PENERAPAN ALGORITMA BFS DFS DAN KNUTH-MORRIS-PRATT PADA PENCARIAN BERKAS DALAM KOMPUTER

Bobby H. Suryanaga / 13508022[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*if18022@if.itb.ac.id*

***Abstrak*—Pada komputer terdapat berbagai macam berkas dalam jumlah yang sangat besar. Struktur berkas pada komputer dapat direpresentasikan menggunakan pohon. Pencarian berkas tanpa menggunakan indeks dapat dilakukan dengan menggunakan algoritma penelusuran graf BFS dan DFS. Untuk pencarian kata kunci pada berkas dapat digunakan algoritma KMP.**

***Kata Kunci*—BFS, DFS, KMP, pencarian berkas.**

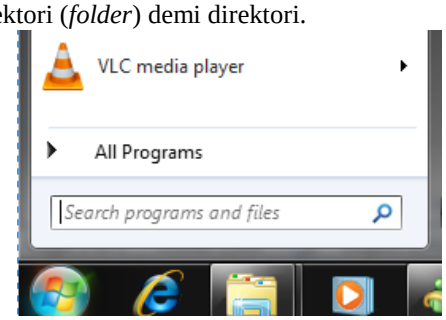
I. PENDAHULUAN

Sekarang ini, komputer tentunya telah menjadi bagian dari keseharian kita. Di mana-mana terdapat komputer yang tentunya memberikan manfaat besar pada hidup manusia. Bahkan ponsel kita juga dapat dikategorikan sebagai sebuah komputer sehubungan dengan fungsinya yang sudah berkembang, tidak hanya sebagai sarana untuk melakukan panggilan telepon, tetapi juga sebagai sarana mengakses internet, sarana multimedia, sarana penunjuk jalan (Global Positioning System / GPS), dan lain-lain.

II. KOMPUTER

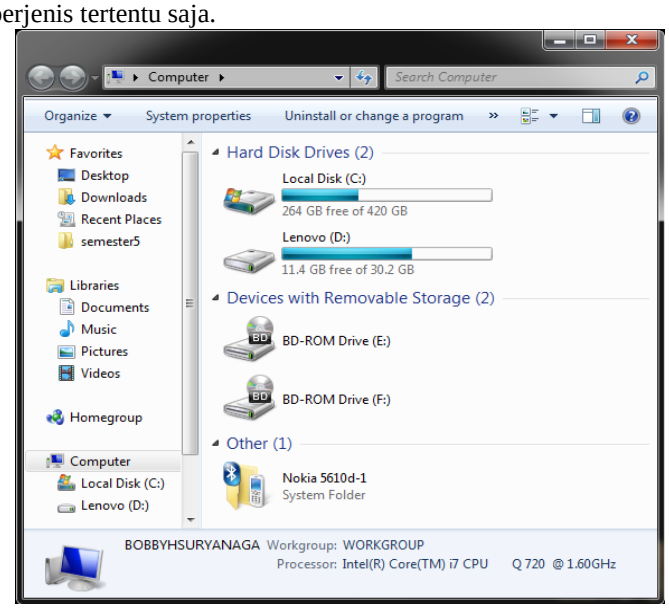
Pada komputer, tentunya terdapat berbagai macam berkas (file) yang membuat komputer dapat berjalan sebagaimana mestinya, dan juga berkas yang merupakan berkas pribadi pemiliki komputer yang dapat berupa teks, dokumen, konten multimedia, dan lain-lain.

Pada umumnya, sistem operasi saat ini sudah dilengkapi dengan fungsi pencarian berkas dengan menggunakan indeks yang memakan waktu cukup cepat, namun ada kerugian yang ditimbulkan dari sistem pencarian seperti itu, yaitu diperlukannya tempat penyimpanan untuk menampung berkas yang berisi indeks-indeks tersebut dan juga membebankan kerja prosesor untuk melakukan pengindeksan. Masalahnya, ukuran berkas tersebut cukup besar karena menampung indeks dari begitu banyak berkas. Untuk itu, bagi pengguna komputer yang jarang menggunakan fungsi pencarian tersebut dapat mematikan fungsi tersebut dan melakukan pencarian berkas secara klasik, yaitu dicari pada direktori (*folder*) demi direktori.



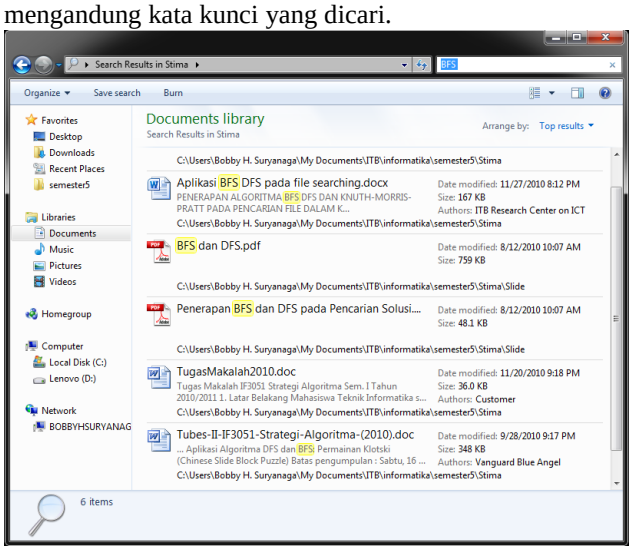
Gambar 1. Fitur pencarian menggunakan indeks pada sistem operasi Windows 7

Pencarian seperti itu masih digunakan hingga sekarang. Sebagai contoh, pada sistem operasi Windows 7, jika kita membuka explorer dan melakukan navigasi ke tempat yang tidak diindeks lalu melakukan pencarian, maka akan dilakukan pencarian berkas per berkas dan menawarkan untuk menambahkan berkas-berkas pada direktori tersebut ke indeks. Secara default, sistem operasi Windows 7 tidak melakukan pengindeksan pada direktori-direktori yang berhubungan dengan sistem. Sistem operasi Windows 7 juga memberikan pilihan pada pengguna untuk melakukan pengindeksan pada berkas berjenis tertentu saja.



Gambar 2. Explorer pada sisem operasi Windows 7 dengan fitur pencarian yang ada di sisi kanan atas jendela

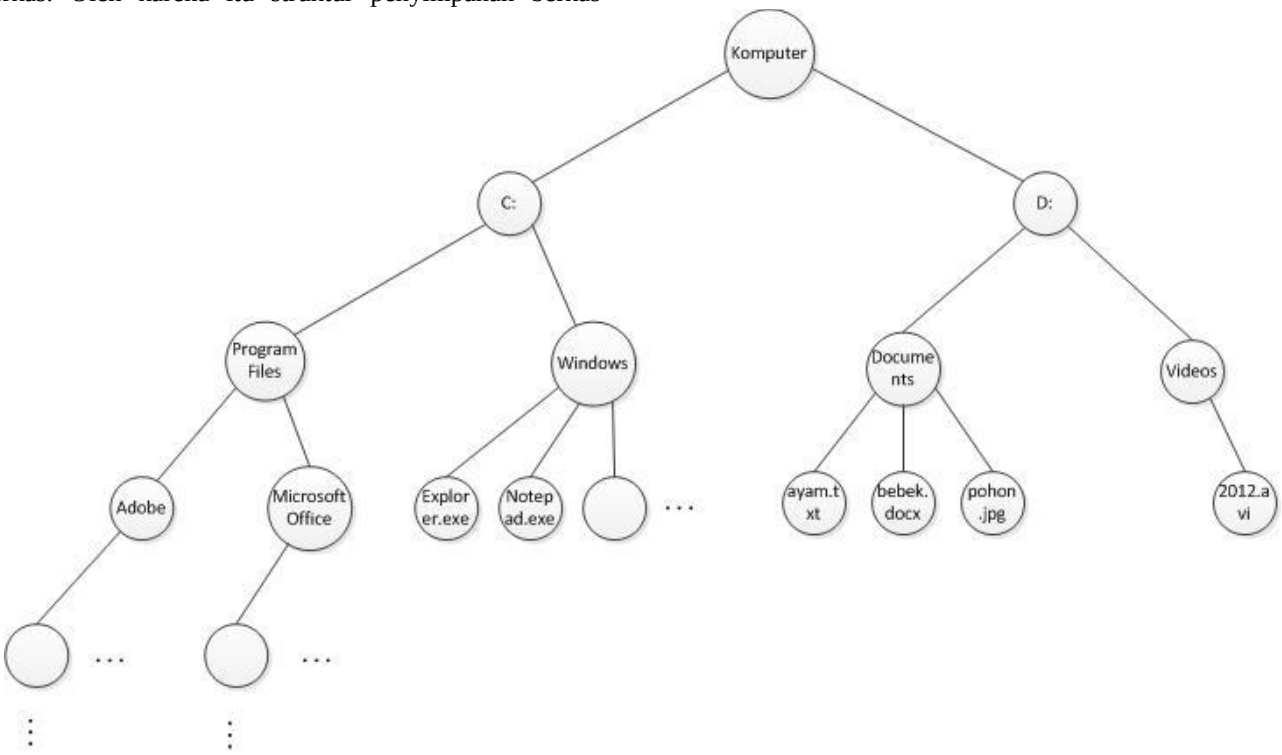
Jika pada suatu direktori dilakukan pencarian, explorer akan menampilkan seluruh berkas dan direktori yang mengandung kata kunci yang dicari.



Gambar 3. Contoh hasil pencarian pada sistem operasi Windows 7

Seperti kita ketahui penyimpanan berkas pada komputer menggunakan direktori-direktori yang di dalamnya mungkin terdapat direktori lain dan juga berkas. Oleh karena itu struktur penyimpanan berkas tersebut dapat kita gambarkan seperti pohon yang memiliki banyak anak. Sebagai contoh, sebuah komputer memiliki 2 tempat penyimpanan berkas, drive C: dan D:. Pada drive C: terdapat 2 buah direktori, direktori "Windows" dan direktori "Program Files". Pada direktori Windows terdapat banyak berkas yang berhubungan dengan kerja operating system Windows, sebut saja explorer.exe, notepad.exe dan masih banyak lainnya. Pada direktori Program Files terdapat banyak direktori yang berisi program-program tambahan di luar OS, sebut saja direktori Adobe, direktori Microsoft Office, dan lainnya yang pada tiap-tiap direktori terdapat berbagai macam berkas yang mendukung berjalannya program yang bersangkutan. Pada drive D: terdapat 2 direktori, direktori Documents dan Videos. Pada direktori Documents terdapat 3 berkas, ayam.txt, bebek.docx, dan pohon.jpg. Pada direktori Videos terdapat sebuah berkas bernama 2012.avi

Struktur berkas seperti di atas dapat digambarkan menjadi pohon sebagai berikut :



Gambar 4. Penggambaran struktur berkas pada computer dengan menggunakan pohon

III. ALGORITMA TRAVERSAL DI DALAM GRAF

A. Graf

Graf adalah pasangan antara himpunan simpul dan sisi. Sebuah graf tidak mungkin memiliki 0 buah simpul *(vertice/node)*, tetapi mungkin memiliki 0 buah sisi *(edge/ arc)*. Graf yang memiliki sebuah simpul tanpa sisi disebut graf trivial.[2]

B. Algoritma pencarian melebar (Breadth First Search / BFS)

Algoritma BFS merupakan mekanisme pencarian sebuah simpul pada graf dengan cara memulai pencarian pada sebuah simpul dan kemudian mengunjungi semua simpul yang bertetangga dengan simpul awal dan baru kemudian semua simpul yang belum dikunjungi yang bertetanggaan dengan simpul-simpul yang tadi

dikunjungi.

Algoritma BFS memerlukan sebuah antrian q untuk menyimpan simpul yang telah dikunjungi. Simpul-simpul yang telah dikunjungi suatu saat diperlukan sebagai acuan untuk mengunjungi simpul-simpul yang bertetangga dengannya. Tiap simpul yang telah dikunjungi masuk ke dalam antrian hanya satu kali.[1]

Algoritma BFS memerlukan tabel Boolean yang bernama `Dikunjungi` untuk menyimpan simpul yang telah dikunjungi agar tidak ada simpul yang dikunjungi lebih dari sekali. Deklarasi tabel tersebut di dalam kamus adalah [1]

```
Deklarasi
    Dikunjungi : array[1..n] of boolean
```

Algoritma untuk traversal graf selengkapnya adalah :

```
procedure BFS(input v:integer)
{ Traversal graf dengan algoritma pencarian BFS.
Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpul yang dikunjungi dicetak ke layar
}

Deklarasi
w : integer
q : antrian;

procedure BuatAntrian(input/output q : antrian)
{ membuat antrian kosong, kepala(q) diisi 0 }

procedure MasukAntrian(input/output q:antrian, input v:integer)
{ memasukkan v ke dalam antrian q pada posisi belakang }

procedure HapusAntrian(input/output q:antrian,output v:integer)
{ menghapus v dari kepala antrian q }

function AntrianKosong(input q:antrian) → boolean
{ true jika antrian q kosong, false jika sebaliknya }

Algoritma:
BuatAntrian(q) { buat antrian kosong }

write(v) { cetak simpul awal yang dikunjungi }
dikunjungi[v]←true { simpul v telah dikunjungi, tandai dengan true}
MasukAntrian(q,v) { masukkan simpul awal kunjungan ke dalam antrian}

{ kunjungi semua simpul graf selama antrian belum kosong }
while not AntrianKosong(q) do
   HapusAntrian(q,v) { simpul v telah dikunjungi, hapus dari antrian }
   for w←1 to n do
      if A[v,w] = 1 then { v dan w bertetangga }
         if not dikunjungi[w] then
            write(w) {cetak simpul yang dikunjungi}
            MasukAntrian(q,w)
            dikunjungi[w]←true
         endif
      endif
   endfor
endwhile
{ AntrianKosong(q) }
{[1]}
```

C. Algoritma pencarian mendalam (Depth First Search / DFS)

Algoritma DFS melakukan pencarian dengan cara menelusuri sebuah simpul awal ke sebuah tetangganya, dan kemudian menelusuri lagi sebuah tetangga dari simpul tersebut secara rekursif. Ketika semua simpul tetangga telah dikunjungi, pencarian dirunut-balik ke simpul terakhir yang dikunjungi yang memiliki tetangga lain yang belum dikunjungi dan pencarian mendalam dilakukan lagi pada simpul tersebut. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.[1]

Algoritma pencarian mendalam selengkapnya adalah sebagai berikut :

```
procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS
Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpulyang dikunjungi ditulis ke layar
}
Deklarasi
w : integer
Algoritma:
write(v)
dikunjungi[v]←true
for w←1 to n do
  if A[v,w]=1 then {simpul v dan simpul w bertetangga }
    if not dikunjungi[w] then
      DFS(w)
    endif
  endif
endfor
{[1]}
```

D. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt merupakan suatu algoritma pencarian string yang dikembangkan oleh D.E. Knuth bersama-sama dengan J. H. Morris dan V. R. Pratt. Pada algoritma brute force, setiap kali ditemukan

ketidakcocokan pattern dengan teks, maka pattern digeser satu karakter ke kanan. Sedangkan pada algoritma HMP, kita memelihara informasi yang digunakan untuk melakukan jumlah pergeseran. Algoritma menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak hanya satu karakter seperti pada algoritma brute force. [1]

Algortima KPM melakukan proses awal terhadap pattern P dengan meghitung fungsi pinggiran yang mengindikasikan pergerseran s terbesar yang mungkin dengan menggunakan perbandingan yang dibentuk sebelum pencarian string. Dengan adanya fungsi pinggiran ini dapat dicegah pergeseran yang tidak berguna, seperti halnya pada algoritma brute force.[1]

Fungsi *pinggiran b*(*j*) didefinisikan sebagai ukuran awalan terpanjang dari *P* yang merupakan akhiran dari *P*[1..*j*].

Algoritma fungsi pinggiran adalah sebagai berikut :

```
procedure HitungPinggiran(input m :
integer, P : array[1..m] of char,
output b : array[1..m] of integer)
{Menghitung nilai b[1..m] untuk
pattern P[1..m]}
Deklarasi
k, q : integer
Algoritma:
b[1] ← 0
q ← 2
k ← 0
for q←2 to m do
  while ((k>0) and (P[q]≠ P[k+1])) do
  k ← b[k]
  endwhile
  if P[q] =P[k+1] then
    k←k+1
  endif
  b[q]=k
endfor
{[1]}
```

Algoritma KMP selengkapnya adalah sebagai berikut :

```
procedure KMPsearch(input m, n :
integer, input P : array[1..m] of
char, input T : array[1..n] of char,
output idx : integer)
{Mencari kecocokan pattern P di dalam
teks T dengan algoritma Knuth-Morris-
Pratt. Jika ditemukan P di dalam T,
lokasi awal kecocokan disimpan di
dalam peubah idx.
Masukan : pattern P yang panjangnya m
dan teks T yang panjangnya n. Teks T
direpresentasikan sebagai string
(array of character)
Keluaran : posisi awal kecocokan
(idx). Jika P tidak ditemukan, idx = -
1.}

Deklarasi
i, j : integer
ketemu : Boolean
b : array[1..m] of integer
procedure HitungPinggiran(input m :
integer, P : array[1..m] of char,
output b : array[1..m] of integer)
{Menghitung nilai b[1..m] untuk
pattern P[1..m]}

Algoritma
HitungPinggiran(m, P, b)
j ← 0
i ← 1
ketemu ← false
while (i ≤ n and not ketemu) do

  while((j > 0) and (P[j+1]≠T[i])) do
    j ← b[j]
  endwhile

  if P[j+i] = T[i] then
    j ← j+1
  endif
  if j = m then
    ketemu ← true
  else
  i ← i+1
  endif

endwhile

if ketemu then
  idx ← i-m+1 {catatan: jika indeks
array dimulai dari 0, maka idx ←i-m}
else
  idx ← -1
endif
{[1]}
```

IV. APLIKASI ALGORITMA BFS, DFS, DAN KMP

Pada proses pencarian berkas atau direktori di dalam komputer, pencarian tidak dihentikan ketika sebuah berkas atau direktori ditemukan sesuai kata kunci yang dipakai, melainkan ke seluruh berkas dan direktori yang terdapat pada lingkup pencarian sehingga pada akhir pencarian didapatkan daftar berkas dan direktori yang memenuhi syarat pencarian. Bila suatu berkas yang terdapat dalam lingkup pencarian merupakan berkas teks atau berkas yang mengandung teks seperti word documents, pdf berkass, dan jenis-jenis berkas lainnya.

Pencarian berkas dapat dilakukan dengan menggunakan algoritma BFS dan KMP atau dengan algoritma DFS dan KMP. Pada pencarian berkas menggunakan algoritma BFS dan KMP digunakan sebuah antrian (queue) yang digunakan untuk menelusuri semua berkas dan direktori yang masuk pada areal pencarian. Areal pencarian merupakan sebuah bagian dari pohon

struktur berkas seperti pada gambar di atas yang merupakan pohon lagi. Sebagai contoh, areal pencarian yang valid adalah pada direktori Documents yang meliputi seluruh berkas dan direktori pada direktori Documents, yaitu ayam.txt, bebek.docx, dan pohon.jpg.

Proses penelusuran pohon menggunakan algoritma BFS adalah dengan memasukkan berkas dan direktori yang ada pada direktori awal pencarian ke dalam antrian, kemudian proses elemen pertama antrian. Jika elemen pertama antrian merupakan berkas yang mengandung teks, cari kata kunci pada nama berkas dan di dalam berkas tersebut. Jika elemen pertama merupakan berkas non teks, cari kata kunci pada nama berkas. Jika elemen pertama merupakan direktori, cari kata kunci pada nama direktori dan masukkan semua berkas dan direktori yang ada pada direktori tersebut ke belakang antrian. Proses pencarian kata kunci dilakukan dengan menggunakan algoritma KMP. Jika kata kunci ditemukan, tampilkan nama berkas atau direktori ke layar.

Pada penelusuran menggunakan BFS ini tidak diperlukan array of boolean Dikunjungi karena tiap berkas dan direktori tidak mungkin dikunjungi lebih dari sekali. Bila pada area pencarian terdapat berkas duplikat, maka kedua berkas akan tetap ditampilkan ke layar.

Berikut ini adalah prosedur untuk melakukan pencarian berkas dengan menggunakan algoritma BFS dan KMP dalam notasi pseudo code :

```
procedure CariBerkasBFS(input dir : string, input P : array[1..m] of char)
{Menampilkan seluruh berkas dan direktori pada direktori dengan path dir yang mengandung pola P ke layar menggunakan BFS.
Masukan : path direktori dir yang isinya akan dicari. Pola P yang merupakan kata kunci pencarian berkas
Keluaran : tidak ada, langsung menuliskan ke layar setiap berkas dan direktori yang mengandung pola P.}

Deklarasi
q : antrian
current : string
teksberkas : string
idx, m, n : integer

procedure BuatAntrian(input/output q : antrian)
{ membuat antrian kosong, kepala(q) diisi sring kosong }

procedure MasukkanIsiKeAntrian(input/output q:antrian, input v:string)
{ memasukkan path dari semua isi dari direktori v ke dalam antrian q pada posisi belakang }

procedure HapusAntrian(input/output q:antrian,output v:string)
{ menghapus v dari kepala antrian q }

function AntrianKosong(input q:antrian) → boolean
{ true jika antrian q kosong, false jika sebaliknya }

function Panjang(input S:string) → integer
{menghasilkan panjang dari string S}

function AdalahDirektori(input S:string) → boolean
{true jika path S merupakan direktori, false jika merupakan berkas}

function BerkasTeks(input file:string) → boolean
{true jika berkas dengan path file merupakan berkas yang mengandung teks}

procedure IsiTeks(input S:string, input/output T : string)
{memasukkan isi dari berkas teks S ke T}
{Catatan : proses memasukkan isi teks dari berkas memiliki mekanisme yang berbeda-beda tergantung jenis berkas teks tersebut.}

procedure Tampilkan(input S:string)
{menampilkan direktori atau berkas dengan path S ke layar)

procedure KMPsearch(input m, n : integer, input P : array[1..m] of char, input T : array[1..n] of char, output idx : integer)
{Mencari kecocokan pattern P di dalam teks T dengan algoritma Knuth-Morris-Pratt. Jika ditemukan P di dalam T, lokasi awal kecocokan disimpan di dalam peubah idx.
Masukan : pattern P yang panjangnya m dan teks T yang panjangnya n. Teks T direpresentasikan sebagai string (array of character)
Keluaran : posisi awal kecocokan (idx). Jika P tidak ditemukan, idx = -1.}

Algoritma
BuatAntrian(q) {buat antrian kosong}

MasukkanIsiKeAntrian(q,dir)
m ← Panjang(P)
while not AntrianKosong(q) do
  HapusAntrian(q,current)
```

```
  n ← Panjang(current)
  KMPsearch(m,n,P,current,idx)
  if idx ≠ -1 then
    Tampilkan(current)
  endif
  if AdalahDirektori(current) then
    MasukkanIsiKeAntrian(q,current)
  else if BerkasTeks(current) then
    IsiTeks(current,teksberkas)
    n ← Panjang(teksberkas)
    KMPsearch(m,n,teksberkas,P,idx)
    if idx ≠ -1 then
      Tampilkan(current)
    endif
  endif
endwhile

{catatan : tipe string dan array of
char dianggap sama}
```

Pada pencarian berkas menggunakan DFS, digunakan fungsi rekursif. Penelusuran dimulai dengan menelusuri isi direktori pencarian. Jika pada penelusuran isi direktori tersebut ditemukan direktori, lakukan penelusuran lagi hingga seluruh isi direktori tersebut ditelusuri. Jika isi seluruh direktori tersebut telah ditelusuri, lanjutkan penelusuran pada direktori sebelumnya.

Sama seperti pada pencarian berkas menggunakan BFS, setiap berkas dan direktori yang ditelusuri diperiksa apakah mengandung kata kunci dengan menggunakan algoritma KMP. Jika ya, tampilkan ke layar.

Pada algoritma pencarian berkas menggunakan DFS ini digunakan struktur data pohon beranak banyak dengan akarnya berisi elemen bertipe string.

Deklarasi

```
typedef struct {
     Akar : string
     Anak : array[1..N] of Pohon
} Pohon
```

Berikut ini adalah fungsi untuk melakukan pencarian berkas dengan menggunakan algoritma DFS dan KMP dalam notasi pseudo code :

```
function CariBerkasDFS(input
dir:string, input P : array[1..m] of
char) → Pohon
{Menampilkan seluruh berkas dan
direktori pada direktori dengan path
dir yang mengandung pola P ke layar
menggunakan BFS.
Masukan : path direktori dir yang
isinya akan dicari. Pola P yang
merupakan kata kunci pencarian berkas
Keluaran : Pohon yang merupakan
struktur berkas dari direktori dir}

Deklarasi
Tree : Pohon
teksberkas : string
Isi : array[1..J] of string
m, n, idx, i : integer

procedure BuatPohon(input/output T :
Pohon)
{membuat Pohon kosong, akar berisi
string kosong dengan 0 anak}

procedure IsiAkar(input/output
T:Pohon, input S:string)
{mengisi Pohon T dengan akar S }

procedure TambahAnak(input/output
T:Pohon, input A:Pohon)
{Menambahkan Pohon A sebagai anak dari
Pohon Tree}

function JumlahIsi(input S:string) →
integer
{menghasilkan banyaknya isi dari
direktori dengan path S, menghasilkan
0 jika path S merupakan berkas}

procedure MasukkanIsi(input S:string,
input/output AS:array[1..J] of string)
{Memasukkan path dari semua isi
direktori dengan path S ke array AS}

function Panjang(input S:string) →
integer
{menghasilkan panjang dari string S}

function BerkasTeks(input file:string)
→ boolean
{true jika berkas dengan path file
merupakan berkas yang mengandung teks}

procedure IsiTeks(input S:string,
input/output T : string)
{memasukkan isi dari berkas teks S ke
T}
{Catatan : proses memasukkan isi teks
dari berkas memiliki mekanisme yang
berbeda-beda tergantung jenis berkas
teks tersebut.}

procedure Tampilkan(input S:string)
{menampilkan direktori atau berkas
dengan path S ke layar)

procedure KMPsearch(input m, n :
integer, input P : array[1..m] of
char, input T : array[1..n] of char,
output idx : integer)
{Mencari kecocokan pattern P di dalam
teks T dengan algoritma Knuth-Morris-
Pratt. Jika ditemukan P di dalam T,
lokasi awal kecocokan disimpan di
dalam peubah idx.
Masukan : pattern P yang panjangnya m
```

```
dan teks T yang panjangnya n. Teks T
direpresentasikan sebagai string
(array of character)
Keluaran : posisi awal kecocokan
(idx). Jika P tidak ditemukan, idx = -
1.}

Algoritma
BuatPohon(Tree)
IsiAkar(Tree,dir)

m ← Panjang(P)
n ← Panjang(dir)
idx = KMPsearch(m,n,P,dir,idx)
if idx ≠ -1 then
  Tampilkan(current)
endif
if BerkasTeks(dir) then
  IsiTeks(dir, teksberkas)
  n ← Panjang(IsiTeks)
  KMPsearch(m,n,P,teksberkas,idx)
  if idx ≠ -1 then
    Tampilkan(dir)
  endif
endif

MasukkanIsi(dir,Isi)
i ← 1
while i ≤ JumlahIsi(dir) do
  TambahAnak(Tree,CariBerkasDFS(Isi[i]
,P))
  i ← i+1
endwhile

→ Tree

{catatan : tipe string dan array of
char dianggap sama}
```

V. KESIMPULAN

Proses pencarian berkas tanpa menggunakan indeks dapat dilakukan dengan menggunakan algoritma penelusuran graf BFS dan DFS dengan algoritma pencocokan string KMP untuk mencari kata kunci yang dimasukkan. Efisiensi kedua algoritma tidak dapat dibandingkan karena jumlah penelusuran bergantung sepenuhnya pada letak berkas.

VII. UCAPAN TERIMA KASIH

Atas terselesaikannya makalah ini, saya mengucapkan terima kasih kepada dosen pembimbing dalam perkuliahan Strategi Algoritma, Pak Rinaldi Munir, karena telah memberikan pengetahuan dasar yang memungkinkannya dibuatnya makalah ini,

DAFTAR PUSTAKA

[1] Ir. Munir, Rinaldi M.T., Strategi Algoritma. Bandung, Indonesia, Januari 2009,hlm.133-139

[2] Ir. Munir, Rinaldi M.T., Struktur Diskrit. Bandung, Indonesia, Januari 2008.

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, 29 April 2010

ttd

Bobby H. Suryanaga / 13508022