Kecerdasan Buatan

Pertemuan 3: Informed Search

Gambaran Umum Materi

* Penggunaan fungsi heuristik
* Best-first Search
* Iterative-improvement Search

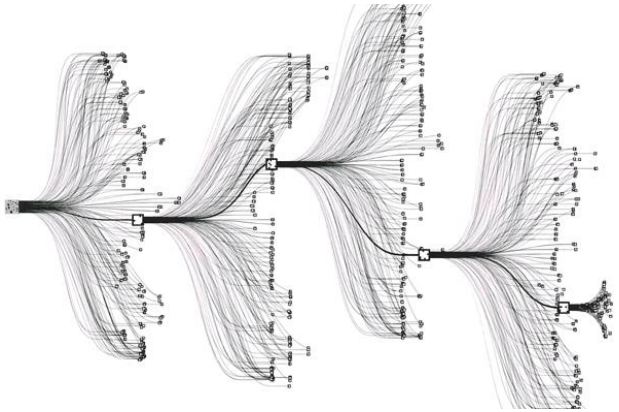
Kompleksitas waktu dari Breadth-first Search

* Jumlah maksimum anak pada tiap node (b): 10
* Asumsi: 1 node membutuhkan waktu 1 ms untuk diekspansi dan 100 bytes untuk menyimpannya (dalam memori komputer)

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

Pencarian Buta/Pencarian Menyeluruh (*Exhaustive Search*)

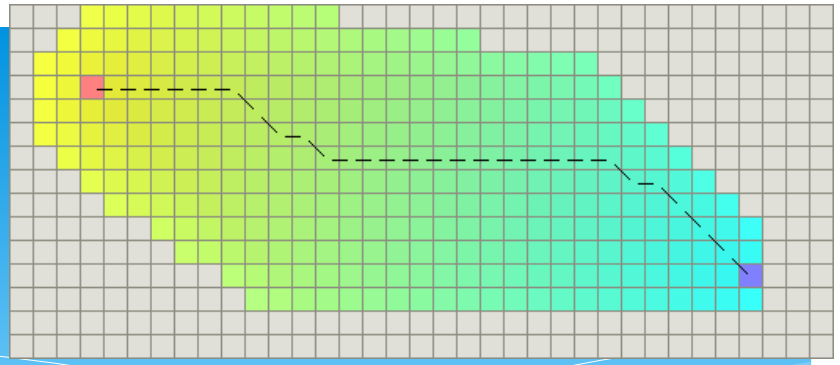
* Mencari solusi dengan menghasilkan state baru dan mengetesnya terhadap state tujuan (*goal state*)
* State berupa state awal, state tujuan, atau lainnya
* Untuk sebagian besar kasus, ini tidak efisien karena jumlah state baru yang dihasilkan dapat sangat banyak dan mengakibatkan ukuran Ruang Keadaan *(state space)* menjadi sangat besar
  * Ukuran Ruang Keadaan permainan catur: ~10^{50}
  * Ukuran Ruang Keadaan permainan Go: ~10^{170}
* Alternatif:
  * Algoritma Informed (heuristic) search
  * Algoritma Local search



Pencarian Tidak Menyeluruh (Non-exhaustive Search)

* Tidak mengeksplorasi node yang "kurang menjanjikan"
* Pencarian yang spesifik pada suatu domain
* Strategi *informed search*:
  * Node *n* dipilih untuk diekspansi berdasarkan penilaian dari sebuah fungsi evaluasi *f(n)*
  * Pengetahuan akan domain dituliskan dalam fungsi heuristik *h(n)*
  * *h(n)* mengestimasi biaya dari node *n* ke node goal terdekat

Fungsi Heuristik



Sumber: http://theory.stanford.edu/~amitp/game-programming/a-star/euclidean.png

* Merupakan estimasi kelayakan
* Digunakan untuk memberikan informasi terhadap proses pencarian
* Harus dapat diterima secara logika, yakni tidak meng-overestimasi biaya untuk mencapai tujuan
* Perhitungan heuristik yang paling sederhana pun akan lebih baik daripada tidak ada sama sekali
* Heuristik tidak boleh sempurna
* Nilai ‘terbaik’ untuk sebuah heuristik adalah biaya untuk mencapai tujuan, tetapi untuk dapat menemukan ini, diperlukan pencarian menyeluruh *(exhaustive search)*

Pencarian Heuristik

* Waktu yang diperlukan untuk mengevaluasi fungsi heuristik untuk memilih node mana yang akan diekspansi harus setidaknya sebanding dengan banyaknya pengurangan pada ukuran ruang keadaan yang dieksplorasi
* Ini adalah syarat minimum sebuah pencarian heuristik

Fungsi Heuristik

* Fungsi Evaluasi:
  * $f(n)$ = kombinasi antara $g(n)$ dengan $h(n)$

* $g(n)$ = biaya yang sudah dikeluarkan untuk mencapai n (sudah diketahui)
* $h(n)$ = estimasi biaya dari n menuju goal : fungsi heuristik
* $f(n)$ = total estimasi biaya yang dikeluarkan jika melewati n untuk menuju goal

Best-first Search

* Mengekspansi node pada list L yang dianggap ‘terbaik’
* Keefektifannya bergantung kepada jenis fungsi heuristik yang diterapkan
* Dapat diimplementasikan dengan menggunakan queue
  * Buat sebuah daftar node yang terurut menurut nilai *f*
* Ada dua jenis algoritma Best-first:
  * **Pencarian Greedy:** mengekspansi node yang paling dekat dengan goal; pilih node *n* dimana $f(n) = h(n)$ yang paling minimal
  * **A*:** mengekspansi node yang memiliki path solusi paling minimum; pilih node *n* dimana $f(n) = h(n) + g(n)$ yang paling minimal

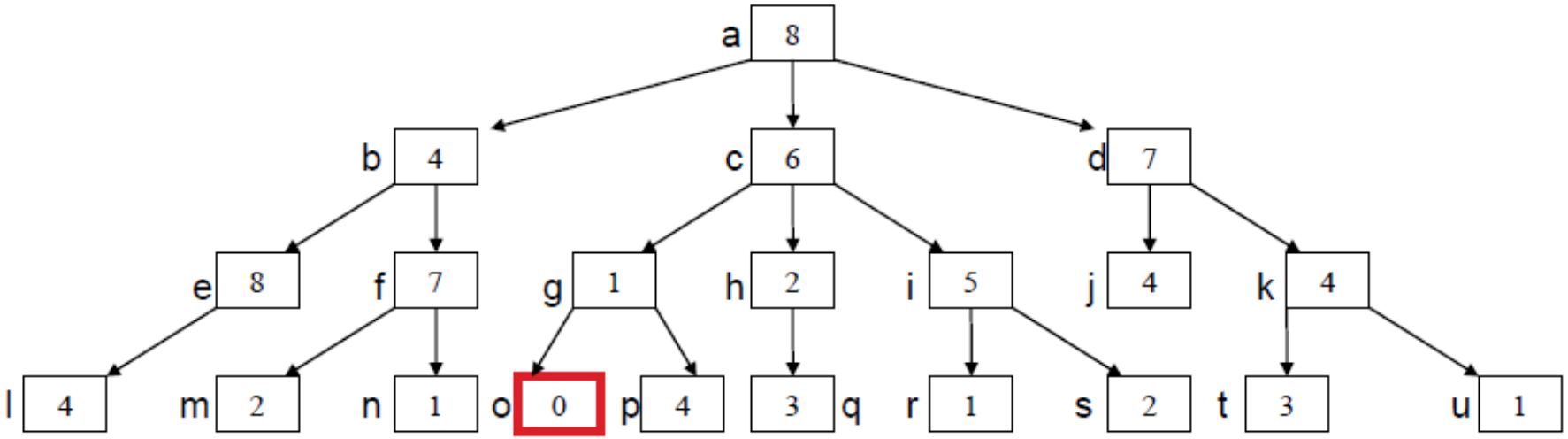
Pencarian Greedy

* Greedy berarti serakah/rakus
* Pencarian Greedy mengekspansi node yang paling dekat dengan goal
* Setiap node dinilai dengan sebuah fungsi evaluasi, dan node dengan nilai tertinggi akan diekspansi pada iterasi berikutnya → $f(n)=h(n)$
* Implementasi: Gunakan queue yang diurutkan berdasarkan nilai fungsi evaluasi (secara menaik/*ascending*)

Pencarian Greedy

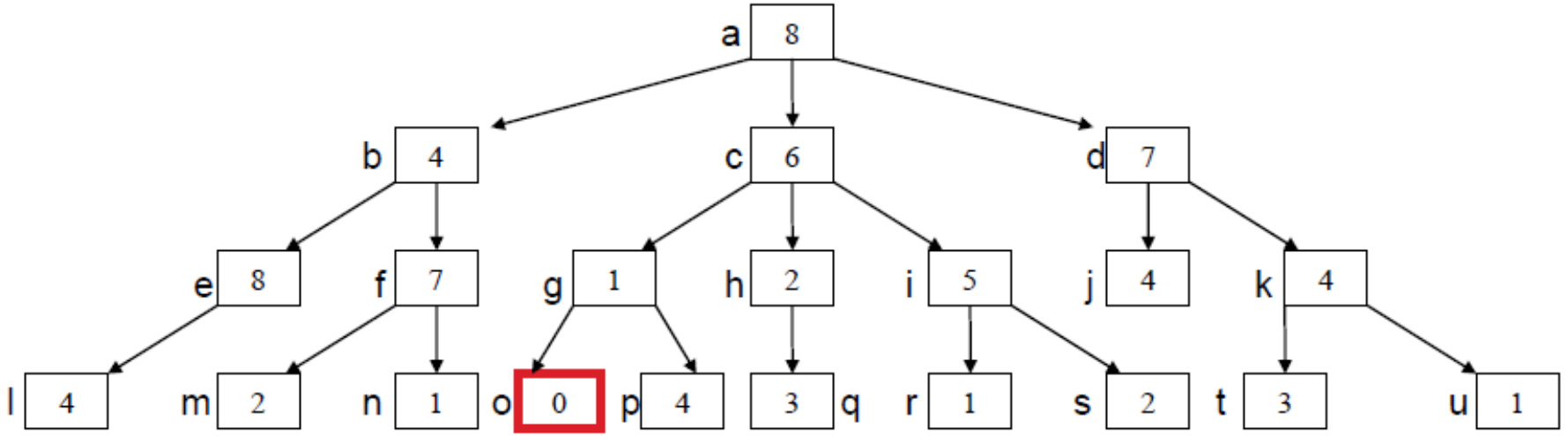
1. Tetapkan *L* sebagai daftar node awal
2. Misal *n* adalah node pertama pada *L* yang dianggap terdekat dengan goal. Jika *L* kosong, maka pencarian gagal.
3. Jika *n* adalah node goal, berhenti, dan kembalikan node tersebut beserta jalur/path yang dilalui dari node awal hingga node *n*
4. Jika *n* bukan node goal, hapus *n* dari *L* dan tambahkan semua anak-anak dari *n* ke dalam *L*. Beri label jalur/path dari node awal menuju semua node anak. Kembali ke langkah 2

Latihan 1



Dengan menggunakan pencarian Greedy dan diberikan nilai heuristik dari tiap node, tentukan urutan pencarian node goal yang dilakukan pada gambar di atas.

Jawaban 1



h(a)=8

h(b)=4 h(c)=6 h(d)=7

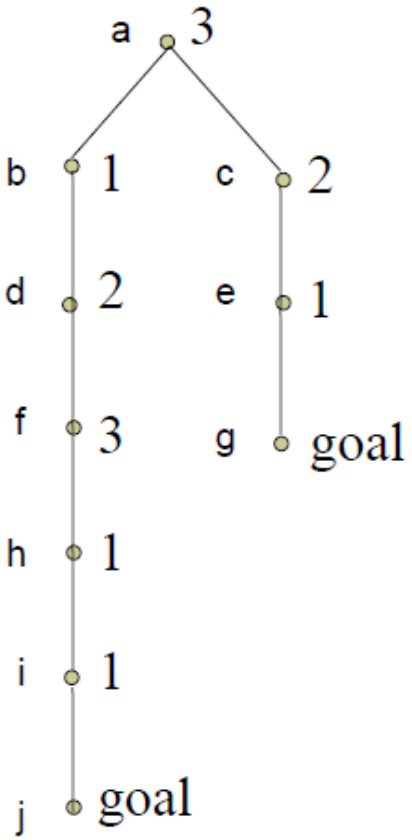
h(c)=6 h(d)=7 h(f)=7 h(e)=8

h(g)=1 h(h)=2 h(i)=5 h(d)=7 h(f)=7 h(e)=8

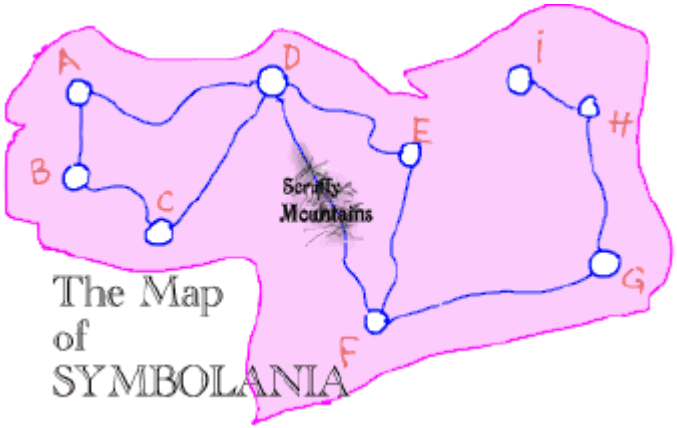
h(o)=0 h(h)=2 h(p)=4 h(i)=5 h(d)=7 h(f)=7 h(e)=8

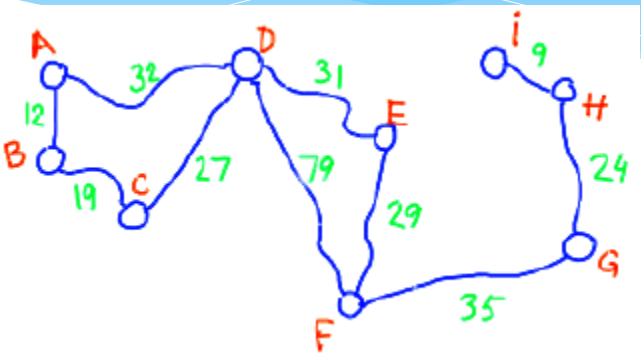
Latihan 2

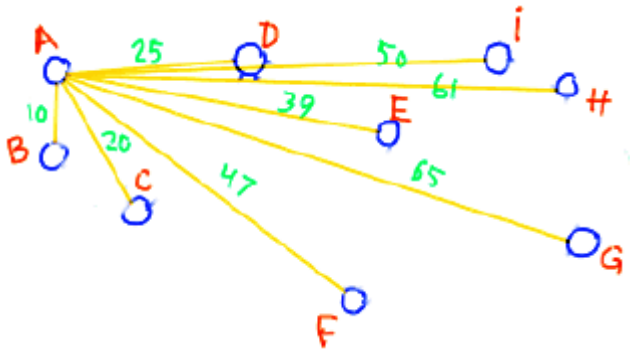
* Dengan pencarian Greedy, tuliskan urutan ekspansi node dalam ruang keadaan berikut:



Studi kasus: Perjalanan di Symbolania

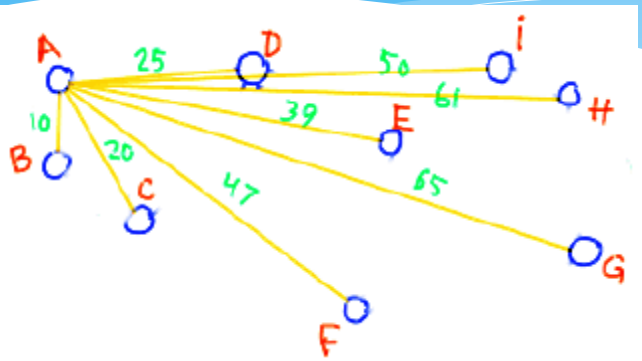


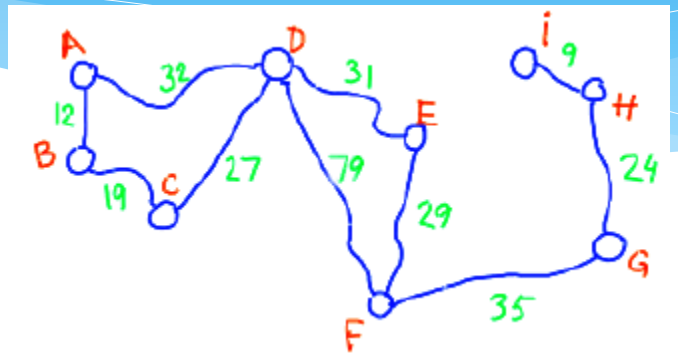




- Negara Symbolania memiliki 9 kota (A-I).
- Tiap kota terhubung dengan jalan. Jarak dituliskan dengan warna hijau. Terdapat pegunungan Scruffy yang menyebabkan jalan dari F ke D lebih panjang dari seharusnya.
- Kita harus menuju kota A. Jarak dari tiap kota menuju A dituliskan dalam warna hijau. Ini adalah informasi yang dibutuhkan dalam mencapai goal dari setiap state yang ada (diasumsikan koordinat kota A dan kota2 lainnya sudah diketahui)
- Sehingga, fungsi heuristiknya, h(), mengembalikan nilai *Euclidean distance* dari tiap kota terhadap kota A. Contoh: h(F)=47, h(H)=61, dst.

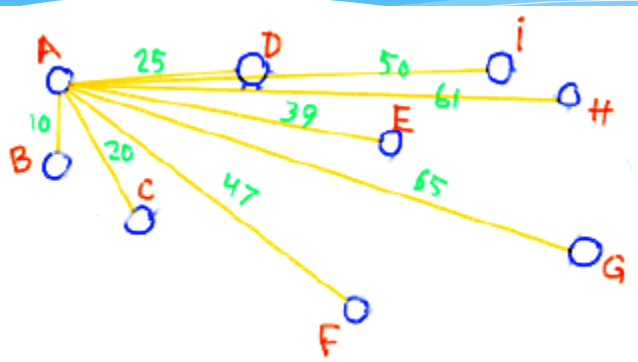
Pencarian Greedy di Symbolania(1)

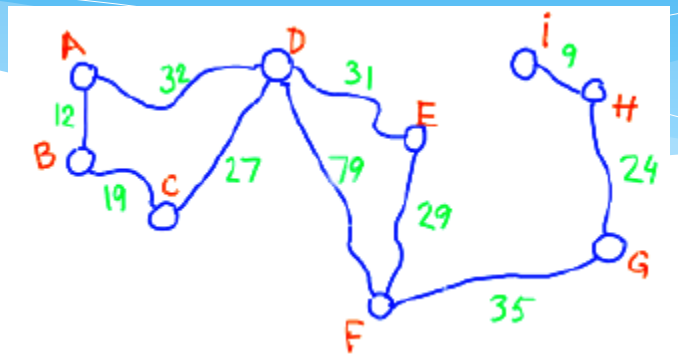




* Misal start dari E menuju A
* h(E)=39
* h(D)=25, h(F)=47
* h(A)=0, h(C)=20, h(E)=39, h(F)=47
* Jalur: E – D – A

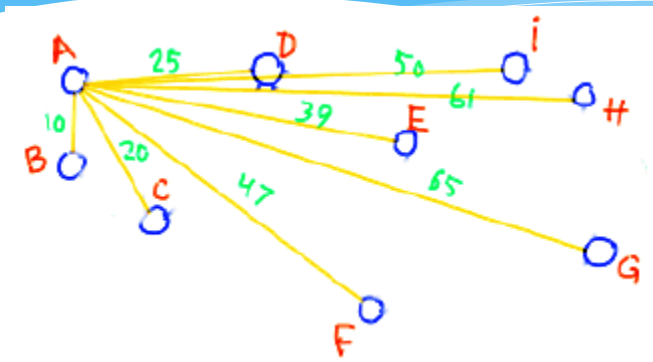
Pencarian Greedy di Symbolania(2)

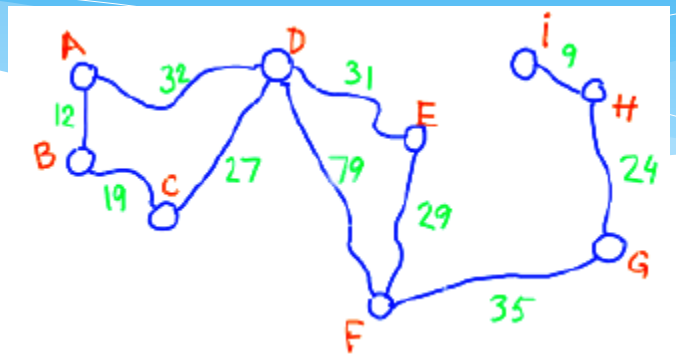




* Misal start dari F menuju A
* h(F)=47
* h(D)=25, h(E)=39, h(G)=65
* h(A)=0, h(C)=20, h(E)=39, h(E)=39, h(F)=47, h(G)=65
* Jalur: F – D – A (jalur yang optimal seharusnya F-E-D-A)
* Pencarian Greedy tidak selalu memberikan hasil yang optimal!

Pencarian Greedy di Symbolania(3)





* Misal start dari H menuju A
* h(H)=61
* h(I)=50, h(G)=65
* h(H)=61, h(G)=65
* h(I)=50, h(G)=65
* ……

Pencarian Greedy tidak selalu menemukan solusi!

Sifat dari Pencarian Greedy

* **Lengkap/Selesai**
  * Tidak, karena dapat terjebak dalam loop
  * Ya, pada ruang keadaan yang dibatasi dan disertai dengan pengecekan state berulang
* **Waktu**
  * $O(b^m)$, tapi heuristik yang baik dapat memberikan peningkatan yang signifikan
* **Ruang**
  * $O(b^m)$, karena menyimpan semua node di memori
* **Optimal**
  * Tidak

Pencarian Greedy tidak optimal, tetapi sering efisien

Pencarian Goal yang Disempurnakan

* Untuk mencari goal ‘terdangkal/terdekat’ secepat mungkin
* Mengekspansi node yang kelihatannya paling dekat dengan goal terdangkal
* Algoritma A*

Pencarian A*

* Menghindari untuk mengekspansi jalur/path yang sudah diketahui mahal
* Fungsi evaluasi $f(n) = g(n) + h(n)$
  * $g(n)$ = biaya yang sudah dikeluarkan untuk mencapai n (sudah diketahui)
  * $h(n)$ = estimasi biaya dari n menuju goal : fungsi heuristik
  * $f(n)$ = total estimasi biaya yang dikeluarkan jika melewati n untuk menuju goal

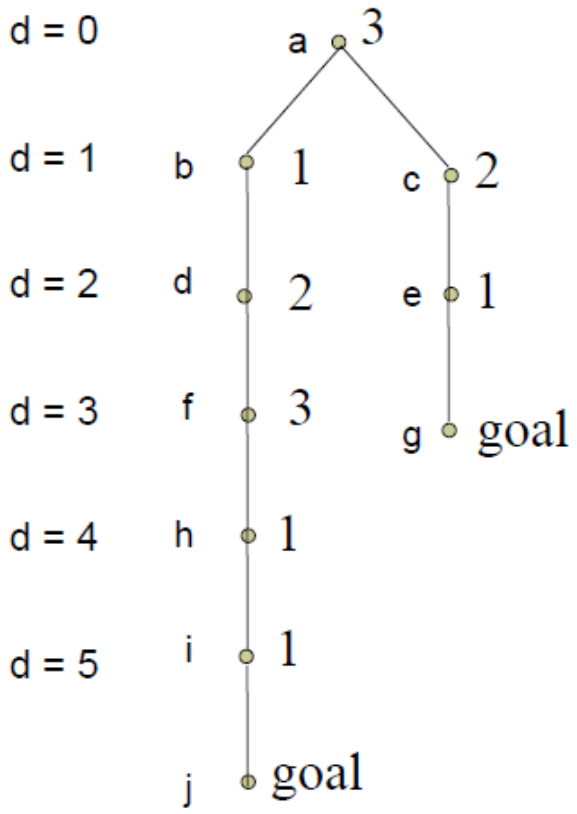
Algoritma A*

1. Tetapkan *L* sebagai daftar node awal
2. Misal *n* adalah node pertama pada *L* dimana *f(n)* yang minimal. Jika *L* kosong, maka pencarian gagal.
3. Jika *n* adalah node goal, berhenti, dan kembalikan node tersebut beserta jalur/path yang dilalui dari node awal hingga node *n*
4. Jika *n* bukan node goal, hapus *n* dari *L* dan tambahkan semua anak-anak dari *n* ke dalam *L*. Beri label jalur/path dari node awal menuju semua node anak. Kembali ke langkah 2

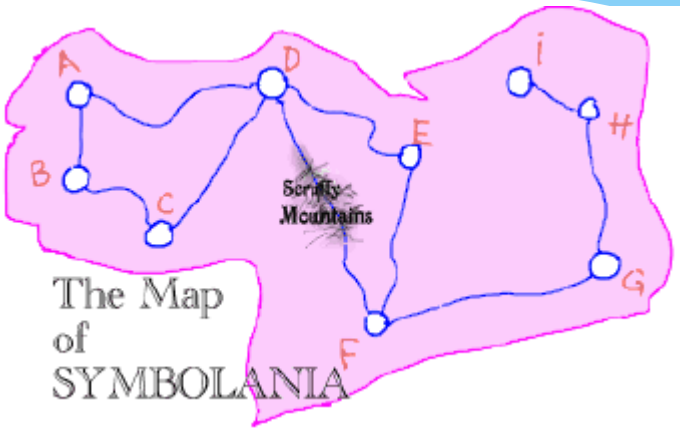
Latihan

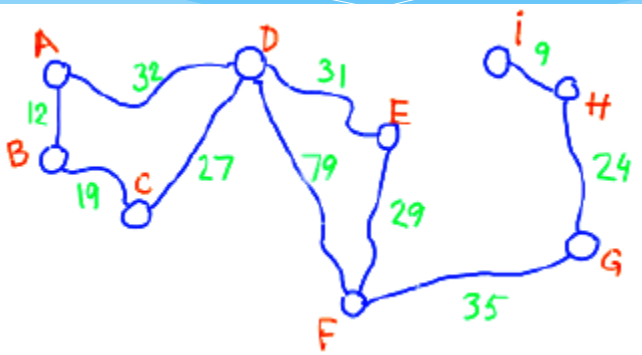
* Diberikan estimasi awal h(n), gunakan algoritma A* untuk menghitung biaya tiap node dan urutan ekspansi node

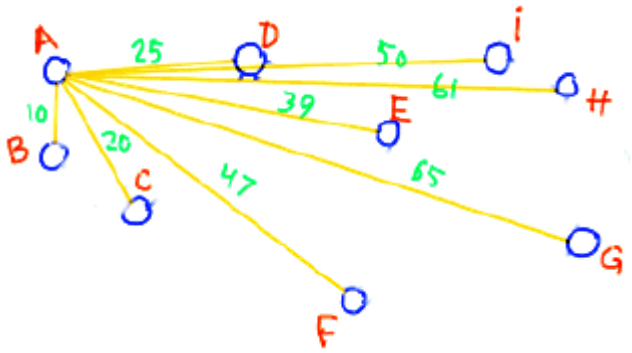
Biaya sampai saat ini, g(n)=kedalaman node



Perjalanan di Symbolania





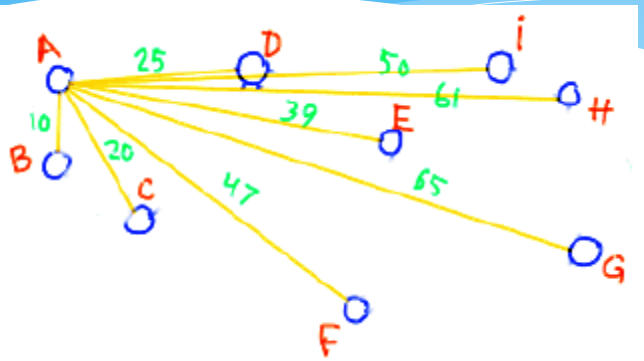


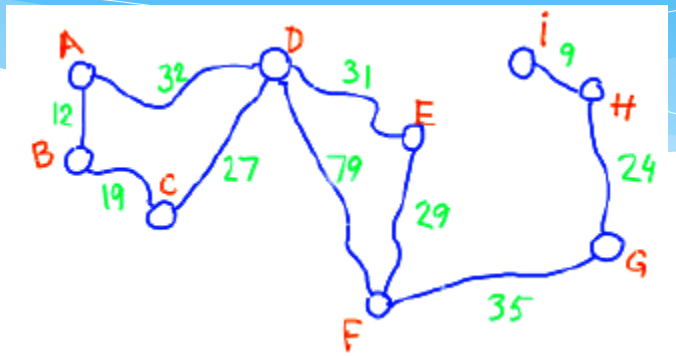
Bagaimana penelusuran:
Dari E ke A?

Dari F ke A?

Dari H ke A?

Pencarian A* di Symbolania (1)

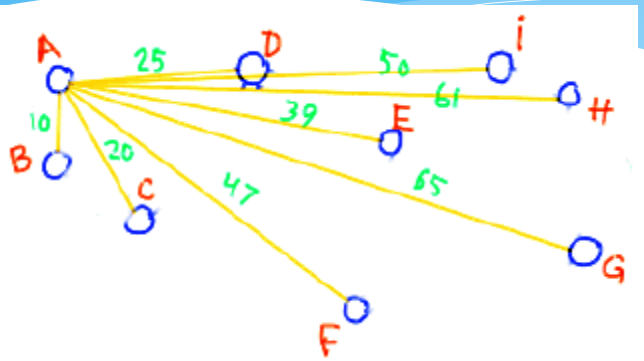


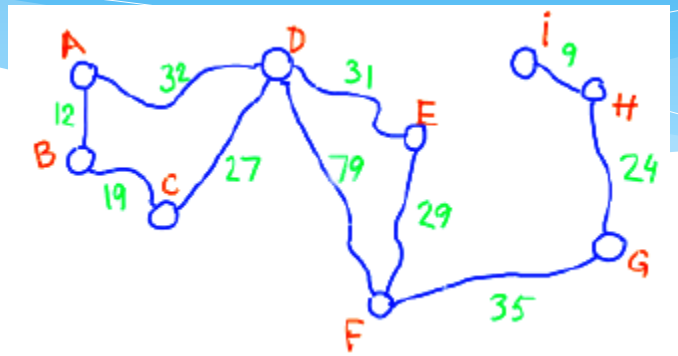


* Dari E ke A
* f(E)=(0+39)=39
* f(D)=31+25=56, f(F)=(29+47)=76
* f(A)=(31+32+0)=63, f(C)=(31+27+20)=78, f(E)=(31+31+39)=101, f(F)=76, f(F)=(31+79+47)=157

* Jalur: E – D – A

Pencarian A* di Symbolania (2)

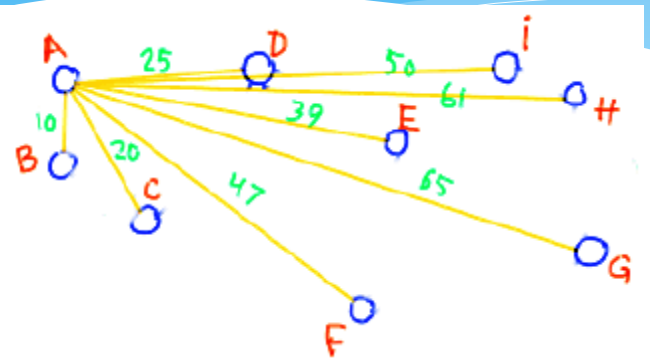


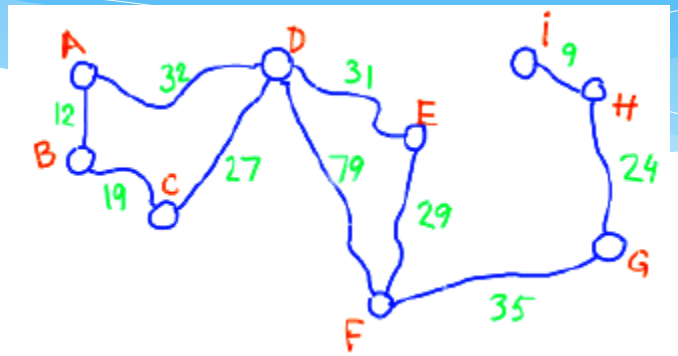


* Dari F ke A
* f(F)=(0+47)=47
* f(E)=(29+39)=68, f(G)=(35+65)=100, f(D)=(79+25)=104
* f(D)=(29+31+25)=85, f(G)=(35+65)=100, f(D)=(79+35)=104, f(F)=(29+29+47)=105
* f(A)=(29+31+32+0)=92, f(G)=(35+65)=100, f(D)=(79+25)=104, f(C)=(29+31+27+20)=107, f(E)=(29+31+31+39)=130, f(F)=(29+31+79+47)=186
* Jalur: F – E – D – A

Pencarian A* optimal!

Pencarian A* di Symbolania (3)





* Dari H ke A
* f(H)=(0+61)=61
* f(I)=(9+50)=59, f(G)=(24+65)=89
* f(H)=(9+9+61)=79, f(G)=89
* f(I)=(9+9+9+50)=77, f(G)=89, f(G)=(9+9+24+65)=107
* f(G)=89, f(H)=(9+9+9+9+61)=97, f(G)=107
* f(H)=97, f(F)=(24+35+47)=106, f(G)=107, f(H)=(24+24+61)=109
* ….

Butuh waktu, tetapi A* dapat menyelesaikannya, sehingga A* dikatakan *complete!*

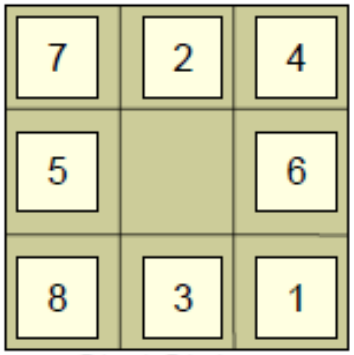
Sifat dari Pencarian A*

* Jika *h(n)* adalah fungsi heuristik yang memungkinkan, maka pencarian A* adalah selesai (*complete)* dan optimal
* Namun dalam sebagian besar kasus, jumlah node di dalam ruang keadaan adalah eksponensial terhadap panjang solusinya
* Waktu komputasi **bukanlah** kendala utama dari A*. Karena A* menyimpan semua node di dalam memori, maka biasanya A* akan kehabisan memori jauh sebelum kehabisan waktu!

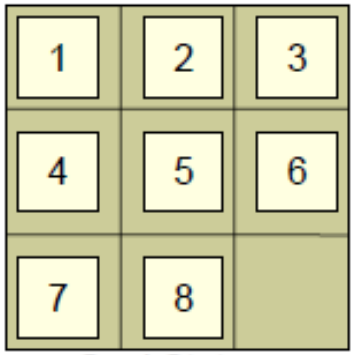
Ringkasan Best-first Search

* Best-first Search mengekspansi node yang memiliki biaya minimal terlebih dahulu (berdasarkan sejumlah perhitungan)
* **Pencarian Greedy** meminimalkan estimasi biaya menuju goal
  * Biasanya dapat mengurangi waktu pencarian
  * Tidak selesai/complete dan tidak optimal
* **Pencarian A*** selesai/complete dan optimal, tetapi memiliki masalah kompleksitas ruang dan waktu

Fungsi Heuristik
Contoh: 8-Puzzle



Start State

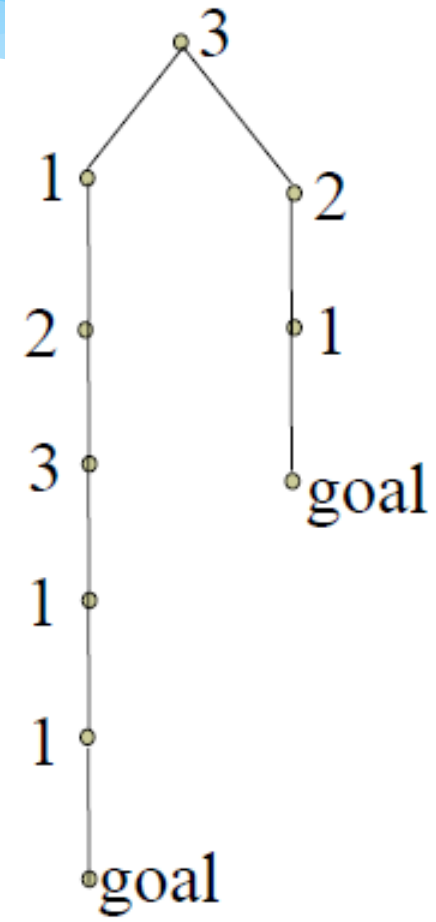


Goal State

* Rata-rata biaya solusi (jumlah langkah yang dibutuhkan) dalam 8-puzzle adalah 22 langkah
* Contoh di atas memiliki biaya solusi sebesar 26 langkah
* Aproksimasi faktor percabangan (*branching factor*) adalah 3

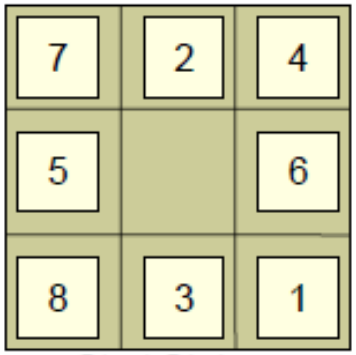
Heuristik yang dapat diterima (1)

* *h(n)* tidak boleh meng-overestimasi biaya untuk mencapai goal
* Heuristik harus optimis. Ia berpikir bahwa biaya untuk mencapai goal lebih kecil dari yang seharusnya
* Pencarian A* adalah optimal jika *h(n)* yang digunakan adalah heuristik yang dapat diterima
* Contoh:
  * Banyaknya garis lurus yang digunakan untuk mencapai Surabaya

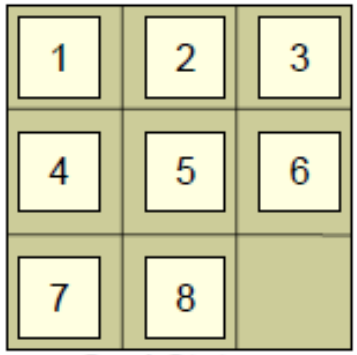


Heuristik yang dapat diterima (2)

* Untuk mendapatkan solusi terpendek dengan A*, diperlukan fungsi heuristik yang tidak meng-overestimasi jumlah langkah menuju goal



Start State

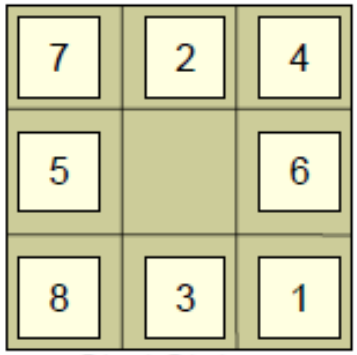


Goal State

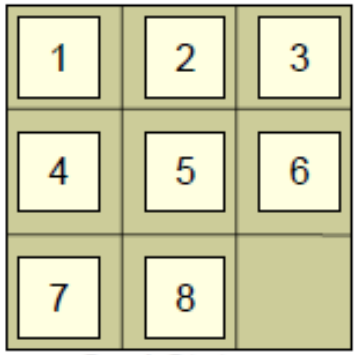
* Rata-rata biaya solusi = 22 langkah

Fungsi Heuristik untuk 8-Puzzle

* Fungsi heuristik
  * Menghitung jumlah keping yang berada pada posisi yang salah
* Tidak ada jaminan bahwa pencarian akan sukses, tapi penentuan fungsi heuristik ini cukup berguna



Start State

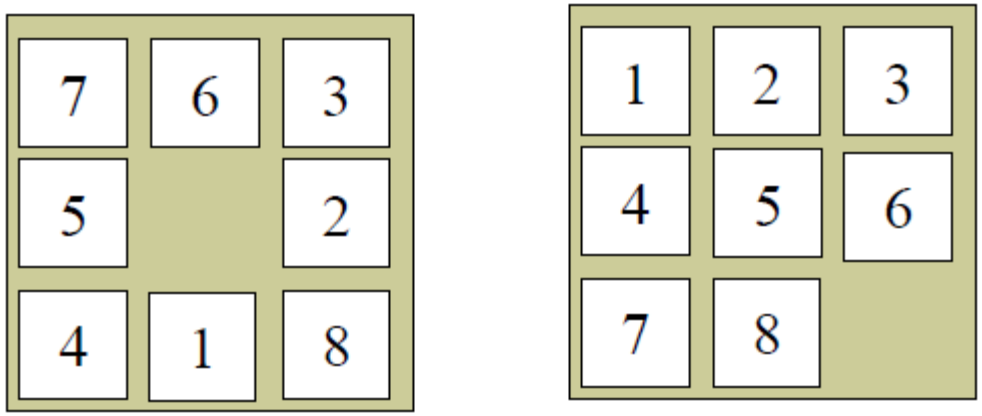


Goal State

Gerakan Keping Kosong	Jumlah Keping Salah Posisi
-	7
Kiri	6
Kanan	8
Atas	8
Bawah	7

Fungsi heuristik 8-puzzle h_1

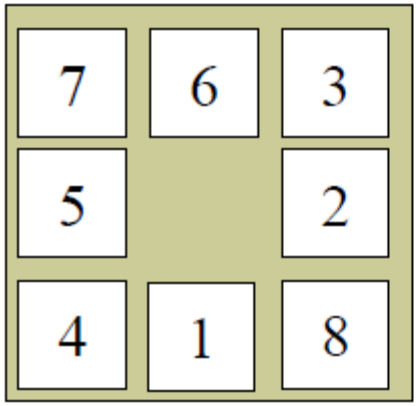
* Untuk 8-puzzle
  * Bagaimana menerjemahkan fungsi heuristik kita – *menghitung jumlah keping yang salah posisi* – menjadi fungsi yang dapat digunakan oleh program komputer?
* *h1(n)*=jumlah keping yang salah posisi

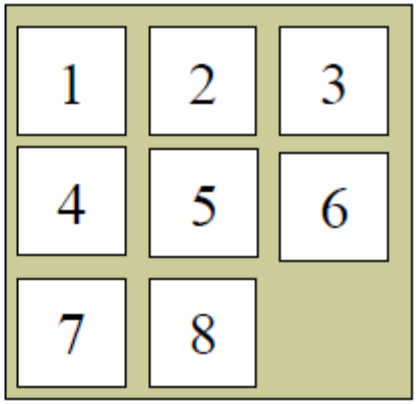


  * $h_1(S)=7$
  * Fungsi heuristik ini dapat diterima, karena sembarang keping yang salah posisi akan digerakkan setidaknya sebanyak satu kali

Fungsi heuristik 8-puzzle h_2

* $h_2(n)$= total jarak Manhattan (Manhattan Distance), yakni jumlah dari kuadrat dari posisi sekarang ke posisi yang dituju

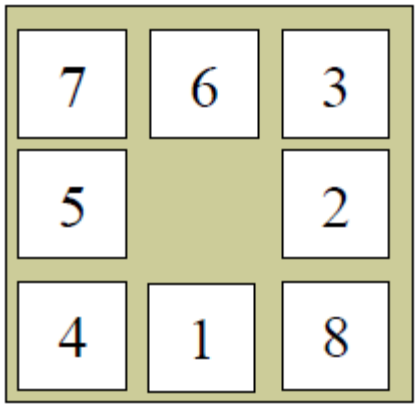


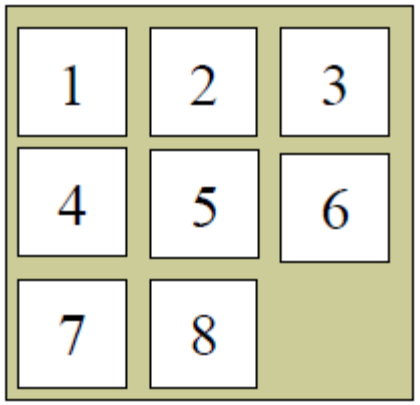


* $h_2(S)$= 3+2+0+1+1+2+2+1 = 12
* Fungsi heuristik ini dapat diterima, karena sembarang langkah yang dilakukan pasti menggerakkan keping mendekati tujuan

Fungsi heuristik 8-puzzle h_3

* *h3(n)*= total jarak Euclidean (Euclidean Distance), yakni panjang garis lurus dari posisi sekarang ke posisi yang dituju





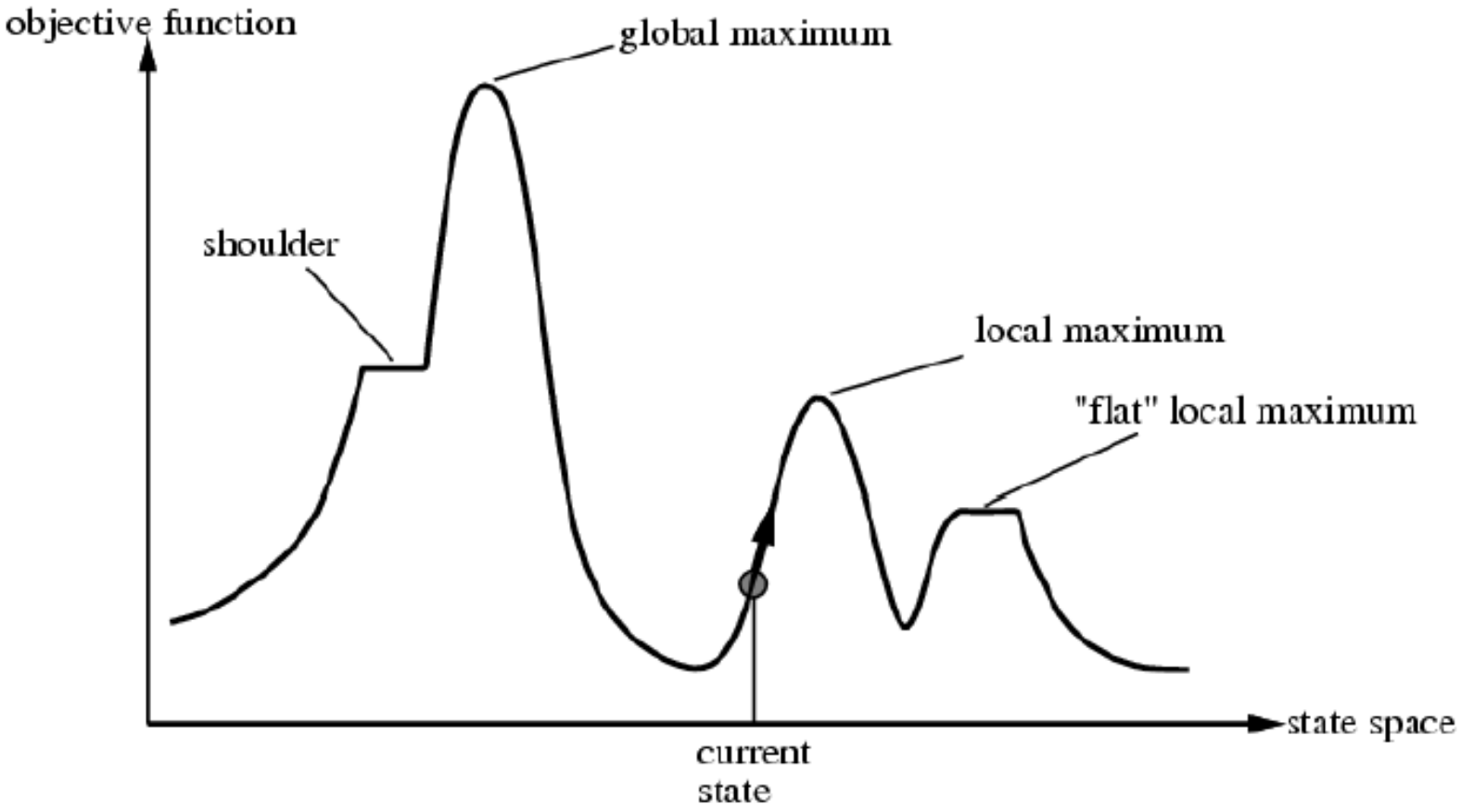
* h3(S)=2+2,23+0+1+1,41+1+2,23+1=10,87
* Fungsi heuristik ini dapat diterima, karena dengan jarak yang ≠ 0 berarti keping akan digerakkan mendekati tujuan

Tugas Demo

* Selesaikan permasalahan 24-Puzzle dengan menggunakan fungsi heuristik h1, h2, dan h3 dan dipasangkan dengan pencarian Greedy dan A*:
  * Pencarian Greedy dengan h1
  * Pencarian Greedy dengan h2
  * Pencarian Greedy dengan h3
  * Pencarian A* dengan h1
  * Pencarian A* dengan h2
  * Pencarian A* dengan h3

Algoritma Local Search: Peningkatan secara Iteratif

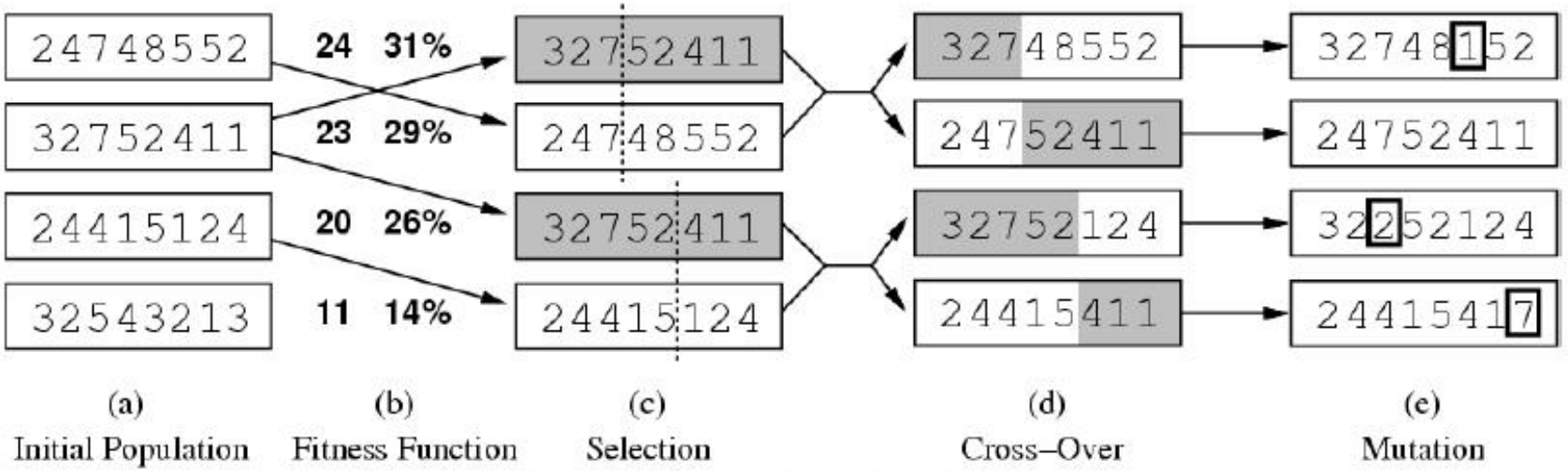
* Mulai dari sebuah konfigurasi dan membuat perubahan secara menaik untuk meningkatkan kualitas solusi
* Tujuan: menemukan *global maximum* dengan memodifikasi state sekarang berdasarkan informasi lokal

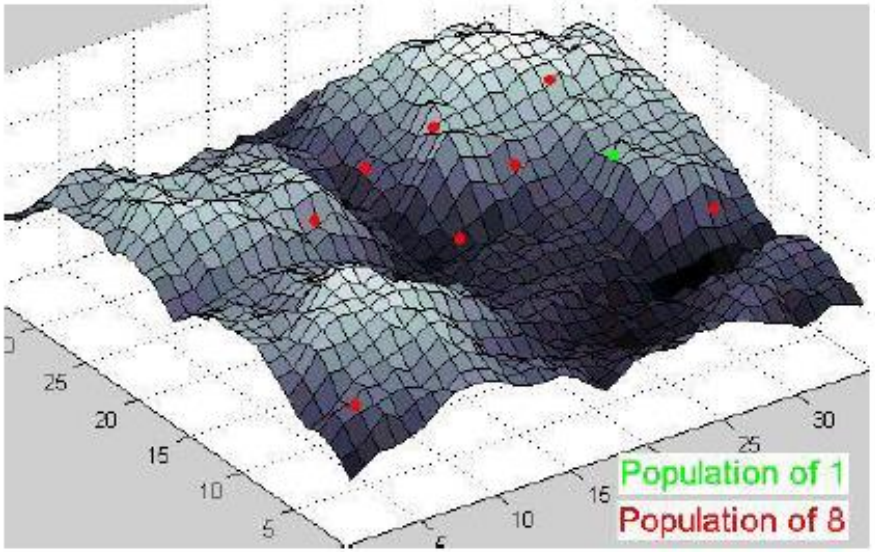

objective function
global maximum
shoulder
local maximum
"flat" local maximum
state space
current
state

Algoritma Local Search

* **Hill Climbing**
  * Mengambil *action* yang dapat meningkatkan state sekarang (pencarian greedy lokal)
* **Simulated Annealing**
  * Memilih *action* secara stokastik (bersifat random atau non-deterministik) dan dapat mentoleransi solusi yang lebih buruk
  * Terinspirasi dari istilah *annealing* pada ilmu metalurgi, yakni teknik yang melibatkan pemanasan dan pendinginan sebuah material untuk meningkatkan ukuran kristal dan mengurangi kerusakan
* **Genetic Algorithm**
  * Pencarian dengan menggunakan populasi state, dimana setiap state dapat mengalami proses "seleksi evolusi" untuk menghasilkan populasi state baru yang lebih "fit/tangguh".
  * Proses evolusi mencakup: seleksi, mutasi, dan cross-over

The Genetic algorithm





Bentuk antena NASA didapat dari perhitungan komputer

Model Pencarian dengan AI lainnya

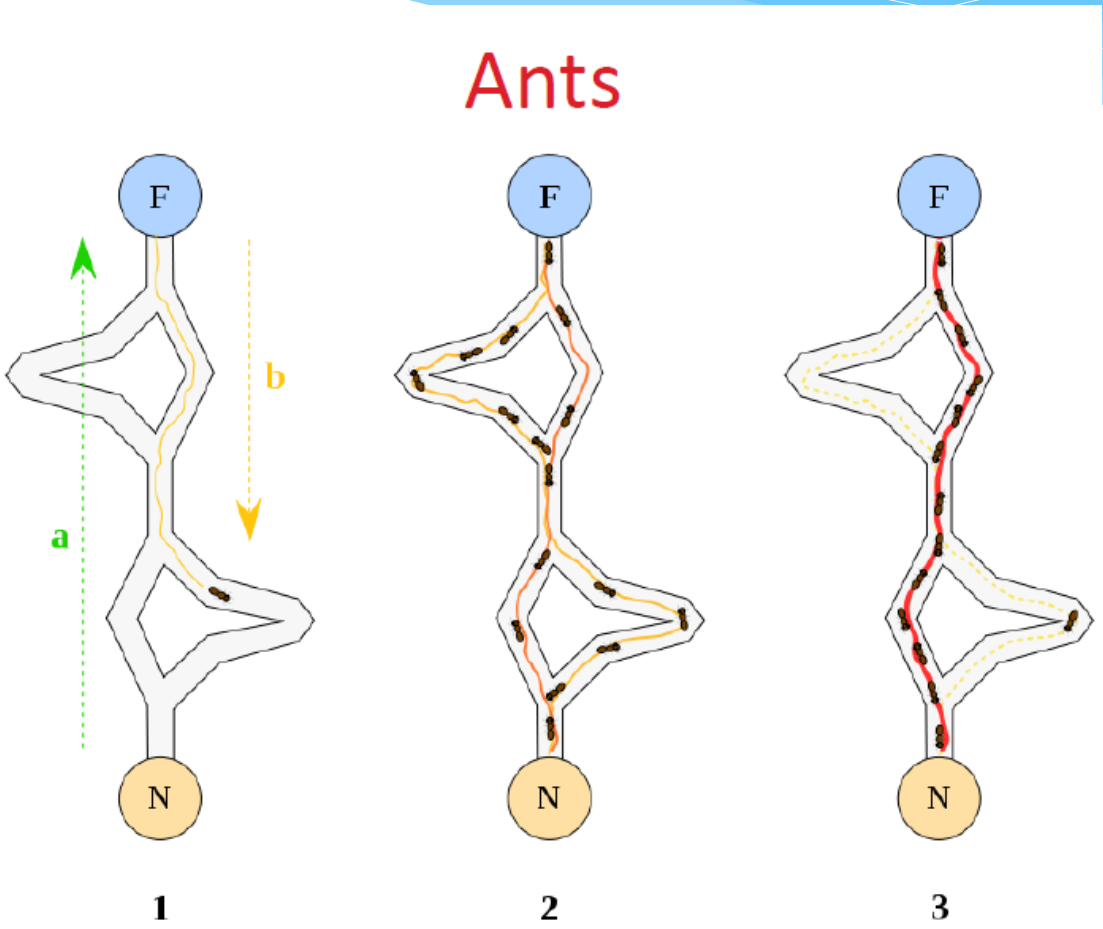
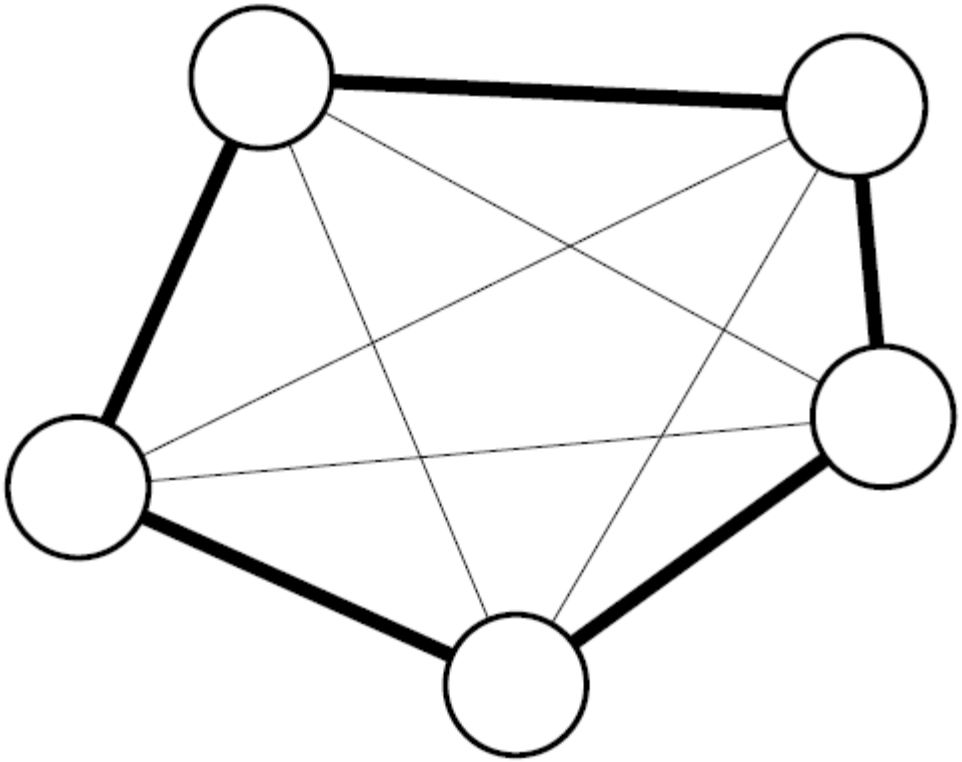


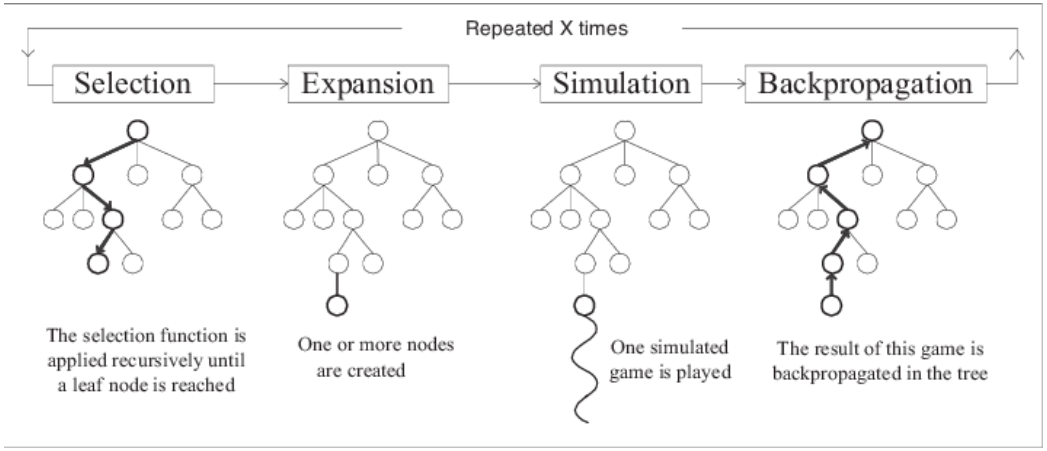
Image from: commons.wikimedia.org

Optimisasi Koloni Semut (Ant Colony) untuk Traveling Salesman Problem



Monte Carlo Tree Search (MCTS)

* Algoritma pencarian heuristic yang digunakan dalam permainan → AlphaGo, Total War: Rome II
* Menganalisis langkah berikutnya yang paling menjanjikan, mengekspansi pohon pencarian berdasarkan *random sampling*
* Aplikasi MCTS dalam permainan didasarkan pada banyak *playout* yang dilakukan
  * Pada setiap *playout*, permainan dimainkan sampai akhir dengan pemilihan langkah yang random
  * Hasil akhir dari setiap *playout* digunakan untuk memberikan bobot node pada tree sehingga node yang "menjanjikan" memiliki kemungkinan yang lebih tinggi untuk terpilih pada *playout* selanjutnya



Ringkasan

* Pencarian Terbimbing (Informed Search)
  * Fungsi heuristic:
    * Best First Search
      * Pencarian Greedy
      * Pencarian A*
    * Monte Carlo Tree Search
* Algoritma Local Search
  * Hill climbing
  * Simulated Annealing
  * Genetic algorithm