AI sebagai Masalah Pelacakan

Lesson 2

Teknik Pencarian

Pendahuluan

- Setelah permasalahan direpresentasikan dalam bentuk state-space, maka selanjutnya dilakukan pencarian (searching) di dalam state-space tersebut untuk menentukan solusi permasalahan
- Hal penting dalam menentukan keberhasilan sistem berdasar kecerdasan adalah kesuksesan dalam pencarian dan pencocokan.

Teknik Pencarian/Pelacakan

PELACAKAN

- **Trial & Error**
- **Uninformed/Blind Search**
  - Depth-first search (Vertical)
  - Breadth-first search (Horizontal)
- **Informed Search/Heuristic Search**
  - Hill Climbing
  - Best-First Search
  - Algoritma A
  - Algoritma A*
- **MINIMAX**
- **.....**

Teknik Pencarian/Pelacakan

- Teknik pencarian dibedakan satu sama lain khususnya dalam hal urutan ekspansi node
- Teknik pencarian dievaluasi dalam beberapa aspek, di antaranya:
  - Completeness

    apakah selalu berhasil menemukan solusi?
  - Time complexity

    berapa lama waktu yang diperlukan untuk menemukan solusi
  - Space complexity

    berapa banyak space memory yang diperlukan untuk mencari solusi?
  - Optimality

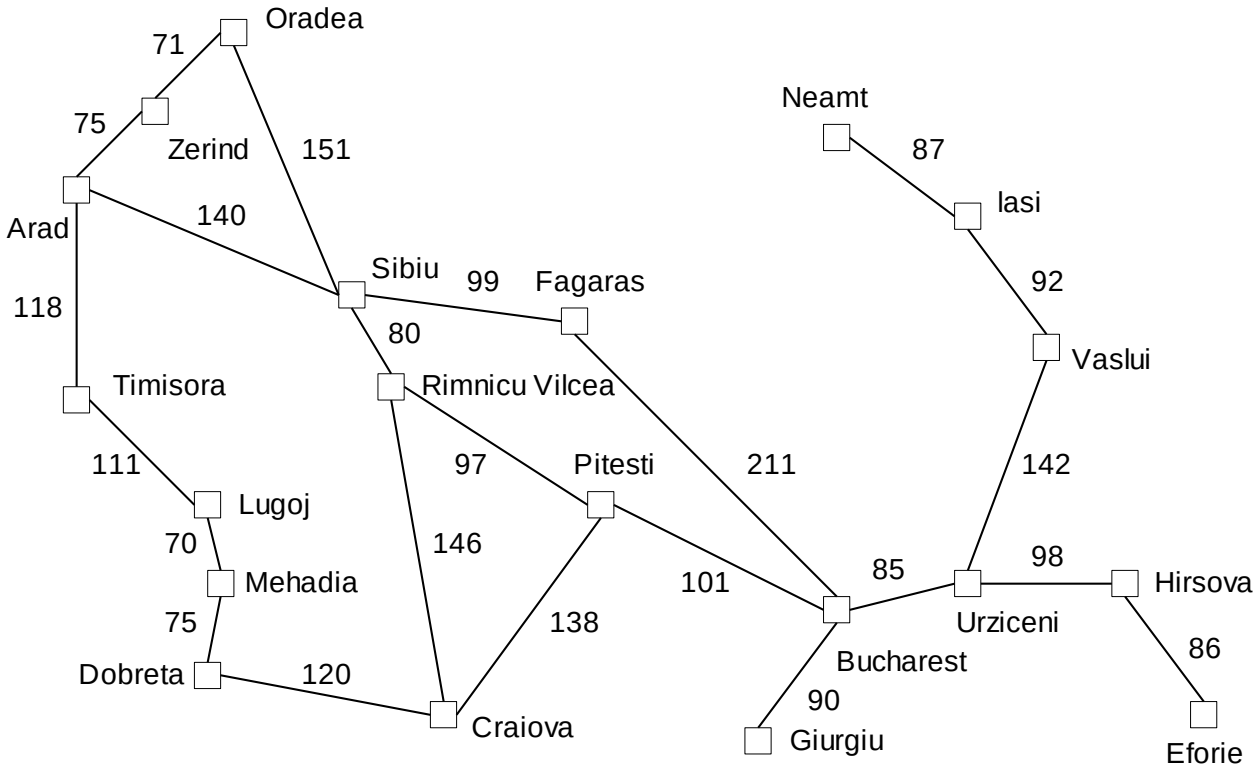
    apakah teknik tersebut dapan menemukan solusi yang paling optimal jika ada beberapa alternatif solusi?

Teknik Pencarian/Pelacakan

- Time dan space complexity diukur menggunakan parameter2:
  - b (branching factor), faktor percabangan maksimum dari search tree
  - d (depth), kedalaman (level tree) dari solusi yang diperoleh

Finding Route

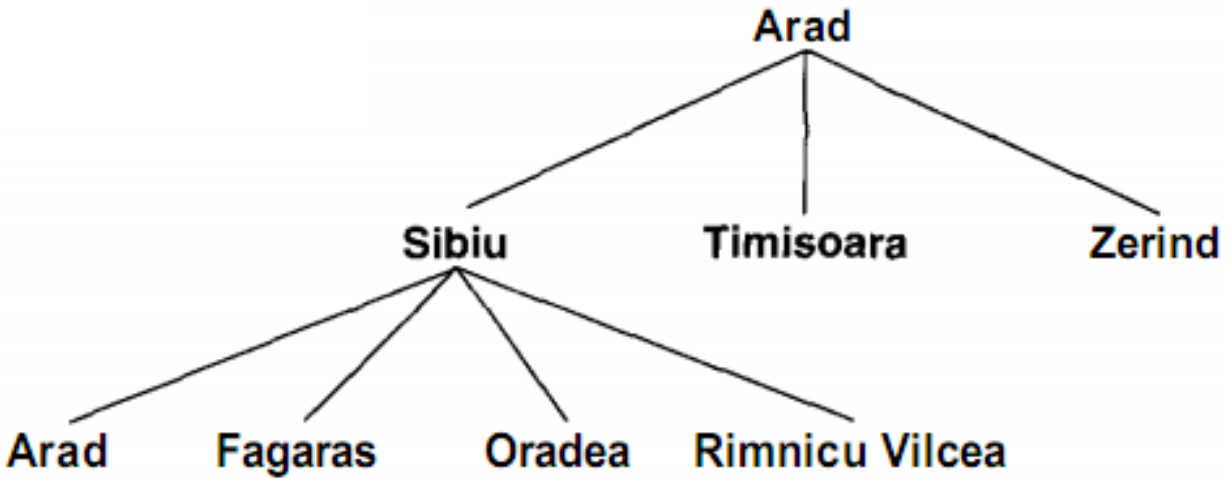
- Perhatikan peta di bawah ini



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Finding Route

- Jika akan dilakukan perjalanan dari Arad ke Bucharest, maka akan diperoleh sebagian search-tree sebagai berikut



Finding Route

- Bagaimana rute dari Arad ke Bucharest?

Trial and Error

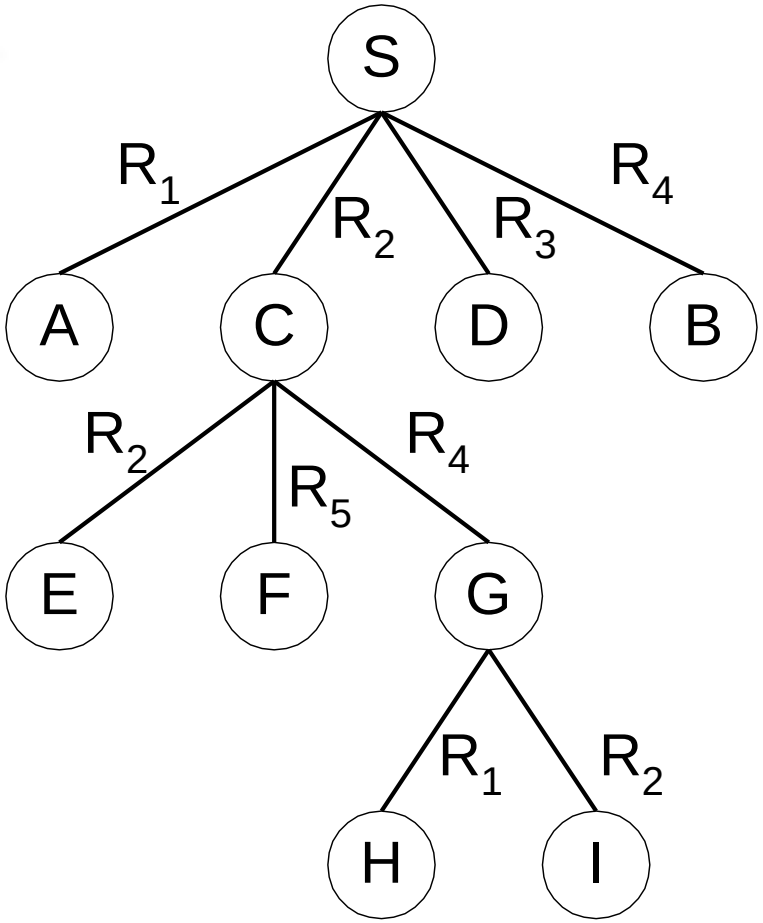
- Metode paling sederhana
- Prosedur pelacakan

1. Tentukan state sebagai keadaan awal
2. While state ≠ keadaan sasaran do
3. Begin
4. Pilih operator yang dapat diterapkan pada state, dan diset sebagai operator
5. State : = operator (state)
6. End

Trial and Error

- Catatan
  - Pada langkah 4, operator dipilih secara acak
  - Pada langkah 5, operator yang dipilih diterapkan pada state membentuk state baru
  - Stokastik, tidak menjamin dicapainya keadaan sasaran (goal)
  - Tidak memperlihatkan karakteristik ‘intelegensi’

Trial and Error



Uninformed x Informed Search

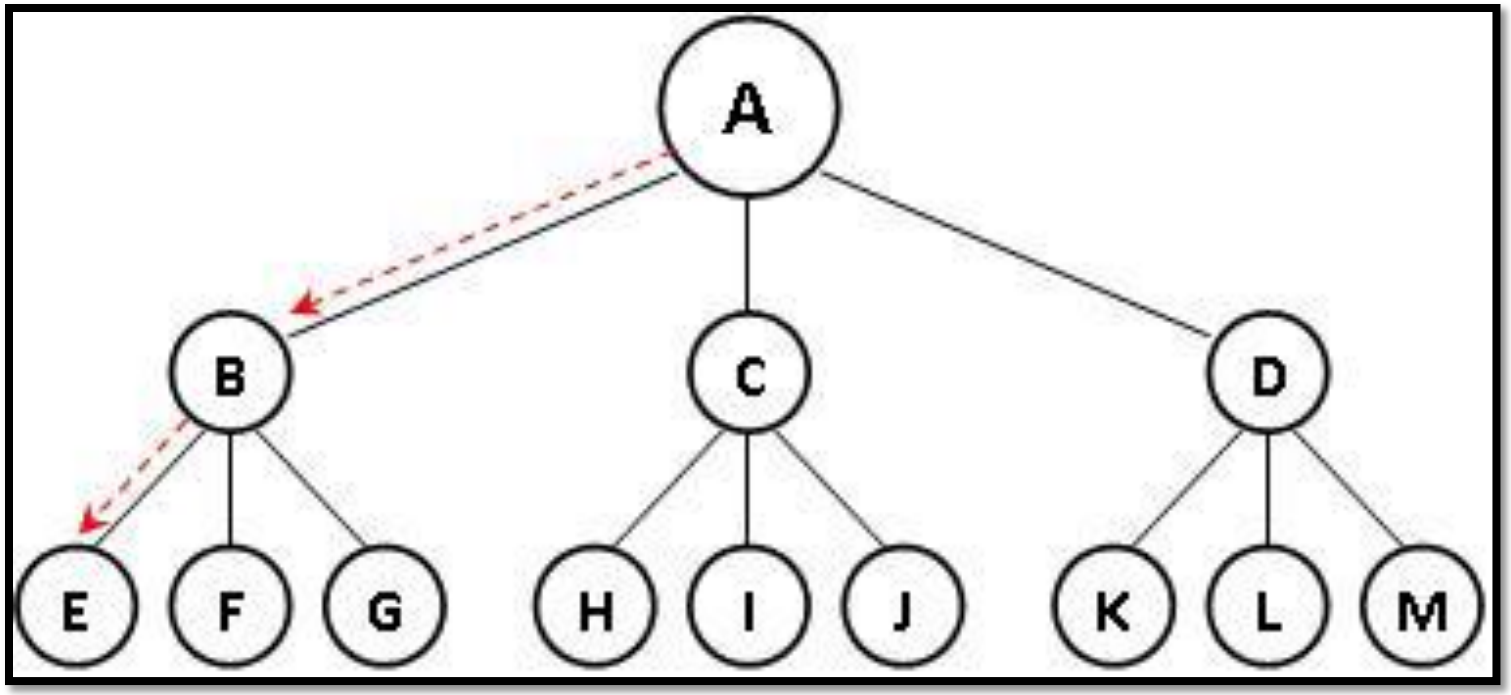
- Pada kasus rute Arad-Bucharest, ada 3 alternatif kota yang dapat dipilih dari Arad, yaitu: Sibiu, Timisoara, Zerind
- Uninformed Search tidak punya preferensi sama sekali di antara 3 pilihan kota tersebut, hanya memilih berdasarkan urutan di dalam teknik pencariannya
- Informed Search akan punya preferensi di antara pilihan tersebut berdasarkan parameter tertentu, misal: jarak

Teknik Sistematis, Blind Search, Uninformed Search

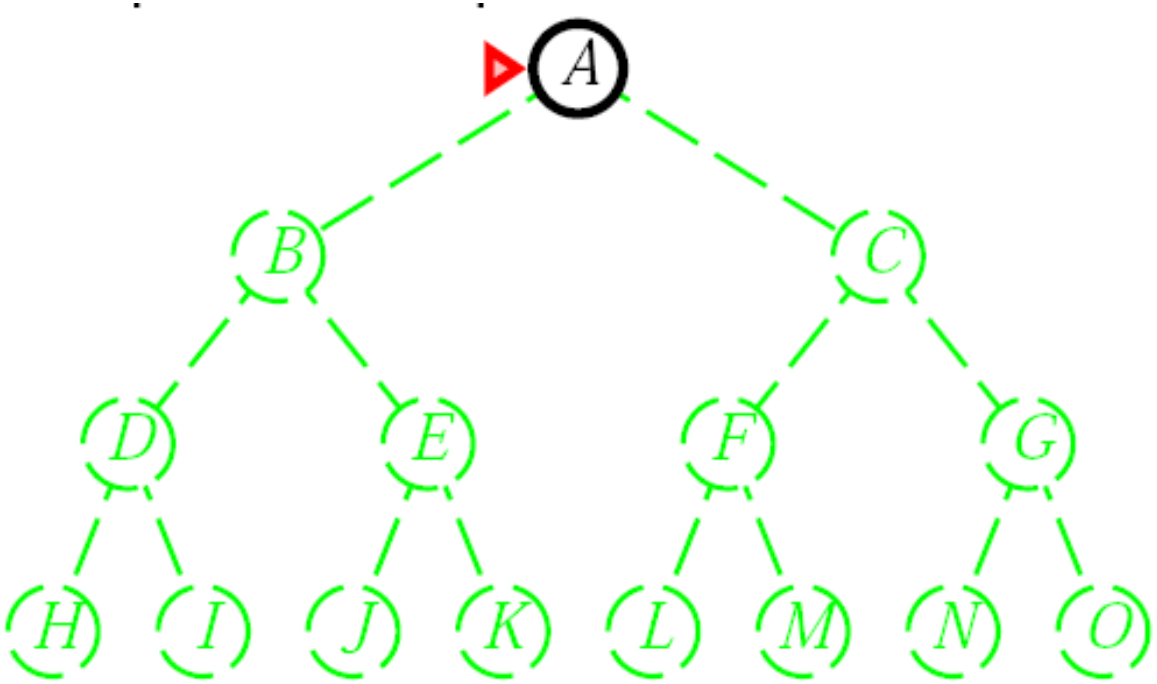
Depth-First Search

- Representasi: Diagram pohon atau grafik
- Pencarian dilakukan dari Simpul Akar ke simpul yang memiliki level lebih tinggi (Simpul Anak).
- Proses pencarian dilakukan pada semua Simpul Anak sebelum dilakukan pencarian ke simpul-simpul yang selevel.

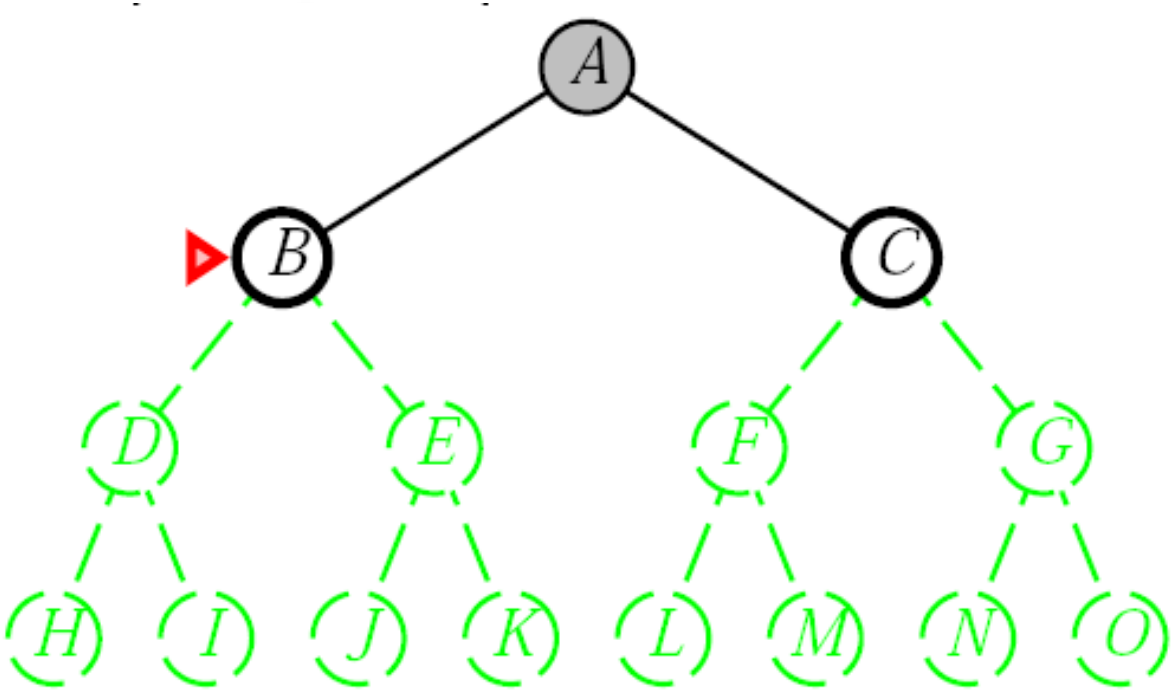
Depth-First Search



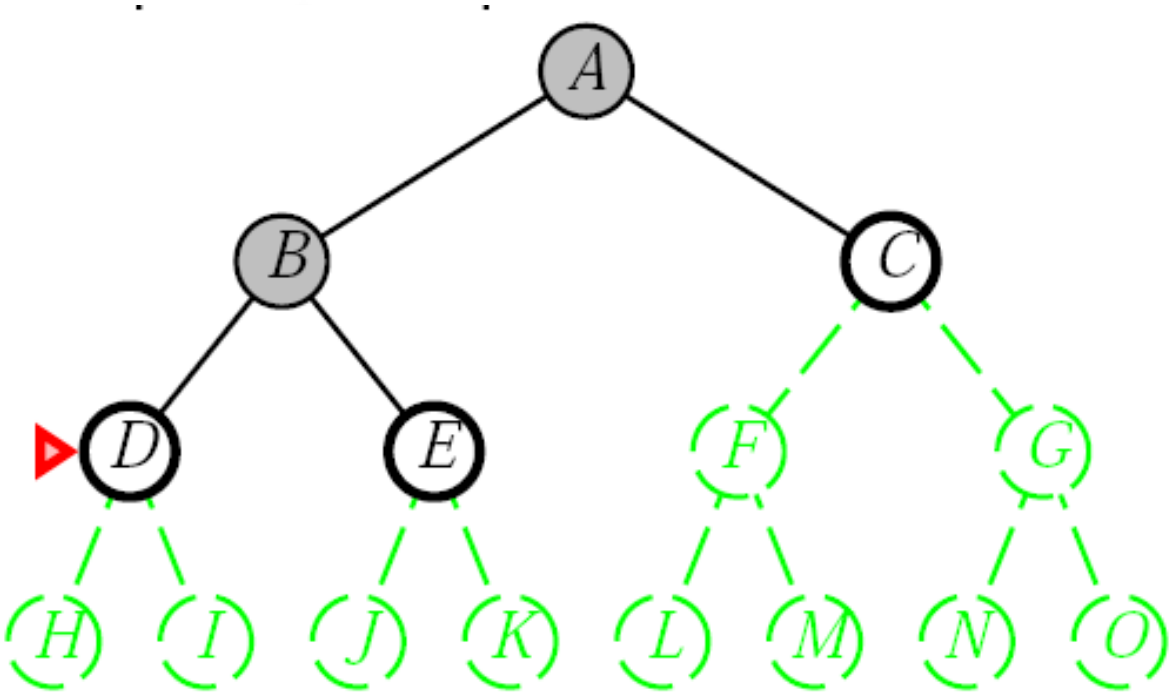
Depth-First Search



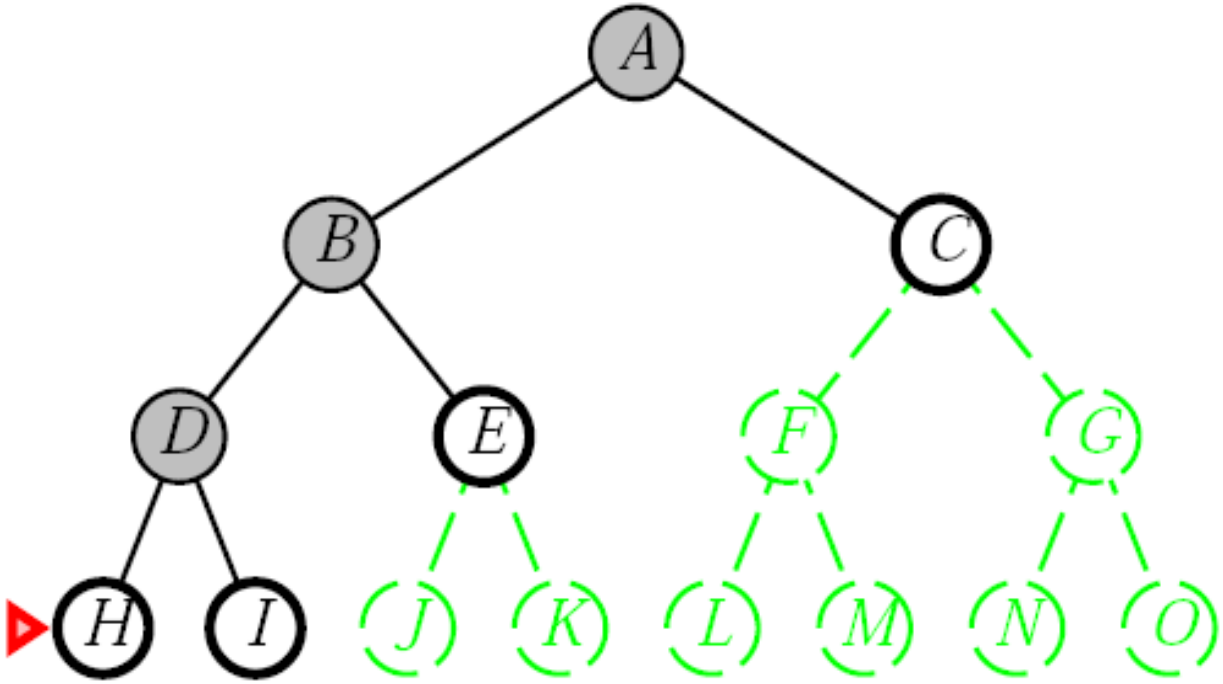
Depth-First Search



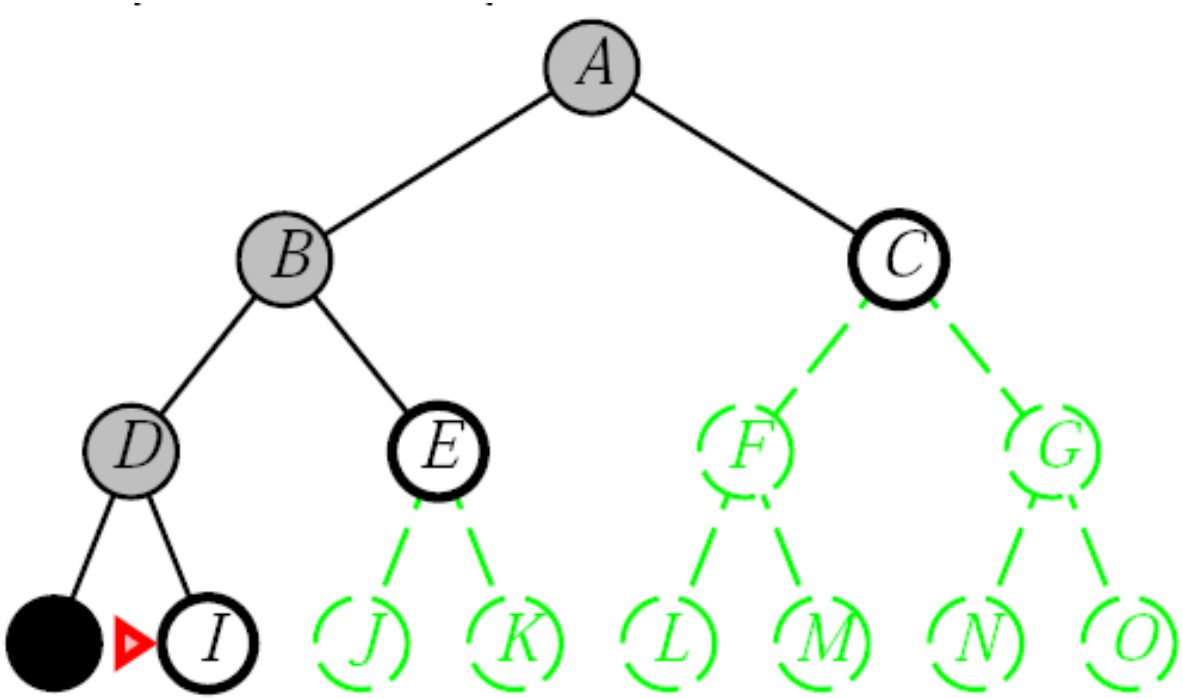
Depth-First Search



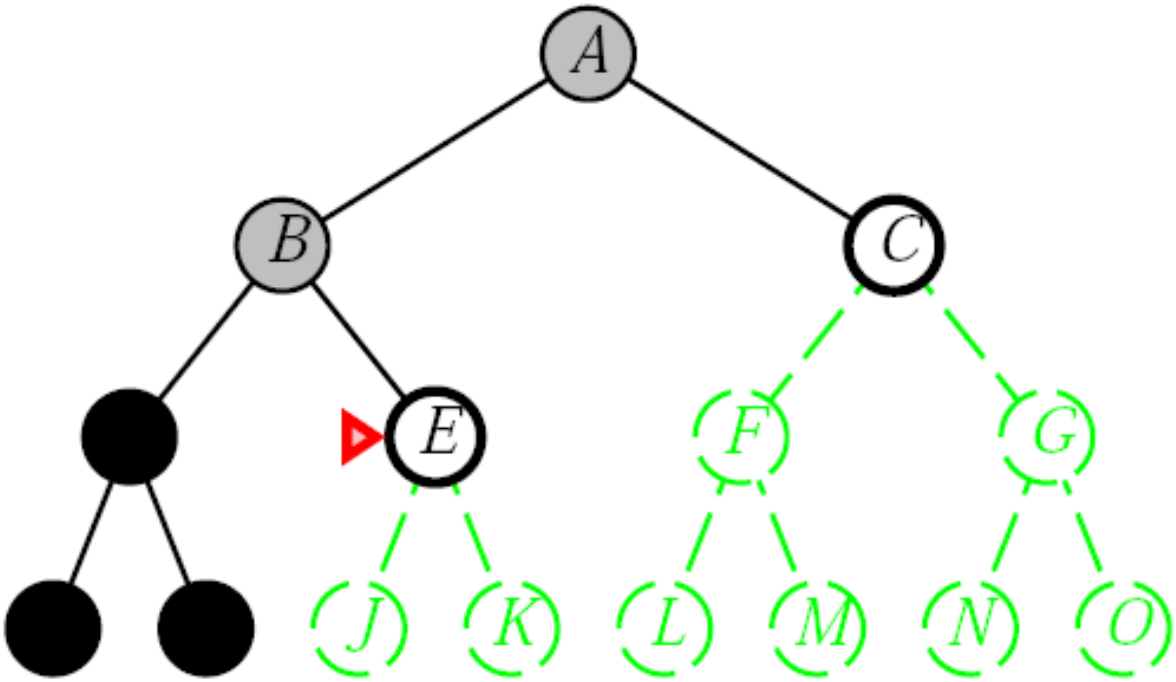
Depth-First Search



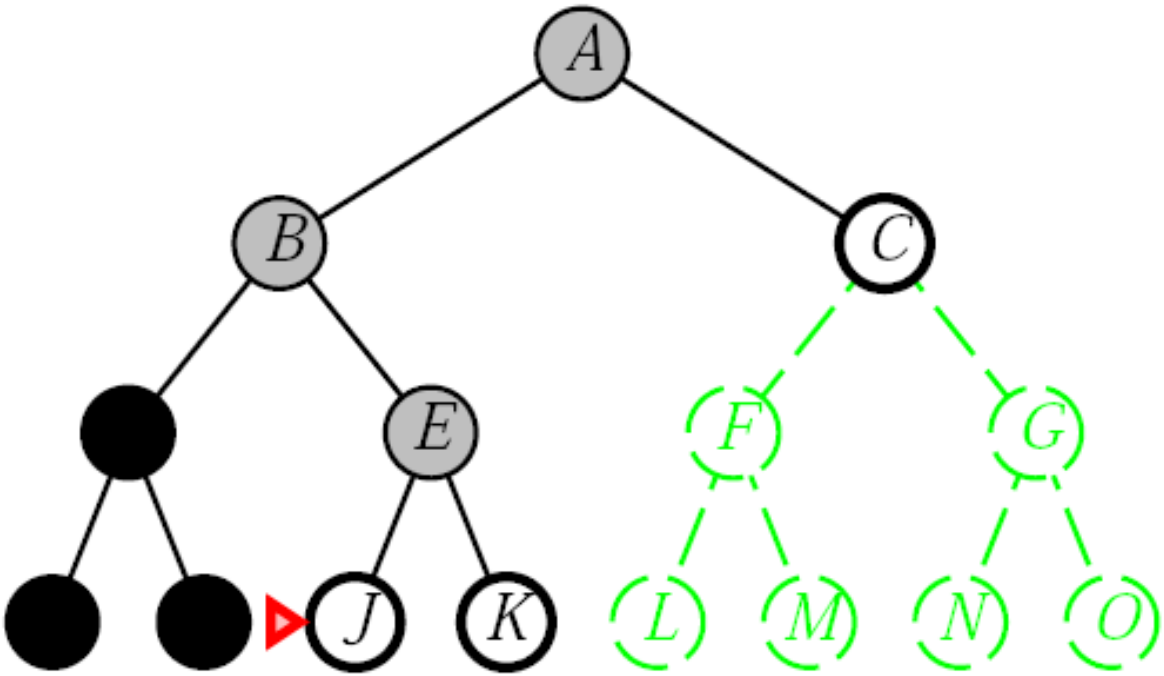
Depth-First Search



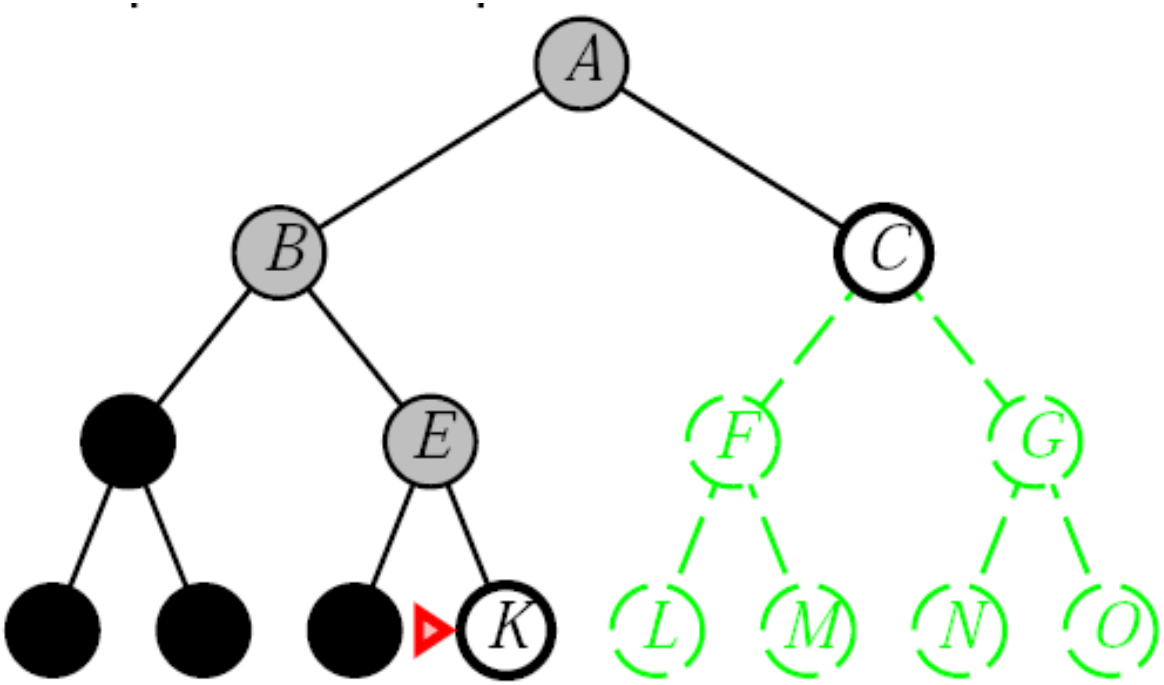
Depth-First Search



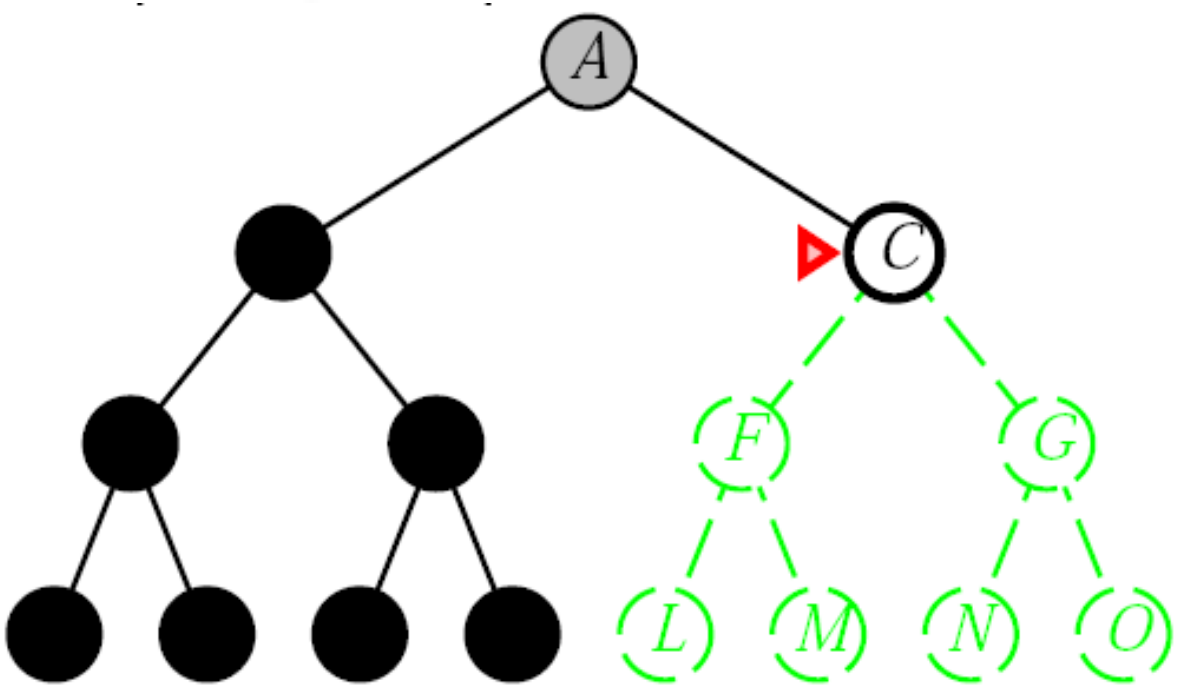
Depth-First Search



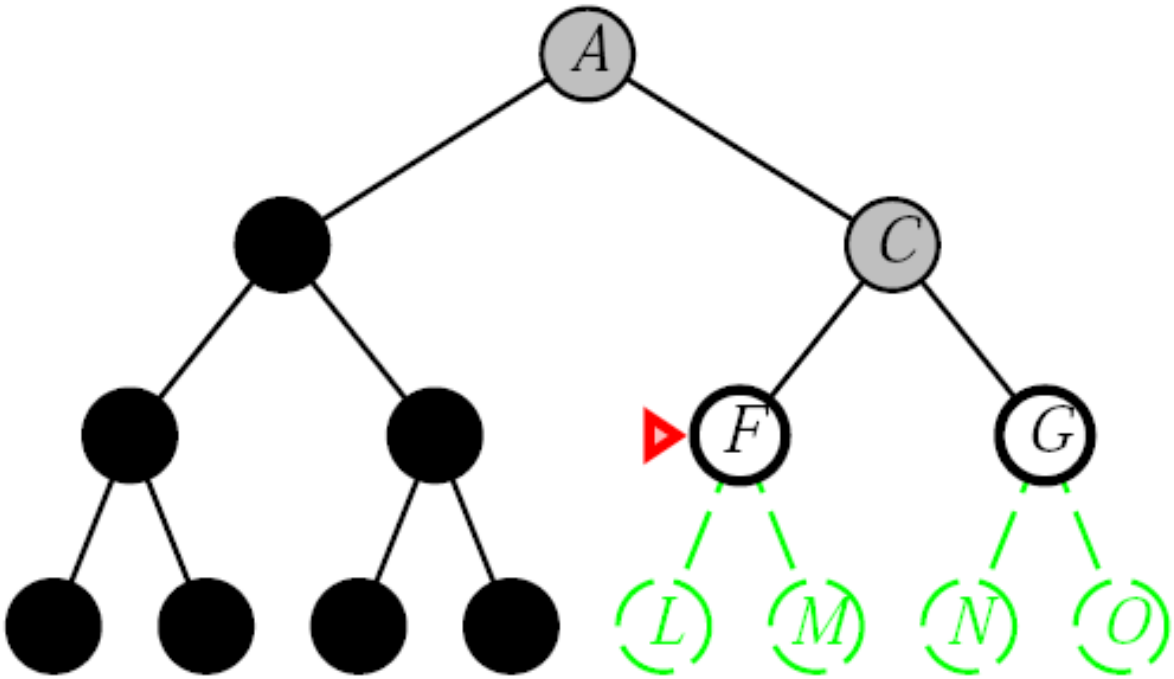
Depth-First Search



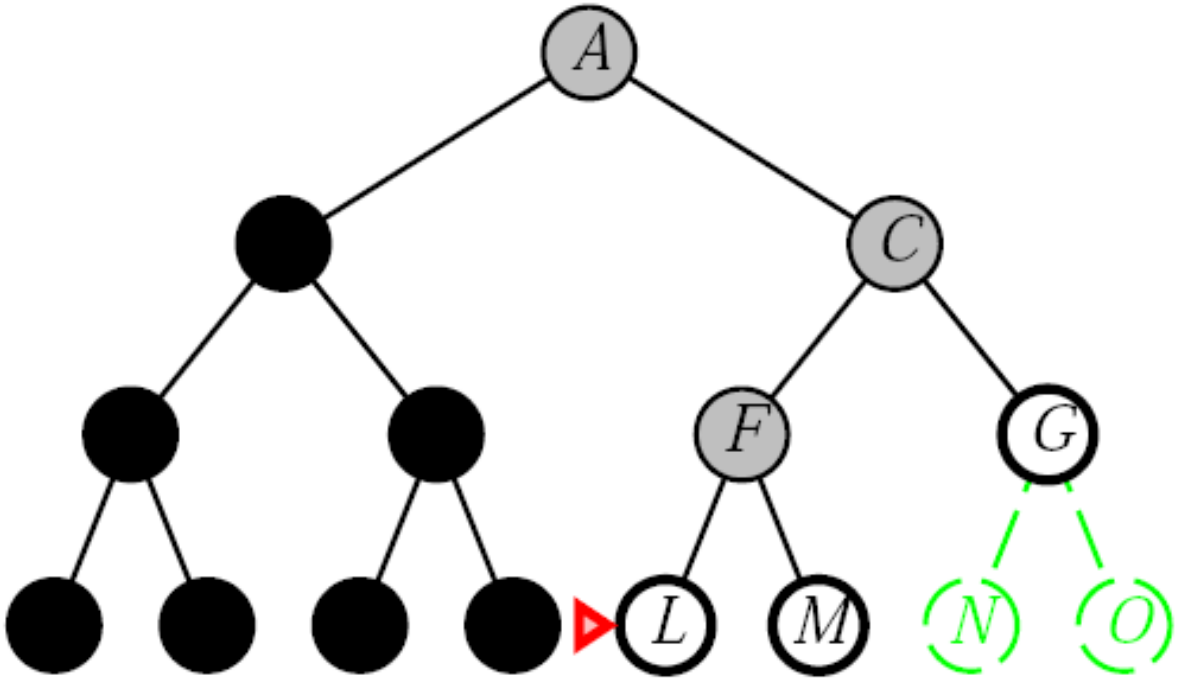
Depth-First Search



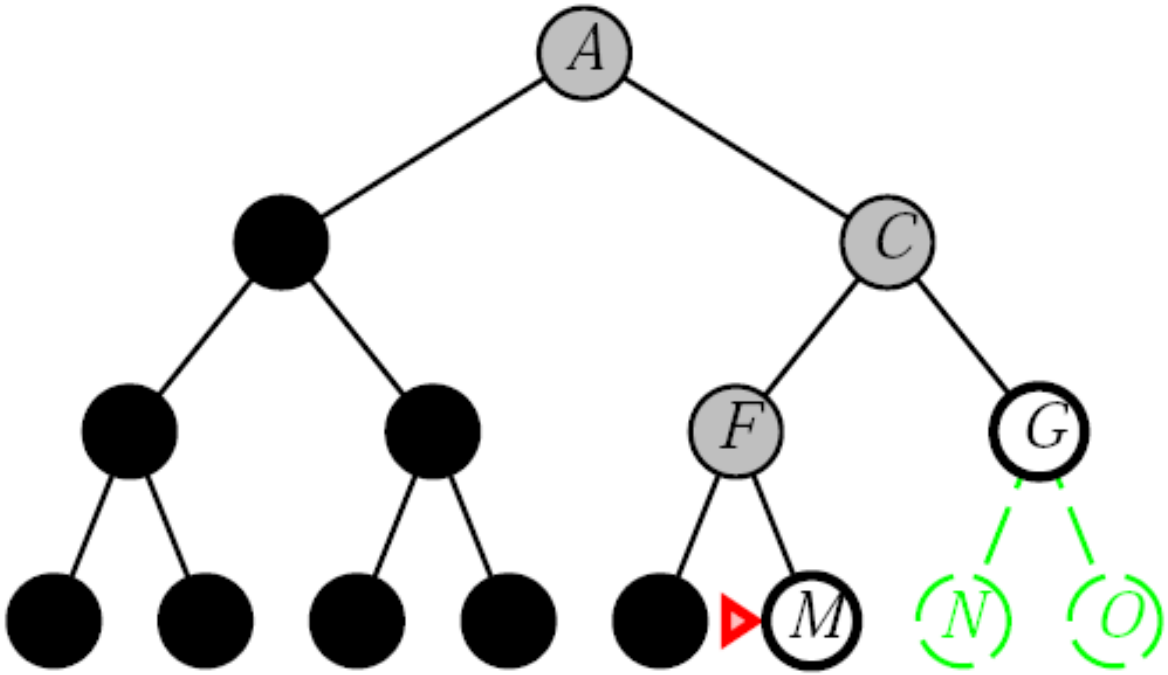
Depth-First Search



Depth-First Search



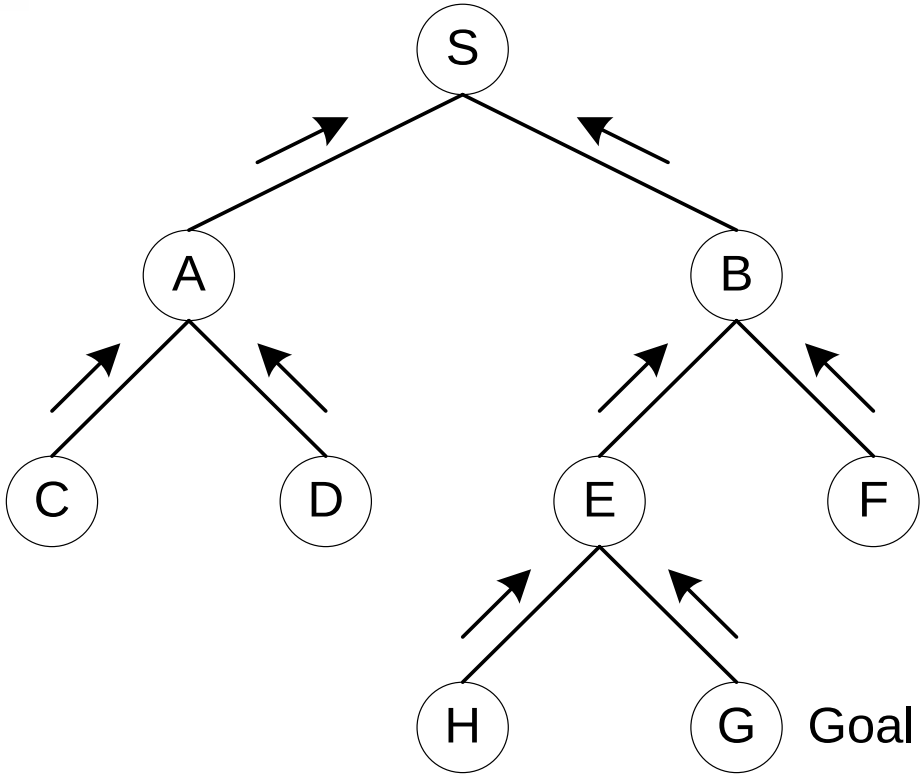
Depth-First Search



Depth-First Search

- Prosedur pelacakan:
  1. Berikan simpul awal pada daftar open
  2. Loop: if open = kosong then exit (fail)
  3. n : = first (open)
  4. if goal (n) then exit (success)
  5. Remove (n, open)
  6. Ekspansikan n, berikan semua simpul anak pada kepala open dan bubuhkan pointer dari simpul anak ke n
  7. Kembali ke Loop
- Penjelasan:
  - Pada langkah 3, elemen pertama daftar open diambil
  - Ekspansi simpul n = pembangkitan simpul-simpul anak dari suatu simpul n

Depth-First Search



Urutan pelacakan:
S, A, C, D, B, E, H, G

Depth-First Search

- **Perubahan daftar ‘open’:**

  (S) $\rightarrow$ (A B) $\rightarrow$ (C D B) $\rightarrow$ (D B) $\rightarrow$ (B) $\rightarrow$ (E F) $\rightarrow$ (H G F) $\rightarrow$ (G F)

- Dengan diagram grafik:

  5. Remove (n, open)

     Add (n, closed)

  6. Ekspansikan n. Berikan pada kepala open semua simpul anak yang belum muncul pada open atau closed dan bubuhkan pointer ke n.

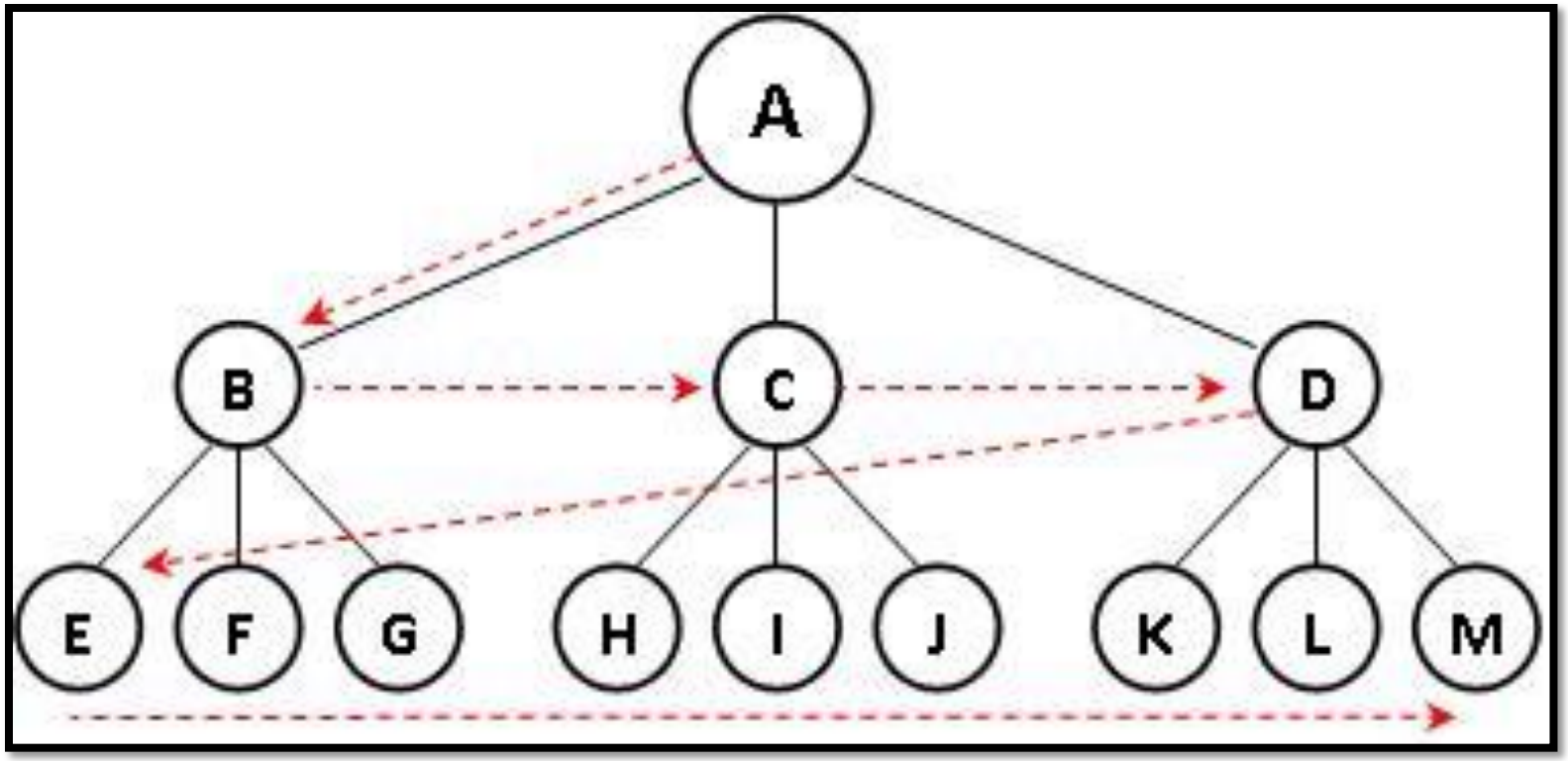
Depth-First Search

- Keuntungan:
  1. Membutuhkan memori yang relatif kecil, karena hanya simpul-simpul pada linatasan yang aktif saja yang disimpan.
  2. Secara kebetulan, metode ini akan menemukan solusi tanpa harus menguji lebih banyak lagi dalam ruang keadaaan.
- Kelemahan:
  1. Memungkinkan tidak ditemukannya tujuan yang diharapkan.
  2. Hanya akan mendapatkan 1 solusi pada setiap pencarian.

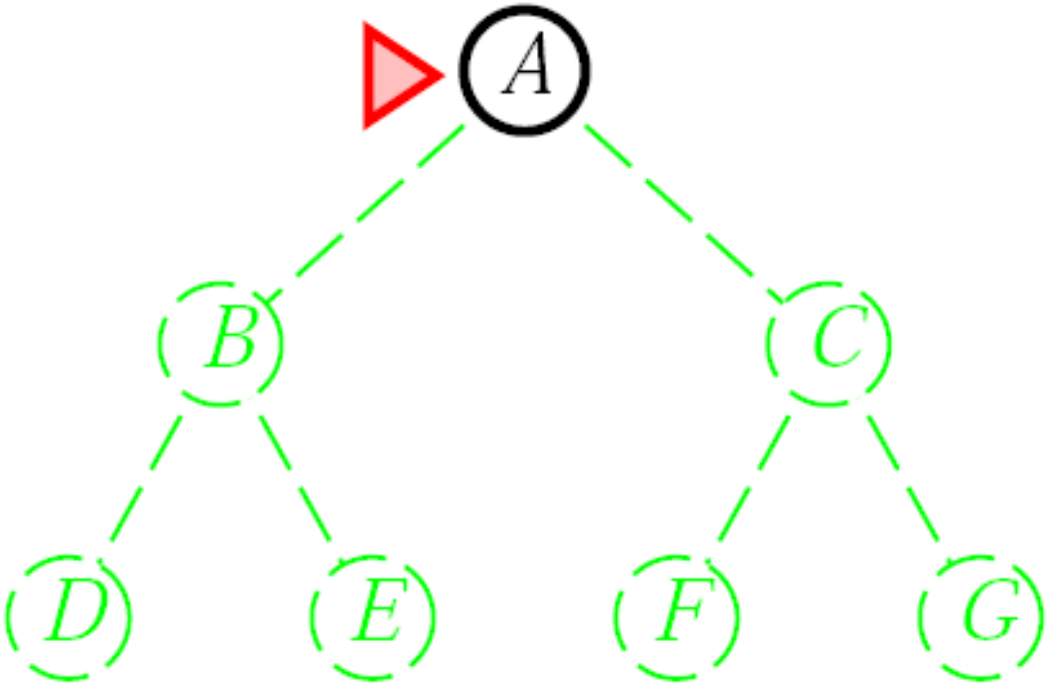
Breadth-First Search

- Bersifat horizontal
- Evaluasi dilakukan terhadap simpul-simpul pada suatu level sebelum dilanjutkan pada level berikutnya
- Pencarian dimulai dari Simpul Akar (Level 0) terus ke Level 1 dari kiri ke kanan sampai semua simpul pada level tersebut dikunjungi, dan seterusnya ke Level 2, 3, ... sampai ditemukannya solusi.

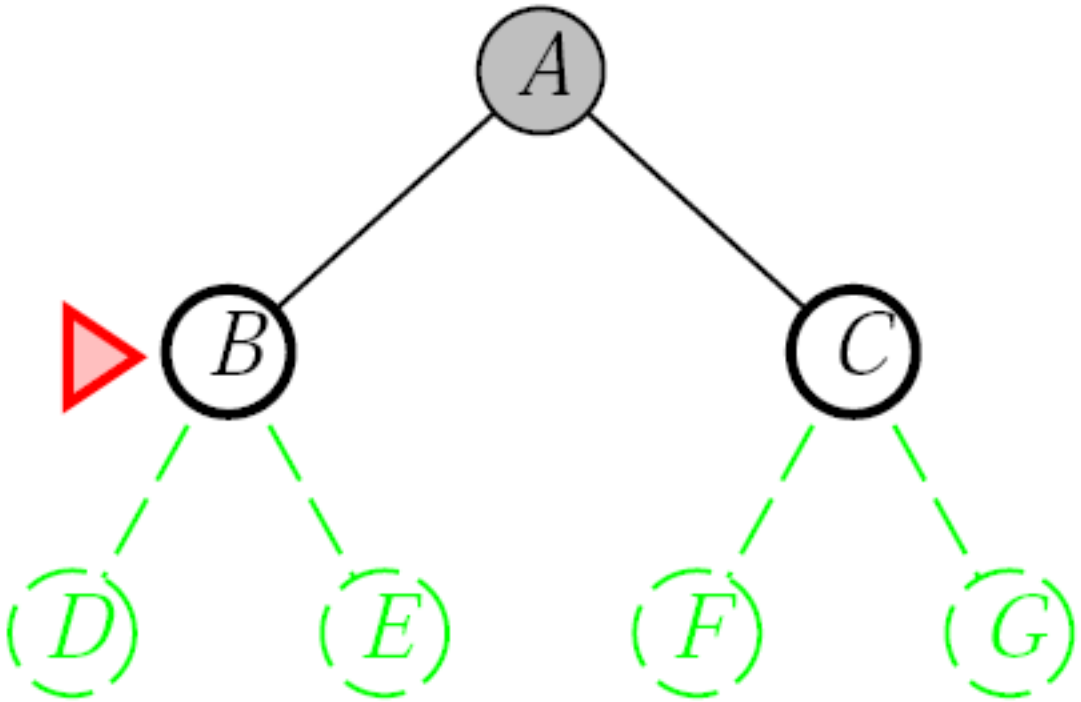
Breadth-First Search



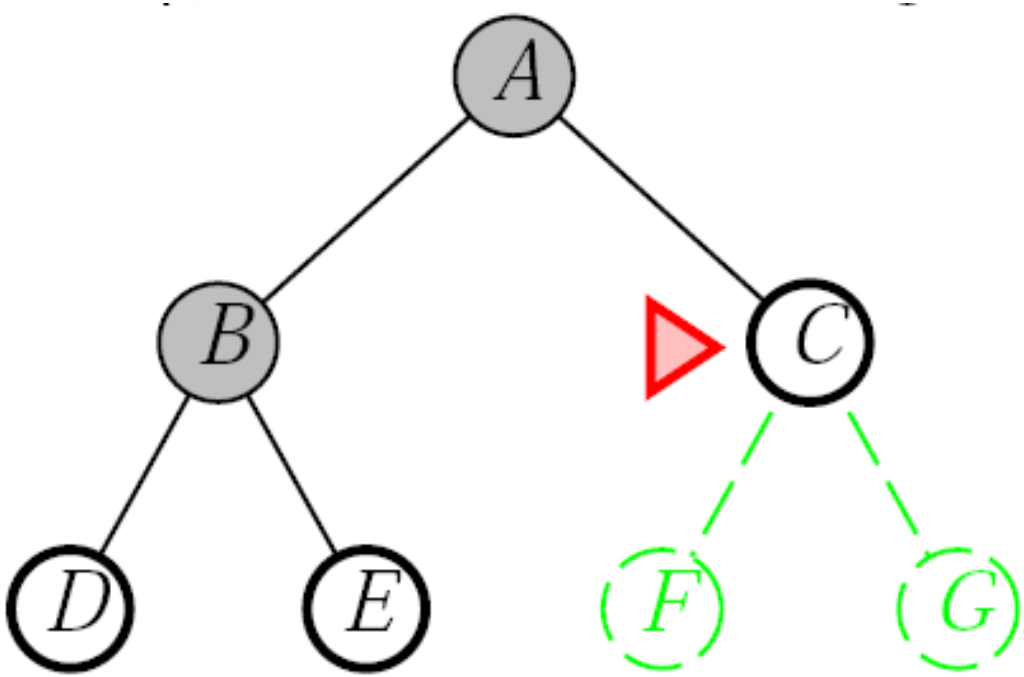
Breadth-First Search



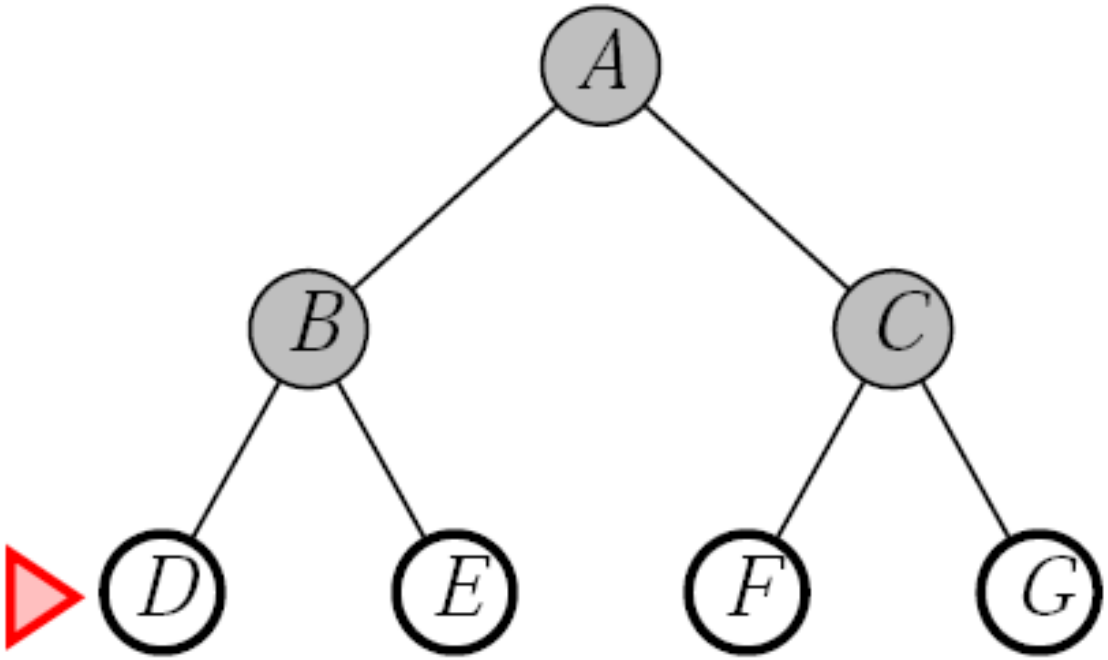
Breadth-First Search



Breadth-First Search



Breadth-First Search



Breadth-First Search

- Prosedur Pelacakan:
  1. Berikan simpul awal pada open
  2. Loop: if open = kosong then exit (fail)
  3. n : = first (open)
  4. If goal (n) then exit (success)
  5. Remove (n, open)
  6. Add (n, closed)
  7. Ekspansikan n. Berikan pada ekonr open semua simpul anak yang belum muncul pada open atau closed dan bubuhkan pointer ke n.
  8. Kembali ke Loop

Breadth-First Search

- Untuk menghitung time complexity dan space complexity dari teknik ini, misalkan branching factor pada tree yang digunakan adalah b, dan goal state terdapat pada level ke-d dari tree
- Maka, banyaknya nodes maksimum yang akan ditelusuri dan disimpan sampai menemukan solusi (mencapai goal state) adalah sbb:

$$1 + b + b^2 + b^3 + \cdots + b^d$$

Breadth-First Search

- Kebutuhan waktu dan memori BFS

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

- Branching factor b=10; 1000 nodes/second; 100 bytes/node

Breadth-First Search

- Keuntungan:
  1. Tidak akan menemui jalan buntu
  2. Jika hanya ada satu solusi, maka akan menemukannya. Dan jika ada lebih dari satu solusi, maka solusi minimum (dalam hal kedalaman tree) akan ditemukan.
- Kelemahan:
  1. Membutuhkan memori yang cukup banyak, karena menyimpan semua simpul dalam satu pohon.
  2. Membutuhkan waktu yang cukup lama, karena akan menguji n level untuk mendapatkan solusi pada level yang ke (n+1).

Teknik Informed Search

Pendahuluan

- Blind Search (uninformed search) tidak selalu dapat diterapkan dengan baik.
- Di antara kelemahannya adalah waktu aksesnya yang cukup lama serta besarnya memori yang diperlukan.
- Kelemahan ini sebenarnya dapat diatasi dengan memberikan informasi tambahan dari domain yang bersangkutan

Pendahuluan

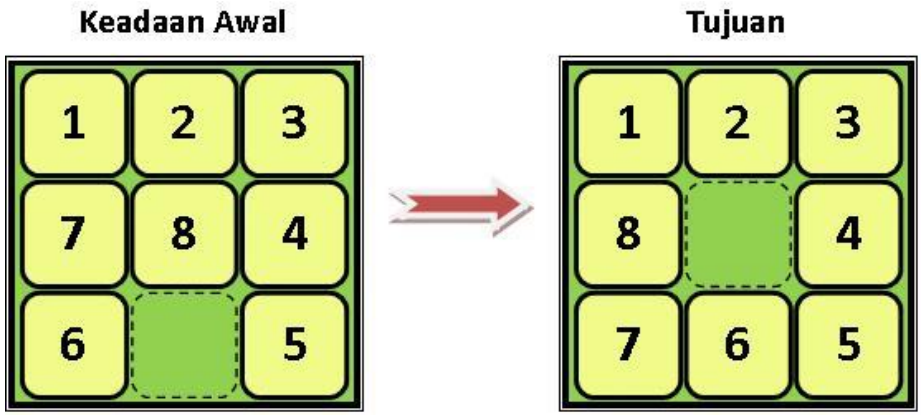
- Heuristik adalah “rule of thumb” berbasis knowledge untuk menentukan pilihan dari sejumlah alternatif untuk mencapai sasaran dengan efektif
- Heuristic bersifat estimasi, bukan perhitungan aktual/real
- Heuristik dipergunakan untuk mempersempit ruang pelacakan → mengurangi waktu dan memory yang dipakai
- Heuristic juga digunakan untuk mendapatkan solusi yang optimal
- Heuristic search → informed search, tidak berlaku sebaliknya

Blind Search x Heuristic Search

- Perhatikan contoh kasus pada masalah 8-puzzle, dimana ada 4 operator yang dapat digunakan (sama seperti 4-puzzle) yaitu:
  1. BU (Blank Up)
  2. BD (Blank Down)
  3. BL (Blank Left)
  4. BR (Blank Right)

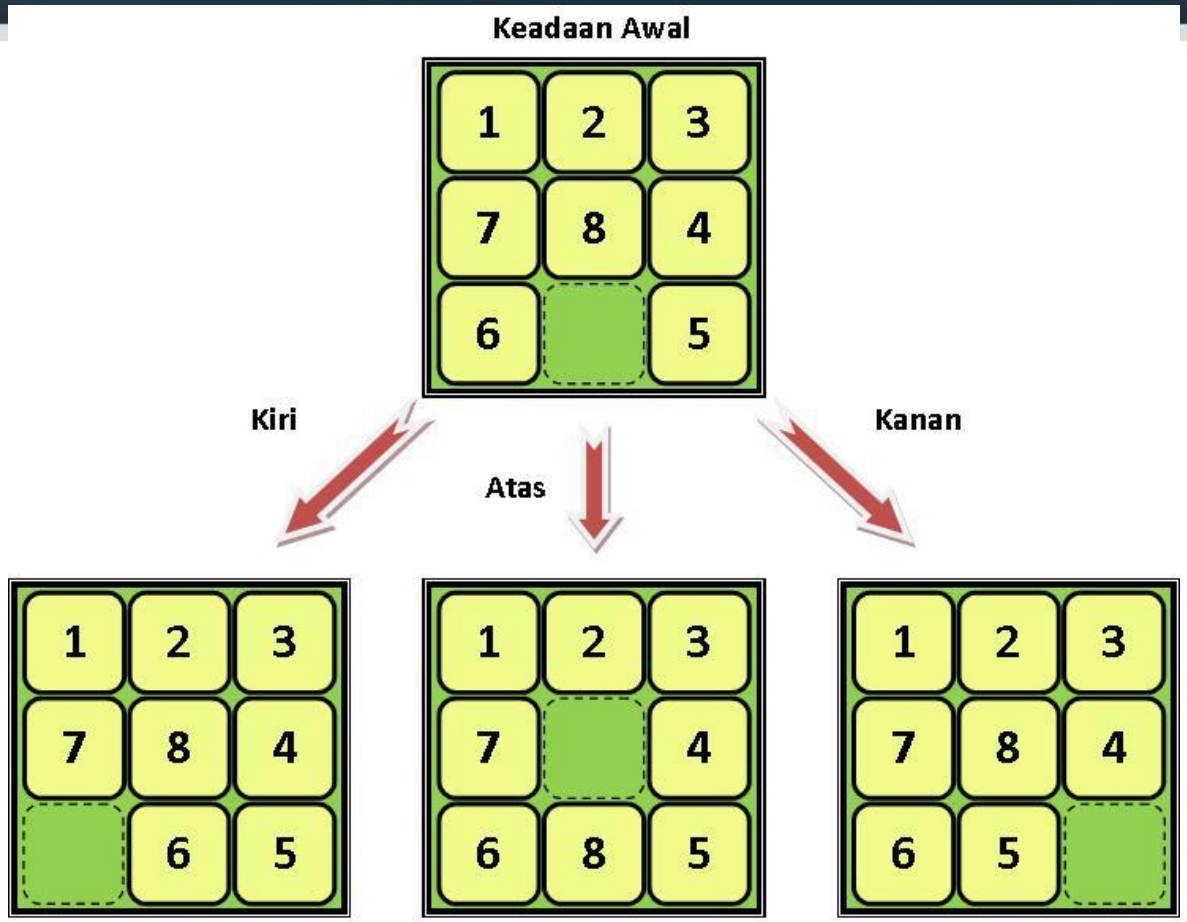
Blind Search x Heuristic Search

- Contoh:



- Dari keadaan (state) awal, hanya 3 operator yang dapat digunakan, yaitu BL atau BR atau BU.

Blind Search x Heuristic Search

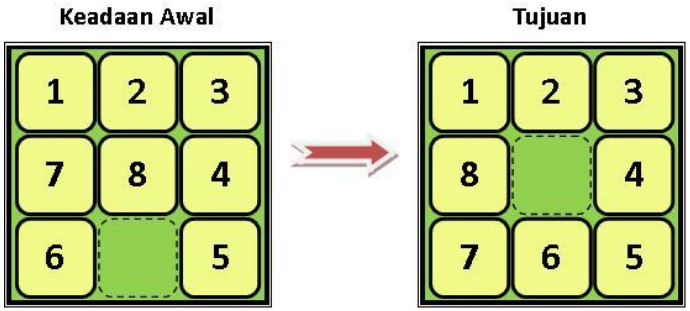


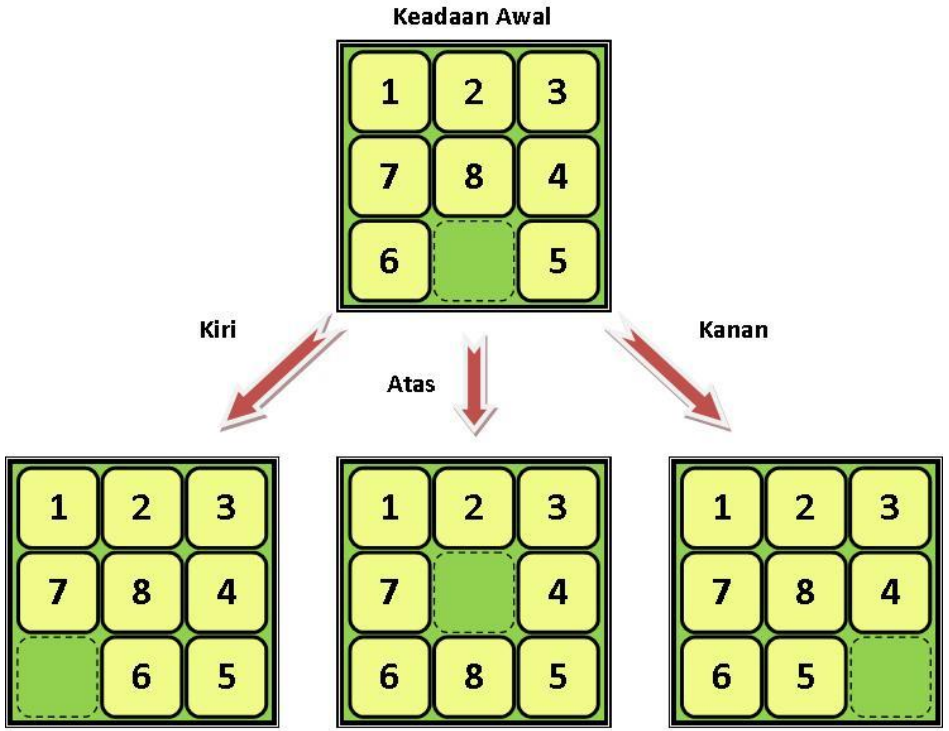
Blind Search x Heuristic Search

- Apabila digunakan blind search, kita tidak perlu mengetahui atau mempertimbangkan next-state mana yang akan dipilih (sembarang next-state dapat dipilih), pemilihan solusi tergantung pada teknik yang digunakan
- Sedangkan pada pencarian heuristik, pemilihan next-state menjadi penting untuk memperoleh solusi yang dianggap paling optimal
- Untuk itu, di dalam pencarian heuristik perlu ditambahkan informasi tertentu dalam domain masalah tersebut

Blind Search x Heuristic Search

- Informasi apa saja yang dapat ditambahkan di dalam pencarian heuristik untuk pemilihan next-state dalam masalah 8-puzzle?

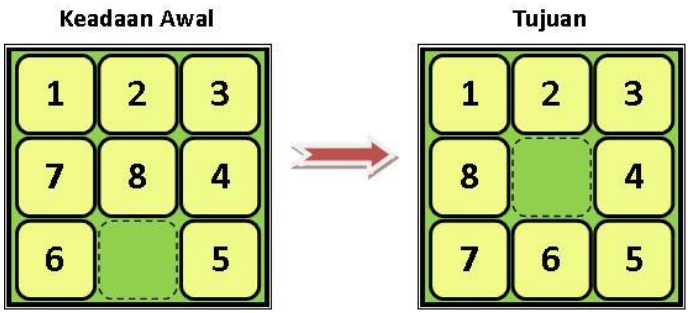


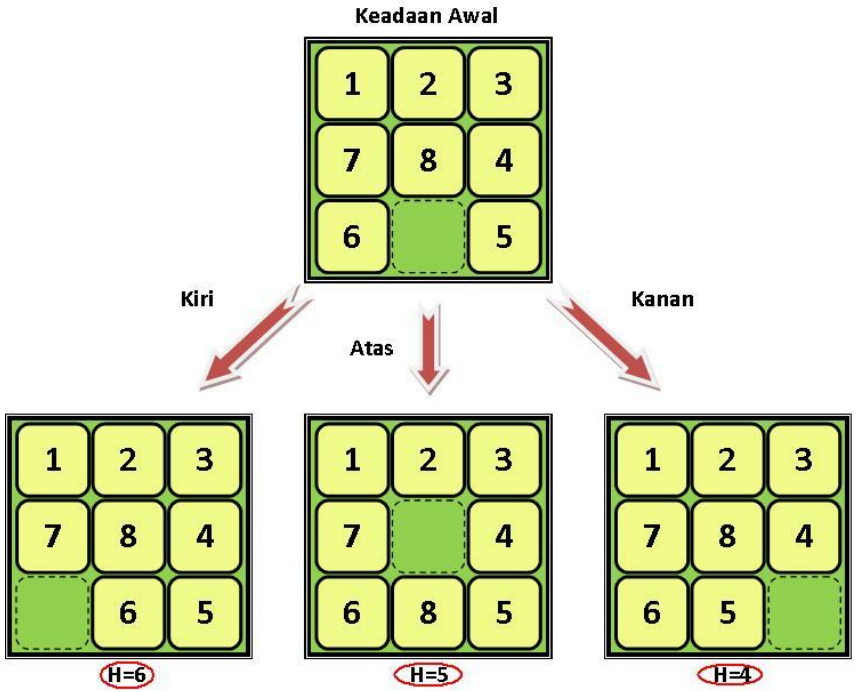


Blind Search x Heuristic Search

- Beberapa informasi yang dapat ditambahkan di dalam pencarian heuristik untuk mendapatkan solusi optimal dalam masalah 8-puzzle:

1. **Jumlah ubin yang menempati posisi yang benar**. Jumlah yang lebih tinggi adalah jumlah yang lebih diharapkan (lebih baik).

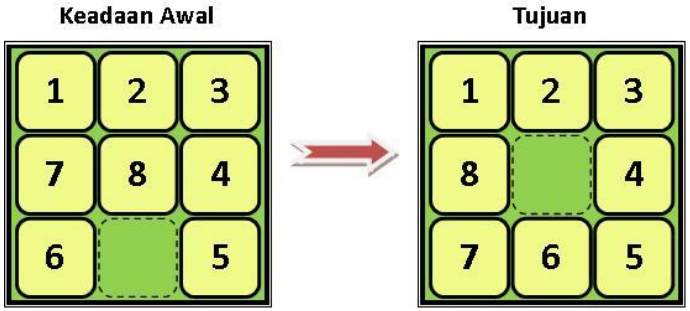


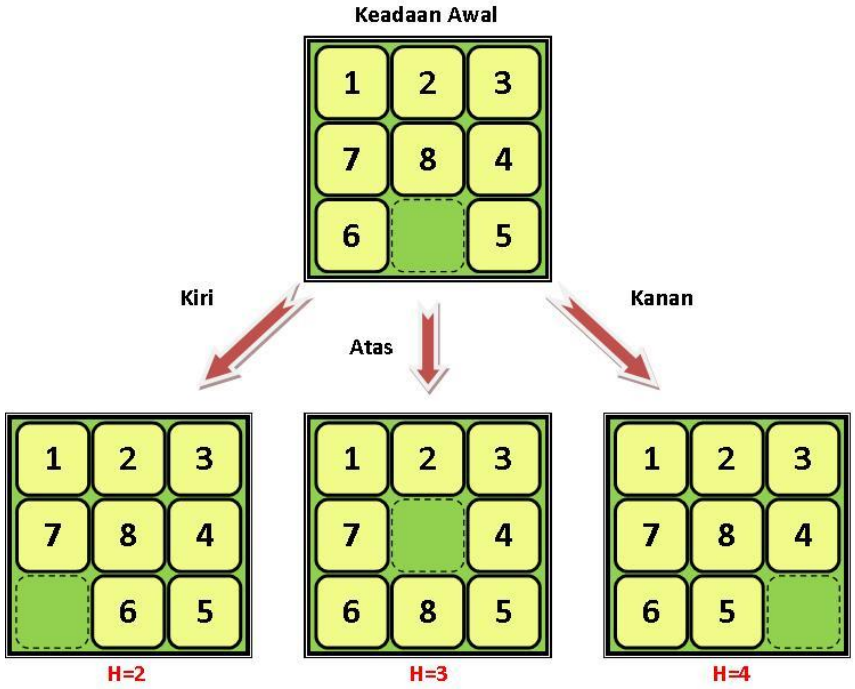


Blind Search x Heuristic Search

2. **Jumlah ubin yang menempati posisi yang salah.**

   Jumlah yang lebih kecil adalah yang diharapkan (lebih baik).

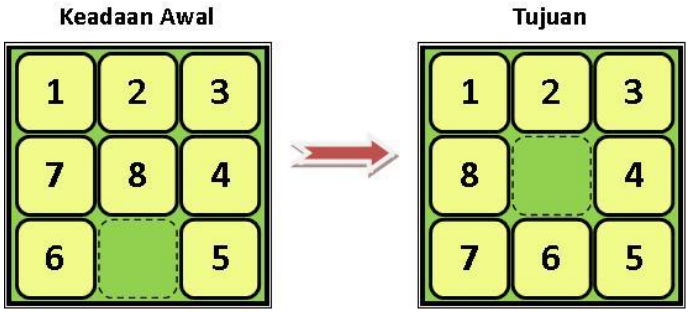


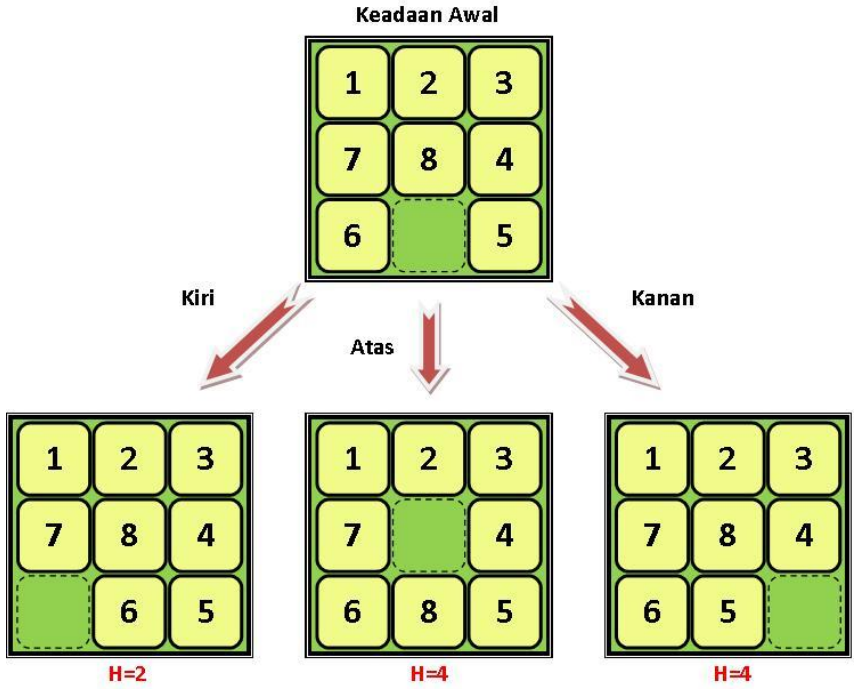


Blind Search x Heuristic Search

3. **Menghitung total gerakan vertikal dan horizontal yang diperlukan oleh ubin selain Blank untuk mencapai tujuan.**

   Jumlah yang lebih kecil adalah yang diharapkan (lebih baik).



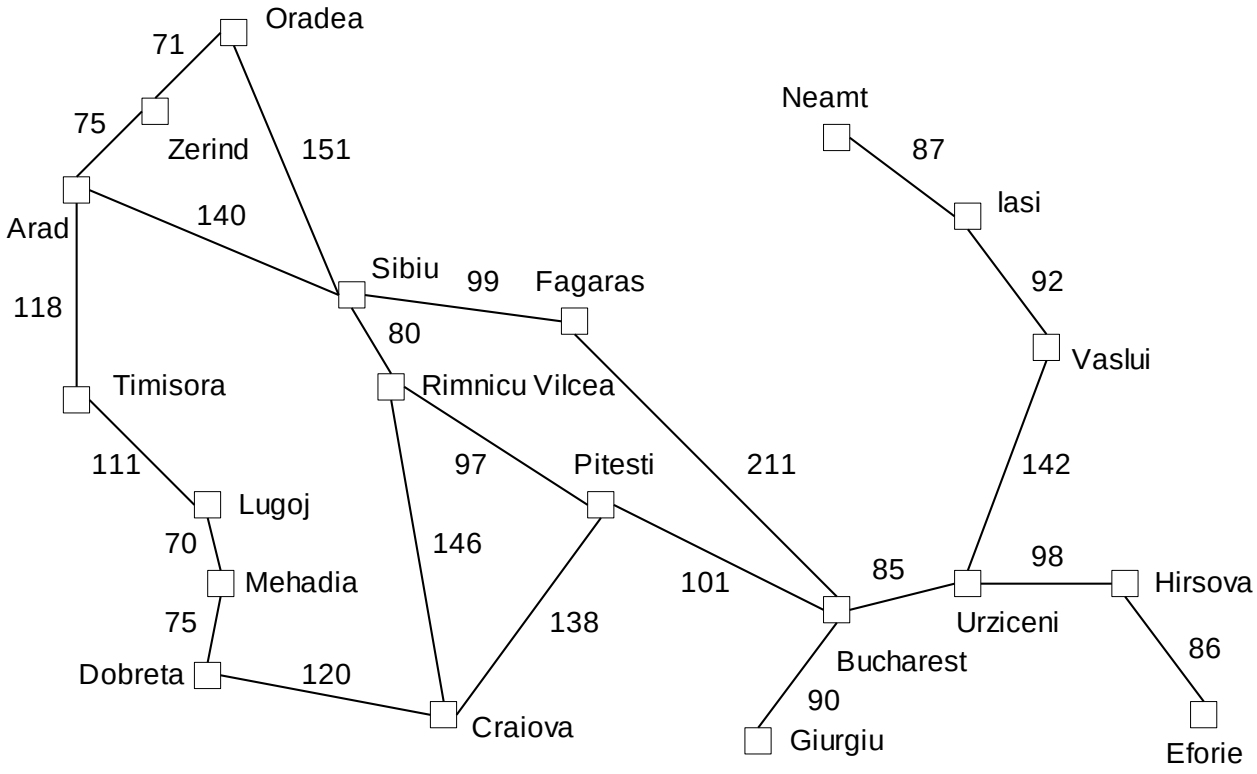


Uniform Cost Search (UCS)

- UCS melakukan searching berdasarkan informasi cost aktual (yang sebenarnya)
- Tidak menggunakan estimasi → bukan heuristic search, tapi termasuk informed search
- Next-state dipilih berdasarkan actual-cost terkecil atau terbesar (tergantung permasalahan)
- Pada akhirnya UCS akan menyisir semua nodes yang ada di dalam tree

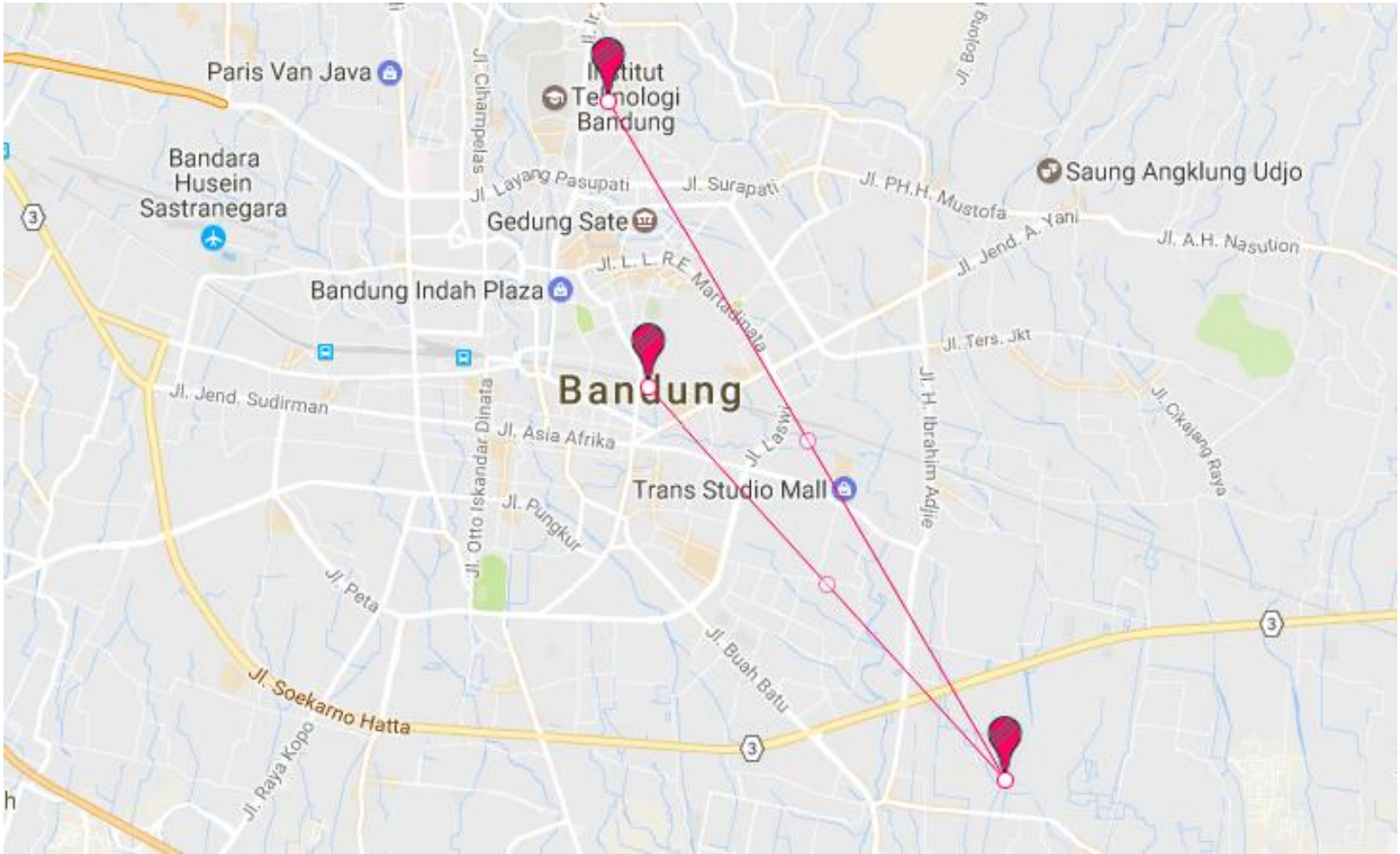
Finding Route

- Perhatikan peta di bawah ini



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Straight-line distance



Uniform Cost Search (UCS)

- Bagaimana mencari rute terpendek dari Arad-Bucharest menggunakan UCS?
- (dijelaskan di kelas, pembahasan di papan tulis ☺)

Uniform Cost Search (UCS)

- Kelebihan UCS:
  - Pasti menghasilkan solusi optimal
  - Optimal → complete, tidak berlaku sebaliknya
- Kekurangan:
  - Time complexity besar
  - Space complexity besar

Teknik Heuristik

- Beberapa teknik yang akan dipelajari:
  1. Best-First Search
     a. Greedy Search
     b. A* search
  2. Hill Climbing

Best First Search

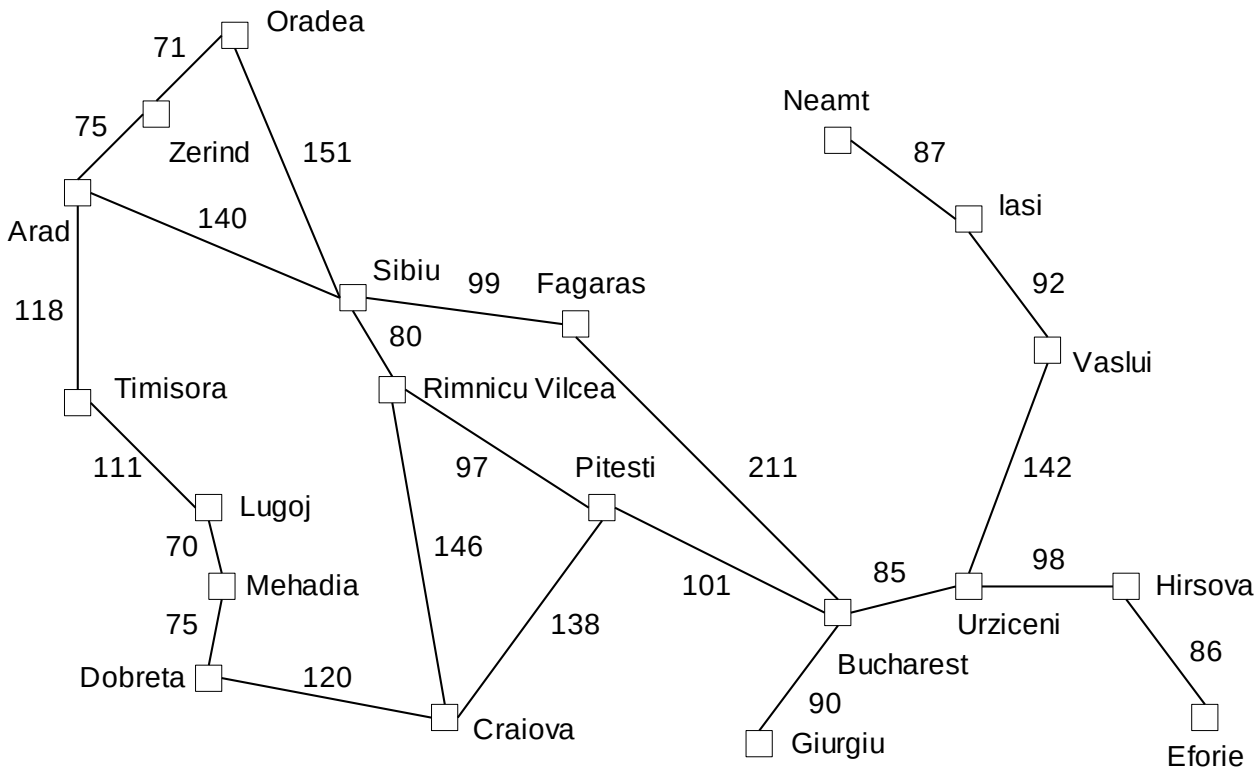
- Secara umum, prinsip yang digunakan:
  - Pada setiap langkah, dipilih simpul yang **diperkirakan** lebih dekat terhadap sasaran dari semua simpul yang dibangkitkan
- Ada 2 teknik yang akan dipelajari:
  1. Greedy Search
  2. A* search

Greedy Best First Search

- Digunakan fungsi evaluasi (fungsi heuristic), yaitu *h(n)*
- *h(n)* **mengestimasikan** “cost” dari n ke titik goal
- Perhatikan bahwa “cost aktual” menunjukkan biaya/cost real dari initial state ke titik n,
- Sementara “estimation cost” mengestimasi biaya/cost dari titik n ke titik goal
- “Cost” dapat berupa jarak, waktu, biaya, atau parameter lainnya tergantung permasalahan yang akan dicari solusinya
- Greedy search mengekspansi node yang ***tampak*** paling dekat ke tujuan (goal state)

Greedy Best First Search

- Carilah rute dari Arad ke Bucharest menggunakan greedy search!



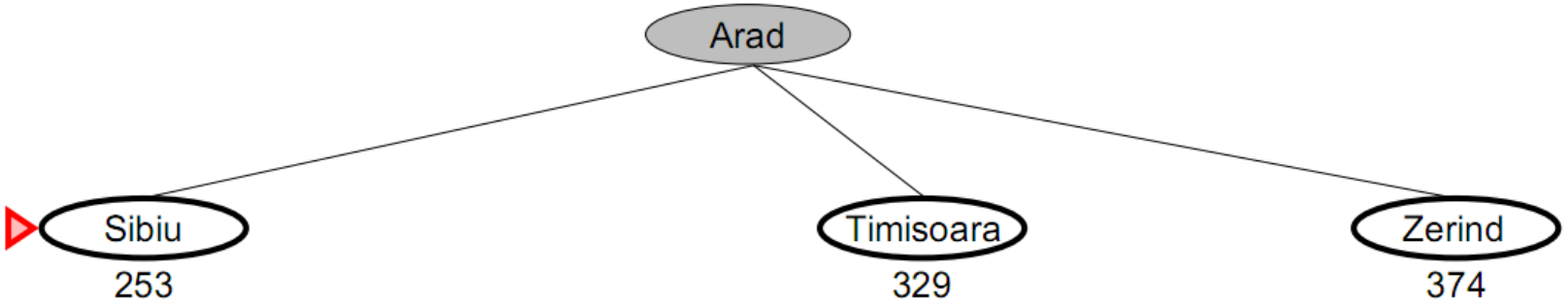
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best First Search

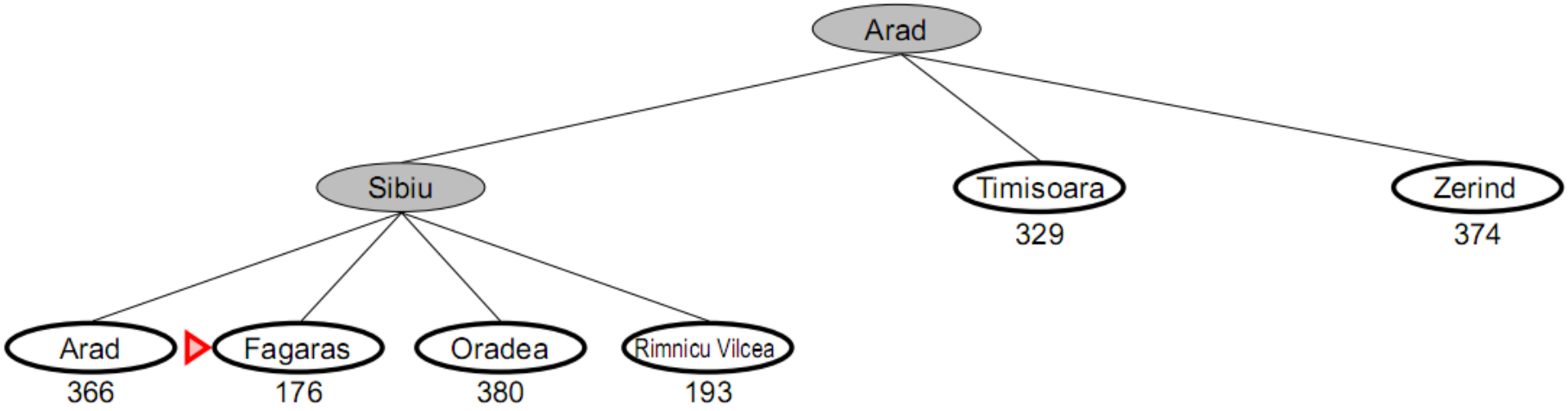
- Parameter apa yang dapat kita gunakan untuk mengestimasi cost dari kota tertentu ke kota tujuan (Bucharest)?
- Apakah jarak antar kota (yang tercantum dalam map) dapat digunakan sebagai estimation cost?
- → Tidak dapat...Mengapa?
- Maka, kita dapat menyatakan estimation cost atau *h(n)* sebagai jarak straight-line dari kota n ke kota tujuan (Bucharest)
- Sehingga node yang akan dipilih ketika ekspansi nodes adalah node yang memiliki *h(n)* atau jarak straight-line terpendek ke Bucharest

Greedy Best First Search

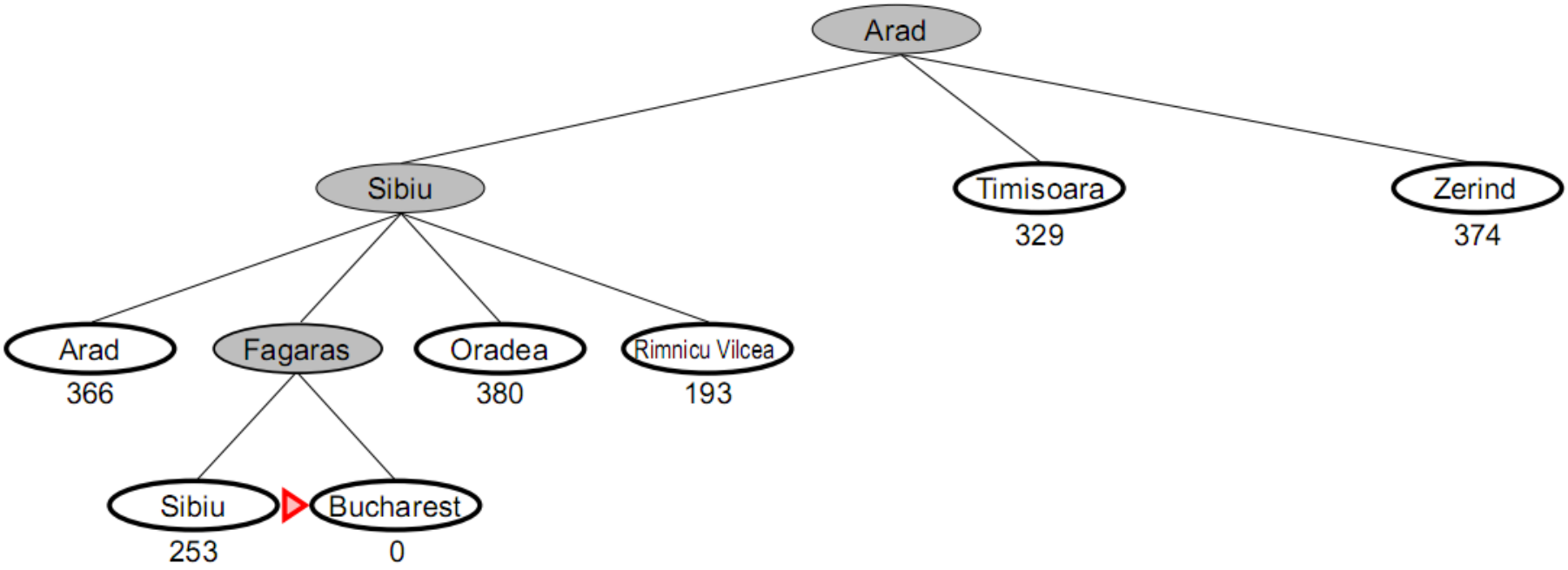
Greedy Best First Search



Greedy Best First Search



Greedy Best First Search



Greedy Best First Search

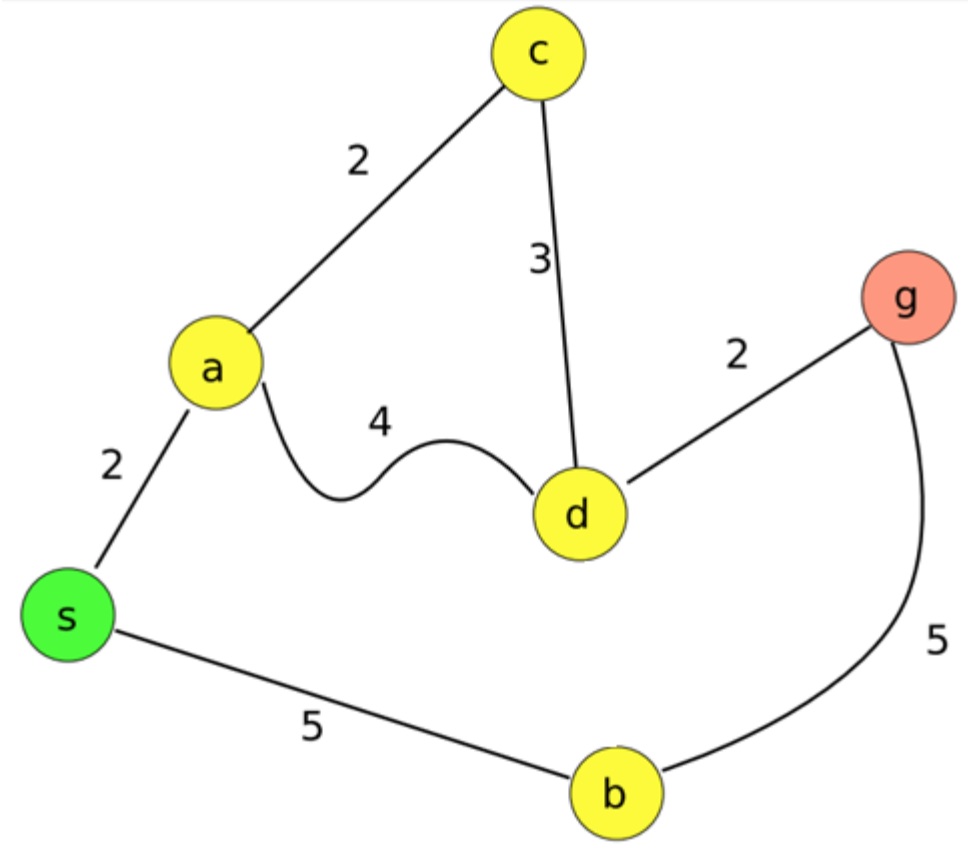
- Salah satu kelemahan greedy search adalah memungkinkan terjadinya stuck in loops dan tidak optimalnya hasil yang diperoleh

What is Admissible Heuristic?

- A heuristic h(n) is **admissible** if for every node n, h(n) ≤ h*(n), where h*(n) is the actual minimum cost to reach the goal state from n
- An admissible heuristic never overestimates the actual cost to reach the goal, i.e., it is optimistic
- Example: straight-line-distance, $h_{SLD}(n)$ (never overestimates the actual road distance)

Contoh-1

- Diketahui state-space seperti di samping. Jika initial state adalah s dan final statenya adalah g, tentukan solusi terbaik menggunakan Uniform Cost Search



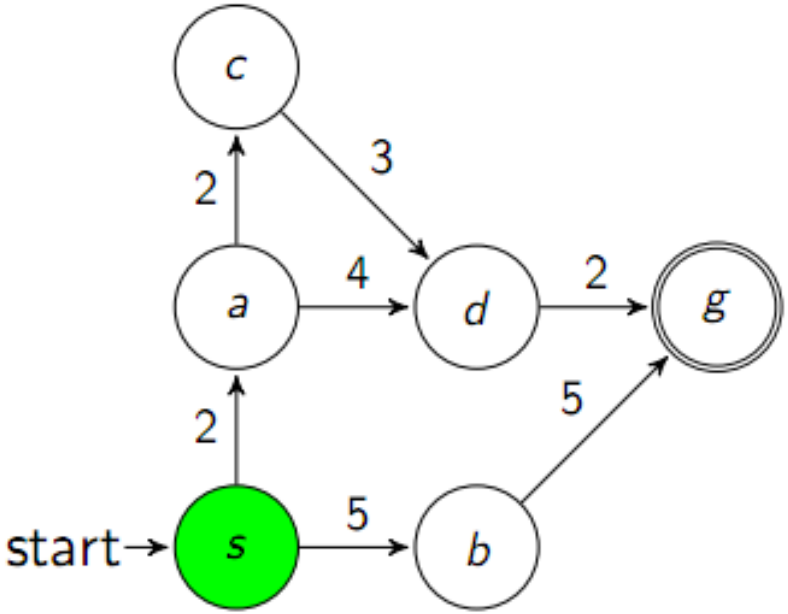
Contoh-2

- Cari solusi dari soal sebelumnya menggunakan greedy dan A/A* search jika diketahui cost menunjukkan jarak, dan fungsi heuristic h(n) dari titik n ke final state (g) adalah sebagai berikut:
  a. h(a) = 2, h(b) = 3, h(c) = 1, h(d) = 4, h(g) = 0, h(s) = 10
  b. h(a) = 2, h(b) = 3, h(c) = 1, h(d) = 1, h(g) = 0, h(s) = 6

Contoh 1- UC Search

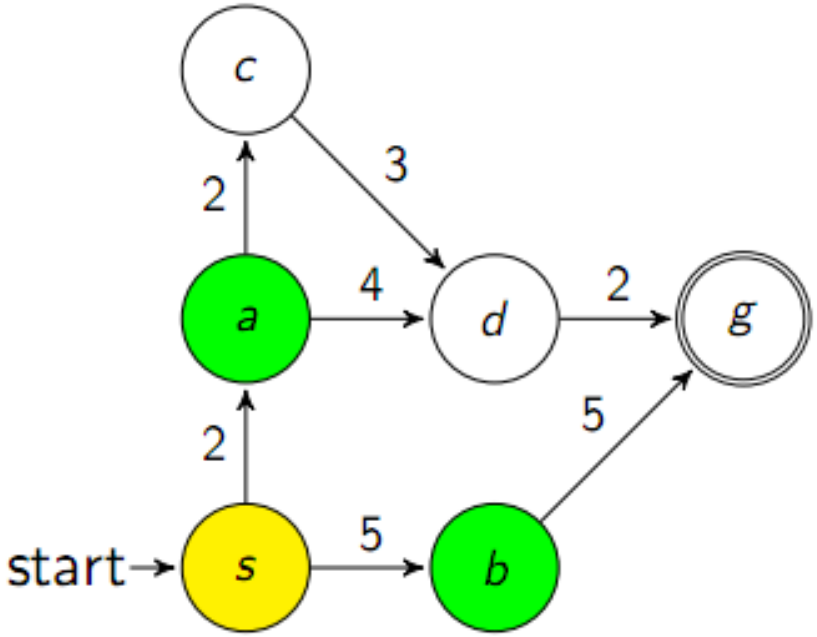
Q:

path	cost
$\langle s \rangle$	0



Contoh 1- UC Search

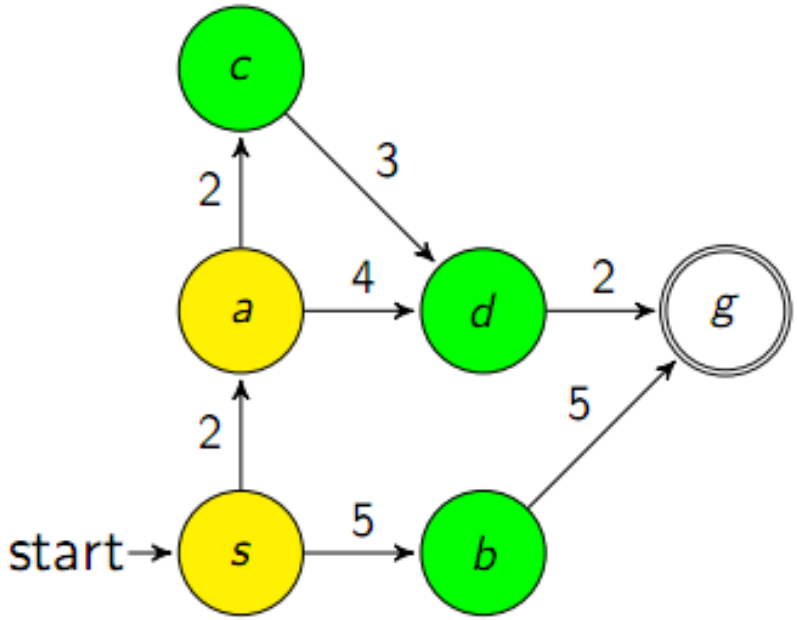
Q:	path	cost
	$\langle a, s \rangle$	2
	$\langle b, s \rangle$	5



Contoh 1- UC Search

Q:

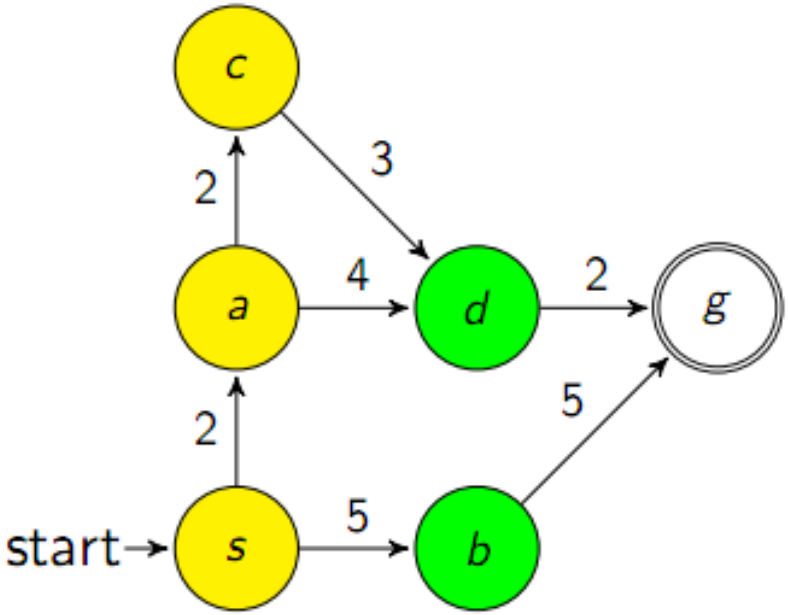
state	cost
$\langle c, a, s \rangle$	4
$\langle b, s \rangle$	5
$\langle d, a, s \rangle$	6



Contoh 1- UC Search

Q:

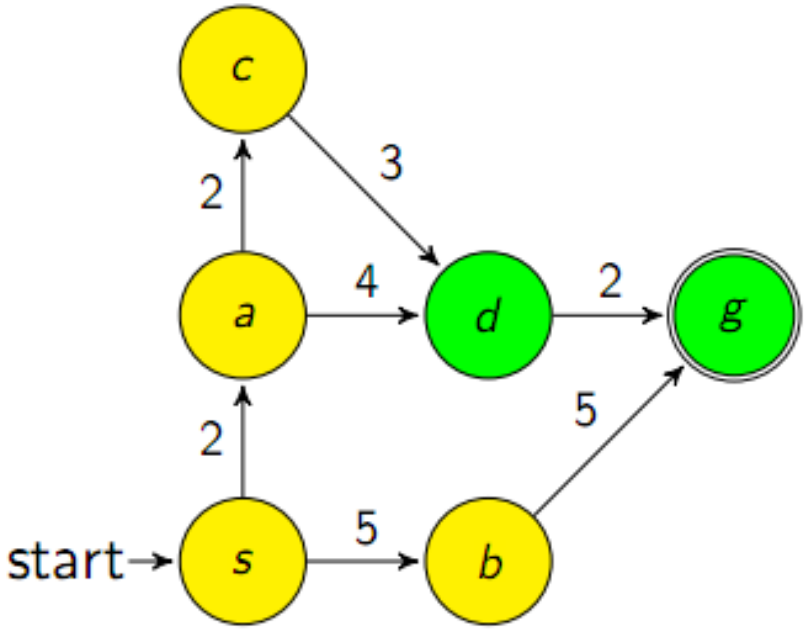
state	cost
$\langle b, s \rangle$	5
$\langle d, a, s \rangle$	6
$\langle d, c, a, s \rangle$	7



Contoh 1- UC Search

Q:

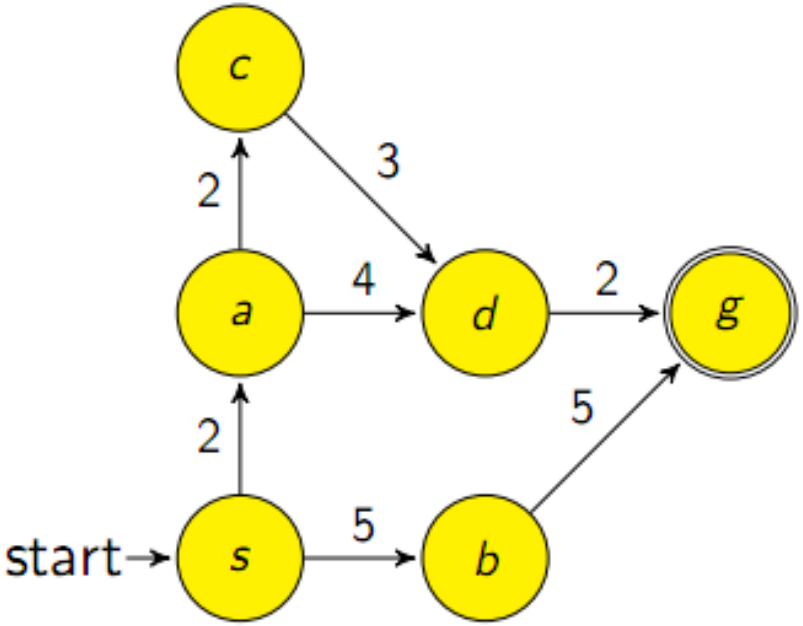
state	cost
$\langle d, a, s \rangle$	6
$\langle d, c, a, s \rangle$	7
$\langle g, b, s \rangle$	10



Contoh 1- UC Search

Q:

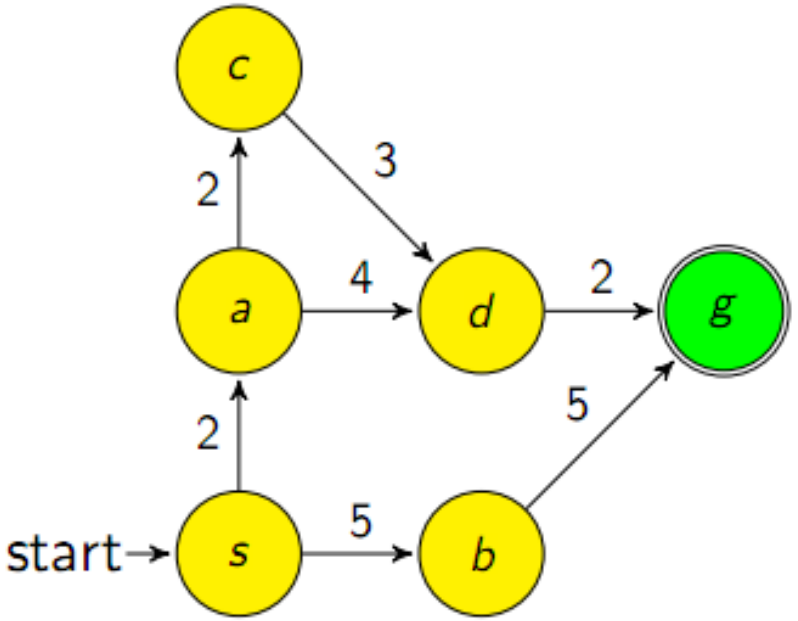
state	cost
$\langle d, c, a, s \rangle$	7
$\langle g, d, a, s \rangle$	8
$\langle g, b, s \rangle$	10



Contoh 1- UC Search

Q:

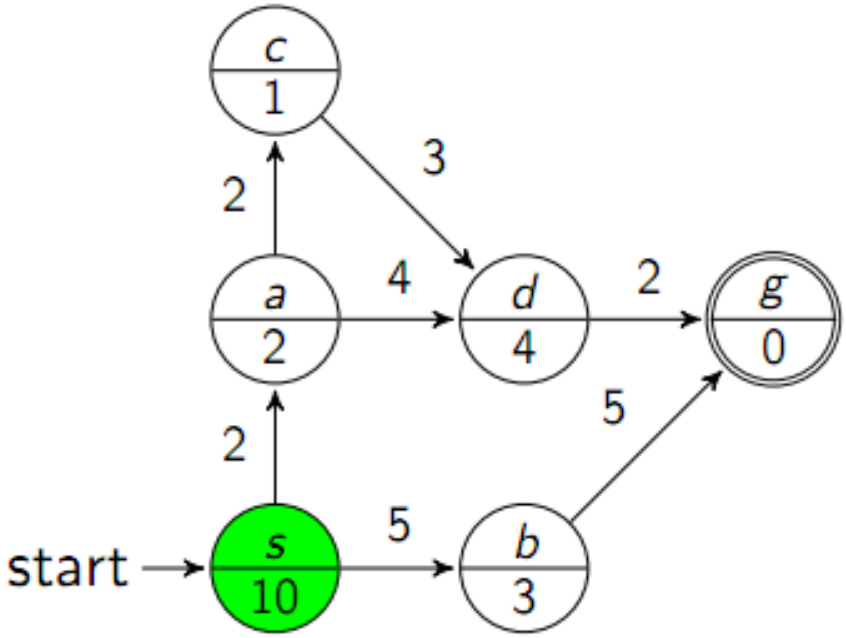
state	cost
$\langle g, d, a, s \rangle$	8
$\langle g, d, c, a, s \rangle$	9
$\langle g, b, s \rangle$	10



Contoh-2a Greedy-Revising

Q:

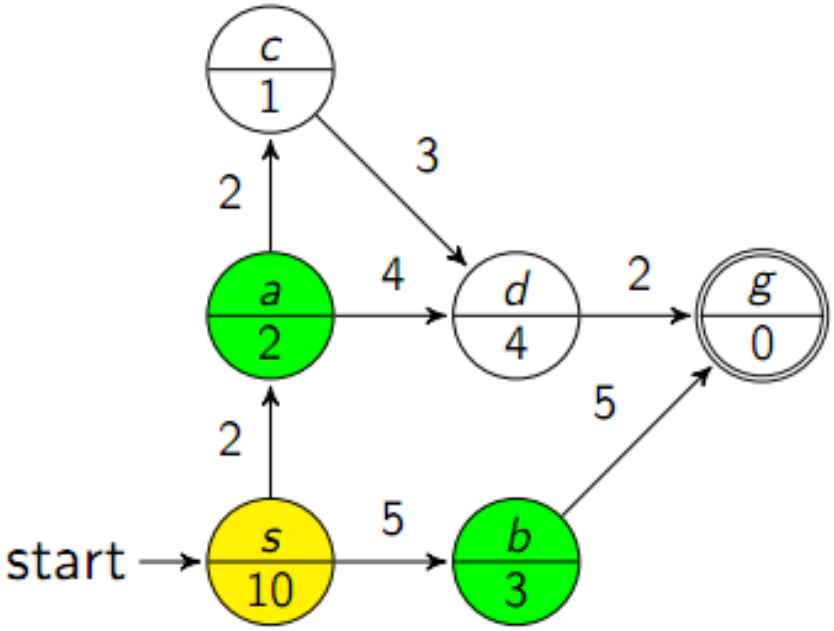
path	cost	h
$\langle s \rangle$	0	10



Contoh-2a Greedy-Revising

Q:

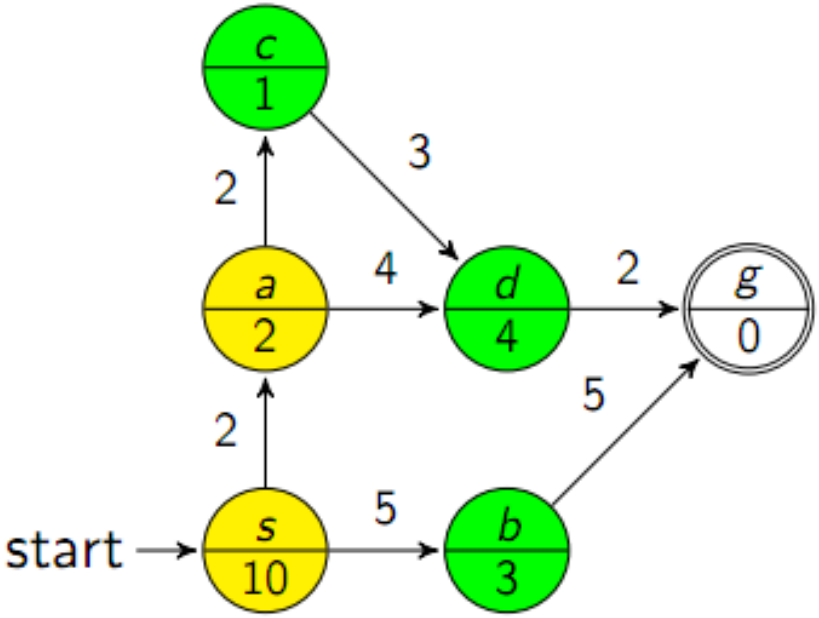
path	cost	h
$\langle a, s \rangle$	2	2
$\langle b, s \rangle$	5	3



Contoh-2a Greedy-Revising

Q:

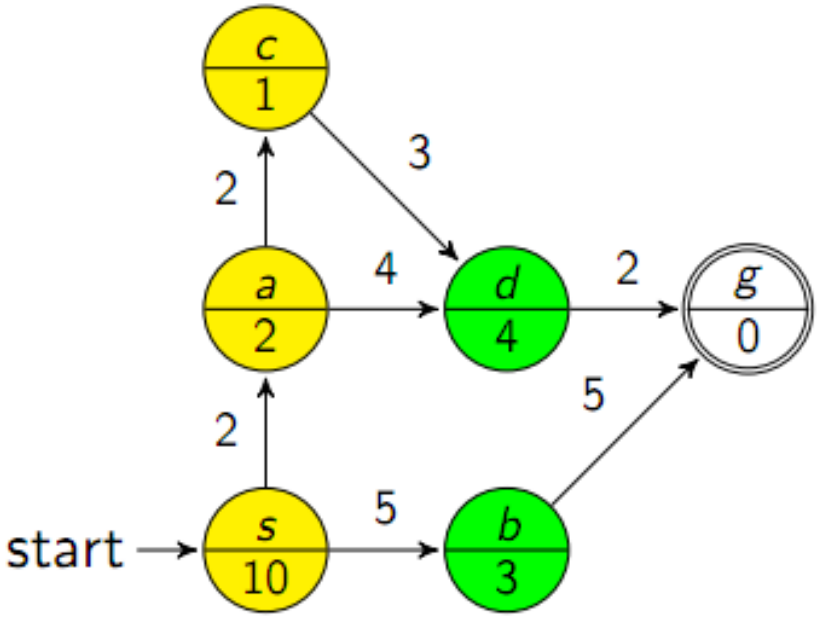
path	cost	h
$\langle c, a, s \rangle$	4	1
$\langle b, s \rangle$	5	3
$\langle d, a, s \rangle$	6	4



Contoh-2a Greedy-Revising

Q:

path	cost	h
$\langle b, s \rangle$	5	3
$\langle d, a, s \rangle$	6	4
$\langle d, c, a, s \rangle$	7	4



Contoh-2a Greedy-Revising

Q:

path	cost	h
$\langle g, b, s \rangle$	10	0
$\langle d, a, s \rangle$	6	4
$\langle d, c, a, s \rangle$	7	4