Sebelumnya...

- **Best-First Search**
- **Greedy Search**
- **A*** Search, karena boros memory, dimunculkan variannya (sekilas):
  - IDA*
  - SMA*
  - D* (DWA*)
  - RBFS
  - Beam

Kecerdasan Buatan

Pertemuan 04

Variasi A* dan Hill Climbing

Kelas 10-S1TI-03, 04, 05

Husni

Lunix96@gmail.com

http://Komputasi.wordpress.com

Outline

- Variasi A*
  - **IDA***
  - **SMA***
  - **D* (DWA*)**
- Pencarian Iteratif
  - ***Hill Climbing***
  - *Simulated Annealing*
  - Pencarian Tabu
  - *Means Ends*

Iterative Deepening A* (IDA*)

- IDA* merupakan gabungan *iterative deepening depth first search* dengan A*.
- Pencarian dilakukan secara iteratif. Batasan/limit pencarian didasarkan pada nilai f(n).
- Dimulai dari status awal, tentukan suatu limit awal.
- Bangkitkan semua node dari status awal tersebut. Hitung f(n) untuk setiap anak yang dibangkitkan. Mana f(n) terkecil?
- Apakah f(n) terkecil lebih kecil dari limit?
  - **IYA**. Bangkitkan semua anak dari node dengan f(n) terkecil tersebut. Lanjutkan.
  - **TIDAK**. Jadikan f(n) tersebut sebagai limit baru. Ulangi pencarian dari status awal (restart).
- Boros waktu tetapi tetap memberikan solusi terbaik.

Algoritma IDA*

Function depth_first_search(n, limit)

If f(n) > limit

Then Return f(n)

If h(n) = 0 **Then** BERHASIL

Return nilai terendah dari depth_first_search(n_i, limit) bagi semua *successor* n_i dari n

end

Procedure IDA*(n)

limit=h(n)

repeat

limit=depth_first_search(n, limit)

until BERHASIL

end

Jalur Terpendek S ke G dengan IDA*

- Mulai dari status awal. Tentukan limit, misalnya 80.
- Bangkitkan semua anak dari S. Hitung f(n). Mana f(n) terkecil? Node E, f(E) = 84.
- Karena f(n) terkecil > limit, limit = f(n) = 84. Restart.
- Bangkitkan semua anak dari status awal. Cari f(n) terkecil, diperoleh f(E)= 84. Karena f(n) belum melebihi limit, kembangkan node E (bangkitkan anak-anaknya).
- Hitung f(n) semua anak E. Apakah masih belum melebihi limit? SUDAH MELEBIHI
- Diantara semua node yang sudah dibuka, berapa f(n) terkecil? 90. limit baru = 90.
- RESTART...

Simplified Memory Bounded A* (SMA*)

- Memory yang tersedia sangat terbatas
- Menghindari ekspansi terhadap node-node yang telah diekspan sebelumnya
- Hanya menyimpan node yang paling menjanjikan
- Harus mengingat “sesuatu” untuk membangkitkan kembali yang tak disimpan dengan cepat
- Jika memory penuh dan masih perlu membangkitkan node tambahan:
  - Hapus node daun dengan nilai f tertinggi
  - Ingat nilai f dari anak terbaik (yang tak disimpan) pada setiap node induk.

Algoritma SMA*

if (status awal adalah status tujuan), **then** return status tersebut

Langkah 1. Tambahkan node root ke antrian Q.

Langkah 2. **While** Q tidak kosong atau Gagal Do

Langkah 2.1 If Q kosong return Gagal

Langkah 2.2. Ambil node berprioritas top dari Q sebagai node terkini

Langkah 2.3 **If** ini adalah status tujuan then return status ini

Langkah 2.4 Ambil *successor* dari node terkini

Langkah 2.5 If successor bukan status tujuan dan batas kedalaman yang dicapai

then set f(successor) ke INFINITE

else f(successor) = MAX(f(current), f(successor))

Langkah 2.6 If successor adalah yang terakhir then update biaya f ancestors menjadi minimum dari biaya f successornya

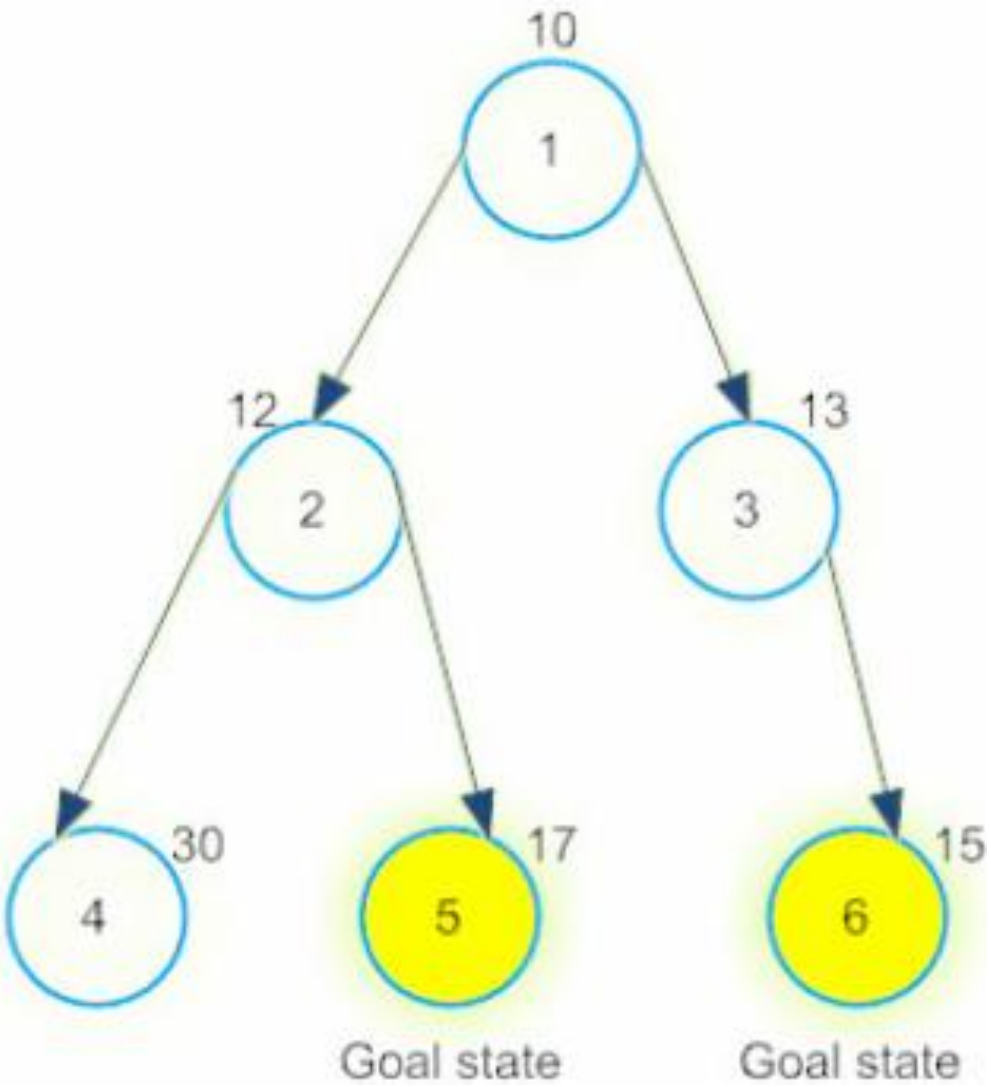
Langkah 2.7 If tidak ada lagi memory bagi successor then hapus node nilai f paling tinggi paling dangkal dan ingatlah biaya f terlupakan paling baik

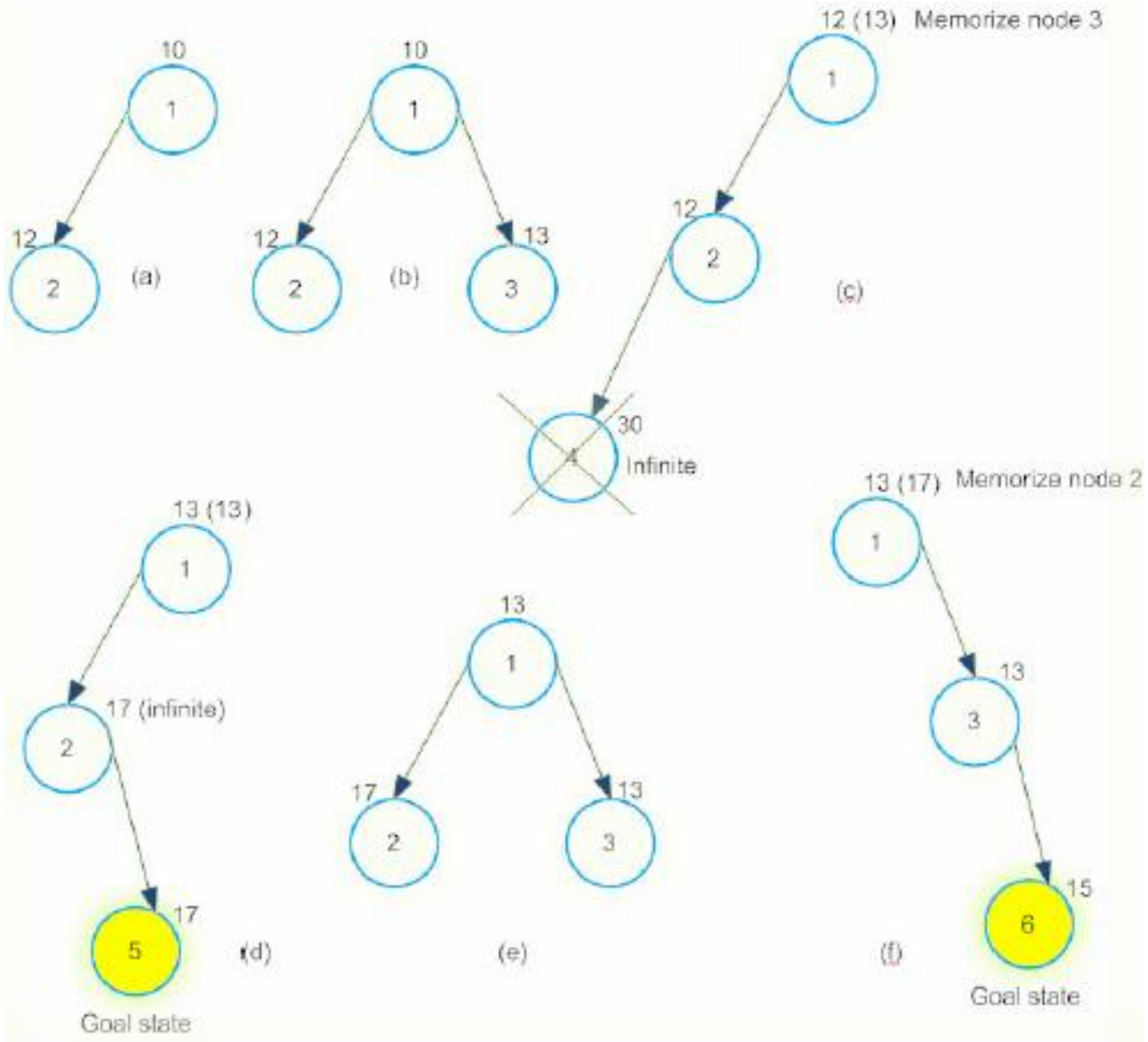
Langkah 2.8 Sisipkan anak ke dalam antrian Q

end

Contoh: Jarak Terpendek

- Pohon dengan 6 node. Node 1 status awal, node 5 dan 6 tujuan.
- Angka di atas node adalah nilai dari fungsi f untuk node tersebut.
- **Cari jalur terpendek menggunakan SMA* dengan memory maksimum 3 node**.




10
1
12
2
(a)
10
1
12
2
13
3
(b)
12 (13) Memorize node 3
1
12
2
(c)
30
Infinite
13 (13)
1
17 (infinite)
2
17
5
(d)
Goal state
13
1
17
2
13
3
(e)
13 (17) Memorize node 2
1
13
3
15
6
(f)
Goal state

Langkah-langkah

- (a) Buka node 1 , diperoleh node 2 dengan f = 12. Simpan ke memory.
- (b) Memory masih dapat menerima satu node. Buka node 1 dan dapatkan node 3 dan masukkan ke memory. Memory penuh dan tidak ditemukan node tujuan.
- Proses selanjutnya:
  - update f dari node 1 dengan f minimum dari anaknya (12/node 2)
  - Buka node 2
  - Keluarkan node daun dengan f paling tertinggi (node 3)
  - Ingat f dari node 3 (13).
- (c) Buka node 4, masukkan ke memory. Bukan tujuan. Selanjutnya:
  - Ingatkan node 3 ke dalam node 1
  - Memory penuh
  - Node 4 bukanlah node tujuan, tandai *infinite.*

Langkah-langkah

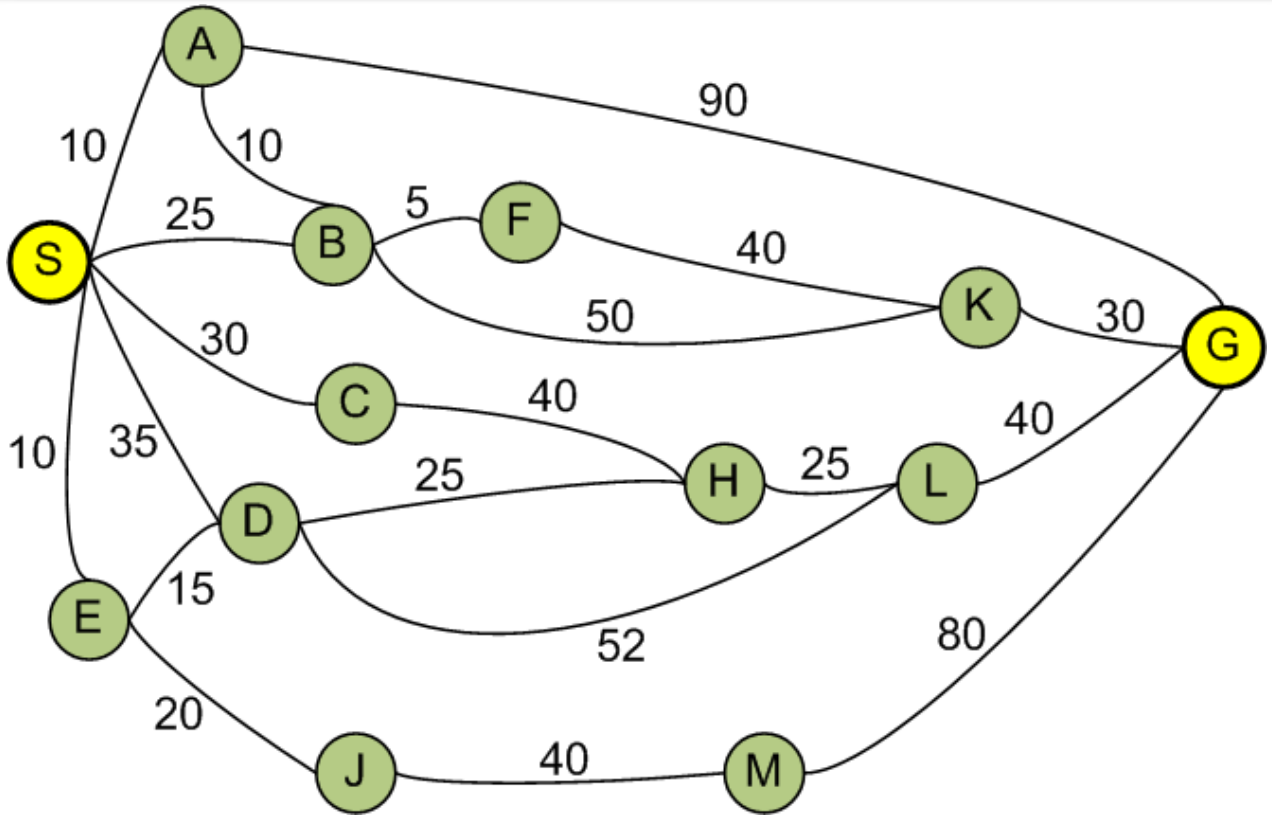
- (d) Lakukan :
  - Keluarkan node 4 dan tambahkan node 5
  - node 2 mengingat node 5
  - update f dari node 2 dengan f minimum dari anak-anaknya
  - update f dari node 1
- Node 5 adalah tujuan. Tetapi masih ada nilai 13 dalam memory, lebih kecil daripada f node tujuan yang diperoleh. Selanjutnya buka node yang diingat.
- (e) Node 3 dibuka lagi, kemudian:
  - Lepaskan node 2 dan tambahkan node 6
  - Ingatkan node 2 ke dalam node 1
  - Node 6 adalah tujuan dan node dengan biaya f terkecil.
  - SELESAI (gambar f).

Algoritma D*

- A* yang dinamis. Parameter biaya dapat berubah selama proses penyelesaian masalah.
- Fungsi heuristik:

  f(n) = g(n) + w(n) * h(n)
- W(n) adalah bobot yang pada langkah pertama ditentukan besar dan semakin kecil pada langkah berikutnya. Saat dianggap mendekati tujuan, nilainya mendekati 1, misalnya 1.1.

Jalur Terpendek dengan SMA* dan D*



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

Pencarian Iteratif

- **Sebelum ini**: mencari jalur menuju solusi.
- **Sekarang**: mungkin hanya tertarik untuk menemukan status tujuan. Digunakan *local search*.
- Analogi *local search* sebagai variasi penyelesaian masalah:
  - *Status awal (Start)*: is a complete configuration in the case of local search compared with a single node on a path (which is a solution);
  - *Operator*: Changes applied to the current state to (heuristically) improve quality
  - *Fungsi evalusi*: Instead of a goal state or goal test, an evaluation function is used. The problem to solve may not have sometimes an exact goal (for instance we are looking for the minimum or maximum value (which is unknown) of a function to optimize);
  - Pencarian: Consists on improving the quality of current state until some number of iterations of algorithm has been reached or some other termination condition has fulfilled. Typically does not keep track of repeated states.

Algoritma Pencarian Iteratif

Langkah 1. Status_Kini = Status_Awal

Langkah 2. **while Status_Kini tidak memenuhi uji tujuan**

DAN batas waktu tak terlewati

Langkah 2.1 Bangkitkan anak (Successor) dari Status_Kini

Langkah 2.2 Set Successor paling menjanjikan (nilai heuristiknya paling rendah) sebagai Status_Kiri baru

Langkah 3. **if** Status_Kini **memenuhi uji tujuan**

then return jalur tindakan yang menyebabkan Status_Kini

else return GAGAL

end

Contoh Pencarian Iteratif

- *Hill Climbing* (pendakian bukit), dapat berupa pencarian mendaki dan menurun
- Tabu
- *Simulated Annealing*
- *Local Beam*

Hill Climbing

- Dikenal juga sebagai *gradient ascent* atau *gradient descent*
- Terus bekerja selama Status_Kini lebih baik dari status sebelumnya. Jika tidak, berhenti.
- Analogi: mendaki gunung tanpa peta
- Dapat berhadapan dengan masalah puncak salah: tidak mencapai tujuan karena sudah menjumpai jalan turun.

Algoritma Hill Climbing

Langkah 1. Set Status_Kini ke status awal

Langkah 2**. loop**

Langkah 2.1 Bangkitkan successor dari Status_Kini;

Langkah 2.2 Ambil successor dengan nilai tertinggi;

Langkah 2.3 **if nilai(successor) < nilai(Status_Kini)**

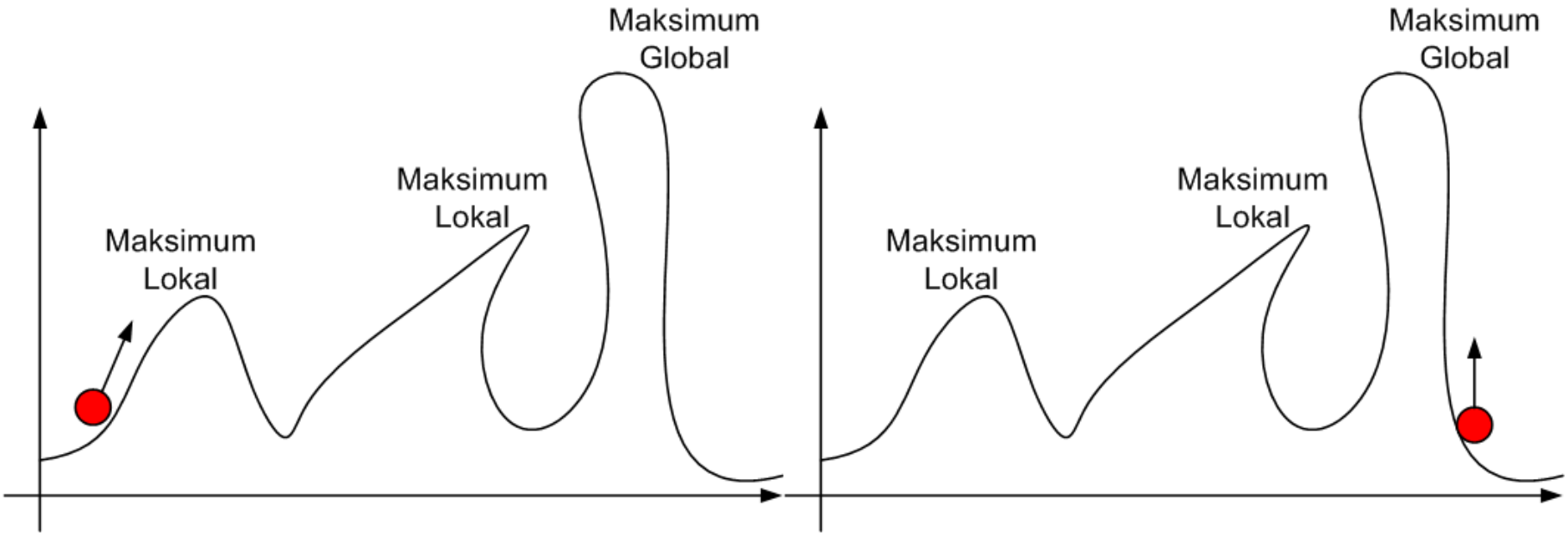
then Return Status_Kini

else Status_Kini = successor

end

Maksimum Lokal atau Global?

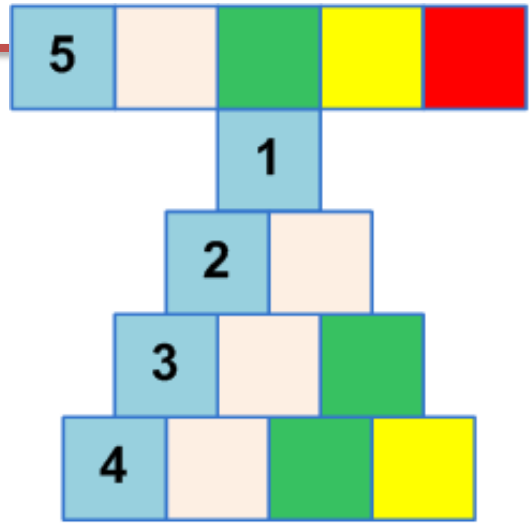
- Sangat tergantung pada status awal, dapat terjebak pada optimum/maksimum lokal
- Algoritma berhenti saat diperoleh puncak (walaupun lokal) karena *successor* sudah terlihat menuruni bukit (gambar kiri). Gambar kanan, puncak global dicapai.



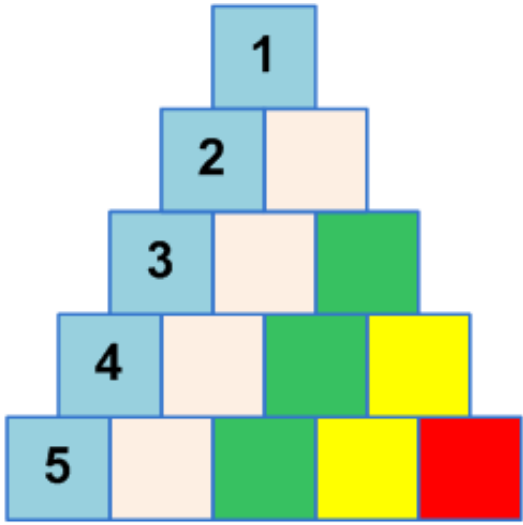
Situasi Berbahaya

- Dataran tinggi (*plateau*): Status-status successor bernilai sama, tidak ada cara memilih;
- Bukit di kaki gunung (*foothill*): maksimum lokal, dapat terjebak pada puncak yang kecil;
- Punggung bukit (*ridge*): bukit dikaki gunung dimana N-langkah di depan adalah (menuju) solusi.

Contoh: Menyusun Balok



Status Awal

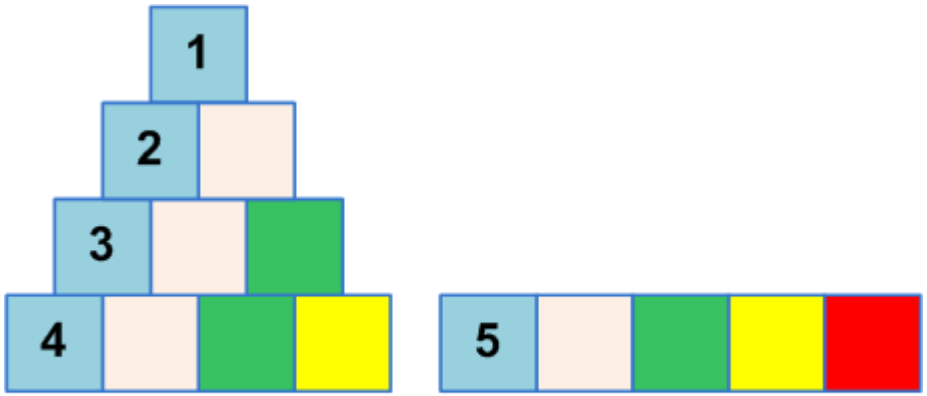


Status Tujuan

- Tentukan langkah-langkah untuk menyusun balok-balok dari status awal hingga dicapai status tujuan
- Hanya satu balok (paling atas) dapat dipindahkan pada satu waktu dan
- Hanya dua tumpukan tambahan dapat digunakan untuk menyusun balok-balok.

Menyusun Balok: Heuristik Pertama

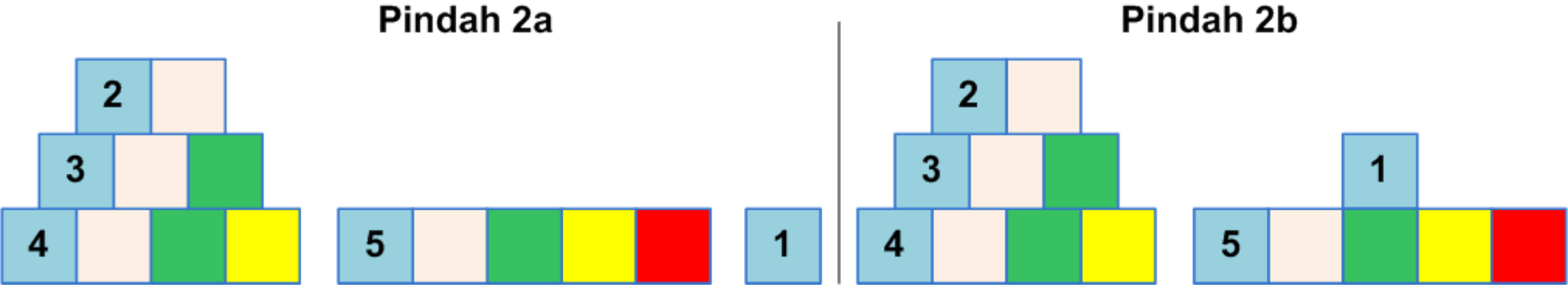
- Menghitung +1 untuk setiap balok yang berada di balok yang benar. Status tujuan memiliki nilai +5;
- Menghitung -1 untuk setiap balok yang berada di atas balok yang salah.
- Status awal (Status_Kini): Balok 1, 2 dan 3 posisinya benar, balok 4 dan 5 salah. Nilainya? +3-2=1.
- Hanya satu balok teratas dapat dipindahkan (pindah 1).



Status_Kini

Contoh: Menyusun Balok

- Berapa nilainya?
- Balok 1, 2, 3 dan 5 berada pada posisi yang benar. Hanya balok 4 yang salah. +4-1 = 3
- Lebih baik dari pada Status awal, dijadikan Status_Kini
- Ada 2 kemungkinan pindah (pindah 2a dan 2b)



Contoh: Menyusun Balok

- Nilai dari *successor* pertama (Pindah 2a) adalah +3 -2 = 1 (2, 3 dan 5 diatas balok yang benar sedangkan 1 dan 4 tidak)
- Nilai dari *successor* kedua (Pindah 2b) is again +3-2=1, sama dengan successor pertama.
- Nilai kedua successor lebih rendah daripada induknya. Kesimpulan: Pindah 1 Optimum.
- Hanya diperoleh optimum lokal. Padahal solusi belum ditemukan.

Menyusun Balok: Heuristik Kedua

- Menghitung +n untuk setiap balok yang berada di atas tumpukan benar dari n balok. Status tujuan bernilai +10.
- Menghitung -n untuk setiap blok yang berada di atas tumpukan yang salah dari n balok.
- Status awal: nilainya (-1)+(-2)+(-3)+(-4) = -10, karena balok 3 di atas satu balok yang salah, balok 2 di atas 2 balok salah, balok 1 di atas 3 balok yang salah dan balok 5 di atas 4 balok salah.
- *Successor* yang diperoleh dengan **pindah 1** memberikan -6 (sama dengan status awal tetapi balok 5 pada posisi yang benar, jadi hanya (-1)+(-2)+(-3)).
- **Pindah 2a** bernilai (-1)+(-2) =-3 karena balok 3 di atas satu balok salah dan balok 2 di atas 2 balok salah. **Pindah 2b** bernilai (-1)+(-2)+(-1) = -4 dari pinalti balok 3, 2 dan 1.
- Hasilnya, optimum lokal terhindari.

Contoh: 3 Gadis & 3 Kanibal

- Contoh heuristik:

  Evaluasi jumlah orang pada sisi kiri: status awal bernilai 6 dan status tujuan 0.
- Langkah pertama, ada 3 kemungkinan (sesuai aturan):
  - 2 kanibal menyeberang
  - 1 kanibal dan 1 gadis menyeberang
  - 1 kanibal menyeberang
- 2 gerakan pertama menyebabkan sisi kiri mendekati tujuan. 4 orang di kiri, 2 di kanan.
- Dilihat kemungkinan langkah ke-2, jumlah di sisi kiri bertambah menjadi 5 atau 6. Menuruni bukit. Selesai.
- Dicapai optimum lokal dan hill climbing berhenti. Solusi tidak diperoleh.

Perbaiki Heuristik

- Pertimbangkan status tujuan setiap sisi sungai. Bandingkan successor dari suatu status dengan status tujuan di sisi dimana perahu berada.
- Contoh:
  - Status awal (Status_Kini):

    Sisi kiri: (3, 3, 1) , sisi kanan: (0, 0, 0)
  - Status tujuan:

    Sisi kiri:(0, 0, 0) , sisi kanan:(3, 3, 1)
- Mulai dari sisi kiri, ada 3 successor. Bandingkan dengan tujuan, berapa yang berada di sisi kanan? 2 successor pertama lebih disukai karena 2 orang pindah ke sisi kanan, pilihan ketiga hanya memindahkan satu orang.

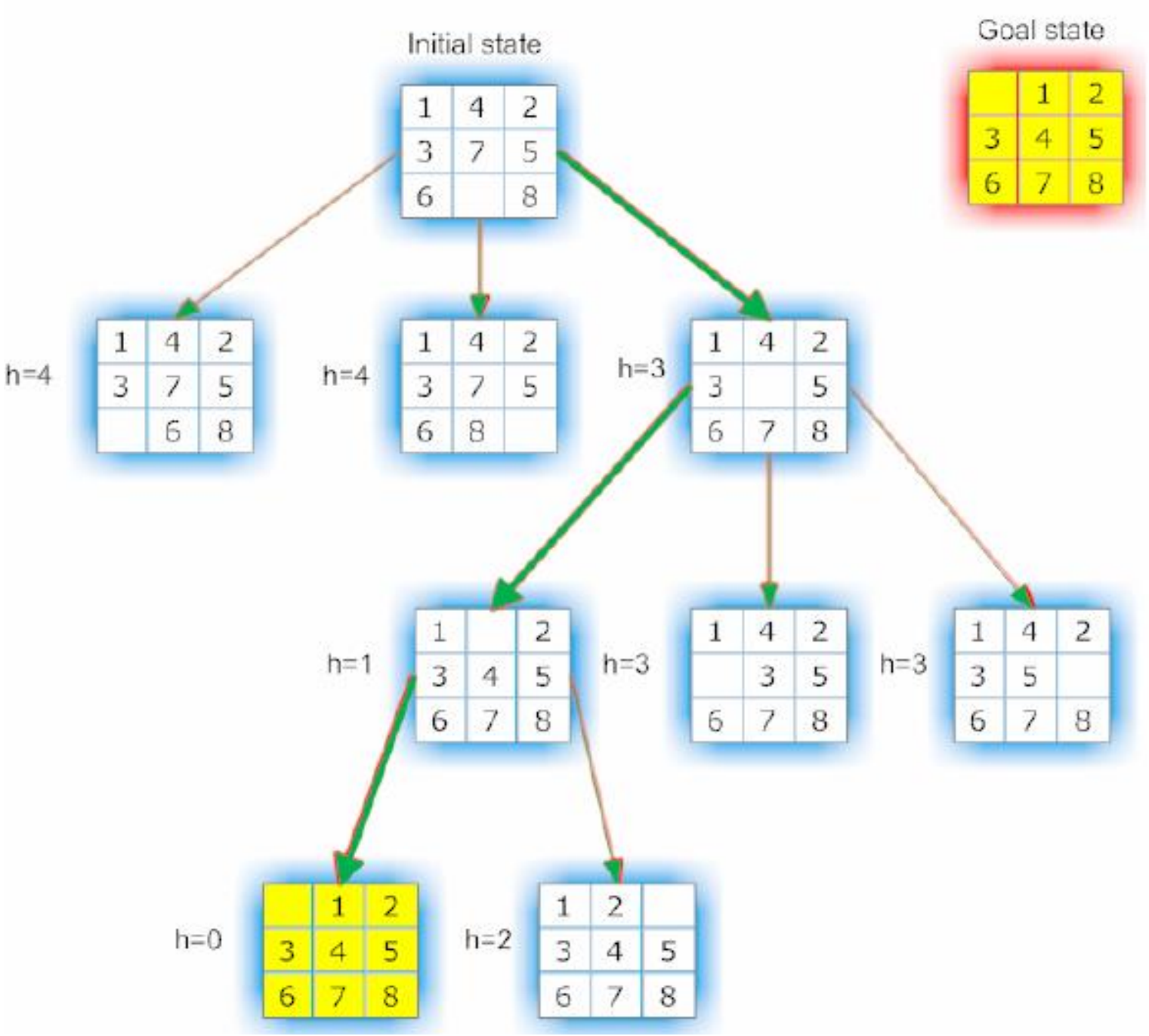
Langkah Pertama & Kedua

- (1) Sisi kiri: (2, 2, 0) Sisi kanan: (1, 1, 1) atau
- (2) Sisi kiri: (3, 1, 0) Sisi kanan: (0, 2, 1)
- Misal, ambil successor (2). 1 atau 2 orang harus mengembalikan perahu ke sisi kiri.
- 2 kemungkinan situasi, langkah ke-2:
- Sisi kiri: (3, 2, 1), Sisi kanan:(0, 1, 0) atau
- Sisi kiri: (3, 3, 1) , Sisi kanan: (0, 0, 0)
- Bandingkan dengan tujuan. Pilihan pertama lebih baik daripada sebelumnya (saat perahu juga di kiri).
- Pencarian berlanjut mengikuti pilihan pertama tersebut.

Langkah Selanjutnya

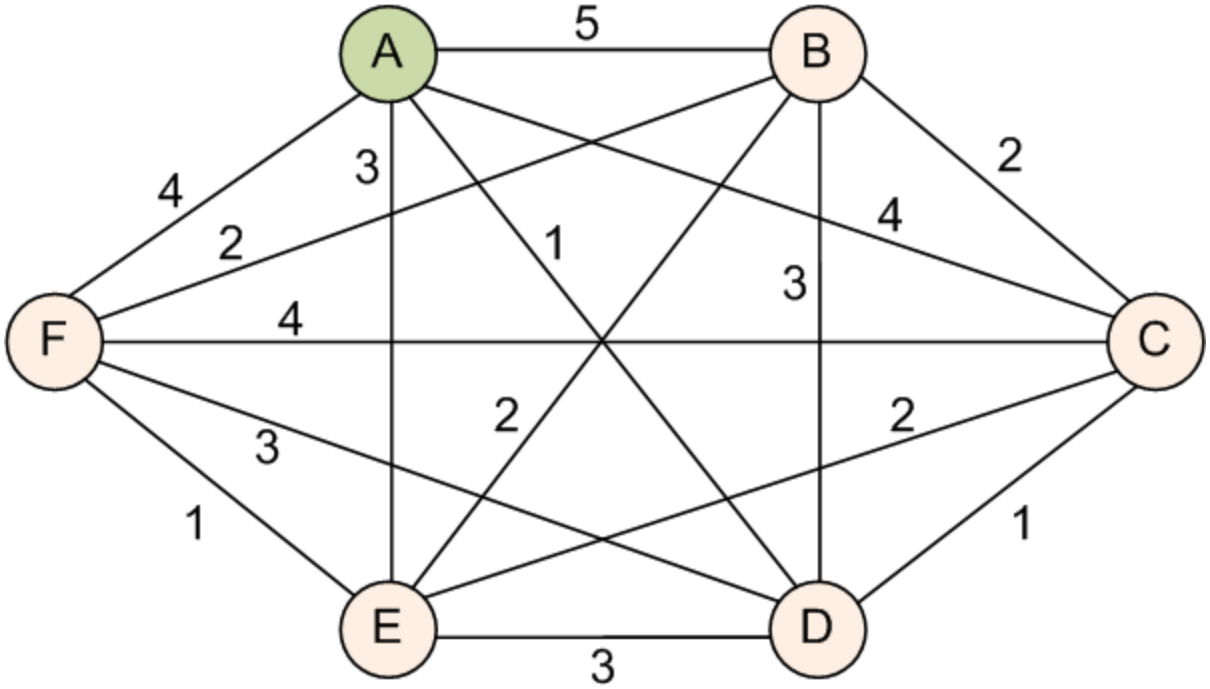
- Langkah 3: Kiri kiri: (3, 0, 0) Kiri kanan: (0, 3, 1)
- Langkah 4: Kiri kiri: (3, 1, 1) Kiri kanan:(0, 2, 0)
- Langkah 5: Kiri kiri: (1, 1, 0) Kiri kanan:(2, 2, 1)
- Langkah 6: Kiri kiri: (2, 2, 1) Kiri kanan:(1, 1, 0)
- Perhitungan pada langkah 6, nilai status tidak lebih baik dari sebelumnya. Hill climbing berhenti pada langkah 5.
- Langkah 5 dianggap sebagai solusi, karena paling dekat dengan status tujuan.
- Hanya dicapai maksimum lokal walaupun lebih dekat ke status tujuan.

Contoh: 8-Puzzle, heuristik: jumlah ubin salah tempat



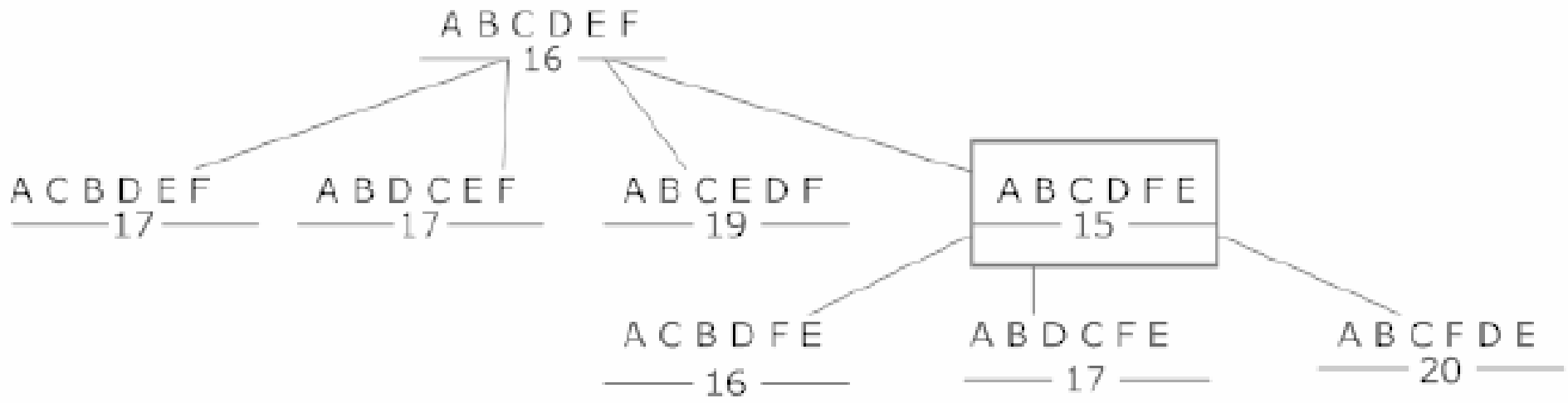
Contoh: *Traveling Salesman* (TSP)

- Jalur mana yang harus ditempuh oleh Salesman untuk menjual dagangannya agar biaya yang dikeluarkan minimal? Status awal: A.



Heuristik 1

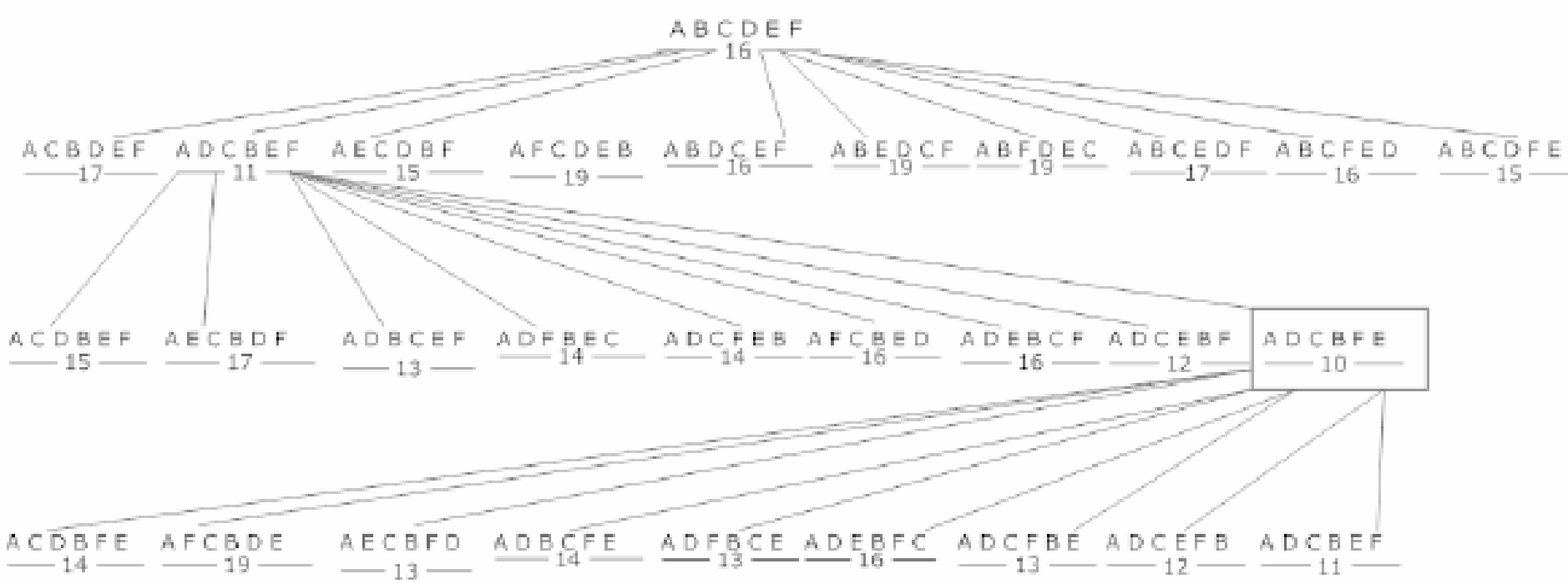
- Hanya boleh 2 kota yang berturutan (berdampingan) dengan biaya ≤ 15
- Kemungkinan solusi:



- Terjebak pada solusi lokal.

Heuristik 2

- Pertukaran kota bebas dengan biaya ≤ 10
- Solusi yang diperoleh:



- Merupakan solusi global.

Tugas

- Bagaimana proses penyelesaian masalah jalur terpendek S ke G menggunakan SMA* dengan memory hanya dapat menampung maksimal 5 node?

  Boleh dikumpulkan via email Lunix96@gmail.com.
- Pelajari tentan **Simulated Annealing** dan **Pencarian Tabu**. Jelaskan disertai contoh. Dikumpulkan *hard copy* Senin depan (22 Oktober 2012).