PRAKTIKUM

**STRATEGI ALGORITMA**

# DAFTAR ISI

[HAK CIPTA 1](#_bookmark0)

[KATA PENGANTAR 2](#_bookmark0)

[DAFTAR PENYUSUN 3](#_bookmark0)

[HALAMAN REVISI 4](#_bookmark0)

[HALAMAN PERNYATAAN 5](#_bookmark0)

[VISI DAN MISI PRODI TEKNIK INFORMATIKA 6](#_bookmark0)

[TATA TERTIB LABORATORIUM TEKNIK INFORMATIKA 7](#_bookmark0)

[DAFTAR ISI 10](#_bookmark1)

[DAFTAR GAMBAR 11](#_bookmark2)

[PRAKTIKUM 1: KOMPLEKSITAS ALGORITMA 12](#_bookmark3)

[PRAKTIKUM 2: ALGORITMA BRUTE FORCE 16](#_bookmark4)

[PRAKTIKUM 3: ALGORITMA GREEDY 20](#_bookmark6)

[PRAKTIKUM 4: ALGORITMA DIVIDE & CONQUER 22](#_bookmark7)

[PRAKTIKUM 5: ALGORITMA DECREASE & CONQUER 31](#_bookmark8)

[PRAKTIKUM 6: ALGORITMA BFS DAN DFS 35](#_bookmark10)

[PRAKTIKUM 7: ALGORITMA BACKTRACKING 45](#_bookmark19)

[PRAKTIKUM 8: ALGORITMA BRANCH AND BOUND 54](#_bookmark28)

[PRAKTIKUM 9: ALGORITMA A\* 64](#_bookmark35)

[PRAKTIKUM 10: ALGORITMA STRING MATCHING 72](#_bookmark41)

[DAFTAR PUSTAKA 79](#_bookmark48)

# DAFTAR GAMBAR

[Gambar 2.1 Ilustrasi perkalian dua matriks 17](#_bookmark5)

[Gambar 5.1. Contoh ilustrasi pengurutan decrease and conquer dengan selection sort 32](#_bookmark9)

[Gambar 6.1. Contoh graf berarah 35](#_bookmark11)

[Gambar 6.2. Algoritma BFS 36](#_bookmark12)

[Gambar 6.3 Algoritma DFS 37](#_bookmark13)

[Gambar 6.4 Graf berarah untuk soal Pre Test dan Post Test Praktikum 6 37](#_bookmark14)

[Gambar 6.5. Contoh isian folder project anda 38](#_bookmark15)

[Gambar 6.6. Setting OpenGL pada DevC++ 38](#_bookmark16)

[Gambar 6.7. Hasil algoritma BFS pada graf. 42](#_bookmark17)

[Gambar 6.8. Hasil algoritma DFS pada graf. 44](#_bookmark18)

[Gambar 7.1. Algoritma Bactracking 46](#_bookmark20)

[Gambar 7.2 Algoritma Runut-balik Untuk Pewarnaan Graf-1 46](#_bookmark21)

[Gambar 7.3. Algoritma Runut-balik Untuk Pewarnaan Graf-2 47](#_bookmark22)

[Gambar 7.4 Contoh kasus pewarnaan graf 47](#_bookmark23)

[Gambar 7.5 Graf untuk soal Pre Test dan Post Test Praktikum 7 48](#_bookmark24)

[Gambar 7.6. Contoh isian folder project 48](#_bookmark25)

[Gambar 7.7. Setting OpenGL pada DevC++ 49](#_bookmark26)

[Gambar 7.8 Hasil penerapan algoritma Bactracking untuk pewarnaan simpul graf 52](#_bookmark27)

[Gambar 8.1 Contoh kasus TSP 55](#_bookmark29)

[Gambar 8.2. Solusi algoritma B&B pada kasus TSP 56](#_bookmark30)

[Gambar 8.3. Kasus TSP untuk Pre Test dan Post Test Praktikum 8 56](#_bookmark31)

[Gambar 8.4. Contoh isian folder project 57](#_bookmark32)

[Gambar 8.5. Setting OpenGL pada DevC++ 57](#_bookmark33)

[Gambar 8.5. Hasil penerapan algoritma B&B pada kasus TSP 62](#_bookmark34)

[Gambar 9.1 Contoh kasus pencarian rute terdekat dari simpul A ke simpul F. 64](#_bookmark36)

[Gambar 9.2. Graf untuk Pre Test dan Post Test Praktikum 9 65](#_bookmark37)

[Gambar 9.3. Contoh isian folder project 66](#_bookmark38)

[Gambar 9.4. Setting OpenGL pada DevC++ 66](#_bookmark39)

[Gambar 9.5. Hasil penerapan algoritma A\* untuk mencari rute terdekat 71](#_bookmark40)

[Gambar 10.1 Ilustrasi algoritma KMP 72](#_bookmark42)

[Gambar 10.2. Ilustrasi fungsi pinggiran KMP 73](#_bookmark43)

[Gambar 10.3. Contoh penerapan algoritma KMP 73](#_bookmark44)

[Gambar 10.4. Contoh isian folder project 74](#_bookmark45)

[Gambar 10.5. Setting OpenGL pada DevC++ 74](#_bookmark46)

[Gambar 10.6. Hasil penerapan algoritma KMP 77](#_bookmark47)

# PRAKTIKUM 1: KOMPLEKSITAS ALGORITMA

Pertemuan ke 1

Total Alokasi Waktu : 90 menit

* Pre-Test : 15 menit
* Praktikum : 60 menit
* Post-Test : 15 menit

Total Skor Penilaian : 100%

* Pre-Test : 20 %
* Praktikum : 30 %
* Post-Test : 50 %

## TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan:

1. Mampu mengetahui contoh praktis penerapan kasus dengan algoritma yang sesuai.
2. Mampu menerapkan perhitungan kompleksitas pada suatu algoritma.

Indikator ketercapaian diukur dengan keberhasilan mahasiswa dalam menerapkan perhitungan kompleksitas dari suatu algoritma.

## TEORI PENDUKUNG

Apakah Strategi Algoritma itu?

Strategi algoritma (algorithm strategies) adalah:

* pendekatan umum
* untuk memecahkan persoalan secara algoritmis
* yang dapat diterapkan pada bermacam-macam persoalan
* dari berbagai bidang komputasi [Levitin, 2003] Nama lain: algorithm design technique

Untuk persoalan dengan instansiasi yang besar, solusinya menjadi lebih sulit ditentukan. Perlu sebuah prosedur umum yang berisi langkahlangkah penyelesaian persoalan  **algoritma.** Analisis algoritma dilakukan untuk mengukur performa algoritma dari segi efisiensinya. Sehingga diketahui algoritma yang tepat untuk solusi suatu persoalan.

Algoritma yang bagus adalah algoritma yang efisien. Efisiensi suatu algoritma diukur dari berapa jumlah waktu dan ruang (*space*) memori yang dibutuhkan untuk menjalankannya. Algoritma yang efisien adalah algoritma yang meminimumkan kebutuhan waktu dan ruang. Kebutuhan waktu dan ruang suatu algoritma bergantung pada ukuran masukan (n), yang menyatakan jumlah data yang diproses. Efisiensi algoritma dapat digunakan untuk menilai algoritma yang terabit.

Mengukur waktu yang diperlukan oleh sebuah algoritma dengan cara menghitung banyaknya operasi/instruksi yang dieksekusi. Jika yang diketahui besaran waktu (dalam satuan detik) untuk melaksanakan sebuah operasi tertentu, maka dapat menghitung berapa waktu sesungguhnya untuk melaksanakan algoritma tersebut.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *a*1 | *a*2 | *a*3 | … | *an* |

Array/Larik bilangan bulat



13

*Contoh 1.* ***Menghitung rata-rata***

1. Operasi pengisian nilai (jumlah=0, k=1, jumlah=jumlah+ak, k=k+1, dan r =jumlah/n). Jumlah seluruh operasi pengisian nilai adalah t1 = 1 + 1 + *n* + *n* + 1 = 3 + 2*n*
2. Operasi penjumlahan (jumlah+ak, dan k+1). Jumlah seluruh operasi penjumlahan adalah t2 = *n* + *n* = 2*n*
3. Operasi pembagian (jumlah/n). Jumlah seluruh operasi pembagian adalah *t*3 = 1

**Total kebutuhan waktu algoritma HitungRataRata:** *t* = *t*1 + *t*2 + *t*3 = (3 + 2*n*) *a* + 2*nb* + *c* detik Model perhitungan kebutuhan waktu seperti di atas kurang dapat diterima:

* 1. Dalam prakteknya contoh diatas tidak mempunyai informasi berapa waktu sesungguhnya untuk

melaksanakan suatu operasi tertentu

* 1. Komputer dengan arsitektur yang berbeda akan berbeda pula lama waktu untuk setiap jenis operasinya.

Selain bergantung pada komputer, kebutuhan waktu sebuah program juga ditentukan oleh *compiler* bahasa yang digunakan.

Model abstrak pengukuran waktu/ruang harus independen dari pertimbangan mesin dan *compiler* Besaran yang dipakai untuk menerangkan model abstrak pengukuran waktu/ruang ini adalah **kompleksitas algoritma**.

Ada dua macam kompleksitas algoritma, yaitu **kompleksitas waktu** dan **kompleksitas ruang**.

* **Kompleksitas waktu, *T*(*n*)**, diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan *n*.
* **Kompleksitas ruang, *S*(*n*)**, diukur dari memori yang digunakan oleh struktur data yang terdapat di dalam algoritma sebagai fungsi dari ukuran masukan *n*.

Dengan menggunakan besaran kompleksitas waktu/ruang algoritma, maka dapat menentukan *laju* peningkatan waktu (ruang) yang diperlukan algoritma dengan meningkatnya ukuran masukan *n*.

**Kompleksitas Waktu**

Dalam praktek, kompleksitas waktu dihitung berdasarkan jumlah operasi abstrak yang *mendasari* suatu algoritma, dan memisahkan analisisnya dari implementasi.

*Contoh 2.* Tinjau algoritma menghitung rata-rata pada *Contoh 1*.

Operasi yang mendasar pada algoritma tersebut adalah operasi penjumlahan elemen-elemen *ak* (yaitu jumlah=jumlah+ak),

Kompleksitas waktu HitungRataRata adalah *T*(*n*) = *n.*

*Contoh 3***.** Algoritma untuk mencari elemen terbesar di dalam sebuah larik (*array*) yang berukuran *n* elemen.

procedure CariElemenTerbesar(input a1, a2, …, an : integer, output maks : integer)

*{ Mencari elemen terbesar dari sekumpulan elemen larik integer a1, a2, …, an. Elemen terbesar akan disimpan di dalam maks.*

*Masukan: a1, a2, …, an*

*Keluaran: maks (nilai terbesar)*

*}*

**Deklarasi** k : integer

**Algoritma** maks=a1 k=2

while k <= n do if ak > maks then maks=ak

endif i-i+1

endwhile

*{ k > n }*

Kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi perbandingan elemen larik (A[i] > maks).

Kompleksitas waktu CariElemenTerbesar : *T*(*n*) = *n* – 1. Kompleksitas waktu dibedakan atas tiga macam :

1. *Tmax(n)* : kompleksitas waktu untuk kasus terburuk *(worst case)*  kebutuhan waktu maksimum.
2. *Tmin(n)* : kompleksitas waktu untuk kasus terbaik *(best case) –>* kebutuhan waktu minimum.
3. *Tavg(n)*: kompleksitas waktu untuk kasus rata-rata *(average case) –>* kebutuhan waktu secara rata-rata

## ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. Dev C++.

## PRE TEST

1. Apa alat ukur sebuah algoritma dikatakan effisien menjadi solusi suatu permasalahan?
2. Mengapa kita perlu algoritma yang efisien?

## LANGKAH PRAKTIKUM

Algoritma pengurutan pilih (Selection Sort):

procedure Urut(input/output a1, a2, …, an : integer)

**Deklarasi**

i, j, imaks, temp: integer **Algoritma**

for i = n downto 2 do *{ pass sebanyak n – 1 kali }*

imaks = 1

for j = 2 to i do

if aj > aimaks then imaks = j

endif endfor

Lakukan analisis algoritma diatas, tentukan:



15

*{ pertukarkan aimaks dengan ai }*

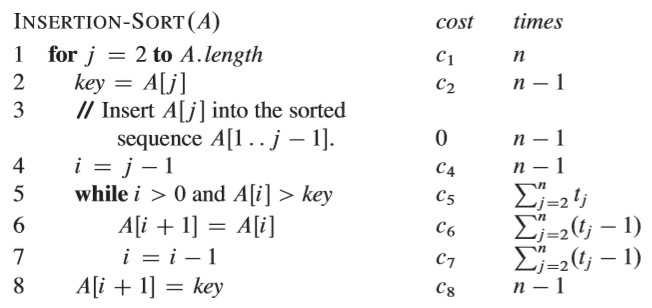
temp = ai ai = aimaks

aimaks = temp endfor

1. Jumlah operasi perbandingan elemen. Untuk setiap *pass* ke-*i?*
2. Jumlah operasi pertukaran elemen?
3. Kompleksitas waktu algoritma diatas?

## TUGAS / POST TEST

Algoritma INSERTION SORT:



1. Tentukan kompleksitas waktunya!
2. Bandingkan kompleksitas waktu algoritma SELECTION SORT dengan INSERTION SORT, menurut kalian mana yang lebih mangkus dan apa alasannya?

# PRAKTIKUM 2: ALGORITMA BRUTE FORCE

Pertemuan ke 2

Total Alokasi Waktu : 90 menit

* Pre-Test : 15 menit
* Praktikum : 60 menit
* Post-Test : 15 menit

Total Skor Penilaian : 100%

* Pre-Test : 20 %
* Praktikum : 30 %
* Post-Test : 50 %

## TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan:

1. Mahasiswa mampu mengaplikasikan notasi algoritma (*pseudocode*) ke coding program.
2. Mahasiswa dapat membuat program dari persoalan dengan strategi algoritma brute force untuk menyelesaikan persoalan pencarian bilangan terbesar dan uji bilangan prima.

Indikator ketercapaian diukur dengan keberhasilan mahasiswa dalam menerapkan algoritma brute force untuk menyelesaikan persoalan pencarian bilangan terbesar dan uji bilangan prima.

## TEORI PENDUKUNG

Brute force adalah algortima pendekatan yang lempang (*straightforward*) untuk memecahkan suatu persoalan dengan sangat sedarhana, langsung dan jelas (*obvious way*). Contoh-contoh persoalan yang dipecahkan dengan *Brute Force*:

1. Perpangkatan

Menghitung *an* (*a>0, n* adalah bilangan bulat tak-negatif)

*an = a x a x … x a* (sebanyak n kali), jika n>0

*=* 1 , jika n = 0

Algoritma: kalikan 1 dengan *a* sebanyak n kali

Algoritma Pangkat:

**Function** pangkat(a : real, n : integer)  real { Menghitung a^n } **Deklarasi:**

i : integer hasil : real

**Algoritma:**

hasil  1

for i  1 to n do hasil  hasil \* a

end

return hasil

17

1. Pencarian elemen terbesar

#### Persoalan:

Diberikan sebuah senarai yang beranggotakan n buah bilangan bulat (*a1,a2,…,an*). Carilah elemen terbesar didalam senarai tersebut.

Algoritma: Bandingkan setiap elemen senarai untuk menemukan elemen terbesar

**procedure** CariElemenTerbesar(input a1, a2, ..., an : integer, output maks : integer)

{ Mencari elemen terbesar di antara elemen a1, a2, ..., an. Elemen terbesar akan disimpan di dalam maks. Masukan: a1, a2, ..., an Keluaran: maks

}

**Deklarasi**

k : integer

**Algoritma**: maksa1

for k2 to n do

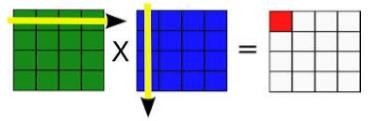
if ak > maks then maksak

endif endfor

1. Perkalian dua buah matriks, A dan B

Misalkan, *C = A × B dan elemen-elemen matrik* dinyatakan sebagai *c*ij*, a*ij*, dan b*ij

Cij = ai1b1j + ai2b2j …. + ain bnj



Gambar 2.1 Ilustrasi perkalian dua matriks

Algoritma: hitung setiap elemen hasil perkalian satu per satu, dengan cara mengalikan dua vector yang panjangnya *n*.

**Procedure** PerkalianMatriks(input A, B : Matriks, input n : integer,output C : Matriks)

*{ Mengalikan matriks A dan B yang berukuran n × n, menghasilkan matriks C yang juga berukuran n × n*

*Masukan: matriks integer A dan B, ukuran matriks n Keluaran: matriks C }*

**Deklarasi**

i, j, k : integer

**Algoritma**

for i  1 to n do for j ← 1 to n do

C[i,j]  0 *{ inisialisasi penjumlah }*

for k  1 to n do

C[i,j]  C[i,j] + A[i,k]\*B[k,j] endfor

endfor endfor

1. Uji Bilangan Prima

**Persoalan** : Diberikan sebuah bilangan-bilangan bulat positif. Ujilah apakah bilangan tersebut merupakan bilangan prima atau bukan.

**Definisi** : Bilangan Prima adalah bilangan yang hanya habis dibagi oleh 1 dan dirinya sendiri.

**Algoritma** : Bagi 𝑛 dengan 2 sampai √𝑛. Jika semuanyan tidak habis dibagi 𝑛, maka 𝑛 adalah bilangan prima.

**function** Prima(input x : integer)-> boolean

*{ Menguji apakah x bilangan prima atau bukan. Masukan: x*

*Keluaran: true jika x prima, atau false jika x tidak prima.}*

**Deklarasi**

k, y : integer test : boolean

**Algoritma:**

if x < 2 then *{ 1 bukan prima }*

return false else

if x = 2 then *{ 2 adalah prima, kasus khusus }*

return true else

y  √x test  true

while (test) and (y ³ 2) do if x mod y = 0 then

test  false else

y  y - 1 endif

endwhile

*{ not test or y < 2 }*

return test endif

endif

**Contoh Program dalam C++:**

#include <iostream.h> #include <conio.h> #include <ctype.h> #include <math.h>

// return 1 if n is prime, 0 otherwise int isPrime(int p);

main()

{

for (int n=1; n< 50; n++)

if (isPrime(n)) cout<<n<<" "; getch(); cout<<endl;

}

int isPrime(int p)

{

float sqrtp=sqrt(p);

if (p<2) return 0; // 2 is the first prime if (p==2) return 1;

if (p%2==0) return 0; //2 is only the evenprime for (int d=3; d<=sqrtp; d+=2)

if (p%d==0) return 0; return 1;

}

## ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. Software Dev C++.

## PRE TEST

1. Jelaskan apa ciri khas dari penyelesaian persoalan dengan algoritma *brute force*?
2. Jelaskan ciri khas algoritma brute force pada contoh diatas:
   1. Algoritma Perpangkatan
   2. Algoritma Pencarian Elemen Terbesar
   3. Algoritma Perkalian dua Matriks
   4. Algorima Uji Bilangan Prima

## LANGKAH PRAKTIKUM

1. Buatlah coding/pseudocode algoritma-algoritma berikut, kemudian lakukan Run program
   1. Algoritma Pencarian Bilangan Terbesar
   2. Algorima Uji Bilangan Prima
2. Lakukanlah analisis hasil output coding

## TUGAS / POST TEST

1. Kompilasi dan eksekusi contoh program diatas
2. Implementasikan dengan C++ algoritma pencarian bilangan terbesar diatas
3. Hitunglah kompleksitas dari algoritma uji bilangan prima.

# PRAKTIKUM 3: ALGORITMA GREEDY

Pertemuan ke 3

Total Alokasi Waktu : 90 menit (Alokasi waktu disesuaikan dengan RPS)

* Pre-Test : 15 menit
* Praktikum : 60 menit
* Post-Test : 15 menit

Total Skor Penilaian : 100%

* Pre-Test : 20 %
* Praktikum : 30 %
* Post-Test : 50 %

## TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan:

1. Mahasiswa mampu menggunakan pendekatan algoritma *greed*y untuk menyelesaikan masalah yang tepat.
2. Mahasiswa dapat membuat program dari persoalan dengan strategi algoritma Greedy untuk menyelesaikan persoalan Penukaran Uang.

Indikator ketercapaian diukur dengan keberhasilan mahasiswa dalam menerapkan algoritma Greedy untuk menyelesaikan persoalan Penukaran Uang.

## TEORI PENDUKUNG

Algoritma *greedy* merupakan metode yang paling popular untuk memecahkan persoalan optimasi (*optimization problems*). Hanya ada dua macam persoalan optimasi yaitu maksimasi (*maximization*) dan minimasi (*minimization*). Greedy = rakus, tamak, loba, …

Prinsip greedy yaitu “*take what you can get now!*”. Algoritma greedy membentuk solusi langkah per langkah (*step by step*). Pada setiap langkah, terdapat banyak pilihan yang perlu dievaluasi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan.

#### Masalah Penukaran Uang

Nilai uang yang ditukar: A Himpunan koin (multiset): {d1, d2, …}.

Himpunan solusi: X = {x1, x2, …, xn}, xi = 1 jika di dipilih, xi = 0 jika di tidak dipilih.

Obyektif persoalan adalah

*n*



Minimisasi *F* = *x*

*i*1 *i*

(fungsi obyektif)

dengan kendala *d*  *A*

*n*



*x*

*i* 1 *i i*

Strategi algoritma greedy: Pada setiap langkah, pilih koin dengan nilai terbesar dari himpunan koin yang tersisa.

function CoinExchange(input C : himpunan\_koin, A : integer)  himpunan\_koin

*{ mengembalikan koin-koin yang total nilainya = A, tetapi jumlah koinnya minimum }*

**Deklarasi**

S : himpunan\_koin x : koin

**Algoritma** S  {}

while ((nilai semua koin di dalam S)  A) and (C  {} ) do x  koin yang mempunyai nilai terbesar

C  C - {x}

if ((nilai semua koin di dalam S) + nilai koin x  A then S  S  {x}

endif endwhile

if ((nilai semua koin di dalam S) = A then return S

else

write(’tidak ada solusi’) endif

## ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. Software Dev C++.

## PRE TEST

Jelaskan apa yang anda ketahui tentang algoritma greedy?

## LANGKAH PRAKTIKUM

1. Buatlah coding/pseudocode algoritma penukaran koin diatas, kemudian lakukan Run program
2. Lakukan pengujian dengan sebuah kasus

## TUGAS / POST TEST

1. Implementasikan dengan C++ algoritma diatas, amati dan analisis output yang dihasilkan serta lakukan tester dalam suatu kasus.
2. Hitung kompleksitas waktunya!

# PRAKTIKUM 4: ALGORITMA DIVIDE & CONQUER

Pertemuan ke 4

Total Alokasi Waktu : 90 menit

* Pre-Test : 15 menit
* Praktikum : 45 menit
* Post-Test : 30 menit

Total Skor Penilaian : 100%

* Pre-Test : 20 %
* Praktikum : 30 %
* Post-Test : 50 %

## TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan:

1. Mampu menggunakan pendekatan skema Divide and Conquer untuk menyelesaikan masaah yang tepat
2. Mampu mengaplikasikan notasi algoritma (pseudocode) ke coding

Indikator ketercapaian diukur dengan keberhasilan mahasiswa dalam menerapkan algoritma Divide and Conquer untuk menyelesaikan masaah yang tepat.

## TEORI PENDUKUNG

**Definisi *Divide and Conquer***

***Divide:*** membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama),

***Conquer:*** memecahkan (menyelesaikan) masing-masing upa-masalah (secara rekursif), dan

***Combine:*** mengabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula.

Obyek permasalahan yang dibagi adalah masukan (input) atau instances yang berukuran n: tabel (larik), matriks, eksponen, dan sebagainya, bergantung pada masalahnya. Tiap-tiap upa-masalah mempunyai karakteristik yang sama (the same type) dengan karakteristik masalah asal, sehingga metode Divide and Conquer lebih natural diungkapkan dalam skema rekursif. Skema Umum Algoritma *Divide and Conquer*, sebagai berikut:

**procedure** DIVIDE\_and\_CONQUER(input n : integer)

*{ Menyelesaikan masalah dengan algoritma D-and-C. Masukan: masukan yang berukuran n*

*Keluaran: solusi dari masalah semula*

*}*

**Deklarasi**

r, k : integer

**Algoritma**

if n  n0 then *{ukuran masalah sudah cukup kecil }*

SOLVE upa-masalah yang berukuran n ini

else

Bagi menjadi r upa-masalah, masing-masing berukuran n/k for masing-masing dari r upa-masalah do

DIVIDE\_and\_CONQUER(n/k) endfor

COMBINE solusi dari r upa-masalah menjadi solusi masalah semula *}*

endif

Permasalahan yang dapat diselesaikan dengan algoritma *divide and conquer*:

#### Mencari nilai Minimum dan Maksimum (MinMaks) Persoalan:

Misalkan diberikan tabel *A* yang berukuran *n* elemen dan sudah berisi nilai *integer*. Carilah nilai minimum dan nilai maksimum sekaligus di dalam tabel tersebut. Penyelesaian dengan Divide and Conquer seperti berikut:

Misalkan tabel *A* berisi elemen-elemen sebagai berikut:

4 12 23 9 21 1 35 2 24

Ide dasar algoritma secara *Divide and Conquer*:

4 12 23 9 21 1 35 2 24

*DIVIDE*

4 12 23 9

21 1 35 2 24

*SOLVE*: tentukan min & maks pada tiap bagian

4 12 23 9 21 1 35 2 24

min = 4 min = 1

maks = 23 maks = 35

*COMBINE*

4 12 23 9 21 1 35 2 24

min = 1

maks = 35

#### Algoritma:

procedure MinMaks2(input A : TabelInt, i, j : integer,

output min, maks : integer)

*{ Mencari nilai maksimum dan minimum di dalam tabel A yang berukuran n elemen secara Divide and Conquer.*

*Masukan: tabel A yang sudah terdefinisi elemen-elemennya Keluaran: nilai maksimum dan nilai minimum tabel*

*}*

**Deklarasi**

min1, min2, maks1, maks2 : integer

**Algoritma:**

if i=j then

minAi maksAi

else

if (i = j-1) then if Ai < Aj then

maksAj

*{ 1 elemen*

*}*

*{ 2 elemen*

*}*

minAi else

maksAi minAj

endif else

k(i+j) div 2

*{ lebih dari 2 elemen }*

*{ bagidua tabel pada posisi k }*

MinMaks2(A, i, k, min1, maks1)

MinMaks2(A, k+1, j, min2, maks2) if min1 < min2 then

minmin1 else

minmin2 endif

if maks1<maks2 then maksmaks2

else

maksmaks2 endif

endif endif

1. **Algoritma Pengurutan dengan *divide and conquer***

Skema Algoritma pengurutan dengan *divide and conquer* secara umum adalah sebagai berikut:

**procedure** Sort(input/output A : TabelInt, input n : integer)

{ Mengurutkan tabel A dengan metode Divide and Conquer Masukan: Tabel A dengan n elemen Keluaran: Tabel A yang terurut

}

**Algoritma:**

if Ukuran(A) > 1 then

Bagi A menjadi dua bagian, A1 dan A2, masing-masing berukuran n1 dan n2 (n = n1 + n2)

Sort(A1, n1) { urut bagian kiri yang berukuran n1 elemen } Sort(A2, n2) { urut bagian kanan yang berukuran n2 elemen }

Combine(A1, A2, A) { gabung hasil pengurutan bagian kiri dan bagian kanan } end

* 1. **Permasalahan *Merge Sort***

Algoritma Merge Sort:

* + 1. Untuk kasus *n* = 1, maka tabel *A* sudah terurut dengan sendirinya (langkah SOLVE).
    2. Untuk kasus *n* > 1, maka
       1. DIVIDE: bagi tabel *A* menjadi dua bagian, bagian kiri dan bagian kanan, masing-masing bagian berukuran *n*/2 elemen.
       2. CONQUER: Secara rekursif, terapkan algoritma *D-and-C* pada masing-masing bagian.
       3. MERGE: gabung hasil pengurutan kedua bagian sehingga diperoleh tabel *A* yang terurut.

Algoritma Merge Sort dengan Divide and Conquer:

**procedure** MergeSort(input/output A : TabelInt, input i, j : integer)

{ Mengurutkan tabel A[i..j] dengan algoritma Merge Sort

Masukan: Tabel A dengan n elemen Keluaran: Tabel A yang terurut

}

**Deklarasi**:

k : integer

**Algoritma**:

if i < j then { Ukuran(A)> 1} k(i+j) div 2

MergeSort(A, i, k) MergeSort(A, k+1, j) Merge(A, i, k, j)

endif

*Pseduocode* Prosedure Merge

**procedure** Merge(input/output A : TabelInt, input kiri,tengah,kanan : integer) { Menggabung tabel

A[kiri..tengah] dan tabel A[tengah+1..kanan] menjadi tabel A[kiri..kanan] yang terurut menaik.

Masukan: A[kiri..tengah] dan tabel A[tengah+1..kanan] yang sudah terurut menaik.

Keluaran: A[kiri..kanan] yang terurut menaik.

}

**Deklarasi**

B : TabelInt

i, kidal1, kidal2 : integer

**Algoritma**:

kidal1kiri

{ A[kiri .. tengah] }

kidal2tengah + 1 { A[tengah+1 .. kanan] } ikiri

while (kidal1 ≤ tengah) and (kidal2 ≤ kanan) do if Akidal1 ≤ Akidal2 then

BiAkidal1

kidal1kidal1 + 1 else

BiAkidal2 kidal2kidal2 + 1

endif ii + 1

endwhile

{ kidal1 > tengah or kidal2 > kanan }

{ salin sisa A bagian kiri ke B, jika ada } while (kidal1 ≤ tengah) do

BiAkidal1 kidal1kidal1 + 1

ii + 1 endwhile

{ kidal1 > tengah }

{ salin sisa A bagian kanan ke B, jika ada } while (kidal2 ≤ kanan) do

BiAkidal2 kidal2kidal2 + 1

ii + 1 endwhile

{ kidal2 > kanan }

{ salin kembali elemen-elemen tabel B ke A } for ikiri to kanan do

AiBi endfor

{ diperoleh tabel A yang terurut membesar }

* 1. **Permasalahan *Quick Sort***

Ditemukan oleh Tony Hoare tahun 1959 dan dipublikasikan tahun 1962. Termasuk pada pendekatan sulit membagi, mudah menggabung (*hard split/easy join*). Tabel A dibagi (istilahnya: dipartisi) menjadi A1 dan A2 sedemikian sehingga elemenelemen A1  elemen- elemen A2.

Teknik mem-partisi tabel:

1. pilih x  { A[1], A[2], ..., A[n] } sebagai pivot,
2. pindai tabel dari kiri sampai ditemukan A[p]  x
3. pindai tabel dari kanan sampai ditemukan A[q]  x
4. pertukarkan A[p]  A[q] (v) ulangi (ii), dari posisi p + 1, dan (iii), dari posisi q – 1 , sampai kedua pemindaian bertemu di tengah tabel (p  q)

Algoritma Merge Sort dengan *Divide and Conquer*:

procedure QuickSort(input/output A : TabelInt, input i,j: integer)

{ Mengurutkan tabel A[i..j] dengan algoritma Quick Sort. Masukan: Tabel A[i..j] yang sudah terdefinisi elemen-elemennya. Keluaran: Tabel A[i..j] yang terurut menaik.

}

Deklarasi k : integer

Algoritma:

if i < j then

{ Ukuran(A) > 1 }

Partisi(A, i, j, k) { Dipartisi pada indeks k }

QuickSort(A, i, k) { Urut A[i..k] dengan Quick Sort } QuickSort(A, k+1, j) { Urut A[k+1..j] dengan Quick Sort }

endif

*Pseudocode* Prosedur Patisi Tabel

procedure Partisi(input/output A : TabelInt, input i, j : integer, output q : integer)

{ Membagi tabel A[i..j] menjadi upatabel A[i..q] dan A[q+1..j] Masukan: Tabel A[i..j]yang sudah terdefinisi harganya.

Keluaran upatabel A[i..q] dan upatabel A[q+1..j] sedemikian sehingga elemen tabel A[i..q] lebih kecil dari elemen tabel A[q+1..j] }

**Deklarasi**

pivot, temp : integer

**Algoritma:**

pivotA[(i + j) div 2] { pivot = elemen tengah} p  i

q  j repeat

while A[p] < pivot do p  p + 1

endwhile

{ A[p] >= pivot}

while A[q] > pivot do q  q – 1

endwhile

{ A[q] <= pivot}

if p < q then

{pertukarkan A[p] dengan A[q] } swap(A[p], A[q])

{tentukan awal pemindaian berikutnya } p  p + 1

q  q - 1 endif

until p > q

## PRE TEST

1. Buatlah fuction quicksort!
2. Buatlah fuction partition!

## ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. Software Dev C++.

## LANGKAH PRAKTIKUM

1. Ketik program algoritma shorting dibawah ini, kemudian lakukan Run program! Program Algoritma Sorting:

#include <cstdlib.h> #include <iostream.h>

using namespace std;

typedef int larik [10]; //tipe data untuk merge void baca\_data(int A[], int n){

//proses input dan baca data

for (int i = 0; i < n; i++) {

cout <<"Data ke - " << i + 1 <<" : "; cin >> A[i];

}

}

void cetak\_data(int A[], int n){

//cetak data

for (int i = 0; i < n; i++) { cout << A[i] <<" ";

}

}

void tukar\_data(int \*a, int \*b) {

//tukar data int temp; temp = \*a;

\*a = \*b;

\*b = temp;

}

void minimum(int A[], int dari, int n, int \*tempat) { int i, min;

min = A[dari];

\*tempat = dari;

for (i = dari + 1; i < n; i++) if (A[i] < min) {

min = A[i];

\*tempat = i;

}

}

void bubble\_sort(int x[], int n) {

// bubble sort

for (int i = 0; i < n - 1; i++) { for (int j = i + 1; j < n; j++) {

if (x[i] > x [j]) tukar\_data (&x[i], &x[j]);

}

}

}

void selection\_sort(int A[], int n) {

// sellection sort int i, t;

for (i = 0; i < n; i++) { minimum (A, i, n, &t);

tukar\_data (&A[i], &A[t]);

}

}

void merge(larik a, int kiri, int tengah, int kanan) { int bagkir, postemp, bykel, i;

larik temp;

bagkir = tengah -1; postemp = kiri;

bykel = kanan - kiri + 1;

while ((kiri <= bagkir) && (tengah <= kanan)) { if ((a[kiri] <= a[tengah])) {

temp[postemp] = a[kiri]; postemp = postemp + 1; kiri = kiri + 1;

}

else {

temp[postemp] = a[tengah]; postemp = postemp + 1; tengah = tengah + 1;

}

//kopi bagian kiri

while ((kiri <= bagkir)) { temp[postemp] = a[kiri]; postemp = postemp + 1; kiri = kiri + 1;

}

//kopi bagian kanan

while ((tengah <= kanan)) { temp[postemp] = a[tengah]; postemp = postemp + 1; tengah = tengah + 1;

}

//kopi ke aaray asal

for (i = 1; i <= bykel; i++) { a[kanan] = temp[kanan]; kanan = kanan -1;

}

}

}

void merge\_sort(larik A, int kiri, int kanan) { int tengah;

if (kiri < kanan) {

tengah = (kiri + kanan) / 2; merge\_sort(A, kiri, tengah); merge\_sort(A, tengah + 1, kanan); merge(A, kiri, tengah + 1, kanan);

}

}

int main(int argc, char \*argv[])

{

int data[10], n; int pilih;

t1:

cout << "1. Bubble Sort\n2. Selection Sort\n3. Merge Sort\n\n"; cout << "Pilihan : ";

cin >> pilih;

switch (pilih) {

case 1: //bubble sort

cout <<"BUBBLE SORT";

cout <<"\n\nMasukan data : "; cin >> n;

baca\_data(data, n);

cout <<"Data yang anda masukan : "; cetak\_data(data, n);

cout<<endl; bubble\_sort(data, n);

cout <<"Setelah ditukar (Bubble Sort) : "; cetak\_data(data, n);

cout <<endl; break;

case 2: //selection sort

cout <<"SELECTION SORT";

cout <<"\n\nMasukan data : "; cin >> n;

baca\_data(data, n);

cout <<"Data yang anda masukan : "; cetak\_data(data, n);

cout<<endl; selection\_sort(data, n);

cout <<"Setelah ditukar (Selection Sort): "; cetak\_data(data, n);

cout <<endl; break;

case 3: //merge sort

cout <<"MERGE SORT";

cout <<"\n\nMasukan data : "; cin >> n;

baca\_data(data, n);

cout <<"Data yang anda masukan : "; cetak\_data(data, n);

cout<<endl; merge\_sort(data, 0, n - 1);

cout <<"Setelah ditukar (Merge Sort): "; cetak\_data(data, n);

cout <<endl; break;

default: cout <<"Ulangi !\n"; goto t1;

}

system("PAUSE"); return EXIT\_SUCCESS;

}

1. Kemudian lakukan Run program!

## TUGAS / POST TEST

1. Pada program algoritma shorting diatas ada 3 algoritma. Buatlah program tersendiri untuk algoritma merge sort dengan solusi divide and conquer!
2. Lakukan Eksekusi sehingga menghasilkan output!

31

# PRAKTIKUM 5: ALGORITMA DECREASE & CONQUER

Pertemuan ke 5

Total Alokasi Waktu : 90 menit

* Pre-Test : 15 menit
* Praktikum : 45 menit
* Post-Test : 30 menit

Total Skor Penilaian : 100%

* Pre-Test : 20 %
* Praktikum : 30 %
* Post-Test : 50 %

## TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan:

1. Mampu menggunakan pendekatan skema Decrease and Conquer untuk menyelesaikan masalah yang tepat
2. Mampu mengaplikasikan notasi algoritma (pseudocode) ke coding

Indikator ketercapaian diukur dengan keberhasilan mahasiswa dalam menerapkan algoritma Decrease and Conquer untuk menyelesaikan masaah yang tepat.

## TEORI PENDUKUNG

Decrease and conquer: metode desain algoritma dengan mereduksi persoalan menjadi beberapa subpersoalan yang lebih kecil, tetapi selanjutnya hanya memproses satu sub-persoalan saja. Berbeda dengan divide and conquer yang memproses semua sub-persoalan dan menggabung semua solusi setiap sub-persoalan. Decrease and conquer terdiri dari dua tahapan:

1. **Decrease**: mereduksi persoalan menjadi beberapa persoalan yang lebih kecil (biasanya dua subpersoalan).
2. **Conquer**: memproses satu sub-persoalan secara rekursif.

Tidak ada tahap combine dalam decrease and conquer. Tiga varian decrease and conquer:

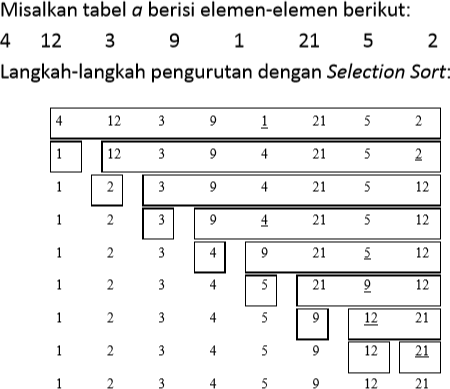
1. Decrease by a constant: ukuran instans persoalan direduksi sebesar konstanta yang sama setiap iterasi algoritma. Biasanya konstanta = 1.
2. Decrease by a constant factor: ukuran instans persoalan direduksi sebesar faktor konstanta yang sama setiap iterasi algoritma. Biasanya faktor konstanta = 2.
3. Decrease by a variable size: ukuran instans persoalan direduksi bervariasi pada setiap iterasi algoritma.

Persoalan dengan solusi Decrease and Conquer:

#### 1. Pengurutan dengan Selection Sort



32



Gambar 5.1. Contoh ilustrasi pengurutan decrease and conquer dengan selection sort Algoritma selection sort:

**procedure** SelectionSort(input/output A : TabelInt, input i,j: integer)

{ Mengurutkan tabel A[i..j] dengan algoritma Selection Sort. Masukan: Tabel A[i..j] yang sudah terdefinisi elemen-elemennya. Keluaran: Tabel A[i..j] yang terurut menaik. }

**Algoritma**:

if i < j then { Ukuran(A) > 1 } Bagi(A, i, j)

SelectionSort(A, i+1, j) endif

Algoritma Table Partition Selection Sort:

**procedure** Bagi(input/output A : TabInt, input i,j: integer)

{ Mencari elemen terkecil di dalam tabel A[i..j], dan menempatkan elemen terkecil sebagai elemen pertama tabel.

Masukan: A[i..j]

Keluaran: A[i..j] dengan Ai adalah elemen terkecil.

}

**Deklarasi**

idxmin, k, temp : integer

**Algoritma**:

Idxmini

for ki+1 to j do if Ak < Aidxmin then

idxmink endif

endfor

{ pertukarkan Ai dengan Aidxmin } tempAi

AiAidxmin Aidxmintemp

## PRE TEST



33

1. Apakah karakteristik dari algoritma decrease and conquer?
2. Jelaskan perbedaan algoritma decrease and conquer dengan divide and conquer!
3. Sebutkan algoritma selain selection sort yang menggunakan teknik decrease and conquer!

## ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. Software Dev C++.

## LANGKAH PRAKTIKUM

1. Ketik program selection sort dibawah ini, kemudian lakukan Run program!

// C++ program for implementation of selection sort #include <bits/stdc++.h>

using namespace std;

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

// One by one move boundary of unsorted subarray for (i = 0; i < n-1; i++)

{

// Find the minimum element in unsorted array min\_idx = i;

for (j = i+1; j < n; j++) if (arr[j] < arr[min\_idx])

min\_idx = j;

// Swap the found minimum element with the first element swap(&arr[min\_idx], &arr[i]);

}

}

/\* Function to print an array \*/ void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

cout << arr[i] << " "; cout << endl;

}

// Driver program to test above functions int main()

{

int arr[] = {64, 25, 12, 22, 11};

1. Kemudian lakukan eksekusi program!



34

int n = sizeof(arr)/sizeof(arr[0]); selectionSort(arr, n);

cout << "Sorted array: \n"; printArray(arr, n);

return 0;

}

1. Analisis hasil output!

## TUGAS / POST TEST

Buatlah program selection sort dengan user sebagai penginput bilangan array!

# PRAKTIKUM 6: ALGORITMA BFS DAN DFS



35

Pertemuan ke 6

Total Alokasi Waktu : 90 menit

* Pre-Test : 15 menit
* Praktikum : 45 menit
* Post-Test : 30 menit

Total Skor Penilaian : 100%

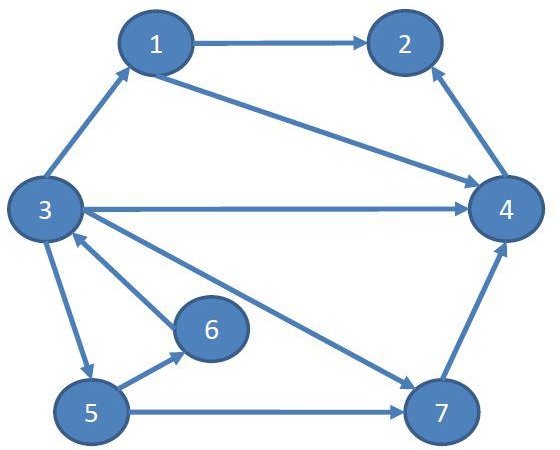
* Pre-Test : 20 %
* Praktikum : 30 %
* Post-Test : 50 %

## TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan mampu menerapkan algoritma BFS dan DFS untuk melakukan transversal pada graf.

Indikator ketercapaian diukur dengan keberhasilan mahasiswa dalam menerapkan algoritma BFS dan DFS untuk melakukan transversal pada graf.

## TEORI PENDUKUNG

Algoritma traversal graf digunakan untuk mengunjungi simpul dengan cara yang sistematik dengan asumsi graf terhubung. Algoritma transversal graf diantaranya pencarian melebar (breadth first search/BFS) dan pencarian mendalam (depth first search/DFS). Algoritma ini termasuk dalam algoritma pencarian tanpa informasi (uninformed/blind search). Dalam proses pencarian solusi, terdapat dua pendekatan yaitu graf statis dan graf dinamis. Graf statis merupakan graf yang sudah terbentuk sebelum proses pencarian dilakukan. Graf ini direpresentasikan sebagai struktur data. Graf dinamis merupkan graf yang terbentuk saat proses pencarian dilakukan. Graf ini tidak tersedia sebelum pencarian, graf dibangun selama pencarian solusi.

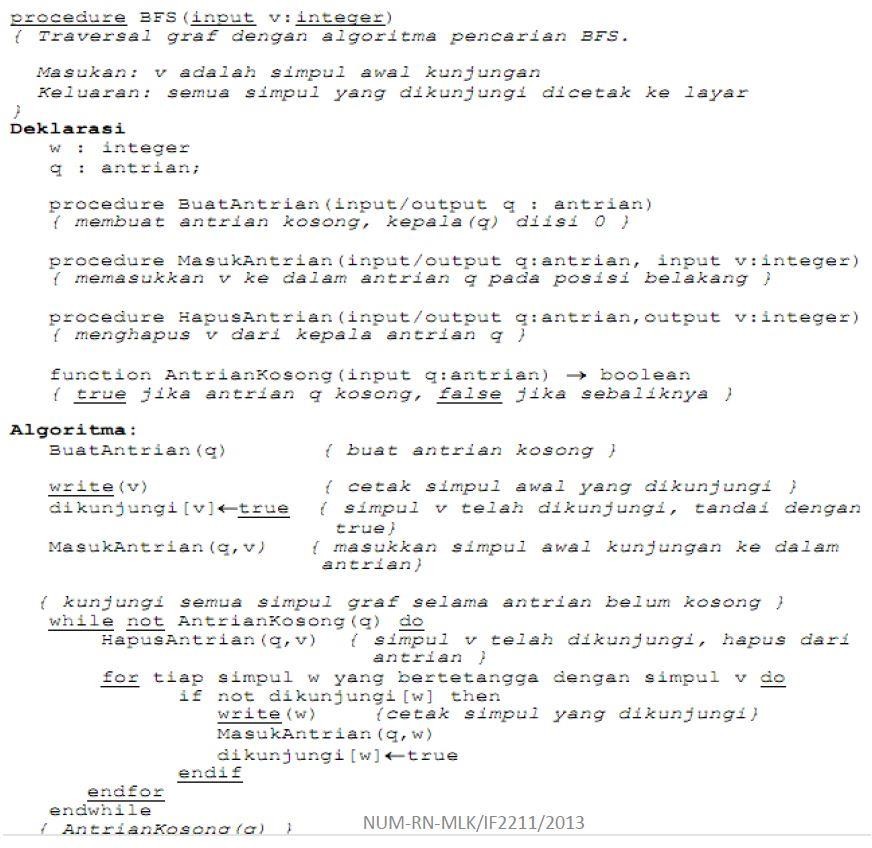
|  |  |
| --- | --- |
| **Breadth First Search (BFS)**  Pada BFS traversal dimulai dari simpul v. Algoritma:   1. Kunjungi simpul v 2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu. 3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.   **Depth First Search (DFS)**  Pada DFS traversal dimulai dari simpul v. Algoritma:  1. Kunjungi simpul v | Gambar 6.1. Contoh graf berarah |



36

|  |  |
| --- | --- |
| 1. Kunjungi simpul w yang bertetangga dengan simpul v. 2. Ulangi DFS mulai dari simpul w. 3. Ketika mencapai simpul usedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi. 4. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai   dari simpul yang telah dikunjungi. | Pada Gambar 6.1 diatas,  Hasil algoritma BFS dimulai dari node-1 yaitu: 1-2-4-3-5-7-6  Hasil algoritma DFS dimulai dari node-1 yaitu: 1-2-4-3-5-6-7 |

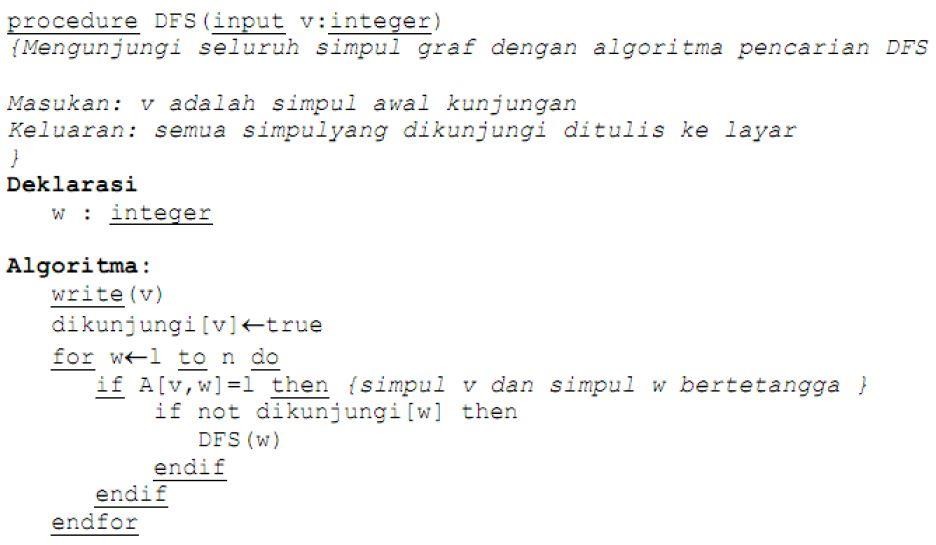
#### Algoritma BFS



Gambar 6.2. Algoritma BFS

(**Sumber**: Slide Mata Kuliah Strategi Algoritmik IF2211– ITB, Rinaldi Munir, 2019)

Gambar 6.3 Algoritma DFS



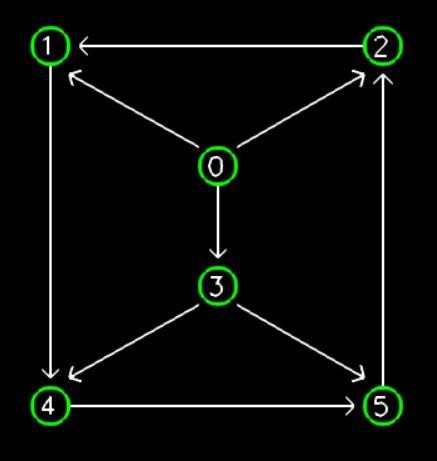
37

**Algoritma DFS**

(**Sumber**: Slide Mata Kuliah Strategi Algoritmik IF2211– ITB, Rinaldi Munir, 2019)

## PRE TEST

1. Analisis graf berarah pada Gambar 6.4 di bawah ini dan tuliskan urutan jalurnya secara manual apabila menggunakan algoritma BFS dan DFS mulai dari simpul/node ke-0!
2. Apa perbedaan utama antara algoritma BFS dan DFS?



Gambar 6.4 Graf berarah untuk soal Pre Test dan Post Test Praktikum 6

## ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. Dev C++.
3. OpenGL Library.

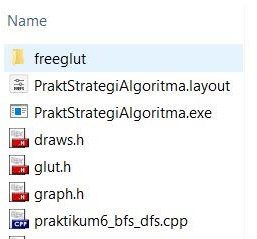
## LANGKAH PRAKTIKUM



38

### PERSIAPAN

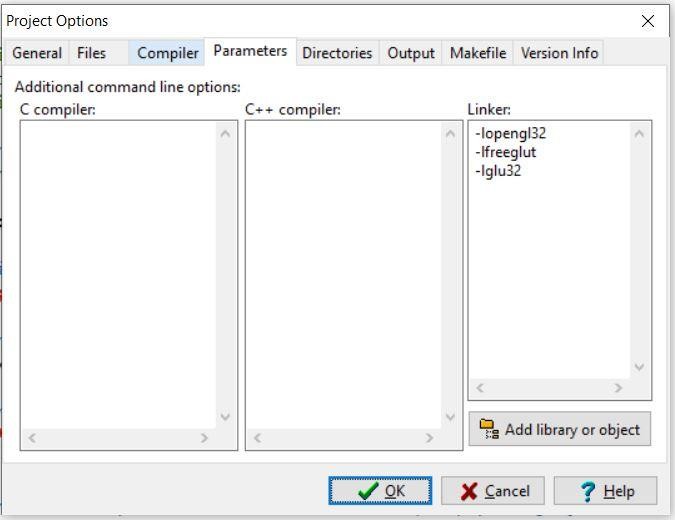
1. Buka DevC++ dan buat project baru dengan nama **praktikum06.**
2. Download library **freeglut** di elearning Strategi Algoritma di bagian atas. Ekstrak .rar nya di lokasi folder project anda berada. Contoh tampilan folder project anda ditunjukkan pada Gambar 6.5.



Gambar 6.5. Contoh isian folder project anda

1. Setting OpenGL library pada DevC++ dengan klik Project – Project options – Parameters lalu isikan

–lopengl32 –lfreeglut –lglu32 pada Linker seperti pada Gambar 6.6.



Gambar 6.6. Setting OpenGL pada DevC++

1. Download file tambahan praktikum yaitu **draws.h** dan **graph.h** untuk menggambar graf dengan OpenGL dari elearning Strategi Algoritma. Ekstrak .rar nya lalu copy kan ke folder project anda.
2. Buat file baru dengan nama **praktikum06.cpp**.

### PRAKTIKUM

1. Tuliskan kode berikut pada **praktikum06.cpp** untuk inisialisasi variable dan jendela OpenGL.

// Praktikum 6: BFS dan DFS

// Penerapan contoh di slide kuliah Strategi Algoritma - BFS dan DFS hal 14



39

// header

#include <windows.h> #include <iostream> #include <list>

// header untuk menggambar graf dengan OpenGL #include "draws.h"

using namespace std;

// deklarasi global Graph Graph graph;

// buffer untuk simpan teks char markText[10];

// hasil dari penerapan algoritma vector<int>\* pathResult; vector<int> pathSequence;

// fungsi untuk menggambar jalur hasil pencarian dengan OpenGL void **drawResult**()

{

glPushMatrix();

// gambar garis dengan anak panah float radius = 15.0f;

for (int i=0; i<graph.getNumNodes(); i++)

{

for (int j=0; j<pathResult[i].size(); j++)

{

int nodeIdx = pathResult[i].at(j); Vec3 color(1.0f,0.0f,1.0f);

drawLine(graph.getNodePosition(), i, nodeIdx, color, radius, 2.0f, graph.getIsDirected());

}

}

// gambar urutan hasil pencarian

for (int i=0; i<pathSequence.size(); i++)

{

int nodeIdx = pathSequence.at(i); if (i == 0)

sprintf(markText,"%s","start");

else

sprintf(markText,"%d",i);

Vec3 position(

graph.getNodePosition()[nodeIdx].getX()+2.0f\*radius, graph.getNodePosition()[nodeIdx].getY()+2.0f\*radius, 0.0f);

Vec3 color(1.0f,0.0f,1.0f);

drawText(position, color, markText, radius, 2.0f);

}

glPopMatrix();

}

// taruh semua obyek yang akan digambar di fungsi display() void **displayGraph**()

{

// bersihkan dan reset layar dan buffer glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

**Note**: Ingat untuk C++ fungsi yang akan digunakan/dipanggil harus berada di atas fungsi yang memanggil kecuali dideklarasikan nama fungsinya terlebih dahulu di bagian atas program setelah header

1. Tuliskan kode fungsi berikut untuk menerapkan algoritma BFS pada graf.



40

glLoadIdentity();

// posisikan kamera pandang

gluLookAt(0.0f, 0.0f, 2.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);

// panggil fungsi untuk menggambar obyek drawNodes();

if (!pathSequence.empty()) drawResult();

drawEdges();

// tampilkan obyek ke layar glutSwapBuffers();

}

// fungsi untuk menerapkan Breadth First Search (BFS) pada graf

// graph = graf

// startIdx = indeks node mulai void **BFS**(Graph graph, int startIdx)

{

// tandai semua node yang belum dikunjungi bool\* visited = new bool[graph.getNumNodes()]; for(int i = 0; i < graph.getNumNodes(); i++)

visited[i] = false;

// buat antrian node list<int> queue;

// tandai node sekarang sebagai node yang dikunjungi visited[startIdx] = true; queue.push\_back(startIdx);

pathSequence.clear();

pathResult = new vector<int>[graph.getNumNodes()]; while(!queue.empty())

{

// keluarkan node dari antrian dan cetak indeksnya startIdx = queue.front();

cout << startIdx << " "; pathSequence.push\_back(startIdx); queue.pop\_front();

// ambil semua node yang bertetangga dengan node sekarang

// dari daftar antrian node

for (int i=0; i<graph.getAdjNodes()[startIdx].size(); i++)

{

// jika node tetangganya belum dikunjungi maka tandai

// telah dikunjungi dan keluarkan dari antrian

int nodeIdx = graph.getAdjNodes()[startIdx].at(i); if (!visited[nodeIdx])

{

visited[nodeIdx] = true; queue.push\_back(nodeIdx); pathResult[startIdx].push\_back(nodeIdx);

}

}

1. Tuliskan kode tester berikut untuk menerapkan algoritma BFS pada graf di Gambar 6.1.



41

// jika antrian kosong tapi masih ada node yang belum dikunjungi

// maka buat node tersebut sebagai titik awal lagi if (queue.empty())

{

int j=0;

// cari node yang belum dikunjungi

while (visited[j] && j<graph.getNumNodes()) j++;

// bila ada node yang belum dikunjungi maka masukan di antrian if (!visited[j] && j<graph.getNumNodes())

{

visited[j] = true; queue.push\_back(j);

}

}

}

}

// kode tester

int **main**(int argc, char\*\* argv)

{

// inisialisasi graf graph.setIsDirected(true); graph.setNumLevels(4); graph.setNumNodes(7);

// tambahkan nodes graph.addNode(0, 0, 1.1f);

graph.addNode(1, 0, 0.95f);

graph.addNode(2, 1, 0.9f);

graph.addNode(3, 1, 1.05f);

graph.addNode(4, 3, 1.1f);

graph.addNode(5, 2, 0.85f);

graph.addNode(6, 3, 0.95f);

// tambahkan edges graph.addEdge(0, 1);

graph.addEdge(0, 3);

graph.addEdge(2, 0);

graph.addEdge(2, 3);

graph.addEdge(2, 4);

graph.addEdge(2, 6);

graph.addEdge(3, 1);

graph.addEdge(4, 5);

graph.addEdge(4, 6);

graph.addEdge(5, 2);

graph.addEdge(6, 3);

// perkiraan posisi node graph.setNodePosition();

**// terapkan BFS int startIdx = 0;**

**cout << "BFS mulai dari node " << startIdx << "\n"; BFS(graph, startIdx);**

// inisialisasi jendela OpenGL glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH);

// set ukuran jendela tampilan dalam piksel glutInitWindowSize(SCREEN\_WIDTH, SCREEN\_HEIGHT);

// posisi jendela dilayar komputer dalam piksel

1. Jalankan kodenya unutuk menerapkan algoritma BFS pada graf diatas. Hasil penerapan ditunjukkan pada Gambar 6.7.



42

glutInitWindowPosition(100, 100);

// judul jendela (isi dengan NAMA / NIM - JUDUL PRAKTIKUM) glutCreateWindow("NAMA / NIM - PRAKTIKUM STIMA");

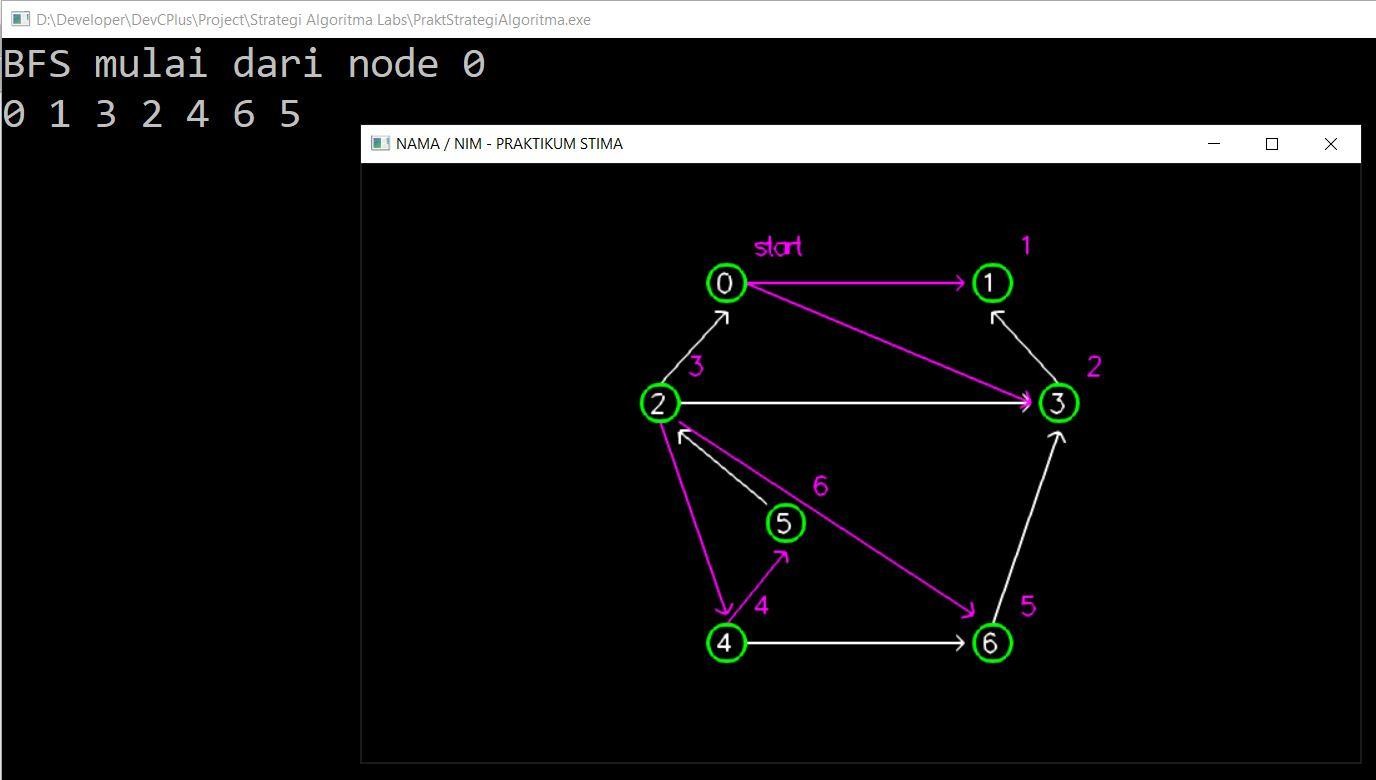
// panggil fungsi init untuk inisialisasi awal initView();

// event handler untuk display glutDisplayFunc(displayGraph); glutReshapeFunc(reshapeView);

// looping glutMainLoop();

return 0;

}



Gambar 6.7. Hasil algoritma BFS pada graf.

**Note:** Kalau waktunya masih cukup silakan dilanjutkan dengan langkah berikut. Apabila tidak maka langkah selanjutnya dijadikan tugas.

1. Tuliskan kode fungsi berikut untuk menerapkan algoritma DFS pada graf.

// fungsi rekursi untuk Depth First Search (DFS)

void **DFSRecursive**(Graph graph, int startIdx, int endIdx, bool visited[])

{

// tandai node sekarang sebagai node yang dikunjungi visited[startIdx] = true;

cout << startIdx << " "; pathSequence.push\_back(startIdx);

// rekursi ke semua node yang bertetangga

for (int i=0; i<graph.getAdjNodes()[startIdx].size(); i++)

{

int nodeIdx = graph.getAdjNodes()[startIdx].at(i); if (!visited[nodeIdx])

{

DFSRecursive(graph, nodeIdx, startIdx, visited); pathResult[startIdx].push\_back(nodeIdx);

1. Ganti kode penerapan algoritma BFS berikut yang ada di kode tester main() menjadi penerapan algoritma DFS.



43

}

}

// jika semua node cabang pada node awal sudah habis maka

// tentukan node baru yang belum dikunjungi sebagai node awal if (startIdx == endIdx)

{

int j=0;

// cari node yang belum dikunjungi

while (visited[j] && j<graph.getNumNodes()) j++;

// bila ada node yang belum dikunjungi maka masukan dalam antrian if (!visited[j] && j<graph.getNumNodes())

DFSRecursive(graph, j, j, visited);

}

}

// fungsi untuk menerapkan Depth First Search (DFS) pada graf

// graph = graf

// startIdx = indeks node mulai void **DFS**(Graph graph, int startIdx)

{

// tandai semua node yang belum dikunjungi bool\* visited = new bool[graph.getNumNodes()]; for(int i = 0; i < graph.getNumNodes(); i++)

visited[i] = false;

pathSequence.clear();

pathResult = new vector<int>[graph.getNumNodes()];

// rekursi DFS

DFSRecursive(graph, startIdx, startIdx, visited);

}

. . .

// terapkan BFS int startIdx = 0;

cout << "BFS mulai dari node " << startIdx << "\n";

**BFS(graph, startIdx);**

. . .

Menjadi

. . .

// terapkan DFS int startIdx = 0;

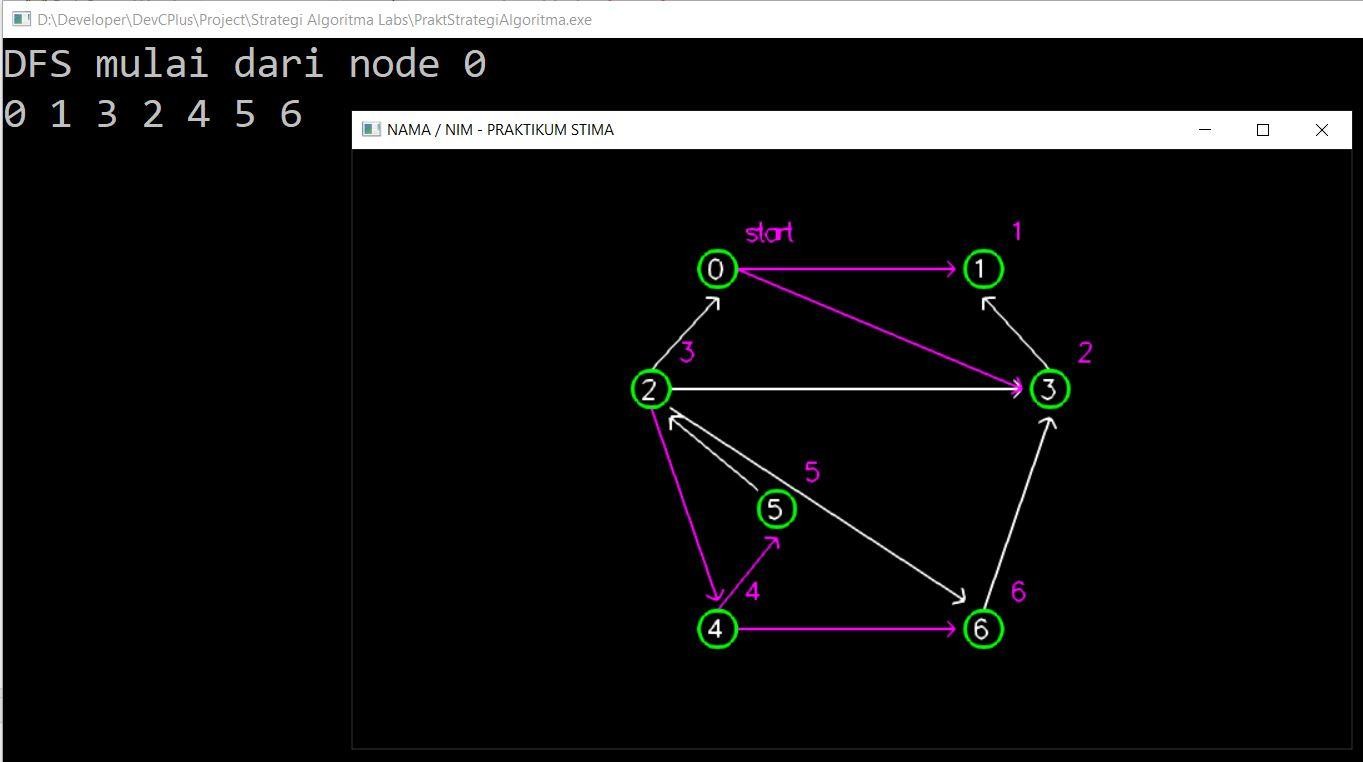
cout << "DFS mulai dari node " << startIdx << "\n";

**DFS(graph, startIdx);**

. . .

1. Jalankan kodenya unutuk menerapkan algoritma DFS pada graf diatas. Hasil penerapan ditunjukkan pada Gambar 6.8.

Gambar 6.8. Hasil algoritma DFS pada graf.



44

1. Amati perbedaan kedua algoritma tersebut dan jelaskan perbedaannya.

## TUGAS / POST TEST

1. Terapkan algoritma BFS dan DFS pada graf berarah di Gambar 6.4 (lihat bagian Pre Test) mulai dari simpul ke-0 dengan memodifikasi kode tester praktikum BFS dan DFS yang sudah anda buat!
2. Analisis apakah hasil dari program sama dengan jawaban pre test anda!

# PRAKTIKUM 7: ALGORITMA BACKTRACKING



45

Pertemuan ke 7

Total Alokasi Waktu : 90 menit

* Pre-Test : 15 menit
* Praktikum : 45 menit
* Post-Test : 30 menit

Total Skor Penilaian : 100%

* Pre-Test : 20 %
* Praktikum : 30 %
* Post-Test : 50 %

## TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan mampu menerapkan algoritma Backtracking untuk menyelesaikan masalah pewarnaan node/simpul pada graf.

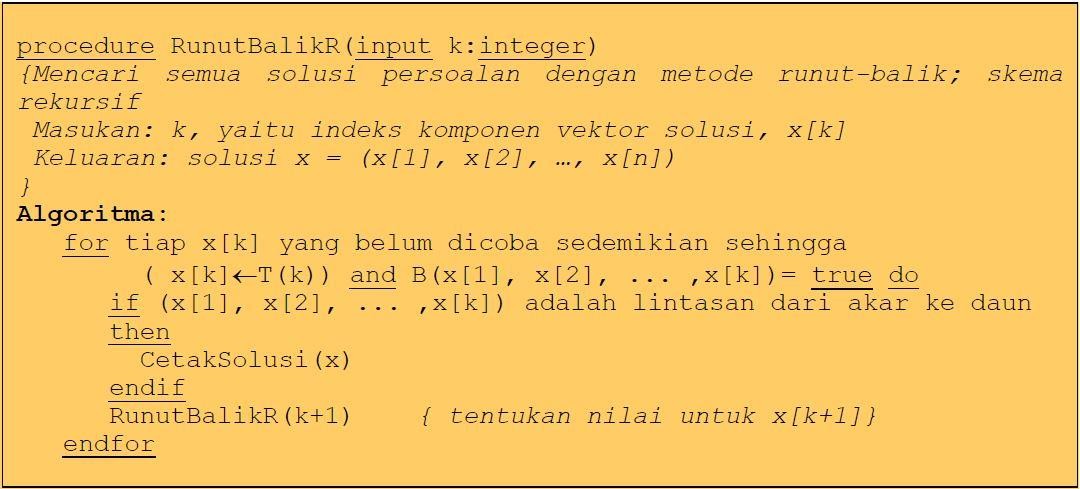
Indikator ketercapaian diukur dengan keberhasilan mahasiswa dalam menerapkan algoritma Backtracking untuk menyelesaikan masalah pewarnaan node/simpul pada graf.

## TEORI PENDUKUNG

Algoritma runut-balik merupakan perbaikan dari exhaustive search. Pada exhaustive search, semua kemungkinan solusi dieksplorasi satu per satu. Pada backtracking, hanya pilihan yang mengarah kesolusi yang dieksplorasi, pilihan yang tidak mengarah ke solusi tidak dipertimbangkan lagi. Dengan kata lain memangkas (pruning) simpul-simpul yang tidak mengarah ke solusi. Algoritma runut-balik pertama kali diperkenalkan oleh D. H. Lehmer pada tahun 1950. Kemudian, R.J Walker, Golomb, dan Baumert menyajikan uraian umum tentang algoritma runut-balik. Pada graf atau pohon, Backtracking dapat dipandang sebagai pencarian di dalam pohon menuju simpul daun (goal) tertentu. Ada tiga macam simpul (node) yaitu Simpul akar, Simpul dalam, dan Simpul daun. Prinsip Pencarian Solusi dengan Metode Runut-balik:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti aturan depht-first order (DFS).
2. Simpul-simpul yang sudah dilahirkan dinamakan simpul hidup (live node).
3. Simpul hidup yang sedangdiperluas dinamakan simpul-E (Expand-node).
4. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang.
5. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut “dibunuh” sehingga menjadi simpul mati (dead node).
6. Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan fungsi pembatas (bounding function).
7. Simpul yang sudah mati tidak akan pernah diperluas lagi.
8. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian backtrackke simpul aras diatasnya
9. Lalu, teruskan dengan membangkitkan simpul anak yang lainnya.
10. Selanjutnya simpul ini menjadi simpul-E yang baru.

46



1. Pencarian dihentikan bila kita telah sampai pada goal node.

#### Algoritma Backtracking:

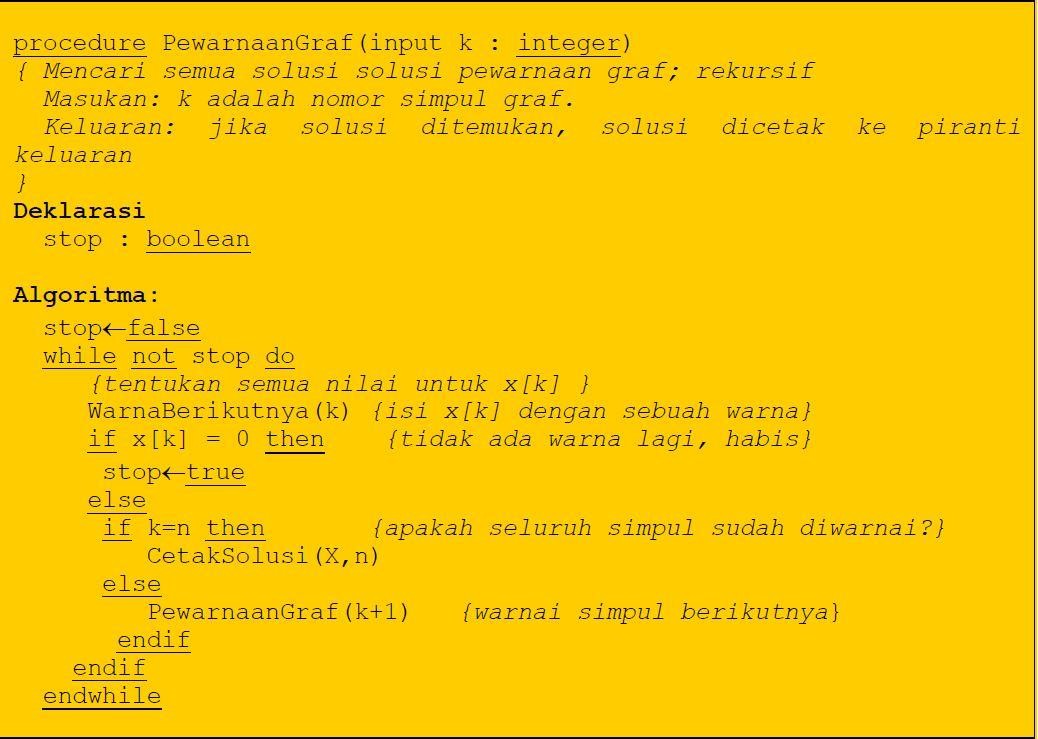
Gambar 7.1. Algoritma Bactracking

(**Sumber**: Slide Mata Kuliah Strategi Algoritmik IF2211– ITB, Rinaldi Munir, 2018)

Setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemanggilan rekursif. Jika jumlah simpul dalam pohon ruang status adalah 2n atau n! maka untuk kasus terburuk, algoritma runut-balik membutuhkan waktu dalam O(p(n)2n) atau O(q(n)n!), denganp(n) dan q(n) adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul.

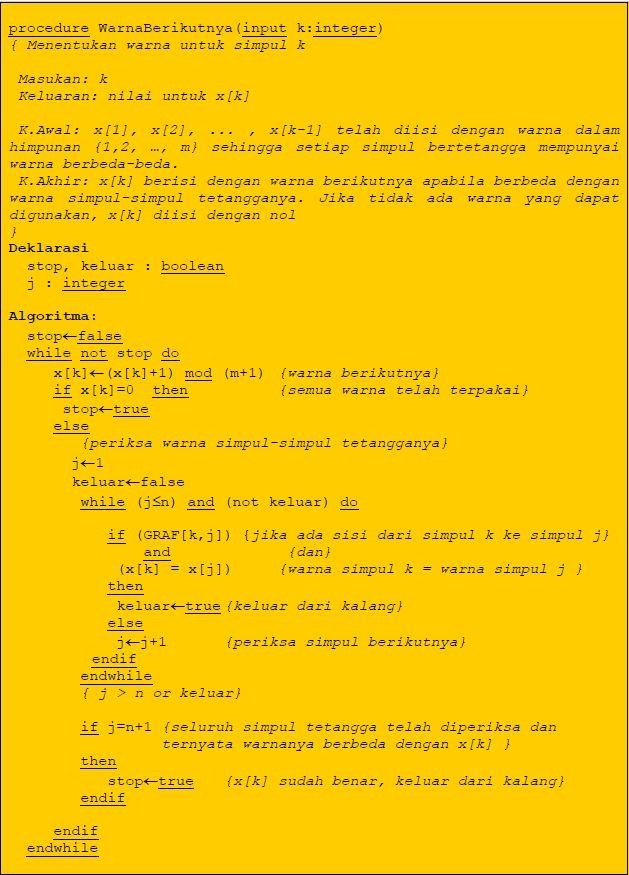
#### Pewarnaan Graf (Graph Colouring)

Diberikan sebuah graf G dengan n buah simpul dan disediakan m buah warna. Bagaimana mewarnai seluruh simpul graf G sedemikian sehingga tidak ada dua buah simpul bertetangga yang mempunyai warna sama (perhatikan juga bahwa tidak seluruh warna harus dipakai).



Gambar 7.2 Algoritma Runut-balik Untuk Pewarnaan Graf-1

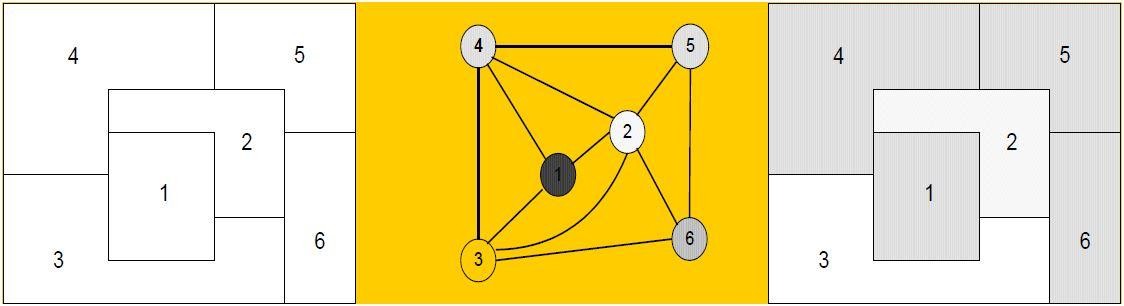
47



Gambar 7.3. Algoritma Runut-balik Untuk Pewarnaan Graf-2

(**Sumber**: Slide Mata Kuliah Strategi Algoritmik IF2211– ITB, Rinaldi Munir, 2018)

#### Contoh:



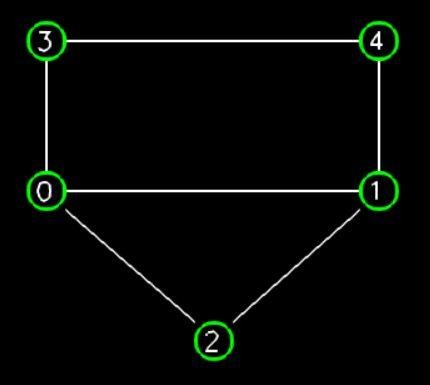
Gambar 7.4 Contoh kasus pewarnaan graf

(**Sumber**: Slide Mata Kuliah Strategi Algoritmik IF2211– ITB, Rinaldi Munir, 2018)

48

## PRE TEST

Analisis graf pada Gambar 7.5 di bawah ini dan tuliskan satu cara urutan pewarnaan simpul yang mungkin secara manual dengan algoritma Backtracking apabila menggunakan 3 warna: merah, hijau dan biru!



Gambar 7.5 Graf untuk soal Pre Test dan Post Test Praktikum 7

## ALAT DAN BAHAN

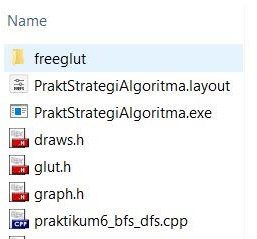
Alat dan bahan yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. DevC++.
3. OpenGL Library.

## LANGKAH PRAKTIKUM

### PERSIAPAN

1. Buka DevC++ dan buat project baru dengan nama **praktikum07.**
2. Download library **freeglut** di elearning Strategi Algoritma di bagian atas. Ekstrak .rar nya di lokasi folder project anda berada. Contoh tampilan folder project ditunjukkan pada Gambar 7.6.

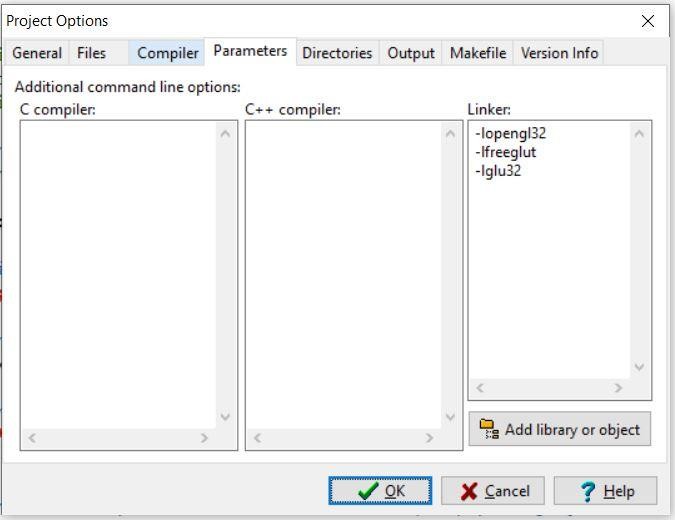


Gambar 7.6. Contoh isian folder project

1. Setting OpenGL library pada DevC++ dengan klik Project – Project options – Parameters lalu isikan

–lopengl32 –lfreeglut –lglu32 pada Linker seperti pada Gambar 7.7.

Gambar 7.7. Setting OpenGL pada DevC++



49

1. Download file tambahan praktikum yaitu **draws.h** dan **graph.h** untuk menggambar graf dengan OpenGL dari elearning Strategi Algoritma. Ekstrak .rar nya lalu copy kan ke folder project anda.
2. Buat file baru dengan nama **praktikum07.cpp**.

### PRAKTIKUM

1. Tuliskan kode berikut pada praktikum07.cpp untuk inisialisasi variable dan jendela OpenGL.

// Praktikum 7: Backtracking

// Penerapan contoh di slide kuliah Strategi Algoritma - Backtracking hal 39

#include <windows.h> #include <iostream> #include <list>

#include "draws.h" using namespace std;

// global Graph Graph graph;

// hasil penerapan algoritma bool colorExist;

int\* colorList;

// fungsi untuk menandai hasil void **drawResult**()

{

glPushMatrix();

float radius = 15.0f; if (colorExist)

{

// gambar nodes

for (int i=0; i<graph.getNumNodes(); i++)

{

1. Tuliskan kode fungsi berikut untuk menerapkan algoritma Backtracking untuk pewarnaan simpul pada graf.



50

sprintf(text,"%d",i);

drawCircle(graph.getNodePosition()[i], getColorTable(colorList[i]-1), radius, 360, 3.0f);

drawText(graph.getNodePosition()[i], Vec3(1.0f,1.0f,1.0f), text, radius, 2.0f);

}

}

glPopMatrix();

}

// taruh semua obyek yang akan digambar di fungsi display() void **displayGraph**()

{

// bersihkan dan reset layar dan buffer glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

// posisikan kamera pandang

gluLookAt(0.0f, 0.0f, 2.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);

// panggil fungsi untuk menggambar obyek if (colorExist)

drawResult(); else

drawNodes(); drawEdges();

// tampilkan obyek ke layar glutSwapBuffers();

}

// cek apakah warna sudah dipakai atau belum di node tetangga

// kalau belum return true dan sebaliknya

bool **assignColor**(int nodeIdx, vector<bool> adjStatus[], int colorList[], int colorIdx)

{

for (int i=0; i<adjStatus[nodeIdx].size(); i++)

{

vector<bool> localAdjStatus = adjStatus[nodeIdx];

if (localAdjStatus.at(i) && colorIdx == colorList[i]) return false;

}

return true;

}

// rekursive pewarnaan graf dengan jumlah warna = numColors;

bool **graphColoringRecursive**(vector<bool> adjStatus[], int numNodes, int numColors, int colorList[], int nodeIdx)

{

// base case: jika semua node sudah diwarnai return true if (nodeIdx == numNodes)

return true;

// coba warna yang lain

for (int colorIdx = 1; colorIdx <= numColors; colorIdx++)

{

// cek apakah warna bisa diterapkan pada node

if (assignColor(nodeIdx, adjStatus, colorList, colorIdx))

{

colorList[nodeIdx] = colorIdx;

1. Tuliskan kode tester berikut untuk menerapkan algoritma Backtracking pada kasus di Gambar 7.4.



51

// rekursi untuk mewarnai semua node

if (graphColoringRecursive(adjStatus, numNodes, numColors, colorList, nodeIdx+1))

return true;

// jika warna tidak mungkin diterapkan maka dihapus colorList[nodeIdx] = 0;

}

}

// jika tidak ada warna yang bisa diterapkan maka return false return false;

}

// fungsi untuk mewarnai node pada graf dengan jumlah warna = numColors void **graphColoring**(vector<bool> adjStatus[], int numColors)

{

// inisialisasi awal semua warna di node = 0 colorList = new int[graph.getNumNodes()];

for (int i = 0; i < graph.getNumNodes(); i++) colorList[i] = 0;

// rekursi pewarnaan graf

if (graphColoringRecursive(adjStatus, graph.getNumNodes(), numColors, colorList, 0) == false)

{

printf("Tidak ada solusi pewarnaan yang mungkin"); colorExist = false;

}

else

{

printf("Solusi pewarnaannya yaitu: \n");

for (int i = 0; i < graph.getNumNodes(); i++) printf(" %d ", colorList[i]);

printf("\n"); colorExist = true;

}

}

// kode tester

int **main**(int argc, char\*\* argv)

{

// inisialisasi graf graph.setIsDirected(false); graph.setNumLevels(3); graph.setNumNodes(6);

// tambahkan node graph.addNode(0, 1, 1.2f);

graph.addNode(1, 1, 0.85f);

graph.addNode(2, 2, 1.0f);

graph.addNode(3, 0, 1.0f);

graph.addNode(4, 0, 1.0f);

graph.addNode(5, 2, 1.0f);

// tambahkan edge graph.addEdge(0, 1);

graph.addEdge(0, 2);

graph.addEdge(0, 3);

graph.addEdge(1, 2);

graph.addEdge(1, 3);

1. Jalankan kodenya untuk menerapkan algoritma Backtracking pada graf diatas. Hasil penerapan ditunjukkan pada Gambar 7.8. Keterangan: Warna 1: merah, Warna 2: hijau, Warna 3: biru, Warna 4: kuning, dst bisa dilihat di draws.h.



52

graph.addEdge(1, 4);

graph.addEdge(1, 5);

graph.addEdge(2, 3);

graph.addEdge(2, 5);

graph.addEdge(3, 4);

graph.addEdge(4, 5);

// estimate node position graph.setNodePosition(); graph.setAdjStatus();

**// banyaknya warna int numColors = 4;**

**graphColoring(graph.getAdjStatus(), numColors);**

// inisialisasi jendela OpenGL glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH);

// set ukuran jendela tampilan dalam piksel glutInitWindowSize(SCREEN\_WIDTH, SCREEN\_HEIGHT);

// posisi jendela dilayar komputer dalam piksel glutInitWindowPosition(100, 100);

// judul jendela (isi dengan NAMA / NIM - JUDUL PRAKTIKUM) glutCreateWindow("NAMA / NIM - PRAKTIKUM STIMA");

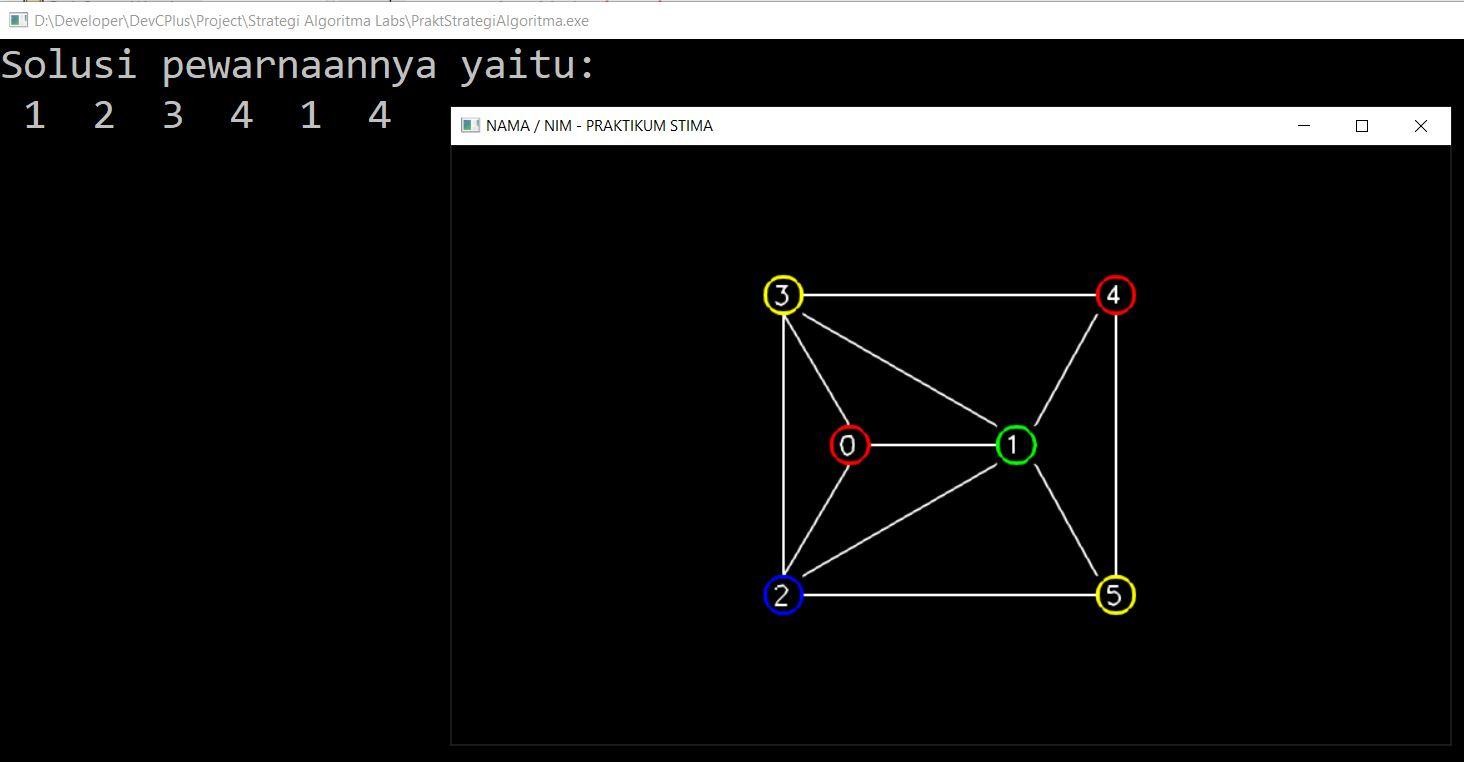
// panggil fungsi init untuk inisialisasi awal initView();

// event handler untuk display glutDisplayFunc(displayGraph); glutReshapeFunc(reshapeView);

// looping glutMainLoop();

return 0;

}



Gambar 7.8 Hasil penerapan algoritma Bactracking untuk pewarnaan simpul graf

1. Amati hasilnya kemudian jelaskan cara kerjanya.

## TUGAS / POST TEST



53

1. Terapkan algoritma backtracking pada graf di Gambar 7.5 (lihat bagian Pre Test) untuk pewarnaan simpul menggunakan 3 warna dengan memodifikasi kode tester praktikum backtracking yang sudah anda buat!
2. Analisis apakah hasil dari program sama dengan jawaban pre test anda!

54

# PRAKTIKUM 8: ALGORITMA BRANCH AND BOUND

Pertemuan ke 8

Total Alokasi Waktu : 90 menit

* Pre-Test : 15 menit
* Praktikum : 45 menit
* Post-Test : 30 menit

Total Skor Penilaian : 100%

* Pre-Test : 20 %
* Praktikum : 30 %
* Post-Test : 50 %

## TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan menerapkan algoritma branch and bound untuk Travelling Salesman Problem.

Indikator ketercapaian diukur dengan keberhasilan mahasiswa dalam menerapkan algoritma branch and bound untuk Travelling Salesman Problem.

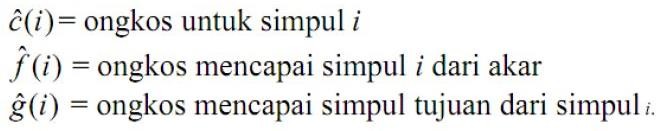
## TEORI PENDUKUNG

Algoritma Branch and Bound digunakan untuk persoalan optimisasi yang meminimalkan atau memaksimalkan suatu fungsi objektif, yang tidak melanggar batasan(constraints) persoalan. Algoritma B&B merupakan BFS + least cost search. Dimana di BFS murni, simpul berikutnya yang akan diekspansi berdasarkan urutan pembangkitannya (FIFO) sedangkan di B&B, setiap simpul diberi sebuah nilai cost: ĉ(i)= nilai taksiran lintasan termurah ke simpul status tujuan yang melalui simpul status i. Simpul berikutnya yang akan di-expand tidak lagi berdasarkan urutan pembangkitannya, tetapi simpul yang memiliki cost yang paling kecil (least cost search) –pada kasus minimasi. Algoritma B&B juga menerapkan “pemangkasan” pada jalur yang dianggap tidak lagi mengarah pada solusi.

Kriteria pemangkasan secara umum:

1. Nilai simpul tidak lebih baik dari nilai terbaik sejauh ini
2. Simpul tidak merepresentasikan solusi yang ‘feasible’ karena ada batasan yang dilanggar
3. Solusi yang feasible pada simpul tersebut hanya terdiri atas satu titik tidak ada pilihan lain; dengan membandingkan nilai fungsi obyektif dengan solusi terbaik saat ini, yang terbaik yang diambil.

Pada umumnya, untuk kebanyakan persoalan, letak simpul solusi tidak diketahui. Misalnya pada persoalan N-Ratu: persoalan yg ideal (letak simpul solusi diketahui). Untuk kasus lain apakah letak simpul solusi diketahui? Misalnya knapsack problem, graph colouring, permainan 8-puzzle, dan TSP. Pada umumnya, untuk kebanyakan persoalan, letak simpul solusi tidak diketahui. Cost setiap simpul umumnya berupa taksiran.



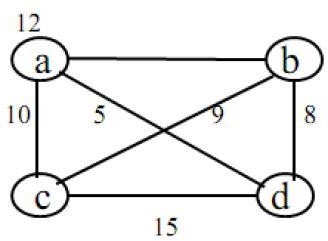
55

Algoritma umum untuk B&B sebagai berikut:

1. Masukkan simpul akar ke dalam antrian Q. Jika simpul akar adalah simpul solusi (goal node), maka solusi telah ditemukan. Stop.
2. Jika Q kosong, tidak ada solusi. Stop.
3. Jika Q tidak kosong, pilih dari antrian Q simpul i yang mempunyai nilai ‘cost’ ĉ(i)paling kecil. Jika terdapat beberapa simpul i yang memenuhi, pilih satu secara sembarang.
4. Jika simpul i adalah simpul solusi, berarti solusi sudah ditemukan, stop. Jika simpul i bukan simpul solusi, maka bangkitkan semua anak-anaknya. Jika i tidak mempunyai anak, kembali ke langkah 2.
5. Untuk setiap anak j dari simpul i, hitung ĉ(j), dan masukkan semua anak-anak tersebut ke dalam Q.
6. Kembali ke langkah 2.

#### Travelling Salesman Problem

Diberikan n buah kota serta diketahui jarak antara setiap kota satu sama lain. Temukan perjalanan (tour) terpendek yang melalui setiap kota lainnya hanya sekali dan kembali lagi ke kota asal keberangkatan. Contoh kasus ditunjukkan pada Gambar 8.1.



Gambar 8.1 Contoh kasus TSP

Kasus TSP dapat diselesaikan dengan B&B reduce cost matrix atau dengan bobot tour lengkap.

#### Branch and Bound dengan Reduce Cost Matrix pada kasus TSP

Sebuah matriks dikatakan tereduksi jika setiap kolom dan barisnya mengandung paling sedikit satu buah nol dan semua elemen lainnya non-negatif. Misalkan:

A: matriks tereduksi untuk simpul R.

S: anak dari simpul R sehingga sisi (R, S) pada pohon ruang status berkoresponden dengan sisi (i, j) pada perjalanan.

Jika S bukan simpul daun, maka matriks bobot tereduksi untuk simpul S dapat dihitung sebagai berikut:

1. ubah semua nilai pada baris i dan kolom j menjadi tak hingga (infinity). Ini untuk mencegah agar tidak ada lintasan yang keluar dari simpul i atau masuk pada simpul j;
2. ubah A(j, 1) menjadi tak hingga. Ini untuk mencegah penggunaan sisi (j, 1);
3. reduksi kembali semua baris dan kolom pada matriks A kecuali untuk elemen tak hingga. Jika r adalah total semua pengurang, maka nilai batas untuk simpul S adalah:

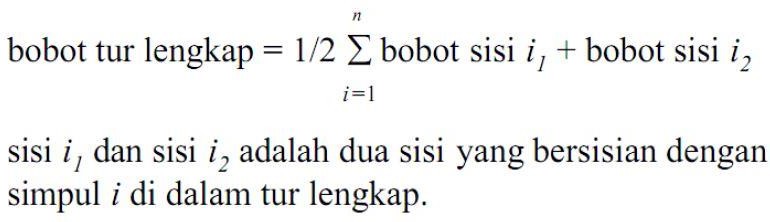
𝑐𝑜𝑠𝑡(𝑆) = 𝑐𝑜𝑠𝑡(𝑅) + 𝐴(𝑖, 𝑗) + 𝑟

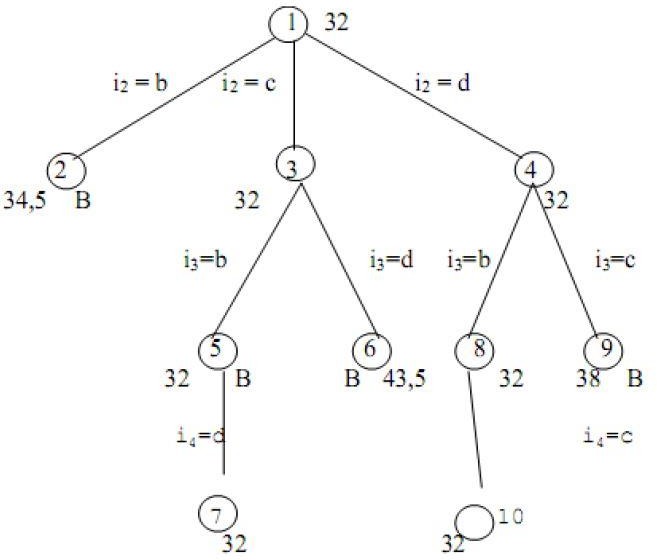
Hasil reduksi ini menghasilkan matriks B.

Ulangi langkah diatas sampai tiba di node daun. Semua simpul hidup yang nilainya lebih besar dari cost tur lengkap akan dibunuh (B) karena tidak mungkin lagi menghasilkan perjalanan dengan bobot < cost tur lengkap.

#### Branch and Bound dengan Bobot Tur Lengkap pada kasus TSP

Apabila menggunakan bobot tour lengkap, taksiran dari batas dapat dihitung dengan:



Solusi dari TSP pada Gambar 8.1 dengan algoritma B&B yaitu:



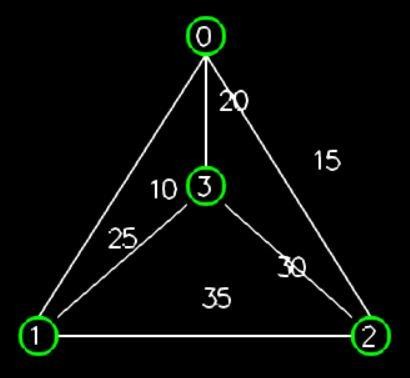
56

Gambar 8.2. Solusi algoritma B&B pada kasus TSP

(**Sumber**: Slide Mata Kuliah Strategi Algoritmik IF2211– ITB, Rinaldi Munir, 2019)

## PRE TEST

Analisis graf pada Gambar 8.3 di bawah ini dan tuliskan satu solusi jalur dengan cost minimum yang mungkin untuk kasus TSP menggunakan algoritma Branch and Bound!



Gambar 8.3. Kasus TSP untuk Pre Test dan Post Test Praktikum 8

## ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. DevC++.
3. OpenGL Library.

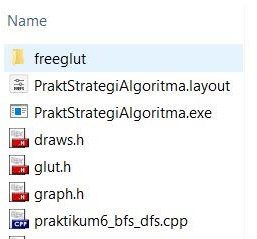
## LANGKAH PRAKTIKUM



57

### PERSIAPAN

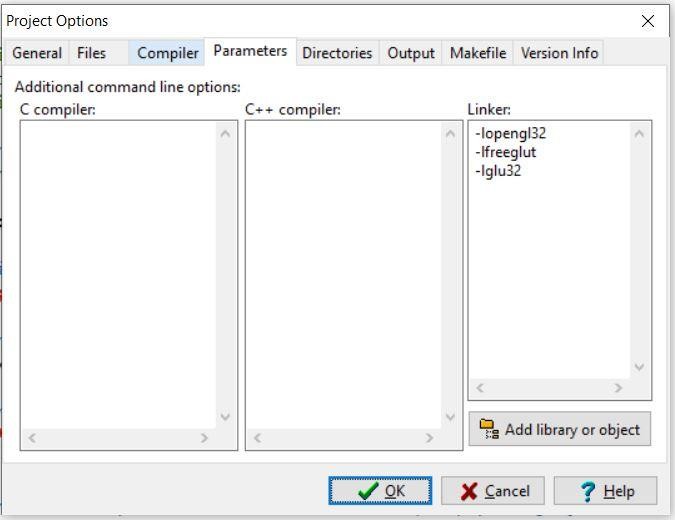
1. Buka DevC++ dan buat project baru dengan nama **praktikum08.**
2. Download library **freeglut** di elearning Strategi Algoritma di bagian atas. Ekstrak .rar nya di lokasi folder project anda berada. Contoh tampilan folder project ditunjukkan pada Gambar 8.4.



Gambar 8.4. Contoh isian folder project

1. Setting OpenGL library pada DevC++ dengan klik Project – Project options – Parameters lalu isikan

–lopengl32 –lfreeglut –lglu32 pada Linker seperti pada Gambar 8.5.



Gambar 8.5. Setting OpenGL pada DevC++

1. Download file tambahan praktikum yaitu **draws.h** dan **graph.h** untuk menggambar graf dengan OpenGL dari elearning Strategi Algoritma. Ekstrak .rar nya lalu copy kan ke folder project anda.
2. Buat file baru dengan nama **praktikum08.cpp**.

### PRAKTIKUM

1. Tuliskan kode berikut pada praktikum08.cpp untuk inisialisasi variable dan jendela OpenGL.

// Praktikum 8: Branch and Bound

// Algoritma Branch and Bound untuk menyelesaikan Travelling Salesman Problem



58

#include <windows.h> #include <iostream> #include <list>

#include "draws.h"

using namespace std;

// global Graph Graph graph;

char markText[10];

// hasil dari algoritma vector<int> pathSequence;

// batas bawah dari cost untuk pruning tree float lowerBound;

// cost tur lengkap float completeCost;

// fungsi untuk menandai hasil void **drawResult**()

{

glPushMatrix();

// gambar edges float radius = 15.0f;

for (int i=1; i<pathSequence.size(); i++)

{

int sourceIdx = pathSequence.at(i-1); int targetIdx = pathSequence.at(i); drawLine(

graph.getNodePosition(), sourceIdx,

targetIdx, Vec3(1.0f,0.0f,1.0f),

radius, 3.0f, true);

}

// gambar teks sprintf(markText,"%s","start"); Vec3 position(

graph.getNodePosition()[pathSequence.at(0)].getX()+2.0f\*radius, graph.getNodePosition()[pathSequence.at(0)].getY()+2.0f\*radius, 0.0f);

drawText(position, Vec3(1.0f,0.0f,1.0f), markText, radius, 2.0f);

glPopMatrix();

}

// taruh semua obyek yang akan digambar di fungsi display() void **displayGraph**()

{

// bersihkan dan reset layar dan buffer glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

// posisikan kamera pandang

gluLookAt(0.0f, 0.0f, 2.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);

// panggil fungsi untuk menggambar obyek drawNodes();

drawResult(); drawEdges(); drawDistances();

1. Tuliskan kode fungsi berikut untuk menerapkan algoritma Branch and Bound untuk kasus TSP.



59

// tampilkan obyek ke layar glutSwapBuffers();

}

// reduce cost pada baris

float **reduceRows**(vector<float> costMatrix[], int rowIdx, int numNodes)

{

float minValue = (float)INT\_MAX; for (int i=0; i<numNodes; i++)

if (costMatrix[rowIdx].at(i) < minValue) minValue = costMatrix[rowIdx].at(i);

for (int i=0; i<numNodes; i++)

{

if (costMatrix[rowIdx].at(i) >= (float)INT\_MAX) continue; costMatrix[rowIdx].at(i) -= minValue;

}

return (minValue >= (float)INT\_MAX ? 0.0f : minValue);

}

// reduce cost pada kolom

float **reduceCols**(vector<float> costMatrix[], int colIdx, int numNodes)

{

float minValue = (float)INT\_MAX; for (int i=0; i<numNodes; i++)

if (costMatrix[i].at(colIdx) < minValue) minValue = costMatrix[i].at(colIdx);

for (int i=0; i<numNodes; i++)

{

if (costMatrix[i].at(colIdx) >= (float)INT\_MAX) continue; costMatrix[i].at(colIdx) -= minValue;

}

return (minValue >= (float)INT\_MAX ? 0.0f : minValue);

}

// reduce cost matrix

float **reduceCostMatrix**(vector<float> costMatrix[], int numNodes)

{

float sums = 0.0f;

for (int i=0; i<numNodes; i++)

sums += reduceRows(costMatrix, i, numNodes); for (int i=0; i<numNodes; i++)

sums += reduceCols(costMatrix, i, numNodes); return sums;

}

// fungsi rekursif dari branch and bound void **BBRecursive**(

int rootIdx, int startIdx, int levelIdx, Graph graph, bool visited[],

vector<int> path, vector<float> costMatrix[], float costRoot)

{

// inisialisasi

int tLevelIdx = levelIdx;



60

bool\* tVisited = new bool[graph.getNumNodes()];

vector<float> \*tCostMatrix = new vector<float>[graph.getNumNodes()]; for (int i=0; i<graph.getNumNodes(); i++)

{

tVisited[i] = visited[i];

for (int j=0; j<graph.getNumNodes(); j++) tCostMatrix[i].push\_back(costMatrix[i].at(j));

}

vector<int> tPathSequence;

for (int i=0; i<path.size(); i++) tPathSequence.push\_back(path.at(i));

// update variabel tLevelIdx++; tVisited[startIdx] = true;

tPathSequence.push\_back(startIdx);

// proses semua node yang bertetangga dengan node sekarang for (int n=0; n<graph.getAdjNodes()[startIdx].size(); n++)

{

int nodeIdx = graph.getAdjNodes()[startIdx].at(n);

// apabila node belum dikunjungi if (!tVisited[nodeIdx])

{

// update cost matrix

for (int i=0; i<graph.getNumNodes(); i++)

{

tCostMatrix[startIdx].at(i) = (float)INT\_MAX; tCostMatrix[i].at(nodeIdx) = (float)INT\_MAX;

}

tCostMatrix[nodeIdx].at(0) = (float)INT\_MAX;

// hitung reduce cost matrix di node cabang

float r = reduceCostMatrix(tCostMatrix, graph.getNumNodes()); float costNode = costRoot + costMatrix[startIdx].at(nodeIdx) + r;

// rekursi node cabang apabila cost <= batas bawah cost if (costNode <= lowerBound)

{

// rekursi node cabang

BBRecursive(rootIdx, nodeIdx, tLevelIdx, graph, tVisited, tPathSequence, tCostMatrix, costNode);

// bila mencapai simpul daun

if (tLevelIdx == graph.getNumNodes()-1)

{

// hitung cost tur lengkap tPathSequence.push\_back(nodeIdx); tPathSequence.push\_back(rootIdx); float finalCost = 0.0f;

for (int i=0; i<tPathSequence.size();i++) if (i >= 1)

finalCost += graph.getNodeDistance()[tPathSequence.at(i-1)].at(tPathSequence.at(i));

// apabila cost < cost tur lengkap sebelumnya if (finalCost < completeCost)

{

lowerBound = costNode; completeCost = finalCost;

1. Tuliskan kode tester berikut untuk menerapkan algoritma B&B pada kasus di Gambar 8.1.



61

// update jalur tur lengkap pathSequence.clear();

for (int i=0; i<tPathSequence.size(); i++) pathSequence.push\_back(tPathSequence.at(i));

}

}

}

// reset cost matrix

for (int i=0; i<graph.getNumNodes(); i++)

for (int j=0; j<graph.getNumNodes(); j++) tCostMatrix[i].at(j) = costMatrix[i].at(j);

}

}

}

// fungsi penerapan branch and bound void **BB**(int startIdx, Graph graph)

{

// inisialisasi

completeCost = (float)INT\_MAX; lowerBound = (float)INT\_MAX; pathSequence.clear();

bool\* visited = new bool[graph.getNumNodes()];

vector<float>\* costMatrix = new vector<float>[graph.getNumNodes()]; for (int i=0; i<graph.getNumNodes(); i++)

{

visited[i] = false;

for (int j=0; j<graph.getNumNodes(); j++) costMatrix[i].push\_back(graph.getCostMatrix()[i].at(j));

}

// hitung reduce cost matrix di node akar

float costRoot = reduceCostMatrix(costMatrix, graph.getNumNodes());

// rekursi BB

BBRecursive(startIdx, startIdx, 0, graph, visited, pathSequence, costMatrix, costRoot);

// cetak hasil jalur tur lengkapnya float finalDistance = 0.0f;

cout << "Final solusi = ";

for (int i=0; i<pathSequence.size();i++)

{

cout << pathSequence.at(i) << " "; if (i >= 1)

finalDistance += graph.getNodeDistance()[pathSequence.at(i- 1)].at(pathSequence.at(i));

}

cout << " = " << finalDistance;

}

// kode tester

int **main**(int argc, char\*\* argv)

{

// inisialisasi graf graph.setIsDirected(false); graph.setNumLevels(2); graph.setNumNodes(4);

1. Jalankan kodenya untuk menerapkan algoritma Branch and Bound pada graf diatas. Hasil penerapan ditunjukkan pada Gambar 8.6.



62

// tambahkan node graph.addNode(0, 0, 1.2f);

graph.addNode(1, 0, 1.0f);

graph.addNode(2, 1, 1.0f);

graph.addNode(3, 1, 1.0f);

// tambahkan edge dengan jarak graph.addEdge(0, 1, 12.0f);

graph.addEdge(0, 2, 10.0f);

graph.addEdge(0, 3, 5.0f);

graph.addEdge(1, 2, 9.0f);

graph.addEdge(1, 3, 8.0f);

graph.addEdge(2, 3, 15.0f);

// estimate node position graph.setNodePosition(); graph.setAdjStatus();

**// terapkan algoritma BB mulai dari indeks node-0 BB(0, graph);**

// inisialisasi jendela OpenGL glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH);

// set ukuran jendela tampilan dalam piksel glutInitWindowSize(SCREEN\_WIDTH, SCREEN\_HEIGHT);

// posisi jendela dilayar komputer dalam piksel glutInitWindowPosition(100, 100);

// judul jendela (isi dengan NAMA / NIM - JUDUL PRAKTIKUM) glutCreateWindow("NAMA / NIM - PRAKTIKUM STIMA");

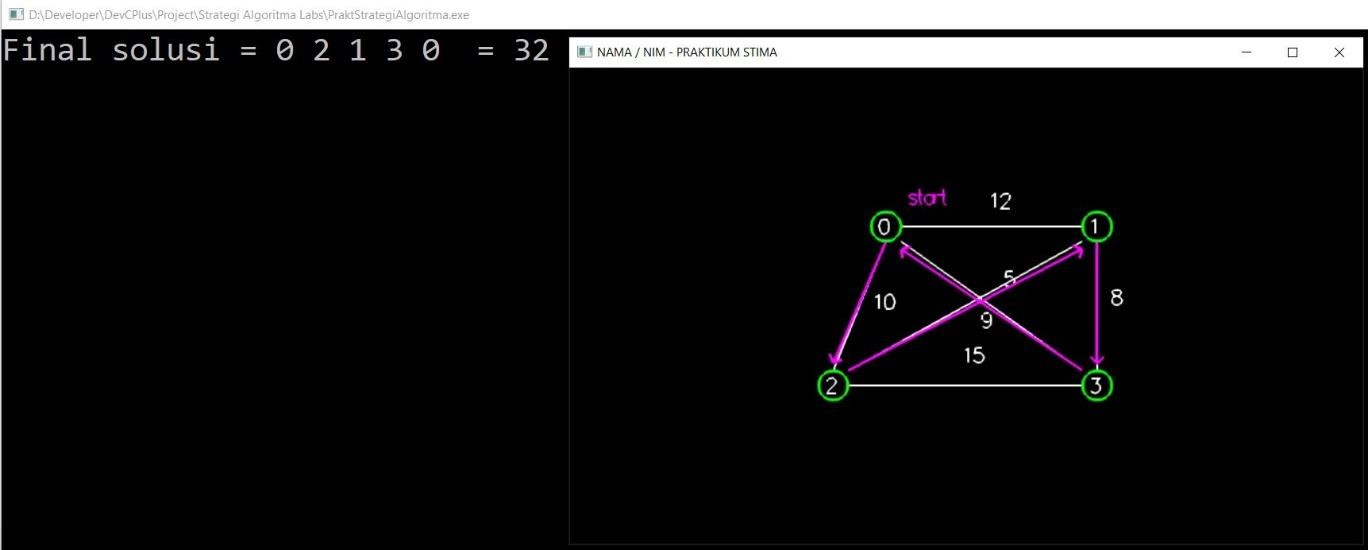
// panggil fungsi init untuk inisialisasi awal initView();

// event handler untuk display glutDisplayFunc(displayGraph); glutReshapeFunc(reshapeView);

// looping glutMainLoop();

return 0;

}



Gambar 8.6. Hasil penerapan algoritma B&B pada kasus TSP

1. Amati hasilnya kemudian jelaskan cara kerjanya.

63

## TUGAS / POST TEST

1. Terapkan algoritma branch and bound pada graf di Gambar 8.3 (lihat bagian Pre Test) untuk kasus TSP dengan memodifikasi kode tester praktikum branch and bound yang sudah anda buat!
2. Analisis apakah hasil dari program sama dengan jawaban pre test anda!

# PRAKTIKUM 9: ALGORITMA A\*



64

Pertemuan ke 9

Total Alokasi Waktu : 90 menit

* Pre-Test : 15 menit
* Praktikum : 45 menit
* Post-Test : 30 menit

Total Skor Penilaian : 100%

* Pre-Test : 20 %
* Praktikum : 30 %
* Post-Test : 50 %

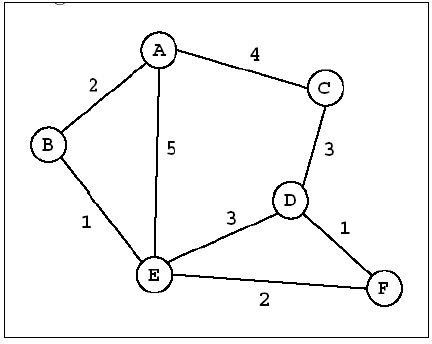
## TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan menerapkan algoritma A\* untuk mencari rute terpendek.

Indikator ketercapaian diukur dengan keberhasilan mahasiswa dalam menerapkan algoritma A\* untuk mencari rute terpendek.

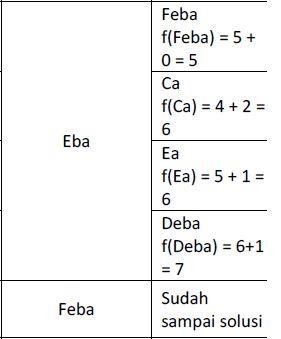
## TEORI PENDUKUNG

Algoritma A\* mempunyai ide untuk menghindari jalur yang terlalu mahal. Untuk melakukan hal tersebut digunakan fungsi evaluasi yaitu f(n) = g(n) + h(n) dimana g(n) merupakan cost untuk mencapai n, h(n) adalah perkiraan cost dari n ke tujuan dan f(n) merupakan perkiraan total cost melalui jalur n ke tujuan. Perkiraan dari cost heuristic h(n) dapat menggunakan rumus pengukuran jarak seperti jarak Manhattan, jarak Euclidean dan lainnya. Untuk memilih jalur, algoritma A\* memilih total cost f(n) yang paling murah diantara jalur yang mungkin ditempuh. Algoritma A\* dapat menghasilkan solusi rute dengan cost terendah. Contoh kasus pencarian rute terdekat ditunjukkan pada Gambar 9.1.

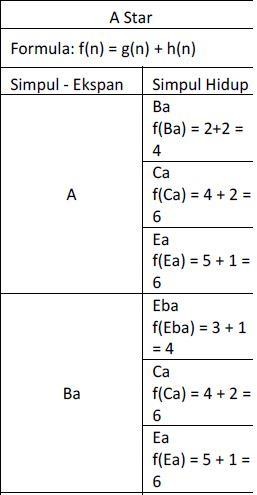


Gambar 9.1 Contoh kasus pencarian rute terdekat dari simpul A ke simpul F. (**Sumber**: Slide Mata Kuliah Strategi Algoritmik IF2211– ITB, Rinaldi Munir, 2019)

65

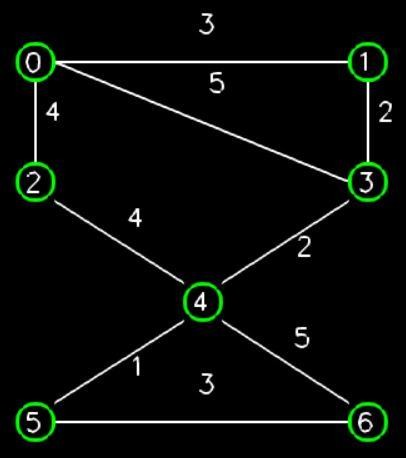


Solusi dari kasus tersebut apabila diselesaikan menggunakan algoritma A\* yaitu:



## PRE TEST

Analisis graf pada Gambar 9.2 di bawah ini dan tuliskan satu solusi jalur terpendek yang mungkin menggunakan algoritma A\* apabila jarak heuristiknya merupakan banyaknya busur minimal dari node asal ke node target!



Gambar 9.2. Graf untuk Pre Test dan Post Test Praktikum 9

## ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. DevC++.
3. OpenGL Library.

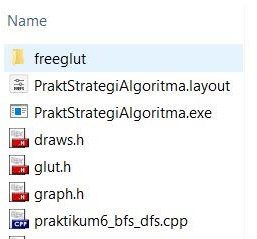
## LANGKAH PRAKTIKUM



66

### PERSIAPAN

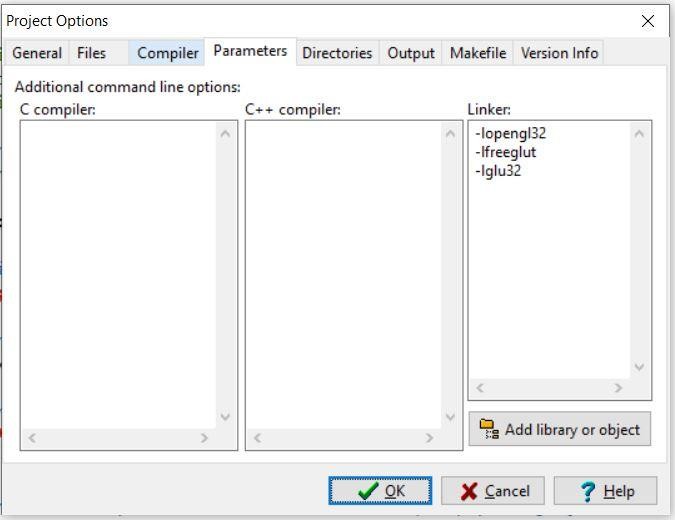
1. Buka DevC++ dan buat project baru dengan nama **praktikum09.**
2. Download library **freeglut** di elearning Strategi Algoritma di bagian atas. Ekstrak .rar nya di lokasi folder project anda berada. Contoh tampilan folder project ditunjukkan pada Gambar 9.3.



Gambar 9.3. Contoh isian folder project

1. Setting OpenGL library pada DevC++ dengan klik Project – Project options – Parameters lalu isikan

–lopengl32 –lfreeglut –lglu32 pada Linker seperti pada Gambar 9.4.



Gambar 9.4. Setting OpenGL pada DevC++

1. Download file tambahan praktikum yaitu **draws.h** dan **graph.h** untuk menggambar graf dengan OpenGL dari elearning Strategi Algoritma. Ekstrak .rar nya lalu copy kan ke folder project anda.
2. Buat file baru dengan nama **praktikum09.cpp**.

### PRAKTIKUM

1. Tuliskan kode berikut pada praktikum09.cpp untuk inisialisasi variable dan jendela OpenGL.

// Praktikum 9: A\*

// Penerapan contoh di slide kuliah materi A\* untuk mencari rute terdekat



67

#include <windows.h> #include <iostream> #include <list>

#include "draws.h"

using namespace std;

// global Graph Graph graph;

char markText[10]; vector<int> pathSequence; float finalCost;

// fungsi untuk menandai jalur hasil pencarian void **drawResult**()

{

glPushMatrix();

// gambar edges float radius = 15.0f;

for (int i=1; i<pathSequence.size(); i++)

{

int sourceIdx = pathSequence.at(i-1); int targetIdx = pathSequence.at(i); drawLine(

graph.getNodePosition(), sourceIdx,

targetIdx, Vec3(1.0f,0.0f,1.0f),

radius, 3.0f, true);

}

// gambar teks sprintf(markText,"%s","start"); Vec3 position(

graph.getNodePosition()[pathSequence.at(0)].getX()+2.0f\*radius, graph.getNodePosition()[pathSequence.at(0)].getY()+2.0f\*radius, 0.0f);

drawText(position, Vec3(1.0f,0.0f,1.0f), markText, radius, 2.0f);

glPopMatrix();

}

// taruh semua obyek yang akan digambar di fungsi display() void **displayGraph**()

{

// bersihkan dan reset layar dan buffer glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

// posisikan kamera pandang

gluLookAt(0.0f, 0.0f, 2.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);

// panggil fungsi untuk menggambar obyek drawNodes();

if (!pathSequence.empty()) drawResult();

drawEdges(); drawDistances();

// tampilkan obyek ke layar glutSwapBuffers();

}



68

1. Tuliskan kode fungsi berikut untuk menerapkan algoritma A\* untuk kasus pencarian rute terdekat.

// hitung jarak heuristik node asal ke node target misalnya

// menggunakan banyaknya busur minimal dari simpul asal ke target float **computeHeuristic**(

Graph graph, bool visited[], int startIdx, int endIdx)

{

// inisialisasi

bool\* localVisited = new bool[graph.getNumNodes()]; for (int i=0; i<graph.getNumNodes(); i++)

localVisited[i] = visited[i];

// jika simpul asal sudah terhubung ke simpul target if (graph.getAdjStatus()[startIdx].at(endIdx))

return 1.0f;

// jika sudah sampai di simpul target if (startIdx == endIdx)

return 0.0f;

// hitung banyaknya busur minimum antara simpul asal dan target float distance = 0.0f;

for (int i=0; i<graph.getAdjNodes()[startIdx].size(); i++)

{

int nodeIdx = graph.getAdjNodes()[startIdx].at(i); if (!localVisited[nodeIdx])

{

localVisited[nodeIdx] = true;

distance = 1.0f + computeHeuristic(graph, localVisited, nodeIdx, endIdx);

}

}

return distance;

}

// hitung cost perjalanan float **computeCost**(

Graph graph, int startIdx, int nodeIdx, int endIdx)

{

// inisialisasi

bool\* visited = new bool[graph.getNumNodes()]; for (int i=0; i<graph.getNumNodes(); i++)

visited[i] = false;

// hitung jarak g(n)

float distance2root = 0.0f; if (pathSequence.size() > 1)

for (int i=1; i<pathSequence.size(); i++)

distance2root += graph.getNodeDistance()[pathSequence.at(i- 1)].at(pathSequence.at(i));

float distance2travel = distance2root + graph.getNodeDistance()[startIdx].at(nodeIdx);

// hitung jarak h(n)



69

float distance2target = computeHeuristic(graph, visited, nodeIdx, endIdx);

// f(n) = g(n) + h(n)

return distance2travel + distance2target;

}

// fungsi rekursif dari branch and bound void **AStarRecursive**(

Graph graph, int startIdx, int endIdx, bool visited[],

vector<int> &pathSequence)

{

// tandai node sekarang sebagai node yang dikunjungi visited[startIdx] = true; pathSequence.push\_back(startIdx);

// base untuk berhenti

if (startIdx == endIdx) return;

// cari cost minimum dari semua node yang bertetangga dengan node sekarang int minIdx;

float minCost = (float)INT\_MAX;

for (int i=0; i<graph.getAdjNodes()[startIdx].size(); i++)

{

int nodeIdx = graph.getAdjNodes()[startIdx].at(i); if (!visited[nodeIdx])

{

float cost = computeCost(graph, startIdx, nodeIdx, endIdx); if (cost < minCost)

{

minCost = cost; minIdx = nodeIdx; finalCost = minCost;

}

}

}

// rekursi ke node dengan cost minimum

AStarRecursive(graph, minIdx, endIdx, visited, pathSequence);

}

// fungsi penerapan branch and bound

void **AStar**(Graph graph, int startIdx, int endIdx)

{

// inisialisasi finalCost = 0.0f;

bool\* visited = new bool[graph.getNumNodes()]; for (int i=0; i<graph.getNumNodes(); i++)

visited[i] = false; pathSequence.clear();

// terapkan rekursi dari BB

AStarRecursive(graph, startIdx, endIdx, visited, pathSequence);

// cetak hasilnya

cout << "Final solusi = ";

for (int i=0; i<pathSequence.size(); i++) cout << pathSequence.at(i) << " ";

cout << "= " << finalCost;

}

70

1. Tuliskan kode tester berikut untuk menerapkan algoritma A\* pada kasus di Gambar 9.1.

// kode tester

int **main**(int argc, char\*\* argv)

{

// inisialisasi graf graph.setIsDirected(false); graph.setNumLevels(4); graph.setNumNodes(6);

// tambahkan node graph.addNode(0, 0, 0.85f);

graph.addNode(1, 1, 0.85f);

graph.addNode(2, 1, 0.9f);

graph.addNode(3, 2, 1.05f);

graph.addNode(4, 3, 1.1f);

graph.addNode(5, 3, 1.0f);

// tambahkan edge graph.addEdge(0, 1, 2.0f);

graph.addEdge(0, 2, 4.0f);

graph.addEdge(0, 4, 5.0f);

graph.addEdge(1, 4, 1.0f);

graph.addEdge(2, 3, 3.0f);

graph.addEdge(3, 4, 3.0f);

graph.addEdge(3, 5, 1.0f);

graph.addEdge(4, 5, 2.0f);

// estimate node position graph.setNodePosition(); graph.setAdjStatus();

**// penerapan algoritma A\* int startIdx = 0;**

**int endIdx = 5;**

**AStar(graph, startIdx, endIdx);**

// inisialisasi jendela OpenGL glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH);

// set ukuran jendela tampilan dalam piksel glutInitWindowSize(SCREEN\_WIDTH, SCREEN\_HEIGHT);

// posisi jendela dilayar komputer dalam piksel glutInitWindowPosition(100, 100);

// judul jendela (isi dengan NAMA / NIM - JUDUL PRAKTIKUM) glutCreateWindow("NAMA / NIM - PRAKTIKUM STIMA");

// panggil fungsi init untuk inisialisasi awal initView();

// event handler untuk display glutDisplayFunc(displayGraph); glutReshapeFunc(reshapeView);

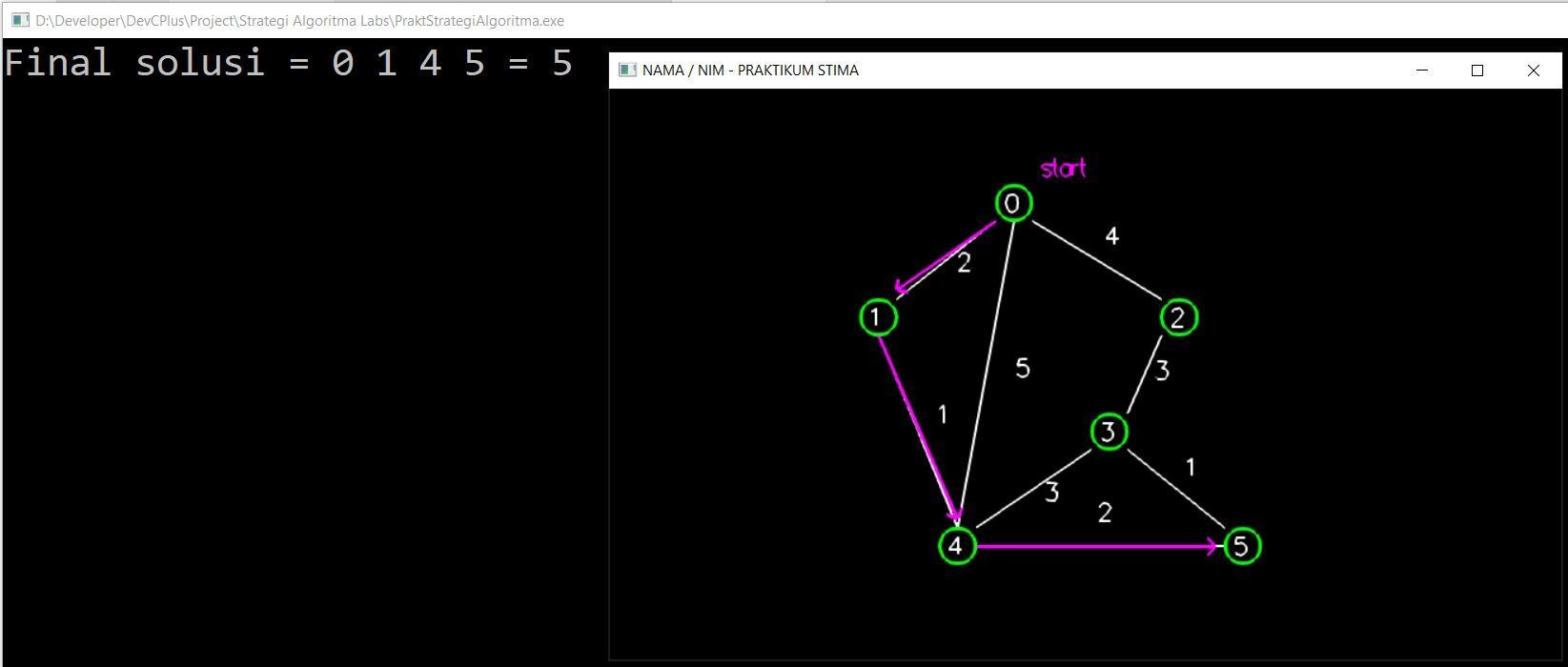
// looping glutMainLoop();

return 0;

}

1. Jalankan kodenya unutuk menerapkan algoritma A\* pada graf diatas. Hasil penerapan ditunjukkan pada Gambar 9.5.

Gambar 9.5. Hasil penerapan algoritma A\* untuk mencari rute terdekat



71

1. Amati hasilnya kemudian jelaskan cara kerjanya.

## TUGAS

1. Terapkan algoritma A\* pada graf di Gambar 9.2 (lihat bagian Pre Test) untuk mencari rute terpendek dengan memodifikasi kode tester praktikum A\* yang sudah anda buat!
2. Analisis apakah hasil dari program sama dengan jawaban pre test anda!

72

# PRAKTIKUM 10: ALGORITMA STRING MATCHING

Pertemuan ke 10

Total Alokasi Waktu : 90 menit

* Pre-Test : 15 menit
* Praktikum : 45 menit
* Post-Test : 30 menit

Total Skor Penilaian : 100%

* Pre-Test : 20 %
* Praktikum : 30 %
* Post-Test : 50 %

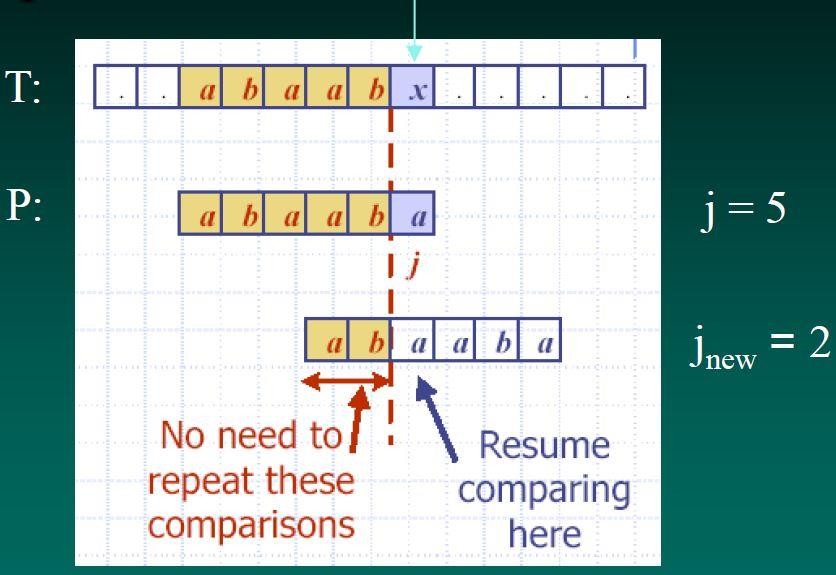
## TUJUAN DAN INDIKATOR CAPAIAN

Setelah mengikuti praktikum ini mahasiswa diharapkan menerapkan algoritma string matching dengan Knuth-Morris-Pratt (KMP) dan algoritma Boyer-Moore.

Indikator ketercapaian diukur dengan keberhasilan mahasiswa dalam menerapkan string matching dengan Knuth-Morris-Pratt (KMP) dan algoritma Boyer-Moore.

## TEORI PENDUKUNG

Pencocokan string merupakan proses pencocokan pola karakter pada teks. Beberapa algoritma dapat digunakan untuk pencocokan string mulai dari brute force, Knuth-Morris-Pratt (KMP), dan Booyer-Moore. Pada praktikum ini akan dibahas dan diimpelemtasikan algoritma KMP untuk pencocokan string. Algoritma KMP mencari pola pada teks mulai dari kiri ke kanan seperti algoritma brute force. Akan tetapi, KMP melakukan pergeseran dalam pencarian pola lebih baik dari pada brute force. Apabila ada ketidakcocokan antar teks T di T(i) dengan pola P di P(j) misalnya T(i) != P(j), algoritma KMP akan menggeser pola untuk menghindari perbandingan yang tidak penting. Algoritma KMP diilustrasikan pada Gambar 10.1.



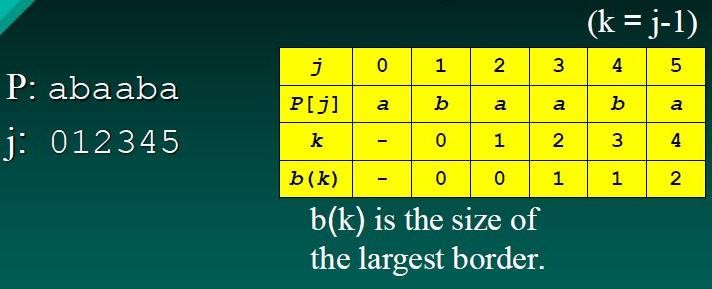
Gambar 10.1 Ilustrasi algoritma KMP

(**Sumber**: Slide Mata Kuliah Strategi Algoritmik IF2211– ITB, Rinaldi Munir, 2019)

73

Algoritma KMP melakukan pra proses terhadap pola untuk menemukan kecocokan dari awalan pola dengan pola itu sendiri. Apabila j merupakan posisi ketidakcocokan pada pola dan k merupakan posisi sebelum ketidakcocokan terjadi (k = j-1) maka fungsi pinggiran KMP (KMP border function) atau b(k) didefinisikan sebagai ukuran dari awalan terbesar dari P[0…k] yang juga merupakan akhiran dari P[1…k]. Gambar 10.2 merupakan ilustrasi dari fungsi pinggiran KMP pada pola “abaaba”. Dari Gambar

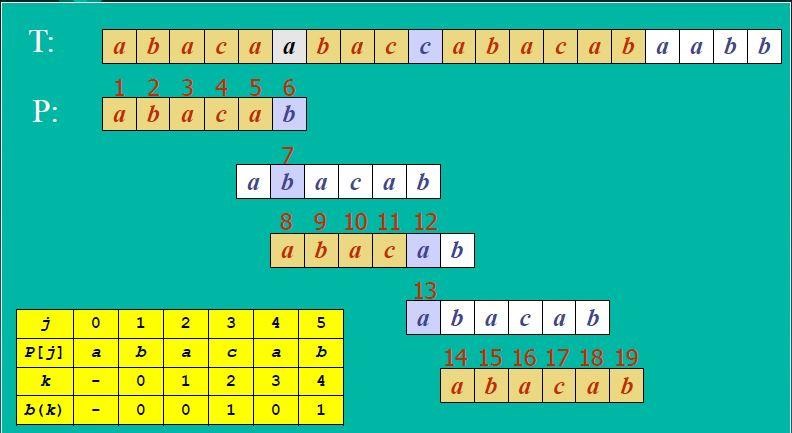
10.2 makna dari b(4) sama dengan 2 berarti menemukan awalan terbesar dari P[0…4] yang juga merupakan akhiran dari P[1…4] atau menemukan awalan terbesar dari “abaab” yang juga merupakan akhiran dari “baab”. Jadi hasilnya yaitu “ab” atau sama dengan 2.



Gambar 10.2. Ilustrasi fungsi pinggiran KMP

(**Sumber**: Slide Mata Kuliah Strategi Algoritmik IF2211– ITB, Rinaldi Munir, 2019)

Contoh penerapan KMP pada teks “abacaabaccabacabaabb” dengan pola yang dicari yaitu “abacab” ditunjukkan pada Gambar 10.3.



Gambar 10.3. Contoh penerapan algoritma KMP

(**Sumber**: Slide Mata Kuliah Strategi Algoritmik IF2211– ITB, Rinaldi Munir, 2019)

## PRE TEST

Baca slide kuliah materi pencocokan string dengan algoritma Boyer-Moore di halaman 41. Tulis notasi algoritma untuk algoritma Boyer-Moore tersebut!

## ALAT DAN BAHAN

Alat dan bahan yang digunakan dalam praktikum ini yaitu:

1. Komputer.
2. Dev C++.



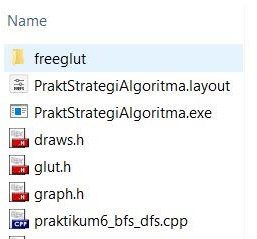
74

1. OpenGL Library.

## LANGKAH PRAKTIKUM

### PERSIAPAN

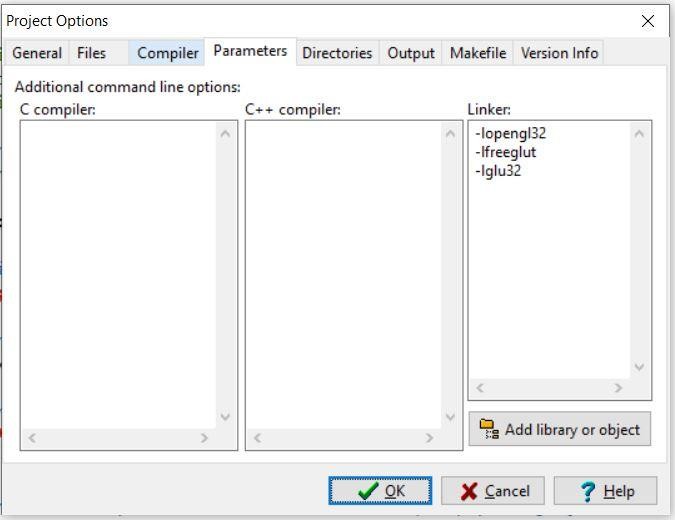
1. Buka DevC++ dan buat project baru dengan nama **praktikum10.**
2. Download library **freeglut** di elearning Strategi Algoritma di bagian atas. Ekstrak .rar nya di lokasi folder project anda berada. Contoh tampilan folder project ditunjukkan pada Gambar 10.4.



Gambar 10.4. Contoh isian folder project

1. Setting OpenGL library pada DevC++ dengan klik Project – Project options – Parameters lalu isikan

–lopengl32 –lfreeglut –lglu32 pada Linker seperti pada Gambar 10.5.



Gambar 10.5. Setting OpenGL pada DevC++

1. Download file tambahan praktikum yaitu **draws.h** dan **graph.h** untuk menggambar graf dengan OpenGL dari elearning Strategi Algoritma. Ekstrak .rar nya lalu copy kan ke folder project anda.
2. Buat file baru dengan nama **praktikum10.cpp**.

### PRAKTIKUM

75

1. Tuliskan kode berikut pada praktikum10.cpp untuk inisialisasi variable dan jendela OpenGL.

// Praktikum 10: String Matching

// Penerapan contoh di slide kuliah materi string matching dengan KMP

#include <windows.h> #include <iostream> #include <string>

#include "draws.h" using namespace std;

// global Graph Graph graph;

char markText[1024]; string ptext; string pattern;

int startIdx;

// fungsi untuk menandai jalur hasil pencarian void **drawResult**()

{

glPushMatrix();

sprintf(markText,"%s",ptext.c\_str()); Vec3 position(-350.0f, 0.0f, 0.0f); Vec3 color1(1.0f,1.0f,1.0f);

Vec3 color2(1.0f,0.0f,1.0f); if (startIdx >= 0)

drawText(position, color1, color2, markText, 2.0f, startIdx, pattern.length());

else

drawText(position, color1, color1, markText, 2.0f, 0, pattern.length());

glPopMatrix();

}

// taruh semua obyek yang akan digambar di fungsi display() void **displayGraph**()

{

// bersihkan dan reset layar dan buffer glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

// posisikan kamera pandang

gluLookAt(0.0f, 0.0f, 2.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);

// panggil fungsi untuk menggambar obyek drawResult();

// tampilkan obyek ke layar glutSwapBuffers();

}

1. Tuliskan kode fungsi berikut untuk menerapkan algoritma KMP untuk pencocokan string.

// cari kecocokan dalam pola itu sendiri int\* **computeFail**(string pattern)

{

int\* fail = new int[pattern.length()]; fail[0] = 0;

int m = pattern.length();

1. Tuliskan kode tester berikut untuk menerapkan algoritma KMP pada kasus di Gambar 10.3.



76

int j = 0; int i = 1; while (i < m)

{

if (pattern.at(j) == pattern.at(i))

{

// karakter j+1 cocok fail[i] = j + 1; i++;

j++;

}

else if (j > 0) // prefiks j j = fail[j-1];

else

{

// tidak cocok fail[i] = 0; i++;

}

}

return fail;

}

// pencocokan string dengan KMP

int **KMPMatch**(string text, string pattern)

{

int n = text.length(); int m = pattern.length();

int\* fail = computeFail(pattern); int i=0;

int j=0; while (i < n)

{

if (pattern.at(j) == text.at(i))

{

if (j == m -1)

return i -m + 1; // cocok i++;

j++;

}

else if (j > 0)

j = fail[j-1]; else

i++;

}

return -1; // tidak cocok

}

// kode tester

int **main**(int argc, char\*\* argv)

{

// teks yang akan dicocokkan ptext = "abacaabacabacababa";

// pola yang akan dicocokkan pattern = "acabaca";

cout << "cari: [" << pattern << "] di dalam teks: [" << ptext << "]\n";

1. Jalankan kodenya unutuk menerapkan algoritma KMP. Hasil penerapan algoritma KMP ditunjukkan pada Gambar 10.6.



77

**// penerapan algoritma KMP**

**startIdx = KMPMatch(ptext, pattern);**

if (startIdx == -1)

cout << "pola tidak ditemukan pada teks\n";

else

cout << "pola ditemukan pada posisi-" << startIdx << "\n";

// inisialisasi jendela OpenGL glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH);

// set ukuran jendela tampilan dalam piksel glutInitWindowSize(SCREEN\_WIDTH, SCREEN\_HEIGHT);

// posisi jendela dilayar komputer dalam piksel glutInitWindowPosition(100, 100);

// judul jendela (isi dengan NAMA / NIM - JUDUL PRAKTIKUM) glutCreateWindow("NAMA / NIM - PRAKTIKUM STIMA");

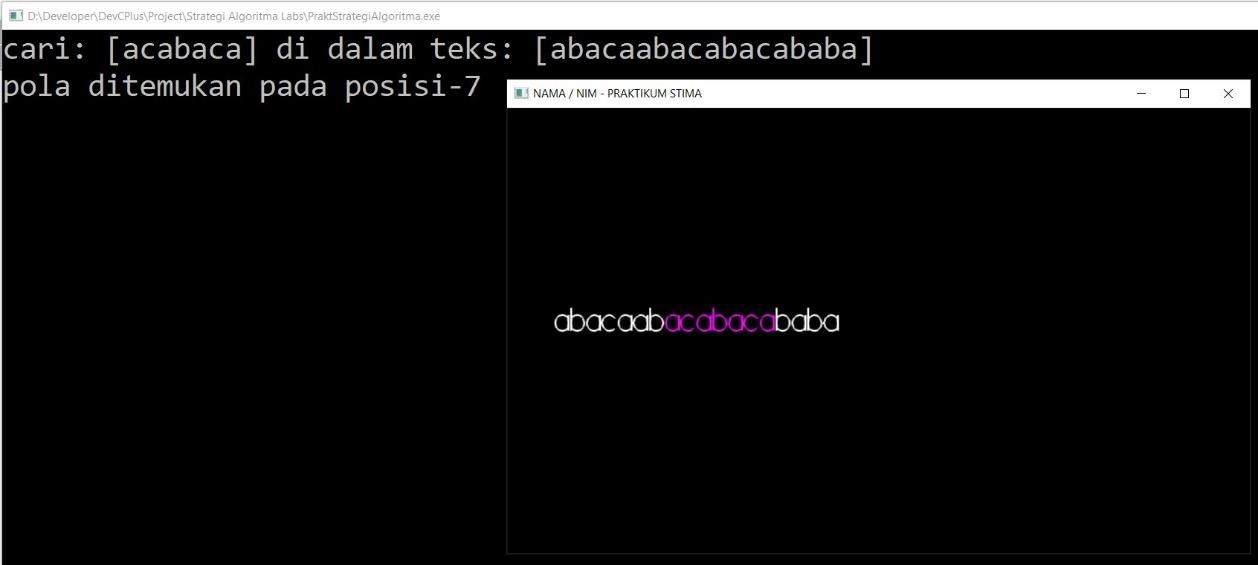
// panggil fungsi init untuk inisialisasi awal initView();

// event handler untuk display glutDisplayFunc(displayGraph); glutReshapeFunc(reshapeView);

// looping glutMainLoop();

return 0;

}



Gambar 10.6. Hasil penerapan algoritma KMP

1. Amati hasilnya kemudian jelaskan cara kerjanya.

## TUGAS / POST TEST

1. Tulis kode program untuk notasi algoritma Boyer-Moore yang anda buat di Pre Test dengan cara memodifikasi kode praktikum yang sudah anda buat diatas!
2. Terapkan untuk kasus text dan pattern yang sama seperti pada praktikum algoritma KMP!

**LEMBAR JAWABAN PRE-TEST DAN POST-TEST PRAKTIKUM**



78

|  |  |  |
| --- | --- | --- |
| **Nama : NIM :** | **Asisten:**  **Paraf Asisten:** | **Tanggal: Nilai:** |

# DAFTAR PUSTAKA



79

1. Design Methods and Analysis of Algorithms, Basu SK, Prentice-Hall of India, 2012
2. Rinaldi Munir, Strategi Algoritmik, Informatika ITB, Bandung 2009.
3. An Introduction to The Analysis of Algorithms, SEDGEWICK, Robert, Addison-Wesley, Massachussets, 1996
4. David Luebke, Algorithms, Virginia University, 2009
5. <https://dosen.perbanas.id/kompleksitas-algoritma/>
6. <http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/stima18-19.htm#SlideKuliah>



80