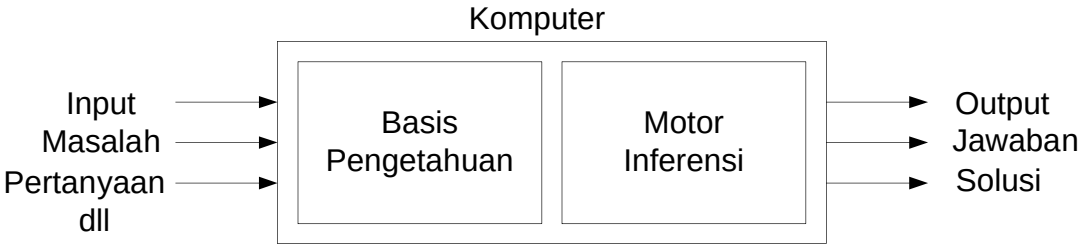
Masalah, Ruang Masalah dan Pencarian

Review : Sistem yang menggunakan AI



Untuk **membangun sistem** yang mampu menyelesaikan masalah **menggunakan AI** :

1. Mendefinisikan masalah dengan tepat, mencakup spesifikasi yang tepat mengenai keadaan awal dan solusi yang diharapkan.
2. Menganalisis masalah tersebut dan mencari beberapa teknik penyelesaian masalah yang sesuai.
3. Merepresentasikan pengetahuan yang perlu untuk menyelesaikan masalah tersebut.
4. Memilih teknik penyelesaian masalah yang terbaik.

Untuk **Mendefinisikan Suatu Masalah**:

- Definisikan/buat '*state space*' atau ruang masalah
- Tentukan keadaan awal (*initial state*)
- Tentukan keadaan akhir/tujuan (*goal state*)
- Tentukan operatornya/aturannya

Contoh 1 : "**Permainan Catur**"
Yang harus ditentukan adalah :

1. Posisi awal pada papan catur
2. Aturan-aturan untuk melakukan gerakan secara legal
3. Tujuan (*goal*) yang ingin dicapai adalah posisi pada papan catur yang menunjukkan kemenangan seseorang terhadap lawannya.

Contoh2 : **A water jug problem**

Initial state:

Diketahui dua buah ember masing-masing berkapasitas 3 gallon dan 4 gallon, dan sebuah pompa air.

Goal state:

Isi ember yang berkapasitas 4 gallon dengan 2 gollon air!

Solusi: Buat asumsi dengan:

X : ember berkapasitas 4 gallon

Y : ember berkapasitas 3 gallon

Production Rules :

Sistem Produksi/*Production System* terdiri dari:

- Sekumpulan Aturan (*a set of rules*)
- Knowledge Base /Data Base
- Sebuah strategi pengontrol (*Control Strategy*)
- Urutan yang dipakai (*a rule applier*)

Untuk kasus di atas, production rules-nya :

1. (X,Y), if (X < 4) → (4,Y)
2. (X,Y), if (Y < 3) → (X,3)
3. (X,Y), if X > 0 → (X-d,Y)
4. (X,Y), if (Y > 0) → (X,Y-d)
5. (X,Y), if X > 0 → (0,Y)
6. (X,Y), if Y > 0 → (X,0)
7. (X,Y) if X+Y >= 4 and Y > 0 → (4, Y-(4-X))
8. (X,Y) if X+Y >= 3 and X > 0 → (X-(3-Y),3)
9. (X,Y) if (X+Y) <= 4 and Y > 0 → (X+Y,0)
10. (X,Y) if X+Y <= 3 and X >0 → (0,X+Y)
11. (0,2) → (2,0)
12. (X,2) → (0,2)

Salah satu solusinya:

X	Y	*Rules* yang digunakan
0	0	
		2
0	3	
		9
3	0	
		2
3	3	
		7
4	2	
		5 atau 12
0	2	
		9 atau 11
2	0	
		solusi

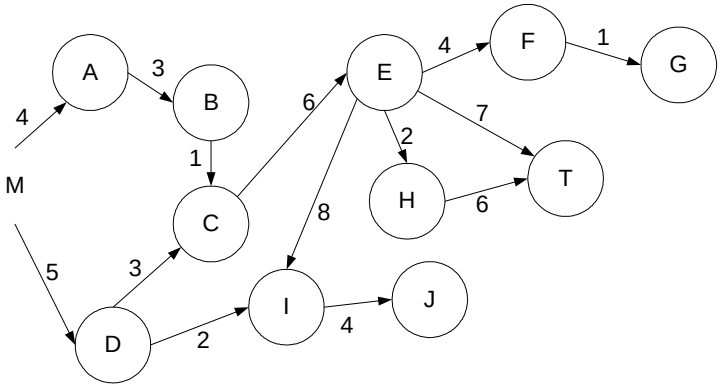
Contoh 3 : Masalah "**Petani, Kambing, Serigala dan Sayuran"**

Seorang petani akan menyebrangkan seekor kambing, seekor serigala dan sayuran dengan sebuah boat yang melalui sungai. Boat hanya bias memuat petani dan satu penumpang lain (kambing, serigala atau sayuran). Jika ditinggalkan oleh petani tersebut, maka sayuran akan dimakan oleh kambing dan kambing akan dimakan oleh serigala. Bagaimana caranya agar petani, kambing, serigala dan sayuran dapat selamat sampai di seberang sungai ?

Beberapa cara **Merepresentasikan Ruang Masalah** :

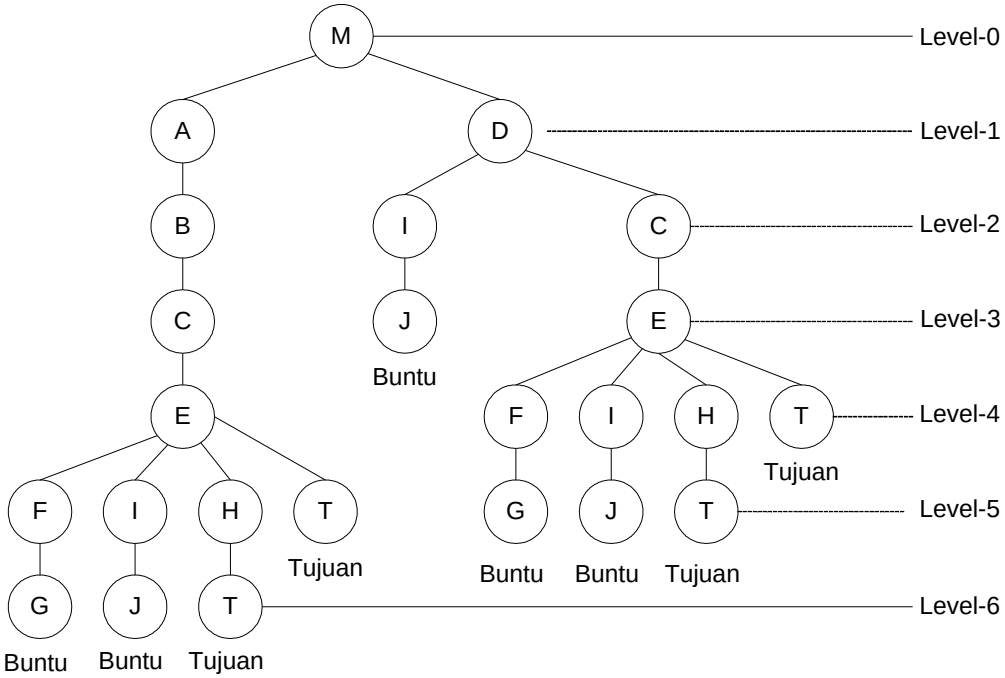
1. Graph Keadaan

   Contoh :

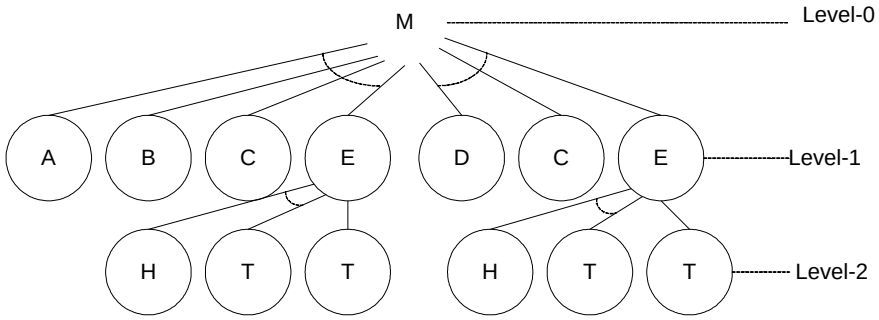


Graph berarah dengan satu tujuan (T)	Graph berarah yang menemui jalan buntu	Graph dengan siklus
• M-A-B-C-E-T • M-A-B-C-E-H-T • M-D-C-E-T • M-D-C-E-H-T	• M-A-B-C-E-F-G • M-A-B-C-E-I-J • M-D-C-E-F-G • M-D-C-E-I-J • M-D-I-J	D-E-C-E-I-D

2. Pohon Pelacakan



3. Pohon AND/OR



KARAKTERISTIK MASALAH/PROBLEMA

Untuk memilih metode yang paling baik untuk memecahkan suatu masalah tertentu, diperlukan suatu analisa masalah. Dalam menganalisa suatu masalah kita perlu mengetahui beberapa karakte –ristik masalah, diantaranya adalah:

1. Apakah masalah dapat dipilah-pilah (*decompose- able*) menjadi sejumlah sub-masalah independent yang lebih kecil atau lebih mudah ?
2. Dapatkah langkah-langkah penyelesaian yang terbukti tidak tepat diabaikan ?
3. Apakah ruang lingkup atau semesta pembicaraan masalah dapat diprakirakan ?
4. Apakah solusi masalah yang baik telah dibanding- kan dengan semua solusi yang dimungkinkan ?
5. Apakah basis pengetahuan yang digunakan untuk memecahkan masalah bersifat konsisten ?
6. Apakah benar-benar dibutuhkan sejumlah besar informasi untuk memecahkan masalah yang sedang dihadapi, atau pengetahuan hanya penting untuk membatasi proses pencarian (*searching*) ?
7. Apakah sebuah komputer sendirian dapat diberi masalah dan kemudian menyajikan solusi secara sederhana, atau akankah solusi dari suatu masalah membutuhkan interaksi antara komputer dan manusia ?

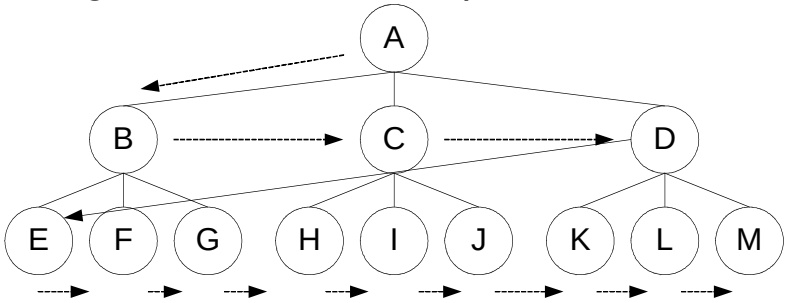
TEKNIKPENCARIAN/PENELUSURAN (*SEARCHING*)

- Pada umumnya manusia mempertimbangkan sejumlah alternatif strategi dalam menyelesaikan suatu problema.
- Dalam permainan catur misalnya, seorang pemain mempertimbangkan sejumlah kemungkinan tentang langkah-langkah berikutnya, memilih yang terbaik menurut kriteria tertentu seperti kemungkinan respon lawannya.
- Aspek tingkahlaku cerdas yang mendasari teknik penyelesaian problema seperti dalam permainan catur tersebut dinamakan proses pencarian ruang keadaan (*space state search*).
- *Exhaustive search* – adalah proses pencarian terhadap seluruh ruang keadaan serangakaian langkah yang paling dimungkinkan untuk menghasilkan kemenangan.
- Walaupun metode ini dapat diterapkan pada setiap ruang keadaan, namum ukuran ruang keadaan yang sangat besar membuat pendekatan ini secara praktis tidak dimungkinkan (dalam permainan catur terdapat 10^{120} keadaan)
- Bila kasus ini diimplementasikan ke dalam sisten komputer, maka akan membutuhkan memori yang sangat besar, dan waktu pencarian yang sangat lama. Dengan kata lain metode *exhaustive search* ini tidak efisien dan tidak efektif, sehingga tidak praktis untuk diimplementasikan.
- Untuk mengatasi kendala tersebut di atas, ada beberapa cara yang dapat dilakukan, diantaranya: pertama teknik pencarian parsial (Blind Search) dan yang kedua teknik pencarian heuristic (Heuristik Search).

Pencarian Parsial (Blind Search)

A. PENCARIAN MELEBAR PERTAMA (*Breadth-First Search*)

- Pada metode *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 ke-1 dari kiri ke kanan, kemudian berpindah ke level berikutnya, demikian pula dari kiri ke kanan hingga ditemukannya solusi (lihat gambar di bawah ini).

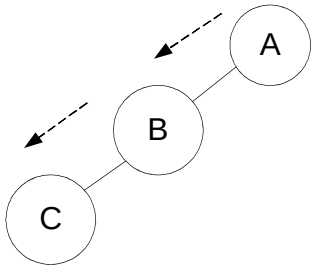


Prosedur breadth_first_search
Inisialisasi : open = [start]; closed [ ]
While open = [ ] do
Begin
Hapuskan keadaan paling kiri dari keadaan open, sebutlah keadaan itu dengan X;
Jika X merupakan tujuan then return (sukses);
Buatlah semua child dari X;
Ambillah X dan masukkan pada closed;
Eliminasilah setiap child X yang telah berada pada open atau closed, yang akan menyebabkan loop dalam search;
Ambillah turunan di ujung kanan open sesuai urutan penemuan-nya;
End;

- Keuntungan :
  - Tidak akan menemui jalan buntu
  - Jika ada satu solusi, maka *breadth-first search* akan menemukannya. Dan, jika ada lebih dari satu solusi, maka solusi minimum akan ditemukan.
- Kelemahan :
  - Membutuhkan memori yang cukup banyak, karena menyimpan semua node dalam satu pohon
  - Membutuhkan waktu yang cukup lama, karena akan menguji n level untuk mendapatkan solusi pada level yang ke-(n+1).

B. PENCARIAN KEDALAM PERTAMA (*Depth-First Search*)

- Pada *Depth-First Search*, proses pencarian akan dilakukanpada semua anaknya sebelum dilakukan pencarian ke node-node yang selevel. Pencarian dimulai dari node akar ke level yang lebih tinggi. Proses ini diulangi terus hingga ditemukannya solusi (lihat gambar di bawah).



sprosedur depth_first_search
inisialisasi: open = [Start]; closed = []
while open x [] do
begin
    hapuskan keadaan berikutnya dari sebelah kiri open, sebutlah keadaan itu dengan X;
    jika X merupakan tujuan then return(sukses);

buatlah semua child yang dimungkinkan dari X;
ambilah X dan masukkan pada closed;
eliminasilah setiap child X yang telah berada pada open atau closed, yang akan menyebabkan loop dalam search;
ambilah child X yang tersisa di ujung kanan open sesuai urutan penemuannya;

end.

- Keuntungan :
  - Membutuhkan memori yang relative kecil, karena hanya node-node pada lintasan yang aktif saja yang disimpan.
  - Secara kebetulan, metode *depth-first search* akan menemukan solusi tanpa harus menguji lebih banyak lagi dalam ruang keadaan.
- Kelemahan :
  - Memungkinkan tidak ditemukannya tujuan yang diharapakan
  - Hanya akan menemukan 1 solusi pada setiap pencarian

Pencarian Heuristik (Heuristic Search)

- Heuristik adalah sebuah teknik yang mengembangkan efisiensi dalam proses pencarian, namum dengan kemungkinan mengorbankan kelengkapan (*completeness*).

- Fungsi heuristik digunakan untuk mengevaluasi keadaan-keadaan problema individual dan menentukan seberapa jauh hal tersebut dapat digunakan untuk mendapatkan solusi yang diinginkan.

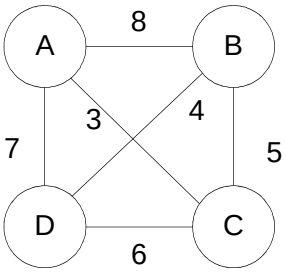
- Jenis-jenis *Heuristic Searching*:
  A. Generate and Test.
  B. Hill Climbing.
  C. Best First Search.
  D. Alpha Beta Prunning,Means-End-Anlysis,Constraint Satisfaction, Simulated Anealing, dll

A. PEMBANGKITAN dan PENGUJIAN (*Generate and Test*)

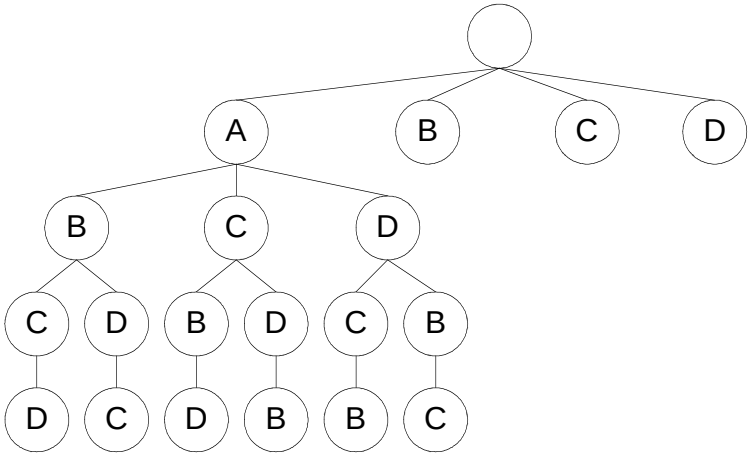
- Metode ini merupakan penggabungan antara *depth-first search* dengan pelacakan mundur (*backtracking*), yaitu bergerak ke belakang menuju pada suatu keadaan awal.

- Algoritma :
  1. Bangkitkan suatu kemungkinan solusi (membangkitkan suatu tititk tertentu atau lintasan tertentu dari keadaan awal).
  2. Uji untuk melihat apakah node tersebut benar-benar merupakan solusinya dengan cara membandingkan node terebut atau node akhir dari suatu lintasan yang dipilih dengan kumpulan tujuan yang diharapkan.
  3. Jika solusi ditemukan, keluar. Jika tidak, ulangi kembali langkah pertama.

Contoh : **“Travelling Salesman Problem (TSP)”**
Seorang salesman ingin mengunjungi n kota. Jarak antara tiap-tiap kota sudah diketahui. Kita ingin mengetahui ruter terpendek dimana setaip kota hanya boleh dikkunjungi tepat 1 kali. Misalkan ada 4 kota dengan jarak antara tiap-tiap kota seperti gambar di bawah ini :



Penyelesaian dengan metode *Generate and Test*



Alur pencarian dengan *Generate and Test*

Pencarian ke-	Lintasan	Panjang Lintasan	Lintasan Terpilih	Panjang Lintasan Terpilih
1	ABCD	19	ABCD	19
2	ABDC	18	ABDC	18
3	ACBD	12	ACBD	12
4	ACDB	13	ACBD	12
5	ADBC	16	ACBD	12
Dst…				

B. PENDAKIAN BUKIT (*Hill Climbing*)

- Metode ini hampir sama dengan metode pembangkitan dan pengujian, hanya saja proses pengujian dilakukan dengan menggunakan fungsi heuristic. Pembangkitan keadaan berikutnya tergantung pada feedback dari prosedur pengetesan. Tes yang berupa fungsi heuristic ini akan menunjukkan seberapa baiknya nilai terkaan yang diambil terhadap keadaan-keadaan lainnyayang mungkin.

- Algoritma *Simple Hill Climbing*
  1. Cari operator yang belum pernah digunakan; gunakan operator ini untuk mendapatkan keadaan yang baru.

  a) Kerjakan langkah-langkah berikut sampai solusinya ditemukan atau sampai tidak ada operator baru yang akan diaplikasikan pada keadaan sekarang : Cari operator yang belum digunakan; gunakan operator ini untuk mendapatkan keadaan yang baru.

  b) Evaluasi keadaan baru tersebut :
     (i) Jika keadaan baru merupakan tujuan, keluar
     (ii) Jika bukan tujuan, namun nilainya lebih baik daripada keadaan sekarang, maka jadikan keadaan baru tersebut menjadi keadaan sekarang.
     (iii) Jika keadaan baru tidak lebih baik daripada keadaan sekarang, maka lanjutkan iterasi.

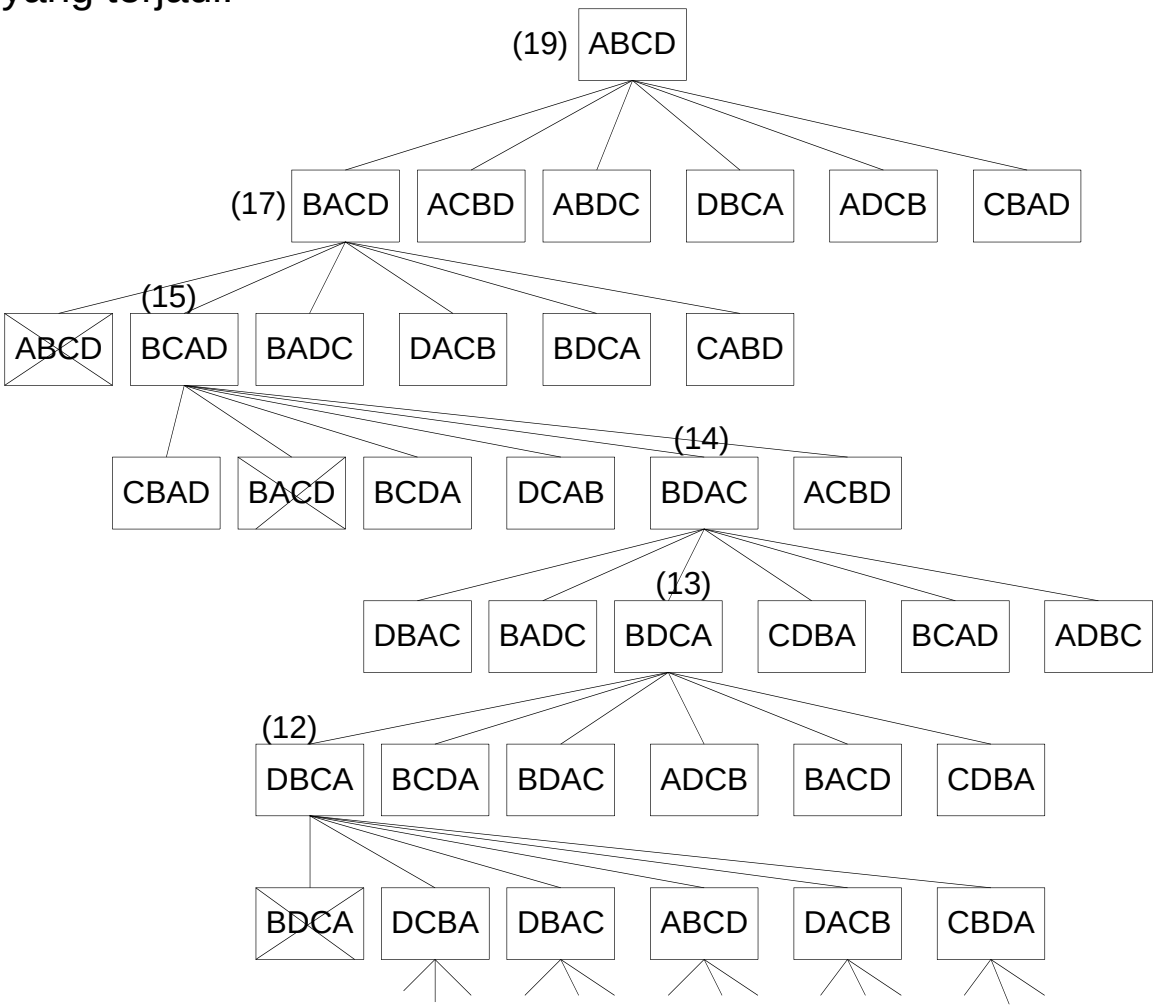
  Pada *simple hill climbing*, ada 3 masalah yang mungkin:
  - Algoritma akan berhenti kalau mencapai nilai optimum local
  - Urutan penggunaan operator akan sangat berpengaruh pada penemuan solusi
  - Tidak diijinkan untuk melihat satupun langkah sebelumnya.

Contoh : TSP dengan *Simple Hill Climbing*

Disini ruang keadaan berisi semua kemungkinan llintasan yang mungkin. Operator digunakan untuk menukar posisi kota-kota yang bersebelahan. Apabila ada n kota, dan kita ingin mencari kombinasi lintasan dengan menukar posisi urutan 2 kota, maka kita akan mendapatkan sebanyak :

$\frac{n!}{2!(n-2)!}$ atau sebanyak 6 kombinasi (lihat gambar di bawah).

Fungsi heuristic yang digunakan adalah panjang lintasan yang terjadi.



C. PENCARIAN TERBAIK PERTAMA (*Best-First Search*)

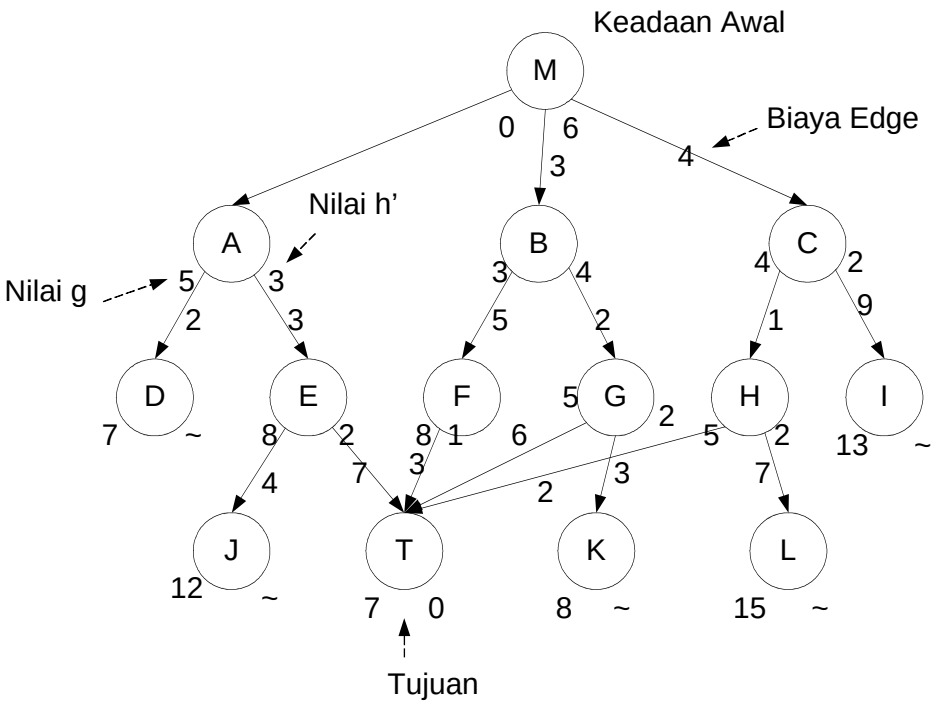
- Metode ini merupakan kombinasi dari metode *depth-first search* dan *breadth-first search*. Pada metode *best-first search*, pencarian diperbolehkan mengunjungi node yang ada di level yang lebih rendah, jika ternyata node pada level yang lebih tinggi ternyata memiliki nilai heuristic yang lebih buruk.

- Fungsi Heuristik yang digunakan merupakan prakiraan (estimasi) *cost* dari *initial state* ke *goal state*, yang dinyatakan dengan :

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

  dimana f' = Fungsi evaluasi
  g = *cost* dari *initial state* ke *current state*
  h' = prakiraan *cost* dari *current state* ke *goal state*

Contoh :
Misalkan kita memiliki ruang pencarian seperti pada gambar di bawah. Node M merupakan keadaan awal dan node T merupakan tujuannya. Biaya edge yang menghubungkan node M dengannode A adalah biaya yang dikeluarkan untuk bergerak dari kota M ke kota A. Nilai g diperoleh berdasarkan biaya edge minimal. Sedangkan nilai h' di node A merupakan hasil perkiraan terhadap biaya yang diperlukan dari node A untuk sampai ke tujuan. h'(n) bernilai ~ jika sudah jelas tidak ada hubungan antara node n dengan node tujuan (jalan buntu). Kita bisa merunut nilai untuk setiap node.



Tabel status tiap node

Node (n)	g(n)	h'(n)	f'(n)
M	0	6	6
A	5	3	8
B	3	4	7
C	4	2	6
D	7	~	~
E	8	2	10
F	8	1	9
G	5	2	7
H	5	2	7
I	13	~	~
J	12	~	~
K	8	~	~
L	15	~	~
T	7	0	7

Solusi : M-C H-T