

PERTEMUAN 3

PENCARIAN (SEARCH)

- 
- Hal penting dalam menentukan keberhasilan sistem cerdas adalah kesuksesan dalam pencarian.

Pencarian adalah suatu proses mencari solusi dari suatu permasalahan melalui sekumpulan kemungkinan ruang keadaan (state space).

Ruang keadaan = merupakan suatu ruang yang berisi semua keadaan yang mungkin.

Untuk mengukur perfomansi metode pencarian, terdapat empat kriteria yang dapat digunakan :

- Completeness :

Apakah metode tersebut **menjamin penemuan solusi jika solusinya memang ada?**

- Time complexity : Berapa lama **waktu yang diperlukan?**
- Space complexity : Berapa banyak **memori yang diperlukan**
- Optimality : Apakah metode tersebut menjamin menemukan **solusi yang terbaik jika terdapat beberapa solusi berbeda?**

- Pencarian atau pelacakan merupakan salah satu teknik untuk menyelesaikan permasalahan dalam bidang kecerdasan buatan.
- Teknik dasar pencarian masalah memberikan suatu kunci bagi banyak sejarah penyelesaian yang penting dalam bidang kecerdasan buatan. Contoh beberapa aplikasi yang menggunakan teknik pencarian yaitu :
 - ✓ Masalah *routing (travelling salesman problem)*
 - ✓ Parsing bahasa dan interpretasinya
 - ✓ Permainan
 - ✓ logika pemrograman (pencarian fakta dan implikasinya)
 - ✓ Pengenalan pola
 - ✓ Sistem pakar berbasis kaidah (*rule based expert system*)

Teknik Pencarian

Pada dasarnya ada dua teknik pencarian yaitu yang biasanya digunakan, yaitu :

1. Pencarian buta (*blind search*)
2. Pencarian terbimbing (*heuristic search*)

Pencarian Buta

- Pencarian buta merupakan sekumpulan prosedur yang digunakan dalam melacak ruang keadaan. Pencarian berlangsung sampai solusi terakhir ditemukan. Idennya adalah menguji seluruh kemungkinan yang ada untuk menemukan solusi.
- Pendekatan ini kurang efisien dan merupakan pemaksaan (*brute force search*). Dalam memecahkan masalah yang sangat besar sejumlah keadaan baru muncul, sehingga alternatif yang perlu dipertimbangkan pun menjadi lebih banyak. Akibatnya diperlukan waktu yang lama untuk menemukan satu solusi.

Pencarian Heuristic

Kata heuristic berasal dari bahasa Yunani heuriskein dari kata dasar eureka atau heurika yang berarti mengungkap atau menemukan.

Dalam AI, heuristic diperkenalkan sebagai suatu teknik yang meningkatkan efisiensi proses pencarian, yang dimungkinkan dengan mengorbankan kelengkapan.

Heuristic seperti pemandu perjalanan, yang baik untuk tujuan pokok mencari arah yang secara umum menarik, tetapi bisa jadi tidak baik jika mempertimbangkan ketertarikan tiap orang berbeda untuk tiap objek berbeda.

Menggunakan heuristic kita berharap mendapatkan solusi yang baik dari masalah yang sulit

Satu contoh general-purpose heuristic yang baik yang berguna untuk banyak kombinasi masalah adalah *nearest neighbor heuristic*, yang bekerja dengan menyeleksi alternatif lokal terbaik pada tiap langkah.


Aplikasinya adalah dalam masalah *Travelling Salesman*, yang menggunakan beberapa prosedur berikut :


1. Pilih secara acak satu kota sebagai awal perjalanan
2. Untuk memilih kota berikut, lihat semua kota yang belum dikunjungi dan pilih yang terdekat lalu kunjungi.
3. Ulangi langkah 2 sampai semua kota dikunjungi.

Karakteristik Masalah

Pencarian heuristic adalah metode yang sangat umum yang dapat diterapkan dalam begitu banyak masalah, meliputi begitu banyak variasi teknik yang spesifik, dimana masing-masing efektif untuk penyelesaian masalah tertentu yang lebih spesifik.

Untuk memilih metode mana (atau kombinasi metode mana) yang akan digunakan untuk menyelesaikan masalah, penting untuk menganalisa masalah pada beberapa dimensi kunci atau karakteristik, sebagai berikut:

- 
- Dapatkah masalah disederhanakan kedalam kelompok terpisah yang lebih kecil atau subprogram yang lebih mudah ?
 - Dapatkah satu tahap penyelesaian solusi diabaikan atau setidaknya tidak dilakukan jika terbukti tidak layak ?
 - Apakah ruang lingkup masalah dapat diprediksi ?
 - Dapatkah dinyatakan sebuah solusi yang baik untuk penyelesaian masalah tanpa membandingkannya dengan solusi lain yang mungkin ?
 - Solusi yang diinginkan adalah sebuah stata atau jalur menuju stata ?

- 
- Apakah sejumlah pengetahuan mutlak diperlukan untuk menyelesaikan masalah atau pengetahuan hanya diperlukan untuk membatasi pencarian ?
 - Dapatkah komputer yang diberikan permasalahan langsung memberikan solusi atau pemecahan masalah memerlukan interaksi antara komputer dan manusia ?

Teknik Search

- Arah search
Dapat dilakukan :
Maju, bermula dari keadaan awal (*start state*)
Mundur, diawali dari keadaan tujuan (*goal state*)
- Topologi proses *search*
Ada dua macam penggambaran problem, yaitu dalam bentuk :
 1. Pohon (*tree*)
 2. Graf (*graph*) :Graf berarah dan Graf tidak berarah

Metode Search

Beberapa metode search yang akan dipelajari :

1. Breadth-First-Search
2. Depth-First-Search
3. Generate-and-Test
4. Hill-Climbing
5. Best-First-Search

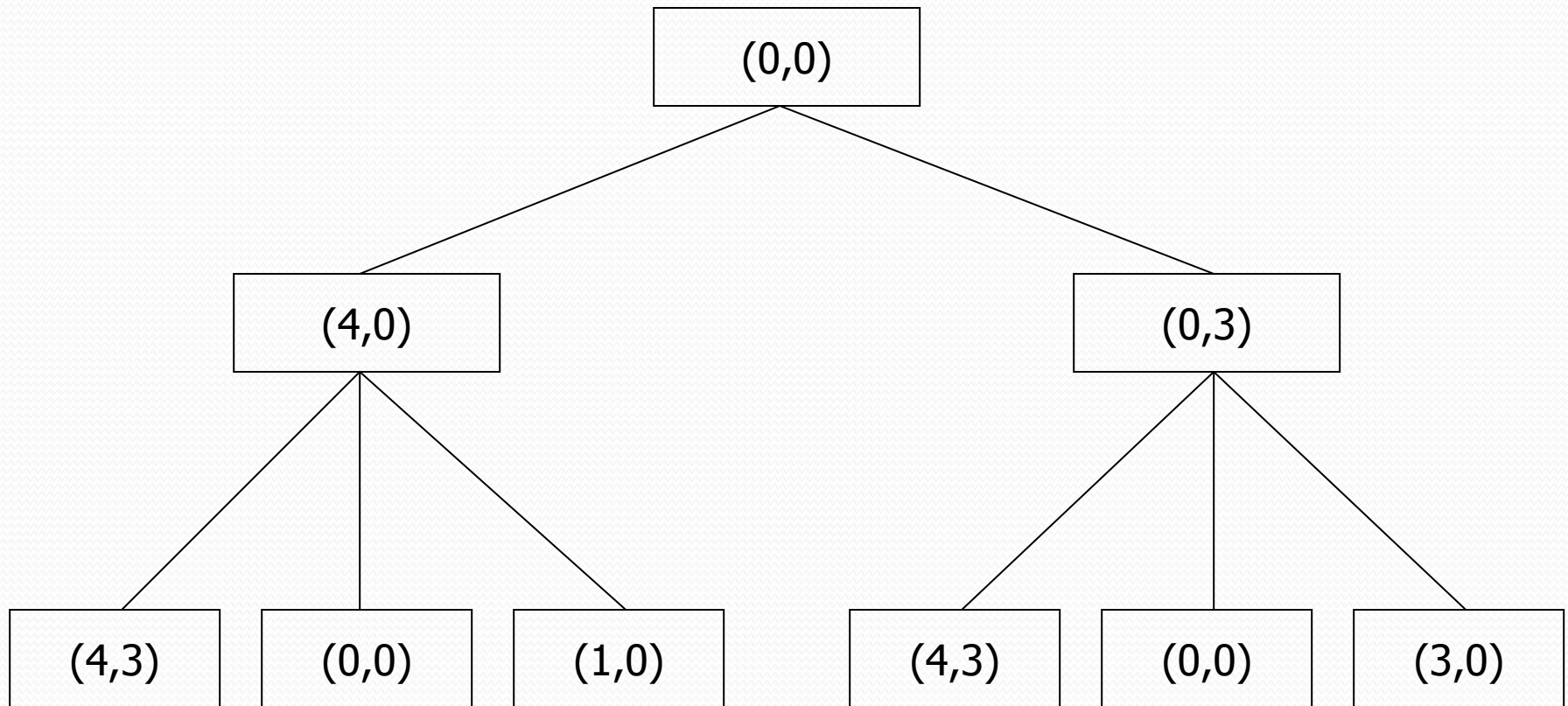
Pencarian buta (1,2), pencarian heuristic
(3,4,5)

Breadth-First-Search

Algoritma Breadth-First-Search :

1. Bentuk variabel dengan nama NODE-LIST dan jadikan sebagai initial state.
2. Sampai goal state ditemukan atau NODE-LIST kosong, lakukan :
 - a. ambil elemen pertama dari NODE-LIST, sebut E.
jika NODE-LIST kosong, quit.
 - b. Untuk tiap cara dimana tiap aturan(fungsi) dapat cocok dengan stata di E, lakukan :
 - i. Gunakan aturan(fungsi) untuk menuju stata baru
 - ii. Jika stata baru adalah goal state, quit return stata ini
 - iii. Jika bukan, tambahkan stata baru di akhir NODE-LIST.

Breadth-First Search Tree untuk masalah bejana air



Kebaikan dan Keburukan Breadth-First-Search

Kebaikan Breadth-First-Search :

- Breadth-First-Search tidak akan terjebak untuk menelusuri satu jalur tertentu saja
- Jika solusi memang ada, maka dijamin Breadth-First-Search akan menemukannya.

Keburukan Breadth-First-Search :

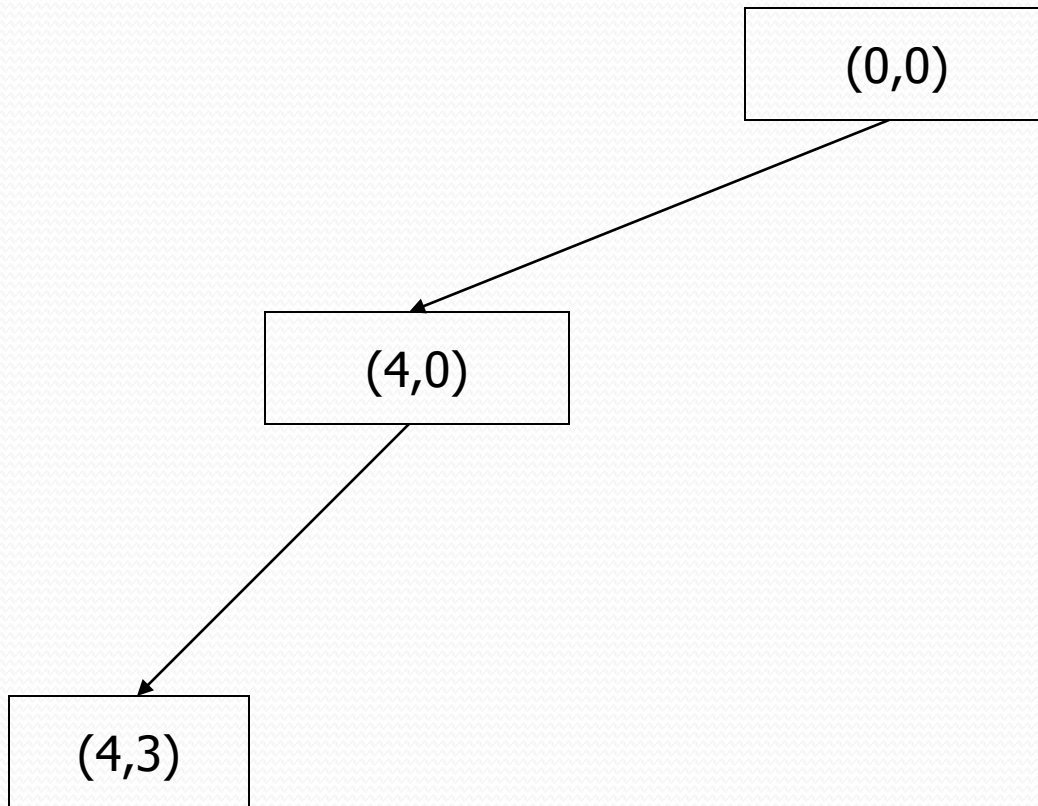
- Memerlukan memori lebih besar karena harus menyimpan semua simpul dari tree yang ditelusuri
- Harus menelusuri semua bagian tree pada level yang sama sebelum beralih ke level berikutnya.

Depth-First-Search

Algoritma Depth-First-Search :

1. Jika initial state adalah goal state, quit dan return success
2. Jika bukan, lakukan dibawah ini sampai dicapai sinyal success atau gagal
 - a. Tentukan successor, E dari initial state. Jika tidak ada lagi successor, maka sinyal gagal
 - b. Jalankan Depth-First-Search dengan E sebagai initial state
 - c. Jika success dihasilkan, sinyal success. Jika tidak maka ulangi langkah 2

Depth-First Search Tree untuk masalah bejana air



Kebaikan dan Keburukan Depth-First-Search

Kebaikan Depth-First-Search :

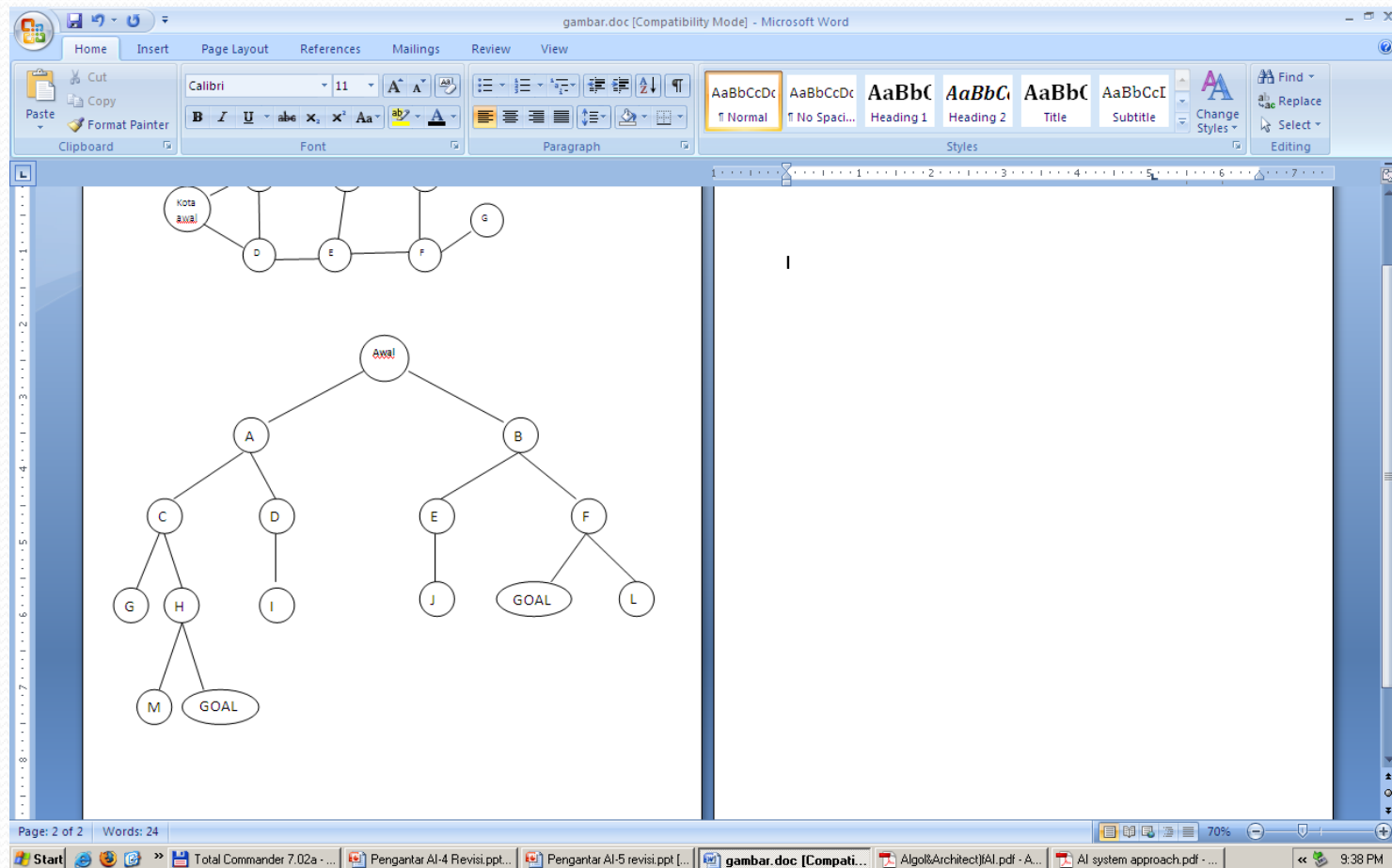
- Depth-First-Search memerlukan ruang memori lebih kecil karena hanya menyimpan simpul-simpul dari path/jalur yang sedang dikerjakan.
- Dapat menemukan solusi tanpa menelusuri terlalu banyak ruang search.

Keburukan Depth-First-Search :

- Ada kemungkinan terjebak pada satu jalur sampai terlalu jauh, bahkan selamanya, sebelum jalur tsb mendapatkan stata yang tidak lagi memiliki successor (buntu).
- Mungkin menemukan jalur panjang ke solusi pada satu bagian dari tree, sementara jalur terpendek tersedia pada bagian lain tree yang belum ditelusuri

Tambahan Contoh Kasus

Representasi masalah dalam tree



Menggunakan teknik *breadth-first search*

Node awal adalah awal dan tujuan adalah Goal, langkah-langkahnya :

1. Open := [Awal]; closed := []
2. Open:= [A,B]; closed:= [Awal]
3. Open:= [B,C,D]; closed:= [A,Awal]
4. Open:= [C,D,E,F]; closed:= [B,A,Awal]
5. Open:= [D,E,F,G,H]; closed:= [D,C,B,A,Awal]
6. Open:= [E,F,G,H,I]; closed:= [D,C,B,A,Awal]
7. Open:= [F,G,H,I,J]; closed:= [E,D,C,B,A,Awal]
8. Open:= [G,H,I,J,GOAL,L]; closed:= [F,E,D,C,B,A,Awal]
9. OPEN := [H,I,J,GOAL,L]; closed:= [G,F,E,D,C,B,A,Awal]
10. Open:= [I,J,GOAL,L]; closed:= [H,G,F,E,D,C,B,A,Awal]
11. Open:= [GOAL,L]; closed:= [I,J,H,G,F,E,D,C,B,A,Awal]
12. Open:= [L]; closed:= [GOAL,I,J,H,G,F,E,D,C,B,A,Awal]

Menggunakan teknik *depth-first search*

Node awal adalah awal dan tujuan adalah Goal, langkah-langkahnya :

1. Open := [Awal]; closed := []
2. Open := [A,B]; closed := [Awal]
3. Open := [C,D,B]; closed := [A,awal]
4. Open := [G,H,D,B]; closed := [C,A,awal]
5. Open := [H,D,B]; closed := [G,C,A,awal]
6. Open := [M,Goal,D,B]; closed := [G,C,A,awal]
7. Open := [Goal,D,B]; closed := [M,G,C,A,awal]
8. Open := [D,B]; closed := [Goal,M,G,C,A,awal]

Generate-and-Test

Teknik Generate-and-Test adalah teknik yang paling mudah dibandingkan teknik search yang lain, namun relatif lebih lama dalam mendapatkan solusi.

Algoritma Generate-and-Test :

1. Bentuk solusi yang mungkin.
Untuk beberapa masalah, ini berarti membentuk poin terpisah dari area permasalahan. Pada masalah lain, ini berarti membentuk jalur dari stata awal.
2. Lakukan test untuk melihat apakah poin yang ditemui adalah solusi dengan membandingkan poin yang dipilih atau poin terakhir dari jalur yang dipilih dengan kumpulan stata tujuan
3. Jika solusi sudah ditemukan, quit. Jika belum kembali ke langkah 1.

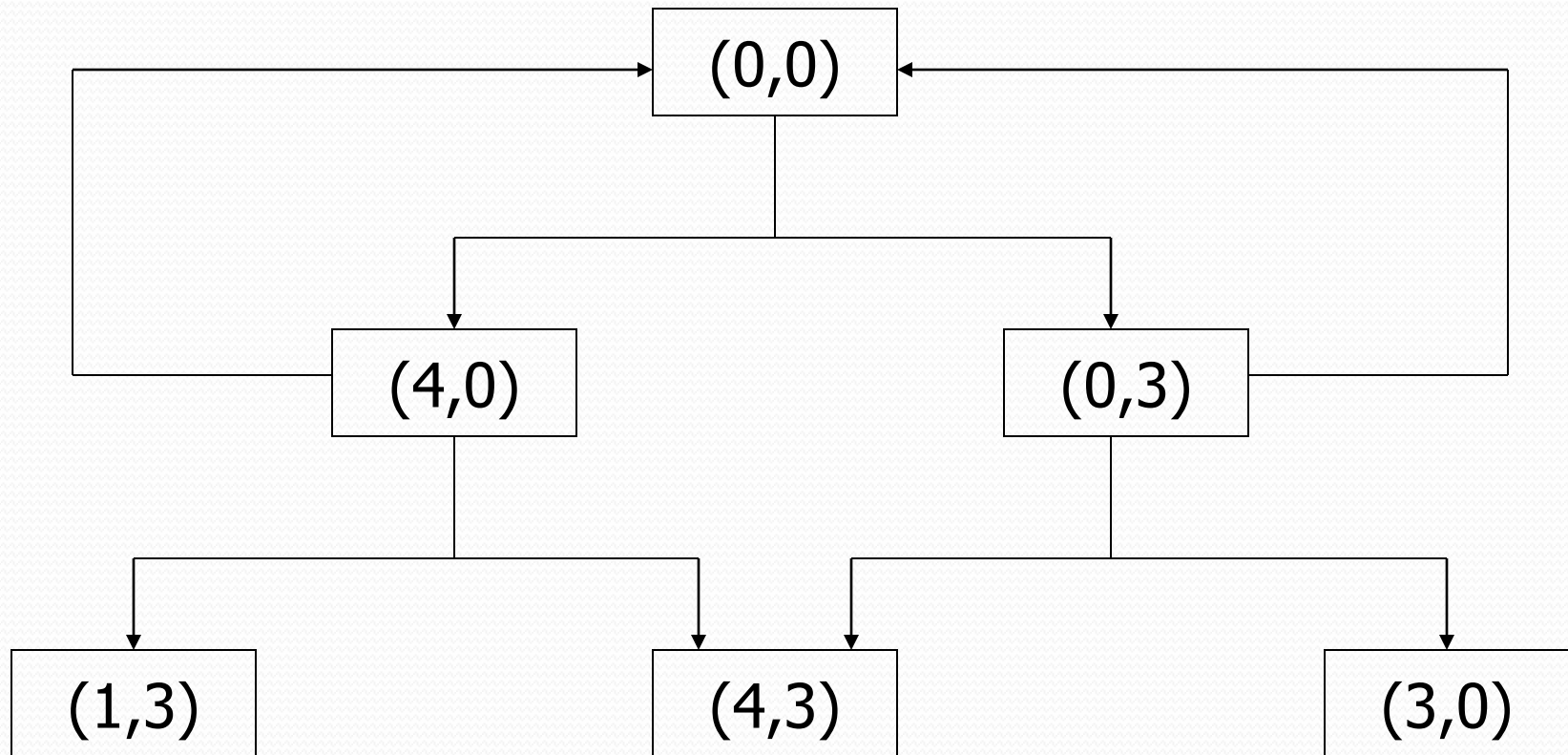
Kebaikan dan Keburukan Generate-and-Test

Jika penurunan solusi yang mungkin dilakukan secara sistematis, maka procedure diatas akan dapat menemukan solusi suatu saat, jika memang ada. Tapi sayangnya jika ruang permasalahan sangat luas maka saat ditemukannya solusi akan menjadi sangat lama.

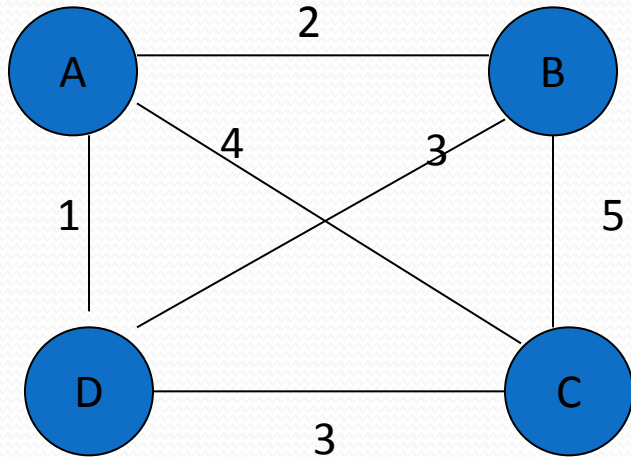
Cara terbaik menerapkan generate-and-test yang sistematis adalah pada tree dari depth-first search dengan backtracking, yaitu kembali ke stata sebelumnya bila ditemui stata yg sudah pernah di test atau memodifikasi prosedurnya untuk menelusuri stata pada bentuk graph.

Contoh Kasus 1

Search Graph untuk masalah bejana air



Contoh kasus TSP




Sebuah rute yang harus dilewati seorang sales dimana sales tersebut harus melewati setiap kota tepat sekali. Terdapat 4 kota, dengan jarak masing-masing kota $AB=2$, $AC=4$, $AD=1$, $BC=5$, $BD=3$, $CD=3$. Tujuannya adalah mencari jarak terpendek bagi sales untuk mengunjungi semua kota sekali.

Penyelesaian menggunakan generate-test adalah dengan membangkitkan solusi-solusi yang mungkin ada sesuai permasalahan yang dihadapi oleh sales tersebut. Kombinasi abjad sebagai solusi yang mungkin adalah $n! = 4! = 24$.

Tujuannya adalah mencari solusi dengan panjang terpendek.

No pencarian	Lintasan	Panjang Lintasan	Lintasan yang dipilih	Panjang lintasan
1	ABCD	10	ABCD	10
2	ABDC	8	ABDC	8
3	ACBD	12	ABDC	8
4	ACDB	10	ABDC	8
5	ADCB	9	ABDC	8
6	ADCB	9	ABDC	8
7	BACD	9	ABDC	8
8	BADC	6	BADC	6
9	BCAD	10	BADC	6
10	BCDA	9	BADC	6
11	BDAC	8	BADC	6
12	BDCA	10	BADC	6
13	CABD	9	BADC	6

14	CADB	8	BADC	6
15	CBAD	8	BADC	6
16	CBDA	9	BADC	6
17	CDAB	6	BADC/CDAB	6
18	CDBA	8	BADC/CDAB	6
19	DABC	8	BADC/CDAB	6
20	DACB	10	BADC/CDAB	6
21	DBAC	9	BADC/CDAB	6
22	DBCA	12	BADC/CDAB	6
23	DCAB	9	BADC/CDAB	6
24	DCBA	10	BADC/CDAB	6




Dari tabel diatas, solusi pertama yang dibangkitkan adalah ABCD = 10, solusi kedua ABDC=8. Ternyata solusi kedua menghasilkan jarak yang lebih pendek sehingga dipilih lintasan ABDC=8. Lakukan untuk langkah selanjutnya. Pada tabel didapat solusi terpendek adalah BADC atau CDBA.

Kelemahan dari teknik ini perlu dibangkitkan semua kemungkinan yang ada sehingga apabila ditambahkan satu kota untuk permasalahan TSP ini diatas 5 kota. Maka akan diperlukan 120 kombinasi lintasan, kecuali diberikan kondisi tertentu misalnya kota awal bagi sales telah ditentukan.

Hill Climbing

Teknik Hill Climbing adalah pengembangan dari teknik Generate-and-Test, dengan penambahan adanya umpan balik dari prosedur test yang sudah digunakan untuk membantu memilih arah mana yang harus ditelusuri pada setiap area search.


Pada prosedur Generate-and-Test yang murni, fungsi test hanya ditanggapi dengan Ya atau Tidak. Tetapi pada Hill-Climbing fungsi test ditambahkan dengan fungsi heuristic atau fungsi objectif yang memungkinkan perkiraan seberapa dekat simpul yang ditelusuri terhadap goal state.



Hill-climbing sering kali digunakan jika fungsi heuristic yang baik tersedia untuk mengevaluasi state, tapi ketika tidak ada lagi pengetahuan yang dapat digunakan.

Sebagai contoh, anda berada disuatu kota yang belum pernah anda kunjungi tanpa memiliki peta. Tujuannya menuju gedung tertinggi yang terlihat dari tempat anda berada.

Fungsi heuristic adalah hanya masalah jarak antara lokasi anda berada dengan letak gedung tertinggi dan bagaimana menemukan jarak yang terdekat atau cara tercepat menuju gedung tertinggi.



Penyelesaian masalah diatas dimulai dengan meninjau karakteristik masalah, apakah solusi yang pertama ditemukan dapat diterima sebagai solusi yang baik ? (mutlak atau relatif ?) Karena tidak ada peta dan tidak ada pengalaman memilih jalan (tidak ada pengetahuan) maka dipilih saja jalan yang arahnya menuju solusi sampai kita tiba di tujuan tanpa mengulangi atau mencoba lagi jalur yang lain dan kita terima itu sebagai solusi terbaik (dgn mengabaikan kemungkinan lain).

Jadi adalah masuk akal menerapkan hill-climbing ketika tidak ada alternatif yang dapat diterima untuk memilih atau menuju pada suatu stata.

Algoritma Simple Hill Climbing :

1. Evaluasi initial state. Jika ini goal state maka return dan keluar. Jika bukan maka lanjutkan dengan initial state sebagai current state.
2. Ulangi langkah berikut sampai menemukan solusi atau sampai tidak ada lagi operator yang dapat digunakan pada current state
 - a. Pilih operator yang belum digunakan pada current state dan gunakan untuk menghasilkan/menuju stata baru
 - b. Evaluasi stata baru.
 - i. Jika ini goal state maka return dan keluar
 - ii. Jika bukan goal state tetapi lebih baik dari current state maka jadikan stata baru sebagai current state
 - iii. Jika tidak lebih baik dari current state lanjutkan perulangan

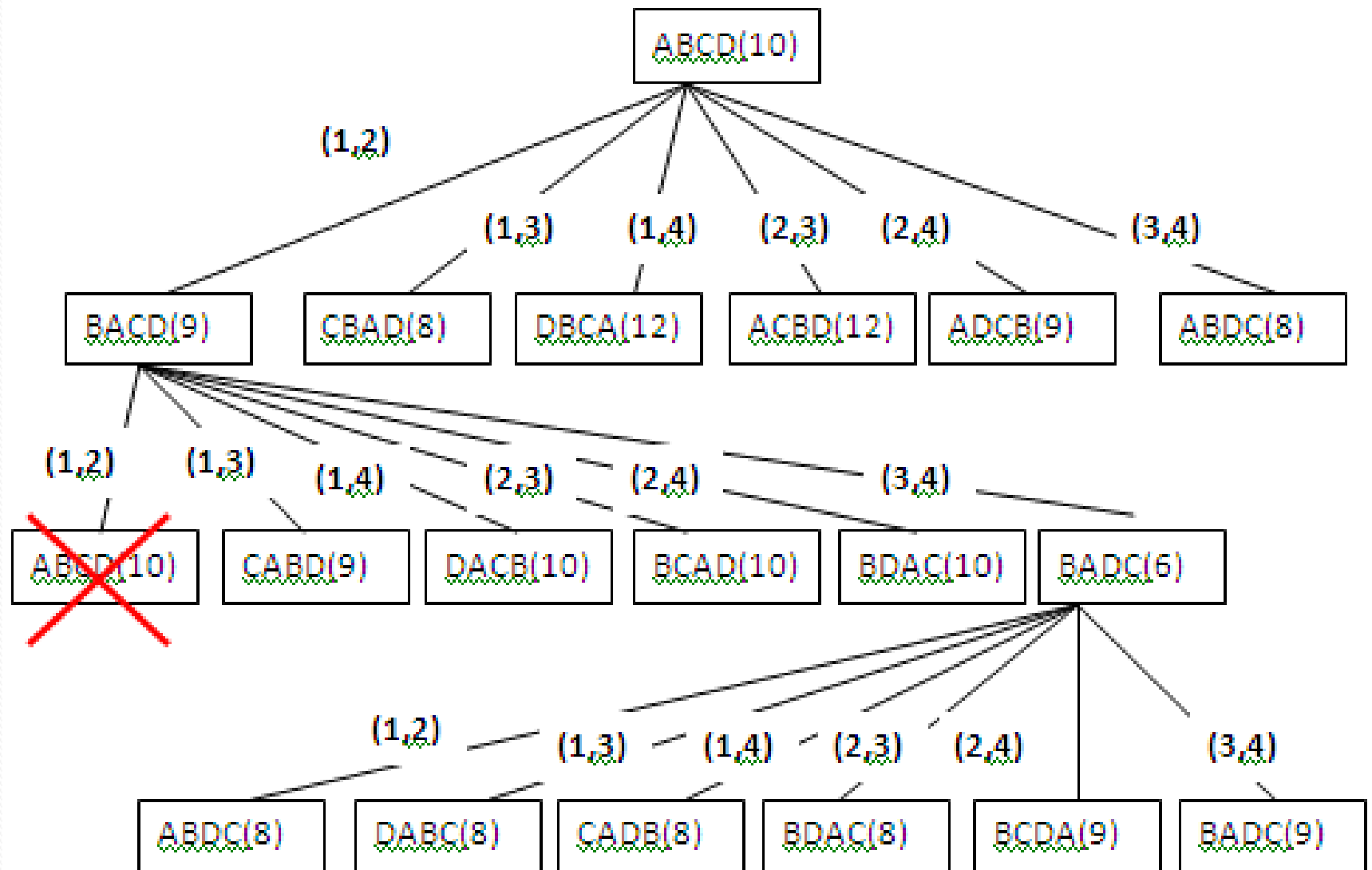
Sebagai ilustrasi teknik pencarian *simple hill climbing* digunakan contoh masalah TSP pada masalah *generate and test* . Operator yang digunakan adalah operator yang dapat menghasilkan kombinasi lintasan kota yang berbeda-beda, yaitu dengan cara menukar posisi masing-masing kota. Untuk mempermudah penukaran posisi, kita cukup menukar posisi 2 kota, operator untuk kombinasi lintasan dengan menukar posisi 2 kota dapat dihitung dengan kalkulasi

$$\frac{4!}{2! (4-2)!}$$

Yaitu :

1. (1,2) menukar posisi kota kesatu dan kedua
2. (1,3) menukar posisi kota kesatu dan ketiga
3. (1,4) menukar posisi kota kesatu dengan keempat
4. (2,3) menukar posisi kota kedua dengan kota ketiga
5. (2,4) menukar posisi kota kedua dengan keempat
6. (3,4) menukar posisi kota ketiga dengan keempat

Penggunaan pengurutan operator harus konsisten, tidak boleh berbeda tiap levelnya. urutan penggunaan operator juga sangat menentukan kecepatan dalam menemukan solusi.



Pencarian *simple hill climbing* dimulai dari anak kiri. Apabila nilai heuristik anak kiri lebih baik maka dibuka untuk pencarian selanjutnya. Jika tidak maka akan dilihat tetangga dari anak kiri tersebut, dan seterusnya.

Level 1 : (ABCD=10 > BACD =9) buka node BACD tanpa harus mencek node yang selevel dengan BACD.

Level 2 : node ABCD dilewati.

(BACD=9 = CABD=9) periksa node tetangga CABD

(BACD=9 < DABC=10) periksa node tetangga DABC

(BACD=9 < BCAD=10) periksa node tetangga BCAD

(BACD=9 < BDAC=10) periksa node tetangga BDAC

(BACD=9 > BADC=6) buka node BADC

Level 3 : (BADC=6 < ABDC=8) periksa tetangga ABDC

(BADC=6 < DABC=8) periksa tetangga DABC

(BADC=6 < CADB=8) periksa tetangga CADB

(BADC=6 < BDAC=8) periksa tetangga BDAC

(BADC=6 < BCDA=9) periksa tetangga BCDA

(BADC=6 < BADC=9) selesai.

Best-First-Search

Teknik Best-First-Search adalah teknik search yang menggabungkan kebaikan yang ada dari teknik Depth-First-Search dan Breadth-First-Search.

Tujuan menggabungkan dua teknik search ini adalah untuk menelusuri satu jalur saja pada satu saat, tapi dapat berpindah ketika jalur lain terlihat lebih menjanjikan dari jalur yang sedang ditelusuri. Untuk mendapatkan jalur yang menjanjikan adalah dengan memberikan skala prioritas pada setiap state saat dihasilkan dengan fungsi heuristic.

Untuk menggunakan Best-First-Search, kita memerlukan dua daftar simpul, yaitu :

1. OPEN

berisi simpul yang dihasilkan dari fungsi heuristic tapi belum dievaluasi, memiliki antrian prioritas dimana elemen dengan prioritas tertinggi adalah yang memiliki nilai paling baik yang dihasilkan fungsi heuristic.

2. CLOSED

berisi simpul yang sudah dievaluasi. Kita perlu tetap menyimpan simpul-simpul ini dalam memori jika kita ingin melakukan search pada Graph, sehingga jika kita menemui suatu simpul kita bisa memeriksa apakah simpul ini sudah pernah dievaluasi atau belum

Algoritma Best-First-Search :

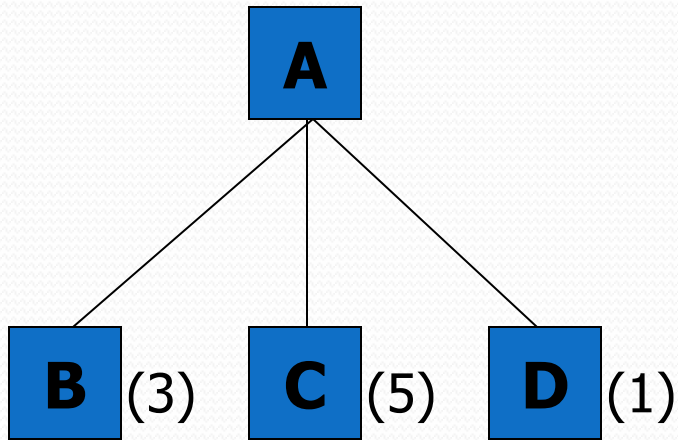
1. Mulai dengan OPEN hanya berisi initial state
2. Sampai goal ditemukan atau tidak ada lagi simpul yang tersisa dalam OPEN, lakukan :
 - a. Pilih simpul terbaik dalam OPEN
 - b. Telusuri successor-nya
 - c. Untuk tiap successor, lakukan :
 - i. Jika belum pernah ditelusuri sebelumnya, evaluasi simpul ini, tambahkan kedalam OPEN dan catat parentnya.
 - ii. Jika sudah pernah ditelusuri, ganti parent nya jika jalur baru lebih baik dari sebelumnya.

Contoh Best First Search

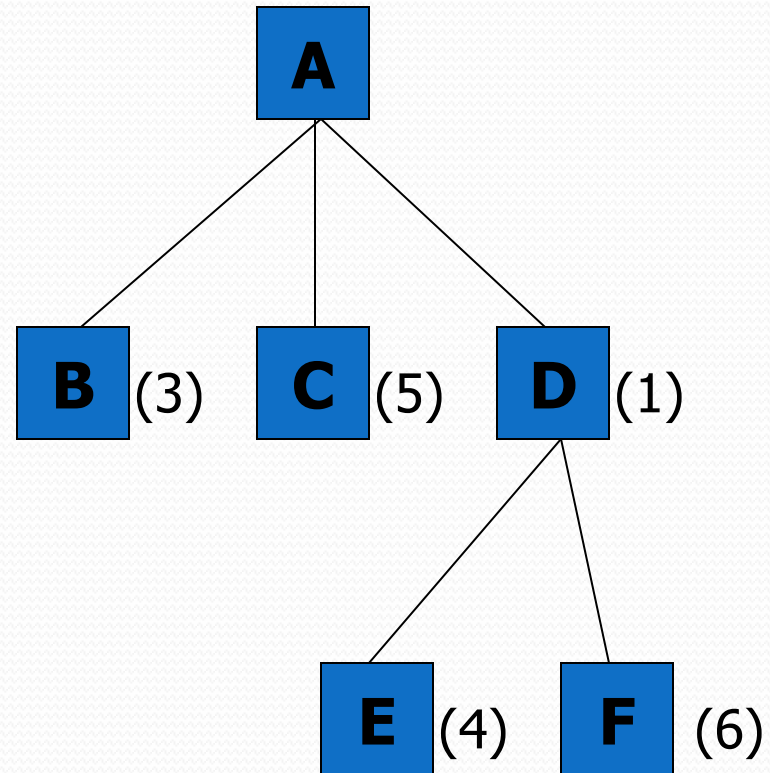
Step 1



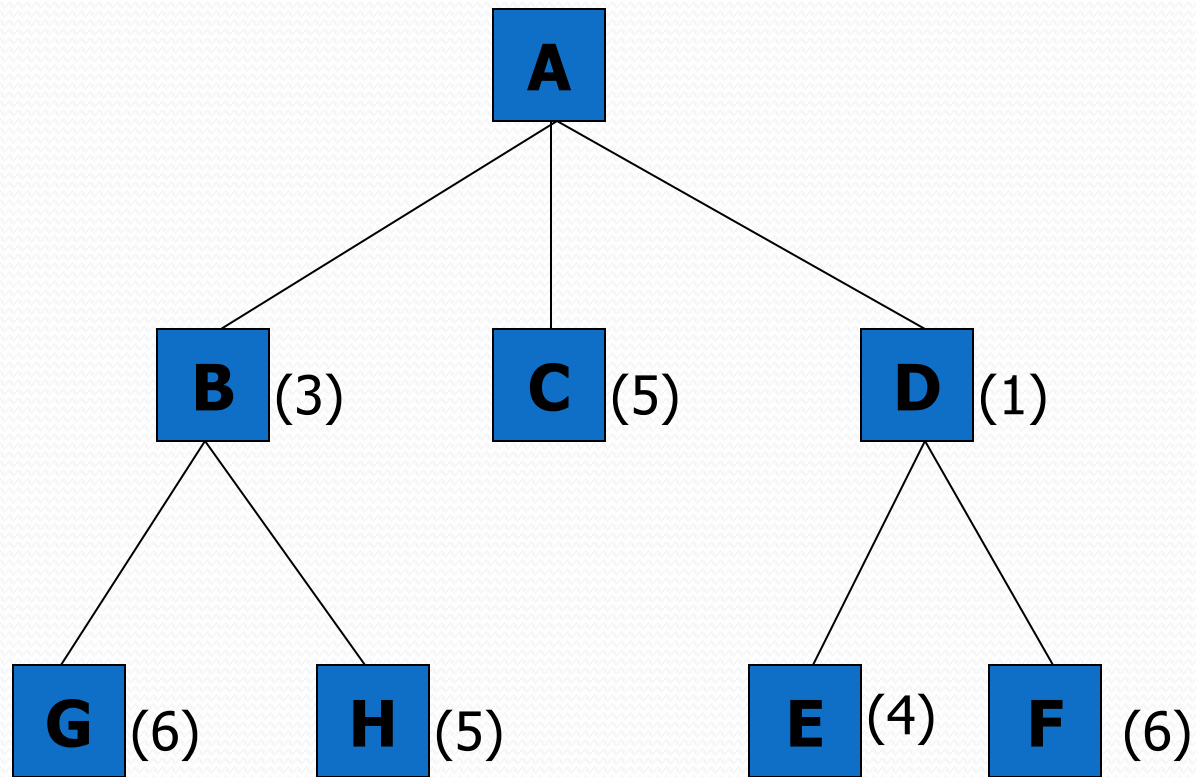
Step 2



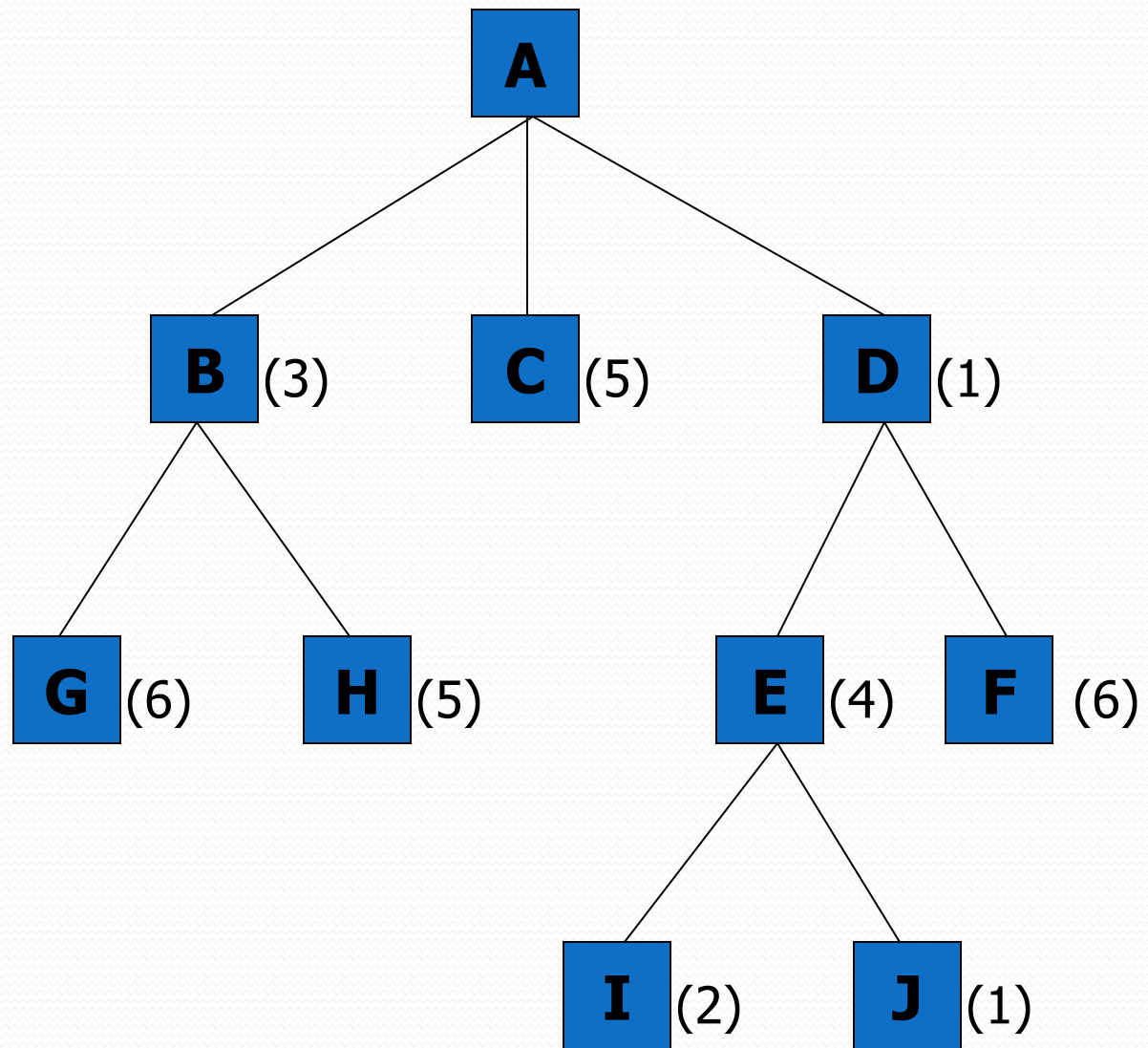
Step 3



Step 4



Step 5



Contoh lain *Best-First-Search*

1	2	
4	5	3
7	8	6

1	2	3
4	5	6
7	8	

Diketahui sebuah puzzle berukuran 3X3 yang berisi angka. Permasalahan adalah angka-angka dalam puzzle tersebut belum teratur.

Nilai awal puzzle :

Goal :

Nilai awal = {1,2,blank,4,5,3,7,8,6} Goal = {1,2,3,4,5,6,7,8,blank}

Nilai heuristic = $f(n) = g(n) + h(n)$

$g(n)$ = kedalaman dari pohon

$h(n)$ = jumlah angka yang masih salah posisi

1	2	
4	5	3
7	8	6

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

{1,2,blank,4,5,3,7,8,6}

$$f(n) = 0 + 3$$

{1,2,3,4,5,blank,7,8,6}

$$f(n) = 1 + 2 = 3$$

1	2	3
4	5	
7	8	6

{1,blank,2,4,5,3,7,8,6}

$$f(n) = 1 + 4 = 5$$

1		2
4	5	3
7	8	6

Level 1

Level 2

1	2	3
4	5	6
7	8	

{1,2,3,4,5,6,7,8,blank}

$$f(n) = 2 + 0 = 2$$

1	2	3
4		5
7	8	6

{1,2,3,4,blank,5,7,8,6}

$$f(n) = 2 + 3 = 5$$

1	2	
4	5	3
7	8	6

{1,2,blank,4,5,3,7,8,6}

$$f(n) = 2 + 3 = 5$$

Latihan soal di rumah

Sebuah puzzle berukuran 3×3

Nilai awal :

2	8	3
1	6	4
7		5

Goal :

1	2	3
8		4
7	6	5

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

$g(n)$ = kedalaman pohon

$h(n)$ = jumlah angka yang salah posisi



THE END