Algoritma Greedy untuk Diimplementasikan Pada Shortest Seek Time First Disk Scheduling

William Juniarta Hadiman - 13516026
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516026@std.stei.itb.ac.id

***Abstract*—Shortest Seek Time First pada Disk Scheduling adalah pemilihan seek time terkecil daripada penjadwalan disk dalam suatu operating system. Algoritma Greedy adalah algoritma yang mengimplementasikan shortest seek time first ini dengan mencari seek time yang paling sedikit dari disk scheduling ini. Dapat disimpulkan juga disk scheduling ini di sort berdasarkan seek time terendah.**

***Keywords*—Greedy, Shortest Seek Time First, Disk Scheduling, Implementation greedy.**

I. PENDAHULUAN

Di Zaman serba modern seperti sekarang, teknologi sudah semakin maju. Kapasitas smartphone sudah seperti komputer PC. Semua perusahaan gadget ingin menampilkan teknologi tertinggi untuk memuaskan pelanggannya.

Manusia sendiri pada jaman ini menggunakan teknologi yang banyak ini untuk melakukan pekerjaan mereka. Pelajar menggunakan komputer untuk membuat dokumen untuk tugas mereka, mahasiswa menggunakan komputer untuk membuat pekerjaan dari para dosen. Baik dari umur yang sudah tua sampai ke anak-anak balita zaman sekarang sudah menggunakan kecanggihan teknologi ini.

Namun dibalik banyaknya orang-orang yang menggunakan teknologi ini, kita dapat melihat bahwa tidak banyak orang yang mengetahui bagaimana teknologi itu bekerja. Mereka tidak tahu bagaimana suatu sistem dalam teknologi itu bekerja. Sistem dalam sistem operasi itu dimakan sistem operasi.

Sistem operasi adalah suatu sistem untuk menghubungkan hardware dari suatu teknologi (pada smartphone contohnya adalah speaker, layar LCD, processor, dll) dan software pada smartphone itu (pada smartphone dimisalkan aplikasi note). Sistem operasi harus mengimplementasikan bagaimana cara mengetik, cara mengambil gambar, design daripada tampilan User Interface, dll. Tanpa suatu sistem operasi, kita tidak akan bisa menggunakan teknologi-teknologi yang ada seperti smartphone, laptop, PC, bahkan teknologi-teknologi seperti televisi pun jaman sekarang banyak yang menggunakan operatins system.

Dalam membuat suatu sistem, perlu adanya pemakaian suatu algoritma untuk menjalankan sistem tersebut secara efektif dan efisien. Scheduling adalah suatu cara suatu sistem operasi untuk memangkuskan cara pembacaan disk oleh suatu sistem. Waktu akses disk terdiri dari *seek time*, waktu yang dibutuhkan oleh head untuk bergerak ke sektor yang dituju dan *rotational latency* dan waktu tambahan untuk memutar suatu disk ke ke sektor tertentu.

Suatu waktu akses disk dapat dioptimasi dengan mengatur pengurutan dari disk yang akan dibaca dahulu. Daftar urutan disk ini adalah schedule dan untuk mendapatkan hasil yang optimal, kita perlu mengatur urutan dari disk ini dengan scheduling. Karena operating system adalah suatu program yang ada pada setiap sistem komputer, maka harus dibuat suatu cara untuk melakukan scheduling tanpa perlu campur tangan manusia. Oleh karena itu diciptakan algoritma scheduling ini.

Ada beberapa cara scheduling salah satunya adalah Shortest Seek Time First dimana cara ini akan kita bahas.

Shortest Seek Time First ini menggunakan algoritma greedy dalam scheduling yaitu mencari request terdekat dari titik saat ini sampai dengan semua request terpenuhi.

II. DASAR TEORI

Algoritma adalah cara yang digunakan untuk menyelesaikan suatu masalah secara sistematis. Ada banyak algoritma yang dipelajari pada mata kuliah IF2211 Stategi Algoritma yaitu algoritma *brute force*, algoritma *greedy*, algoritma *divide and conquer*, algoritma *decrease and conquer*, algoritma *breath first search* dan *depth first search*, algoritma runut balik (*backtracking*), algoritma *branch and bound*, algoritma pencocokan string, dan algoritma program dinamis. Untuk disk scheduling SSTF ini kita akan menggunakan algoritma greedy.

1. Algoritma Greedy

Algoritma greedy adalah salah satu algoritma yang banyak digunakan meskipun algoritma ini tidak selalu menghasilkan solusi optimal. Hal ini dikarenakan algoritma ini masih berarah tidak seperti algoritma brute force yang mengiterasi tiap kemungkinan tetapi dalam pemikiran logikanya juga tidak terlalu rumit seperti algoritma yang memerlukan hirarki seperti

backtracking.

Menurut kamus oxford, greedy dapat didefinisikan sebagai berikut.

"Having an excessive desire or appetite for food."

yang berarti memiliki keinginan yang besar atau napsu yang besar (pada makanan). Dalam algoritma greedy juga definisi algoritma greedy tidak berbeda yaitu rakus atau dengan kata lain mengambil sebanyak-banyaknya. Algoritma greedy juga memiliki suatu prinsip yaitu "Take what you can get now".

Algoritma ini memiliki 2 buah tujuan yang tertama adalah untuk maksimasi dan minimasi. Maksimasi disini berarti memaksimalkan suatu keperluan dan minimasi berarti meminimalkan suatu keperluan. Untuk jelasnya akan dijelaskan di bagian selanjutnya.

Cara algoritma ini bekerja adalah mengambil data yang dibutuhkan dengan mencari nilai optimum lokal dari tiap-tiap langkah dengan harapan di akhir akan mendapatkan nilai optimum global. Tapi ada kalanya nilai optimum dari optimum lokal bukanlah merupakan nilai optimum global dari suatu permasalahan.

Menurut rinaldi, algoritma greedy ini memiliki 5 buah elemen.

1. Himpunan Kandidat
   Himpunan kandidat adalah suatu himpunan yang mungkin menjadi solusinya.
2. Himpunan Solusi
   Himpunan solusi adalah suatu himpunan yang merupakan hasil langkah-langkah yang merupakan jawaban dari algorima ini. Himpunan solusi juga merupakan himpunan bagian dari himpunan kandidat.
3. Fungsi Seleksi
   Fungsi seleksi adalah fungsi yang menentukan nilai terbaik yang akan dipilih sebagai solusi lokal yang optimum. Solusi lokal yang di dapat dari tiap langkah tidak akan diperiksa atau dipertimbangkan lagi setelah dipilih.
4. Fungsi Kelayakan
   Fungsi kelayakan memeriksa apakah himpunan yang sudah diambil setelah di seleksi totalnya melebihi batas atau tidak. Jika total dari himpunan itu melebihi batas maka akan dibuang dan tidak digunakan, tetapi jika masih terdapat dalam batas artinya akan dimasukkan ke dalam himpunan kandidat.
5. Fungsi Obyektif
   Fungsi obyektif adalah suatu fungsi yang meminimalkan atau memaksimalkan atau meminimalkan solusi. Fungsi ini juga merupakan inti dari masalah persoalan greedy.

Kita akan menggunakan contoh mencari koin untuk ditukar dengan jumlah minimum untuk mencapai nilai sekian. Misal himpunan kandidat adalah himpunan yang merepresentasikan nilai 100, 200, 500, dan 1000 paling sedikit mengandung 1 koin untuk setiap nilai. Himpunan solusinya adalah total nilai koin yang dipilihtepat sama jumlahnya dengan nilai uang yang ditukarkan. Fungsi seleksinya adalah pilihlah koin yang bernilai tertinggi dari himpunan kandidat yang tersisa. Fungsi kelayakannya adalah memeriksa apakah nilai total dari himpunan koin yang dipilih tidak melebihi jumlah uang yang harus dibayar. Fungsi obyektifnya adalah jumlah koin yang digunakan minimum.

Solusi yang diberikan algoritma greedy tidak selalu merupakan hasil yang optimum tapi pasti merupakan hasil yang pesudo-optimum karena algoritma ini tidak beroperasi secara menyeluruh terhadap alternatif solusi yang ada. Selain itu terdapat pula banyak fungsi seleksi yang berbeda. Sehingga untuk mendapatkan solusi optimal maka fungsi seleksi yang dipilihlah haruslah fungsi seleksi yang benar.

Algoritma greedy memang belum tentu memberikan solusiyang optimal, tetapi algoritma ini sering digunakan untuk menghasilkan solusi hampiran, bukan untuk menghasilkan solusi yang eksak.

Ada banyak masalah yang dapat diselesaikan dengan algoritma ini misalnya masalah activity selection problem, masalah integer knapsack, atau misalnya masalah job scheduling.

Sekarang kita akan melihat masalah integer knapsack. Dalam permasalahan ini, terdapat 3 jenis pembagian algoritma greedy yaitu

1. Greedy by profit
   Pada setiap langkah, akan dipilih suatu objek yang memiliki keuntungan terbesar. Greedy disini berarti mengambil keuntungan sebesar mungkin dengan langsung melihat untung terbesar.
2. Greedy by weight
   Pada setiap langkah, akan dipilih suatu objek yang memiliki berat yang paling kecil. Greedy disini berarti mengambil keuntungan terbesar dengan memilih berat terkecil yang menyebabkan akan makin banyak barang yang dapat dibawa. Algoritma ini berasumsi bahwa makin banyak bawang yang dibawa, maka akan makin besar keuntungannya.
3. Greedy by density
   Pada setiap langkah, akan dipilih suatu objek yang memiliki nilai keuntungan per unit berat paling besar. Greedy disini berarti mengambil keuntungan terbesar dengan memlilih nilai keuntungan per unit berat paling besar.

Properti objek				*Greedy by*			Solusi
i	w_i	p_i	p_i/w_i	*profit*	*weight*	*density*	Optimal
1	6	12	2	0	1	0	0
2	5	15	3	1	1	1	1
3	10	50	5	1	0	1	1
4	5	10	2	0	1	0	0
Total bobot				15	16	15	15
Total keuntungan				65	37	65	65

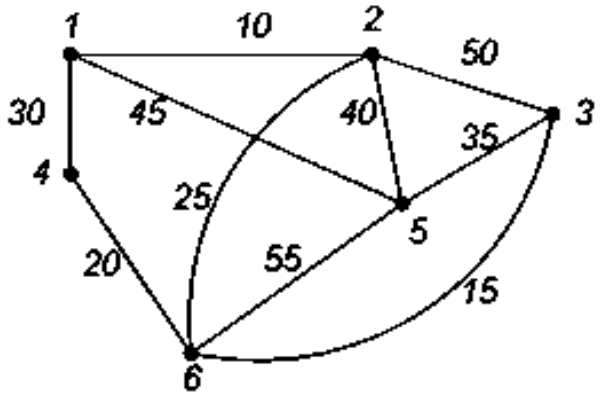
Tabel 1. Contoh penerapan algoritma greedy pada integer knapsack

Selain persolan integer knapsack, terdapat juga persoalan job scheduling yaitu persoalan memilih job yang akan dikerjakan oleh mesin sehingga hasil keuntungan yang dihasilkan optimum. Berikut contoh hasil daripada job scheduling.

Langkah	J	$F = \sum p_i$	Keterangan
0	{}	0	-
1	{1}	50	layak
2	{4,1}	50 + 30 = 80	layak
3	{4, 1, 3}	-	tidak layak
4	{4, 1, 2}	-	tidak layak

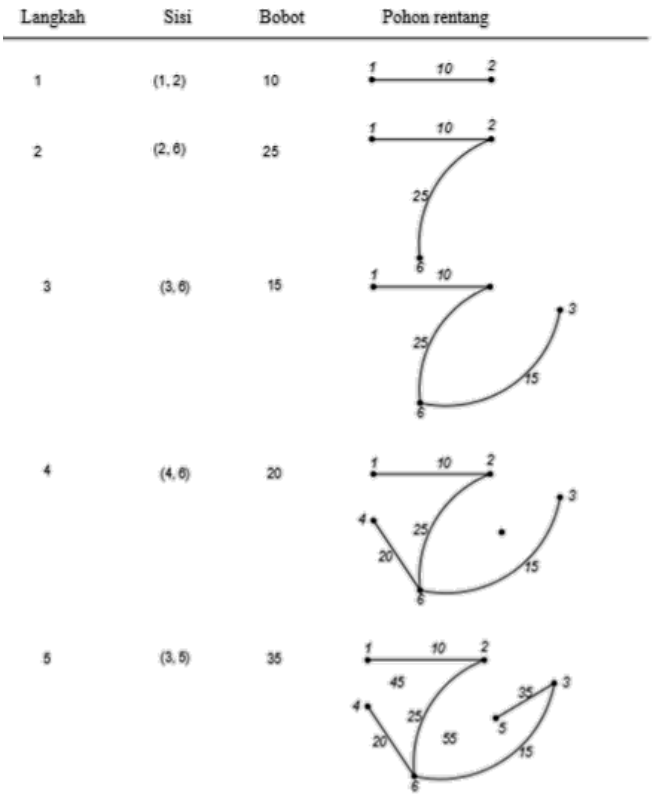
Tabel 2. Contoh penerapan algoritma greedy pada job scheduling

Kemudian contoh penggunaan algoritma greedy dapat kita lihat pada Algoritma prim seperti contoh yang akan diberikan pada gambar berikut.



Gambar 1. Contoh soal untuk algoritma prim

Dengan contoh soal seperti itu, maka akan diselesaikan dengan langkah-langkah yang akan dituliskan pada gambar berikut.

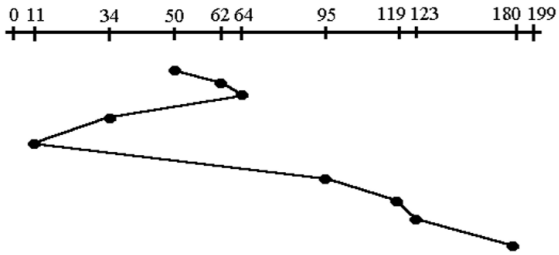


Gambar 2. Jawaban diatas dengan menggunakan persoalan prim

2. Algoritma Shortest Seek Time First

Algoritma shortest seek time first adalah algoritma yang didasarkan pada prinsip "Ambil yang terdekat". Maksudnya adalah untuk request yang diambil adalah request yang terdekat dengan head. Sama pula dengan namanya, algoritma ini mengambil request dengan seek time terendah dari semuanya.

Berikut adalah contoh dari penggunaan algoritma shortest seek time first dari request 62, 64, 34, 11, 95, 119, 123, dan 180 dengan head diawali dari 50.



Gambar 3. Gambar jalur urutan request yang diambil menggunakan algoritma shortest seek time first

Kita dapat melihat pada contoh soal diatas bahwa head pertama diawali dari 50. Kemudian algoritma Shortest time seek first akan mencari request dengan seek time terkecil. Karena dari gambar diatas kita sudah tahu urutan requestnya, maka kita tinggal membandingkan request tepat di sebelah kiri dan sebelah kanan request 50. Jika kita bandingkan, 50 tepat besebelahan dengan 34 dan 62. Kita akan mencari seektimenya terlebih dahulu. Jika dari head di 50 menuju ke request 34 maka akan memerlukan seek time 16. Sedangkan dari head 50 menuju request ke 62, kita memerlukan seek time 12. Dengan membandingkan seek time yang telah kita hitung yaitu 12 dan 16, kita tahu bahwa 12 lebih kecil dari 16 sehingga kita akan memilih urutan 62 setelah request ke 50.

Kemudian sekarang kita harus mencari urutan setelah request ke 62. Kita akan mencari dari sebelah kiri dan kanan. Pertama kita akan melihat dari sisi kiri, request terdekat dari sisi kiri adalah request ke 34 dan dari sisi kanan adalah request ke 64. Jika kita lihat selisihnya dari bagian kiri, kita melihat selisih dari head di 62 ke request ke 34 adalah 28. Sedangkan di bagian kanan, kita melihat selisih dari head di 62 ke request ke 64 adalah 2. Jika kita bandingkan dari selisih 2, dengan selisih 28 maka kita dapat melihat bahwa 2 lebih kecil dari 28 sehingga langkah selanjutnya yang akan diambil adalah 64.

Kemudian sekarang kita harus melihat lagi urutan setelah request ke 64. Kita akan mencari dari sebelah kiri dan kanan. Pertama kita akan melihat dari sisi kiri, request terdekat dari sisi kiri adalah request ke 23 dan dari sisi kanan adalah request ke 95. Jika kita lihat selisihnya dari bagian kiri, kita melihat selisih dari head di 64 ke request ke 34 adalah 30. Sedangkan di bagian kanan, kita melihat selisih dari head di 64 ke request ke 95 adalah 31. Jika kita bandingkan dari selisih 30, dengan selisih 31 maka kita dapat melihat bahwa 30 lebih kecil dari 31 sehingga langkah selanjutnya yang akan diambil adalah 34.

Kemudian sekarang kita harus melihat lagi urutan setelah request ke 34. Kita akan mencari dari sebelah kiri dan kanan.

Pertama kita akan melihat dari sisi kiri, request terdekat dari sisi kiri adalah request ke 11 dan dari sisi kanan adalah request ke 95. Jika kita lihat selisihnya dari bagian kiri, kita melihat selisih dari head di 34 ke request ke 11 adalah 23. Sedangkan di bagian kanan, kita melihat selisih dari head di 34 ke request ke 95 adalah 61. Jika kita bandingkan dari selisih 23, dengan selisih 61 maka kita dapat melihat bahwa 23 lebih kecil dari 61 sehingga langkah selanjutnya yang akan diambil adalah 11.

Kemudian sekarang kita harus melihat lagi urutan setelah request ke 11. Kita akan mencari dari sebelah kiri dan kanan. Pertama kita akan melihat dari sisi kiri, tidak ada request terdekat di kiri dan dari sisi kanan adalah request ke 95. Jika kita lihat selisihnya dari bagian kiri, tidak ada perbandingan yang dapat dilakukan. Sedangkan di bagian kanan, kita melihat selisih dari head di 11 ke request ke 95 adalah 84. Karena tidak ada perbandingan lagi yang dapat dilakukan maka langkah selanjutnya yang akan diambil adalah 95.

Kemudian sekarang kita harus melihat lagi urutan setelah request ke 95. Kita akan mencari dari sebelah kiri dan kanan. Pertama kita akan melihat dari sisi kiri, tidak ada request terdekat di kiri dan dari sisi kanan adalah request ke 119. Jika kita lihat selisihnya dari bagian kiri, tidak ada perbandingan yang dapat dilakukan. Sedangkan di bagian kanan, kita melihat selisih dari head di 95 ke request ke 119 adalah 24. Karena tidak ada perbandingan lagi yang dapat dilakukan maka langkah selanjutnya yang akan diambil adalah 119.

Kemudian sekarang kita harus melihat lagi urutan setelah request ke 119. Kita akan mencari dari sebelah kiri dan kanan. Pertama kita akan melihat dari sisi kiri, tidak ada request terdekat di kiri dan dari sisi kanan adalah request ke 123. Jika kita lihat selisihnya dari bagian kiri, tidak ada perbandingan yang dapat dilakukan. Sedangkan di bagian kanan, kita melihat selisih dari head di 119 ke request ke 123 adalah 4. Karena tidak ada perbandingan lagi yang dapat dilakukan maka langkah selanjutnya yang akan diambil adalah 123.

Kemudian sekarang kita harus melihat lagi urutan setelah request ke 123. Kita akan mencari dari sebelah kiri dan kanan. Pertama kita akan melihat dari sisi kiri, tidak ada request terdekat di kiri dan dari sisi kanan adalah request ke 180. Jika kita lihat selisihnya dari bagian kiri, tidak ada perbandingan yang dapat dilakukan. Sedangkan di bagian kanan, kita melihat selisih dari head di 123 ke request ke 180 adalah 57. Karena tidak ada perbandingan lagi yang dapat dilakukan maka langkah selanjutnya yang akan diambil adalah 180.

Tidak ada request yang dapat diambil lagi sehingga didapatkan urutan yang dihasilkan adalah 50, 62, 64, 34, 11, 95, 119, 123, dan 180 dengan head berakhir di 180. Jika dihitung seek time yang terjadi, kita dapat melihat seek timenya adalah 12 + 2 + 30 + 23 + 84 + 24 + 57 = 232.

III. IMPLEMENTASI GREEDY PADA SHORTEST SEEK TIME FIRST

Sekarang akan dibahas per langkah cara mengimplementasikan algoritma greedy pada algoritma Shortest seek time first.

Misalkan ada sebuah disk dengan space 0-50. Terdapat list request yaitu 10,5,35,50,40,15,0,45,25,30,20. Asumsikan head terdapat pada posisi ke 7.

Kita akan menentukkan elemen-elemen dari permasalahan ini terlebih dahulu.

1. Himpunan Kandidat
   Himpunan kandidat dari permasalahan ini adalah himpunan request dimana tiap himpunan terdiri dari semua request yang ada.
2. Himpunan Solusi
   Himpunan solusi adalah himpunan yang menjadi solusi dari permasalahan ini
3. Fungsi Seleksi
   Fungsi seleksi dari permasalahan ini adalah pilih request yang memiliki absolut dari selisih terkecil dari posisi head.
4. Fungsi Kelayakan
   Fungsi kelayakan dari permasalahan ini adalah apakah request yang dipilih sudah pernah dipilih sebelumnya.
5. Fungsi Obyektif
   Fungsi obyektif dari permasalahan ini adalah cari total seek time terkecil.

Sekarang akan dijelaskan per langkah pemilihan daripada Shortest Seek Time First menggunakan algoritma greedy. Diasumsikan fungsi kelayakan dilakukan secara langsung setiap kali melakukan langkah sehingga request yang sudah diakses tidak akan diakses lagi.

1. Langkah 1
   Dengan posisi head berada di 7.

Request	Jarak
0	7
5	2
10	3
15	8
20	13
25	18
30	23
35	28
40	33
45	38
50	43

Tabel 3. Langkah 1 SSTF

Dapat kita lihat bahwa request terdekat dari posisi head adalah request 5 dengan jarak 2. Maka posisi selanjutnya head berada pada posisi 5.

2. Langkah 2
   Dengan posisi head berada di 5.

Request	Jarak
0	5
10	5
15	10
20	15
25	20

30	25
35	30
40	35
45	40
50	45

Tabel 4. Langkah 2 SSTF

Dapat kita lihat bahwa request terdekat dari posisi head adalah adalah request 0 dan 10 dengan jarak 5. Karena jarak request tersebut sama, diasumsikan kita akan mengambil request dengan posisi ke 0 sehingga sekarang posisi head berada di 0.

3. Langkah 3

   Dengan posisi head berada di 0.

Request	Jarak
10	10
15	15
20	20
25	25
30	30
35	35
40	40
45	45
50	50

Tabel 5. Langkah 3 SSTF

Dapat kita lihat bahwa request terdekat dari posisi head adalah request 10 dengan jarak 10. Maka posisi head sekarang berada di 10.

4. Langkah 4

   Dengan posisi head berada di 10.

Request	Jarak
15	5
20	10
25	15
30	20
35	25
40	30
45	35
50	40

Tabel 6. Langkah 4 SSTF

Dapat kita lihat bahwa request terdekat dari posisi head adalah request 15 dengan jarak 5. Maka posisi head sekarang berada di 15.

5. Langkah 5

   Dengan posisi head berada di 15.

Request	Jarak
20	5
25	10
30	15
35	20
40	25
45	30
50	35

Tabel 7. Langkah 5 SSTF

Dapat kita lihat bahwa request terdekat dari posisi head adalah request 20 dengan jarak 5. Maka posisi head sekarang berada di 20.

6. Langkah 6

   Dengan posisi head berada di 20.

Request	Jarak
25	5
30	10
35	15
40	20
45	25
50	30

Tabel 8. Langkah 6 SSTF

Dapat kita lihat bahwa request terdekat dari posisi head adalah request 25 dengan jarak 5. Maka posisi head sekarang berada di 25.

7. Langkah 7

   Dengan posisi head berada di 25.

Request	Jarak
30	5
35	10
40	15
45	20
50	25

Tabel 9. Langkah 7 SSTF

Dapat kita lihat bahwa request terdekat dari posisi head adalah request 30 dengan jarak 5. Maka posisi head sekarang berada di 30.

8. Langkah 8

   Dengan posisi head berada di 30.

Request	Jarak
35	5
40	10
45	15
50	20

Tabel 10. Langkah 8 SSTF

Dapat kita lihat bahwa request terdekat dari posisi head adalah request 35 dengan jarak 5. Maka posisi head sekarang berada di 35.

9. Langkah 9

   Dengan posisi head berada di 35.

Request	Jarak
40	5
45	10
50	15

Tabel 11. Langkah 9 SSTF

Dapat kita lihat bahwa request terdekat dari posisi head adalah request 40 dengan jarak 5. Maka posisi head sekarang berada di 40.

10. Langkah 10

    Dengan posisi head berada di 40.

Request	Jarak
45	5
50	10

Tabel 12. Langkah 10 SSTF

Dapat kita lihat bahwa request terdekat dari posisi head adalah request 45 dengan jarak 5. Maka posisi head sekarang berada di 45.

11. Langkah 11

Dengan posisi head berada di 45.

Request	Jarak
50	10

Tabel 13. Langkah 11 SSTF

Dapat kita lihat bahwa request terdekat dari posisi head adalah request 50 dengan jarak 5. Maka posisi head sekarang berada di 50.

Semua request sudah terpenuhi, sekarang akan ditunjukkan tabel total jarak yang ditempuh oleh head (total seek time). Posisi head pertama di 7.

Urutan ke-	Request	Jarak
1	5	2
2	0	5
3	10	10
4	15	5
5	20	5
6	25	5
7	30	5
8	35	5
9	40	5
10	45	5
11	50	5
	Total	57

Tabel 14. Tabel Seek Time dari SSTF

Maka, langkah yang akan diambil oleh head dalam kasus ini adalah 5 – 0 – 10 – 15 – 20 – 25 – 30 – 35 – 40 – 45 – 50 dengan nilai total seek time 57.

IV. Analisis

Jika kita lihat pada contoh di point 4, kita dapat melihat bahwa semua aspek dalam algoritma greedy haruslah mengikuti elemen-elemen yang ada. Fungsi kelayakan diterapkan di awal karena untuk langsung menyeleksi keadaan dimana request bisa kembali ke posisi semula. Bahkan jika pada algoritma komputer dapat menyebabkan busy waiting (infinite loop).

V. Kesimpulan

Penggunaan algoritma greedy dapat digunakan untuk mengimplementasikan algoritma shortest seek time first pada disk scheduling. Algoritma greedy juga dapat menghasilkan solusi yang optimum karena pada hakekatnya, shortest seek time first adalah implementasi nyata dari algoritma greedy yang dapat mencari nilai secara langsung benar, bukan merupakan aproksimasi dari suatu masalah.

References

[1] Munir, Rinaldi. Mengambil data dari slide pada laman http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Greedy-(2018).pdf dikutip pada tanggal 13 mei 2018.

[2] Silberschatz, Abraham. Operating System Concepts. Wiley, 2013.

[3] Oxford University. Mengambil data dari laman https://en.oxforddictionaries.com/definition/greedy dikutip tanggal 13 mei 2018

[4] Ikaprama. Mengambil data dari laman http://ikaprama.org/headline/perkembangan-teknologi-informasi-menuju-indonesia-terkoneksi/ dikutip tanggal 14 mei 2018

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 14 Mei 2018

William Juniarta Hadiman - 13516026