

ILK2102P

DESAIN & ANALISIS ALGORITMA

Disusun oleh:

1. Dr. Mohammad Andri Budiman, S.T., M.Comp.Sc., M.E.M.
2. Anandhini Medianty Nababan, S. Kom., M. T.
3. Arian Syah Putra
4. Ilma Sakinah Parinduri
5. Tamir Rusydi Hega
6. Petrus Marcellino H. Tampubolon
7. Wilson

No . Dokumen : IK-GKM-Fasilkom-TI-009-DAA

Berlaku Efektif : 29 September 2022

Revisi : 09


Direvisi : TIM GKM ILK Fasilkom-TI

Diperiksa/Disetujui : GJM dan GKM ILK Fasilkom-TI USU

Disahkan Oleh : Dr. Maya Silvi Lydia B.Sc., M.Sc.
Dekan Fasilkom-TI USU



**S-1 ILMU KOMPUTER
FASILKOM-TI
USU**

	MODUL PRAKTIKUM	No. Dokumen	:	IK-GKM-ILK-Fasilkom-TI-00-DAA
		Edisi	:	01
		Revisi	:	09
		Berlaku Efektif	:	26 September 2022
		Halaman	:	i dari ii
KATA PENGANTAR				

Puji syukur kehadiraT Allah SWT. Tuhan Yang Maha Esa karena atas berkat dan rahmat-Nya sehingga penyusun dapat menyelesaikan buku penuntun praktikum ini. Sholawat dan salam kepada Nabi Muhammad saw. Sosok manusia teladan bagi hidup dan kehidupan manusia sampai akhir zaman.

Modul praktikum ini disusun berdasarkan silabus mata kuliah praktikum Program Studi S1 Ilmu Komputer Fasilkom-TI USU. Buku ini terdiri 8 modul atau judul percobaan yang terdiri dari tujuan, teori, latihan dan tugas praktikum dengan harapan mahasiswa dapat membuat jurnal sebagai tolak ukur bagi keberhasilan pencapaian tujuan praktikum. Buku penuntun praktikum ini merupakan buku pedoman dasar dan bahan ajar dalam pelaksanaan praktikum.

Atas tersusunnya buku ini, penyusun mengucapkan terima kasih kepada Ketua dan Sekretaris Program Studi S1 Ilmu Komputer, Bapak/Ibu Dosen, Staf Tata Usaha Program Studi S1 Ilmu Komputer dan semua pihak yang telah membantu dalam penyusunan buku penuntun praktikum ini. Buku penuntun praktikum digunakan oleh Program Studi S1 Ilmu Komputer Fasilkom-TI USU dan merupakan implementasi dari pelaksanaan Sistem Manajemen Mutu Perguruan Tinggi.

Buku penuntun ini adalah revisi dari buku penuntun praktikum terdahulu. Penyusun menyadari buku ini masih jauh dari kesempurnaan oleh karena itu penyusun sangat mengharapkan kritik dan saran dari para pembaca demi menyempurnakan buku penuntun praktikum ini. Dengan adanya buku penuntun praktikum diharapkan dapat membantu mahasiswa dalam pelaksanaan praktikum dan bermanfaat secara maksimal bagi semua pihak.

Medan, 26 September 2022

Ketua,

Dr. Amalia

NIP.

Program Studi S1-Ilmu Komputer, Fasilkom-TI USU

	MODUL PRAKTIKUM	No. Dokumen	:	IK-GKM-ILK-Fasilkom-TI-009-DAA
		Edisi	:	01
		Revisi	:	09
		Berlaku Efektif	:	26 September 2022
		Halaman	:	ii dari ii
DAFTAR ISI				

KATA PENGANTAR.....	2
MODUL I STRUKTUR DATA STATIS DAN DINAMIS.....	4
MODUL II STRUKTUR DATA STACK DAN QUEUE	8
MODUL III SENARAI BERANTAI (LINKED LIST)	13
MODUL IV SORTING	19
MODUL V MINIMUM SPANNING TREE	27
MODUL VI GRAPH	42
MODUL VII KOMPRESI DATA: ALGORITMA END TAGGED DENSE CODE (ETDC).55	
MODUL VIII STRING MATCHING.....	62

MODUL I

STRUKTUR DATA STATIS DAN DINAMIS

I. TUJUAN

1. Praktikan dapat menjelaskan konsep struktur data statis.
2. Praktikan dapat menjelaskan konsep struktur data dinamis.
3. Praktikan dapat menjelaskan perbedaan struktur data statis dan struktur data dinamis.
4. Praktikan dapat memberikan contoh penggunaan struktur data statis dan struktur data dinamis

II. TEORI

1. Struktur Data dan Algoritma

Berbicara mengenai struktur data dalam pemrograman, takkan terlepas dari kata algoritma. Struktur data adalah pengaturan data dalam memori komputer atau terkadang didalam disk dengan tujuan agar data dapat diakses secara efisien. Sedangkan algoritma sendiri merupakan langkah-langkah logis untuk memecahkan suatu masalah secara sistematis. Didalam struktur data algoritma digunakan untuk memanipulasi data yang tersimpan, mulai dari insert, delete, update, dan sebagainya.

2. STRUCT

Struct merupakan komponen yang sangat penting dalam struktur data, karena struct dapat menampung banyak data (variable) dengan tipe yang berbeda. Dalam mendeklarasikan struct, ada beberapa cara penulisan yang biasa umum digunakan.

```
Struct nama_struct{
    tipe_data_1 nama_var_1;
    tipe_data_2 nama_var_2;
    tipe_data_3 nama_var_3;
    .....
};

typedef struct {
    tipe_data_1 nama_var_1;
    .
    .
    .
    tipe_data_n nama_var_n;
} nama_struct;
```

3. Struktur Data Statis

Struktur data statis adalah struktur data yang memiliki alokasi data yang tetap. Data yang dapat dimasukkan tidak dapat ditambah lagi jika data yang telah ada telah mencapai nilai maksimum alokasi data dan jika data yang ada kurang dari jumlah maksimum alokasi data, maka alokasi memori yang disediakan tetap (statis).

```
/* Contoh Program ke : 1 */
/* Program : struktur data statis mahasiswa */
#include <conio.h>
#include <stdio.h>
#include <iostream>
#include <string.h>
#define nmax 5 // banyak data yang dapat ditampung
using namespace std;

int n = 0; // variable n sebagai banyak data yang telah tersimpan
struct data
{ // struktur data mahasiswa yg akan diinput
    int nim;
    char nama[20];
    char kom;
};
struct data maba[nmax]; // deklarasi variabel bertipe struct
void tambah_data(); // prototype fungsi void
void hapus_data();
void tampilkan_data();
main()
{
    int pil;
menu:
    cout << "\t\t MENU \n";
    cout << "1. tambah data\n";
    cout << "2. hapus data \n";
    cout << "3. tampilkan data \n";
    cout << "4. keluar \n";
    cout << "\n Pilih menu (1/2/3/4) ? ";
    cin >> pil;
    if (pil == 1)
        tambah_data();
    else if (pil == 2)
        hapus_data();
    else if (pil == 3)
        tampilkan_data();
    else if (pil == 4)
        exit(1); // atau bisa pakai return(0)
    else
    {
        cout << "pilihan tak tersedia...!!\n";
    }
}
```

```

    goto menu;
    getch();
}

void tambah_data()
{
    if (n < nmax)
    {
        cout << "nama : ";
        cin >> maba[n].nama;
        cout << "NIM : ";
        cin >> maba[n].nim;
        cout << "Kom : ";
        cin >> maba[n].kom;
        n++;
    }
    else
        cout << "\n data telah melebihi...!!\n maksimal data = " << nmax;
}

void hapus_data()
{
    int x;
    cout << "pilih data yang akan dihapus (1 s.d. 5) : ";
    cin >> x;
    if (x < n && x > 0)
    {
        for (int i = x; i < n; i++)
        {
            strcpy(maba[i - 1].nama, maba[x].nama);
            maba[i - 1].nim = maba[x].nim;
            maba[i - 1].kom = maba[x].kom;
        }
        n--;
    }
    else if (x == n)
        n--;
    else
        cout << "\n data yang ingin dihapus tidak ada...!!\n";
}

void tampilkan_data()
{
    if (n == 0)
        cout << "Tidak ada data yang disimpan...!!\n";
    else
    {
        for (int i = 0; i < n; i++)
        {
            cout << "Data ke-" << i + 1 << " : ";

```

```

cout << "\nNama : " << maba[i].nama;
cout << "\nNIM : " << maba[i].nim;
cout << "\nKom : " << maba[i].kom;
cout << endl;
    }
}
}

```

4. Struktur data Dinamis

Struktur data yang dinamis, yang mana struktur data yang bisa berubah, seperti list/senarai, queue/antrian/giliran, tumpukan/stack/timbunan. Konsep dasar struktur data dinamis adalah alokasi memori yang dilakukan secara dinamis. Pada konsep ini, terdapat suatu struktur yang disebut dengan struktur referensi diri (self-referential structure), mempunyai anggota pointer yang menunjuk ke struktur yang sama dengan dirinya sendiri.

Struktur data dinamis sederhana dapat dibagi menjadi empat jenis, yaitu :

1. Linked list
2. Stack
3. Queue
4. Binary tree

Definisi ini dapat dituliskan secara sederhana dengan struktur :

```

struct node {
    int info;
    struct node *nextPtr;
}

```

III. Tugas dan Latihan

Pada contoh program 1, tambahkan 4 fungsi lagi :

- a. Fungsi update data
- b. Fungsi bersihkan data
- c. Fungsi menyisipkan data
- d. Fungsi men-swap (menukar posisi) data

MODUL II

STRUKTUR DATA STACK DAN QUEUE

I. Tujuan

1. Praktikan dapat memahami definisi dan konsep Stack dan Queue.
2. Praktikan dapat mengimplementasikan konsep Stack dan Queue.

II. Teori

1. Stack (Tumpukan)

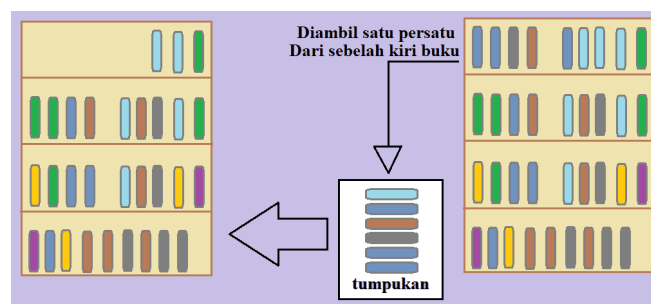
A. Pengertian

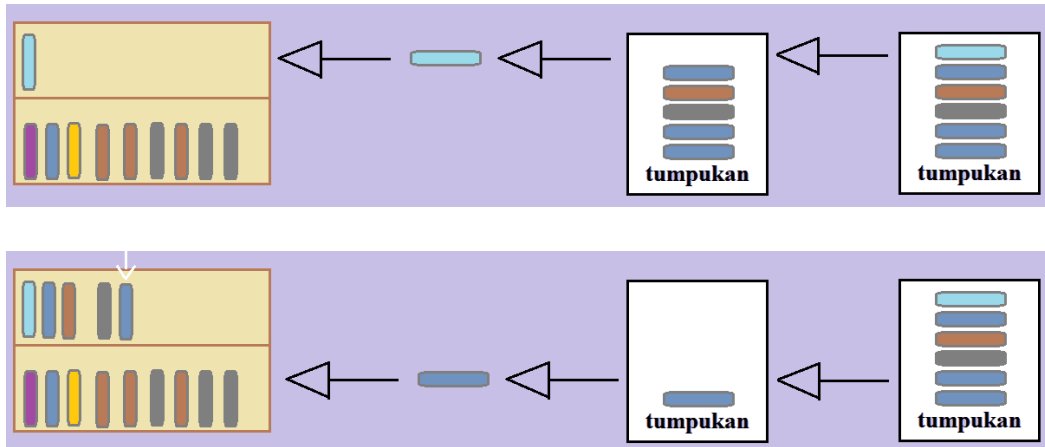
Stack adalah suatu tumpukan dari benda. Konsep Utamanya adalah LIFO (*Last In First Out*), maksudnya, benda yang terakhir masuk dalam stack akan menjadi benda yang pertama sekali yang akan dikeluarkan dari stack. Ada 2 operasi dasar yang bisa dilaksanakan pada sebuah tumpukan, yaitu operasi menyisipkan data (*push*) dan operasi menghapus data (*pop*). Selain itu stack mempunyai operasi mengosongkan (*empty*), melihat banyak tumpukan (*size*), dan melihat tumpukan paling atas (*top*). Konsep stack yang anda bisa terapkan dalam c++, dengan konsep stack beberapa peran dan fungsi yang harus dipahami ialah :

- *push* adalah operasi untuk menambahkan *stack* (data tumpukan) dalam program, kita harus bisa menambahkan satu persatu elemen dalam program untuk bisa dijadikan data *stack*, yang otomatis menambahkan data tanpa *me-replacenya* kita bisa menggunakan *array* untuk menampung datangnya.
- *pop* adalah operasi penghapusan elemen yang sudah ditambahkan dalam *stack* (tumpukan) yang kita tambahkan sebelumnya pada posisi paling atas, seperti pada implementasi peletakan buku perpustakaan.

Implementasi dari stack dapat dilakukan dengan membuat linked list, namun untuk bahasa pemrograman seperti C++ sudah ada class yang menampung data layaknya stack.

B. Implementasi konsep *stack*





C. Contoh Program

```
#include<iostream>
#include<stack>
using namespace std;

int main()
{
    stack<int> s;
    int input;
    //Hentikan input dengan ctrl+z atau tekan huruf
    while(cin>>input)
    {
        s.push(input);
    }
    do
    {
        cout<<s.top()<<" ";
        s.pop();
    } while (s.size() != 0);
    cout<<endl;
    return 0;
}
```

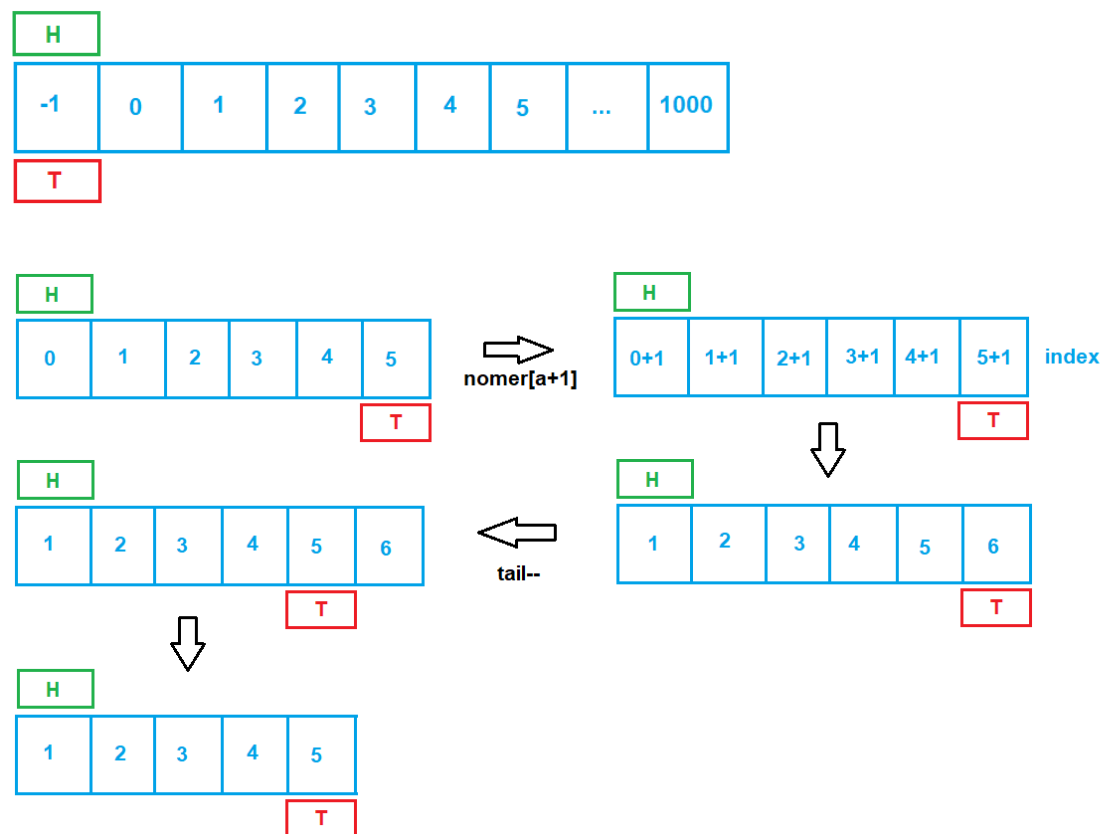
2. Queue

A. Pengertian

Queue secara harafiah diartikan sebagai antrian. *Queue* merupakan salah satu contoh aplikasi dari pembuatan double linked list yang cukup sering kita temui dalam kehidupan sehari-hari, misalnya pada saat anda mengantri di Mesin ATM. Apabila seseorang masuk dalam sebuah antrian disebut *Enqueue*. Dalam suatu antrian, ada prinsip FIFO (*First In First Out*) merupakan antrian biasa dimana yang pertama datang akan menjadi yang pertama keluar dari antrian. Antrian juga merupakan suatu kumpulan data yang penambahan elemennya hanya bisa dilakukan pada suatau ujung (disebut dengan belakang atau *rear*), dan penghapusan atau pengambilan elemen dilakukan lewat ujung yang lain (disebut dengan isi depan atau *front*). Dalam antrian juga ada istilah lain yaitu LIFO (*Last In First Out*) dimana yang duluan masuk akan terakhir keluar dari antrian.

Implementasi dari queue juga dapat dilakukan dengan membuat *linked list*, namun untuk bahasa pemrograman seperti C++ sudah ada class yang menampung data layaknya *queue*.

B. Implementasi



C. Contoh program

```
#include<iostream>
#include<queue>
using namespace std;

int main()
{
    queue<int> q;
    int input;
    //Hentikan input dengan ctrl+z atau tekan huruf
    while(cin>>input)
    {
        q.push(input);
    }
    do
    {
        cout<<q.front()<<" ";
        q.pop();
    } while (q.size() != 0);
    cout<<endl;
    return 0;
}
```

III.Tugas

1. Buatlah program yang dapat mengecek tanda kurung sejar atau tidak

Format input :

Satu baris string yang berisi huruf, angka, simbol '+', '-', '*', '/' dan tanda kurung '(', '{', dan '['

Format output :

“Ya” untuk string tersebut sejar, “Tidak” untuk tidak.

Contoh input :

(tanda kurung)

Contoh output :

Ya

Contoh input 2:

(tanda (kurung)

Contoh output 2:

Tidak

Contoh input 2:

$\{(2*5) + (4-2)\} + \{4 - 2 * [2 + 1]\}$

Contoh output 2:

Ya

2. Buatlah program yang dapat mengecek tanda kurung sejajar atau tidak

MODUL III

SENARAI BERANTAI (LINKED LIST)

I. Tujuan

1. Praktikan dapat memahami pengertian linked list, gunanya dan dapat mengimplementasikannya dalam pemrograman.
2. Praktikan dapat mengidentifikasi permasalahan-permasalahan pemrograman yang harus diselesaikan dengan menggunakan linked list, sekaligus menyelesaikannya.

II. Teori

Linked list merupakan suatu struktur data yang membentuk suatu untaian yang saling berhubungan. Tiap untaian tersebut diletakkan pada memory. Tempat yang disediakan pada suatu area memori tertentu untuk menyimpan data dikenal dengan sebutan Node/Simpul. Linked list juga disebut dengan senarai berantai merupakan suatu variabel pointer yang simpulnya bertipe Record.

1. Senarai Berantai Tunggal (Single Linked List)

Single Linked List terdiri dari elemen-elemen individu, dimana masing-masing dihubungkan dengan pointer tunggal. Masing-masing elemen terdiri dari dua bagian, yaitu sebuah data dan sebuah pointer yang disebut dengan pointer next.

data	next
20	

```
//Membuat struktur node
class Node {
    int data;
    Node* next;
}

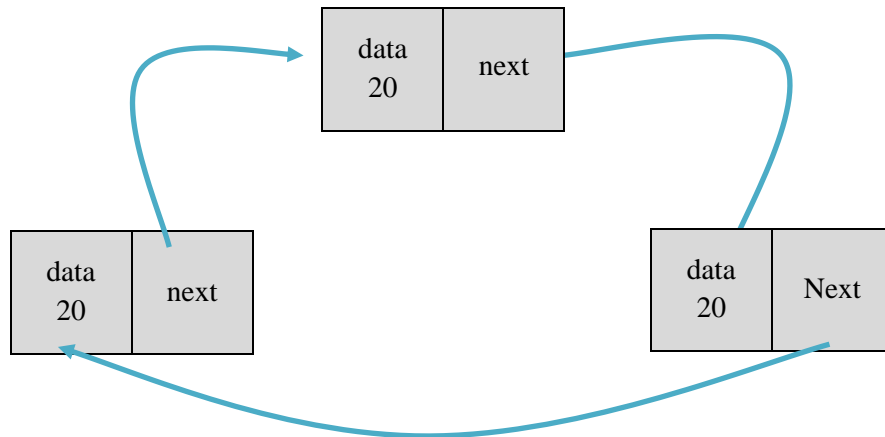
Node(int n) {
    data = n;
    next = NULL;
}

int main() {
    //buat node kepala dengan nilai 10
    Node *head = new Node(10);
}
```

Single linked list terbagi atas dua jenis yaitu Single Linked List Circular dan Single Linked List non Circular.

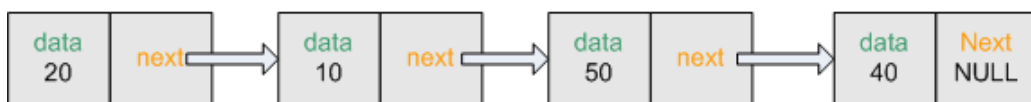
A. Single Linked List Circular

Adalah pointer pada node terakhir terakhir dalam list menunjuk kepada node awal , seakan-akan seperti sebuah lingkaran. Berikut ini adalah ilustrasi dari Single Linked List Circular .



B. Single Linked List NonCircular

Adalah pointer pada node terakhir terakhir dalam list tidak menunjuk kepada lain , sehingga bernilai NULL. Berikut ini adalah ilustrasi dari Single Linked List NonCircular



Linked list

Elemen pada awal suatu list disebut head, dan elemen terakhir dari suatu list disebut tail. Untuk mengakses elemen dalam linked list, dimulai dari head dan menggunakan pointer next dari elemen selanjutnya untuk berpindah dari elemen ke elemen berikutnya sampai elemen yang diminta dicapai.

Ada beberapa hal yang harus diketahui mengenai linked list, diantaranya adalah :

1. Linked list selalu memiliki pointer petunjuk yang selalu menunjuk pada awal dari list yang disebut Head.
2. Linked list juga selalu memiliki pointer petunjuk menunjuk pada akhir dari list yang disebut Tail, kecuali untuk jenis circular.
3. Setiap simpul yang terbentuk selalu memiliki nilai NULL, kecuali jika simpul tersebut sudah ditunjuk oleh simpul yang lainnya (Linked list belum terhubung).
4. Posisi simpul terakhir pada linked list selalu bernilai NULL karena ia tidak menunjuk pada simpul yang lainnya, kecuali bentuk circular.

Dalam pembuatan Single Linked List dapat menggunakan 2 metode, yaitu:

- **LIFO (Last In First Out), aplikasinya : Stack (Tumpukan)**

LIFO adalah suatu metode pembuatan Linked List dimana data yang masuk paling akhir adalah data yang keluar paling awal.

- **FIFO (First In First Out), aplikasinya : Queue (Antrian)**

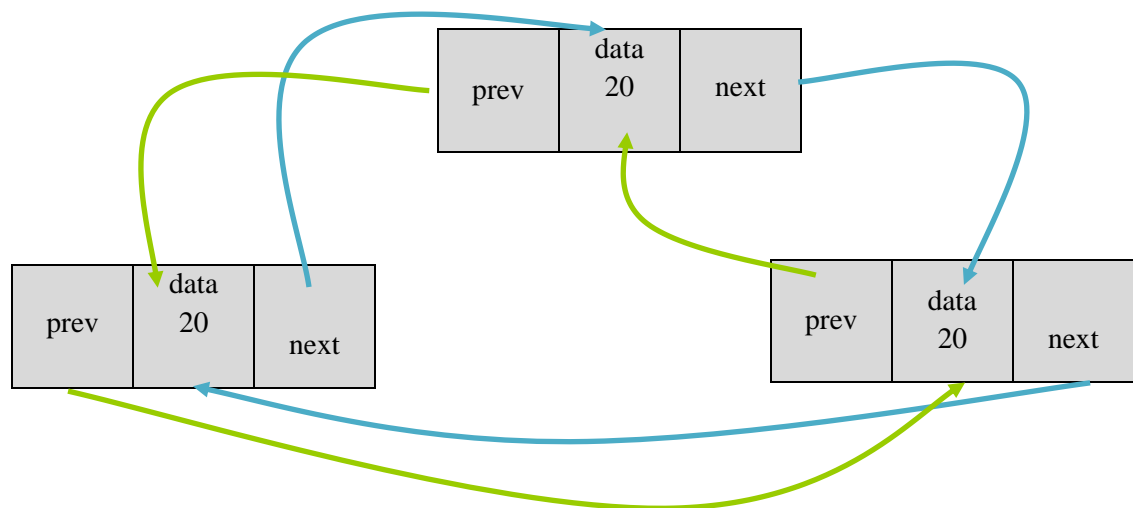
FIFO adalah suatu metode pembuatan Linked List dimana data yang masuk paling awal adalah data yang keluar paling awal juga.

2. Senarai Berantai Ganda (Double Linked List)

Secara garis besar Double linked list adalah linked list yang memiliki dua buah pointer yang menunjuk ke simpul sebelumnya (Prev) dan yang menunjuk ke simpul sesudahnya (Next). Seperti halnya diatas, double linked list juga memiliki 2 jenis yaitu Double Linked List Circular dan Double Linked List NonCircular.

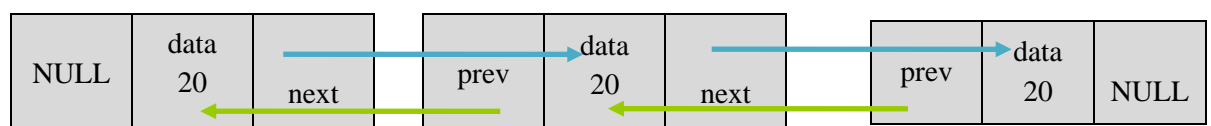
A. Double Linked List Circular

Pada double linked list circular, pointer next pada node terakhir menunjuk kepada node awal dan pointer prev pada node awal menunjuk kepada node akhir dalam rangkaian node-node tersebut.



B. Double Linked List NonCircular

Untuk menunjukkan *head* dari *double linked list circular*, maka pointer *prev* dari node pertama menunjuk NULL dan untuk menunjukkan *tail* dari *double linked list* tersebut, maka pointer *next* dari node terakhir menunjuk NULL. Gambar berikut menunjukkan gambaran Double Linked list Non Circular .



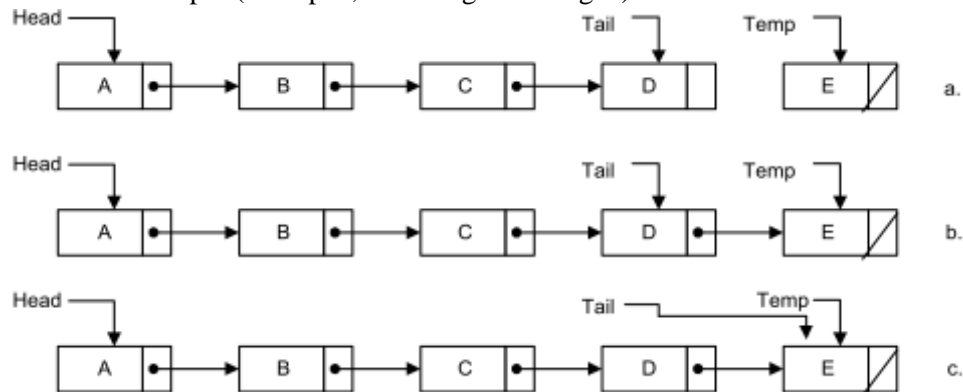
Ada beberapa hal yang harus diketahui mengenai Double linked list, diantaranya

adalah :

1. Double Linked list selalu memiliki pointer petunjuk yang selalu menunjuk pada awal dari list yang disebut Head.
2. Double Linked list juga selalu memiliki pointer petunjuk menunjuk pada akhir dari list yang disebut Tail.
3. Setiap simpul yang terbentuk selalu memiliki nilai NIL, kecuali jika simpul tersebut sudah ditunjuk oleh simpul yang lainnya (Double Linked list belum terhubung).
4. Posisi simpul terakhir pada Double linked list selalu bernilai NIL karena ia tidak menunjuk pada simpul yang lainnya, kecuali bentuk circular.

3. Operasi Linked List dan Beberapa Implementasi

- a. Menambah Simpul (di Depan, Belakang dan Tengah).



Ilustrasi Penambahan Simpul di Posisi Tail

```
//Fungsi menambah simpul di kepala single linked list
void deleteHead(Node** head)
{
    if (*head==NULL)
    {
        cout<<"Head tidak boleh NULL";
        return; .
    }
    //simpan dahulu head yang dihapus
    Node* temp = *head;
    //pindahkan head ke yang baru
    *head = temp->next;
    //hapus head sebelumnya
    delete temp;
}
}
```

- b. Menghapus Simpul (di Depan, Belakang dan Tengah).

- c. Membaca isi linked list (Membaca maju dan mundur).



Ilustrasi Pembacaan dari Head ke Tail.

```
//mencetak isi di single linked list
void print(Node* head)
{
    if (head==NULL)
    {
        cout<<"linked list kosong"<<endl;
        return;
    }
    Node* temp = head;
    while(temp!=NULL)
    {
        cout<<(temp->data)<<" ";
        temp = temp->next;
    }
    cout<<endl;
}
```

4. Latihan

```
int main()
{
    Node* head = NULL;
    string input;
    while (cin>>input)
    {
        if (input=="insert")
        {
            int data;
            cin>>data;
            insertTail(&head,data);
            print(head);
        }
        else if (input=="delete")
        {
            deleteHead(&head);
            print(head);
        }
        else if (input=="print")
        {
            print(head);
        }
    }
    return 0;
}
```

5. Tugas

1. Buatlah single linked list dengan operasi seperti stack (push, pop, dll)
2. Buatlah Sebuah program untuk menampilkan deret angka yang tampak bergeser (Membaca dari titik yang ditentukan)

Contoh :

Data : 1 2 3 4 5 Pergeseran : 3

Hasil : 4 5 1 2 3

3. Buatlah Sebuah program double linked list dengan kriteria menu sebagai berikut :

- a. Insert data dari belakang.
- b. Insert data ke indeks yang diberikan

Contoh :

List : 1 2 3 4 5 . Hapus indeks 3

Hasil : 1 2 3 5

- c. Hapus data dari indeks yang diberikan
 - d. Tampilkan data dari indeks yang diberikan.
4. Buatlah implementasi insertion sort menggunakan linked list

Contoh input :

4 9 2 10 5 8 3

Contoh output:

2 3 4 5 8 9 10

MODUL IV SORTING

I. TUJUAN

1. Praktikan dapat memahami dan menjelaskan algoritma dari Bubble Sort, Selection Sort, Insertion Sort, Shell Sort dan Quick Sort. Praktikan dapat menjelaskan konsep struktur data dinamis.
2. Praktikan dapat membuat implementasi pribadi menggunakan algoritma yang ada.

II. TEORI

Sorting adalah proses menyusun elemen-elemen dengan tata urut tertentu dan proses tersebut terimplementasi dalam bermacam aplikasi. Beberapa macam algoritma sorting telah dibuat karena proses tersebut sangat mendasar dan sering digunakan. Oleh karena itu, pemahaman atas beberapa algoritma yang ada sangatlah berguna.

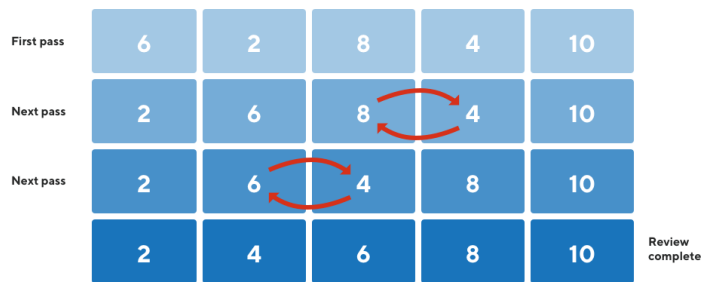
1. Bubble Sort

Bubble Sort merupakan cara pengurutan yang sederhana. Konsep dari ide dasarnya adalah seperti “gelembung air” untuk elemen struktur data yang semestinya berada pada posisi awal. Cara kerjanya adalah dengan berulang-ulang melakukan traversal (proses looping) terhadap elemen-elemen struktur data yang belum diurutkan. Di dalam traversal tersebut, nilai dari dua elemen struktur data dibandingkan. Jika ternyata urutannya tidak sesuai dengan “pesanan”, maka dilakukan pertukaran (swap). Algoritma sorting ini disebut juga dengan comparison sort dikarenakan hanya mengandalkan perbandingan nilai elemen untuk mengoperasikan elemennya.

Pseudocode Bubble Sort :

```
procedure bubbleSort( A : list of sortable items )
n = length(A)
repeat
    swapped = false
    for i = 1 to n-1 inclusive do
        if A[i-1] > A[i] then
            swap(A[i-1], A[i])
            swapped = true
        end if
    end for
    n = n - 1
until not swapped
end procedure
```

Bubble Sort



Algoritma Bubble Sort

Algoritma bubble sort dapat diringkas sebagai berikut, jika N adalah panjang elemen struktur data, dengan elemen-elemennya adalah T1, T2, T3, ..., TN-1, TN, maka:

1. Lakukan traversal untuk membandingkan dua elemen berdekatan. Traversal ini dilakukan dari belakang.
2. Jika elemen pada TN-1 > TN, maka lakukan pertukaran (swap). Jika tidak, lanjutkan ke proses traversal berikutnya sampai bertemu dengan bagian struktur data yang telah diurutkan.
3. Ulangi langkah di atas untuk struktur data yang tersisa.

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Bubble sort\n\n";

    int n = 5;
    int data[5];
    cout << "input data: \n\n";
    for (int i = 0; i < 5; i++)
    {
        cout << "Data- " << i + 1 << " : ";
        cin >> data[i];
    }
    // ---- bubble sort
    for (int i = 1; i < n; i++)
    {
        for (int j = n - 1; j >= i; j--)
        {
            if (data[j] < data[j - 1]) // ascending
                swap(data[j], data[j - 1]);
        }
    }
    //
    cout << "\n sorted data\n";
    for (int i = 0; i < n; i++)
```

```

{
    cout << data[i] << " ";
}
cout << endl;
return 0;
}

```

5. Selection Sort

Algoritma Sorting sederhana yang lain adalah Selection Sort. Ide dasarnya adalah melakukan beberapa kali pass untuk melakukan penyeleksian elemen struktur data. Untuk sorting ascending (menaik), elemen yang paling kecil di antara elemen-elemen yang belumurut, disimpan indeksinya, kemudian dilakukan pertukaran nilai elemen dengan indeks yang disimpan tersebut dengan elemen yang paling depan yang belumurut. Sebaliknya, untuk sorting descending (menurun), elemen yang paling besar yang disimpan indeksinya kemudian ditukar.

Pseudocode Selection Sort :

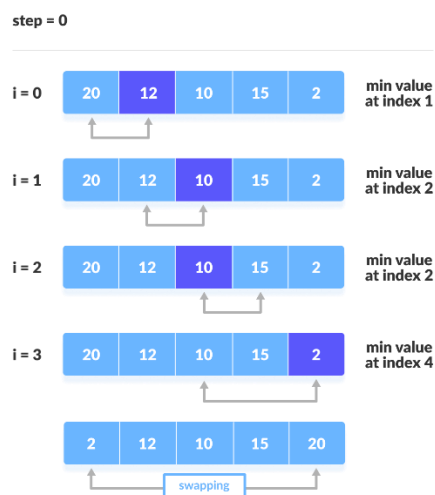
```

for i = 1 to n,
    k = i
    for j = i+1 to n,
        if a[j] < a[k],
            k = j
        → invariant: a[k] smallest of a[i..n]
        swap a[i,k]
    → invariant: a[1..i] in final position
end

```

Algoritma Selection Sort

1. Temukan nilai yang paling minimum (atau sesuai keinginan) di dalam struktur data. Jika ascending, maka yang harus ditemukan adalah nilai yang paling minimum. Jika descending, maka temukan nilai yang paling maksimum.
2. Tukar nilai tersebut dengan nilai pada posisi pertama di bagian struktur data yang belum diurutkan.
3. Ulangi langkah di atas untuk bagian struktur data yang tersisa.



```

#include <iostream>

using namespace std;
int main()
{
    cout << "Selection Sort \n\n";
    int n = 5, pos;
    int data[5];
    cout << "Input Data : \n\n";
    for (int i = 0; i < n; i++)
    {
        cout << "Data-" << i + 1 << " : ";
        cin >> data[i];
    }

    //-----Selection Sort
    for (int i = 0; i < n - 1; i++)
    {
        pos = i;
        for (int j = i + 1; j < n; j++)
        {
            if (data[j] < data[pos]) // Ascending
                pos = j;
        }
        if (pos != i)
            swap(data[pos], data[i]);
    }

    //-----
    cout << "\nSorted Data\n";
    for (int i = 0; i < n; i++)
    {
        cout << data[i] << " ";
    }
    cout << "\n";
    return 0;
}

```

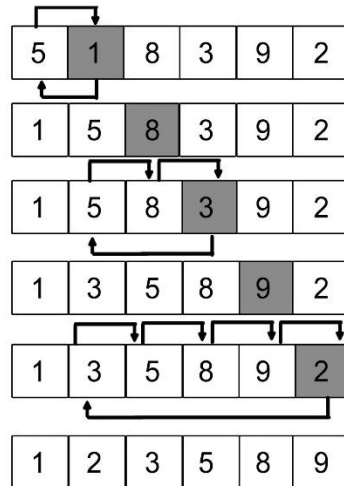
3. Insertion Sort

Insertion sort merupakan salah satu algoritma pengurutan dengan cara menyisipkan insert jika menemukan nilai yang lebih kecil, sehingga algoritma ini menyelesaikan pengurutan secara bertahap dari kiri ke kanan atau sebaliknya. Dengan syarat elemen yang akan diurutkan selanjutnya diurutkan dulu pada elemen-elemen yang telah terurut sebelumnya dengan cara langsung menyisipkan diantara elemen tersebut yang sesuai. Cara kerja insertion sort sebagaimana namanya. Pertama-tama, dilakukan iterasi, dimana di setiap iterasi insertion sort memindahkan nilai elemen, kemudian menyisipkannya berulang-ulang sampai ketempat yang tepat. Begitu seterusnya dilakukan. Dari proses

iterasi, seperti biasa, terbentuklah bagian yang telah di-sorting dan bagian yang belum di-sorting.

Pseudocode Insertion Sort :

```
for i = 2:n,  
    for (k = i; k > 1 and a[k] < a[k-1]; k--)  
        swap a[k,k-1]  
    → invariant: a[1..i] is sorted  
end
```



Algoritma Insertion Sort

Algoritma Insertion Sort dapat dirangkum sebagai berikut:

1. Simpan nilai T_i ke dalam variabel sementara, dengan $i = 1$.
2. Bandingkan nilainya dengan elemen sebelumnya.
3. Jika elemen sebelumnya (T_{i-1}) lebih besar nilainya daripada T_i , maka tindih nilai T_i dengan nilai T_{i-1} tersebut. Decrement i (kurangi nilainya dengan 1).
4. Lakukan terus poin ke-tiga, sampai $T_{i-1} \leq T_i$.
5. Jika $T_{i-1} \leq T_i$ terpenuhi, tindih nilai di T_i dengan variabel sementara yang disimpan sebelumnya.
6. Ulangi langkah dari poin 1 di atas dengan i di-increment (ditambah satu).

```
#include <iostream>  
using namespace std;  
int main()  
{  
    cout << "Insertion Sort \n\n";  
}
```

```

cout << "Input banyak data : ";
int n = 5, pos;

int data[5];
cout << "Input Data : \n\n";
for (int i = 0; i < n; i++)
{
    cout << "Data-" << i + 1 << " : ";
    cin >> data[i];
}
//-----Insertion Sort
int temp, j;
for (int i = 1; i < n; i++)
{
    temp = data[i];
    j = i - 1;
    while (data[j] > temp && j >= 0)
    { // Ascending
        data[j + 1] = data[j];
        j--;
    }
    data[j + 1] = temp;
}
//-----
cout << "\nSorted Data\n";
for (int i = 0; i < n; i++)
{
    cout << data[i] << " ";
}
cout << "\n";
return 0;
}

```

4. Quick Sort

Quick Sort adalah metode pengurutan data yang dikemukakan pertama kali oleh C.A.R Hoare pada tahun 1962. Metode ini menggunakan strategi “pecah-pecah” dengan mekanisme seperti berikut : Larik L[p..r] (dengan indeks terkecil adalah p dan indeks terbesar yaitu r) disusun ulang (dipartisi) menjadi dua buah larik A[p..q] dan A[q+1..r] sehingga setiap elemen dalam A[q+1..r]. Selanjutnya kedua larik tersebut diurutkan secara rekursif. Dengan sendirinya kombinasi kedua larik tersebut membentuk larik dengan data yang telah urut.

Pseudocode QuickSort :

```

void quicksort (int a[], int left, int right)
begin procedure
    int i, j, v;
    if (right > left)
        v = a[right]; i = left - 1; j = right;
        for(;;)

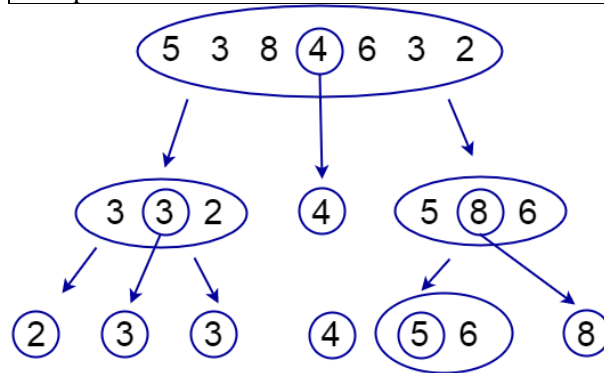
```



```

while(a[++i] < v);
while(a[--j] > v);
if(i >= j) break;
swap(a, i, j);
swap(a, i, right);
quicksort(a, left, i-1);
quicksort(a, i+1, right);
end procedure

```



Algoritma Quick Sort

Algoritma ini terdiri dari 4 langkah utama:

1. Jika struktur data terdiri dari 1 atau 0 elemen yang harus diurutkan, kembalikan struktur data itu apa adanya.
2. Ambil sebuah elemen yang akan digunakan sebagai pivot point (poin poros).
(Biasanya elemen yang paling kiri.)
3. Bagi struktur data menjadi dua bagian – satu dengan elemen-elemen yang lebih besar daripada pivot point, dan yang lainnya dengan elemen-elemen yang lebih kecil dari pada pivot point.
4. Ulangi algoritma secara rekursif terhadap kedua paruh struktur data.

```

#include <iostream>
#define n 12
using namespace std;
int A[n] = {32, 5, 13, 2, 35, 29, 17, 14, 8, 20, 11, 23};
void sort(int l, int r);
main()
{
    int i, kiri, kanan, kali, X;
    cout << "Sebelum di Sort\n";
    for (i = 0; i <= n - 1; i++)
        cout << A[i] << " ";
    cout << endl;
    sort(0, n - 1);
    cout << "\nSetelah di Sort\n";
    for (i = 0; i <= n - 1; i++)

```

```

        cout << A[i] << " ";
    cout << endl;
    return 0;
}

void sort(int kiri, int kanan)
{
    int i, ii, j, x, pivot, w, k;
    i = kiri;
    j = kanan;
    pivot = A[i];
    // pivot = A[i + (rand() % (j-i + 1))];
    while (i <= j)
    {
        while (A[i] < pivot)
            i++;
        while (pivot < A[j])
            j--;
        if (i <= j)
        {
            swap(A[i], A[j]);
            i++;
            j--;
        }
    }
    if (kiri < j)
        sort(kiri, j);
    if (i < kanan)
        sort(i, kanan);
}

```

Tugas dan Latihan

1. Buatlah proses pengurutan secara descending dan ascending pada data **10 11 23 21 0 0 8 7** dengan menggunakan :
 - a. Bubble Sort.
 - b. Selection Sort.
 - c. Insertion Sort.

MODUL V

MINIMUM SPANNING TREE

I. Tujuan

1. Praktikan dapat memahami Minimum Spanning Tree.
2. Praktikan dapat memahami Algoritma Reverse-Delete
3. Praktikan dapat menyelesaikan kasus Minimum Spanning Tree dengan menggunakan Algoritma Reverse-Delete.
4. Praktikan dapat mengimplementasikan Algoritma Reverse-Delete pada program yang kompleks.

II. Teori

Minimum Spanning Tree (MST) adalah teknik mencari jalan penghubung yang dapat menghubungkan semua titik dalam jaringan secara bersamaan sampai diperoleh jarak minimum. Masalah MST serupa dengan masalah shortest path (jalur terpendek), perbedaannya yaitu :

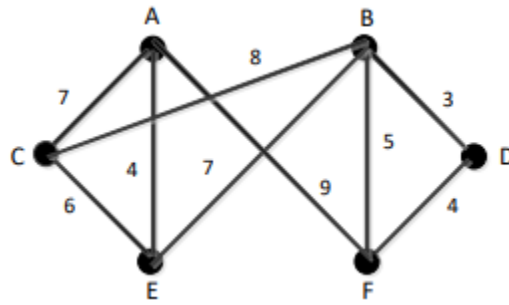
- Dalam MST, kita harus mengunjungi semua titik yang ada dengan biaya yang paling minimum untuk mencapai titik tujuan.
- Sedangkan pada shortest path, kita tidak perlu mengunjungi semua titik yang ada. Kita hanya perlu memastikan biaya untuk mencapai titik tujuan adalah biaya yang paling rendah.

Pada MST dan shortest path, setiap titik hanya boleh dikunjungi sekali. Algoritma yang paling terkenal untuk menyelesaikan kasus MST adalah Algoritma Prim dan Kruskal. Kali ini, kita akan membahas Algoritma Reverse-Delete, Algoritma Prim dan Kruskal.

1. Algoritma Reverse-Delete

Algoritma Reverse-Delete adalah algoritma dalam teori grafik yang digunakan untuk mendapatkan minimum spanning tree dari graf terhubung-berbobot tepi yang diberikan. Algoritma ini pertama kali muncul di Kruskal (1956) dan merupakan algoritma serakah karena memilih pilihan terbaik yang diberikan pada situasi apa pun. Algoritma ini adalah kebalikan dari algoritma Kruskal, yang merupakan algoritma serakah lain untuk menemukan minimum spanning tree. Algoritma Kruskal dimulai dengan grafik kosong dan menambahkan tepi sedangkan algoritma Reverse-Delete dimulai dengan grafik asli dan menghapus tepi dari grafik.

Langkah-langkah mencari minimum spanning tree dengan algoritma Reverse-Delete sebagai berikut:

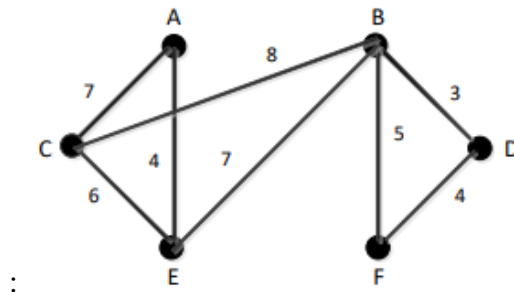


Gambar 5.1 Graf berbobot G

Penyelesaian :

1. Urutkan ruas-ruas dari graf berbobot G dengan bobot terbesar sampai yang terkecil, diperoleh :
 AF BC AC BE CE BF AE DF BD
 9 8 7 7 6 5 4 4 3
2. Lakukan penghapusan ruas berdasarkan urutan yang sudah dilakukan, dengan ketentuan bahwa penghapusan ruas tersebut tidak menyebabkan graf menjadi tidak terhubung atau membentuk sirkuit.

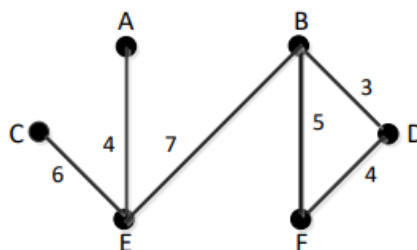
- Hapus ruas AF, karena tidak memutus graf G, diperoleh



:

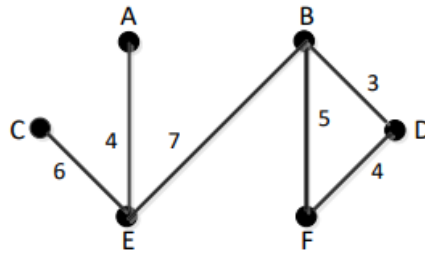
Gambar 5.2 Penghapusan ruas AF pada graf berbobot G

- Hapus ruas BC (boleh), karena tidak memutus graf G, diperoleh :



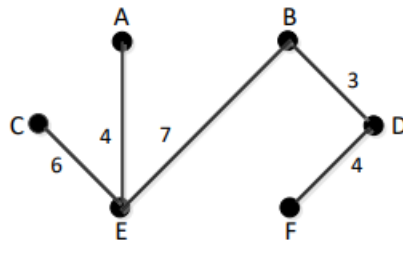
Gambar 5.3 Penghapusan ruas BC pada graf berbobot G

- Hapus ruas AC (boleh), karena tidak memutus graf G, diperoleh :



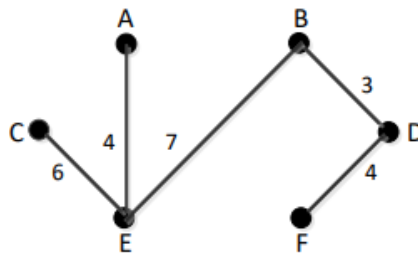
Gambar 5.4 Penghapusan ruas AC pada graf berbobot G

- Hapus ruas BE (tidak boleh), karena memutus graf G
- Hapus ruas CE (tidak boleh), karena memutus graf G
- Hapus ruas BF (boleh), karena tidak memutus graf G, diperoleh :



Gambar 5.5 Penghapusan ruas BF pada graf berbobot G

- Berikutnya ruas AE, DF dan BD tidak dapat dihapus karena memutus graf G. Jadi minimum spanning tree dari graf berbobot G adalah sebagai berikut :



Gambar 5.6 Hasil minimum spanning tree pada graf berbobot G

- Jadi jumlah bobot *minimal spanning tree* dari graf G adalah $6 + 4 + 7 + 3 + 4 = 24$

2. Program Algoritma Reverse-Delete

```
// C++ program to find Minimum Spanning Tree
// of a graph using Reverse Delete Algorithm
#include<bits/stdc++.h>
using namespace std;
```

```

// Creating shortcut for an integer pair
typedef pair<int, int> iPair;

// Graph class represents a directed graph
// using adjacency list representation
class Graph
{
int V; // No. of vertices
list<int> *adj;
vector< pair<int, iPair> > edges;
void DFS(int v, bool visited[]);

public:
Graph(int V); // Constructor

// function to add an edge to graph
void addEdge(int u, int v, int w);

// Returns true if graph is connected
bool isConnected();

void reverseDeleteMST();
};

Graph::Graph(int V)
{
this->V = V;
adj = new list<int>[V];
}

void Graph::addEdge(int u, int v, int w)
{

```

```

adj[u].push_back(v); // Add w to v's list.

adj[v].push_back(u); // Add w to v's list.
edges.push_back({w, {u, v}});
}

void Graph::DFS(int v, bool visited[])
{
// Mark the current node as visited and print it
visited[v] = true;

// Recur for all the vertices adjacent to
// this vertex
list<int>::iterator i;
for (i = adj[v].begin(); i != adj[v].end(); ++i)
if (!visited[*i])

DFS(*i, visited);
}

// Returns true if given graph is connected, else false
bool Graph::isConnected()
{
bool visited[V];
memset(visited, false, sizeof(visited));

// Find all reachable vertices from first vertex
DFS(0, visited);

// If set of reachable vertices includes all,
// return true.
for (int i=1; i<V; i++)
if (visited[i] == false)
return false;

```

```

return true;
}

// This function assumes that edge (u, v)
// exists in graph or not,
void Graph::reverseDeleteMST()
{
// Sort edges in increasing order on basis of cost
sort(edges.begin(), edges.end());

int mst_wt = 0; // Initialize weight of MST

cout << "Edges in MST\n";

// Iterate through all sorted edges in
// decreasing order of weights
for (int i=edges.size()-1; i>=0; i--)
{
int u = edges[i].second.first;
int v = edges[i].second.second;

// Remove edge from undirected graph
adj[u].remove(v);
adj[v].remove(u);

// Adding the edge back if removing it
// causes disconnection. In this case this
// edge becomes part of MST.
if (isConnected() == false)
{
adj[u].push_back(v);
adj[v].push_back(u);
}
}
}

```



```

// This edge is part of MST
cout << "(" << u << ", " << v << ") \n";
mst_wt += edges[i].first;
}
}

cout << "Total weight of MST is " << mst_wt;
}

// Driver code
int main()
{
// create the graph given in above figure
int V = 9;
Graph g(V);

// making above shown graph
g.addEdge(0, 1, 4);
g.addEdge(0, 7, 8);
g.addEdge(1, 2, 8);
g.addEdge(1, 7, 11);
g.addEdge(2, 3, 7);
g.addEdge(2, 8, 2);
g.addEdge(2, 5, 4);
g.addEdge(3, 4, 9);
g.addEdge(3, 5, 14);
g.addEdge(4, 5, 10);
g.addEdge(5, 6, 2);
g.addEdge(6, 7, 1);
g.addEdge(6, 8, 6);
g.addEdge(7, 8, 7);

```

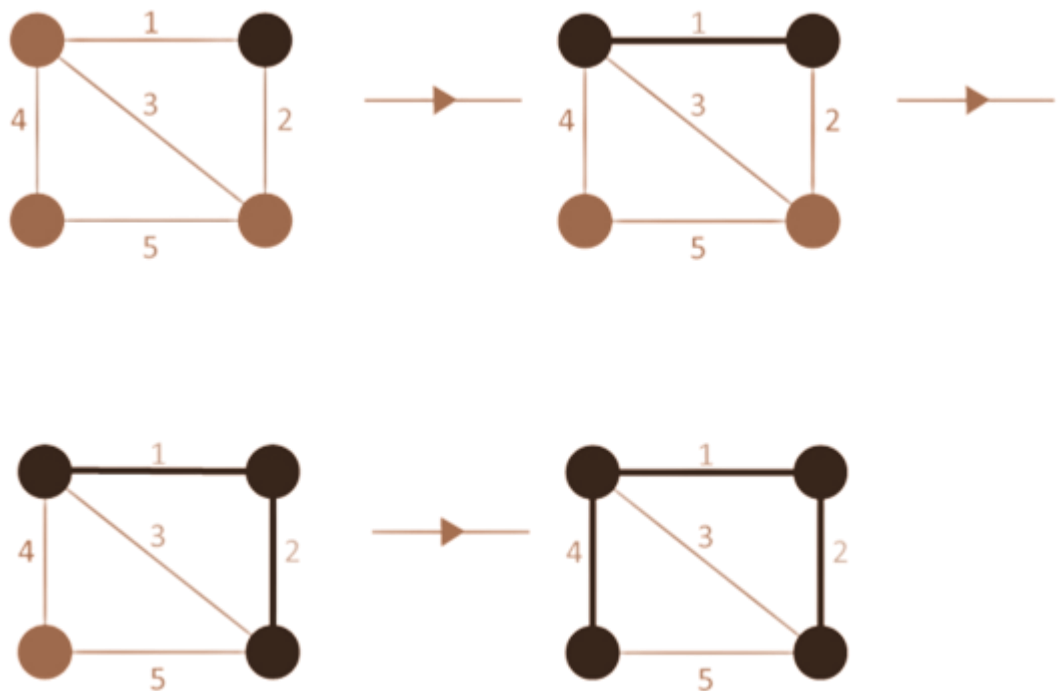
```
g.reverseDeleteMST();  
return 0;  
}
```

3. Algoritma Prim

Algoritma Prim menggunakan pendekatan serakah untuk menemukan pohon merentang minimum. Algoritma Prim menumbuhkan pohon rentang dari posisi awal. Tidak seperti tepi Kruskal, prim menambahkan simpul ke pohon merentang yang sedang tumbuh.

Langkah-langkah algoritma:

- Memegang dua set simpul terputus-putus. Satu berisi simpul yang ada di pohon merentang yang sedang tumbuh dan yang lainnya berisi simpul yang tidak ada di pohon merentang yang sedang tumbuh.
- Pilih simpul termurah yang tidak ada di pohon rentang yang sedang tumbuh dan terhubung ke pohon rentang yang sedang tumbuh dan tambahkan ke pohon rentang yang sedang tumbuh. Ini dapat dilakukan dengan menggunakan antrian prioritas. Sertakan simpul yang terkait dengan pohon rentang yang tumbuh dalam antrian prioritas.
- Periksa siklus Anda. Untuk melakukan ini, tandai node yang sudah dipilih dan tambahkan hanya node yang tidak ditandai ke antrian prioritas.



Dalam Algoritma Prim, kita akan mulai dengan simpul sembarang (tidak peduli yang mana) dan menandainya. Dalam setiap iterasi kita akan menandai sebuah simpul baru yang berdekatan dengan simpul yang telah kita tandai. Sebagai algoritma serakah, algoritma Prim akan memilih tepi termurah dan menandai titik tersebut. Jadi kita hanya akan memilih edge dengan bobot 1. Pada iterasi berikutnya kita memiliki tiga opsi, edge dengan bobot 2, 3 dan 4. Jadi, kita akan memilih edge dengan bobot 2 dan menandai titik tersebut. Sekarang lagi kita memiliki tiga pilihan, tepi dengan bobot 3, 4 dan 5. Tapi kita tidak bisa memilih tepi dengan bobot 3 karena menciptakan sebuah siklus. Jadi kami akan memilih tepi dengan bobot 4 dan kami mendapatkan pohon merentang minimum dari total biaya 7 ($= 1 + 2 + 4$).

4. Program Algoritma Prim

```
#include <iostream>
#include <vector>
#include <queue>
#include <functional>
#include <utility>

using namespace std;
const int MAX = 1e4 + 5;
typedef pair<long long, int> PII;
bool marked[MAX];
vector <PII> adj[MAX];

long long prim(int x)
{
    priority_queue<PII, vector<PII>, greater<PII> > Q;
    int y;
    long long minimumCost = 0;
    PII p;
    Q.push(make_pair(0, x));
    while(!Q.empty())
    {
        // Select the edge with minimum weight
        p = Q.top();
```

```

        Q.pop();

        x = p.second;

        // Checking for cycle
        if(marked[x] == true)
            continue;

        minimumCost += p.first;
        marked[x] = true;

        for(int i = 0; i < adj[x].size(); ++i)
        {
            y = adj[x][i].second;

            if(marked[y] == false)
                Q.push(adj[x][i]);
        }
    }

    return minimumCost;
}

int main()
{
    int nodes, edges, x, y;
    long long weight, minimumCost;

    cin >> nodes >> edges;

    for(int i = 0; i < edges; ++i)
    {
        cin >> x >> y >> weight;

        adj[x].push_back(make_pair(weight, y));
        adj[y].push_back(make_pair(weight, x));
    }

    // Selecting 1 as the starting node
    minimumCost = prim(1);

    cout << minimumCost << endl;

    return 0;
}

```

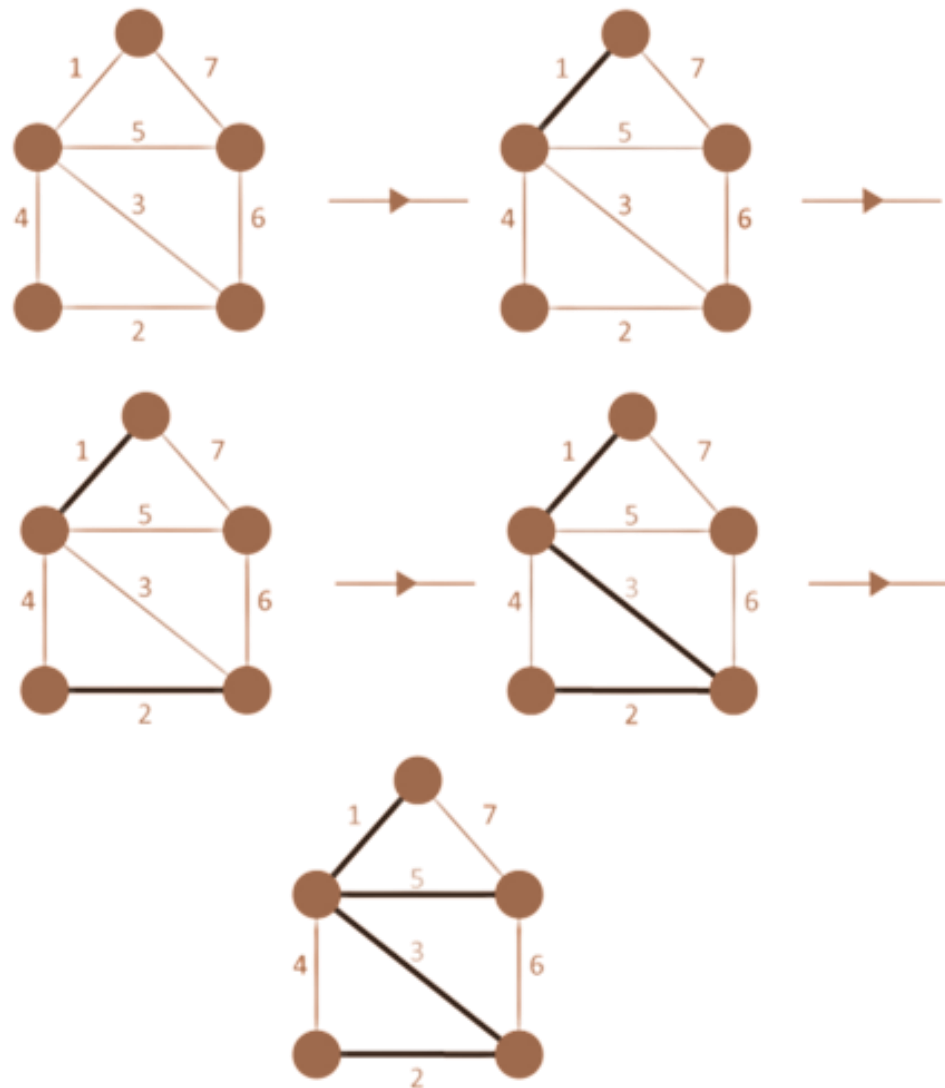
5. Algoritma Kruskal

Algoritma Kruskal membangun spanning tree dengan menambahkan satu sisi pada satu waktu ke spanning tree yang sedang tumbuh. Algoritma Kruskal mengikuti pendekatan tercepat, mencari tepi dengan bobot terendah pada setiap iterasi dan menambahkannya ke pohon rentang yang sedang tumbuh.

Langkah-langkah algoritma:

- Urutkan tepi grafik menurut bobotnya.
- Mulai tambahkan tepi ke MST dari tepi dengan bobot terendah ke tepi dengan bobot tertinggi.
- Menambahkan hanya tepi yang tidak membentuk lingkaran, hanya tepi yang menghubungkan komponen individu. Pertanyaannya adalah bagaimana memeriksa apakah simpul terhubung.

Ini dapat dilakukan dengan DFS mulai dari simpul pertama dan memeriksa apakah simpul kedua telah dikunjungi. Namun, karena DFS memiliki pengurutan di mana jumlah simpul adalah jumlah tepi, itu adalah kompleks waktu. Jadi solusi terbaik adalah "Set Terpisah". Himpunan lepas adalah himpunan yang titik potongnya adalah himpunan kosong, artinya tidak memiliki elemen yang sama.



Algoritma Kruskal memilih tepi dengan bobot terendah pada setiap iterasi. Jadi mulailah dengan tepi dengan bobot terendah. Artinya, Sisi dengan bobot 1. Kemudian pilih tepi dengan bobot terendah kedua. Sisi dengan bobot 2. Perhatikan bahwa kedua tepi ini benar-benar terlepas. Ruas berikutnya adalah rusuk dengan bobot terendah ketiga, yaitu rusuk dengan bobot 3 yang menghubungkan dua bagian graf H. yang saling lepas. Di sini tidak diperbolehkan untuk memilih tepi bobot 4 yang membuat siklus dan tidak dapat memiliki siklus. Oleh karena itu, pilih tepi dengan bobot terendah kelima. Artinya, Sisi dengan bobot 5 . Abaikan dua sisi lainnya saat mereka menghasilkan siklus. Kemudian, berakhir dengan pohon merentang minimal dengan total 11 ($= 1 + 2 + 3 + 5$).

6. Program Algoritma Kruskal

```
#include <iostream>
#include <vector>
#include <utility>
#include <algorithm>

using namespace std;
const int MAX = 1e4 + 5;
int id[MAX], nodes, edges;
pair <long long, pair<int, int> > p[MAX];

void initialize()
{
    for(int i = 0; i < MAX; ++i)
        id[i] = i;
}

int root(int x)
{
    while(id[x] != x)
    {
        id[x] = id[id[x]];
        x = id[x];
    }
    return x;
}

void union1(int x, int y)
{
    int p = root(x);
    int q = root(y);
    id[p] = id[q];
}
```

```

long long kruskal(pair<long long, pair<int, int> > p[])
{
    int x, y;
    long long cost, minimumCost = 0;
    for(int i = 0; i < edges; ++i)
    {
        // Selecting edges one by one in increasing order from the beginning
        x = p[i].second.first;
        y = p[i].second.second;
        cost = p[i].first;
        // Check if the selected edge is creating a cycle or not
        if(root(x) != root(y))
        {
            minimumCost += cost;
            union1(x, y);
        }
    }
    return minimumCost;
}

int main()
{
    int x, y;
    long long weight, cost, minimumCost;
    initialize();
    cin >> nodes >> edges;
    for(int i = 0; i < edges; ++i)
    {
        cin >> x >> y >> weight;
        p[i] = make_pair(weight, make_pair(x, y));
    }
    // Sort the edges in the ascending order
    sort(p, p + edges);
    minimumCost = kruskal(p);
}

```



```
cout << minimumCost << endl;  
  
return 0;  
  
}
```

III. Tugas

1. Buatlah implementasi MST menggunakan Algoritma Reverse-Delete, Algoritma Prim, atau Algoritma Kruskal secara manual dan program.

MODUL VI

GRAPH

I. TUJUAN

1. Praktikan memahami cara pembuatan struktur data graph
2. Praktikan dapat memahami algoritma *graph traversal Breadth First Search* dan *Depth First Search*.
3. Praktikan dapat mengimplementasikan algoritma *graph transversal Breadth First Search* dan *Depth First Search* ke dalam program.

II. TEORI & PRAKTIK

Struktur data graph merupakan struktur data yang berasal dari teori graph. Teori graph adalah sebuah bidang studi tersendiri dalam matematika yang berkaitan dengan geometri. Graph dalam matematika merupakan sebuah struktur pemetaan berpasangan antara relasi objek - objek.

Graph sangat berguna dalam memahami sebuah sistem yang sangat kompleks. Karena itu, pemrograman khusus graph berkembang pesat dalam dunia riset maupun kajian sistem sangat kompleks. Hal yang menjadi penghalang antara paradigma pemrograman jenis ini adalah matematika yang sangat kompleks, sehingga hal tersebut membuat praktisi menjauhi pemrograman graph.

Paradigma pemrograman graph hampir sama dengan paradigma pemrograman fungsional. Dapat dikatakan bahwa program graph adalah berbasis aturan, bahasa pemrograman non-deterministik untuk menyelesaikan masalah pada jenis abstraksi yang sangat tinggi, membebaskan programmer untuk tidak melakukan manajemen sumber daya komputasi dan struktur data.

Berikut adalah perbedaan paradigma pemrograman klasik dengan paradigma pemrograman graph:

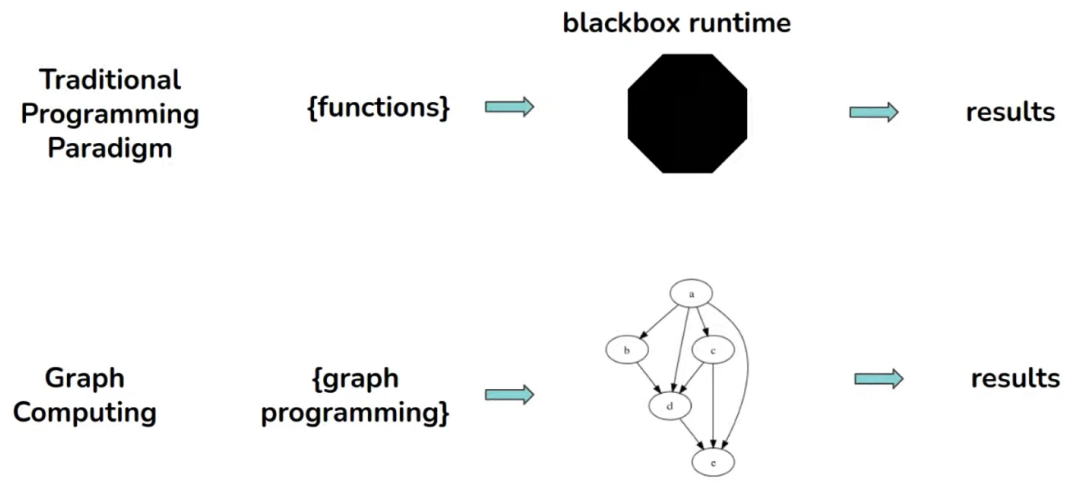


Figure 1: Difference between classical programming and graph programming

Kegunaan graph dalam menyelesaikan masalah adalah sangat penting, seperti pada kasus social network analysis, model bahasa, base knowledge, dll. Perlu diingat bahwa pemetaan graph tidak mengikuti planar 2 dimensi, graph sendiri tidak memiliki koordinat daripada kartesius dan order. Akan tetapi, secara normal manusia akan menggambarkan graph dalam bentuk dua dimensi.

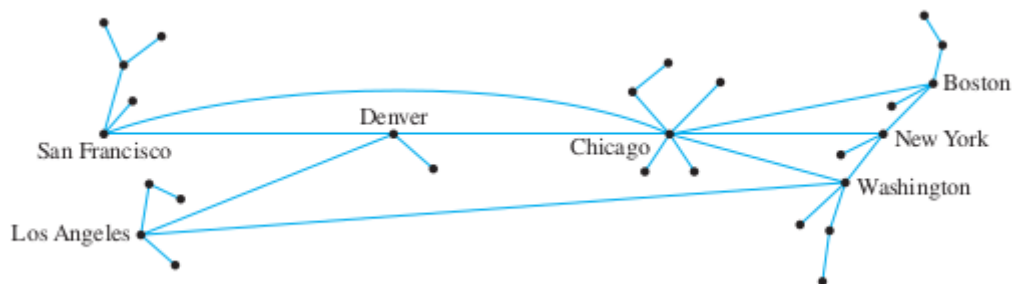
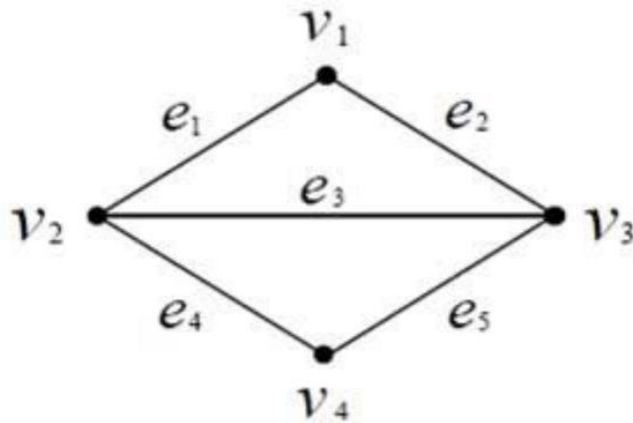


Figure 2: Graph sebagai jalan antara kota - kota

Sebuah graph G harus memiliki sebuah set V yang terbatas daripada objek - objek yang disebut sebagai vertices, dan sebuah set E yang terbatas daripada objek - objek yang disebut dengan edges, dan sebuah fungsi y yang diberikan pada setiap subset edge (v, w) . Dimana v dan w adalah vertices. Sehingga kita dapat menulis secara matematis yaitu $G = (V, E, y)$.



Untuk penulisan graph diatas, kita dapat menulisnya sebagai:

Let:

$$V = \{v1, v2, v3, v4\}$$

$$E = \{e1, e2, e3, e4, e5\}$$

y sebagai:

$$y(e1) = \{v1, v2\},$$

$$y(e2) = \{v1, v3\},$$

$$y(e3) = \{v2, v3\},$$

$$y(e4) = \{v2, v4\},$$

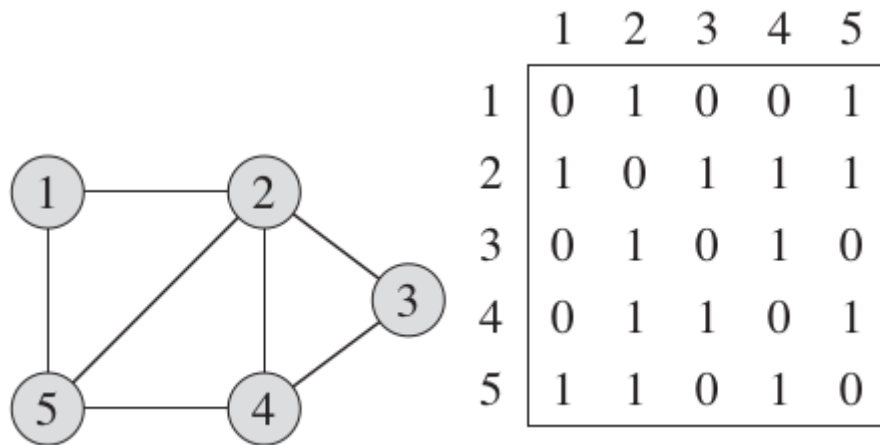
$$\text{dan } y(e5) = \{v3, v4\}$$

A. Representasi Graph

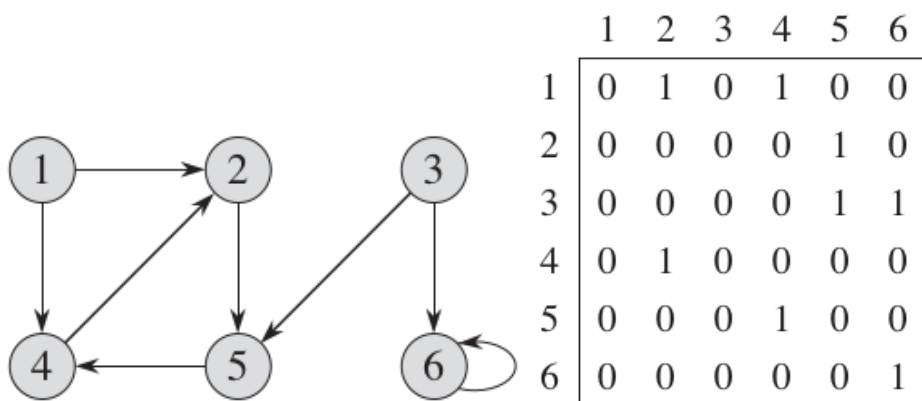
Terdapat banyak cara dalam merepresentasikan graph kedalam komputer. Akan tetapi, cara yang paling sering digunakan adalah adjacency matrix dan adjacency list. Kedua cara tersebut dapat dikatakan sebagai cara yang sederhana dengan konsep yang mudah.

1. Adjacency Matrix

Pada metode ini graph direpresentasikan dengan bentuk matriks atau larik dua dimensi, dimana baris dan kolom menggambarkan vertex dan isi daripada matriks tersebut atau nilai atribut adalah pasangan posisi dimana dua verteks terhubung.



Urutan dari beberapa vertex sehingga terdapat beberapa edge ke setiap vertex yang berurutan disebut Path. Vertex pertama disebut start vertex (vertex awal), vertex terakhir disebut end vertex (vertex akhir). Jika start vertex dan end vertex sama, maka Path disebut Cycle. Path disebut sederhana jika setiap vertex terhubung hanya sekali. Cycle disebut sederhana jika setiap vertex kecuali start vertex terhubung hanya sekali.



Metode ini jarang digunakan untuk permasalahan yang lebih kompleks, seperti gambar matriks diatas. Hal penting yang ingin kita ketahui adalah matriks dengan isi kolom yang bernilai 1, akan tetapi dikarenakan matriks tersebut kebanyakan 0. Maka hal ini membuat matriks tersebut menjadi sparse dan memakan banyak memori atau tempat penyimpanan. Metode ini hanya cocok digunakan untuk representasi matriks yang bersifat dense.

```
#include <iostream>

using namespace std;

bool** adjacencyMatrix;

int vertexCount;

void Graph() {
    adjacencyMatrix = new bool*[vertexCount];
    for (int i = 0; i < vertexCount; i++) {
        adjacencyMatrix[i] = new bool[vertexCount];
        for (int j = 0; j < vertexCount; j++)
            adjacencyMatrix[i][j] = false;
    }
}

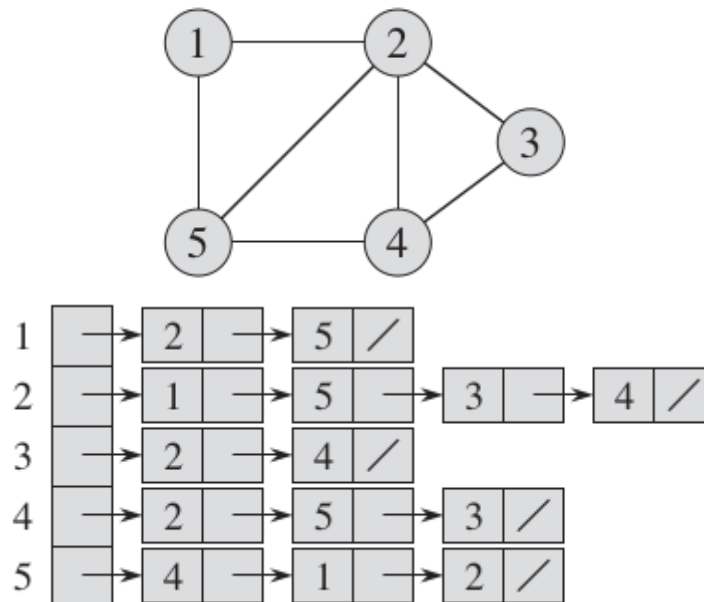
void addEdge(int i, int j) {
    if (i >= 0 && i < vertexCount && j > 0 && j < vertexCount) {
        adjacencyMatrix[i][j] = true;
        adjacencyMatrix[j][i] = true;
    }
}

void removeEdge(int i, int j) {
    if (i >= 0 && i < vertexCount && j > 0 && j < vertexCount) {
        adjacencyMatrix[i][j] = false;
        adjacencyMatrix[j][i] = false;
    }
}
```

```
int main() {
    vertexCount = 5;
    Graph();
    addEdge(0,1);
    addEdge(1,2);
    addEdge(2,3);
    addEdge(3,4);
}
```

2. Adjacency List

Pada representasi ini, vertex disimpan sebagai record atau objek, dan setiap vertex menyimpan daftar simpul yang berdekatan atau ber adjacent. Struktur data ini memungkinkan penyimpanan data tambahan pada vertex. Data tambahan ini dapat disimpan jika edge-edge juga disimpan sebagai objek, dimana dalam setiap vertex menyimpan incident edges dan tiap edge menyimpan incident vertex.



Tetapi dalam dalam hal merepresentasikan ke komputer lumayan sulit. Untuk menambahkan dan mengurangi adjacent list ataupun edge membutuhkan rata-rata $O(|E|/|V|)$ kali yang dapat menambahkan kompleksitas cubic pada setiap edge. Singkatnya adjacency list adalah solusi yang baik untuk graph yang tidak terlalu padat dan mengefisienkan pengubahan jumlah vertex daripada adjacency matrix. Walaupun masih ada solusi yang lebih baik lagi untuk menyimpan graph secara lebih dinamis.

```
#include <iostream>

using namespace std;

struct AdjListNode {
    int dest;
    struct AdjListNode* next;
};

struct AdjList {
    struct AdjListNode *head;
};

struct Graph {
    int V;
    struct AdjList* array;
};

struct AdjListNode* newAdjListNode(int dest) {
    struct AdjListNode* newNode = (struct AdjListNode*) malloc(sizeof(struct
AdjListNode));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
};

struct Graph* createGraph(int V) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->V = V;
```



```
void printGraph(struct Graph* graph) {  
    int v;  
    for (v = 0; v < graph->V; ++v) {  
        struct AdjListNode* pCrawl = graph->array[v].head;  
        printf("\n Adjacency list of vertex %d\n head ", v);  
        while (pCrawl) {  
            printf("-> %d", pCrawl->dest);  
            pCrawl = pCrawl->next;  
        }  
        printf("\n");  
    }  
}  
  
int main() {  
    struct Graph *graph = createGraph(5);  
    addEdge(graph, 1, 2);  
    addEdge(graph, 2, 3);  
    addEdge(graph, 3, 4);  
    printGraph(graph);  
}
```

B. Graph Transversal

Ada dua algoritma graf traversal yang umum digunakan, yaitu Breadth First Search dan Depth First Search. Perbedaan di antara keduanya terletak di bagaimana masing - masing algoritma nantinya mengeksplorasi tiap-tiap vertex yang ada di dalam graf.

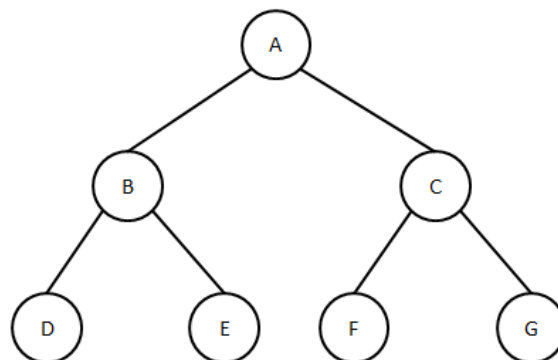
a. Breadth First Search

Pada metode BFS, semua node pada level n akan dikunjungi terlebih dahulu sebelum mengunjungi node-node pada level $n+1$. Pencarian dimulai dari node akar terus ke level ke-1 dari kiri ke kanan, kemudian berpindah ke level berikutnya demikian pula dari kiri ke kanan hingga ditemukannya solusi.

Algoritma:

- Buat sebuah antrian, inialisasi node pertama dengan root dari tree.
- Bila node pertama \neq goal, diganti dengan anak-anaknya yang diletakkan di belakang per level
- Bila node pertama = goal, maka selesai.

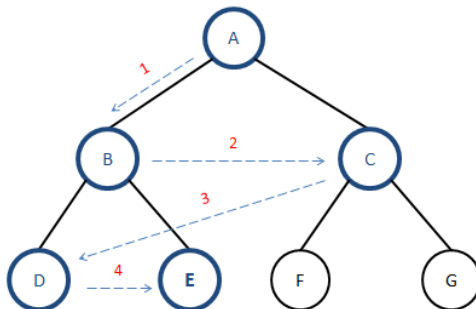
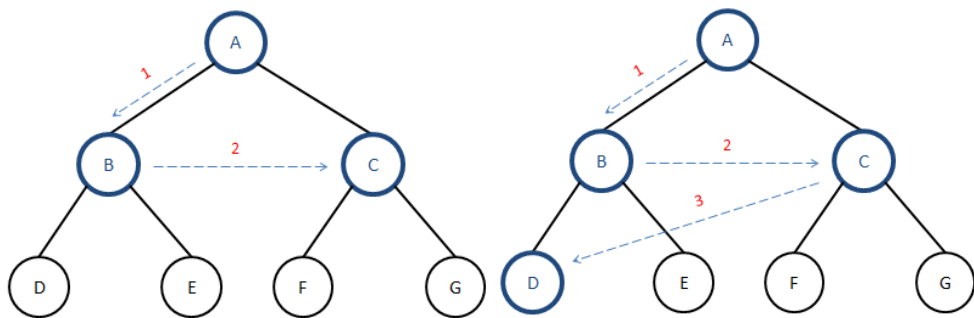
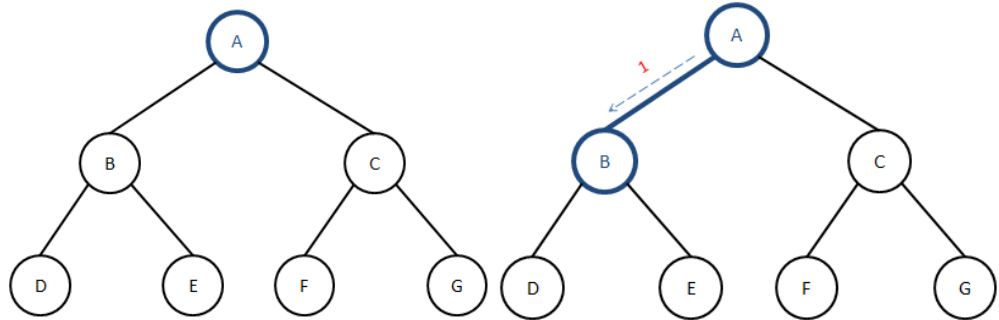
Seperti terlihat pada gambar, tujuan dari pencarian adalah menemukan titik E.



Langkah - langkah penyelesaian:

- Pencarian diawali dengan menelusuri titik pangkal yaitu titik A, karena titik A merupakan titik dengan level tertinggi,
- maka penelusuran selanjutnya adalah pada level kedua yaitu diawali dengan titik B, karena pada level ini terdapat titik lain yang selevel,
- maka penelusuran dilanjutkan dengan menjelajahi titik C, karena titik C titik terakhir di level 2,

4. maka penelusuran selanjutnya adalah pada level ketiga yaitu, diaawali dengan titik D,
5. selajutnya penelusuran dengan menjelajahi titik E, titik E merupakan titik tujuan, maka dengan demikian proses pencarian selesai pada level ketiga.



Pseudocode Breath First Search:

```
procedure BFS(G,v) is
  create a queue Q
  create a set V
  add v to V
  enqueue v onto Q
  while Q is not empty loop
    t ← Q.dequeue()
    if t is what we are looking for then
      return t
    end if
    for all edges e in G.adjacentEdges(t) loop
      u ← G.adjacentVertex(t,e)
      if u is not in V then
        add u to V
        enqueue u onto Q
      end if
    end loop
  end loop
  return none
end BFS
```

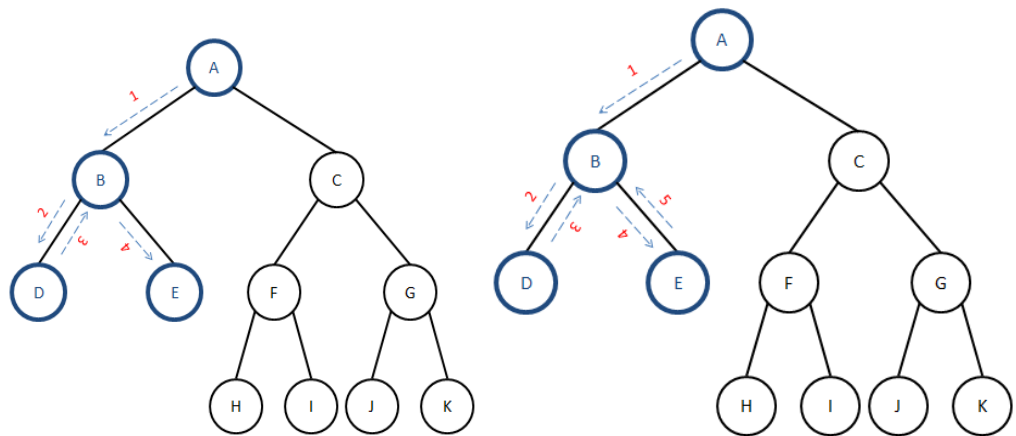
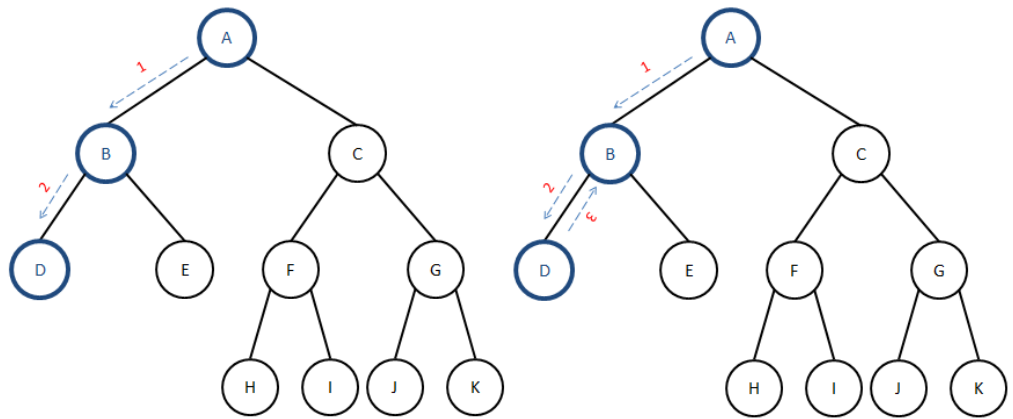
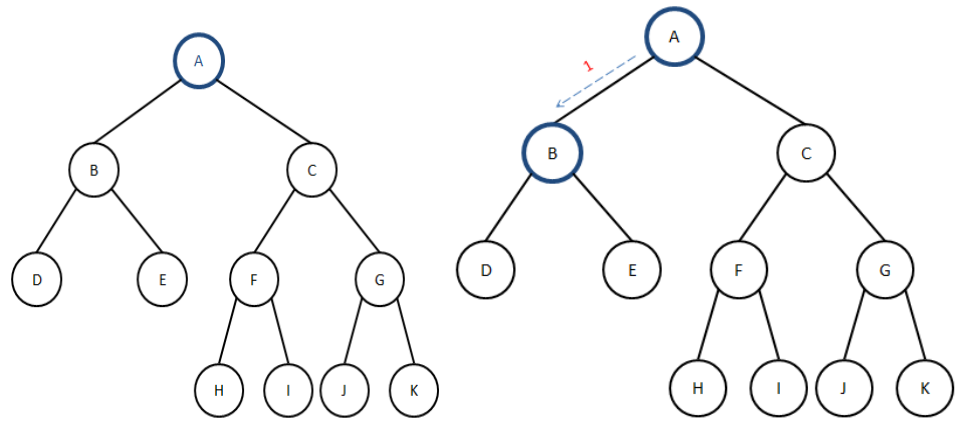
b. Depth First Search

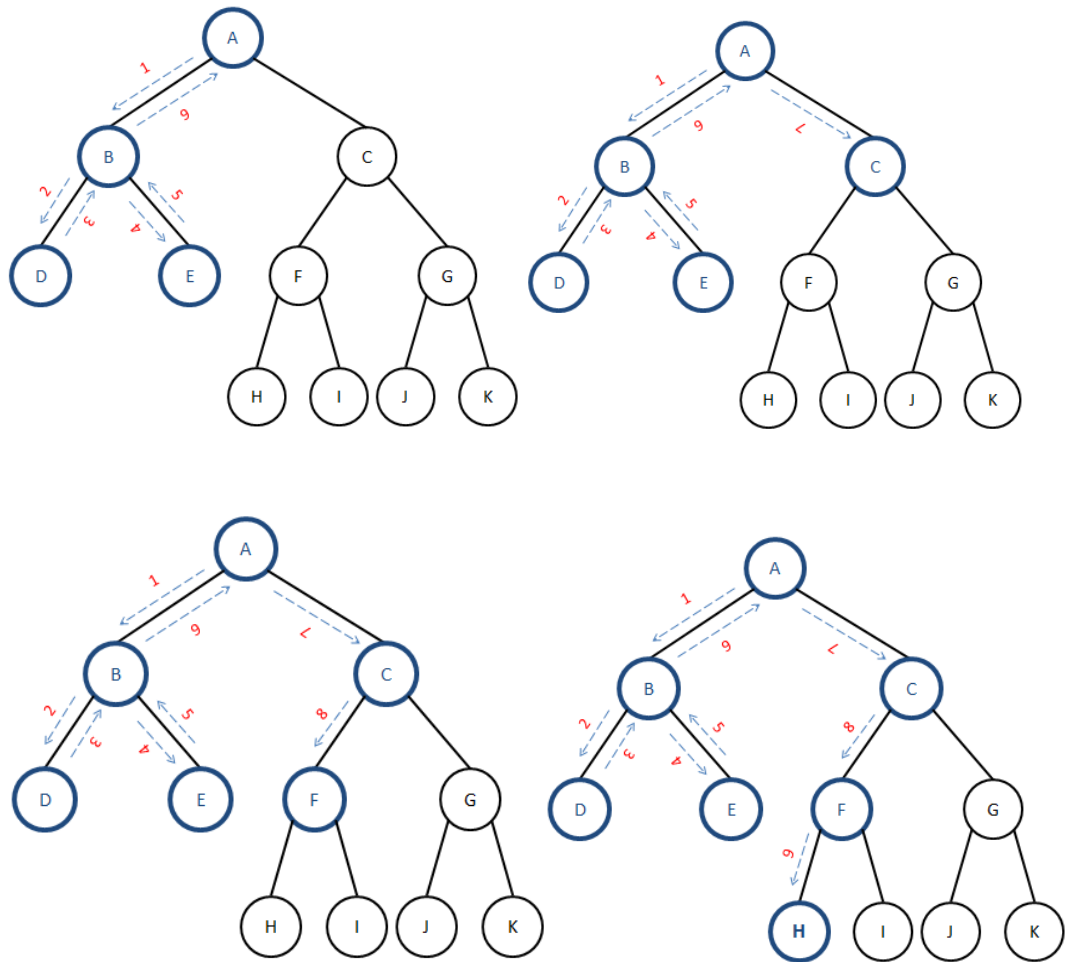
Pada DFS, proses pencarian akan dilakukan pada semua anaknya sebelum dilakukan ke node-node yang selevel. Pencarian dimulai dari node akar ke level yang lebih tinggi. Proses ini diulangi terus hingga ditemukannya solusi.

Algoritma:

- Buat sebuah antrian, inisialisasi node pertama dengan root dari tree.
- Bila node pertama \neq goal, node dihapus diganti dengan anak-anaknya dengan urutan Lchild.
- Bila node pertama = goal, maka selesai.

Pencarian dengan metode Depth First mencoba melintasi keseluruhan graf dengan urutan ABDBEBACFH. Metode ini hampir mirip dengan metode pohon. Penelusuran diawali dengan cabang yang kiri setiap node dilalui hingga mencapai node yang terakhir, jika node yang terakhir sudah dilalui akan tetapi belum menemukan titik tujuan maka penelusuran kembali melalui ke titik pangkal yaitu titik A. Pencarian dilanjutkan dengan menelusuri setiap node hingga menemukan titik tujuan. Prosedur ini diulang-ulang sehingga setiap cabang dilalui dan sampai menemukan titik tujuan.





Pseudocode Depth First Search (DFS)

procedure DFS(G, v):

 label v as discovered

 for all edges from v to w in $G.\text{adjacentEdges}(v)$ do

 if vertex w is not labeled as discovered then

 recursively call DFS(G, w)

III. TUGAS

1. Implementasikan BFS seperti yang sudah dipaparkan materi
2. Implementasikan DFS seperti yang sudah dipaparkan materi

MODUL VII

KOMPRESI DATA: ALGORITMA END TAGGED DENSE CODE (ETDC)

I. Tujuan

1. Praktikan dapat memahami algoritma End-Tagged Dense Code (ETDC)
2. Praktikan dapat menganalisis algoritma End-Tagged Dense Code (ETDC)
3. Praktikan dapat membuat implementasi pribadi menggunakan algoritma tersebut.

II. Teori

Kompresi data adalah sebuah cara untuk memadatkan data sehingga hanya memerlukan ruangan penyimpanan yang lebih kecil dan lebih efisien dalam penyimpanannya atau untuk mempersingkat waktu pertukaran data tersebut. Terdapat dua jenis kompresi data yaitu kompresi data lossless (tidak ada data yang hilang) dan kompresi data lossy (ada data yang hilang).

End-Tagged Dense Code (ETDC) merupakan salah satu algoritma kompresi data yang bekerja menggunakan bit tertinggi untuk memberi sinyal permulaan dari codeword. ETDC digunakan untuk memberi sinyal akhir codeword. Artinya adalah bit tertinggi dari byte codeword adalah 1 untuk byte terakhir dan 0 untuk byte lainnya.

b-ary End-Tagged Dense Code menentukan simbol sumber paling sering ke-I (dimulai dari $i=0$), codeword dan k digit dalam baris $2b$, dimana

$$\frac{2^{b-1}2^{(b-1)(k-1)} - 1}{2^{b-1} - 1} \leq i < 2^{b-1} \frac{2^{(b-1)k} - 1}{2^{b-1} - 1}$$

variabel k diisi dengan i-a dalam baris $2b-1$ (digit paling signifikan dan terkecil) dan kita menambahkan $2b-1$ ke digit yang paling signifikan yaitu digit terakhir. Artinya untuk $b=8$, kata pertama ($i=0$) dikodekan sebagai 10000000, yang kedua ($i=1$) menjadi 10000001 hingga ke 128 ($i=127$) dikodekan sebagai 11111111.

Word rank	Codeword assigned	Bytes	words
0	10000000	1	
....	
$2^7 - 1 = 127$	11111111	1	
$2^7 = 128$	00000000:10000000	2	$2^7 2^7$
....	

256	00000001:10000000	2	
....	
$2^7 2^7 + 2^7 - 1 = 16511$	01111111:11111111	2	
$2^7 2^7 + 2^7 = 16512$	00000000:00000000:10000000	3	$(2^7)^3$
....	
$(2^7)^3 + (2^7)^2 + 2^7 - 1$	01111111:01111111:11111111	3	

Tabel Code assignment in *End-Tagged Dense Code*.

Pembuatan codeword jika b=3

i	codeword
0	100
1	101
2	110
3	111
4	000 100
5	000 101
6	000 110
7	000 111
8	001 100
9	001 101
10	001 110
11	001 111
12	010 100
13	010 101
14	010 110
15	010 111
16	011 100
17	011 101
18	011 110

19	011 111
20	000 000 100
21	000 000 101
22	000 000 110
23	000 000 111
24	000 001 100
25	000 001 101
26	000 001 110
27	000 001 111
28	000 010 100
29	000 010 101
30	000 010 110
31	000 010 111
32	000 011 100
33	000 011 101
34	000 011 110
35	000 011 111

Pembuatan codeword jika b=4

i	codeword
0	1000
1	1001
2	1010
3	1011
4	1100
5	1101
6	1110
7	1111

8	0000 1000
9	0000 1001
10	0000 1010
11	0000 1011
12	0000 1100
13	0000 1101
14	0000 1110
15	0000 1111

Pembuatan codeword jika b=5

i	Codeword
0	10000
1	10001
2	10010
3	10011
4	10100
5	10101
6	10110
7	10111
8	11000
9	11001
10	11010
11	11011
12	11100
13	11101
14	11110
15	11111

Langkah encoding algoritma End-Tagged Dense Code. dengan string contoh “AFIFAH RIANTI” adalah sebagai berikut:

1. Membuat tabel data sebelum dikompresi dengan urutan charset berdasarkan frekuensi terbesar.

Tabel Data sebelum dikompresi

Char	Freq	ASCII Code	Bit	Bit x Freq
A	3	01000001	8	24
I	3	01001110	8	24
F	2	01001011	8	16
H	1	01001000	8	8
space	1	00100000	8	8
R	1	01001001	8	8
N	1	01010010	8	8
T	1	01010101	8	8
total				104

2. Tentukan b yang akan kalian pakai. Disini b yang dipakai adalah 3, jadi saat proses pengubahan kedalam End-Tagged Dense Code setiap karakter akan diubah menjadi codeword yang jumlah bit minimal 3 dan selanjutnya akan memiliki jumlah bit kelipatan 3(6,9,dst)
3. Ubahlah ASCII Code menjadi ETDC, dengan b=3

Tabel Data setelah dikompresi

i	Char	Freq	ETDC	Bit	Bit x Freq
0	A	3	100	3	9
1	I	3	101	3	9
2	F	2	110	3	6
3	H	1	111	3	3
4	space	1	000100	6	6
5	R	1	000101	6	6
6	N	1	000110	6	6
7	T	1	000111	6	6

	total	51
--	-------	----

4. Susun stringbit hasil kompresi dan tambahkan padding bit dan flag bit, sehingga stringbit hasil kompresi menjadi:

A	F	I	F	A	H	SP
100	110	101	110	100	111	000100
R	I	A	N	T	I	
000101	101	100	000110	000111	101	

Menambahkan padding bit dan flag bit pada string bit. Proses ini dilakukan dengan cara membagi total bita setelah encode dengan 8. Total bit hasil kompresi adalah 51. Maka 51 dibagi 8 menyisakan 5. Sehingga menghasilkan padding yakni 00000 dan flag bitnya adalah 00000101 (biner dari 5). Dan diperoleh hasil kompresinya :

100 110 101 110 100 111 000100 000101 101 100 000110 000111 101
00000 00000101

NOTE : penulisan sebenarnya tidak disertai spasi. Penulisan ini dilakukan agar praktikan lebih mudah memerhatikannya

Proses dekompresi menggunakan algoritma *End-Tagged Dense Code*

Melakukan pembacaan string bit yang didapat pada kompresi terhadap tabel data setelah dikompresi dimulai dari indeks terkecil hingga indeks terakhir dengan terus menerus menambahkan nilai pada indeks sebelumnya yang tidak mewakili karakter.

1. Indeks 0 adalah 1, pada tabel data setelah dikompresi kode 1 tidak mewakili karakter apapun, maka diikuti oleh indeks 1 yaitu 0. pada tabel data setelah dikompresi kode 10 tidak mewakili karakter apapun, maka diikuti oleh indeks 2 yaitu 0. pada tabel data setelah dikompresi kode 100 mewakili huruf A
2. Pembacaan selanjutnya dimulai pada indeks ke-3 yaitu 1, pada tabel data setelah dikompresi kode 1 tidak mewakili karakter apapun, maka diikuti oleh indeks ke-4 yaitu 1. pada tabel data setelah dikompresi kode 11 tidak mewakili karakter apapun, maka diikuti oleh indeks ke-5 yaitu 0. pada tabel data setelah dikompresi kode 110 mewakili karakter F
3. Langkah tersebut terus berlangsung hingga string bit habis.

Menganalisis algoritma *End-Tagged Dense Code*

Compression Ratio(Cr)

$$Cr = \frac{\text{ukuran bit sebelum dikompresi}}{\text{ukuran bit setelah dikompresi}}$$

$$Cr = \frac{104}{51}$$

$$Cr = 2.03$$

Space Savings

$$SS = \left(1 - \frac{\text{ukuran bit setelah dikompresi}}{\text{ukuran bit sebelum dikompresi}}\right) \times 100\%$$

$$SS = \left(1 - \frac{51}{104}\right) \times 100\%$$

$$SS = (1 - 0.490) \times 100\%$$

$$SS = 51\%$$

Bit Rate

$$\text{Bit rate} = \frac{\text{jumlah bit terkompresi}}{\text{jumlah jenis karakter}}$$

$$\text{Bit rate} = \frac{51}{8}$$

$$\text{Bit rate} = 6.375 \text{ bit/char}$$

III. Tugas

1. Kompresilah string nama lengkap kamu dengan algoritma End-Tagged Dense Code dengan nilai b nya adalah angka terakhir nim kalian. Khusus untuk angka terakhir nim kalian yang 0, maka b=3, angka terakhir nim kalian yang 1, maka b=4, angka terakhir nim kalian yang 2 , maka b=5 dan angka terakhir nim kalian yang 9, maka b=8.

MODUL VIII

STRING MATCHING

I. Tujuan

1. Praktikan dapat memahami Algoritma Knuth-Morris-Pratt (KMP).
2. Praktikan dapat mengimplementasikan algoritma KMP ke dalam program.

II. Teori

Algoritma Knuth-Morris-Pratt adalah salah satu algoritma pencarian string, dikembangkan secara terpisah oleh Donald E. Knuth pada tahun 1967 dan James H. Morris bersama Vaughan R. Pratt pada tahun 1966, namun keduanya mempublikasikannya secara bersamaan pada tahun 1977. Secara sistematis, langkah-langkah yang dilakukan algoritma Knuth-Morris-Pratt pada saat mencocokkan string:

Algoritma Knuth-Morris-Pratt mulai mencocokkan *pattern* pada awal teks.

Dari kiri ke kanan, algoritma ini akan mencocokkan karakter per karakter *pattern* dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:

Karakter di *pattern* dan di teks yang dibandingkan tidak cocok (*mismatch*).

Semua karakter di *pattern* cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.

Algoritma kemudian menggeser *pattern* berdasarkan tabel next, lalu mengulangi langkah 2 sampai *pattern* berada di ujung teks.

Contoh :

Kita akan mencari kata GCAGAGAG di dalam GCATCGCAGAGAGTATACAGTACG.

1. Membuat tabel kmpNext dari *pattern* (kata yang mau dicari).

G	C	A	G	A	G	A	G	
-1	0	0	-1	1	-1	1	-1	1

2. Melakukan pencarian.

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
G	C	A	G	A	G	A	G																

Karena huruf keempat tidak sama maka geser *pattern* sebesar $(i - \text{kmpNext}[i])$ dimana i adalah indeks pada *pattern* yang terakhir dikunjungi.

$$\text{Geser} = 3 - \text{kmpNext}[3] = 3 - -1 = 4$$

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Karena panjang *pattern* sudah melewati panjang string yang mau dicari maka proses berhenti.

Program Algoritma Knuth-Morris-Pratt

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

using namespace std;

void preKmp(string x, int m, int kmpNext[])
{
    int i,j;
    i=0;
    j=kmpNext[0]=-1;
    while(i<m)
    {
        while(j>-1&& x[i]!=x[j])
        {
            j=kmpNext[j];
        }
        i++;
        j++;
        if (x[i]==x[j])
            kmpNext[i]=kmpNext[j];
        else
            kmpNext[i]=j;
    }
}

void KMP(string x, int m, string y, int n)
```



```

int i,j,kmpNext[m];

preKmp(x,m,kmpNext);

cout<<"Tabel kmpNext : "<<endl;

for (int a=0; a<m; a++)
{
    cout<<kmpNext[a]<<" ";

}

cout<<endl<<endl;

i=j=0;

while(j<n)
{
    while (i>-1&& x[i]!=y[j])
    {
        i=kmpNext[i];
    }

    i++;
    j++;

    if (i==m)
    {
        cout<<"Ditemukan di indeks : "<<(j-i)<<endl;

        i=kmpNext[i];
    }

    else if (i<m && j==n)
    {
        cout<<endl;

        cout<<"Data Tidak Ditemukan ";

    }

}

}

int main()
{

```

```
string text ;  
  
string pattern;  
  
cout<<"Masukkan Kata/Kalimat : ";  
  
getline(cin,text);  
  
cout<<endl;  
  
cout<<"Masukkan Kata/Kalimat Yang Mau Dicari : ";  
  
getline(cin,pattern);  
  
cout<<endl;  
  
KMP(pattern,pattern.length(),text,text.length());  
  
return 0;  
  
}
```

III. Tugas

1. Buatlah sebuah string acak dan carilah sebuah kata dari string tersebut. Lakukan pencarian secara manual dan program menggunakan Algoritma Knuth-Morris-Pratt.