Bahasan Terakhir...

- Pencarian dan Klasifikasinya
- *Breadth-first Search*
- *Depth-first Search*
- Variasi *Breadth* & *Depth-first Search*:
  - *Backtracking*
  - *Depth Bounded/Limited*
  - *Iterative Deepening (Breadth + Depth First)*
  - *Branch & Bound (Uniform Cost)*
  - *Bidirectional*
- Kinerja Pencarian *Uninformed/Blind*

Pencarian *Branch and Bound* (*Uniform Cost Search*)

- Node dengan biaya minimum selalu di*expand*, meskipun harus kembali ke level sebelumnya.
- Saat status tujuan dicapai, proses belum tentu berhenti. Harus dievaluasi, apakah sudah optimal atau biaya minimum?.
- Solusi optimal dijamin dengan melanjutkan pembangkitan jalur-jalur lain dengan biaya yang mungkin lebih kecil. Pencarian berhenti saat diketahui bahwa semua jalur lain biayanya lebih atau sama dengan jalur solusi yang ditemukan.

Algoritma Pencarian *Branch and Bound*

Kembalikan SOLUSI atau GAGAL

Q adalah antrian prioritas, diurutkan berdasarkan biaya terkini dari awal (*start*) ke tujuan (*goal*)

Langkah 1. Tambahkan status awal (*root*) ke Q.

Langkah 2. Sampai tujuan dicapai atau Q kosong

do

Langkah 2.1 Hapus jalur (*path*) pertama dari antrian Q;

Langkah 2.2 Buat jalur baru dengan memperluas jalur pertama ke semua tetangga dari node terminal.

Langkah 2.3 Hapus semua jalur yang ber-*loop*.

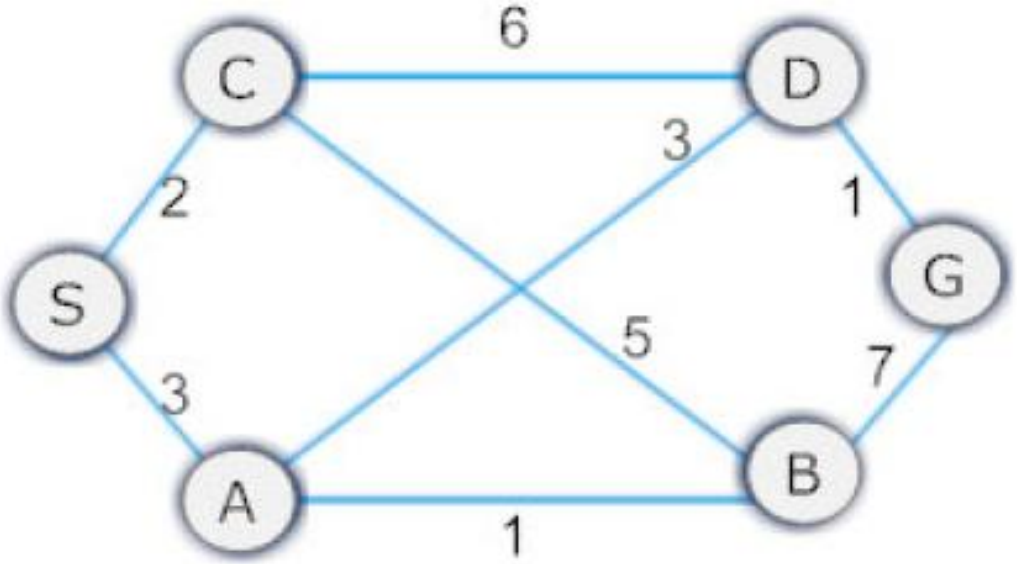
Langkah 2.4 Tambahkan jalur baru yang tersisa, jika ada, ke Q.

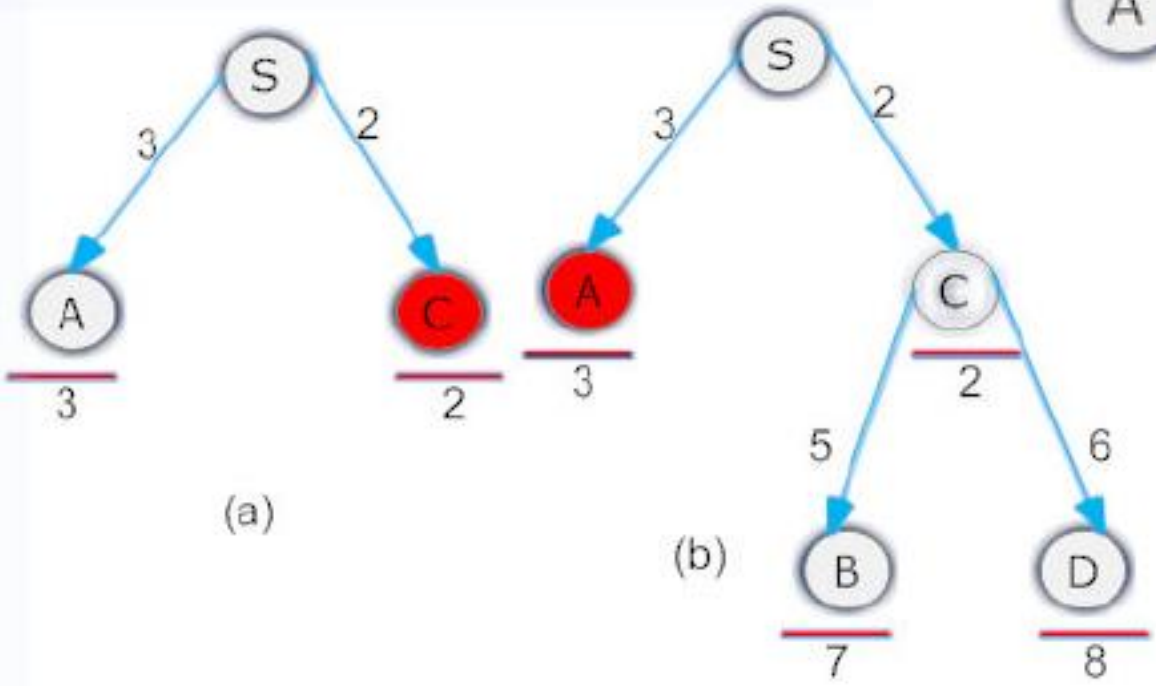
Langkah 2.5 Urutkan Q, jalur berbiaya murah ada di depan.

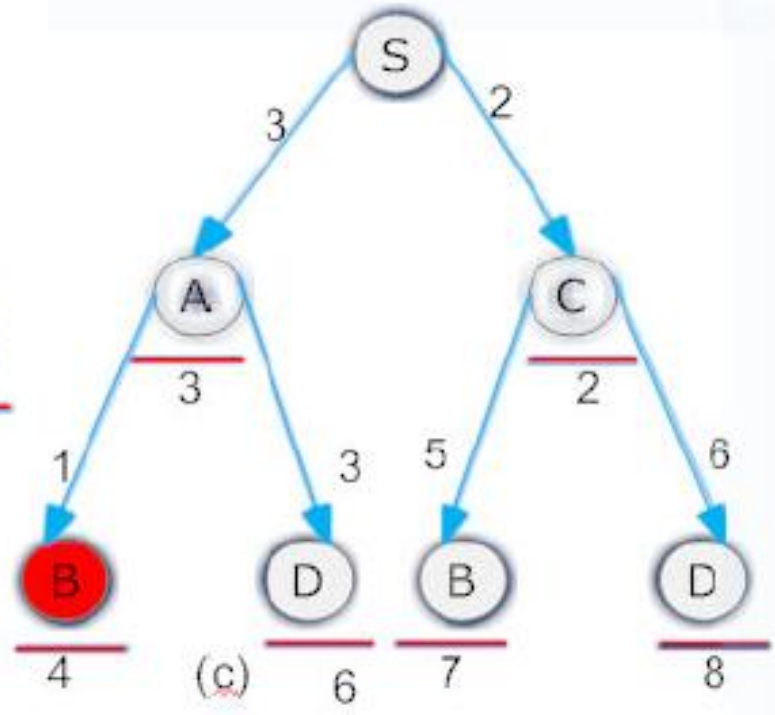
End

Contoh

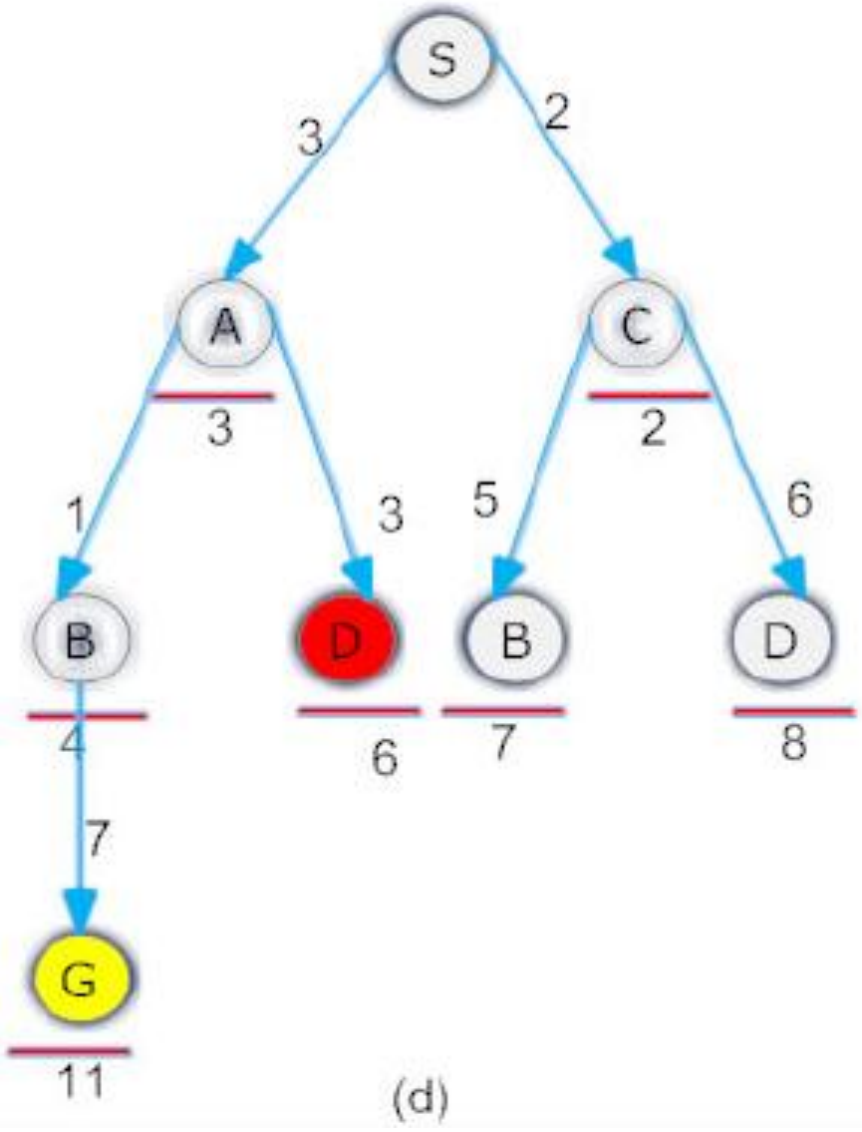
- Mana jalur terbaik?

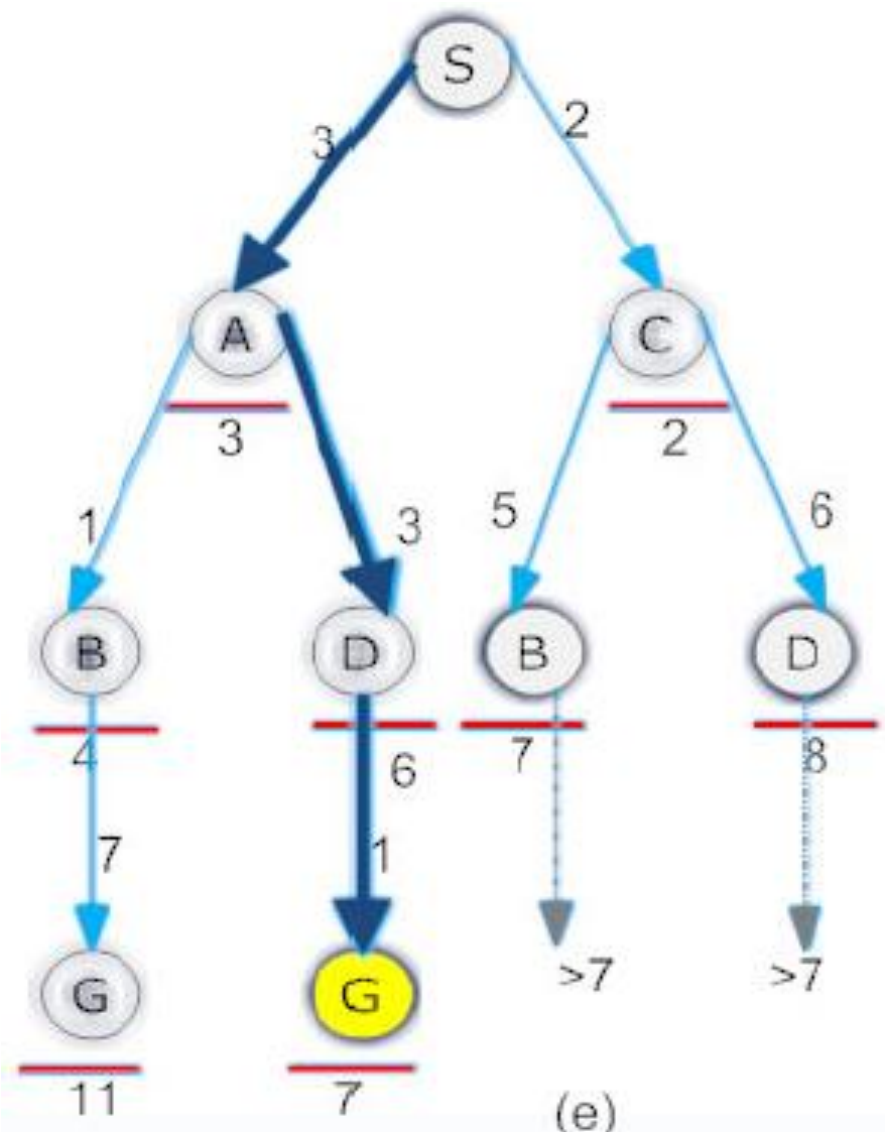






Contoh: Menentukan Jalur Terbaik





Kecerdasan Buatan

Pertemuan 03

Pencarian Heuristik (*Informed*)

Kelas 10-S1TI-03, 04, 05

Husni
Lunix96@gmail.com
http://www.facebook.com/lunix96
http://Komputasi.wordpress.com

Outline

- Pendahuluan
- Heuristik
- Pencarian *Best First*
- Pencarian *Greedy*
- Pencarian A*
- Variasi A*
- Rangkuman
- Tugas

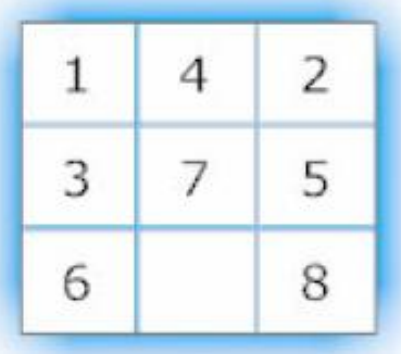
Pendahuluan

- *Informed search* (pencarian terarah, pencarian heuristik), mencoba mengurangi jumlah pencarian dengan memilih secara cerdas node-node yang harus dibuka (di*expand*).
- Pengetahuan spesifik masalah digunakan untuk menemukan solusi lebih cepat.
- Diperlukan suatu antrian berurut.
- Urutan ekspansi ditentukan oleh fungsi evaluasi. Setiap node yang terlibat mengembalikan suatu bilangan yang menunjukkan harapan atau potensi dari node tersebut.
- Fungsi evaluasi, disebut fungsi heuristik, komponen paling penting di sini, memperkiraan biaya jalur termurah dari status terkini ke status tujuan.

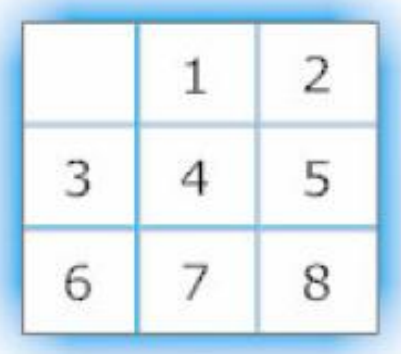
Definisi Heuristik

- Kajian mengenai metode dan aturan penemuan atau rekaan.
- Strategi pencarian cerdas untuk menyelesaikan masalah komputer
- Heuristik dapat berupa suatu *rule of thumb* yang digunakan untuk memandu suatu aksi.

Contoh



Current state



Goal state

- h1(n) = jumlah kotak yang tidak sama dengan status tujuan (*goal state*).

  h1(n)= 3 karena kotak 1, 4 dan 7 diluar status tujuan.
- h2(n) : Jumlah jarak/pergeseran yang diperlukan untuk mencapai status tujuan.

  Kotak 1, perlu 1 langkah. Kotak 2 tetap, 0. kotak 7 perlu 1 langkah, dst... Jadi:

  h2(n) = 1+0+0+1+0+0+1+0 = 3
- Nilai h1(n) atau h2(n) makin kecil adalah harapan. Nilai 0 mewakili status tujuan.

Definisi-Definisi

- **Definition 1:**

  A heuristic function h(n) is called admissible if for all nodes one has h(n) < k(n) where k(n) is the actual distance to the goal from n.

- **Definition 2:**

  Let h1(n) and h2(n) be two heuristic functions, both admissible. If h2(n) ≥ h1(n) for all n then h2(n) **dominates h1(n) and is better for search.**

Best First Search

- *Best first search* menggunakan fungsi evaluasi dan selalu memilih node berikutnya yang bernilai terbaik (*the best score*).
- Antrian digunakan sebagaimana pada pencarian breadth/depth first. Tidak mengambil node pertama (dan membangkitkan successornya). Tetapi mengambil node terbaik *(*atau diurutkan dan kemudian diambil node pertama).
- S*uccessor* dari node terbaik akan dievaluasi (diberi nilai) dan ditambahkan ke list.
- Suatu fungsi biaya f(n) diberlakukan terhadap setiap node.
- Node-node diletakkan dalam OPEN sesuai urutan nilai f-nya.
- Node-node dengan nilai f(n) lebih kecil di*expand* lebih awal.
- Dua pendekatan mendefinisikan fungsi f:
  - Mengekspan node yang paling dekat ke tujuan, atau
  - Mengekspan node pada jalur solusi *least-cost* (biaya paling murah)

Algorithm *Best first search*

Langkah 1. Buat antrian prioritas Q dan masukkan status awal.

Langkah 2. Sampai Q kosong atau Gagal

Langkah 2.1 **If** Q kosong **return** Gagal

Langkah 2.2 Hapus node pertama dari Q (pindahkan dari daftar OPEN ke dalam daftar CLOSED)

Langkah 2.3 **If** node pertama adalah tujuan **then return** jalur ke dirinya dari status awal

Else bangkitkan semua *successor* dari node tersebut dan letakkan ke dalam Q sesuai nilai f(n)-nya (yang terbaik di depan).

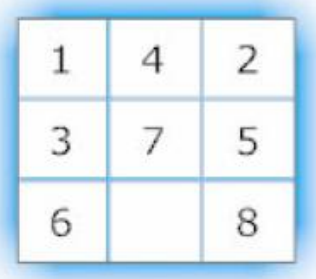
Langkah 3. **If** suatu solusi ditemukan, **return** solusi tersebut, **else return** Gagal.

Pencarian *Greedy*

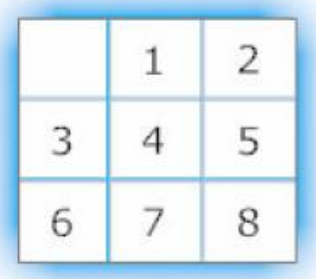
- Ekspansi node dengan biaya diperkirakan paling kecil untuk mencapai tujuan (atau node yang terlihat paling dekat ke tujuan).
- Fungsi heuristiknya adalah:

  $f(n) = h(n)$
- Dimana $h(n)$ perkiraan/estimasi jarak tersisa ke tujuan.

Contoh 1: 8-puzzle



Current state



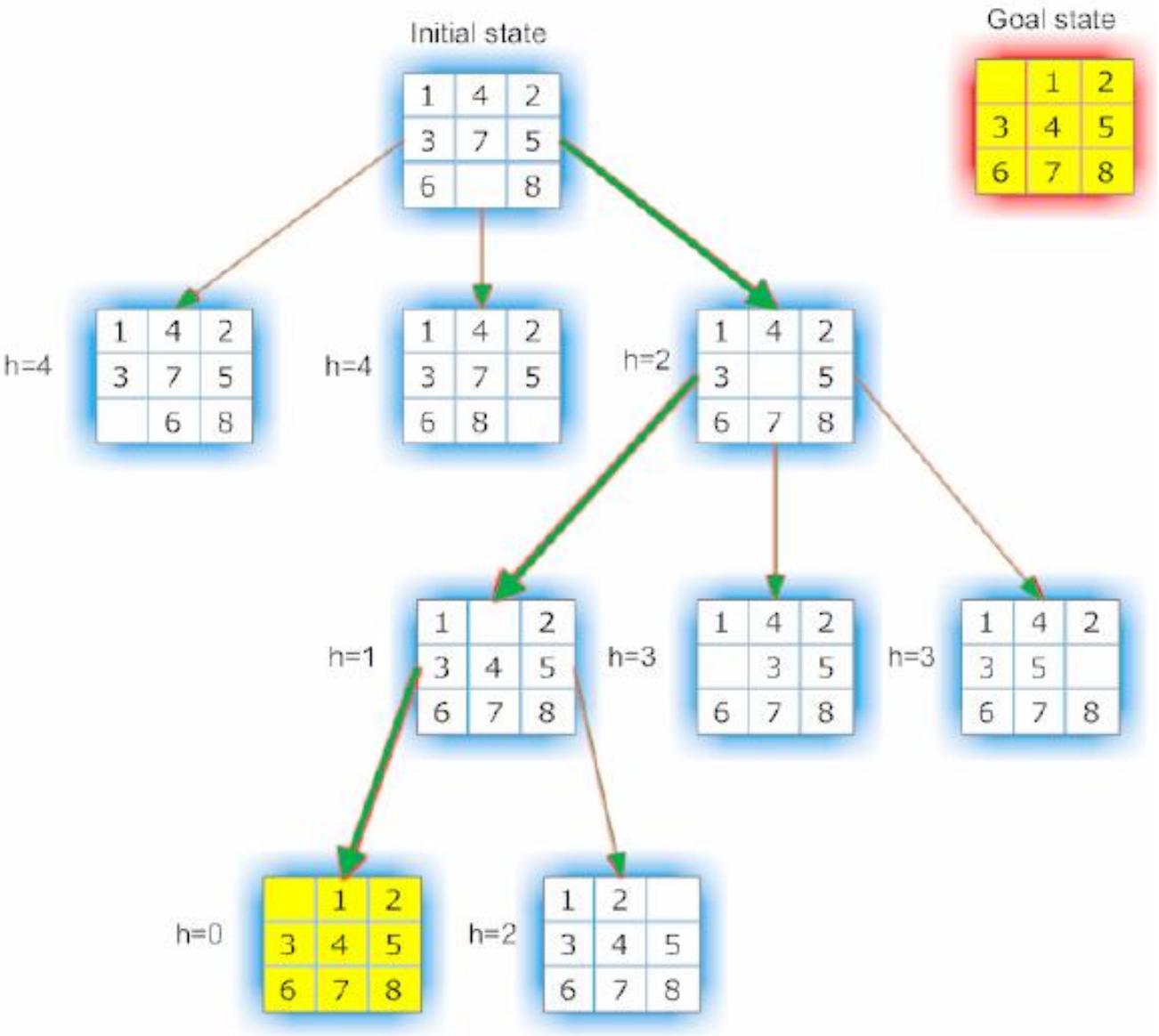
Goal state

- Digunakan dua fungsi heuristik:
  - banyaknya kotak yang salah tempat atau
  - jumlah geser yang diperlukan semua kotak untuk mencapai tujuan (jarak Manhattan).

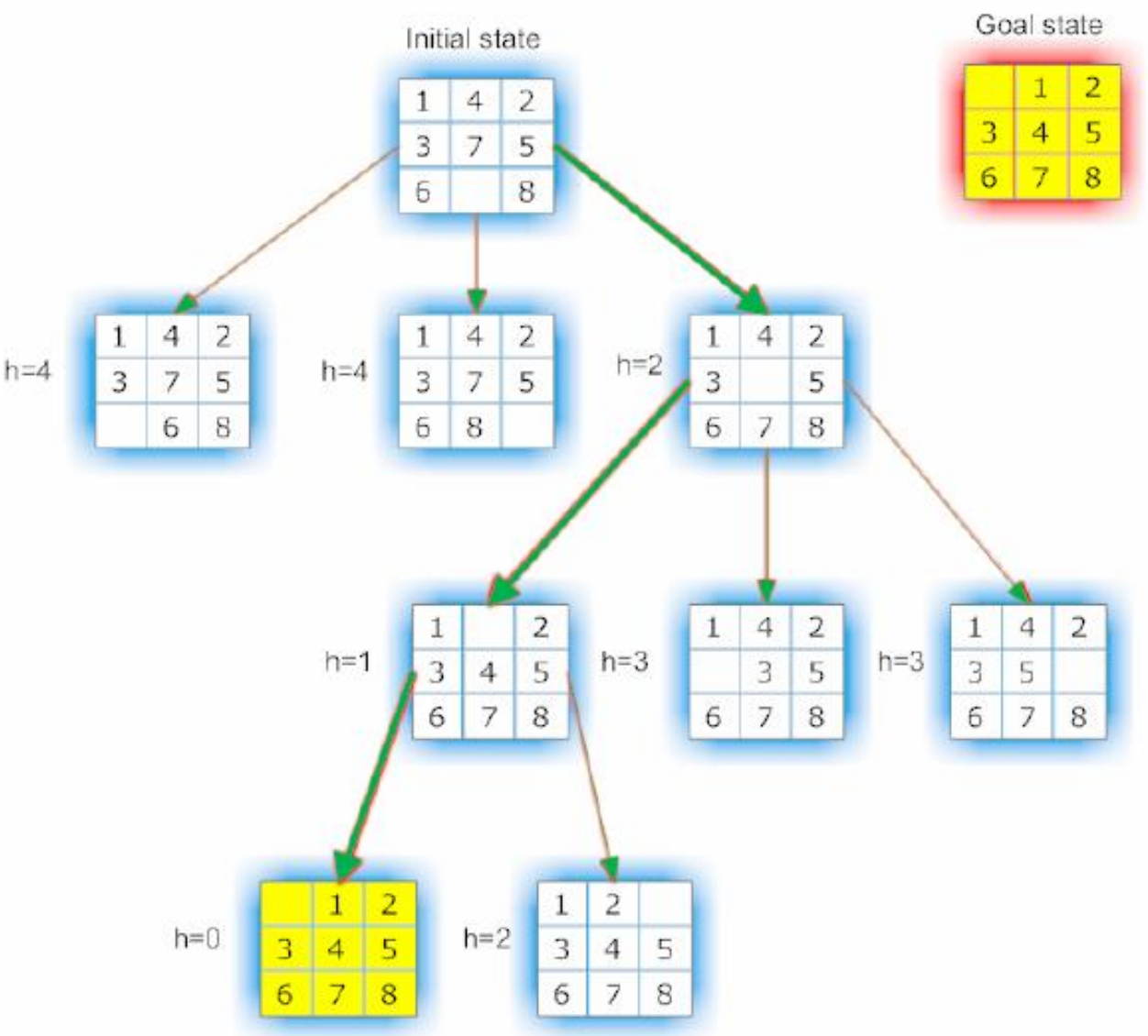
Menggunakan fungsi heuristik Jumlah Kotak Salah Tempat:

- Ada 3 kemungkinan gerakan dari status awal: kosong gerak ke kiri, ke kanan dan ke atas
- Saat kosong digeser ke kiri, status baru: 3 kotak salah tempat (1, 4 dan 7).
- Jika kosong digeser ke kanan, terdapat 4 kotak salah tempat (1, 4, 7, dan 8).
- Ketika kosong digeser ke atas, diperoleh status 2 kotak salah tempat (1 dan 4).
- Algoritma Greedy akan memutuskan untuk mengikuti jalur yang paling dekat dengan status tujuan.
- **Dst....**

Heuristik 1: Jumlah Kotak Salah Tempat/Posisi

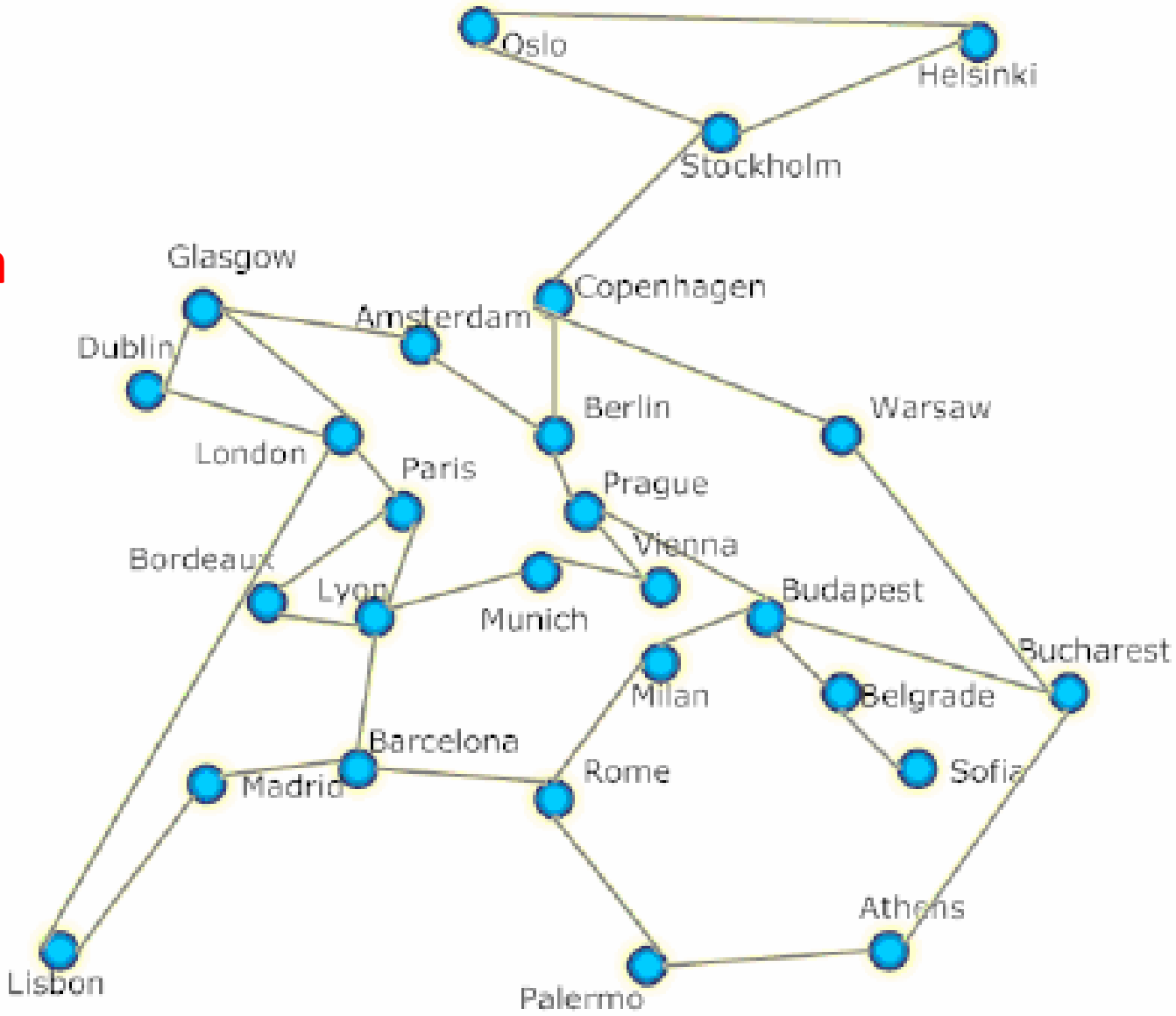


Heuristik 2: Jumlah Pergeseran Semua Kotak Menuju Tujuan



Contoh 2: Jarak Terpendek

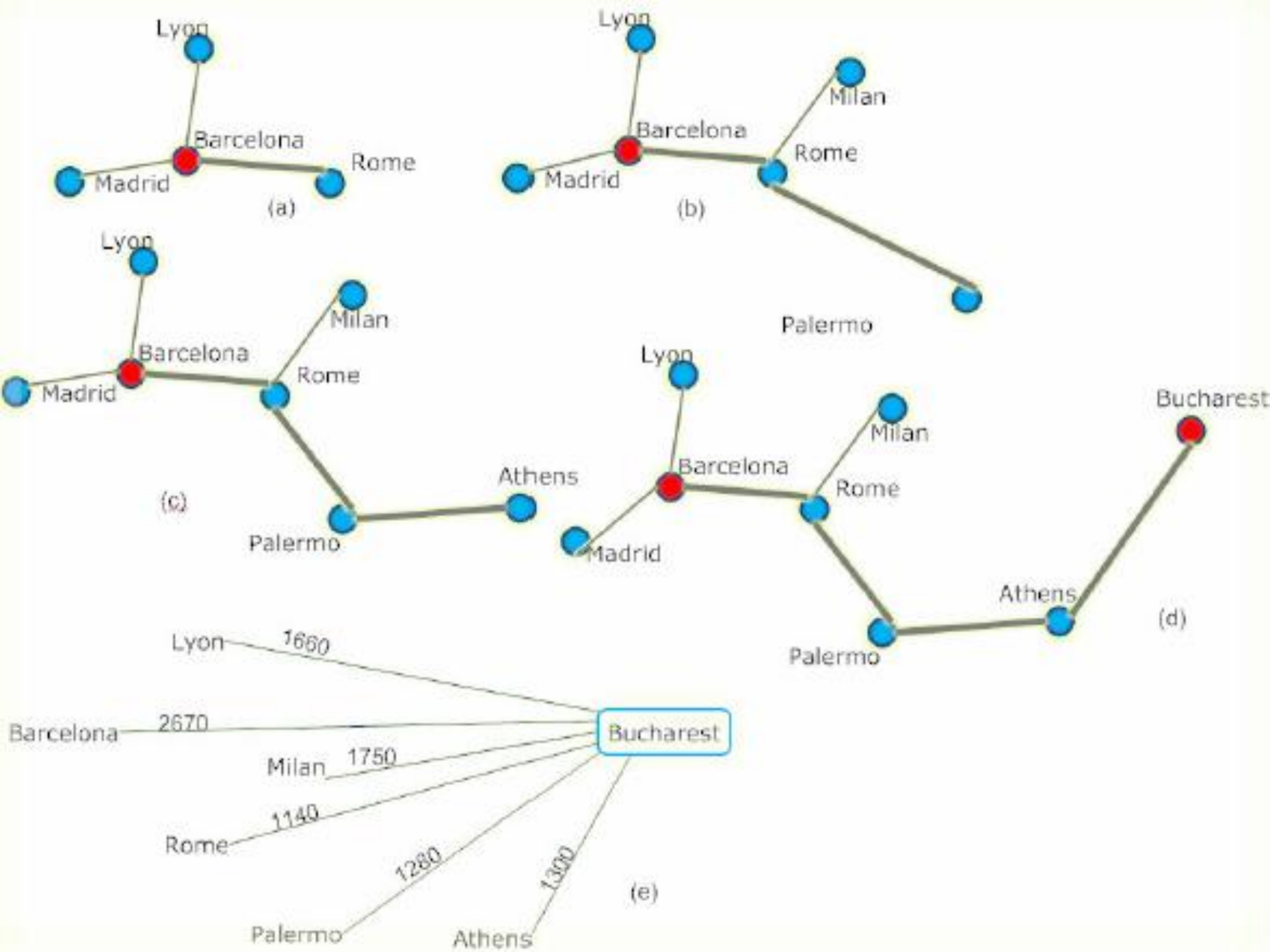
- Tentukan jalur terpendek dari **Barcelona** ke **Buchrest**



Kota dan Jarak langsung ke Bucharest			
Amsterdam	2280	Lyon	1660
Athens	1300	Madrid	3300
Barcelona	2670	Milan	1750
Belgrade	630	Munich	1600
Berlin	1800	Oslo	2870
Bordeaux	2100	Palermo	1280
Budapest	900	Paris	2970
Copenhagen	2250	Prague	1490
Dublin	2530	Rome	1140
Glasgow	2470	Sofia	390
Helsinki	2820	Stockholm	2890
Lisbon	3950	Vienna	1150
London	2590	Warsaw	946

Jarak Langsung Antar Kota					
Oslo-Helsinki	970	Rome-Milan:	681	Madrid -Barcelona	628
Helsinki-Stockholm	400	Milan-Budapest	789	Madrid-Lisbon	638
Oslo-Stockholm	570	Vienna-Budapest	217	Lisbon-London	2210
Stockholm-Copenhagen	522	Vienna-Munich	458	Barcelona-Lyon	644
Copenhagen-Warsaw	668	Prague-Vienna	312	Paris-London	414
Warsaw-Bucharest	946	Prague-Berlin	354	London-Dublin	463
Bucharest-Athens	1300	Berlin-Copenhagen	743	London-Glasgow	667
Budapest-Bucharest	900	Berlin-Amsterdam	648	Glasgow-Amsterdam	711
Budapest-Belgrade	316	Munich-Lyon	753	Budapest-Prague	443
Belgrade-Sofia	330	Lyon-Paris	481	Barcelona-Rome	1471
Rome-Palermo	1043	Lyon-Bordeaux	542	Paris-Bordeaux	579
Palermo-Athens	907			Glasgow-Dublin	306

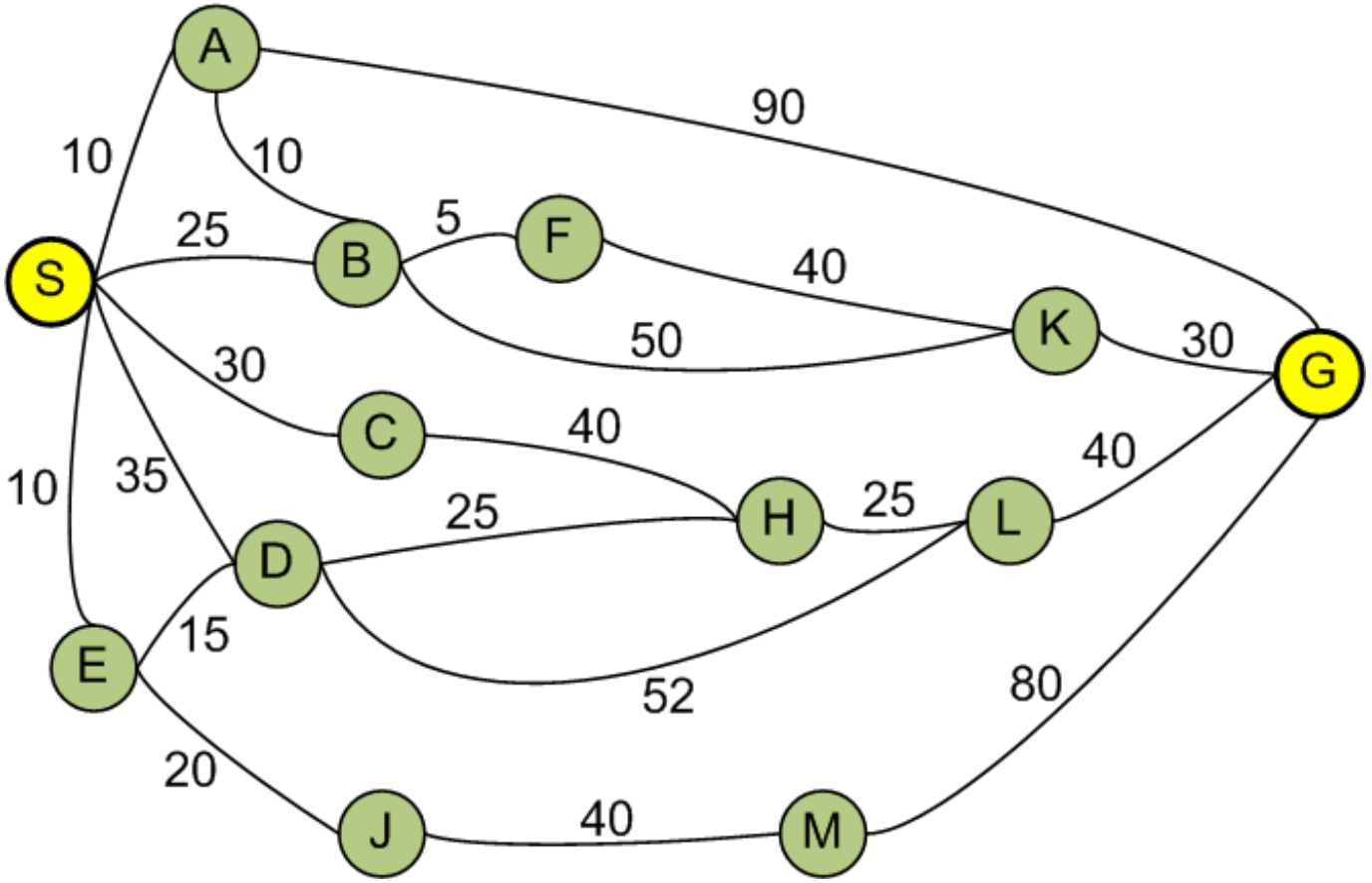
- Temukan jalur terpendek Barcelona – Bucharest!
- Solusi Greedy: **Barcelona – Rome – Palermo – Athens – Bucharest**, dengan biaya 1471+1043+907+1300 = **4,721**


Lyon
Barcelona
Madrid
Rome
(a)
Lyon
Milan
Barcelona
Rome
Madrid
(b)
Palermo
Lyon
Milan
Barcelona
Rome
Madrid
Athens
(c)
Palermo
Lyon
Bucharest
Milan
Barcelona
Rome
Madrid
Athens
(d)
Palermo
Lyon
1660
Barcelona
2670
Bucharest
Milan
1750
1140
Rome
1280
1300
(e)
Palermo
Athens

Penjelasan Contoh 2

- Mulai dari **Barcelona**. Tujuan? Bukan. Buka node-node yang tersambung dengan Barcelona. Diperoleh **Madrid**, **Lyon** dan **Rome**.
- Dari 3 node baru, pilih yang terdekat dengan Bucharest, yaitu **Rome**.
- Apakah Rome tujuan? **Bukan**. Buka node-node yang mungkin. Diperoleh **Milan** dan **Palermo**.
- Mana yang terdekat dengan Bucharest? **Palermo**.
- Buka node **Palermo**. Diperoleh **Athens**.
- Apakah **Athens** tujuan? Bukan. Buka node anaknya, diperoleh **Bucharest**.
- Apakah Bucharest tujuan? **Ya**. **Selesai**.

Mana Rute Terpendek S menuju G?



Jarak Garis Lurus Kota n ke Kota G													
n	S	A	B	C	D	E	F	G	H	J	K	L	M
h(n)	80	80	60	70	85	74	70	0	40	100	30	20	70

Solusi dengan Greedy

- Hanya menghitung berapa jarak lurus kota ke tujuan (h(n)). Misal: h(A) = 80.
- Fungsi heuristik: f(n) = h(n)
- Bangkitkan semua node (anak) dari root (status awal). Node mana yang mempunyai h(n) terkecil? Ambil.
- Bangkitkan node-node dari h(n) terkecil tersebut. Ambil h(n) terkecil dari anak-anaknya.
- Lakukan terus sampai status tujuan dicapai. Apakah jalur yang diperoleh optimal? Terbaik?
- Tidak boleh kembali ke level sebelumnya.

Contoh 3: Penukaran Koin

- Misalkan C = {c1, c2, …, ck} suatu himpunan berhingga satuan koin berbeda. Buat asumsi:
  - Setiap ci, i=1,.., k adalah integer dan c1 > c2 > ... > ck;
  - Setiap satuan tersedia dalam jumlah tak terbatas.
- Masalah: **tukarkan suatu nominal N dengan jumlah koin minimal**!
- Fungsi heuristik dasar: pilih secara berulang koin yang paling besar yang kurang atau sama dengan sisa nomimal, sampai nominal total yang diharapkan dicapai.

Greedy: Bukan Solusi Optimal

- Jika himpunan koin {25, 10, 5, 1} dan nomimal adalah 30, metode greedy selalu mendapatkan solusi optimal, yaitu **1 x 25 + 0 x 10 + 1 x 5 (2 koin)**
- Tetapi jika nominal yang diharapkan adalah 30 dan koin {25, 10, 1} maka Greedy mengembalikan:

  1 × 25 + 0 ×10 + 5 × 1 (6 koin)
- Padahal solusi optimal:

  0 × 25 +3 × 10 + 0 × 1 (3 koin).
- Pencarian Greedy juga tidak menemukan solusi optimal untuk himpunan K {12, 5, 1} dan N = 15. Greedy memberikan solusi **1 × 12 + 0 × 5 + 3 × 1**, padahal yang optimal adalah **0 × 12 + 3 × 5 + 0 × 1** (3 koin, bukan 4).

Rangkuman: Teknik Pencarian *Greedy*

- Dimaksudkan untuk menemukan solusi dengan cepat, meskipun tidak selalu optimal;
- Dapat terjebak loop (perulangan), tidak selsai;
- Tidak *admissible*; kadang heuristiknya *underestimate*;
- Jika terlalu banyak node, pencarian dapat menjadi *exponential*;
- Kompleksitas waktu terburuknya sama dengan pencarian *depth first*;
- Kompleksitas ruang terburuknya sama dengan pencarian *breadth first*;
- Heuristik yang bagus dapat memberikan perbaikan signifikan;
- Pencarian Greedy cocok untuk masalah kecil jawaban cepat.

Pencarian A*

- Mengkombinasikan pencarian uniform cost dan Greedy dalam hal menggunakan antrian berurut prioritas (biaya) dan fungsi evaluasi untuk menentukan pengurutan.
- Fungsi evaluasi/heuristik f:

  f(n)= h(n) + g(n)

  Dimana:
  - h(n) adalah fungsi heuristik yang memperkirakan biaya dari n ke tujuan dan g(n) biaya mencapai n dari titik awal.
  - Jadi, f(n) memperkirakan biaya jalur melewati n ke tujuan.

Algorithm 3.2. A* search

Langkah 1. Buat antrian jalur parsial Q (awalnya root ke root, panjang 0);

Langkah 2. While Q Tidak kosong atau Gagal

Langkah 2.1 if jalur pertama P mencapai node tujuan then return sukses

Langkah 2.2. Hapus jalur P dari Q;

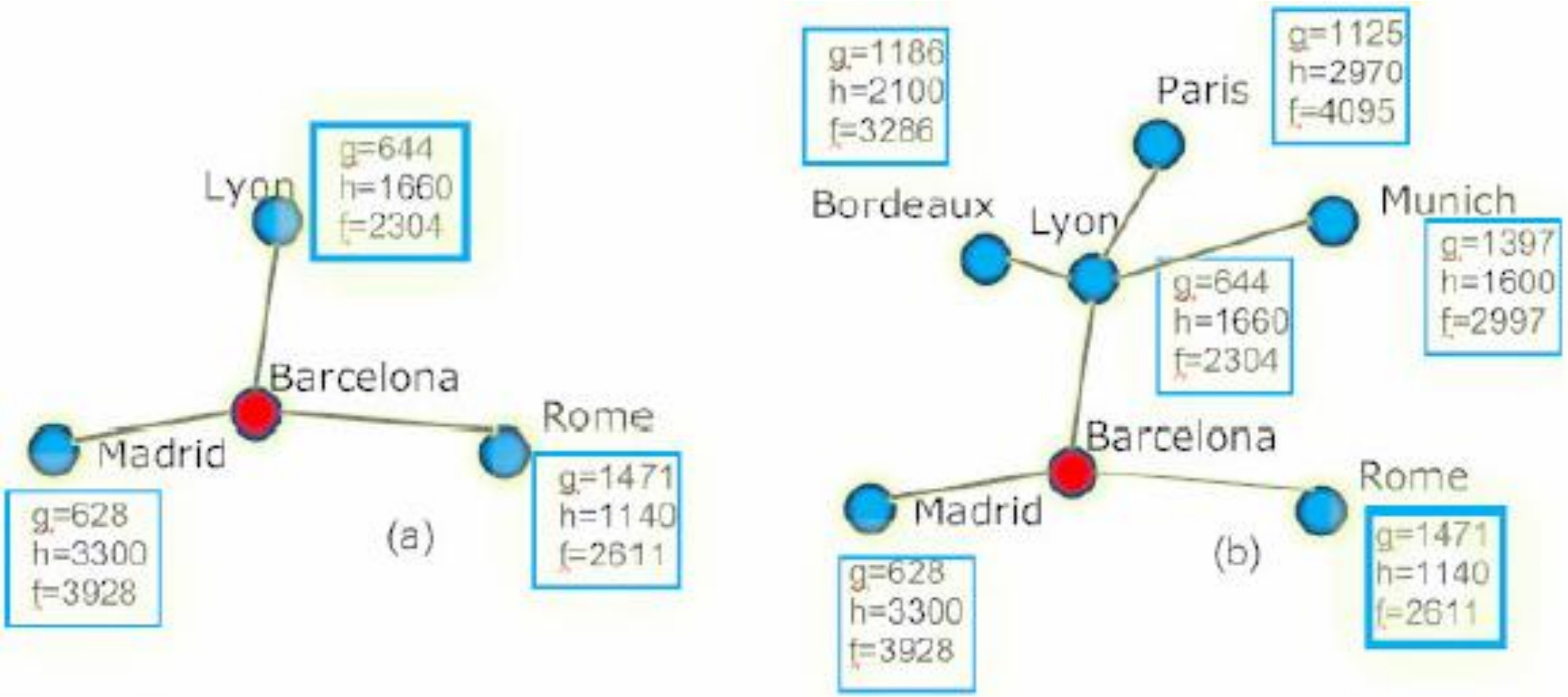
Langkah 2.3 Buka P yang mungkin & tambahkan jalur baru ke Q

Langkah 2.4 Urutkan Q dengan menjumlahkan dua nilai: Biaya dari P sampai sekarang (g) fdan estimasi jarak tersisa (h);

Langkah 2.5 Pangkas (*prune*) Q dengan meninggalkan hanya jalur terpendek bagi setiap node yang dicapai sejauh ini;

Langkah 3. Return jalur terpendek (jika berhasil) atau Gagal.

Contoh 4: Jalur Terpendek (Langkah 1 & 2)



- (a) Barcelona di*expand*, f(Lyon) paling kecil.
- (b) Lyon di*expand.*

Penjelasan

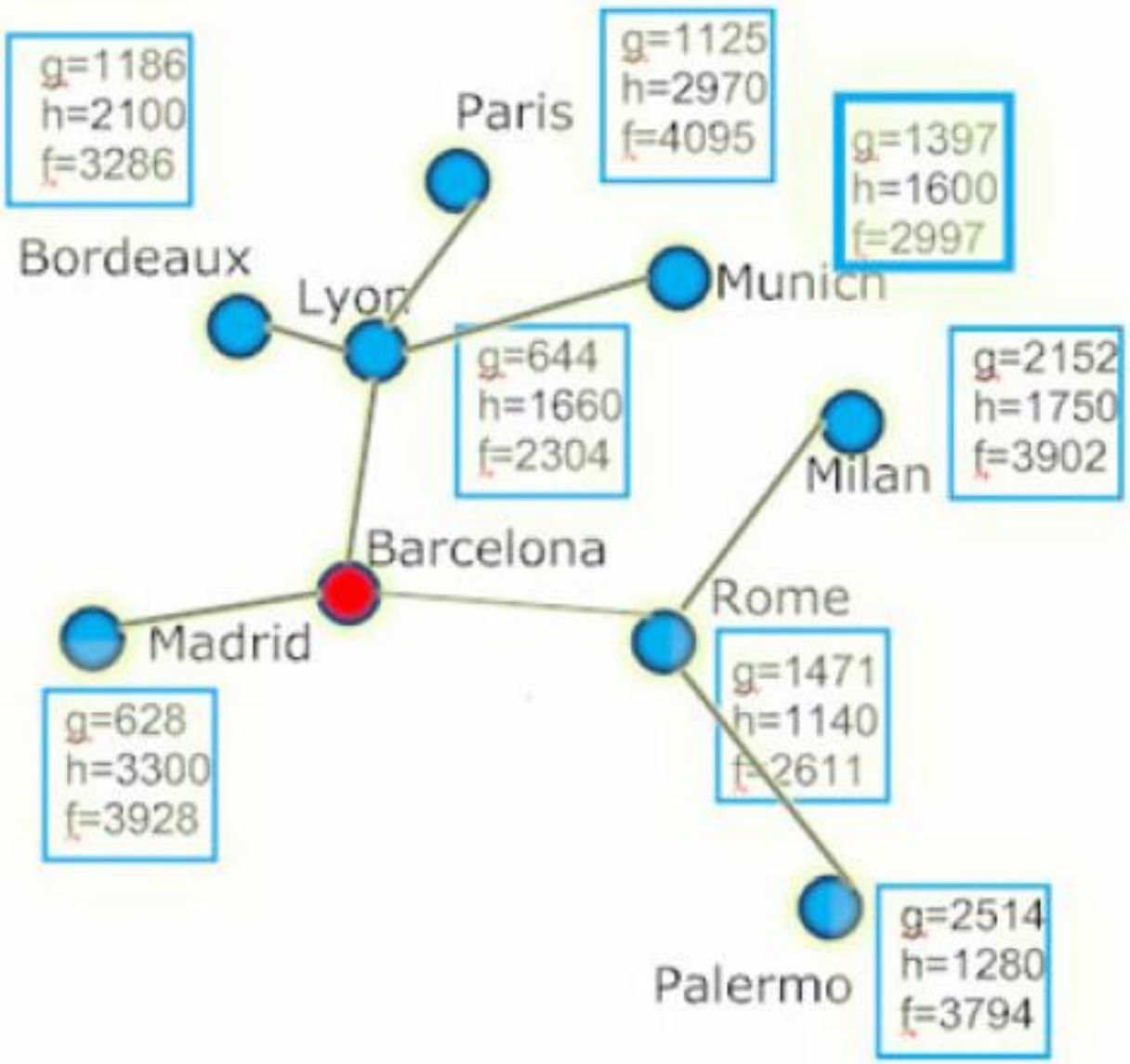
- Setiap node n harus menghitung 3 entitas: h(n): jarak langsung ke Bucharest dari node n, g(n): biaya jalur sampai di status terkini. Dan f(n) dengan formula f(n) = g(n)+h(n).
- Untuk Madrid, diperoleh:
  - g = 628 (jarak dari Barcelona ke Madrid)
  - h=3,300 (jarak garis lurus dari Madrid ke Bucharest)
  - f= 3,300+628=3,928

  Lyon: g=644, h=1,660 dan f=2,304;

  Rome: g=1,471, h=1,140 dan f=2,611;
- Karena Lyon adalah node dengan nilai f terendah, Lyon akan di*expand* berikutnya.

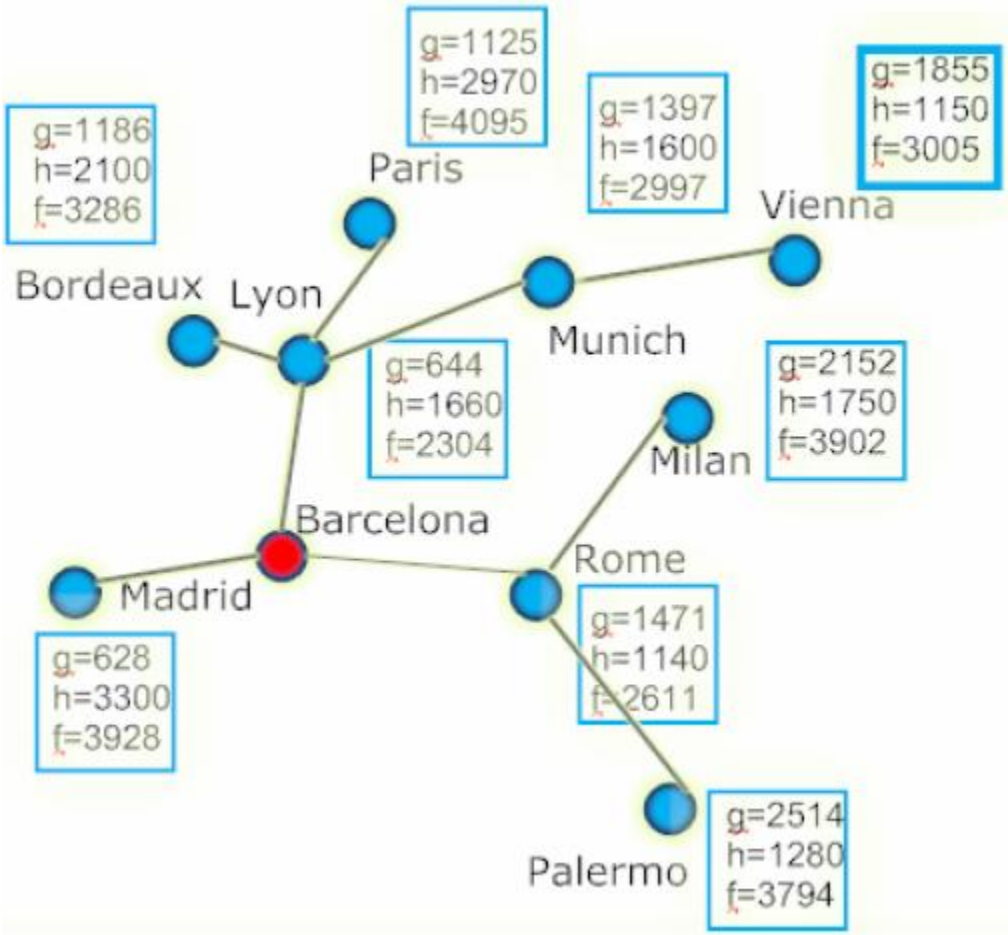
Contoh 4: Jalur Terpendek (Langkah 3)

- Roma di*expand*, karena f(Roma) lebih kecil daripada f(Munich)



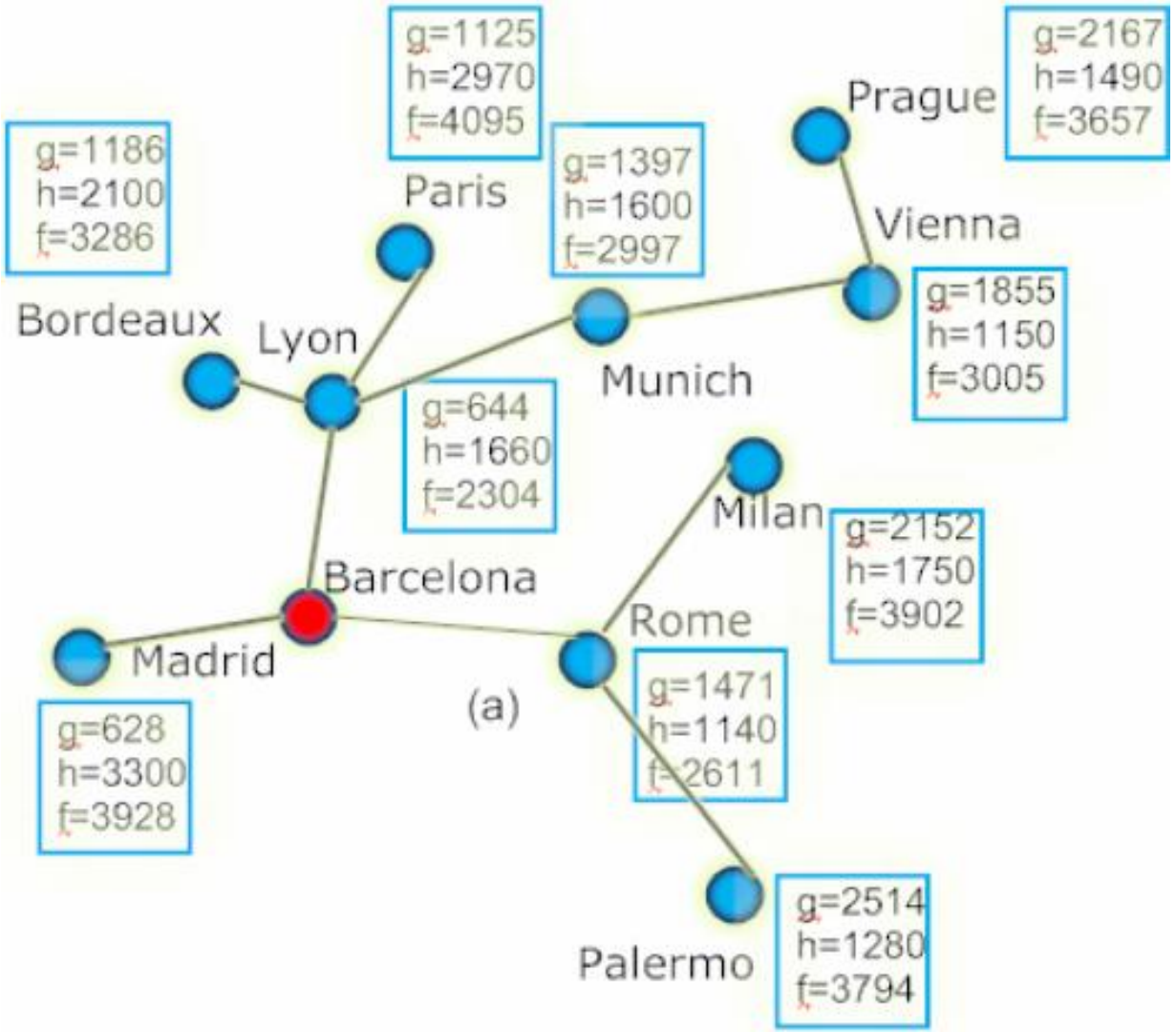
Contoh 4: Jalur Terpendek (Langkah 4)

- F(Munich) lebih kecil daripada f(Palermo)
- Munich di*expand*



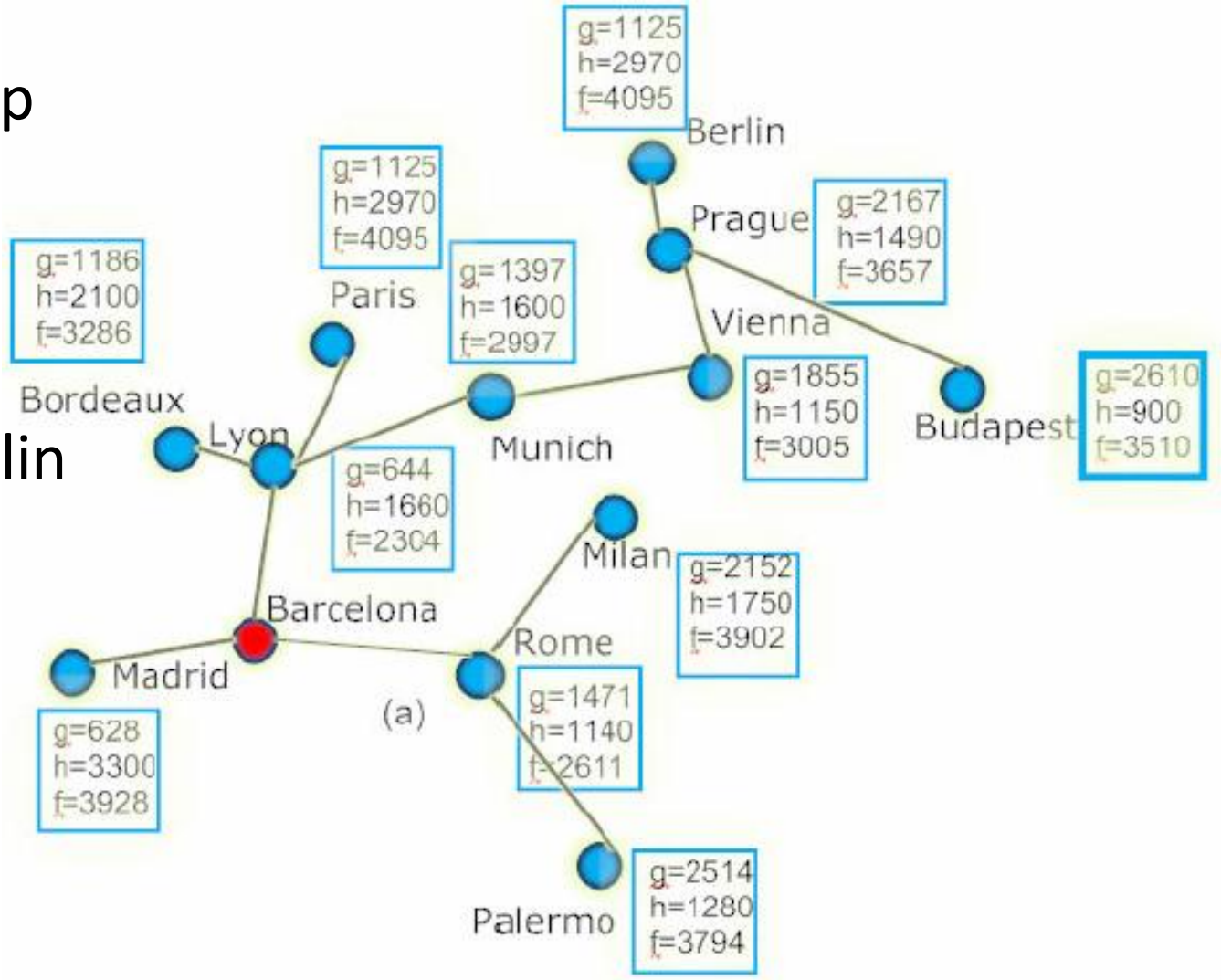
Contoh 4: Jalur Terpendek (Langkah 5)

- f(Vienna) paling kecil
- Node Vienna di*expand*
- Diperoleh Prague

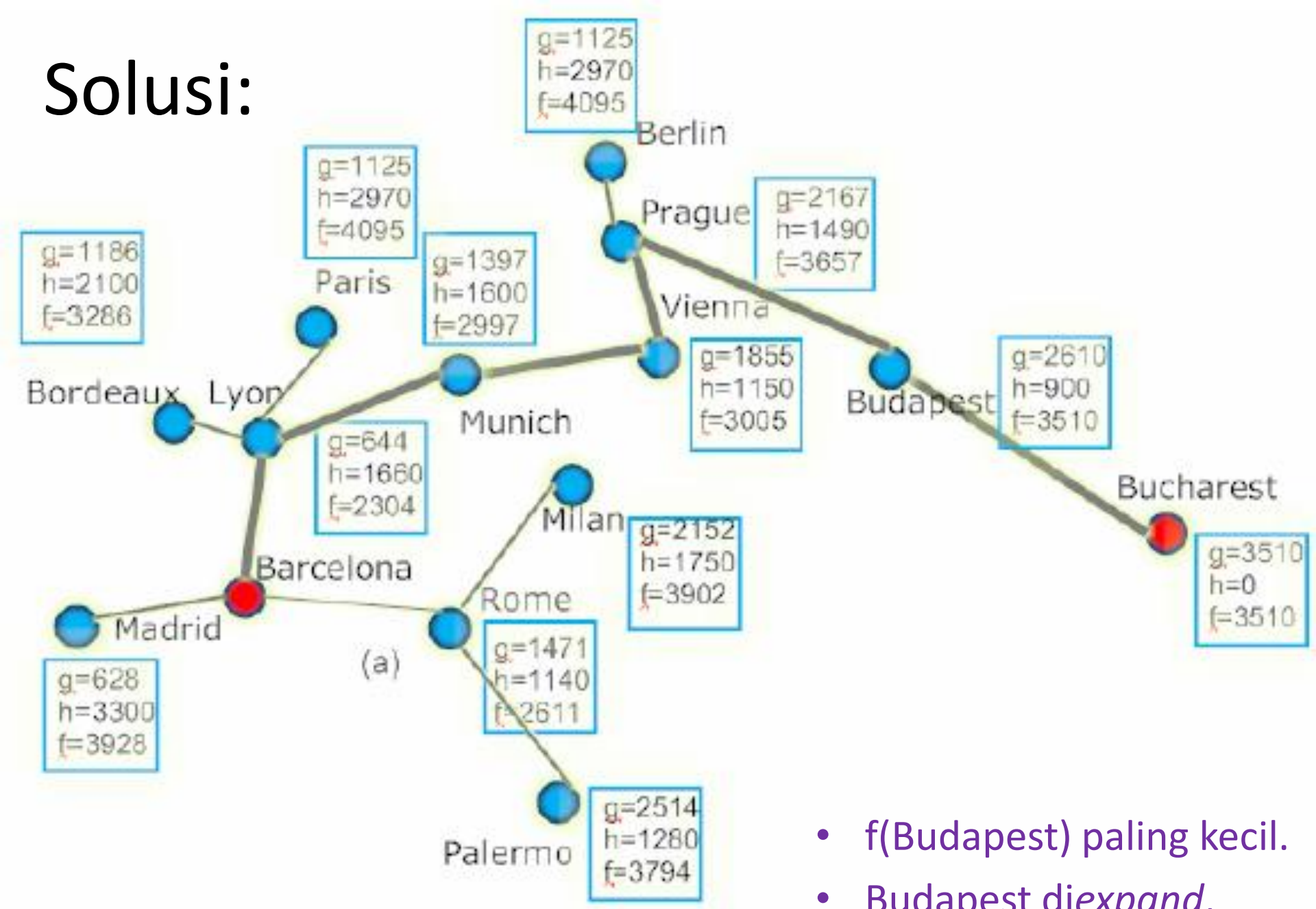


Contoh 4: Jalur Terpendek (Langkah 6)

- f(Prague) tetap paling kecil
- Node Prague di*expand*
- Diperoleh Berlin dan Budapest



Solusi:



- f(Budapest) paling kecil.
- Budapest di*expand*, Diperoleh Bucharest

Solusi Jalur Terpendek S ke G dengan A*

- Evaluasi suatu status melibatkan jarak langsung ke tujuan dan jarak yang telah ditempuh dari status awal.
- Fungsi heuristik: f(n) = h(n) + g(n)
- Misal: f(A) = 80 + 10 = 90.
- Bangkitkan semua node dari status awal. Hitung f(n) untuk setiap node. Berapa f(n) terkecil? Ambil.
- Bangkitkan semua node dari node dengan f(n) terkecil tersebut. Hitung f(n) dari semua anaknya. Lihat semua node yang sudah dibuka. Mana node yang mempunyai f(n) terkecil? Ambil.
- Lanjutkan... seperti pada pencarian *uniform cost*.

Varian dari A*

- Pencarian A* memerlukan banyak memory, seperti pencarian *breadth first*.
- Berbagai varian A* dikembangkan untuk mengatasi masalah ini, seperti:
  - *Iterative deepening* A* (IDA*)
  - *Memory-bounded* A* (MA*)
  - *Simplified memory bounded* A* (SMA*)
  - *Recursive best-first search* (RBFS) dan
  - *Dynamic* A* (D*).

Pencarian *Recursive Best-First* (RBFS)

- Membuka node-node dalam urutan best-first
- Mirip depth first search, jejak f dari jalur alteratif terbaik
- Jika node terkini melampaui limit, rekursi menuju jalur alternatif, mengganti nilai f setiap node bersama path dengan f terbaik dari nak-anaknya.
- Juga mengingat f dari daun terbaik yang “terlupakan”

Algoritma RBFS

RBFS (node: n ,limit l)

if f(n) > l **return** f(n)

if n adalah tujuan, **then** *exit*

if n tidak punya anak, **return** *infinity*

else for each anak n_i dari n, set $f_i = f(n_i)$

Urutkan n_i dan f_i (*ascending* berdasarkan nilai f_i)

if hanya ada satu anak **then** f_2 = *infinity*

while $f_1 \leq l$ dan $f_1 <$ *infinity*

$f_1 = RBFS(n_1, min(l, f_2))$

sisipkan n_1 dan f_1 dalam daftar terurut (*sorted list*)

return f_1

Pencarian *Beam*

- Pada perulangan A*, daftar OPEN menyimpan semua node masih dapat digunakan untuk menemukan suatu jalur.
- Pencarian Beam membatasi ukuran OPEN. Daftar/list terlalu besar dapat menyebabkan lepasnya node-node yang dianggap buruk (padahal mungkin menuju ke solusi terbaik)
- Pencarian Beam menggunakan heuristik f(n) = g(n) + h(n). Tetapi diparameterkan dengan integer positif k (seperti pencarian *depth limited*, berparameter l).
- S*uccessors* dari suatu *node* dihitung, hanya k anak dengan nilai f(n) terbaik (terkecil) yang dimasukkan dalam agenda (*list* OPEN).
- Tidak dijamin lengkap dan optimal.

Rangkuman

- Pencarian *Informed* memanfaatkan pengetahuan *problem-specific* untuk memandu proses pencarian dan ini memberikan peningkatan signifikan pada kinerja pencarian.
- Telah dipelajari konsep fungsi heuristik dan beberapa pendekatan pencarian *well known* seperti Greedy dan A* (termasuk beberapa variannya).
- Prakteknya, ingin diperoleh tujuan dengan jalur berbiaya minimum, dapat tersusun oleh pencarian lengkap untuk masalah tertentu tetapi tidak mungkin untuk masalah lain.
- Heuristik yang memperkirakan biaya jalur dari suatu node ke tujuan dapat diefisienkan dengan mengurangi ruang pencarian.
- Heuristik dapat meningkatkan kecepatan pencarian buta yang lengkap, seperti *depth first* dan *breadth first*.
- Menemukan heuristik terbaik adalah tantangan: semakin baik heuristik maka semakin baik metode pencariannya.

Rangkuman

- Pencarian Greedy memilih perkiraan biaya minimal ke tujuan, *f(n)*, dan menurunkan waktu pencarian tetapi tidak lengkap, tidak optimal.
- Jika *h(n)* adalah fungsi heuristik *admissible*, pencarian A* lengkap optimal.
- Ruang memory adalah kekurangan utama pencarian A* (bukan kompleksitas waktu) karena menampung semua node yang dibangkitkan dalam memory.
- Beberapa varian pencarian A* diusulkan untuk mengatasi masalah di atas, seperti *iterative deepening* A* (IDA*), *memory-bounded* A* (MA*), *simplified memory bounded* A* (SMA*), *recursive best-first search* (RBFS) dan *Dynamic* A* (D*).
- Heuristik yang *admissible* bersifat *optimistic*: biaya penyelesaian masalah kurang dari sesungguhnya.
- Algoritma pencarian dan heuristik dapat memberikan solusi berbeda.
- Kualitas heuristik diukur dengan *effective branching factor*. Heuristik yang dirancang baik mempunyai nilai *branching factor* mendekati 1.

Tugas

- Gunakan pendekatan RBFS untuk menyelesaikan masalah penentuan jalur terpendek S ke G di atas.
- Dapat dikumpulkan via email **lunix96@gmail.com**