Program Studi Teknik Informatika – Universitas Widyatama

Strategi Algortima

Pertemuan : 09 - 12

Dosen Pembina: Danang Junaedi

Overview

- **Tujuan Instruksional**
- **Definisi Strategi Algortima**
- **Jenis Strategi Algoritma**
- *Direct solution strategies*
- *State-space base strategies*
- *Top-down solution strategies*
- *Bottom-up solution strategies*

Tujuan Instruksional

- **Mahasiswa akan dapat**
  - **Menjelaskan pengertian dan manfaat strategi algoritma**
  - **Menjelaskan aplikasi strategi algoritma**
  - **Menggunakan aplikasi strategi algoritma**

Definisi Strategi Algortima

- **Strategi** : rencana yang cermat mengenai kegiatan untuk mencapai sasaran khusus
- **Algoritma** : urutan langkah-langkah untuk memecahkan suatu masalah
- **Strategi Algoritma** : kumpulan metode atau teknik untuk memecahkan masalah guna mencapai tujuan yang ditentukan, yang dalam hal ini deskripsi metode atau teknik tersebut dinyatakan dalam suatu urutan langkah-langkah penyelesaian

Jenis Strategi Algoritma

1. Strategi solusi langsung *(direct solution strategies)*
   - *Brute force*
   - *Greedy*
2. Strategi berbasis pencarian pada ruang status *(state-space base strategies)*
   - *Backtracking*
   - *Branch and Bound*
3. Strategi solusi atas-bawah *(top-down solution strategies)*
   - *Divide and Conquer*
   - *Transform and Conquer*
   - *Increase and Conquer*
4. Strategi solusi bawah-atas *(bottom-up solution strategies)*
   - *Dynamic Programming.*

Brute Force

- *Brute force* : pendekatan yang lempang (*straightforward*) untuk memecahkan suatu masalah
- Biasanya didasarkan pada:
  - pernyataan masalah (*problem statement*)
  - definisi konsep yang dilibatkan.
- Algoritma *brute force* memecahkan masalah dengan
  - sangat sederhana,
  - langsung,
  - jelas (*obvious way*).
- *Just do it!*

Contoh Brute Force

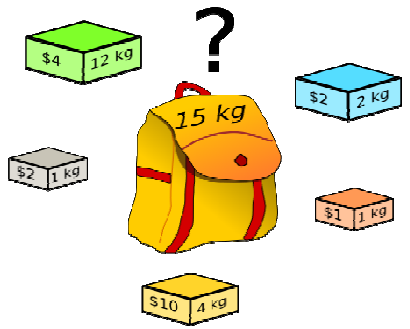
- Menghitung an (a > 0, n adalah bilangan bulat tak-negatif)
  Definisi:
  a^n = a x a x ... x a (n kali) , jika n > 0
  = 1 , jika n = 0
  Algoritma brute force: kalikan 1 dengan a sebanyak n kali
- Algoritma Pengurutan Brute Force
  - Algoritma apa yang paling lempang dalam memecahkan masalah pengurutan?
    Bubble sort dan selection sort!
  - Kedua algoritma ini memperlihatkan teknik brute force dengan jelas sekali.

Exhaustive Search

- A brute force solution to a problem involving search for an element with a special property, usually among combinatorial objects such as permutations, combinations, or subsets of a set.
- Method:
  - generate a list of all potential solutions to the problem in a systematic manner
  - evaluate potential solutions one by one, disqualifying infeasible ones and, for an optimization problem, keeping track of the best one found so far
  - when search ends, announce the solution(s) found

Contoh Exhaustive Search

- Persoalan 0/1 *Knapsack* dapat kita pandang sebagai mencari himpunan bagian (*subset*) dari keseluruhan objek yang muat ke dalam *knapsack* dan memberikan total keuntungan terbesar.



Contoh Exhaustive Search(contd)

- Contoh: n = 4.

  w1 = 2; p1 = 20
  w2 = 5; p2 = 30
  w3 = 10; p3 = 50
  w4 = 5; p4 = 10

  Kapasitas knapsack K = 16
- Langkah-langkah pencarian solusi 0/1 Knapsack secara exhaustive search dirangkum dalam tabel di sebelah kanan
- Himpunan bagian objek yang memberikan keuntungan maksimum adalah {2, 3} dengan total keuntungan adalah 80.
- Solusi: $X = \{0, 1, 1, 0\}$

Himpunan Bagian	Total Bobot	Total keuntungan
{}	0	0
{1}	2	20
{2}	5	30
{3}	10	50
{4}	5	10
{1, 2}	7	50
{1, 3}	12	70
{1, 4}	7	30
{2, 3}	**15**	**80**
{2, 4}	10	40
{3, 4}	15	60
{1, 2, 3}	17	tidak layak
{1, 2, 4}	12	60
{1, 3, 4}	17	tidak layak
{2, 3, 4}	20	tidak layak
{1, 2, 3, 4}	22	tidak layak

Brute-Force Strengths and Weaknesses

- Strengths
  - wide applicability
  - simplicity
  - yields reasonable algorithms for some important problems (e.g., matrix multiplication, sorting, searching, string matching)
- Weaknesses
  - rarely yields efficient algorithms
  - some brute-force algorithms are unacceptably slow
  - not as constructive as some other design techniques

Greedy

- *A greedy algorithm always makes the choice that looks best at the moment.*
- *The greedy algorithm obtains an optimal solution to a problem by making a sequence of choices. For each decision point in the algorithm, the choice that seems best at the moment is chosen. This strategy does not always produce an optimal solution. But in some cases it does.*
- *Optimization problem: there can be many possible solution. Each solution has a value, and we wish to find a solution with the optimal (minimum or maximum) value*
- *Greedy Principal: "take what you can get now!".*
- *Greedy algorithm: an algorithmic technique to solve optimization problems*
  - *Always makes the choice that looks best at the moment.*
  - *Makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution*

Pseudo Code Greedy

procedure greedy(input C: himpunan_kandidat; output S : himpunan_solusi)
{ menentukan solusi optimum dari persoalan optimasi dengan algoritma greedy
Masukan: himpunan kandidat C
Keluaran: himpunan solusi S }
Deklarasi
x : kandidat;
Algoritma:
S←{} *{ inisialisasi S dengan kosong }*
while (belum SOLUSI(S)) and (C ≠ {}) do
x←SELEKSI(C); *{ pilih sebuah kandidat dari C}*
C← C - {x} *{ elemen himpunan kandidat berkurang satu }*
if LAYAK(S ∪ {x}) then S←S ∪ {x} endif
endwhile
{SOLUSI(S) sudah diperoleh or C = {} }

Elemen-Elemen *Greedy*

- Himpunan kandidat : berisi elemen-elemen pembentuk solusi.
- Himpunan solusi : berisi kandidat-kandidat yang terpilih sebagai solusi persoalan.
- Fungsi seleksi (*selection function*) : fungsi untuk memilih kandidat yang paling memungkinkan mencapai solusi optimal. Kandidat yang sudah dipilih pada suatu langkah tidak pernah dipertimbangkan lagi pada langkah selanjutnya.
- Fungsi kelayakan (*feasible*): fungsi untuk 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 kandidat yang tidak layak dibuang dan tidak pernah dipertimbangkan lagi.
- Fungsi obyektif, yaitu fungsi yang memaksimumkan atau meminimumkan nilai solusi (misalnya panjang lintasan, keuntungan, dan lain-lain)

Contoh Greedy

- Penukaran Uang
  Diberikan uang senilai *32*. Tukar *uang tersebut* dengan koin-koin uang yang ada (misalnya terdiri dari pecahan 1, 5, 10, dan 25). Berapa jumlah minimum koin yang diperlukan untuk penukaran tersebut?

Penyelesaian :

- Strategi *greedy* yang digunakan adalah: Pada setiap langkah, pilihlah koin dengan nilai sebesar mungkin dari himpunan koin yang tersisa dengan syarat (kendala) tidak melebihi nilai uang yang ditukarkan.
  - *Langkah 1*: pilih 1 buah koin 25 (Total = 25)
  - *Langkah 2*: pilih 1 buah koin 5 (Total = 25 + 5 = 30)
  - *Langkah 3*: pilih 2 buah koin 1 (Total = 25+5+1+1= 32)
- Solusi: Jumlah koin minimum = 4 (solusi optimal!)
- Pada setiap langkah di atas kita memperoleh optimum lokal, dan pada akhir algoritma kita memperoleh optimum global (yang pada contoh ini merupakan solusi optimum).

Contoh Greedy (Contd)

- **Himpunan kandidat :** himpunan koin yang merepresentasikan nilai 1, 5, 10, 25, paling sedikit mengandung satu koin untuk setiap nilai.
- **Himpunan solusi :** total nilai koin yang dipilih tepat sama jumlahnya dengan nilai uang yang ditukarkan.
- **Fungsi seleksi :** pilihlah koin yang bernilai tertinggi dari himpunan kandidat yang tersisa.
- **Fungsi layak :** memeriksa apakah nilai total dari himpunan koin yang dipilih tidak melebihi jumlah uang yang harus dibayar.
- **Fungsi obyektif :** jumlah koin yang digunakan minimum.

Contoh Greedy(Contd)

- Contoh Persoalan 0/1 *Knapsack*

  n = 4

  w1 = 6; p1 = 12

  w2 = 5; p2 = 15

  w3 = 10; p3 = 50

  w4 = 5; p4 = 10

  Kapasitas knapsack K = 16

Properti objek				*Greedy by*			Solusi
i	w_i	p_i	p_i/w_i	*profit*	*weight*	*density*	Optimal
1	6	12	2	0	1	0	0
2	5	15	3	1	1	1	1
3	10	50	5	1	0	1	1
4	5	10	2	0	1	0	0
Total bobot				15	16	15	15
Total keuntungan				65	37	65	65

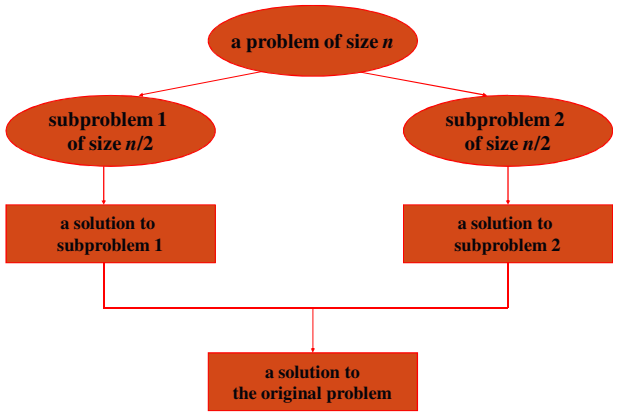
- Langkah-langkah pencarian solusi 0/1 Knapsack menggunakan strategi greedy dirangkum dalam tabel di sebelah kanan
- Himpunan bagian objek yang memberikan keuntungan maksimum adalah {2, 3} dengan total keuntungan adalah 65.
- Solusi: $X = \{0, 1, 1, 0\}$

Devide & Conquer

- **Definisi**
  - *Divide:* membagi masalah menjadi beberapa bagian-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama),
  - *Conquer:* memecahkan/menyelesaikan masing-masing bagian-masalah (**secara rekursif**), dan
  - *Combine:* mengabungkan solusi masing-masing bagian-masalah sehingga membentuk solusi masalah semula.
- **Proses**

  if $n \le n_0$ **then** ***{ukuran masalah sudah cukup kecil}***
  SOLVE bagian-masalah yang berukuran n ini
  else
  Bagi menjadi 2 bagian-masalah, masing-masing berukuran n/2
  DIVIDE_and_CONQUER(bagian-masalah pertama yang berukuran n/2)
  DIVIDE_and_CONQUER(bagian-masalah kedua yang berukuran n/2)
  COMBINE solusi dari 2 bagian-masalah di atas
  endif

Divide-and-Conquer Technique



Divide-and-Conquer Examples

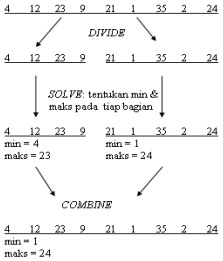
- Sorting: Mergesort and Quicksort
- Binary tree traversals
- Binary search (?)
- Etc…

Contoh Divide-and-Conquer

Mencari nilai minimum dan maksimum pada array yang tidak terurut

Proses :

1. Untuk kasus $n = 1$ atau $n = 2$,
   Jika $n = 1$, maka $min = maks = An$.
   Jika $n = 2$, maka bandingkan kedua elemen untuk menentukan *min* dan *maks*.
2. Untuk kasus $n > 2$,
   - DIVIDE: Bagi dua tabel *A* secara rekursif menjadi dua bagian yang berukuran sama, yaitu bagian kiri dan bagian kanan.
   - CONQUER: Terapkan algoritma *Divide and Conquer* untuk masing-masing bagian, dalam hal ini *min* dan *maks* dari tabel bagian kiri dinyatakan dalam peubah *min1* dan *maks1*, dan *min* dan *maks* dari tabel bagian kanan dinyatakan dalam peubah *min2* dan *maks2*.
   - COMBINE: Bandingkan *min1* dengan *min2* untuk menentukan *min* tabel *A* dan/atau Bandingkan *maks1* dengan *maks2* untuk menentukan *maks* tabel *A*.

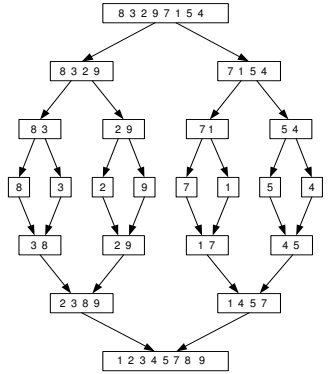


Contoh Divide-and-Conquer

Merge Sort

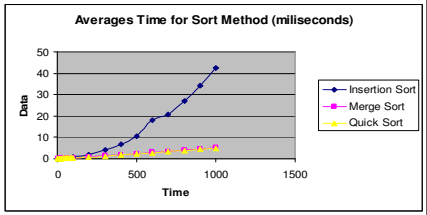
Proses :

1. Untuk kasus $n = 1$, maka tabel *A* sudah terurut dengan sendirinya
2. Untuk kasus $n > 1$,
   - DIVIDE: bagi tabel *A* menjadi dua bagian, bagian kiri dan bagian kanan, masing-masingbagian berukuran $n/2$ elemen.
   - CONQUER: Terapkan algoritma *Divide and Conquer* untuk masing-masing bagian, secara rekursif.
   - COMBINE: gabung hasil pengurutan kedua bagian sehingga diperoleh tabel *A* yang terurut.



Perbandingan Metoda *Sorting*

N	*Insertion Sort*	*Merge Sort*	*Quick Sort*
0	0.09	0.09	0.09
10	0.12	0.13	0.12
20	0.14	0.16	0.15
30	0.18	0.2	0.18
40	0.22	0.24	0.22
50	0.27	0.27	0.26
60	0.33	0.32	0.31
70	0.4	0.37	0.33
80	0.48	0.41	0.37
90	0.56	0.45	0.41
100	0.65	0.49	0.44
200	1.99	0.93	0.86
300	4.08	1.45	1.29
400	6.9	1.95	1.75
500	10.7	2.43	2.23
600	18.15	3.01	2.73
700	20.6	3.56	3.5
800	26.9	4.12	3.67
900	34.35	4.62	4.38
1000	42.3	5.2	4.73



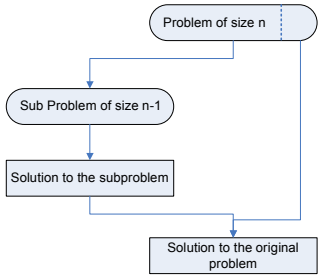
Decrease-and-Conquer

1. Reduce problem instance to smaller instance of the same problem
2. Solve smaller instance
3. Extend solution of smaller instance to obtain solution to original instance

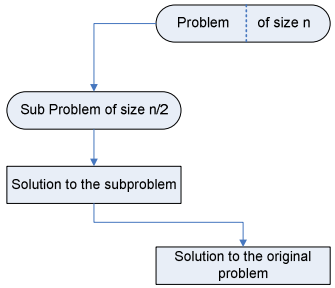
- Can be implemented either top-down or bottom-up
- Also referred to as *inductive* or *incremental* approach

Decrease-and-Conquer Technique

- Decrease (by one)-and-conquer technique



- Decrease (by half)-and-conquer technique



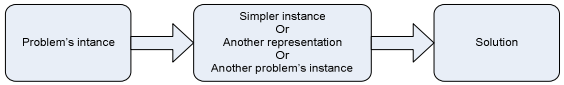
3 Types of Decrease and Conquer

- *Decrease by a constant* (usually by 1):
  - insertion sort
  - graph traversal algorithms (DFS and BFS)
  - topological sorting
  - algorithms for generating permutations, subsets
- *Decrease by a constant factor* (usually by half)
  - binary search and bisection method
  - exponentiation by squaring
  - multiplication à la russe
- *Variable-size decrease*
  - Euclid's algorithm
  - selection by partition
  - Nim-like games

Transform and Conquer

This group of techniques solves a problem by a *transformation*

- to a simpler/more convenient instance of the same problem (*instance simplification*)
- to a different representation of the same instance (*representation change*)
- to a different problem for which an algorithm is already available (*problem reduction*)



Taxonomy of Searching Algorithms

- List searching
  - sequential search
  - binary search
  - interpolation search
- Tree searching
  - binary search tree
  - binary balanced trees: AVL trees, red-black trees
  - multiway balanced trees: 2-3 trees, 2-3-4 trees, B trees
- Hashing
  - open hashing (separate chaining)
  - closed hashing (open addressing)

Studi Kasus

Cari Program untuk kasus di bawah ini (pilih salah satu)

- ***Greedy***
  - Kasus Penukaran Uang
  - *Penjadwalan*
  - *Schedulling with Deadlines*
  - *Knapscak 1/0*
  - *Dst*
- ***Devide & Conquer***
  - *Merge Sort*
  - *Shell Sort*
  - *Quick Sort*
  - *Dst*

Bactracking

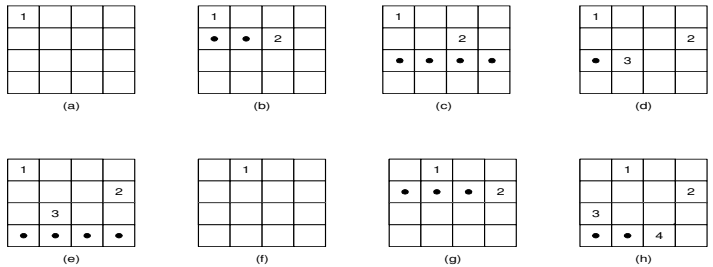
- **Definisi**
  - *Backtracking*, yang merupakan perbaikan dari algoritma *brute-force*, secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada.
  - Dengan metode *Backtracking*, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan. Akibatnya, waktu pencarian dapat dihemat.
  - *Backtracking* merupakan bentuk tipikal dari algoritma **rekursif**.
  - Saat ini algoritma *Backtracking* banyak diterapkan untuk program *games* (seperti permainan *tic-tac-toe*, menemukan jalan keluar dalam sebuah labirin, catur, dll) dan masalah-masalah pada bidang kecerdasan buatan (*artificial intelligence*).
- **Proses**

```
for tiap X bagian/solusi yang belum dicobasedemikian do
    if X merupakan solusi then
        CetakSolusi(X)
    endif
    Backtracking(k+1)
endfor
```

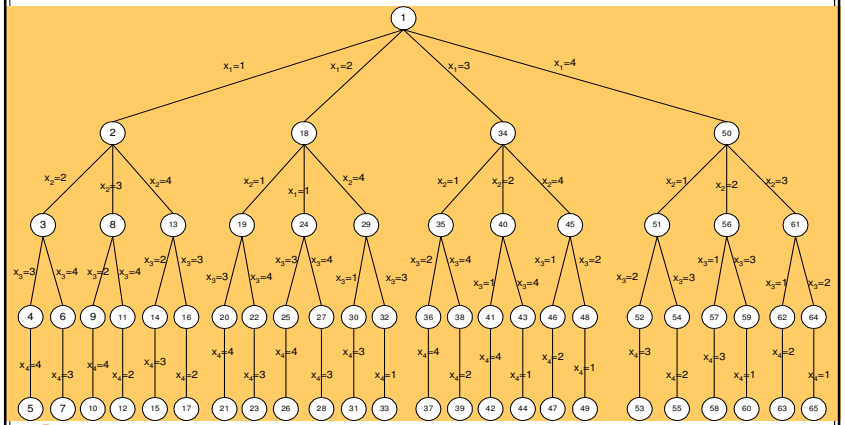

Contoh Backtracking

Kasus Backtracking : n - *queen problem*

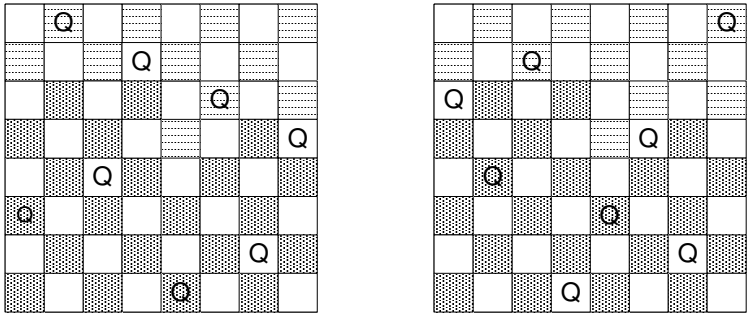
- Misalkan disediakan sebuah papan catur dengan ukuran 4 x 4 (4 - *queen problem*)
- Hasil yang diinginkan adalah menempatkan setidaknya 4 buah ratu pada papan catur tersebut sesuai ketentuan yang berlaku pada permainan catur
- *Contoh Solusi :....Proses?!?(Cari sendiri)*



Contoh: Pohon ruang-status persoalan 4-Ratu

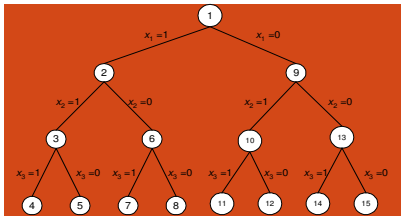


Contoh 2 buah solusi *8-queen problem*:



Contoh Bactracking (Contd)

- Contoh Persoalan 0/1 *Knapsack*
  n = 3
  $(w_1, w_2, w_3) = (35, 32, 25)$
  $(p_1, p_2, p_3) = (40, 25, 50)$
  Kapasitas knapsack K = 30
- Solusi dinyatakan sebagai $X = (x_1, x_2, x_3)$, $x_i \in \{0, 1\}$.
- Ruang solusi 0/1 Knapsack menggunakan strategi backtracking dirangkum dalam gambar di sebelah kanan
- Himpunan bagian objek yang memberikan keuntungan maksimum adalah {3} dengan total keuntungan adalah 50
- Solusi: $X = \{0, 0, 1\}$



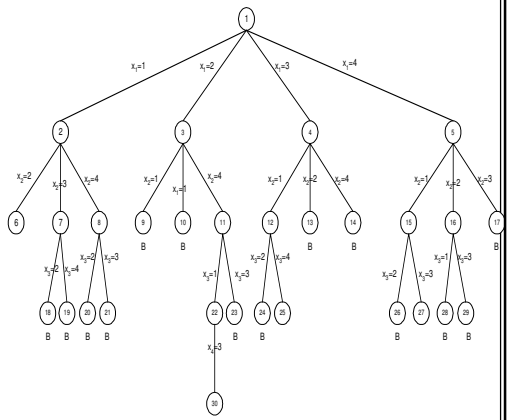
Branch & Bound

- *Branch and Bound is a general search method.*
- *Starting by considering the root problem (the original problem with the complete feasible region), the lower-bounding and upper-bounding procedures are applied to the root problem.*
- *If the bounds match, then an optimal solution has been found and the procedure terminates*
- *Otherwise, the feasible region is divided into two or more regions, these subproblems partition the feasible region.*
- *The algorithm is applied recursively to the subproblems. If an optimal solution is found to a subproblem, it is a feasible solution to the full problem, but not necessarily globally optimal.*
- *We Can search for an optimal solution in three steps:*
  - *Define the cost variable.*
  - *Link the cost variable to other variables in the problem.*
  - *Define the search for optimal solutions.*

Contoh Branch & Bound

Kasus Backtracking : 4 - *queen problem*

- *Pohon ruang status yang terbentuk untuk persoalan 4-Ratu dengan metode BFS*, dapat dilihat pada gambar sebelah kanan
- Solusi dinyatakan sebagai $X = (x_1, x_2, x_3, x_4)$, $x_i \in \{0, 1\}$.
- Solusi pertama dicapai pada simpul 30, yaitu $X = (2, 4, 1, 3)$
- Simpul hidup yang menjadi simpul-E ialah simpul yang mempunyai nilai batas terkecil (strategi **pencarian berdasarkan biaya terkecil** (*least cost search*)



Contoh *Branch & Bound*(Contd)

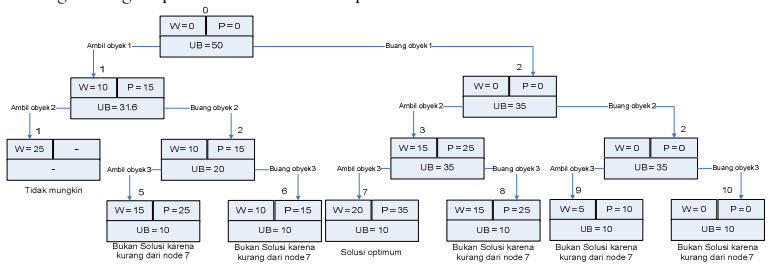
- Knapsack 1/0

  Jika kita mempunyai N buah objek yang memiliki berat W dan harga P. Objek-objek tersebut akan dimasukkan ke dalam suatu kantung yang memiliki kapasitas K. permasalahannya adalah objek-objek mana yang dipilih untuk dimasukkan agar kantong dapat terisi penuh dan dengan keuntungan tertinggi. Contoh data :
  - K = 20 {kapasitas kantong}
  - N = 3 {jumlah objek}
  - P = {15, 25, 10}, harga untuk masing-masing objek
  - W = {10, 15, 5}, berat untuk masing-masing objek

  Objek yang diperkenankan untuk dimasukkan ke dalam kantong adalah dalam keadaan utuh (tidak sebagian). Objek i → Xi maka (Xi = 0 atau Xi = 1) dari objek i yang dapat dimasukkan ke dalam kantong.

Contoh *Branch & Bound*(Contd)

- Contoh Persoalan 0/1 *Knapsack dengan* n = 3; $(w_1, w_2, w_3) = (10, 15, 5)$; $(p_1, p_2, p_3) = (15, 25, 10)$; dan Kapasitas knapsack K = 20; Solusi dinyatakan sebagai $X = (x_1, x_2, x_3)$, $x_i \in \{0, 1\}$.
- Langkah-langkah pencarian solusi 0/1 Knapsack :



- UB dihitung dengan rumus $p_i+(K-w)*(p_{i+1}/w_{i+1})$
- Solusi: $X = \{0, 1, 1\}$, Himpunan bagian objek yang memberikan keuntungan maksimum adalah {2,3} dengan total keuntungan adalah 35

Dynamic Programming

- **Program Dinamis** (*dynamic programming*):
  - 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.
- **Pada penyelesaian persoalan dengan metode ini:**
  1. Terdapat sejumlah berhingga pilihan yang mungkin,
  2. Solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya,
  3. Kita menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

Pendekatan *Dynamic Programming*

- Dua pendekatan yang digunakan
  - maju (*forward* atau *up-down*) dan
  - 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.

Langkah-langkah 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 Dynamic Programming

Kasus : Integer (1/0) *Knapsack*

- Pada persoalan ini,
  1. Tahap (k) adalah proses memasukkan barang ke dalam karung (*knapsack*) (ada 3 tahap).
  2. Status (y) menyatakan kapasitas muat karung yang tersisa setelah memasukkan barang pada tahap sebelumnya.
- Dari tahap ke-1, kita masukkan objek ke-1 ke dalam karung untuk setiap satuan kapasitas karung sampai batas kapasitas maksimumnya. Karena kapasitas karung adalah bilangan bulat, maka pendekatan ini praktis.
- Misalkan ketika memasukkan objek pada tahap k, kapasitas muat karung sekarang adalah $y - w_k$.
- Untuk mengisi kapasitas sisanya, kita menerapkan prinsip optimalitas dengan mengacu pada nilai optimum dari tahap sebelumnya untuk kapasitas sisa $y - w_k$ (yaitu $f_{k-1}(y - w_k)$).

Contoh Dynamic Programming(Contd)

- Selanjutnya, kita bandingkan nilai keuntungan dari objek pada tahap k (yaitu pk) plus nilai $f_{k-1}(y - w_k)$ dengan keuntungan pengisian hanya $k - 1$ macam objek, $f_{k-1}(y)$.
- Jika $p_k + f_{k-1}(y - w_k)$ lebih kecil dari $f_{k-1}(y)$, maka objek yang ke-k tidak dimasukkan ke dalam karung, tetapi jika lebih besar, maka objek yang ke-k dimasukkan
- Relasi rekurens untuk persoalan ini adalah

$$f_0(y) = 0, \quad y = 0, 1, 2, \ldots, M \qquad \text{(basis)}$$
$$f_k(y) = -\infty, \quad y < 0 \qquad \text{(basis)}$$

$$f_k(y) = \max\{f_{k-1}(y), p_k + f_{k-1}(y - w_k)\}, \quad \text{dimana } k = 1, 2, \ldots, n \qquad \text{(rekurens)}$$

- $f_k(y)$ adalah keuntungan optimum dari persoalan 0/1 *Knapsack* pada tahap k untuk kapasitas karung sebesar y.
- $f_0(y) = 0$ adalah nilai dari persoalan *knapsack* kosong (tidak ada persoalan *knapscak*) dengan kapasitas y,
- $f_k(y) = -\infty$ adalah nilai dari persoalan knapsack untuk kapasitas negatif. Solusi optimum dari persoalan 0/1 *Knapsack* adalah $f_n(M)$.

Contoh Dynamic Programming (Contd)

- Contoh Persoalan 0/1 *Knapsack untuk* n = 3; (w_1, w_2, w_3) = (2, 3, 1); (p_1, p_2, p_3) = (65, 80, 30); dan Kapasitas knapsack K = 5
- Langkah-langkah pencarian solusi 0/1 Knapsack menggunakan strategi dynamic programming adalah sebagai berikut

Tahap 1:

$$f_1(y) = \max\{f_0(y), p_1 + f_0(y - w_1)\}$$
$$= \max\{f_0(y), 65 + f_0(y - 2)\}$$

y	$f_0(y)$	$65 + f_0(y-2)$	Solusi Optimum $f_1(y)$	Solusi Optimum (x_1^*, x_2^*, x_3^*)
0	**0**	-∞	0	(0, 0, 0)
1	**0**	-∞	0	(0, 0, 0)
2	0	**65**	65	(1, 0, 0)
3	0	**65**	65	(1, 0, 0)
4	0	**65**	65	(1, 0, 0)
5	0	**65**	65	(1, 0, 0)

Contoh (Contd)

Tahap 2:

$f_2(y) = \max\{f_1(y), p_2 + f_1(y - w_2)\}$
$= \max\{f_1(y), 80 + f_1(y - 3)\}$

			Solusi Optimum	
y	$f_1(y)$	$80 + f_1(y-3)$	$f_2(y)$	(x_1^*, x_2^*, x_3^*)
0	**0**	$80 + (-\infty) = -\infty$	0	(0, 0, 0)
1	**0**	$80 + (-\infty) = -\infty$	0	(0, 0, 0)
2	**65**	$80 + (-\infty) = -\infty$	65	(1, 0, 0)
3	65	$80 + 0 = \mathbf{80}$	80	(0, 1, 0)
4	65	$80 + 0 = \mathbf{80}$	80	(0, 1, 0)
5	65	$80 + 65 = \mathbf{145}$	145	(1, 1, 0)

Tahap 3:

$f_3(y) = \max\{f_2(y), p_3 + f_2(y - w_3)\}$
$= \max\{f_2(y), 30 + f_2(y - 1)\}$

			Solusi Optimum	
y	$f_2(y)$	$30 + f_2(y-1)$	$f_3(y)$	(x_1^*, x_2^*, x_3^*)
0	**0**	$30 + (-\infty) = -\infty$	0	(0, 0, 0)
1	**0**	$30 + (-\infty) = -\infty$	0	(0, 0, 0)
2	**65**	$30 + 0 = 30$	65	(1, 0, 0)
3	80	$30 + 65 = \mathbf{95}$	95	(1, 0, 1)
4	80	$30 + 80 = \mathbf{110}$	110	(0, 1, 1)
5	**145**	$30 + 80 = 110$	145	(1, 1, 0)

Solusi optimum $X = (1, 1, 0)$ dengan $\sum p = f = 145$.

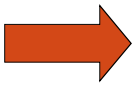
Studi Kasus

Cari Program untuk kasus di bawah ini (pilih salah satu)

- ***Backtracking***
  - Langkah Ratu (4x4, 8x8 dst) atau Langkah Kuda atau Langkah Benteng
  - Puzzle
  - Knapsack 1/0, dst
- ***Branch & Bound***
  - Knapsack 1/0
  - Puzzle, dst

Susunan Awal

1	3	4	15
2		5	12
7	6	11	14
8	9	10	13



Susunan Akhir

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- ***Dynamic Programming***
  - Fibonacci
  - Knapsack 1/0, dst

Referensi

- Levitin, Anany, Introduction to Design and Analysis of Algorithms, Pearson Addison-Wesley, 2007
- Rinaldi Munir, Strategi Algoritma, ITB, 2004

Untuk bahan renungan bersama

- Ketika satu pintu kebahagiaan tertutup, pintu yang lain dibukakan. Tetapi sering kali kita terpaku terlalu lama pada pintu yang tertutup sehingga tidak melihat pintu lain yang dibukakan bagi kita.
- Doa memberikan kekuatan pada orang yang lemah, membuat orang tidak percaya menjadi percaya dan memberikan keberanian pada orang yang ketakutan. Janganlah berputus asa. Tetapi kalau kamu sampai berada dalam keadaan putus asa, berjuanglah terus meskipun dalam keadaan putus asa.
- Aku tidak pernah menyesal dalam menjalani hidup kecuali saat aku tidak bisa mengambil pelajaran dan bersyukur
- Seseorang manusia harus cukup rendah hati untuk mengakui kesalahan ,cukup bijak untuk mengambil manfaat daripada kegagalannya dan cukup berani untuk membetulkan kesalahan
- Dalam hidup,terkadang kita lebih banyak mendapatkan apa yang tidak kita inginkan. Dan ketika kita mendapatkan apa yang kita inginkan, akhirnya kita tahu bahwa yang kita inginkan terkadang tidak dapat membuat hidup kita menjadi lebih bahagia.
- Jadikan dirimu bagai pohon yang rendang di mana orang dapat berteduh. Jangan seperti pohon kering tempat sang pungguk melepas rindu dan hanya layak dibuat kayu bakar
- Bermimpilah tentang apa yang ingin kamu impikan, pergilah ke tempat-tempat kamu ingin pergi. Jadilah seperti yang kamu inginkan, karena kamu hanya memiliki satu kehidupan dan satu kesempatan untuk melakukan hal-hal yang ingin kamu lakukan.
- Masa depan yang cerah berdasarkan pada masa lalu yang telah dilupakan. Kamu tidak dapat melangkah dengan baik dalam kehidupan kamu sampai kamu melupakan kegagalan kamu dan rasa sakit hati.

Untuk bahan renungan bersama

- Dunia ini umpama lautan yg luas. Kita adalah kapal yg belayar dilautan, telah banyak kapal karam didalamnya. Andai muatan kita adalah iman dan layarnya takwa niscaya kita akan selamat dari tersesat di lautan hidup ini.
- Siapakah orang yang BODOH ?

  Orang yang boDoh ialah orang yang suka menceritakan dirinya sendiri, padahal orang lain ga pernah tanya. Maka jangan sekali kali menceritakan dirimu sendiri klo ga ditanya.
- Siapakah orang yang PANDAI ?

  Orang yang pandai adalah orang2 yang suka bekerja tanpa pamrih dan suka menolong tanpa diminta dan tanpa melihat asal dan golongan nya. Maka segeralah menolong tanpa diminta
- Siapakah orang yang sibuk?

  Orang yang sibuk adalah orang yang suka menyepelekan waktu solatnya seolah-olah ia mempunyai kerajaan seperti kerajaan Nabi Sulaiman a.s. Maka sempatkanlah bagimu untuk beribadah.....dan bersegeralah!
- Siapakah orang yang manis senyumanya?

  Orang yang mempunyai senyuman yang manis adalah orang yang ditimpa musibah lalu dia berucap "Inna lillahi wainna illaihi rajiuun." Kemudian berkata,"Ya Rabbi, Aku ridha dengan ketentuanMu ini", sambil mengukir senyuman. Maka berbaik hatilah dan bersabar.....
- Siapakah orang yang kaya ?

  Orang yang kaya adalah orang yang bersyukur dengan apa yang ada dan tidak lupa akan kenikmatan dunia yang sementara ini. Maka bersyukurlah atas nikmat yang kau terima dan berbagilah.....
- Siapakah orang yang miskin?

  Orang yang miskin adalah orang tidak puas dengan nikmat yang ada, selalu menumpuk-numpukkan harta. Maka janganlah kau menjadi kikir juga dengki..... .