Modifikasi Algoritma Steepest-Ascent Hill Climbing Dan Backtracking Untuk Pencarian Lintasan Kritis Proyek

Modified Steepest-Ascent Hill Climbing and Backtracking Algorithm for Project Critical Path Finding

Elvina[1], Lukman Hakim[2]
[1,2] Program Studi Teknik Informatika, Universitas Bunda Mulia, Jakarta
e-mail: [1]aux.elvina@gmail.com, [2] irhakim77@yahoo.com

Abstrak

Lintasan kritis merupakan suatu hal yang sangat penting dan perlu diperhatikan dalam penjadwalan proyek, karena lintasan kritis mempunyai dampak terhadap terlambat atau tidaknya suatu proyek. Terdapat kenaikan pada angka pertumbuhan proyek di Indonesia. Oleh karena itu, untuk mendukung keberhasilan proyek tersebut, maka dilakukanlah penelitian terhadap pencarian lintasan kritis. Sehingga nantinya hasil dari penelitian ini berguna bagi para developer yang ingin membuat aplikasi yang menerapkan pencarian lintasan kritis. Parameter dari algoritma ini adalah waktu, yaitu : Earliest Start (ES), Early Finish (EF), Last Start (LS), dan Last Finish (LF). Algoritma Steepest-ascent Hill Climbing berguna untuk mencari goal berdasarkan nilai heuristik terbaik. Nilai heuristik terbaik yang dijadikan acuan adalah slack time dari kegiatan. Algoritma Backtracking merupakan perbaikan dari algoritma Brute-Force yang berbasis DFS (Deep-First Search). Jurnal ini membahas tentang algoritma Steepest-ascent Hill Climbing yang berguna untuk mencari slack (keterlambatan) guna menjadi tolak ukur dari lintasan kritis, dan backtracking yang berguna untuk mencari ES, EF, LS, dan LF guna menjadi parameter dalam mencari slack (keterlambatan). Angka keberhasilan dari penggabungan algoritma ini untuk mencari lintasan kritis adalah sebesar 80%.

Kata kunci—Steepest-ascent Hill Climbing, algoritma, backtracking, manajemen proyek, lintasan kritis.

Abstract

Critical path is very important and need to be considered in project scheduling, because critical path have an impact on the delay or failure of a project. There is an increase in project growth rates in Indonesia. Therefore, to support the success of the project, research was conducted on the search for critical paths. So that later the results of this study are useful for developers who want to create applications that implement critical path search. The parameters of this algorithm are time, namely: Earliest Start (ES), Early Finish (EF), Last Start (LS), and Last Finish (LF). The Steepest-ascent Hill Climbing algorithm is useful for finding goal based on the best heuristic values. The best heuristic value that is used as a reference is slack time from the activity. The Backtracking algorithm is an improvement from the DFS-based (Deep-First Search) Brute-Force algorithm. This journal discusses the Steepest-ascent Hill Climbing algorithm which is useful for finding slacks to become a benchmark for critical paths, and backtracking which is useful for finding ES, EF, LS, and LF to be used as parameters in finding slack. The success rate of combining this algorithm to find a critical path is 80%.

Keywords—*Steepest-ascent Hill Climbing, algorithm, backtracking, project management, critical path.*

1. PENDAHULUAN

Dari sisi pembangunan, terdapat kenaikan penjualan semen di pasar domestik sebesar 15,7 juta ton atau tumbuh sekitar 6,6% tahun ke tahun [1]. Hal ini berarti adanya proyek pembangunan di Indonesia yang sedang berlangsung, dan dari data tersebut diketahui bahwa pertumbuhan proyek di Indonesia semakin banyak dari tahun ke tahun.

Selain dari sisi pembangunan, juga terdapat proyek-proyek dalam perkembangan teknologi informasi di suatu perusahaan. Teknologi informasi menunjang produktifitas dari sebuah perushaan, seperti menjadikan dokumen perushaan menjadi lebih rapi dan terstruktur, dapat diubah sesuai keperluan, dan arsip digital [2]. Berdasarkan dari hasil survei yang dilakukan oleh kemkominfo (Kementrian Komunikasi dan Komunikasi) di sektor bisnis, 92% persen dari 803 perusahaan sudah menggunakan komputer [3]. Dan masih banyak proyek-proyek dibidang lainnya.

Dalam *memanage* sebuah proyek, tentunya ada hal-hal yang perlu diperhatikan. Salah satunya adalah dari segi waktu. Seorang manajer proyek haruslah merencanakan sebuah proyek tersebut dengan matang, agar proyek tersebut dapat diselesaikan dengan baik dan tidak adanya keterlambatan proyek. Dalam mengantisipasi keterlambatan proyek, seorang manajer proyek harus memperhatikan lintasan kritis dari proyek tersebut. Karena, bila pelaksanaan kegiatan-kegiatan dalam lintasan kritis mengalami keterlambatan, maka akan mengakibatkan keterlambatan proyek secara keseluruhan [4]. Berdasarkan hasil penelitian penyebab keterlambatan waktu pelaksanaan proyek di peringkat kedua adalah perencanaan dan penjadwalan pekerjaan [5].

Di zaman yang modern dan sudah maju ini, tentunya sebagian besar masyarakat sudah memiliki gadget untuk menunjang aktivitas yang dilakukan. Khususnya para manajer proyek, tentu memiliki gadget yang selalu dibawa untuk tetap dapat terhubung dengan timnya dan client. Untuk itu, diperlukan sebuah solusi penyelesaian yang dapat diterapkan dalam Teknologi Informasi untuk kasus pencarian lintasan kritis dalam manajemen proyek secara tepat dan akurat. Sehingga, nantinya dapat diterapkan dalam sebuah aplikasi dan berguna bagi manajer proyek dalam mengantisipasi keterlambatan proyek dengan memberikan perhatian lebih terhadap lintasan kritis.

2. METODE PENELITIAN

2.1 Kajian Pustaka

Dalam penelitian ini digunakan algoritma Steepest-ascent Hill Climbing dan algoritma Backtracking untuk mencari lintasan kritis.

2.1.1 Manajemen Proyek

Manajemen proyek adalah serangkaian aktivitas perencanaan, pelaksanaan dan pengendalian untuk memastikan agar sasaran waktu, anggaran dan spesifikasi tertentu dalam sebuah pekerjaan yang telah ditetapkan dapat tercapai secara efektif dan efisien [6]. Terdapat beberapa hal yang perlu diperhatikan dalam proyek, salah satunya adalah waktu. Waktu tenggang dari kegiatan yang terdapat dalam proyek dapat diukur dengan cara sebagai berikut :

$$S = LF - EF = LS - ES \quad (1)$$

Yang dimana :
S (*slack*) = waktu tenggang dari kegiatan dalam proyek.
LF (*latest activity finish time*) = waktu paling lambat dari penyelesaian kegiatan dalam proyek.
EF (*earliest activity finish time*) = waktu paling awal dari penyelesaian kegiatan dalam proyek.
ES (*earliest activity start time*) = waktu paling awal memulai kegiatan dalam proyek.
LS (*latest activity start time*) = waktu paling lama memulai kegiatan dalam proyek.

Terdapat 3 tahap untuk mencari *slack*. Yaitu:

1. *Forward Pass* yaitu perhitungan dilakukan dengan bergerak maju dari kegiatan paling awal hingga kegiatan paling akhir. Perhitungan ini berguna untuk mencari ES dan EF.
2. *Backward Pass*. Perhitungan ini berguna untuk mencari LS dan LF. Berbeda dengan tahap sebelumnya yaitu forward pass, perhitungan ini dilakukan dari kanan ke kiri.
3. Pencarian Waktu tenggang, perhitungan ini dilakukan mulai dari paling kegiatan paling awal menuju kegatan paling akhir. Perhitungan ini dilakukan dengan mencari selisih LF dengan EF atau selisih dari LS dengan ES, yang ditunjukan oleh persamaan (1). Kegiatan-kegiatan yang masuk ke dalam lintasan kritis memiliki Slack bernilai 0, sehingga lintasan kritis merupakan kumpulan kegiatan-kegiatan yang mempunyai waktu tenggang 0 dari awal hingga akhir.

Untuk mencari EF dari suatu kegiatan dalam proyek, digunakan persamaan sebagai berikut:

$$EF = ES + t \quad (2)$$

Dalam mencari LS dari suatu kegiatan maka digunakan persamaan seperti dibawah :

$$LS = LF - t \quad (3)$$

Sedangkan ES dapat diketahui dengan melihat ES yang paling tinggi dari kegiatan sebelumnya, dan LF dapat diketahui dengan melihat LS paling rendah dari kegiatan selanjutnya.

Keterangan :

t = waktu untuk melaksanakan proyek ; waktu pengerjaan.

2.1.2 Algoritma Steepest-ascent Hill Climbing

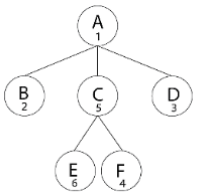
Terdapat 2 macam metode dalam algoritma pencarian *Hill Climbing* [7], yaitu:

1. Pencarian *Simple Hill Climbing*
2. Pencarian *Steepest-ascent Hill Climbing*

Dalam penelitian ini, algoritma pencarian yang digunakan untuk mencari *slack time* adalah *Steepest-ascent Hill Climbing*. Algoritma pencarian *Steepest-ascent Hill Climbing* pada dasarnya hampir sama dengan Algoritma pencarian *Simple Hill Climbing*, yang membedakannya adalah gerakan pencarian yang tidak dimulai dari posisi paling kiri namun gerakan selajutnya dicari berdasarkan nilai heuristik terbaik. Algoritma dari *Steepest Ascent Hill Climbing Search* adalah [8] :

1. Mulai dari keadaan awal, lakukan pengujian. Jika merupakan tujuan maka berhenti, dan jika tidak lanjutkan dengan keadaan sekarang sebagai keadaan awal.
2. Ulangi hingga tujuan tercapai atau hingga pencarian tidak memberikan perubahaan pada keadaan sekarang.
   a. Tentukan SUCC sebagai nilai heuristik dari *successor-successor*.
   b. Lakukan untuk tiap node yang digunakan oleh keadaan sekarang.
      i. Gunakan node tersebut dan bentuk keadaan baru.
      ii. Evaluasi keadaan baru tersebut jika merupakan tujuan keluar. Jika bukan, bandingkan nilai heuristiknya dengan SUCC. Jika lebih baik, jadikan nilai heuristik keadaan baru tersebut sebagai SUCC, tetapi jika tidak lebih baik, nilai SUCC tidak berubah.
      iii. Jika SUCC lebih baik daripada nilai heuristik keadaan sekarang, ubah node SUCC menjadi keadaan sekarang.

Contoh:



Gambar 1. Contoh *Steepest-ascent Hill Climbing*

Angka pada gambar 1, merupakan nilai heuristik. *Goal* / tujuan akhir dari kasus di atas dimisalkan adalah node E. Sehingga, langkah-langkah dalam algoritma Steepest-ascent Hill Climbing untuk contoh di atas adalah :

1. Node A dianggap sebagai CURRENT (keadaan sekarang).
2. Lakukan pengecekan terhadap node B.
   Apakah node B merupakan tujuan ? Jika bukan, maka apakah nilai heuristik node B lebih besar daripada CURRENT ? Jika iya maka CURRENT = B.
3. Lakukan pengecekan terhadap node C. Node C bukan merupakan tujuan, dan nilai heuristik node C > CURRENT. Maka CURRENT = C.
4. Lakukan pengecekan terhadap node D. Node D bukan merupakan tujuan, dan nilai heuristik node D < CURRENT. Maka CURRENT = C.
5. Lakukan pengecekan terhadap node E. Node E merupakan tujuan. Lalu keluar (Selesai).

2.1.3 Algoritma Backtracking

Algoritma *backtracking* merupakan algoritma yang berbasis pada *Depth First Search* (DFS) untuk mencari solusi persoalan secara lebih efisien. Algoritma *backtracking* melakukan pencarian solusi persoalan secara sistematis pada semua kemungkinan persoalan yang ada pada setiap node dengan berbasis DFS secara [9]. DFS merupakan metode pencarian setiap level dimulai dari sisi kiri atau paling awal. Jika ternyata tidak ditemukan solusi dan tidak terdapat langkah selanjutnya (jalan buntu), maka akan dilakukan *backtracking* ke node 1 level di bawahnya (node sebelumnya), dilanjutkan dengan mengunjungi node lain yang belum dikunjungi hingga menemukan solusi atau sudah tidak terdapat node lain yang belum dikunjungi. Langkah-langkah pencarian solusi menggunakan algoritma *backtracking* adalah [10]:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Simpul yang sudah dilahirkan dinamakan simpul hidup dan simpul hidup yang diperluas dinamakan simpul-E (*Expand-node*).
2. Jika lintasan yang diperoleh dari perluasan simpul-E tidak mengarah ke solusi maka simpul itu akan menjadi simpul mati dimana simpul itu tidak akan diperluas lagi.
3. Jika posisi terakhir ada di simpul mati, maka pencarian dilakukan dengan membangkitkan simpul anak yang lainnya dan jika tidak ada simpul child maka dilakukan *backtracking* ke simpul parent.
4. Pencarian dihentikan jika kita telah menemukan solusi atau tidak ada simpul hidup.

Sebagai contoh, misalkan terdapat seperti pada gambar 2.1, dan angka yang tercantum merupakan angka yang dimiliki oleh masing-masing node, dengan *goal* / tujuan akhir adalah menemukan angka 3. Sehingga langkah-langkah dari contoh tersebut adalah :

1. Kondisi sekarang adalah node A.
2. Lakukan pengecekan pada node B sehingga keadaan sekarang adalah node B.
   Node B bukan merupakan tujuan, dan ternyata node B tidak memiliki *child*. Sehingga dilakukan *backtracking* ke node *parent* dari node B yaitu node A.
3. Keadaan sekarang berada di node A kembali.
4. Lakukan pengecekan pada node C. Keadaan sekarang adalah node C. Node C bukan merupakan tujuan namun ternyata node C memiliki *child.*
5. Lakukan pengecekan pada *child* dari node C yaitu node E. Node E bukan merupakan tujuan dan tidak memiliki *child*. Sehingga dilakukan *backtracking* ke parent dari node E yaitu node C.
6. Keadaan sekarang berada di node C kembali. Node C masih memiliki *child* yang belum dikunjungi, yaitu node F. Lakukan pengecekan pada node F, sehingga keadaan sekarang adalah node F. Node F bukan merupakan tujuan dan tidak memiliki *child*. Sehingga dilakukan *backtracking* ke *parent* dari node F yaitu node C.

7. Keadaan sekarang adalah node C kembali. Node C tidak memiliki *child* lain yang belum dikunjungi, sehingga dilakukan *backtracking* ke *parent* dari node C, yaitu node A.
8. Keadaan sekarang berada di node A. Node A masih memiliki *child* yang belum dikunjungi yaitu node D.
9. Lakukan pengecekan pada node D. Node D merupakan tujuan. Sehingga, *goal* tercapai dan pencarian berhenti.

2.2 Perancangan Penelitian

Dalam melakukan penelitian ini, adapun tahapan yang dilalui adalah sebagai berikut:

a. Pengumpulan Data

Dalam mengumpulkan data, metode yang digunakan dalam penelitian ini adalah kualitatif, yaitu dengan cara studi kasus. Masalah dalam penelitian ini didapat dari portal berita yang terpercaya, dan juga berasal dari penelitian yang dilakukan oleh seseorang dan didokumentasikan dalam bentuk jurnal.

b. Analisis Data

Metode analisis data yang digunakan dalam penelitian ini adalah dengan cara studi literatur, yaitu dengan mencari referensi teori yang relefan dengan kasus permasalahan yang ditemukan. Teori yang dijadikan referensi berasal dari buku, ataupun jurnal.

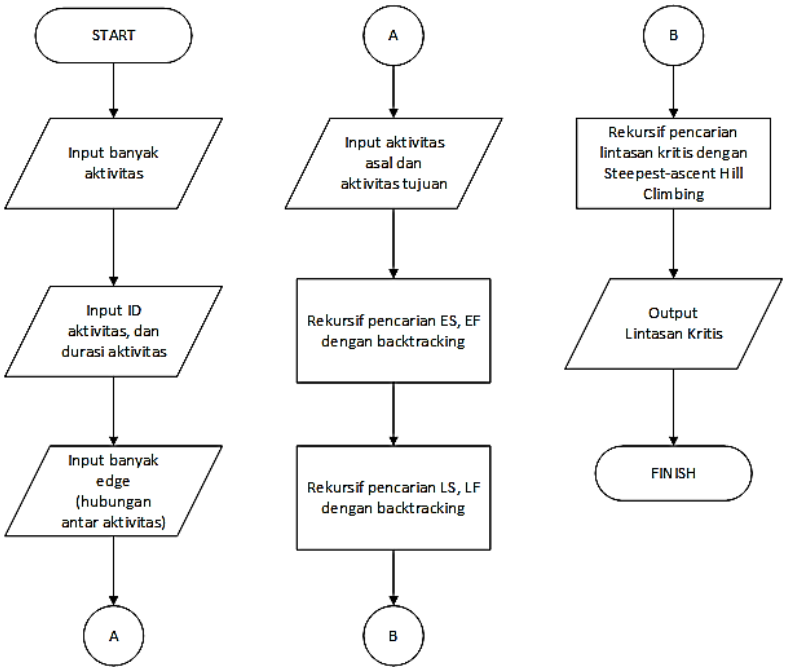
c. Implementasi

Setelah mendapatkan algoritma yang dirasa cocok berdasarkan hasil studi literatur, maka penulis mencoba untuk mengimplementasikan algoritma tersebut untuk kasus pencarian lintasan kritis dalam kode program yang ditulis menggunakan bahasa C++.

d. Pengujian

Pengujian algoritma dilakukan dengan menginputkan sampel uji berupa kasus. Perbandingan dilakukan antara hasil perhitungan lintasan kritis menggunakan algoritma tersebut dan perhitungan manual dengan pencocokan hasil sebagai tolak ukur.

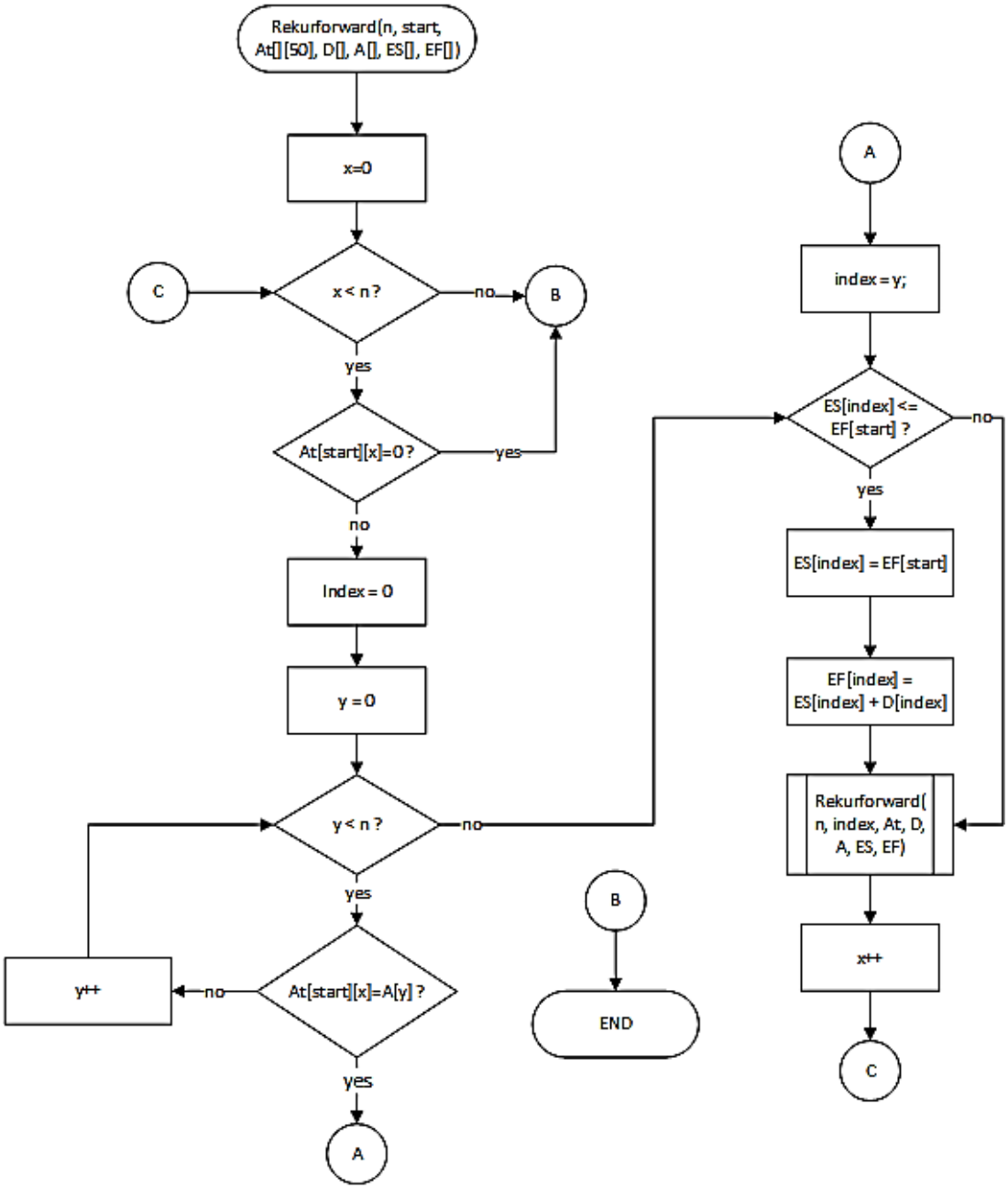
2.2.1 Perancangan alur program



Gambar 2. *Flowchart* Program

Adapun diagram alir dari program yang mengimplementasikan algoritma pencarian Steepest ascent Hill Climbing dan algoritma Backtracking ini dapat dilihat pada gambar 2, Dimulai dari user melakukan *input*, proses rekursif, hingga program menampilkan *output.*

2.2.2 Perancangan Alur Prosedur Forward Pass



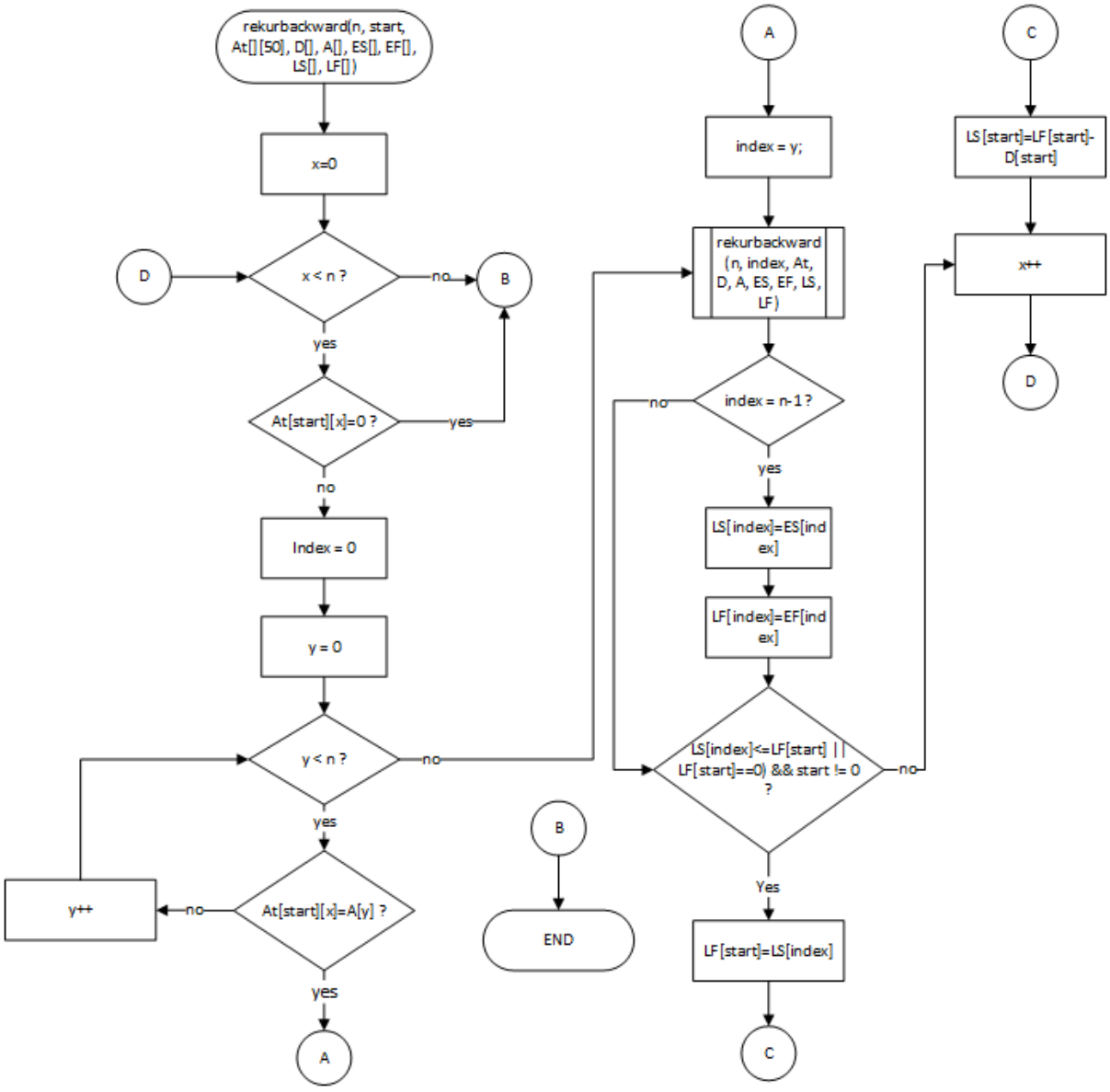
Gambar 3. *Flowchart* Prosedur *Forward Pass.*

Pada gambar 3, dapat dilihat bahwa nama dari prosedur tersebut adalah rekurforward yang mempunyai 7 parameter. Adapun keterangan dari masing-masing parameter dalam prosedur rekurforward adalah sebagai berikut.

1. n = jumlah kegiatan

2. start = posisi sekarang
3. At = *relationship* masing-masing kegiatan
4. D = durasi
5. A = id kegiatan
6. ES = *Early Start*
7. EF = *Early Finish*

2.2.3 Perancangan Alur Prosedur Backward Pass

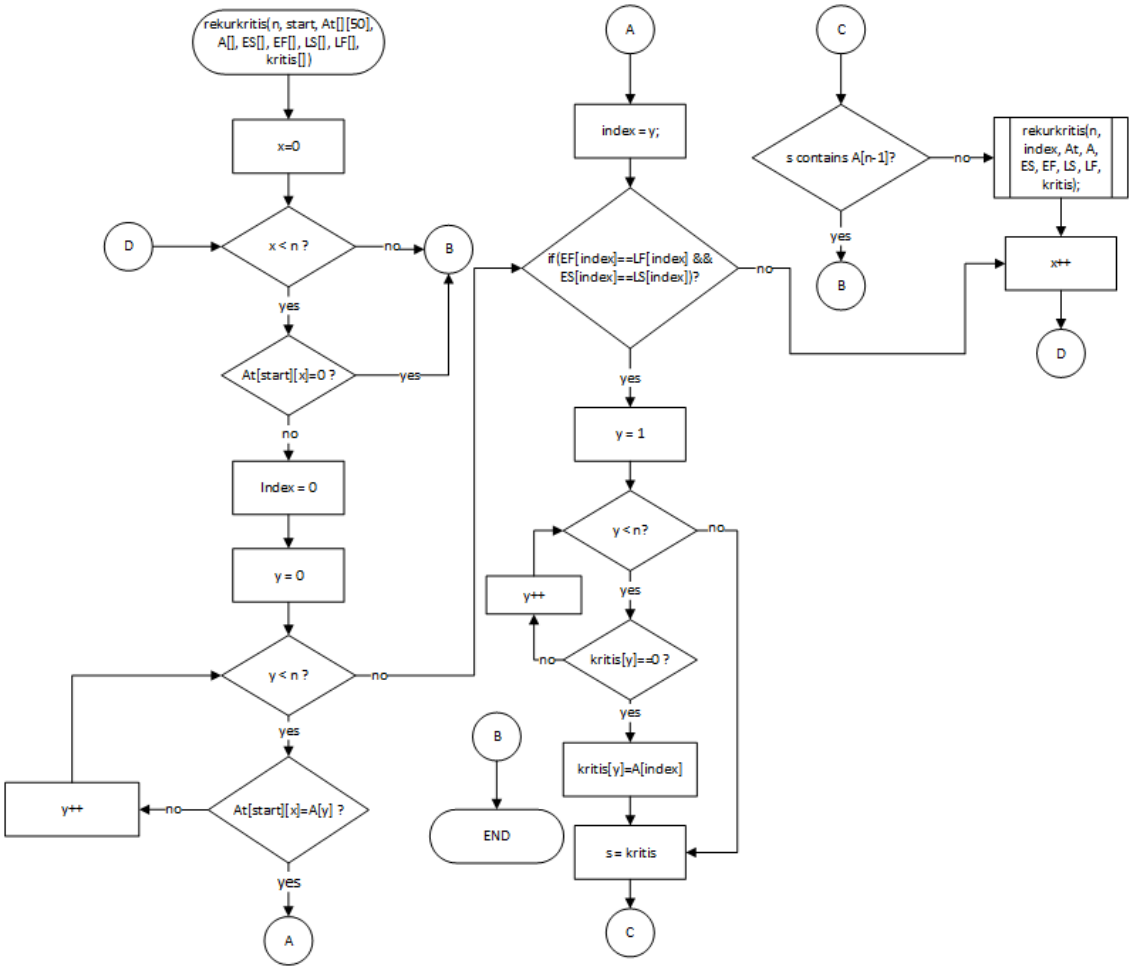


Gambar 4. *Flowchart* Prosedur *Backward Pass.*

Pada gambar 4, dapat dilihat bahwa terdapat perbedaan parameter antara prosedur *forward pass* dengan prosedur *backward pass*. Pada prosedur backwardpass terdapat 9 parameter. Terdapat penambahan 2 parameter, yaitu :

1. LS = *Last Start*
2. LF = *Last Finish*

2.2.4 Perancangan Alur Prosedur Rekursif Pencarian Lintasan Kritis



Gambar 5. Flowchart Prosedur Rekursif Pencarian Lintasan Kritis.

Prosedur pada gambar 5, mempunyai 9 parameter, adapun parameter yang berbeda dari parameter sebelumnya adalah adanya parameter yang bernama kritis. Adapun parameter kritis bertujuan untuk menyimpan lintasan kritis, yang diakumulasi dalam setiap proses rekursif.

3. HASIL DAN PEMBAHASAN

3.1 Hasil

Dari penelitian ini, terbukti bahwa penggabungan kedua algoritma tersebut dapat digunakan untuk mencari lintasan kritis dalam manajemen proyek. Penulis menggunakan 5 *testcase* sebagai *sampling* untuk membuktikan bahwa algoritma dapat berjalan d dapat menemukan lintasan kritis. *Berikut* adalah 2 dari 5 *test case* yang digunakan untuk menguji algoritma tersebut.

Tabel 1. Tabel *Test-Case*

No. *Test-Case*	BanyakKegiatan	ID Kegiatan	Durasi	Relationship
1	12	S	0	a, b, c
		a	3	d, e, f
		b	2	g
		c	1	h
		d	2	j
		e	3	i
		f	4	i
		g	3	j
		h	5	F
		i	3	j
		j	4	F
		F	0	-
2	10	S	0	a
		a	10	b, f, h
		b	20	c, g
		c	5	d
		d	10	e
		e	20	F
		f	15	g
		g	5	e
		h	15	e
		F	0	-

Data *test-case* tersebut diinputkan ke dalam program, adapun *output* dari program tersebut adalah :

Tabel 2. Tabel *Output*

No. *Test-Case*	*ID Kegiatan*	*Early Start*	*Late Start*	*Early Finish*	*Late Finish*	Lintasan Kritis
1	S	0	0	0	0	S-a-f-i-j-F
	a	0	3	0	3	
	b	0	2	5	7	
	c	0	1	8	9	
	d	3	5	8	10	
	e	3	6	4	7	
	f	3	7	3	7	
	g	2	5	7	10	
	h	1	6	9	14	
	i	7	10	7	10	
	j	10	14	10	14	
	F	14	14	14	14	
2	S	0	0	0	0	S-a-b-c-d-e-F
	a	0	10	0	10	
	b	10	30	10	30	
	c	30	35	30	35	
	d	35	45	35	45	
	e	45	65	45	65	

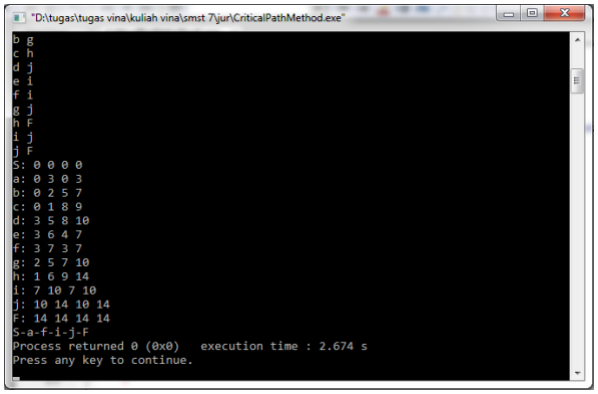
	f	10	25	25	40	
	g	35	40	40	45	
	h	10	25	30	45	
	F	65	65	65	65	

Pembanding yang digunakan untuk melihat seberapa tepat program ini adalah dengan menghitung manual. Adapun perbandingan hasil pencarian lintasan kritis dengan menggunakan program dan dengan perhitungan manual adalah sebagai berikut :

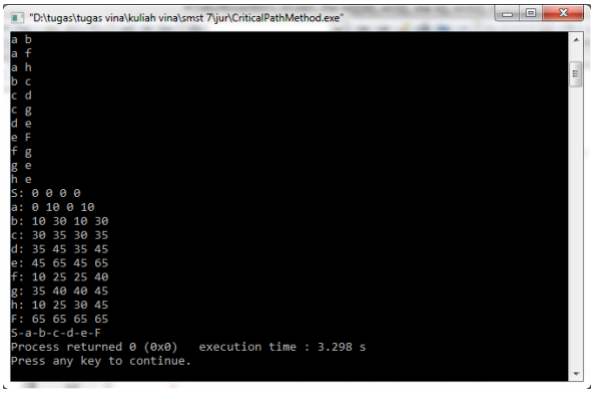
Tabel 3. Tabel Perbandingan

No. *Test-Case*	Lintasan Kritis Hasil Perhitungan Program	Lintasan Kritis Hasil Perhitungan Manual	Ketepatan (tepat / tidak)
1	S-a-f-i-j-F	S-a-f-i-j-F	tepat
2	S-a-b-c-d-e-F	S-a-b-c-d-e-F	tepat
3	S-a-c-f-h-i-F	S-a-c-f-h-i-F	tepat
4	S-a-e-i-F	S-a-e-i-F	tepat
5	S-a-b-e-F-c	S-a-c-b-e-F	tidak tepat

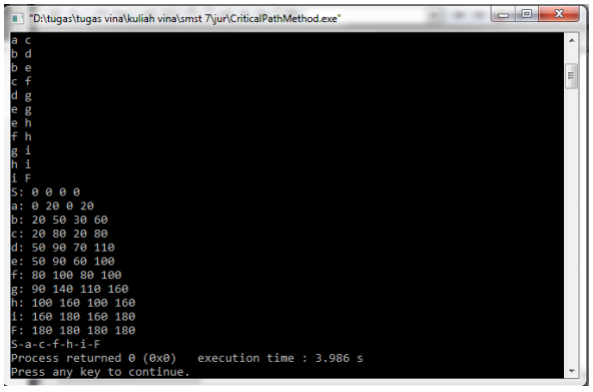
Sehingga tingkat ketepatan perhitungan lintasan kritis berdasarkan hasil perhitungan program jika dibandingkan dengan perhitungan manual adalah sebesar 80% dari 5 contoh kasus (*test-case*). Sedangkan untuk perhitungan ES, EF, LS, dan LF, algoritma *backtracking* mampu memecahkan 5 *test-case* dengan benar.
Berikut adalah tampilan hasil program dengan menggunakan *test-case* tersebut:



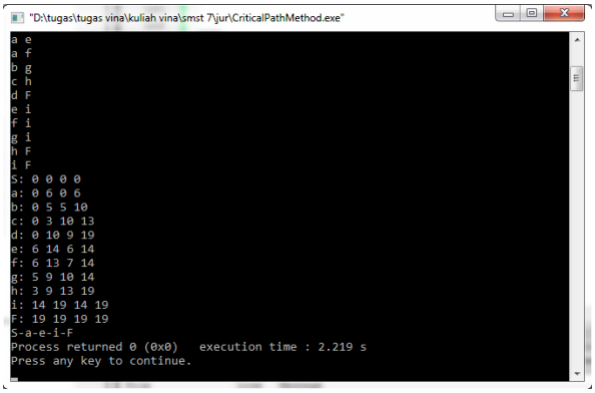
Gambar 6. *Test-Case* Pertama



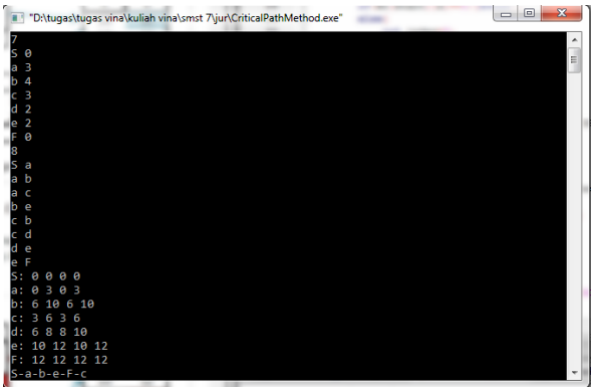
Gambar 7.*Test-Case* Kedua



Gambar 8. *Test-Case* Ketiga



Gambar 9. *Test-Case* Keempat



Gambar 10. *Test-Case* Kelima

3.2 Pembahasan

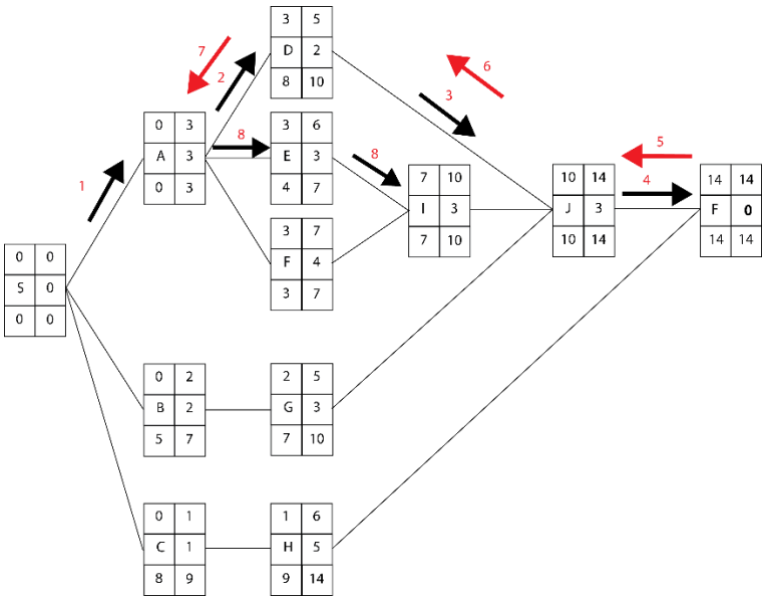
Terdapat 2 algoritma yang digunakan, yaitu :

a. Algoritma Backtracking
b. ModifikasiAlgoritma Steepest-ascent Hill Climbing

Berikut penjelasan mengenai masing-masing algoritma.

1. Algoritma *Backtracing*

Algoritma *Backtracking* pada kasus ini digunakan untuk mencari ES, EF, LS, LF. Cara kerja dari algoritma ini dapat dilihat pada gambar 11.



Gambar 11. Ilustrasi algoritma *Backtracking*

Tahap pertama dimulai dari id kegiatan S, yang kemudian dilanjutkan ke titik A→D→J→F. Disetiap proses rekursif, dilakukan pengecekkan apakah tujuan selanjutnya merupakan titik terakhir (tidak ada jalan lagi). Jika ternyata titik sekarang merupakan titik akhir, maka akan dilakukan proses *backtracking* ke titik sebelumnya hingga semua titik telah berhasil dikunjungi. Pada tahapan ke-4 dapat dilihat bahwa posisi sekarang berada di titik F (id kegiatan F). Maka dilakukan pengecekkan apakah masih terdapat titik selanjutnya yang terhubung dengan titik F, dan jawabannya adalah tidak. Maka, pada tahap ke-5 akan dilakukan proses *backtracking*

ke titik sebelumnya yaitu titik J (id kegiatan J). Kemudian pada titik J akan dilakukan pengecekkan kembali apakah masih ada titik lain yang terhubung dengan titik J selain titik F, dan jawabannya adalah tidak. Maka dilanjutkan dengan proses *backtracking* ke titik sebelumnya yaitu titik D. Pada titik A, dilakukan pengecekkan yang serupa yaitu apakah ada titik lain yang terhubung dengan titik A selain titik D, dan jawabannya adalah ada yaitu titik E. Maka akan dilakukan proses rekursif menuju titik E. Pada gambar 11, proses *backtracking* ditandai dengan panah berwarna merah.

Proses pencarian ES dan EF dilakukan pada saat proses *forward-pass* (bergerak maju) yang dapat dilihat pada gambar 11 dengan panah berwarna hitam. Namun untuk dapat mengisi ES dan EF secara keseluruhan, maka diperlukan peranan proses *backtracking*. Hal ini disebabkan sifatnya yang *brute-force*, yaitu mencoba semua kemungkinan yang ada. Adapun pseudocode dari algoritma backtracking untuk proses *forward-pass* yang digunakan dalam kasus ini adalah sebagai berikut :

```
procedure rekurforward (n : integer, start : integer, At[][50] : char, D[] : integer, A[] : char, ES[]
: integer, EF[] : integer)
{
        for x : integer ← 0 to n do
                if At[start][x] = 0 then
                        break
                else
                        index : integer ← 0
                        for y : integer ← 0 to n
                                if At[start][x] = A[y] then
                                        index ← y
                                        break
                                end if
                        end for
                        if ES[index] <= EF[start] then
                                ES[index] ← EF[start]
                                EF[index] ← ES[index] + D [index]
                        end if
                        rekurforward(n, index, At, D, A, ES, EF)
                end if
        end for
}
```

Sedangkan untuk mencari LS dan LF, dilakukan pada saat proses backtracking (*backward-pass*). Adapun pseudocode untuk mencari LS dan LF adalah sebagai berikut :

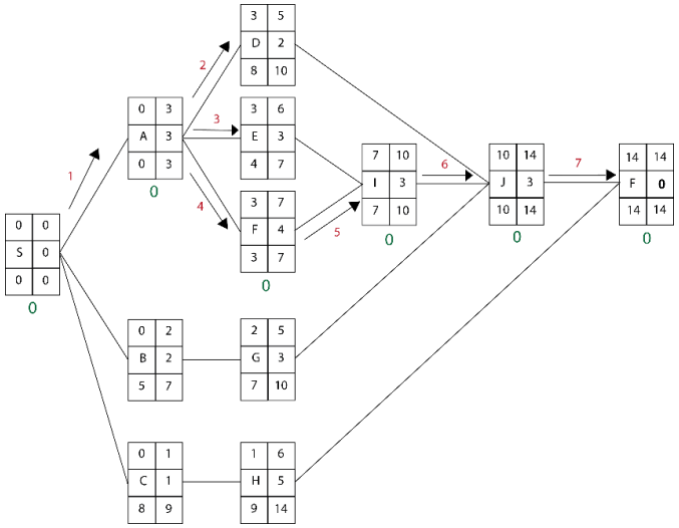
```
procedure rekurbackward (n : integer, start : integer, At[][50] : char, D[] : integer,  A[] : char,
ES[] : integer, EF[] : integer, LS[] : integer, LF[] : integer)
{
        for x : integer ← 0 to n do
                if At[start][x] = 0 then
                        break
                else
                        index : integer ← 0
                        for y : integer ← 0 to n
                                if At[start][x] = A[y] then
```

```
                                    index ← y
                                    break
                            end if
                    end for
                    rekurbackward(n, index, At, D, A, ES, EF, LS, LF)
                    if index = n-1 then
                            LS[index]←ES[index];
                            LF[index]←EF[index];
                    end if
                    if (LS[index] <= LF [start] or LF[start] = 0) and start != 0 then
                            LF[start]←LS[index]
                            LS[start←LF[start] - D[start]
                    end if
            end if
    end for
}
```

Dimana :
A = id kegiatan
D = durasi
n = jumlah kegiatan
At = Relasi

2. Modifikasi Algoritma *Steepest-ascent Hill Climbing*

Algoritma ini digunakan untuk mencari *slack*. Adapun tahapan dalam pencarian slack dapat diilustrasikan seperti gambar 12.



Gambar 12. Ilustrasi algoritma Modifikasi *Steepest-Ascent Hill Climbing*.

Perbedaan dalam algoritma Modifikasi *Steepest-ascent Hill Climbing* dapat dilihat pada gambar 12. Proses pencarian *goals* / tujuan dilakukan dengan mencari nilai heuristic tertinggi untuk mecapai titik akhir (*Finish*). Dalam kasus ini, nilai heuristik tertiggi adalah jika *slack* bernilai 0. Selain itu, proses pencarian *goals* dalam algoritma yang sudah dimodifikasi ini tidak terdapat proses *backtracking*. Sedangkan algoritma *Steepest-ascent Hill Climbing* dengan versi asli, terdapat proses *backtracking* ke titik sebelumnya, jika *goals* tidak ditemukan. Tentunya pencarian dengan menggunakan algoritma *Steepest-ascent Hill Climbing* membutuhkan *cost* proses yang lebih besar dibandingkan dengan hasil modifikasi *Steepest-ascent Hill Climbing*.

Proses *backtracking* ini didasari oleh nilai heuristik. Sehingga proses akan berjalan *backtrack* menuju titik 1 level sebelumya yang memiliki nilai heuristik tertinggi yang belum dikunjungi. Tidak adanya proses *backtracking* pada modifikasi Steepest-Ascent Hill Climbing ini dikarenakan tidak diperlukannya proses untuk kembali ke kegiatan sebelumnya. Karena sudah dipastikan akan ada *slack* bernilai 0 untuk setiap level. Berikut adalah *pseudocode* untuk mencari lintasan kritis dengan memodifikasi algoritma *Steepest-Ascent Hill Climbing* :

```
procedure rekurkritis (n : integer, start : integer, At[][50] : char, A[] : char, ES[] : integer, EF[] : integer, LS[] : integer, LF[] : integer, kritis[] : char)
{
        for x : integer ← 0 to n do
                if At[start][x] = 0 then
                        break
                else
                        index : integer ← 0
                        for y : integer ← 0 to n
                                if At[start][x] = A[y] then
                                        index ← y
                                        break
                                end if
                        end for
                        if EF[index] = LF[index] and ES[index] = LS[index]
                                for y : integer ← 1 to n do
                                        if kritis[y] = 0
                                                kritis[y] ← A[index]
                                                break
                                        end if
                                end for
                                s : string ← kritis
                                if s contains A[n-1] then // jika s sudah mencapai titik finish
                                        break
                                end if
                                rekurkritis(n, index, At, A, ES, EF, LS, LF, kritis)
                        end if
                end if
        end for
}
```

Berdasarkan hasil percobaan menggunakan 5 *test-case*, modifikasi algoritma *Steepest-ascent Hill Climbing* berhasil menemukan lintasan kritis dengan tepat sebanyak 4 kasus, dan 1 kasus lainnya tidak dapat dipecahkan dengan tepat. Hal ini disebabkan modifikasi algoritma *Steepest-ascent Hill Climbing* tidak melakukan *backtracking* untuk mencoba semua kemungkinan yang ada. Sehingga apa bila terdapat 2 lintasan kritis (*slack* dari kegiatan bernilai 0), modifikasi algoritma ini tidak membandingkan antara 1 lintasan kritis dengan lintasan kritis lainnya untuk mencapai hasil maksimal.

4. KESIMPULAN

Hasil dari penelitian ini menunjukkan bahwa angka keberhasilan dalam menangani 5 *test-case* adalah sebesar 80%. Hal ini disebabkan tidak adanya proses *backtracking* dari modifikasi algoritma *Steepest-ascent Hill Climbing* untuk membandingkan setiap kemungkinan lintasan kritis yang ada. Namun dengan tidak adanya proses *backtracking* pada modifikasi algoritma

Steepes-ascent Hill Climbing menjadikan algoritma ini mempunyai *process cost* yang lebih rendah jika dibandingkan dengan algoritma yang tidak dimodifikasi. Kekurangan dari modifikasi algoritma *Steepest-ascent Hill Climbing* ini mampu memecahkan *test-case* dengan tepat jika terdapat 1 lintasan kritis saja (lintasan dengan *slack* 0), jika terdapat lebih dari satu kemungkinan maka algoritma hasil modifikasi ini akan mengambil lintasan kritis yang pertama kali ditemukan. Sedangkan untuk algoritma *backtracking* mampu menangani semua *test-case* dengan tepat untuk mencari ES, EF, LS, dan LF. Kesulitan yang dialami penulis adalah penulis sulit untuk mendapatkan artikel yang membahas tentang pencarian lintasan kritis, padahal lintasan kritis merupakan hal fatal yang perlu diberikan perhatian lebih dalam manajemen proyek.

5. SARAN

Dari hasil penelitian ini, penulis menyarankan untuk penelitian selanjutnya dapat mengembangkan algoritma yang digunakan untuk mencari lintasan kritis ini sehingga dapat menemukan lintasan kritis dengan hasil maksimal jika terdapat lebih dari satu kemungkinan lintasan kritis. Sehingga bagi para *developer* yang ingin menerapkan pencarian *critical path* dalam aplikasinya, dapat menjadikan artikel ini atau artikel pada penelitian selanjutnya sebagai referensi.

DAFTAR PUSTAKA

[1] A. Hidayat and B. T. Rafie, "Penjualan semen domestik kuartal I naik 6,6%," 19 April 2018. [Online]. Available: https://industri.kontan.co.id/news/penjualan-semen-domestik-kuartal-i-naik-66#. [Accessed 21 September 2018].

[2] Y. Maryono and B. P. Istiana, Teknologi Informasi & Komunikasi 1, Bogor: Yudhistira, 2008.

[3] K. K. d. Informatika, Hasil Survei Penggunaan Teknologi Informasi dan Komunikasi di Sektor Bisnis Indonesia, Jakarta: Pusat Data dan Sarana Informatika, 2011.

[4] R. Nur and M. A. Suyuti, Pengantar Sistem Manufaktur, Yogyakarta: Deepublish, 2017.

[5] B. Proboyo, "Keterlambatan Waktu Pelaksanaan Proyek : Klasifikasi dan Peringkat dari Penyebab-penyebabnya," *Civil Engineering Dimension*, pp. 49-58, 1999.

[6] B. Harsanto, Dasar Ilmu Manajemen Operasi, Sumedang: Unpad Press, 2013.

[7] S. Kusumadewi, Artificial Intelligence, Yogyakarta: Graha Ilmu, 2003.

[8] M. R. Firdaus, S. I. Halim and D. , "Penerapan Metode Hill Climbing Search Untuk Pencarian Lokasi Terdekat pada Aplikasi Toko Virtual Berbasis Android," *Seminar Perkembangan dan Hasil Penelitian Ilmu Komputer (SPHP-ILKOM)*, pp. 88-97, 2014.

[9] R. Lumbantoruan, Y. N. Simatupang, M. Siahaan, M. H. Pardede and J. Pakpahan, "Penjadwalan Kuliah dengan Algoritma Backtracking," *Konferensi Nasional ICT-M Politeknik Telkom (KNIP)*, pp. 256-264, 2012.

[10] Teneng, J. Purwadi and E. Kurniawan, "Penerapan Algoritma Backtracking pada Permainan Math Maze," *Jurnal Informatika*, pp. 56-67, 2010.