Algoritma Bellman-Ford dalam Distance Vector Routing Protocol

Galih Andana – NIM : 13507069

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
E-mail : if17069@students.if.itb.ac.id

ABSTRAK

Sebuah routing vektor jarak-protokol adalah salah satu dari dua kelas utama dari routing protokol yang digunakan dalam paket-switched network untuk komunikasi komputer, dan kelas lainnya adalah protokol link-state. Algoritma Bellman-Ford di sini dipakai untuk menghitung path terpendek.
Sebuah vektor jarak memerlukan routing protokol yang menginformasikan router tetangganya dari perubahan topologi secara berkala atau bila ada perubahan yang terdeteksi dalam topologi jaringan. Dibandingkan dengan protokol link-state, yang membutuhkan sebuah router untuk menginformasikan semua simpul dalam tiap perubahan topologi jaringan, routing vektor jarak-protokol ini memiliki komputasi yang lebih mudah.
Pada makalah ini, penulis akan membahas lebih jauh mengenai apa itu Bellman-Ford dan memberikan sedikit perbandingan dengan algoritma routing Djikstra. Di sini juga akan diberi contoh seperti apa penerapan Bellman-Ford dengan studi kasus yang telah penulis praktekan menggunakan bahasa pemrograman java.

Kata kunci: Routing, Bellman-Ford, Djikstra, tetangga.

1. PENDAHULUAN

Para algoritma Bellman-Ford, kita menghitung satu sumber jalan terpendek dalam sebuah graf berbobot (bahkan di sini beberapa sisi dimungkin memiliki bobot negatif). Algoritma Dijkstra menyelesaikan masalah yang sama dengan berjalan waktu yang lebih rendah, tetapi memerlukan batasan di mana bobot pathnya harus non-negatif. Jadi, Bellman-Ford biasanya digunakan hanya bila ada sisi negatif. Algoritma ini diberi nama Bellman-Ford karena dikembangkan oleh Richard Bellman dan Lester Ford, Jr.

Menurut Robert Sedgewick, "Bobot negatif bukan hanya hasil keingintahuan matematis; mereka muncul secara alami ketika kita mempertimbangkan masalah-masalah lain untuk menentukan jalan mana yang terpendek", beliau memberikan contoh spesifik suatu pengurangan dari masalah NP-lengkap graf Hamilton untuk mencari jalan terpendek dengan beban umum. Jika sebuah graf berisi siklus dari total bobot jarak yang negatif kemudian dianggap sebagai beban terendah sehingga tidak ada solusi; Bellman-Ford mendeteksi kasus ini.

Jika grafik berisi siklus bobot negatif, Bellman-Ford dapat mengatasi hal ini; Tapi Bellman-Ford tidak dapat menemukan jalan terpendek yang tidak melalui sisi yang sama pada setiap sisi dalam suatu grafik. Masalah ini disebut masalah NP-lengkap jalan terpanjang.

Metode ini digunakan untuk menghitung jalan terbaik untuk jaringan yang berbeda antara routing protokol yang berbeda namun fitur dasar dari algoritma ini sama dengan algoritma lain yang berbasis *Distance Vector* di semua protokol.

Distance Vector berarti bahwa Router diposisikan sebagai vektor yang memiliki jarak dan arah. Arah hanyalah alamat hop berikutnya dan keluar interface dan Jarak berarti bobot menuju hop tersebut.

Router yang menggunakan protokol vektor jarak tidak memiliki cukup pengetahuan **seluruh jalur dari *source* ke *destination*,** bahkan pada awalnya ia hanya mengetahui jarak antara dia dan tetangganya. Oleh sebab itu DV memiliki dua keterangan.

1. Ke mana arah atau *interface* yang harus dilalui oleh paket untuk dapat diteruskan.
2. Bobot jarak dari tujuan tersebut.

Seperti namanya DV protokol ini menitik beratkan pada perhitungan arah dan jarak untuk setiap link dalam sebuah jaringan. Biaya untuk mencapai tujuan dihitung menggunakan berbagai rute metrik. RIP menggunakan hop tujuan sedangkan IGRP mempertimbangkan informasi lain seperti delay simpul dan bandwidth yang tersedia.

Update dilakukan secara periodik pada protokol vektor-jarak di mana semua atau bagian dari tabel routing milik router dikirim ke semua tetangga yang dikonfigurasi untuk menggunakan bobot tersebut dari protokol routing vektor-jarak yang sama. RIP mendukung cross-platform, sedangkan routing vektor jarak IGRP adalah sebuah Cisco Systems protokol routing vektor jarak proprietary. Sekali sebuah router yang memiliki informasi ini dapat mengubah tabel routing miliknya sendiri untuk

mencerminkan perubahan, ia kemudian memberitahu para tetangganya mengenai perubahan tersebut. Proses ini telah digambarkan sebagai '*routing by rumor*' karena router hanya mengandalkan informasi yang mereka terima dari router lain dan tidak bisa menentukan apakah informasi tersebut benar-benar sah atau tidak.

Algoritma Bellman-Ford tidak mencegah terjadinya *routing loop* dan konsekuensi dari perhitungan untuk bobot tak terhingga. Inti dari hitungan untuk bobot tak terhingga adalah bahwa jika A mengatakan B yang memiliki jalan di suatu tempat, tidak ada cara untuk B untuk tahu apakah jalan memiliki B sebagai sebuah bagian dari itu. Untuk melihat masalah dengan jelas, bayangkan sebuah subnet terhubung seperti ABCDEF, dan membiarkan metrik antara router menjadi "jumlah lompatan". Sekarang anggaplah bahwa A akan turun (dari order). Dalam vektor-update-proses B pemberitahuan bahwa rute terpendek dari 1 sampai A sedang *down* - B tidak menerima update dari vektor A. Masalahnya adalah, B masih mendapatkan sebuah update dari C, dan C masih belum sadar fakta bahwa A kini sedang *down* - sehingga memberitahu B bahwa untuk pergi ke A, C hanya perlu melompat dua kali. Hal ini perlahan-lahan menjalar melalui jaringan sampai mencapai infinity (dalam hal ini mengoreksi algoritma itu sendiri, karena adanya "*Rilex property*" dari Bellman Ford).

Bellman-Ford memiliki struktur dasar yang sangat mirip dengan Algoritma Dijkstra, tapi ia tidak dengan rakus memilih simpul dengan berat minimum yang belum diproses, dan melakukan ini | V | - 1 kali, di mana | V | adalah jumlah simpul pada grafik. Repetisi tersebut memungkinkan jarak terpendek secara akurat menyebar ke seluruh grafik, karena, dengan tidak adanya siklus negatif, jalan terpendek hanya dapat mengunjungi setiap simpul paling banyak satu kali. Berbeda dengan Algoritma Greedy pada umumnya, yang tergantung pada asumsi struktural tertentu berasal dari bobot positif, pendekatan langsung ini meluas ke kasus umum.

Bellman-Ford berjalan dalam O(|V| · |E|) waktu, dimana |V| dan |E| adalah jumlah simpul dan sisi masing-masing. Ide penyelesaiannya adalah :

- Inisialisasi tabel shortest path dari source ke semua verteks lain kecuali dari source ∞ ke source 0.
- Relaksasi tabel shortest path dalam n-1 iterasi setiap pasangan verteks u dan v jika sisi (u,v) ada.

Namun dengan syarat : tidak ada siklus negatif karena tidak ada solusi.

2. METODE

Algoritma Bellman-Ford dipergunakan untuk mencari jarak terpendek pada router apabila ada sisi yang negatif sehingga tidak dapat diselesaikan dengan algoritma Djikstra. Namun algoritma ini tidak mampu menyelesaikan masalah apabila ada siklus yang bernilai negatif.

Bobot negatif pada sebuah *path* mungkin saja akan muncul dalam masalah modeling (misalnya, seorang sopir harus membayar uang masuk ketika melalui jalan yang lebih singkat, tetapi tidak untuk jalan yang lain).

2.1 Penjelasan Langkah Algoritma

Anggap Γ = (*V,A*) adalah graf berarah dan $\ell : A \rightarrow \mathbb{R}$ adalah fungsi jarak pada sisi Γ. Ibaratkan kita mencari jarak terpendek, w.r.t.ℓ, *path* langsung dari $v \in V$ ke semua verteks Γ. Kita definisikan $d_k(u)$ menjadi panjang minimal dari seluruh *k*-sisi dari v ke $u \in V$.

Tentunya, $d_0(v) = 0$ (karena graf tidak boleh memiliki siklus negatif dan $d_0(u) = \infty$ untuk semua $v \neq u \in V$.

Kemudian dengan menggunakan induksi, $d_k(u) = min(d_{k-1}(u), \ell(w,u) : (w,u) \in A)$. Hal ini baik, tapi apakah k dapat tumbuh tiba-tiba? Jika kita mengunjungi verteks lebih dari sekali, ini berarti graf mengandung siklik, kecuali bila siklik tersebut mengandung bobot negatif, kita dapat menghilangkannya, dan tidak menambah panjang jalan. Jadi, tanpa kehilangan secara general, sebuah jalan terpendek ke u memiliki hampir semua verteks |V|, dan juga $d_n(u)$ adalah jarak dari v ke u.

Maka pada hampir semua O(|V| · |E|) operasi, kita akan menemukan jarak dari v ke setiap verteks dari Γ.

2.2 Langkah Algoritma

Algoritma ini, seperti algoritma djikstra menggunakan tepi relaksasi tetapi tidak memakai metode greedy. Bahkan, ini menggunakan d[u] sebagai batas atas pada jarak d[u,v] dari u ke v.

Algortima ini secara berkala mengurangi perkiraan d[v] pada bobot dari jalan terpendek dari sumber verteks s ke tiap verteks v pada V hingga memperoleh jalan terpendek. Berikut ini penulis melampirkan langkah-langkah penyusunan algoritmanya yang akan mengembalikan nilai TRUE jika pada graf tidak terdapat siklus yang negatif dan akan mengembalikan nilai FALSE jika sebaliknya.

BELLMAN-FORD (G, *w*, *s*)

1. INITIALIZE-SINGLE-SOURCE (G, *s*)
2. for each vertex *i* = 1 to V[G] - 1 do
3. for each edge (u, *v*) in E[G] do
4. RELAX (u, *v*, *w*)
5. For each edge (u, *v*) in E[G] do
6. if d[u] + *w*(u, *v*) < d[*v*] then
7. return FALSE
8. return TRUE

Inisialisasi pada line 1 memerlukan waktu O(v) time. Untuk loop dari line 2-4 memerlukan waktu O(E) dan untuk line 5-7 memerlukan O(E) time pula. Jadi, Bellman-Ford berjalan dalam kompleksitas waktu O(E) time.

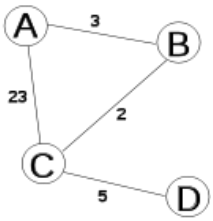
2.3 Pseudocode Algoritma Bellman-Ford

```
procedure BellmanFord(list vertices, list edges, vertex source)
   // This implementation takes in a graph, represented as lists of vertices
   // and edges, and modifies the vertices so that their distance and
   // predecessor attributes store the shortest paths.

   // Step 1: Initialize graph
   for each vertex v in vertices:
       if v is source then v.distance := 0
       else v.distance := infinity
       v.predecessor := null

   // Step 2: relax edges repeatedly
   for i from 1 to size(vertices)-1:
       for each edge uv in edges: // uv is the edge from u to v
           u := uv.source
           v := uv.destination
           if u.distance + uv.weight < v.distance:
               v.distance := u.distance + uv.weight
               v.predecessor := u

   // Step 3: check for negative-weight cycles
   for each edge uv in edges:
       u := uv.source
       v := uv.destination
       if u.distance + uv.weight < v.distance:
           error "Graph contains a negative-weight cycle"
```

Gambar 1. Pseudocode menurut wikipedia

2.4 Contoh Routing dengan Bellman-Ford

Dalam jaringan ini kita punya 4 router A, B, C, D:



Gambar 2. Router A, B, C, D dan Bobot Sisinya

Penulis menandai waktu saat ini (atau iterasi) dalam algoritma dengan T, dan akan mulai (pada waktu 0, atau T = 0) dengan membuat matriks jarak untuk setiap router ke tetangga terdekatnya. tabel routing di bawah ini, jalan terpendek akan disorot dengan warna hijau, dan update untuk jalan terpendek baru disorot dengan warna kuning.

dari A	via A	via B	via C	via D
A				
B		3		
C			23	
D				

dari B	via A	via B	via C	via D
A	3			
B				
C			2	
D				

dari C	via A	via B	via C	via D
A	23			
B		2		
C				
D				5

dari D	via A	via B	via C	via D
A				
B				
C			5	
D				

Tabel 1 Routing Table saat T=0

Pada titik ini, semua router (A, B, C, D) memiliki "jalan-terpendek" untuk DV mereka (daftar jarak yang berasal dari mereka ke router lain melalui tetangga). Mereka masing-masing mengirimkan DV baru ini kepada semua tetangga mereka: A ke B dan C, B ke C dan A, C ke A, B, dan D, dan D ke C. tetangga masing-masing menerima informasi ini, kemudian menghitung ulang yang jalur terpendek digunakannya.

Sebagai contoh: A menerima sebuah DV dari C yang menyatakan A ada jalur melalui C ke D, dengan jarak (atau biaya) 5. Karena saat ini "jalan terpendek" ke C adalah 23, maka A tahu itu jalan ke D bahwa biaya 23 +5 = 28. Karena tidak ada jalan lain yang lebih pendek yang diketahui A, ia menempatkan ini sebagai perkiraan arus-jalan terpendek dari dirinya sendiri (A) ke D, via C.

from A	via A	via B	via C	via D
to A				
to B		3	25	
to C		5	23	
to D			28	

from B	via A	via B	via C	via D
to A	3		25	
to B				
to C	26		2	
to D			7	

from C	via A	via B	via C	via D
to A	23	5		
to B	26	2		
to C				
to D				5

from D	via A	via B	via C	via D
to A			28	
to B			7	
to C			5	
to D				

Tabel 2 Routing Table saat T=1

Sekali lagi, semua router telah diperoleh dalam iterasi terakhir (pada T = 1) "jalan terpendek", sehingga mereka semua menyiarkan DV mereka pada tetangga mereka; Hal ini mendorong masing-masing tetangga untuk kembali menghitung jarak terpendek mereka lagi.

Sebagai contoh: A menerima sebuah DV dari B yang memberitahu A ada jalur melalui B ke D, dengan jarak (atau biaya) dari 7. Karena saat ini "jalan-terpendek" untuk B adalah 3, maka A mengetahui ada jalan ke D dengan biaya 7 +3 = 10. Jalan ke D ini dengan panjang 10 (melalui B) lebih pendek daripada "jalan-terpendek" ke D dengan panjang 28 (melalui C), sehingga 10 tersebut menjadi "jalan-terpendek" ke D yang baru.

from A	via A	via B	via C	via D
to A				
to B		3	25	
to C		5	23	
to D		10	28	

from B	via A	via B	via C	via D
to A	3		7	
to B				
to C	8		2	
to D	31		7	

from C	via A	via B	via C	via D
to A	23	5		33
to B	26	2		12
to C				
to D	51	9		5

from D	via A	via B	via C	via D
to A			10	
to B			7	
to C			5	
to D				

Tabel 3 Routing Table saat T=2

Kali ini, hanya router A dan D yang memiliki jalan terpendek baru untuk dvs. Jadi, mereka menyiarkan DV baru mereka kepada tetangga mereka: A mengirim ke B

dan C, dan D mengirim pada C. Hal ini menyebabkan masing-masing dari tetangga menerima DV baru, dan kembali menghitung jalan terpendek mereka. Namun, karena informasi dari DV tidak menghasilkan apa pun jalan-jalan lebih pendek daripada yang sudah ada di tabel routing mereka, maka tidak ada perubahan pada tabel routing.

from A	via A	via B	via C	via D
to A				
to B		3	25	
to C		5	23	
to D		10	28	

from B	via A	via B	via C	via D
to A	3		7	
to B				
to C	8		2	
to D	13		7	

from C	via A	via B	via C	via D
to A	23	5		15
to B	26	2		12
to C				
to D	33	9		5

from D	via A	via B	via C	via D
to A			10	
to B			7	
to C			5	
to D				

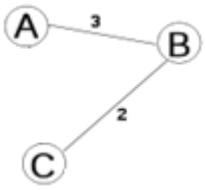
Tabel 4 Routing Table saat T=3

Tidak ada router lagi yang memiliki "jalan terpendek" baru. Maka tidak ada router yang memperoleh informasi baru yang memungkinkan terjadinya perubahan pada routing table mereka. Dengan kata lain algortima ini selesai.

2.5 Kelemahan Algoritma dan Solusinya

Pada algoritma ini apabila kita menghadapi perubahan link cost pada suatu node, kita harus melakukan update pada DV yang ada. Bila cost yang diubah menjadi lebih kecil, seluruh DV akan otomatis berubah karena ada penggantian link cost yang lebih minimum. Namun, apabila kita mengganti link cost dengan angka yang lebih besar akan ada kemungkinan terjadinya *infinite loop* (loop abadi). Hal tersebut juga dapat terjadi apabila ada permasalah pemutusan link dari suatu node dengan mengganti bobot linknya menjadi 999.

Berikut penjelasan secara rincinya dengan contoh berikut.



Gambar 3. Kasus pada Router A, B, C

Pada mulanya routing table yang terbentuk di A adalah

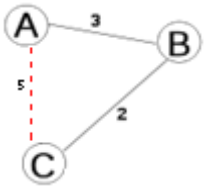
A-B = 3
A-C = 5

sedangkan di B adalah

B-A = 3
B-C = 2

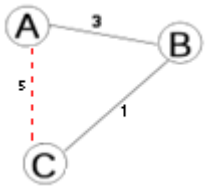
dan di C adalah

C-A = 5
C-B = 2



Gambar 4. Seakan-akan Ada Path pada Router A-C, C-A

Hal ini akan terus berlangsung selama tidak terjadi perubahan. Bila terjadi proses update link cost antara B dan C menjadi 1.



Gambar 5. Update Cost pada Router B-C menjadi 1

Maka B akan mengirimkan paket ke tetangganya (A, C) bahwa ada jalan baru dari B-C dengan cost 1. Karena C memiliki cost dari C-B dengan bobot minimum 2, sedangkan ada update dari B bahwa ada jalan yang melalui B-C dengan bobot 1 (di mana 1 < 2), C akan mengganti cost "jalan-terpendek" dari C-B menjadi 1. Dan hal ini mempengaruhi Kemudian router A yang tadinya memiliki cost link antara A-C yang sebelumnya bernilai 5, mendapat informasi dari B bahwa ada *path* baru dari B-C dengan nilai 1, ia akan mengganti cost A-C yang sebelumnya 5 dengan 3+1=4, karena 4 < 5. Sehingga kini routing table yang terbentuk di A adalah

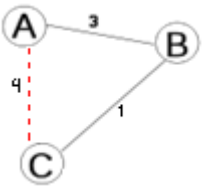
A-B = 3
A-C = 4

sedangkan di B adalah

B-A = 3
B-C = 1

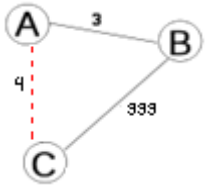
dan di C adalah

C-A = 4
C-B = 1



Gambar 6. Distance Cost yang Baru

Namun hal tersebut akan berdampak buruk apabila penggantian cost tersebut melebihi *pseudo path* yang terbentuk pada node A-C. Misalkan kita mematikan link antara B-C dan mengesetnya menjadi 999.



Gambar 7. Link Antara B-C Diset Menjadi 999

Hal pertama yang dilakukan router B adalah mengubah cost dari B-C menjadi 999. routing table yang terbentuk di A adalah tetap

A-B = 3
A-C = 4

sedangkan di B adalah

B-A = 3
B-C = 999

dan di C pun ikut mengganti bobot path C-B

C-A = 4
C-B = 999.

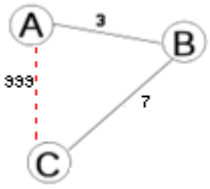
Tetapi ketika router B akan mengirimkan paket update kepada tetangganya, router A dan C akan mengirimkan paket DV mereka terlebih dahulu di mana masih terdapat *pseudo-path* antara A-C dengan bobot 4. B yang mengetahui hal ini akan memilih jalur B-A-C dengan bobot 3+4=7 karena lebih kecil dari 999. B akan mengganti path yang ia miliki menjadi 7.

B-A = 3
B-C = 7

Setelah itu barulah router A berubah menjadi

A-B = 3
A-C = 999

Router C kini telah terputus



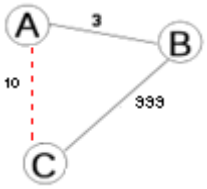
Gambar 8. Perubahan Link Hasil Update

Masalah tersebut belum berhenti karena sebelum B sempat mengganti link cost ke C miliknya menjadi 999, A telah menerima update bahwa ada link cost antara B-C yang bernilai 7, sehingga A akan menemukan jalan bahwa ternyata ada jalan ke router C yang lebih minimum melalui A-B-C sebesar 3+7=10, (10 < 999) sehingga router A akan mengganti cost "jalan-terpendek"nya menjadi

A-B = 3
A-C = 10

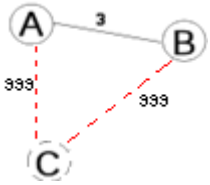
Setelah itu barulah router B berubah menjadi

B-A = 3
B-C = 999



Gambar 9. Perubahan Selanjutnya

Dan hal ini kan berlangsung secara terus menerus hingga kedua sisi mencapai angka maksimum putus yaitu 999. Hal inilah yang dikatakan sebagai *infinite loop.*



Gambar 10. Kondisi Akhir

Untungnya, hal tersebut dapat diatasi dengan menggunakan perintah untuk melakukan Bellman-Ford ulang setiap kali ada perubahan link cost. Kita dapat menggunakan boolean UPDATE yang bernilai FALSE saat awal dan bernilai TRUE bila ada perubahan link cost yang terjadi. Setiap kita melakukan update link cost kita akan mengeset boolean UPDATE menjadi TRUE. Apabila boolean UPDATE bernilai TRUE, seluruh router wajib melakukan Bellman-Ford ulang dari susunan toplogy router yang baru, oleh sebab itu kita perlu mencatat link cost antar tetangga terbaru yang ada pada setiap router. Apabila kita melakukan Bellman-Ford ulang, otomatis masalah mengenai perubahan cost dari B-C di atas tidak akan berpengaruh sebab sejak awal C sudah dianggap telah *down* dan tidak diperhitungkan dalam algoritma ini.

IV. KESIMPULAN

Sebuah varian dari algoritma Bellman-Ford biasanya digunakan dalam protokol routing vektor jarak, misalnya Routing Information Protocol (RIP). Algoritma ini didistribusikan karena melibatkan sejumlah node (router) dalam sebuah sistem Otonom, dan koleksi jaringan IPnya biasanya dimiliki oleh sebuah ISP. Algoritma ini terdiri dari langkah-langkah berikut:

1. Setiap simpul menghitung jarak antara dirinya dan semua node lain dalam AS dan menyimpan informasi ini dalam sebuah tabel.
2. Setiap node mengirimkan tabel untuk semua node tetangganya.
3. Ketika sebuah node menerima tabel jarak dari tetangga-tetangganya, ia menghitung rute terpendek ke semua node lainnya dan melakukan updating tabelnya sendiri untuk mencerminkan perubahan apapun.

Kelemahan utama dari algoritma Bellman-Ford dalam latar protokol routing untuk vektor jarak adalah:

1. Perhitungan skala kurang baik
2. Perubahan dalam topologi jaringan tidak tercermin dengan cepat sejak pembaruan menyebar karena dilakukan node-by-node bukan broadcasting.
3. *Counting to infinity* (jika link atau node mengalami kegagalan dan menjadikan sebuah simpul tidak terjangkau dari beberapa set node lain, mereka menghabiskan waktu dan secara bertahap meningkatkan perkiraan jarak ke sana, padahal sementara itu mungkin ada routing loop).

REFERENSI

[1] http://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorith.

[2] http://en.wikipedia.org/wiki/Distance-vector_routing_protocol

[3] http://equatorialmaths.wordpress.com/2009/09/18/bellman-ford-algorithm-for-shortest-paths/

[4] Leiserson, Charles. E, "Shortest Path 2", "Introduction to Algorithms", 18, 2001.

[5] Munir, Rinaldi. (2006), "Diktat Kuliah IF2153 Matematika Diskrit Edisi Keempat", Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.

[6] ranau.cs.ui.ac.id/sda/archive/2006/ShortestPathAdd-2pp.pdf

[7] www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/bellFordAlgor.htm