

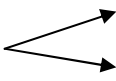
BAB IV SEARCH

Teknik-Teknik Search

Sebelum membahas tentang teknik-teknik pencarian, maka perlu dibicarakan tentang hal-hal penting yang muncul:

- Arah search
- Topologi proses search tersebut
- Memilih aturan-aturan yang dapat diterapkan
- Penggunaan fungsi-fungsi heuristik untuk memandu proses search tersebut.

Arah search

Dapat dilakukan 
Maju, bermula dari keadaan awal (*start state*)
Mundur, diawali dari keadaan tujuan (*goal state*)

TOPOLOGI PROSES SEARCH:

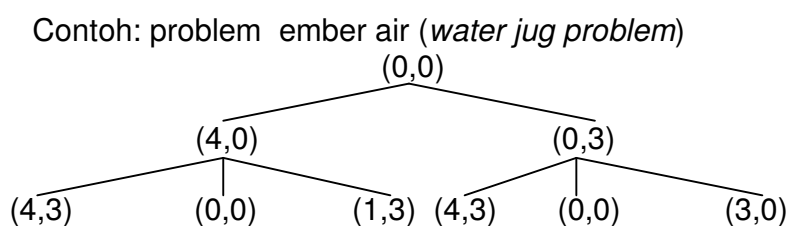
Ada dua macam penggambaran problem, yaitu dalam bentuk:

Pohon (tree)

Graf (graph)

POHON (TREE)

Merupakan graf dimana dua simbol memiliki paling banyak satu lintasan yang menghubungkannya. Tidak dimungkinkan adanya loop pada pohon.



GRAF (GRAPH)

Graf dibedakan antara:

- Graf berarah dan
- Graf tidak berarah

Graf disebut berarah bila lintasannya mempunyai arah, umumnya digambarkan dengan anak panah. Untuk graf berakar mempunyai simpul unik yang disebut akar sedemikian rupa sehingga terdapat lintasan dari akar tersebut ke semua simpul pada graf.

Ada beberapa cara untuk mencari kemungkinan penyelesaian, yaitu:

1. Depth-First Search
2. Breadth-First Search
3. Hill-Climbing Search
4. Least-Cost Search
5. Best-First Search

Evaluasi sebuah pencarian (*search*)

Evaluasi penampilan sebuah teknik pencarian (*search*) akan sangat kompleks.

Dasar pengukuran dari evaluasi:

- a. seberapa cepat search menemukan penyelesaian
- b. seberapa cepat search menemukan penyelesaian yang baik

Kecepatan search ditentukan:

- panjang lintasan
- jumlah sesungguhnya penelusuran node

Kasus:	Jakarta	- Solo	: 1000 KM
	Solo	- Yogyakarta	: 1000 KM
	Jakarta	- Purwokerto	: 800 KM
	Jakarta	- Yogyakarta	: 1900 KM
	Purwokerto	- Semarang	: 1500 KM
	Purwokerto	- Surabaya	: 1800 KM
	Purwokerto	- Solo	: 500 KM
	Yogyakarta	- Bandung	: 1000 KM
	Yogyakarta	- Kediri	: 1500 KM
	Kediri	- Surabaya	: 1500 KM
	Yogyakarta	- Surabaya	: 1000 KM

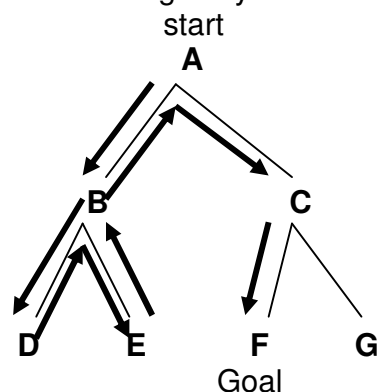
Graph:



4.1. Teknik Pencarian *Depth-First*

Penelusuran(pencarian) *Depth First* berarti setiap kemungkinan path ke goal digali(eksplorasi) ke kesimpulannya sebelum path lainnya dicoba.

Contoh: di mana F adalah goalnya.



Pencarian Depth-First akan menelusuri Graph di atas dengan susunan: ABD BEBACF

Pencarian Depth-First adalah kemungkinan metoda terbaik yang dapat diikuti di mana Heuristic tidak digunakan. Turbo Prolog menggunakan pencarian Depth-First.

Pada Graph Kasus: misalnya kita ingin pergi dari Jakarta ke Surabaya:

Percobaan 1: Jarak 3000

Jakarta - Solo - Yogyakarta - Surabaya

Percobaan 2: Jarak 5000

Jakarta - Solo - Yogyakarta - Kediri - Surabaya

Percobaan 3: Jarak 2600

Jakarta - Purwokerto - Surabaya

Evaluasi pencarian Depth First:

Percobaan 1: Penyelesaian yang baik karena tidak ada backtracking

Percobaan 2: Penyelesaian yang jelek

Percobaan 3: Penyelesaian Optimal

Contoh 8 puzzle:

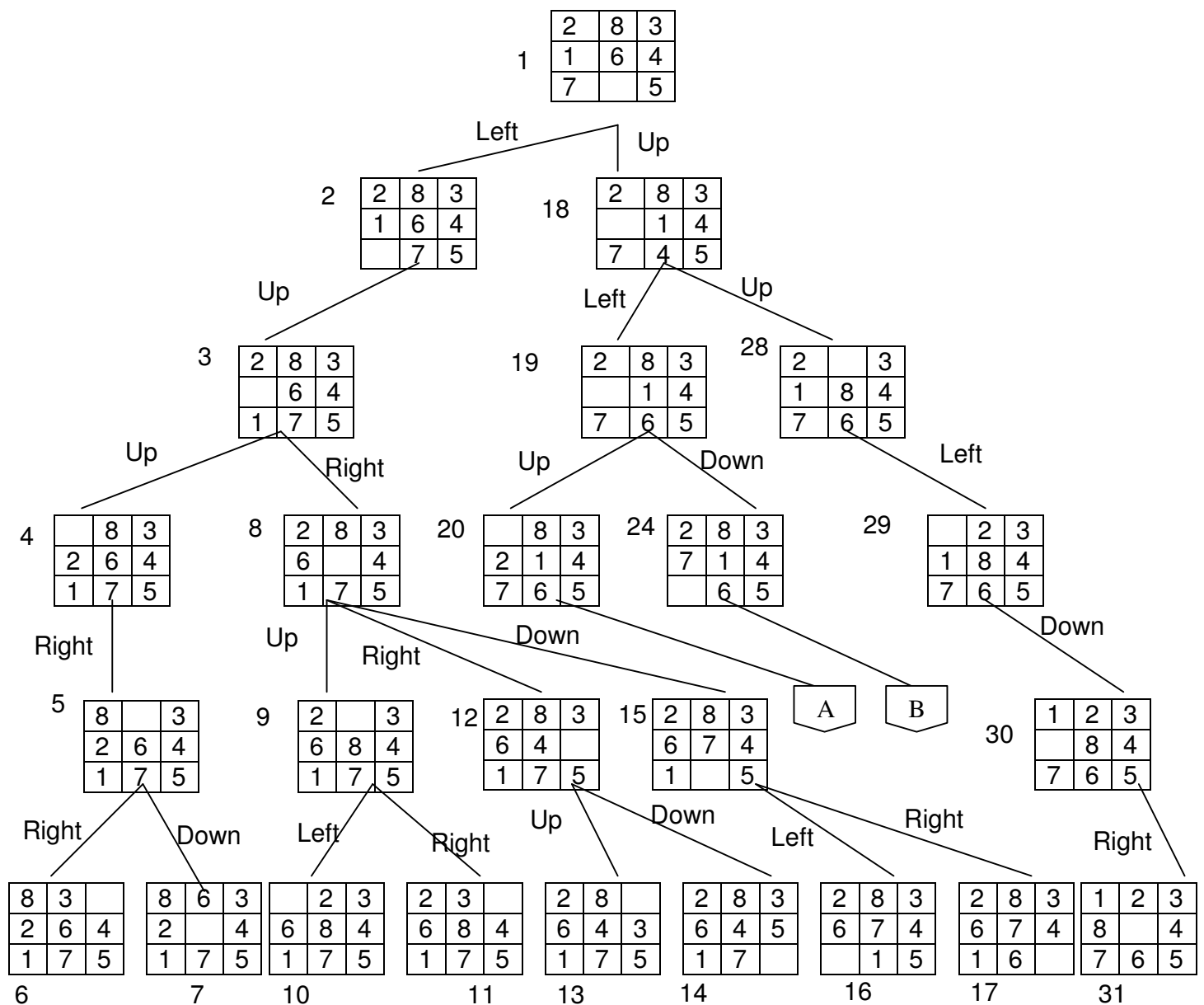
Start state:

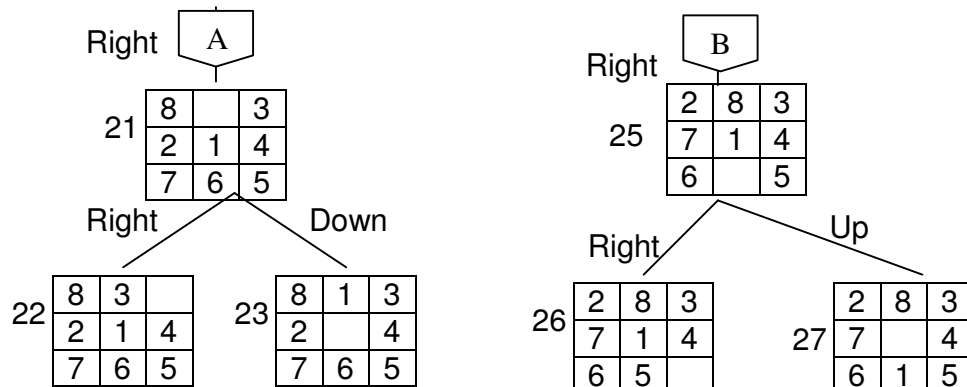
2	8	3
1	6	4
7		5

Goal State:

1	2	3
8		4
7	6	5

MENGGUNAKAN TEKNIK PENCARIAN DEPTH-FIRST:





PROSEDURE DEPTH-FIRST SEARCH ¹⁶⁾

Begin

Open := [Start]; {inisialisasi}

Closes := [];

While open # [] do

Begin

Remove leftmost state from, call it X;

If X is a goal then return (succes)

Else begin

Generate children of X;

Put X in closed;

Eliminate children of X on open or closed;

Put remaining children on left and of open

End

End;

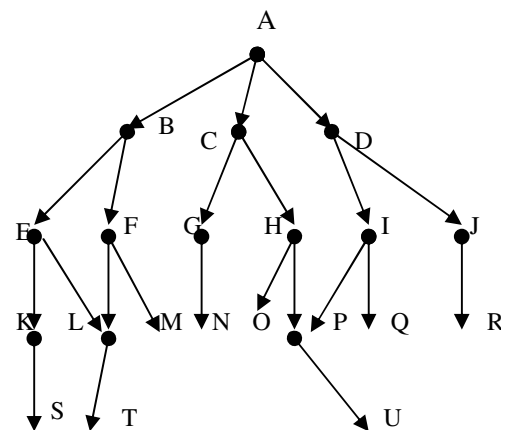
Return (failure)

End.

Trace of Depth-First Search,

Asumsi goal statenya adalah U:

1. OPEN = [A]; CLOSED = []
 2. OPEN = [B,C,D]; CLOSED = [A]
 3. OPEN = [E,F,C,D]; CLOSED = [B,A]
 4. OPEN = [K,L,F,C,D]; CLOSED = [E,B,A]
 5. OPEN = [S,L,F,C,D]; CLOSED = [K,E,B,A]
 6. OPEN = [L,F,C,D]; CLOSED = [S,K,E,B,A]
 7. OPEN = [T,F,C,D]; CLOSED = [L,S,K,E,B,A]
 8. OPEN = [F,C,D]; CLOSED = [T,L,S,K,E,B,A]
 9. OPEN = [M,C,D]; CLOSED = [F,T,L,S,K,E,B,A]
 10. OPEN = [C,D]; CLOSED = [M,F,T,L,S,K,E,B,A]
 11. OPEN = [G,H,D]; CLOSED = [C,M,F,T,L,S,K,E,B,A]
 12. OPEN = [N,H,D]; CLOSED = [G,C,M,F,T,L,S,K,E,B,A]
 13. OPEN = [H,D]; CLOSED = [N,G,C,M,F,T,L,S,K,E,B,A]
 14. OPEN = [O,P,D]; CLOSED = [H,N,G,C,M,F,T,L,S,K,E,B,A]
 15. OPEN = [P,O]; CLOSED = [O,H,N,G,C,M,F,T,L,S,K,E,B,A]
 16. OPEN = [U, D]; CLOSED=[P,O,H,N,G,C,M,F,T,L,S,K,E,B,A]
- Success

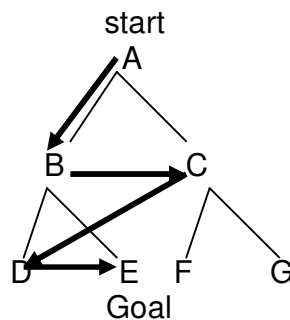


¹⁶⁾ Luger, George F. & Stubblefield, William A., "Artificial Intelligence: Structured & Strategies for Complex Problem Solving", 2nd Edition, page 96-98, Benjamin/Cumming Publishing Company, California, 1993

4.2. Teknik Pencarian Breadth-First

Merupakan kebalikan dari teknik pencarian Depth-First. Pada metoda ini diperiksa setiap node pada level yang sama sebelum mengolah ke level berikut yang lebih dalam.

Contoh: E adalah : Goalnya



node yang dilewati adalah: ABCDE

Pada Graph Kasus:

Percobaan 1: Jarak 2600

Jakarta – Purwokerto - Surabaya

Percobaan 2: Jarak 2900

Jakarta – Yogyakarta - Surabaya

Percobaan 3: jarak 3000

Jakarta - Solo – Yogyakarta – Surabaya

Penambahan Heuristic

Pada metoda depth-First dan Breadth-First: Pencarian terletak pada pemindahan dari satu goal ke goal yang lain tanpa menggunakan tebakan yang terarah. Kedua metoda tersebut baik untuk situasi-situasi yang terkendali. Bila tidak maka dibutuhkan penambahan heuristic.

Dasar dari metoda pencarian heuristic:

Maximizing atau minimizing beberapa aspek dari problem(masalah)

HEURISTIK

- Dari kata Yunani : *heuriskein* artinya “**to discover**” = menemukan
- adalah : sebuah teknik yang memperbaiki hasil efisiensi dari sebuah proses penelusuran / pencarian, kemungkinan dengan klaim-klaim korban dari kesempurnaan.

Keuntungan dari Depth first search :

- ❖ *depth-first search* membutuhkan sedikit memory, karena hanya node-node pada path aktif (*current*) yang disimpan. Ini sangat berbeda dengan *breadth-first search*, dimana semua part dari tree sepanjang telah dihasilkan harus disimpan.
- ❖ Secara kebetulan (atau jika penanganan diambil dalam urutan state pengganti alternatif). *Depth first search* dapat menemukan sebuah solusi tanpa uji coba beberapa penelusuran tempat kosong pada keseluruhannya. Ini berbeda dengan *breadth-first search*, semua bagian dari tree harus di ujicoba pada level n sebelum beberapa node pada level $n+1$ dapat di ujicoba.
Ini secara khusus pasti jika beberapa solusi dapat diterima. *Depth-first-search* dapat berhenti bila salah satu dari mereka ditentukan.

Fungsi/Function Heuristik adalah : Fungsi yang melakukan pemetaan (*mapping*) dari diskripsi keadaan masalah (*problema*) ke pengukur kebutuhan, umumnya direpresentasikan berupa angka.

Sangat penting dalam *Expert system* sebagai komponen esensial dalam memecahkan persoalan.

George Polya mendefinisikan heuristik sebagai studi metode & aturan penemuan.

Dalam proses search ruang keadaan (*state space*), heuristik dinyatakan sebagai aturan untuk melakukan pemilihan cabang dalam ruang keadaan yang paling dapat diharapkan mencapai pemecahan masalah yang dapat diterima.

AI menggunakan heuristik dalam dua situasi dasar :

- a. Persoalan/problema yang mungkin memiliki solusi eksak, namun biaya perhitungan untuk menemukan solusi tersebut sangat tinggi dalam kebanyakan persoalan (seperti catur), ruang keadaan bertambah secara luar biasa seiring dengan jumlah.
- b. Persoalan yang mungkin tidak memiliki solusi eksak karena ambiguitas (keraguan atau ketidakpastian) mendasar dalam pernyataan persoalan atau data yang tersedia, Diagnosa medis merupakan salah satu contohnya.

Heuristik menangani kerumitan masalah dengan cara memadu proses search pada sepanjang lintas yang paling dapat diharapkan. Namun heuristik juga bisa salah. Oleh karena itu heuristik hanyalah sebuah cara menerka langkah berikutnya yang harus diambil dalam memecahkan suatu persoalan berdasarkan informasi yang ada/tersedia.

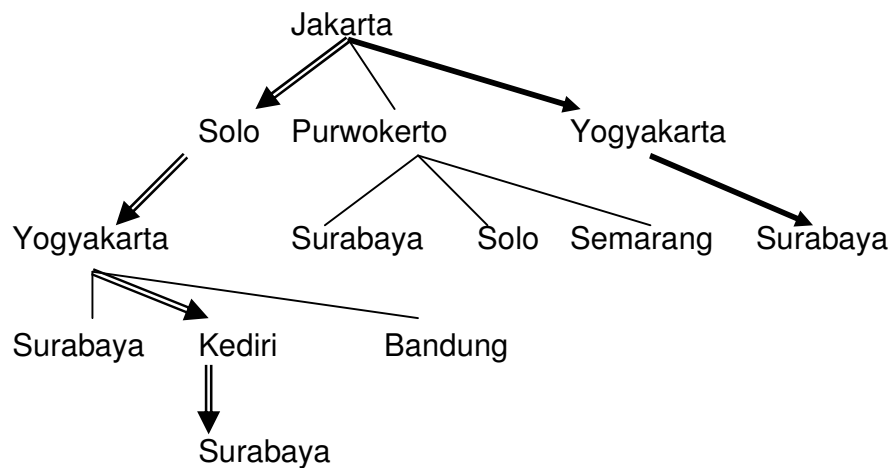
4.3. Teknik Pencarian *Hill Climbing*

Pada tabel jarak dari Jakarta ke Surabaya terdapat 2 batasan kemungkinan di mana seseorang ingin meminimalkan hal:

1. Jumlah hubungan(*connection*) yang harus dibuat
2. Panjang *Route*
(*Route* terpendek tidak berarti hubungan yang paling sedikit)

Algoritma pencarian mencoba untuk menemukan solusi pertama yang meminimalkan jumlah hubungan dengan menggunakan informasi heuristic. Dalam AI metoda pencarian yang menggunakan informasi heuristic disebut: *Hill Climbing*.

Pada umumnya algoritma *Hill Climbing* memilih langkah berikut node yang berada ditempat yang terdekat dengan goal.



Percobaan pertama : 2900 Jakarta – Yogyakarta - Surabaya

Percobaan kedua : 4900 Jakarta - Yogyakarta- Kediri - Surabaya

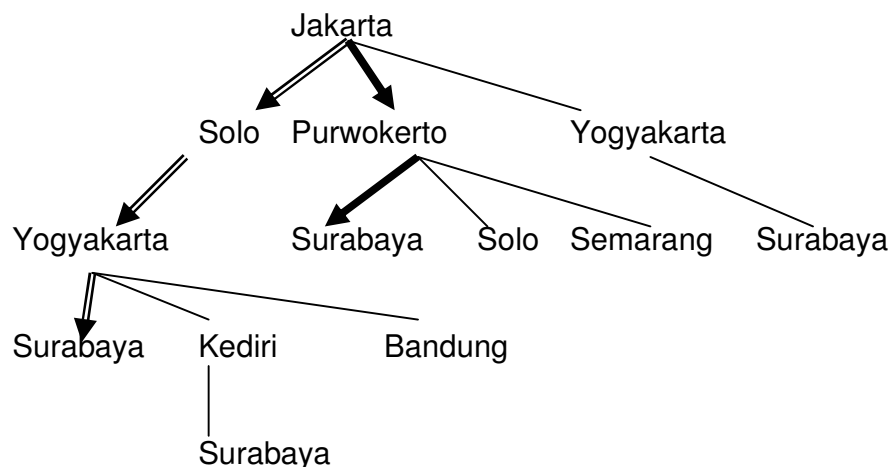
4.4. Teknik Pencarian *Least Cost*

Kebalikan dari pencarian *Hill Climbing* adalah pencarian *Least Cost*.

Pada metoda ini : hubungan terpendek akan diambil sehingga route yang ditemukan mempunyai kemungkinan yang baik untuk mendapat jarak terpendek.

Jadi *Hill Climbing* : meminimkan jumlah hubungan

Least Cost : meminimkan jumlah jarak yang ditelusuri



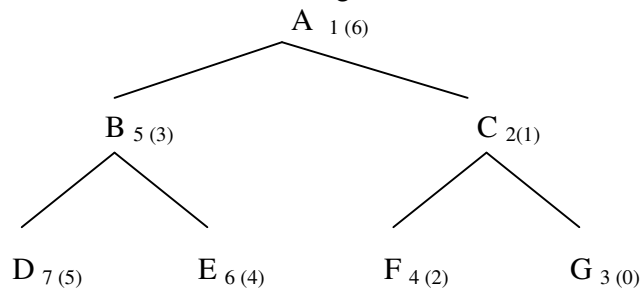
Percobaan pertama : 2600 Jakarta – Purwokerto - Surabaya

Percobaan kedua : 4900 Jakarta - Solo – Yogyakarta - Surabaya

4.5. Best First Search

Algoritma best first search menurut Thomas Dean, et all, AI teori & practice, page 145, sebagai berikut:

1. Himpun N menjadi sebuah list berurutan dari node awal(*initial nodes*)
2. Jika N adalah kosong, maka keluar dan berikan pesan gagal/*failure*
3. Himpun n menjadi node pertama dalam N, dan hapus n dari N
4. Jika n adalah node tujuan/*goal* maka keluar dan berikan pesan sukses
5. Selain itu, tambahkan anak dari n ke N, urutkan node-node dalam N menurut jarak estimasi dari sebuah goal, dan kembali ke langkah 2.



Implementasi best-first search dalam fungsi LISP (*LISP FUNCTION*):

```

(defun best (nodes goalp next comparep)
  (cond ((null nodes) nil)
        ;; return the first node if it is a goal node
        ((funcall goalp (first nodes)) (first nodes))
        ;; append the children to the set of old nodes
        ;; and then sort them all according to value.
        (t ( best (sort (append (funcall next (first nodes))
                                (rest nodes))
                      comparep)
                  goalp
                  next
                  comparep))))
  
```

```

(defun TREE-comparator (tree1 tree2)
  (< (TREE-value tree1) (TREE-value tree2)))
  
```

setelah didemonstrasikan dengan goals sebagai berikut:

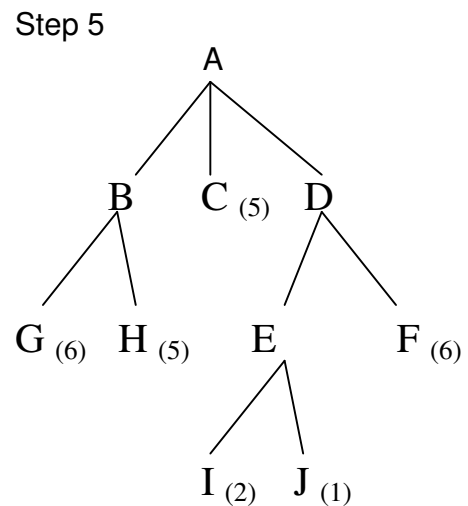
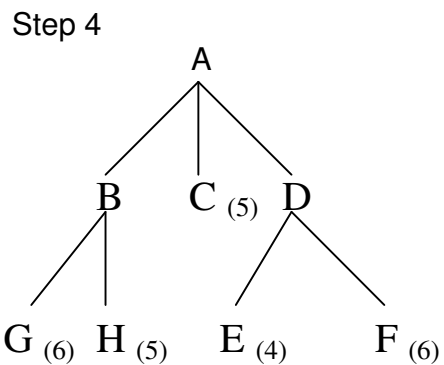
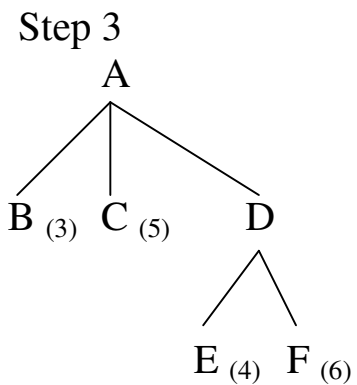
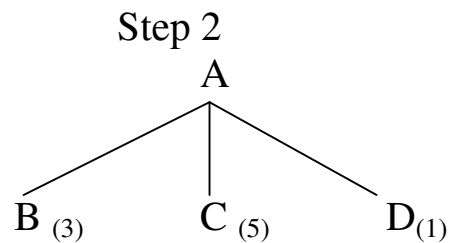
```

> (best (list tree)
      #'(lambda (x) (TREE-print x) nil)
      #'TREE-children
      #'TREE-comparator)
ACGFBED
NIL
  
```

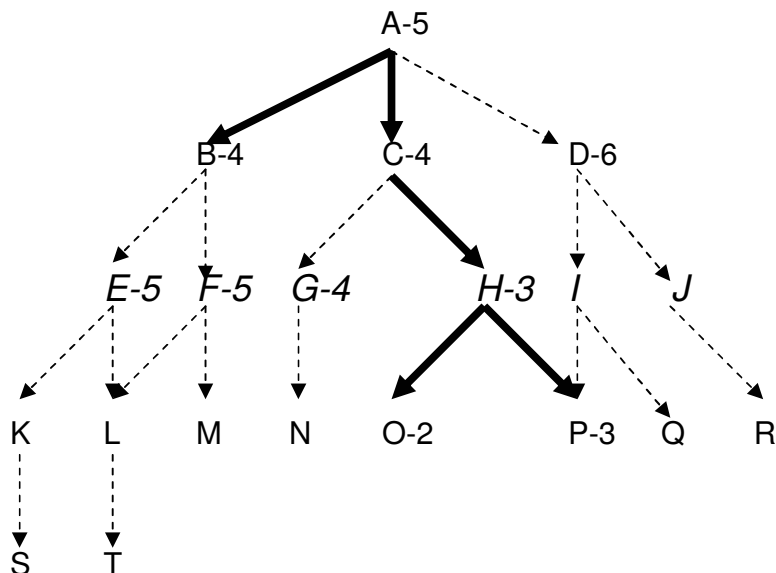
Algoritma *Best First Search* menurut Elaine Rich & Kevin Knight, "Artificial Intelligence", page 73, sebagai berikut:

1. Start with OPEN containing just the initial state
2. Until a goal is found or there are no nodes left on OPEN do:
 - (a) Pick the best node on OPEN
 - (b) Generate its successors
 - (c) For each successor do :
 - i. if it has not been generated before, evaluate it, add it to OPEN, and record its parent.
 - ii. If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

Step 1
A



Algoritma *Best First Search* menurut George F. Luger, & William A. Stubblefield, “*Artificial Intelligence: Structured & Strategies for Complex Problem Solving*”, 2nd Edition, page 120-123, sebagai berikut:



1. OPEN = [A-5]; CLOSED = []
 2. Evaluate A-5; OPEN = [B-4, C-4, D-6];
CLOSED = [A-5]
 3. Evaluate B-4; OPEN = [C-4, E-5, F-5, D-6];
CLOSED = [B-4, A-5]
 4. Evaluate C-4; OPEN = [H-3, G-4, E-5, F-5, D-6];
CLOSED = [C-4, B-4, A-5]
 5. Evaluate H-3; OPEN = [O-2, P-3, G-4, E-5, F-5, D-6];
CLOSED = [H-3, C-4, B-4, A-5]
 6. Evaluate O-2; OPEN = [P-3, G-4, E-5, F-5, D-6];
CLOSED = [O-2, H-3, C-4, B-4, A-5]
- Evaluate P-3; The Solution is found!

Kesimpulan : *best first search* merupakan suatu teknik pencarian yang mengkombinasikan yang terbaik diperoleh dari teknik *depth-first search* dan teknik *breadth-first search* ke dalam sebuah metode tunggal.

4.6. Teknik Pencarian Minimax

Prosedur Pencarian Minimax adalah langkah *depth-first*, idenya adalah memulai pada saat sebagian posisi dan menggunakan *plausible-move generator*

untuk menggerakkan himpunan pada posisi *possible successor*. Sekarang kita dapat mengaplikasikan fungsi *evaluasi statis* ke posisinya dan memilih salah satunya yang terbaik.

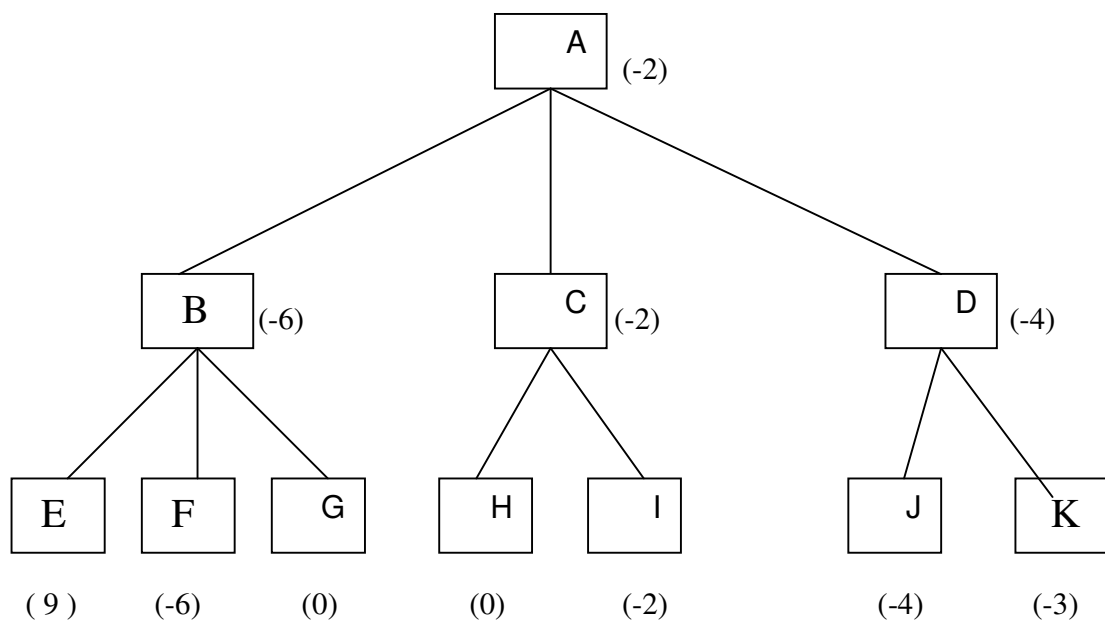
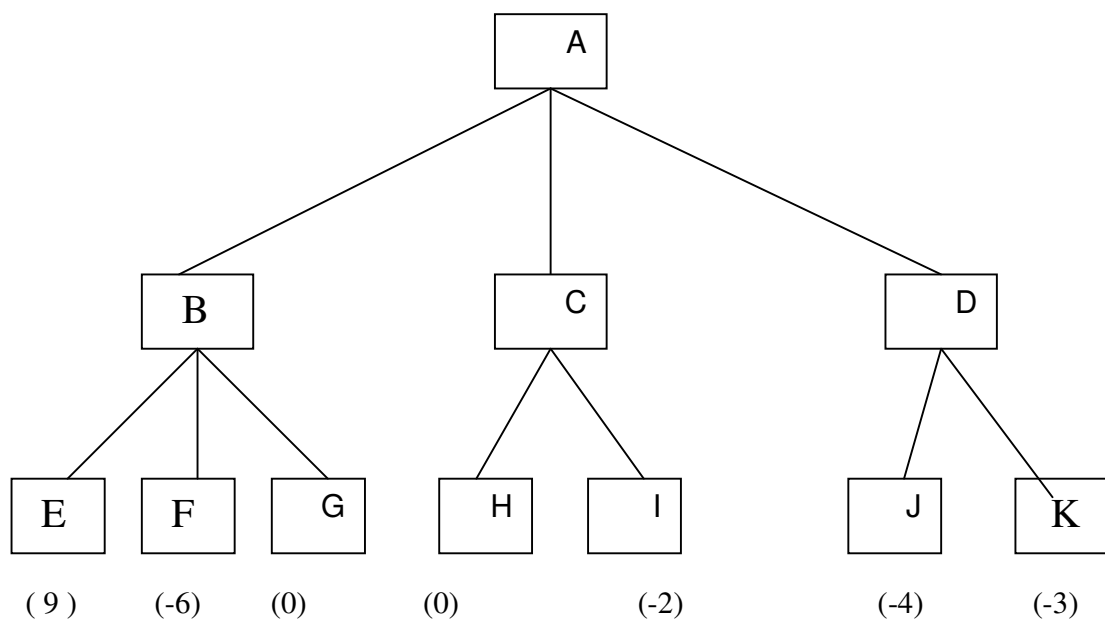
Setelah melakukannya, kita dapat mengembalikan nilai pada permulaan posisi untuk memperoleh hasil evaluasi. Permulaan posisi sangat tepat sekali untuk dilakukan oleh kita sebagai posisi yang digerakkan oleh langkah kita yang terbaik untuk selanjutnya. Disini kita mengamsumsi bahwa fungsi evaluasi statis akan kembali ke nilai yang lebih besar yang mengidentifikasi kita untuk situasi yang tepat, lalu tujuan akhir(goal) adalah memaksimalkan nilai pada fungsi evaluasi statis pada posisi berikutnya.

Contohnya dapat kita lihat pada gambar 4.1. pada gambar tersebut diamsumsikan bahwa fungsi evaluasi statis akan kembali pada nilai yang ditunjukkan pada -10 to 10, dengan 10 mengidentifikasi kemenangan untuk kita, -10 menang untuk opponent, dan 0 untuk pertandingan berikutnya. Sejak tujuan akhir kita (goal) adalah untuk memaksimalkan nilai pada fungsi heuristic, kita memilih untuk mengerakkannya ke B, simpan nilai B ke A, kita dapat menyatakan bahwa nilai A adalah 8, sejak kita mengetahuinya kita dapat mengerakkannya ke posisi yang bernilai 8.

Tetapi sejak kita mengetahui bahwa fungsi evaluasi statis tidaklah lengkap, kita memutuskan untuk membawanya ke tempat yang lebih jauh di banding satu permainan. Ini penting misalnya pada permainan catur dimana kita berada ditengah diantara benteng.

Sesudah kita bergerak, situasinya haruslah baik, tetapi jika kita melihat satu pergerakan raja, kita dapat melihat bahwa salah satu benteng kita akan di makan dan situasi ini tidaklah baik untuk dilihat, lalu kita dapat melihat raja lihatlah apa yang terjadi pada langkah berikutnya yang dapt dilihat oleh opponent-nya.

Paling sedikit pada pengaplikasian fungsi evaluasi statis untuk beberapa posisi yang kita kembangkan, kita dapat mengaplikasikan *plausible-move generator*, pergerakan posisi successor untuk beberapa posisi.



Gambar 4.1.Memback Up Nilai Pada Two-Ply Search

Tetapi kita harus dapat membawa ke dalam perhitungan berarti *opponent* dapat kita pilih berdasarkan *successor* yang bergerak untuk membuat dan demikian yang mana nilai terminal seharusnya di *backup* untuk level berikutnya.

Berdasarkan yang kita buat pada langkah B. lalu *opponent* harus memilih diantara langkah E,F dan G . pada *goal opponent* adalah meminimalkan nilai pada fungsi evaluasi, lalu dia dapat memilih untuk meggerakkan F. ini berarti jika kita

menggerakkan B, posisi aktualnya memungkinkan suatu konfigurasi yang dipersembahkan oleh E, yang sangat baik untuk kita .

Tetapi sejak kita berada pada level tersebut kita tidak dapat melakukan satu langkah, kita tidak akan melakukan pilihan tersebut. Pada gambar 4.1. kita dapat melihat hasil propaganda pada nilai yang baru pada sebuah tree. Pada level ini mempersembahkan pilihan opponent, nilai minimum dapat dipilih dan di back up. Pada level ini kita mempersembahkan pilihan kita, nilai maksimum yang kita pilih.

Pada nilai yang pertama dari permainan kedua di backup, ini akan menjadi langkah yang tepat untuk kita yang kita buat pada level pertama, memberikan informasi yang memungkinkan adalah C. Sejak tidak terdapat opponent maka tidak dapat dilakukan dari sini untuk memproduksi nilai salah lebih dari -2 . Proses ini dapat diulang untuk beberapa waktu, dan lebih akurat evaluasinya, sehingga hasilnya dapat digunakan untuk langkah berikutnya pada level teratas.

Alternatif pada maximizing dan minimizing pada alternatif permainan, ketika evaluasi ditekan oleh korespondennya untuk strategi mengopponent pada dua pemain dan memberikan pengertian nama minimax.

Memiliki informasi yang menjelaskan pengoperasian prosedur minimax, kita sekarang menggambarkan secara lengkap. Ini merupakan langkah tepat pada prosedur rekursif bahwa ada dua alasan yang menjelaskan secara spesifik mengenai permainan yang dimainkan :

1. MOVEGEN (position, Player)

Plausible-move generator, yang mengembalikan node yang mempersembahkan langkah yang dapat dibuat oleh pemain dalam suatu posisi.

Kita menyebutnya sebagai dua pemain yaitu pemain 1 dan pemain 2; pada program catur, kita seharusnya menggunakan nama HITAM dan PUTIH.

2. STATIC (position, player)

Fungsi evaluasi statis, yang mengembalikan angka yang mempersembahkan kebaikan pada suatu posisi dari suatu penilaian oleh pemain.

Dengan beberapa program rekursif, isu kritikal dalam suatu design prosedur minimax dapat diberhentikan pada pengrekursiannya dan dapat dipanggil dengan mudah oleh fungsi evaluasi statis. Dapat dinilai factor varietasnya yang menyebabkan suatu keputusan. Hal tersebut ialah :

- Apakah satu sisi menang ?
- Berapa banyak ply yang dimiliki kita siap dieksplorasi ?
- Apakah path menjanjikan ?
- Berapa waktu yang diperlukan ?
- Apakah dapat menjamin konfigurasinya ?

Untuk pusat dari prosedur minimax dibahas disini , kita dapat menggunakan DEEP-ENOUGH, yang mengasumsikan untuk mengevaluasi semua faktor dan memberikan TRUE jika pencarian dapat dihentikan pada sebagian level dan sebaliknya FALSE.

Kita menggunakan sangat sederhana pada DEEP-ENOUGH yang dapat mengambil dua parameters, posisi dan Depth. Ini dapat menyebabkan ketidakperdulian pada posisi parameters dalam pemberian nilai TRUE, jika pada DEPTH parameter memotong nilai yang konstan.

Suatu permasalahan yang menjelaskan minimax sebagai prosedur rekursif yang dibutuhkan untuk tidak mengembalikan salah satunya tetapi terdapat dua hasil :

- Pemback up-an nilai pada path yang dipilih.
- Path tersebut. Kita mengembalikannya kedalam bagian path yang memungkinkan hanya sebagian elemen, yang mempersembahkan langkah yang baik dari sebagian posisi, yang sangat dibutuhkan.

Kita mengasumsikannya bahwa pengembalian struktur minimax menghubungkan hasil keduanya dan terdapat dua fungsi yaitu VALUE dan PATH, yang mengekstrak sebagian komponen.

Sejak kita menemukan struktur prosedur minimax sebagai fungsi rekursif, kita harus dapat menjelaskan secara spesifik bagaimana ia dapat dipanggil inisialnya. Ini membutuhkan tree parameter, pada board posisi, pada sebagian

DEPTH pada suatu pencarian dan pemain untuk menjalankannya. Maka penginisialannya dapat dipanggil untuk menghitung langkah yang terbaik pada posisi CURRENT seharusnya.

```

MINIMAX( CURRENT, 0, PLAYER-ONE)
IF PLAYER – ONE IS TO MOVE, OR
    MINIMAX( CURRENT, 0, PLAYER-TWO)
IF PLAYER – TWO IS TO MOVE, OR

```

ALGORITMA : MINIMAX (POSITION, DEPTH, PLAYER)

1. IF DEEP-ENOUGH(POSITION, DEPTH) THEN RETURN THE STRUCTURE

VALUE = STATIC (POSITION, PALYER)

PATH = NIL

Ini mengidentifikasi bahwa tidak terdapat path dari node ini dan bahwa nilai dideterminasikan oleh fungsi evaluasi statis.

2. Sebaliknya, satu perkembangan ply lagi pada suatu tree oleh pemanggilan fungsi MOVEGEN(Posistion, Player) dan pengesetan SUCCESSORS pada suatu list.
3. jika SUCCESSORS kosong, maka tidak pergerakan yang dibuat, lalu kembali ke struktur yang sama bahwa akan terjadi perubahan yang dikembalikan jika DEEP-ENOUGH telah dinyatakan benar.
4. Jika SUCCESSOR tidak kosong, maka pelaksanaan beberapa elemen yang turun dan menjaga agar track tersebut adalah yang terbaik. Sesuai dengan yang mengikutinya.

Penginisialisasian skor terbaik untuk nilai minimum bahwa STATIC dapat dikembalikan. Ini dapat menjadi refleksi dalam mengupdate nilai terbaik sehingga dapat mencapai suatu elemen pada SUCCESSORS.

Untuk beberapa elemen SUCC pada SUCCESSOR, dapat mengikuti :

- SET RESULT-SUCC TO
MINIMAX(SUCC,DEPTH+1,OPPOSITE(PLAYER))³

Pada rekursif memanggil minimax dapat secara aktual membawa pengekplorasian pada SUCC.

- SET NEW-VALUE TO-VALUE(RESULT-SUCC).

Ini dapat menimbulkan refleksi dari kebaikan pada suatu posisi dari tempat yang prepektif pada level terbawah berikutnya.

- IF NEW-VALUE > BEST-SCORE, maka dapat kita temukan pada successor bahwa lebih baik daripada beberapa hal yang kita pelajari selama ini. Pada Record ini dapat dilakukan dengan mengikuti :

- SET BEST-SCORE TO NEW-VALUE

Yang terbaik yang perlu kita ketahui dari path ini adalah dari CURRENT ke SUCC dan maka untuk path ke bawah dengan tepat dari SUCC sebagai determinasi dari pemanggilan rekursif untuk minimax. Lalu set BEST-PATH untuk hasil pada lampiran SUCC ke depan pada PATH(RESULT-SUCC).

5. Sekarang semua Successor telah dipelajari, kita tahu nilai pada posisi baik seperti yang dapat diambil pada path tersebut. Lalu kembalikan ke struktur :

VALUE = BEST-SCORE

PATH = BEST-PATH

Ketika penginisialisasian pemanggilan pada minimax, langkah yang terbaik dari CURRENT adalah elemen pertama pada suatu path. Untuk melihatnya dapat dilihat pada saat prosedur ini bekerja, anda seharusnya dapat melacak pada suatu eksekusi pada permainan game tree yang diperlihatkan pada gambar 4.1

Prosedur minimax dapat dijabarkan kedalam bentuk yang sederhana. Tetapi performancenya dapat di buktikan secara significant dengan beberapa perbaikan. Beberapa diantaranya menjelaskan langkah beberapa seksi.