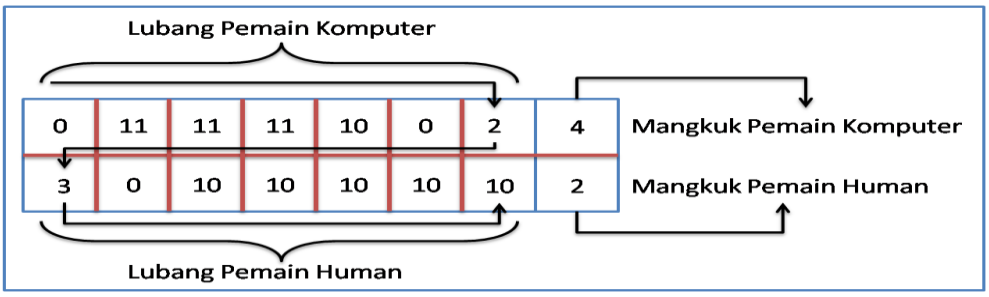
BAB 3

ANALISIS ALGORITMA

3.1 Analisis Masalah

Permainan congklak merupakan permainan yang diperlukan strategi dan kemampuan matematika untuk bisa memenangkan permainan. Umumnya congklak menggunakan papan dengan 16 lubang yang diantaranya adalah 7 lubang masing-masing untuk menyimpan biji yang berjumlah 7, sedangkan kedua lubang besar digunakan untuk menampung biji yang dihasilkan milik pemain. Penelitian ini pemain diijinkan untuk memilih jumlah lubang yang diinginkan yang akan dibatasi hingga 7 lubang agar dalam percobaan ujinya bisa dinamis dan terlihat hasil dari perbedaannya, dengan banyaknya lubang maka akan semakin lama permainan berlangsung dan akan lebih diperlukan kemampuan yang lebih dalam perhitungan matematika untuk memilih biji mana yang akan diambil, dan pada penelitian sebelumnya pun menjelaskan bahwa komputer akan sangat lama merespon jika jumlah lubang yang digunakan banyak.

Penelitian sebelumnya hanya menggunakan algoritma *Minimax*, namun pada penelitian sebelumnya memiliki kelemahan karena algoritma *Minimax* memiliki kelemahan dalam waktu eksekusi yang dibutuhkan sebanding dengan jumlah leaf-nya. Sehingga jika jumlah leaf-nya lebih besar, maka permasalahan akan semakin kombionatorik, oleh karena itu untuk mengoptimalkannya akan menggunakan algoritma *Alpha-Beta Pruning*, serta jalan permainan congklak pada penelitian sebelumnya tidak searah dengan jarum jam yang akan ditunjukan pada gambar 3.1 di bawah ini.

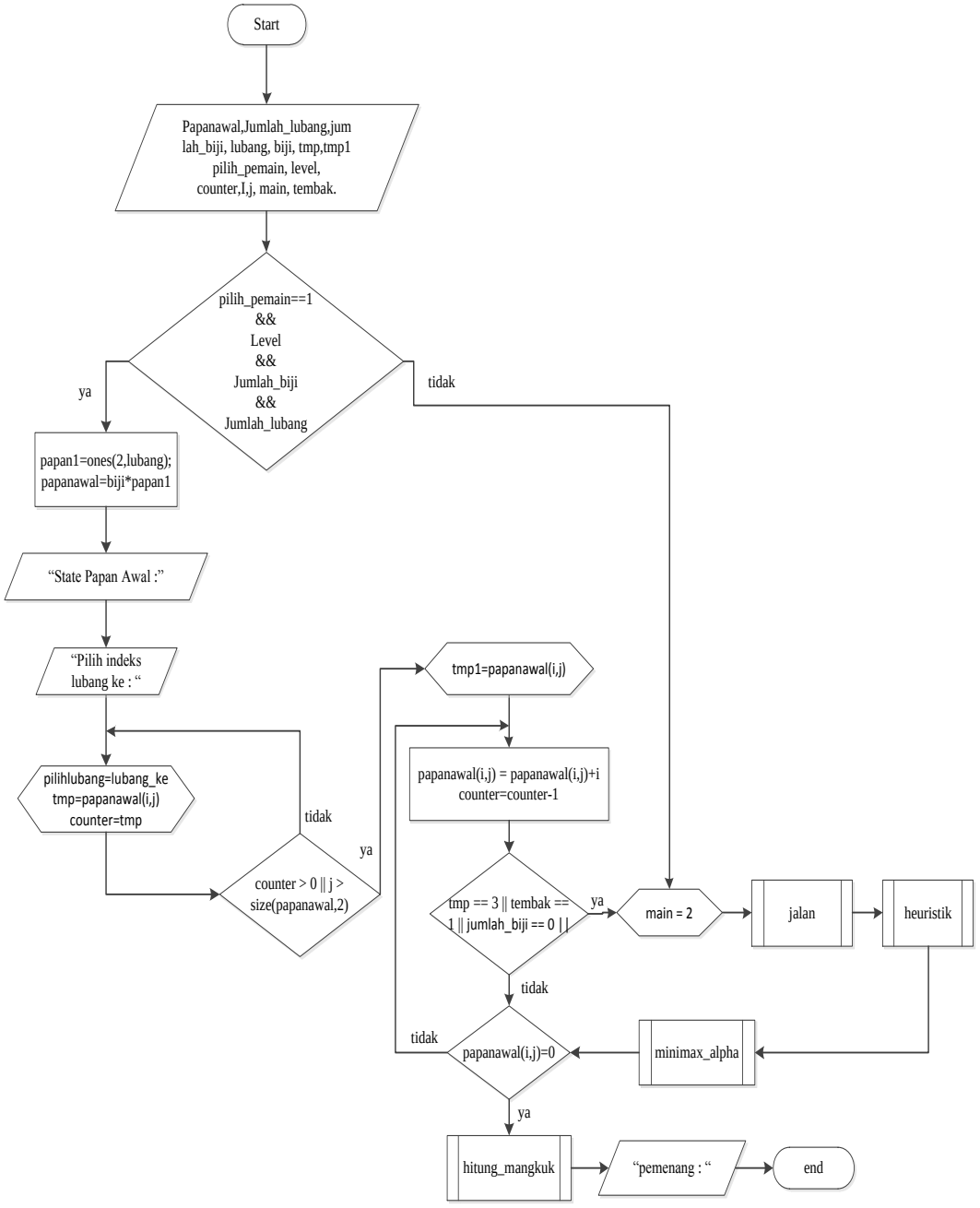


Gambar 3.1 Representasi Matriks Penelitian Sebelumnya

Uraian di atas yang telah disampaikan akan menganalisis beberapa hal yang diperlukan dalam penelitian dalam BAB 3 ini yaitu analisis algoritma *Minimax* dan algoritma *Alpha-Beta Pruning*, algoritma *Minimax* akan digunakan pada proses pencarian solusi pada permainan Congklak yang akan menentukan langkah awal yang akan memberikan kesempatan untuk mendapatkan kemenangan lebih besar, sedangkan algoritma *Alpha-Beta pruning* akan digunakan untuk proses pemotongan simpul yang tidak perlu diproses pada saat pencarian solusi, serta untuk menggambarkan keadaan secara hirarkis untuk algoritma *Minimax* dengan Optimasi *Alpha-Beta Pruning* ini akan dilakukan dengan menggambarkan sebuah pohon pencarian (*search tree*) dengan menggunakan DFS(*depth-first-search)*, dan untuk menghasilkan keputusan terbaik maka akan dilakukan sebuah fungsi evaluasi heuristik untuk membandingkan nilai-nilai heuristik pada *Minimax*.

3.2 Alur Permainan pada Congklak

Subbab ini akan menjelaskan alur dari keseluruhan permainan yang berdasarkan pada aturan-aturan permainan congklak yang akan ditunjukan dengan menggunakan *flowchart* pada gambar 3.2 di bawah ini.



Gambar 3.2 *Flowchort* Keseluruhan Permanainan pada Congklak

Gambar 3.2 di atas menjelaskan alur dari keseluruhan permainan congklak. Aturan-aturan congklak mengacu pada dakon asli jawa seperrti yang telah dijelaskan di BAB 2, namun akan sedikit diubah atau dimodifikasi aturan-aturan dari congklak yang akan dijelaskan pada subbab berikutnya. Dalam keseluruhan

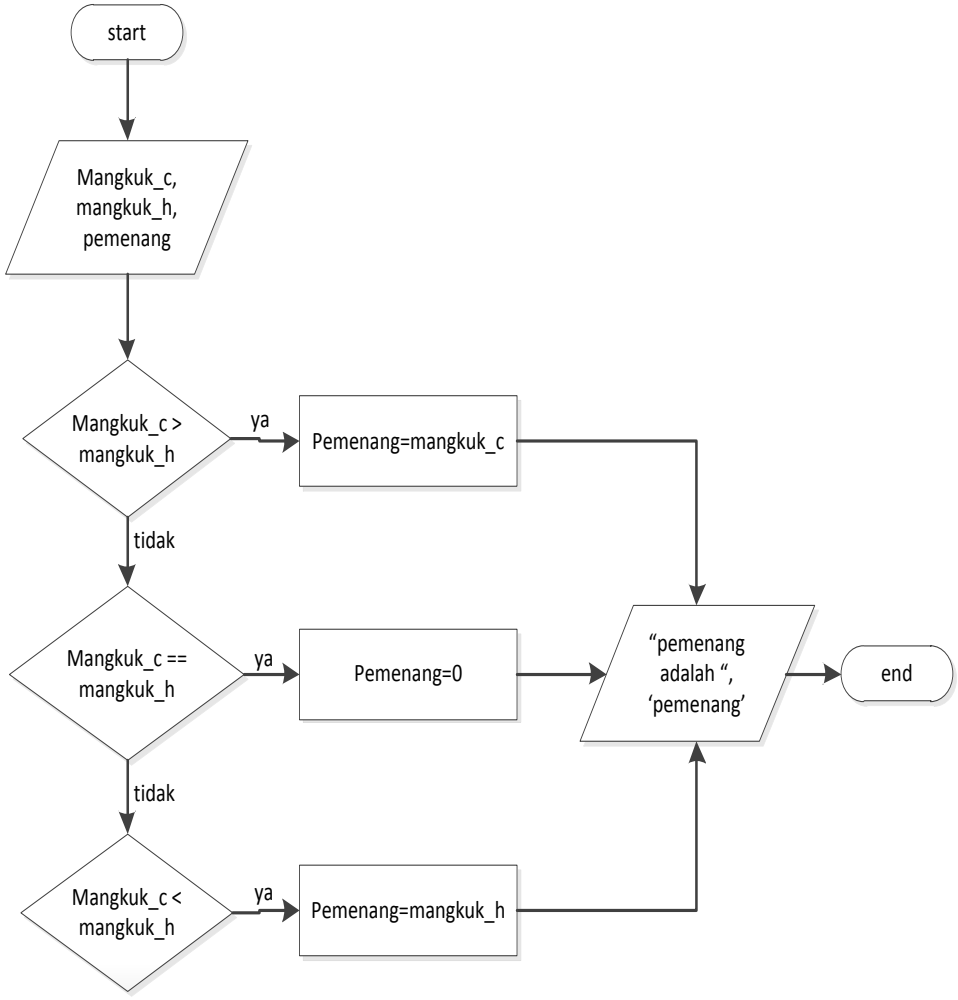
permainannya jika pemain memilih komputer bermain pertama maka akan masuk ke prosedur jalan arti jalan disini adalah membuat pohon pencarian terlebih dahulu, kemudian akan menghitung heuristiknya seteelah dilakukan heuristik maka akan masuk ke algoritma *Minimax* dengan *Alpha-Beta Pruning.* Pada algoritma *Minimax* dan *Alpha-Beta Pruning* akan dianalisis dengan kompleksitas algritma dengan menggunakan Big *O* yang akan dijelaskan pada subbab di bawah.

3.2.1 Aturan Permainanan Congklak

Aturan-aturan permainan congklak telah dijelaskan pada BAB 2, namun tidak semua cara di terapkan dalam model permainan congklak yang dikembangkan menggunakan algoritma *Minimax* dengan Optimasi *Alpha-Beta Pruning*. Congklak adalah permainan yang dimainkan oleh 2 orang. adapun asumsi aturan-aturan bermain Congklak yaitu sebagai berikut :

1. Permainan dimainkan oleh dua pemain yaitu, pemain *human* dan pemain *computer*.
2. Awal permainan :
   a. Pemain bisa memilih level yang akan dimainkan. Terdapat tiga level yaitu “Mudah” dengan dibatasi 3 kedalaman, “Sedang” dengan dibatasi 6 kedalaman, dan “Sulit” akan dibatasi hingga kedalaman 10.
   b. Pemain bisa memilih jumlah lubang yang akan dimainkan.
   c. Pemain bisa memilih siapa yang akan bermain pertama.
3. Ketika permainan dimulai pemain memilih biji pada lubang miliknya dan menjatuhkan satu persatu biji tersebut ke lubang miliknya searah jarum jam, dan menjatuhkan biji ke mangkuk miliknya.
4. Proses “Tembak”, yaitu pada proses ini jika pemain menjatuhkan biji terakhirnya di lubang miliknya yang kosong maka pemain dapat mengambil biji di lubang lawan yang sejajar dan dimasukan ke mangkuk miliknya dan jika menjatuhkan biji terakhirnya di lubang milik pemain walaupun di lubang milik lawan yang sejajar tidak terdapat biji maka biji terakhir yang dijatuhkan tersebut akan masuk ke dalam mangkuk pemain.

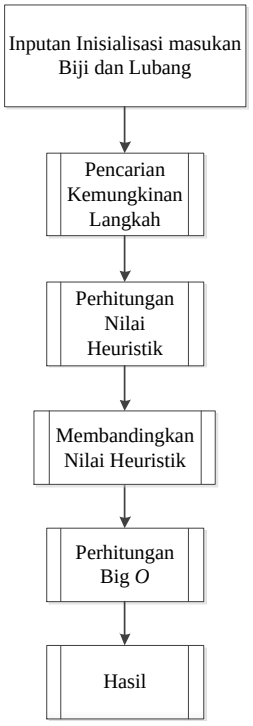
5. Akhir giliran pemain terdapat 3 kondisi yaitu :
   a. Jika biji terakhir yang dijatuhkan pemain berada pada lubang kosong miliknya dan sudah mengambil 3 kali biji maka gilirannya berakhir.
   b. Jika biji terakhir yang dijatuhkan pemain berada pada lubang kosong milik lawan maka gilirannya berakhir.
   c. Pemain sudah mengambil biji sebanyak 3 kali.
6. Akhir permainan, permainan dikatakan selesai jika tidak adalagi biji yang terdapat di lubang milik pemain *human* atau pemain *computer* dan pemain dikatakan menang jika jumlah biji yang terdapat di mangkuknya mendapatkan jumlah terbanyak, yang akan ditunjukan pada *flowchart* di bawah ini.



Gambar 3.3 *Flowchort* Hitung mangkuk

3.3 Tahapan Analisis Algoritma

Sub-bab ini akan menjelaskan tahapan-tahapan apa saja untuk analisis algoritma pada permainan congklak yang akan ditunjukan dengan *block* diagram pada gambar di bawah 3.3 ini.



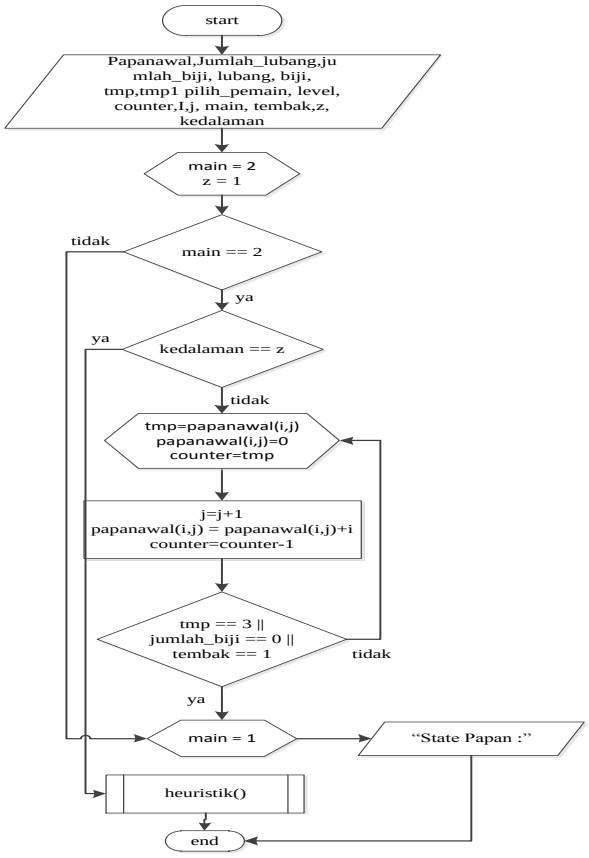
Gambar 3.4 Tahapan Analisis Algoritma pada Permainan Congklak

Gambar 3.4 di atas menjelaskan tahapan-tahapan analisis algoritma mulai dari menentukan biji dan lubang yang akan dimasukan serta level untuk menganalisis algoritma *Minimax* dan *Minimax* optimasi *Alpha-Beta Pruning* yang akan di cari kemungkinan-kemungkinan langkah dari permainan congklak dengan pohon pencarian, kemudian dari level dapat dilihat batas dari kedalaman pohon pencarian dari setiap level yang ditentukan, dari level kedalamannya dapat dihitung nilai heuristiknya pada akar dari level kedalaman, setelah itu dibandingkan nilai heuristiknya dengan menggunakan algoritma *Minimax* dan

Alpha-Beta Pruning untuk mendapatkan solusi atau langkah. Kemudian dicari perhitungan Big*O*-nya untuk mengetahui kompleksitas dari suatu algoritma, setelah menganalisis keempat tahapan tersebut kemudian di dapat sebuah hasil dari analisis yang telah dilakukan. Tahapan-tahapan di atas akan dijelaskan pada sub-bab selanjutnya.

3.3.1 Pencarian Kemungkinan-kemungkinan langkah

Pohon pencarian untuk menggambarkan semua kemungkinan posisi dan langkah-langkah yang dapat dicapai dalam sebuah *game*. Alur pohon pencariannya akan ditunjukan dengan *flowchart* di bawah ini.



Gambar 3.5 *Flowchort* Pohon Pencarian

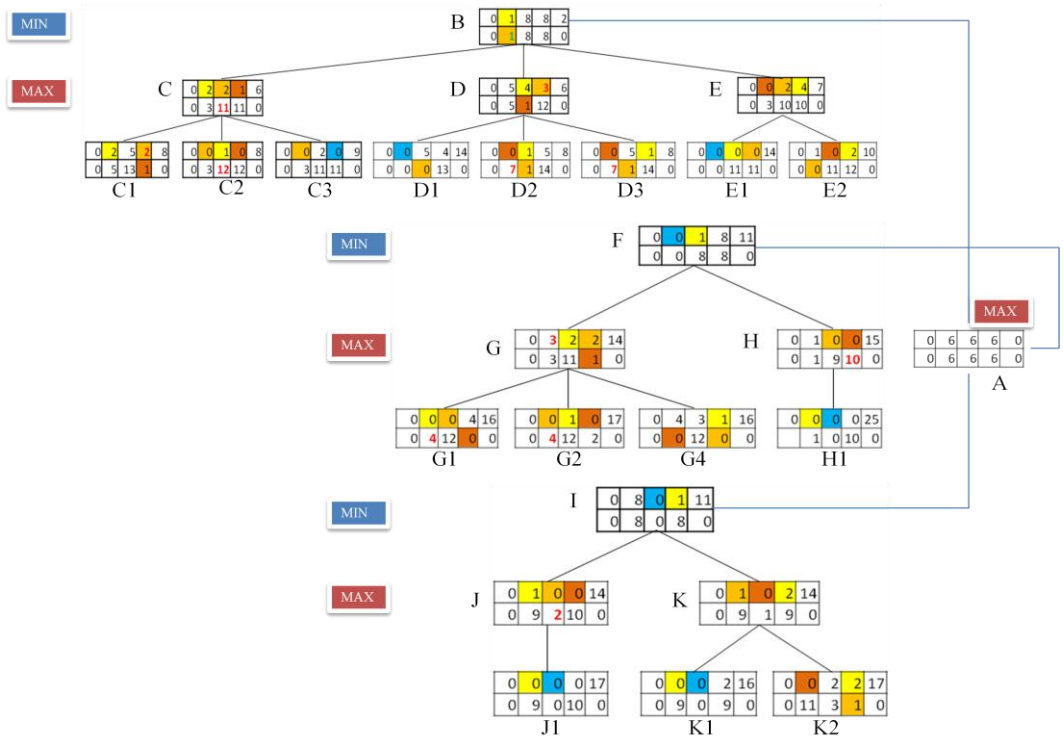
Gambar 3.5 di atas menjelaskan bagaimana pohon pencarian dibentuk, pohon pencarian akan dilakukan dengan penelusuran DFS(*depth-first-search*)

karena pada dasarnya algoritma *Minimax* merupakan perkembangan dari algoritma DFS, algoritma *Minimax* untuk melakukan pencariannya akan ditelusuri setiap simpul mulai dari simpul kiri selanjutnya ke bawah dan apabila solusinya belum ditemukan maka akan kembali ke atas kemudian membandingkan nilai yang ada pada simpul-simpul tetangganya dan seterusnya hingga batas kedalaman yang telah ditentukan hingga menemukan solusi, metode penelusuran pencarian pohon ini sama dengan DFS, pohon pencarian pada algoritma *Minimax* bertugas untuk melakukan pencarian seluruh kemungkinan langkah permainan (panjang cabang pohon ditentukan oleh level permainan). Dalam permainan congklak, setiap simpul pohon pencarian dapat berupa posisi papan congklak yang mungkin. Serta untuk Optimasi *Alpha-Beta Pruning* ini bertugas untuk memangkas simpul-simpul yang mungkin tidak perlu diproses pada saat pencarian.

Pohon pencarian hanya akan dilakukan pada pemain "Komputer" saja. Untuk simulasi permainan congklak akan disimulasikan menggunakan *state matrix* yang merupakan papan dari Congklak dimana untuk *baris pertama* merupakan *papan komputer* dan *baris kedua* merupakan *papan lawan*, untuk mangkuknya akan ada 2 kotak yaitu yang terdapat di kiri dan kanan yang menyatakan jumlah biji dari mangkuk pemain, serta akan menggunakan lubang yang bejumlah 3, 4, 5, 6, dan 7, serta biji 3, 4, 5, 6, dan 7, untuk analisisnya hanya akan dilakukan pada permainan congklak dengan 3 lubang dan 6 biji pada level "Mudah" dengan 3 kedalaman yang akan dituangkan ke dalam gambar dan tabel.

3.3.1.1 Analisis Pohon Pencarian 3 lubang 6 Biji

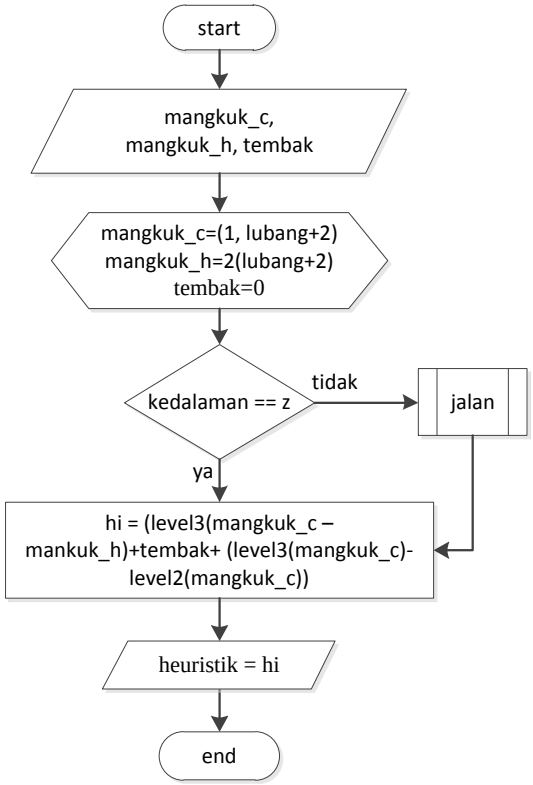
Pohon pencarian pada papan Congklak dengan 3 lubang dan 6 biji yang akan digambarkan di bawah ini dengan level mudah dengan 3 kedalaman.



Gambar 3.6 Pohon Pencarian dengan 3 lubang 6 biji pada 3 kedalaman

Gambar 3.6 di atas merupakan pohon pencarian untuk *Minimax* pada congklak untuk jalan komputer. Warna-warna yang ada di state matriks keterangan akan di jelaskan di bawah ini.

3.3.1.2 Perhitungan Nilai Heuristik



Gambar 3.7 *Flowchart* Perhitungan Nilai Heuristik

Gambar 3.7 menjelaskan alur untuk menghitung heuristiknya yang di mulai dengan meload pohon pencarian yang sudah di *record* dan pada level 3 di dalam programnya akan di cek jumlah biji pada pemain *human* dan pemain komputer kemudian akan di hitung sesuai rumus.

Sebuah Fungsi Evaluasi Heuristik dapat ditentukan dengan melihat ke dalam beberapa faktor, proses menghitung fungsi evaluasi heuristik dilakukan dengan melihat 3 faktor penting dalam congklak yaitu:

1. Jumlah biji yang terdapat dalam mangkuk pemain komputer dan lawan.
2. Kondisi kemungkinan tembak dalam satu state, dan jika ada kondisi tembak maka state tersebut bernilai t=1 dan jika tidak ada t=0.

3. Dan biji komputer sebelum permainan berjalan yaitu biji komputer yang ada dalam mangkuk pada level 3 dikurangi dengan biji komputer pada level 2.

Rumus fungsi evaluasi heuristik pada Minimax untuk permainan congklak mengacu pada 3 faktor di atas yang merupakan total dari beberapa heuristik, rumusnya yaitu sebagai berikut :

$$H = \sum_{i=1}^{n} hi$$

Dimana,

1. H= Total jumlah nilai heuristik permainan congklak
2. hi= Jumlah nilai heuristik dari kondisi tertentu
3. n= Total/batas heuristik

Proses menghitung heuristik akan berjalan dengan baik setelah dilakukan formulasi heuristik, formulasi heuristik adalah mengubah heuristik ke dalam sebuah fungsi yang dapat dibaca oleh komputer, berikut adalah rumus-rumus heuristik dari beberapa kondisi.

1. Total jumlah biji di mangkuk komputer pada akar kedalaman

   h1 = mangkuk_c – mangkuk_h

   Ket :

   h1 = Jumlah biji dalam mangkuk

   Mangkuk_c = jumlah biji di mangkuk komputer

   Mangkuk_h = jumlah biji di mangkuk *human*

2. Kondisi Tembak

   h2 = Kondisi tembak

   Ket :

   h2 adalah kondisi tembak jika ada kondisi tembak maka bernilai 1 dan jika tidak bernilai 0

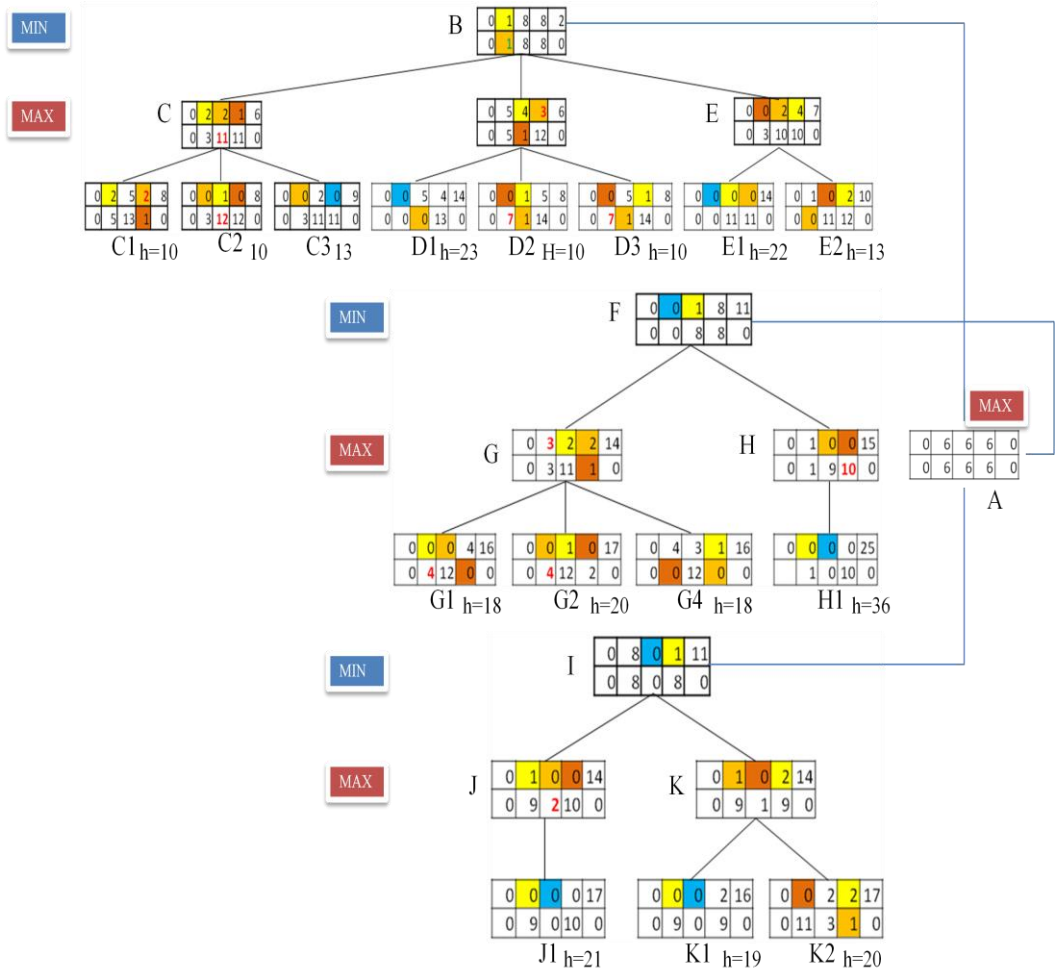
3. Jumlah mangkuk komputer pada level sebelum akar kedalaman

h3 = levelhi(mangkuk_c – mangkuk_c)

h3 = jumlah biji di mangkuk komputer

levelhi = level sebelum akar kedalaman

Dalam pohon pencarian pada penjelasan sebelumnya pada subbab 3.3.1.1 akan dihitung nilai heuristik yang akan dihitung pada akar kedalaman yang akan ditunjukan pada gambar di bawah ini.



Gambar 3.8 Pohon Pencarian dengan Nilai Heuristik

Gambar 3.8 di atas memperlihatkan pohon pencarian yang telah diperoleh nilai heuristiknya, untuk cara menghitungnya akan dimisalkan dengan setiap simpul dimisalkan dengan diberikan abjad.

Adapun cara perhitungan evaluasi heuritiknya yaitu sebagai berikut.

A. 

Gambar di atas merupakan matriks *state* untuk akar C1, sebelum dihitung untuk akar C1 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar C1 terdapat kondisi Tembak atau tidak jika ada maka bernilai "1" jika tidak maka bernilai "0", kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

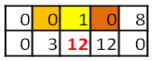
Cara menghitung heuristik akar C1 :

h1 = 8-0 =8

h2 = 0

h3 = 8-6 = 2

Nilai Heuristik C1 adalah Hi = h1+h2+h3 = 10.

B. 

Gambar di atas merupakan matriks *state* untuk akar C2, sebelum dihitung untuk akar C2 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar C2 terdapat kondisi Tembak atau tidak jika ada maka bernilai "1" jika tidak maka bernilai "0", kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek

maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

Cara menghitung heuristik akar C1 :

h1 = 8-0 =8

h2 = 1

h3 = 8-6 = 2

Nilai Heuristik C2 adalah Hi = h1+h2+h3 = 11.

C.

0	0	2	0	9
0	3	11	11	0

Gambar di atas merupakan matriks *state* untuk akar C3, sebelum dihitung untuk akar C3 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar C3 terdapat kondisi Tembak atau tidak jika ada maka bernilai “1” jika tidak maka bernilai “0”, kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

Cara menghitung heuristik akar C1 :

h1 = 9-0 =9

h2 = 1

h3 = 9-6 = 3

Nilai Heuristik C3 adalah Hi = h1+h2+h3 = 13.

D.

0	0	5	4	14
0	0	0	13	0

Gambar di atas merupakan matriks *state* untuk akar D3, sebelum dihitung untuk akar D3 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah

biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar D3 terdapat kondisi Tembak atau tidak jika ada maka bernilai “1” jika tidak maka bernilai “0”, kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

Cara menghitung heuristik akar D1 :

h1 = 14-0 =14

h2 = 1

h3 = 14-6 = 8

Nilai Heuristik C3 adalah Hi = h1+h2+h3 = 23.

E.

0	0	1	5	8
0	7	1	14	0

Gambar di atas merupakan matriks *state* untuk akar D2, sebelum dihitung untuk akar D2 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar D2 terdapat kondisi Tembak atau tidak jika ada maka bernilai “1” jika tidak maka bernilai “0”, kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

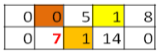
Cara menghitung heuristik akar D2 :

h1 = 8-0 =8

h2 = 0

h3 = 8-6 = 2

Nilai Heuristik C3 adalah Hi = h1+h2+h3 = 10.

F. 

Gambar di atas merupakan matriks *state* untuk akar D3, sebelum dihitung untuk akar D3 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar D3 terdapat kondisi Tembak atau tidak jika ada maka bernilai "1" jika tidak maka bernilai "0", kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

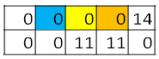
Cara menghitung heuristik akar D3 :

h1 = 8-0 =0

h2 = 0

h3 = 8-6 =2

Nilai Heuristik D3 adalah Hi = h1+h2+h3 = 10.

G. 

Gambar di atas merupakan matriks *state* untuk akar E1, sebelum dihitung untuk akar E1 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar E1terdapat kondisi Tembak atau tidak jika ada maka bernilai "1" jika tidak maka bernilai "0", kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

Cara menghitung heuristik akar E1 :

h1 = 14-0 =0

h2 = 1

h3 = 14-7 = 7

Nilai Heuristik E1 adalah Hi = h1+h2+h3 = 22.

H.

0	1	0	2	10
0	0	11	12	0

Gambar di atas merupakan matriks *state* untuk akar E2, sebelum dihitung untuk akar E2 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar E2 terdapat kondisi Tembak atau tidak jika ada maka bernilai "1" jika tidak maka bernilai "0", kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

Cara menghitung heuristik akar E2 :

h1 = 10-0 =10

h2 = 0

h3 = 10-7 = 3

Nilai Heuristik E2 adalah Hi = h1+h2+h3 = 13.

I.

0	0	0	4	16
0	4	12	0	0

Gambar di atas merupakan matriks *state* untuk akar G1, sebelum dihitung untuk akar G1 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar G1 terdapat kondisi Tembak atau tidak jika ada maka bernilai "1" jika tidak maka bernilai "0", kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

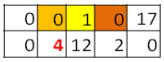
Cara menghitung heuristik akar G1 :

h1 = 16-0 =16

h2 = 0

h3 = 16-14 = 3

Nilai Heuristik G1 adalah Hi = h1+h2+h3 = 18.

J. 

Gambar di atas merupakan matriks *state* untuk akar G2, sebelum dihitung untuk akar G2 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar G2 terdapat kondisi Tembak atau tidak jika ada maka bernilai "1" jika tidak maka bernilai "0", kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

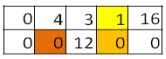
Cara menghitung heuristik akar G2 :

h1 = 17-0 =17

h2 = 0

h3 = 17-14 = 3

Nilai Heuristik G2 adalah Hi = h1+h2+h3 = 20.

K. 

Gambar di atas merupakan matriks *state* untuk akar G3, sebelum dihitung untuk akar G3 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar G3 terdapat kondisi Tembak atau tidak jika ada maka bernilai "1" jika tidak maka bernilai "0", kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek

maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

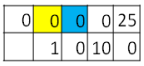
Cara menghitung heuristik akar G3 :

h1 = 16-0 =16

h2 = 0

h3 = 16-14 = 2

Nilai Heuristik G1 adalah Hi = h1+h2+h3 = 18.

L. 

Gambar di atas merupakan matriks *state* untuk akar H1, sebelum dihitung untuk akar H1 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar H1 terdapat kondisi Tembak atau tidak jika ada maka bernilai "1" jika tidak maka bernilai "0", kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

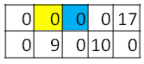
Cara menghitung heuristik akar H1 :

h1 = 25-0 =25

h2 = 1

h3 = 25-15 = 0

Nilai Heuristik H1adalah Hi = h1+h2+h3 = 36.

M. 

Gambar di atas merupakan matriks *state* untuk akar J1, sebelum dihitung untuk akar J1 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar

J1 terdapat kondisi Tembak atau tidak jika ada maka bernilai “1” jika tidak maka bernilai “0”, kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

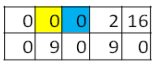
Cara menghitung heuristik akar J1 :

h1 = 17-0 =17

h2 = 1

h3 = 17-14 = 3

Nilai Heuristik J1 adalah Hi = h1+h2+h3 = 21.

N. 

Gambar di atas merupakan matriks *state* untuk akar K1, sebelum dihitung untuk akar K1 akan dilihat terlebih dahulu jumlah biji yang ada pada mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar K1 terdapat kondisi Tembak atau tidak jika ada maka bernilai “1” jika tidak maka bernilai “0”, kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

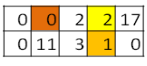
Cara menghitung heuristik akar K1 :

h1 = 16-0 =16

h2 = 1

h3 = 16-14 = 2

Nilai Heuristik K1 adalah Hi = h1+h2+h3 = 19.

O. 

Gambar di atas merupakan matriks *state* untuk akar K2, sebelum dihitung untuk akar K2 akan dilihat terlebih dahulu jumlah biji yang ada pada

mangkuk komputer kemudian setelah mendapatkan jumlah biji dari mangkuk Komputer, mangkuk lawan di cek jumlah bijinya, setelah mengecek jumlah biji yang terdapat di mangkuk Komputer dan lawan maka di cek apakah akar K2 terdapat kondisi Tembak atau tidak jika ada maka bernilai "1" jika tidak maka bernilai "0", kemudian di cek jumlah mangkuk komputer pada akar kedalaman di kurangi dengan *parent*-nya atau level seelumnya. Setelah di cek maka tahap selanjutnya adalah menghitung heuristiknya, caranya adalah seperti berikut :

Cara menghitung heuristik akar K2:

h1 = 17-0 =17

h2 = 0

h3 = 17-14 = 3

Nilai Heuristik K2 adalah Hi = h1+h2+h3 = 20.

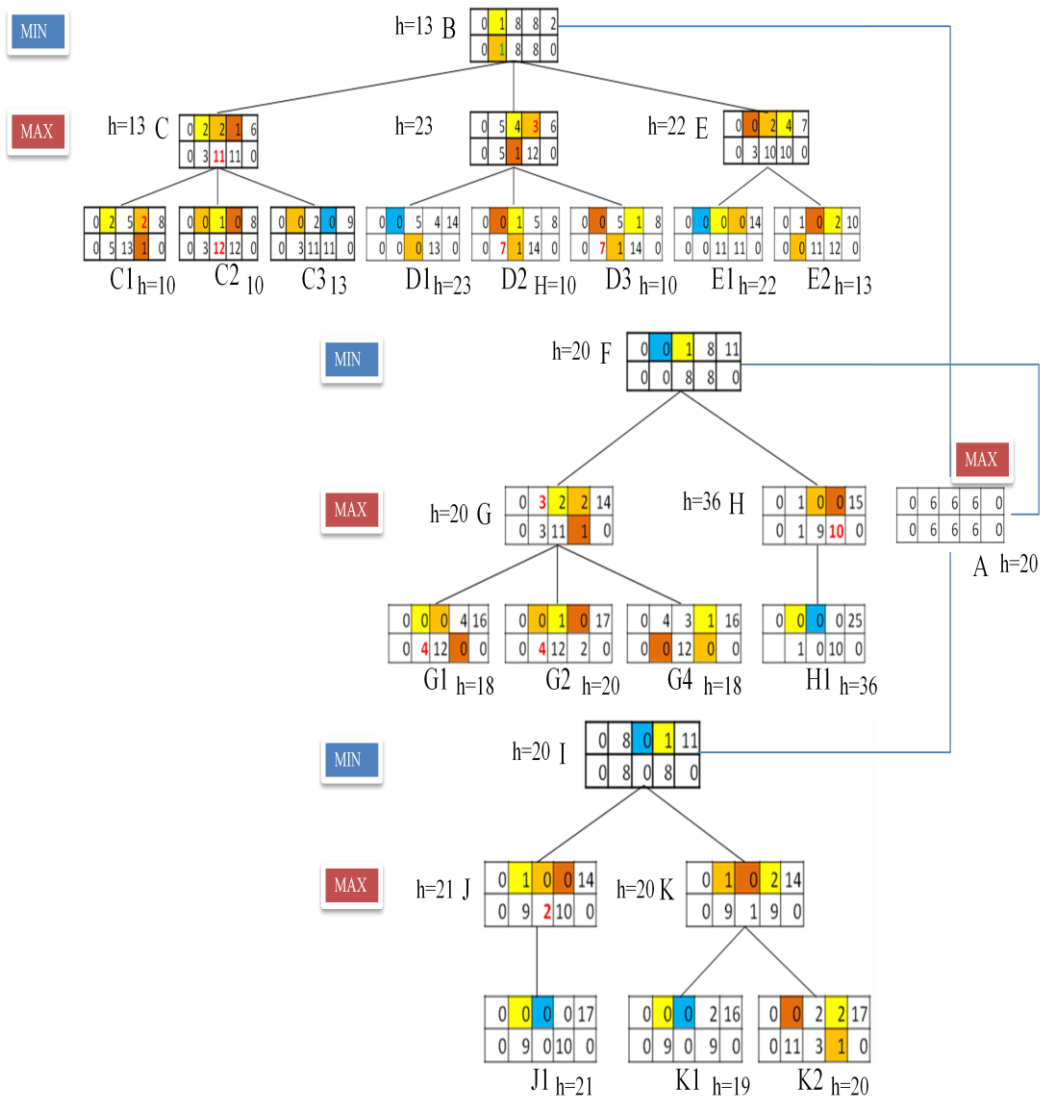
3.3.1.3 Membandingkan Nilai Heuristik

Setelah menghitung heuristik maka tahap selanjutnya adalah membandingkan nilai-nilai heuristiknya pada algoritma *Minimax* dan *Alpha-Beta Pruning* hingga mendapatkan solusi, solusi yang di ambil oleh komputer adalah bukan pada level terakhir tapi pada level 1 karena pada analisis pohon pencariannya hanya untuk kemungkinan jalan komputer saja sampai kedalaman level yang dipilih oleh *user* ketika awal permainan. Karena untuk permainan congklak *goal state* yang diperoleh adalah dilihat dari banyaknya biji dalam mangkuk maka dibuat semua kemungkinan jalan untuk komputer yang nantinya akan menghasilkan biji lebih banyak dari lawan.

1. **Membandingkan Nilai Heuristik pada Algoritma *Minimax***

   Membandingkan nilai heuristik pada *minimax* akan dilakukan sesuai dengan alur algoritmanya yaitu diasumsikan *Max* akan selalu ada pada Komputer dan *Min* akan diasumsikan pada pemain *Human*, penelusurannya akan dilakukan secara DFS yaitu dimulai dari atas ke kiri kemudian ke bawah

hingga ke akar kemudian dibandingkan dan dilihat nilai heuristiknya kemudian naik ke atas kemudian di telusuri hingga ke kanan sampai mendapatkan solusi. Perbandingannya akan ditunjukan pada gambar 3.8 di bawah ini



Gambar 3.9 Membandingkan Nilai Heuristik pada Minimax

Gambar 3.9 akan membandingkan nilai heuristik, pertama penelusuran akan ditelusuri pada level 0 atau Max, kemudian ditelusuri ke kiri pada level 1 atau Min kemudian di telusuri pada level 2 atau Max,

kemudian pada level 3 yaitu C1 akan dievaluasi nilai heuristiknya yaitu h=10 setelah evaluasi penelusuran akan kembali ke atas kemudian disimpan nilai heuristiknya pada *parent*-nya yaitu level 2, kemudian dievaluasi nilai heuristiknya pada level 3 yaitu C2, nilai dari C2 yaitu h=10, setelah itu nilai heuristik dari C2 dibandingkan dengan nilai heuristik yang di dapat pada level 2 apabila C2 lebih besar dari pada *parent*-nya maka nilai heuristik pada level 2 diganti menjadi nilai heuristic yang di dapat dari C2 karena pada level 2 merupakan Max, karena Max akan selalu bernilai lebih besar ”>”, dan apabila tidak maka nilai di *parent*-nya akan tetap dan tidak diubah, kemudian dibandingkan lagi pada simpul sebelahnya dan seterusnya, hingga mendapatkan solusi pada level 1 yaitu “F”, yang akan ditunjukan pada tabel 3.1 di bawah ini. Gambar 3.8 merupakan solusi untuk tabel dengan No.1 pada tabel 3.1.

Tabel 3.1 Hasil Permainan Congklak Algoritma *Minimax* 3 lubang dan 6 biji

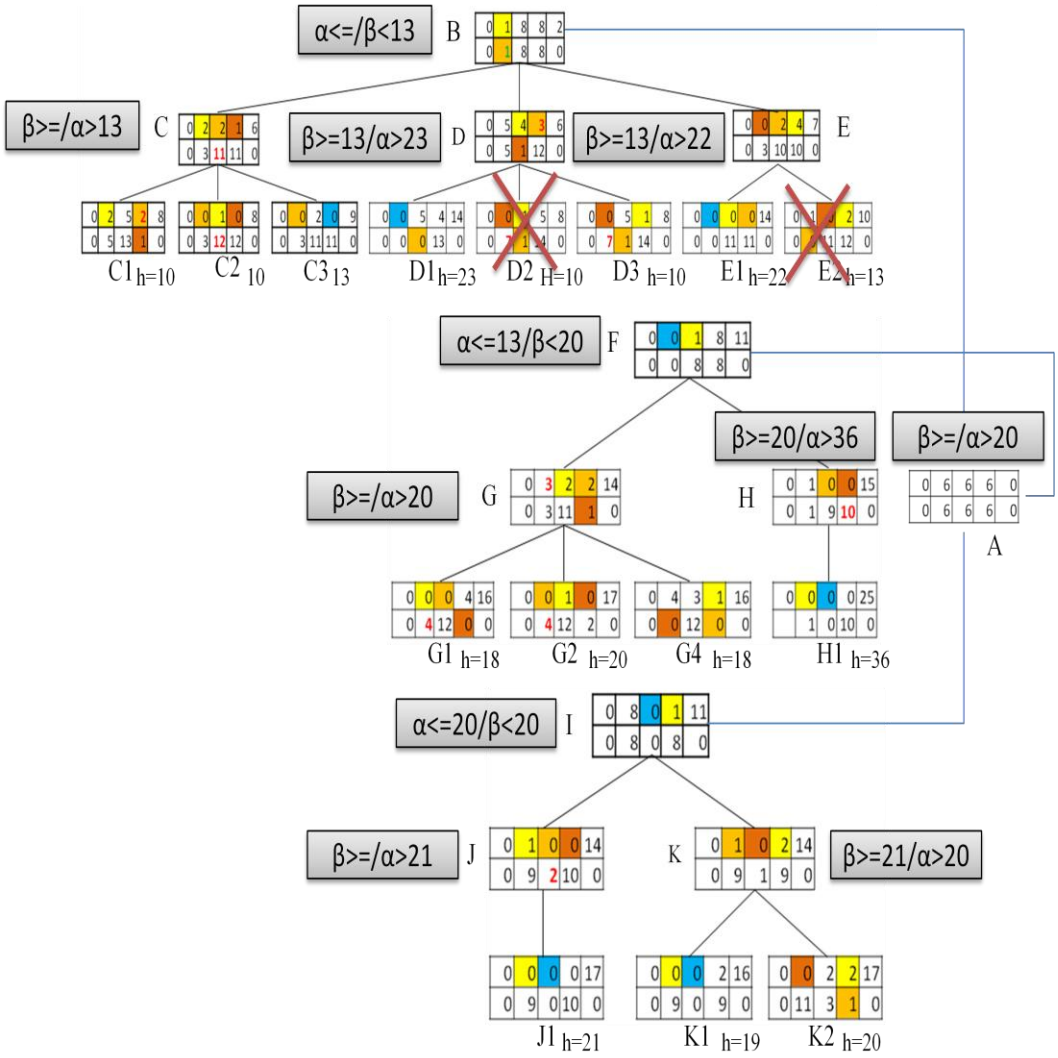
No	Kasus Congklak Lubang 3 Biji 6 pada Level Mudah							
	Pemain	State Matrik	∑ pengambilan biji	Total biji di mangkuk	∑ langkah algoritma	∑Langkah Dasar	Nilai Heuristik
1	komputer	0 0 1 8 11 0 0 8 8 0	2x	11	50	30	20
	human	0 0 4 11 11 3 3 1 **3** 0	3x	3	-	-	-
2	komputer	0 0 0 12 17 3 0 0 4 0	2x	17	28	16	19
	human	0 **1** 0 12 17 4 1 1 0 0	1x	4	-	-	-
3	komputer	0 0 0 12 19 4 1 0 0 0	1x	19	10	6	24
	human	0 0 0 12 19 5 0 0 0 0	1x	5	-	-	-
4	komputer	0 0 2 0 22	3x	22	4	4	24

		5 2 2 3							
	human	0 0 2 0 7 5 0 0	22	2x	7	-	-	-	

No	Kasus Congklak Lubang 3 Biji 6 pada Level Mudah							
	Pemain	State Matrik	∑ pengambilan biji	Total biji di mangkuk	∑ langkah algoritma	∑Langkah Dasar	Nilai Heuristik
5	komputer	0 0 0 0 24 7 5 0 0 0	2x	24	2	2	19
	human	0 1 1 0 24 10 0 0 0 0	2x	10	-	-	-
6	komputer	0 0 0 0 26 10 0 0 0 0	1x	26	4	4	16

2. Membandingkan Nilai Heuristik pada Algoritma *Minimax* dengan Optimasi *Alpha-Beta Pruning*

Membandingkan nilai heuristik pada *Alpha-Beta Pruning* akan dilakukan sesuai dengan alur algoritmanya yaitu diasumsikan *Max* akan selalu ada pada Komputer dan *Max* akan selalu bernilai "α" karena *alpha* akan selalu bernilai lebih besar, *Min* akan diasumsikan pada pemain *Human* dan akan selalu bernilai "β", penelusurannya akan dilakukan dimulai dari atas ke kiri kemudian ke bawah hingga ke akar kemudian dibandingkan dan dilihat nilai heuristiknya kemudian naik ke atas kemudian di telusuri hingga ke kanan sampai mendapatkan solusi. Perbandingannya akan ditunjukan pada gambar di bawah ini



Gambar 3.10 Membandingkan Nilai Heuristik pada Minimax Optimasi Alpha-Beta Pruning

Gambar 3.10 akan membandingkan nilai heuristik, pertama penelusuran akan ditelusuri pada level 0 atau ≥β/α>, kemudian ditelusuri ke kiri pada level 1 atau ≤α/<β kemudian di telusuri pada level 2 atau ≥β/α>, kemudian pada level 3 yaitu C1 akan dievaluasi nilai heuristiknya yaitu h=8, kemudian nilai tersebut akan dibandingkan dengan *parent*-nya yaitu β apabila nilai tersebut lebih besar sama dengan beta maka simpul berikutnya akan di *pruning* apabila tidak maka akan di bandingkan dengan alpha apakah lebih

besar kalau iya nilai heuristiknya akan diganti dengan nilai dari akar C2 apabila tidak maka *alpha* di *parent*-nya tidak akan diganti, kemudian dibandingkan lagi pada simpul sebelahnya dan seterusnya, hingga mendapatkan solusi pada level 1 yaitu "F", hasil untuk *alpha-beta pruning* tidak merubah solusi, karena dilihat dari hasil *minimax* yaitu "F" dan hasil tersebut sama seperti *alpha-beta pruning*, perbedaanya adalah dalam langkah penelusurannya yang akan diperlihatkan pada tabel 3.2 dibawah ini. Gambar 3.9 merupakan solusi untuk tabel dengan No.1 pada tabel 3.2.

Tabel 3.2 Analisis Permainan Congklak Algoritma *Minimax* dengan optimasi *Alpha-Beta Pruning* 3 lubang dan 6 biji

No	Kasus Congklak Lubang 3 Biiji 6 pada Level Mudah						
	Pemain	State Matrik	∑ pengambilan biji	Total biji di mangkuk	∑ langkah algoritma	∑Langkah Dasar	Nilai Heuristik
1	komputer	0 0 1 8 11 0 0 8 8 0	2x	11	40	22	20
	human	0 0 4 11 11 3 3 1 **3** 0	3x	3	-		-
2	komputer	0 0 0 12 17 3 0 0 4 0	2x	17	18	10	19
	human	0 **1** 0 12 17 4 1 1 0 0	1x	4	-		-
3	komputer	0 0 0 12 19 4 1 0 0 0	1x	19	10	6	24
	human	0 0 0 12 19 5 0 0 0 0	1x	5	-		-
4	komputer	0 0 2 0 22 5 2 2 3	3x	22	4	3	24
	human	0 0 2 0 22 7 5 0 0	2x	7	-		-

No	Kasus Congklak Lubang 3 Biiji 6 pada Level Mudah						
	Pemain	State Matrik	∑ pengambilan biji	Total biji di mangkuk	∑ langkah algoritma	∑Langkah Dasar	Nilai Heuristik
5	komputer	0 0 0 0 24 7 5 0 0 0	2x	24	2	2	19
	human	0 1 1 0 24 10 0 0 0 0	2x	10	-		-
6	komputer	0 0 0 0 26 10 0 0 0 0	1x	26	4	4	16

3.3.1.4 Menghitung Kompleksitas Algoritma

Sebuah algoritma tidak saja harus benar tetapi juga harus mangkus(efisien), algoritma yang mangkus adalah algoritma yang meminimumkan penggunaan *space(*ruang) dan waktu. Selanjutnya, untuk mendapatkan algoritma yang efisien serta mendapatkan rumusan matematika sebagai ukuran kerumitan (kompleksitas) maka untuk analaisis algoritma akan menggunakan notasi O (big O) pada algoritma *Minimax* dan algoritma *Minimax* dengan di optimasi *Alpha-Beta Pruning*.

Minimax akan mengevaluasi seluruh *node* yang ada pada pohon pencarian satu persatu hingga selesai, sedangkan jumlah *node* yang akan dievaluasi tidaklah sedikit. Berikut adalah tabel jumlah *node* yang akan dievaluasi menurut tingkat kedalamannya, dimana *branching factor b* adalah jumlah pergerakan maksimum masing-masing *node* dan *depth d* adalah kedalaman pohon pencarian[15].

Tabel 3.3 Hubungan Kedalaman dan Jumlah *Node*

Kedalaman	**Jumlah *node* $O(b^d)$**
1	3
2	9

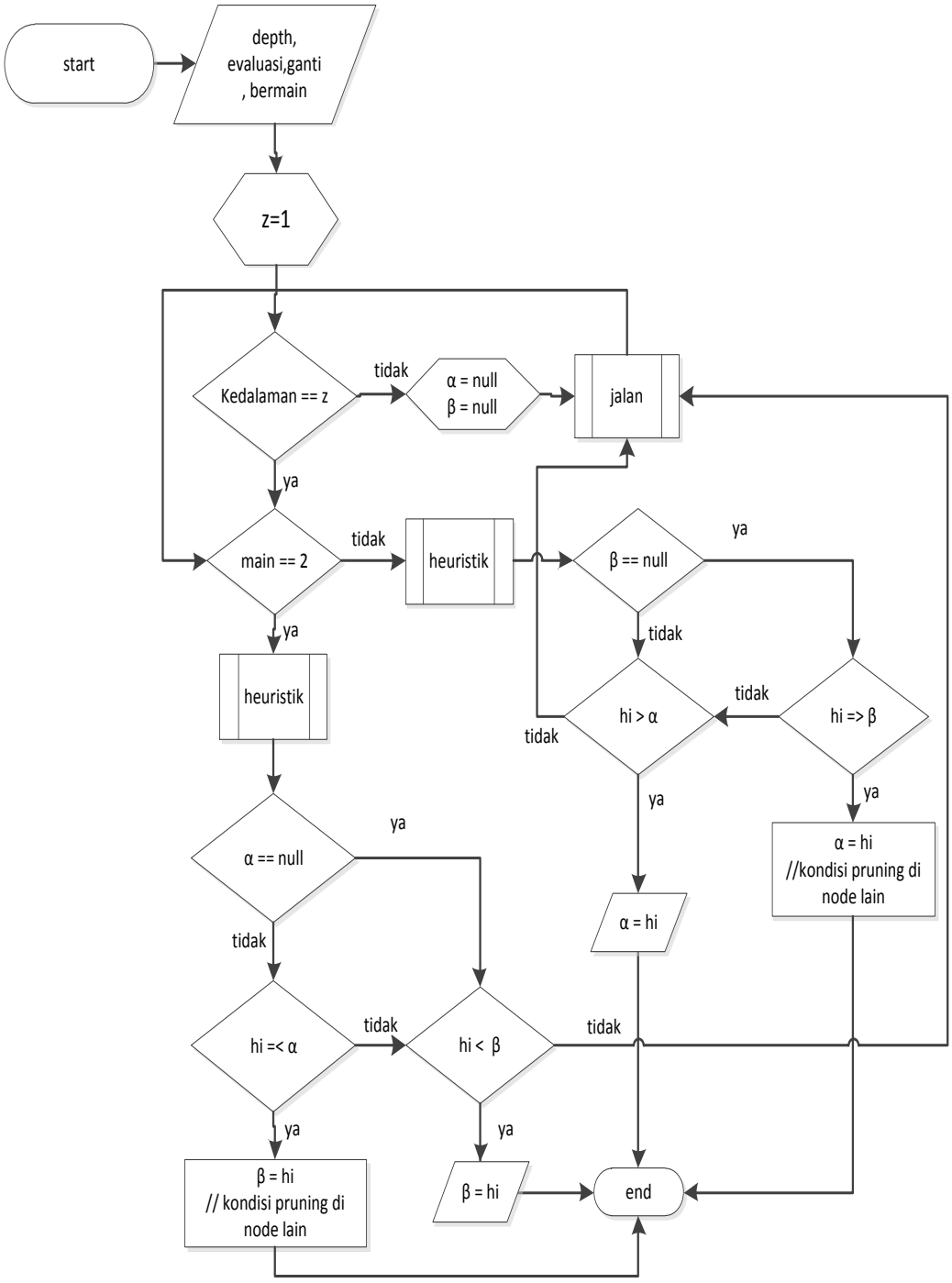
3	81

Ukuran papan permainan Congklak adalah lubang x lubang yang berarti berjumlah n lubang kotak, lubang disini tergantung dari *user* yang akan memilih berapa lubang yang diinginkan. Dengan demikian jumlah pergerakan maksimum yang dimiliki masing-masing *node* adalah n lubang, sebab Congklak tidak memiliki *illegal move* seperti halnya permainan Catur ataupun *Othello*, yang dimaksud dengan *illegal move* adalah gerakan ilegal atau batasan pergerakan pemain, misalnya kuda hanya boleh melangkah sesuai dengan huruf L pada catur. Jumlah pergerakan maksimum ini akan dijadikan percabangan pada pohon pencarian.

Dengan adanya *Alpha-Beta Pruning* diharapkan waktu pencarian akan jauh lebih singkat karena tujuan utama dari algoritma ini adalah mengabaikan *subtree* atau percabangan yang tidak mempengaruhi hasil akhir. Berikut beberapa ketentuan dalam algoritma *Alpha-Beta Pruning* :

1. Alpha (α) merupakan nilai batas bawah maksimum atau nilai terbaik sementara pada max. Alpha digunakan pada *node* min dan diset pada *node* max.
2. Beta (β) merupakan nilai batas atas minimum atau nilai terbaik sementara pada min. Beta digunakan pada *node* max dan diset pada *node* min.
3. Max bertujuan untuk memaksimalkan nilai dengan cara memilih *node* anak dengan nilai paling besar. Nilai awal max adalah $-\infty$ dan akan bertambah seiring dengan pencarian.
4. Min bertujuan untuk meminimalkan nilai dengan cara memilih *node* anak dengan nilai paling kecil. Nilai awal min adalah $+\infty$ dan akan berkurang seiring dengan berjalannya pencarian.
5. Max adalah agen yang mewakili komputer, sedangkan min adalah agen yang mewakili gerakan lawan dalam hal ini *user*.

Berikut adalah *flowchart* algoritma *Minimax* dengan optimasi *Alpha-Beta Pruning*.



Gambar 3.11 Flowchart Algoritma Minimax dengan Optimasi Alpha-Beta Pruning

3.3.1.4.1 Analisis Notasi Asimptotik Big *O Minimax* dan *Minimax* Optimasi Alpha Beta

Dalam praktek, kompleksitas waktu dihitung berdasarkan jumlah operasi abstrak yang *mendasari* suatu algoritma, dan memisahkan analisisnya dari implementasi. Kompleksitas waktu yang dilakukan adalah pada algoritma *Minimax* dan *Minimax* dengan optimasi *Alpha-Beta Pruning*.

1. **Kompleksitas waktu Algoritma *Minimax***

Berikut ini adalah *pseudo-code* untuk algoritma *Minimax* akan diperlihatkan pada gambar 3.12 di bawah ini.

```
1. Function Minimax( input file_arr, endarr
2.          output papan, chosen_score, chosen_move, ct, jmlkl)

   Deklarasi :
3.      a, child_awal, chosen_score : integer
4.      chosen_move, ct, tmp_score, tmp_move : integer

Algooritma :
5.   while N <= end_arr do                              o(n)
6.   tmpScore ← -99999                                  o(1)
7.   for a=1 to child_awal do                           o(1)
8.        chosen_score, chosen_move                     o(1)
9.        if chosen_score > tmpScore then               o(1)
10.           ct = ct + 1                               o(1)
11.           tmpScore = chosen_score                   o(1)
12.           tmpMove = chosen_move                     o(1)
13.       endif
14. chosen_score = tmpScore                             o(1)
15. chosen_move = tmpMove                               o(1)
16. papan = papanawal;                                  o(1)
17. Endfor
18. Endwhile

                                          Jumlah : O(n)
```

Gambar 3.12 *Pseudo-Code* Algoritma *Minimax*

Dari *pseudo-code* di atas operasi dasar dari *Minimax* adalah yang diberi tulisan warna merah dengan Kompleksitas *running time* ***T(n)=n***, karena operasi dasar diulang dari i=1 sampai panjang langkah. Notasi Asimptotiknya adalah *O*(n), karena pada pengulangan **while N <= end_arr do** tidak dapat

ditentukan panjangnya maka kompleksitasnya diambil dari kompleksitas waktu terburuk **(Tmax(n))** atau ***O(n).***

2. Kompleksitas waktu Algoritma *Minimax* optimasi Alpha Beta

Setelah didapat kompleksitas waktu *Minimax* maka selanjutnya adalah mendapatkan kompleksitas waktu *Minimax* optimasi *Alpha-Beta Pruning*.

Berikut ini adalah *pseudo-code* untuk algoritma *Minimax* optimasi *Alpha-Beta Pruning* akan diperlihatkan pada gambar 3.13 di bawah ini

```
1. Function AlphaBeta(input file_arr, endarr
2.          output papan, chosen_score, chosen_move, ct, jmlkl)

Deklarasi :
3. parent, chosen_score, tmpCounterVisit, comp : integer
4.  chosen_move, ct, tmp_score, tmp_move, counterVisit : integer

Algoritma :
5. tmpScore = chosen_score;                          o(1)
6. tmpMove = chosen_move;                            o(1)
7. while (parent, 'awal.mat') == 0 do                o(1)
8.    tmpCounterVisit = counterVisit                 o(1)
9.    if comp==1 then                                o(1)
10.       if tmpScore > chosen_score then            o(1)
11.          Pruning(next) // di pruning             o(1)
12.          chosen_score = tmpScore                 o(1)
13.          chosen_move = tmpMove                   o(1)
14.          ct = ct + 1                             o(1)
15.       endif
16.       else
17.       if tmpScore < chosen_score then            o(1)
18.          Pruning(next) // di pruning             o(1)
19.          chosen_score = tmpScore                 o(1)
20.          chosen_move = tmpMove                   o(1)
21.          ct = ct + 1                             o(1)
22.       endif
23.     counterVisit = tmpCounterVisit               o(1)
24.     tmpScore = chosen_score                      o(1)
25.     tmpMove = chosen_move                        o(1)
29.   endif
30. Endwhile
                                        Jumlah : O(n)
```

Gambar 3.13 Pseudo-Code Algoritma Minimax Alpha-Beta Pruning

Dari *pseudo-code* di atas operasi dasar dari *Minimax* Optimasi *Alpha-Beta pruning* adalah yang diberi tanda kotak merah dengan Kompleksitas *running time* ***T(n)=n-1***, Kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi

perbandingan *tmpscore* > *chosen score* dan *tmpscore* < *chosen score*. Serta untuk perhitungan Notasi Asimptotik Big *O* nya akan di jelaskan di bawah.

$= O(1) + O(1) + (n\text{-}1)\{ O(1) + O(1) + O(1) + O(1) + O(1) + O(1) +$

$O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1)$

$= O(1) + (n\text{-}1)\, O(1)$

$= O(1) + O(n\text{-}1)$

$= O(1) + O(n)$

$= \mathbf{O(n)}$

3.3.1.4.2 Kompleksitas Waktu Asimptotik *Minimax* dan *Minimax* Optimasi Alpha Beta

Dari hasil kompleksitas waktu yang di dapat dari penjelasan sebelumnya adalah :

1. Algoritma *Minimax*
   T(n)=n,
2. Algoritma *Minimax* Optimasi Alpha Beta
   T(n)=n-1,

Sehingga untuk perbandingan pertumbuhan T(n) dengan n untuk algoritma *Minimax*

Tabel 3.4 Perbandingan pertumbuhan *T*(*n*) dengan *n*

n	T(n)=n	n
3	3	3
6	6	6

10	10	10

Untuk *n* yang besar, pertumbuhan *T*(*n*) sebanding dengan *n*. Pada kasus ini, *T*(*n*) tumbuh seperti *n* tumbuh. *T*(*n*) tumbuh seperti *n* tumbuh saat *n* bertambah. Maka dikatakan bahwa *T*(*n*) berorde *n* dan tuliskan : *Big O nya adalah T(n)=O(n).* Bahwa Algoritma *Minimax* termasuk ke dalam algoritma *Lanjar.*

Sedangkan untuk perbandingan pertumbuhan T(n) dengan n untuk algoritma *Minimax* Optimasi Alpha Beta adalah sebagai berikut.

Tabel 3.5 Perbandingan pertumbuhan *T*(*n*) dengan *n*

n	T(n)=n-1	n
3	2	3
6	5	6
10	9	10

Untuk *n* yang besar, pertumbuhan *T*(*n*) sebanding dengan *n*. Pada kasus ini, *T*(*n*) tumbuh seperti *n* tumbuh. *T*(*n*) tumbuh seperti *n* tumbuh saat *n* bertambah. Maka dikatakan bahwa *T*(*n*) berorde *n* dan tuliskan : *Big O nya adalah T(n)=n-1=O(n).* Bahwa Algoritma *Minimax* Optimasi *Alpha-Beta Pruning* termasuk ke dalam algoritma *Lanjar.*

3.3.1.4.3 Kesimpulan Analisis Algoritma dengan Big *O*

Hasil dari analisis Big *O* untuk *Minimax* dan *Minimax* optimasi Alpha-Beta Pruning yang sudah digambarkan dan dihitung menggunakan Big *O* dengan notasi asimptotik pada penjelasan sebelumnya bahwa Big *O Minimax* termasuk ke dalam *time complexity function* "n" atau Big *O* nya adalah Big *O*(n) dan untuk *Minimax* optimasi Alpha-Beta Pruning termasuk ke dalam *time complexity function* "n" atau Big *O* nya adalah Big *O*(n), artinya algoritma tersebut termasuk ke dalam *time complexity function* "n", sehingga hasil dari perbedaan waktu antara *Minimax* dan *Minimax* optimasi Alpha-Beta Pruning tidak akan terlalu

berbeda jauh atau signifikan dilihat dari kompleksitas waktu antara *Minimax* yaitu t(n)=n dan *Minimax* optimasi *Alpha-Beta Pruning* t(n)=n-1. Kelompok Algoritma untuk *Minimax* dan *Minimax Alpha-Beta pruning* adalah sebagai berikut.

Tabel 3.6 Pengelompokan Algoritma

Algoritma	Kelompok Algoritma	Nama
Minimax	*O*(n)	Lanjar
Minimax Optimasi *Alpha-Beta Pruning*		

3.3.2 Kesimpulan Analisis Algoritma pada Permainan Congklak

Berdasarkan analisis yang ditabelkan dan digambarkan di atas pada sub-bab sebelumnya bahwa dapat di tarik sebuah kesimpulan, pada analisis lubang 3,6 dengan level mudah bahwa langkah yang ditelusuri oleh algoritma *Minimax* lebih banyak dari pada algoritma *Minimax Optimasi Alpha-Beta Pruning* walaupun perbedaannya hanya sedikit, namun *alpha-beta pruning* akan semakin terlihat *pruning*-nya jika kedalaman pohon semakin dalam dan lubang semakin banyak. Sedangkan dari hasil kompleksitas waktu dan notasi asimptotik terlihat bahwa kedua algoritma ini akan masuk ke dalam kategori algoritma Big *O(n)* atau *Lanjar*, sehingga hasil dari perbedaan waktu antara *Minimax* dan *Minimax* optimasi Alpha-Beta Pruning tidak akan terlalu berbeda jauh atau signifikan, hal tersebut dapat dilihat dari kompleksitas algoritmanya yaitu algoritma *Minimax* menghasilkan t(n)=n dan algoritma *Minimax* optimasi *Alpha-Beta Pruning* menghasilkan t(n)=n-1.