Penerapan Algoritma Greedy Pada Pembacaan Blok di Hard Disk

Muhammad Furqan Habibi 13511002[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]furqan.habibi@students.itb.ac.id

***Abstrak*—Pada makalah ini akan dibahas salah satu strategi algoritma yang idenya cukup sederhana, namun memiliki pengaplikasian yang luas dan implementasi yang tidak sulit. Algoritma ini dikenal dengan nama algoritma greedy. Algoritma Greedy merupakan algoritma yang digunakan pada masalah-masalah optimasi. Biasanya masalah optimasi berkaitan dengan pencarian solusi yang memberikan hasil yang maksimum atau minimum. Di makalah ini juga akan dibahas salah satu implementasi algoritma greedy yang memiliki peran yang sangat penting di arsitektur komputer. Di sini algoritma greedy berperan dalam menentukan jalur mana yang akan diambil dalam pembacaan blok di harddisk.**

***Index Terms*—Algoritma Greedy, Optimasi, Pembacaan blok.**

I. PENDAHULUAN

Komputer adalah sebuah perangkat *general purpose* yang dapat di program untuk melakukan operasi-operasi logika dan aritmatik. Pada awal perkembangannya, komputer digunakan pada bidang-bidang kritikal yang menuntut ketepatan seperti di bidang militer. Bahkan alasan berkembangnya komputer dapat dikatakan didasari dari alasan militer. Pada saat itu, komputer digunakan sebagai media penyaluran informasi strategis, untuk menganalisis data dalam jumlah banyak, untuk menyimpan data-data kritikal yang bersifat rahasia, dan masih banyak lagi.

Setelah masa perang dunia berakhir, komputer mulai beralih fungsi menjadi mesin yang lebih *general purpose*. Komputer tidak lagi hanya digunakan oleh organisasi-organisasi militer, namun mulai merambah ke bidang-bidang aplikasi lain. Pada awalnya komputer mulai diperkenalkan di perguruan-perguruan tinggi sebagai sarana untuk melakukan komputasi yang besar, yang tidak mungkin dilakukan oleh manusia. Kemudian dari penggunaannya di dunia akademisi tersebut, mulai dirasakan bahwa komputer juga dapat dimanfaatkan di bidang lain seperti dunia bisnis. Dari sini kemudian komputer mulai digunakan sebagai alat untuk mengorganisir dan mengelola data. Pengolahan data yang sebelumnya dilakukan secara manual menjadi sangat terbantu dengan dilakukannya otomasi oleh komputer.

Dari tujuan pengorganisasian dan pengolahan data inilah kemudian berkembang teknologi penyimpanan data. Pada awal perkembangannya teknologi penyimpanan data hanya dapat menyimpan data dengan kapasitas kecil. Semakin lama perkembangan hardware dan software komputer yang semakin kompleks, serta tuntutan pengelolaan data yang semakin banyak, maka dibutuhkan teknologi penyimpanan yang dapat menyimpan data yang lebih besar. Dari kebutuhan ini maka teknologi penyimpanan data mulai dikembangkan secara berkesinambungan.

Jika dilihat dari sejarahnya, berikut perkembangan teknologi penyimpanan data dari masa ke masa [1] :

- Magnetic Tape (1928)
- Hard Disk (1956)
- Music Tape (1963)
- Floppy Disk (1971)
- CD (1980)
- DVD (1995)
- Flash Disk (1999)
- SD Card (2000)
- Blue Ray (2003)
- SSD (2005)

Dari semua jenis teknologi penyimpanan data di atas, teknologi yang paling umum digunakan dan paling memenuhi kebutuhan penyimpanan data di era sekarang adalah Hard Disk. Hal ini disebabkan oleh beberapa hal, antara lain :

- Harga relative murah
- Cukup reliable
- Kapasitas penyimpanan besar

Oleh karena penggunaannya yang umum dan penggunanya yang paling banyak, maka produsen harddisk berusaha memaksimalkan kemampuan dari harddisk yang dibuatnya. Selain dengan menambah kapasitas penyimpanan, mereka juga berlomba-lomba memproduksi harddisk dengan waktu akses yang cepat. Di antara solusi untuk mempercepat waktu akses hard disk ini adalah dengan mengimplementasikan algoritma greedy dalam pembacaan file dari hard disk. Penjelasan lebih lanjut akan dibahas di subbab lain dari makalah ini.

II. DASAR TEORI

A. Algoritma Greedy

Algoritma greedy adalah salah satu algoritma yang paling umum digunakan dalam menyelesaikan masalah optimasi. Dalam masalah optimasi, dibutuhkan penyelesaian masalah yang dapat meberikan hasil yang maksimum atau minimum pada masalah tersebut. Contoh masalah yang paling sederhana adalah masalah penukaran uang. Dengan diberikan sejumlah pecahan koin dengan masing-masing koin tersedia cukup banyak, tentukan banyak koin yang paling sedikit untuk ditukarkan dengan suatu nilai uang tertentu. Masalah ini adalah masalah optimasi, lebih tepatnya masalah minimasi.

Algoritma greedy meberikan solusi dalam bentuk langkah per langkah. Pada setiap langkah, terdapat banyak pilihan solusi yang dapat diambil. Semua pilihan tersebut dapat berkontribusi untuk menghasilkan suatu solusi. Dengan algoritma greedy, pada setiap langkahnya, diambil solusi yang paling optimum (maksimum/minimum) untuk langkah itu. Pengambilan ini tidak memperhatikan pengambilan langkah sesudahnya, sebelumnya, atau pengambilan langkah yang lain. Pengambilan pilihan ini didasarkan pada harapan bahwa dengan diambilnya pilihan optimum pada langkah sekarang, maka pengambilan pilihan untuk semua langkah juga akan memberikan hasil yang optimum. Faktanya harapan ini tidak selalu benar, bahkan bisa saja pengambilan langkah ini tidak menghasilkan solusi. Namun pada kasus yang umum, penggunaan algoritma greedy memberikan hasil yang cukup mengestimasi hasil optimum yang sebenarnya. Bahkan untuk beberapa kasus, penerapan algoritma greedy dapat dibuktikan selalu menghasilkan solusi yang optimum.

A.1. Skema Umum Algoritma Greedy

Dalam memeecahkan masalah optimasi dengan algoritma greedy, masalah dapat diklasifikasikan menjadi elemn-elemen berikut [2] :

1. Himpunan kandidat, C
   Himpunan ini berisi elemen-elemen yang akan dipilih untu menjadi solusi.
2. Himpunan solusi, S
   Himpunan ini berisi elemen-elemen yang telah dipilih untuk menjadi solusi
3. Fungsi seleksi
   Fungsi yang pada seiap langkahnya memilih elemen dari C yang bersifat optimum pada langkah tersebut. Elemen tersebut kemudian dimasukkan ke S. Elemen yang sudah dipilih pada suatu langkah tidak akan dipertimbangkan lagi pada langkah selanjutnya.
4. Fungsi kelayakan
   Fungsi ini memeriksa apakah gabungan suatu elemen dari C dan semua elemen dari S tidak melanggar kendala.
5. Fungsi obyektif
   Fungsi ini adalah fungsi yang menjadi dasar meminimumkan atau memaksimumkan solusi.

A.2. Penerapan Algoritma Greedy

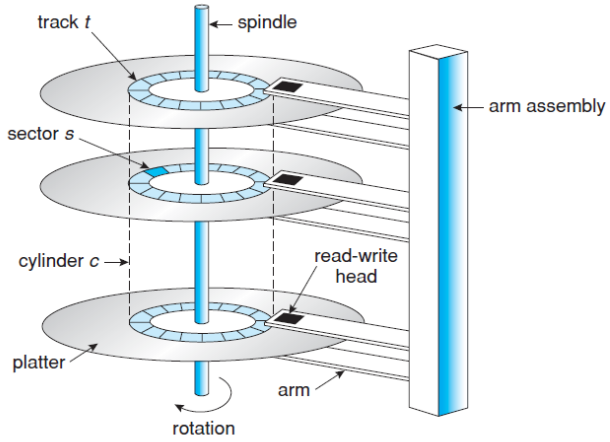
Pada ilmu komputer, terdapat banyak permasalahan yang dapat dipecahkan dengan cukup baik dengan algoritma greedy. Seperti yang dijelaskan sebelumnya, masalah-masalah ini selalu merupakan masalah pencaraian solusi yang maksimum atau minimum. Namun algoritma greedy tidak selalu menghasilkan hasil yang optimum, sehingga pada masalah yang solusinya harus eksak maka perlu digunakan algoritma lain. Meskupin demikian, algoritma greedy tetap menjadi pilihan yang baik dalam memecahkan masalah komputasi yang berkaitan dengan optimasi.

Di antara masalah yang umum dipecahkan dengan algoritma greedy adalah :

1. Masalah penukaran uang
2. Minimisasi waktu di dalam sistem penjadwalan
3. Activity selection problem
4. Integer knapsack
5. Fractional Knapsack
6. Penjadwalan job dengan tenggat waktu
7. Pohon merentang minimum
8. Lintasan terpendek
9. Pemampatan data dengan Algoritma Huffman
10. Pecahan Mesir
11. Dll.

B. Hard Disk (Magnetic Disk)

Seperti yang telah dijelaskan pada subbab pendahuluan, teknologi penyimpanan yang paling umum digunakan pada sistem komputer modern adalah hard disk. Secara konsep, sebenarnya arsitektur hard disk cukup sederhana. Di dalam sebuah hard disk terdapat beberapa cakram magnetic. Setiap cakram tersebut memiliki bentuk yang datar dan bulat, mirip seperti sebuah CD. Cakram hard disk pada umumnya memiliki diameter antara 1.8 hingga 3.5 inci atau antara 4.572 hingga 8.89 cm. Kedua permukaan cakram tersbut dilapisi oleh material yang bersifat magnetik. Material inilah yang dapat menyimpan data. Kita menyimpan data dengan merekamnya pada permukaan cakram ini secara magnetic. Karena sifat magnetisnya dalam penyimpanan data, maka hard disk dapat tetap menyimpan data meskipun tidak dialiri listrik. Karakteristik ini dikenal dengan istilah *nonvolatile storage*. Hal ini berbeda dengan teknologi penyimpanan lain seperti RAM (*Random Access Memory*) yang kehilangan datanya ketika aliran listrik diputus.



Gambar 1

Sebuah *head* (kepala) di atur sedemikian sehingga melayang di atas permukan setiap cakram. *Head* ini berfungsi untuk melakukan pembacaaan dan penulisan ke hard disk. *Head* ini terpasang pada sebuah *arm* (lengan) yang menghubungkan seluruh *head* harddisk dan menggerakkan mereka secara bersamaan. Permukaan dari cakram hard disk ini dibagi menjadi *track-track* sirkular. Selanjutnya setiap *track* tersebut dibagi lagi menjadi *sector-sector*. Kumpulan *track* yang berada pada posisi yang sama untuk setiap *arm* nya disebut sebagai *cylinder*. Biasanya terdapat ribuan *cylinder* yang ada pada sebuah hard disk, dengan setiap *track* mengandung ratusan *sector*. Jika kerapatan setiap *cylinder*, *track*, *sector*, dikalikan akan menghasilkan jumlah total kapasitas yang didukung suatu hard disk. Kapasitas penyimpanan suatu hard disk pada umumnya diukur dalam satuan GB *(gigabytes)*.

Ketika sebuah hard disk digunakan, motor harddisk tersebut akan memutar seluruh cakram dengan kecepatan tinggi. Kebanyakan hard disk berputar sebanyak 60 sampai 250 putaran per detik, satuan yang bisa digunakan adalah RPM *(rotation per minute)*. Hard disk yang umum biasanya memiliki kecepatan putar 5,400, 7,200, 10,000, dan 15,000 RPM.

Kecepatan hard disk biasa dikelompokkan menjadi dua kecepatan. Kecepatan itu adalah :

1. Kecepatan transfer data.
   Kecepatan transfer data adalah kecepatan data mengalir antara hard disk dan komputer.
2. Waktu akses acak *(random acess time).*
   Waktu ini terbagi lagi menjadi dua, yaitu :
   - *Seek time*
     Waktu yang dibutuhkan untuk menggerakan *arm* dari disk ke *cylinder* yang diinginkan
   - *Rotational latency*
     Waktu yang dibutuhkan untuk memutar cakram hingga *sector* yang diinginkan berada tepat dibawah head.

Karena *head* dari disk melayang di atas cakram dengan jarak yang sangat dekat (dalam satuan micron), maka akan terjadi bahaya jika *head* bersentuhan langsung dengan permukaan cakram. Meskipun permukaan cakram dilengkapi dengan lapisan pelindung, masih terdapat kemungkinan bahwa suatu saat *head* akan merusak permukaan magnetic cakram. Situasi tersebut disebut *head crash*. Jika *head crash* terjadi, biasanya akan berakibat fatal pada hard disk, dan hard disk tidak dapat diperbaiki lagi.

III. Pengaplikasian Greedy Pada Penjadwalan Pembacaan Blok Di Harddisk

Seperti yang telah dijelaskan sebelumnya, terdapat dua jenis spesifikasi kecepatan pada harddisk, yaitu *seek time* dan *rotational latency*. Pada algoritma penjadwalan pembacaan blok di hard disk, algoritma ini berusaha menentukan jadwal mana yang meminimumkan *seek time* yang diperlukan dalam melakukan pambacaan sekumpulan request terhadap beberapa blok. Sebagai contoh, misal pada suatu waktu terdapat request yang dilakukan oleh sistem operasi untuk pembacaan hard disk di blok-blok yang ada pada silinder berikut :

98, 183, 37, 122, 14, 124, 65, 67

Kemudian asumsikan bahwa saat awal request-request tersebut diterima, *head* dari disk berada pada silinder 53. Pertanyaannya adalah, bagaimanakah penjadwalan pembacaan blok-blok tersebut, jika diinginkan seek time total dalam melayani seluruh pembacaan blok adalah minimum? Perhatikan bahwa sebenarnya masalah ini adalah masalah optimasi, dengan obyektif meminimumkan seek time yang terjadi selama melayani request-request tersebut. Seperti yang telah dijelaskan panjang lebar di subbab sebelumnya, salah satu algoritma yang cukup baik untuk menyelesaikan masalah optimasi adalah algoritma greedy. Di mana algoritma greedy dapat menghasilkan himpunan solusi yang cukup baik dalam mengestimasi solusi yang paling optimum. Selanjutnya, bagaimanakah cara menerapkan greedy pada permasalahan ini?

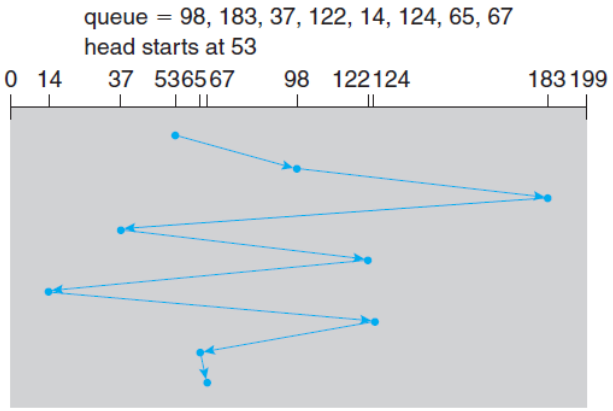
Terdapat berbagai strategi greedy yang dapat diterapkan untuk menyelesaikan masalah penjadwalan pembacaan blok ini. Di antara strategi tersebut ada yang merupakan solusi umum yang diakui para perancang sistem operasi, sehingga diterapkan secara langsung pada sistem operasi untuk menangani masalah penjadwalan ini. Pada sistem operasi dengan sumber terbuka, solusi dengan strategi greedy tersebut dapat dilihat dan dipelajari oleh umum, bahkan dapat di perbaharui jika ada yang menemukan strategi yang lebih baik.

Di antara strategi-strategi yang umum digunakan adalah FCFS, yang pada dasarnya merupakan strategi *greedy by request time*, SSTF yang merupakan strategi *greedy by seek time*, SCAN yang merupakan strategi *greedy by position*, C-SCAN yang merupakan modifikasi SCAN.[3]

A. FCFS (First Come First Serve)

Seperti yang telah saya utarakan di atas, FCFS adalah salah satu strategi greedy untuk penjadwalan pembacaan disk, di mana pada setiap langkahnya dipilih blok yang

datang meminta request terlebih dahulu. Artinya yang dipilih adalah yang *request time*-nya yang paling kecil. Berikut penjelasan algoritma FCFS.



Gambar 2

Pada saat terjadi request kepada harddisk, request tersebut akan ditampung pada sebuah *request queue*. Request yang datang terlebih dahulu akan berada di depan, sedangkan request yang datangnya di belakang akan berada di belakang. Selanjutnya jika harddisk telah siap untuk melayani request tersebut, head akan langsung bergerak dengan mengikuti penjadwalan FCFS. Artinya adalah, head akan bergerak ke blok yang datang pertama, melayani requestnya, kemudian head bergerak ke blok yang datang kedua, melayani requestnya, demikian seterusnya hingga *request queue* kosong.

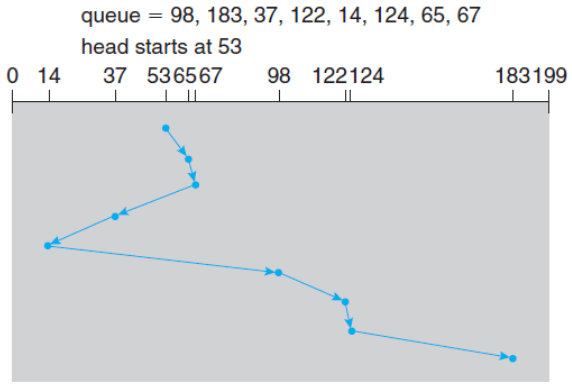
Selanjutnya misalkan diberikan contoh seperti di gambar 2 di atas. Bagaimanakah gerakan head jika menggunakan algoritma FCFS? Pertama head yang berada di blok pada silinder 53 akan bergerak ke blok yang berada di kepala queue, yaitu ke silinder 98. Setelah head selesai melakukan operasi pada blok tersebut, maka ia akan dihapus dari queue. Selanjutnya yang sekarang berada di kepala queue adalah blok dengan silinder no 183. Maka head akan bergerak dari silinder 98 ke 183. Begitu seterusnya hingga semua blok telah selesai dilayani. Untuk mengukur performansi algoritma ini untuk kasus di atas, maka kita akan menghitung total seek time yang terjadi selama head melayani seluruh request. Total seek time ini adalah total panjang jalur pada gambar 2, yaitu :

Total seek time = (98-53) + (183-98) + (183-37) + (122-37) + (122-14) + (124-14) + (124-65) + (67-65)

Total seek time = 640

B. SSTF (Shortest Seek Time First)

SSTF adalah salah satu strategi greedy untuk penjadwalan pembacaan disk, di mana pada setiap langkahnya dipilih blok yang paling dekat dengan *head*. Artinya yang dipilih adalah yang selisih posisinya dengan posisi *head* paling kecil. Berikut penjelasan algoritma SSTF.



Gambar 3

Ketika harddisk telah siap untuk melayani request pada *request queue*, head akan bergerak dengan mengikuti penjadwalan SSTF. Artinya adalah, head akan bergerak ke blok yang jaraknya paling dekat dengan posisi head sekarang, melayani requestnya, kemudian head bergerak ke blok lain yang belum dilayani yang jaraknya paling dekat dengan posisi head sekarang, melayani requestnya, demikian seterusnya hingga *request queue* kosong.

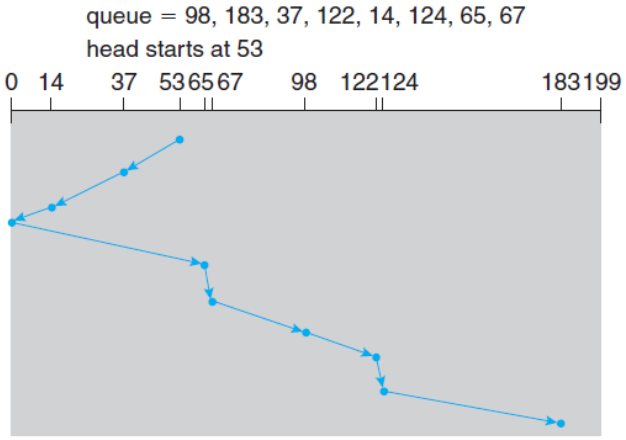
Selanjutnya misalkan diberikan contoh seperti di gambar 3 di atas. Bagaimanakah gerakan head jika menggunakan algoritma SSTF? Pertama head yang berada di blok pada silinder 53 akan bergerak ke blok yang terdekat. Dari inspeksi terhadap setiap blok, ditemukan blok terdekat adalah di silinder 65. Setelah head selesai melakukan operasi pada blok tersebut, maka akan dicari lagi blok terdekat dari head. Artinya harus dilakukan traversal lagi untuk menentukan blok yang terdekat. Begitu seterusnya hingga semua blok telah selesai dilayani. Untuk mengukur performansi algoritma ini untuk kasus di atas, maka kita akan menghitung total seek time yang terjadi selama head melayani seluruh request. Total seek time ini adalah total panjang jalur pada gambar 3, yaitu :

Total seek time = (65-53) + (67-65) + (67-37) + (37-14) + (98-14) + (122-98) + (124-122) + (183-124)

Total seek time = 236

C. SCAN

SCAN adalah salah satu strategi greedy untuk penjadwalan pembacaan disk, di mana pada setiap langkahnya dipilih blok yang posisinya berada di depan / di belakang head. Pemilihan belakang atau depan dilakukan oleh pengimplementasi algoritma. Ketika head telah mencapai ujung dari harddisk, maka arah pencarian dibalik. Berikut penjelasan algoritma SCAN dengan pemilihan blok di belakang head.



Gambar 4

Ketika harddisk telah siap untuk melayani request pada *request queue*, head akan bergerak dengan mengikuti penjadwalan SCAN. Karena dipilih bahwa SCAN dilakukan dengan melihat ke belakang, maka head akan bergerak ke blok yang posisinya di belakang posisi head sekarang, melayani requestnya, kemudian head bergerak ke blok lain yang posisinya di belakang posisi head sekarang, melayani requestnya, demikian seterusnya hingga *request queue* kosong.

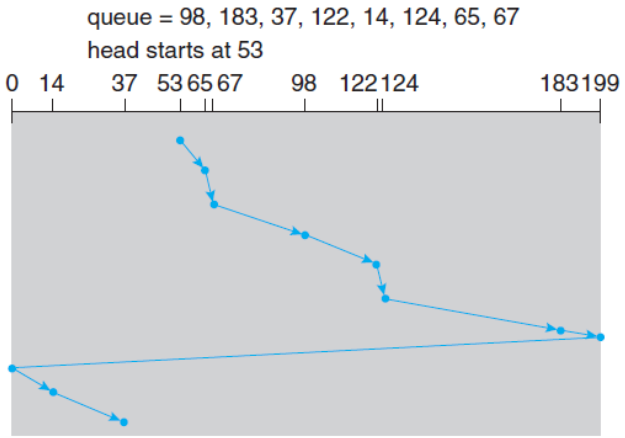
Selanjutnya misalkan diberikan contoh seperti di gambar 4 di atas. Bagaimanakah gerakan head jika menggunakan algoritma SCAN? Pertama head yang berada di blok pada silinder 53 akan bergerak ke blok terdekat yang ada di belakangnya. Dari inspeksi terhadap setiap blok, ditemukan blok terdekat adalah di silinder 37. Setelah head selesai melakukan operasi pada blok tersebut, maka akan dicari lagi blok terdekat yang berada di belakang head. Begitu seterusnya hingga semua blok telah selesai dilayani. Jika head telah mencapai pinggir dari hard disk, maka head akan berubah arah, dan selanjutnya memilih blok terdekat yang berada di depannya. Untuk mengukur performansi algoritma ini untuk kasus di atas, maka kita akan menghitung total seek time yang terjadi selama head melayani seluruh request. Total seek time ini adalah total panjang jalur pada gambar 4, yaitu :

Total seek time = (53-0) + (183-0)

Total seek time = 236

D. C-SCAN

C-SCAN sebenarnya adalah modifikasi dari SCAN. C-SCAN adalah salah satu strategi greedy untuk penjadwalan pembacaan disk, di mana pada setiap langkahnya dipilih blok yang posisinya berada di depan / di belakang head. Pemilihan belakang atau depan dilakukan oleh pengimplementasi algoritma. Ketika head telah mencapai ujung dari harddisk, maka head akan bergerak ke pinggir harddisk lainnya, lalu melanjutkan algoritmanya. Berikut penjelasan algoritma C-SCAN dengan pemilihan blok di depan head.



Gambar 5

Ketika harddisk telah siap untuk melayani request pada *request queue*, head akan bergerak dengan mengikuti penjadwalan C-SCAN. Karena dipilih bahwa C-SCAN dilakukan dengan melihat ke depan, maka head akan bergerak ke blok yang posisinya di depan posisi head sekarang, melayani requestnya, kemudian head bergerak ke blok lain yang posisinya di depan posisi head sekarang, melayani requestnya, demikian seterusnya hingga *request queue* kosong.

Selanjutnya misalkan diberikan contoh seperti di gambar 5 di atas. Bagaimanakah gerakan head jika menggunakan algoritma C-SCAN? Pertama head yang berada di blok pada silinder 53 akan bergerak ke blok terdekat yang ada di depannya. Dari inspeksi terhadap setiap blok, ditemukan blok terdekat adalah di silinder 65. Setelah head selesai melakukan operasi pada blok tersebut, maka akan dicari lagi blok terdekat yang berada di depan head. Begitu seterusnya hingga semua blok telah selesai dilayani. Jika head telah mencapai pinggir dari hard disk, maka head akan bergerak ke pinggir lainnya, kemudian melanjutkan algoritmanya seperti biasa. *Head* kembali akan mencari blok terdekat yang berada di depan posisinya sekarang. Untuk mengukur performansi algoritma ini untuk kasus di atas, maka kita akan menghitung total seek time yang terjadi selama head melayani seluruh request. Total seek time ini adalah total panjang jalur pada gambar 4, yaitu :

Total seek time = (199-53)+(199-0)+(37-0)

Total seek time = 382

IV. Analisis Strategi Greedy

Dari seluruh strategi greedy di atas, masing-masing menghasilkan solusi penjadwalan yang berbeda-beda. Demikian juga dengan total seek time yang dialami selama melayani seluruh *request*. Di sini saya akan mencoba sedikit menganalisis mengapa terjadi perbedaan tersebut, dan strategi mana yang paling baik untuk

penjadwalan pembacaan disk.

Dari semua strategi greedy di atas, yang memberikan total seek time yang paling minimum adalah algoritma SSTF dan SCAN. Mengapa bisa begitu?

Secara natural, dengan mengikuti asumsi dari harapan algoritma greedy, yaitu dengan memilih solusi optimum pada langkah sekarang, akan menghasilkan solusi optimum untuk semua solusi, kita dapat melihat bahwa SSTF adalah strategi yang terbaik. Hal ini karena, setiap langkahnya, SSTF memilih blok yang paling dekat dengan head. Artinya pada setiap langkahnya head akan bergerak seminimum mungkin. Atau dengan kata lain, pada setiap langkah head mngalami seek time yang paling minimum. Dengan meminimumkan seek time pada setiap langkah, maka diharapkan seek time total juga akan minimum.

Sementara mengapa SCAN juga memberi hasil yang baik? Hal ini bisa disebabkan karena pada algoritma SCAN, head tidak melakukan maju-mundur melewati jalur yang sebelumnya telah ia lewati. Pada SCAN, head akan selalu maju terus atau mundur terus, hanya berubah arah saat sampai di pinggir hard disk. Namun secara umum, SSTF-lah yang akan memberikan hasil yang lebih baik.

V. KESIMPULAN

1. Algoritma greedy merupakan salah satu algoritma yang cukup baik dalam memberikan solusi pada masalah optimasi
2. Algoritma greedy dapat diterapkan pada banyak masalah optimasi
3. Pada masalah penjadwalan pada pembacaan blok di hard disk, algoritma greedy dapat digunakan sebagai pendekatan untuk menghasilkan algoritma penjadwalan yang baik
4. Secara umum, SSTF memberikan hasil yang lebih baik dibandingkan FCFS, SCAN, dan C-SCAN.

REFERENCES

[1] http://www.zetta.net/history-of-computer-storage/

[2] Munir, Rinaldi. 2009. Strategi Algoritma. Bandung: Penerbit Institut Teknologi Bandung.

[3] Silberschatz, Abraham, *OperatingSystem Concepts*. New York: Wiley, 2008, ch. 10.

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, 20 Desember 2013

Muhammad Furqan Habibi (13511002)