**PROBLEM SOLVING DENGAN PENCARIAN**

**3**

TUJUAN INSTRUKSIONAL KHUSUS :

1. Mendeskripsikan masalah dan mencari penyelesaiannya dengan algoritma pencarian
2. Memilih algoritma pencarian buta yang tepat untuk sebuah permasalahan, mengimplementasikan dan karakteristiknya pada kompleksitas waktu dan ruang
3. Mengimplementasikan pencarian minimax dan alpha-beta pruning pada beberapa game 2 pemain.

**Materi :**

* 1. **Problems Solving Agent**

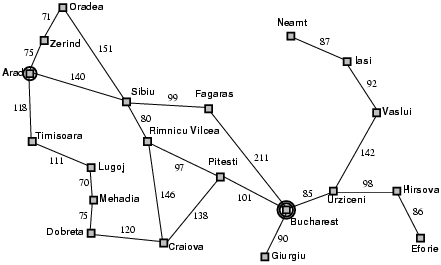
Problem solving agent memutuskan apa yang akan dilakukan dengan mencari serangkaian aksi yang mengarah pada kondisi yang diharapkan. Intelligence agent seharusnya memaksimalkan ukuran kinerja mereka.

Contoh :

* Sebuah agent melakukan perjalanan ke Romania. Agent saat ini ada di Arad.
* Pesawat terbang besok dari Bucharest.
* **Formulated Goal**, didasarkan pada situasi saat ini dan ukuran kinerja agent, adalah langkah pertama dalam menyelesaikan masalah. Tugas agent adalah untuk menemukan rangkaian aksi yang mana yang membawa ke goal kondisi. Sebelum melakukan itu, perlu diputuskan aksi dan kondisi apa yang dipertimbangkan. Untuk agent pada contoh, Formulated Goal adalah :
* Ada di Bucharest
* **Formulated Problem** adalah proses memutuskan aksi dan kondisi apa yang dipertimbangkan, yang membawa ke goal. Agent pada contoh memiliki Formulated Problem sebagai berikut :
* Kondisi : bermacam-macam kota
* Aksi : berkendaraan antara kota

Agent sekarang memiliki goal untuk berkendaraan ke Bucharest dan mempertimbangkan akan kemana pergi dari Arad. Ada beberapa jalur alternatif menuju Arad. Tetapi agent tidak tahu jalan terbaik karena tidak cukup tahu tentang kondisi yang hasilnya dari pengambilan aksi. Jika agent tidak memiliki pengetahuan tambahan maka akan stuck. Cara terbaik adalah memilih secara acak.

* **Find Solution**. Proses untuk mencari rangkaian penyelesaian disebut Search (pencarian). Suatu algoritma pencarian menganggap masalah sebagai input dan menghasilkan solusi dalam bentuk rangkaian aksi. Sekali solusi ditemukan, aksi yangmenyarankan dikeluarkan. Hal ini disebut eksekusi. Pada contoh, agent memiliki solusi, yaitu :
* Serangkaian kota, misal : Arad, Sibiu, Faragas, Bucharest.

****

Gambar 3.1 Peta jalan Romania yang disederhanakan

**Masalah dan Penyelesaian**

Suatu masalah dapat didefinisikan secara formal oleh 4 komponen :

1. **Initial states** (kondisi awal) dimana agent mulai. Contoh : “di Arad”
2. Deskripsi aksi yang memungkinkan yang tersedia untuk agent. Umumnya **actions** atau **successor function** S(x) = himpunan aksi-pasangan kondisi. Contoh :
3. **Test goal** yang menentukan apakah kondisi yang diberikan adalah kondisi tujuan goal. Bentuknya dapat berupa :

* Eksplisit : x = “at Bucharest”
* Implicit : Checkmate(x)

1. **Path cost**. Fungsi path cost yang memberikan cost (nilai) numeric pada tiap path (jalur).

* Jumlah jarak, banyaknya aksi yang dilakukan
* C(x,a,y) adalah step cost, diasumsikan 0.

Solusi adalah sebuah rangkaian aksi mulai dari kondisi awal sampai ke sebuah kondisi tujuan.

* 1. **Jenis-jenis algoritma pencarian**

Pencarian adalah masalah yang paling penting dan paling sering ditemui dalam bidang kecerdasan buatan. Permasalahan ini dapat menjadi penentu keberhasilan sistem kecerdasan buatan.

Pada bagian akan dibahas 3 jenis metoda pencarian, yaitu :

1. Metoda pencarian sederhana yang hanya mencari penyelesaian, misalnya : depth-first search, breadth-first search, hill climbing, dan beam search.

Metoda pencarian yang mencari jarak atau jalur terpendek, misalnya British Museum Procedure, Branch and Bound, Dynamic Programming dan A\*. metoda-metoda ini mencari kemungkinan optimal dari harga perjalanan.

1. Metoda yang digunakan untuk berhadapan dengan musuh, misalnya minimax search, alpha-betha prunning.

Gambar 3.1 Diagram metoda pencarian

Manfaat metoda pencarian :

1. Untuk menyelesaikan masalah setiap state (keadaan) untuk menggambarkan langkah-langkah penyelesaian masalah
2. Untuk perencanaan, menentukan apa ynag harus dilakukan, dimana setiap state menggambarkan kemungkinan posisi pada suatu saat.
3. Bagian dari kesimpulan, dimana setiap state menggambarkan hipotesa dalam rangkain deduktif.

* 1. **Mendefinisikan masalah dalam suatu ruang keadaan**

Untuk mendeskripsikan suatu masalah dengan baik secara umum harus :

1. Mendefinisikan suatu ruang keadaan
2. Menerapkan satu atau lebih keadaan awal
3. Menetapkan satu atau lebih tujuan
4. Menetapkan kumpulan aturan

Contoh kasus I :

Seorang petani ingin memindahkan dirinya, seekor serigala, seekor ayam gemuk, dan seikat padi menyeberangi sungai. Sayangnya kapasitas perahu terbatas, perahu hanya dapat membawa satu objek untuk menyeberangi sungai dalam satu waktu. Selain kapasitas perahu, petani tidak dapat meninggalkan serigala dengan ayam, atau ayam dengan padi.

Berdasarkan masalah tersebut, untuk mendefiniskan masalah sebagai ruang keadaan, dapat ditentukan hal-hal sebagai berikut :

1. Identifikasi ruang keadaan

Masalah tersebut dapat dilambangkan dengan JumlahSerigala, JumlahAyam, JumlahPadi, JumlahPetani. Contoh : Daerah asal (0,1,1,1) berarti daerah asal tidak ada serigala, ada ayam ada padi dan ada petani.

1. Keadaan awal dan tujuan
   1. Keadaan awal, pada kedua sisi sungai :
      1. Daerah asal :(1,1,1,1)
      2. Daerah seberang : (0,0,0,0)
   2. Tujuan, pada kedua sisi sungai :
      1. Daerah asal : (0,0,0,0)
      2. Daerah seberang : (1,1,1,1)
2. Aturan

Aturan dapat di tentukan dalam Tabel 3.1 berikut :

Tabel 3.1 Aturan-aturan penyeberangan petani dan bawaannya

|  |  |
| --- | --- |
| **Aturan** | **Keterangan** |
| 1 | Ayam menyeberang |
| 2 | Padi menyeberang |
| 3 | Serigala menyeberang |
| 4 | Angsa kembali |
| 5 | Padi kembali |
| 6 | Serigala kembali |
| 7 | Petani kembali |

Salah satu solusi dapat dilihat pada Tabel 3.2

Tabel 3.2 Contoh solusi masalah petani dan bawaannya

|  |  |  |
| --- | --- | --- |
| **Daerah asal** | **Daerah seberang** | **Aturan yang digunakan** |
| (1,1,1,1) | (0,0,0,0) | 1 |
| (1,0,1,0) | (0,1,0,1) | 7 |
| (1,0,1,1) | (0,1,0,0) | 3 |
| (0,0,1,0) | (1,1,0,1) | 4 |
| (0,1,1,1) | (1,0,0,0) | 2 |
| (0,1,0,0) | (1,0,1,1) | 7 |
| (0,1,0,1) | (1,0,1,0) | 1 |
| (0,0,0,0) | (1,1,1,1) | solusi |

**Contoh kasus II :**

Ada 2 buah teko masing-masing memiliki kapasitas 4 galon untuk teko A dan 3 galon untuk teko B. tidak ada batas yang menunjukkan batas ukuran pada kedua teko tersebut. Masalah : bagaimana dapat mengisi tepat 2 galon air kedalam teko yang berkapasitas 4 galon ?

Dari masalah tersebut, didefinisikan masalah sebagai ruang keadaan ditentukan langkah-langkah berikut :

1. Identifikasi ruang keadaan

Masalah ini dapat dilambangkan dengan (x,y), dimana :

* + X = air yang diisikan pada teko A
  + Y = air yang diisikan pada teko B

1. Keadaan awal dan tujuan
   * Keadaan awal, keduaa teko dalam keadaan kosong (0,0)
   * Tujuan, keadaan dimana pada teko 4 galon berisi tepat 2 galon air : (2,n) untuk sembarang n.
2. Aturan

Aturan-aturan pada masalah teko dapat dilihat pada Tabel 3.3

Tabel 3.3 Aturan-aturan masalah teko air

|  |  |  |  |
| --- | --- | --- | --- |
| **Aturan** | **Kondisi awal** | **Kondisi tujuan** | **Keterangan** |
| 1 | (x,y), x < 4 | (4 , y) | Isi teko A sampai penuh |
| 2 | (x,y), y < 3 | (x , 3) | Isi teko B sampai penuh |
| 3 | (x,y), x > 0 | (x-d , y) | Tuang sebagian air keluar dari teko A |
| 4 | (x,y), y > 0 | (x , y-d) | Tuang sebagian air keluar dari teko B |
| 5 | (x,y), x > 0 | (0 , y) | Kosongkan teko A dengan membuang airnya |
| 6 | (x,y), y > 0 | (x , 0) | Kosongkan teko B dengan membuang airnya |
| 7 | (x,y), x+y ≥ 4 dan y > 0 | (4, y-(4-x)) | Tuang air dari teko B ke A sampai teko A penuh |
| 8 | (x,y), x+y ≥ 3 dan x > 0 | (x-(3-y), 3 ) | Tuang air dari teko A ke B sampai teko B penuh |
| 9 | (x,y), x+y ≤ 4 dan y > 0 | (x+y+ , 0) | Tuang seluruh air dari teko B ke teko A |
| 10 | (x,y), x+y ≤ 3 dan x > 0 | (0 , x-y) | Tuang seluruh ait dari teko A ke teko B |
| 11 | (0 , 2) | (2 , 0) | Tuang 2 galon air dari teko B ke teko A |
| 12 | (2 , y) | (0 , y) | Kosongkan 2 galon air dari teko A dengan membuang airnya |

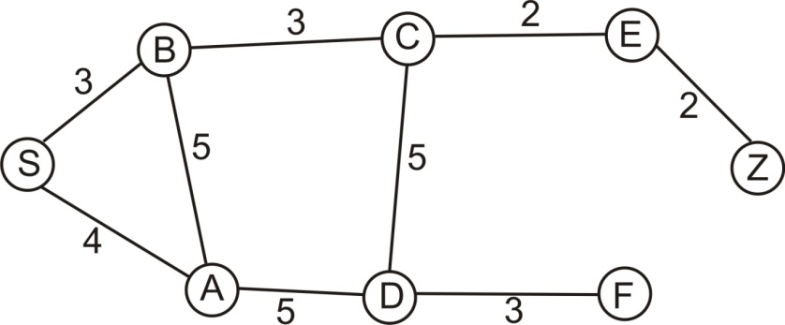
Salah satu solusi dapat ditemukan pada Tabel 3.3

Tabel 3.4 Contoh solusi masalah teko air

|  |  |  |
| --- | --- | --- |
| **Kondisi awal** | **Kondisi tujuan** | **Aturan yang digunakan** |
| (0,0) | (0,3) | 2 |
| (0,3) | (3,0) | 9 |
| (3,0) | (3,3) | 2 |
| (3,3) | (4,2) | 7 |
| (4,2) | (0,2) | 5 |
| (0,2) | (2,0) | 9 |

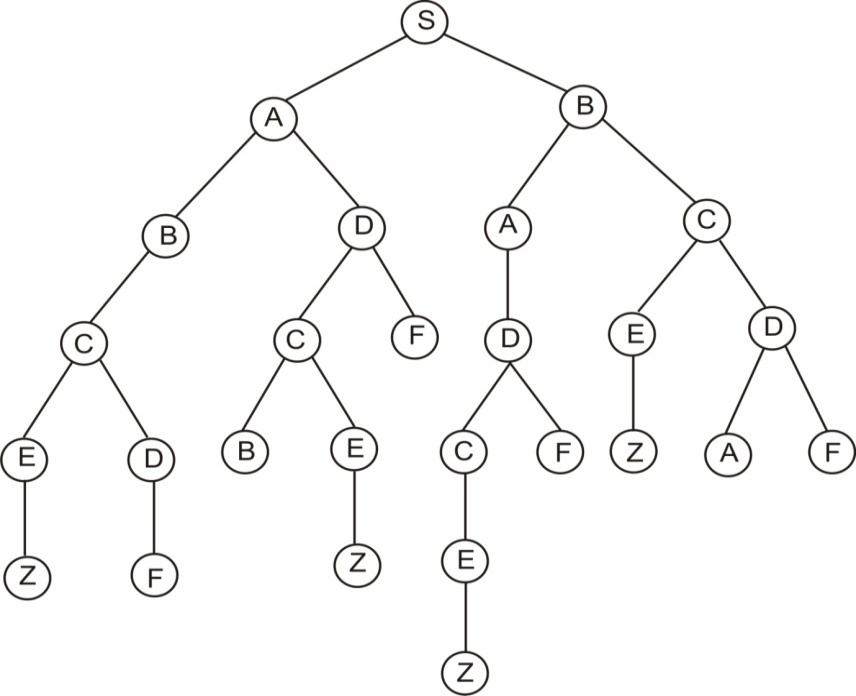
* 1. **Pohon pelacakan**

Struktur pohon digunakan untuk menggambarkan keadaan secara hirarkis. Pohon terdiri dari beberapa node. Node akar menunjukan keadaan awal yang biasanya sebuah topik atau objek. Node akar memiliki percabangan. Masalah pencarian maju, node cabang disebut node successor. Pencarian mundur, node cabang disebut node predecessor. Node tanpa cabang atau node daun adalah akhir pencarian, dapat berupa tujuan yang diharapkan atau cabang buntu.



Gambar 3.2 Contoh graph yang berisi path antar kota.

Graph pada Gambar 3.2 dibuat struktur tree seperti pada Gambar 3.3.

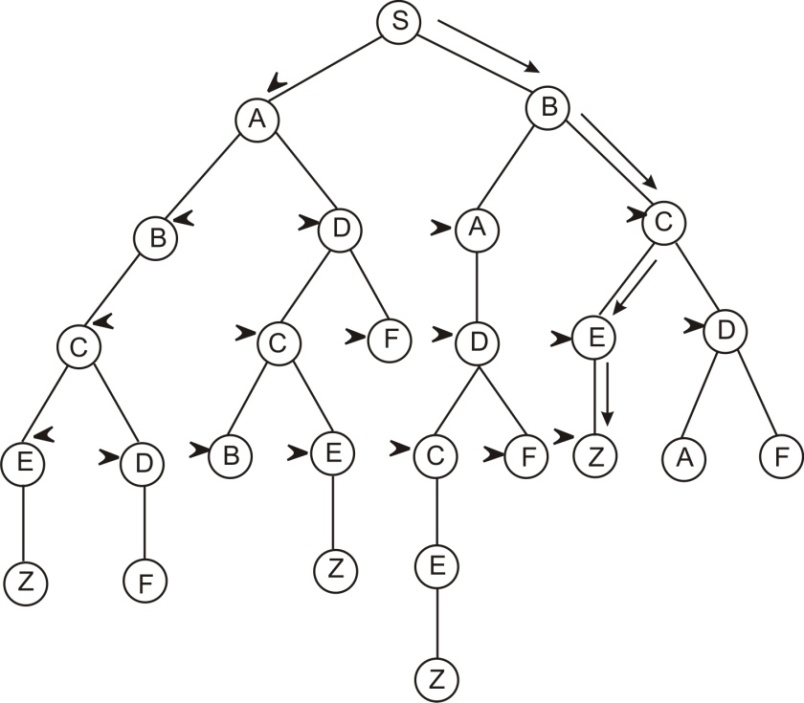


Gambar 3.3 Struktur tree untuk graph pada Gambar 3.2

* 1. **Pencarian buta (Breadth First, Depth First, Hill Climbing, Best First)**

1. Breadth-First Search

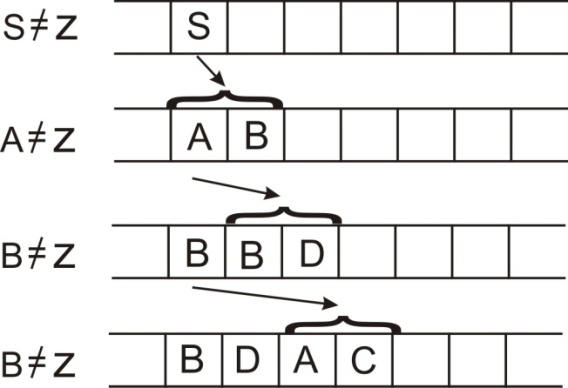
Pada metode Breadth-First Search, semua node level n akan dikunjungi terlebih dahulu sebelum mengunjungi node-node pada level n+1. Pencarian dimulai dari node akar terus ke level ke-1 dari kiri ke kanan, lalu pindah ke level berikutnya demikian pula dari kiri ke kanan sampai ditemukannya solusi.



Gambar 3.4 Penelusuran metoda Breadth-First Search

* + Algoritma

1. Buat sebuah antrian, inisialisasi node pertama dengan Root dari tree
2. Bila node pertama, jika ≠ GOAL, digantikan dengan anak-anaknya dan diletakkan di belakang PER LEVEL
3. Bila node pertama = GOAL, selesai

****

Lintasan yang didapat : S – B – C – E – Z

* + Keuntungan

1. Tidak akan menemui jalan buntu
2. Jika ada satu solusi, maka Breadth First Search akan menemukannya. Dan jika ada lebih dari satu solusi, maka solusi minimum akan ditemukan.
   * Kelemahan
3. Membutuhkan memori yang cukup banyak, karena menyimpan semua node dalam satu pohon
4. Kemungkinan ditemukan optimal lokal
5. Depth-First Search

Proses pencarian akan dilaksanakan pada semua anaknya sebelum dilakukan pencarian ke node-node yang selevel. Pencarian dimulai terus hingga ditemukannya solusi.

* + Algoritma

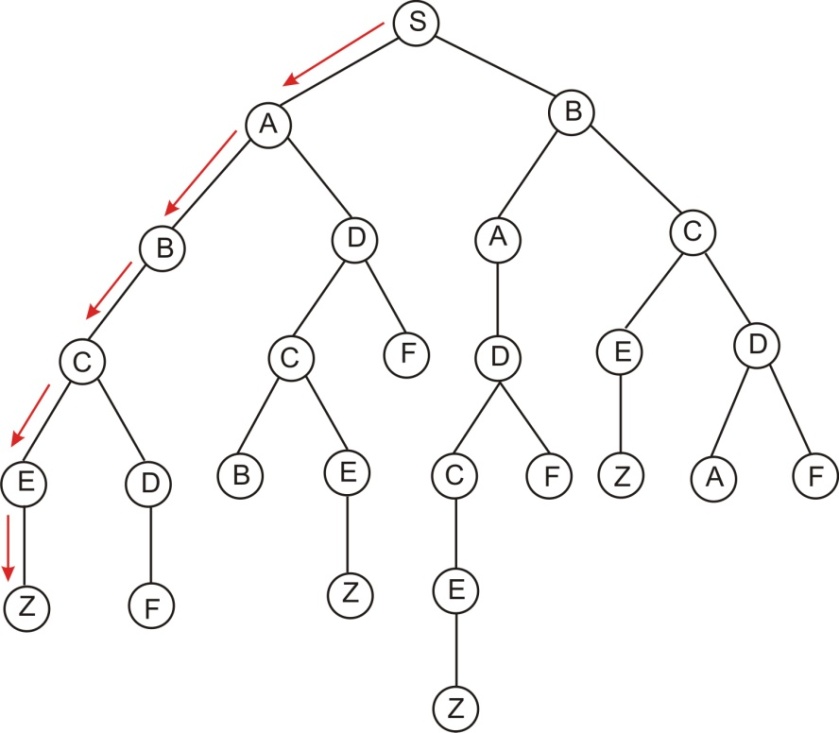
1. Buat sebuah antrian, inisialisasi node pertama dengan Root dari tree
2. Bila node pertama, jika ≠ GOAL, node dihapus diganti dengan anak-anaknya dengan urutan Lchild
3. Bila node pertama = GOAL, selesai

* Keuntungan

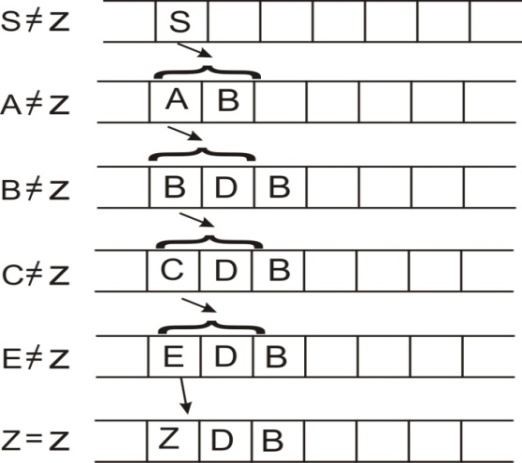
1. Membutuhkan memori yang relatif kecil, karena hanya node-node pada lintasan yang aktif saja yang disimpan.
2. Menemukan solusi tanpa harus menguji lebih banyak lagi ruang keadaan

* Kelemahan

1. Kemungkinan terjebak pada optimal lokal
2. Hanya akan mendapatkan 1 solusi pada setiap pencarian

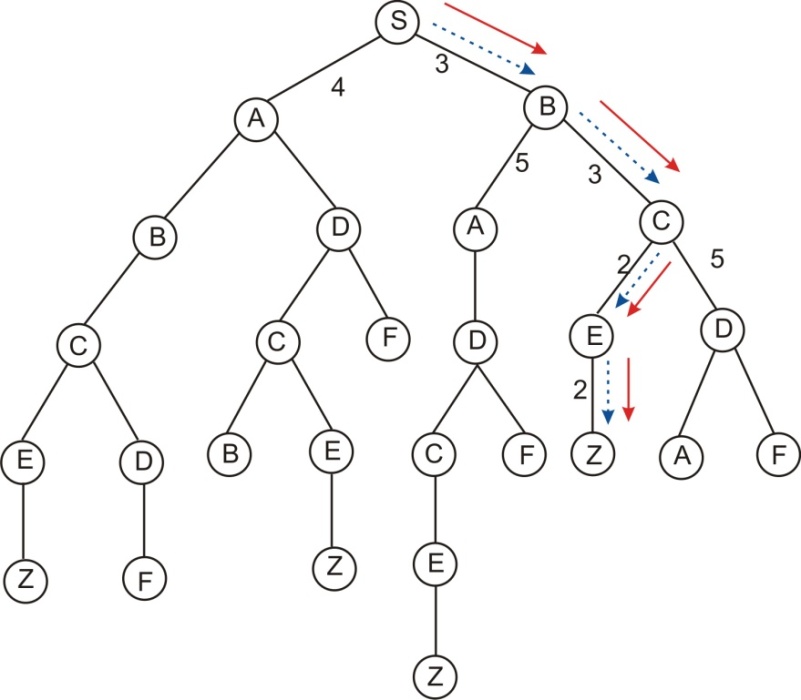
****

Gambar 3.5 Depth-First Search Tree

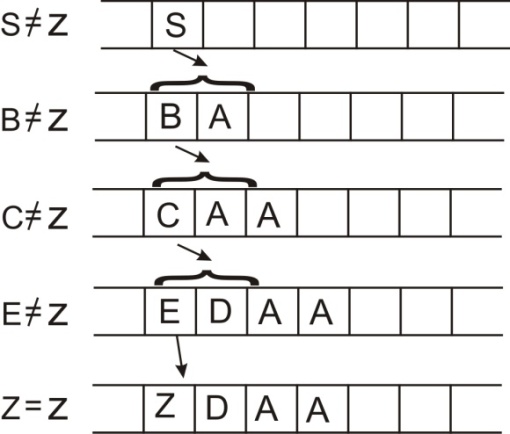


Lintasan yang didapat : S – A – B – C – E – Z

1. Hill Climbing
   * Algoritma
2. Buat sebuah antrian, inisialisasi node pertama dengan Root dari tree
3. Bila node pertama, jika ≠ GOAL ; node dihapus diganti dengan anak-anaknya dengan urutan yang paling kecil jaraknya
4. Bila node pertama = GOAL, selesai



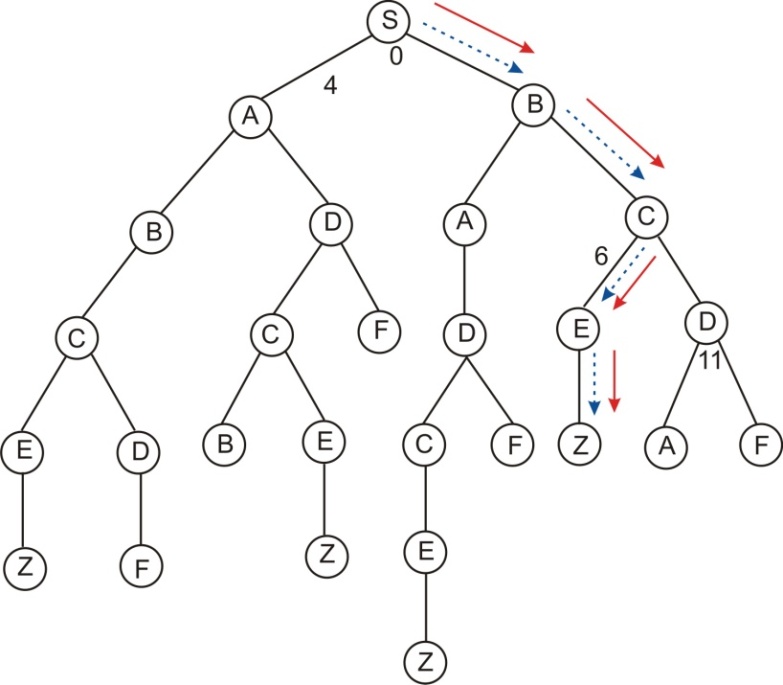
Gambar 3.6 Hill Climbing Tree



Lintasan yang didapat : S – A – B – C – E – Z

* + Keuntungan

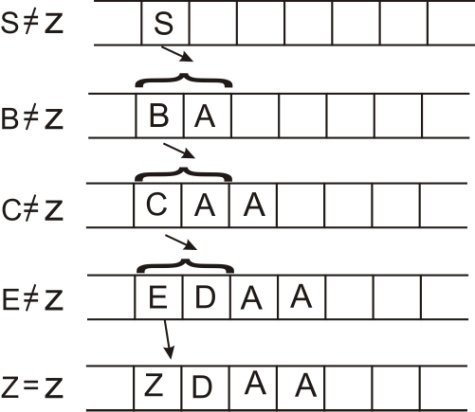
1. Membutuhkan memori yang relaatif kecil, karena hanya node-node pada lintasan yang aktif saja yang disimpan
2. Metoda hill climbing search akan menemukan solusi tanpa harua menguji lebih banyak lagi ruang keadaan.
   * Kerugian
3. Algoritma akan berhenti kalau mencapai nilai optimum lokal
4. Perlu menentukan aturan yang tepat
5. Best-First Search

****

Gambar 3.7 Best-First Search Tree

* + Algoritma

1. Buat sebuah antrian, inisialisasi node pertama dengan Root dan tree
2. Buat node pertama, jika ≠ GOAL, node dihapus dan diganti dengan anak-anaknya. Selanjutnya keseluruhan node yang ada di Queue di-sort Ascending
3. Bila node pertama = GOAL, selesai



Lintasan yang didapat : S – A – B – C – E – Z

* + Keuntungan

1. Membutuhkan memori yang relatif kecil, karena hanya node-node pada lintasan yang aktif saja yang disimpan
2. Secara kebetulan. Metoda best first search akan menemukan solusi tanpa harus menguji lebih banyak lagi dalam ruang keadaan
   * Kelemahan
3. Algoritma akan berhenti kalau mencapai nilai optimum lokal
4. Tidak diijinkan untuk melihat satupun langkah sebelumnya.
   1. **Pencarian**

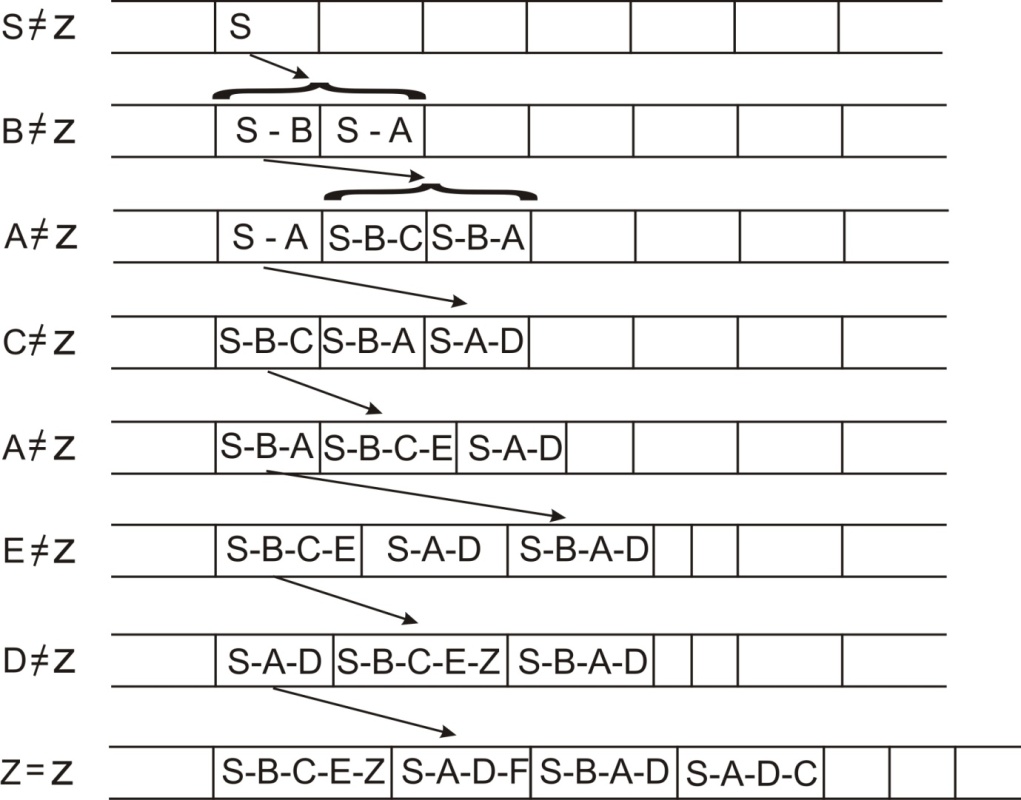
Pada pencarian sebelumnya adalah pencarian non-oprimal, maka pada sub bab ini dipresentasikan metoda optimal.

1. Branch and Bound

Metoda ini tidak memerlukan estimasi jarak tiap node menuju GOAL. Elemen- elemen pada queue bukan tiap node, melainkan lintasan parsial yang sudah tercapai.

* + Algoritma

1. Buat sebuah antrian, inisialisasi node pertama dengan Root dan tree
2. Bila lintasan parsial ≠ lintasan GOAL, maka lintasan parsial diganti dengan lintasan parsial + node child, semuanya diatur berdasarkan harga yang diurut secara Ascending
3. Bila node pertama = lintasan GOAL, selesai



* + Keuntungan

Algoritma berhenti pada nilai optimum sebenarnya (menemukan optimum global

* + Kelemahan

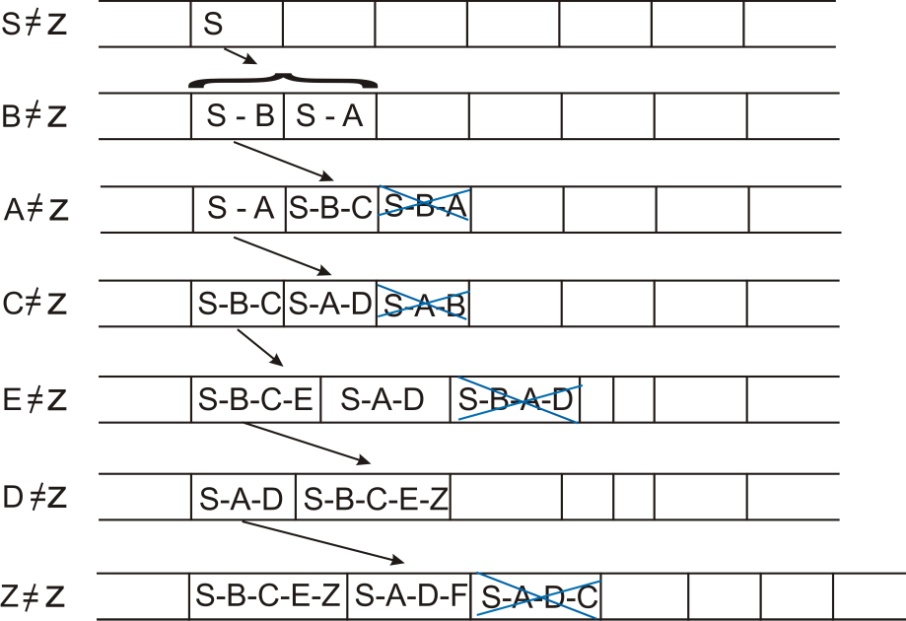
Membutuhkan memori yang cukup banyak, karena bisa jadi menyimpan semua lintasan parsial yang memungkinkan.

1. Branch and Bound dengan Dynamic Programming

Metoda ini hampir sama dengan branch and bound, metoda ini dapat mengurangi lebar tree. Hal ini dilakukan dengan cara mereduksi lintasan parsial yang menuju ke node yang sudah pernah dikunjungi sebelumnya.

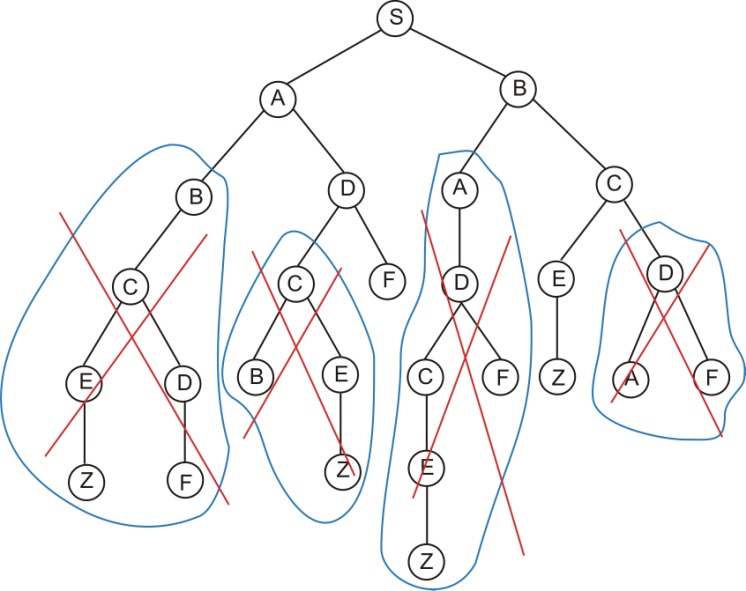
* + Algoritma

1. Buat sebuah antrian, inisialisasi node pertama dengan Root dari tree.
2. Bila lintasan parsial ≠ lintasan GOAL, jika ada lintasan parsial dengan node terakhir yang sama (dalam satu queue) maka diambil yang harganya paling minimal, sedangkan yang lebih mahal dihapus dari queue, sehingga tree akan lebih kurus.
3. Bila node pertama = lintasan GOAL, selesai



* + Keuntungan

1. Algoritma berhenti pada nilai optimum sebenarnya
2. Lebih efisien dari Branch and Bound dalam penggunaan memori dan waktu eksekusi karena ada pemotongan.
   * Kelemahan
3. Harus mengingat node terakhir dari lintasan parsial yang sudah dicapai sebelumnya.

****

Gambar 3.8 Tree untuk Metoda Branch and Bound dengan Dynamic Programming

* 1. **Game**

Saat ini game adalah topik yang menarik untuk dibahas dalam kecerdasan buatan. Game akan semakin menarik dengan kecerdasan buatan yang tinggi. Kecerdasan buatan dibutuhkan dalam sebuah game bila salah satu pemain adalah komputer atau game tersebut memberikan state-state yang berubah-ubah sesuai aksi yang ditentukan agent. Terdapat beberapa hal yang harus diperhatikan untuk menjadikan komputer sebagai salah satu pemain dalam game, yaitu :

* Cara bermain game :
  + Pertimbangkan semua kemungkinan jalan
  + Berikan nilai pada semua kemungkinan jalan
  + Jalankan pada kemungkinan yang mempunyai nilai terbaik
  + Tunggu giliran pihak lawan jalan
  + Ulangi cara diatas
* Permasalahan kunci :
  + Representasikan “bound” atau “state”
  + Buat next bound yang legal
  + Lakukan evaluasi pada posisi

1. **Fungsi Evaluasi**

Evaluation function atau static evaluator digunakan untuk mengevaluasi nilai posisi yang baik. Zero-sum assumption (kerugian seorang pemain adalah keuntungan pemain lain) memperbolehkan untuk menggunakan single evaluation function untuk mendeskripsikan nilai posisi

* + F(n) >> 0 : posisi n baik untuk saya dan jelek untuk lawan
  + F(n) << 0 : posisi n jelek untuk saya dan baik untuk lawan
  + F(n) near 0 : posisi n adalah posisi netral / seri
  + F(n) =+infinity : saya menang
  + F(n) = -infinity : lawan menang

Contoh fungsi evaluasi yang digunakan pada game :

* + Tic-tac-toe
    - F(n) = [# of 3-lengths open for me] – [# of 3-lengths open for you] dimana 3-length adalah complete row, column atau diagonal yang terisi.
  + Alan Turing’s function untuk catur
    - F(n) = w(n)/b(n) dimana w(n) – jumlah point value bidak putih dan b(n) = jumlah point value dari bidak hitam
  + Deep Blue (yang mengalahkan Gary Kasparov tahun 1997) mempunyai lebih dari 8000 features untuk evaluation function

1. **Sistem sistem pencarian pada game**

Ada dua metode mendasar dalam pencarian untuk game ini, yaitu :

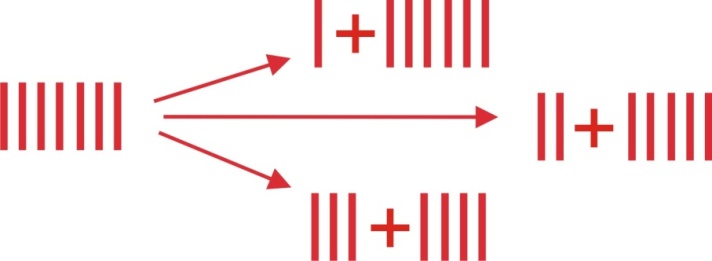
1. Metode minimax

Diuraikan pertama kali oleh John von Neumann pada tahun 1944, algoritma ini memaksimalkan posisi pemain dan meminimalkan posisi lawan.

Metode ini melakukan proses pada tiap level. Proses mencari nilai minimum, kemudian pada node parent-nya dilakukan pencarian nilai maksimum.

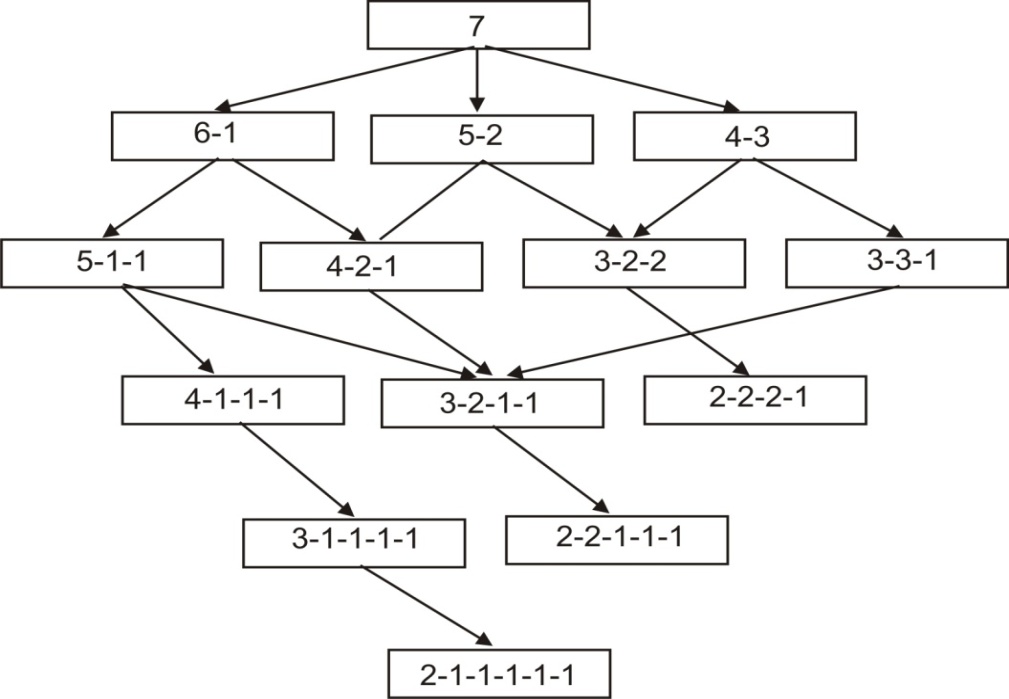
Contoh penggunaan metode ini pada game Nim, dengan aturan sebagai berikut :

* + Diawali serangkaian batang
  + Setiap pemain harus memecah serangkaian batang menjadi 2 kumpulan dimana jumlah batang di tiap kumpulan tidak boleh sama dan atau kosong



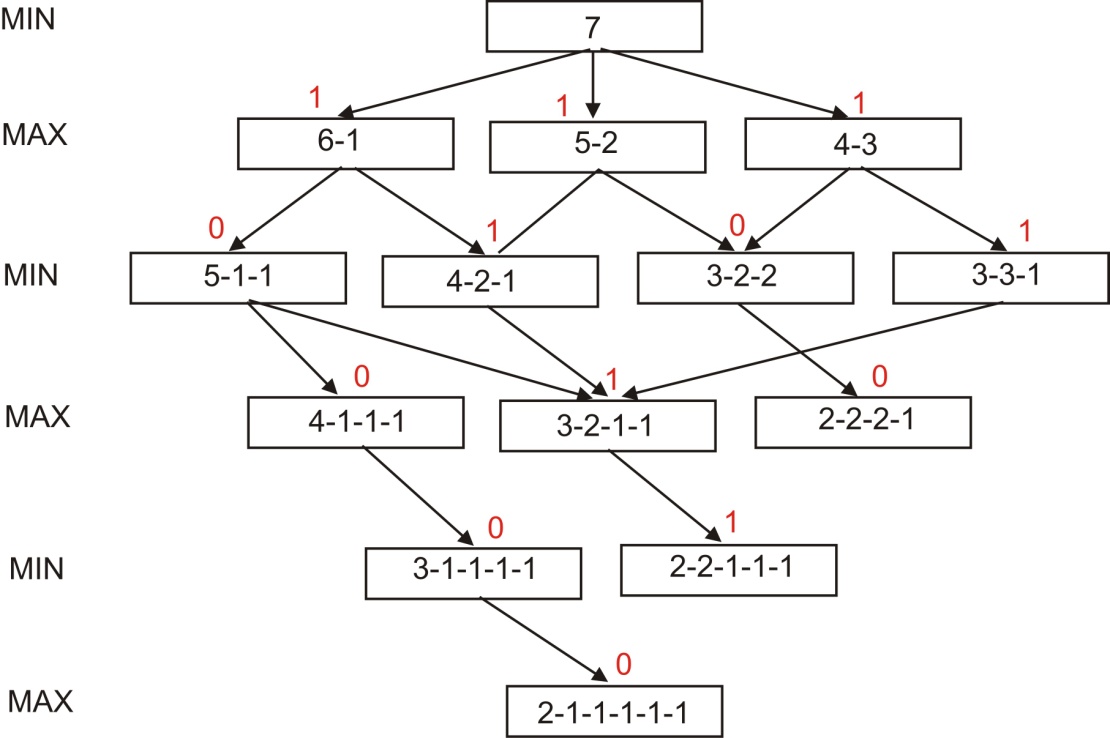
Gambar 3.9 Ilustrasi Game Nim dengan jumlah batang 7

Ilustrasi pada Gambar 3.9 menghasilkan semua kemungkinan jalan seperti yang ditunjuk pada Gambar 3.10



Gambar 3.10 kemungkinan jalan pada Game Nim dengan jumlah batang 7

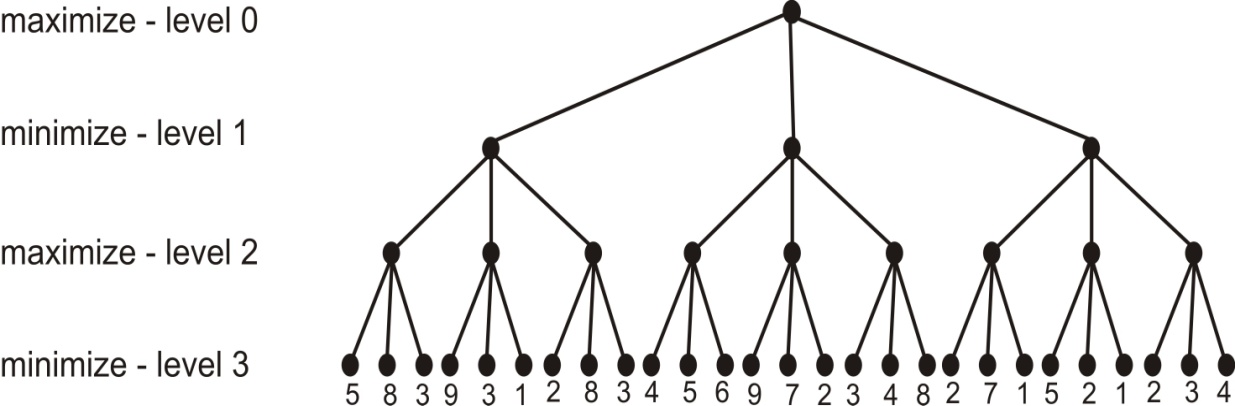
Asumsi yang diberikan MIN bermain dulu dan fungsi evaluasi 0 -> MIN menang dan 1 -> MAX menang. Langkah selanjutnya adalah memberikan nilai pada semua kemungkinan. Pemberian nilai dimulai dari simpul yang menjadi daun(leaf) dengan nilai kemenangan berdasarkan pemain yang bermain terakhir.



Gambar 3.11 Nilai pada semua kemungkinan jalan pada Game Nim dengan jumlah batang 7

1. **Alpha-Beta Prunning**

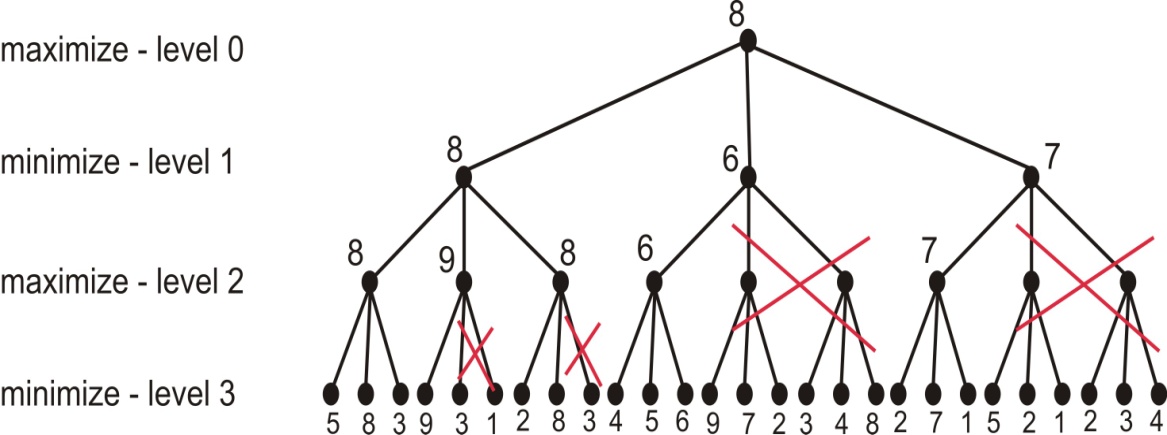
Kondisi awal pada Gambar 3.12 akan didapat hasil dengan metoda Alpha Beta Prunning seperti pada Gambar 3.13.



Gambar 3.12 Kondisi awal metoda Alpha Beta Prunning

Algoritma

1. Hampiri node pertama pada leaf dengan nilai 5, naik ke parent pada level 2 masukkan nilai 5, hampiri 8, karena 8 > 5 maka ganti parent dengan 8, hampiri 3.
2. Setelah ketiga leaf pertama terhampiri, naik lebih tinggi lagi ke level 1 masukkan nilai 8.
3. Hampiri node keempat pada leaf dengan nilai 9, naik ke parent pada level 2 masukkan nilai 9. Jika menghampiri leaf berikutnya, cari nilai yang lebih tinggi dari 9, sementara pada level 1 cari yang lebih kecil dari 8, maka leaf 3 dan 1 potong (tidak dihampiri). Demikian seterusnya hingga kita dapat hasil akhir dari tree di atas adalah Gambar 4.13.



Gambar 3.13 Hasil Alpha Beta Prunning

**Latihan**

1. Buat graph ruang keadaan untuk permasalahan petani dan bawaannya. Buat pula tree-nya
2. Buat perubahan path untuk mencari GOAL dari tree soal no. 1. Gunakan algoritma Depth First.
3. Cari algoritma pencarian lain dan buat graph serta tree-nya
4. Cari jurnal ilmiah dengan topik kecerdasan buatan yang menggunakan algoritma pencarian untuk mencari solusi permasalahannya. Buat resume algoritma pencarian tersebut.