#10 PEMROGRAMAN DINAMIS

Materi Pertemuan #10 (Online #8)

Kemampuan Akhir Yang Diharapkan

Mampu membandingkan antara kondisi nyata dengan penerapan teori yang telah dipelajari dan menghitung serta menganalisis permasalahan dengan pendekatan metode keilmuan teknik industri terkait dengan pemrograman dinamis.

Indikator Penilaian

Ketepatan dalam membandingkan antara kondisi nyata dengan penerapan teori yang telah dipelajari dan menghitung serta menganalisis permasalahan dengan pendekatan metode keilmuan teknik industri terkait dengan pemrograman dinamis.

10.1. Pendahuluan

Istilah Pemrograman Dinamis (*Dynamic Programming*) awalnya digunakan pada 1940-an oleh Richard Bellman untuk menggambarkan proses pemecahan masalah tahap demi tahap untuk menemukan satu keputusan terbaik. Pada tahun 1953, istilah tersebut disempurnakan untuk makna modern, dengan mengacu secara khusus untuk bersarang pada masalah keputusan kecil dalam keputusan yang lebih besar, dan lapangan, sehingga kemudian diakui oleh IEEE sebagai topik analisis sistem dan rekayasa. Kontribusi Bellman akan diingat dalam nama persamaan Bellman, hasil pemrograman dinamis yang menyatakan kembali masalah optimasi dalam bentuk rekursif. Kata “Dinamis” dipilih oleh Bellman untuk menangkap aspek waktu bervariasi dari masalah, dan juga karena terdengar mengesankan. Kata “Pemrograman” mengacu pada penggunaan metode untuk menemukan program yang optimal, untuk pelatihan atau logistik dalam jadwal kemiliteran. Penggunaannya sama seperti dalam pemrograman linear dan pemrograman matematika, sebuah sinonim untuk optimasi matematika.

Pemrograman Dinamis atau *Dynamic Programming* (biasa disingkat DP) adalah suatu teknik algoritma untuk memecahkan masalah dimana solusi optimal dari masalah tersebut dapat dipandang sebagai suatu deret keputusan. Persoalan partisi merupakan persoalan yang sering diterapkan dalam kehidupan sehari-hari seperti misalnya pada persoalan pembagian pekerjaan. Persoalan partisi sendiri memiliki banyak model dan karateristik. Persoalan yang dibahas dalam makalah ini adalah persoalan membagi pekerjaan untuk dikerjakan oleh N pekerja secara efisien sedemikian sehingga setiap pekerja mendapat pekerjaan yang relatif sama (perbedaan bobot yang diterima pekerja dibuat seminimum mungkin). Penyelesaian persoalan ini salah satunya adalah dengan menggunakan algoritma pemrograman dinamis (*dynamic programming*).

Algoritma pemrograman dinamis adalah penyelesaian persoalan dengan cara menguraikan solusi menjadi sekumpulan langkah dan memandang solusi dari persoalan tersebut sebagai serangkaian keputusan yang saling berkaitan.

Suatu pekerjaan yang berskala besar seringkali harus dikerjakan oleh banyak pekerja. Pekerjaan berskala besar itu biasanya dapat dibagi-bagi menjadi beberapa bagian pekerjaan yang masing-masing bagian memiliki bobot tertentu. Pembagian

bobot pekerjaan selayaknya relatif sama (perbedaaan bobot yang diterima setiap pekerja dibuat seminimum mungkin) untuk setiap pekerja.

Cara pembagian pekerjaan dengan memperhitungkan bobot tersebut dapat mengefisiensikan waktu pengerjaan dan juga mendapatkan hasil yang optimal.

Sebagai contoh, terdapat sebuah pekerjaan untuk melakukan pencarian informasi yang ada pada beberapa buku yang memiliki jumlah halaman berbeda kepada tiga pekerja. Buku-buku ini dibagi ke setiap pekerja dengan jumlah halaman buku relatif sama (perbedaan halaman yang diterima setiap pekerja dibuat seminimum mungkin). Untuk membagi jumlah halaman tersebut dengan perbedaan halaman seminimal mungkin diperlukan sebuah metode pembagian partisi. Proses pembagian partisi ini dapat menerapkan berbagai algoritma salah satunya adalah menggunakan pemrograman dinamis.

Dalam matematika dan ilmu komputer, pemrograman dinamis adalah metode untuk memecahkan masalah yang kompleks dengan melanggar mereka ke dalam *sub-problems* yang lebih sederhana. Hal ini berlaku untuk masalah menunjukkan sifat *overlapping sub-problem* yang hanya sedikit lebih kecil dan optimal sub-struktur. Ketika diterapkan, metode ini membutuhkan waktu jauh lebih sedikit daripada metode naif.

Ide kunci di balik pemrograman dinamis adalah cukup sederhana. Secara umum, untuk memecahkan persoalan yang diberikan, maka perlu untuk memecahkan berbagai masalah (sub-masalah), kemudian menggabungkan solusi dari sub-masalah untuk mencapai solusi secara keseluruhan. Seringkali, banyak dari sub-masalah ini benar-benar sama. Pendekatan pemrograman dinamis berusaha untuk memecahkan setiap sub-masalah hanya sekali, sehingga mengurangi jumlah perhitungan. Hal ini sangat berguna ketika jumlah sub-masalah mengulangi secara eksponensial besar.

Untuk menyelesaikan suatu masalah tentunya membutuhkan suatu cara atau solusi agar masalah tersebut terselesaikan dengan baik. Setiap masalah apapun tentunya membutuhkan yang namanya solusi atau cara. Dalam sebuah penelitianpun diperlukan solusi untuk membantu memecahkan suatu masalah. Sama halnya dengan laporan ini yang membahas tentang "*Dynamic Programming*".

Dynamic Programming mirip seperti metode *divide-and-conquer* yang menyelesaikan suatu problem dengan mengkobinasikan solusi menjadi sub-problem. *Divide-and-conquer* membagi problem menjadi sub-problem yang independen. Kemudian menyelesaikan sub-problem secara rekursif dan mengkombinasikan solusi tersebut untuk menyelesaikan problem utama. Sedangkan *dynamic programming* cocok digunakan ketika sub-problem tidak independen, jadi ketika sub-problem terbagi menjadi sub-sub-problem.

Pemrograman dinamis adalah suatu teknik matematis yang biasanya digunakan untuk membuat suatu keputusan dari serangkaian keputusan yang saling berkaitan. Dalam hal ini pemrograman dinamis menyediakan prosedur sistematis untuk menentukan kombinasi keputusan yang optimal. Tujuan utama model ini ialah untuk mempermudah penyelesaian persoalan optimasi yang mempunyai karakteristik tertentu. Tidak seperti pemrograman linier, tidak ada bentuk matematis standar untuk perumusan pemrograman dinamis. Akan tetapi, pemrograman dinamis adalah pendekatan umum untuk pemecahan masalah dan persamaan tertentu yang digunakan di dalamnya harus dibentuk sesuai dengan situasi masalah yang dihadapi.

10.2. Pengertian *Dynamic Programming*

Dynamic programming problems adalah masalah multi tahap (*multistage*) dimana keputusan dibuat secara berurutan (*in sequence*).

Pemrograman dinamis (*dynamic programming*) adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah (*step*) atau tahapan (*stage*) sedemikian rupa sehingga solusi dari permasalahan ini dapat dipandang dari serangkaian keputusan-keputusan kecil yang saling berkaitan satu dengan yang lain. Penyelesaian persoalan dengan pemrograman dinamis ini akan menghasilkan sejumlah berhingga pilihan yang mungkin dipilih, lalu solusi pada setiap tahap-tahap yang dibangun dari solusi pada tahap sebelumnya, dan dengan metode ini kita menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

10.3. Konsep Dasar Dan Karakteristik *Dynamic Programming*

Untuk konsep dasar yang terdapat dalam pemrograman dinamis atau *dynamic programming* (DP), antara lain:

1) Dekomposisi
   Persoalan DP dapat dipecah-pecah menjadi sub-persoalan atau tahapan (*stage*) yang lebih kecil dan berurutan. Setiap tahap disebut juga sebagai titik keputusan. Setiap keputusan yang dibuat pada suatu tahap akan mempengaruhi keputusan-keputusan pada tahap berikutnya.

2) Status
   Status adalah kondisi awal ($\boldsymbol{S_n}$) dan kondisi akhir ($\boldsymbol{S_{n-1}}$) pada setiap tahap, dimana pada tahap tersebut keputusan dibuat ($\boldsymbol{D_n}$). Status akhir pada sebuah tahap tergantung kepada status awal dan keputusan yang dibuat pada tahap yang bersangkutan. Status akhir pada suatu tahap merupakan input bagi tahap berikutnya.

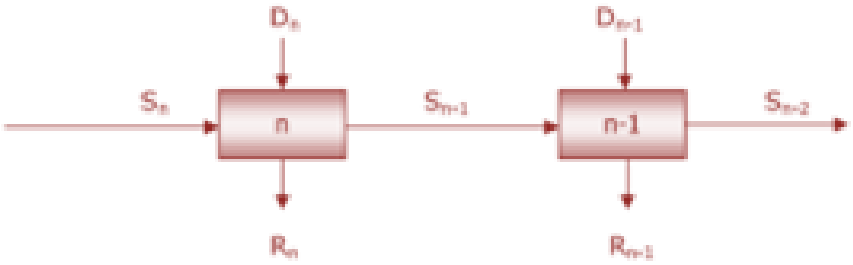
3) Variabel Keputusan dan Hasil
   Keputusan yang dibuat pada setiap tahap ($\boldsymbol{D_n}$) merupakan keputusan yang berorientasi kepada *return* yang diakibatkannya ($\boldsymbol{R_n|D_n}$), yaitu tingkat maksimal atau minimal.

4) Fungsi Transisi
   Fungsi transisi menjelaskan secara pasti bagaimana tahap-tahap saling berhubungan. Fungsi ini berbentuk fungsi hubungan antar status pada setiap tahap yang berurutan. Fungsi transisi secara umum berbentuk:

   $$\boldsymbol{S_{n-1} = S_n - D_n}$$

   Di mana:
   - $\boldsymbol{S_{n-1}}$ = status pada tahap $\boldsymbol{n-1}$, atau status akhir pada tahap-*n*.
   - $\boldsymbol{S_n}$ = status awal pada tahap-*n*.

   Komponen pada setiap tahap dapat digambarkan seperti pada Gambar 10.1.



Gambar 10.1. Komponen Setiap Tahap Pada Pemrograman Dinamis

5) Optimasi Tahap
   Optimasi tahap dalam DP adalah menentukan keputusan optimal pada setiap tahap dari berbagai kemungkinan nilai status inputnya.
   Fungsi umum dari keputusan optimal adalah:

   – $\boldsymbol{f_n(S_n, D_n)}$ = return pada tahap-*n* dari nilai status input S_n, dan keputusan D_n.

   – $\boldsymbol{f_n^*(S_n)}$ = return optimal pada tahap-*n* dari nilai input status S_n.

6) Fungsi Rekursif
   Fungsi rekursif biasanya digunakan pada berbagai program komputer, di mana nilai sebuah variabel pada fungsi itu merupakan nilai kumulatif dari nilai variabel tersebut pada tahap sebelumnya. Pada DP, fungsi umum dituliskan sebagai:

   $$\boldsymbol{f_n(S_n, D_n) = R_n + f_{n-1}^*(S_{n-1}, D_{n-1})}$$

Untuk karakteristik persoalan yang dimiliki oleh pemrograman dinamis, antara lain:

1) Persoalan dapat dibagi menjadi beberapa tahap (*stage*), yang pada setiap tahap hanya dapat 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. Jumlahnya bisa berhingga atau tak berhingga.
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 mengidentifikasikan 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.

Prosedur optimasi diawali dari tahap akhir menuju tahap awal (*backward*), dengan karakteristik pemrograman dinamis adalah:

1) Persoalan dapat dipisahkan menjadi beberapa tahap (*stages*), di mana setiap tahap membutuhkan keputusan kebijakan yang standar dan saling berhubungan, seperti pada Gambar 10.2.

Gambar 10.2. Tahap (Stage) Pada Pemrograman Dinamis

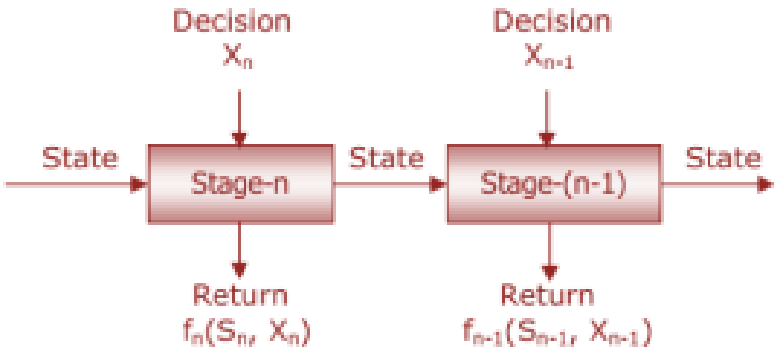
2) Setiap tahap memiliki sejumlah status (*state*). Secara umum, sekumpulan status ini merupakan berbagai kemungkinan kondisi yang timbul dari sistim persoalannya. Status ini memberikan informasi yang dibutuhkan setiap keputusan dan dampaknya pada tahap berikutnya. Jumlah status pada setiap tahap bisa definit atau infinit.

3) Setiap keputusan kebijakan yang dibuat pada suatu tahap, status pada tahap tersebut ditransformasi ke dalam status yang berkaitan pada tahap berikutnya. Hubungan antar status pada tahap yang berurutan bias bersifat deterministik atau probabilistik.

   Pada sebuah persoalan dengan n-tahap, ada dua input, yaitu:
   a) *State* pada tahap-n ($\boldsymbol{S_n}$)
   b) *Decision variable* ($\boldsymbol{X_n}$)

   Sedang outputnya adalah:
   a) *Return* atau akibat dari setiap $\boldsymbol{X_n}$ yang dipilih, $\boldsymbol{f_n(S_n, X_n)}$
   b) Status baru yang menjadi input pada tahap berikutnya $(\boldsymbol{S_{n-1}})$.

   Hubungan antara $\boldsymbol{X_n}$ dan $\boldsymbol{f_n(S_n, X_n)}$ ditentukan oleh *return function*. Sedangkan hubungan antar status pada tahap tertentu ditentukan oleh *transition function* sepert pada Gambar 10.3.



Gambar 10.3. Hubungan antara $\boldsymbol{X_n}$ dan $\boldsymbol{f_n(S_n, X_n)}$

4) Solusi pada programa dinamis berprinsip kepada optimalitas yang dikembangkan oleh Bellman: "*An optimal policy must have the property that, regardless of the decision to enter a particular state, the remaining decisions must consitute an optimal policy for leaving that state*".

5) Keputusan pada tahap berikutnya bersifat independen terhadap keputusan sebelumnya. Untuk menyelesaikan persoalan pemrograman dinamis, dimulai dari solusi awal pada suatu tahap, dan secara berurutan menuju tahap berikutnya dengan proses yang terbalik (*backward induction process*).
6) Solusi optimal yang dihasilkan pada setiap tahap berprinsip kepada hubungan dalam bentuk fungsi rekursif (*recursion relationship*). Secara umum bentuk fungsi rekursif adalah:

$$f_n^*(S_n) = max/min\{f_n(S_n, X_n)\}$$

Di mana: $f_n^*(S_n)$ adalah hasil optimal dari keputusan pada tahap-n.

10.4. Kelebihan Dan Kekurangan Pada *Dynamic Programming*

Adapun kelebihan dari Pemrograman Dinamis (*Dynamic Programming*) antara lain:

1) Mengoptimalkan penyelesaian suatu masalah tertentu yang diuraikan menjadi sub-submasalah yang lebih kecil yang terkait satu sama lain dengan tetap memperhatikan kondisi dan batasan permasalahan tersebut.
2) Proses pemecahan suatu masalah yang kompleks menjadi sub-sub masalah yang lebih kecil membuat sumber permasalahan dalam rangkaian proses masalah tersebut menjadi lebih jelas untuk diketahui.
3) Pendekatan *dynamic programming* dapat diaplikasikan untuk berbagai macam masalah pemrograman matematik, karena *dynamic programming* cenderung lebih fleksibel daripada teknik optimasi lain.
4) Prosedur perhitungan *dynamic programming* juga memperkenankan bentuk analisissensitivitas terdapat pada setiap variabel status (*state*) maupun pada variabel yang ada di masing-masing tahap keputusan (*stage*).
5) *Dynamic programming* dapat menyesuaikan sistematika perhitungannya menurut ukuran masalah yang tidak selalu tetap dengan tetap melakukan perhitungan satu persatu secara lengkap dan menyeluruh.

Sedangkan kelemahan dari Pemrograman Dinamis (*Dynamic Programming*) adalah penggunaan *dynamic programming* jika tidak dilakukan secara tepat, akan mengakibatkan ketidakefisienan biaya maupun waktu. Karena dalam menggunakan *dynamic programming* diperlukan keahlian, pengetahuan, dan seni untuk merumuskansuatu masalah yang kompleks, terutama yang berkaitan dengan penetapan fungsi transformasi dari permasalahan tersebut.

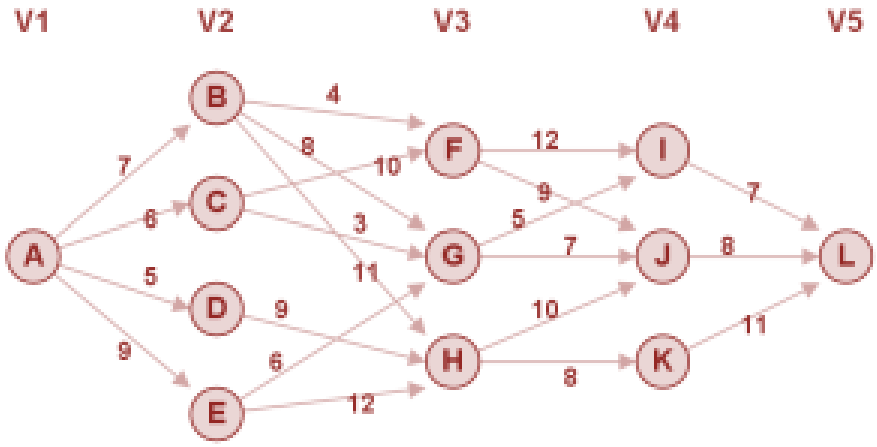
10.5. *Multi Stage Graph*

Multistage Graph adalah grafik dengan sifat-sifat khusus, antara lain:

1) Grafik berarah (*Directed Graph*)
2) Setiap *edge*-nya memiliki *weight* (bobot)
3) Hanya terdapat 1 *source* (disebut ***s***) dan 1 *sink* (disebut ***t***)
4) Lintasan dari *source* ke *sink* terdiri atas beberapa *stage* V_1 sampai V_k
5) Semua *edge* menghubungkan *node* di V_i ke sebuah node di V_{i+1} dimana $1 \leq i \leq k$
6) Terdapat *stage* sebanyak k, dimana $k \geq 2$
7) Setiap *path* dari ***s*** ke ***t*** merupakan konsekuensi dari pilihan sebanyak $k-2$

Multistage Graph merupakan bentuk permodelan yang dapat digunakan untuk menyelesaikan berbagai permasalahan dunia nyata. Contoh: pemilihan proyek untuk mendapatkan keuntungan maksimal; serta pemilihan langkah-langkah yang harus dipilih dalam menyelesaikan sebuah tugas.

Tahapan dalam penggunaan *Multistage Graph Problem*, antara lain:
1) *Problem* mencari lintasan terpendek dari *source* ke *sink* pada sebuah *Multistage Graph*.
2) *Problem* ini merupakan salah satu contoh penerapan yang bagus dari *Dynamic Programming*. Gambar 10.4 merupakan contoh dari *Multistage Graph*.



Gambar 10.4. Contoh *Multistage Graph*.

10.6. *Dynamic Programming* Pada *Multistage Graph Problem*

Teknik penyelesaian *Multistage Graph Problem* dengan *Dynamic Programming* berdasar pada sebuah prinsip bahwa jalur terpendek dari satu *node* (awal atau akhir) ke *node* lain di *stage* tertentu merupakan jalur terpendek dari *stage* sebelumnya ditambah panjang salah satu *edge* penghubung *stage*.

Terdapat 2 (dua) pendekatan, yaitu:

1) Metode *Forward*
Menghitung jarak ke depan (menuju *sink*)

2) Metode *Backward*
Menghitung jarak ke belakang (dari *source*)

Beberapa hal penting yang terdapat pada metode *Forward*, antara lain:
1) Prinsip: analisis dilakukan dengan menghitung *path* (jalur) dari suatu *node* ke *sink*
2) Rumus:

$$\boldsymbol{cost}\,(\boldsymbol{i}\,,\boldsymbol{j}) = \boldsymbol{min}\{\boldsymbol{c}(\boldsymbol{j}\,,\boldsymbol{l}) + \boldsymbol{cost}(\boldsymbol{i}+\boldsymbol{1}\,,\boldsymbol{l})\}$$

3) Perhitungan dimulai dari *node–node* di stage ***k–2***
4) $\boldsymbol{cost}\,(\boldsymbol{i}\,,\boldsymbol{j})$ artinya panjang lintasan dari *node* ***j*** di stage ***i*** menuju *sink* ($\boldsymbol{t}$)
5) $\boldsymbol{c}(\boldsymbol{j}\,,\boldsymbol{l})$ artinya panjang lintasan dari *node* ***j*** ke *node* ***l***

Sedangkan beberapa hal penting yang terdapat pada metode *Backward*, antara lain:

1) Prinsip: analisis dilakukan dengan menghitung *path* (jalur) dari *source* ke suatu *node*
2) Rumus:

$$\boldsymbol{bcost}\,(\boldsymbol{i}\,,\boldsymbol{j}) = \boldsymbol{min}\{\boldsymbol{cost}(\boldsymbol{i}-\mathbf{1}\,,\boldsymbol{l}) + \boldsymbol{c}(\boldsymbol{l}\,,\boldsymbol{j})\}$$

3) Perhitungan dimulai dari *node–node* di *stage* 3
4) $\boldsymbol{bcost}\,(\boldsymbol{i}\,,\boldsymbol{j})$ artinya panjang lintasan *backward* dari *source* (**s**) menuju *node* $\boldsymbol{j}$ di *stage* $\boldsymbol{i}$
5) $\boldsymbol{c}(\boldsymbol{l}\,,\boldsymbol{j})$ artinya panjang lintasan dari *node* $\boldsymbol{l}$ ke *node* $\boldsymbol{j}$

10.7. Contoh Permasalahan

Contoh 10.1: The Roadrunner Transmission Company

The Roadrunner Transmission Company adalah perusahaan yang beroperasi dalam bidang sistim penyaluran BBM dalam mobil. Ada empat bagian berurutan dalam pabrik, yaitu: (1)Towing, (2)Inspection and Diagnostic, (3)Disassembling and Repair, dan (4)Reassembling and Testing. Biaya per unit mobil yang diperbaiki sistim penyaluran aliran BBM dari Towing ke Inspection and Diagnostic Station seperti pada Tabel 10.1.

Tabel 10.1. Biaya/Unit dari Towing ke Inspection and Diagnostic Station

Dari Towing	Ke Inspection and Diagnostic Station			
	1	**2**	**3**	**4**
	35	40	30	45

Sedangkan biaya per unit mobil yang diperbaiki sistim penyaluran aliran BBM dari Inspection and Diagnostic Station ke Disassembling and Repair Station seperti pada Tabel 10.2.

Tabel 10.2. Biaya/Unit dari Inspection and Diagnostic ke Disassembling and Repair Station

Dari Inspection and Diagnostic Station	Ke Disassembling and Repair Station			
	1	**2**	**3**	**4**
1	105	100	85	90
2	90	85	100	95
3	100	90	95	105
4	110	105	120	110

Untuk biaya per unit mobil yang diperbaiki sistim penyaluran aliran BBM dari Disassembling and Repair Station ke Reassembling and Testing Station seperti pada Tabel 10.3.

Tabel 10.3. Biaya/Unit dari Disassembling and Repair ke Reassembling and Testing Station

Dari Disassembling and Repair Station	Ke Reassembling and Testing Station			
	1	**2**	**3**	**4**
1	70	75	85	80
2	85	90	80	95
3	90	70	85	80
4	80	85	90	75

Perbedaan biaya terjadi karena persoalan teknologi mesin pada masing masing station dan jarak antar mesin-mesin. Untuk optimalisasi biaya dapat dilakukan pendekatan pemrograman dinamis. Dengan *backward induction process*, skema persoalan dapat digambarkan seperti pada Gambar 10.5.

Gambar 10.5. Skema Persoalan Contoh 10.1

Solusi Contoh 10.1

Dengan pendekatan pemrograman dinamis metode *backward*, maka harus di mulai pada stage-1 dan hasilnya seperti pada Tabel 10.4.

Tabel 10.4. Hasil Perhitungan Stage-1

X_1 / S_1	$f_1(S_1, X_1) = c_{s_1, x_1}$				$f_1^*(S_1)$	x_1^*
	1	**2**	**3**	**4**		
1	70	75	85	80	**70**	**1**
2	85	90	80	95	**80**	**3**
3	90	70	85	80	**70**	**2**
4	80	85	90	75	**75**	**4**

Kemudian dilanjutkan pada stage-2 dan hasilnya seperti pada Tabel 10.5.

Tabel 10.5. Hasil Perhitungan Stage-2

X_2 / S_2	$f_2(S_2, X_2) = c_{s_2,x_2} + f_1^*(S_1)$				$f_2^*(S_2)$	x_2^*
	1	**2**	**3**	**4**		
1	105 + 70 = 175	100 + 80 = 180	85 + 70 = 155	90 + 75 =165	**155**	**3**
2	90 + 70 = 160	85 + 80 = 165	100 + 70 = 170	95 + 75 = 170	**160**	**1**
3	100 + 70 = 170	90 + 80 = 170	95 + 70 = 165	105 + 75 = 180	**165**	**3**
4	110 + 70 = 180	105 + 80 = 185	120 + 70 = 190	110 + 75 = 185	**180**	**1**

Kemudian dilanjutkan pada stage-3 dan hasilnya seperti pada Tabel 10.6.

Tabel 10.6. Hasil Perhitungan Stage-3

X_3 / S_3	$f_3(S_3, X_3) = c_{s_3,x_3} + f_2^*(S_2)$				$f_3^*(S_3)$	x_3
	1	**2**	**3**	**4**		
Dari Towning	35 + 155 = 190	40 + 160 = 200	30 + 165 = 195	45 + 180 = 225	**190**	**1**

Pembacaan tabel-tabel ini untuk menentukan tingkat optimal adalah dari:

stage-3 → stage-2 → stage-1.

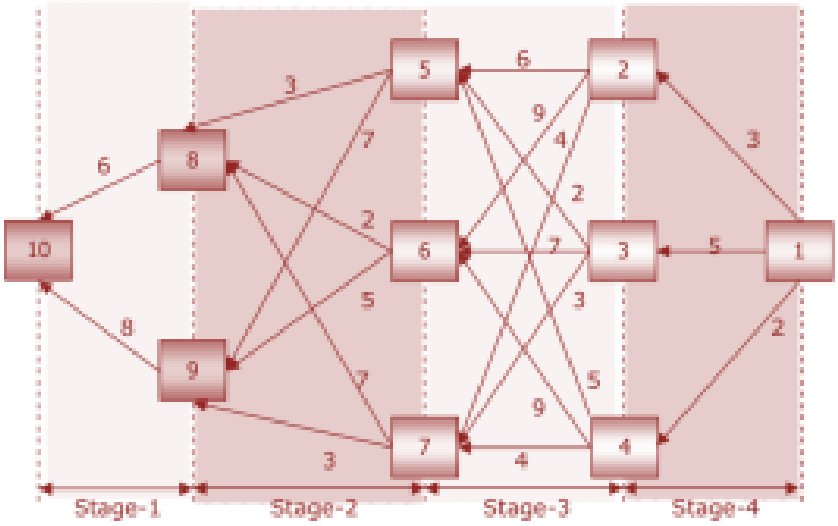
Dari Towing ke R&T Station melalui mesin-1, dengan biaya = $35. Dari R&T Station di mesin-1 ke D&R Station melalui mesin-3 dengan biaya = $85. Dari D&R Station di mesin-3 ke I&D Station melalui mesin-2 dengan biaya = $70. Total biaya = $35 + $85 + $70 = $190.

Jadwal optimal dari contoh 10.1 seperti pada Tabel 10.7.

Tabel 10.7. Jadwal Optimal Contoh 10.1

Dari Towing	I&D Station	D&R Station	R&T Station
Mesin	1	3	2
Biaya	35	85	70
Kumulatif Biaya	35	120	190

Contoh 10.2: The Shortest Route
Sebuah rute dari St. Louis ke San Francisco melalui beberapa kota antara seperti yang terlihat pada gambar jaringan berikut :



Solusi Contoh 10.2.

Stage-1:

X_1 / S_1	$f_1(S_1, X_1) = c_{s_1, x_1}$	$f_1^*(S_1)$	x_1^*
	10		
8	6	**6**	**8**
9	8	**8**	**9**

Angka pada setiap garis penghubung merupakan jarak. Dari gambar rute tersebut, maka persoalan ini dapat dibagi menjadi empat tahap (nomor tahap dimulai dari kota tujuan beurutan ke kota asal).

Stage-2:

X_2 / S_2	$f_2(S_2, X_2) = c_{s_2, x_2} + f_1^*(S_1)$		$f_2^*(S_2)$	x_2^*
	8	**9**		
5	3 + 6 = 9	7 + 8 = 15	**9**	**8**
6	2 + 6 = 8	5 + 8 = 13	**8**	**8**
7	7 + 6 = 13	3 + 8 = 11	**11**	**9**

Stage-3:

X_3 / S_3	$f_3(S_3, X_3) = c_{s_3, x_3} + f_2^*(S_2)$			$f_3^*(S_3)$	x_3^*
	5	**6**	**7**		
2	6 + 9 = 15	9 + 8 = 17	4 + 11 = 15	**15**	**5 , 7**
3	2 + 9 = 11	7 + 8 = 15	3 + 11 = 14	**11**	**5**
4	5 + 9 = 14	9 + 8 = 17	4 + 11 = 15	**14**	**5**

Stage-4:

X_4 / S_4	$f_4(S_4, X_4) = c_{s_4, x_4} + f_3^*(S_3)$			$f_4^*(S_4)$	x_4^*
	2	**3**	**4**		
1	3 + 15 = 18	5 + 11 = 16	2 + 14 = 16	**16**	**3 , 4**

Pembacaan tabel-tabel ini untuk menentukan tingkat optimal adalah dari:

stage-4 → stage-3 → stage-2 → stage-1.

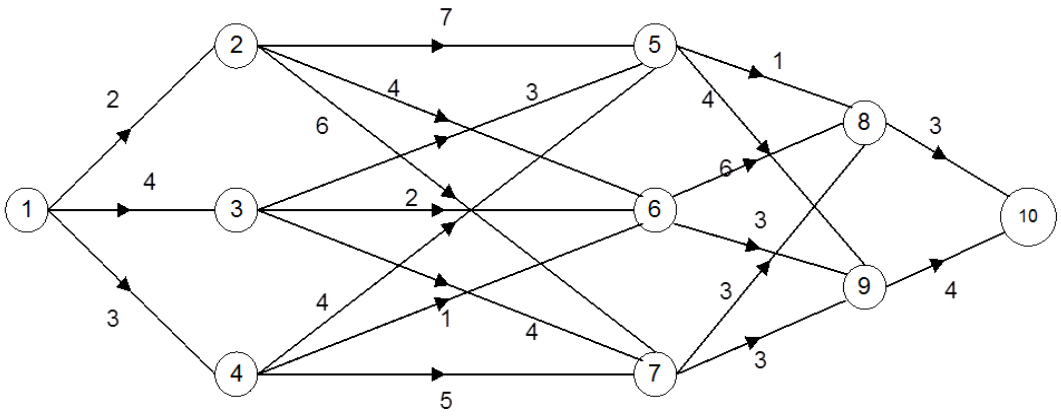
Rute optimal dengan jarak minimal adalah:

1 → 3 → 5 → 8 → 10, dengan total jarak = 5 + 2 + 3 + 6 = 16

Atau

1 → 4 → 5 → 8 → 10, dengan total jarak = 2 + 5 + 3 + 6 = 16

Contoh 10.3.
Tentukan lintasan terpendek dari simpul 1 ke simpul 10 sesuai dengan gambar berikut ini.



Solusi Contoh 10.3.
Penyelesaian dengan pemrograman 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:

$1 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$

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 di dalam grafik.

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

$f_4(s) = c_{sx4}$ (basis)

$f_k(s) = min_{xk} \{c_{sxk} + f_{k+1}(x_k)\}$, k = 1, 2, 3 (rekurens)

Keterangan:
– x_k : peubah keputusan pada tahap k (k = 1, 2, 3)
– c_{sxk} : 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 $f_1(1)$ dengan cara mencari $f_4(s)$, $f_3(s)$, $f_2(s)$ terlebih dahulu.

Tahap 4

$f_4(s) = c_{sx4}$

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

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

Tahap 3

$f_3(s) = min_{x3} \{c_{sx3} + f_4(x_3)\}$

x_3 / s	$f_3(s,x_3) = c_{s,x3} + f_4(x_3)$		Solusi Optimum	
	8	**9**	$f_3(s)$	x_3^*
5	4	8	4	8
6	9	7	7	9
7	6	7	6	8

Tahap 2

$f_2(s) = min_{x2} \{c_{sx2} + f_3(x_2)\}$

x_2 / s	$f_2(s,x_2) = c_{s,x2} + f_3(x_2)$			Solusi Optimum	
	5	**6**	**7**	$f_2(s)$	x_2^*
2	11	11	12	11	5 atau 6
3	7	9	10	7	5
4	8	8	11	8	5 atau 6

Tahap 1

$f_1(s) = min_{x1} \{c_{sx1} + f_2(x_1)\}$

x_1 / s	$f_1(s,x_1) = c_{s,x1} + f_2(x_1)$			Solusi Optimum	
	2	**3**	**4**	$f_1(s)$	x_1^*
1	13	11	11	11	3 atau 4

Solusi optimum telah diperoleh. Jadi ada tiga lintasan terpendek dari 1 ke 10, yaitu:

1) $1 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 10$

2) $1 \rightarrow 4 \rightarrow 5 \rightarrow 8 \rightarrow 10$

3) $1 \rightarrow 4 \rightarrow 6 \rightarrow 9 \rightarrow 10$

Panjang ketiga lintasan tersebut sama, yaitu 11.

Link Jurnal
https://jurnal.usu.ac.id/index.php/jti/article/view/6038/pdf

Kuis

1. Apa yang dimaksud dengan Pemrograman Dinamis (*Dynamic Programming*):
   a. Masalah multi tahap (*multistage*) dimana keputusan dibuat secara berurutan (*in sequence*)
   b. Metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah (*step*) atau tahapan (*stage*) sedemikian rupa sehingga solusi dari permasalahan ini dapat dipandang dari serangkaian keputusan-keputusan kecil yang saling berkaitan satu dengan yang lain
   c. Penyelesaian persoalan yang akan menghasilkan sejumlah berhingga pilihan yang mungkin dipilih, lalu solusi pada setiap tahap-tahap yang dibangun dari solusi pada tahap sebelumnya
   d. Metode yang menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap

2. Konsep dasar yang terdapat dalam pemrograman dinamis atau *dynamic programming* (DP), **kecuali**:
   a. Fungsi Rekursif
   b. *Multistage Graph*
   c. Fungsi Transisi
   d. Dekomposisi

3. Input dari persoalan dengan n-tahap pada pemrograman dinamis atau *dynamic programming* (DP), adalah:
   a. Ongkos (*cost*)
   b. *Decision variable*
   c. *Return* atau akibat dari setiap variabel keputusan yang dipilih
   d. Status baru yang menjadi input pada tahap berikutnya

4. Output dari persoalan dengan n-tahap pada pemrograman dinamis atau *dynamic programming* (DP), adalah:
   a. *State* pada tahap-n
   b. *Return* atau akibat dari setiap variabel keputusan yang dipilih
   c. *Decision variable*
   d. Ongkos (*cost*)

5. Sifat-sifat khusus yang terdapat *Multistage Graph*, **kecuali**:
   a. Setiap *edge*-nya memiliki *weight* (bobot)
   b. *Problem* mencari lintasan terpendek dari *source* ke *sink* pada sebuah *Multistage Graph*
   c. Hanya terdapat 1 *source* dan 1 *sink*
   d. Grafik berarah (*Directed Graph*)

Tugas

Jawablah pertanyaan dibawah ini yang bersumber dari modul dan jurnal yang saudara baca sebelumnya:

1. Dari link jurnal dalam pembelajaran ini, jelaskan:
   a. Latar belakang dan tujuan dari penelitian tersebut.
   b. Metode yang digunakan pada penelitian tersebut.
   c. Hasil dari penelitian tersebut.
   d. Manfaat dari hasil penelitian tersebut.

Referensi

Arga, W., 1985, Dinamika Dan Integer Programming, BPFE, Yogyakarta

Dimyati, Tjutju Tarliah dan Dimyati, Ahmad, 2006, Operation Research: Model-Model Pengambilan keputusan, Sinar Baru Algesindo, Bandung

Heizer, Jay dan Barry Render. 2009. *Operation Management*. Terjemahan oleh Dwianoegrawati Setyoningsih dan Indra Almahdy. Edisi 7. Buku I. Jakarta: Salemba Empat.

Lieberman, Gerald J., and Hiller, Fredrick S., 2008, Introduction operation research, edisi ke-8, ANDI, Yogyakarta

Nurcahyo. Widyat, 2005, Pengantar Teknik Industri, Modul Perkuliahan, Fakultas Teknik Universitas Tama Jagakarsa

Wignjosoebroto. S, 2003, Pengantar Teknik dan Manajemen Industri, Guna Widya