WXES 3182

Simulasi Jarak Terdekat Menggunakan Algoritma Dijkstra
(*Simulation of Shortest Path Using Dijkstra's Algorithm*)

Oleh:
Mohd Yohan Bin Ibrahim
WEK 010407

Penyelia: En Mohd Yamani Idna Idris
Moderator: Dr. Phang Keat Keong

ABSTRAK

Simulasi algoritma dijkstra dibangunkan dan langkah-langkah dalam membuat simulasi tersebut diterangkan. Simulasi yang dibangunkan adalah satu modul atau sebahagian daripada Sistem Automasi Meletak kenderaan. Jadual perjalanan sepanjang projek ini adalah seperti di bahagian lampiran. Algoritma Dijkstra atau lebih dikenali dengan *Dijkstra's Algorithm* diterangkan secara terperinci dalam laporan ini. Ini termasuk rumus,kelebihan,kekurangan,kekompleksan rumus Dijkstra dan ciri-ciri algoritma ini. Cara pengiraan *Shortest Path*(jarak terdekat) menggunakan *Dijkstra's Algorithm* turut diterangkan dengan menggunakan contoh-contoh yang sesuai. Proses pembangunan simulasi ini adalah dengan menggunakan aturcara C dengan menggunakan Microsoft Visual C++ 6.0. Carta alir dan pseudocode rumus Dijkstra turut. Contoh-contoh paparan output jangkaan turut disertakan dalam laporan ini. Proses perlaksanaan simulasi diterangkan secara mendalam. Kaedah-kaedah dan proses pengujian simulasi juga diterangkan. Fasa ini dibuat bagi mengesahkan ketepatan algoritma dan aturcara yang dibangunkan. Fasa terakhir iaitu fasa penilaian dan perbincangan turut dibincangkan dalam laporan ini. Di dalam bab ini, kelebihan dan kelemahan simulasi dibincangkan dan diterangkan. Secara keseluruhannya laporan ini lengkap dengan pelbagai maklumat dan kajian tentang *Dijkstra's Algorithm* dalam Bahasa Melayu. Ini menjadikan ia unik kerana tiada lagi rekod yang lengkap tentang algorima ini dalam Bahasa Melayu. Semua sumber maklumat untuk kajian ini adalah dalam bahasa inggeris dan ia diterjemahkan dan diterangkan dalam Bahasa Melayu untuk pemahaman yang mutlak.

PENGHARGAAN

Pertama sekali saya ingin mengucapkan ribuan terima kasih kepada kedua ibu bapa saya yang banyak memberikan banyak sumbangan baik dari segi sokongan moral dan juga material dalam menjayakan dan seterusnya menyiapkan Laporan Latihan Ilmiah Tahap Akhir WXES3182 ini.

Ucapan ribuan terima kasih juga saya tujukan khas buat Penyelia Projek Latihan Ilmiah II WXES3182 iaitu En Yamani Idna Bin Idris yang telah banyak membantu dalam menjayakan projek Latihan Ilmiah tahap awal ini. Tanpa tunjuk ajar dan bimbingan daripada beliau maka projek ini tidak akan dapat direalisasikan. Terima kasih kerana meluangkan masa yang banyak dalam sesi pertemuan dan perbincangan mingguan yang dibuat sepanjang semester ini. Tanpa bantuan, cadangan, idea, komen dan penerangan daripada beliau projek Latihan Ilmiah ini tidak akan mungkin dapat dilaksanakan sebaiknya.

Ucapan terima kasih juga saya tujukan kepada Moderator saya iaitu Dr. Phang Keat Keong yang turut bersama menjayakan Laporan Latihan Ilmiah tahap awal. Terima kasih diucapkan sekali lagi kerana telah memberikan reaksi positif serta memberikan cadangan dan komen yang berguna ketika sesi viva. Tanpa komen dan pertanyaan daripada beliau dapat saya memperbaiki Laporan Latihan Ilmiah Tahap Akhir ini.

Selain daripada itu, turut tidak dilupakan rakan-rakan yang telah turut membantu dalam menjayakan laporan ini dalam sesi perbincangan dan diskusi yang diadakan.Hanya ucapan ribuan terima kasih dapat saya ucapkan.

Isi Kandungan

SENARAI JADUAL

SENARAI RAJAH

Rajah 8

BAB 1 :

PENGENALAN

TAJUK : Simulasi Jarak Terdekat Menggunakan Algoritma Dijkstra

(*Simulation of Shortest Path Using Dijkstra's Algorithm*)

BAB 1 : PENGENALAN

1.1 Definisi Algorithm

Algoritma Dijkstra atau lebih dikenali sebagai *Dijkstra's algorithm* merupakan satu algoritma/rumus yang terbaik dalam mencari *shortest path*(jarak terdekat) di antara dua node/titik/*vertex/network*/router. Edsger Wybe Dijkstra seorang saintis computer Belanda telah dilahirkan pada 11 Mei 1930 dan meninggal dunia pada 6 Ogos 2002.Algoritmanya mudah, senang untuk difahami dan diimplimentasi, malah juga efisien. Pembangun sistem yang dapat menguasai algoritma ini, akan dapat menyelesaikan banyak masalah mengira jarak terdekat(*shortest path*)

Dijkstra's algorithm, apabila diimplementasi pada graf, akan mencari *shortest path*(jarak terdekat) dari titik mula dan destinasi yang diberikan. Algoritma ini terlalu kuat sehinggakan ia mencari semua *shortest path* dari sumber ke semua destinasi. Ini dikenali sebagai *single-source shortest paths problem.*

Algoritma ini dibahagi kepada dua set *vertical* berbeza, set sementara(unsettled/A) dan set kekal(settled/B) Pada awalnya, semua *vertical* adalah sementara(unsettled). Setelah jarak terdekat diketahui pada setiap nod, nod tersebut yang sementara akan bertukar menjadi nod kekal. Apabila semua nod menjadi kekal, algoritma ini tamat.

1.2 Definisi Masalah

Pengguna sering menghadapi masalah meletak kenderaan terutamanya di kawasan Komersial di bandar seperti di pusat-pusat membeli belah. Mereka mengambil masa yang lama untuk mencari tempat meletak kenderaan yang kosong.

Sistem-sistem yang sedia ada, tidak lagi menunjukkan simulasi menggunakan algoritma Dijkstra. Banyak sistem pada masa ini menggunakan algoritma-algoritma lain seperti A*. Jika ketepatan untuk menentukan jarak terdekat(*shortest path*) diperlukan, hanya rumus ataupun algoritma ini sahaja yang dapat menjamin jarak terdekat diketahui secara 100% tepat. Ini hanya mungkin terjadi jika destinasi diberikan(weighted network) dan jarak antara nod adalah positif.

1.3 Objektif Projek

i) Membangunkan satu Simulasi yang boleh mengira jalan dan jarak yang terdekat(shortesrt path) antara dua titik apabila titik awal dan titik destinasi diberikan

ii) Membangunkan satu Simulasi menggunakan *Dijkstra's Algorithm* yang kemudiannya akan digabungkan dengan modul-modul lain dan melengkapkan keseluruhan rekabentuk Sistem Automasi Meletak Kenderaan. Ini akan membolehkan sistem meletak kereta beroperasi secara lebih sistematik.

iii) Memudahkan pembangun sistem untuk membuat sistem yang memerlukan rumus algoritma Dijkstra

iv) Membagunkan satu simulasi yang akan memberitahu langkah-langkah dan jalan yang perlu dilalui untuk sampai ke desinasi dalam jarak terdekat ataupun masa yang paling cepat

1.4 Skop Projek

Satu sistem simulasi yang akan memberitahu pengguna jalan yang paling dekat yang perlu dilalui untuk sampai ke lot parking yang kosong. Hasil daripada simulasi ini akan dibaca oleh C*hain Code*. Kemudiannya *Chain Code* ini akan menyahkod(decode) hasil daripada simulasi menjadi arah dah pergerakan. Hasil ini akan dihantar kepada pangkalan data yang kemudiannya akan menghantar data tersebut kepada robot untuk simulasi pergerakan.Proses keseluruhan sistem ini dan Skop Dijkstra dapat dilihat dengan lebih jelas dalam *Rajah 2.1 Sistem Automasi Meletak Kenderaan.*

Simulasi yang akan dibangunkan ini akan menerangkan konsep dan rumus dijkstra secara keseluruhannya. Walaupun terdapat pelbagai cara untuk mengira jarak terdekat antara dua nod atau titik seperti A* dan lain2, Dijkstra adalah asas kepada semua rumus2 lain. Algoritma Dijkstra juga merupakan antara algoritma terawal yang digunakan dalam dunia sains komputer.Oleh itu projek simulasi ini hanya akan menggunakan rumus Dijkstra.Kajian Literasi tentang Rumus Dijkstra akan diterangkan dengan lebih lanjut dalam Bab 2.

Luas/had algoritma ini adalah berdasarkan jumlah saiz tempat letak kereta. Contohnya tempat letak kereta yang mempunyai keluasan 200*100meter. Halangan seperti dinding, kereta lain di tempat parking dan pintu juga diambil kira untuk menghadkan pergerakan simulati algoritma Dijkstra yang akan dibangunkan.

Pixel akan digunakan untuk mewakili jarak sebenar.Tatasusunan(array) akan digunakan untuk mewakili nod dalam grid. Dengan menggunakan tatasusunan, saiz adalah sekata.Jarak antara 2 nod bersebelahan secara mendatar(horizontal) dan

menegak(vertical) ditetapkan kepada 10 manakala nod bersebelan secara condong(diagonal) ditetapkan kepada 14.

1.5 Pengguna Sasaran

Simulasi algoritma yang akan dibangunkan ini akan digunakan dalam satu sistem mencari tempat letak kenderaan.Ia hanya merupakan satu modul daripada satu sistem yang besar. Dengan adanya simulasi ini, ia melengkapkan sistem Automasi Meletak Kenderaan. Oleh itu, pengguna sasaran akan menjadi pengunjung pusat membeli belah dan komersial yang menggunakan tempat meletak kenderaan komplex yang disediakan. Sistem ini tidak sesuai untuk kegunaan pejabat, kerana biasanya ruang letak kereta untuk pejabat telah dikhaskan untuk sesuatu kenderaan berdasarkan jawatan dan juga staf-staf.

1.6 Hasil yang dijangka

Satu simulasi yang apabila diberi 2 titik, akan menjana satu simulasi yang mencari jarak terdekat antara dua titik dan menunjukkan nod-nod atau jalan yang perlu dilalui sebelum sampai ke parking kosong. Ia akan ditunjukkan secara paparan dos. Paparan ini akan menunjukkan grid/susunan jalan yang di lalui untuk sampai ke destinasi melalui jalan terdekat(*shortest path*).

1.7 Penjadualan Projek

Jadual Perjalanan Projek yang lengkap adalah seperti di lampiran.

BULAN	7				8				9				10			
MINGGU	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Penyiasatan Awalan																
Analisa Masalah																
Analisa Keperluan Simulasi																
Analisa Simulasi																
Rekabentuk																

BULAN	10				11				12				1			
MINGGU	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Pembangunan																
Implementasi dan pengujian																
Dolumentasi																

Jadual 1.1 Jadual Perjalanan Projek

1.8 Ringkasan

Dalam bab1 ini menerangkan secara ringkas tentang simulasi masalah dan rumus dijkstra. Ia juga memberikan gambaran skop simulasi yang akan dibangunkan dan hasil jangkaan. Jadual perjalanan projek juga diberikan. Jadual Perjalanan Projek Simulasi Jarak Terdekat Menggunakan Algoritma Dijkstra adalah seperti di lampiran.

BAB 2 :

KAJIAN LITERASI

BAB 2 : KAJIAN LITERASI

2.1 Masalah dalam Sistem Tempat Meletak Kenderaan

Pengguna sering menghadapi masalah meletak kenderaan terutamanya di kawasan Komersial di bandar seperti di pusat-pusat membeli belah. Mereka mengambil masa yang lama untuk mencari tempat meletak kenderaan yang kosong.Rata-rata pengguna menggunakan secara puratanya selama ½ jam untuk mencari tempat meletak kenderaan. Terdapat juga pengguna mengambil masa yang lebih lama untuk mencari tempat meletak kenderaan daripada mengambil masa membeli belah di pusat membeli belah sahaja.Ini hanya akan merugikan kedua-dua pihak,pengguna dan juga pemilik.

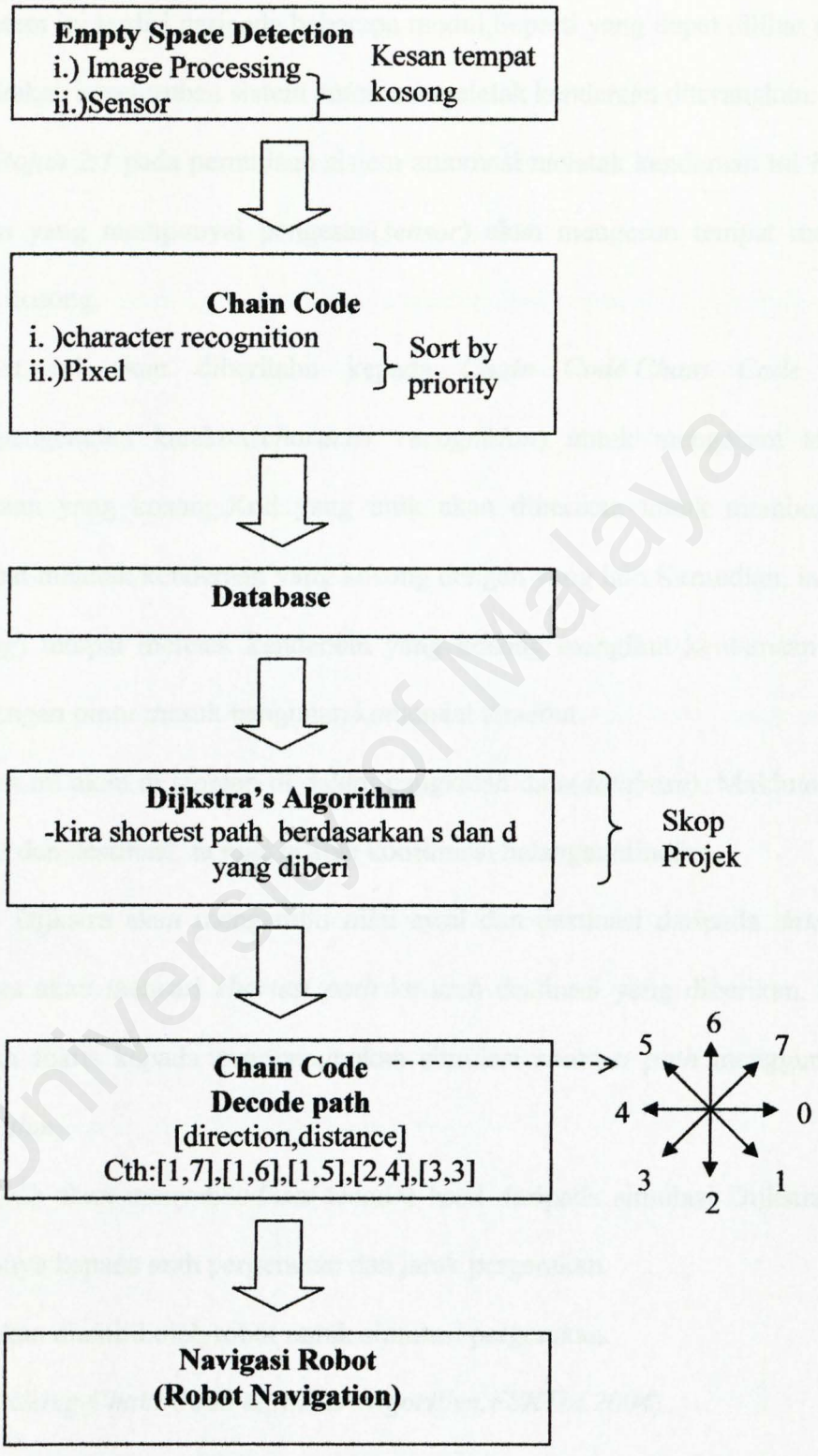
Ini juga mendatangkan masalah kepada Pemilik tempat meletak kenderaan di premis komersial kerana sistem meletak kenderaan yang kurang effisien dan pergerakan traffik dalam tempat meletak kenderaan yang selalu sesak.Ketidak puasan pengguna boleh mengakibatkan pengguna mencari alternatif lain untuk meletak kenderaan mereka di tempat meletak kenderaan lain walaupun agak jauh daripada premis perniagaan. Ini akan mendatangkan kerugian kepada pemilik tempat meletak kenderaan dan juga tempat komersial apabila pengguna terasa bosan akan sistem meletak kenderaan yang tidak sistematik. Ini akan mengurangkan daya tarikan pengunjung ke sesuatu pusat membeli belah.

2.2.Cara Lama yang digunakan

Pada masa tertentu pengawal atau penjaga tempat meletak kenderaan perlu membantu dalam mengawal aliran trafik dalam tempat meletak kenderaan untuk memastikan pengguna mendapat tempat meletak kenderaan.

Ketiadaan sistem yang dapat menunjukkan atau memberitahu pengguna arah ke tempat letak kenderaan yang kosong yang terdekat.

2.3 Cadangan untuk Mengatasi Masalah



Rajah 2.1 Sistem Automasi Meletak Kenderaan

Oleh itu satu sistem automasi meletak kenderaan dicadangkan untuk mengatasi masalah ini. Sistem ini terdiri daripada beberapa modul.Seperti yang dapat dilihat dalam *Rajah 2.1* pergerakan keseluruhan sistem automasi meletak kenderaan diterangkan.

Seperti *Rajah 2.1* pada permulaan sistem automasi meletak kenderaan ini *Empty Space Detection* yang mempunyai pengesan(*sensor*) akan mengesan tempat meletak kenderaan yang kosong.

Maklumat ini akan diberitahu kepada *Chain Code.Chain Code* akan menggunakan pengenalan karakter(*character recognition*) untuk mengecam tempat melatak kenderaan yang kosong.Kod yang unik akan diberikan untuk membezakan antara satu tempat meletak kenderaan yang kosong dengan yang lain.Kemudian, ia akan menyorot(*sorting*) tempat meletak kenderaan yang kosong mengikut keutamaan iaitu jarak terdekat dengan pintu masuk bangunan komersial tersebut.

Maklumat ini akan di simpan di dalam pangkalan data(*database*). Maklumat ini adalah nilai awal dan destinasi. Ia juga adalah koordinasi halangan/dinding.

Simulasi Dijkstra akan mengambil nilai awal dan destinasi daripada *database* dan kemudiannya akan mencari *shortest path* ke arah destinasi yang diberikan. Skop projek ini adalah fokus kepada membangunkan simulasi *shortest path* menggunakan *Dijkstra's Algorithm.*

Chain Code akan menyahkodkan(decode) hasil daripada simulasi Dijkstra dan menterjemahkannya kepada arah pergerakan dan jarak pergerakan.

Arahan akan diambil oleh robot untuk simulasi pergerakan.

(*Parking System Using Chain Code & A-Star Algorithm,FSKTM.2004*)

2.4 Latar belakang Algoritma yang Ingin Dibangunkan.

Pada tahun 1959 satu artikel 3 muka surat yang bertajuk *A Note on Two Problems in Connexion with Graphs* telah diterbitkan dalam satu journal *Numerische Mathematik.* Dalam kertas ini, Edsger W. Dijkstra seorang saintis komputer Belanda yang berumur 29 tahun telah mengemukakan algoritma cadangan untuk penyelesaiaan masalah 2 teori graf asas (the *minimum weight spanning tree problem* dan *shortest path problem.*) Pada hari ini *Dijkstra's Algorithm* untuk mencari jarak terdekat antara dua nod adalah algoritma yang paling popular dalam dunia sains komputer(CS) dan juga kajian operasi(Operation Research)

Sejak 1959, *Edsger Wybe Dijkstra* telah memperkenalkan satu cara atau kaedah yang telah digunakan secara global sehingga hari ini iaitu *Shortest Route menggunakan Dijkstra's Algorithm.* Ia merupakan satu kaedah untuk mencari jarak antara dua nod atau titik apabila titik mula(sourcw) dan sasaran(destination) ditentukan.

Shortest path(Jarak Terdekat) boleh bermaksud jarak terdekat secara geometri dari satu sumber ke destinasi dalam satu rangkaian yang bersambung. Ia juga boleh dikatakan jumlah terdekat antara sumber(source,origon) dan destinasi(destination).

Anggapan: Nilai pemberat mestilah positif dan bukan n.

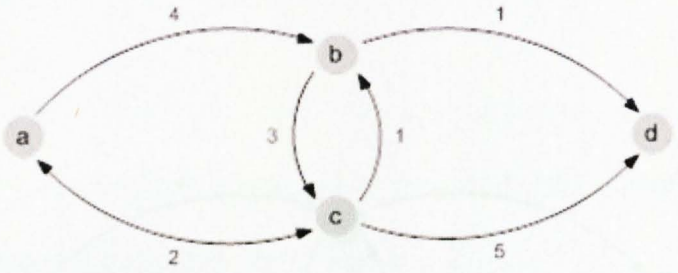
Rumus Dijkstra F(n)= G(n)

D(v)=min {D(v) , D(w)+l(w,v)}

```
     distance(s,u)      weight(u,v)        distance(s,u) + weight(u,v)
s ------------------> u ----------> v  ==>  s -------------------> v
       path                edge                        path
```

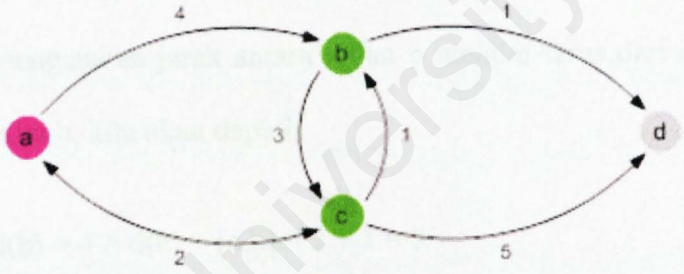
2.5 Contoh Penerangan Rumus Dijkstra

Untuk memahami algoritma ini, satu contoh ringkas diberi. *Dikjstra's shortest path algorithm* akan ditunjukkan dalam graf berikut bermula dari nod/sumber/titik mula vertex *a*.



Rajah 2.5.1 Penerangan Contoh

Kita mulakan dengan menambah titik mula a kepada set Q(unsettled vertices). Q bukan kosong. a menjadi S(settled vertices) dan kita rehatkan nod bersebelahannya.



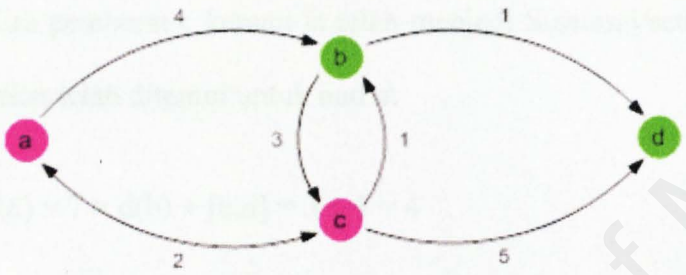
Rajah 2.5.2 Penerangan Contoh

Nod bersebelahan a adalah nod b dan c. Jarak di antara a dan b dikira. d(b) disetkan kepada infiniti, olehitu kita guna

$d(b) = d(a) + [a,b] = 0 + 4 = 4$.

$\pi(b)$ diset kepada a, dan b ditambah kepada Q. Sama dengan c kita setkan d(c) kepada 2 dan $\pi(c)$ kepada a.

Sekarang, Q mengandungi b dan c.Seperti rajah di atas c adalah vertex dengan jarak terdekat bernilai 2.Ia di tukar kepada S(settled nodes). Kemudian node/vertex bersebelahan c iaitu b,d dan a direhatkan.



Rajah 2.5.3 Penerangan Contoh

a diabaikan kerana ia telah menjadi S(settled set). Langkah algoritma pertama telah mengatakan jarak antara a dan b melalui terus dari a ke b.Jika kita bandingkan jiran c iaitu b, kita akan dapati;

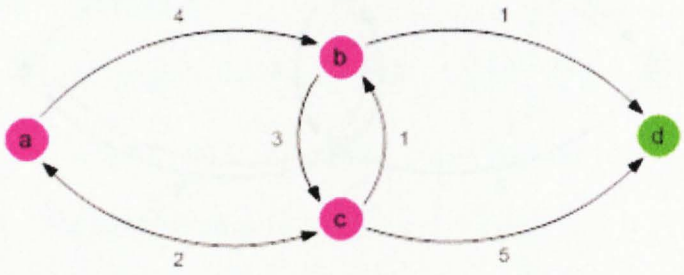
$d(b) = 4 > d(c) + [c,b] = 2 + 1 = 3$

Sekarang kita telah mengetahui bahawa jarak terdekat sebenarnya melalui c wujud di antara a dan b diberi nilai d(b) = 3 dan $\pi(b)$ diupdate kepada c. nod b sekali lagi menjadi Q.

Vertex berdekatan seterusnya ialah d, $d(d)$ disettkan kepada 7 dan $\pi(d)$ kepada c.

Vertex Q dengan jarak terdekat dikeluarkan daripada queue, menjadikannya sekrang b.

Kita tukarnya kepada S(settled set) dan rehatkan nod-nod bersebelahannya iaitu *c* dan *d*.



Rajah 2.5.4 Penerangan Contoh

Kira pemberat c kerana ia telah menjadi S(selesai/settled) Akan tetapi jarak yang lebih dekat telah ditemui untuk nod *d*:

$d(d) = 7 > d(b) + [b,d] = 3 + 1 = 4$

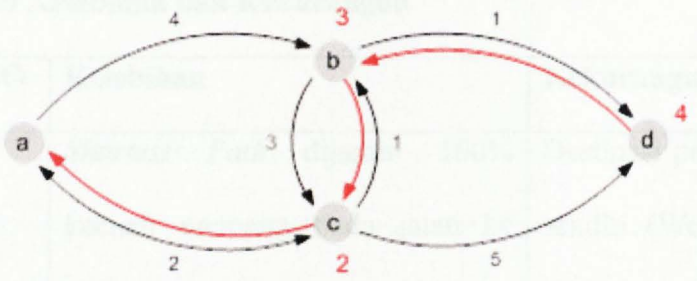
Olehitu kite tukarkan $d(d)$ kepada 4 and $\pi(d)$ kepada *b*. D dimasukkan ke dalam set Q.

Pada tahap ini, satu-satu vertex yang tinggal di Q(unsettled set) adalah d, dan semua nod bersebelahan(jirannya) telah menjadi S(settled)

Algoritma ini tamat kerana semua nod telah menjadi selesai/settled S.

Hasil akhir ditunjukkan seperti *Rajah 2.5.5* di bawah.

- π - the *shortest path*, in predecessor fashion
- *d* – jarak terdekat(*shortest distance*) daripada sumber(source) untuk setiap verrtex



Rajah2.5.5 Penerangan Contoh

Di antara algoritma untuk mencari jarak terdekat antara dua nod Dijkstra's merupakan algoritma yang paling mudah dan mempunyai keupayaan yang baik dalam mencapai matlamatnya.

2.6 Kelebihan dan Kekurangan

NO	Kelebihan	Kekurangan
1	*Shortest Path* dijamin 100% kecuali memang tiada jalan ke destinasi yang diberikan.	Destinasi perlu diberikan.Tidak boleh dicari sendiri. (Weighted Network)
2.	Coding yang mudah	Kurang fungsi tambahan dan hanya maklumat ringkas dapat dipaparkan
3.	*Shortest Path* untuk semua destinasi akan dicari kerana menyemak seluruh rangkaian jalan yang mungkin.	Ambil masa yang lama-Kerana tiada estimation(jangkaan) dan juga menyemak seluruh rangkaian jalan yang mungkin.
4.	Mengira jalan terdekat bermula daripada sumber atau titik permulaan	Jarak antara nod bersebelahan perlu ditentukan terlebih dahulu.
7.	Ia boleh ditamatkan sejurus selepas jarak terdekat dari dua nod diketahui	Nod negatif tidak boleh dikira,oleh itu semua jarak antara satu nod perlulah positif

Jadual 2.5 Kelebihan dan Kekurangan Algoritma Dijkstra

2.7 Kekangan Dijkstra

Masa yang lama diambil untuk mendapatkan *shortest path.*Ini adalah kerana ia terpaksa menyemak seluruh rangkaian jalan yang ada untuk mendapatkan *shortest path* ke destinasi yang diberikan.Ketiadaan Heuristic(jankaan) menyebabkannya tidak dapat mengagak arah yang perlu dilaluinya.

Kekonpleksan Dijkstra akan bertambah dengan saiz Tempat Meletak kenderaan. Ini bermakna pengiraan akan bertambah sukar untuk luas permukaan yang lebih besar.

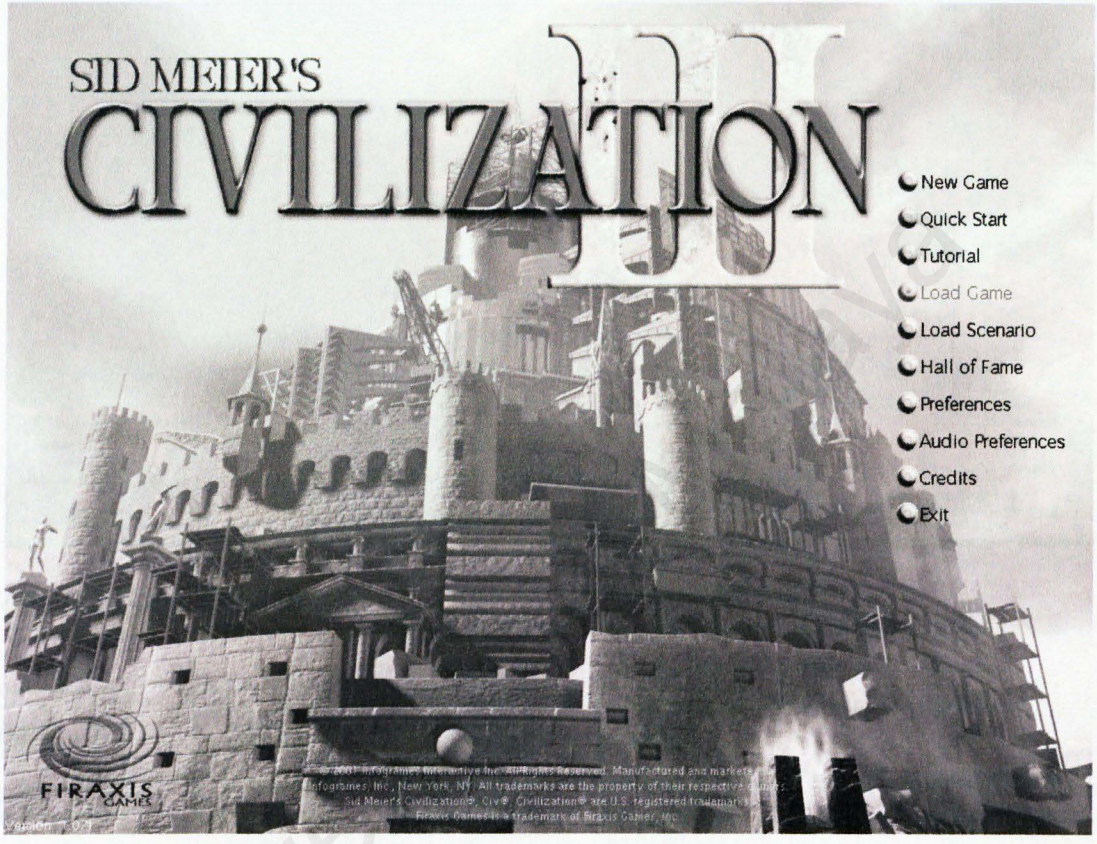
Masalah penggunaan pixel untuk output bagi simulasi adalah ia terlalu kecil apabila hendak dilihat dengan mata kasar.

Output MFC mungkin sukar dihasilkan. Oleh itu Output jangkaan utama simulasi adalah hanya dengan MS-DOS sahaja.

Aturcara C yang kompleks perlu dihasilkan daripada rumus Dijkstra. Setiap kemungkinan perlu diaturcarakan termasuk halangan, cara pergerakan dan saiz bagi templet dalam tatasusunan yang digunakan.

2.8 Implementasi dan Sistem-sistem yang menggunakan algorithm

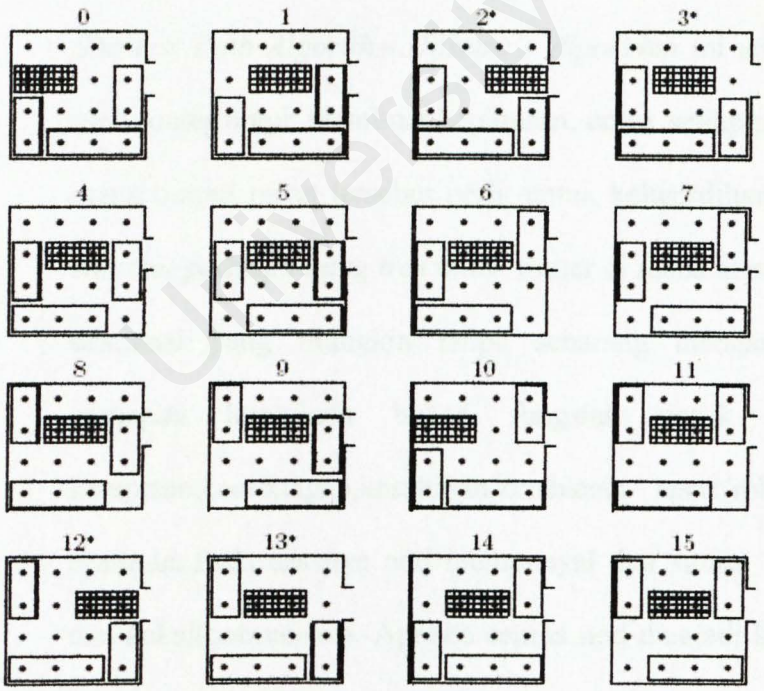
2.8.1 *Game* yang berbentuk Grid seperti 2D contohnya : *Game Civilazation*



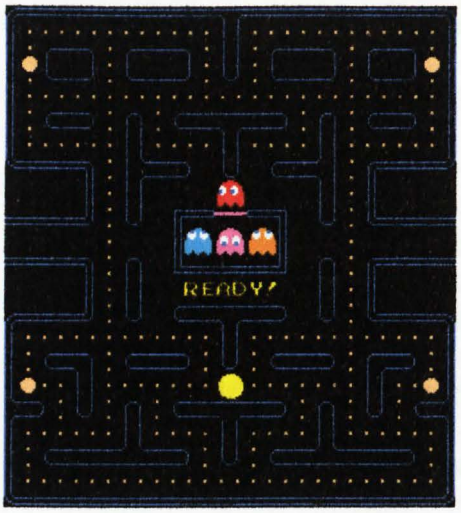
Rajah 2.2.1 Game Civilazation

Seperti dalam *Rajah 2.2.2* setiap pergerakan adalah dalam grid. Setiap Pergerakan oleh pemain akan mencari jarak terdekat apabila pemain menclik tetikus di satu titik dalam peta. Pergerakan pemain akan melalui jalan yang paling dekat dengan titik awal tersebut. Konsep *shortest path* digunakan dalam permainan komputer ini.

Rajah 2.2.2 Game Civilazation



Rajah 2.2.3 Game Rush Hour



Rajah 2.2.4 Game Pac-Man

Seperti dalam *Rajah 2.2.4* raksaksa(merah) akan mencari jalan terdekat untuk sampai kepada pemain(bola kuning) Ini menggunakan rumus *shortest path.*

2.8.2 Implementasi *Dijkstra's algorithm* digunakan dalam *routing* internet. –untuk mengira jadual routing(routing table) pada satu router menggunakan *Dijkstra's Shortest Path Algorithm.Dijkstra's Algorithm* ini adalah logik yang digunakan oleh router untuk membuat keputusan, untuk setiap paket yang datang/masuk ke mana output paket tersebut perlu untuk keluar/dihantar.Algoritma ini membina *shortest-path spanning tree* untuk router di mana ia mengandungi jalan ke semua destinasi yang mungkin tanpa sebarang ulangan(loops). Rumus Dijkstra menepati keperluan bukan fungsian untuk algoritma routing iaitu ketepatan,kesenangan,kesahihan,ketahanan lasak(robustness), kestabilan dan optimum.Pada asasnya nod mempunyai dua situasi iaitu sementara(temporary) dan kekal(permanent). Apabila semua nod menjadi kekal algoritma ditamatkan.

Apabila router telah menjumpai *shortest-path spanning tree* ia boleh membina jadual routing(routing table)

2.8.3 Automated Speech recognition(ASR)- Ia membolehkan mengecam suara melalui sistem komputer dibuat. Hidden Markov Models (HMM) dalam konteks *shortest path,* boleh dikatakan graf pemberat(*weighted graph*) dengan kemungkinan pertukaran=nod berpemberat dan pernyataan=hubungan. Tambahan pula, setiap keadaan HMM mempunyai nilai kos positif. Prinsip Algoritma Dijkstra diaplikasikan.

2.8.4 Implementasi OSPF (*Open shortest path First*)-*Link state routing*

2.8.5 MapQuest –mencari jarak terdekat diantara dua titik dalam peta banyak digunakan di antara bandar-bandar di seluruh Amerika Syarikat(AS)

2.9 Kekompleksan Dijkstra(Complexity)

Prestasi(*performance*) *Dijkstra's Algorithm* bergantung kepada bagaimana ia diimplementasikan. Ini banyak bergantung kepada struktur data(*data structure*) yang digunakan di set sempadan(*frontier set*).

Seperti rumus dijkstra, set A adalah set nod yang belum selesai(unsetled). Proses mengalihkan dan mengeluarkan set minimum dari set A ke set B memerlukan satu operasi *delete_min* pada struktur data yang digunakan oleh set A. Mengemaskini nilai pemberat terkini dan yang paling minimum(setelah rumus dijkstra digunakan) memerlukan operasi *decrease_key* dan proses memasukkan nilai set A memerlukan operasi *insert.* Adalah amat penting struktur data yang digunakan untuk set A menyokong semua operasi-operasi ini dalam kekompleksan masa yang boleh diterima. (*reasonable time complexity*)

Terdapat 3 jenis struktur data yang biasa digunakan iaitu *Fibonacci heap 2-3 heap* dan *binary heap..Complexity* atau kekompleksan Struktur-struktur data ini menyokong operasi-operasi yang diperlukan.

Rumus pengiraan kekompleksan(*computational complexity*) bagi Dijkstra adalah O((E + V) log V)

Jenis Struktur Data	Rumus	Rumus untuk graf yang besar
binary heap time complexity	O(mlog n)	$O(n^2 \log n)$
Fibonacci heap	=O (m+n log n)	$O(n^2+n \log n)$
2-3 heap	=O (m+n log n)	$O(n^2+n \log n)$
Dijkstra	O (n2)	= O((E + V) log V)

Rajah 2.3 Rumus-Rumus Kekompleksan

n=bilangan nod, m=bilangan penyambung nod/edges/arcs

V=bilangan nod,E=bilangan penyambung nod/edges/arcs

Daripada *Rajah 2.3* dapat disimpulkan bahawa kekompleksan rumus dijkstra apabila menggunakan *binary heap* adalah tinggi jika saiz graf besar. Ia mungkin lebih pantas daripada Fibonacci dan 2-3 heap jika saiz graf adalah kecil. Walaubagaimanapun, jika saiz graf kecil *Fibonacci heap* dan *2-3 heap* juga sesuai digunakan kerana mempunyao pengiraan *overhead* yang tinggi.

Fibonacci heap → s = 3m + 1:44n log2 n

2-3 heap→s = 2m+ 2n log2 n

complexity untuk *Dijkstra's Algorithm* adalah O (n2)

complexity untuk *Dijkstra's Algorithm* menjadi $O(V \lg V + E)$

2.10 Algoritma-Algoritma Lain

2.10.1 A*: Ada fungsi heuristic $h(v)$ (jangkaan)- Ia lebih cepat daripada Algoritma Dijkstra kerana ia ada nilai jangkaan. Bellman-Ford Algorithm- Berlainan daripada Dijkstra, Bellman-Ford boleh digunakan untuk graf yang mempunyai nilai kitaran negatif selagi ia tidak berulang dari sumber S.

2.10.2 Floyd's -Warshall Algorithm- Untuk menyelesaikan *All Pairs Shortest Path Problem*

2.10.3 Kruskal's Algorithm –adalah satu algoritma dari teori graf yang mencari *minimum spanning-tree* dalam graf berpemberat.

2.10.4 Edmonds-Karp algorithm-adalah implementasi daripada Ford-Fulkerson algorithm

2.10.5 Boruvka's Algorithm-untuk mencari *minimum spanning tree*

2.10.6 Ford-Fulkerson algorithm

2.10.7 Prim's Algorithm

2.11 Ringkasan

Dalam Bab 2 ini, analisa terhadap masalah yang dihadapi dibuat dan cara-cara yang dicadangkan untuk mengatasi masalah tersebut dicadangkan. Bab ini juga memberi penerangan yang lebih mendalam ke atas Algoritma yang akan digunakan dalam membangunkan Simulasi iaitu *Dijkstra's Algotrithm.*

BAB 3:

METODOLOGI

3.0 METODOLOGI

3.1 Kaedah Dan Teknik Pelaksanaan

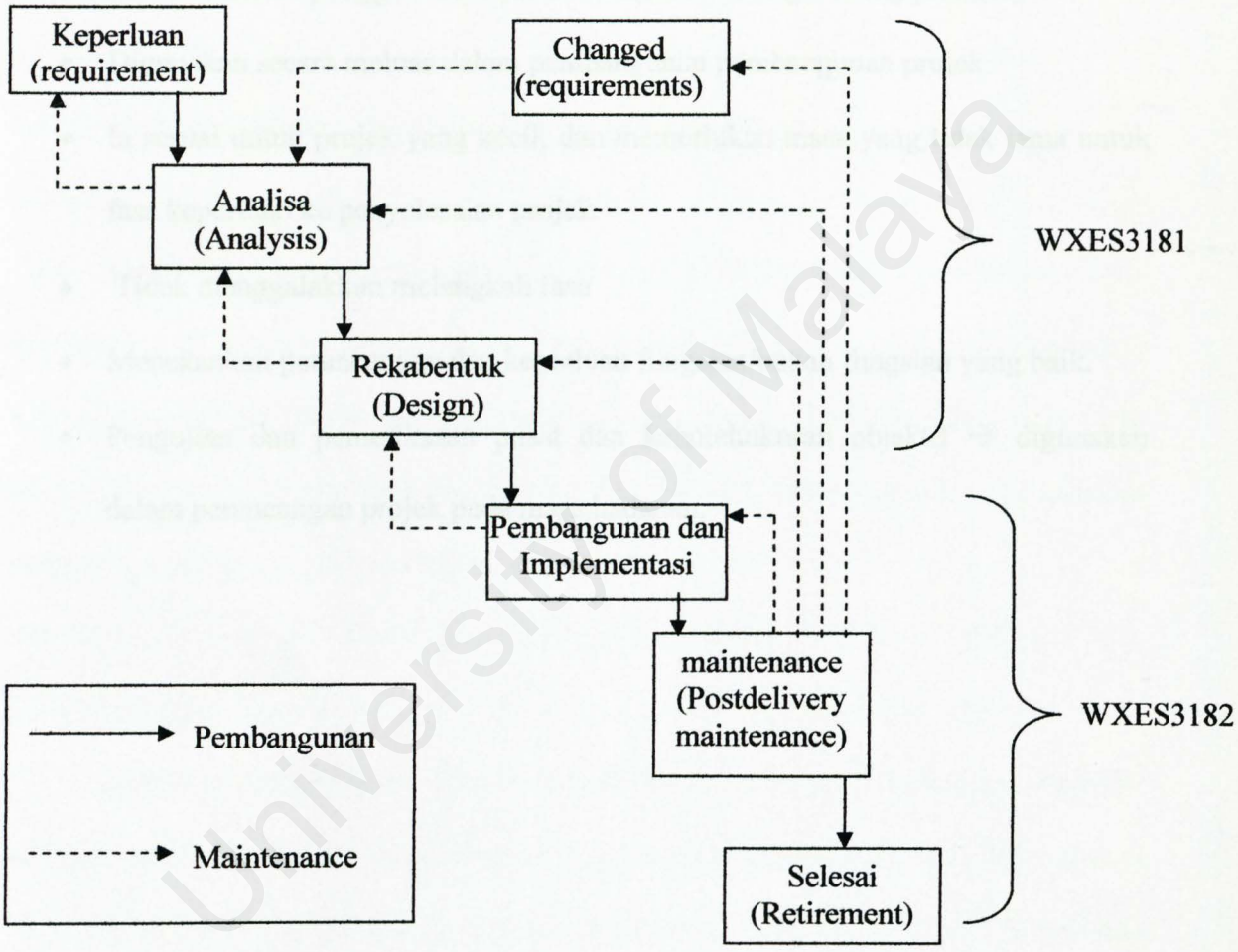
Untuk membangunkan simulasi ini, beberapa kaedah digunakan. Metodologi dalam membangunkan projek *Simulasi Shortest Path menggunakan Dijkstra's Algorithm* adalah satu model Air Terjun(Waterfall life-cycle model).Model ini mempunyai 5 anak tangga yang penting.

Untuk WXES3181 3 tangga teratas model Air Terjun telah dijalankan. Manakala sambungannya akan dilakukan dalam WXES3182.

Pengaturcaraan C di pilih sebagai bahasa pengaturcaraan utama dalam membangunkan simulasi ini.

3.2 Kaedah Dan Teknik Pelaksanaan:Waterfall life-cycle model(Model Air Terjun)

Sepanjang projek ini, kaedah-kaedah yang perlu dilalui dan dijalankan adalah seperti *Rajah 3.1 Model Air Terjun* di bawah.



Rajah 3.1 Model Air Terjun

Model Air Terjun(*Waterfall Model)* di gunakan kerana beberapa sebab.

- Seperti mana model air terjun dikenali kerana ia mengalir ke bawah seperti air terjun dari satu fasa ke fasa berikutnya.
- Ia mengambil proses-proses aktiviti asas seperti keperluan,pembangunan, pengesahan dan perkembagan dan mewakilkannye kepada fasa proses berlainan.
- Mencerminkan penggunaan kejuruteraan(Reflects engineering practice)
- Digunakan secara meluas dalam pembangunan pembangunan projek
- Ia sesuai untuk projek yang kecil, dan memerlukan masa yang tidak lama untuk fasa keperluan ke penyelesaian projek.
- Tidak menggalakkan melangkah fasa
- Menekankan perancangan dan keperluan fungsian/bukan fungsian yang baik.
- Pengujian dan pemeriksaan pusat dan kebolehukuran objektif → digunakan dalam perancangan projek pada masa hadapan.

3.3 C

Simulasi ini akan dibangunkan menggunakan platform C kerana ia menggunakan konsep berorientasikan objek. Ini membolehkan menggabungkan beberapa fungsi dalam satu aturcara. Penggunaan kelas juga merupakan satu kelebihan. Pengaturcaraan C atas bahasa peringkat tinggi(high level language)

3.4 Kaedah Penyelidikan: Kaedah Mengumpul Maklumat.

Terdapat banyak kaedah mencari maklumat yang digunakan dalam pembagunan simulasi ini. Maklumat yang diperolehi berguna sebagai rujukan dalam membagunkan simulasi ini. Dengan kaedah mencari maklumat yang betul, ia dapat menambahkan dan memudahkan pemahaman terhadap simulasi ini. Antara kaedah yang digunakan dalam mencari maklumat adakah seperti berikut:

3.4.1 Membaca

Membaca bahan-bahan seperti buku-buku rujukan yang berkaitan, majalah-majalah computer, kamus dan beberapa lagi yang diperoleh daripada Perpustakaan Utama Universiti Malaya dan juga bilik dokumen di Fakulti Sains Komputer dan Teknologi Maklumat. Pembacaan daripada majalah-majalah dan buku-buku rujukan yang dibeli sendiri turut digunakan.Selain daripada itu,contoh-contoh laporan latihan ilmiah senior-senior lepas turut dibaca dan diteliti,untuk memberikan gambaran sebenar dan idea tentang apa yang perlu dilakukan dalam membangunkan projek ini.

3.4.2 Internet

Kebanyakan maklumat yang diperoleh adalah daripada internet.Ini adalah kerana internet mengandungi pelbagai maklumat yang terkini tentang teknologi-teknologi baru dan juga sumber-sumber yang berkaitan dengan simulasi yang ingin dibangunkan. Selain itu,maklumat yang diperoleh daripada sumber digital/elektronik(internet) adalah murah jika dibandingkan dengan maklumat di atas kertas. Maklumat yang diperoleh daripada internet ditapis melalui skop yang berkaitan simulasi projek yang dibangunkan. Forum-forum di internet juga digunakan sebagai sumber rujukan dengan memerhatikan ruangan soal jawab dan menggunakan link-link yang diberikan.

3.4.3 Perbincangan dan Diskusi

Sesi perbincangan dilakukan dengan supervisor seminggu sekali pada hari Jumaat petang. Ini dilakukan untuk mengenal pasti obkektif-objektif projek, skop projek, kekangan projek, kajian literasi projek, keperluan fungsian, keperluan bukan fungsian, rekabentuk simulasi(dari sudut algoritma, kod pseudo, dan carta alir) dan sebagainya.

Selain itu, perbincangan turut dilakukan dengan seorang rakan yang mengambil tajuk *Shortest Route* menggunakan *A* Algorithm*. Banyak masa diperuntukkan dalam sesi berbincangan untuk memahami lagi konsep *shortest path finding* dan algoritma-algoritma yang terlibat.

3.4.5 Pemerhatian dan kajian ke atas sistem meletak kenderaan

Satu pemerhatian secara rambang telah dibuat di tempat-tempat meletak kenderaan yang sentiasa penuh.Didapati tiada lagi sistem elektronik atau sebarang peranti yang membantu atau memberikan informasi tentang tempat letak kenderaan wujud.Ini menjadikan sistem yang ingin dibangunkan masih belum ada atau diimplimentasi di Malaysia.

3.5 Ringkasan

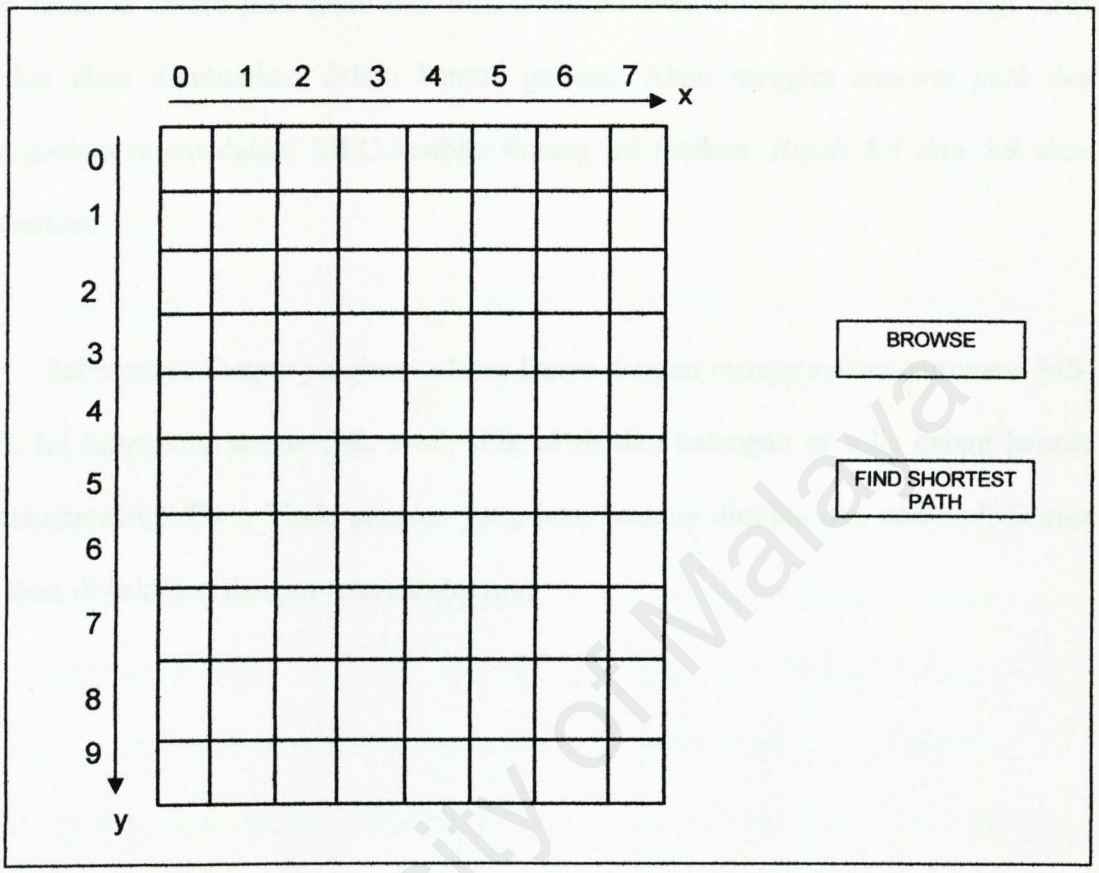
Bab 3 ini menerangkan secara metodologi dan kaedah-kaedah yang digunakan dalam membangunkan simulasi ini.Metodologi utama yang dipilih adalah Model Air Terjun(Waterfall model).Bab ini juga menerangkan cara-cara maklumat dikumpulkan.

BAB 4 :

ANALISA SIMULASI

BAB 4 ANALISA SIMULASI

4.1 Keperluan Fungsian



Rajah 4.1 Keperluan Fungsian Simulasi

Seperti *Rajah 4.1 Keperluan Fungsian Simulasi,* terdapat dua butang penting iaitu butang *BROWSE* dan butang *FIND SHORTEST PATH.*

Butang BROWSE : Apabila butang ini ditekan simulasi akan mengambil *template* yang memberikan nilai titik awal dan titik akhir.Template ini juga mengandungi halangan/dinding bagi simulasi. Apabila butang ini ditekan *Rajah 5.7* akan dipaparkan.

Butang: FIND SHORTEST PATH : Apabila ini ditekan Jarak terdekat(*shortest path*) akan dikira dari dua titik/nod yang diberikan oleh template(selepas menekan Browse). Jarak terdekat antara titik mula dan titik destinasi akan dikira dan output bagi jarak terdekat akan dikeluarkan dalam bentuk garisan. Akan mengira *shortest path* dan memaparkan ouput dalam MFC.Apabila butang ini ditekan, *Rajah 5.8 dan 5.9* akan dipaparkan.

Sebenarnya Output jangkaan utama hanya dengan menggunakan command MS-DOS. Ini bermakna semua titik awal, titik akhir dan halangan di tulis dalam bentuk pengaturcaraan C/C++. Tiada paparan yang jelas tentang dinding dan nod-nod. Semua nod akan diwakilkan dengan tatasusan(*array*)

4.2 Keperluan Bukan Fungsian

- Mudah alih/perihal mudah dibawa (portability)-Ini adalah kerana saiznya kecil. Ia adalah simulasi.
- Kebolehpercayaan(reliability)- simulator yang akan dibina boleh dipercayai 80% tepat.
- Ketahan lasakan (robustness) – Oleh kerana ia menggunakan C ia agak lasak.
- Boleh diubahsuai (maintainability)-Supaya ia mudah diubahsuai pada masa hadapan jikalau berlaku perubahan keperluan.
- Masa tindak balas(response time)-akan mengambil masa lebih kurang dari 10saat untuk mendapat output. Saiz memory yang digunakan hanya kecil teetapi ia bergantung juga pada saiz kekomplesan rumus ini. Untuk tempat meletak kenderaan yang sangat besar, masa yang diperlukan akan bertambah sedikit.
- Paparan yang mudah difahami-oleh kerana MS-DOS digunakan dan keseluruhan simulasi adalah dalam C,senarai pengaturcaraan perlu dibuat dengan teratur dan menggunakan fungsi-fungsi dan objek-objek yang teratur supaya pengubahsuaian di masa hadapan dapat dilakukan jika diperlukan.

4.3 Keperluan Perkakasan

- Pentium II dan ke atas
- Ada MS-DOS
- Mcrosoft 98 dan ke atas.
- 128 MB RAM
- 1 G memori kosong

4.4 Keperluan Perisian

Microsoft Visual Studio 6.0 Visual C++ 6.0:- C/C++

Program dalam C++ mengandungi *classes*(kelas) dan *functions*(fungsi). Satu kelebihan utama menggunakan C++ adalah kebolehannya untuk berorientasian objek.Objek adalah komponen perisian yang boleh diguna semula yang mewakilkan benda di dalam kehidupan sebenar. Aturcara berorientasikan objek mudah difahami,mudah disemak dan juga mudah diubahsuai. Saiz library yang besar membolehkan penambahan pelbagai jenis fungsi. Bahasa pengaturcaraan C++ adalah bahasa peringkat tinggi dan banyak fungsinya. Ia sesuai digunakan dalam pelbagai jenis aplikasi. Ia adalah antara bahasa pengaturcaraan yang paling meluas digunakan. Microsoft Visual C++ adalah versi C++ baru yang digunapakai pada masa kini.

Aturcara C juga masih digunakan secara meluas pada masa kini. Banyak contoh-contoh dan dan algoritma-algoritma menggunakan bahasa aturcara C. Ini adalah kerana kesesuaian untuk diimplementasi di dalam pelbagai sistem dan suasana berbeza.

4.5 Ringkasan

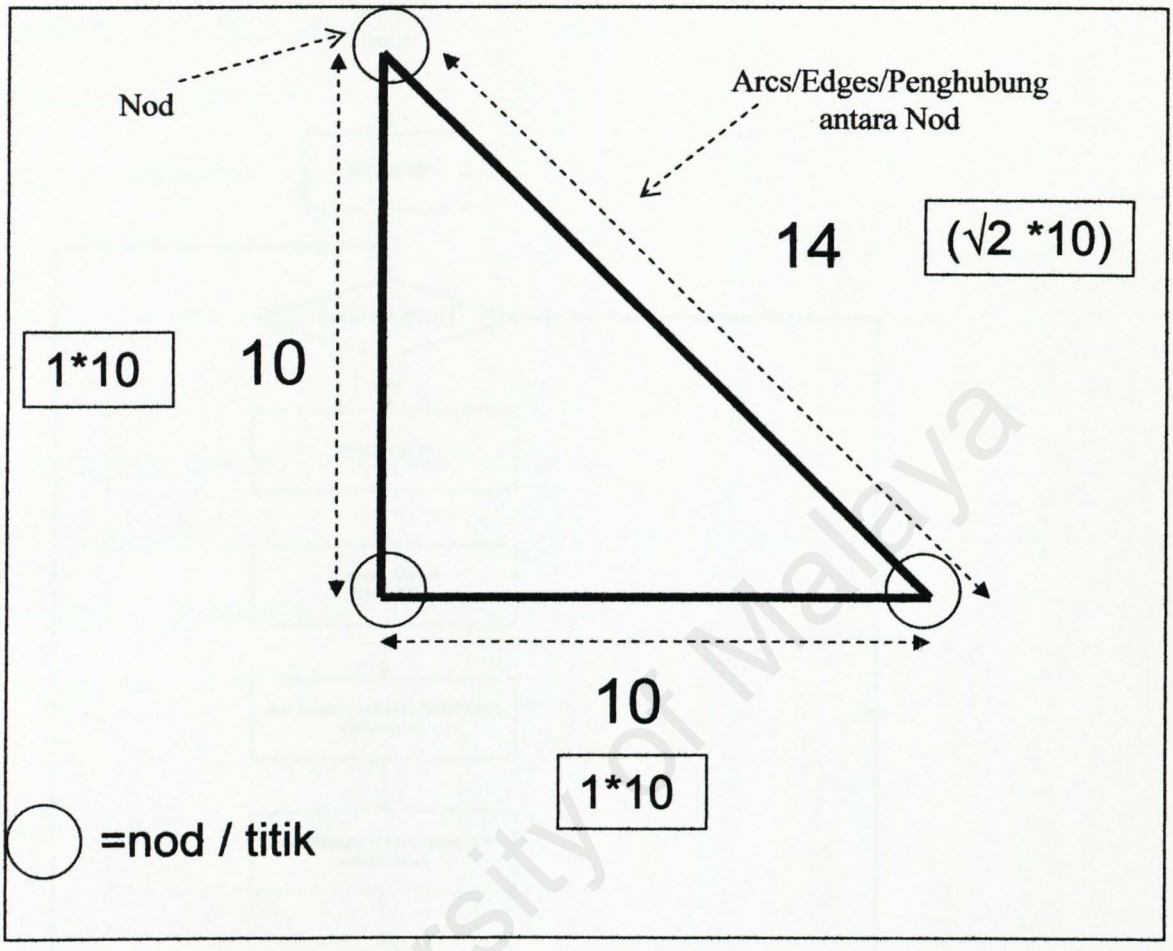
Bab 4 ini menerangkan keperluan fungsian dan bukan fungsian bagi simulasi yang akan dibangunkan. Ia juga menyatakan keperluan perkakasan dan keperluan perisian simulasi.

BAB 5 :

REKABENTUK SIMULASI

BAB 5 : REKABENTUK SIMULASI

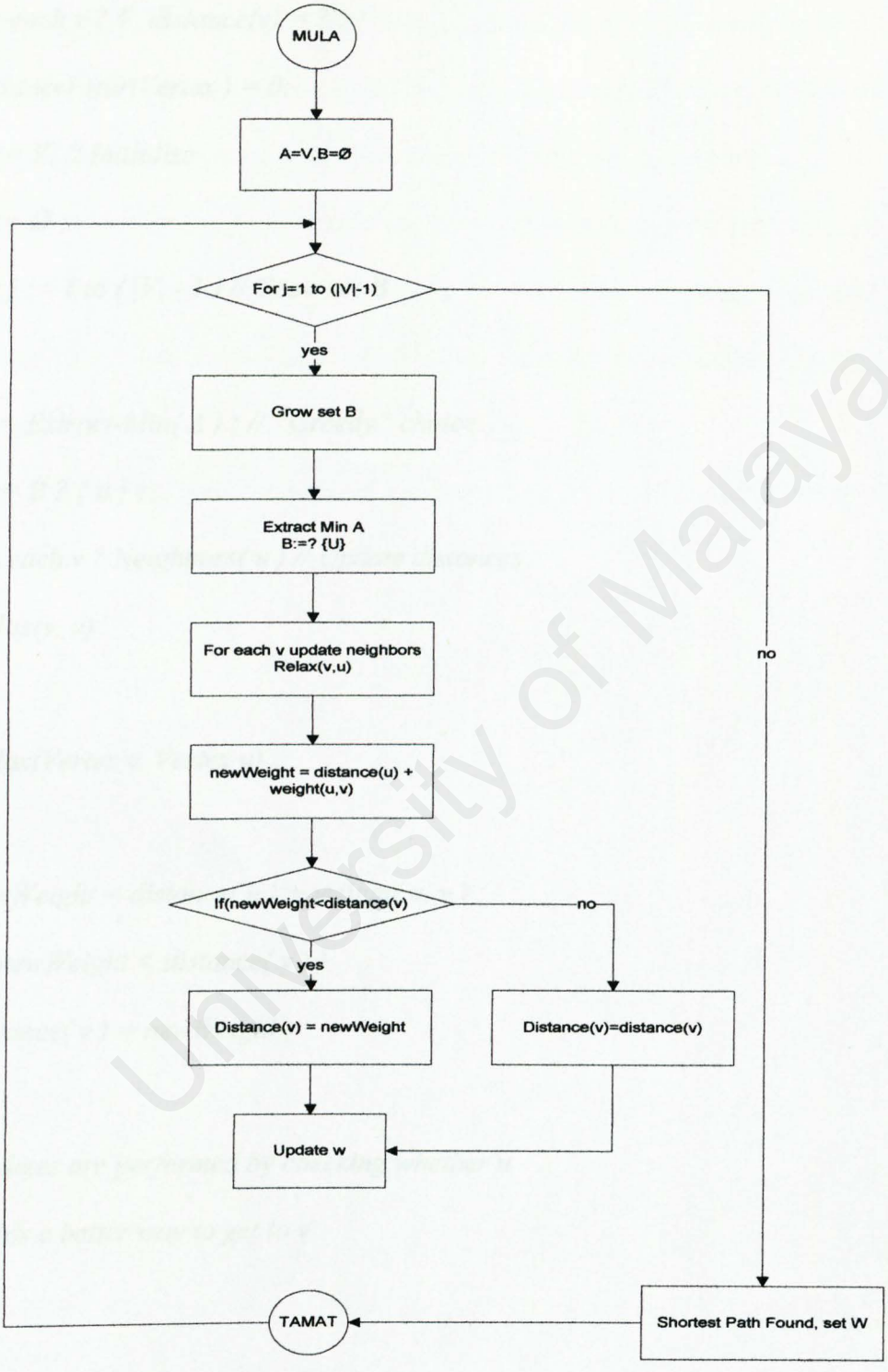
5.1 Perwakilan Jarak antara Dua Nod Bersebelahan.



Rajah 5.1 Perwakilan untuk kiraan jarak antara nod

Seperti *Rajah 5.1 Perwakilan untuk kiraan jarak antara nod* di atas, nilai jarak antara dua nod ditetapkan kepada 10(melintang dan mendatar) dan 14(condong). Nilai awal ini ditetapkan untuk saiz tatasusunan dalam pengaturcaraan. Jarak antara dua nod bersebelahan ditetapkan seperti *Rajah 5.1* Ini bagi memudahkan proses pembangunan aturcara simulasi rumus dijkstra. Nilai yang sekata diperlukan untuk tatasusunan. Selain itu ia menepati ciri-ciri rumus dijkstra yang memerlukan arcs/edges/jarak antara dua nod diketahui sebelum boleh menggunakan rumus dijkstra.

5.2 Carta Alir Rumus Dijkstra



Rajah 5.3 Carta Alir Rumus Dijkstra

5.3 Pseudocode

Darripada *Rajah 5.3 Carta Alir Rumus Dijkstra* dapat diterangkan pseudocode berikut:-

for each v ? V, distance(v) = ? ;

distance(startVertex) = 0;

A := V; // Initialize

B := Ø ;

for j := 1 to (|V| - 1) // Grow set B

{

u := Extract-Min(A) ; // "Greedy" choice

B := B ? { u } ;

for each v ? Neighbors(u) // Update distances

Relax(v, u)

}

Relax(Vertex v, Vertex u)

{

newWeight = distance(u) + weight(u, v) ;

if (newWeight < distance(v))

distance(v) = newWeight ;

}

Updates are performed by checking whether u

yields a better way to get to v

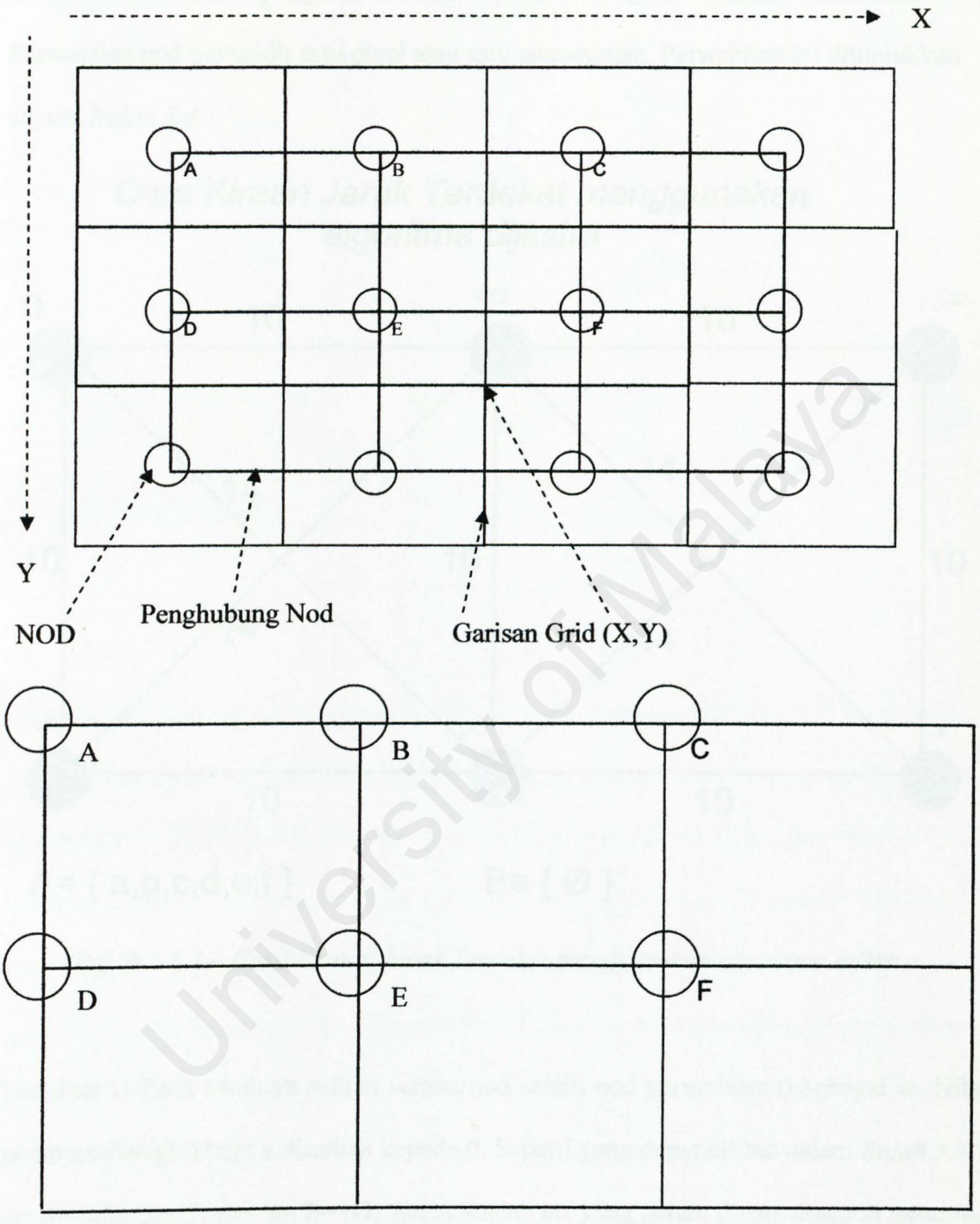
Pseudocode(Kod Pseudo) ini adalah penerangan kepada *Rajah 5.3 Carta Alir Rumus Dijkstra* . Pseudocode/Carta Alir ini amat berguna dalam pembangunan aplikasi dan pengaturcaraan simulasi algoritma dijkstra yang akan dibangunkan.

Carta alir dan *pseudocode(*kod-pseudo) yang diberikan adalah dalam Bahasa Inggeris kerana ia adalah bahasa rasmi dalam bahasa pengaturcaraan.

Carta alir dan *pseudocode* ini memberi gambaran awal tentang bagamana proses pengkodan dilakukan. Pelbagai aspek perlu ditekankan. Proses pengaturcaraan C akan dibangunkan dan dijalankan dalam bab akan datang dalam Latihan Ilmiah Tahap Akhir WXES3182.

Penerangan tentang Carta alir dan *Pseudocode* diterangkan dalam bahagian 5.6 iaitu Cara Pengiraan.

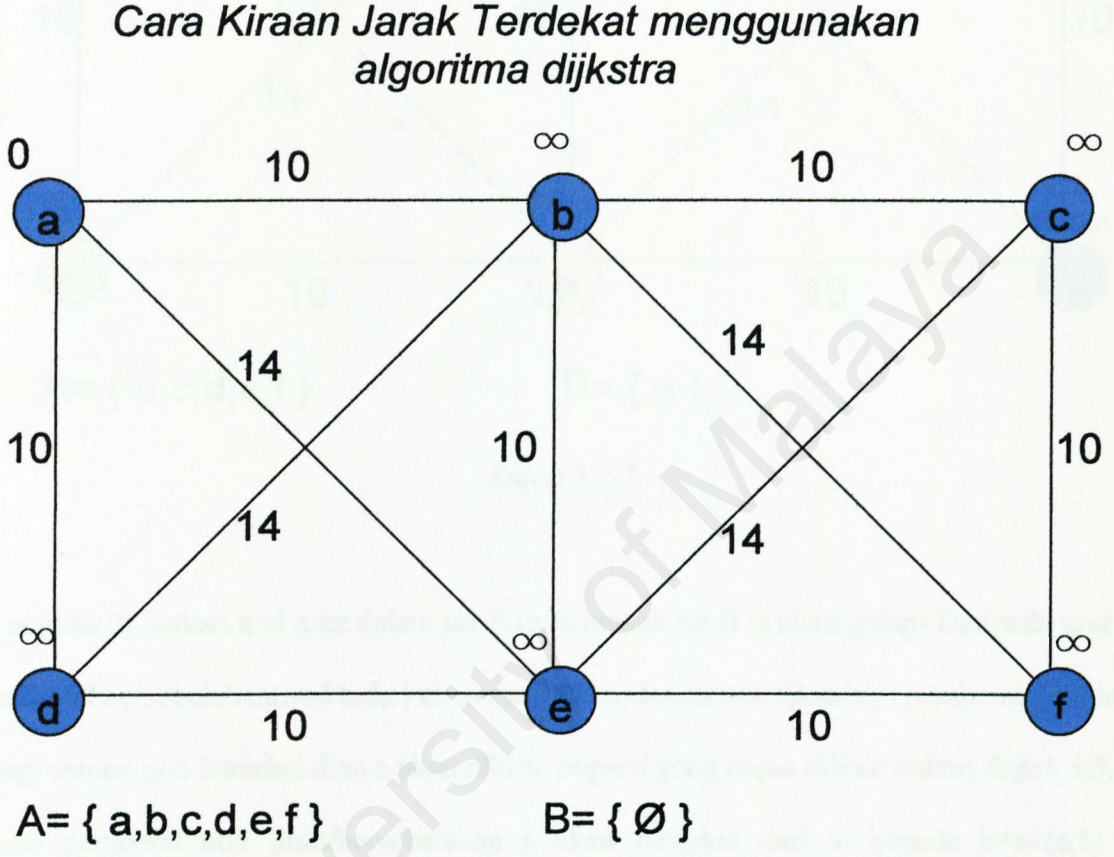
5.4 Contoh Nod dan hubungannya dengan Grid.



Rajah 5.4 Perwakilan Nod dengan Grid

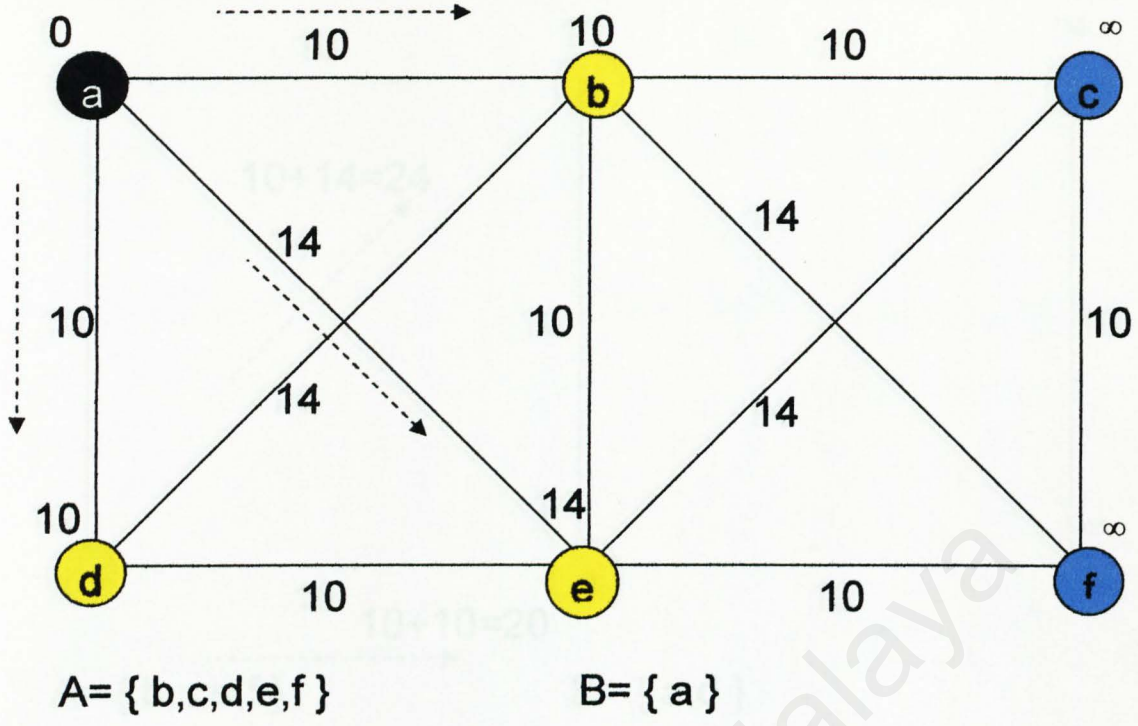
5.5 Cara Pengiraan

Berikut adalah contoh pengiraan mencari *Shortest Path*(jarak terdekat) antara nod. Perwakilan nod mewakili satu pixel atau satu tatasusunan. Perwakilan ini ditunjukkan seperti *Rajah 5.4.*



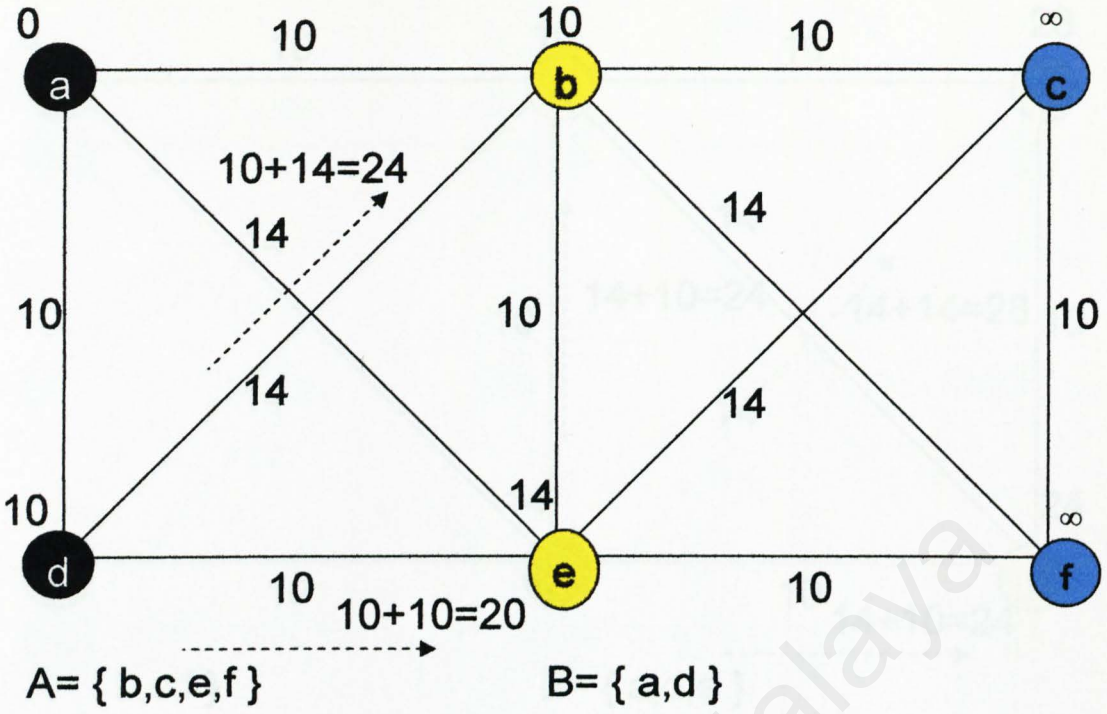
Rajah 5.5.1 – Cara Kiraan Jarak Terdekat menggunakan algoritma dijkstra

Langkah 1: Pada awalnya setkan semua nod selain nod permulaan(a) sebagai ∞. Nilai pemberat(weight) bagi a disetkan kepada 0. Seperti yang dapat dilihat dalam *Rajah 5.5.1* set A={a,b,c,d,e,f} dan set B={Ø}.Set A adalah set yang belum diusik ataupun *unsettled* manakala set B adalah set yang selah selesai *settled.* Seperti yang dapat dilihat dalam rajah di atas, pada permulaannya semua nod adalah dalam set A dan tiada dalam set B.



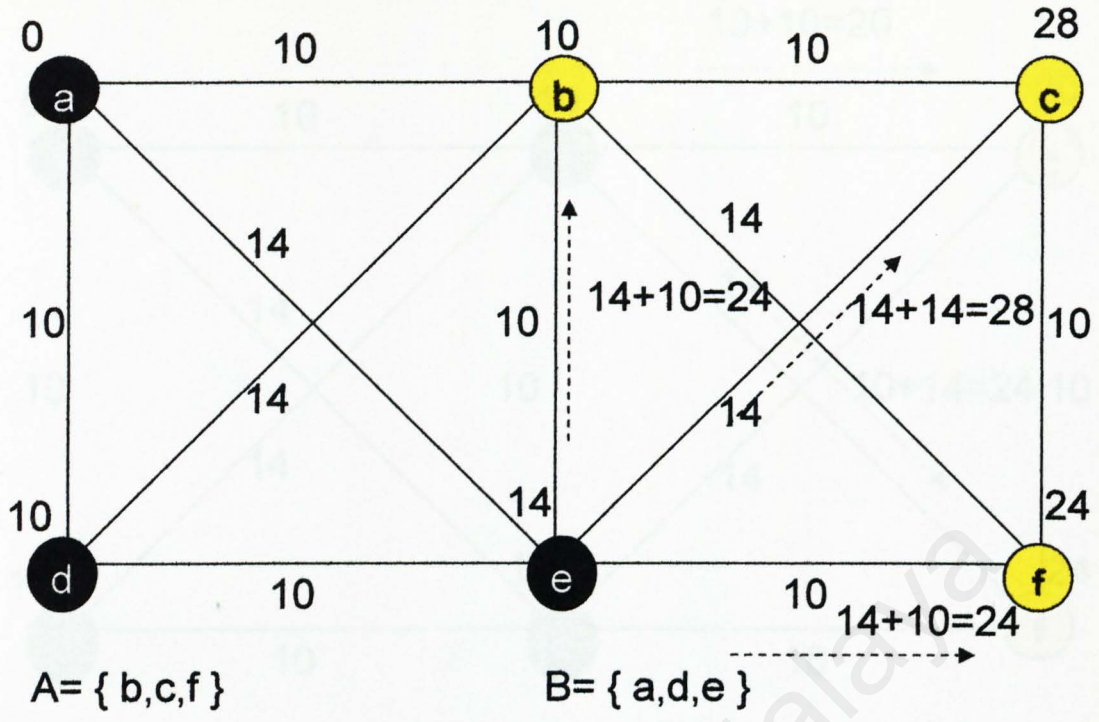
Rajah 5.5.2

Langkah 2: Setkan nod **a** ke dalam set B.(bila masuk set B ia akan gelap) Daripada nod a nod-nod bersebelahan(nod b,d,e) akan menjadi nod sementara(kuning) pemberat(weight) bagi semua nod bersebelahan a akan dikira. Seperti yang dapat dilihat dalam *Rajah 5.5.2* nilai pemberat nod jiran/bersebelahan a akan bertukar dari ∞ kepada b=a+(a,b) , d=a+(a,d) dan e=a+(a,e). Nilai pemberat bagi nod b,d dan e akan di kemasmini(*update*). Seperti yang dapat dilihat dalam *Rajah 5.5.2* nilai nod a diekstrak keluar dari A ke dalam set B.Sekarang nilai pemberat bagi nod b ialah 10, pemberat nod d 10 dan pemberat nod e adalah 14. Nilai 10 dan 14 diantara dua nod digunakan seperti penerangan di *Rajah 5.1 Perwakilan untuk kiraan jarak antara nod.*



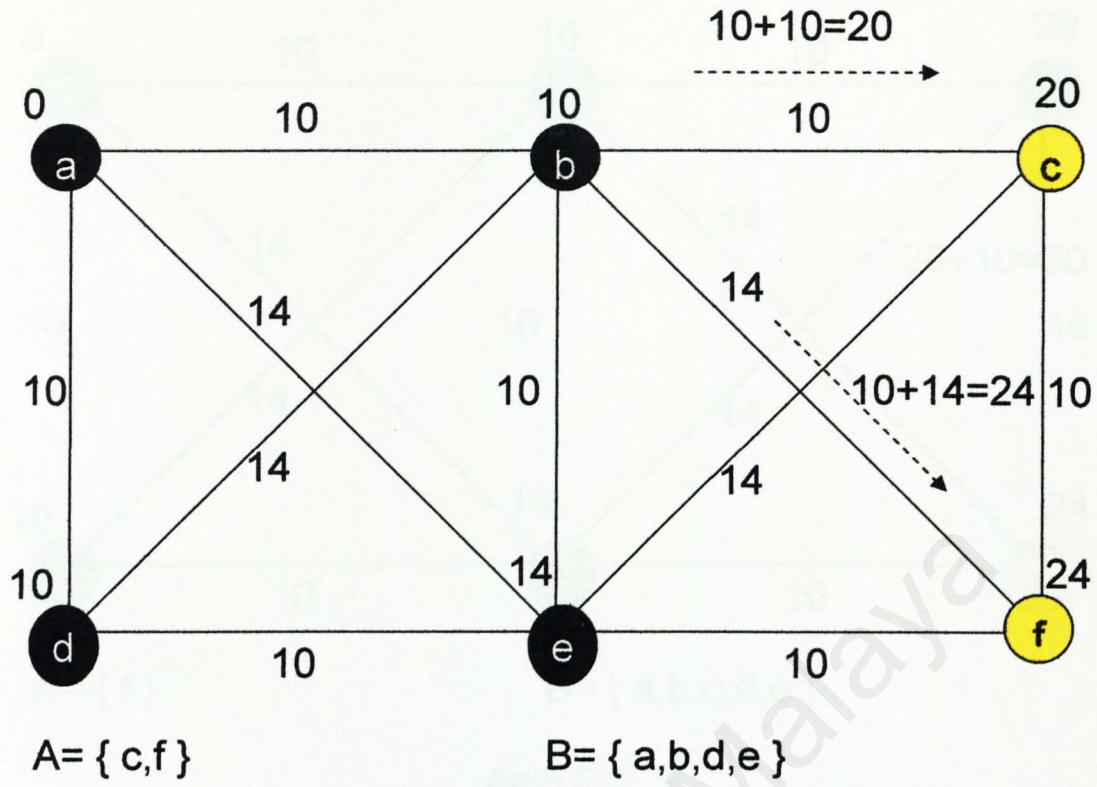
Rajah 5.5.3

Langkah 3: Seperti yang dapat dilihat dalam *Rajah 5.5.3* nod **d** diekstrak dan dimasukkan ke dalam set B. Seperti juga langkah 2, nod-nod jiran/bersebelahan dengan nod d iaitu nod b dan e ditambah nilai pemberatnya. Seperti yang dapat dilihat dalam *Rajah 5.5.3* walaupun nod a adalah jiran bagi nod d, tetapi ia tidak akan dipedulikan. Nilai pemberat bagi nod a akan kekal 0. Ini adalah kerana nod a telah masuk ke dalam senarai set B(*settled*) Ini bermakna set d, akan hanya mencari dan menambah nilai pemberatnya dengan jaraknya antara nod yang bukan berada dalam senarai set B. Set B boleh juga dikenali sebagai *Close list* manakala set A boleh dikenali sebagai *Open List.* Nilai pemberat set b tetap 10 walaupun w(d)+14=24. Ini adalah kerana menurut rumus dijkstra nilai minimum D(v)=min {D(v) , D(w)+l(w,v)}akan dipilih. Ini adalah sama di nod e, nilai pemberat akan kekal 14.



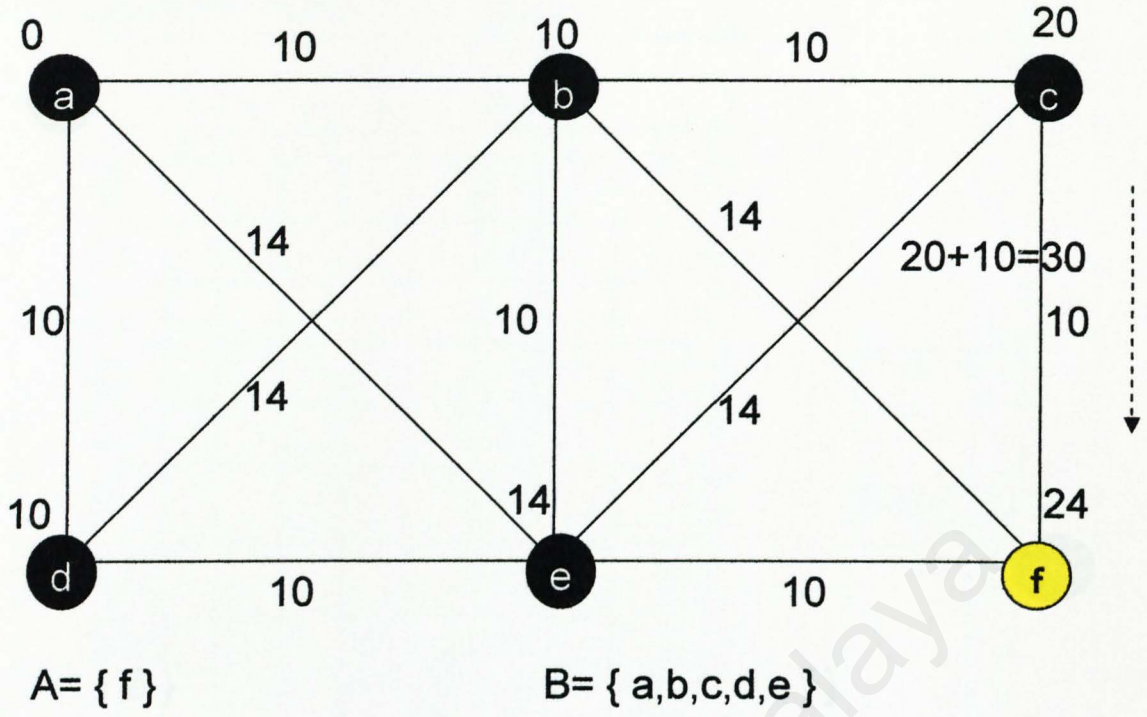
Rajah 5.5.4

Langkah 4: Seperti *Rajah 5.5.4* sekarang, nod **e** diekstrak dari set A dan dimasukkan ke dalam *close list* set B. Nod jiran/bersebelahan kepada nod e iaitu nod b,c dan no f akan ditambah pemberatnya dengan nilai arcs/edges/jarak dari nod e. Ia seperti menganggap nod e sebagai titik mula(starting node) dan jirannya sebagai destinasi. Dengan ini pengiraan pemberat b=pemberat e+(e,b)→14+10=24. Walaubagaimanapun nilai 24 ini akan dibandingkan dengan nilai pemberat sedia ada di nod b iaitu 10.Nilai pemberat no b iaitu 10 didapati daripada langkah 2. D(v)=min {D(v) , D(w)+l(w,v)}Ia akan memilih 10 sebagai pemberat kerana nilainya lebih kecil daripada 24.Maka nilai pemberat(weight) bagi nod b=10. Nilai pemberat bagi nod c pula adalah w(e)+l(e,c) → 14+14=28. Nilai pemberat bagi nod f pula ialah w(e)+l(e,f)→14+10=24.



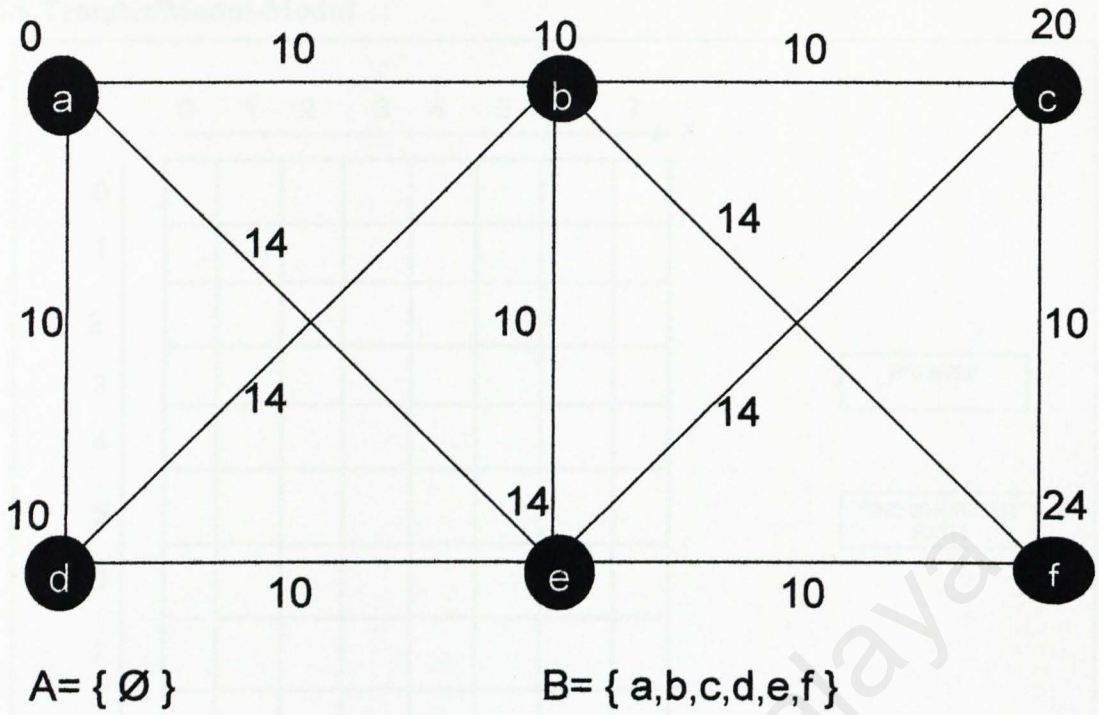
Rajah 5.5.5

Langkah 5: Seperti *Rajah 5.5.5* nod **b** diekstrak daripada set A dan dimasukkan ke dalam set B.Nilai pemberat bagi jiran nod b dikira. Jiran bersebelahannya nod c dikira pemberatnya. Pemberat bagi nod c adalah w(b)+l(b,c)=10+10=20. Nilai ini akan dibandingkan dengan nilai pemberat nod c yang sedia ada iaitu 28. Berdasarkan rumus dijkstra D(v)=min {D(v) , D(w)+l(w,v)}nilai terkecil akan dipilih.Oleh itu nilai pemberat baru bagi nod c adalah 20. Pemberat bagi nod f juga dikira. Pemberat bagi nod f ialah w(b)+l(b,f)→10+14=24. Nilai ini akan dibandingkan dengan nilai asal iaitu 24. Olehkerana ia mempunyai nilai yang sama, nilai pemberat bagi f akan kekal sebagai 24.



Rajah 5.5.6

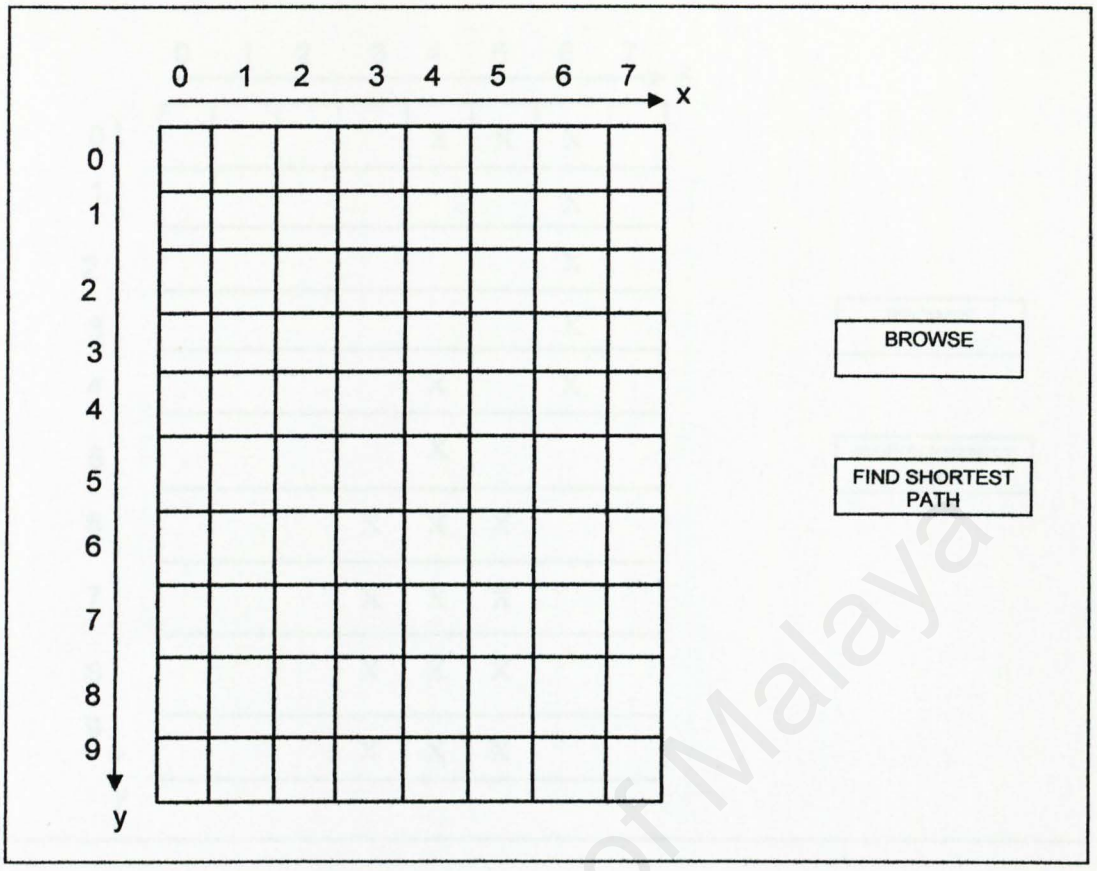
Langkah 6: Seperti *Rajah 5.5.6* nod c akan diekstrak daripada set A dan dimasukkan ke dalam set B.Seperti yang dilihat dar langkah 1-5, setiap nod bermula daripada nod permulaan akan mengekstrak nilai nod bersebelahannya. Nod-nod jiran/bersebelahan ini akan menjadi titik mula ke destinasi titik bersebelahannya pula. Seperti dalam *Rajah 5.5.6* jiran bagi nod c iaitu nod f akan ditambah nilai pemberatnya. Hanya tinggal nod f sahaja dalam senarai set A(*Open list/Unsettled*) Nilai pemberat bagi nod f adalah w(c)+(c,f)→20+10=30.Nilai ini akan dibandigkan dengan nilai pemberat asal iaitu 24. Berdasarkan rumus dijkstra D(v)=min {D(v) , D(w)+l(w,v)}, nilai 24 di ambil kerana 24<30. Olehitu nilai pemberat bagi nod f akan disetkan kepada 24.



Rajah 5.5.7

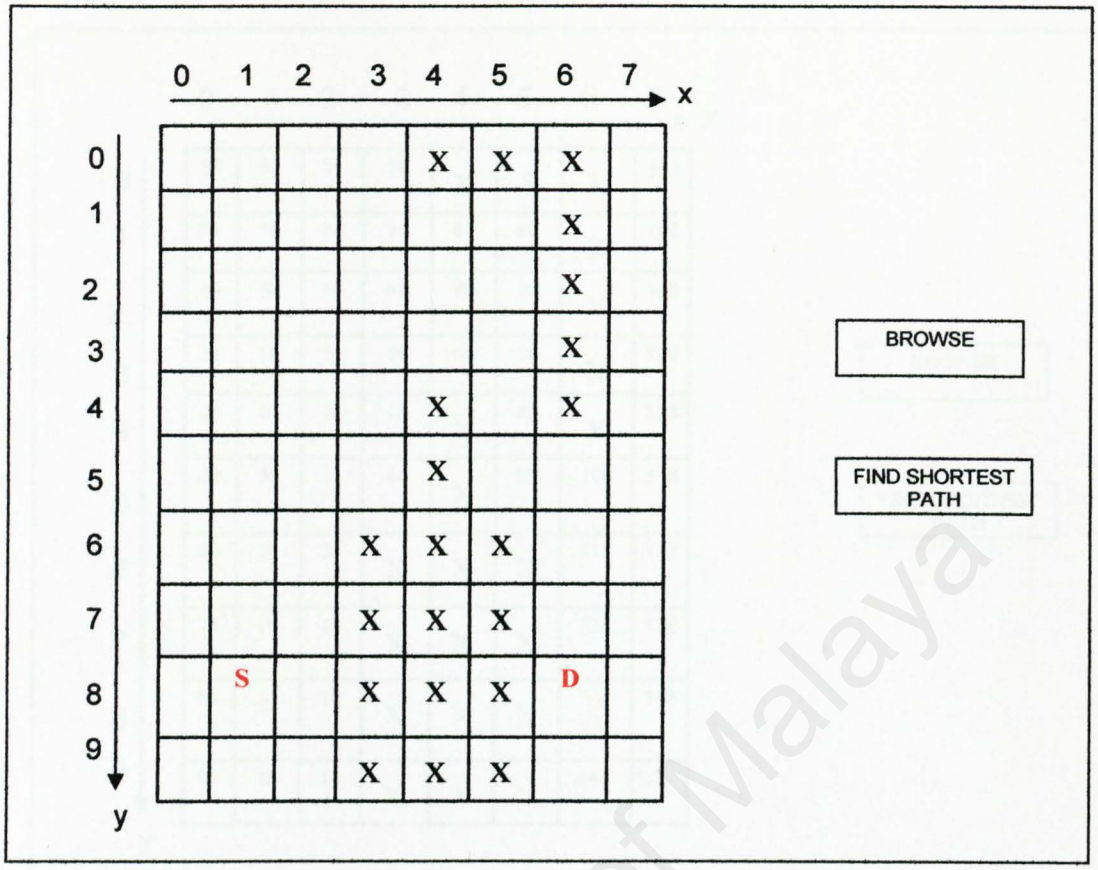
Langkah 7/Kesimpulan pengiraan: Setelah semua nod diekstrak daripada set A(*unsettled/open list*), dan dimasukkan ke dalam set B(*settled/closem list*) dan semua nilai pemberat badi semua nod dikira dan dikemaskini berdasarkan rumus dijkstra D(v)=min {D(v) , D(w)+l(w,v)}. *Shortest Path* ataupun jarak terdekat antara nod a dan nod-nod lain dalam set A telah ditemui. Nilai pemberat bagi setiap nod adalah nilai bagi jarak terdekat antara nod tersebut dengan nod permulaan iaitu nod a. Cara yang sama digunakan untuk nod-nod lain. Penerangan menggunakan 6 nod ini sama sahaja jikalau nod-nod yang banyak digunakan. Algoritma Dijkstra berhenti sebaik sahaja nilai pemberat bagi setiap nod dikemaskini. Penerangan ini juga mewakili carta alir dan *pseudocode*(kod pseudo) yang diberikan dalam bahagian sebelum ini.

5.6 Templet/Modul-Modul



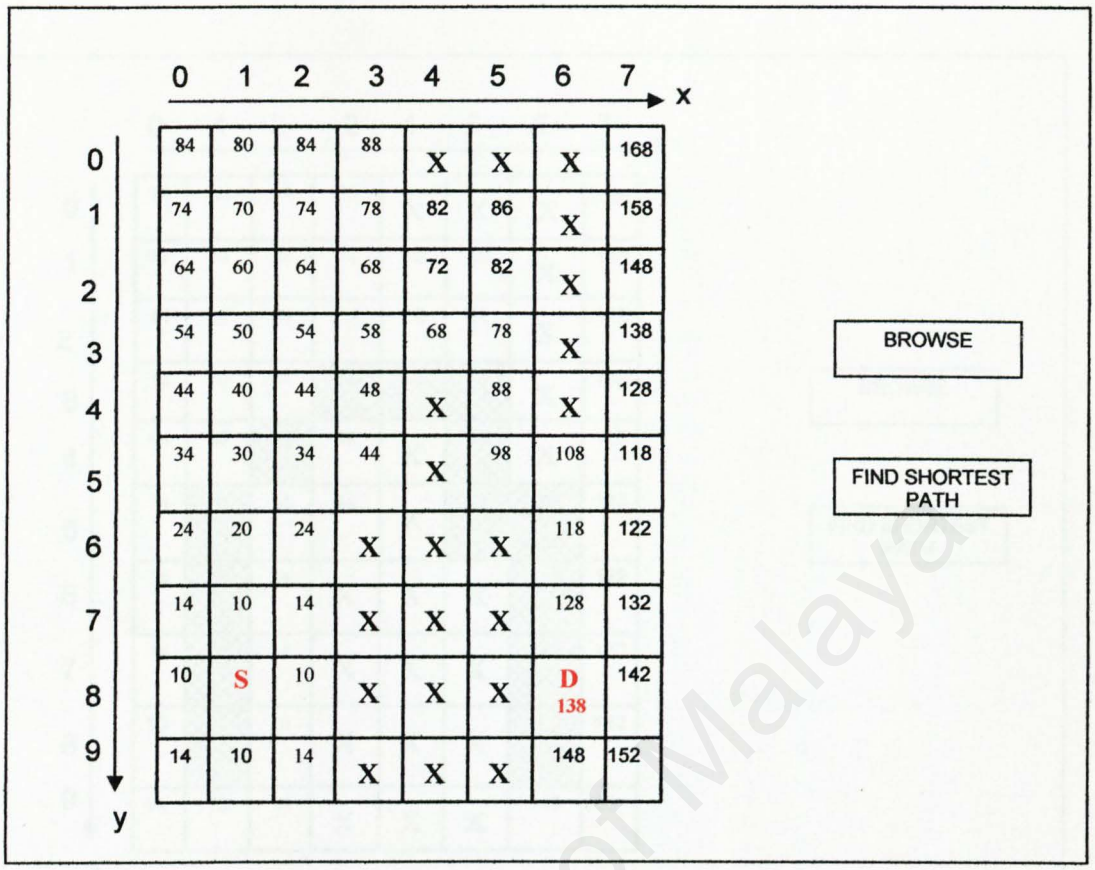
Rajah 5.6 Contoh Template Kosong

Lagent:	
Butang Browse:	Membuka tamplate yang sedia ada
Butang Find Shortest Path:	Mencari jarak terdekat antara dua nod berdasarkan sumber dan destinasi yang diberikan dalam tamplate yang telah disediakan.



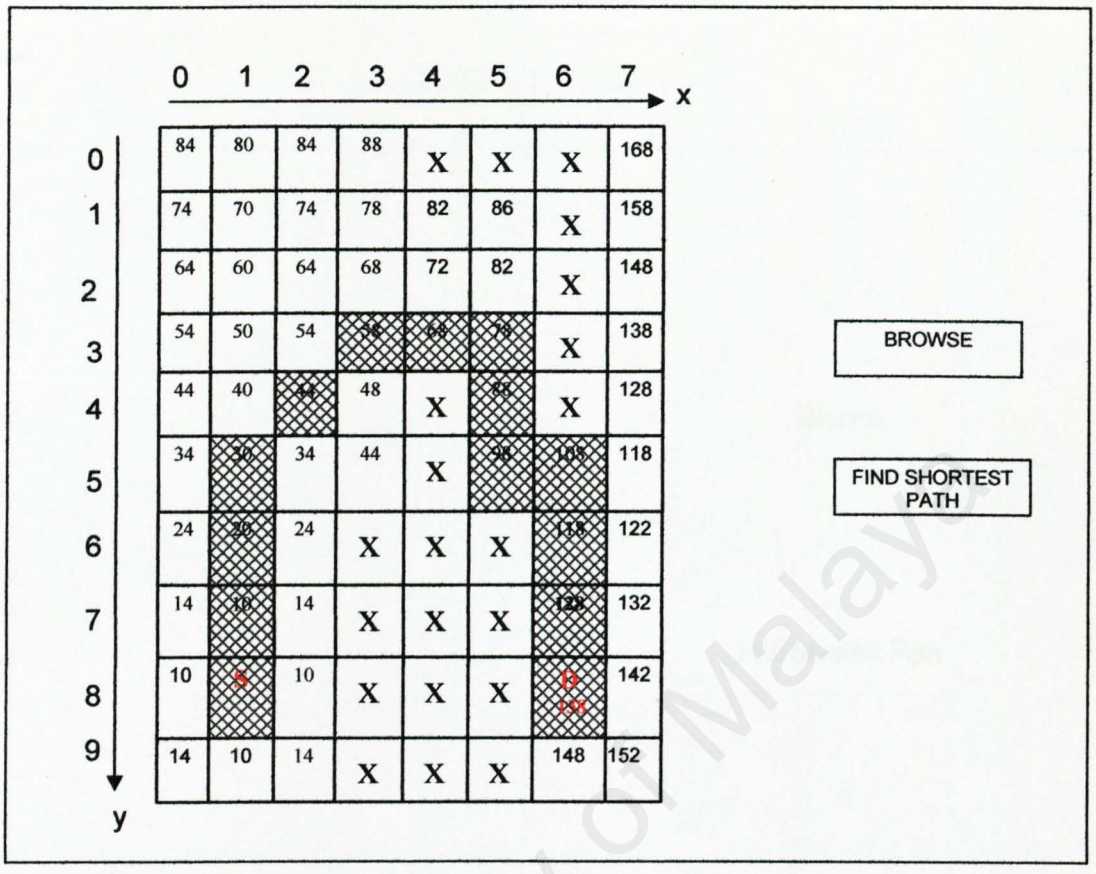
Simbol	Makna
X	Halangan, spt dinding
S	Titik @ Nilai awal
D	Destinasi @ nod sasaran
10/nilai	Nilai Pemberat=Weight bagi nod

Rajah 5.7 Contoh Template Dengan Nilai S,D dan Halangan(dinding/wall)



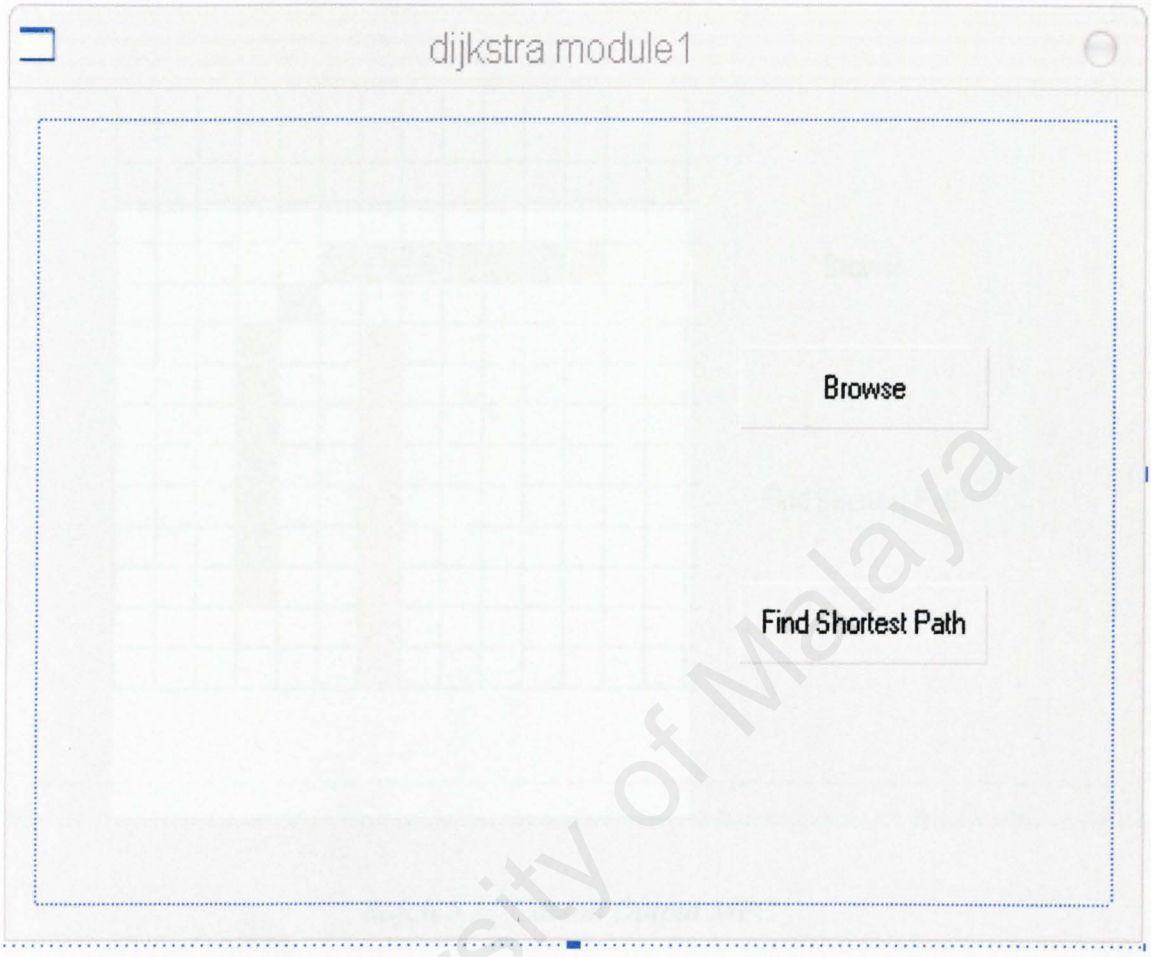
Simbol	Makna
X	Halangan, spt dinding
S	Titik @ Nilai awal
D	Destinasi @ nod sasaran
10/nilai	Nilai Pemberat=Weight bagi nod

Rajah 5.8 Contoh Template Selepas Jarak Terdekat Dikira

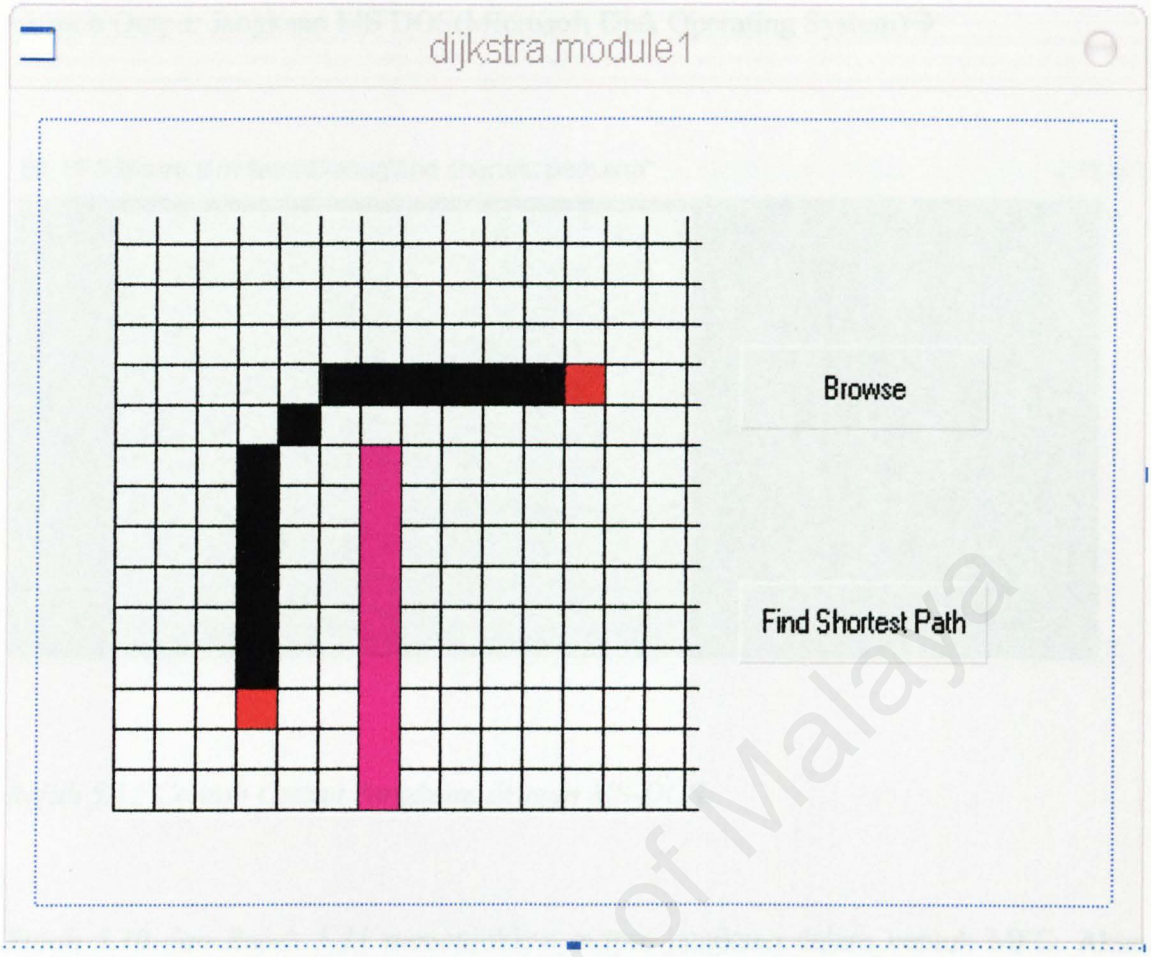


Rajah 5.9 Shortest Path Found

5.7Contoh Output Jangkaan

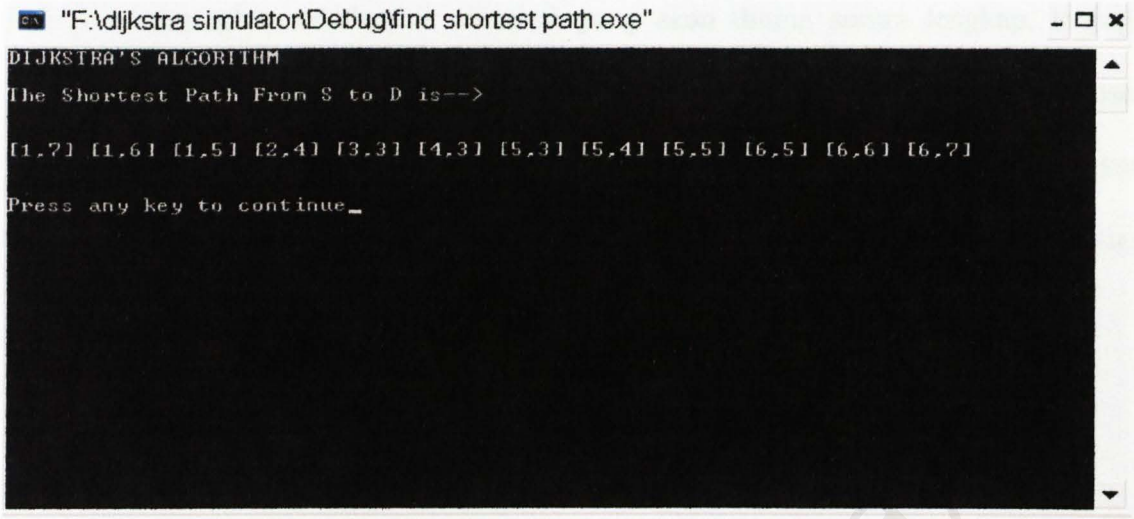


Rajah 5.10 Contoh Output Microsoft Foundation Class Library (MFC)



Rajah 5.11 Contoh Output MFC

Contoh Output Jangkaan MS DOS(Microsoft Disk Operating System)→



Rajah 5.12 Contoh Output Jangkaan dengan MS-DOS.

Rajah 5.10 dan Rajah 5.11 menunjukkan output jangkaan dalam bentuk MFC. Akan tetapi, output sasaran utama bagi simulasi yang akan dibangunkan adalah dengan MS-DOS sahaja. Seperti *Rajah 5.12* di atas, output jangkaan adalah dalam bentuk tatasusunan(*array*) . Output jangkaan yang menggunakan MS-DOS dipaparkan. Output dan paparan MS-DOS digunakan walaupun ia kurang mesra pengguna(*user-friendly*) kerana hasil daripada simulasi ini akan dibaca oleh modul *Chain Code* dan bukannya pengguna biasa. *Chain Code* seperti ditunjukkan dalam *Rajah 2.1 Sistem Automasi Meletak Kenderaan* akan menyahcode(decode) hasil daripada simulasi kepada nilai jarak dan arah. *Chain Code* akan hanya membaca koordinat dalam tatasusunan jarak terdekat yang diperoleh.

5.9 Ringkasan

Bab 5 menerangkan rekabentuk simulasi yang akan dibina secara lengkap. Ia juga menunjukkan carta alir dan kod-pseudo(*pseudocode*) yang akan digunakan dalam proses pembangunan dan aturcara. Cara pengiraan menggunakan rumus dijkstra diberikan dalam bentuk jadual untuk menerangkan carta alir dan kod-pseudo.Bab ini juga mengandungi output-output yang dijangka dijana setelah simulasi berjaya dihasilkan.

BAB 6 :

PERLAKSANAAN SIMULASI

BAB 6 : PERLAKSANAAN SIMULASI

6.1 Pengenalan

Dalam fasa Pelaksanaan, **Simulasi Jarak Terdekat Menggunakan Algoritma Dijkstra(*Simulation of Shortest Path Using Dijkstra's Algorithm*)** yang dicadangkan dibangunkan. Fasa ini direalisasikan selepas proses menganalisa dan rekabentuk sistem berjaya dilakukan. Akan tetapi terdapat beberapa perubahan dan pengubahsuaian dalam fasa implimentasi ini. Sebahagian daripada rekabentuk diubah berdasarkan kesesuaian simulasi. Konsep asal masih digunakan tetapi ia ditambah dengan beberapa fungsi atau model yang lebih baik.

Pembangunan dimulakan dengan menghasilkan modul-modul secara berasingan menggunakan pengaturcara bahasa C. Ianya menggunakan platform Microsoft Visual C++ 6.0. Tiga fail dibuat iaitu sp.c , pq.c dan sp.h dibuat.

Seterusnya, didalam fasa ini juga segala langkah dan strategi pembangunan akan dibincangkan bagi memastikan kelancaran pembangunan simulasi. Dalam fasa ini, aktiviti yang paling terpenting adalah pengaturcaraan C dan juga Struktur Data(Data Structure). Ini adalah simulasi ini adalah 100% daripada bahasa pengaturcaraan. Seperti rekabentuk,output adalah dalam bentuk DOS. Output jarak terdekat di dalam notepad/ fail yang bernama "output.txt" juga dihasilkan. Ini adalah kerana simulasi ini akan diintegrasi dengan sistem meletak kenderaan. Simulasi ini akan berhubungkait dengan Chain Code. Ouput dari simulasi ini akan digunakan oleh Chain Code dan diterjemah

kepada pergerakan sebelum membolehkan robot simusi bergerak dari titik awal ke titik akhir menggunakan jalan terpantas/terpendek.

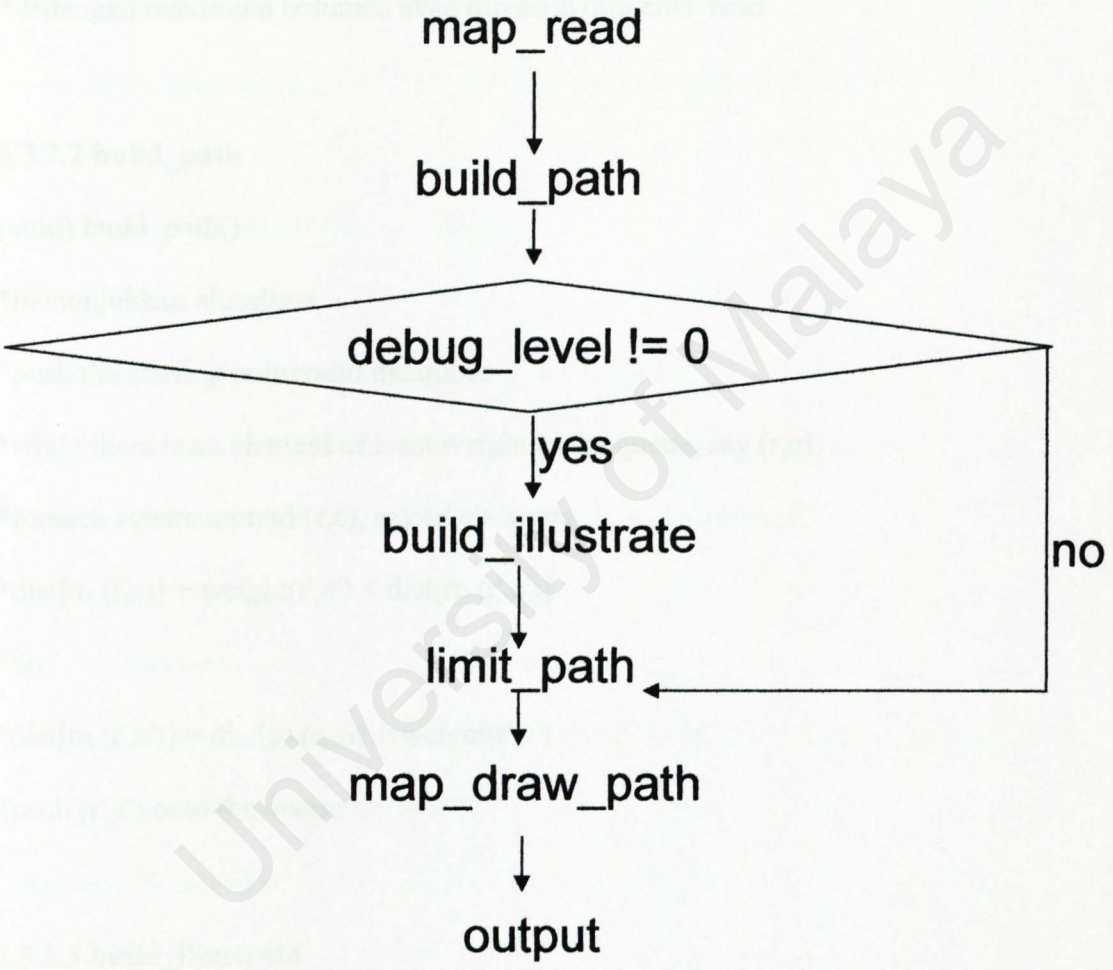
Dalam peringkat ini, segalanya adalah bergantung kepada pengaturcaraan C. Ianya memerlukan ketekunan,ketelitian dan kesabaran yang tinggi. Proses try and error banyak dilakukan dimana, program di tulis menggunakan kefahaman dan keparan sedia ada. Kemudian satu per satu fungsi ditambah,diubah atau dikemaskini. Dalam masa yang sama rujukan dibuat bagi mengkaji cara dan teknik yang bersesuaian dengan simulasi. Rujukan berkenaan bahasa pengaturcaraan C dan data struktur menggunakan C juga dibuat untuk memastikan coding yang digunakan betul dan mengikuti piwaian. Kesilapan dan sebarang ralat akan mengakibatkan lengahan dalam pembangunan simulasi.

6.2 Perisian Pengaturcaraan

Persian Microsoft Visual C++ 6.0 digunakan dalam membangunkan simulasi ini. Walaubagaimanapun, bahasa aturcara C dipilih dan digunakan dalam membangunkan dan melaksanakan Simulasi Jarak Terdekat Menggunakan Algoritma Dijkstra ini. Bahasa Pengaturcaraan C ini boleh di kompil di dalam Microsoft C++ menjadikannya satu kelebihan kerana ia serasi dengan persekitaran ini.

6.3 Pengaturcaraan Sistem

6.3.1 Struktur Aturcara



Rajah 6.1 Flow Aturcara

6.3.2 Fungsi-fungsi yang digunakan

6.3.2.1 map_read

* map_read(nama_fail)

* Baca nama_fail/file_name dan save info dlm global map structure.

* Selepas baca map file, initialize dist array kepada INFINITY dan parent array kpd -1.

* Bilangan rows yg yg di read akan direkod dlm rows_read

* Bilangan maximum columns akan direkord dlm cols_read

6.3.2.2 build_path

(void) build_path()

*menunjukkan algoritma

*push the starting point onto the queue

*while there is an element of least weight in the queue, say (r,c)

*foreach square around (r,c), say (r',c') where

*dist[to (r,c)] + weight(r',c') < dist[to (r',c')]

*do

*dist[to (r',c')] = dist[to (r,c)] + weight(r',c')

*push (r',c') onto the queue

6.3.2.3 build_illustrate

(void) build_illustrate();

Menunjukkan weights utk semua shortest paths.

6.3.2.4 limit_path

void limit_path(void)

Ia akan mensetkan parent untuk semua titik yang bukan berada dalam jalan shortest path yg dipilih.

6.3.2.5 map_draw_path

Ia akan dipanggil selepas limit_path() dipanggil, jikalau tidak user akan ditunjukkan semua path dari starting point. map_draw_path akan menunjukkan:-->

'^' -->titik mula(starting point)

'$' -->titik tamat(ending point)

'+' -->jalan terdekat(intermediate squares in the path)

'.' -->jalan lain yg boleh dilalui(the walkable squares in the map)

#-->jalan yg mustahil/dinding.halangan(the impassible squares)

6.3.3 Algorithma

Algoritma Dijkstra digunakan dalam membangunkan simulasi C ini. Algoritma yang dibangunkan terdapat dalam fungsi build_path.

```
void build_path(void)
{
PQUEUE * pq;
int     row, col;
int       i;
```

```
pq = pqueue_new();
if (startr < 0 || startr >= rows_read || startc < 0 || startc >= cols_read)
    return;
dist[startr][startc] = 0;
parent[startr][startc][0] = -1;
parent[startr][startc][1] = -1;
pqueue_insert(pq, startr, startc, 0);

while (pqueue_popmin(pq, &row, &col) == 0) {
    /* DEBUG INFORMATION */
    pops++;
    if (debug_level > 0)
      printf("\nSelected (row,col)=(%d,%d). Adding:\n", row, col);

    /* FOREACH square adjacent to (row,col), call it drow(row,i), dcol(col,i) */
    for (i = 0; i < 8; i++) {
      /* Check to make sure the square is on the map */
      if (drow(row,i) < 0 || drow(row,i) >= rows_read ||
          dcol(col,i) < 0 || dcol(col,i) >= cols_read)
          continue;

      /* Semak samada untuk lihat samaada shorters path dijumpai*/
      if (dist[row][col] + map[drow(row,i)][dcol(col,i)] <
          dist[drow(row,i)][dcol(col,i)]) {
```

```
                /* DEBUG INFORMATION */
                if (debug_level > 0) {
                    printf("\t\t(%d,%d) Weight was/is: ", drow(row,i), dcol(col,i));
                printf("%d/%d\n", dist[drow(row,i)][dcol(col,i)],
map[drow(row,i)][dcol(col,i)] + dist[row][col]);
                }

                /* We have a shorter path, update the information */
                dist[drow(row,i)][dcol(col,i)] = map[drow(row,i)][dcol(col,i)] +
dist[row][col];
                parent[drow(row,i)][dcol(col,i)][0] = row;
                parent[drow(row,i)][dcol(col,i)][1] = col;
                /* Push the modified square onto the priority queue */
                pqueue_insert(pq, drow(row,i), dcol(col,i),
dist[drow(row,i)][dcol(col,i)]);
            } else if (debug_level >= 5) {
                printf("\t\t(%d,%d) Weight Stays: %d (would have been %d)\n",
drow(row,i), dcol(col,i), dist[drow(row,i)][dcol(col,i)], dist[row][col] +
map[drow(row,i)][dcol(col,i)]);
            }
        }
        if (debug_level >= 10) build_illustrate();
    }
    return; }
```

6.3.4

Coding untuk 8titik keliling satu titik→

```
#define drow(row,i) (row+delta[i][0])

#define dcol(col,i) (col+delta[i][1])

int delta[8][2] = { -1, -1, -1, 1, 1, -1, 1, 1, -1, 0, 0, 1, 1, 0, 0, -1 };

//ini adalah array yang menunjukkan 8 titik bersebelahan dengan satu titik.
```

1 [-1,1]	5 [-1,0]	2 [-1,1]
8 [0,-1]	(Row,Coloum) [0,0]	6 [0,1]
3 [1,-1]	7 [1,0]	4 [1,1]

6.3.5 How to read [row,column]

Column1 → **Column5**

[0,0], [0,1],[0,2],[0,3],[0,4],[0,5] – ROW 0

[1,0], [1,1],[1,2],[1,3],[1,4],[1,5] – ROW 1

[2.0], [2,1],[2,2],[2,3],[2,4],[2,5] – ROW 2

[3,0], [3,1],[3,2],[3,3],[3,4],[3,5] – ROW 3

[4,0], [4,1],[4,2],[4,3],[4,4],[4,5] – ROW 4

######

#1111#

#1111#

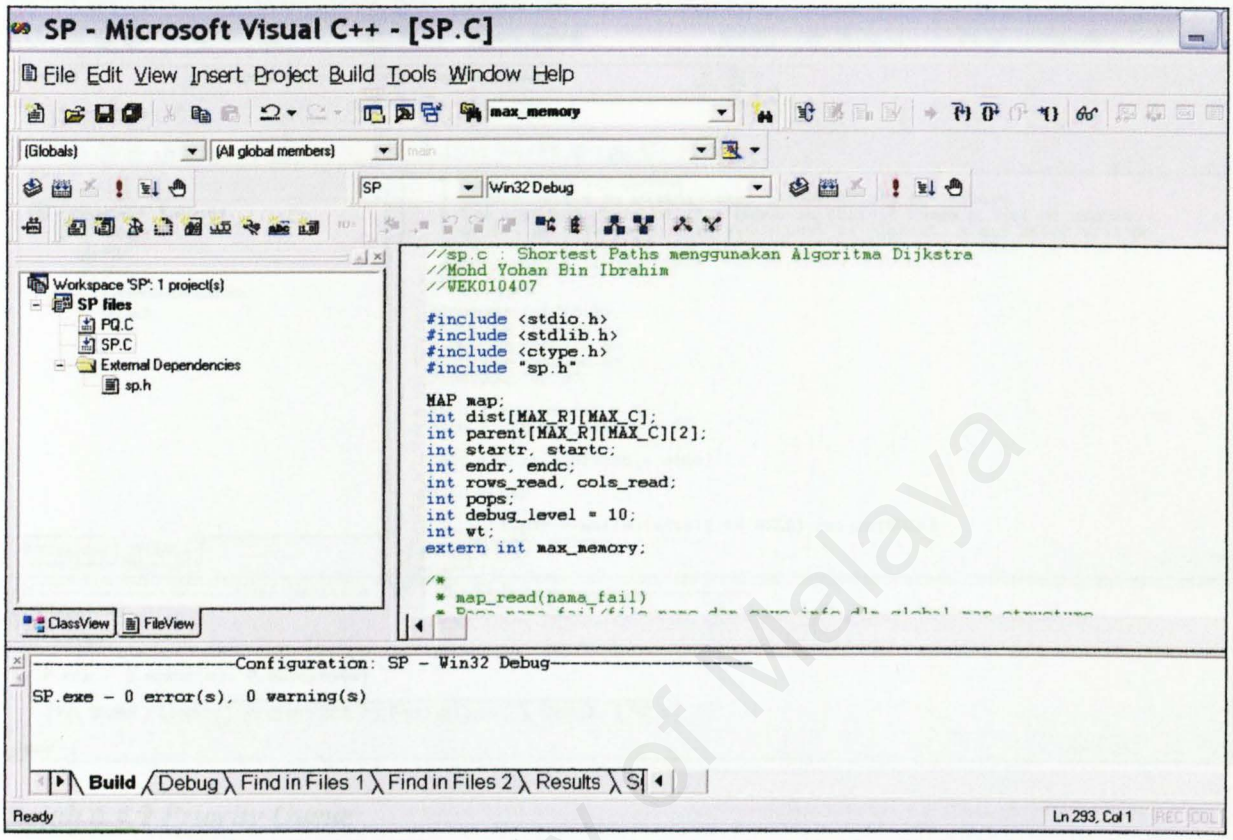
#1111#

######

Daripada map diatas, semua column 0 dan 5 adalah dinding dan tiada nilai pemberat.

6.4 Dokumentasi

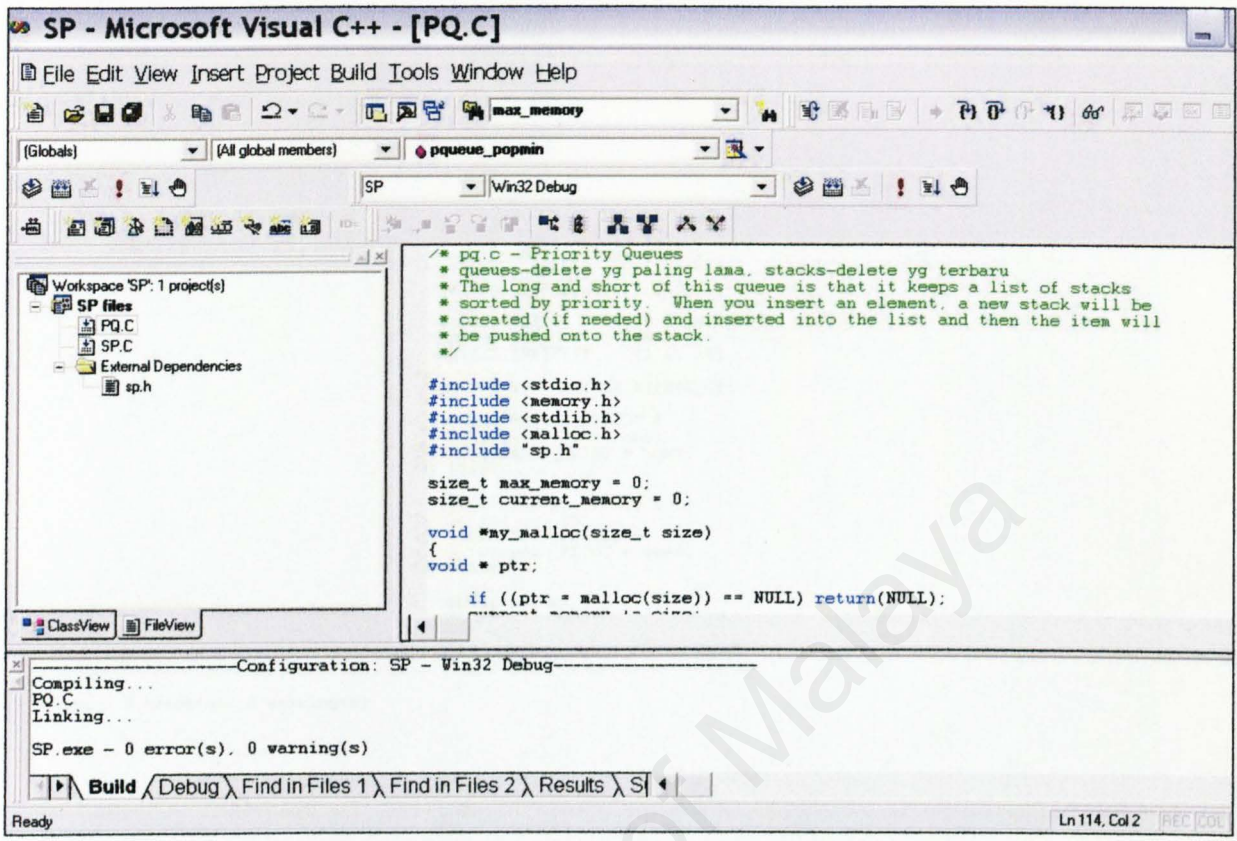
6.4.1 sp.c



Rajah 6.4.1

Fungsi-fungsi seperti yang terdapat di dalam dalam Struktur Aturcara.

6.4.2 pq.c



Rajah 6.4.2 Priority Queue

Fail ini mengandungi fungsi dan data yang bernama

i) my_malloc – memory allocation, turut mengira nilai max_memory

ii) my_free

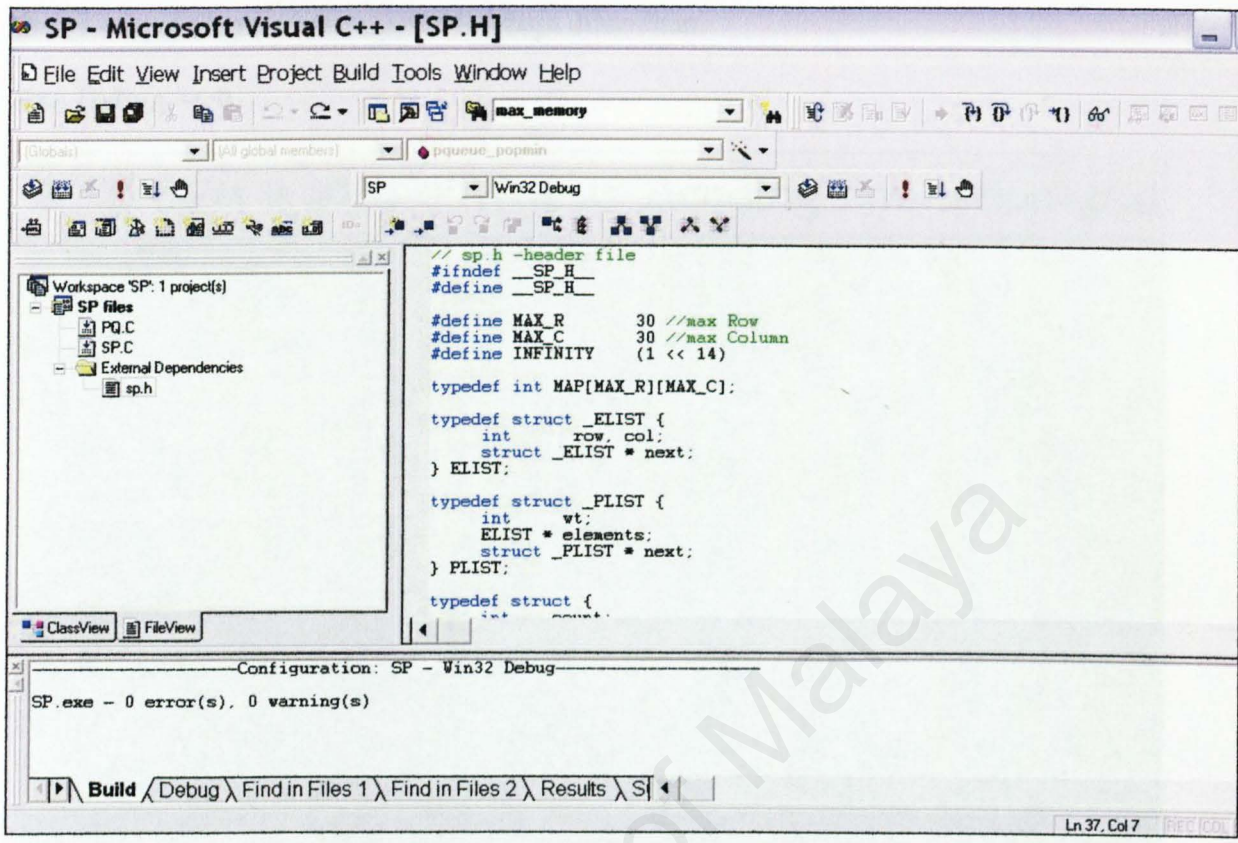
iii) pqueue_new

iv) pqueue_insert

v) pqueue_peekmin

vi) pqueue_popmin

6.4.3 sp.h



Rajah 6.4.3

Fail ini adalah header bagi dua cp.c dan pq.h

#define MAX_R 30 //mensetkan nilai maksimum Row kepada 30

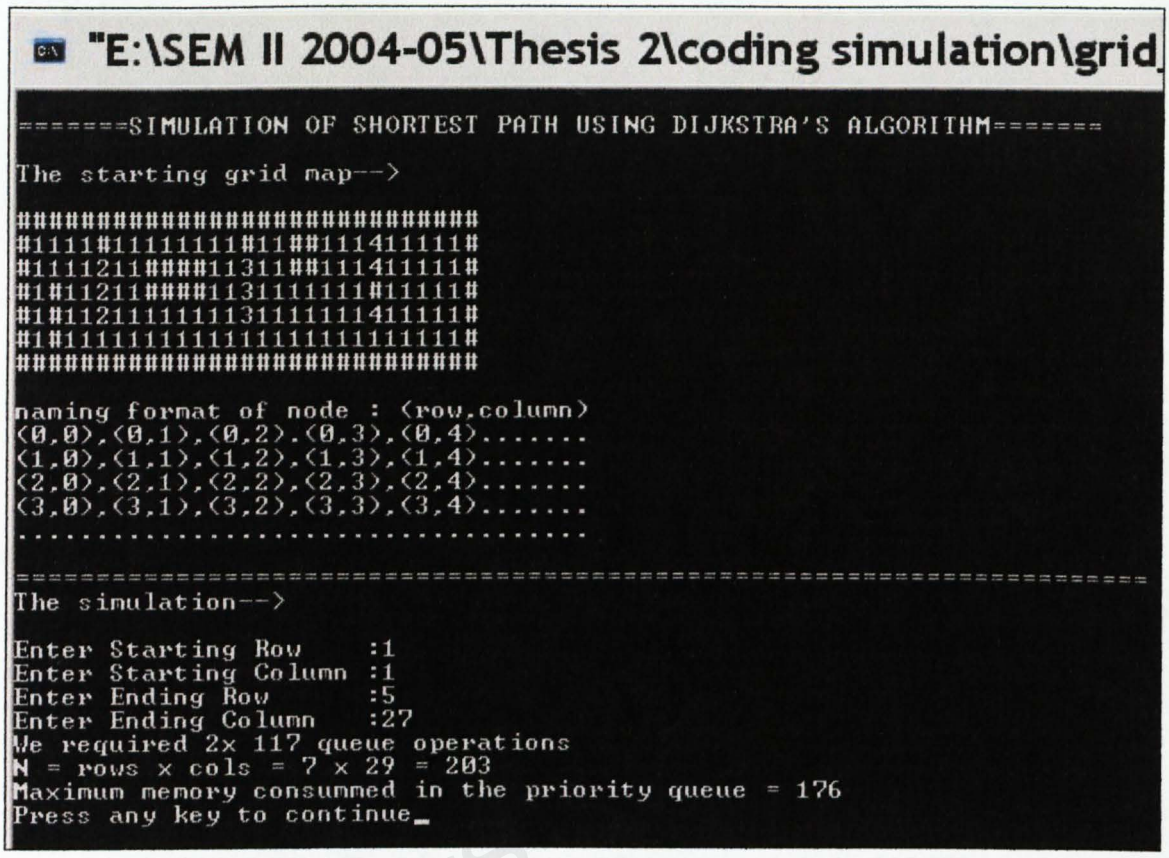
#define MAX_C 30 //mensetkan nilai maksimum Column kepada 30

#define INFINITY (1 << 14) →mensetkan nilai INFINITI kepada 2kuasa14

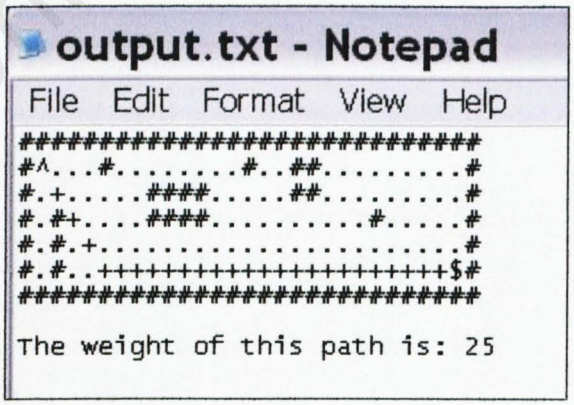
6.5 Pembangunan Antaramuka Pengguna

Hanya output Comand prompt DOS sahaja dihasilkan.

Jika Debug = 0

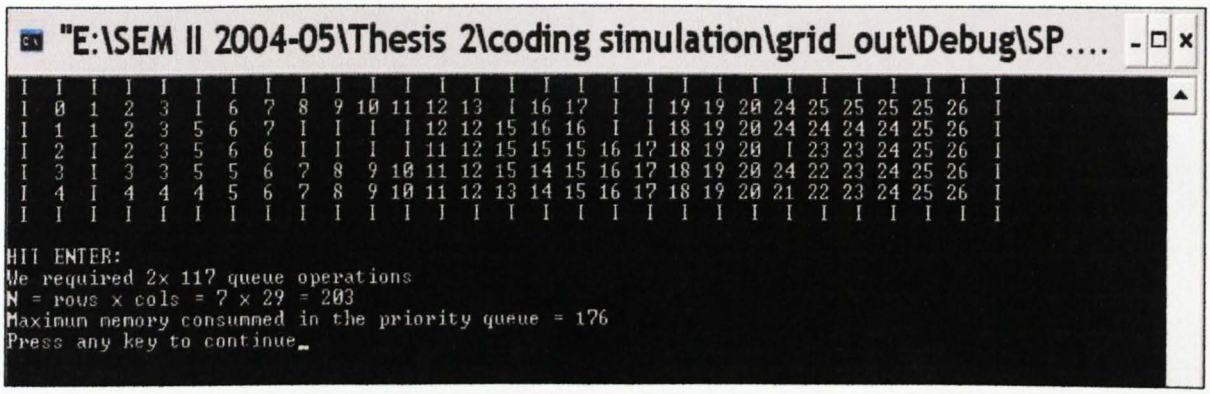


Rajah 6.5.1 Contoh output DOS untuk [1,1]→[5,27]

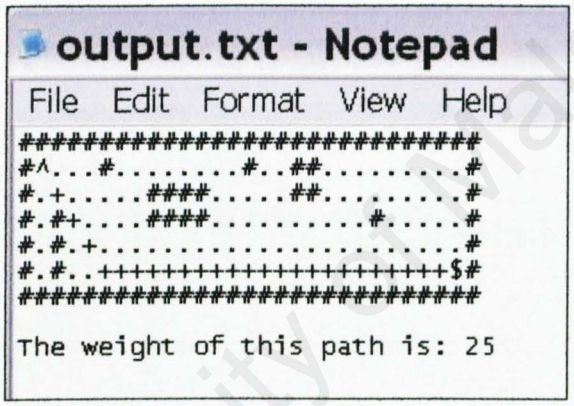


Rajah 6.5.2 Contoh output untuk fail output.txt [1,1]→[5,27]

Jika Debug = 5



Rajah 6.5.3 Contoh output DOS untuk [1,1]→[5,27]



Rajah 6.5.4 Contoh output untuk fail output.txt [1,1]→[5,27]

6.7 Ringkasan

Akhirnya setelah usaha yang dilakukan satu **Simulasi Jarak Terdekat Menggunakan Algoritma Dijkstra (*Simulation of Shortest Path Using Dijkstra's Algorithm*)** berjaya dihasilkan.

BAB 7 :

PENGUJIAN SIMULASI

BAB 7 : PENGUJIAN SIMULASI

7.1 Pengenalan dan objektif pengujian

Fasa Pengujian dilakukan untuk memastikan simulasi bebas daripada ralat sebelum diintegerasi dengan sistem meletak kenderaan. Fasa ini juga dilakukan bagi memastikan simulasi yang dibangunkan menepati piawaian. Data dimasukkan ke dalam input command prompt dan di compile, di build dan di execute. Pada peringkat ini aturcara boleh diexecute kerana tiada error ketika compile dan build. Akan tetapi nilai-nilai yang diperoleh daripada simulasi dikira secara manual dan dibandingkan dengan rumus. Sekiranya nilai yang sama diperoleh, maka algoritma yang digunakan adalah betul. Sekiranya nilai yang salah diperoleh, algoritma yang salah atau aturcara yang salah telah ditulis dan perlu diperbetulkan.

Sebenarnya proses ini adalah proses yang sering dengan fasa/proses Perlaksanaan Simulasi

Terdapat 6 map/input/templet/peta iaitu map1.txt, map2.txt, map3.txt, map4.txt, map5.txt dan map6.txt

7.2. Pengujian Sistem

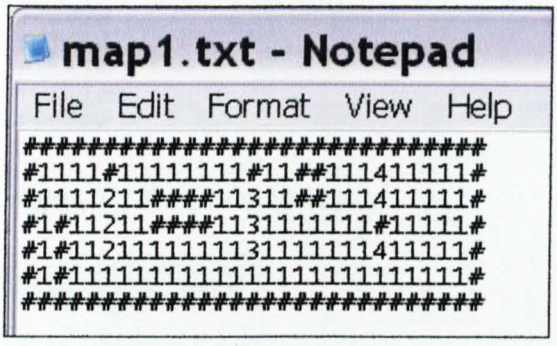
Di dalam fungsi build_path terdapat debug untuk untuk membuktikan aturcara ini.

```
void build_path(void)
{
PQUEUE * pq;
int     row, col;
int       i;

   pq = pqueue_new();
   if (startr < 0 || startr >= rows_read || startc < 0 || startc >= cols_read)
         return;
   dist[startr][startc] = 0;
   parent[startr][startc][0] = -1;
   parent[startr][startc][1] = -1;
   pqueue_insert(pq, startr, startc, 0);

   while (pqueue_popmin(pq, &row, &col) == 0) {
         /* DEBUG INFORMATION */
         pops++;
         if (debug_level > 0)
            printf("\nSelected (row,col)=(%d,%d).  Adding:\n", row, col);

         /* FOREACH square adjacent to (row,col), call it drow(row,i), dcol(col,i) */
         for (i = 0; i < 8; i++) {
            /* Check to make sure the square is on the map */
            if (drow(row,i) < 0 || drow(row,i) >= rows_read ||
                  dcol(col,i) < 0 || dcol(col,i) >= cols_read)
                  continue;

            /* Semak samada untuk lihat samaada shorters path dijumpai*/
            if (dist[row][col] + map[drow(row,i)][dcol(col,i)] <
                  dist[drow(row,i)][dcol(col,i)]) {
                  /* DEBUG INFORMATION */
                  if (debug_level > 0) {
                     printf("\t\t(%d,%d) Weight was/is: ", drow(row,i), dcol(col,i));
               printf("%d/%d\n", dist[drow(row,i)][dcol(col,i)],
map[drow(row,i)][dcol(col,i)] + dist[row][col]);
                  }

                  /* We have a shorter path, update the information */
                  dist[drow(row,i)][dcol(col,i)] = map[drow(row,i)][dcol(col,i)] +
dist[row][col];
                  parent[drow(row,i)][dcol(col,i)][0] = row;
                  parent[drow(row,i)][dcol(col,i)][1] = col;
                  /* Push the modified square onto the priority queue */
```

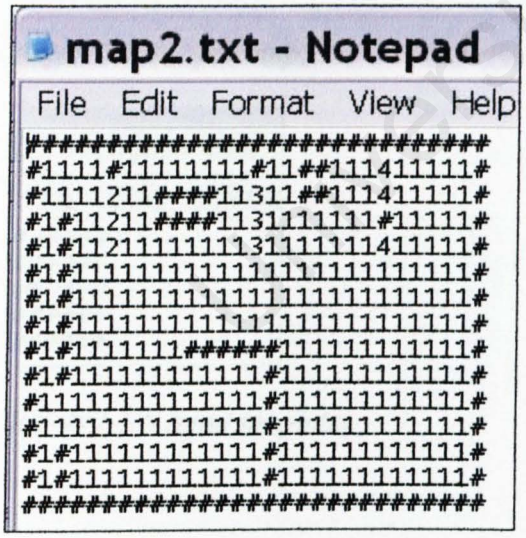
```
                pqueue_insert(pq, drow(row,i), dcol(col,i),
dist[drow(row,i)][dcol(col,i)]);
            } else if (debug_level >= 5) {
                printf("\t\t(%d,%d) Weight Stays: %d (would have been %d)\n",
drow(row,i), dcol(col,i), dist[drow(row,i)][dcol(col,i)], dist[row][col] +
map[drow(row,i)][dcol(col,i)]);
            }
        }
        if (debug_level >= 10) build_illustrate();
    }
    return;
}
```

Fungsi map_read(nama_fail) akan membaca nama_fail. Selepas membaca map file, ia akan settkan(initialize) dist array kepada INFINITY dan parent array kpd -1.Bilangan rows yg dibaca akan direkod dlm rows_read manakala bilangan maximum columns akan direkord dlm cols_read

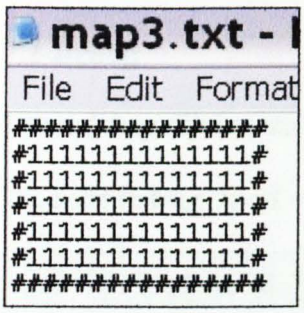
Terdapat 6 nama_fail/map/input/templet/peta iaitu map1.txt, map2.txt, map3.txt, map4.txt, map5.txt dan map6.txt.



Rajah 7.1.1 Input map1.txt

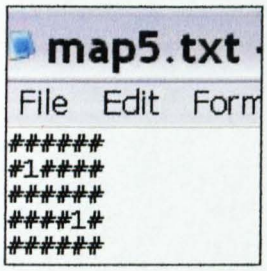


Rajah 7.1.2 Input map2.txt

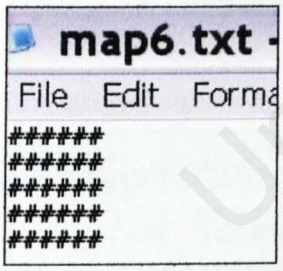


Rajah 7.1.3 Input map3.txt

Rajah 7.1.4 Input map4.txt



Rajah 7.1.5 Input map5.txt

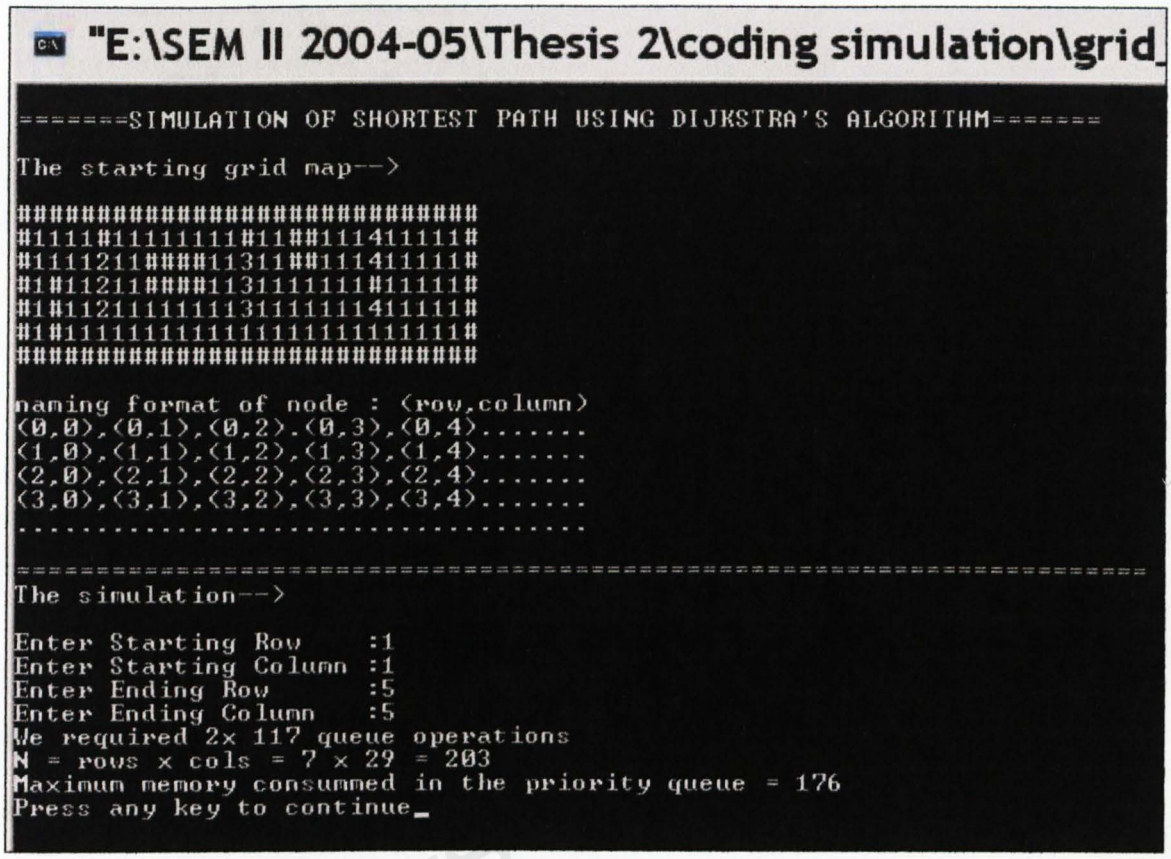


Rajah 7.1.6 Input map6.txt

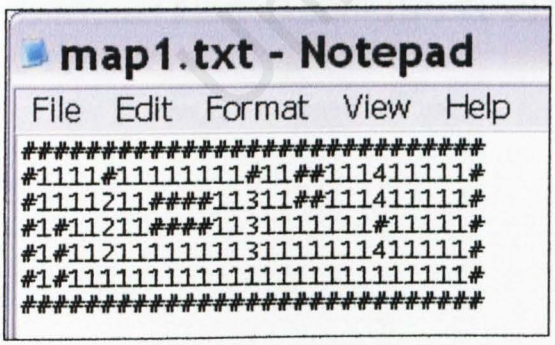
7.2 Debug

Ujian debug dijalankan bagi memastikan output aturcara dan algoritma tepat.

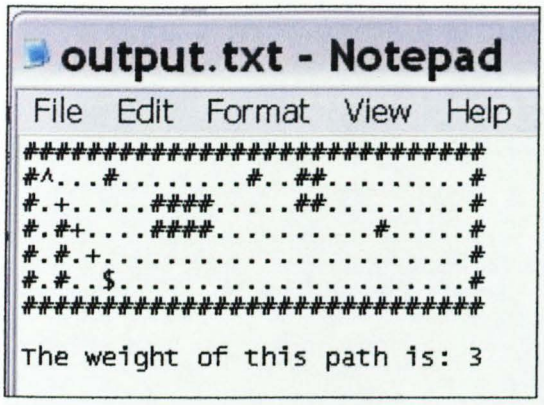
7.2.1 int debug_level = 0;



Rajah 7.2.1.1 output DOS debug 0



Rajah 7.2.1.2 Input map1.txt



Rajah 7.2.1.3 ouput.txt debug 0

Daripada contoh pengujian dalam *Rajah 7.1 output DOS debug 0* didapati :

titik mula = [1,1], titik akhir = [5,5] Fail output.txt akan dijana seperti yang ditunjukkan dalam *Rajah 7.3*

'^' -->titik mula(starting point)

'$' -->titik tamat(ending point)

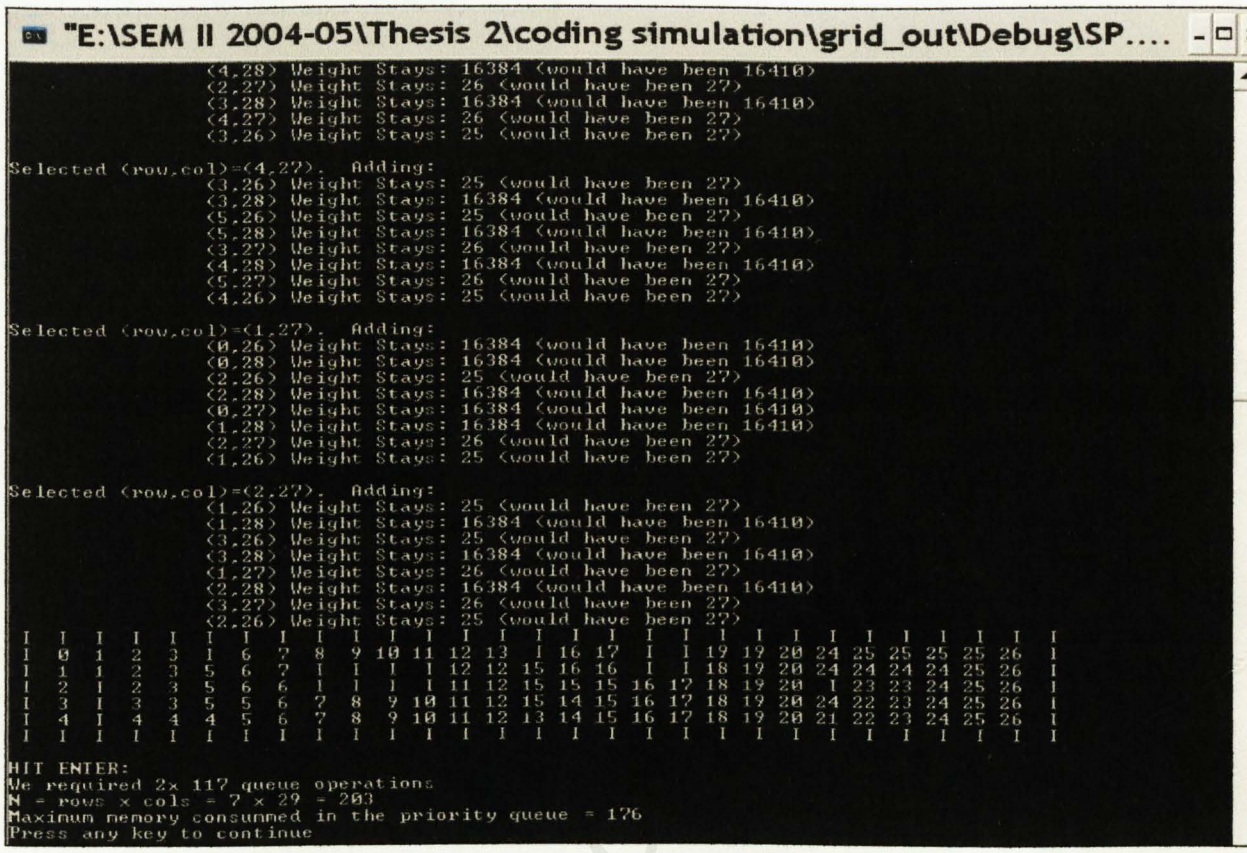
'+' -->jalan terdekat(intermediate squares in the path)

'.' -->jalan lain yg boleh dilalui(the walkable squares in the map)

#-->jalan yg mustahil/dinding.halangan(the impassible squares)

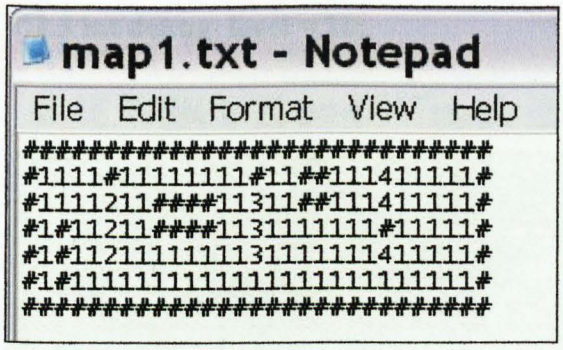
jarak/weight pemberat antara [1,1]→[5,5] = 1+1+1=3

7.2.2 int debug_level = 5;

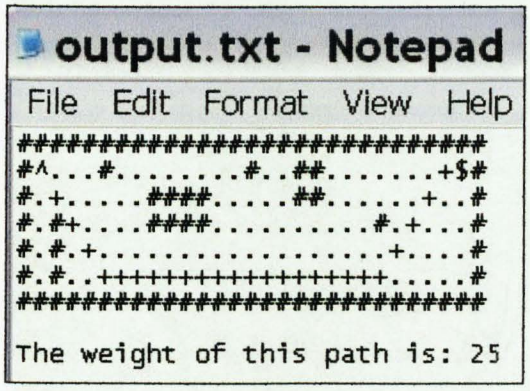


Rajah 7.2.2.1 output DOS debug 5

Jika debug_level disetkan kepada 5, shortest akan ditunjukkan tetapi bukan satu persatu. Ia akan terus memaparkan sume langkah-langkah yang dilalui untuk mencapai matlamat Algoritma Dijkstra iaitu mencari jarak terdekat antara satu titik dan semua titik lain dalam grid yang diberikan.



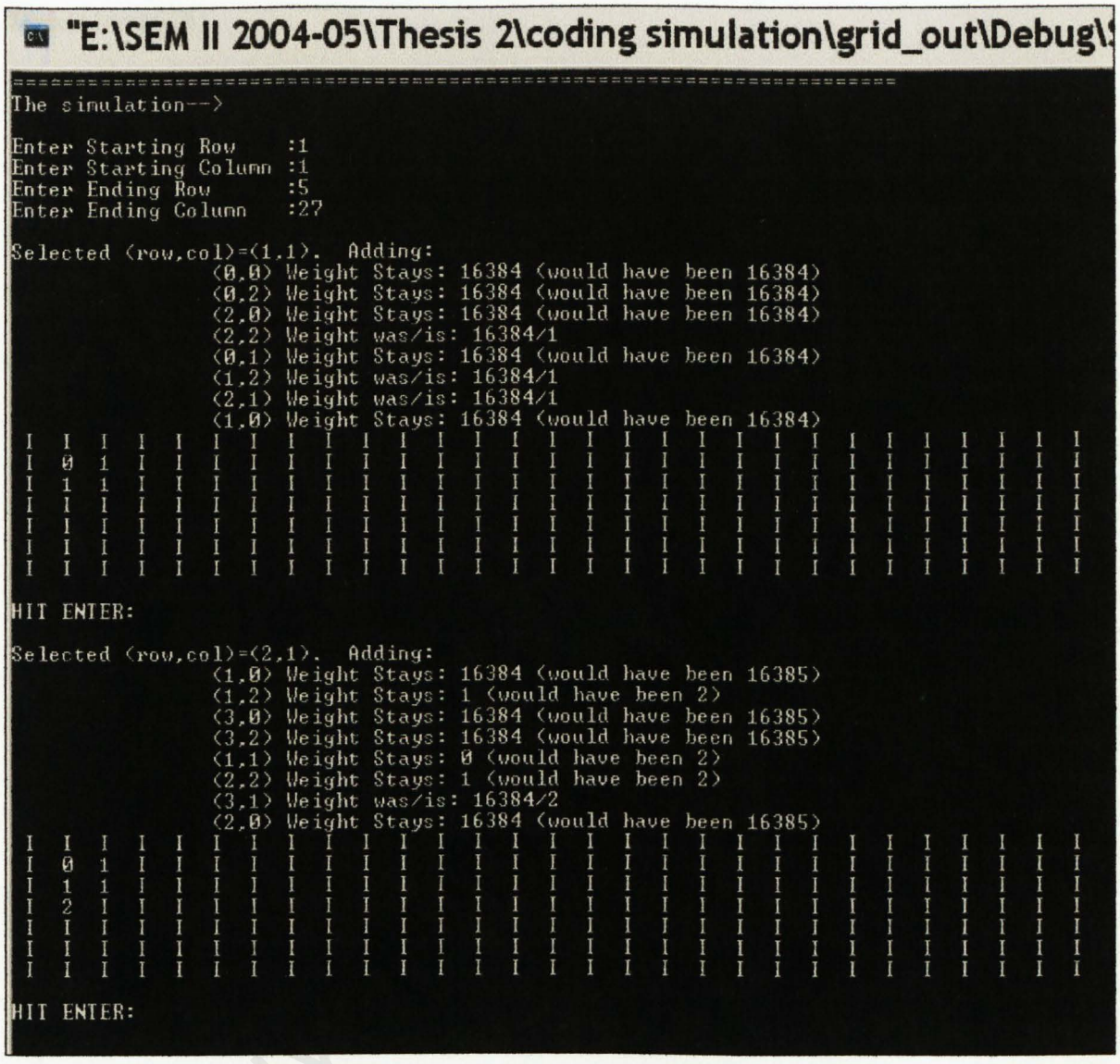
Rajah 7.2.2.2 Input map1.txt



Rajah 7.2.2.3 output.txt

Output di atas menunjukkan titik [1,1] ke [1,27] ia melalui titik-titik yang ditandakan dengan '+'

7.2.3 int debug_level = 10;

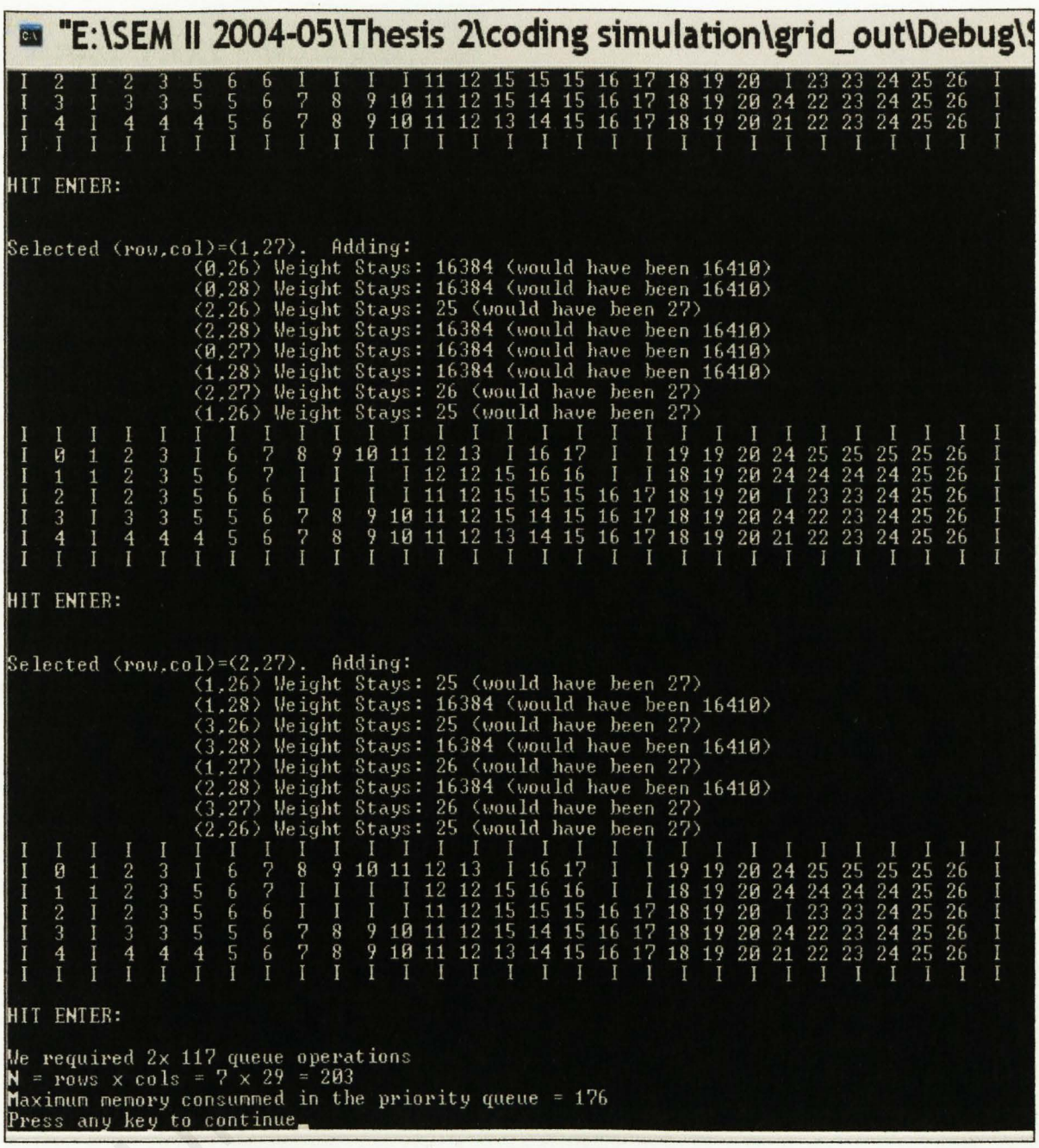


Rajah 7.2.3.1 output DOS debug 10

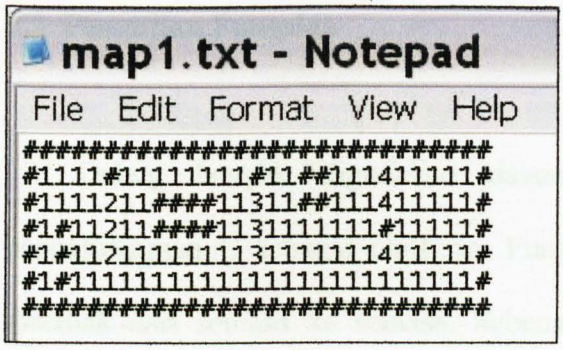
Apabila debug_level disetkan pada 10. fungsi build_illustrate akan dijalankan. Ia akan menyemak satu persatu titik bermula daripada titik awal. Seperti Rajah 7.2.3.1 di atas, titik awal ialah [1,1]. 8 titik di sekelilingnya dikira/disemak mengikut queue. Nilai terkecil akan diambil. Contoh di atas, titik [2,2] Weight was/is : 16384/1 . Ini bermakna nilai baru weight untuk titik [2,2] adalah 1 menggantikan 16384 nilai sebelum ini.

Nilai 16384 ialah nilai INFINITI yang telah disetkan di fail header sp.h. 2kuasa14=16385.

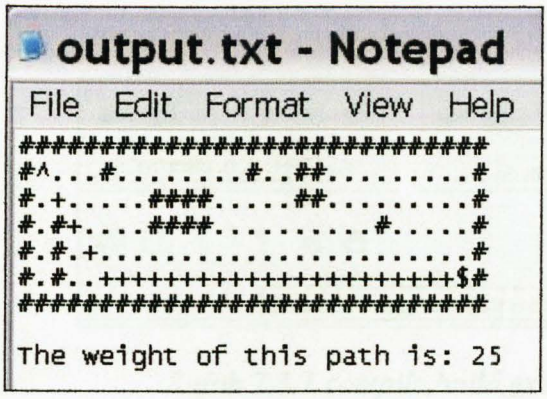
```
int delta[8][2] = { -1, -1, -1, 1, 1, -1, 1, 1, -1, 0, 0, 1, 1, 0, 0, -1 };


void build_illustrate(void){
int i, j;
char line[256];


  for (i = 0; i < rows_read; i++) {
        for (j = 0; j < cols_read; j++)
          if (dist[i][j] < INFINITY)
                printf("%2d ", dist[i][j]);
          else
                printf(" I "); // dapat dilihat dalam Rajah 7.2.3.1
        printf("\n");
  }
  printf("\nHIT ENTER:\n"); fflush(stdout);
  gets(line);
}
```



Rajah 7.2.3.2 output DOS debug 10



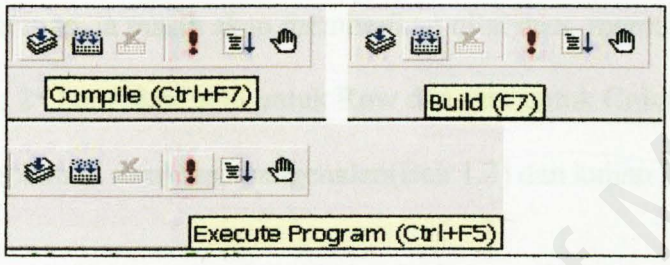
Rajah 7.2.3.3 Input map1.txt



Rajah 7.2.3.4 output.txt

7.3 Pengujian Fungsian

Debug banyak digunakan dalam simulasi ini kerana ia menggunakan pengaturcaraan C sebagai platform. Fungsi-fungsi dalam program pengaturcaraan C disemak dari semasa ke semasa. Sebenarnya proses pengujian dilakukan sepanjang proses pengaturcaraan dengan cara Complile(Ctrl+F7),Build(F7) dan Execute Program(Ctr F5)



Rajah 7.3.1 compile,build,execute

7.4 Ringkasan

Bab ini telah menunjukkan langkah-langkah yang dijalankan dalam proses pengujian. Melalui proses-proses dan peringkat-peringkat pengujian yang dijalankan, dapatlah disahkan bahawa simulasi ini berfungsi mengikut spesifikasi yang telah ditentukan. Rumus dijkstra juga telah berjaya dibuktikan kerana ia mencari jarak terdekat untuk semua titik dari titik awal. Ini dapat dibuktikan dengan mensetkan debug_level kepada 10 dan juga proses meninputkan data dari [1,1] ke mana-mana titik dalam map, ia masih akan memberikan nilai max_memory dan bilangan queue yang sama. 2* bermakna satu untuk Row dan satu untuk Column. Objektif yang telah ditetapkan di awal fasa pengenalan(Bab 1.3) dan kajian literasi telah berjaya tercapai.

BAB 8 :

PENILAIAN DAN PERBINCANGAN

BAB 8 : PENILAIAN DAN PERBINCANGAN

8.1 Pengenalan

Setelah simulasi ini diuji dan proses pengubahsuaian dilakukan, maka semua objektif yang telah disasarkan pada fasa perancangan tercapai. Ini dibuktikan dengan data-data yang diperoleh hasil daripda fasa Pengujian simulasi. Ini membuktikan simulasi yang dirancang berjaya dibangunkan mengikut spesifikasi yang telah ditetapkan diperingkat perancangan. Tambahan pula terdapat beberapa pertambahan dalam simulasi yang dibangunkan.

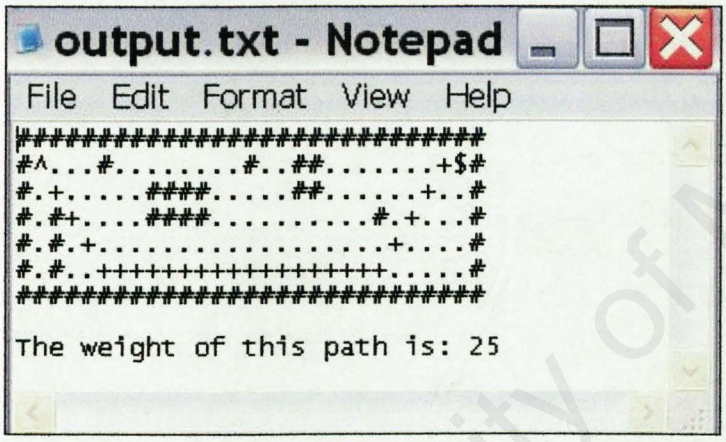
Dalam bab ini akan menerangkan kelebihan dan kekurangan simulasi secara terperinci.

8.2 Kelebihan Simulasi

Terdapat beberapa kelebihan simulasi yang dibangunkan. Diantaranya adalah:

8.2.1 Jalan terdekat dalam fail output.txt

Jalan atau shortest path yang dihasilkan adalah dalam file.txt / notepad yang dinamakan output.txt . Dalam DOS juga terdapat sebahagian daripada output yang menunjukkan jumlah bilangan queue yang diperlukan untuk mencari semua shortest path, row * column map.txt dan maksimum memori yang digunakan dalam mencari jarak terdekat. Fungsi memory allocation malloc digunakan dalam fail pq.h.



Rajah 8.2.1

Fail output.txt ini akan dibaca oleh Chain Code yang akan menterjemahkannya kepada pergerakan. Seperti yang dapat dilihat dalam *Rajah 8.2.1* di atas,

'^' -->titik mula(starting point)

'$' -->titik tamat(ending point)

'+' -->jalan terdekat(intermediate squares in the path)

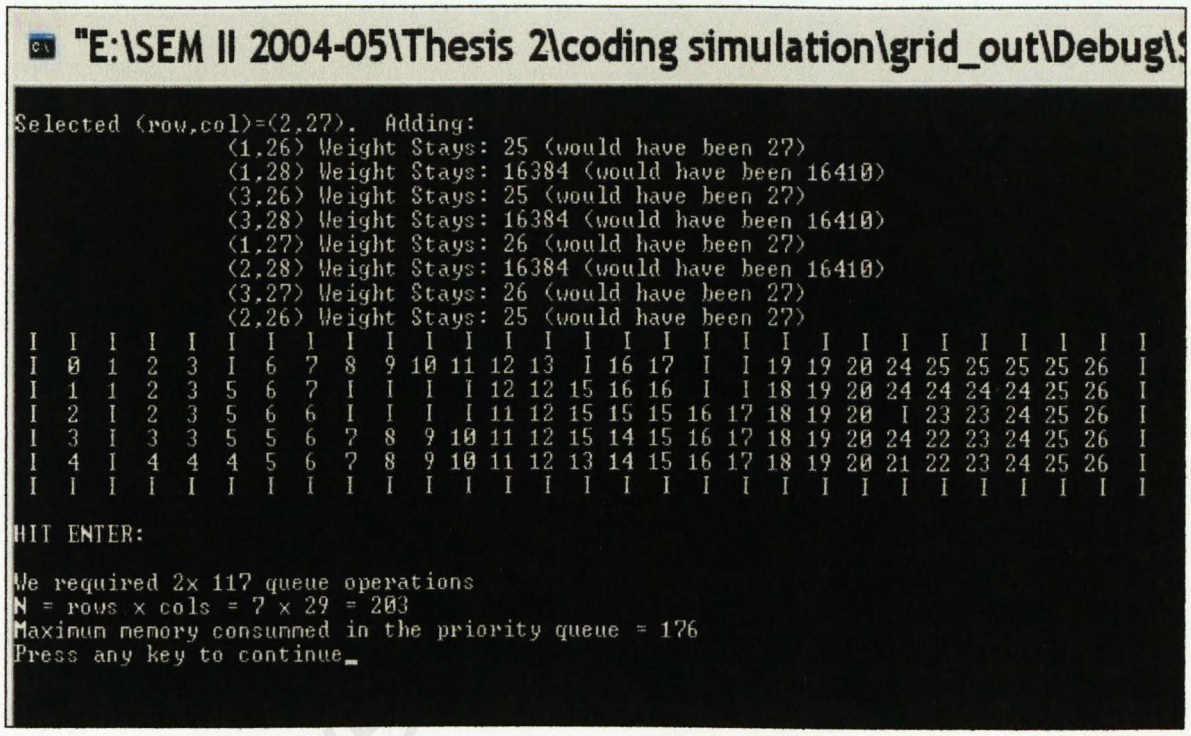
'.' -->jalan lain yg boleh dilalui(the walkable squares in the map)

#-->jalan yg mustahil/dinding.halangan(the impassible squares)

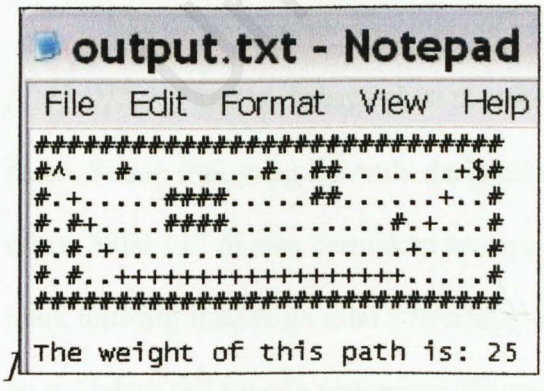
(The Weight of this path is : 25) Nilai pemberat untuk jalan ini ialah : 25

8.2.2 Semua Jalan Terdekat dicari

Simulasi ini akan mencari jarak terdekat antara titik mula dan titik tamat. Ia bukan sahaja mencari shortest path antara 2 titik yang dimasukkan tetapi ia akan mencari nilai jarak terdekat daripada titik mula yang dimasukkan untuk semua titik/nod di dalam map file tersebut. Ini dapat dilihat jika nilai debug_level disetkan kepada 10. Jarak terdekat antara titik yang dipilih ke semua titik di dalam map/fail output akan dicari.



Rajah 8.2.2 Shortest path dari [1,1]→[1,27



Rajah 8.2.3 output.txt

```
Selected (row,col)=(2,27).  Adding:
                (1,26) Weight Stays: 25 (would have been 27)
                (1,28) Weight Stays: 16384 (would have been 16410)
                (3,26) Weight Stays: 25 (would have been 27)
                (3,28) Weight Stays: 16384 (would have been 16410)
                (1,27) Weight Stays: 26 (would have been 27)
                (2,28) Weight Stays: 16384 (would have been 16410)
                (3,27) Weight Stays: 26 (would have been 27)
                (2,26) Weight Stays: 25 (would have been 27)
 I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I
 I  0  1  2  3  I  6  7  8  9 10 11 12 13  I 16 17  I  I 19 19 20 24 25 25 25 25 26  I
 I  1  1  2  3  5  6  7  I  I  I  I 12 12 15 16 16  I  I 18 19 20 24 24 24 24 25 26  I
 I  2  I  2  3  5  6  6  I  I  I  I 11 12 15 15 15 16 17 18 19 20  I 23 23 24 25 26  I
 I  3  I  3  3  5  5  6  7  8  9 10 11 12 15 14 15 16 17 18 19 20 24 22 23 24 25 26  I
 I  4  I  4  4  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26  I
 I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I  I

HIT ENTER:

We required 2x 117 queue operations
N = rows x cols = 7 x 29 = 203
Maximum memory consumned in the priority queue = 176
Press any key to continue
```

Rajah 8.2.4 Shortest path dari [1,1]→[5,20]

output.txt - Notepad

File Edit Format View Help

```
##############################
#^...#........#..##.........#
#.+.....####.....##.........#
#.#+....####..........#.....#
#.#.+.......................#
#.#..+++++++++++++++$.......#
##############################

The weight of this path is: 18
```

Rajah 8.2.5 output.txt

Jika dibandingkan *Shortest path dari [1,1]→[1,27 dan Shortest path dari [1,1]→[5,20]* kedua-duanya akan memberikan nilai queue dan maximum memory yang sama. Setiap titik yang bermula daripada titik [1,1] akan mempunyai kedua nilai ini yang sama. Nilai 117 di atas bermakna terdapat 117 langkah yang digunakan dalam mencari jarak terdekat manakala nilai 176 adalah nilai maksimum memori yang ditambah di fail pq.c. Dalam fail ini ada penyataan if(current_memory>max_memory) max_memory=

current_memory; Nilai ini juga sama kerana ia telah shortest path hanya dikira sekali daripada satu titik dan disimpan dalam memori.

8.3 Kelemahan Simulasi

8.3.1 Nilai menegak dan melintang sama

Kelemahan simulasi yang paling utama adalah nilai weight/pemberatnya. Jika nilai weight/pemberat satu titik(berdasarkan input fileMap.txt) adalah 1. Nilai 1 akan dikira dari semua posisi, samada dari kiri ke kanan,kanan ke kiri, bawak ke atas, atas ke bawah, bucu kiri atas ke bucu kanan bawah, bucu kiri bawah ke bucu kanan atas, bucu kanan atas ke bucu kiri bawah, dan bucu kanan bawah ke bucu kiri atas.Semua nilai yang akan dikira adalah 1. Ini berbeza dengan rumus tiagonometri yang mengatakan bahawa dalam satu segitiga kaki sama, yang mepunya dua kaki sama contohnya bernilai1, nilai condong akan jadi $\sqrt{(1+1)} = \sqrt{2}$.

8.3.2 Melanggar Bucu

Satu lagi kelemahan sistem ini adalah, ia boleh melanggar bucu. Jika ada halangan di hadapan, ia akan menggunakan jarak terdekat tanpa menghiraukan bucu dinding. Ini dapat dilihat dalam fasa pengujian. Rajah di bawah menjelaskan lagi masalah ini.

Rajah 8.3.1

Output jarak terdekat sebenarnya dalam file text. Rajah diatas cuma untuk menunjukkan kelemahan simulasi yang dibangunkan. Oleh itu jarak terdekat diRUN dalam DOS hanya untuk tujuan pembuktian. Dapat dilihat dari simulasi ini, jika dimasukkan nilai titik mula=[5,1] dan titik akhir=[5,3], jarak terdekat/shortest path akan dihasilkan dan diwakili dengan tanda '+'. Daripada *Rajah 8.5.1* di atas dapat dilihat jalan yang yang dilalui adalah [5,1]titik mula->[4,1]->[3,1]->[2,2]->[3,3]->[4,3]->[5,3]titik tamat. Daripada senarai ini dapat dilihat pergerakan daripada titik [3,1]->[2,2]->[3,3] Dimana titik [3,2] adalah dinding dan titik[3,1] melanggar bucu dinding[3,2] untuk sampai ke [2,2] dan dari titik ini ia melanggar bucu [3,2] untuk sampai kepada titik [3,3]

8.4 Masalah dan Penyelesaian

Sepanjang menjalankan latihan ilmiah simulasi ini, pelbagai masalah telah timbul dan setelah berusaha dengan gigih, masalah-masalah ini telah berjaya dihadapi dan ditangani secara efekstif. Antara masalah-masalah yang dihadapi dan penyelesaian bagi masalah-masalah tersebut adalah :

8.4.1

Masalah: Memahami Algoritma Dijkstra. Rumus yang kompleks dan coding/aturcara yang berbeza-beza.

Penyelesaian :

Merujuk kepada buku-buku, forum-forum dan internet untuk semua kemusyhkilan dan masalah berkenaan Algoritma Dijkstra.

8.4.2

Masalah:

Mencari buku-buku yang berkenaan Algoritma Dijkstra. Dalam proses mencari maklumat tambahan saya telah pergi mencari bahan rujukan di Perpustakaan Negara tetapi hampa kerana tidak berjaya menjumpai apa yang saya perlukan. Buku-buku yang berkaitan robotik sahaja ditemui. Tiada buku Algoritma Dijkstra's.

Penyelesaian:

Mencari buku-buku berkenaan Aturcara C,Data Struktur,Algoritma Analisi C di perpustakaan Universiti Malaya.

8.4.3

Masalah:

Rujukan Dijkstra, seorang tokoh sains komputer yang telah meninggal dunia pada tahun 2002 menyebabkan saya tidak beroleh peluan untuk berhubung dengan beliau.

Penyelesaian:

Membuat rujukan dengan penyelidik-penyelidik lain di dunia ini.

8.4.4

Masalah

Masalah komunikasi dengan penulis-penulis buku dan laman web yang berkaitan dengan aturcara Dijkstra. Ini berlaku apabila saya telah berhubung dengan seorang penyelidik daripada Romania yang kurang fasih berbahasa inggeris.

Penyelesaian:

Meminta kebenaran mengambil aturcara beliau untuk dianasila sendiri.

8.4.5

Masalah

Rujukan internet yang berbeza-beza menyebabkan saya terpaksa menganalisa teknik-teknik dan kesahihan rumus dijkstra yang digunakan.

Penyelesaian:

Rujukan-rujukan yang tidak ada kaitan dengan simulasi ditapis seperti Rumus Dijkstra yang berkaitan dengan teknologi Rangkaian. Rumus Dijkstra yang digunakan dalam Routing Table. Rumus ini diabaikan. Perbezaan algoritma dan rumus dijkstra berdasarkan pemahaman dilakukan.

8.4.6

Masalah:

Rujukan internet dalam C. Banyak laman web membekalkan aturcara dan coding yang terlalu panjang dan sukar difahami.

Penyelesaian:

Membuat aturcara daripada bawah. Mulakan dengan menamakan fungsi-fungsi yang diperlukan.

8.4.7

Masalah:

Banyak sumber yang diperoleh/dijumpai adalah ditulis dalam bahasa pengaturcaraan Java, dan juga Microsoft Foundation Class(MFC). Objektif utama saya adalah menghasilkan output menggunakan command prompt DOS. Oleh itu bahasa C dan C++ sahaja boleh dipertimbangkan.

Penyelesaian:

Menganalisis aturcara-aturcara yang ditemui.

8.4.8

Masalah:

Tidak ramai pakar di Malaysia yang boleh saya rujuk. Simulasi atau algoritma ini sudah jarang dibincangkan di Malaysia kerana algoritma A* lebih popular dikalangan penyelidik di Malaysia.

Penyelesaian:

Mendapatkan khidmat nasihat daripada penyelidik dan pensyarah luar negara.

8.4.9

Masalah :

Masa yang kurang dan terhad. Ini disebabkan peperiksaan yang dijalani pada awal bulan mac, menyebabkan masa terpaksa diperuntukkan untuk menelaah pelajaran.

Penyelesaian:

Membuat jadual kerja untuk membahagikan masa di antara membuat latihan ilmiah dan pelajaran.

8.5.10

Masalah:

Masalah menggunakan pengaturcaraan C++. Kekurangan pengetahuan dan kemahiran dalam penggunaan bahasa ini. Kebanyakan sumber dan contoh-contoh algoritma dan priority queue menggunakan aturcara C. Algoritma dijkstra wujud sejak tahun 1970an, dan pada ketika itu Aturcara C digunakan secara meluas. Algoritma Dijkstra lebih sesuai dibangunkan dengan bahasa pengaturcaraan C.

Penyelesaian:

Menggunakan Bahasa Pengaturcaraan C sebagai bahasa pengaturcaraan utama dalam membangunkan simulasi. Ini ditambah dengan pengetahuan sedia ada dan pengalaman ketika praktikal.

8.5 Cadangan Masa Depan

Walaupun dengan beberapa kebaikan dan kelebihan simulasi ini, masih terdapat kekurangan dan ruang untuk diperbaiki. Simulasi Rumus dijkstra yang mengambil kira faktor yang lebih kompleks di mana ia menggunakan nilai condong yang berbeza dengan nilai menegak dan nilai mendatar. Bucu juga diambil kira. Masalah ini timbul kerana algoritma yang mudah digunakan dalam pembangunan simulasi. Priority Queue yang lebih kompleks perlu digunakan seperti menggunakan heap Fibonacci.

8.6 Pengalaman Yang Diperoleh

Daripada proses-proses/fasa-fasa dalam menyiapkan simulasi ini pelbagai pengalaman yang sangat beharga dan pengetahuan yang berguna berjaya diperoleh. Pengalaman dalam membangunkan dan menjayakan latihan ilimiah ini sangat bermakna kepada diri saya.

8.7 Ringkasan

Fasa Penilaian dan Perbincangan ini berjaya menerangkan kelebihan-kelebihan simulasi dan juga menerangkan kekurangan-kekurangan simulasi dan cadangan untuk menambahbaikkan simulasi ini pada masa hadapan.

Secara keseluruhannya Simulasi menggunakan *Dijkstra's Algorithm* yang dibangunkan ini berjaya mencapai objektif iaitu mencari shortest path antara dua titik dan memberikan jalan terdekat yang perlu dilalui. Fail output.txt di reka daripada aturcara dan diSAVE secara automatik di dalam directory yang sama dengan fail sp.c, pq.c dan sp.h. Fail ini akan dibaca oleh Chain Code yang akan menterjemahkannya kepada pergerakan robotik. Simulasi ini telah siap untuk diintegrasi dan digabungkan dengan modul-modul lain untuk melengkapkan keseluruhan rekabentuk Sistem Automasi Meletak Kenderaan.

Simulasi siap dan sedia untuk digunakan.

LAMPIRAN

~Manual Pengguna/Cara Menggunakan Simulasi Jarak Terdekat Menggunakan Algoritma Dijkstra~

Simulasi ini mudah digunakan. Ia hanya menggunakan Command Prompt DOS.
User hanya perlu membuka Microsoft Visual C++ dan membuka fail/directory grid_out. Di dalam fail grid_out akan ada 3 fail aturcara iaitu SP.C,PQ.C dan SP.H.
(Di dalam fail ini buka fail sp.c, pq.c dan sp.c Kemudian debug, built dan execute.) Selain itu terdapat juga 7 fail text iaitu map1.txt, map2.txt, map3.txt, map4.txt,map5.txt,map6.txt dan output.txt.
Mula-Mula,buka Program/Perisian Microsoft Visual C++.Kemudian pergi ke fail, open dan highlight 3 fail SP.C, PQ.C dan SP.H. Clik Open. Apabila butang yang bertulis "This build command requires an active project workspace.

Would you like to create a default project workspace?"dipaparkan, tekan butang yes. Compile fail pq.c. Butang "PQ.C This file is not included in the project. Would you like to add it? " akan muncul. Telan butang Yes. Kemudian Build. Kemudian pergi ke SP.C dan tekan butang execute. Simulasi akan dijalankan.

User hanya perlu mengikuti arahan memasukan nilai [Row awal,Colomn awal] dan nilai [Row akhir,Coloum akhir] Untuk membaca output pergerakan, user perlu membuka fail yang bernama output.h .(fail ini akan diReWrite setiap kali selepas program diexecute)

Secara default nya jika debug_level disetkan kepada 0. output akan keluar secara terus tanpa langkah-langkah menunjukkan bagaimana shortest path ditemui. Oleh itu setkan debug level kepada 5 atau 10. Terdapat pilihan utama debug level iaitu 0,5,dan 10. selain itu, boleh juga diRUN akan tetapi sedikit ralat akan berlaku. Jika debug_level disetkan kepada 10 semua pergerakan daripada awal, dari titik mula ke titik tamat dan semua shortest path bagi titik mula keseluruhan grid yang dikira akan ditunjukkan.

Sekali lagi untuk membaca jalan/shortest path dan weight, sila buka fail output.txt Ini sebenarnya dilakukan untuk dibaca dan diterjemahkan oleh Chain Code. Oleh itu simulasi ini tidak perlu mesra pengguna. Walaubagaimanapun simulasi ini sangat mudah difahami dan diikuti oleh pengguna lain selain pembangun Sistem Meletak kenderaan Automasi.

RUJUKAN

1) Robert Segewick.(1998). *Algorithms In C Part 1-4 Fundamentals,Data Structure, Sorting,Searching* 3rd ed. Addison-Wesley

2) Eric S.Roberts.(1995).*The Art of C An Introduction to Computer Science.* Addison-Wesley

3) Mark Allen Weiss.(1996).*Data Structures and Algorithm Analysis in C.* 2nd ed. Addision-Wesley

4) Tony Zang.(2000).*SAMS Teach Yourself C in 24 Hours.* 2nd ed. SAMS.

5) Bryon S. Gottfried, Ph.D.(1996). *Programming with C.* 2nd ed. Schaum's Outline

6) Marni Abu Bakar.(1998).*Struktur Data Menggunakan C.* Pentice Hall

7) Wong Chun Keong.(2000).*Data Structures With C.* 1st ed.Sejana Publishing

8) Dr.P.Sellapanan.(1999).*Programming in C.* 1st ed.Sejana Publishing

9) Deitel&Deitel.(2001).*C++ How To Program.*3rd ed.Dietel&Dietel

10) Cisco Systems.(2001).*Dictionary of Internetworking Terms And Acronyms.* Cisco Press

RUJUKAN INTERNET

Algoritma Dijkstra

http://www.cs.utexas.edu/users/EWD/

http://rollerjm.free.fr/pro/graphs.html#7

http://renaud.waldura.com/doc/java/dijkstra/

http://www.cs.mcgill.ca/~cs251/OldCourses/1997/topic29/

http://en.wikipedia.org/wiki/Dijkstra's_algorithm

http://publish.uwo.ca/~jmalczew/gida_1/Zhan/Zhan.htm

http://www.informit.com/articles/article.asp?p=169575&seqNum=3

http://encyclopedia.thefreedictionary.com/Dijkstras%20Algorithm

http://www.thecodeproject.com/csharp/graphs_astar.asp

http://cpcug.org/user/scifair/Preygel/Preygel.html

http://www.devmaster.net/articles/pathfinding/

http://www.devmaster.net/forums/index.php?showtopic=421

http://www.cs.mcgill.ca/~cs203/lectures/lecture16/tsld001.htm

http://www.cs.mcgill.ca/~cs203/lectures/lecture16/sld015.htm

http://students.hamilton.edu/2004/agconway/algos/pseudo.htm

http://www.seas.gwu.edu/~simhaweb/cs151/lectures/module9/module9.html

http://metcs.bu.edu/ghuang/cs342/chap2a/2a-5-8.html

http://www.worldhistory.com/wiki/D/Dijkstra%27s-algorithm.htm

http://www.cs.utexas.edu/users/EWD

http://www.cs.chalmers.se/Cs/Grundutb/Kurser/ndg/De11/lec-20030310-Mon.html

http://www.nist.gov/dads/HTML/dijkstraalgo.html

http://www.cs.umd.edu/~shankar/417-F01/Slides/chapter4a-aus/tsld014.htm

http://www.cs.wustl.edu/cs/cs/archive/CS423_FL00/Chapter4a/text13.html

http://www.cs.virginia.edu/~mngroup/projects/mpls/documents/thesis/node21.html

http://www.cs.chalmers.se/Cs/Grundutb/Kurser/ndg/Del1/lec-20030310-Mon.html

http://web.umr.edu/~ercal/387/class.Graph

http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/dij-proof.htm
l

http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/dijks
traAlgor.htm

http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/dij-op.html

http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/dijkstra.html

http://www.arl.wustl.edu/~jst/cse/541/src/intro.html

http://www.arl.wustl.edu/~jst/cse/541/src/cmds/spt_dijkstra.c

http://rollerjm.free.fr/pro/graphs.html

http://www-unix.mcs.anl.gov/dbpp/text/node35.html

http://www.redbrick.dcu.ie/~hazy/networks/dijkstra.html

http://rollerjm.free.fr/pro/graphs.html

http://www.brpreiss.com/books/opus4/html/page566.html

http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/dijkstra.html

http://www.brpreiss.com/books/opus4/html/page566.html

http://www.answers.com/topic/dijkstra-s-algorithm

http://216.5.163.53/DirectX4VB/Tutorials/GeneralVB/GM_Dijkstra.asp

http://www.cs.utexas.edu/users/EWD/transcriptions/EWD11xx/EWD1166.html

http://www.cs.usask.ca/resources/tutorials/csconcepts/1999_8/tutorial/advanced/dijkstra/dijkstra.html

http://www.ibiblio.org/links/devmodules/graph_networking/compat/page13.html

http://nz.cosc.canterbury.ac.nz/tad.takaoka/alg/

http://nz.cosc.canterbury.ac.nz/tad.takaoka/alg/spalgs/spalgs.html

http://www.cs.ucsd.edu/users/calder/classes/win05/190/notes/graphs.html

http://www.codeproject.com/cpp/GcDijkstra.asp

Struktur Data dan Pengaturcaraan C

http://computer.howstuffworks.com/c29.htm

http://publications.gbdirect.co.uk/c_book/chapter9/input_and_output.html

http://pbpl.physics.ucla.edu/Computing/C_Tutorial/Section_6_(Input_and_Output)/

http://courses.cs.vt.edu/~cs3304/FreePascal/doc/ref/node11.html#SECTION038200000000000000

http://vergil.chemistry.gatech.edu/resources/programming/c-tutorial/io.html

http://www-vs.informatik.uni-ulm.de:81/projekte/Oberon-2.Report/Chapter10-2.html

http://www.doc.ic.ac.uk/lab/secondyear/cstyle/node9.html

http://docs.hp.com/en/B3901-90003/ch03s13.html

http://bincang.net/forum/articles.php?action=viewarticle&artid=8

http://www.lsc-group.phys.uwm.edu/~duncan/binary_inspiral/downloads/gauss.c

http://www.cs.cf.ac.uk/Dave/C/node18.html

http://ncca.bournemouth.ac.uk/CourseInfo/MAVisAn/C/InputOutput/

http://www.calculator.org/CalcHelp/matrix.html

http://www.eschertech.com/product_documentation/Language%20Reference/Language ReferenceManual06.htm

http://ciips.ee.uwa.edu.au/~morris/Year1/CLP110/io_intro.html

http://www.infosys.utas.edu.au/info/documentation/C/CStdLib.html

http://libslack.org/manpages/lim.3.html

Memory Allocation

http://cplus.about.com/od/beginnerctutoria1/l/aa042402a.htm

http://www.cs.cf.ac.uk/Dave/C/node11.html

http://www.cs.cf.ac.uk/Dave/C/section2_21_8.html

http://www.eskimo.com/~scs/cclass/notes/sx11.html

http://www.eskimo.com/~scs/cclass/notes/sx11a.html

http://www.eskimo.com/~scs/cclass/notes/sx11b.html

http://www.eskimo.com/~scs/cclass/notes/sx11c.html

http://www.eskimo.com/~scs/cclass/notes/sx11d.html

http://www.answers.com/topic/dynamic-memory-allocation

http://www.google.com.my/search?hl=en&lr=&oi=defmore&q=define:heap

Simulasi

http://carbon.cudenver.edu/~hgreenbe/sessions/dijkstra/DijkstraApplet.html

http://students.ceid.upatras.gr/~papagel/project/kef5_7_1.htm

http://www.cs.usask.ca/resources/tutorials/csconcepts/1999_8/tutorial/advanced/dijkstra/dijkstra_applet.html

http://www.ifors.ms.unimelb.edu.au/tutorial/dijkstra/island.html

http://www.ifors.ms.unimelb.edu.au/tutorial/dijkstra/dp_frame.html

http://www.lupinho.de/gishur/html/DijkstraApplet.html

```
//sp.c
//Shortest Paths menggunakan Algoritma Dijkstra
//Mohd Yohan Bin Ibrahim
//WEK010407

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "sp.h"

MAP map;
int dist[MAX_R][MAX_C];
int parent[MAX_R][MAX_C][2];
int startr, startc;
int endr, endc;
int rows_read, cols_read;
int pops;
int debug_level = 0;
int wt;
extern int max_memory;

/*
 * map_read(nama_fail)
 * Baca nama_fail/file_name dan save info dlm global map structure.
 * Selepas baca map file, initialize dist array kepada INFINITY dan parent array kpd -1.
 * Bilangan rows yg yg di read akan direkod dlm rows_read
 * Bilangan maximum columns akan direkord dlm cols_read
*/

int
map_read(char * fname)
{
FILE * file;
int   row, col;
char  line[MAX_C*2+1], * line_ptr;

   if ((file = fopen(fname, "r")) == NULL) return(-1);

   cols_read = 0;
   for (row = 0; row < MAX_R && ! feof(file); row++) {
        if (fgets(line, MAX_C*2, file) == NULL) break;
        printf("%s",line);
        for (col = 0, line_ptr = line; col < MAX_C && *line_ptr; col++, line_ptr++)
            if (isdigit(*line_ptr))
                map[row][col] = *line_ptr - '0';
            else
                map[row][col] = INFINITY;
```

```
        if (col > cols_read) cols_read = col - 1;
        for (; col < MAX_C; col++) map[row][col] = INFINITY;
    }
    rows_read = row;
    for ( ; row < MAX_R; row++)
        for (col = 0; col < MAX_C; col++)
            map[row][col] = INFINITY;
    fclose(file);

    for (row = 0; row < MAX_R; row++)
        for (col = 0; col < MAX_C; col++) {
            dist[row][col] = INFINITY;
        parent[row][col][0] = -1;
        parent[row][col][1] = -1;
      }
    startr = startc = endr = endc = -1;
    return(0);
}

/* (void) map_draw_path()
   Ia akan dipanggil selepas limit_path() dipanggil, jikalau tidak
   user akan ditunjukkan semua path dari starting point.

   map_draw_path akan menunjukkan:-->

   '^' -->titik mula=starting point sebagai
   '$' -->titik tamat=ending point sebagai
   '+' -->jalan terdekat/intermediate squares in the path
   '.' -->jalan lain yg boleh dilalui/the walkable squares in the map
   '#' -->jalan yg mustahil/dinding.halangan/the impassible squares

   A square is considered to be in the path if it has a parent.
*/


void map_draw_path(char *fname){

int row, col;
int wt;
FILE *ip;
ip=fopen(fname,"w");
    wt = 0;
    for (row = 0; row < rows_read; row++) {
        for (col = 0; col < cols_read; col++) {
            if (startr == row && startc == col)
//                  fputc(ip,'^', stdout);
                    fprintf(ip,"^");
```

```
            else if (endr == row && endc == col)
//                  fputc(ip,'$', stdout);
                    fprintf(ip,"$");
            else if (parent[row][col][0] >= 0 && parent[row][col][1] >= 0) {
//                  fputc(ip,'+', stdout);
                    fprintf(ip,"+");
                    wt += map[row][col];
            } else if (map[row][col] < INFINITY)
//                  fputc(ip,'.', stdout);
                    fprintf(ip,".");
            else
//                  fputc(ip,'#', stdout);
                    fprintf(ip,"#");
        }
        fprintf(ip,"\n");
    }
    fprintf (ip,"\nThe weight of this path is: %d\n", wt);
        fclose(ip);
}


/* drow(row,i) dan * dcol(col,i) akan apply delta table utk menunjukkan
   row/column utk return petak/square around/disekeling 1 titik yg dispesifikkan
   delta table melistkan row/column utk 8 petak/squares disekeliling
   square/kotak (cth atas-kiri,atas,atas-kanan,kiri,kanan,bawah kiri,bawah,bawah kanan)
 */
#define drow(row,i) (row+delta[i][0])
#define dcol(col,i) (col+delta[i][1])

int delta[8][2] = { -1, -1, -1, 1, 1, -1, 1, 1, -1, 0, 0, 1, 1, 0, 0, -1 };

/* (void) build_illustrate();
 * Menunjukkan weights utk semua shortest paths.
 */
void
build_illustrate(void)
{
int i, j;
char line[256];

    for (i = 0; i < rows_read; i++) {
        for (j = 0; j < cols_read; j++)
            if (dist[i][j] < INFINITY)
                printf("%2d ", dist[i][j]);
            else
                printf(" I ");
        printf("\n");
```

```
    }
    printf("\nHIT ENTER:\n"); fflush(stdout);
    gets(line);
}

/*
* (void) build_path()
* menunjukkan algorithm:
* push the starting point onto the queue
* while there is an element of least weight in the queue, say (r,c)
* foreach square around (r,c), say (r',c') where
* dist[to (r,c)] + weight(r',c') < dist[to (r',c')]* do
* dist[to (r',c')] = dist[to (r,c)] + weight(r',c')
* push (r',c') onto the queue
*/


void
build_path(void)
{
PQUEUE * pq;
int     row, col;
int       i;

    pq = pqueue_new();
    if (startr < 0 || startr >= rows_read || startc < 0 || startc >= cols_read)
        return;
    dist[startr][startc] = 0;
    parent[startr][startc][0] = -1;
    parent[startr][startc][1] = -1;
    pqueue_insert(pq, startr, startc, 0);

    while (pqueue_popmin(pq, &row, &col) == 0) {
        /* DEBUG INFORMATION */
        pops++;
        if (debug_level > 0)
            printf("\nSelected (row,col)=(%d,%d).  Adding:\n", row, col);

        /* FOREACH square adjacent to (row,col), call it drow(row,i), dcol(col,i) */
        for (i = 0; i < 8; i++) {
            /* Check to make sure the square is on the map */
            if (drow(row,i) < 0 || drow(row,i) >= rows_read ||
                dcol(col,i) < 0 || dcol(col,i) >= cols_read)
                continue;

            /* Semak samada untuk lihat samaada shorters path dijumpai*/
            if (dist[row][col] + map[drow(row,i)][dcol(col,i)] <
```

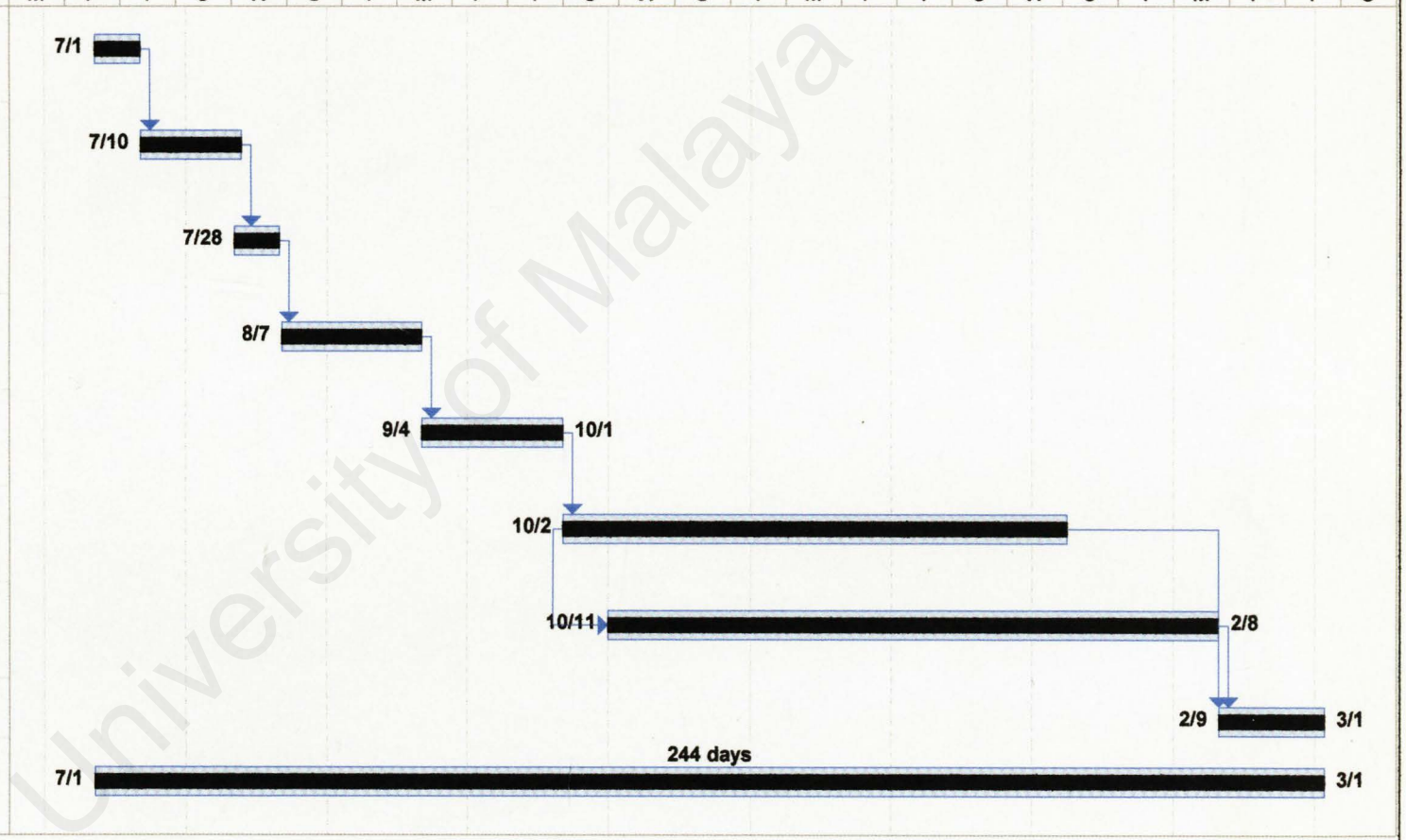
```
                dist[drow(row,i)][dcol(col,i)]) {
                /* DEBUG INFORMATION */
                if (debug_level > 0) {
                   printf("\t\t(%d,%d) Weight was/is: ", drow(row,i), dcol(col,i));
             printf("%d/%d\n", dist[drow(row,i)][dcol(col,i)],
map[drow(row,i)][dcol(col,i)] + dist[row][col]);
                }

                /* We have a shorter path, update the information */
                dist[drow(row,i)][dcol(col,i)] = map[drow(row,i)][dcol(col,i)] +
dist[row][col];
                parent[drow(row,i)][dcol(col,i)][0] = row;
                parent[drow(row,i)][dcol(col,i)][1] = col;
                /* Push the modified square onto the priority queue */
                pqueue_insert(pq, drow(row,i), dcol(col,i),
dist[drow(row,i)][dcol(col,i)]);
          } else if (debug_level >= 5) {
                printf("\t\t(%d,%d) Weight Stays: %d (would have been %d)\n",
drow(row,i), dcol(col,i), dist[drow(row,i)][dcol(col,i)], dist[row][col] +
map[drow(row,i)][dcol(col,i)]);
          }
       }
       if (debug_level >= 10) build_illustrate();
   }
   return;
}

// (void) limit_path(void)
// This routine will set the parent of every square which is not on the shortest path yg
dipilih


void
limit_path(void)
{
int row, col;
int tmp;

   if (endr < 0 || endr >= rows_read || endc < 0 || endc >= cols_read) return;
   row = endr;
   col = endc;
   do {
       dist[row][col] = -dist[row][col];
       tmp = parent[row][col][0];
       col = parent[row][col][1];
       row = tmp;
   } while(row >= 0 && col >= 0);
```

```
    for (row = 0; row < rows_read; row++)
        for (col = 0; col < cols_read; col++)
            if (dist[row][col] < 0)
                dist[row][col] = -dist[row][col];
            else
                parent[row][col][0] = -1;
    return;
}


int main(){

        printf ("\n========SIMULATION OF SHORTEST PATH USING
DIJKSTRA'S ALGORITHM========\n\n");
        printf("The starting grid map-->\n\n");

        map_read("map1.txt");

    printf("\nnaming format of node : (row,column)\n");
        printf("(0,0),(0,1),(0,2).(0,3),(0,4).....\n(1,0),(1,1),(1,2),(1,3),(1,4).....\n(2,0),(2,
1),(2,2),(2,3),(2,4).....\n(3,0),(3,1),(3,2),(3,3),(3,4).....\n.....");

        printf("\n\n=============================================================
==========================\nThe simulation-->");

        printf("\nMasukkan nilai debug level 0 , 5 atau 10 : ");
        scanf("%d",&debug_level);

        printf("\n\nEnter Starting Row    :");
        scanf("%d",&startr);

        printf("Enter Starting Column :");
        scanf("%d",&startc);

        printf("Enter Ending Row      :");
        scanf("%d",&endr);

        printf("Enter Ending Column   :");
        scanf("%d",&endc);

    pops = 0;
    build_path();
    if (debug_level < 10 && debug_level > 0) build_illustrate();
    limit_path();
    map_draw_path("output.txt");
    printf ("We required 2x %d queue operations\n", pops);
```

```
    printf ("N = rows x cols = %d x %d = %d\n", rows_read, cols_read, rows_read *
cols_read);
    printf ("Maximum memory consummed in the priority queue = %d\n",
max_memory);

        return 0;
}
```

JADUAL PERJALANAN PROJEK SIMULASI JARAK TERDEKAT MENGGUNAKAN ALGORITMA DIJKSTRA

ID	Task Name	Duration
1	Penyisatan Awalan	9 days
2	Analisa Masalah	20 days
3	Analisa Keperluan Simulasi	9 days
4	Analisa Sistem	28 days
5	Rekabentuk Simulasi	28 days
6	Pembangunan Simulasi	100 days
7	Pengujian Simulasi	121 days
8	Perbincangan/Kesimpulan	21 days
9	Dokumentasi	244 days



TARIKH MULA : Thu 7/1/04
TARIKH DIJANGKA HABIS : Tue 3/1/05

Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline