**NAMA : MERIZA CORRY NURSHINTA**

**NIM : 2014157027**

**KECERDASAN BUATAN**

**BEST FIRST SEARCH, ALGORITMA A\*, ALGORITMA DJIKSTRA**

**Metode Pencarian Best First Search**

|  |
| --- |
| **f’ = g + h’** |

*Best First Search* (BFS) merupakan suatu cara yang menggabungkan keuntungan atau kelebihan dari pencarian *Breadth First Search* dan *Depth First Search*. Pada setiap langkah proses pencarian terbaik pertama, kita memilih node-node dengan menerapkan fungsi heuristik yang memadai pada setiap node/simpul yang kita pilih dengan menggunakan aturan-aturan tertentu untuk menghasilkan penggantinya. Fungsi Heuristik yang digunakan merupakan prakiraan (estimasi) *cost* dari *initial state* ke *goal state*, yang dinyatakan dengan :

Dimana f’ = prakiraan cost dari initial ke goal g = cost dari initial state ke current state  
h’ = prakiraan cost dari *current state* ke *goal state*

*Best first search* (BFS)juga merupakan sebuah metode yang membangkit kan simpul dari simpul sebelumnya. *Best first search* memilih simpul baru yang memiliki biaya terkecil diantara  semua *leaf nodes* (simpul-simpul pada level terdalam) yang pernah dibangkitkan. Penentuan simpul terbaik dilakukan dengan menggunakan sebuah fungsi yang disebut fungsi evaluasi *f(n).* fungsi evaluasi best-first search dapat berupa biaya perkiraan dari suatu simpul menuju ke goal atau gabungan antara biaya sebenarnya dan biaya perkiraan tersebut.

Pada setiap langkah proses pencarian terbaik pertama, kita memilih node-node dengan menerapkan fungsi heuristik yang memadai pada setiap node/simpul yang kita pilih dengan menggunakan aturan-aturan tertentu untuk menghasilkan penggantinya. Fungsi heuristic merupakan suatu strategi untuk melakukan proses pencarian ruang keadaan suatu problema secara selektif, yang memandu proses pencarian yang kita lakukan sepanjang jalur yang memiliki kemungkinan sukses paling besar.

Ada beberapa istilah yang sering digunakan pada metode *best first search,* yaitu:

1.      Start node adalah sebuah terminology untuk posisi awal sebuah pencarian

2.      Curret node adalah simpul yang sedang dijalankan dalam algoritma pencarian jalan terpendek

3.      Suksesor adalah simpul-simpul yang yang akan diperiksa setelah current node

4.      Simpul (node) merupakan representasi dari area pencarian

5.      Open list adalah tempat menyimpan data simpul yang mungkin diakses dari starting node maupun simpul yang sedang dijalankan

6.      Closed list adalah tempat menyimpan data simpul yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan

7.      Goal node yaitu simpul tujuan

8.      Parent adalah curret node dari suksesor.

**2.3 Algoritma *Best First Search***

Algoritma best first search ini merupakan kombinasi dari algoritma depth first search  dengan algoritma breadth first search  dengan mengambil kelebihan dari kedua  algoritma tersebut. Apabila pada pencarian dengan algoritma  hill climbing tidak  diperbolehkan untuk kembali ke node pada level yang lebih rendah meskipun node di  level yang lebih rendah tersebut memiliki nilai heuristik yang lebih baik, lain halnya pada algoritma best first search, pencarian diperbolehkan mengunjungi node yang ada di level yang lebih rendah, jika ternyata node di level yang lebih tinggi memiliki nilai heuristik yang lebih buruk.

Algoritma best first search merupakan salah satu bagian dari tipe informed search. Algoritma ini menggunakan nilai-nilai heuristik tiap simpul yang dibuka. Simpul dengan nilai heuristik terbaik akan dibuka lebih dahulu. Bila goal state belum ditemukan, akan dilakukan pemeriksaan pada simpul berikutnya dengan nilai heuristik terbaik pada kedalaman yang sama. Simpul tersebut kemudian dibuka dan diperiksa apakah terdapat goal state pada cabang-cabangnya. Bila goal state belum ditemukan,  akan dilakukan proses yang sama pada simpul berikutnya.

Merupakan metode yang membangkitkan suksesor dengan mempertim- bangkan harga (didapat dari fungsi heuristik tertentu) dari setiap node, bukan dari aturan baku seperti DFS maupun BFS. Gambar 3.4 mengilustrasikan langkah-langkah yang dilakukan oleh algoritma *Best First Search*. Pertama kali, dibangkitkan node A. Kemudian semua suksesor A dibangkitan, dan dicari harga paling minimal. Pada langkah 2, node D terpilih karena harganya paling rendah, yakni 1. Langkah 3, semua suksesor D dibangkitkan, kemudian harganya akan dibandingkan dengan harga node B dan C. Ternyata harga node B paling kecil dibandingkan harga node C, E, dan F. Sehingga B terpilih dan selanjutnya akan dibangkitkan semua suksesor B. Demikian seterusnya sampai ditemukan node Tujuan.

Untuk mengimplementasikan algoritma pencarian ini, diperlukan dua buah senarai, yaitu: OPEN untuk mengelola node-node yang pernah dibangkitkan tetapi belum dievaluasi dan CLOSE untuk mengelola node-node yang pernah dibangkitkan dan sudah dievaluasi. Algoritma selengkapnya adalah sebagai berikut.

1.      OPEN berisi initial state dan CLOSED masih kosong.

2.      Ulangi sampai goal ditemukan atau sampai tidak ada di dalam OPEN.  
      a. Ambil simpul terbaik yang ada di OPEN.  
      b.  Jika simpul tersebut sama dengan goal, maka sukses  
      c.  Jika tidak, masukkan simpul tersebut ke dalam CLOSED  
      d. Bangkitkan semua aksesor dari simpul tersebut  
      e. Untuk setiap suksesor kerjakan.

Algoritma yang menggunakan metode best-first search, yaitu:

**a.      Greedy Best-First**

Greedy Best-First adalah algoritma best-first yang paling sederhana dengan hanya memperhitungkan biaya perkiraan (estimated cost) saja, yakni *f(n) = h(n).* Biaya yang sebenarnya (actual cost) tidak diperhitungkan. Dengan hanya memperhitungkan biaya perkiraan yang belum tentu kebenarannya, maka algoritma ini menjadi tidak optimal.

**b.      A\***

Algoritma A\* merupakan algoritma best first search dengan modifikasian fungsi heuristik, yang akan meminimumkan total biaya lintasan, dan pada kondisi yang tepat  akan memberikan solusi yang terbaik dalam waktu yang optimal.

Algoritma A juga membutuhkan dua antrian, yaitu OPEN dan CLOSED. Selain itu, ada juga fungsi heuristik yang memprediksi keuntungan tiap node yang di buat. Yang akan memungkinkan algoritma untuk melakukan pencarian-pencarian lintasan yang lebih di harapkan. Fungsi ini di sebut f’(n) sebagai pendekatan dari fungsi f(n) yang merupakan fungsi evaluasi yang sebenarnya terhadap node n. dalam banyak penarapan, akan lebih baik jika fungsi di definisikan sebagai kombinasi atau jumlah dua komponen yaitu g(n) dan h(n). Fungsi g(n) merupakan ukuran biaya yang di keluarkan dari keadaan awal sampai ke node n. Nilai yang didapat g(n) merupakan jumlahan biaya penerapan setiap aturan yang dilakukan pada sepanjang lintasan trbaik menuju suatu simpul dan bukan merupakan hasil estimasi.

Fungsi h(n) merupakan pengukur biaya tambahan yang harus dikeluarkan dari node n sampai mendapatkan tujuan. Perlu diketahui bahwa g(n), tidak negatif karena bila negatif maka lintasan yang membalik siklus pada graf akan tampak lebih baik dengan semakin panjangnya lintasan.

Secara matematis,fungsi F sebagai estimasi fungsi evaluasi terhadap node ndapat di tuliskan :

        f’(n) = g(n) + h’(n)

Dengan f’(n) = fungsi evaluasi

    g(n) = biaya yang sudah di keluarkan dari keadaan awal sampai

                keadaan n

     h’(n) = estimasi biaya untuk sampai pada suatu tujuan mulai dari n

dari fungsi di atas maka ada beberapa kondisi yang perlu diperhatikan, yaitu:

Jika h = h’, berarti proses pencarian telah sampai ke tujuan ( goal ).

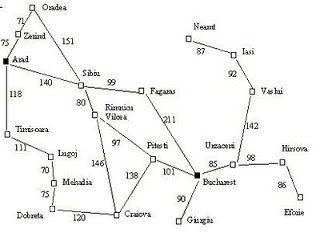
Jika g = h’ = 0 maka f’ random, artinya system tidak dapat di kendalikan.

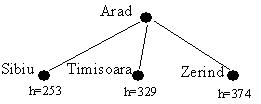
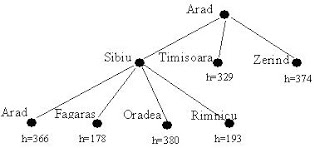
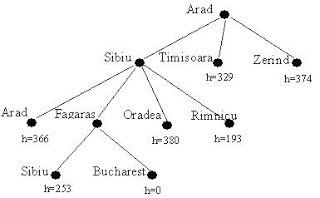
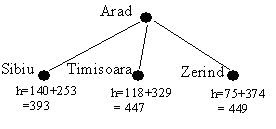
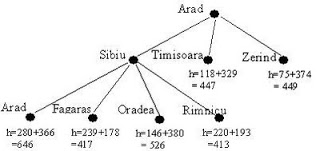
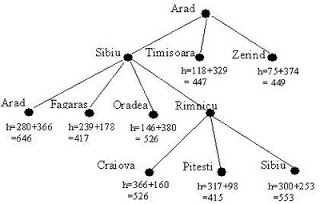
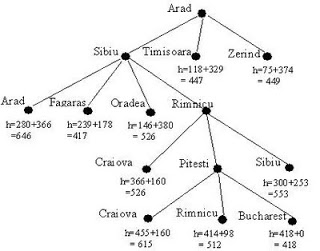
  Jika g = k, k adalah konstanta dan biasanya bernilai 1, h’ = 0, artinya system

menggunakan breadth first search.

**CONTOH KASUS BFS & A\*:**

## Best-First Search VS A\* (Arad-Bucharest)

Tidak seperti Depth-First Search (DFS) atau Breadth-First Search BFS), Best-First Search adalah algoritma pencarian dengan menggunakan heuristic. Berikut adalah contoh penggunaan metode BFS untuk permasalahan Arad-Bucharest.  
Berikut ini adalah peta Romania dengan jarak jalan-jalan yang menghubungkan kota-kota dalam km.  
[](http://4.bp.blogspot.com/_NpbmFwsFQhU/Sp7zQbeKksI/AAAAAAAAAA8/eV5TMj5lYjs/s1600-h/arad1.JPG)  
Adapun jarak kota-kota terhadap kota Bucharest jika ditarik garis lurus adalah sebagai berikut  
Arad 366  
Bucharest 0  
Craiova 160  
Dobreta 242  
Eforie 161  
Fagaras 178  
Giurgiu 77  
Hirsova 151  
Iasi 226  
Lugoj 244  
Mehadia 241  
Neamt 234  
Oradea 380  
Pitesti 98  
Rimnicu Vilcea 193  
Sibiu 253  
Timisoara 329  
Urziceni 80  
Vaslui 199  
Zerind 374  
Permasalahannya adalah untuk mencari jalan terdekat dari kota Arad menuju kota Bucharest dengan menggunakan metoda Best First Search dan A\*.

**Best First Search**  
  
Heuristik yang digunakan adalah jarak kota-kota terhadap kota Bucharest jika ditarik garis lurus yang jaraknya seperti yang tertera di atas dengan asumsi kota terhubung yang letaknya paling dekat dengan kota Bucharest adalah jalan yang paling optimal.  
  
Diagram pohon langkah-langkah penelusuran dengan metode Best First Search adalah sebagai berikut :  
  
Langkah 1  
[http://3.bp.blogspot.com/_NpbmFwsFQhU/Sp7zppf41gI/AAAAAAAAABE/UcSP11E_WLY/s320/arad2.JPG](http://3.bp.blogspot.com/_NpbmFwsFQhU/Sp7zppf41gI/AAAAAAAAABE/UcSP11E_WLY/s1600-h/arad2.JPG)  
Langkah 2  
  
Langkah 3  
[](http://1.bp.blogspot.com/_NpbmFwsFQhU/Sp7z9U6n-eI/AAAAAAAAABU/IeJUExBdJ2s/s1600-h/arad4.JPG)  
Langkah 4  
[](http://3.bp.blogspot.com/_NpbmFwsFQhU/Sp70FlhYHyI/AAAAAAAAABc/X6v-UJbXbhs/s1600-h/arad5.JPG)  
Dari Langkah-langkah di atas, dengan metode Best First Search didapatkan kota-kota yang harus dilalui untuk mendapatkan jalan yang paling dekat jaraknya dari Arad ke Bucharest adalah : Arad – Sibiu – Fagaras – Bucharest. Dari peta di atas, panjang jalan yang dilalui adalah 140+99+211 = 450 km.  
  
**A\* Algorthm**  
A\* Algorithm menggunakan dua fungsi cost sebagai acuan penelusuran yaitu jarak yang telah dilalui dari kota asal Arad ke kota tersebut ditambah dengan heuristik jarak kota-kota terhadap kota Bucharest jika ditarik garis lurus seperti yang digunakan pada Best First Search, jadi fungsi cost f(n) = g(n) + h(n).  
  
Diagram pohon langkah-langkah penelusuran dengan metode A\* adalah sebagai berikut :  
Langkah 1  
[](http://2.bp.blogspot.com/_NpbmFwsFQhU/Sp8rDcynDlI/AAAAAAAAABk/O6eh4OH33V8/s1600-h/arad2.JPG)Langkah 2  
[](http://4.bp.blogspot.com/_NpbmFwsFQhU/Sp8rM7gHq-I/AAAAAAAAABs/hNkcuIbNGZQ/s1600-h/arad3.JPG)Langkah 3  
[](http://1.bp.blogspot.com/_NpbmFwsFQhU/Sp8rVrK1oII/AAAAAAAAAB0/qZOkjASEDNQ/s1600-h/arad4.JPG)Langkah 4  
[](http://3.bp.blogspot.com/_NpbmFwsFQhU/Sp8rjwpquYI/AAAAAAAAAB8/5njFmryF0ug/s1600-h/arad5.JPG)Langkah 5  
[](http://2.bp.blogspot.com/_NpbmFwsFQhU/Sp8rvfqXcoI/AAAAAAAAACE/jUNpQABjmSE/s1600-h/arad6.JPG)  
Dari Langkah-langkah di atas, dengan metode A\* didapatkan kota-kota yang harus dilalui untuk mendapatkan rute jalan yang paling dekat jaraknya dari Arad ke Bucharest adalah : Arad – Sibiu – Rimnicu – Pitesti – Bucharest. Dari peta di atas, panjang jalan yang dilalui adalah 140+80+97+101 = 418 km.  
  
Jika dibandingkan dengan hasil metode Best First Search, penelusuran dengan metode A\* untuk mencapai goal lebih besar kedalamannya (BFS=4 langkah, A\* = 5 langkah). Begitu juga Node yang dieksplore, metode A\* lebih banyak daripada Best First Search (BFS=10 node, A\*=14 node). Akan tetapi, rute yang dihasilkan dengan metode A\* lebih baik dari Best First Search (BFS=450 km, A\*= 418 km).  
  
Jelas bahwa jika masalah yang dihadapi diprioritaskan kepada cost yang optimum, metode A\* lebih baik digunakan, sedangkan jika kecepatan mencapai goal lebih diprioritaskan maka metode Best First Search lebih baik digunakan.

**Pengertian dan Teori Algoritma Dijkstra**

**Algoritma Dijkstra**, (dinamai menurut penemunya, seorang ilmuwan komputer, Edsger Dijkstra), adalah sebuah algoritma rakus (*greedy algorithm*) yang dipakai dalam memecahkan permasalahan jarak terpendek (*shortest path problem*) untuk sebuah graf berarah (*directed graph*) dengan bobot-bobot sisi (*edge weights*) yang bernilai tak-negatif.  
Misalnya, bila *vertices* dari sebuah graf melambangkan kota-kota dan bobot sisi (*edge weights*) melambangkan jarak antara kota-kota tersebut, maka algoritma Dijkstra dapat digunakan untuk menemukan jarak terpendek antara dua kota.  
  
Input algoritma ini adalah sebuah graf berarah yang berbobot (*weighted directed graph*) *G* dan sebuah sumber *vertex* *s* dalam *G* dan *V* adalah himpunan semua vertices dalam graph *G*.  
Setiap sisi dari graf ini adalah pasangan vertices (*u*,*v*) yang melambangkan hubungan dari *vertex* *u* ke *vertex* *v*. Himpunan semua tepi disebut *E*.  
Bobot (*weights*) dari semua sisi dihitung dengan fungsi

*w*: *E* → [0, ∞)

jadi *w*(*u*,*v*) adalah jarak tak-negatif dari vertex *u* ke vertex *v*.  
Ongkos (*cost*) dari sebuah sisi dapat dianggap sebagai jarak antara dua *vertex*, yaitu jumlah jarak semua sisi dalam jalur tersebut. Untuk sepasang vertex *s* dan *t* dalam *V*, algoritma ini menghitung jarak terpendek dari *s* ke *t*.

**Tujan Algoritma Dijkstra**

* Tujuan Algoritma Dijkstra yaitu untuk menemukan jalur terpendek berdasarkan bobot terkecil dari satu titik ke titik lainnya.
* Kelemahan algoritma ini adalah semakin
* banyak titik akan semakin memakan waktu proses.
* Jumlah titik menentukan tingkat efektifitas dari algoritma djikstra.

**Urutan Logika Algoritma Dijkstra**

1. Beri nilai bobot (jarak) untuk setiap titik ke titik lainnya, lalu set nilai 0 pada node awal dan nilai tak hingga terhadap node lain (yang belum terisi).
2. Set semua node “Belum terjamah” dan set node awal sebagai “Node keberangkatan”.
3. Dari node keberangkatan, pertimbangkan node tetangga yang belum terjamah dan hitung jaraknya dari titik keberangkatan.
4. Setelah selesai mempertimbangkan setiap jarak terhadap node tetangga, tandai node yang telah terjamah sebagai “Node terjamah”. Node terjamah tidak akan pernah di cek kembali, jarak yang disimpan adalah jarak terakhir dan yang paling minimal bobotnya.
5. Set “Node belum terjamah” dengan jarak terkecil (dari node keberangkatan) sebagai “Node Keberangkatan” selanjutnya dan lanjutkan dengan kembali ke step 3.

**Contoh :**

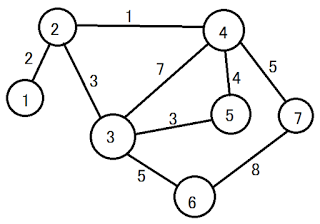
Titik mengambarkan lokasi dan garis menggambarkan jalan, maka algoritma Dijkstra melakukan kalkulasi terhadap semua kemungkinan bobot terkecil dari setiap titik.

Pertama tentukan titik yang akan menjadi nomor kode (node) awal, lalu beri bobot jarak pada node pertama ke node terdekat satu per satu, lalu akan dilakukan pengembangan pencarian dari satu titik ke titik selanjutnya (tahap demi tahap).

1. Node awal 1, Node tujuan 5. Setiap edge yang terhubung antar node telah diberi nilai.
2. Dijkstra melakukan kalkulasi terhadap node tetangga yang terhubung langsung dengan node keberangkatan (node 1), dan hasil yang didapat adalah node 2 karena bobot nilai node 2 paling kecil dibandingkan nilai pada node lain, nilai = 7 (0+7).
3. Node 2 diset menjadi node keberangkatan dan ditandai sebagi node yang telah terdatangi. Dijkstra melakukan kalkulasi kembali terhadap node-node tetangga yang terhubung langsung dengan node yang telah terdatangi. Dan kalkulasi dijkstra menunjukan bahwa node 3 yang menjadi node keberangkatan selanjutnya karena bobotnya yang paling kecil dari hasil kalkulasi terakhir, nilai 9 (0+9).
4. Perhitungan berlanjut dengan node 3 ditandai menjadi node yang telah terjamah. Dari semua node tetangga belum terjamah yang terhubung langsung dengan node terjamah, node selanjutnya yang ditandai menjadi node terjamah adalah node 6 karena nilai bobot yang terkecil, nilai 11 (9+2).
5. Node 6 menjadi node terjamah, dijkstra melakukan kalkulasi kembali, dan menemukan bahwa node 5 (node tujuan ) telah tercapai lewat node 6. Jalur terpendeknya adalah 1-3-6-5, dan niilai bobot yang didapat adalah 20 (11+9). Bila node tujuan telah tercapai maka kalkulasi dijkstra dinyatakan selesai.

Contoh soal :

Mari kita pakai contoh jaringan yang sama dengan yang ada pada postingan Bellman-Ford yang lalu:



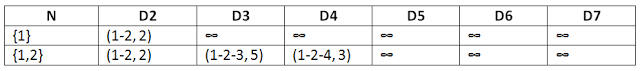
Misalnya kita akan menggunakan algoritma Dijkstra untuk mencari path terpendek dari node 1.

Untuk mempermudah, buatlah tabel seperti berikut ini:

(Notasi dalam tabel algoritma Dijkstra memiliki format (s-j,D), dimana s-j menunjukkan rute dari node s menuju node j, sementara D menunjukkan jarak total antara kedua node tersebut)

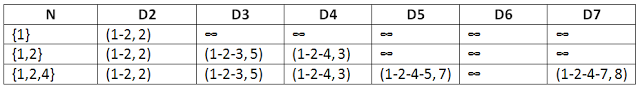
Baris pertama masih berupa inisialisasi, yaitu Dj akan memiliki nilai jika tersambung langsung dan tidak memiliki nilai jika tidak tersambung langsung.

Karena node 1 kebetulan hanya memiliki 1 tetangga yaitu node 2, maka i = 2 dimasukkan pada himpunan N.



Node 2 sudah berperan sebagai "perpanjangan" node sumber (node 1), sehingga sekarang node-node yang terhubung dengan node 2 sudah bisa "dijangkau" oleh node 1 via node 2. Diketahui node 3 dan node 4 terhubung langsung dengan node 2, sehingga rutenya ditulis (1-2-3) dan (1-2-4).

Untuk langkah selanjutnya dipilih node i yang telah tersambung dengan node s namun belum masuk dalam himpunan N. Diketahui yaitu node 3 dan node 4. Node yang dipilih adalah yang memiliki jumlah jarak yang paling minimum, yaitu node 4. Sehingga didapat baris tabel berikutnya seperti berikut.



Seperti langkah yang sebelumnya, sekarang node 4 ikut berperan sebagai "perpanjangan" dari node 1 sehingga node 5 dan node 7 yang terhubung langsung dengan node 4 sudah bisa "dijangkau" oleh node 1 dengan rute yang tertampil.

Sebenarnya disini mulai terjadi perbandingan nilai jarak. Dengan dimasukkannya node 4 dalam himpunan N, maka node 4 berlaku sebagai "penembak sementara" (maafkan saya kalau istilahnya alay -\_-). Maksudnya adalah node 4 dapat melakukan perhitungan minimum menuju node tertentu meskipun node tersebut telah diketahui rute dan jaraknya.

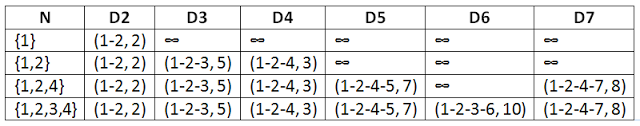
Dalam kasus ini dapat diambil node 3. Node 3 sudah diketahui rute dan jaraknya pada baris ke-2. Dengan masuknya node 4 pada himpunan N, maka node 4 akan menghitung jarak minimum antara entry D3 yang terdahulu dengan yang baru. (Perbandingan via node 2 dengan via node 4):

Via node 2 --> D2 + C23 = 2 + 3 = 5

Via node 4 --> D4 + C43 = 3 + 7 = 10

Maka entry yang dipertahankan adalah via node 2 dengan jarak 5.

Sebelumnya telah dipilih node 4 sebagai anggota N karena bertetangga dengan node 2. Sekarang dipilih tetangga node 2 yang lainya yaitu node 3.



Perbandingan yang terjadi:

Pada D4

Via node 2 --> D2 + C24 = 2 + 1 = 3

Via node 3 --> D3 + C34 = 5 + 7 = 12

Maka entry yang dipertahankan adalah via node 2 dengan jarak 3.

Pada D5

Via node 4 --> D4 + C45 = 3 + 4 = 7

Via node 3 --> D3 + C35 = 5 + 3 = 8

Maka entry yang dipertahankan adalah via node 4 dengan jarak 7.

Singkat cerita, tabel hasil akhirnya adalah sebagai berikut.

