BAB III
ALGORITMA GREEDY DAN PROGRAM DINAMIS

3.1 Algoritma Greedy

Algoritma Greedy merupakan metode yang paling populer dalam memecahkan persoalan optimasi. Hanya ada dua macam persoalan optimasi, yaitu maksimasi dan minimasi. Algoritma Greedy adalah algoritma yang memecahkan masalah langkah per langkah. Algoritma Greedy membentuk solusi langkah per langkah. Pada setiap langkah, terdapat banyak pilihan yang perlu dieksplorasi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Pada setiap langkahya merupakan pilihan, untuk membuat langkah optimum lokal (*local optimum*) dengan harapan bahwa langkah sisanya mengarah ke solusi optimum global (*global optimum*). Prinsip Greedy adalah "*take what you can get now*", mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (Munir).

3.1.1 Skema Umum Algoritma Greedy

Persoalan optimasi dalam konteks algoritma greedy disusun oleh elemen-elemen sebagai berikut :

1. Himpunan kandidat, C.
   Himpunan ini berisi elemen-elemen pembentuk solusi. Pada setiap langkah, satu buah kandidat diambil dari himpunannya.
2. Himpunan solusi, S.
   Himpunan ini berisi kandidat-kandidat yang terpilih sebagai solusi persoalan. Dengan kata lain, himpunan solusi adalah himpunan bagian dari himpunan kandidat.
3. Fungsi seleksi (*selection function*)
   Fungsi ini dinyatakan dengan predikat seleksi. Merupakan fungsi yang pada setiap langkah memilih kandidat yang paling memungkinkan

mencapai solusi optimal. Kandidat yang sudah dipilih pada suatu langkah tidak pernah dipertimbangkan lagi pada langkah selanjutnya.

4. Fungsi kelayakan (*feasible*)

   Fungsi ini dinyatakan dengan predikat layak. Fungsi kelayakan ini merupakan fungsi yang memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yakni kandidat tersebut bersama-sama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala (*constraints*) yang ada. Kandidat yang layak dimasukkan ke dalam himpunan solusi, sedangkan yang tidak layak dibuang dan tidak pernah dipertimbangkan lagi.

5. Fungsi obyektif

   Fungsi obyektif ini merupakan sebuah fungsi yang memaksimumkan atau meminimumkan nilai solusi.

Dengan kata lain, algoritma greedy melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan S dioptimasi oleh fungsi obyektif.

Adakalanya, solusi optimum global yang diperoleh dari algoritma greedy yang diharapakan sebagai solusi optimum dari persoalan, belum tentu merupakan solusi optimum (terbaik), tetapi solusi *sub-optimum* atau *pseudo-optimum*. Hal ini dikarenakan algoritma greedy tidak beroperasi secara menyeluruh terhadap semua alternatif solusi yang ada dan terdapat beberapa fungsi seleksi yang berbeda, yaitu jika fungsi seleksi tidak identik dengan fungsi obyektif karena fungsi seleksi biasanya didasarkan pada fungsi obyektif. Sehingga harus dipilih fungsi yang tepat jika menginginkan algoritma menghasilkan solusi optimal atau nilai yang optimum. Jadi, pada sebagian masalah algoritma greedy tidak selalu berhasil memberikan solusi yang benar-benar optimum, tetapi algoritma greedy pasti memberikan solusi yang mendekati (*approximation*) nilai optimum.

Jika jawaban terbaik mutlak tidak diperlukan, maka algoritma greedy sering berguna untuk menghasilkan solusi hampiran (*approximation*), daripada menggunakan algoritma yang lebih rumit untuk menghasilkan solusi yang eksak. Bila algoritma greedy optimum, maka keoptimalannya itu dapat dibuktikan secara matematis (Munir).

3.2 Program Dinamis (Dynamic Programming)

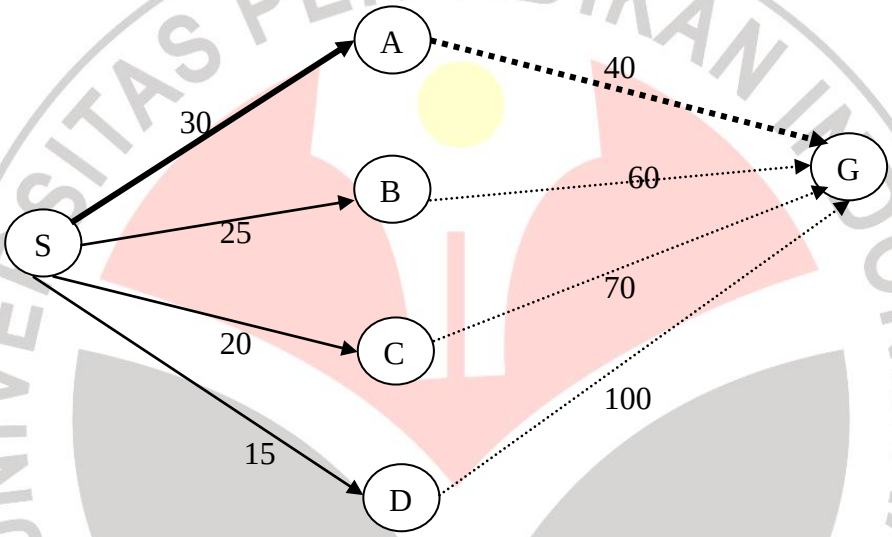
Di dalam ilmu matematika dan ilmu komputer, program dinamis (*dynamic programming*) didefinisikan sebagai suatu metode penyelesaian masalah untuk permasalahan yang memiliki sifat-sifat ***overlapping subproblems*** dan ***optimal substructure***. Istilah *dynamic programming* pertama kali digunakan pada era 1940-an oleh Richard Bellman.

Program Dinamis digunakan untuk menggambarkan proses penyelesaian masalah dimana solusi yang diinginkan harus benar-benar paling optimum atau biasanya disebut sebagai optimum global (*global optimum*). Pada *Dynamic Programming*, istilah "*programming*" sama sekali tidak ada hubungannya dengan pemograman komputer (*computer programming*). Penggunaan kata "*programming*" justru berasal dari istilah "*mathematical programming*", yang merupakan sinonim untuk kata "*optimization*". Dengan demikian, kata "*programming*" disini berarti rencana yang optimal (*optimal plan*) untuk aksi-aksi yang dihasilkan. Sehingga "*programming*" bisa diartikan sebagai pencarian rencana aksi-aksi yang dapat diterima (Suyanto, 2010).

3.2.1 Konsep Dasar

Optimal substructure berarti bahwa solusi optimal untuk submasalah-submasalah (*subproblems*) dapat digunakan untuk menemukan solusi optimal yang utuh untuk masalah yang dihadapi. Sebagai contoh, dalam suatu graf, jalur terpendek (*shortest path*) dari suatu simpul asal (misalnya S) menuju *goal* (misalnya G) dapat ditemukan dengan terlebih dulu menghitung sub jalur terpendek dari semua simpul tetangga S menuju *goal*, dan selanjutnya

menggunakan sub jalur terpendek tersebut untuk mendapatkan jalur terpendek yang utuh. Perhatikan Gambar 3-1 di bawah ini



Gambar 3-1
(Suyanto, 2010)

Gambar di atas merupakan contoh pencarian jalur terpendek (*shortest path*) dalam suatu graf menggunakan *optimal substructure*. Garis yang tebal menunjukkan suatu busur tunggal antara dua simpul. Sedangkan garis putus-putus menunjukkan *shortest path* antara dua simpul, dimana simpul-simpul yang berada diantara kedua simpul tersebut tidak diperlihatkan. Garis tebal menyatakan *shortest path* yang utuh dari simpul asal S ke simpul tujuan G.

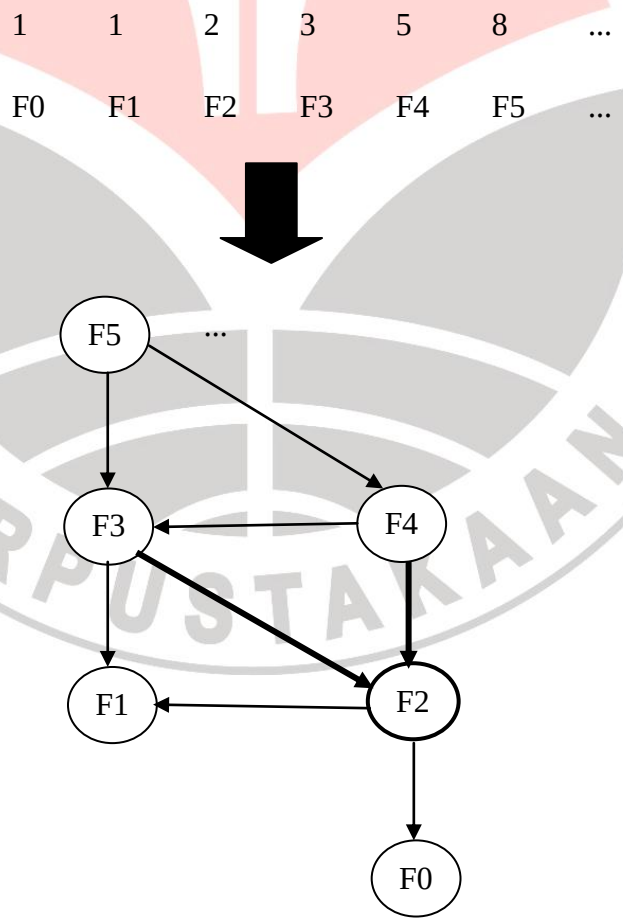
Secara umum, untuk menyelesaikan suatu masalah dengan *optimal substructure* dapat digunakan tiga langkah berikut :

1. Pecah masalah menjadi submasalah-submasalah yang lebih kecil

2. Selesaikan submasalah-submasalah secara optimal menggunakan tiga langkah ini secara rekursif
3. Gunakan solusi-solusi optimal untuk submasalah-submasalah tersebut untuk membangun solusi optimal yang utuh untuk masalah yang dihadapi.

Setiap submasalah diselesaikan dengan cara membaginya kedalam submasalah-submasalah yang lebih kecil, dan seterusnya, sampai mencapai suatu kasus sederhana yang dapat diselesaikan dalam waktu singkat dan konstan.

Suatu masalah dikatakan memiliki ***overlapping subproblems*** jika *subproblems* yang sama digunakan untuk menyelesaikan banyak masalah lain yang lebih besar. Sebagai contoh, perhatikan gambar deret fibonacci pada Gambar 3-2 dibawah ini.



Gambar 3-2
(Suyanto, 2010)

Gambar di atas merupakan graf *subproblem* untuk deret fibonacci yang menunjukkan adanya *overlapping subproblems*. Pada deret ini, F0 = F1 = 1. Selanjutnya, F2 = F1 + F0, F3 = F2 + F1, F4 = FF3 + F2, dan seterusnya. Secara umum, deret ini dapat dirumuskan sebagai F0 = F1 = 1 dan F_n = F(n - 1) + F(n - 2) untuk n lebih dari atau sama dengan 2.

Pada deret di atas, F3 = F1 + F2 dan F4 = F2 + F3. Penghitungan F3 dan F4 memerlukan penghitungan F2. Karena F3 dan F4 diperlukan untuk menghitung F5, metode naif untuk menghitung F5 mungkin memerlukan penghitungan F2 sebanyak 2 kali atau lebih. Hal ini dilakukan setiap kali muncul *subproblems*. Dengan demikian, metode naif akan menghabiskan banyak waktu penghitungan, secara berulang-ulang, solusi-solusi optimal untuk *subproblems* yang sebenarnya sudah diselesaikan.

Untuk mencegah hal ini, solusi-solusi untuk submasalah-submasalah yang sudah diselesaiakan dapat disimpan. Selanjutnya, jika perlu untuk menyelesaikan masalah yang sama, dapat mengambil dan menggunakan solusi tersebut. Pendekatan ini disebut ***memoization*** (bukan *memorization*, meskipun istilah ini juga sesuai). Jika yakin bahwa tidak akan membutuhkan solusi yang disimpan tersebut, solusi tersebut dapat dihapus untuk menghemat *memory*. Bahkan dalam beberapa kasus, dapat dihitung terlebih dahulu solusi-solusi mana yang akan dibutuhkan untuk submasalah-submasalah yang diketahui yang nantinya akan diperlukan.

Program dinamis biasanya menggunakan salah satu dari dua pendekatan berikut ini :

- ***Top-down***

  Masalah dipecah menjadi beberapa submasalah, kemudian submasalah-submasalah tersebut diselesaikan dan solusi-solusinya diingat (dicatat)

agar dapat digunakan kembali. Dalam hal ini, *recursion* dan *memoization* dikombinasikan secara bersamaan.

- ***Bottom-up***

  Semua submasalah-submasalah, yang mungkin diperlukan, diselesaikan terlebih dahulu dan kemudian digunakan untuk membangun solusi-solusi untuk masalah-masalah yang lebih besar. Pendekatan ini sedikit lebih baik dalam hal ruang tumpukan (*stack space*) dan jumlah *function calls*, tetapi kadang-kadang tidak intuitif untuk menggambarkan semua submasalah yang diperlukan untuk menyelesaikan masalah yang diberikan (Suyanto, 2010).

3.2.2 Skema Umun Program Dinamis

Program Dinamis (*Dynamic Programming*) merupakan suatu metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah (*step*) atau tahapan (*stage*) sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Karakteristik penyelesaian persoalan dengan program dinamis:

1. Terdapat sejumlah berhingga pilihan yang mungkin
2. Solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya
3. Menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap (Munir).

3.2.2.1 Prinsip Optimalitas

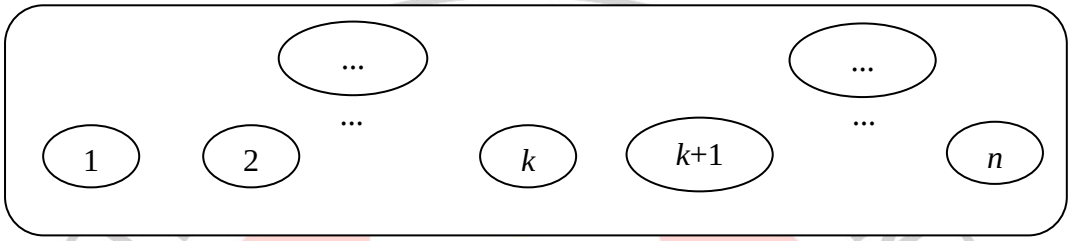
Pada program dinamis, rangkaian keputusan yang optimal dibuat dengan menggunakan **Prinsip Optimalitas**, dengan prinsip optimalitas ini dijamin bahwa pengambilan keputusan pada suatu tahap adalah keputusan yang benar untuk tahap-tahap selanjutnya. Dikatakan prinsip optimalitas, jika solusi total optimal, maka bagian solusi sampai tahap ke-*k* juga optimal. Prinsip optimalitas berarti

bahwa jika dari tahap *k* ke tahap *k*+1, dapat menggunakan hasil optimal dari tahap *k* tanpa harus kembali ke tahap awal.

Ongkos pada tahap *k*+1 = (ongkos yang dihasilkan pada tahap *k*)

+

(ongkos dari tahap *k* ke tahap *k*+1)



Gambar 3-3
(Munir)

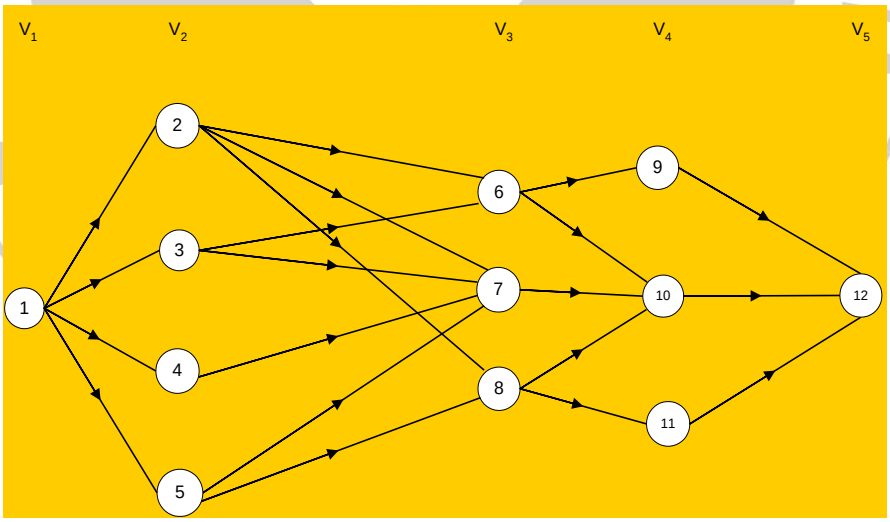
Dengan prinsip optimalitas ini dijamin bahwa pengambilan keputusan pada suatu tahap adalah keputusan yang benar untuk tahap-tahap selanjutnya. Perbedaan antara algoritma greedy dan program dinamis adalah pada metode greedy hanya satu rangkaian keputusan yang pernah dihasilkan, sedangkan pada metode program dinamis lebih dari satu rangkaian keputusan yang memenuhi prinsip optimalitas yang akan dihasilkan (Munir).

3.2.2.2 Karakteristik Persoalan Program Dinamis

Ada beberapa karakteristik persoalan yang dalam penyelesaian dengan menggunakan program dinamis yaitu:

1. Persoalan dapat dibagi menjadi beberapa tahap (*stage*), yang pada setiap tahap hanya diambil satu keputusan.
2. Masing-masing tahap terdiri dari sejumlah status (*state*) yang berhubungan dengan tahap tersebut. Secara umum, status merupakan bermacam kemungkinan masukan yang ada pada tahap tersebut.

3. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya.
4. Ongkos (*cost*) pada suatu tahap meningkat secara teratur (*steadily*) dengan bertambahnya jumlah tahapan.
5. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos pada tahap tersebut.
6. Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusan yang dilakukan pada tahap sebelumnya.
7. Adanya hubungan rekursif yang menidentifikasikan keputusan terbaik untuk setiap status pada tahap *k* memberikan keputusan terbaik untuk setiap status pada tahap *k*+1.
8. Prinsip optimalitas berlaku pada persoalan tersebut.



Gambar 3-4 Graf Multitahap (*multistage graph*)
(Munir)

Pada gambar di atas, tiap simpul didalam graf tersebut menyatakan status, sedangkan V_1, V_2, ... menyatakan tahap (Munir).

3.2.2.3 Dua Pendekatan Program Dinamis

Dua pendekatan yang digunakan dalam program dinamis:

1. Program dinamis maju (*forward* atau *up-down*)
2. Program dinamis mundur (*backward* atau *bottom-up*)

Misalkan x_1, x_2, ..., x_n menyatakan peubah (*variable*) keputusan yang harus dibuat masing-masing untuk tahap 1, 2, ..., *n*. Maka,

1. Program dinamis maju. Program dinamis bergerak mulai dari tahap 1, terus maju ke tahap 2, 3, dan seterusnya sampai tahap *n*. Runtunan peubah keputusan adalah x_1, x_2, ..., x_n.
2. Program dinamis mundur. Program dinamis bergerak mulai dari tahap *n*, terus mundur ke tahap *n*-1, *n*-2, dan seterusnya sampai tahap 1. Runtunan peubah keputusan adalah x_n, x_{n-1}, ..., x_1.

- Prinsip optimalitas pada program dinamis maju:
  ongkos pada tahap *k*+1 = (ongkos yang dihasilkan pada tahap k)
  +
  (ongkos dari tahap k ke tahap *k*+1),
  dengan *k* = 1, 2, ..., n-1.
- Prinsip optimalitas pada program dinamis mundur:
  ongkos pada tahap *k* = (ongkos yang dihasilkan pada tahap k+1)
  +
  (ongkos dari tahap k+1 ke tahap k),
  dengan k = n, n-1, ..., 1 (Munir).

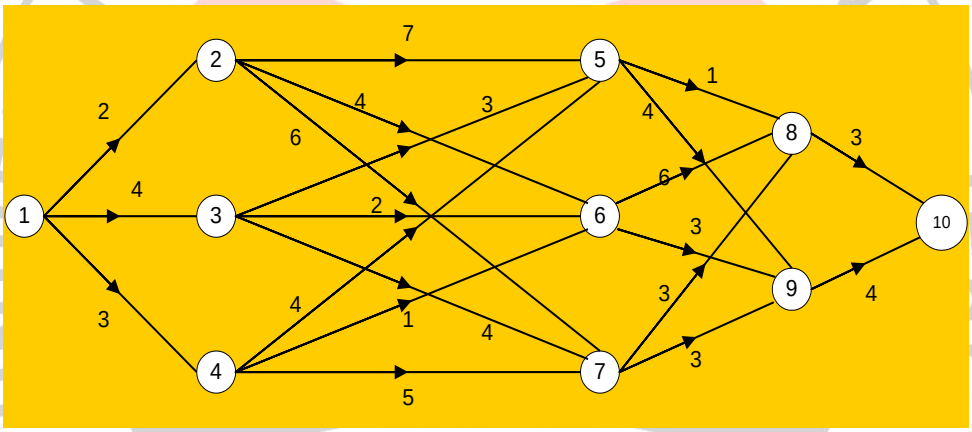
3.2.2.4 Langkah-langkah Pengembangan Algoritma Program Dinamis

Adapun bagaimana langkah-langkah dalam pengembangan algoritma program dinamis:

1. Karakteristikkan struktur solusi optimal.
2. Definisikan secara rekursif nilai solusi optimal.
3. Hitung nilai solusi optimal secara maju atau mundur.
4. Konstruksi solusi optimal.

Contoh kasus

Tentukan lintasan terpendek dari simpul 1 ke simpul 10 :



Gambar 3-5 Lintasan Terpendek (*shortest path*)
(Munir)

Penyelesaian dengan menggunakan program dinamis mundur

- Misalkan x_1, x_2, ..., x_4 adalah simpul-simpul yang dikunjungi pada tahap k (k = 1, 2, 3, 4).
- Maka rute yang dilalui adalah

  1→ $x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$, yang dalam hal ini x_4 = 10.

Pada persoalan ini,

- Tahap (k) adalah proses memilih simpul tujuan berikutnya (ada 4 tahap).

- Status (s) yang berhubungan dengan masing-masing tahap adalah simpul-simpul didalam graf.

Relasi rekurens berikut menyatakan lintasan terpendek dari status s ke x_4 pada tahap k:

$f_4(s) = C_{sx_4}$ (basis)

$f_k(s) = min_{x_k}\{C_{sx_k} + f_{k+1}(x_k)\}$, (rekurens)

k = 1, 2, 3

Keterangan:

x_k : peubah keputusan pada tahap k (k = 1, 2, 3).

C_{sx_k} : bobot (*cost*) sisi dari s ke x_k

$f_k(s, x_k)$: total bobot lintasan dari s ke x_k

$f_k(s)$: nilai minimum dari $f_k(s, x_k)$

Tujuan program dinamis mundur: mendapatkan $f_k(s)$, dengan mencari $f_k(s)$, $f_k(s)$, $f_k(s)$ terlebih dahulu.

Tahap 4:

$f_4(s) = C_{sx_4}$

s	Solusi Optimum	
	$f_4(s)$	x_4^*
8	3	10
9	4	10

Catatan: x_4^* adalah nilai x_k yang meminimumkan $f_k(s, x_k)$.

Tahap 3:

$f_3(s) = min_{x_3}\{C_{sx_3} + f_4(x_3)\}$

s	x_3			
	$f_3(s, x_3) = C_{s,x_3} + f_4(x_3)$		Solusi Optimum	
	8	9	$f_3(s)$	x_3^*
5	4	8	4	8

6	9	7	7	9
7	6	7	6	7

Tahap 2:

$f_2(s) = min_{x_2}\{C_{sx_2} + f_3(x_2)\}$

s	x_2			
	$f_2(s, x_2) = C_{s,x_2} + f_3(x_2)$		Solusi Optimum	
	5	6	$f_2(s)$	x_2^*
2	11	11	12	5 atau 6
3	7	9	10	5
4	8	8	11	5 atau 6

Tahap 1:

$f_1(s) = min_{x_1}\{C_{sx_1} + f_2(x_1)\}$

s	x_1				
	$f_1(s, x_1) = C_{s,x_1} + f_1(x_1)$			Solusi Optimum	
	2	3	4	$f_1(s)$	x_1^*
1	13	11	11	11	3 atau 4

Solusi optimum dapat dibaca pada tabel dibawah ini:

	x_1	x_2	x_3	x_4	Panjang Lintasan Terpendek
1	3	5	8	10	11
	4 →	5	8	10	11

		6	9	10	11

Jadi ada tiga lintasan terpendek dari 1 ke 10, yaitu:
1→ 3 → 5 → 8 → 10

1 → 4 → 5 → 8 → 10

1 → 4 → 6 → 9 → 10

Panjang ketiga lintasan tersebut sama, yaitu 11 (Munir).