URAIAN MATERI

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 :

a. ***Completeness*** : apakah metode tersebut **menjamin penemuan solusi jika solusinya memang ada?**
b. ***Time complexity*** : berapa lama **waktu yang diperlukan?**
c. ***Space complexity*** : berapa banyak **memori yang diperlukan**
d. ***Optimality*** : apakah metode tersebut menjamin menemukan **solusi yang terbaik jika terdapat** beberapa solusi berbeda?

Teknik Pencarian

A. *Blind search* (Pencarian buta)

Tidak ada informasi awal yang digunakan dalam proses pencarian

1. Pencarian melebar pertama **(*Breadth – First Search*)**
2. Pencarian mendalam pertama **(*Depth – First Search*)**

B. *Heuristic Seacrh* (Pencarian terbimbing)

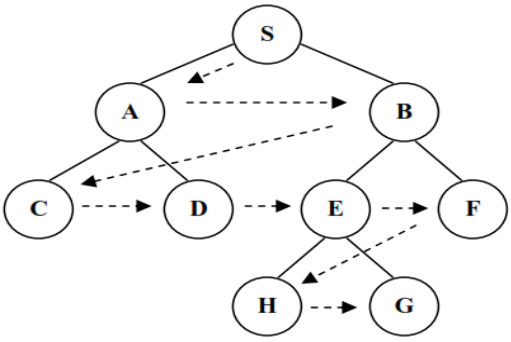
Adanya informasi awal yang digunakan dalam proses pencarian

1. Pendakian Bukit **(*Hill Climbing*)**
2. Pencarian Terbaik Pertama **(*Best First Search*)**

Pencarian Buta (Blind Search)

1. Breadth – First search

Semua node pada level n akan dikunjungi terlebih dahulu sebelum mengunjungi node-node pada level n+1. Pencarian dimulai dari node akar terus ke level 1 dari kiri ke kanan, kemudian berpindah ke level berikutnya dari kiri ke kanan hingga solusi ditemukan.



Keuntungan :

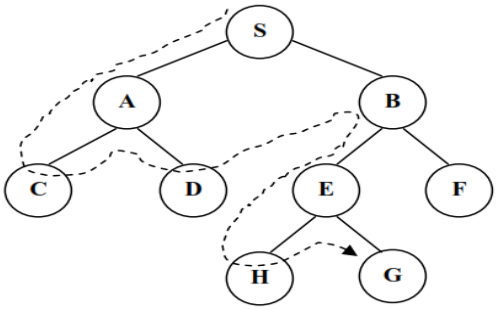
- tidak akan menemui jalan buntu, menjamin ditemukannya solusi (jika solusinya memang ada) dan solusi yang ditemukan pasti yang paling baik jika ada 1 solusi, maka breadth – first search akan menemukannya, jika ada lebih dari 1 solusi, maka solusi minimum akan ditemukan.
- Kesimpulan : **complete dan optimal**

Kelemahan :

- membutuhkan **memori yang banyak, karena harus menyimpan semua simpul yang pernah** dibangkitkan. Hal ini harus dilakukan agar BFS dapat melakukan penelusuran simpul-simpul sampai di level bawah membutuhkan **waktu yang cukup lama**

2. Depth – First search

Pencarian dilakukan pada suatu simpul dalam setiap level dari yang paling kiri. Jika pada level yang paling dalam tidak ditemukan solusi, maka pencarian dilanjutkan pada simpul sebelah kanan dan simpul yang kiri dapat dihapus dari memori. Jika pada level yang paling dalam tidak ditemukan solusi, maka pencarian dilanjutkan pada level sebelumnya. Demikian seterusnya sampai ditemukan.



Keuntungan :

- membutuhkan **memori relatif kecil, karena hanya node-node pada lintasan yang aktif saja** yang disimpan
- Secara kebetulan, akan menemukan solusi tanpa harus menguji lebih banyak lagi dalam ruang keadaan, jadi jika solusi yang dicari berada pada level yang dalam dan paling kiri, maka DFS akan menemukannya dengan cepat **waktu cepat**

Kelemahan :

- Memungkinkan tidak ditemukannya tujuan yang diharapkan, karena jika pohon yang dibangkitkan mempunyai level yang sangat dalam (tak terhingga) **tidak complete karena** tidak ada jaminan menemukan solusi
- Hanya mendapat 1 solusi pada setiap pencarian, karena jika terdapat lebih dari satu solusi yang sama tetapi berada pada level yang berbeda, maka DFS tidak menjamin untuk menemukan solusi yang paling baik **tidak optimal.**

Pencarian Heuristik

Heuristik adalah sebuah teknik yang mengembangkan efisiensi dalam proses pencarian. Fungsi heuristik digunakan untuk mengevaluasi keadaan-keadaan problema individual dan menentukan seberapa jauh hal tersebut dapat digunakan untuk mendapatkan solusi yang diinginkan. Metode ini menggunakan suatu fungsi yang menghitung estimasi biaya dari suatu simpul tertentu menuju ke simpul tujuan, disebut **fungsi heuristic.** Jenis-jenis pencarian heuristik:

a. *Generate and Test*
b. *Hill Climbing* :
    i. *Simple Hill Climbing*
    ii. *Steepest – Ascent Hill Climbing*
c. *Best First Search*
d. *Means-End Analysis, Constraint Satisfaction*, etc.

a. *Generate and Test*

Gabungan antara *Depth-First Search* dengan pelacakan mundur (*backtracking*).

Algoritma :

a. Bangkitkan suatu kemungkinan solusi (titik tertentu / lintasan tertentu / keadaan awal).

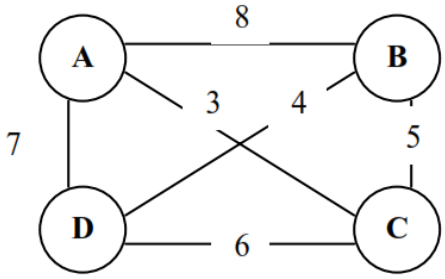
b. Uji untuk melihat apakah node tersebut merupakan solusinya dengan cara membandingkan node tersebut / node akhir dari suatu lintasan yang dipilih dengan kumpulan tujuan yang diharapkan

c. Jika solusi ditemukan, keluar. Jika tidak, ulangi kembali langkah pertama.

Contoh : "*Travelling Salesman Problem* (TSP)".

Contoh : "*Travelling Salesman Problem* (TSP)".

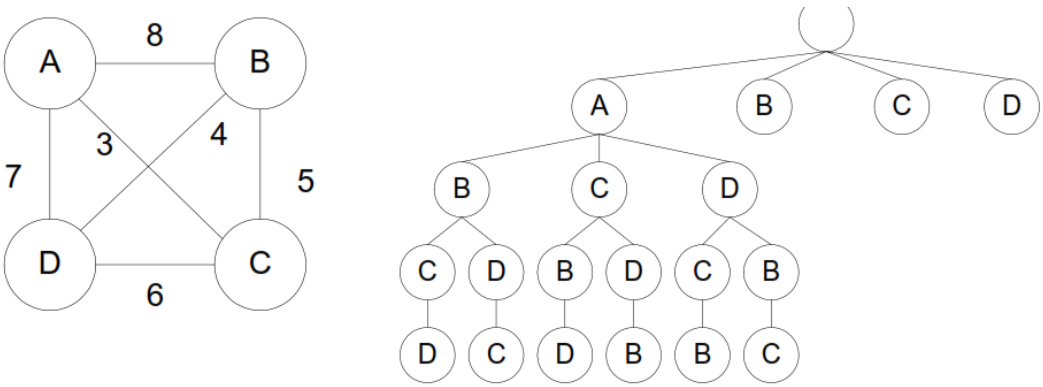
Seorang Salesman ingin mengunjungi n kota. Jarak antara tiap-tiap kota sudah diketahui. Kita ingin mengetahui rute terpendek dimana setiap kota hanya boleh dikunjungi tepat 1 kali. Misalkan ada 4 kota dengan jarak antara tiap-tiap kota seperti berikut :



Alur Pencarian

Pencarian Ke	Lintasan	Panjang lintasan	Lintasan terpilih
1	ABCD	19	ABCD
2	ABDC	18	ABDC
3	ACBD	12	ACBD
4	ACDB	13	ACBD
5	ADBC	16	ACBD
Dst ...			

Kemudian, dapat dibuat pohon keputusannya



b. *Hill Climbing*

Proses pengujian yang dilakukan dengan menggunakan fungsi heuristik. Pembangkitan keadaan berikutnya sangat tergantung pada *feedback* dari prosedur pengetesan. Tes yang berupa fungsi heuristik akan menunjukkan seberapa baik nilai terkaan yang diambil terhadap keadaan-keadaan lain yang mungkin. Metode ini mengeksplorasi keadaan secara *depth-first search* dengan mencari path yang bertujuan menurunkan *cost* untuk ke *goal path*.

Algoritma :

- Mulai dari keadaan awal (*initial state*), lakukan pengujian: jika merupakan tujuan, maka berhenti; dan jika tidak, lanjutkan dengan keadaan sekarang (*current state*) sebagai keadaan awal.
- Kerjakan langkah-langkah berikut sampai solusinya ditemukan, atau sampai tidak ada operator baru yang akan diaplikasikan pada keadaan sekarang:
  - Cari operator yang belum pernah digunakan; gunakan operator ini untuk mendapatkan keadaan yang baru.
  - Evaluasi keadaan baru tersebut.
  - Jika keadaan baru merupakan tujuan, keluar.
  - Jika bukan tujuan, namun nilainya lebih baik daripada keadaan sekarang, maka jadikan keadaan baru tersebut menjadi keadaan sekarang.
  - Jika keadaan baru tidak lebih baik daripada keadaan sekarang, maka lanjutkan iterasi.

Operator yang digunakan adalah menukar urutan posisi 2 kota dalam suatu lintasan.

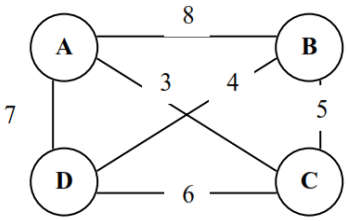
Apabila ada *n* kota, & ingin mencari kombinasi lintasan, maka akan didapat kombinasi sebanyak:

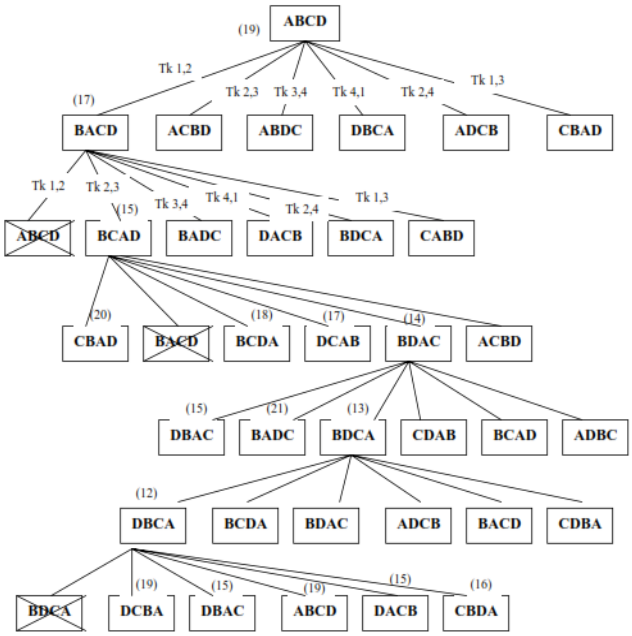
Keenam kombinasi ini akan dipakai semuanya sebagai operator, yaitu:

- Tukar 1,2 = menukar urutan posisi kota ke – 1 dengan kota ke – 2
- Tukar 2,3 = menukar urutan posisi kota ke – 2 dengan kota ke – 3
- Tukar 3,4 = menukar urutan posisi kota ke – 3 dengan kota ke – 4
- Tukar 4,1 = menukar urutan posisi kota ke – 4 dengan kota ke – 1
- Tukar 2,4 = menukar urutan posisi kota ke – 2 dengan kota ke – 4
- Tukar 1,3 = menukar urutan posisi kota ke – 1 dengan kota ke – 3

Contoh:

TSP dengan *Hill Climbing :* Ruang keadaan berisi semua kemungkinan lintasan yag mungkin.





Maka, Didapatlah lintasan terpendek DBCA =12

c. Best Fisrt Search

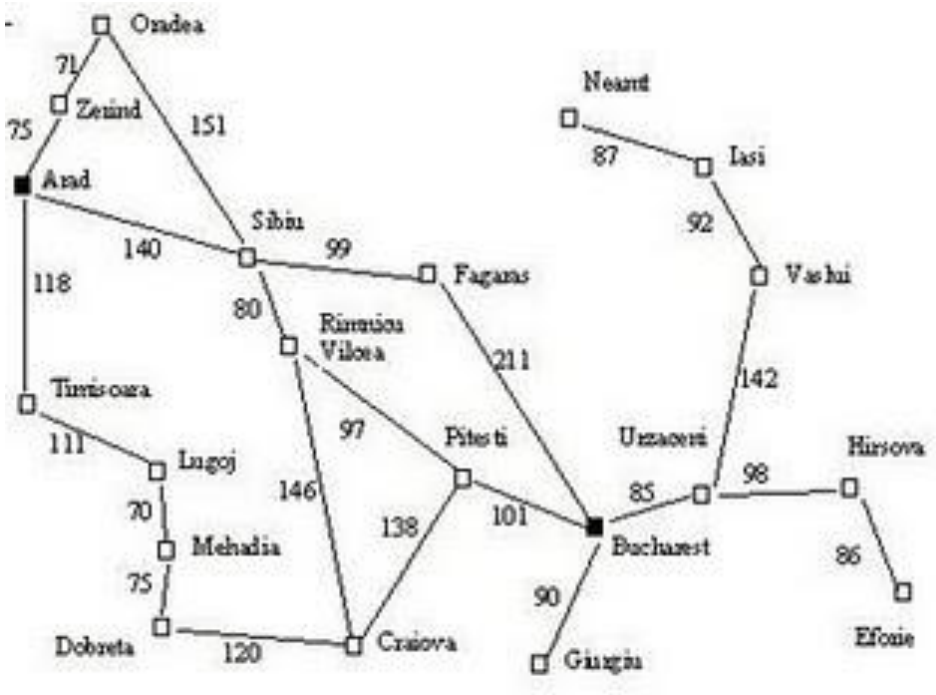
Metode ini merupakan kombinasi dari metode *depth first search* dan *breadth first search*.

Algoritma :

- Mulai dengan OPEN yang hanya berisi keadaan awal
- Ulangi langkah-langkah berikut hingga tujuan ditemukan atau antrian OPEN sudah kosong
- Ambil node terbaik dari OPEN
- Bangkitkan semua *successor*-nya
- Untuk tiap-tiap *successor*, kerjakan:
    - jika node tersebut belum pernah dibangkitkan sebelumnya, evaluasi node tersebut dan masukkan ke OPEN;
    - jika node tersebut sudah pernah dibangkitkan sebelumnya, ubah parent jika lintasan baru lebih baik daripada sebelumnya.

Contoh:

Berikut adalah peta Romania dengan jarak jalan-jalan yang menghubungkan kota-kota dalam km. Permasalahannya adalah untuk mencari jalan terdekat dari Arad menuju kota Bucharest.



Adapun jarak kota-kota terhadap kota Bucharest jika ditarik garis lurus adalah sebagai berikut :

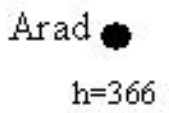
Arad 366	Hirsova 15	Timisoara 329
Bucharest 0	Iasi 226	Urziceni 80
Craiova 160	Lugoj 244	Vaslui 199
Dobreta 242	Mehadia 241	Zerind 374
Eforie 161	Neamt 234	Rimnicu Vilcea 193
Fagaras 178	Oradea 380	Sibiu 253
Giurgiu 77	Pitesti 98	

Permasalahannya adalah untuk mencari jalan terdekat dari kota Arad menuju kota Bucharest dengan menggunakan metoda *Best First Search*

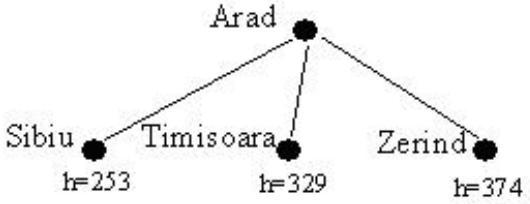
Heuristik yang digunakan adalah jarak kota-kota terhadap Bucharest jika ditarik garis lurus dengan asumsi kota terhubung yang letaknya paling dekat dengan Bucharest adalah jalan yang paling optimal.

Langkah-langkah penelusurannya adalah

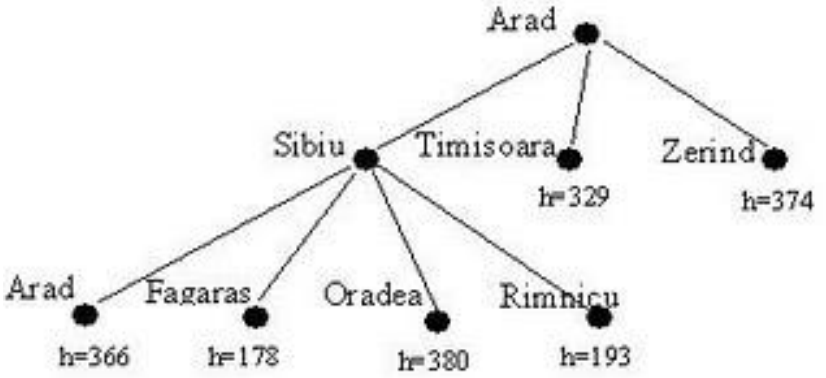
Langkah 1



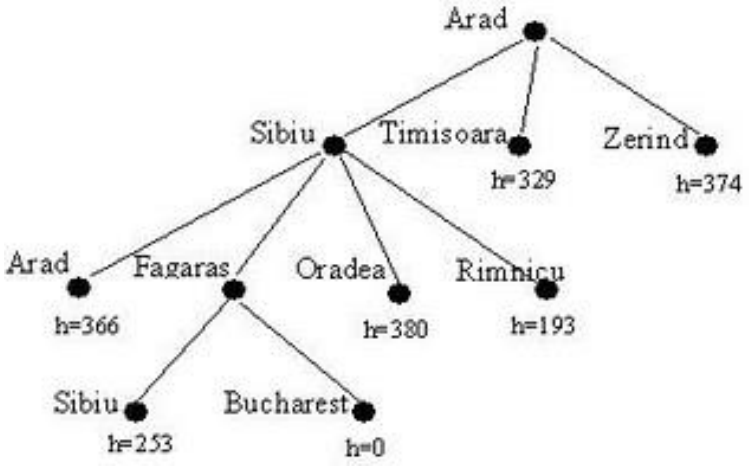
langkah 2



Langkah 3



Langkah 4



Dari langkah2 diatas, maka didapatkan path solusi dari Arad ke Bucharest adalah:

Arad – Sibiu – Fagarast – Bucharest.

140 + 99 + 221 = 450 km.