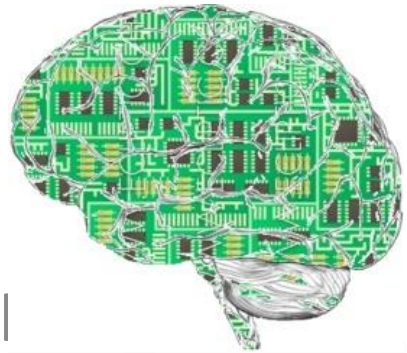
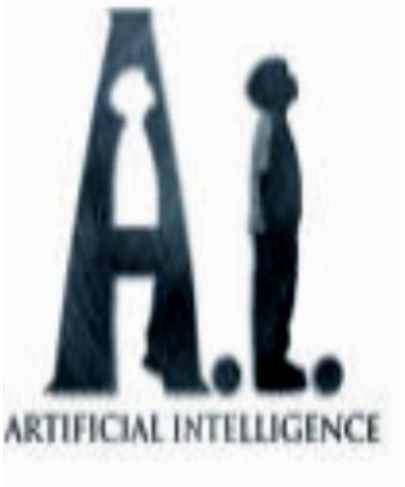
KECERDASAN BUATAN

SEARCHING

SKS/JS : 3/3



DR. Eng. ANIK NUR HANDAYANI



SEARCHING

Search technique that is solving problems that present a problem into a state space (state) and are systematically generating and testing state-of the initial state to find a goal state.

Metode-metode pencarian

Dibedakan dalam dua jenis:

- Pencarian buta/tanpa informasi (*Blind Un-informed search*)
- Pencarian Heuristik/dengan informasi (*Informed search*)

Setiap metode mempunyai karakteristik yang berbeda-beda dengan kelebihan dan kekurangan masing-masing.

Untuk mengukur performansi metode pencarian, terdapat empat kriteria yang dapat digunakan:

- **Completeness :** Apakah metode tersebut menjamin penemuan solusi jika solusinya memang ada?
- **Time complexity :** Berapa lama waktu yang diperlukan?
- **Space complexity :** Berapa banyak memori yang diperlukan?
- **Optimality :** Apakah metode tersebut menjamin menemukan solusi yang terbaik jika terdapat beberapa solusi berbeda?

b = faktor percabangan (juml. Simpul child/ sucessor node yang dimiliki dari suatu simpul parent/root node)

d = kedalaman solusi

Jumlah simpul yang harus disimpan sebanyak O(b pangkat d)

Misal : b = 10 dan d = 8

Maka BFS membangkitkan dan menyimpan sebanyak $10^0 + ... + 10^8 = 111.111.111 = 10^8$

Depth	Nodes	Time	Memory
2	1100	11 seconds	1 megabytes
4	111.100	11 seconds	106 megabytes
6	10^{7}	19 minutes	10 gigabytes
8	10^{9}	31 hours	1 tetabytes
10	10^{11}	129 days	101 terbytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3.523 years	1 exabyte

Time and memory requirements for breadth-firs serach. The numbers shown assuming brnching factor b = 10; 10.000 nodes/second; 1000 bytes/node

BLIND/UN-INFORMED SEARCH

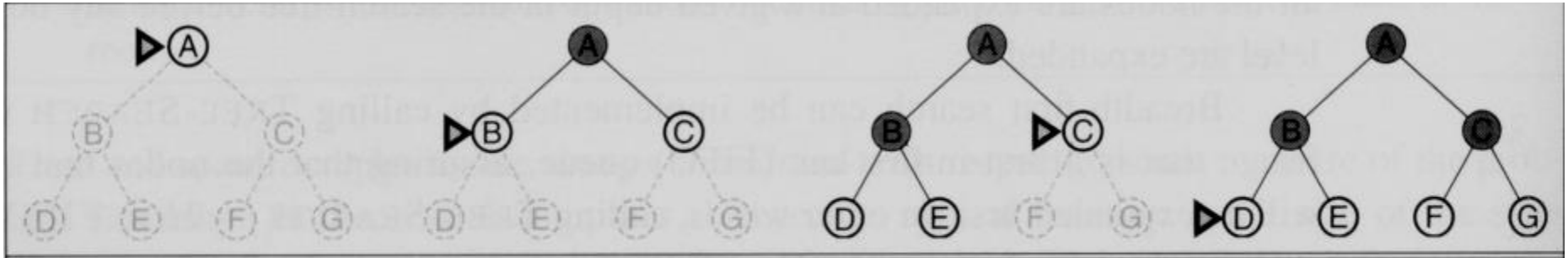
Tidak ada informasi awal yang digunakan dalam proses pencarian.

- √ Breadth-First Search (BFS)
- √ Depth-First Search (DFS)
- Depth-Limited Search (DLS)
- Uniform Cost Search (UCS)
- Iterative-Deepening Search (IDS)
- Bi-Directional Search(BDS)

Breadth-First Search (BFS)

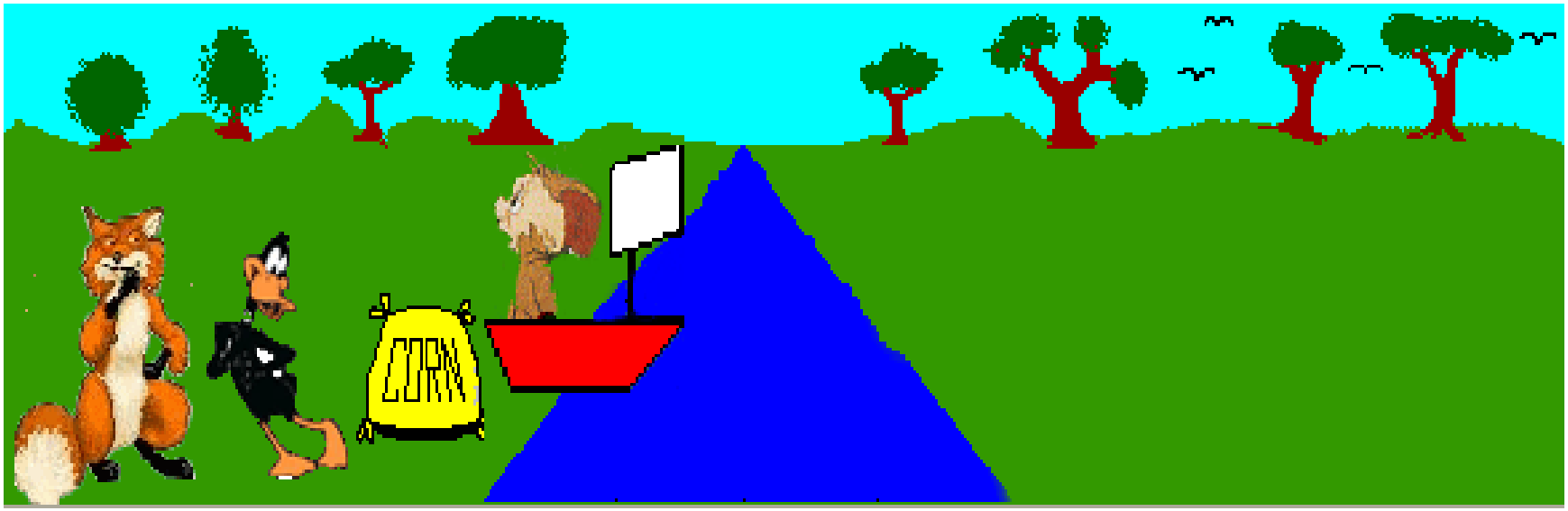
- Breadth-first serach is a simple strategy in which the root node is expanded firs, then all the successors of the root node are expanded next, then their successor and so on.
- Pencarian dilakukan pada semua simpul dalam setiap level secara berurutan **dari** **kiri ke kanan**.
- Jika pada satu level belum ditemukan solusi, maka pencarian dilanjutkan pada level berikutnya dst… sampai solusi terpenuhi
- Membutuhkan memori besar, sulit diimplementasikan pada dunia nyata.

Breadth-First Search (BFS)

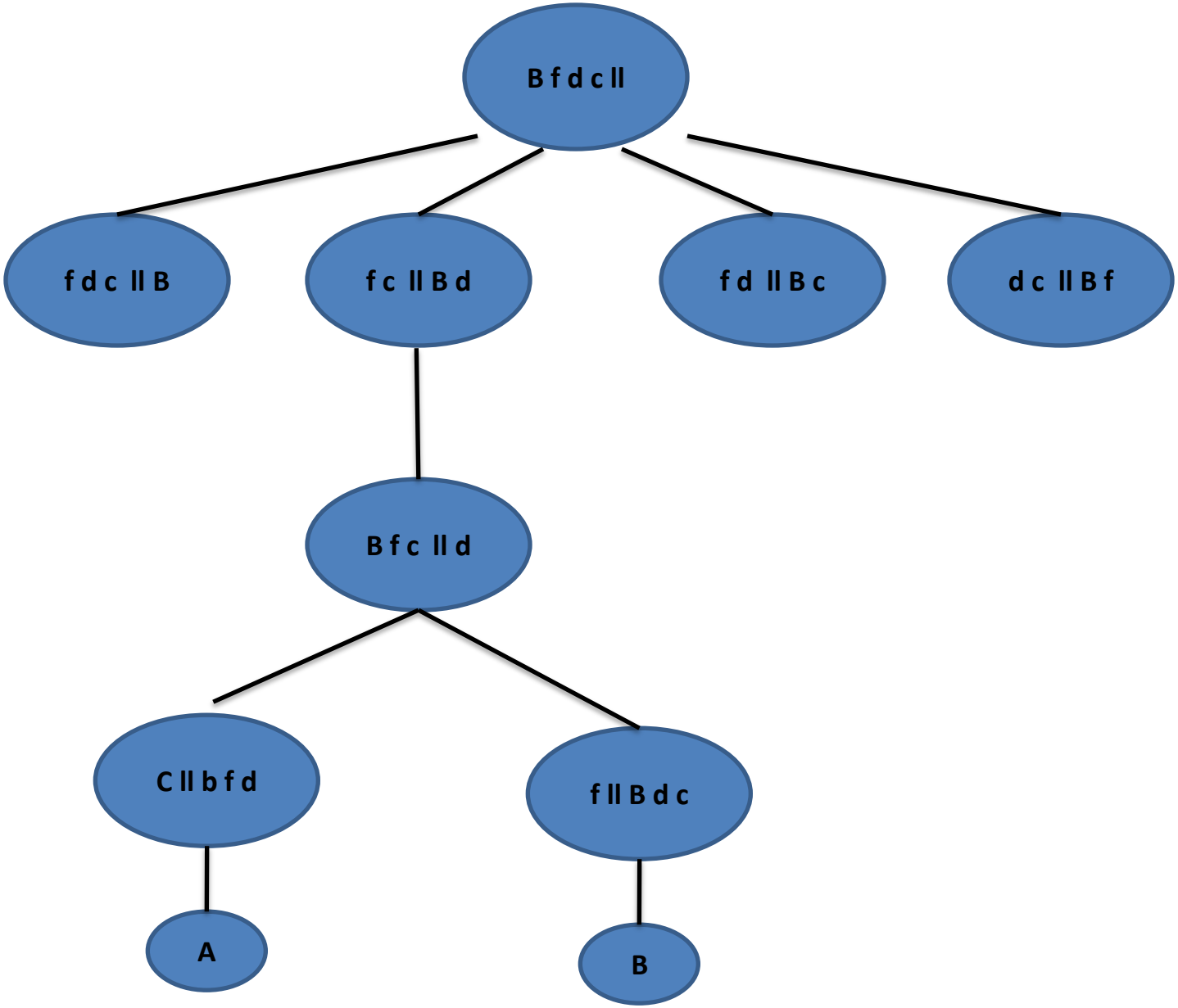


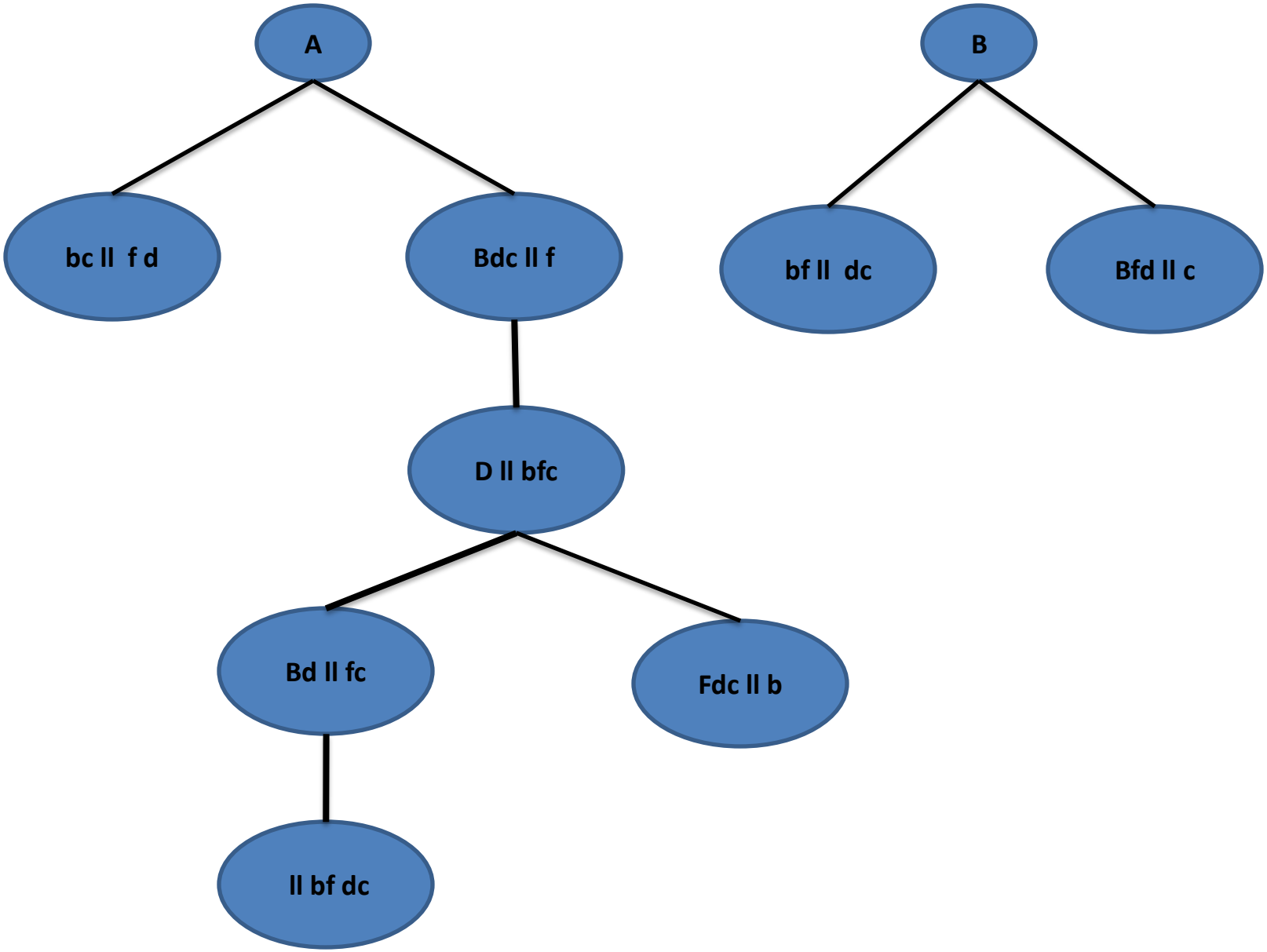
Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

Breadth-First Search (BFS)



Boat fox duck corn | | _ _ _ _

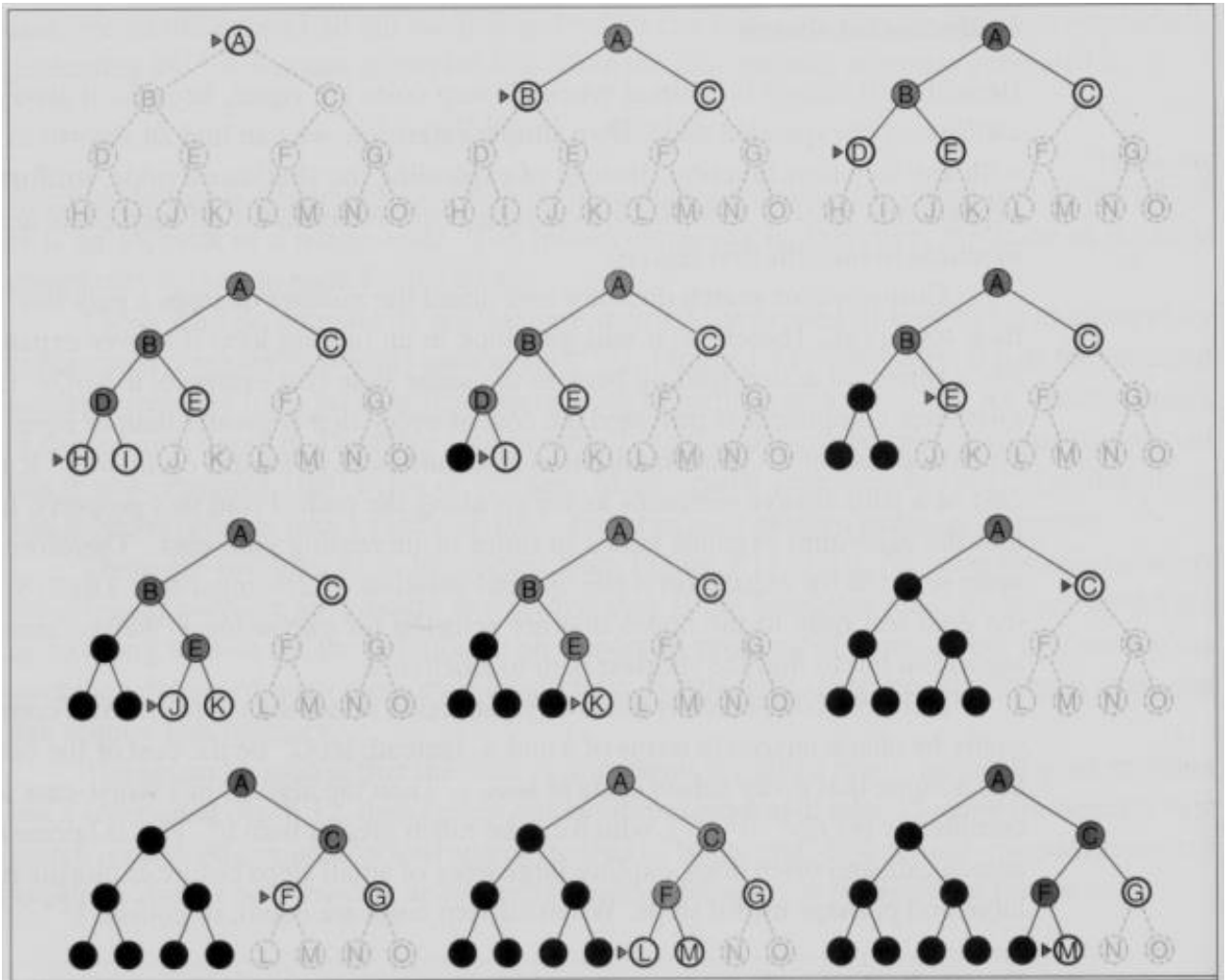

B f d c ll
f d c ll B
f c ll B d
f d ll B c
d c ll B f
B f c ll d
C ll b f d
f ll B d c
A
B


A
B
bc ll f d
Bdc ll f
bf ll dc
Bfd ll c
D ll bfc
Bd ll fc
Fdc ll b
ll bf dc

Depth-First Search (DFS)

- Depth-first search always expands the deepest node in the current fringe of the search tree.
- Pencarian dilakukan pada suatu simpul dalam setiap level dari yang paling kiri.
- Jika pada level yang terdalam solusi belum ditemukan, maka pencarian dilanjutkan pada simpul sebelah kanan dan simpul yang kiri dapat dihaps dari memori. Demikian seterusnya sampai ditemukan solusi.
- Kelebihan →Pemakaian memori lebih sedikit, jika solusi yang dicari berada pada level yang dalm paling kiri, mka DFS akan menemukannya dengan cepat.

Depth-First Search (DFS)



Depth-first search on a binary tree. Nodes that have been expanded and have no descendants in the fringe can be removed from memory; these are shown in black. Nodes at depth 3 are assumed to have no successors and M is the only goal node.

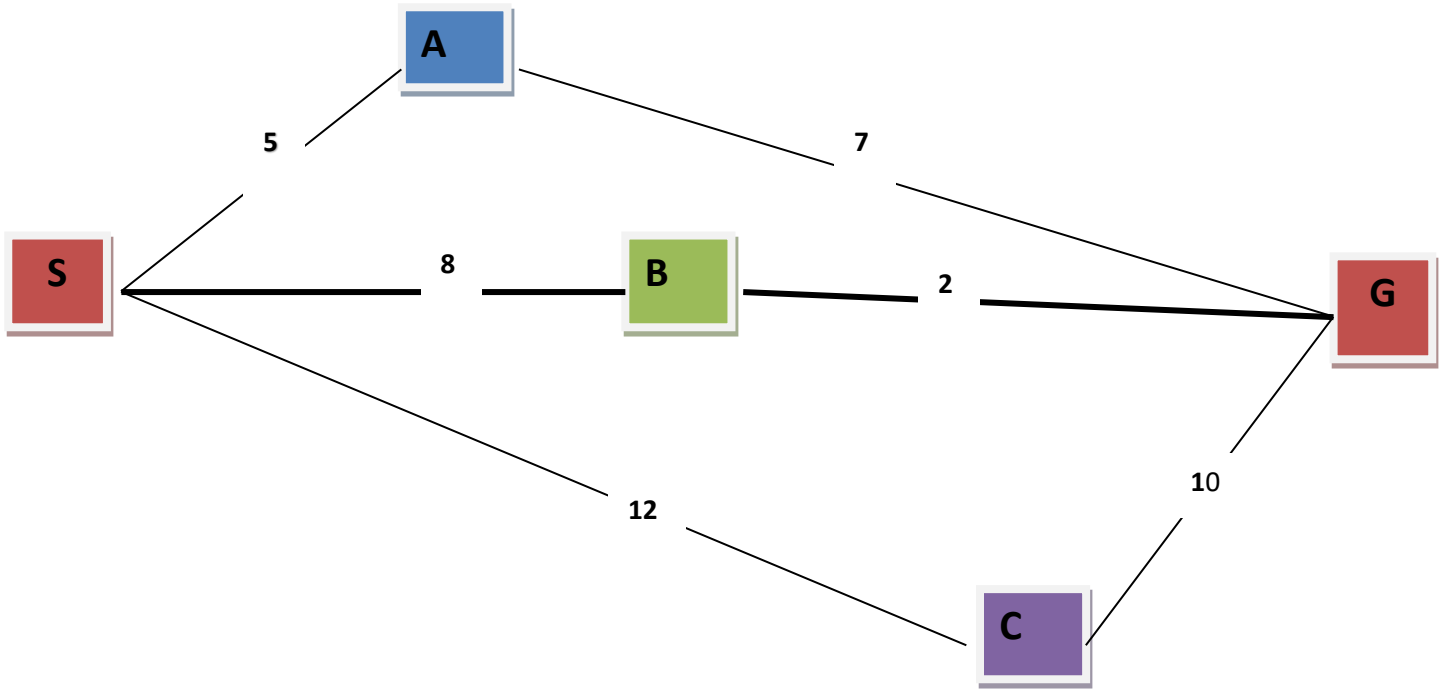
Depth-Limited Search (DLS)

- For the depth limited search, the problem of unbounded tress can be alleviated by supplying depth-first-search with pre-determined depth limit ℓ are treated as if they have no successors.
- Berusaha mengatasi kelemahan DFS (yg mempunyai kelemahan tidak *complete*) dengan membatasi kedalaman maksimum dari suatu jalur solusi. Dengan syarat harus mengetahui berapa level maksimum dari suatu solusi.
- Jika batasan kedalaman terlalu kecil, DLS tidak dapat menemukan solusi yang ada.

Uniform Cost Search (UCS)

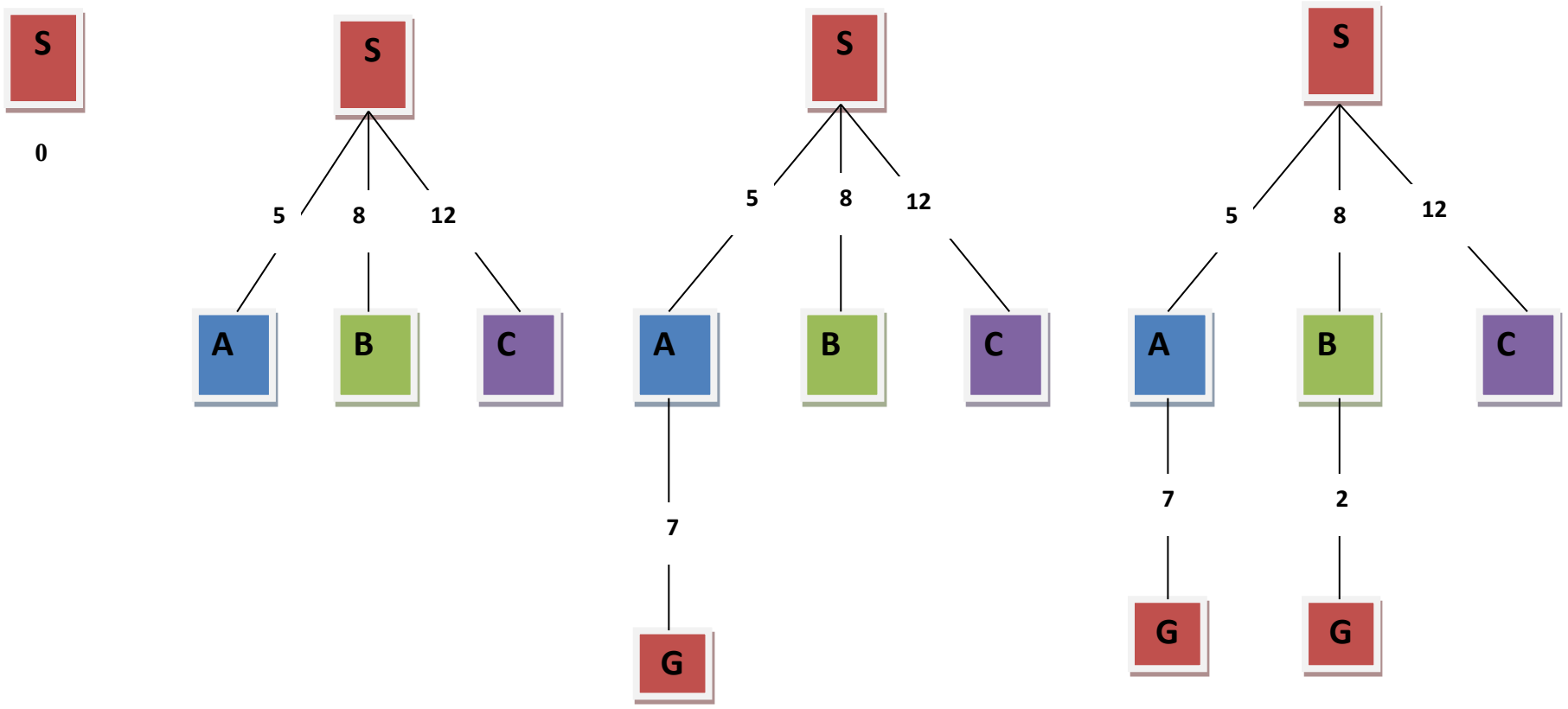
- Hampir sama dengan BFS
- Menggunakan urutan *path cost* dari paling kecil sampai yang terbesar.
- UCS berusaha menemukan solusi dengan total *path cost* yang terendah yang dihitung bedasarkan path cost dari node asal s.d node tujuan.

Uniform cost search (UCS)



Sebuah masalah pencarian rute dari kota S menuju kota G. Pada masalah ini digunakan basis data berupa path cost antara satu kota dengan kota lainnya. (Suyanto, 2007:10)

Uniform Cost Search (UCS)

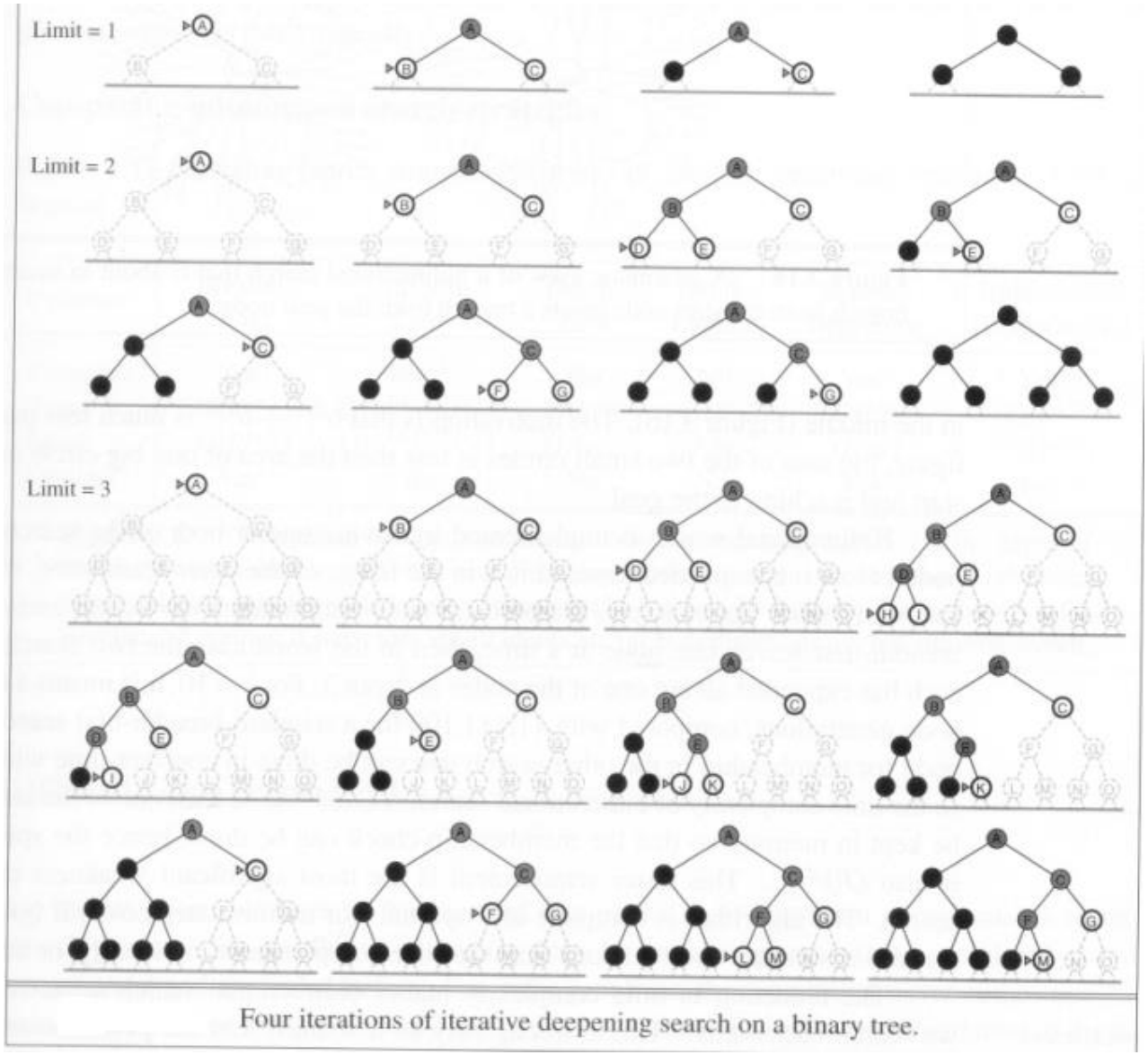


Angka pada setiap simpul menyatakan *path cost* . Pada akhirnya UCS menemukan S-B-G sebagai rute dengan minimum *path cost* 10. (Suyanto, 2007:10)

Iterative Deepening Search (IDS)

- IDS merupakan metode yang menggabungkan kelebihan BFS (*complete* dan *optimal*) dan DFS (*space complexiti* rendah atau membutuhkan sedikit memori).
- IDS melakukan pencarian secara iteratif menggunakan penelusuran DLS dimulai dengan batasan level 0.
- Jika belum ditemukan solusi, maka dilanjutkan pada level 1, dst sampai ditemukan solusi.

Iterative Deepening Search (IDS)



Four iterations of iterative deepening search on a binary tree.

SUMMARY

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.17 Evaluation of search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Metode Pencarian Heuristik

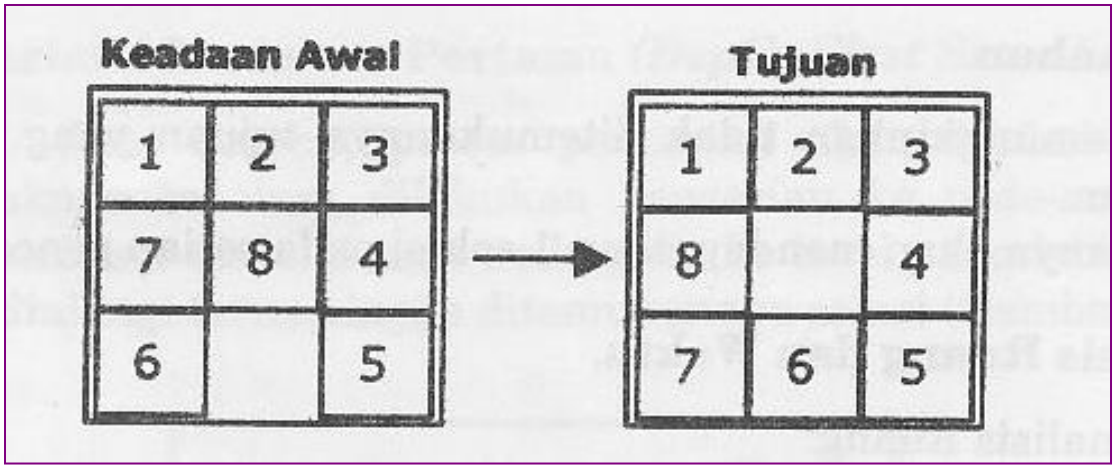
- Pencarian buta tidak selalu dapat diterapkan dengan baik, hal ini disebabkan waktu aksesnya yang cukup lama serta besarnya memori yang dibutuhkan.
- Kelemahan ini sebenarnya dapat diatasi jika ada informasi tambahan (fungsi heuristik) dari domain yang bersangkutan.

Metode Pencarian Heuristik

- Heuristik adalah sebuah teknik yang mengembangkan efisiensi dalam proses pencarian, namun dengan kemungkinan mengorbankan kelengkapan (*completeness*).
- Untuk dapat menerapkan heuristik tersebut dengan baik dalam suatu domain tertentu,

  diperlukan suatu Fungsi Heuristik.
- Fungsi heuristik digunakan untuk menghitung *path cost* suatu node tertentu menuju ke node tujuan.

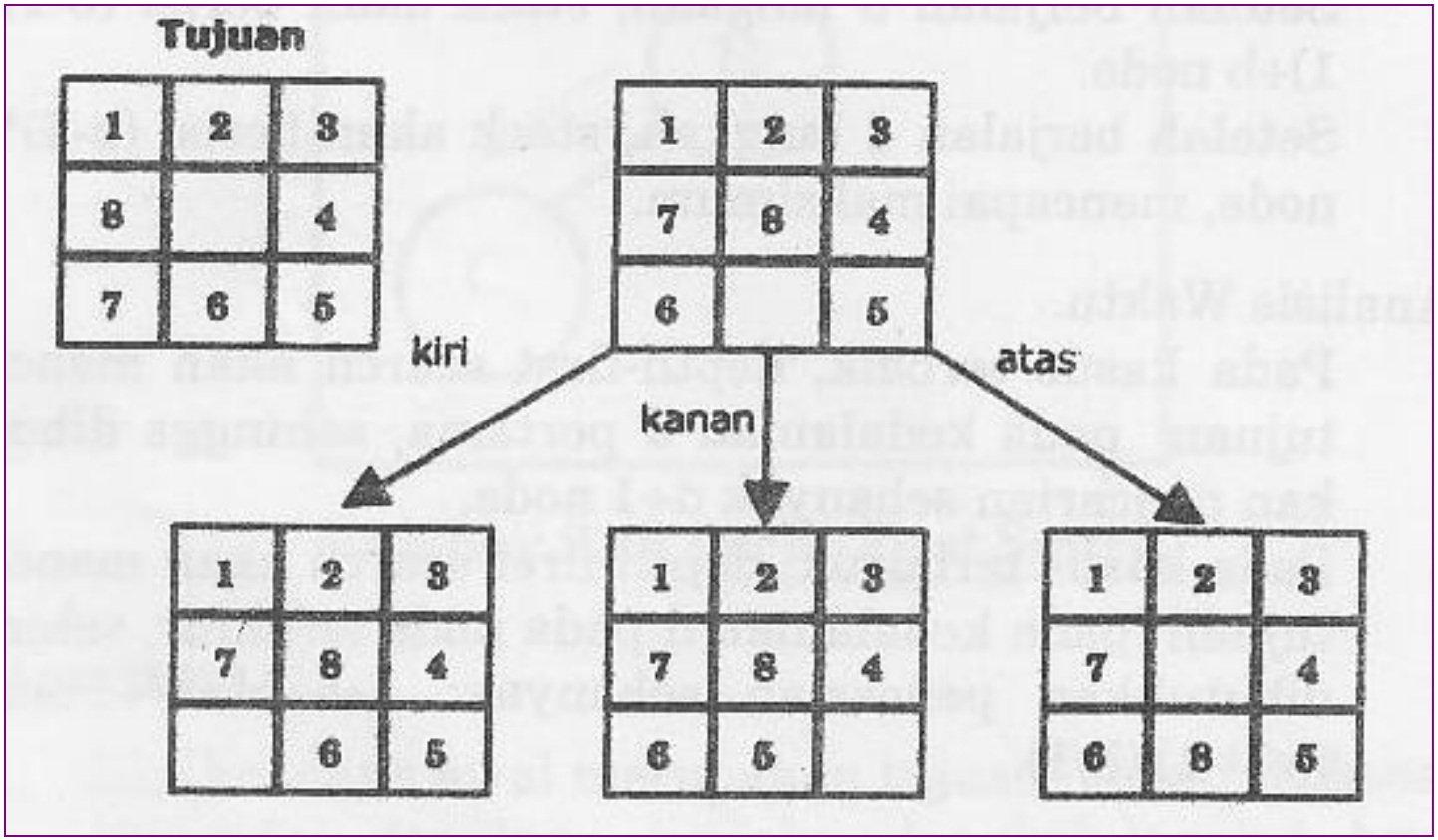
Fungsi Heuristik

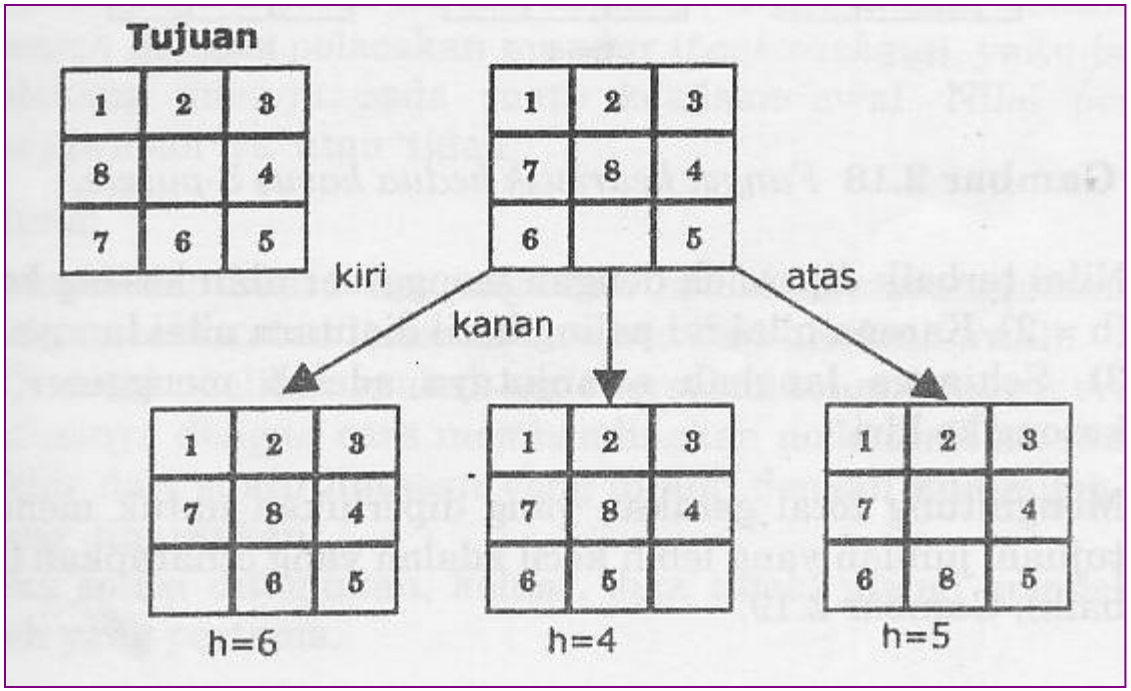


- Kasus 8-puzzle

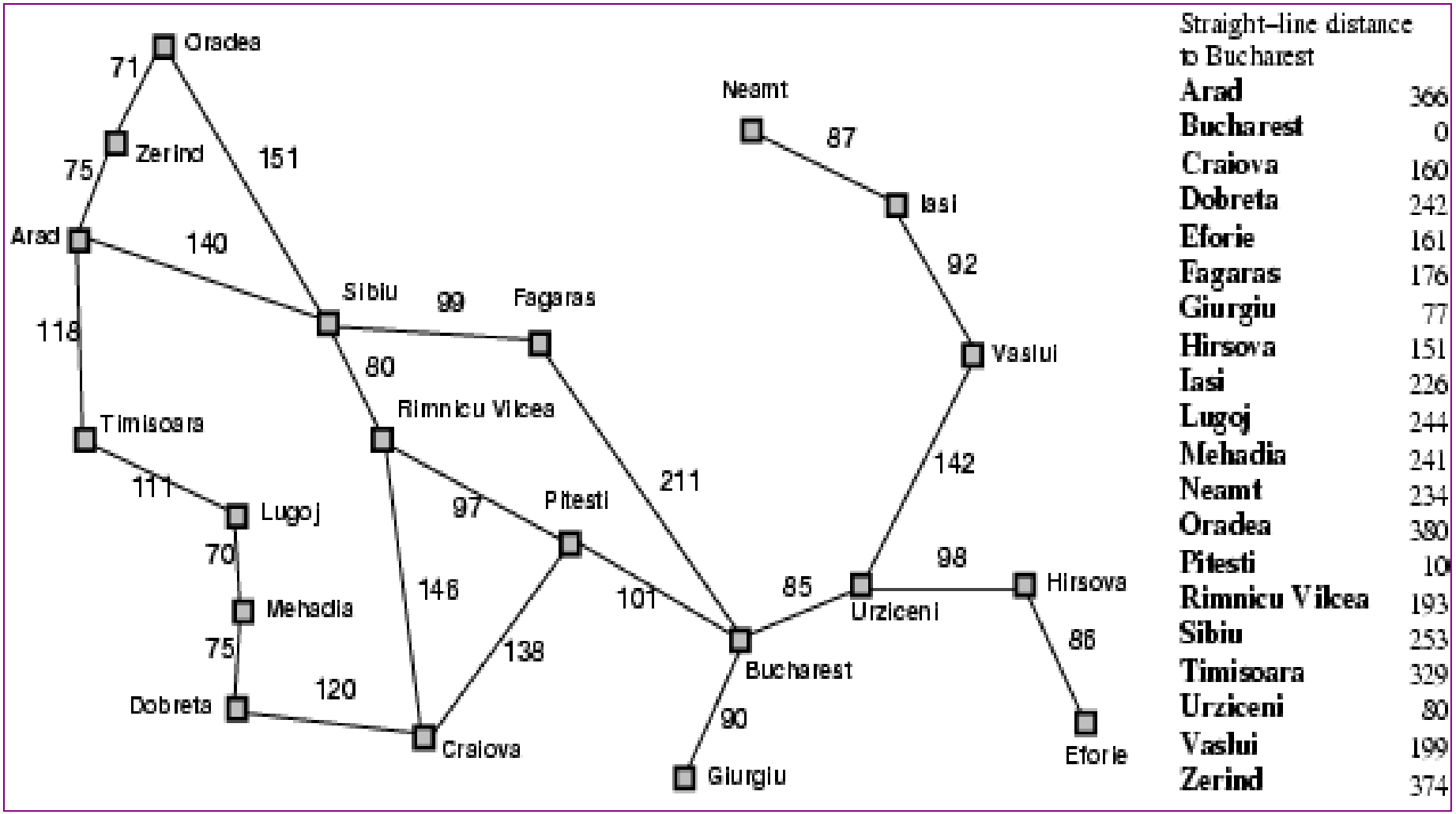
  Ada 4 operator yang dapat digunakan untuk menggerakkan dari satu keadaan (state) ke keadaan yang baru.

  - Geser ubin kosong ke kiri
  - Geser ubin kosong ke kanan
  - Geser ubin kosong ke atas
  - Geser ubin kosong ke bawah


Tujuan
1 2 3
8 4
7 6 5
1 2 3
7 6 4
6 5
kiri
kanan
atas
1 2 3
7 8 4
6 5
1 2 3
7 8 4
6 5
1 2 3
7 4
6 8 5



- Informasi yang diberikan dapat berupa jumlah ubin yang menempati posisi yang benar. Jumlah yang lebih tinggi adalah yang diharapkan.
- Sehingga langkah selanjutnya yang harus dilakukan adalah menggeser ubin kosong ke kiri.



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Informasi yang diberikan berupa *straight-line distance* (jarak dalam garis lurus) antara tiap kota dengan Bucharest.

Metode Pencarian Heuristik

1. *Generate and Test* (Pembangkit dan Pengujian)
2. *Hill Climbing* (Pendakian Bukit)
3. *Best First Search* (Pencarian Terbaik Pertama)
4. *Simulated Annealing*
5. *A**
6. *Djikstra*

Generate and Test

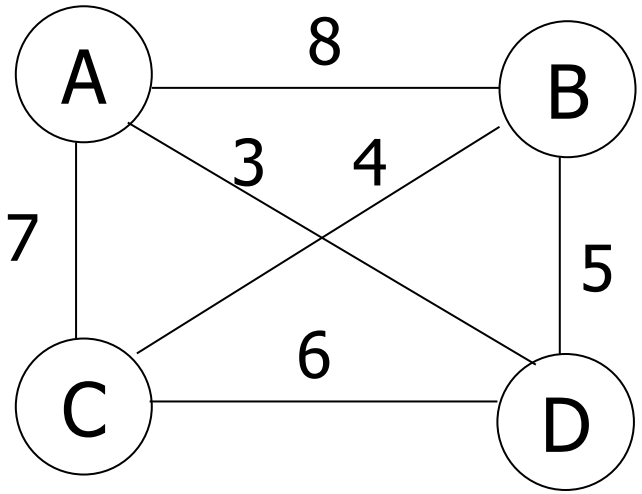
- Metode Generate-and-Test (GT) adalah metode yang paling sederhana dalam teknik pencarian heuristik.
- Di dalam GT, terdapat dua prosedur penting:
  - Pembangkit (generate), yang membangkitkan semua solusi yang mungkin.
  - Test, yang menguji solusi yang dibangkitkan tersebut.
- Algoritma GT menggunakan prosedur Depth First Search karena suatu solusi harus dibangkitkan secara lengkap sebelum dilakukan Test.

Generate and Test

- Jika pembangkitan atau pembuatan solusi–solusi yang dimungkinkan dapat dilakukan secara sistematis, maka prosedur ini akan dapat segera menemukan solusinya, (bila ada).
- Namun, jika ruang problema sangat besar, maka proses ini akan membutuhkan waktu yang lama.
- Metode *generate* and *test* ini kurang efisien untuk masalah yang besar atau kompleks

Generate and Test

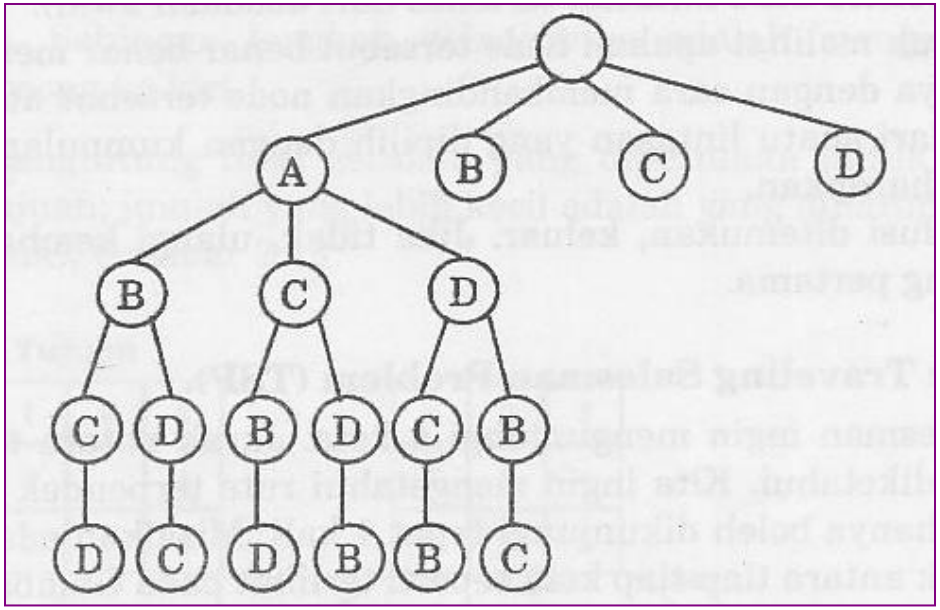
- **Travelling Salesman Problem (TSP)**
- **Seorang salesman ingin mengunjungi sejumlah n kota. Akan dicari rute terpendek di mana setiap kota hanya boleh dikunjungi tepat 1 kali.**
- **Jarak antara tiap-tiap kota sudah diketahui. Misalkan ada 4 kota dengan jarak antara tiap-tiap kota seperti terlihat pada gambar berikut.**



Generate and Test

- Penyelesaian dengan menggunakan Generate-and-Test dilakukan dengan membangkitkan solusi-solusi yang mungkin dengan menyusun kota-kota dalam urutan abjad, yaitu:
  - A-B-C-D
  - A-B-D-C
  - A-C-B-D
  - A-C-D-B
  - dan seterusnya

Generate and Test



- Misalkan kita mulai dari node A. Kita pilih sebagai keadaan awal adalah lintasan ABCD dengan panjang lintasan = 18.
- Kemudian kita lakukan *backtracking* untuk mendapatkan lintasan ABDC dengan panjang lintasan = 19.

Generate and Test

- Lintasan ini kita bandingkan dengan lintasan ABCD, ternyata ABDC > ABCD, sehingga lintasan terpilih adalah ABCD.
- Kita lakukan lagi backtracking untuk mendapatkan lintasan ACBD (=16), ternyata ACBD < ABCD, maka lintasan terpilih sekarang adalah ACBD.
- Demikian seterusnya hingga ditemukan solusi yang sebenarnya.
- Salah satu kelemahan dari metode ini adalah perlunya dibangkitkan semua kemungkinan solusi sehingga membutuhkan waktu yang cukup besar dalam pencariannya.

Generate and Test

ALGORITMA :

- Initial State : Keadaan awal yg diberikan
- Goal State : Rute terpendek
- Buatlah/bangkitkan semua solusi yang memungkinkan. Solusi bisa berupa suatu keadaan (state) tertentu. Solusi juga bisa berupa sebuah jalur dari satu posisi asal ke posisi tujuan, seperti dalam kasus pencarian rute dari satu kota asal ke kota tujuan.

Generate and Test

- Analisa solusi awal dan hitung jarak yang dibutuhkan untuk menyelesaikan solusi tersebut. (dibandingkan dg nilai takberhingga)
- Analisa solusi berikutnya. Jika jarak dari solusi berikutnya lebih pendek dari pada jarak solusi sebelumnya, maka solusi berikutnya adalah solusi terbaik.
- Lakukan sampai seluruh solusi selesai dianalisa

HILL CLIMBING

- Hampir sama *Generate* and *Test*, perbedaan terjadi pada *feedback* dari prosedur test untuk pembangkitan keadaan berikutnya.

- Tes yang berupa fungsi heuristik akan menunjukkan seberapa baik nilai terkaan yang diambil terhadap keadaan lain yang mungkin

Metode simple hill climbing

- Ruang keadaan berisi semua kemungkinan lintasan yang mungkin. Operator digunakan untuk menukar posisi kota-kota yang bersebelahan. Fungsi heuristik yang digunakan adalah panjang lintasan yang terjadi.
- Operator yang akan digunakan adalah menukar urutan posisi 2 kota dalam 1 lintasan. Bila ada n kota, dan ingin mencari kombinasi lintasan dengan menukar posisi urutan 2 kota, maka akan didapat sebanyak :

$$\frac{n!}{2!(n-2)!} = \frac{4!}{2!(4-2)!} = 6 \text{ kombinasi}$$

Metode simple hill climbing

Keenam kombinasi ini akan dipakai semuanya sebagai operator, yaitu :

Tukar 1,2 = menukar urutan posisi kota ke – 1 dengan kota ke – 2

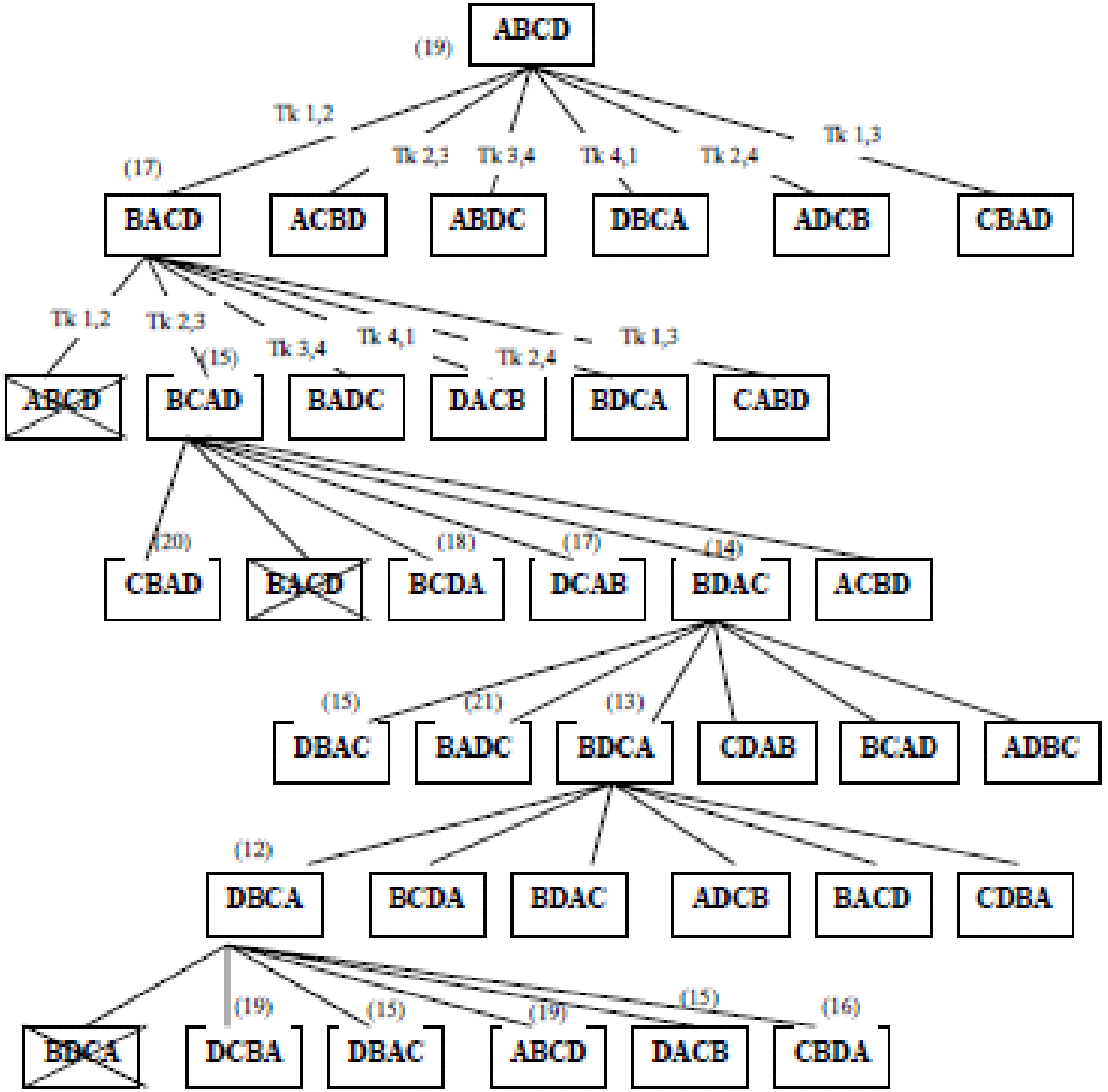
Tukar 2,3 = menukar urutan posisi kota ke – 2 dengan kota ke – 3

Tukar 3,4 = menukar urutan posisi kota ke – 3 dengan kota ke – 4

Tukar 4,1 = menukar urutan posisi kota ke – 4 dengan kota ke – 1

Tukar 2,4 = menukar urutan posisi kota ke – 2 dengan kota ke – 4

Tukar 1,3 = menukar urutan posisi kota ke – 1 dengan kota ke – 3


(19)
ABCD
Tk 1,2
Tk 2,3
Tk 3,4
Tk 4,1
Tk 2,4
Tk 1,3
(17)
BACD
ACBD
ABDC
DBCA
ADCB
CBAD
Tk 1,2
Tk 2,3
Tk 3,4
Tk 4,1
Tk 2,4
Tk 1,3
(15)
ABCD
BCAD
BADC
DACB
BDCA
CABD
(20)
(18)
(17)
(14)
CBAD
BACD
BCDA
DCAB
BDAC
ACBD
(15)
(21)
(13)
DBAC
BADC
BDCA
CDAB
BCAD
ADBC
(12)
DBCA
BCDA
BDAC
ADCB
BACD
CDBA
(19)
(15)
(19)
(15)
(16)
BDCA
DCBA
DBAC
ABCD
DACB
CBDA

- Keadaan awal, lintasan ABCD (=19).
- Level pertama, hill climbing mengunjungi BACD (=17), BACD (=17) < ABCD (=19), sehingga
  - BACD menjadi pilihan selanjutnya dengan operator Tukar 1,2
- Level kedua, mengunjungi ABCD, karena operator Tukar 1,2 sudah dipakai BACD, maka pilih node
  - lain yaitu BCAD (=15), BCAD (=15) < BACD (=17)
- Level ketiga, mengunjungi CBAD (=20), CBAD (=20) > BCAD (=15), maka pilih node lain yaitu
    - BCDA (=18), pilih node lain yaitu DCAB (=17), pilih node lain yaitu BDAC (=14), BDAC (=14) < BCAD (=15)
- Level keempat, mengunjungi DBAC (=15), DBAC(=15) > BDAC (=14), maka pilih node lain yaitu
  - BADC (=21), pilih node lain yaitu BDCA (=13), BDCA (=13) < BDAC (=14)
- Level kelima, mengunjungi DBCA (=12), DBCA (=12) < BDCA (=13)
- Level keenam, mengunjungi BDCA, karena operator Tukar 1,2 sudah dipakai DBCA, maka pilih node
  - lain yaitu DCBA, pilih DBAC, pilih ABCD, pilih DACB, pilih CBDA
- Karena sudah tidak ada node yang memiliki nilai heuristik yang lebih kecil dibanding nilai heuristik DBCA, maka **node DBCA** (=12) adalah **lintasan terpendek (SOLUSI)**

Metode simple hill climbing

- Algoritma:
  1. Initial State
  2. Goal State
  3. Evaluasi keadaan awal, jika tujuan berhenti, jika tidak lanjut dengan keadaan sekarang sebagai keadaan awal
  4. Kerjakan langkah berikut sampai solusi ditemukan atau tidak ada lagi operator baru sebagai keadaan sekarang :

Metode simple hill climbing

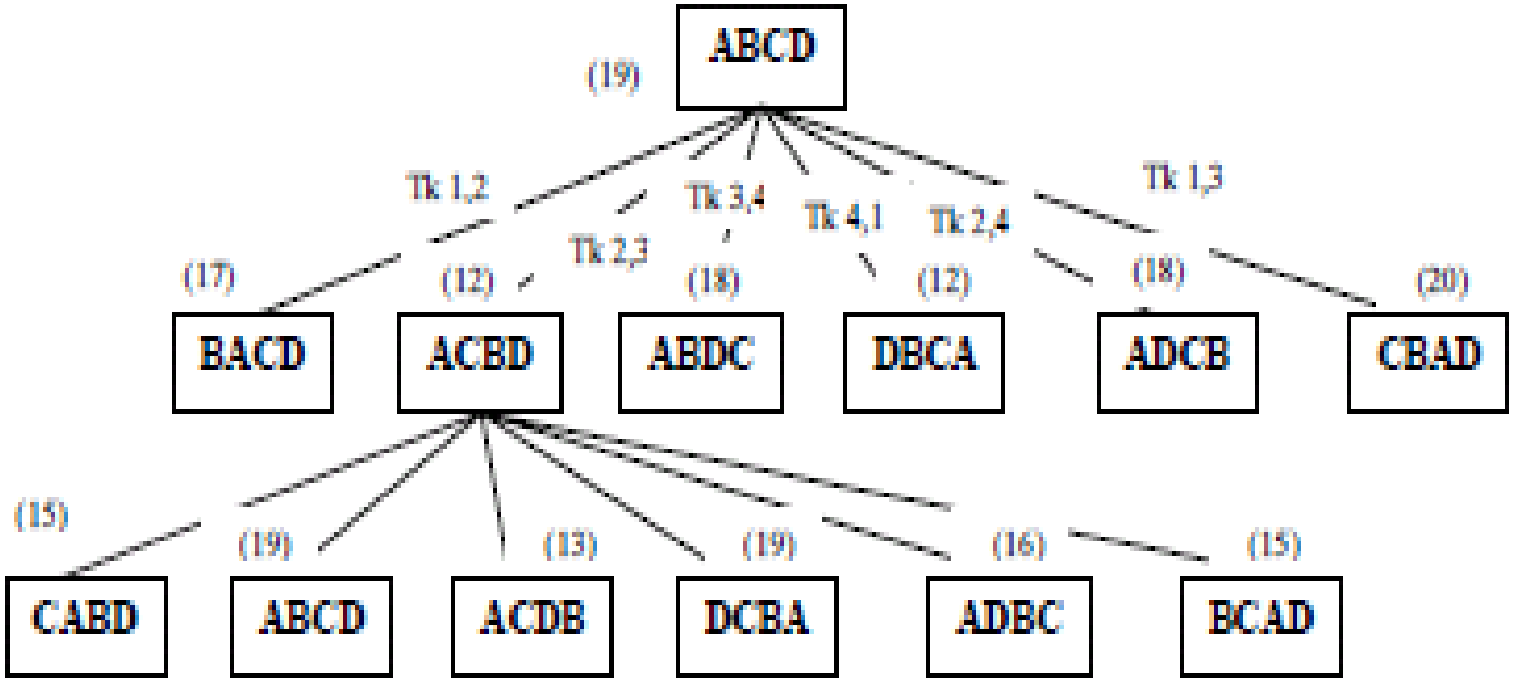
i. Cari operator yang belum pernah digunakan. Gunakan operator untuk keadaan yang baru.

ii. Evaluasi keadaan sekarang:

   a) Jika keadaan tujuan , keluar.
   b) Jika bukan tujuan, namun nilainya lebih baik dari sekarang, maka jadikan keadaan tersebut sebagai keadaaan sekarang
   c) Jika keadaan baru tidak lebih baik daripada keadaan sekarang, maka llanjutkan iterasi.

Metode steepest – ascent hill climbing

- Steepest – ascent hill climbing hampir sama dengan simple – ascent hill climbing, hanya saja gerakan pencarian tidak dimulai dari kiri, tetapi berdasarkan nilai heuristik terbaik.
- Gerakan pencarian selanjutnya berdasar nilai heuristik terbaik



- Keadaan awal, lintasan ABCD (=19).
- Level pertama, hill climbing memilih nilai heuristik terbaik yaitu ACBD (=12) sehingga ACBD
- menjadi pilihan selanjutnya.
- Level kedua, hill climbing memilih nilai heuristik terbaik, karena nilai heuristik lebih besar dibanding ACBD, maka hasil yang diperoleh lintasannya tetap ACBD (=12)

Metode steepest – ascent hill climbing

- Algoritma:
  1) Initial State
  2) Goal State
  3) Evaluasi keadaan awal, jika tujuan berhenti jika tidak lanjut dengan keadaan sekarang sebagai keadaan awal
  4) Kerjakan hingga tujuan tercapai atau hingga iterasi tidak memberi perubahan sekarang :

Metode steepest – ascent hill climbing

i. Tentukan SUCC sebagai nilai heuristik terbaik dari successor-successor
ii. Kerjakan tiap operator yang digunakan oleh keadaan sekarang.

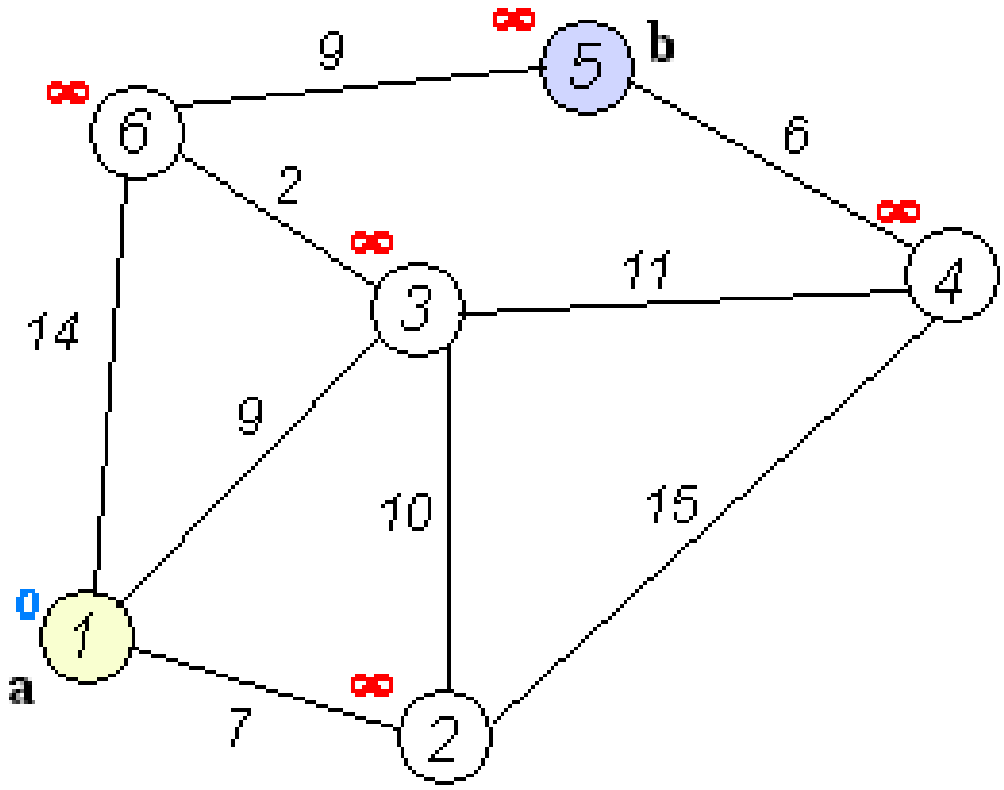
    a. Gunakan operator tersebut dan bentuk keadan baru
    b. Evaluasi keadaan baru. Jika tujuan keluar, jika bukan bandingkan nilai heuristiknya dengan SUCC. Jika lebih baik jadikan nilai heuristik keadaan baru ter sebut sebagai SUCC. Jika tidak, nilai SUCC tidak berubah.

iii. Jika SUCC lebih baik dari nilai heuristik keadaan sekarang, ubah SUCC menjadi keadaan sekarang.

DIJKSTRA

- Dijkstra's algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1959,[1] is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree. This algorithm is often used in routing. An equivalent algorithm was developed by Edward F. Moore in 1957.[2]

DIJKSTRA



DIJKSTRA

Yang harus diketahui pada metode Dijkstra:

- Nilai jarak : Nilai jarak dari satu node dengan node yang lain
- Jarak : Jarak antara satu node dengan node yang lain
- Nilai : Nilai bisa berupa nama, angka pada node dan lain sebagainya.

DIJKSTRA

Algoritma:

1. Beri nilai tiap node dengan nilai.
2. Beri informasi jarak antar node
3. Beri nilai jarak 0 untuk initial node dan ∞ untuk node yang lain.
4. Tandai semua node sebagai *unvisited.* Set initial node sebagai current (sekarang).
5. Untuk current node, kunjungi node tetanga dan hitung jarak (dari initial node). Contoh : Jika current node A nilai jarak = 6, dan node tetangga (B) jaraknya 2. Jarak B lewat A adalah 6 + 2 = 8. Jika jarak ini kurang dari jarak sebelumnya yang disimpan (∞), ganti jaraknya.

DIJKSTRA

4. Jika sudah dihitung jaraknya, tandai sebagai visited. Node yang sudah divisited tidak perlu dicek lagi.
5. Set node unvisited dengan jarak terpendek (dengan initial node) sebagai current node dan ulangi step 3.