# Kecerdasan Buatan

## Algoritma Pencarian Heuristik

Oleh Politeknik Elektronika Negeri Surabaya

2017

Politeknik Elektronika Negeri Surabaya
Departemen Teknik Informatika dan Komputer

# Heuristic Search

- Best First Search
- A*
- Hill climbing
- Branch and Bound
- Dynamic Programming

# Tujuan Instruksi Umum

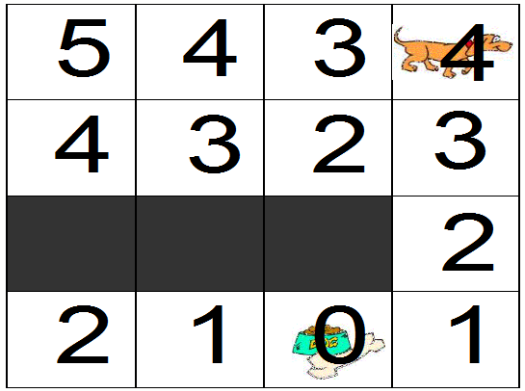
Mahasiswa memahami filosofi Kecerdasan Buatan dan mampu menerapkan beberapa metode Kecerdasan Komputasional dalam menyelesaikan sebuah permasalahan, baik secara individu maupun berkelompok/kerjasama tim.

# Tujuan Instruksi Khusus

- Mengetahui definisi Algoritma Pencarian
- Mengetahui contoh algoritma pencarian yang mempertimbangkan bobot

# Heuristic Search

- Pencarian heuristik memanfaatkan “knowledge” tambahan tentang masalah yang membantu pencarian langsung ke jalur yang lebih menjanjikan.
- Fungsi heuristik, melakukan estimasi seberapa jauh kita dari sebuah tujuan. Contoh:



# Menentukan sebuah state lebih baik dibandingkan dengan state lainnya.

| | | |
|---|---|---|
| 7 | 8 | 4 |
| 3 | 5 | 1 |
| 6 | 2 | |

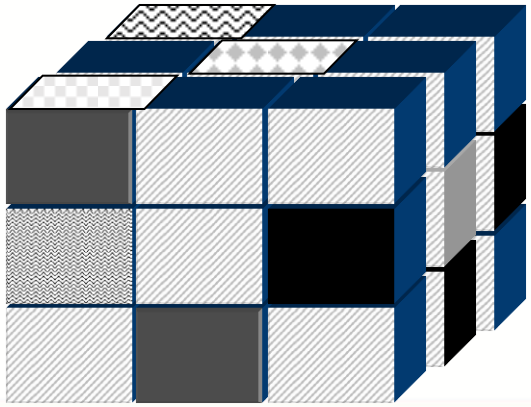
| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | | 8 |

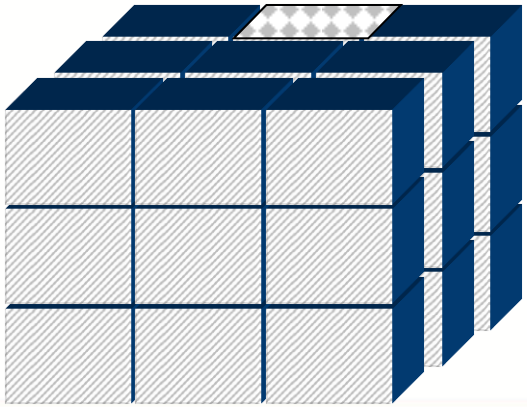
FWD

C

D

FW C





# Pencarian dengan Informasi (*Informed Search*)

- Tambahkan informasi pada spesifik domain untuk memilih jalur terbaik, pada saat melakukan  pencarian.
- Tentukan fungsi heuristik, h(n), untuk memperkirakan “goodness” dari sebuah node n.
- Secara khusus, h(n) = perkiraan cost (jarak)  dari lintasan dengan minimal cost dari node n ke node tujuan (***goal state***)
- Fungsi heuristik adalah perkiraan, berdasarkan informasi spesifik domain yang dapat dihitung dari state/node saat ini, seberapa dekat state tersebut dengan tujuan

# Greedy Search

- f(N) = h(N) → greedy best-first search

# Fungsi Heuristik (1)

- Fungsi h (N) yang memperkirakan cost jalur terpendek dari node N ke node tujuan.
- Contoh: 8-puzzle

| | | |
|---|---|---|
| 5 | | 8 |
| 4 | 2 | 1 |
| 7 | 3 | 6 |

N

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

goal

h(N) = jumlah tile yang tidak sesuai
= 6

# Fungsi Heuristik (2) – Manhattan Distance

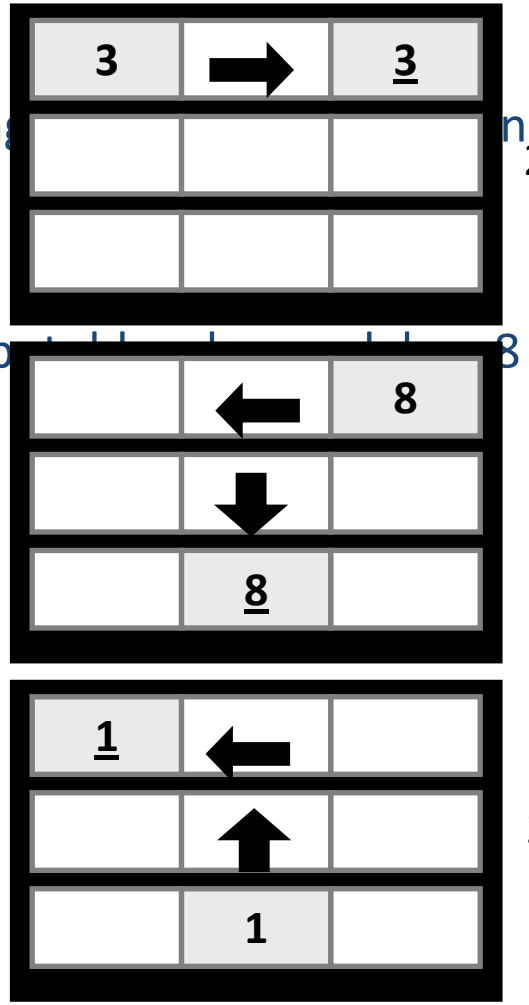
- Tile yang tidak sesuai dengan node tujuan adalah [illegible] 3 tile.
- Jadi fungsi heuristiknya akan mencapai 8.
- Dengan kata lain, heuristik memberitahukan bahwa solusi tersedia memb[illegible] langkah lagi.

Current State

| | | |
|---|---|---|
| 3 | 2 | 8 |
| 4 | 5 | 6 |
| 7 | 1 | |

Goal State

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |




Catatan : $h$(n) $h$(current state) = 8

# Fungsi Heuristik (2) – Manhattan Distance

- Fungsi h (N) yang memperkirakan cost jalur terpendek dari node N ke node tujuan.
- Contoh: 8-puzzle

| | | |
|---|---|---|
| 5 | | 8 |
| 4 | 2 | 1 |
| 7 | 3 | 6 |

N

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

goal

h(N) = jumlah jarak untuk tiap tile yang berbeda dengan tile pada node tujuan

= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1

= 13

# 8-Puzzle

| | | |
|---|---|---|
| 5 | | 8 |
| 4 | 2 | 1 |
| 7 | 3 | 6 |

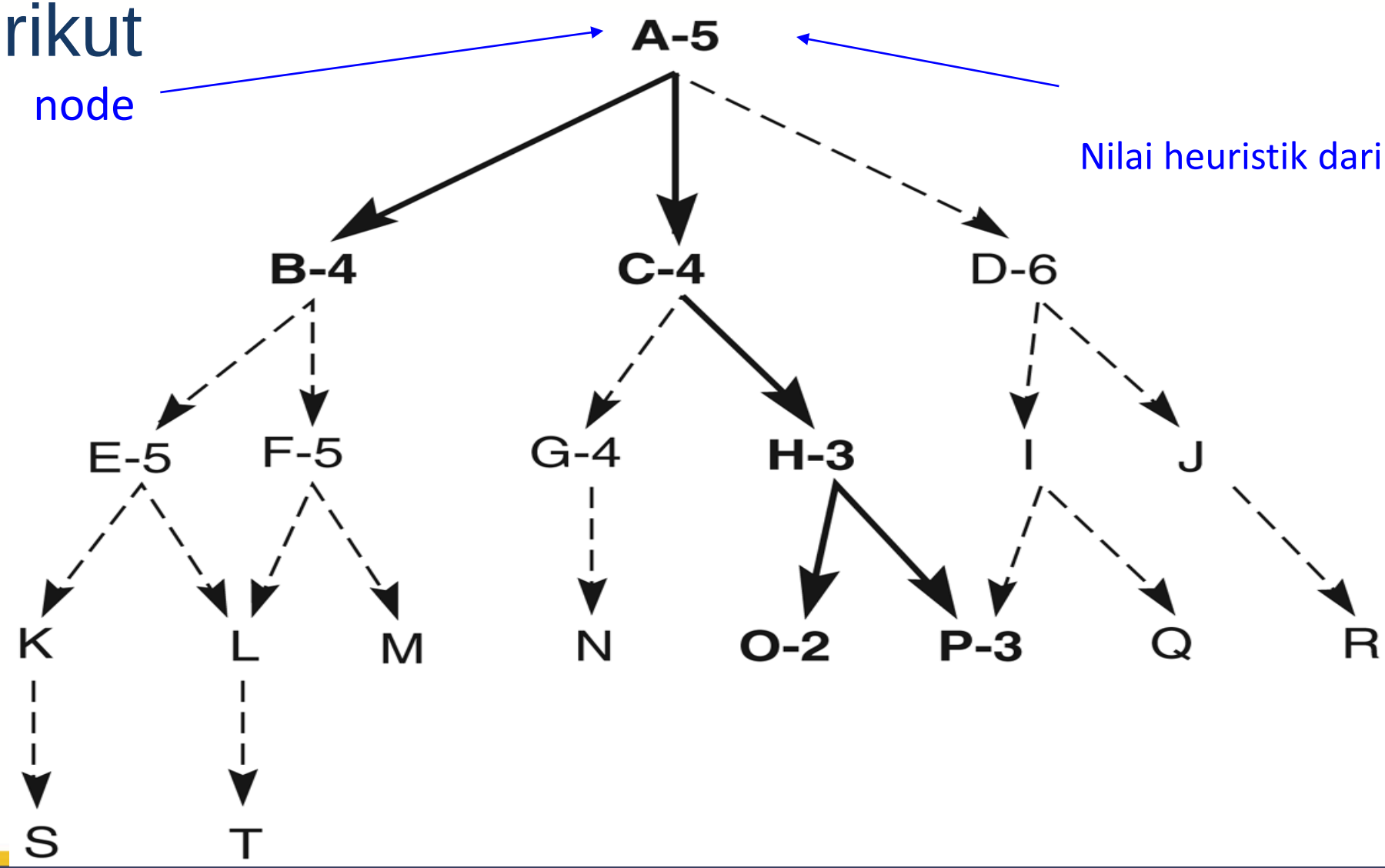
N

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

goal

- h1(N) = Jumlah tile yang berbeda = 6
- h2(N) = jumlah jarak untuk tiap tile yang berbeda dengan tile pada node tujuan = 13

# Pencarian Heuristik dengan Ruang Keadaan Berikut



# Algoritma Depth First Search (DFS)

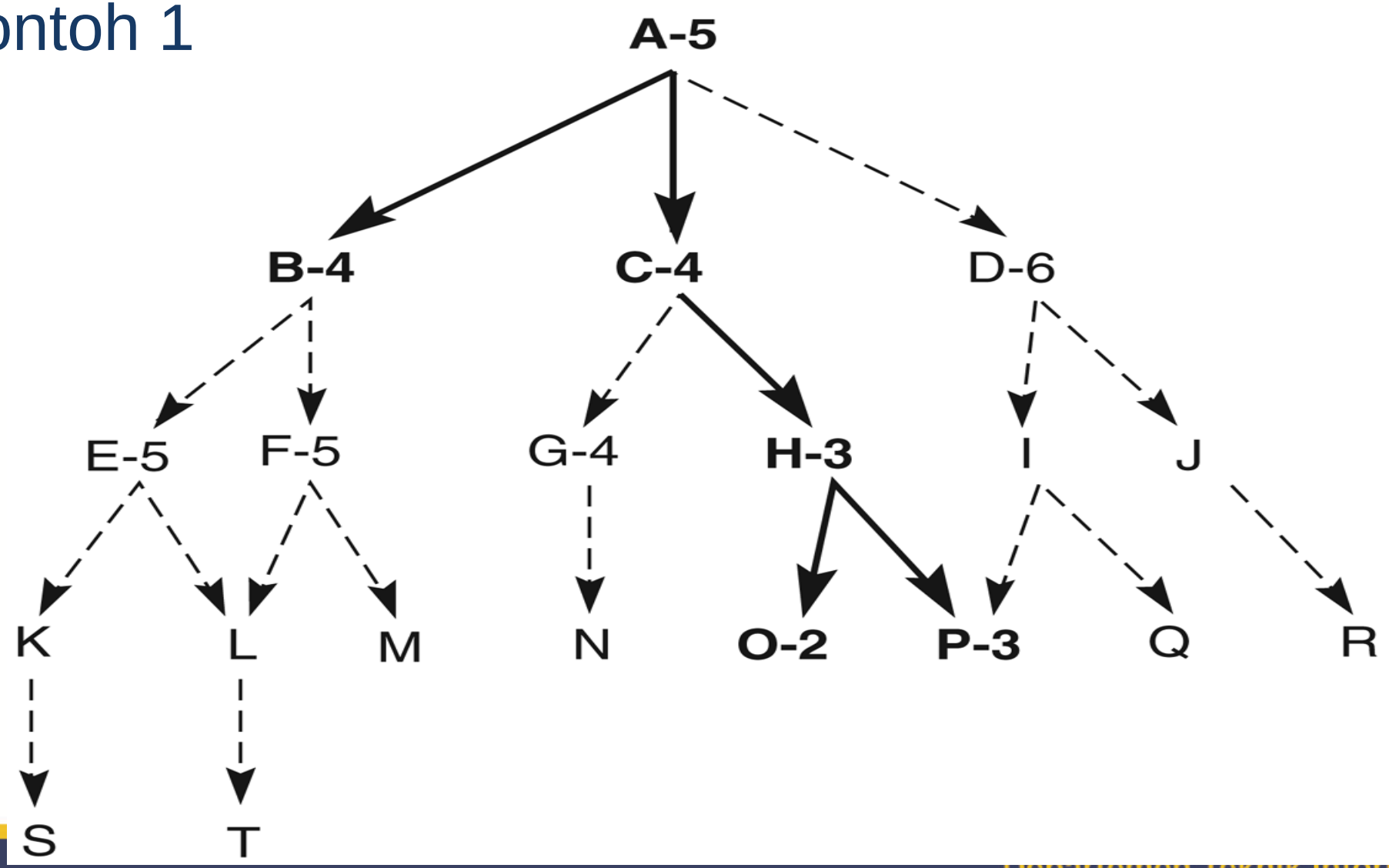
- Function depth_first_search;
- begin
  open := [Start];
  closed := [ ];
  while open ≠ [ ] do
  begin
  remove leftmost state from open, call it X;
  if X is a goal then return SUCCESS
  else begin
  generate children of X;
  put X on closed;
  discard remaining children of X if already on open or closed
  put remaining children on left end of open
  end
  end;
  return FAIL
  end.

# Algoritma Best First Search (Pengembangan Algoritma DFS)

- Function best_first_search;
- begin
  open := [Start];
  closed := [ ];
  while open ≠ [ ] do
  begin
  remove leftmost state from open, call it X;
  if X is a goal then return SUCCESS
  else begin
  generate children of X;
  assign each child their heuristic value;
  put X on closed;
  (discard remaining children of X if already on open or closed)
  put remaining children on open
  **sort open by heuristic merit (best leftmost)**
  end
  end;
  return FAIL
  end.

# Pencarian Heuristik dengan Ruang Keadaan Berikut

- Contoh 1

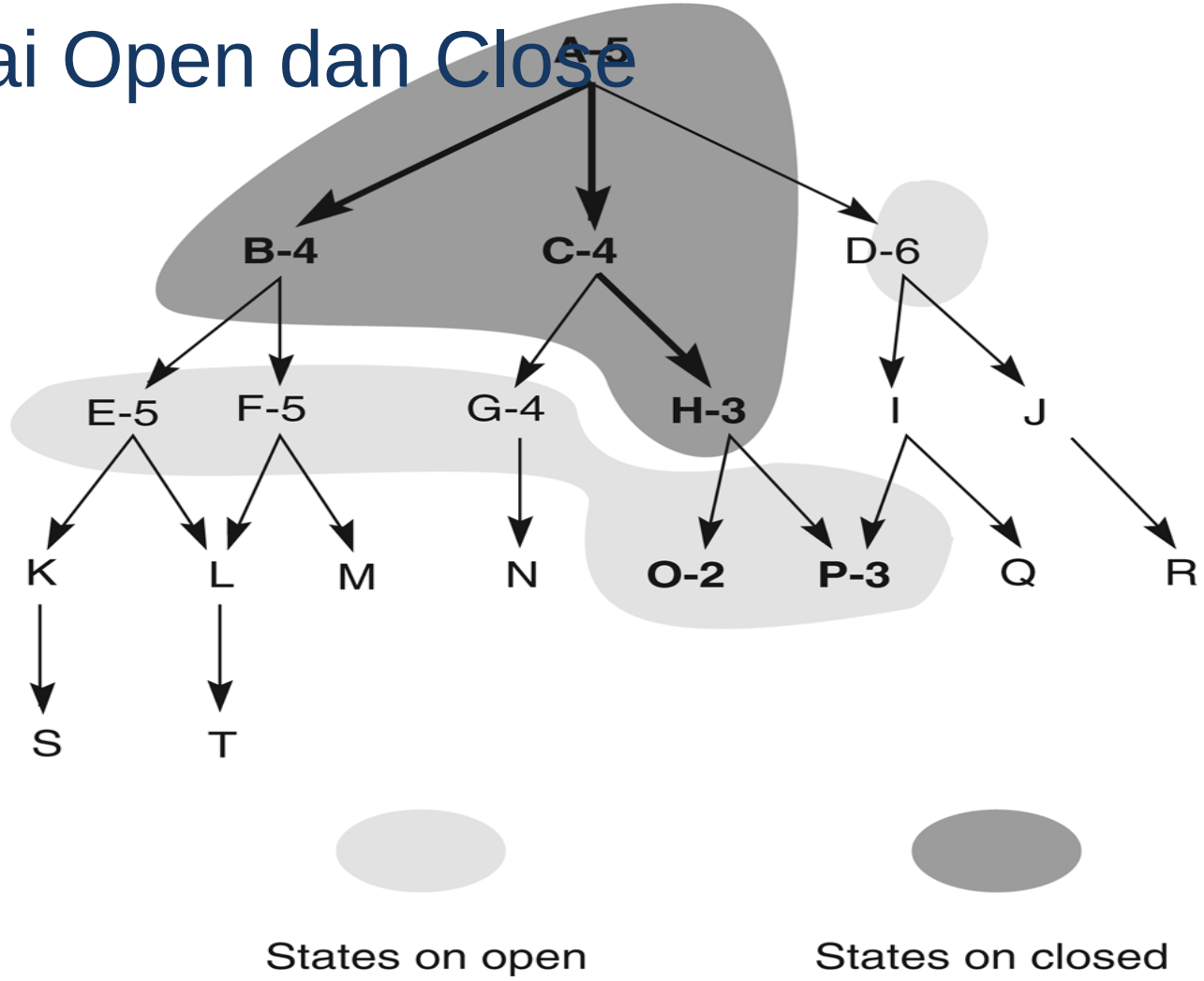


# Proses Trace untuk Algoritma Best First Search

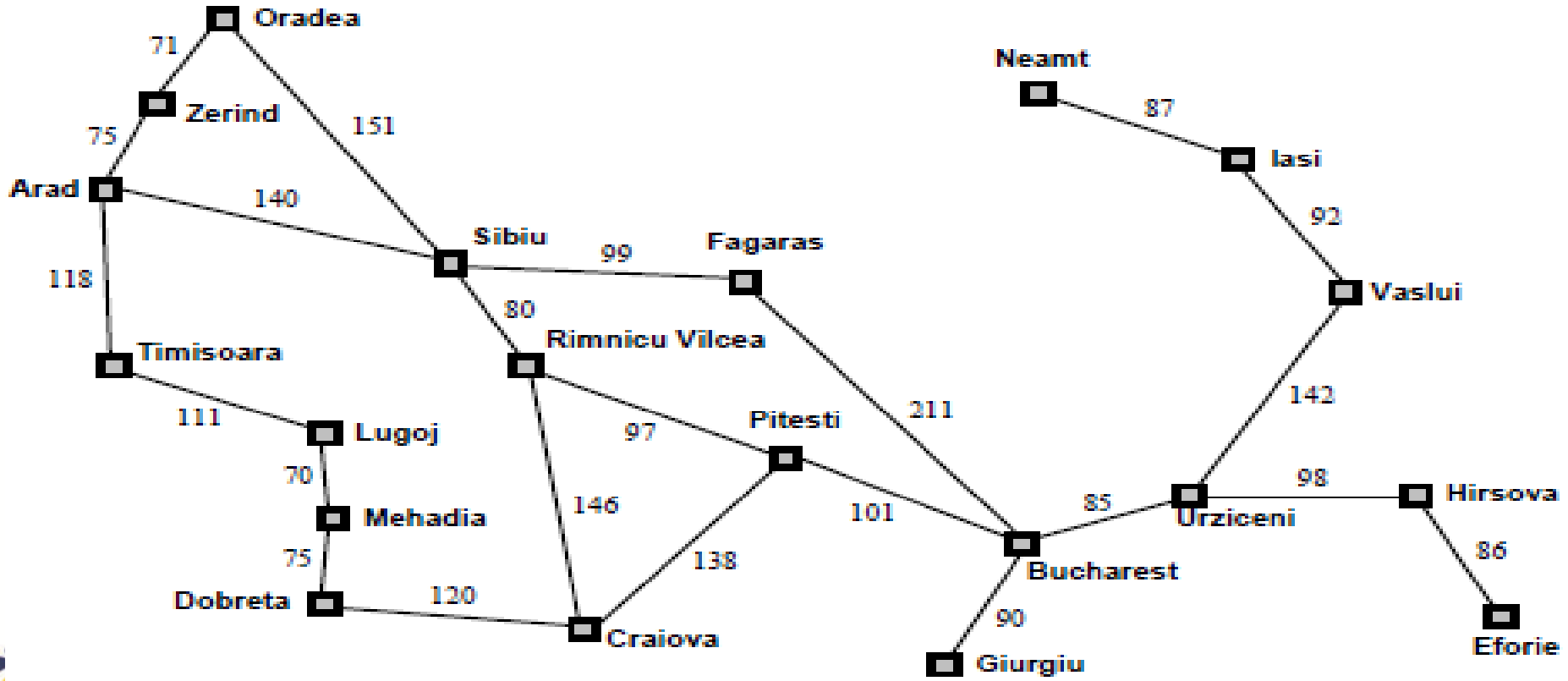
1. **open = [A5]; closed = [ ]**
2. **evaluate A5; open = [B4,C4,D6]; closed = [A5]**
3. **evaluate B4; open = [C4,E5,F5,D6]; closed = [B4,A5]**
4. **evaluate C4; open = [H3,G4,E5,F5,D6]; closed = [C4,B4,A5]**
5. **evaluate H3; open = [O2,P3,G4,E5,F5,D6]; closed = [H3,C4,B4,A5]**
6. **evaluate O2; open = [P3,G4,E5,F5,D6]; closed = [O2,H3,C4,B4,A5]**
7. **evaluate P3; the solution is found!**

# Pencarian Heuristik dengan Ruang Keadaan Berikut

- Menandai Open dan Close



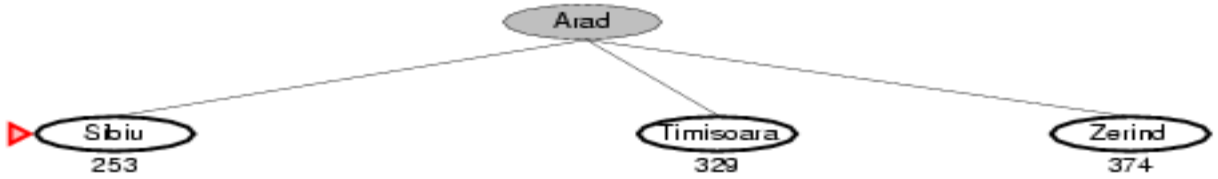
# Contoh 2. Algoritma Best First Search



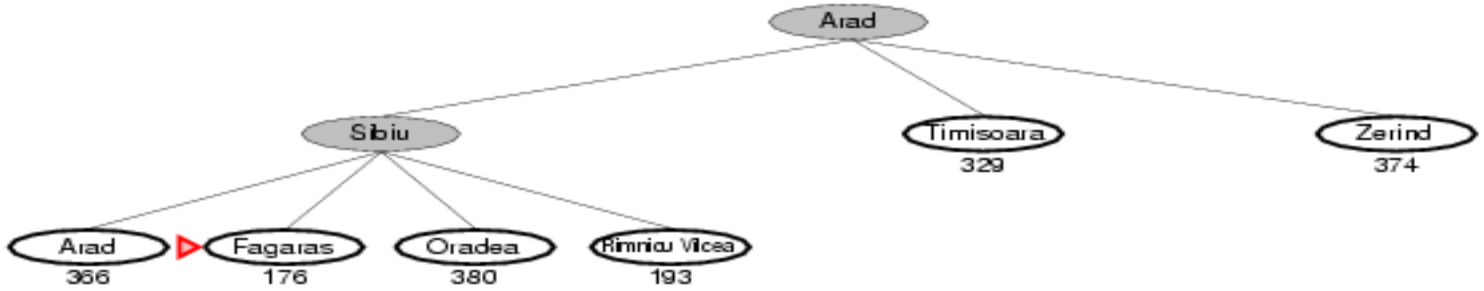

| Straight–line distance to Bucharest | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 178 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 98 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |

# Contoh 2. Algoritma Best First Search

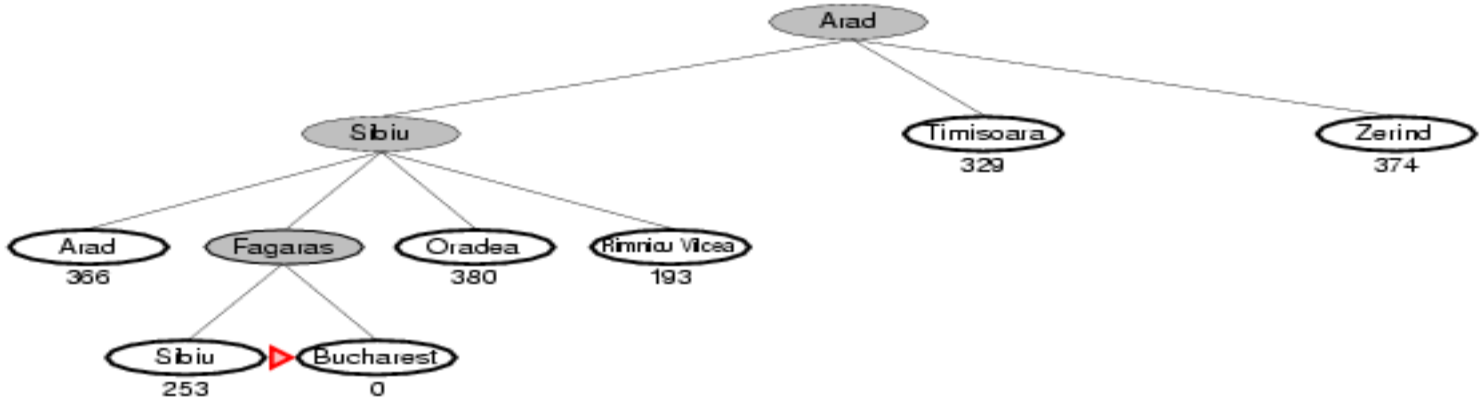
# Contoh 2. Algoritma Best First Search



# Contoh 2. Algoritma Best First Search



# Contoh 2. Algoritma Best First Search



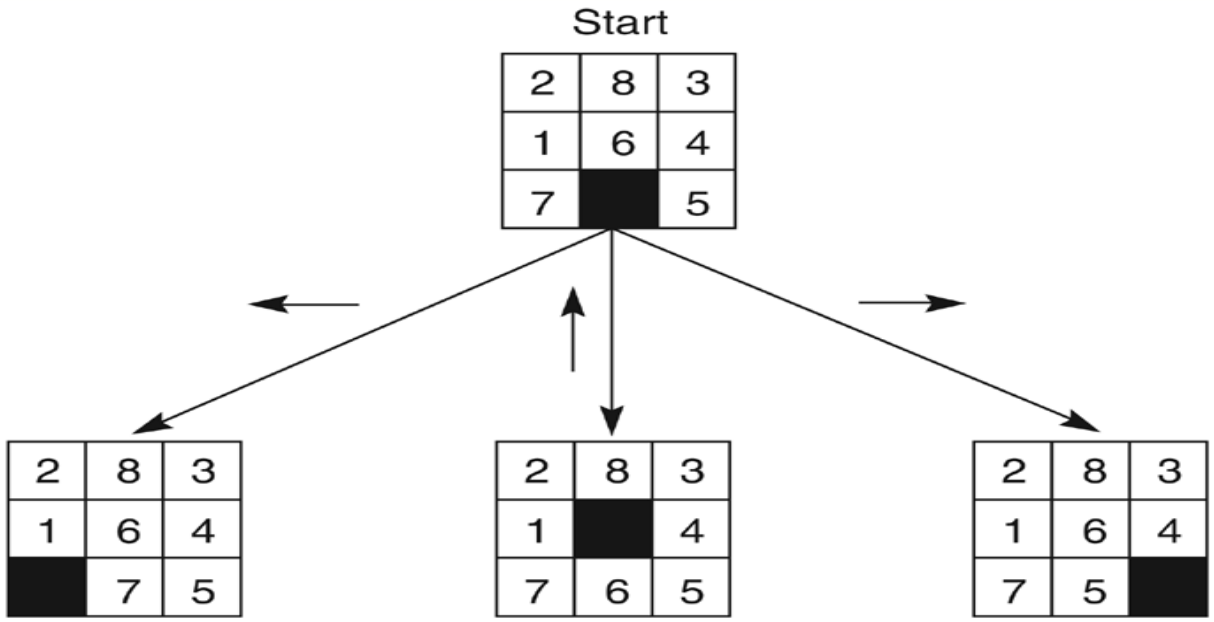
# Analisa Best First Search

- Kelebihan
  - Butuh memori kecil
  - Menemukan solusi tanpa harus menguji lebih banyak lagi
- Kelemahan
  - Mungkin terjebak pada local optima

# Algoritma A*

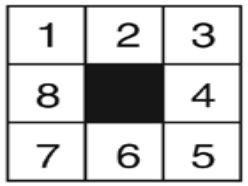
- A * (A star) adalah bentuk pencarian Best-First yang paling banyak dikenal
  - Mengevaluasi node dengan menggabungkan g (n) dan h (n)
  - f (n) = g (n) + h (n)
  - Dimana
    - g(n) = cost dari node asal ke node saat ini yaitu n
    - h(n) = perkiraan cost dengan lintasan terdekat dari node n ke node tujuan
    - f(n) = perkiraan total cost pada lintasan yang melalui node n

# Fungsi Heuristik pada 8-Puzzle



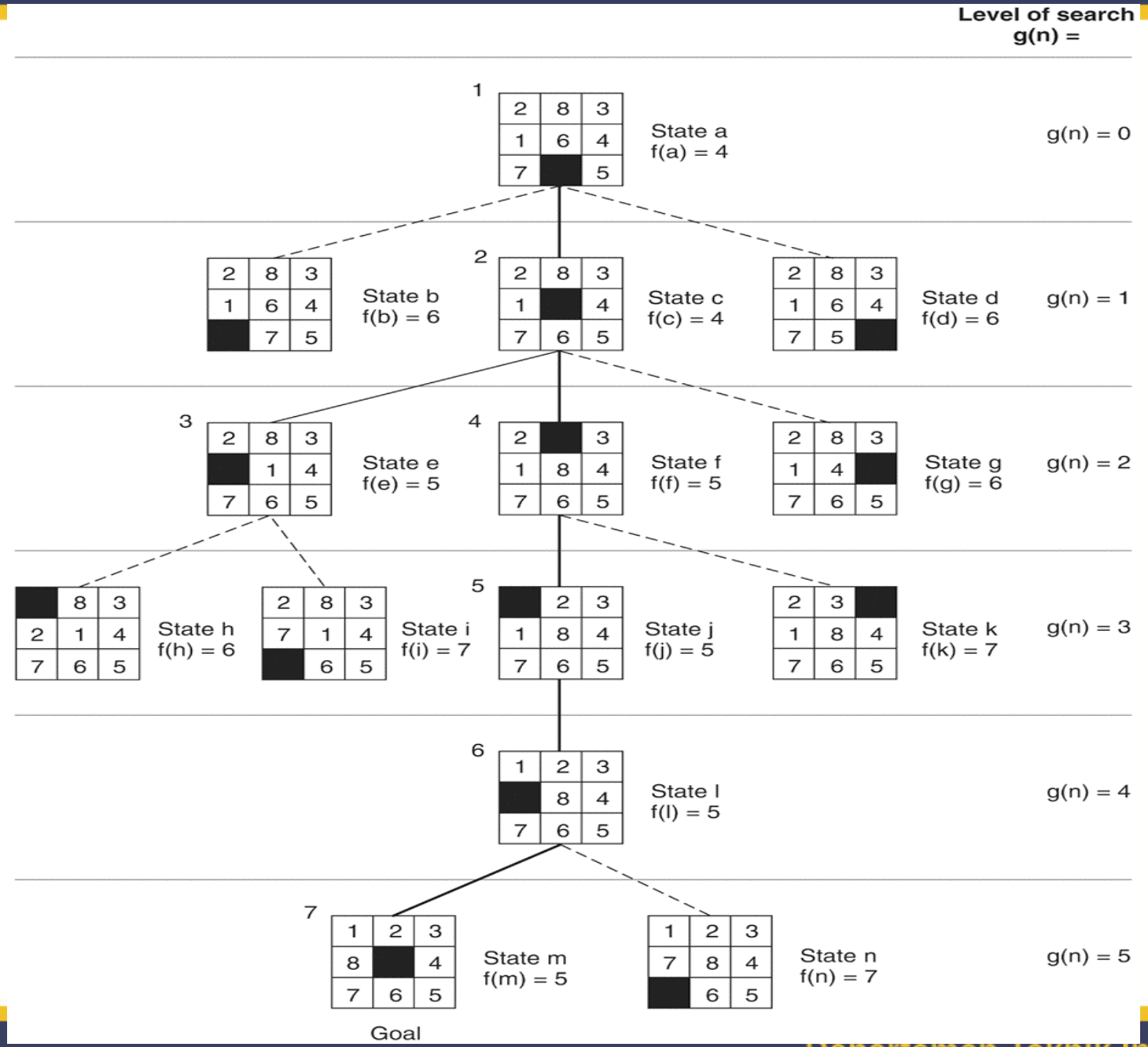

**Values of f(n) for each state,** **6** **4** **6**

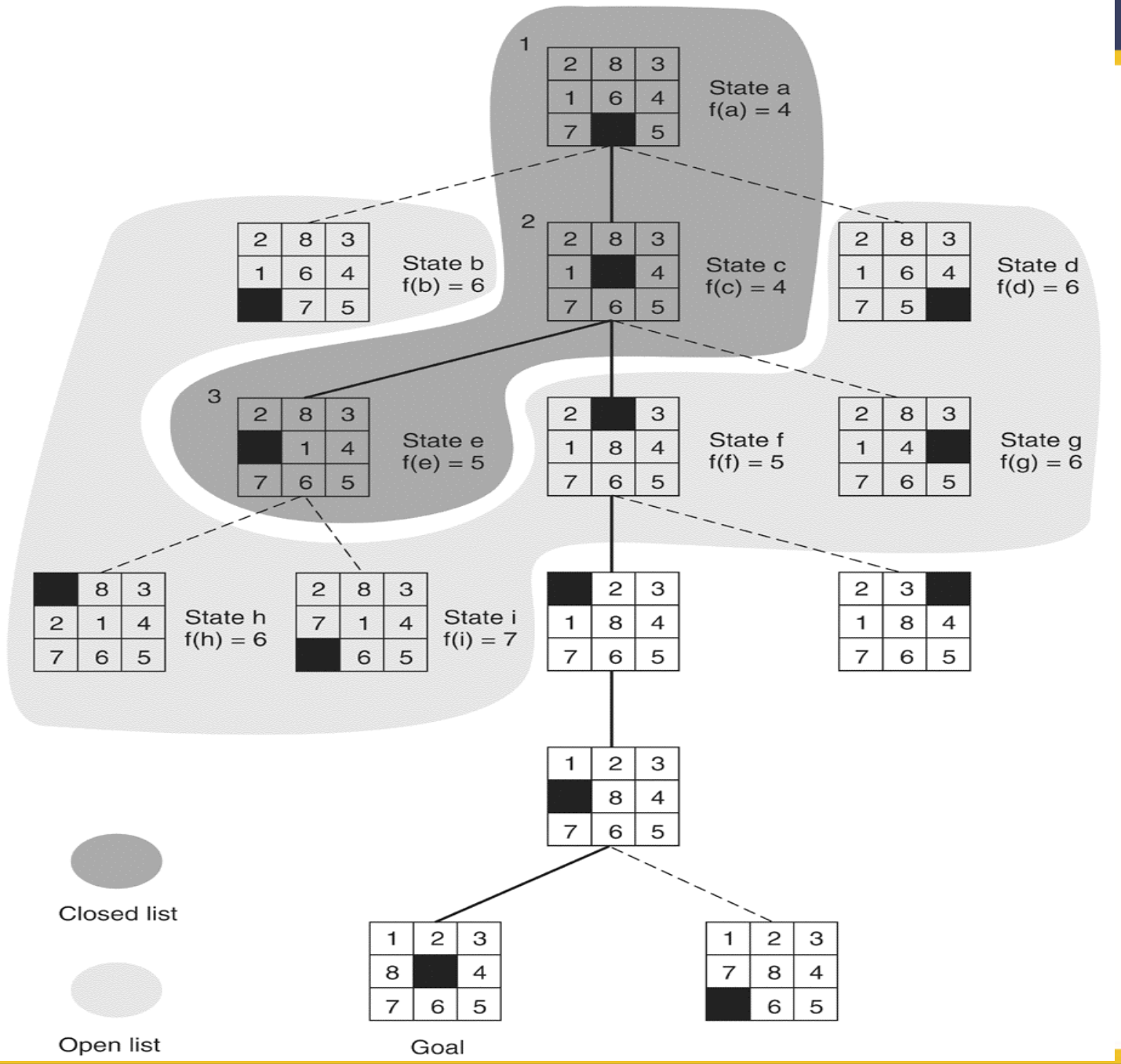
where:
f(n) = g(n) + h(n),
g(n) = actual distance from n to the start state, and
h(n) = number of tiles out of place.



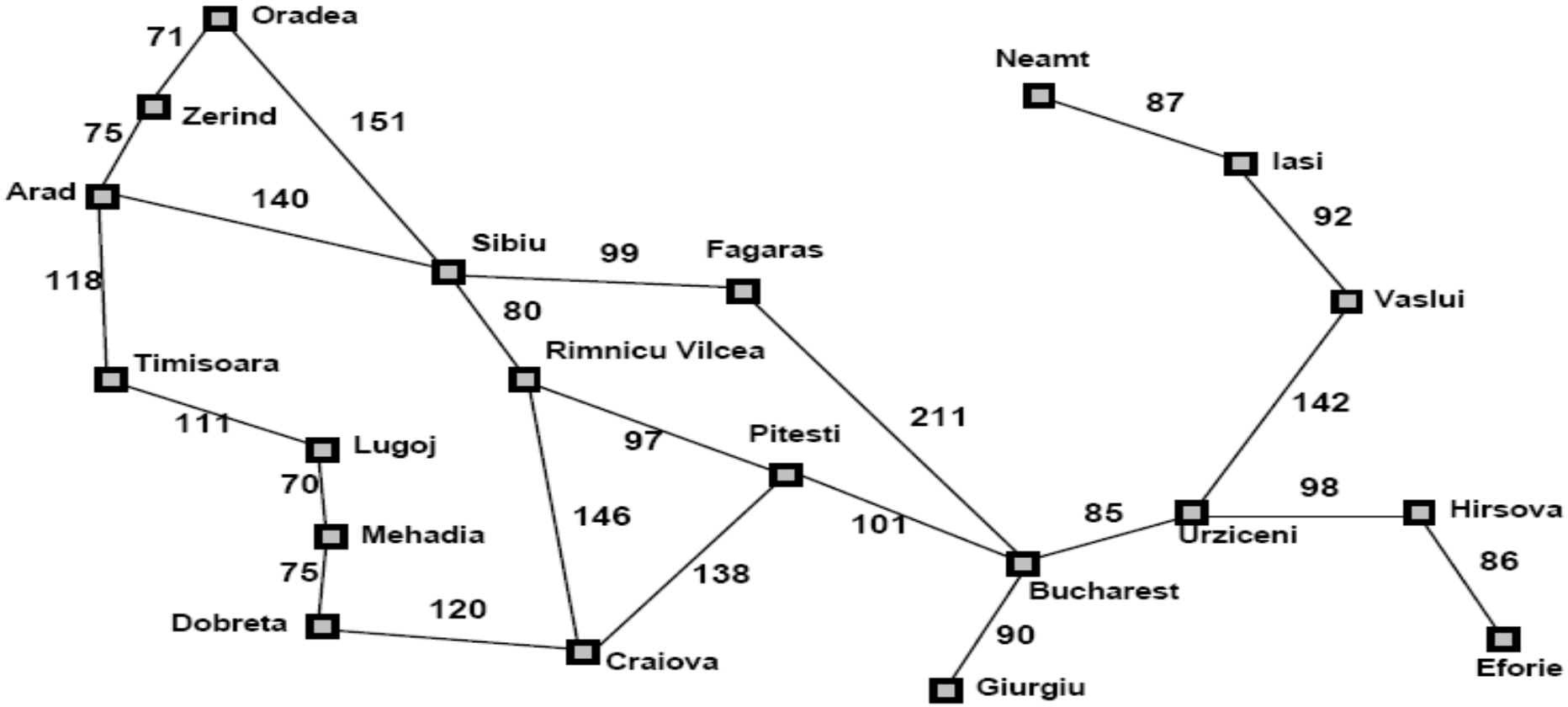
# Proses Trace pada Algoritma A*

1. **open = [a4];**
   **closed = [ ]**
2. **open = [c4, b6, d6];**
   **closed = [a4]**
3. **open = [e5, f5, b6, d6, g6];**
   **closed = [a4, c4]**
4. **open = [f5, h6, b6, d6, g6, l7];**
   **closed = [a4, c4, e5]**
5. **open = [ j5, h6, b6, d6, g6, k7, l7];**
   **closed = [a4, c4, e5, f5]**
6. **open = [l5, h6, b6, d6, g6, k7, l7];**
   **closed = [a4, c4, e5, f5, j5]**
7. **open = [m5, h6, b6, d6, g6, n7, k7, l7];**
   **closed = [a4, c4, e5, f5, j5, l5]**
8. **success, m = goal!**


Level of search
g(n) =
1
State a
f(a) = 4
g(n) = 0
2
State b
f(b) = 6
State c
f(c) = 4
State d
f(d) = 6
g(n) = 1
3
State e
f(e) = 5
4
State f
f(f) = 5
State g
f(g) = 6
g(n) = 2
State h
f(h) = 6
State i
f(i) = 7
5
State j
f(j) = 5
State k
f(k) = 7
g(n) = 3
6
State l
f(l) = 5
g(n) = 4
7
State m
f(m) = 5
State n
f(n) = 7
g(n) = 5
Goal

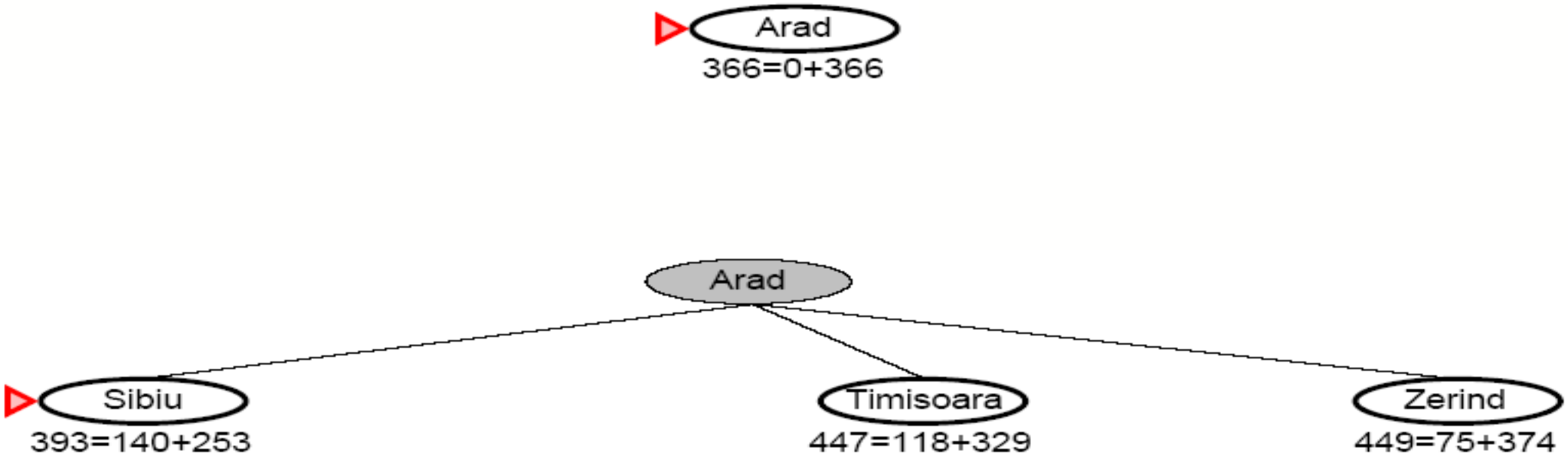

1
2 8 3
1 6 4
7 5
State a
f(a) = 4
2
2 8 3
1 6 4
7 5
State b
f(b) = 6
2 8 3
1 4
7 6 5
State c
f(c) = 4
2 8 3
1 6 4
7 5
State d
f(d) = 6
3
2 8 3
1 4
7 6 5
State e
f(e) = 5
2 3
1 8 4
7 6 5
State f
f(f) = 5
2 8 3
1 4
7 6 5
State g
f(g) = 6
8 3
2 1 4
7 6 5
State h
f(h) = 6
2 8 3
7 1 4
6 5
State i
f(i) = 7
2 3
1 8 4
7 6 5
2 3
1 8 4
7 6 5
1 2 3
8 4
7 6 5
Closed list
Open list
1 2 3
8 4
7 6 5
Goal
1 2 3
7 8 4
6 5

# Contoh 2

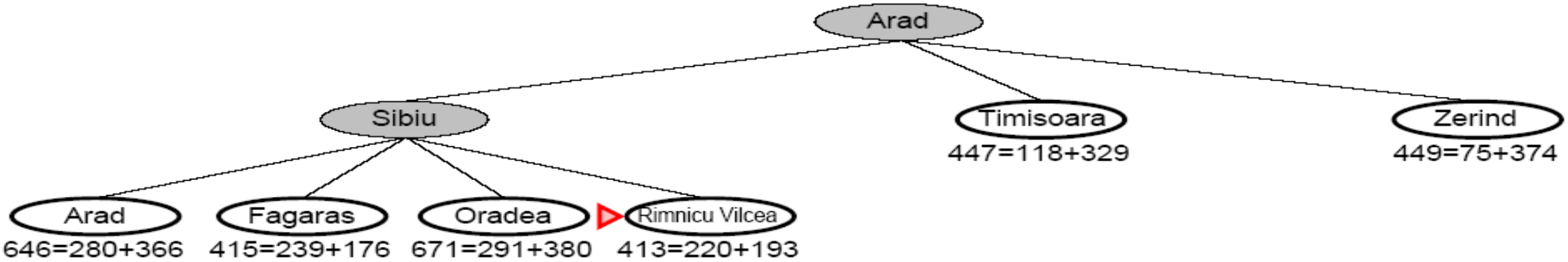



| Straight–line distance to Bucharest | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 178 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 98 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |

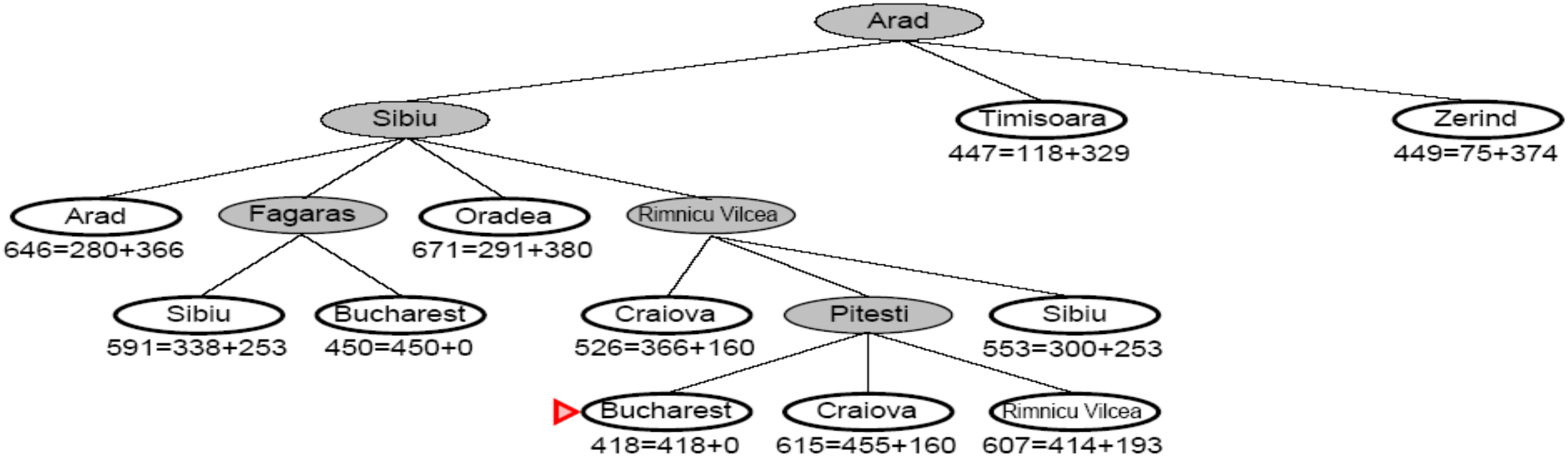
# A* Search


Arad
366=0+366
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374

# A* Search

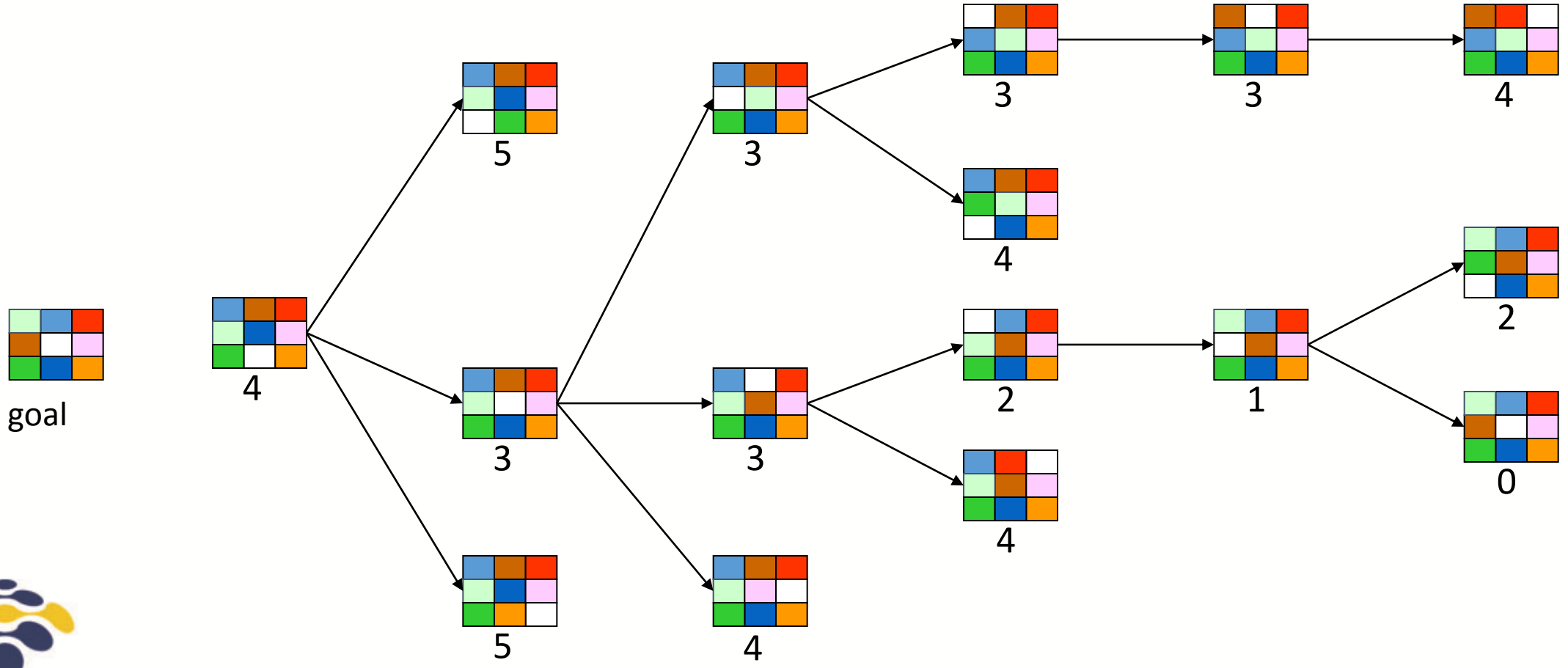

Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193

# A* Search


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

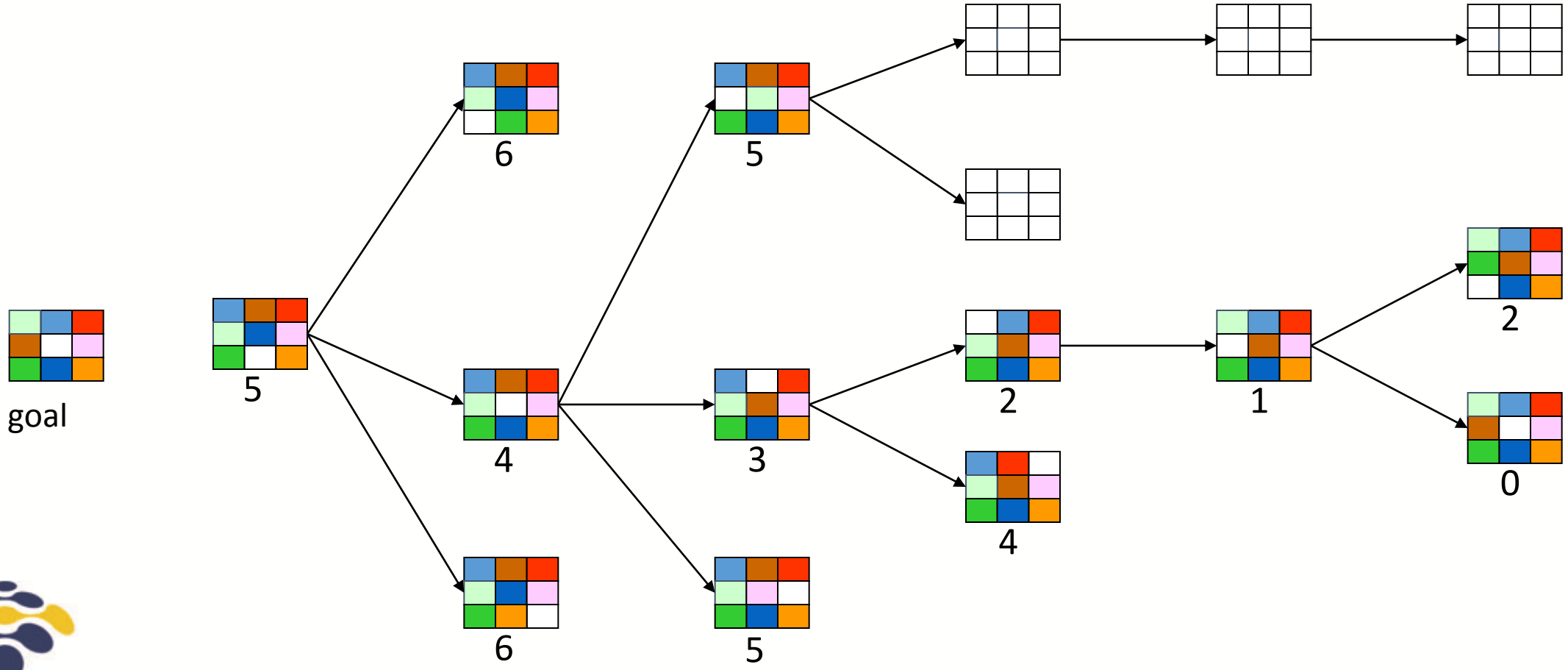
# Algoritma Best First Search (8-Puzzle)

f(N) = h(N) = jumlah tile yang berbeda



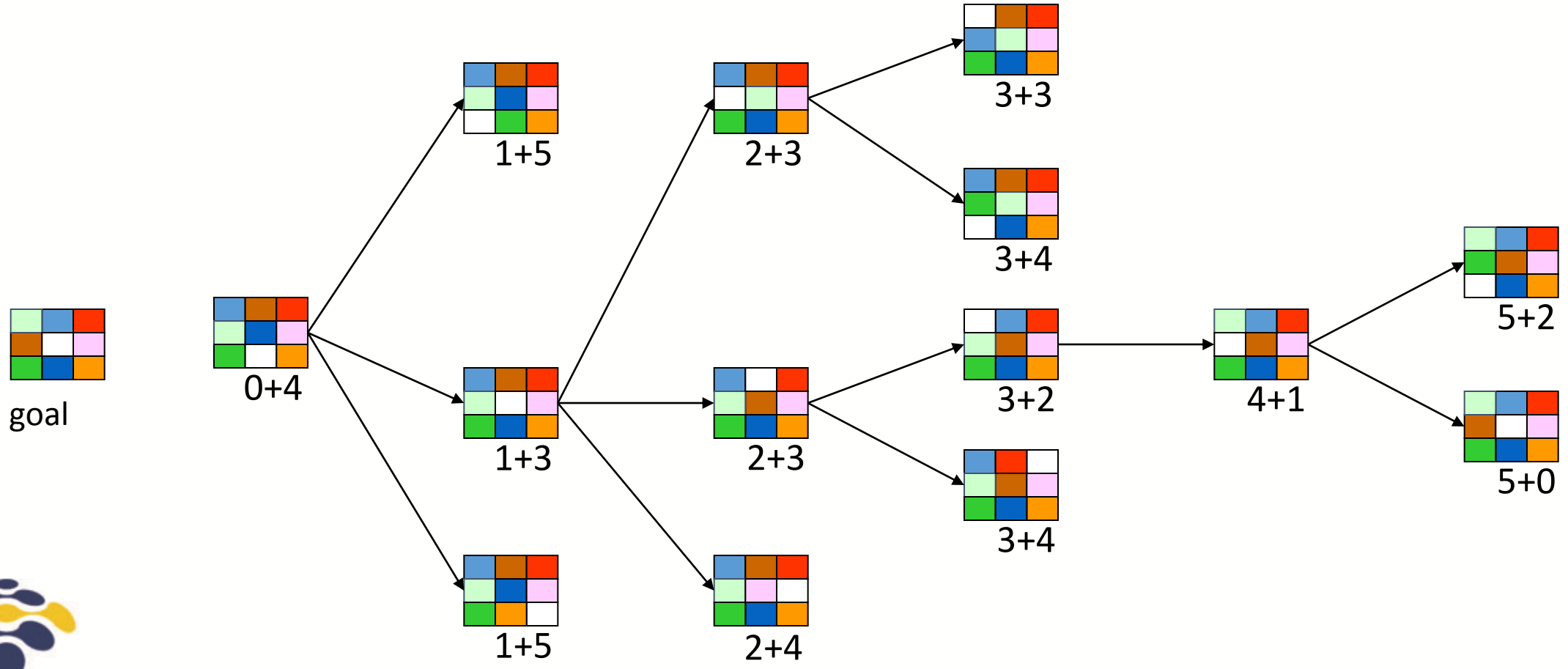
# Algoritma Best First Search (8-Puzzle)

f(N) = h(N) = ∑jarak tile yang berbeda dengan tile yang terdapat pada node tujuan



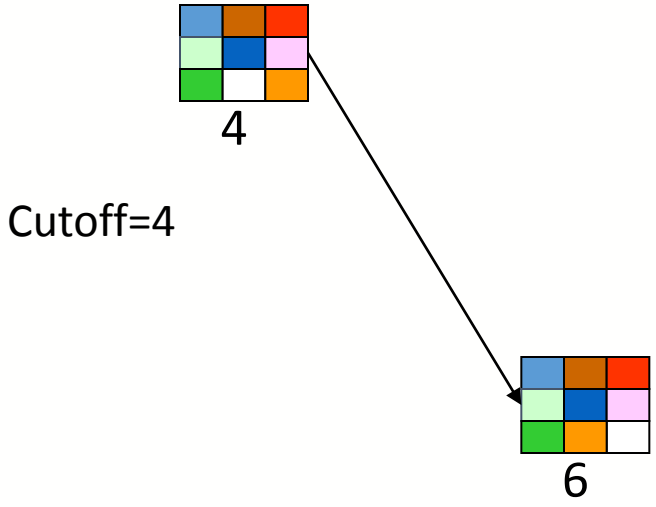
# Algoritma A* (8-Puzzle)

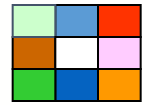
f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda



# Algoritma A* (8-Puzzle) - Dengan Cutoff 4

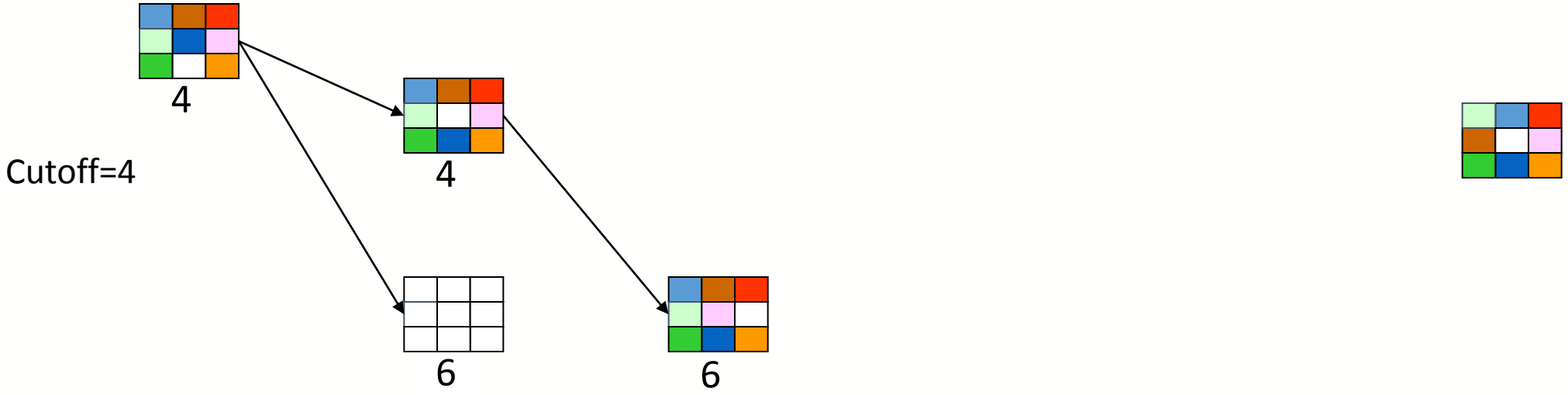
f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda



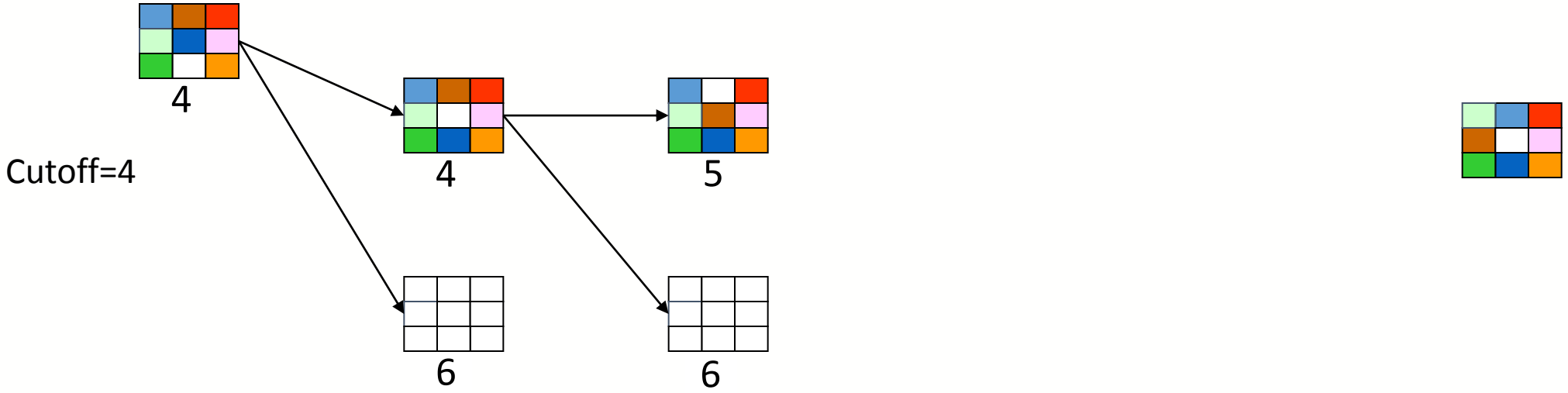
# Algoritma A* (8-Puzzle) - Dengan Cutoff 5

f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda



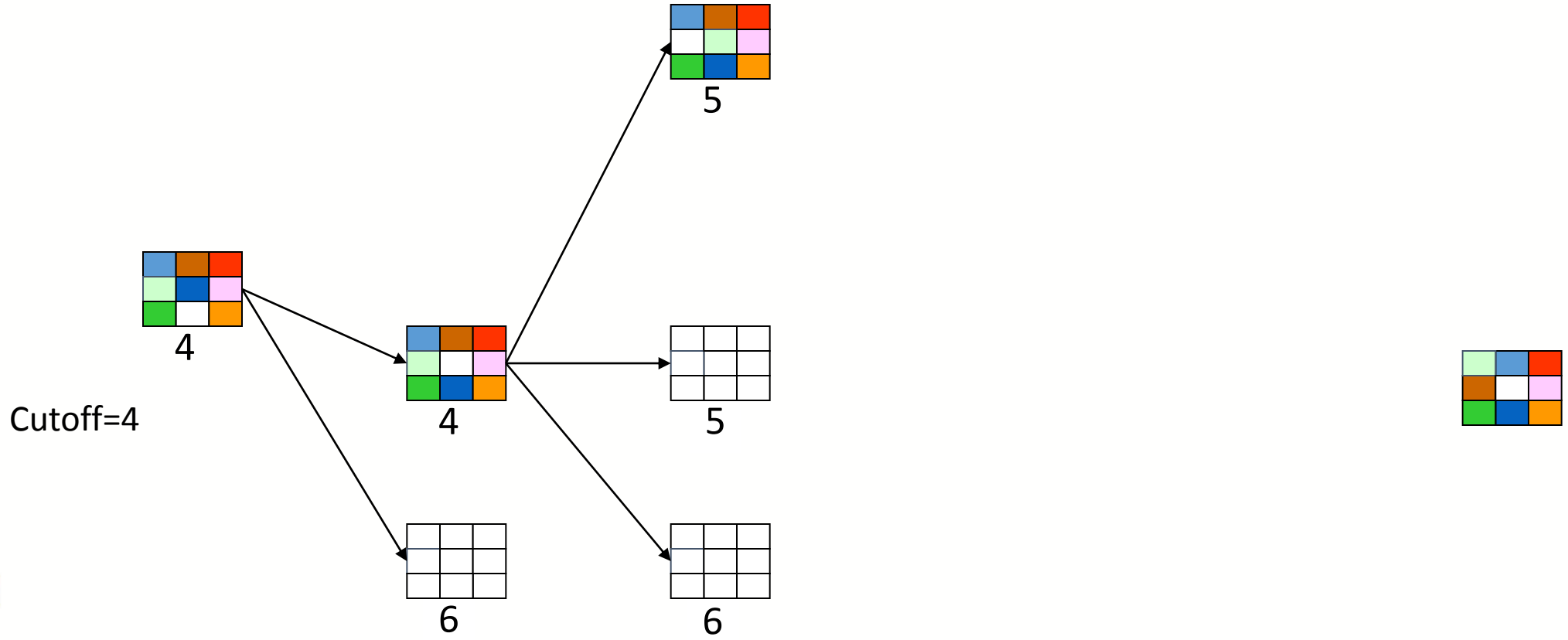
# Algoritma A* (8-Puzzle) - Dengan Cutoff 4

f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda



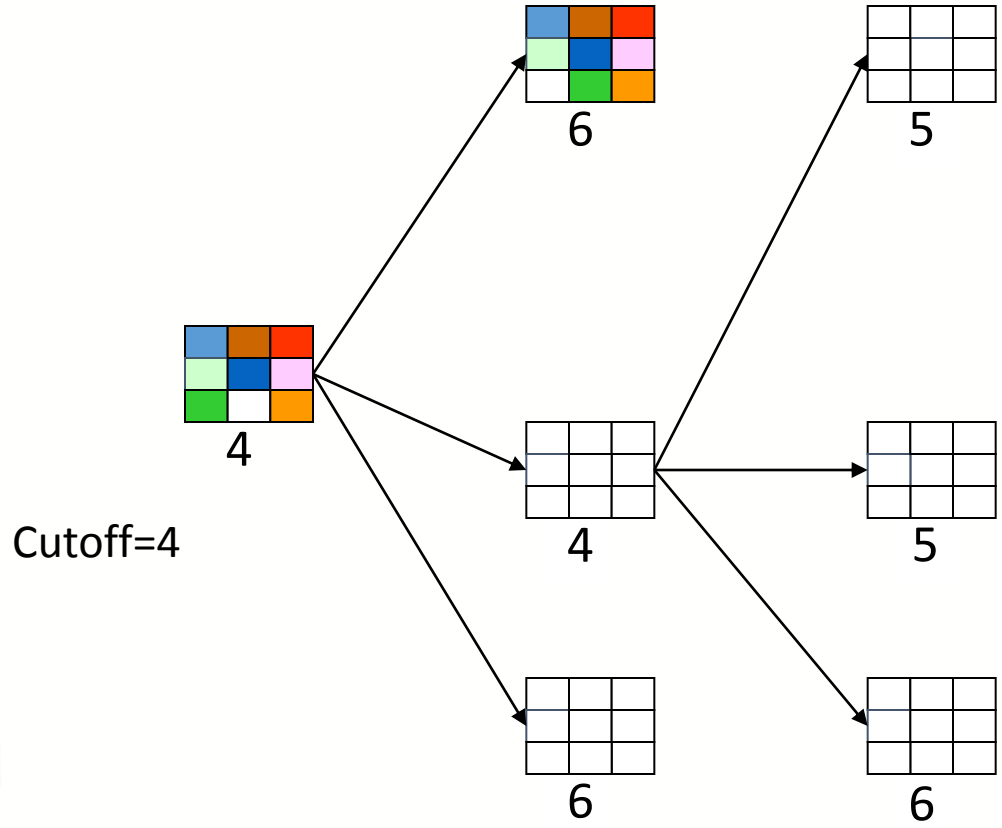
# Algoritma A* (8-Puzzle) - Dengan Cutoff 4

f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda

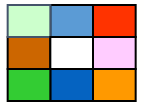


# Algoritma A* (8-Puzzle) - Dengan Cutoff 4

f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda

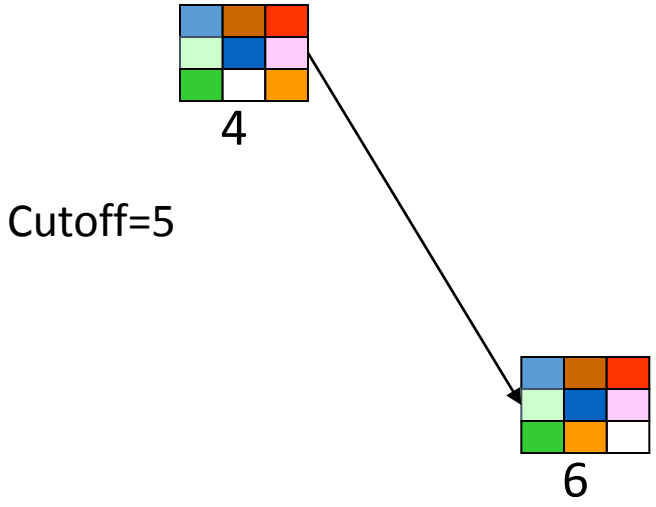


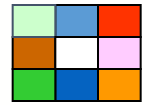

Cutoff=4



# Algoritma A* (8-Puzzle) - Dengan Cutoff 5

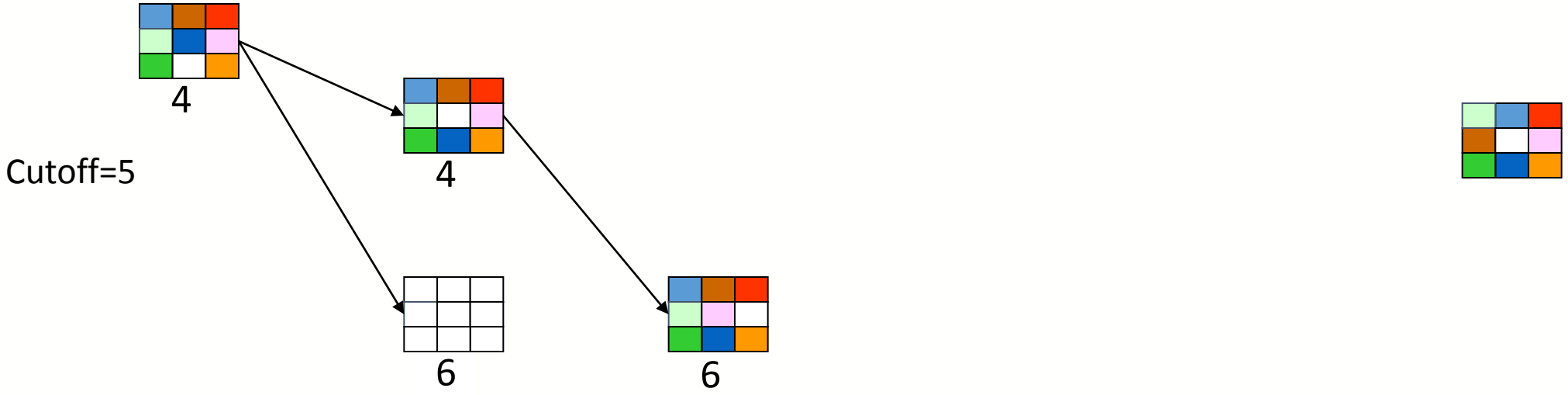
f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda



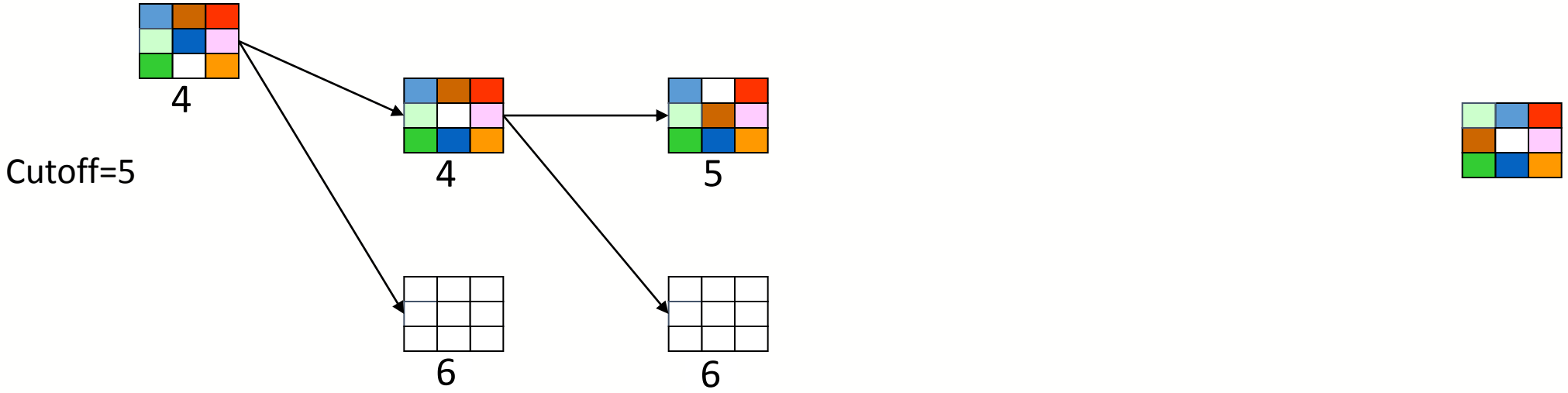
# Algoritma A* (8-Puzzle) - Dengan Cutoff 5

f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda



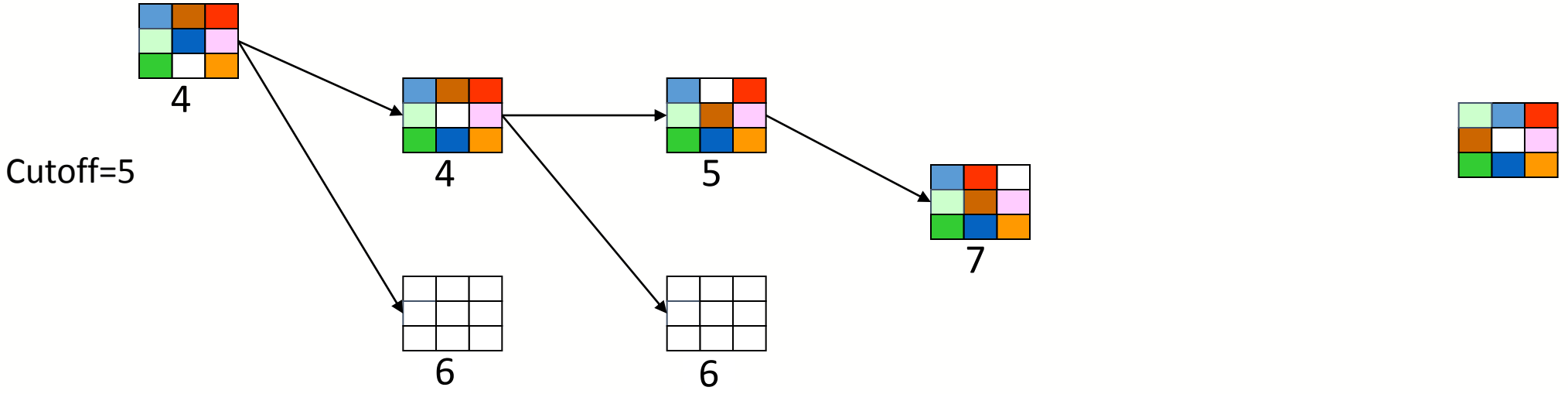
# Algoritma A* (8-Puzzle) - Dengan Cutoff 5

f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda



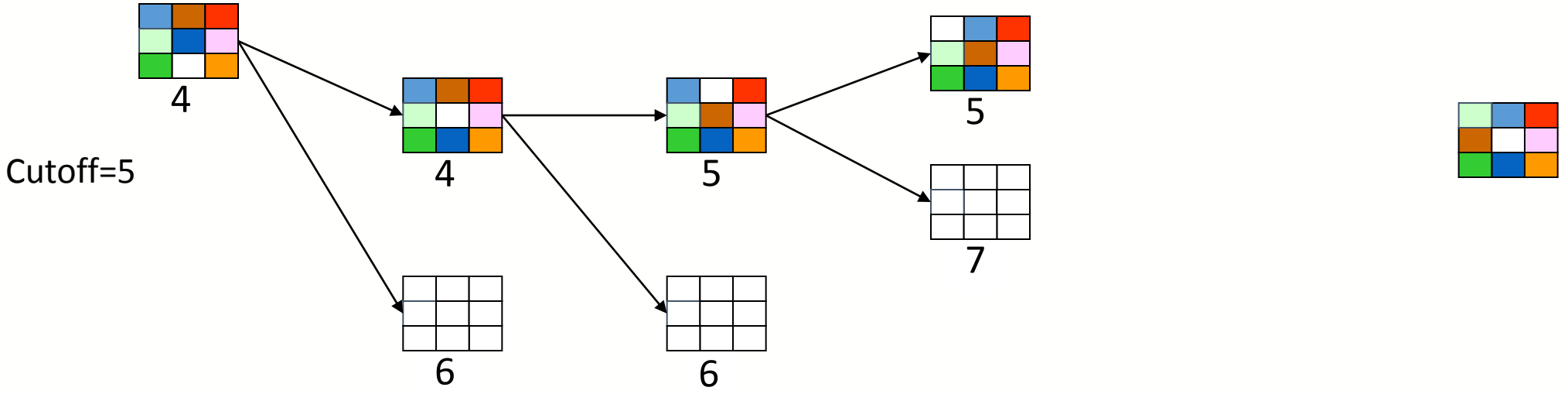
# Algoritma A* (8-Puzzle) - Dengan Cutoff 5

f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda



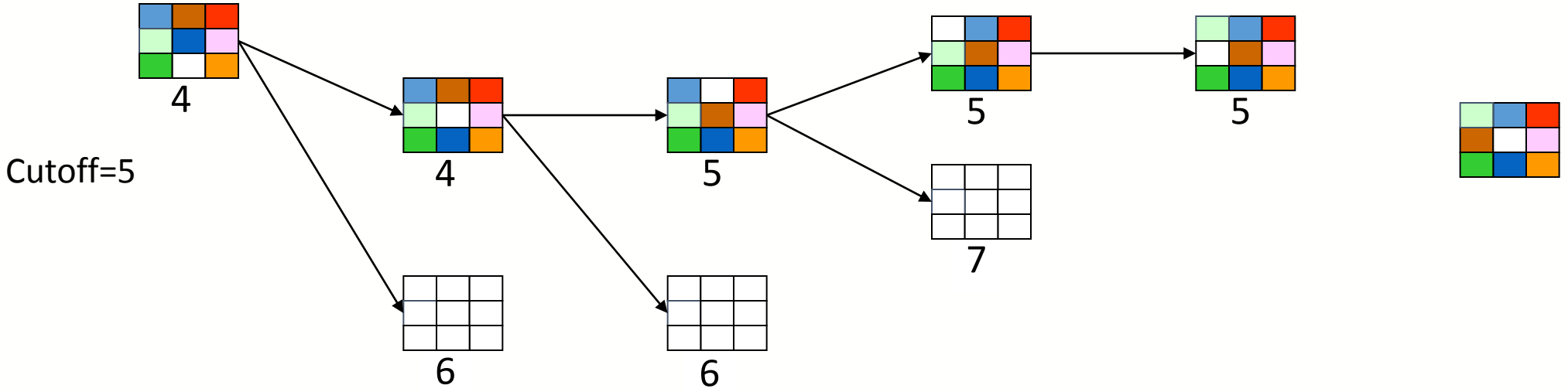
# Algoritma A* (8-Puzzle) - Dengan Cutoff 5

f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda



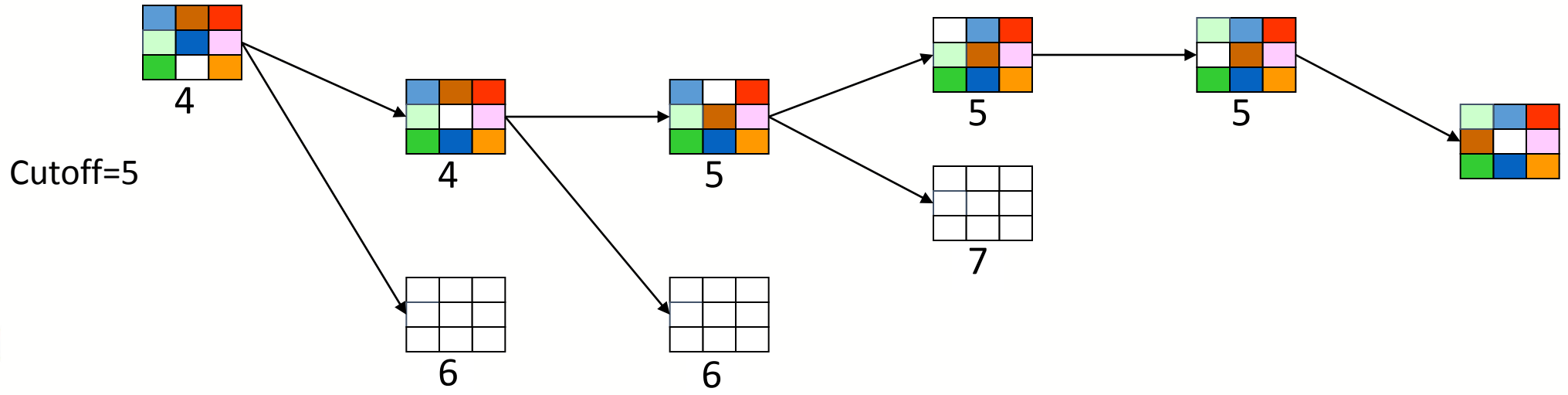
# Algoritma A* (8-Puzzle) - Dengan Cutoff 5

f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda



# Algoritma A* (8-Puzzle) - Dengan Cutoff 5

f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda



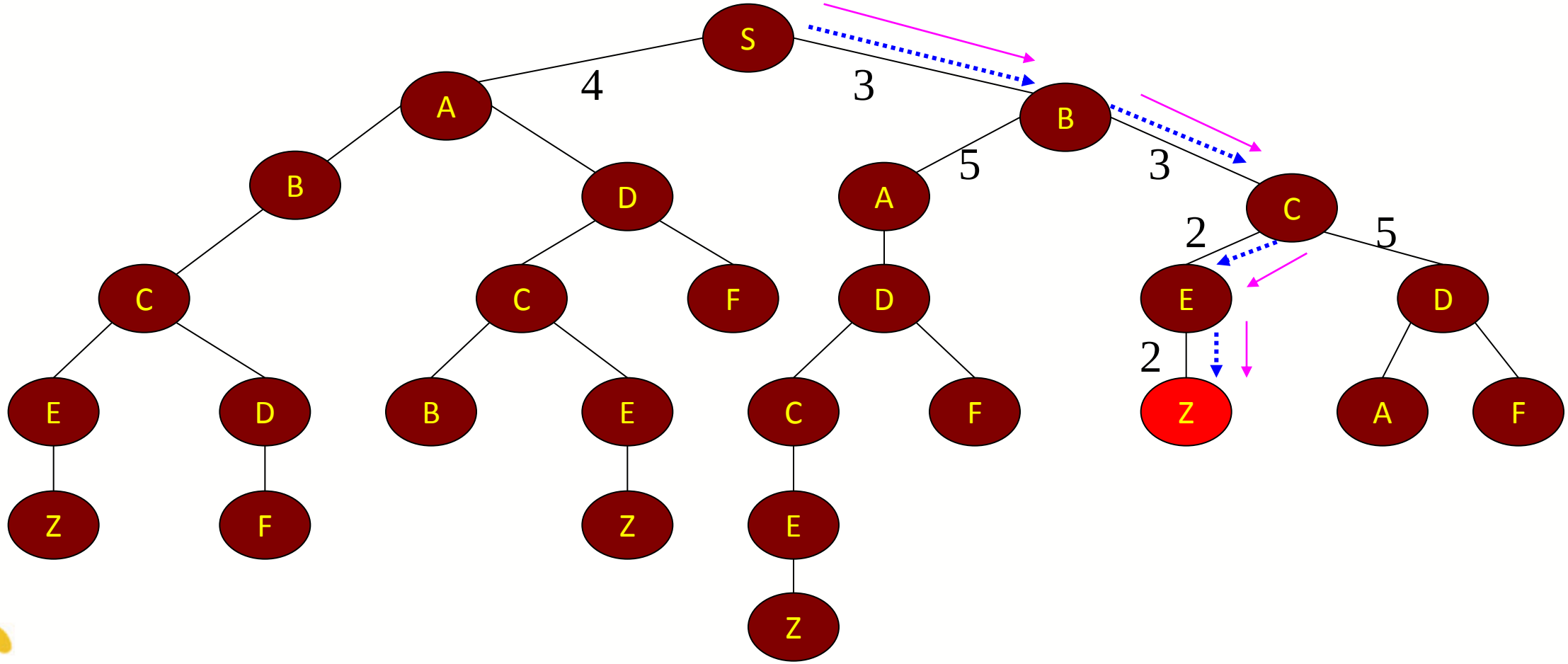
# Mengenai Algoritma Heuristik

- Fungsi Heuristik untuk mengarahkan pencarian ke lintasan yang menjanjikan
- Waktu yang dihabiskan untuk menghitung fungsi heuristik akan mendapatkan pencarian yang lebih baik
- Fungsi heuristik dapat menyelesaikan permasalahan, dapat mengarahkan pencarian menuju ke node tujuan
- Menentukan node mana yang akan diperluas disebut meta-reasoning
- Heuristik mungkin tidak selalu terlihat seperti angka dan mungkin melibatkan sejumlah besar knowledge

# Kapan Menggunakan Algoritma Pencarian?

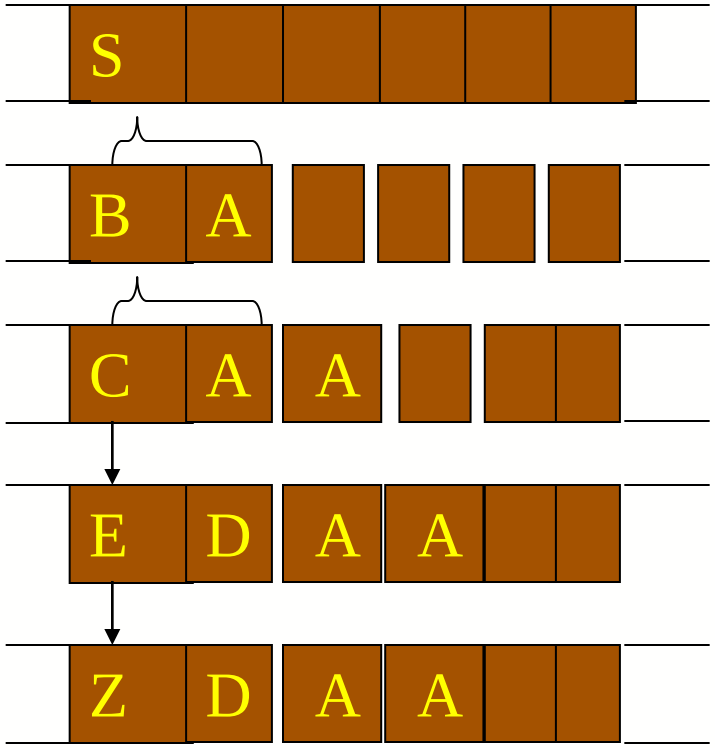
- Ruang pencarian kecil
  - Tidak ada teknik lain yang tersedia, atau
  - Tidak sepadan dengan usaha untuk mengembangkan teknik yang lebih efisien
- Ruang pencariannya besar, dan
  - Tidak ada teknik lain yang tersedia, dan
  - Terdapat heuristik “good“

# Hill climbing



# Hill climbing

Mirip dengan Depth First Search, hanya saja pemilihan node anak disertai dengan aturan

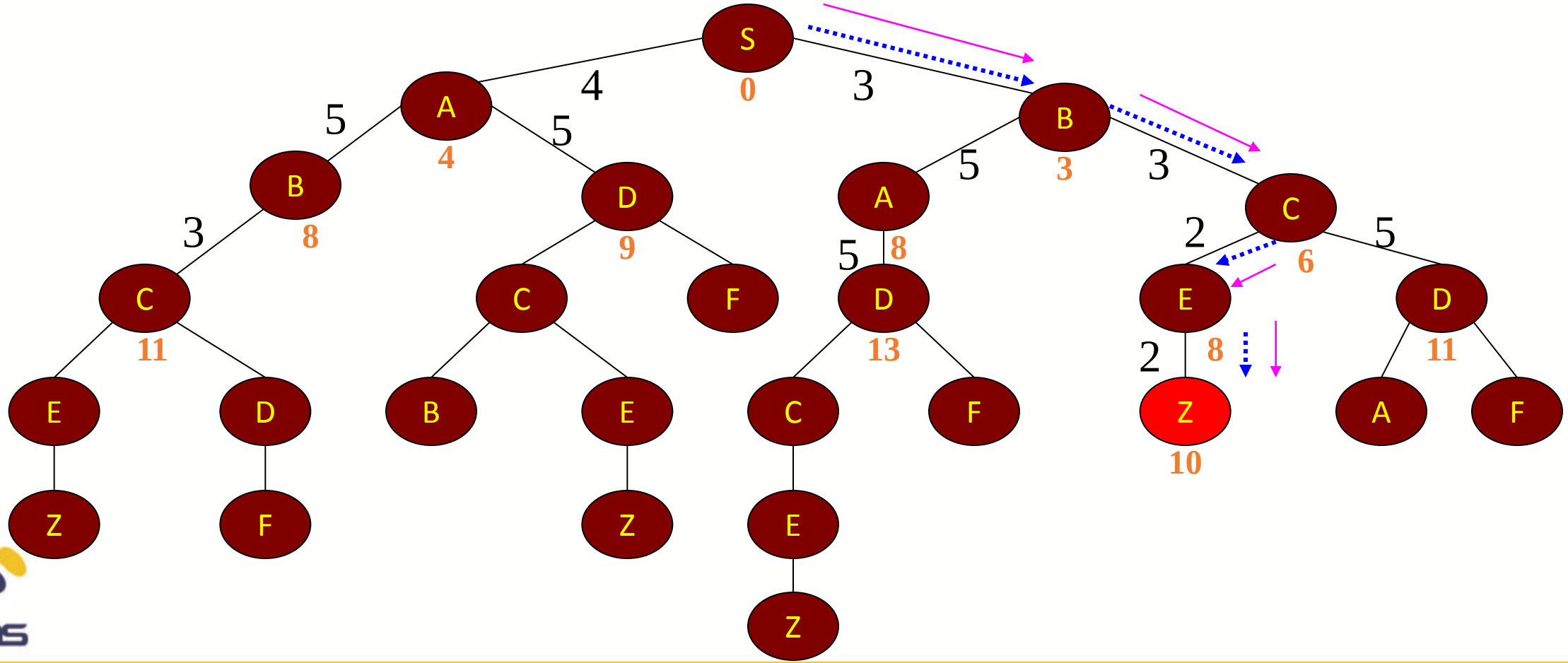



Rule: yang paling kecil jaraknya

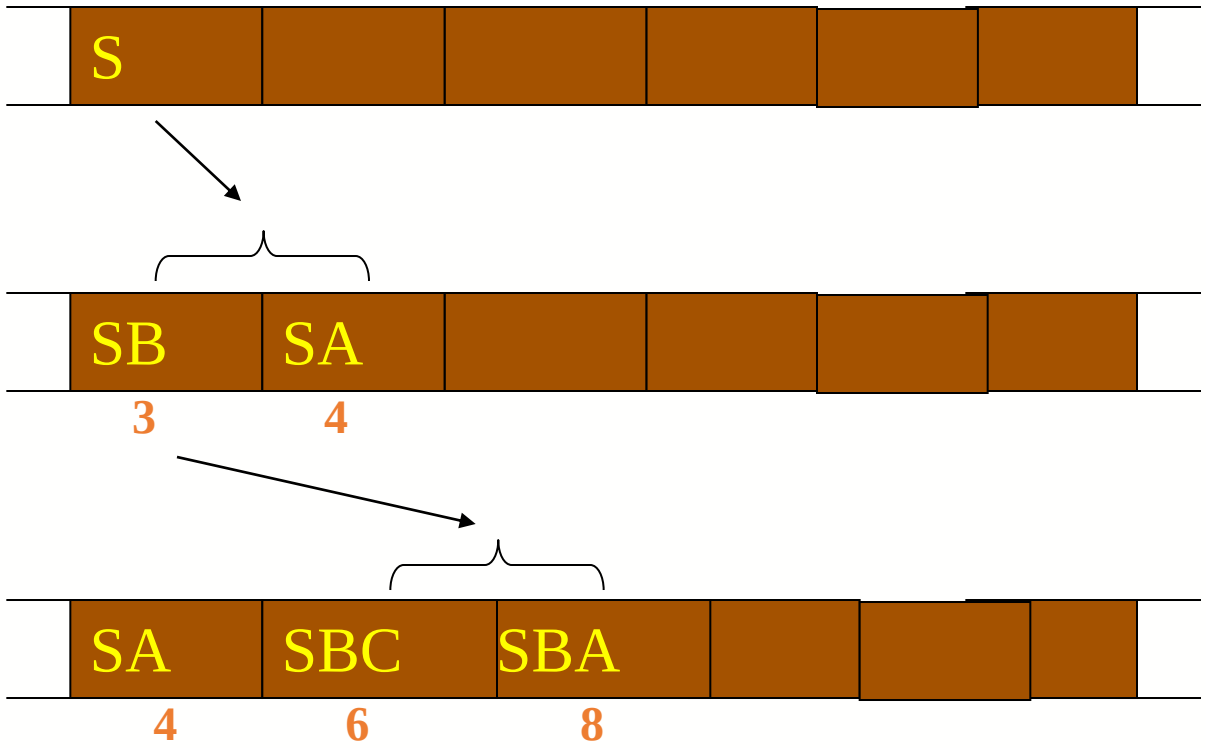
# Analisa Hill climbing

- Kelebihan
  - Butuh memori kecil
  - Menemukan solusi tanpa harus menguji lebih banyak lagi
- Kelemahan
  - Mungkin terjebak pada local optima
  - Perlu menentukan aturan yang tepat

# Branch and Bound



# Branch and Bound

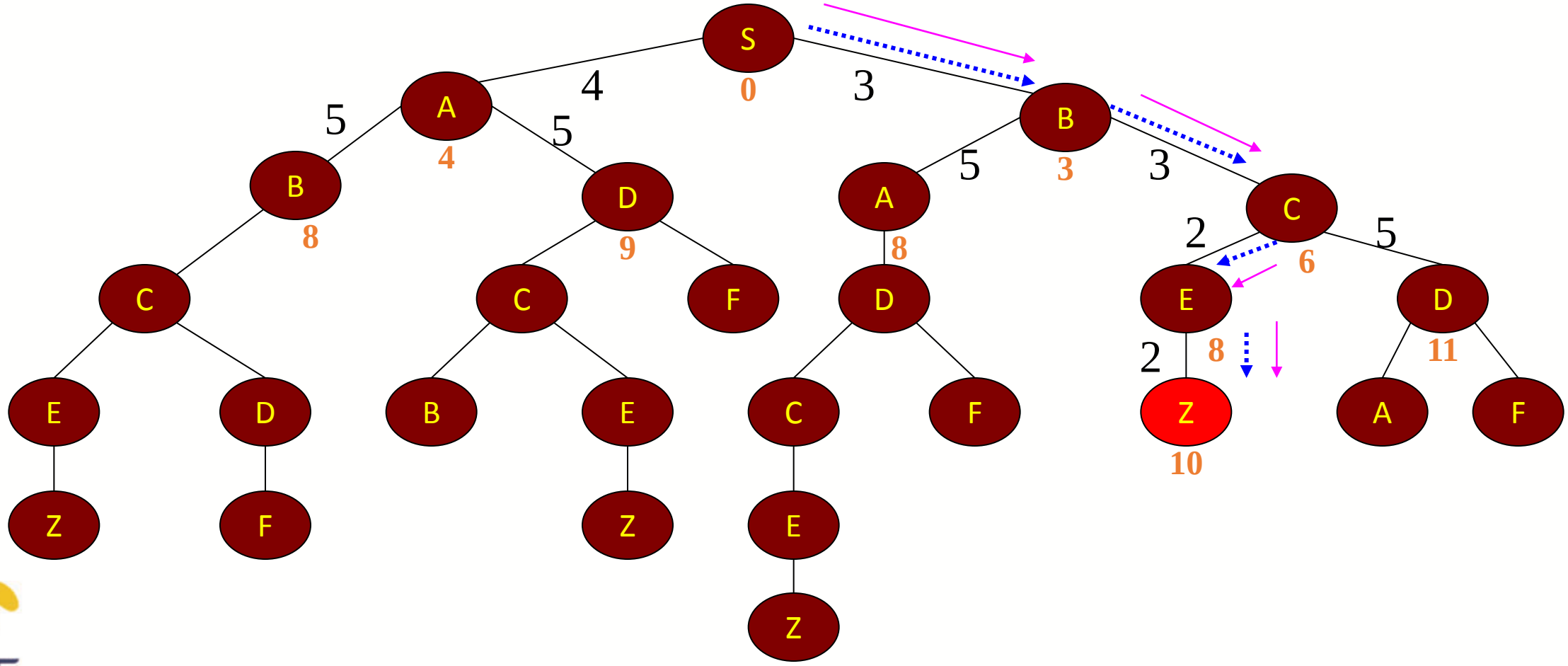



dan seterusnya…

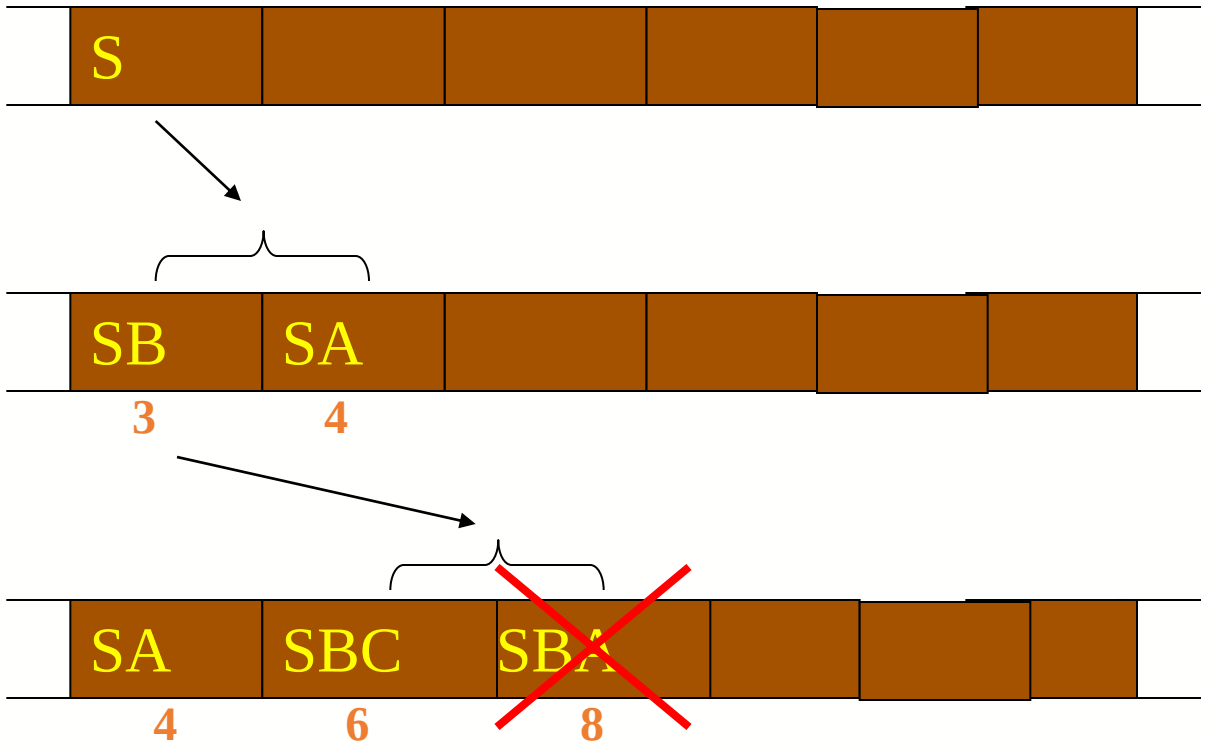
# Analisa Branch and Bound

- Kelebihan
  - Selalu menemukan global optimum
- Kelemahan
  - Boros memori karena menyimpan lintasan partial lebih dari 1 kali

# Dynamic Programming



# Dynamic Programming

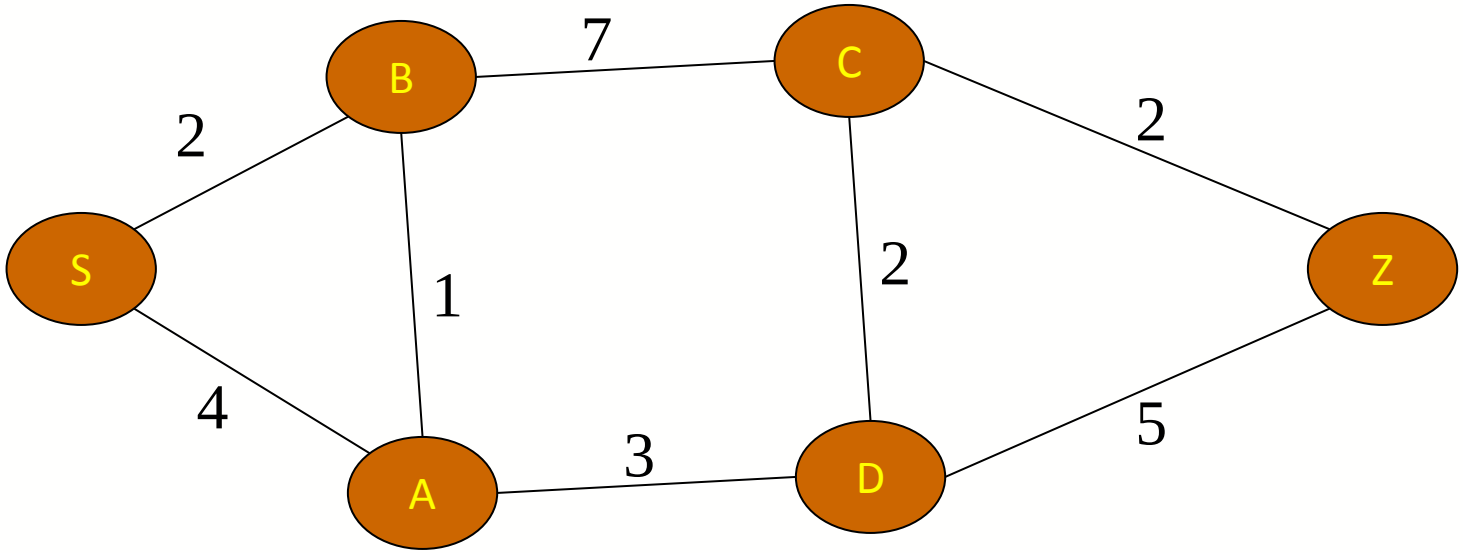



dan seterusnya…

# Dynamic Programming

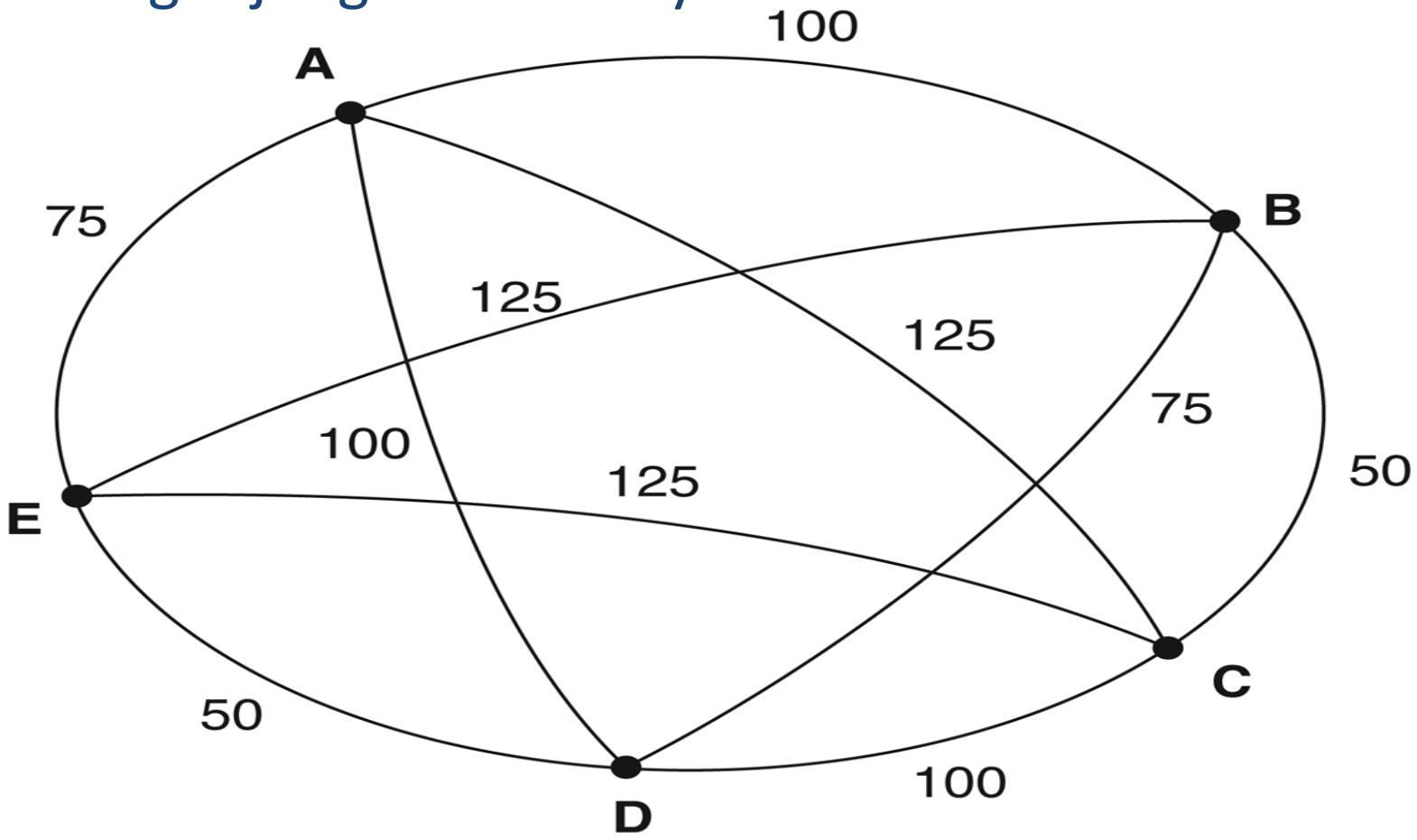
- Kelebihan
  - Selalu menemukan global optimum
  - Lebih cepat dan hemat memori karena hanya 1 kali menyimpan lintasan partial
- Kelemahan
  - Harus mengingat node terakhir dari lintasan partial yang sudah dicapai sebelumnya

# Tugas 1



# Tugas 2
## Traveling Salesperson Problem

- Dari kota A mengunjungi kota lainnya dan kembali ke kota A



# Tugas

Selesaikan Tugas 1 dan Tugas 2:

- Representasikan kasus diatas dengan tree
- Selesaikan kasus diatas dengan metode:
  - Best First Search
  - Hill climbing
  - Branch and Bound
  - Dynamic Programming

bridge to the future

http://www.eepis-its.edu