PERTEMUAN 4

SEARCH

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

Pencarian = 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 interprestasinya
- 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 serach)*

Pencarian Buta

Pencarian buta merupakan sekumpulan prosedur yang digunakan dalam melacak ruang keadaan. Pencarian berlangsung sampai solusi terakhir ditemukan. Idenya 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 pengatahuan 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-Fisrt-Search
3. Generate-and-Test
4. Hill-Climbing
5. Best-First-Search

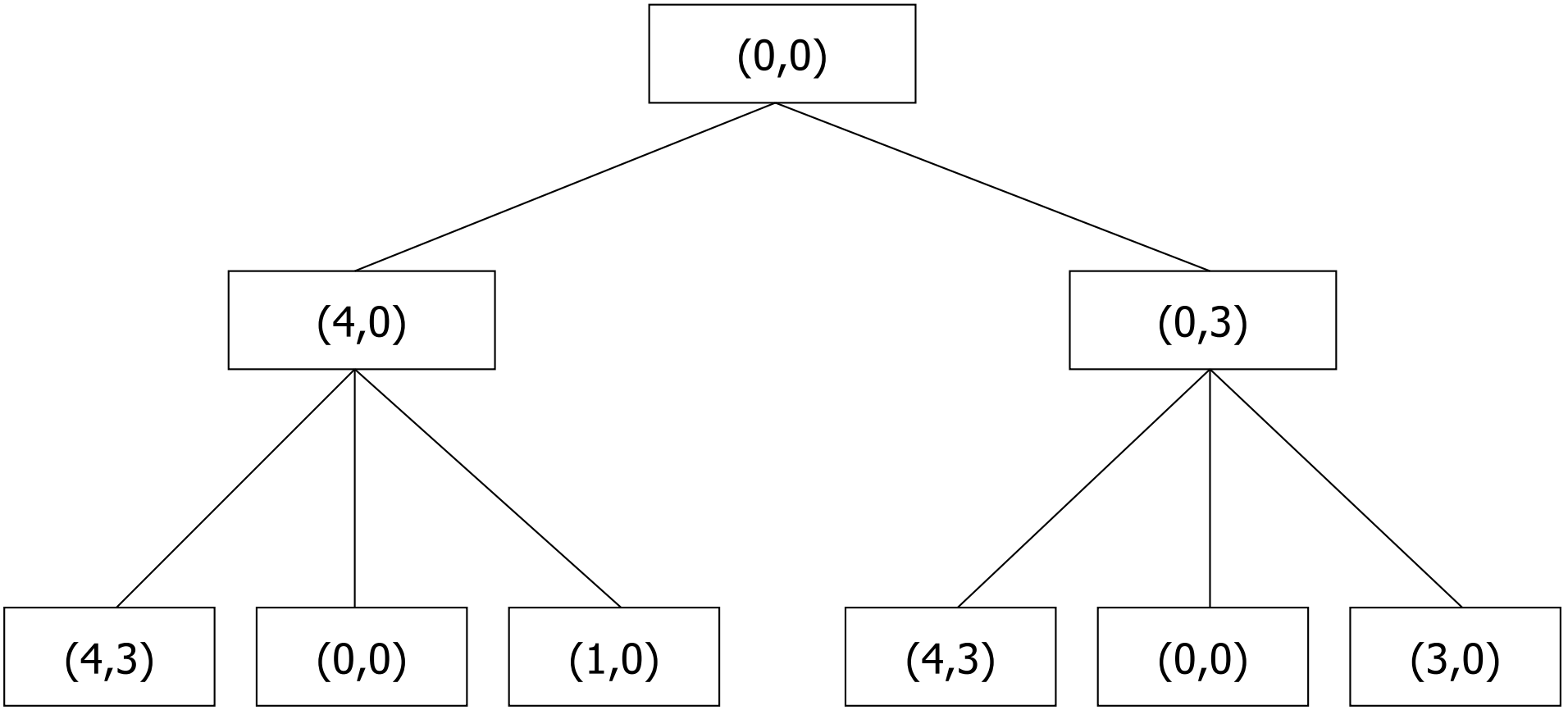
Pencarian buta (1,2), pencarian heuristic (3,4,5)

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 :

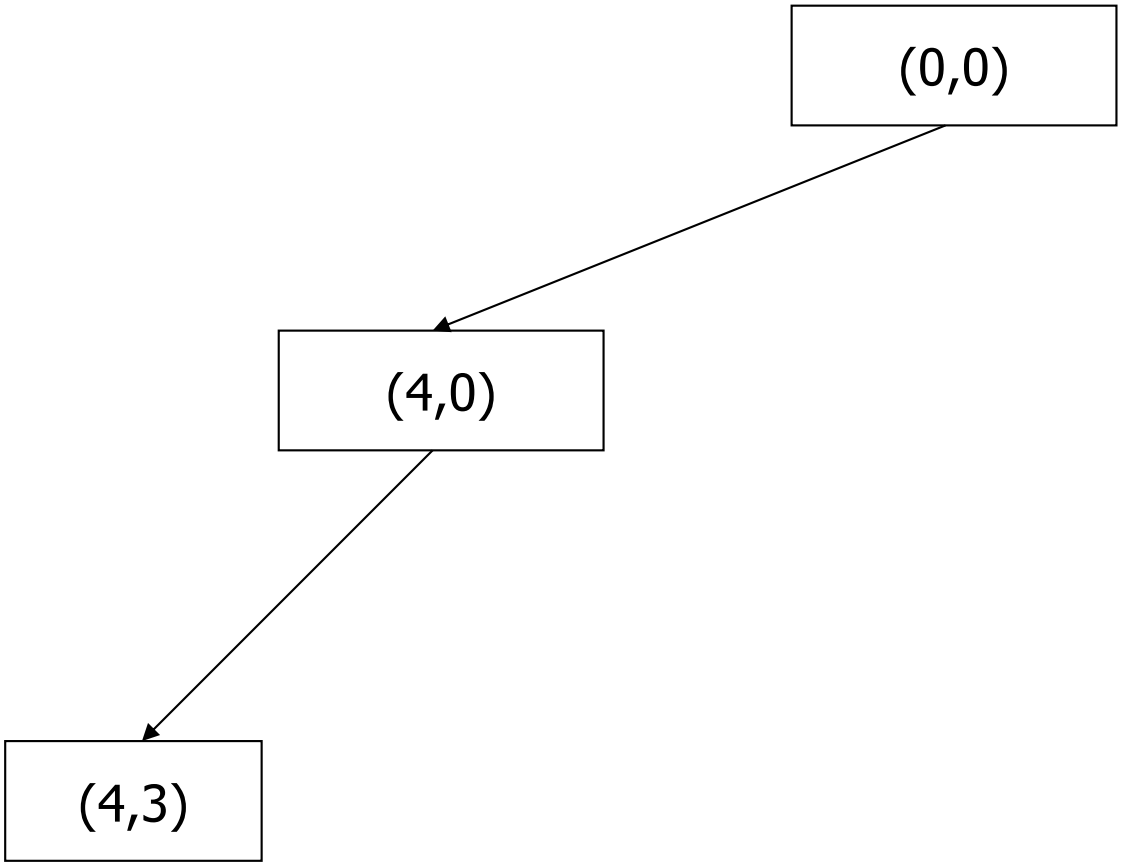
- Memerlukan memori lebih besar karena harus menyimpan semua simpul dari tree yang ditelusuri
- Harus menelusuri semua bagian tree pada level yang sama sebelum beralih ke level berikutnya.

Depth-Fisrt-Search

Algoritma Depth-Fisrt-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-Fisrt-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

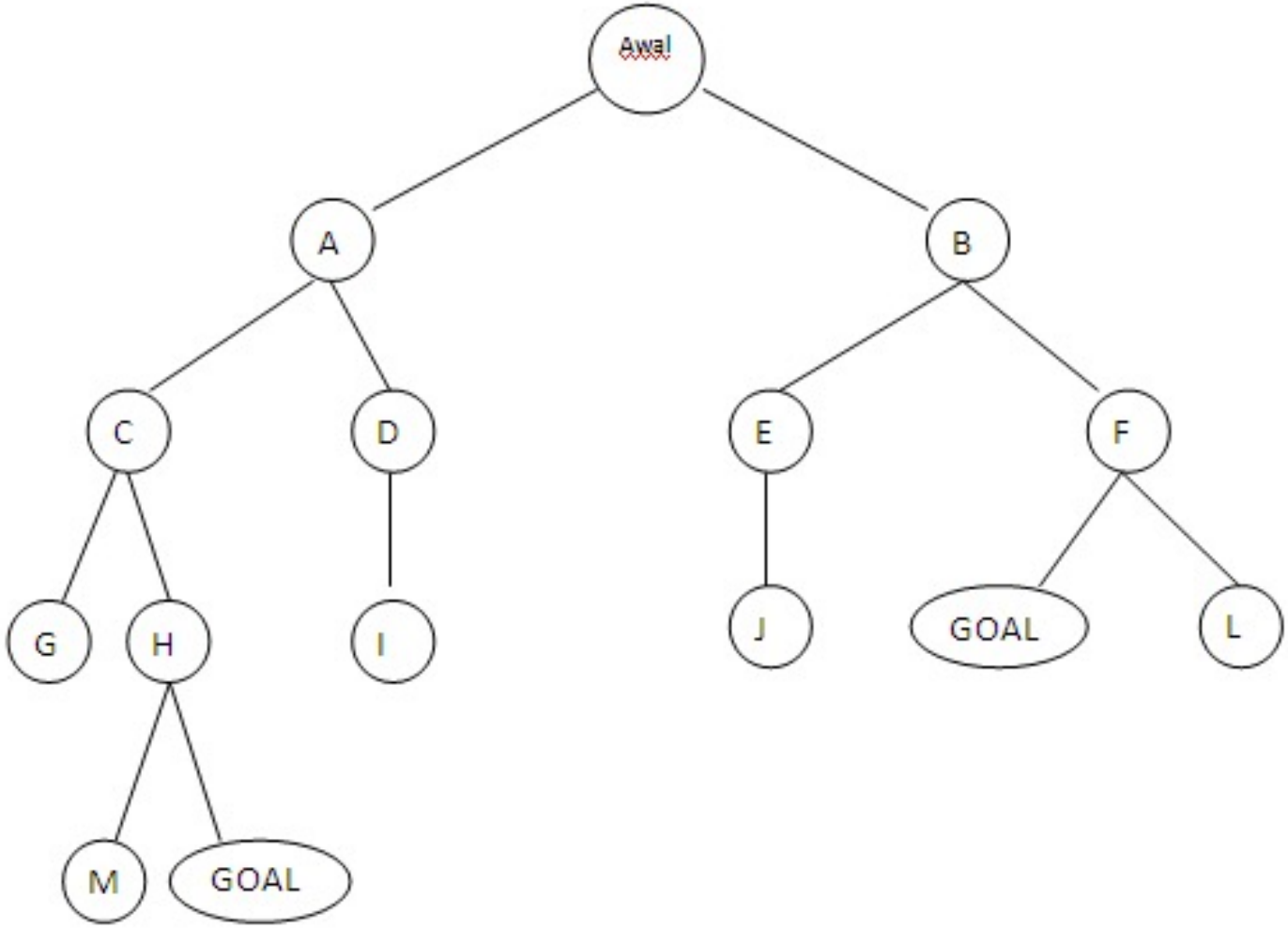
Kebaikan Depth-First-Search :

- 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]

THE END