Bermanfaat
dan Mendunia

Universitas Padjadjaran

Searching: Heuristik

Dr. Intan Nurma Yulita, M.T

unpad.ac.id

Tentang Saya

Dr. Intan Nurma Yulita, M.T

Dosen Departemen Ilmu Komputer Universitas Padjadjaran
Ketua Pusat Riset Kecerdasan Artifisial dan Big Data Universitas Padjadjaran (2021-2023)
Presiden Indonesian Association for Pattern Recognition (INAPR) (2023-sekarang)
Kepala Pusat Inovasi Pengajaran dan Pembelajaran Universitas Padjadjaran (2024-sekarang)
intan.nurma@unpad.ac.id

Teknik Dasar AI

- **Searching (Pencarian)**
  AI mencari solusi terbaik dari berbagai kemungkinan.
- **Reasoning (Penalaran)**
  AI menganalisis informasi untuk mengambil keputusan.
- **Planning (Perencanaan)**
  AI menyusun langkah-langkah untuk mencapai tujuan.
- **Learning (Pembelajaran)**
  AI belajar dari data untuk meningkatkan kinerjanya.

Apa itu Algoritma Pencarian?

Algoritma pencarian digunakan dalam kecerdasan buatan (AI) untuk menemukan solusi dari suatu permasalahan dengan menjelajahi ruang keadaan (*state space*).

Secara umum, terdapat dua kategori pencarian:

1. **Pencarian Buta (Blind Search)**
   Tidak memiliki informasi tambahan tentang jalur terbaik menuju solusi selain ruang keadaan yang didefinisikan.
2. **Pencarian Heuristik (Informed Search)**
   Menggunakan fungsi heuristik untuk mempercepat pencarian solusi.

Pendahuluan

Algoritma pencarian heuristik digunakan untuk menemukan solusi dalam ruang pencarian yang besar dengan memanfaatkan "**pengetahuan**" **tambahan**. Teknik ini mempercepat proses pencarian dengan memilih jalur yang lebih menjanjikan.

Konsep Dasar Pencarian Heuristik

- **Pencarian heuristik:** Menggunakan informasi tambahan untuk memilih jalur terbaik.
- **Fungsi heuristik (h(n)):** Perkiraan biaya atau jarak dari suatu node ke tujuan.
- **Tujuan:** Menemukan solusi dengan meminimalkan jumlah langkah atau biaya.

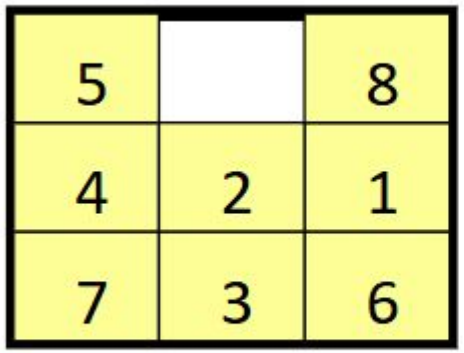
Algoritma

Algoritma	Konsep	Kelebihan	Kekurangan
Greedy Best-First Search	Menggunakan fungsi heuristik f(N) = h(N). Memilih jalur dengan nilai heuristik terendah.	Cepat dalam menemukan solusi.	Bisa terjebak dalam local optima.
Algoritma A*	Menggunakan kombinasi g(n) (biaya dari awal ke node saat ini) dan h(n) (perkiraan ke tujuan): f(n) = g(n) + h(n)	Optimal jika h(n) tidak melebih-lebihkan jarak ke tujuan.	Boros memori.
Hill Climbing	Memilih langkah dengan nilai heuristik terbaik saat ini tanpa melihat langkah berikutnya.	Mudah diimplementasikan.	Bisa terjebak dalam local optima.
Branch and Bound	Menggunakan tree search dengan memangkas cabang yang memiliki nilai lebih tinggi.	Menjamin global optima.	Boros memori karena menyimpan banyak jalur.
Dynamic Programming	Menyimpan hasil perhitungan sebelumnya untuk menghindari perhitungan ulang	Lebih cepat dibanding metode lain	Membutuhkan penyimpanan tambahan

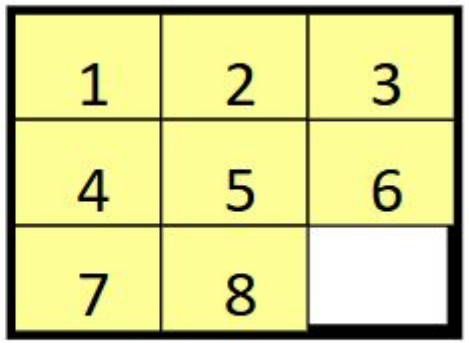
Fungsi Heuristik (1)

Fungsi h (N) yang memperkirakan cost jalur terpendek dari node N ke node tujuan.

Contoh: 8-puzzle



N



goal

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

ngsi Heuristik (2) – Manhattan Distance

Current State

3	2	8
4	5	6
7	1	

Tile yang tidak sesuai dengan node tujuan adalah tile "3", "8" dan "1" dengan perbedaan 2, 3, dan 3 tile.

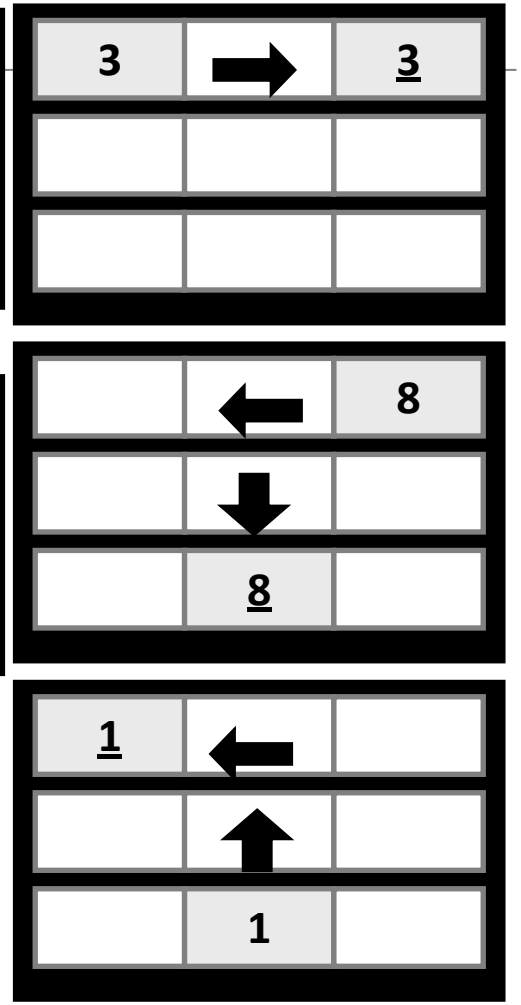
Jadi fungsi heuristiknya akan mencapai 8.

Dengan kata lain, heuristik memberitahukan bahwa solusi tersedia membutuhkan hanya dalam 8 langkah lagi.

Catatan : $h(n)$ $h(\text{current state}) = 8$

1	2	3
4	5	6
7	8	

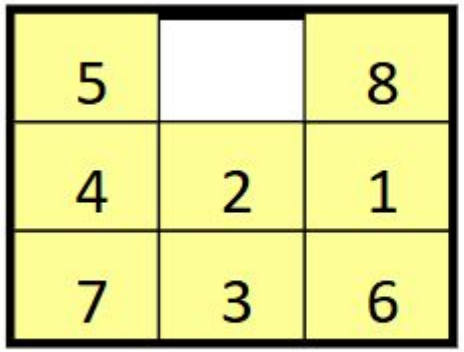
Goal State



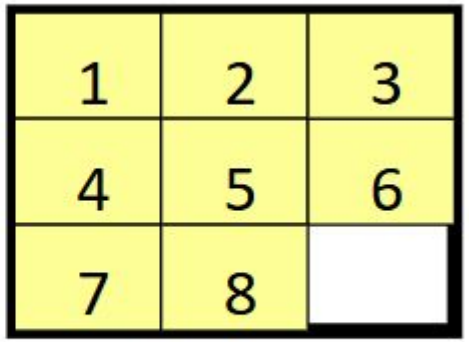
Fungsi Heuristik (2) – Manhattan Distance

Fungsi h (N) yang memperkirakan cost jalur terpendek dari node N ke node tujuan.

Contoh: 8-puzzle



N



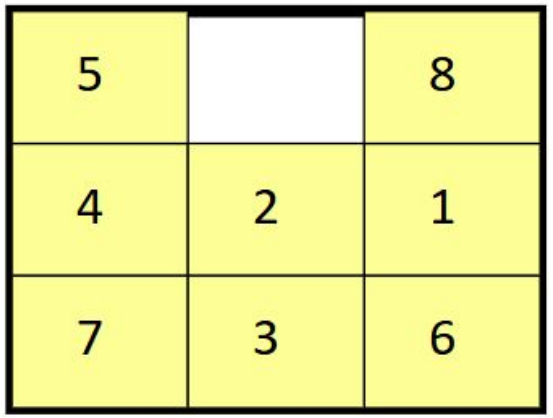
goal

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

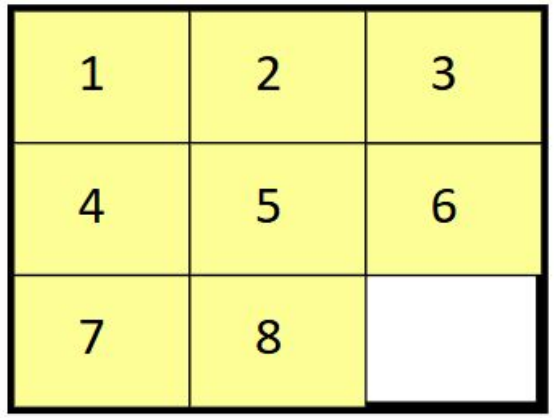
= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1

= 13

8-Puzzle



N

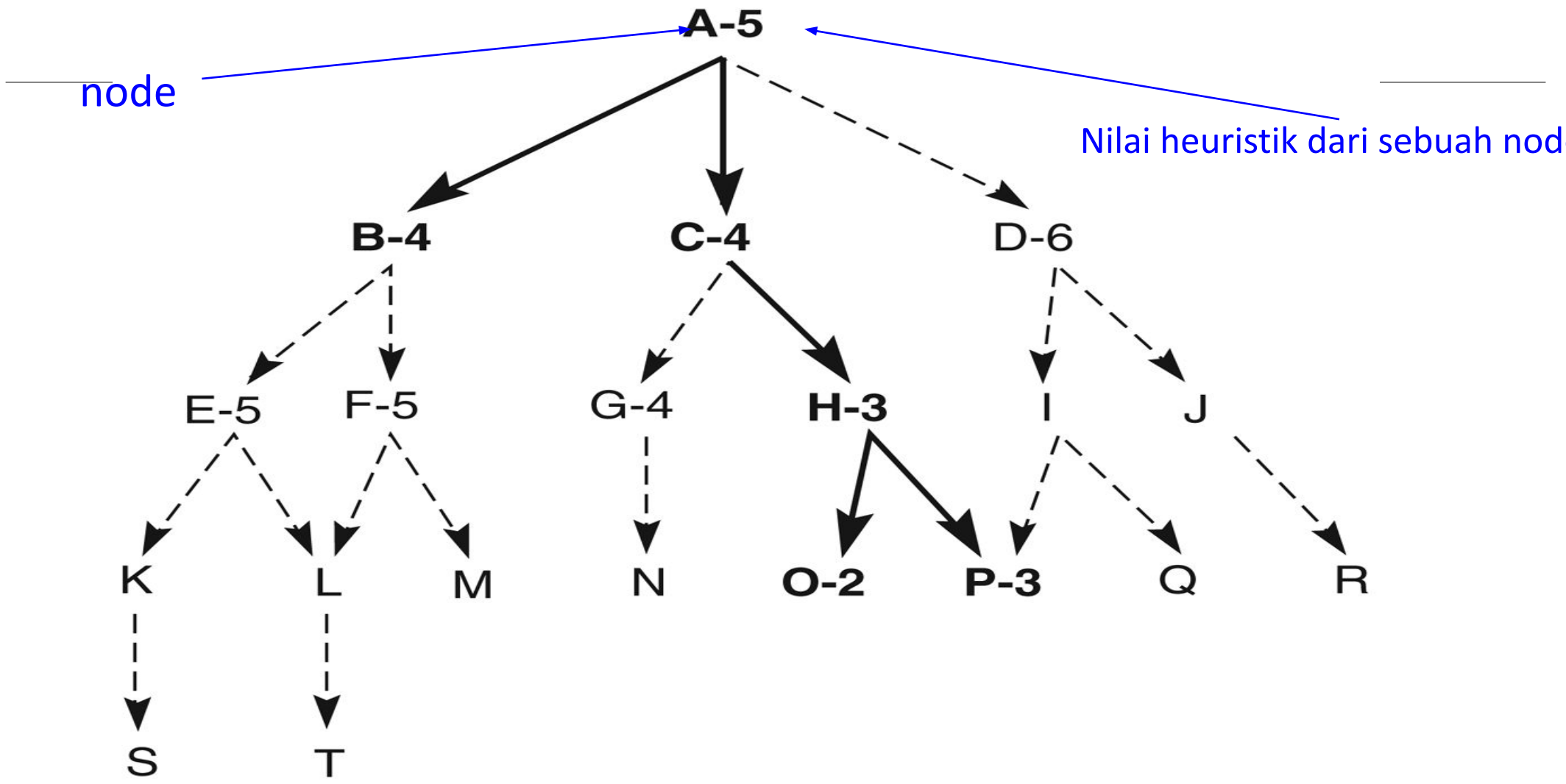


goal

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;

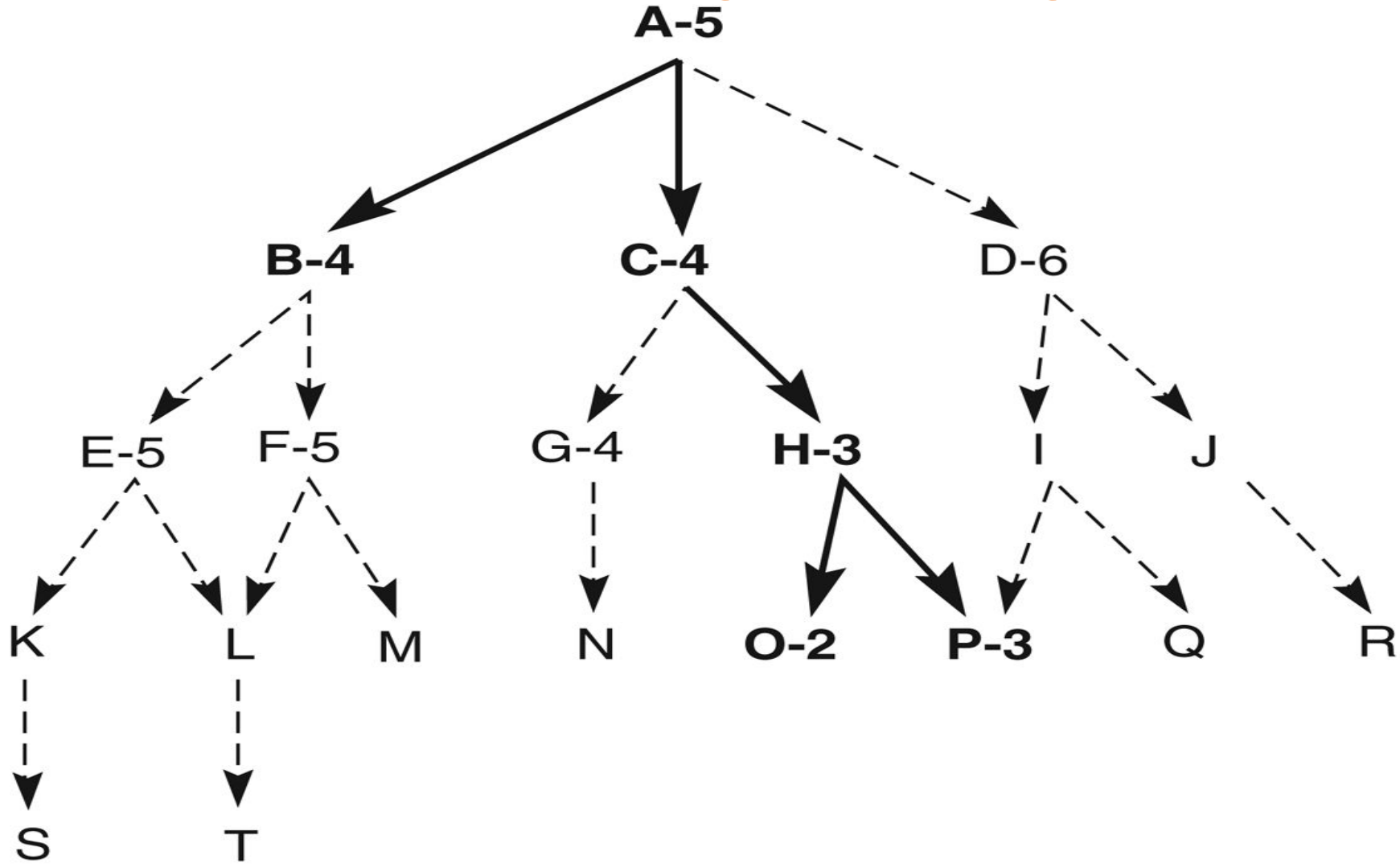
```
begin
  open := [Start];
  closed := [ ];
  while open ≠ [ ] do
    begin
      remove leftmost state from open, call it X;
      if X is a goal then return SUCCESS
        else begin
          generate children of X;
          put X on closed;
          discard remaining children of X if already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

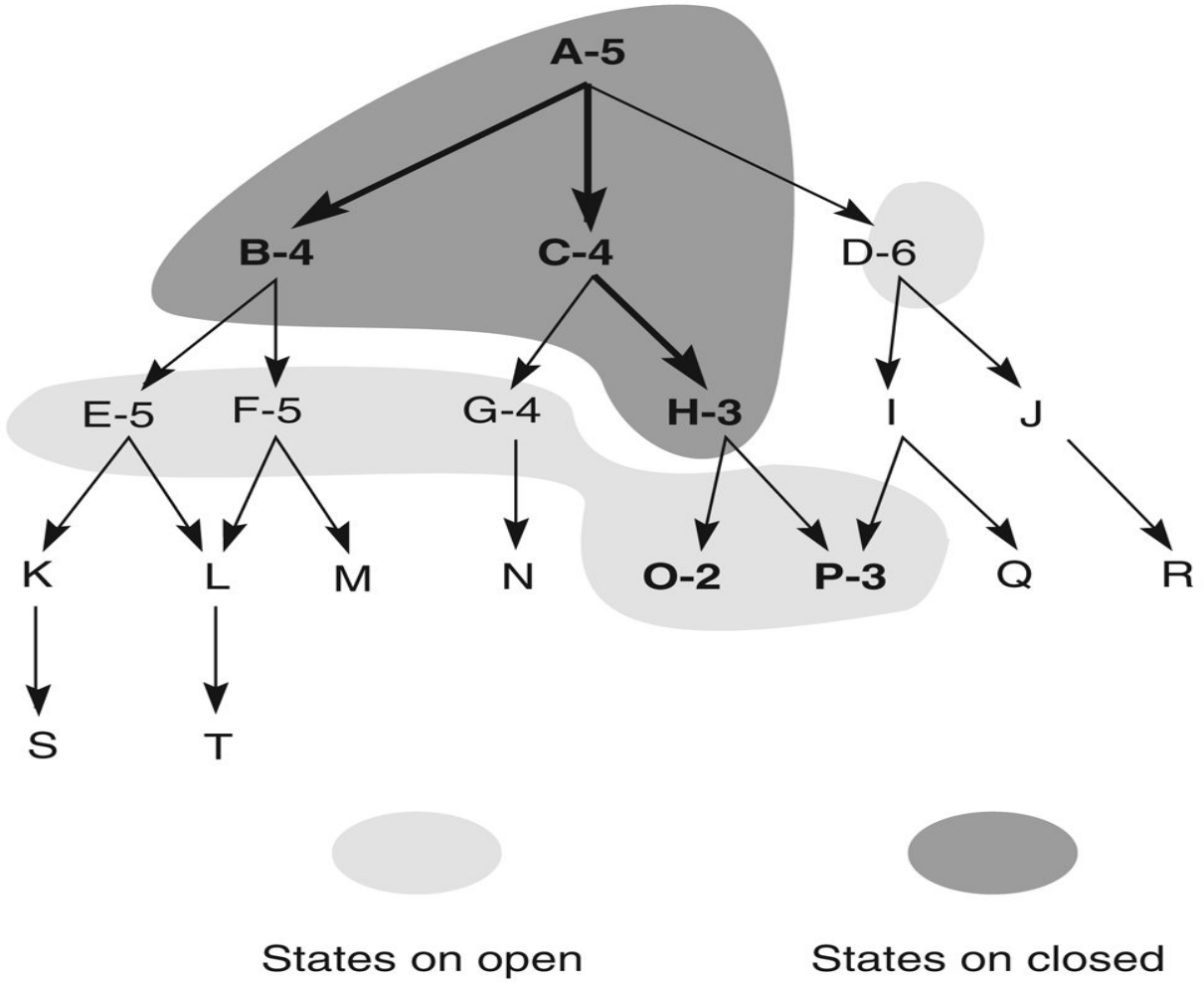
Pencarian Heuristik dengan Ruang Keadaan Berikut



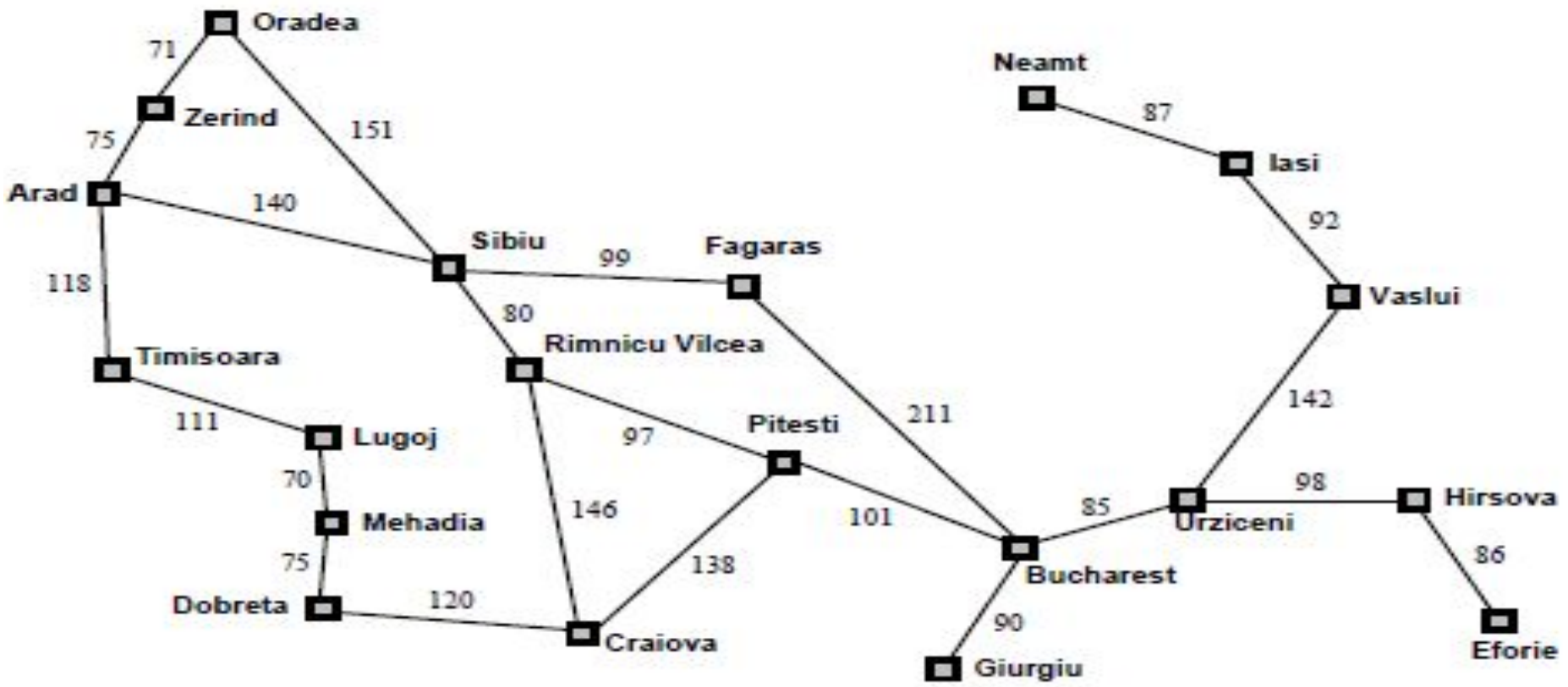
Proses Trace untuk Algoritma Best First Search

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 - Menandai Open dan Close



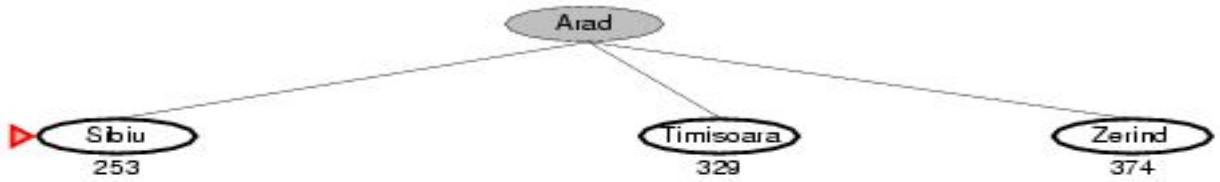
Contoh 2. Algoritma Best First Search



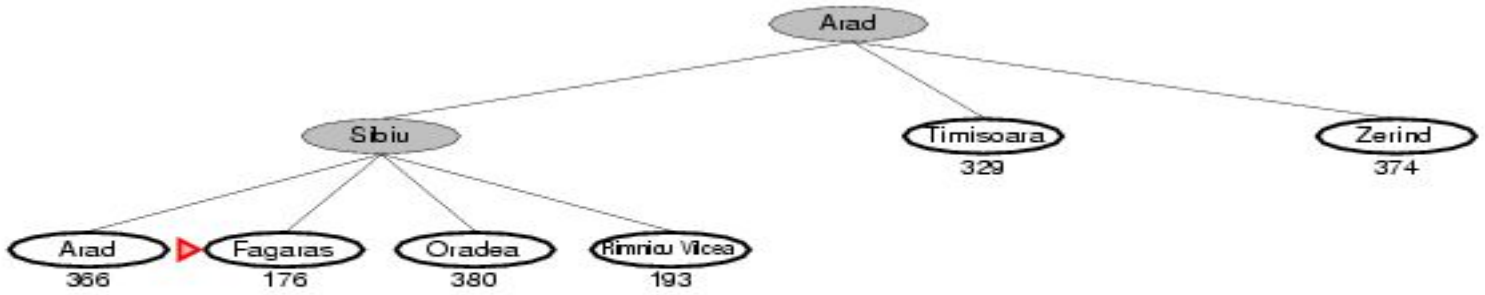
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Contoh 2. Algoritma Best First Search

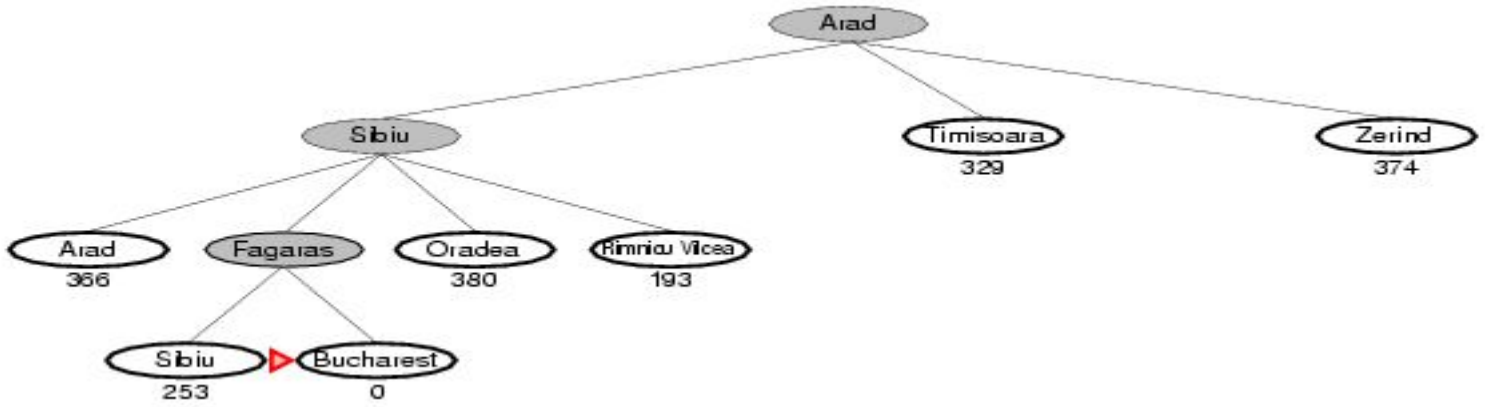
Contoh 2. Algoritma Best First Search



Contoh 2. Algoritma Best First Search



Contoh 2. Algoritma Best First Search

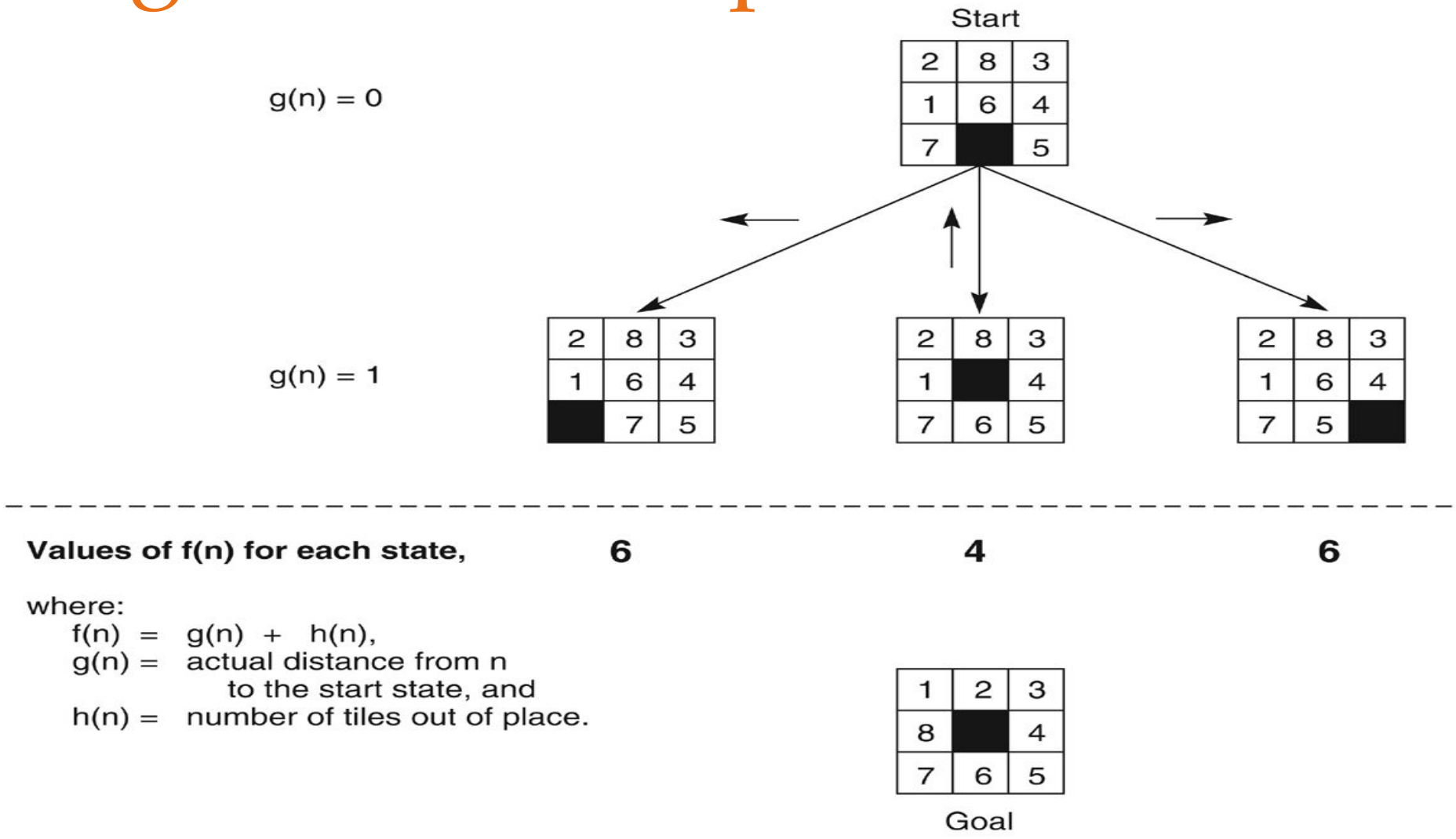


Algoritma A*

A * (A star) adalah bentuk pencarian Best-First yang paling banyak dikenal

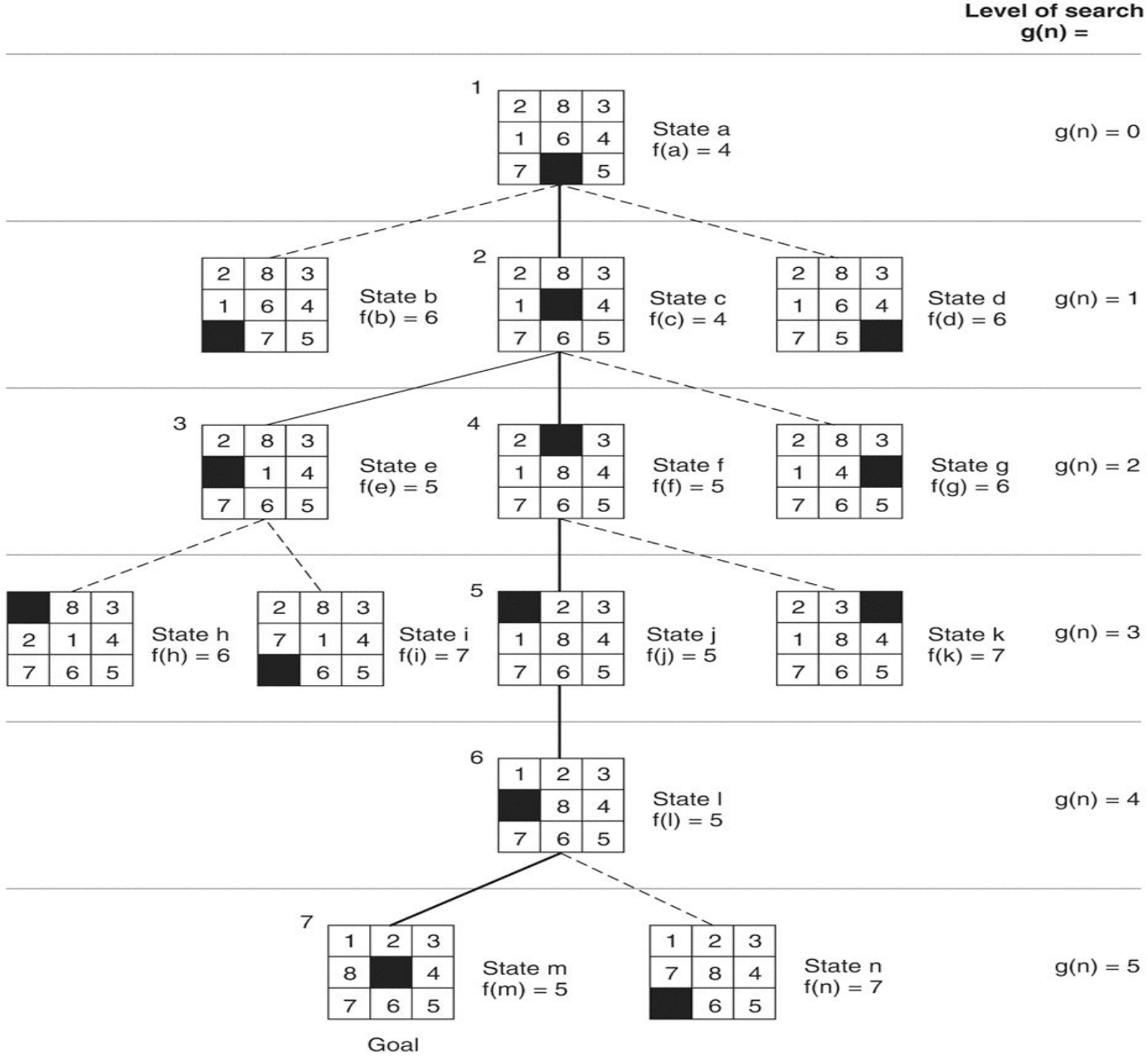
- Mengevaluasi node dengan menggabungkan g (n) dan h (n)
- f (n) = g (n) + h (n)
- Dimana
  - g(n) = cost dari node asal ke node saat ini yaitu n
  - h(n) = perkiraan cost dengan lintasan terdekat dari node n ke node tujuan
  - f(n) = perkiraan total cost pada lintasan yang melalui node n

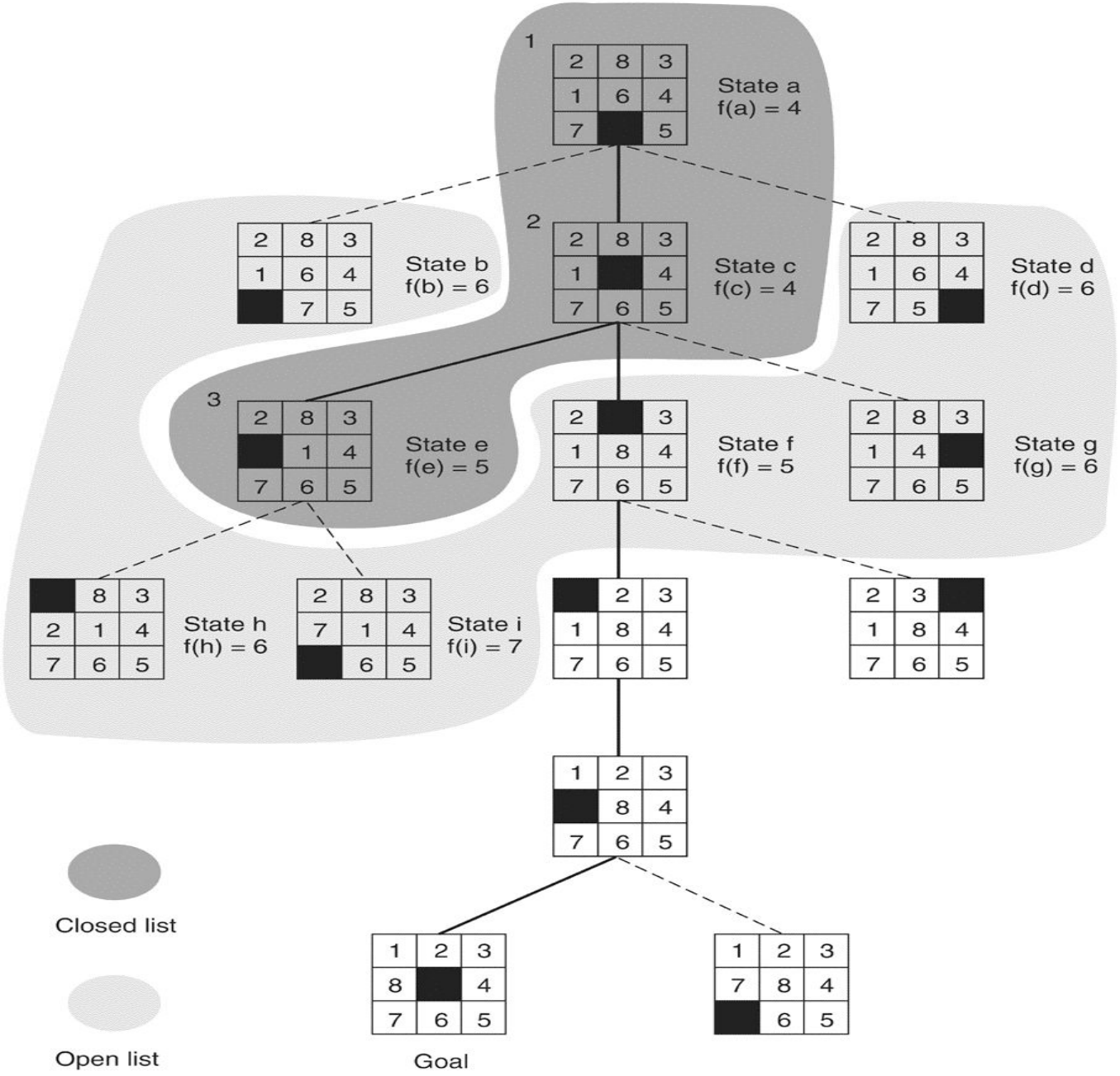
Fungsi Heuristik pada 8-Puzzle



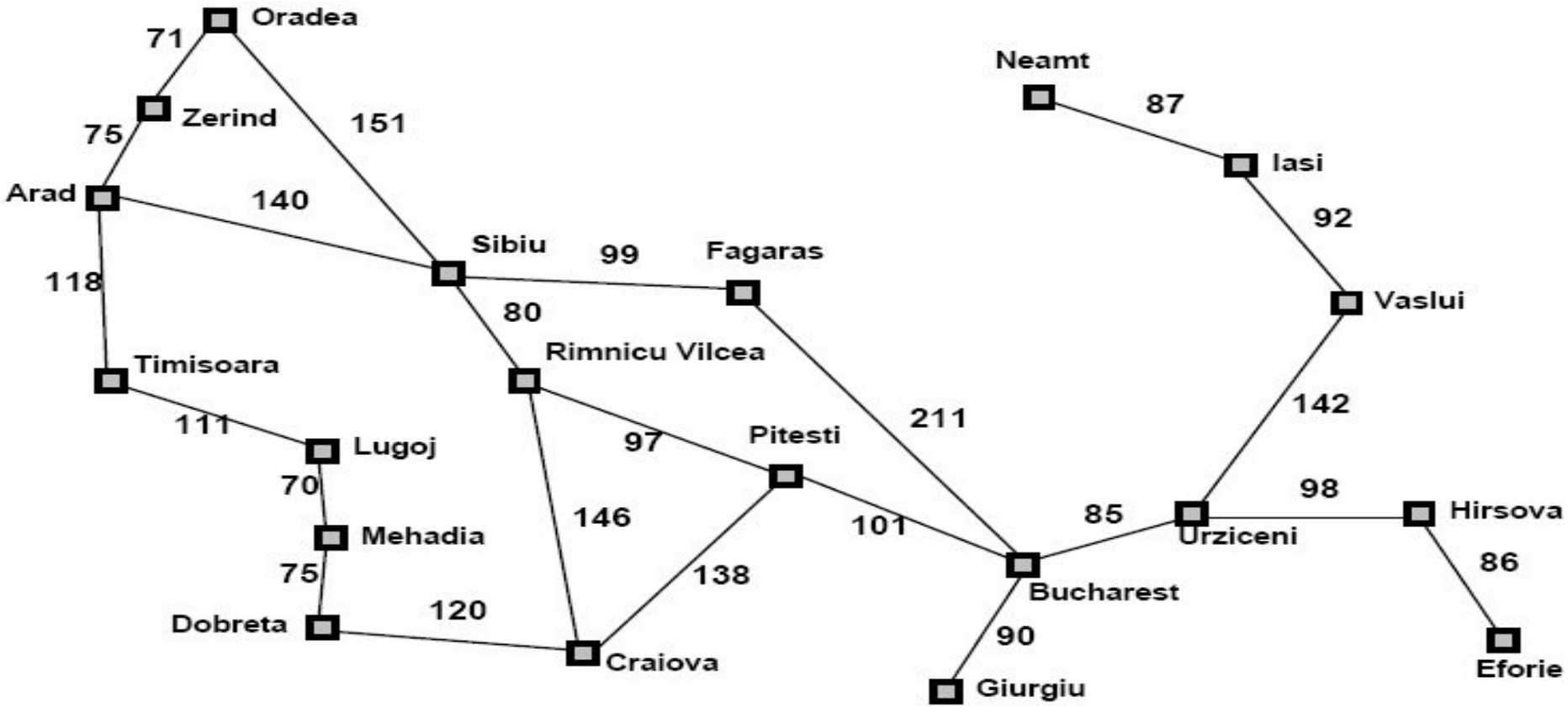
Proses Trace pada Algoritma A*

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

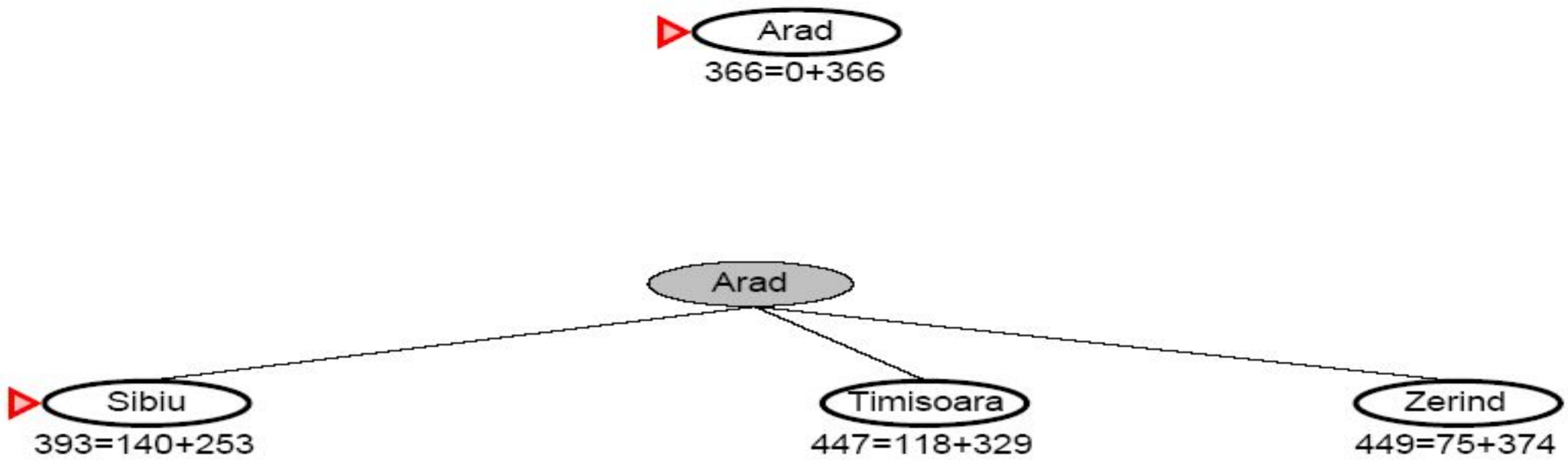

1
2 8 3
1 6 4
7 5
State a
f(a) = 4
2
2 8 3
1 4
7 6 5
State c
f(c) = 4
2 8 3
1 6 4
7 5
State b
f(b) = 6
State d
f(d) = 6
3
2 8 3
1 4
7 6 5
State e
f(e) = 5
2 3
1 8 4
7 6 5
State f
f(f) = 5
2 8 3
1 4
7 6 5
State g
f(g) = 6
8 3
2 1 4
7 6 5
State h
f(h) = 6
2 8 3
7 1 4
6 5
State i
f(i) = 7
2 3
1 8 4
7 6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

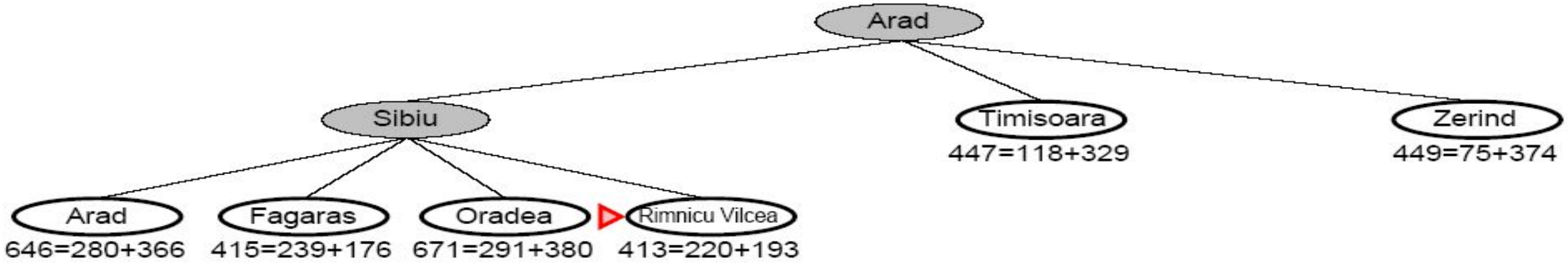


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

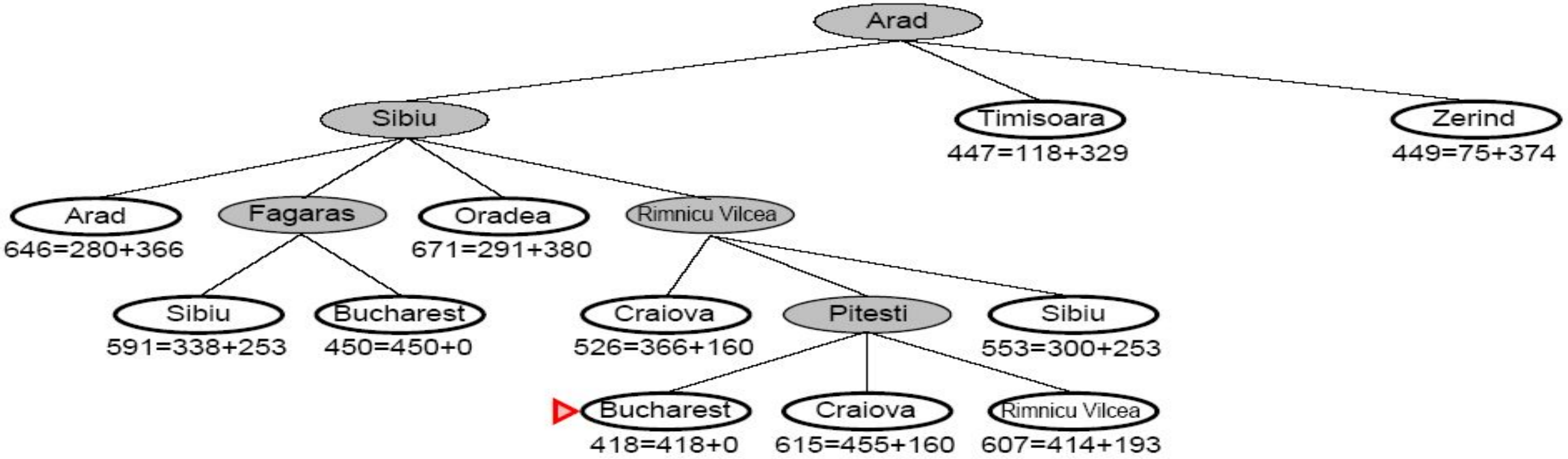
A* Search


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

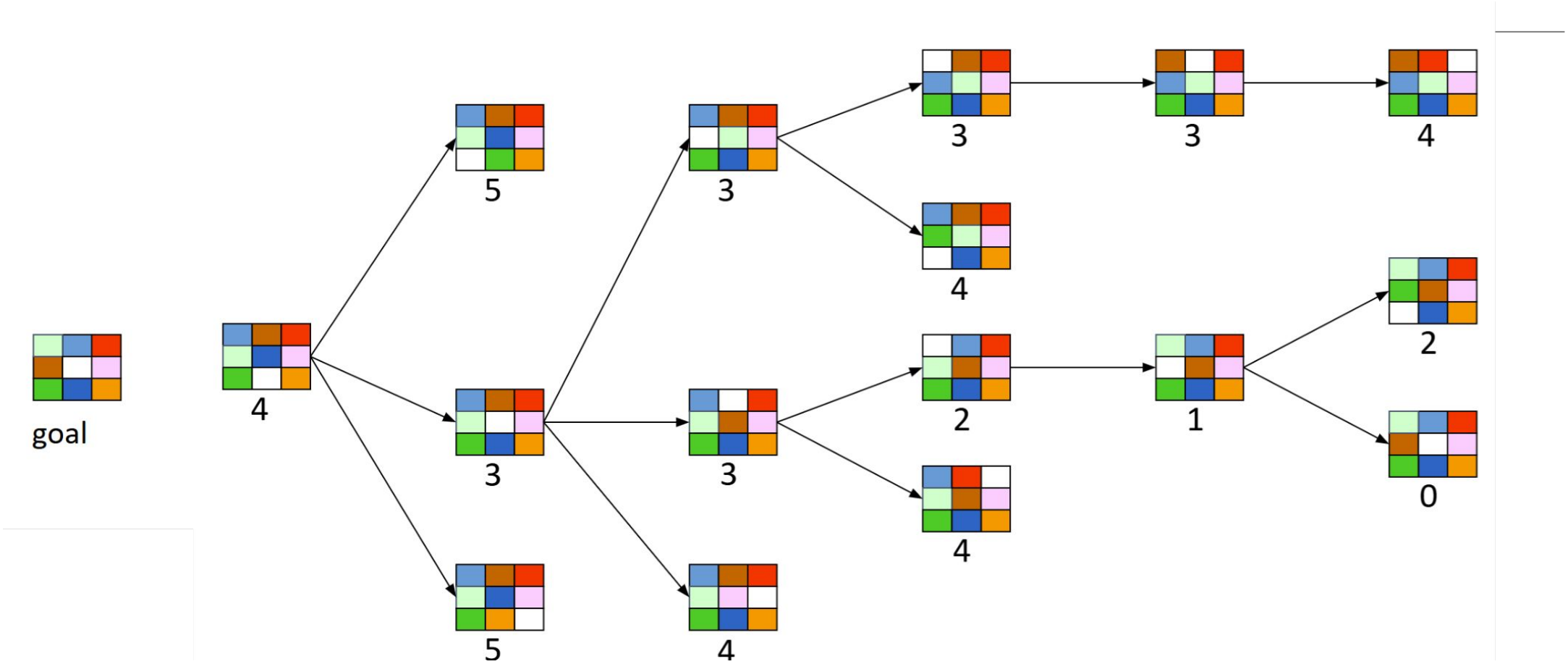
A* Search


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

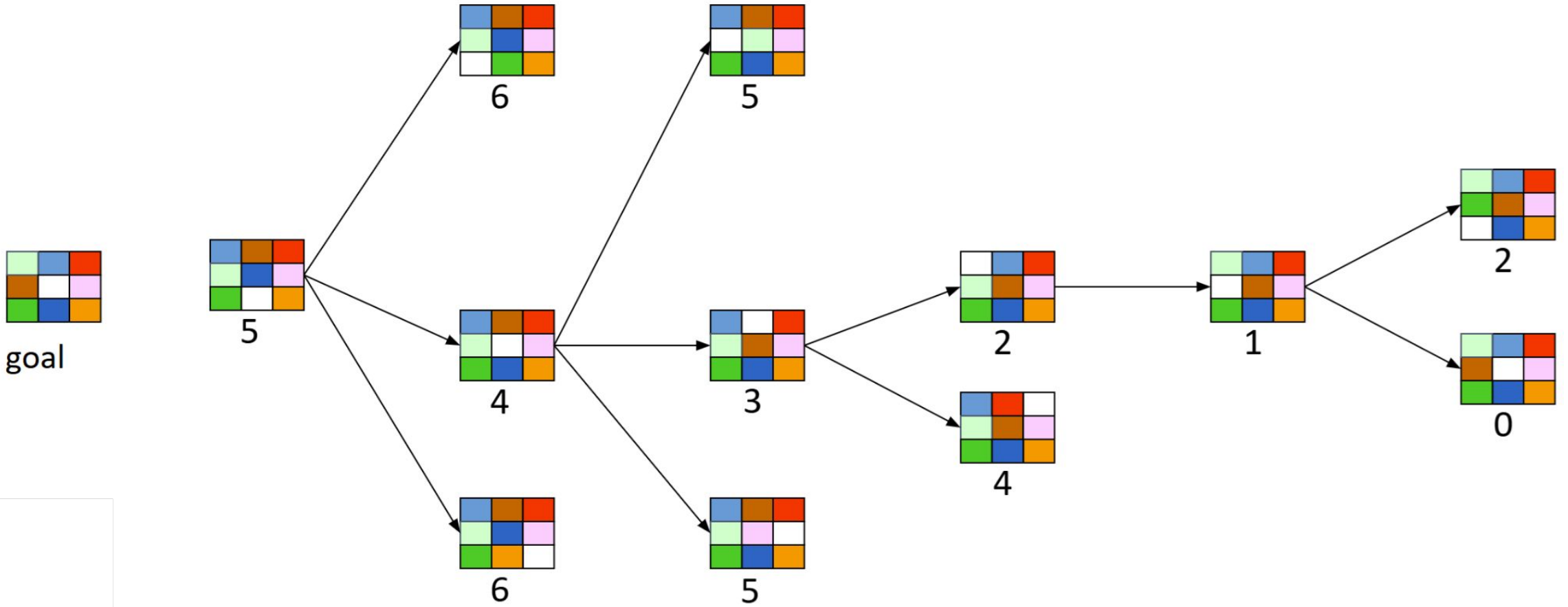

Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

Algoritma Best First Search (8-Puzzle)



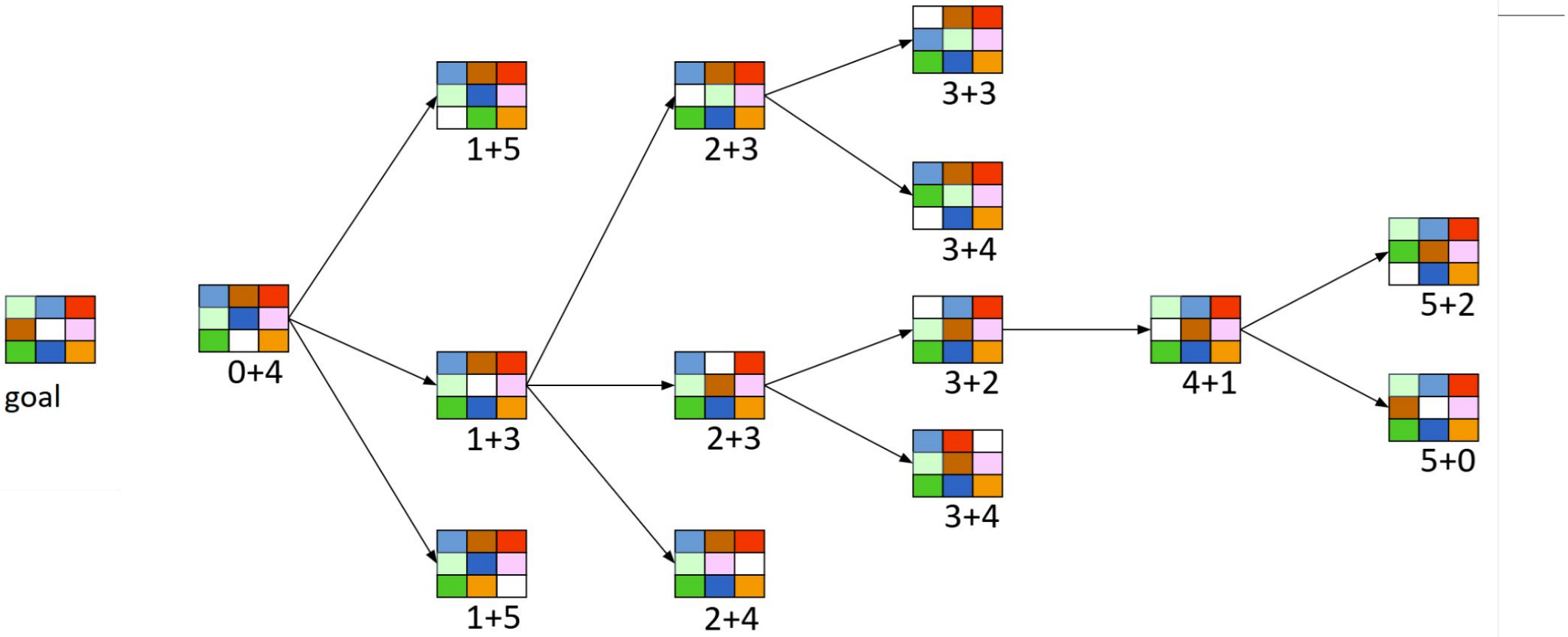
f(N) = h(N) = jumlah tile yang berbeda

Algoritma Best First Search (8-Puzzle)



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

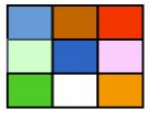
Algoritma Best First Search (8-Puzzle)



(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

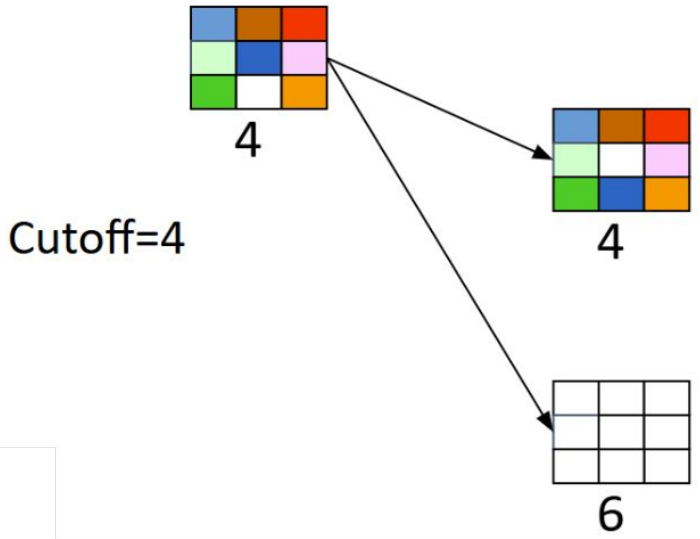


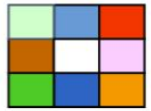
4

Cutoff=4

Algoritma A* (8-Puzzle) dengan Cutoff 5

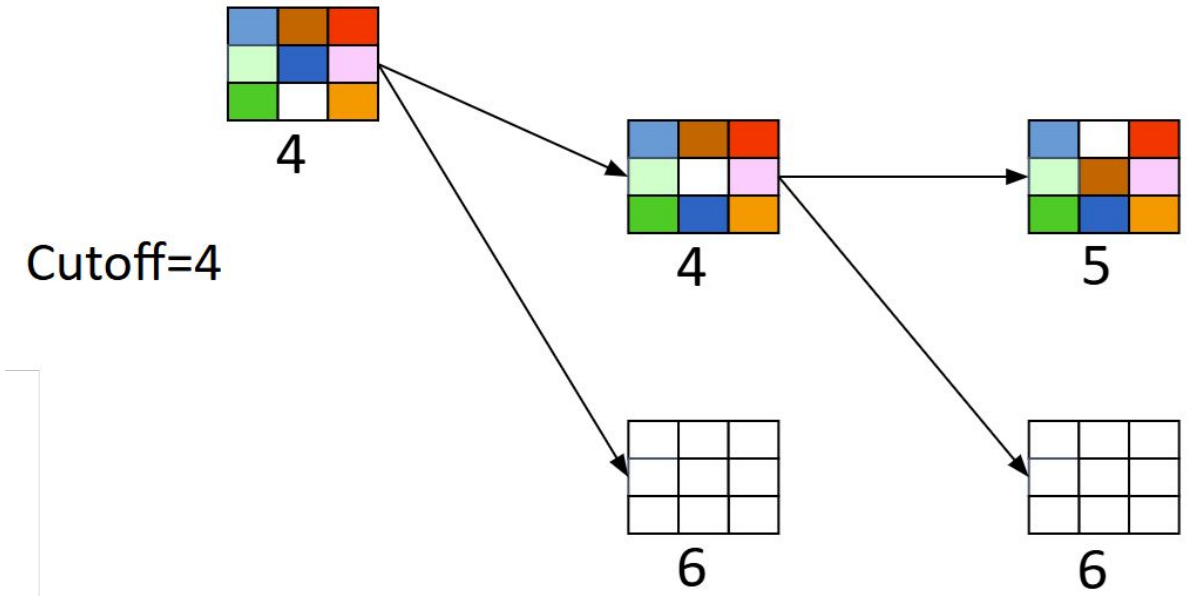
f(N) = g(N) + h(N)
dengan h(N) = jumlah tile yang berbeda

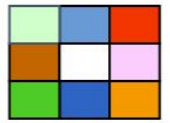




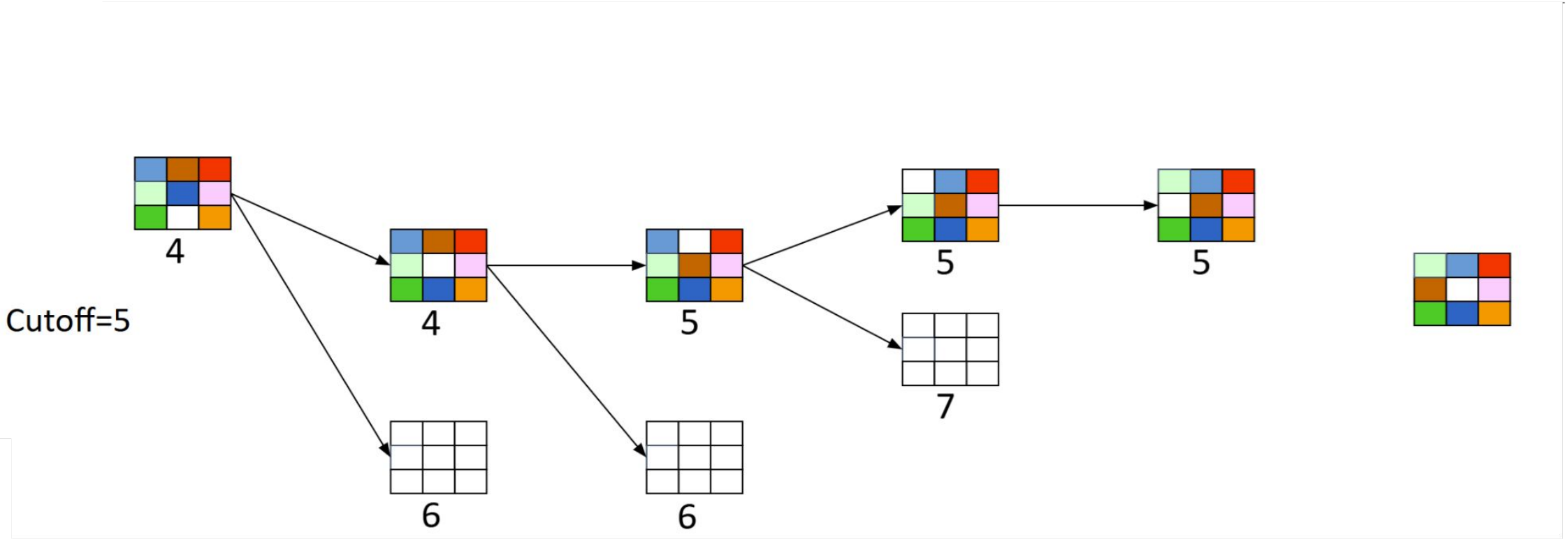
Algoritma A* (8-Puzzle) dengan Cutoff 5

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

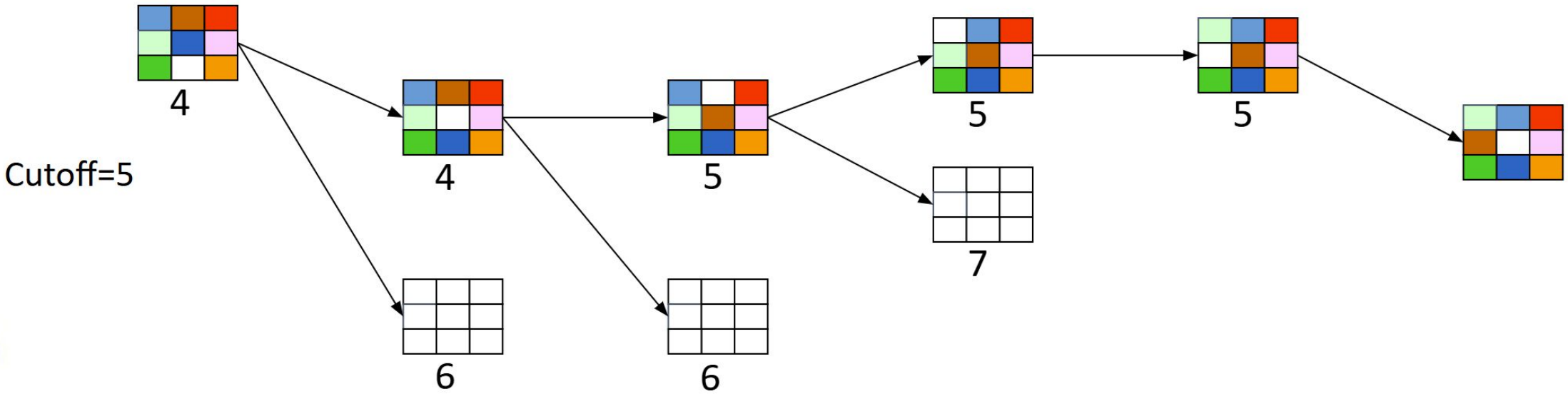




Algoritma A* (8-Puzzle) dengan Cutoff 5



Algoritma A* (8-Puzzle) dengan Cutoff 5



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

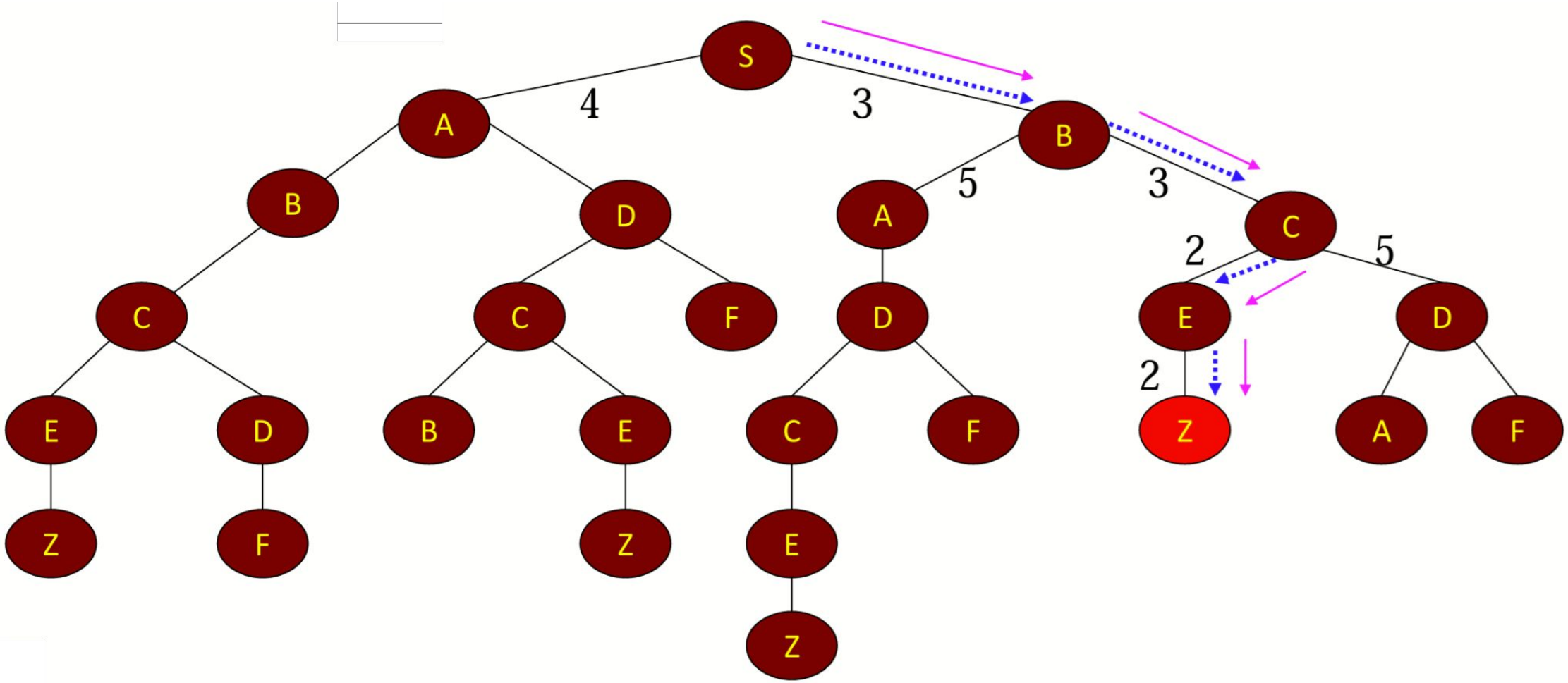
Konsep Dasar

Hill Climbing adalah algoritma pencarian heuristik yang digunakan untuk optimasi dan pencarian solusi terbaik. Algoritma ini bekerja dengan **memulai dari suatu solusi awal** dan **terus bergerak ke arah solusi yang lebih baik** berdasarkan nilai fungsi evaluasi (heuristic function).

Prinsip Kerja

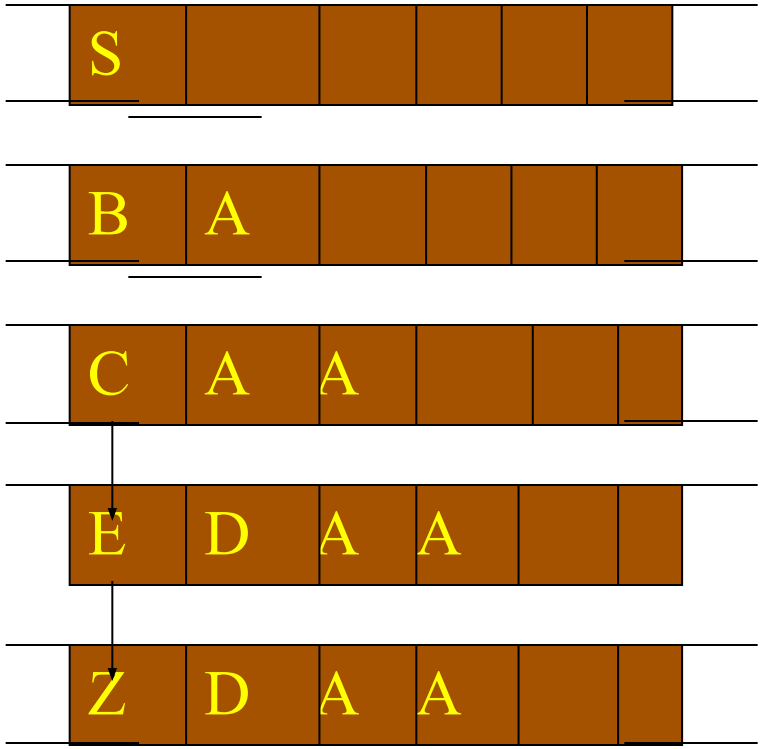
1. **Inisialisasi**: Memulai dari titik awal (solusi awal).
2. **Evaluasi**: Menghitung nilai fungsi evaluasi dari solusi saat ini.
3. **Pencarian Tetangga**: Memilih solusi dari tetangga yang memiliki nilai fungsi evaluasi lebih baik.
4. **Perbandingan**: Jika solusi tetangga lebih baik, pindah ke solusi tersebut. Jika tidak ada solusi tetangga yang lebih baik, berhenti (terjebak di local optimum).
5. **Iterasi**: Ulangi proses sampai tidak ada perbaikan yang bisa dilakukan

Hill climbing



Hill climbing

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



Rule: yang paling kecil jaraknya

Hill climbing

Perbedaannya dengan DFS

1. **Aturan dalam Pemilihan Node Anak**
   - Dalam **DFS**, eksplorasi dilakukan berdasarkan urutan tanpa mempertimbangkan kualitas solusi.
   - Dalam **Hill Climbing**, pemilihan node anak dilakukan **berdasarkan nilai fungsi heuristik** (hanya memilih solusi yang lebih baik).
2. **Tidak Kembali ke Solusi Sebelumnya**
   - DFS dapat kembali ke node sebelumnya jika menemui jalan buntu (backtracking).
   - Hill Climbing **tidak melakukan backtracking**, sehingga bisa terjebak di **local optimum** atau **plateau**.
3. **Arah Pencarian**
   - DFS mencoba semua kemungkinan tanpa mempertimbangkan apakah sedang menuju solusi terbaik.
   - Hill Climbing selalu memilih **solusi terbaik yang tersedia di lingkungan saat ini**.

DFS seperti seseorang yang menjelajahi hutan tanpa peduli apakah jalur yang diambil adalah yang terbaik.

Hill Climbing seperti seseorang yang mendaki gunung dengan hanya memilih jalur yang terlihat lebih tinggi, tetapi bisa terjebak di puncak yang lebih rendah (**local optimum**).

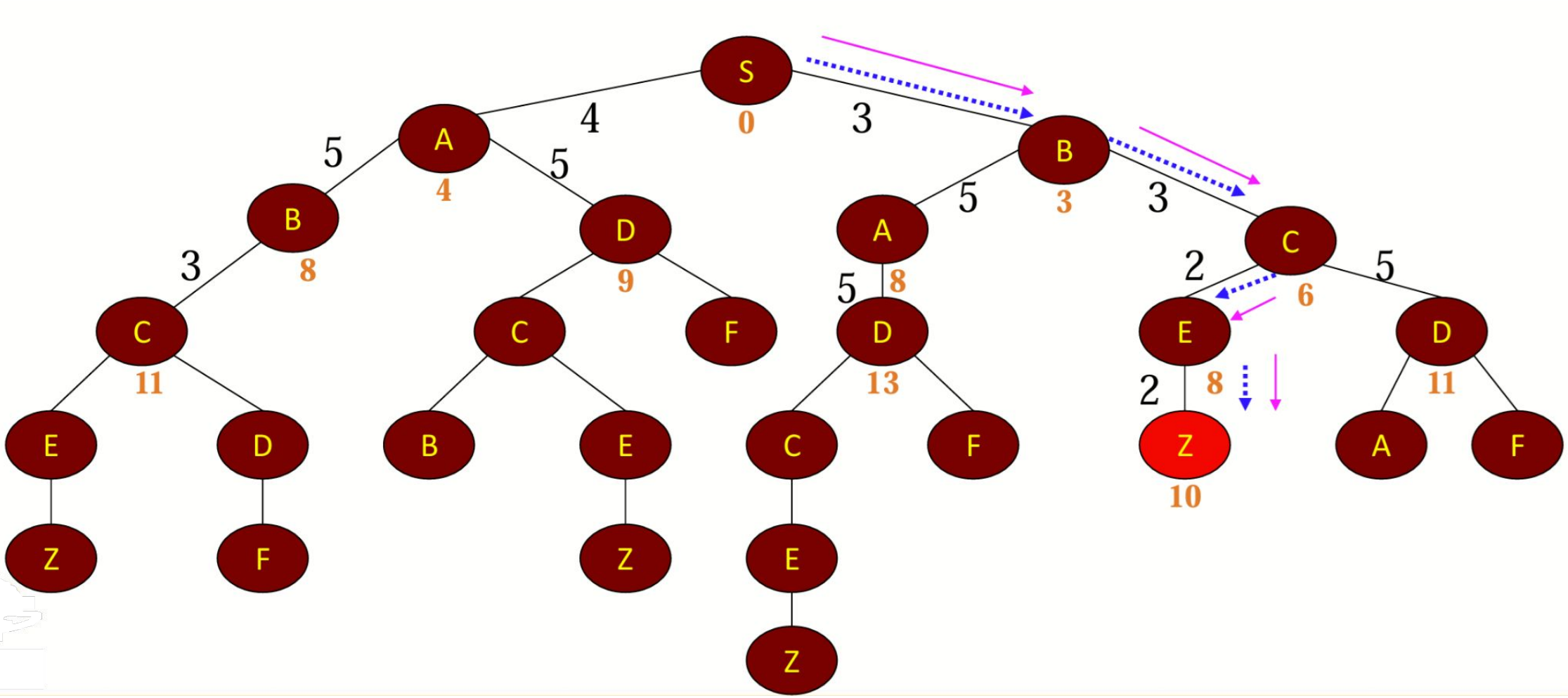
Branch and Bound

B&B menggunakan pendekatan **Depth First Search (DFS) atau Breadth First Search (BFS)** dengan **pemangkasan (bounding)** untuk menghindari eksplorasi cabang yang tidak mengarah ke solusi optimal.

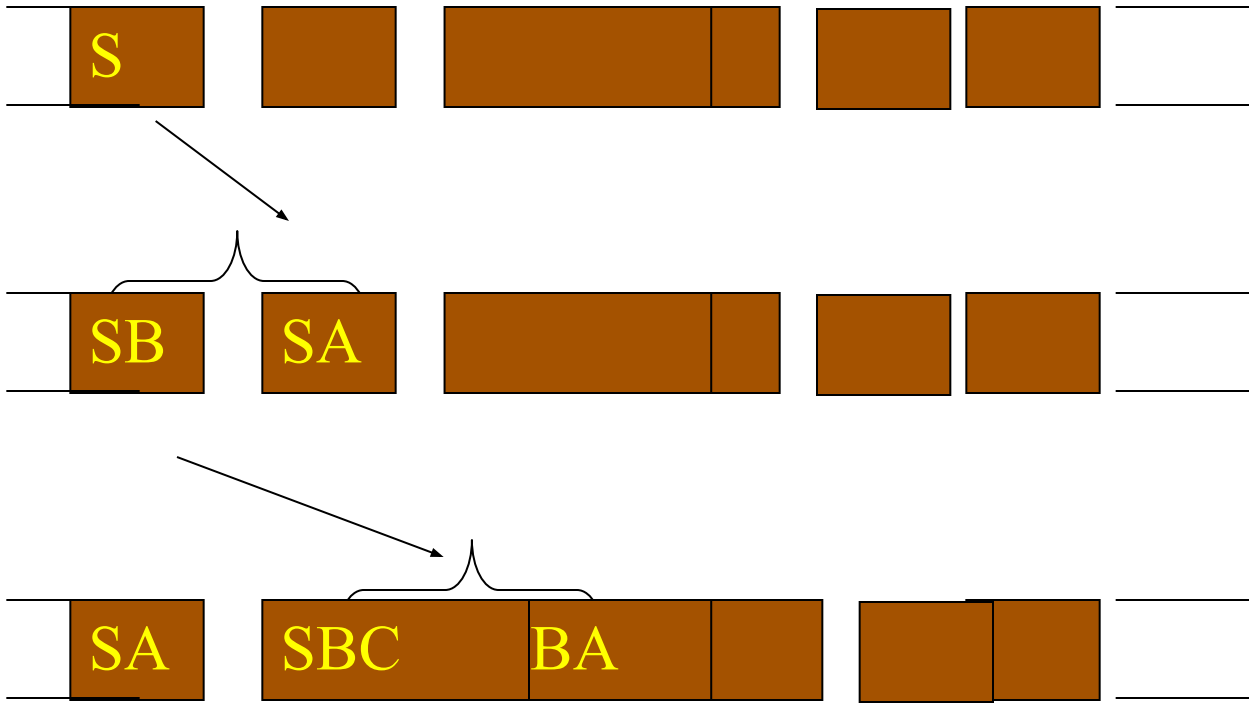
Langkah-langkah utama:

1. **Branching (Pencabangan)**
   - Memecah masalah utama menjadi submasalah yang lebih kecil (mirip rekursi atau tree traversal).
2. **Bounding (Pembatasan)**
   - Menggunakan **fungsi batas (bound function)** untuk memperkirakan solusi terbaik dari setiap cabang.
   - Jika batas suatu cabang lebih buruk daripada solusi terbaik yang ditemukan sejauh ini, cabang tersebut **dipangkas (pruned)**.
3. **Exploration (Penjelajahan Node)**
   - Melakukan pencarian menggunakan **DFS (lebih umum) atau BFS**.
   - Memilih node terbaik berdasarkan strategi seperti **Least Cost Search** atau **Best First Search**.

Branch and Bound



Branch and Bound



dan seterusnya…

Dynamic Programming

Dynamic Programming (DP) adalah teknik pemrograman yang digunakan untuk menyelesaikan **masalah optimasi** dengan cara **membagi masalah menjadi submasalah yang lebih kecil**, menyelesaikan masing-masing submasalah **sekali**, dan menyimpan hasilnya untuk menghindari perhitungan berulang (**memoization** atau **tabulation**).

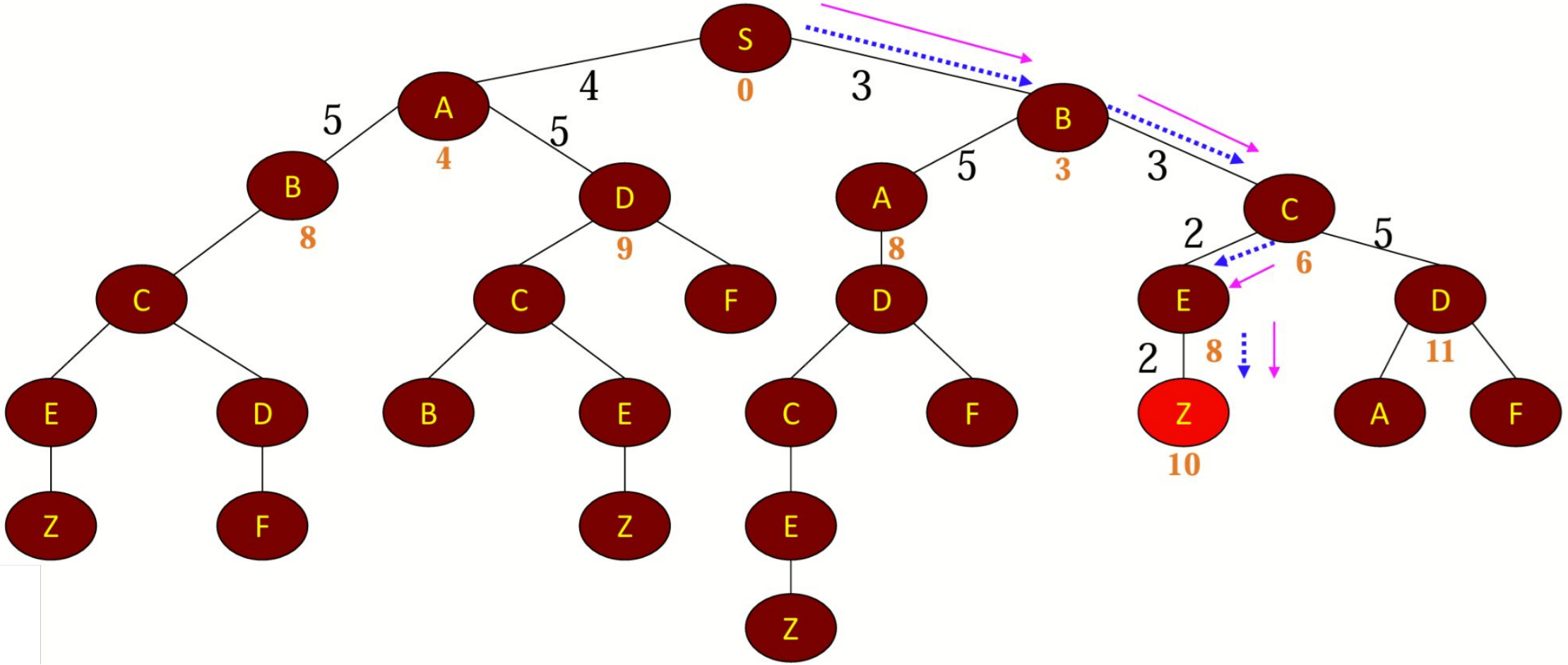
DP digunakan ketika sebuah masalah memiliki:

1. **Overlapping Subproblems:** Submasalah yang sama muncul berulang kali.
2. **Optimal Substructure:** Solusi optimal dari masalah utama bisa dibangun dari solusi optimal submasalahnya.

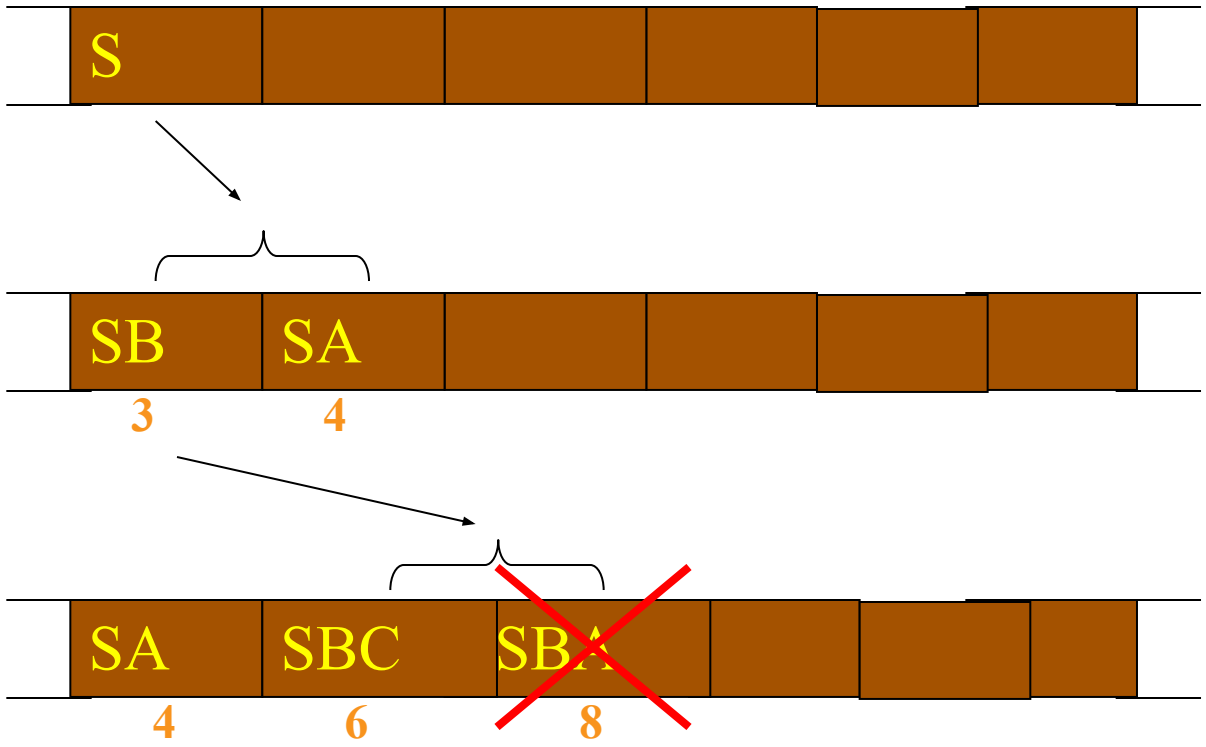
Cara kerja DP:

1. Identifikasi submasalah
2. Definisikan relasi rekursif
3. Gunakan teknik penyimpanan hasil (Memoization atau Tabulation)
4. Bangun solusi dari hasil yang disimpan

Dynamic Programming



Dynamic Programming



dan seterusnya…

Algoritma

Algoritma	Konsep	Kelebihan	Kekurangan
Greedy Best-First Search	Menggunakan fungsi heuristik f(N) = h(N). Memilih jalur dengan nilai heuristik terendah.	Cepat dalam menemukan solusi.	Bisa terjebak dalam local optima.
Algoritma A*	Menggunakan kombinasi g(n) (biaya dari awal ke node saat ini) dan h(n) (perkiraan ke tujuan): f(n) = g(n) + h(n)	Optimal jika h(n) tidak melebih-lebihkan jarak ke tujuan.	Boros memori.
Hill Climbing	Memilih langkah dengan nilai heuristik terbaik saat ini tanpa melihat langkah berikutnya.	Mudah diimplementasikan.	Bisa terjebak dalam local optima.
Branch and Bound	Menggunakan tree search dengan memangkas cabang yang memiliki nilai lebih tinggi.	Menjamin global optima.	Boros memori karena menyimpan banyak jalur.
Dynamic Programming	Menyimpan hasil perhitungan sebelumnya untuk menghindari perhitungan ulang	Lebih cepat dibanding metode lain	Membutuhkan penyimpanan tambahan

REFERENSI

Politeknik Elektronika Negeri Surabaya. (2017). *Algoritma Pencarian Heuristik*. Departemen Teknik Informatika dan Komputer.

Suyanto. (2021). *Artificial intelligence: Searching, reasoning, planning, dan learning*. Informatika Bandung.

TERIMA KASIH