TRƯỜNG ĐẠI HỌC ĐÀ LẠT

KHOA CÔNG NGHỆ THÔNG TIN

BÀI LAB MÔN HỌC

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2

Đà Lạt – 03/2018

Bài thực hành **"Các cấu trúc dữ liệu cơ bản"**

Thời lượng 4 tiết

I. Mục tiêu:
II. Nội dung ôn tập:
III. Luyện tập

I. Mục tiêu

Các cấu trúc dữ liệu cơ bản và cài đặt các thuật giải tìm kiếm trong.

II. Nội dung ôn tập

- Các cấu trúc dữ liệu cơ bản (ôn tập tại lớp)
- Thuật giải tìm kiếm: Tìm kiếm tuyến tính, tìm kiếm nhị phân

1. Thuật giải tìm kiếm tuyến tính

a. Phát biểu bài toán :

Tìm x có trong dãy a?

- Input : $a_0, a_2, \ldots a_{n-1}$, x

  int a[n], x;
- Output :
  - Nếu có, trả về chỉ số i đầu tiên để a[i] = x ;
  - Nếu không có, trả về -1

b. Mô tả thuật giải:

- Bước 1: Xuất phát từ phần tử đầu tiên của dãy: **i=0**
- Bước 2: So sánh a[i] với giá trị x, có 2 trường hợp:
  - a[i] = x: tìm thấy, dừng thuật giải
  - a[i] ≠ x: sang bước 3
- Bước 3: Xét phần tử kế tiếp trong mảng: **i = i+1**
  - nếu i>n-1: hết mảng, không tìm thấy, dừng thuật giải
  - ngược lại: quay lại bước 2.

b. Cài đặt:

- TH1: Không dùng lính canh

```
int LinearSearch (int a[], int n, int x)
{
        int i = 0;
        while ((i<n) && (a[i])!=x)
                i++;
        if (i==n)
                return -1;  // tìm hết mảng nhưng không có x
        return  i;  // tìm thấy x tại vị trí x
}
```

- TH2: Dùng lính canh

- Đặt thêm phần tử có giá trị x vào cuối mảng (luôn tìm thấy x trong mảng)
- Dựa vào vị trí tìm thấy x để kết luận.

```
int LinearSearch (int a[], int n, int x)
{
        int i = 0;
        a[n] = x; // đặt phần tử lính canh
        while (a[i]!=x)
```

```
        i++;
    if (i==n)
        return -1;  // tìm hết mảng nhưng không có x
    return i;  // tìm thấy x ở vị trí i
}
```

2. Thuật giải tìm kiếm nhị phân

(Chỉ sử dụng cho các dãy đã có thứ tự)

a. Phát biểu bài toán :

Tìm x có trong dãy tăng a?

- Input : $a_0, a_2, \ldots a_{n-1}$, x

  int a[n], x;
- Output :
  - Nếu có, trả về chỉ số i để a[i] = x ;
  - Nếu không có, trả về -1

b. Ý tưởng:

Giả sử dãy đã có thứ tự tăng : $i < j \Rightarrow a_i \leq a_j$

- Nếu $x > a_k$ thì x chỉ có thể xuất hiện trong đoạn $[a_{k+1}, a_{n-1}]$
- Nếu $x < a_k$ thì x chỉ có thể xuất hiện trong đoạn $[a_0, a_{k-1}]$

c. Mô tả thuật giải:

- Bước 1: left=0; right = n-1; // tìm trên tất cả các phần tử
- Bước 2: mid = (left+right)/2; // lấy mốc so sánh

  So sánh a[mid] với giá trị x, có 3 trường hợp:
  - a[mid] = x: tìm thấy, dừng thuật giải
  - a[mid] > x: righ = mid-1 // tìm tiếp trong dãy con $a_{left} .. a_{mid-1}$
  - a[mid] < x: left = mid+1 // tìm tiếp trong dãy con $a_{mid+1} .. a_{right}$
- Bước 3:
  - nếu left≤right: lặp lại bước 2 // còn phần tử chưa xét, tìm tiếp
  - ngược lại: dừng // đã xét hết mọi phần tử

d. Cài đặt:

```
int BinarySearch (int a[], int n, int x)
{
    int left = 0, right = n-1, mid;
    do
    {
        mid = (left+right)/2;
        if (x==a[mid])
            return mid; // tìm thấy x tại vị trí mid
        else
            if (x<a[mid])
                right = mid -1;
        else
            left = mid+1;
    }
    while (left<=right);
    return -1;  // tìm hết dãy mà không có x
}
```

II. Luyện tập:

Bài 1: (Bài toán Đếm) Viết chương trình nhập vào một mảng a gồm tối thiểu 10 số nguyên từ một file. Thực hiện các thao tác sau trên mảng a:

1. Dem : Đếm số lần xuất hiện của x trong mảng a.
2. Dem_Am: Đếm các số âm
3. Dem_Duong: Đếm các số dương.
4. Dem_Nt : Đếm các số nguyên tố.
5. Đếm số lượng các đường chạy.

Đường chạy: Dãy con có thứ tự dài nhất gồm những phần tử kế tiếp.

Bài 2: (Kiểm tra tính đúng sai) Viết chương trình nhập vào một mảng a gồm tối thiểu 10 số nguyên từ một file. Thực hiện kiểm tra các phát biểu sau trên mảng a:

1. a không chứa 0.
2. a có thứ tự tăng.
3. a chứa ít nhất 3 phần tử liên tiếp trùng nhau.
4. a chỉ chứa 2 giá trị.
5. a chỉ chứa các giá trị từ 0 đến n-1.
6. Nếu a có chưa phần tử 0 thì phải chứa phần tử có giá trị 1.
7. Giả thiết a, b cùng có n phần tử . Kiểm tra a, b có phải là hoán vị của nhau.

Bài 3: (Bài toán tính Max) Viết chương trình nhập vào một mảng a gồm tối thiểu 10 số nguyên từ một file. Thực hiện các thao tác sau trên mảng a:

1. Max: Tính max(a_0,..,a_{n-1}).
2. Cs_Max: Tìm chỉ max: Trả về chỉ số đầu tiên đạt max(a_1,..,a_n).
3. Cs_Am_Max: Tìm chỉ số (đầu tiên) của số âm lớn nhất, nếu có. Nếu không, trả về 0.
4. Kc_Max: Khoảng cách lớn nhất giữa số x và các phần tử trong a.
5. Kc_Nt_Max: Trả về chỉ số của số nguyên tố có khoảng cách lớn nhất đến 1 phần tử trong a. (nếu có);

Bài thực hành: **"Cấu trúc dữ liệu Danh sách liên kết đơn: Tổ chức, các thao tác, ứng dụng"**

(Thời lượng: 4 tiết)

I. Mục tiêu
II. Ôn tập
III. Luyện tập
IV. Bài tập

I. Mục tiêu:

- Tìm hiểu cách tổ chức kiểu DSLK đơn
- Thực hiện thành thục các thao tác cơ bản trên DSLK đơn.

II. Ôn tập:

A. *Cách tổ chức kiểu dữ liệu DSLK đơn*
B. *Cài đặt các thao tác thường dùng trên DSLK đơn*
C. *Tổ chức thư viện DSLK đơn*

A. Tổ chức kiểu dữ liệu DSLK đơn:

- DSLK đơn liên kết các phần tử (là các biến động) của DS theo một chiều đi từ phần tử đầu đến phần tử cuối, trong đó phần tử trước chứa thành phần liên kết (là con trỏ) chứa địa chỉ của phần tử kế sau. DSLK đơn được quản lý bằng 2 con trỏ: một chứa địa chỉ của phần tử đầu, một chứa địa chỉ phần tử cuối.
  Ta sẽ đặt tên Kiểu DSLK đơn (khi cài đặt) là: **LIST**
- Phần tử (biến động) của DS là một cấu trúc có 2 thành phần: Một thành phần chứa dữ liệu của phần tử, thành phần còn lại là con trỏ, được dùng để chứa địa chỉ của cấu trúc cùng kiểu (Cấu trúc tự trỏ)
  Ta sẽ đặt tên Kiểu phần tử (khi cài đặt) là: **NODE**
- Thành phần dữ liệu của phần tử (nút) của DS có kiểu dữ liệu là các kiểu đơn (ký tự, nguyên, thực) hay là các kiểu có cấu trúc tự đặt . . .
  Đặt tên Kiểu của thành phần dữ liệu của phần tử trong DS là: **Data**
  (Data thay đổi tùy theo các ứng dụng)
  Data → NODE → LIST

B. Cài đặt kiểu dữ liệu DSLK đơn:

- Kiểu của thành phần dữ liệu trong DS:
  (Giả sử là kiểu int)

```
typedef  int  Data;
```

- Kiểu Phần tử (Nút) của DS:
  - Định nghĩa ban đầu

```
struct  tagNode
{
    Data        Info;
    tagNode*    pNext;
};
```

  - Đổi lại tên:

```
typedef tagNode NODE;
```

- Kiểu DSLK đơn:

```
struct  LIST
{
        NODE* pHead;//Con trỏ lưu địa chỉ phần tử đầu  DSLK
        NODE* pTail; //Con trỏ lưu địa chỉ phần tử cuối DSLK
};
```

C. *Các thao tác thường dùng trên DSLK đơn (trên các nút, trên danh sách)*

(Giả sử ta đã cài đặt kiểu DSLK đơn như trên)

1. Tạo nút mới: **GetNode (x):**
   - Chức năng: Tạo ra một phần tử (nút) của DSLK với thành phần dữ liệu là x
     (Thành phần liên kết của nút mới là con trỏ có giá trị NULL)
   - Input: x
   - Output: trả về con trỏ - lưu trử địa chỉ của pt vừa tạo (nếu tạo thành công) - có giá trị NULL (nếu ngược lại).
   - Nguyên mẫu của hàm: NODE* GetNode(Data x);

2. Khởi tạo DS rỗng: **CreatList (l)**
   - Chức năng: Tạo ra một DSLK l rỗng.
   - Input: l
   - Output: l
   - Nguyên mẫu của hàm: void CreatList(LIST &l);

3. Kiểm tra DSLK l có rỗng: **IsEmpty(l)**
   - Chức năng: Kiểm tra DSLK l có rỗng rỗng?
   - Input: l
   - Output: 1; nếu l rỗng
     0; ngược lại
   - Nguyên mẫu của hàm: int IsEmpty(LIST l);

4. Duyệt danh sách:
   - Chức năng: Duyệt danh sách từ đầu DS để xử lý dữ liệu của nút (có thể là xuất dữ liệu ra màn hình, đếm số nút, . . .)
   - Input: l
   - Nguyên mẫu của hàm: void ProcessList (LIST l);

5. Tìm nút có Info là x:
   - Chức năng: Tìm trong DS có nút chứa thành phần Info là x?
   - Input: l, x
   - Output: p (con trỏ p chứa nút đầu tiên có Info là x); nếu có
     NULL; ngược lại
   - Nguyên mẫu của hàm: NODE *Search(LIST l, Data x);

6. Chèn nút (đã có) vào đầu DSLK đơn:
   - Chức năng: Chèn một nút (đã có trước) vào đầu DS
     (Chú ý: thành phần liên kết của nút này là con trỏ có giá trị NULL)
   - Input: l, New_Ele
   - Output: l (Nút đầu là New_Ele)
   - Nguyên mẫu của hàm: void AddFirst(LIST &l, NODE* new_ele);

7. Chèn một giá trị dữ liệu vào đầu DSLK đơn:
   - Chức năng: Tạo trước nút New_Ele có info là x, con trỏ liên kết có giá trị NULL; sau đó Chèn nút này vào đầu DS.
   - Input: l, x (có kiểu data)
   - Output: - l (Nút đầu có Info là x kiểu Data)

- Con trỏ chứa địa chỉ của nút vừa tạo.

- Nguyên mẫu của hàm: NODE* InsertHead(LIST &l, Data x);

8. Chèn nút (đã có) vào Cuối DSLK đơn:
   - Chức năng: Chèn một nút (đã có trước) vào cuối DS
     (Chú ý: thành phần liên kết của nút này là con trỏ có giá trị NULL)
   - Input: l, New_Ele
   - Output: l (Nút cuối là New_Ele)
   - Nguyên mẫu của hàm: void AddTail(LIST &l, NODE *new_ele);

9. Chèn một giá trị dữ liệu vào cuối DSLK đơn:
   - Chức năng: Tạo trước nút New_Ele có info là x, con trỏ liên kết có giá trị NULL; sau đó Chèn nút này vào cuối DS.
   - Input: l, x (có kiểu data)
   - Output: - l (Nút cuối có Info là x kiểu Data)
     - Con trỏ chứa địa chỉ của nút vừa tạo.
   - Nguyên mẫu của hàm: NODE* InsertTail (LIST &l, Data x);

10. Chèn một nút (chưa có trước) vào sau nút do con trỏ q trỏ tới:
    - Chức năng: Tạo trước nút New_Ele có info là x, con trỏ liên kết có giá trị NULL; sau đó Chèn nút này vào sau nút do con trỏ q trỏ tới.
    - Input: l, q
    - Output: - l (Nút cuối có Info là x kiểu Data)
      - Con trỏ chứa địa chỉ của nút vừa tạo.
    - Nguyên mẫu của hàm: void InsertAfter(LIST &l,NODE *q, Data x);

11. Hủy nút đầu ra khỏi DSLK đơn:
    - Chức năng: Hủy nút đầu ra khỏi DSLK đơn
    - Input: l
    - Output: - l (bớt nút đầu của l input)
    - Nguyên mẫu của hàm: void RemoveHead(LIST &l);

12. Hủy nút ở vị trí sau nút do con trỏ q trỏ tới:
    - Chức năng: Hủy nút sau nút có vị trí do con trỏ q trỏ tới.
    - Input: l, q
    - Output: - l (bớt 1 nút)
    - Nguyên mẫu của hàm: void RemoveAfter (LIST &l, NODE *q);

13. Hủy nút có thành phần info là x:
    - Chức năng: Hủy nút có info là x.
    - Input: l, x
    - Output: 1; Nếu có nút
      0; Nếu ngược lại
    - Nguyên mẫu của hàm: int RemoveNode(LIST &l, Data x);

14. Hủy toàn bộ danh sách
    - Chức năng: Hủy toàn bộ danh sách
    - Input: l
    - Output: l rỗng

- Nguyên mẫu của hàm: void RemoveList(LIST &l);

15. Nhập dữ liệu từ tập tin

```
 //Tap tin chỉ chứa dữ liệu (số), chẳng hạn:
//10    1    0    9    5    8    4
//Chuyển dữ liệu tập tin f vào DSLK đơn l

void File_List(char *f, LIST &l)
{
	ifstream in(f);    //Mo de doc
	if (!in)
	{
		cout << "\nLoi mo file !";
		exit(1);
	}
	CreatList(l);
	Data x;
	in >> x;
	InsertTail(l, x);//InsertHead(l,x)
	while (!in.eof())
	{
		in >> x;
		InsertTail(l, x);
	}
	in.close();
}
```

16. Xuất dữ liệu

```
void XuatDS(LIST l)
{
	NODE  *p;
	if(IsEmpty(l))
	{	cout<<"\nDS rong!\n";
		return;
	}
	cout<<"\nDu lieu cua DS:\n";
	p = l.pHead;
	while (p!= NULL)
	{
		cout<<(p->Info)<<'\t';
		p = p->pNext;
	}
}
```

III. Luyện tập

Các bài sau dữ liệu được tổ chức bằng DSLK đơn.

Mỗi bài sau thực hiện theo yêu cầu:

Tạo một Project gồm 2 tập tin:

- Tập tin thư viện *h: Chứa các hàm chức năng của chương trình
- Tập tin chương trình *cpp: Chứa hàm main(), các hàm tổ chức menu, nhập xuất dữ liệu.
- Dữ liệu lưu trử trong một tập tin, chuyển dữ liệu vào danh sách liên kết để xử lý.

Bài 1:

Tổ chức chương trình menu thực hiện trên DSLK đơn với các chức năng:

1. Xem danh sách
2. Tìm nút đầu tiên có dữ liệu là x
3. Tìm nút cuối cùng có dữ liệu là x
4. Chèn giá trị dữ liệu x vào đầu danh sách
5. Chèn giá trị dữ liệu x vào cuối danh sách
6. Hủy nút đầu
7. Hủy nút cuối
8. Hủy các nút có dữ liệu là x
9. Hủy toàn bộ danh sách – khôi phục lại danh sách ban đầu
10. Tính số nút của danh sách
11. Tính số nút của danh sách có giá trị dữ liệu bằng x
12. Tìm giá trị nhỏ nhất của danh sách
13. Đếm số các số nguyên tố trong danh sách

14. Đếm xem giá trị lớn nhất trong danh sách xuất hiện bao nhiêu lần
15. Tính tổng các giá trị trong danh sách
16. Tính tổng các giá trị phân biệt trong danh sách
17. Tính tổng các giá trị chỉ xuất hiện 1 lần
18. Sắp danh sách tang dần theo giá trị của các nút
19. Đảo ngược danh sách l.
20. Tách Danh sách l thành 2 nửa trước sau lưu trử vào các danh sách l1, l2
21. Tách luân phiên từng nút trong Danh sách l vào 2 danh sách l1, l2.
22. . . .

Dữ liệu của các nút là số nguyên, được cho trong tập tin "Dayso.in" sau đây:

10 1 0 9 5 8 4 0 9 1

Bài 2:

Tổ chức chương trình menu thực hiện trên DSLK đơn với các chức năng:

1. Xem danh sách
2. Chèn một sinh viên vào đầu danh sách
3. Chèn một sinh viên vào cuối danh sách
4. Hủy nút đầu
5. Hủy nút cuối
6. Hủy tất cả các sinh viên có tên cho trước
7. Xuất các sinh viên có tên cho trước
8. Xuất các sinh viên có năm sinh cho trước
9. Tìm các sinh viên có năm sinh nhỏ nhất của danh sách
10. Sắp tang danh sách nhân viên theo năm sinh

Dữ liệu của các nút là các cấu trúc chứa thông tin của sinh viên, được cho trong tập tin "Sinhvien.in" sau đây:

Mã SV	Họ	Tên Lót	Ten	Năm sinh
1010123	Hoang	Ngoc	Trung	1996
1012100	Nguyen	Minh	Tuan	1995
1212121	Van	Thai	Lai	1994
1210110	Ly	Thanh	Tuan	1992
1212122	Doan	_	Trong	1994
1212131	Vo	Thi	Gai	1990
1213110	Le	Van	Vuong	1992
1213122	Do	Thi	Tinh	1995

Ghi chú:

- Về dữ liệu của họ, chữ lót, tên: nếu nhiều hơn 2 từ thì các từ kết nối nhau bằng dấu "_". Nếu chữ lót không có thì điền dấu "_"

IV. Bài tập:

Bài 1:

Giả sử một công ty quản lý nhân viên theo các thông tin sau:
- Mã nhân viên: chuỗi 8 ký tự

- Họ của nhân viên: chuỗi 8 ký tự
- Tên lót nhân viên: chuỗi 8 ký tự
- Tên nhân viên: chuỗi 8 ký tự
- Năm sinh: số nguyên
- Trình độ văn hoá: số nguyên từ 1 đến 6
  (1 = cấp 1 ; 2 = cấp 2 ; 3 = cấp 3 ;4 = đại học; 5 = cao học, 6 = Tiến sĩ)
- Lương: số nguyên ≤ 5 000 000

Viết chương trình tùy chọn thực hiện các chức năng sau:

- Tính tổng lương mà công ty trả cho nhân viên
- Tìm các nhân viên có lương cao nhất
- Liệt kê danh sách các nhân viên có năm sinh cho trước.
- Sắp danh sách tăng dần theo mức lương
- Sắp danh sách tang dần theo tên, họ, chu lot.

- Dữ liệu về nhân viên của công ty được cho trong tập tin "nhanvien.txt":
  (Gồm các thông tin: Mã nhân viên, họ, chữ lót, tên, năm sinh, trình độ, lương cơ bản)

123456	Tran	Thi	Thuy_Hoa	1980	6	4567893
123345	Nguyen	Ngoc	Minh	1985	3	2345678
113245	Hoang	Hoa	Tien	1990	2	1436784
142365	Vuong	Anh	Tuan	1966	1	1500000
154876	Vo_Tran	Ngoc	Duy	1979	5	4445670
165234	Truong	Minh	Tung	1990	2	2000000
123215	Nguyen	Minh	Nhat	1979	4	3122130
123226	Nguyen	Thi_Thuy	Tien	1980	3	3132145
114245	Hoang	_	Tuan	1966	6	4567893
160234	Tran	Minh	Tien	1980	1	2000000
123220	Nguyen	Minh	Thu	1970	6	3122130
123124	Nguyen	Thi	Thanh	1980	1	1500000

Ghi chú:

- Về dữ liệu của họ, chữ lót, tên: nếu nhiều hơn 2 từ thì các từ kết nối nhau bằng dấu "_". Nếu chữ lót không có thì điền dấu "_"
- Dữ liệu tổ chức bằng kiểu danh sách liên kết đơn

Bài 2:

Điểm số các môn học của sinh viên trong học kỳ được cho trong tập tin "Diemso.in" sau đây (gồm 4 cột):

Mã SV	Điểm M1	Điểm M2	Điểm M3
1010123	7.5	8	5
1012100	5	6.5	7
1212121	3	7.5	8.5
1012121	3.5	9.5	5
1111100	5	8	7.5
1212021	10	9	8.5
1010121	8.5	2.5	10

Số tín chỉ của mỗi môn học được cho trong bảng sau:

Môn 1 4
Môn 2 4
Môn 3 2

Điểm trung bình học kỳ của sinh viên tính theo công thức:

$$DTB\_Chung = \frac{\sum_i (So\_TC\_Mon\_i \ * Diem\_So - Mon\_i)}{\sum_i So\_TC\_Mon\_i}$$

Tính điểm trung bình học kỳ của các sinh viên, xuất ra dưới dạng:

Mã SV	Điểm M1	Điểm M2	Điểm M3	Điểm TBHK

Bài thực hành: **Lập trình với nhập xuất tập tin văn bản**

A. Mục tiêu

- Giúp sinh viên biết cách thao tác trên file văn bản
- Cung cấp sinh viên kiến thức về đọc và ghi file văn bản
- Kết quả sau khi sinh viên thực hiện bài lab: Đọc và ghi file:
  - Mảng 1 chiều
  - Ma trận
  - Mảng cấu trúc

B. Yêu cầu thực hành

- Sinh viên hiểu được các thao tác đọc và ghi file văn bản
- Thực hiện được bài thực hành về đọc và ghi file văn bản
  - Mảng 1 chiều
  - Ma trận
  - Mảng cấu trúc

C. Hướng dẫn lý thuyết

1. Khai báo thư viện sử dụng <fstream>
2. Sử dụng luồng ghi/đọc:

- Đối tượng
  - fstream //Luồng đọc và ghi
  - ofstream // Luồng ghi
  - ifstream //Luồng đọc

3. Tạo luồng
   - fstream Tên_Luong; //Nhập /Xuất.
   - ifstream Tên_Luong; //Nhập .
   - ofstream Tên_Luong; //Xuất.

4. Mở tập tin (đọc – nhập, ghi-xuất)

a. Mở để đọc:

```
fstream in; //Tạo luồng nhập, xuất
in.open("test", ios::in, 0);
```

Hoặc:

```
ifstream in; //Tạo luồng nhập
in.open("test");
```

b. Mở để ghi: Các cách mở sau là tương đương: Khi dùng ofstream

```
fstream out; //Tạo luồng nhập, xuất
out.open("test", ios::out, 0);
```

Hoặc:

```
ofstream out; //Tạo luồng xuất
out.open("test");
```

c. Mở tập tin nhập/xuất: dùng fstream

```
fstream mystream;
mystream.open("test", ios:: in | ios:: out);//Kết hợp in,out
```

d. Kết hợp tạo luồng vừa mở luôn tập tin.

Khởi tạo một đối tượng fstream ngay khi nó được khai báo.

- Đọc:

  fstream in ("test",ios::in);

  hoặc

  ifstream in("test")

- Ghi:

  fstream out ("test",ios:out);

  hoặc

  ofstream out("test");

Tóm lại, các cách viết sau là tương đương:

fstream in; in.open("test",ios::in);	fstream in ("test",ios::in);	ifstream in("test")
fstream out; out.open("test",ios::out);	fstream out ("test",ios:out);	ofstream out("test")

5. *Kết quả của việc mở tập tin tin.*

Nếu việc mở tập tin (đọc hay ghi, gắn với luồng đã mở) không thành công, luồng sẽ trả về giá trị 0

6. Kiểm tra mở tập tin có thành công hay không:

Xem khai báo:

```
fstream mystream;
mystream.open("test", ios:: in | ios:: out);
```

Cách 1:

Kiểm tra giá trị của luồng (Giá trị luồng trả về 0 nếu không mở được tập tin)

```
if ( !mystream)
{
    cout<<"\nTập tin không mở được!";
    exit(1);
}
```

Cách 2:

Dùng hàm thành phần fail(). Hàm trả về giá trị 1 nếu việc mở tập tin không thành công.

```
if (mystream.fail())
{
    cout<<  "\nTập tin không mở được!";
    exit(1);
}
```

7. Kiểm tra hết tập tin.

Dùng hàm thành phần eof() của luồng ?fstream

$$?fstreamName.eof() = \begin{cases} 1; Hết\ dữ\ liệu \\ 0; Ngược\ lại \end{cases}$$

trong đó ? là ***i, o*** hoặc ***không có ký tự nào***

Với khai báo:

```
ifstream in("test");
if(!in.eof()) //chưa hết dữ liệu
{
    //Xử lý dữ liệu
}
```

8. *Thao tác đọc dữ liệu từ tập tin, ghi dữ liệu vào tập tin.*

Với các khai báo:

```
ifstream in("test1");
ofstream out("test2");
int sonhap;
```

- Đọc:

```
in>>sonhap; // đọc dữ liệu trong test 1, lưu trử trong sonhap
```

- Ghi:

```
out<<sonhap; //ghi sonhap vao test2
```

9. *Đóng tập tin:*

Dùng hàm thành phần close()

```
mystream.close ();
```

Lưu ý: Đối với tập tin, đã có thao tác mở thì nên có thao tác đóng.

D. Hướng dẫn thực hành

Bài 1: (Chuyển dữ liệu của Tập tin vào Mảng 1 chiều)

Đọc dữ liệu của tập tin văn bản – ghi vào mảng một chiều

Giả sử tập tin văn bản \test1.txt có nội dung sau:

Tập tin:

```
9
1   2   3   4   5   6   7   8   9
```

Trong đó:

- Hàng 1: chứa số lượng các phần tử của tập tin (9)
- Hàng 2: Các giá trị của tập tin.

Viết chương trình đọc các giá trị trong tập tin \test1.txt rồi lưu trữ vào mảng 1 chiều các số nguyên.

```
#include <iostream>
#include <fstream>
#include <conio.h>
#define MAX 100
```

```
using namespace std;
void File_Array(char *filename, int Arr[MAX], int &n);
int main()
{
	int n, Arr[MAX];
	char filename[MAX];
	system("cls");
	cout<<"Nhap ten file mo de doc:";
	cin>>filename;
	File_Array(filename,Arr,n);
	cout<<endl;
	cout<<n<<endl;
	for (int i = 0; i < n; i++)
		cout<<Arr[i]<<'\t';
	system("PAUSE");
	return 0;
}
void File_Array(char *filename, int Arr[MAX], int &n)
{
	ifstream in(filename);		//Mo tập tin filename de doc
	if (!in) //Kiểm tra việc mở tập tin
	{
		cout<<"\nLoi mo file !";
		exit(1);
	}
	in>>n; //đọc dữ liệu đầu tiên của tập tin (hàng đầu), xác định kích thước mảng
	for (int i = 0; i < n; i++)
	{
		in>>Arr[i]; // lần lượt đọc dữ liệu tập tin ghi vảo mảng 1 chiều
	}
	in.close();
}
```

Bài 2: (Chuyển dữ liệu của Mảng 1 chiều vào Tập tin)

Đọc dữ liệu của mảng 1 chiều rồi ghi vào tập tin "\Test.txt" theo định dạng :

- Hàng đầu : số phần tử của tập tin (kích thước của mảng)
- Các hàng sau là các phần tử của tập tin (các phần tử của mảng)

Mảng a :

int a[9] = {1,2,3,4,5,6,7,8,9};

Tập tin :

9

1 2 3 4 5 6 7 8 9

Viết hàm thực hiện thao tác trên và có kiểm tra lại bằng chương trình.

```
#include <iostream>
#include <fstream>
#include <conio.h>
using namespace std;
void write_int(int a[], int n, char *filename);
int main()
{
	int n = 9;
	int a[] = {1,2,3,4,5,6,7,8,9};

	char filename[80];
	system("cls");
	cout<<"Nhap ten file mo de ghi:";cin>>filename;
	write_int(a,n,filename);
	system("PAUSE");
	return 0;
}
void write_int(int a[], int n, char *filename)
{
	ofstream out(filename);	//Mo de ghi
	if (!out)
	{
		cout<<"\nLoi mo file !";
		exit(1);
	}
	out<<n;
	for(int i = 0; < n;i++)
	{
		out<<a[i];
		out<<'\t';
	}
	cout<<"\nghi xong du lieu vao tep "<<filename;
```

```
        out.close();
}
```

Bài 3 : (Chuyển dữ liệu của Tập tin vào Mảng 1 chiều, định dạng tập tin chỉ chứa dữ liệu – không có số phần tử tập tin)

Giả sử tập tin văn bản \test2.txt có nội dung là các số nguyên sau:

1 2 3 4 5 6 7 8 9

Viết hàm đọc các giá trị trong tập tin \test2.txt rồi lưu trữ vào mảng 1 chiều các số nguyên (hoặc xuất ra màn hình).

```
#include <iostream>
#include <fstream>
#include <conio.h>
#define MAX 100
using namespace std;

void File_Array1(char *filename, int arr[MAX], int &n);

int main()
{
        int n, arr[MAX];
        char filename[80];
        system("cls");
        cout<<"Nhap ten file mo de doc:";cin>>filename;
        File_Array1(filename,arr,n);
        cout<<"\nN="<<n;
        cout<<endl;
        for (int i = 0; i < n; i++)
                cout<<arr[i]<<'\t';

        system("PAUSE");
        return 0;
}
void File_Array1(char *filename, int arr[MAX], int &n)
{
        ifstream in(filename);          //Mo de doc
        if (!in)
        {
                cout<<"\nLoi mo file !";
                exit(1);
```

```
        }
        n = 0;
        in>>arr[n];
        while (!in.eof())
        {
                n++;
                in>>arr[n];
        }
        n++;
        in.close();
}
```

Bài 4: (Chuyển dữ liệu của *Tập tin vào Ma trận vuông)*

Đọc dữ liệu tập tin (hàng đầu của tập tin là cấp ma trận vuông) rồi ghi vào ma trận vuông)

Giả sử tập tin văn bản \test3.txt có nội dung sau:

```
3
1       2       3
4       5       6
7       8       9
```

Trong đó:

- Hàng 1: Chỉ cỡ của ma trận vuông.
- Các hàng sau là các phần tử của tập tin

Viết hàm đọc các giá trị trong tập tin \test3.txt rồi lưu trữ vào ma trận vuông cấp n.

```
#include <iostream>
#include <fstream>
#include <conio.h>
#define MAX 100
using namespace std;

void File_Mat(char *filename, int a[MAX][MAX], int &n);

int main()
{
        int n, a[MAX][MAX],i,j;
        char filename[80];
```

```
	system("cls");
	cout<<"Nhap ten file mo de doc:";cin>>filename;
	File_Mat(filename,a,n);
	cout<<"\nN="<<n;
	cout<<endl;
	for ( i = 0; i < n; i++)
	{
		cout<<"\n";
		for (j = 0; j < n; j++)
			cout<<a[i][j]<<'\t';
	}
	system("PAUSE");
	return 0;
}
void File_Mat(char *filename, int a[MAX][MAX], int &n)
{
	ifstream in(filename);		//Mo de doc
	if (!in)
	{
		cout<<"\nLoi mo file !";
		exit(1);
	}
	in>>n;
	int i, j;
	for ( i = 0; i < n; i++)
		for (j = 0; j < n; j++)
			in>>a[i][j];
	in.close();
}
```

Bài 5: (Chuyển dữ liệu ma trận vuông vào Tập tin theo định dạng)

Duyệt ma trận vuông, ghi dữ liệu của ma trận vào tập tin theo định dạng như sau:

- Dòng 1: n //Cấp ma trận vuông
- Các dòng sau: Ghi các giá trị ma trận theo n dòng và n cột tương ứng. Các giá trị cách nhau 1 tab.

Chẳng hạn:

```
3
1	2	3
```

4 5 6

7 8 9

```
#include <iostream>
#include <fstream>
#include <conio.h>
#define MAX 100
using namespace std;

void Input_Mat(int a[MAX][MAX], int n);
void Mat_File(char *Filename, int a[MAX][MAX], int n);

int main()
{
	int n, a[MAX][MAX];
	char filename[80];
	system("cls");
	cout<<"\nNhap so phan tu ma tran:";
	cin>>n;
	Input_Mat(a, n);
	cout<<"Nhap ten file mo de luu:";cin>>filename;
	Mat_File(filename,a,n);
	System(“PAUSE”);
	return 0;
}
void Input_Mat(int a[MAX][MAX], int n)
{
	int i, j;
	for ( i = 0; i < n; i++)
	{
		for (j = 0; j < n; j++)
		{
			cout<<"a["<<i<<"]["<<j<<"]=";
			cin>>a[i][j];
		}
	}
}
void Mat_File(char *filename, int a[MAX][MAX], int n)
{
	ofstream out(filename);
```

```
	if (!out)
	{
		cout<<"\nLoi mo file !";
		exit(1);
	}
	out<<n;
	int i, j;
	for(i = 0; i < n; i++)
	{
		out<<endl;
		for(j = 0; j < n; j++)
			out<<a[i][j]<<"\t";
	}
	out.close();
	cout<<"\nLuu file thanh cong!";
}
```

Bài 6: ***(Chuyển dữ liệu của tập tin cấu trúc vào Mảng cấu trúc)***

Giả sử tập tin văn bản \tnhanvien.txt chứa những thông tin về nhân viên của một công ty. Mỗi dòng trong tập tin chứa các thông tin: Mã số, họ tên, Ngay thang năm sinh, địa chỉ, Lương.

Viết hàm đọc các thông tin trong tập tin \nhanvien.txt rồi lưu trữ vào mảng 1 chiều các cấu trúc có trường tương ứng

```
#include <iostream>
#include <fstream>
#include <conio.h>
#include <string.h>
#include<iomanip>
#define MAX 20
#define THOAT  0
using namespace std;
struct date
{
	int ngay;
	int thang;
	int nam;
};
struct nhanvien
{
```

```
	int ms;
	char hoten[MAX];
	date ntn;
	char diachi[MAX];
	double luong;
};
void xuat (nhanvien ds[MAX], int n);
int read_struct(char *filename,nhanvien ds[MAX]);
//*************************
int main()
{
	char filename[80];
	nhanvien ds[MAX];
	system("cls");
	cout<<"Nhap ten file mo de doc:";cin>>filename;
	int n = read_struct(filename,ds);
	xuat(ds,n);
	system("PAUSE");
	return 0;
}
int read_struct(char *filename,nhanvien ds[MAX])
{
	ifstream in(filename);
	if (!in)
	{
		cout<<"\nLoi mo file !";
		exit(1);
	}
	int ms;
	char hoten[MAX];
	char diachi[MAX];
	double luong;
	int ngay, thang, nam;
	int i = 0;
	in>>ms; ds[i].ms =  ms;
	in>>hoten; strcpy_s(ds[i].hoten,hoten);
	in>>ngay; ds[i].ntn.ngay = ngay;
	in>>thang; ds[i].ntn.thang = thang;
	in>>nam; ds[i].ntn.nam = nam;
```

```
        in>>diachi; strcpy_s(ds[i].diachi,diachi);
        in>>luong;ds[i].luong = luong;
        while (!in.eof())
        {
                i++;
                in>>ms; ds[i].ms =  ms;
                in>>hoten; strcpy_s(ds[i].hoten,hoten);
                in>>ngay; ds[i].ntn.ngay = ngay;
                in>>thang; ds[i].ntn.thang = thang;
                in>>nam; ds[i].ntn.nam = nam;
                in>>diachi; strcpy_s(ds[i].diachi,diachi);
                in>>luong;ds[i].luong = luong;

        }
        in.close();
        return  i;
}
void xuat (nhanvien ds[MAX], int n)
{
        cout<<setiosflags(ios:: left);
        cout<<setw(20)<<"MS"
                <<setw(20)<<"Ho Ten"
                <<setw(20)<<"NTN Sinh"
                <<setw(20)<<"Dia chi"
                <<setw(20)<<"Luong"
        cout<<endl;
        for(int i = 0; i < n; i++)
        {
                cout<<setw(20)<<ds[i].ms
                        <<setw(20)<<ds[i].hoten
                        <<setw(2)<<ds[i].ntn.ngay<<'/'
                        <<setw(2)<<ds[i].ntn.thang<<'/'
                        <<setw(16)<<ds[i].ntn.nam
                        <<setw(20)<<ds[i].diachi
                        <<setw(20)<<ds[i].luong
                 cout<<endl;
        }
}
```

Bài 7: ***(Chuyển dữ liệu của Mảng cấu trúc vào tập tin)***

Duyệt dữ liệu của mảng cấu trúc rồi ghi dữ liệu vào tập tin theo định dạng :

- Mỗi dòng gồm dữ liệu của các thành phần một cấu trúc.
- Trên mỗi dòng dữ liệu của các thành phần cấu trúc là tách biệt nhau (ít nhất 1 khoảng trắng)
- Khoảng trắng của các từ trong một chuỗi sẽ thay thế bằng dấu _.

Chẳng hạn, ta có một mảng các cấu trúc có kiểu NHANVIEN chứa các trường dữ liệu: Mã nhân viên, Họ tên, Năm sinh, địa chỉ, lương,

Ta chuyển dữ liệu mảng cấu trúc này vào tập tin định dạng như trên.

```
#include<iostream>
#include<fstream>
#include <iomanip>
using namespace std;

#define MAX 100
struct NHANVIEN
{
	char MaNV[10];
	char Hoten[20];
	int Namsinh;
	char Diachi[20];
	double LuongCB;
};

void MangCT_TTCT(char *f, NHANVIEN DS[MAX], int n);
int TTCT_MangCT(char *f, NHANVIEN TEST[MAX]);
void XuatMang(NHANVIEN TEST[MAX], int m);

void main()
{
	NHANVIEN DS[MAX] =  {
		{ "123456", "Tran_Tuan", 1960, "Da_Lat", 12500000 },
		{ "103456", "Truong_Tuan_Hoc", 1961, "Sai_Gon", 12500000 },
		{ "123450", "Nguyen_Van_Nam", 1970, "Nha_Trang", 20200000 }
					};
	int n = 3;
	cout << "\nDu lieu mang cau truc DS:\n";
	XuatMang(DS, n);
	system("PAUSE");
	//Chuyển mảng cấu trúc DS vào tập tin "Bai7test"
	MangCT_TTCT("Bai7test", DS, n);

	NHANVIEN TEST[MAX];
	//Chuyển lại dữ liệu tập tin "Bai7test" vào mảng cấu trúc TEST
```

```
	int m = TTCT_MangCT("Bai7test", TEST);
	cout << "\nDu lieu mang cau truc TEST:\n";
	XuatMang(TEST, m);
}
//Mang cau truc  -> Tap tin cau truc
void MangCT_TTCT(char *f, NHANVIEN DS[MAX], int n)
{
	ofstream out(f);
	if (!out)
	{
		cout << "\nLoi mo file !";
		exit(1);
	}
	int i;
	for (i = 0; i < n; i++)
	{
		out << setiosflags(ios::left)
			<< setw(10) << DS[i].MaNV
			<< setw(20) << DS[i].Hoten
			<< setw(10) << DS[i].Namsinh
			<< setw(20) << DS[i].Diachi
			<< setiosflags(ios::fixed) << setprecision(2)
			<< setw(10) << DS[i].LuongCB << '\n';
	}
	out.close();
}
//Tap tin -> Mang cau truc
int TTCT_MangCT(char *f, NHANVIEN TEST[MAX])
{
	ifstream in(f);
	if (!in)
	{
		cout << "\nLoi mo file !";
		exit(1);
	}
	char MaNV[10];
	char Hoten[20];
	int Namsinh;
	char Diachi[20]; //Tinh
	double LuongCB;

	int i = 0;
	in >> MaNV; strcpy_s(TEST[i].MaNV, MaNV);
	in >> Hoten; strcpy_s(TEST[i].Hoten, Hoten);
	in >> Namsinh; TEST[i].Namsinh = Namsinh;
```

```
        in >> Diachi; strcpy_s(TEST[i].Diachi, Diachi);
        in >> LuongCB; TEST[i].LuongCB = LuongCB;

        while (!in.eof())
        {
                i++;
                in >> MaNV; strcpy_s(TEST[i].MaNV, MaNV);
                in >> Hoten; strcpy_s(TEST[i].Hoten, Hoten);
                in >> Namsinh; TEST[i].Namsinh = Namsinh;
                in >> Diachi; strcpy_s(TEST[i].Diachi, Diachi);
                in >> LuongCB; TEST[i].LuongCB = LuongCB;

        }
        i++;
        in.close();
        return i;
}

void XuatMang(NHANVIEN TEST[MAX], int m)
{
        int i;
        cout << setiosflags(ios::left)
                << setw(10) << "MaNV"
                << setw(20) << "Hoten"
                << setw(10) << "NS"
                << setw(20) << "Diachi"
                << setw(10) << "LuongCB";
        cout << endl;
        for (i = 0; i < m; i++)
        {
                cout << setiosflags(ios::left)
                        << setw(10) << TEST[i].MaNV
                        << setw(20) << TEST[i].Hoten
                        << setw(10) << TEST[i].Namsinh
                        << setw(20) << TEST[i].Diachi
                        << setiosflags(ios::fixed) << setprecision(2) << setw(10)
                        << TEST[i].LuongCB << '\n';
        }
}
```

E. LUYỆN TẬP

Chạy kiểm tra lại các bài trên.

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2

LAB 4 – ĐỒ THỊ VÀ BIỂU DIỄN ĐỒ THỊ BẰNG MA TRẬN KỀ (4 TIẾT)

I. Mục tiêu

Sau khi thực hành, sinh viên cần:

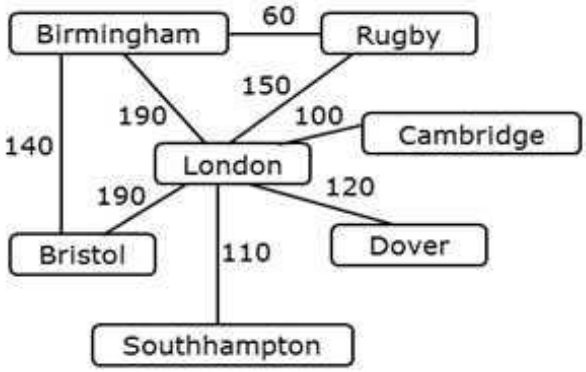
- Nắm vững định nghĩa đồ thị, các khái niệm liên quan đến đồ thị.
- Cài đặt được kiểu dữ liệu đồ thị và các thao tác, phép toán trên đồ thị được biểu diễn bởi ma trận kề.
- Vận dụng kiến thức đã học để giải một số bài toán thực tế.

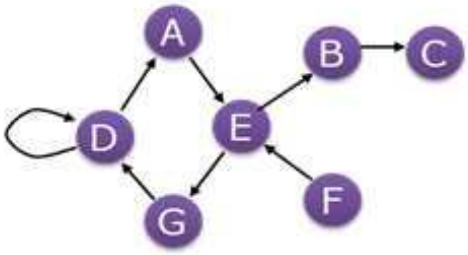
II. Yêu cầu

- Sinh viên phải hoàn thành mục IV và ***4 bài*** thuộc mục V. Mỗi bài tập tạo một project, xóa các thư mục debug của project này. Sau đó chép các project vào thư mục: Lab4_CTK40_HoTen_MSSV_Nhom#. Nén thư mục, đặt tên tập tin nén theo dạng sau: Lab4_CTK40_HoTen_MSSV_Nhom#.rar.
  Ví dụ: Lab4_CTK40_NguyenVanA_161111_Nhom4.rar.
- Sinh viên sẽ nộp bài Lab qua mạng tại phòng lab theo hướng dẫn của giáo viên.

III. Ôn tập lý thuyết

1. Đồ thị và các định nghĩa





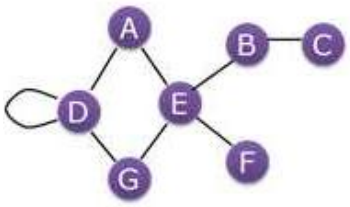
Một đồ thị G được định nghĩa là G = (V, E). Trong đó:

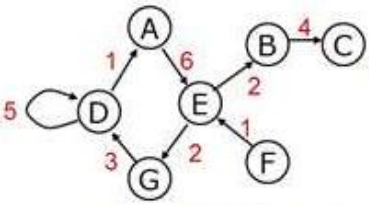
- V(G) là tập hữu hạn các đỉnh và khác rỗng. E(G) là tập các cung (cạnh) = Tập các cặp đỉnh (u, v) mà u, v ϵ V.
- Các đỉnh còn gọi là các nút (node) hay điểm (point)
- Mỗi cung nối giữa hai đỉnh v, w có thể ký hiệu là cặp (v,w). Hai đỉnh có thể trùng nhau. Nếu cặp (v,w) có thứ tự thì gọi là cung có thứ tự hay cạnh có hướng. Ngược lại ta nói là cung không có thứ tự hay cạnh vô hướng.
- Đỉnh kề: Hai đỉnh được gọi là kề nhau nếu chúng được nối với nhau bởi một cạnh. Trong trường hợp này, hai nút đỉnh được gọi là hàng xóm (neighbors) của nhau.
- Đường đi là một dãy tuần tự các đỉnh v_1, v_2, …, v_n sao cho (v_i, v_{i+1}) là một cung trên đồ thị. Đỉnh v_1 được gọi là đỉnh đầu, đỉnh v_n được gọi là đỉnh cuối. Độ dài đường đi là số cạnh trên đường đi.
- Đỉnh X gọi là có thể đi đến được từ đỉnh Y nếu tồn tại một đường đi từ đỉnh Y đến đỉnh X.
- Đường đi đơn là đường đi mà mọi đỉnh trên đó đều khác nhau, ngoại trừ đỉnh đầu và cuối có thể trùng nhau.

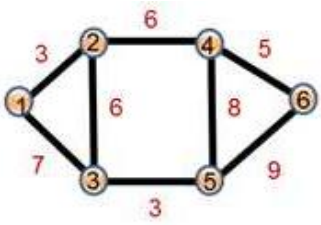
- Chu trình là đường đi có đỉnh đầu trùng với đỉnh cuối. Chu trình đơn là một đường đi đơn có đỉnh đầu và đỉnh cuối trùng nhau và có độ dài ít nhất là 1.
- Cấp của đồ thị là số đỉnh của đồ thị. Kích thước của đồ thị là số cạnh của đồ thị. Đồ thị rỗng là đồ thị có kích thước bằng 0
- Bậc của một đỉnh là số cạnh nối với đỉnh đó
- Trong nhiều ứng dụng, ta thường kết hợp các giá trị (value) hoặc nhãn (label) cho các cạnh hoặc đỉnh. Khi đó, ta gọi là đồ thị có trọng số.
- Nhãn có thể có kiểu tùy ý và dùng để biểu diễn tên, chi phí, khoảng cách, …
- Đồ thị có hướng là đồ thị mà các cạnh của nó là có hướng. Nghĩa là các cặp đỉnh (v,w) có phân biệt thứ tự.
- Đồ thị vô hướng là đồ thị mà các cặp đỉnh tương ứng với các cạnh của nó không phân biệt thứ tự.

2. *Biểu diễn đồ thị*

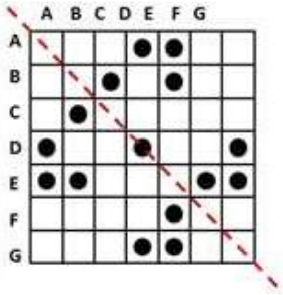
Biểu diễn đồ thị bằng ma trận kề





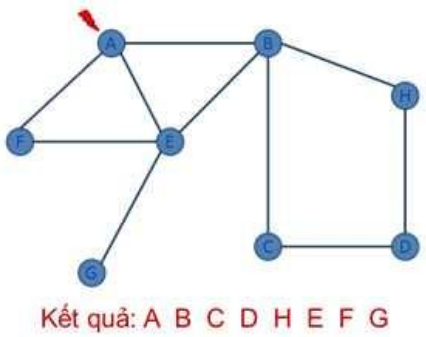


	A	B	C	D	E	F	G
A				1	1		
B			1		1		
C		1					
D	1			1			1
E	1	1				1	1
F					1		
G				1	1		

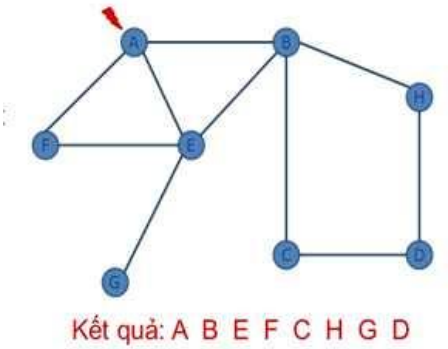


	A	B	C	D	E	F	G
A	0	0	0	0	6	0	0
B	0	0	4	0	0	0	0
C	0	0	0	0	0	0	0
D	1	0	0	5	0	0	0
E	0	2	0	0	0	0	2
F	0	0	0	0	1	0	0
G	0	0	0	3	0	0	0

	1	2	3	4	5	6
1	0	3	7	0	0	0
2	3	0	6	6	0	0
3	7	6	0	0	3	0
4	0	6	0	0	8	5
5	0	0	3	8	0	9
6	0	0	0	5	9	0



Kết quả: A B C D H E F G



Kết quả: A B E F C H G D

3. *Các phương pháp duyệt đồ thị*

Duyệt đồ thị là một thủ tục có hệ thống để khám phá đồ thị bằng cách kiểm tra tất cả các đỉnh và các cạnh của nó.

Có hai cách duyệt đồ thị phổ biến

- Duyệt đồ thị theo chiều sâu (Depth First Search - DFS)
- Duyệt đồ thị theo chiều rộng (Breadth First Search - BFS)

Cách 1: Duyệt đồ thị theo chiều sâu

- B1. Khởi gán tất cả các đỉnh đều chưa được duyệt
- B2. Xuất phát từ một đỉnh **v** bất kỳ của đồ thị, đánh dấu đỉnh v đã được duyệt
- B3. Xử lý đỉnh v
- B4. Với mỗi đỉnh w chưa được duyệt và kề với v, ta thực hiện đệ quy quá trình trên cho w.

Cách 2: Duyệt đồ thị theo chiều rộng

- B1. Khởi gán tất cả các đỉnh đều chưa được duyệt

- B2. Xuất phát từ một đỉnh **v** bất kỳ của đồ thị, đánh dấu đỉnh v đã được duyệt. Đưa v vào hàng đợi.
- B3. Lấy x ra khỏi hàng đợi. Xử lý đỉnh x.
- B4. Với mỗi đỉnh w chưa được duyệt và kề với x, ta đánh dấu w đã được duyệt. Đưa w vào hàng đợi.
- B5. Quay lại B3.

IV. Hướng dẫn thực hành

1. Tạo dự án

Sinh viên có thể chọn cài đặt kiểu dữ liệu đồ thị theo ngôn ngữ C++.

- Tạo dự án mới, đặt tên là Lab4_GraphAsMatrix
- Trong thư mục Header Files, tạo ra 4 files sau:
  - menu.h: Định nghĩa các thao tác thực thi của chương trình
  - common.h: Định nghĩa các hằng số và kiểu dữ liệu đồ thị
  - graph.h: Định nghĩa các thao tác trên đồ thị
- Trong thư mục Source Files, tạo tập tin: program.cpp

2. Định nghĩa kiểu dữ liệu đồ thị (Biểu diễn bằng ma trận kề)

- Nhấp đôi chuột vào file common.h, định nghĩa các hằng số và kiểu dữ liệu như sau:

```
#ifndef _GRAPH_
#define _GRAPH_

// Định nghĩa hằng số
#define  UPPER          100      // Số ptử tối đa
#define  ZERO           0        // Giá trị 0
#define  MAX            20       // Số đỉnh tối đa
#define  INF            1000     // Vô cùng
#define  YES            1        // Đã xét
#define  NO             0        // Chưa xét
#define  NULLDATA       -1       // Giả giá trị rỗng

// ================================================
// Dùng cho kiểu dữ liệu đồ thị
// ================================================
// Định nghĩa các kiểu dữ liệu
typedef  char           LabelType;
typedef  int            CostType;
typedef  CostType       MaTrix[MAX][MAX]; // Ma trận

// Định nghĩa cấu trúc của một đỉnh
struct Vertex
{
        LabelType       Label;   // Nhãn của đỉnh
        int             Visited; // Trạng thái
};

// Định nghĩa cấu trúc một cạnh
struct Edge
{
        int             Source;  // Đỉnh đầu
        int             Target;  // Đỉnh cuối
        CostType        Weight;  // Trọng số
        int             Marked; // Trạng thái
};
```

```
// Định nghĩa cấu trúc một đoạn đường đi
struct Path
{
	CostType	Length;  // Độ dài đđi
	int		Parent;  // Đỉnh trước
};

// Định nghĩa kiểu dữ liệu đồ thị
struct	Graph
{
	bool	Directed;		// ĐT có hướng?
	int	NumVertices;	// Số đỉnh
	int	NumEdges;		// Số cạnh
	MaTrix	Cost;			// MTrận kề
	Vertex	Vertices[MAX];	// DS đỉnh
};

// ================================================
// Dùng cho Stack và Queue hoặc dùng thư viện có sẵn
// ================================================

// Kiểu dữ liệu phần tử của Queue, Stack
struct Entry
{
	// Dữ liệu chứa trong một nút của Queue
	int		Data;		// hoặc Stack
	Entry*	Next;		// Con trỏ Next
};

// Định nghĩa kiểu con trỏ tới một Entry
typedef Entry*	EntryPtr;

// Kiểu dữ liệu ngăn xếp (Stack)
typedef EntryPtr Stack;

// Tạo một phần tử của Stack chứa
// dữ liệu là data
EntryPtr CreateEntry(int data)
{
	EntryPtr item = new Entry;
	if (item)
	{
		item->Data = data;
		item->Next = NULL;
	}
	return item;
}

#endif
```

- Trong tập tin program.cpp, nhập đoạn mã sau:

```
#include <iostream>
#include <conio.h>
#include <fstream>
using namespace std;
#include "common.h"
#include "queue.h" // Nếu dùng thư viện có sẵn thì #include <queue>
#include "stack.h" // Nếu dùng thư viện có sẵn thì #include <stack>
#include "graph.h"
```

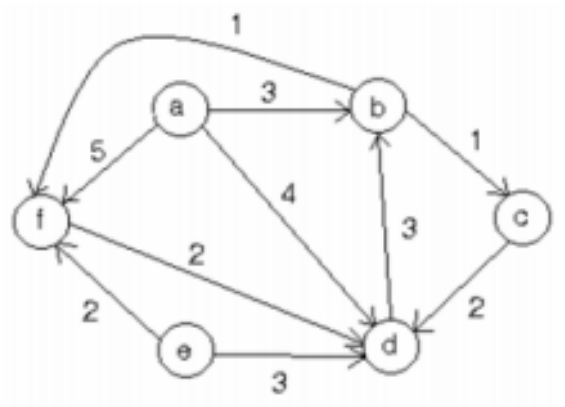
```
void main()
{
	_getch();
}
```

Vào menu Build > chọn Build Solution để biên dịch chương trình, kiểm tra lỗi. Nếu có lỗi, kiểm tra lại mã nguồn bạn đã viết. Nếu chương trình không có lỗi, ta sẽ cài đặt các thao tác, phép toán trên đồ thị.

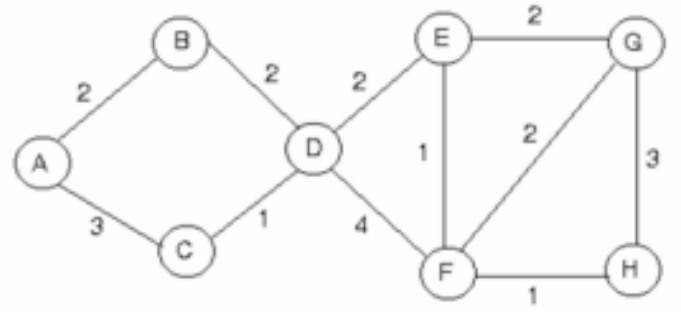
V. Bài tập thực hành (có HD)

Bài 1. Với mỗi đồ thị sau, hãy tạo các tập tin (sử dụng chương trình notepad, wordpad, …) lưu trữ chúng theo định dạng:

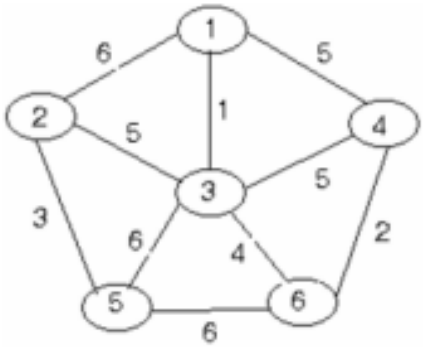
- Dòng đầu: Ghi số N (số đỉnh của đồ thị)
- Dòng thứ 2: Ghi số M (số cạnh của đồ thị)
- Dòng thứ 3: Ghi số 0 (nếu đồ thị vô hướng) hoặc 1 (nếu đồ thị có hướng)
- N dòng kế tiếp, mỗi dòng ghi nhãn của 1 đỉnh
- N dòng tiếp theo, mỗi dòng ghi N giá trị biểu diễn cho ma trận kề hay ma trận trọng số (gồm N dòng, N cột). Quy ước, nếu giữa hai đỉnh v và w có cạnh nối thì ghi giá trị trọng số của cạnh đó. Nếu không có cạnh nối thì ghi 0 (nếu v = w) hoặc 1000 (=INF nếu v khác w).



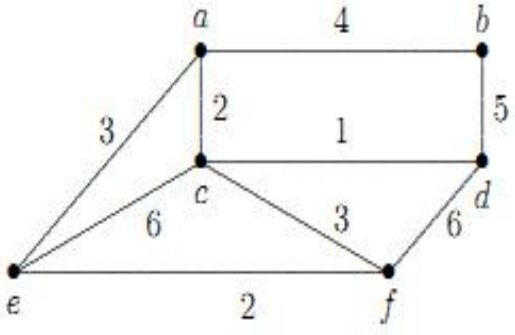
a)



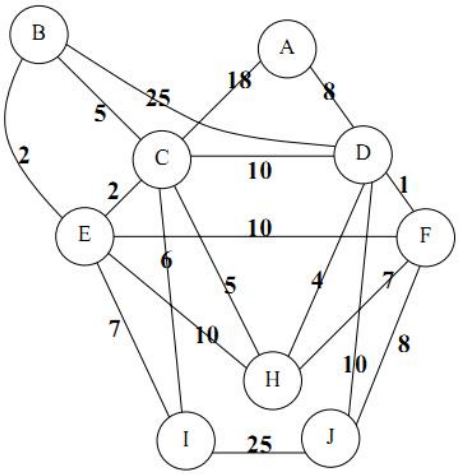
b)

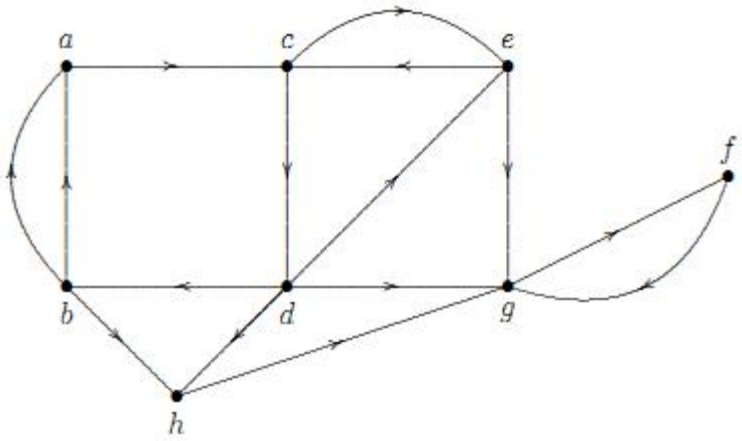


c)



d)





e) **f)**

Bài 2. Cài đặt các thao tác cơ bản trên kiểu dữ liệu đồ thị – Viết hàm trong tập tin graph.h (C++)

a. Tạo một đỉnh có nhãn lab

```
// Tạo một đỉnh có nhãn lab
Vertex CreateVertex(LabelType lab)
{
        Khai báo 1 biến v kiểu Vertex
        Gán nhãn lab cho đỉnh v
        Gán đỉnh v chưa xét
        Trả về v;
}
```

b. Hiển thị thông tin của một đỉnh

```
// Hiển thị thông tin đỉnh thứ pos trong đồ thị
void DisplayVertex(Graph g, int pos)
{
        cout << g.Vertices[pos].Label << "\t";
}
```

c. Hàm khởi tạo một đồ thị

```
// Khởi tạo một đồ thị
// directed = true: Đồ thị có hướng
Graph InitGraph(bool directed)
{
        Khai báo và khởi tạo một biến g kiểu Graph;
        Khởi tạo số cạnh của g = 0;
        Khởi tạo số đỉnh của g = 0;
        Gán loại đồ thị = directed; // Có hướng hay ko

        // Khởi tạo ma trận kề (ma trận trọng số, chi phí)
        for (int i=0; i < Số đỉnh của g; i++)
                for (int j=0; j< Số đỉnh của g; j++)
                        if (i == j)                     // ĐĐi từ một đỉnh tới chính nó thì =0
                                Khởi tạo đường đi từ đỉnh i đến j = 0;
                        else
                                Khởi tạo đường đi từ đỉnh i đến j = INF;
        return g;
}
```

d. Hàm thiết lập lại trạng thái của các đỉnh trong đồ thị

```
// Thiết lập lại trạng thái của các đỉnh
void ResetFlags(Graph &g)
{
        for (int i=0; i < Số đỉnh của g; i++)
                Thiết lập trường Visited của đỉnh i = NO;
}
```

e. Hàm kiểm tra 2 đỉnh có kề nhau hay không (2 đỉnh được nối với nhau bởi ít nhất một cạnh)

```
// Kiểm tra hai đỉnh start và end có được nối với nhau bởi 1 cạnh hay không
int IsConnected(Graph g, int start, int end)
{
        if (Chi phí đi từ start đến end = 0 hoặc INF)
                Trả về 0;
        else
                Trả về 1;
}
```

f. Hàm thêm một đỉnh có nhãn lab vào đồ thị

```
// Thêm một đỉnh có nhãn lab vào đồ thị g
void AddVertex(Graph &g, LabelType lab)
{
        Gọi hàm tạo một đỉnh v với nhãn lab;
        Đưa đỉnh v vào đồ thị g;
        Tăng số đỉnh của g lên 1;
}
```

g. Hàm thêm một cạnh nối giữa hai đỉnh của đồ thị

```
// Thêm một cạnh có trọng số là weight bắt đầu
// từ đỉnh start và kết thúc tại đỉnh end
// directed=false: Thêm cạnh vô hướng
void AddEdge(Graph &g, int start, int end,
                        CostType weight, bool directed)
{
        if (Hai đỉnh start, end không kề nhau)
                Tăng số cạnh lên 1;

        Gán chi phí đi từ start đến end = weight;
        if (Nếu đồ thị là vô hướng)
                Gán chi phí đi từ end đến start = weight;
}

// Thêm một cạnh có trọng số là weight bắt đầu
// từ đỉnh start và kết thúc tại đỉnh end
void AddEdge(Graph &g, int start, int end, CostType weight)
{
        AddEdge(g, start, end, weight, g.Directed);
}

// Thêm một cạnh bắt đầu từ đỉnh start và kết thúc tại
// đỉnh end. Dùng cho đồ thị không có trọng số.
void AddEdge(Graph &g, int start, int end)
{
        AddEdge(g, start, end, 1);
}
```

h. Hàm lưu đồ thị xuống file

```
// Lưu đồ thị xuống file
void SaveGraph(Graph g, char* fileName)
```

```
{
	// Khai báo biến và mở tập tin để ghi
	ofstream os(fileName);

	// Lưu số đỉnh
	os << g.NumVertices << '\n';

	// Lưu số cạnh
	os << g.NumEdges << '\n';

	// Lưu loại đồ thị
	os << g.Directed << '\n';

	// Lưu tên các đỉnh
	for (int i=0; i<g.NumVertices; i++)
		os << g.Vertices[i].Label << '\n';

	// Lưu ma trận kề
	for (int i=0; i<g.NumVertices; i++)
	{
		for (int j=0; j<g.NumVertices; j++)
		{
			os << g.Cost[i][j] << '\t';
		}
		os << '\n';
	}
	// Đóng tập tin
	os.close();
}
```

i. Hàm tạo đồ thị từ dữ liệu được lưu trong file

```
// Đọc dữ liệu từ file để tạo đồ thị
void OpenGraph(Graph &g, char* fileName)
{
	// Khai báo biến và mở file để đọc
	ifstream is(fileName);
	// kiểm tra đã mở được file chưa?
	if (is.is_open()) // Nếu mở file thành công
	{
		int n = 0, m = 0;
		bool d = false;
		LabelType lab;

		Đọc số đỉnh của đồ thị đưa vào biến n
		Đọc số cạnh của đồ thị đưa vào biến m
		Đọc loại đồ thị đưa vào biến d;

		Khởi tạo đồ thị g
		Gán số cạnh m của đồ thị vào g.NumEdges

		// Khởi tạo nhãn của các đỉnh
		for (int i=0; i< Số đỉnh của đồ thị; i++)
		{
			Đọc nhãn đưa vào biến lab;
			Gọi hàm thêm đỉnh có nhãn lab vào đồ thị
		}

		// Đọc ma trận kề từ file
		for (int i=0; i< Số đỉnh của đồ thị; i++)
		{
			for (int j=0; j< Số đỉnh của đồ thị; j++)
			{
```

```
                                        Và lưu vào đồ thị g.Cost[i][j]
                        }
                }
                is.close();                             // Đóng file
        }
        else
        {
                cout << "\nLoi doc file nhap lai ten file\n";
                system("pause");
                return 0;
        }

}
```

j. Hàm tìm đỉnh đầu tiên chưa được xét và kề với đỉnh hiện tại (curr)

```
// Tìm đỉnh đầu tiên kề với curr mà chưa xét
int FindFirstAdjacentVertex(Graph g, int curr)
{
        // Duyệt qua mọi đỉnh
        for (int i=0; i < Số đỉnh của g; i++)
        {
                // Kiểm tra đỉnh đã xét chưa và kề với curr ko?
                if (Đỉnh i chưa được xét và
                        Có cạnh nối từ curr đến i))
                        return i;               // Thỏa đk -> tìm thấy
        }
        return NULLDATA;                        // Không tìm thấy
}
```

Bài 3. Các phương pháp duyệt đồ thị.

a. Hàm duyệt đồ thị theo chiều sâu (dạng đệ quy)

```
// Duyệt đồ thị theo chiều sâu dạng đệ quy
void DFS_Recursion(Graph &g, int start)
{
        Đánh dấu đỉnh start đã xét : gán Visited = YES;
        Xuất thông tin đỉnh start;

        while (true)
        {
                Tìm đỉnh adj đầu tiên kề với start và chưa xét;
                if (Không tìm thấy đỉnh adj như vậy)
                        break;          // thì dừng, quay lui
                else
                        Gọi đệ quy, duyệt từ đỉnh adj;
        }
}
```

b. Hàm duyệt đồ thị theo chiều sâu (dạng lặp)

```
// Duyệt đồ thị theo chiều sâu (Depth First Search)
// Dạng lặp, sử dụng Stack
void DFS_Loop(Graph g, int start)
{
        Đánh dấu đỉnh start đã xét;
        Xuất thông tin đỉnh start;

        Khởi tạo Stack s;
        Đưa đỉnh start vào Stack s, start;

        int curr, adj;          // curr : Đỉnh đang xét
                                // adj  : Đỉnh kề với curr
```

```
        while (Stack s khác rỗng)
        {
                Xem phần tử đầu tiên curr từ Stack s;

                Tìm đỉnh adj đầu tiên kề với curr và chưa xét;
                if (Không tìm thấy đỉnh adj như vậy)
                        Pop(s);   // Loại bỏ 1 đỉnh trong Stack
                else                        // nhằm quay lại đỉnh trước
                {
                        Đánh dấu đỉnh adj đã xét,gán Visited=YES;
                        Hiển thị thông tin đỉnh adj;
                        Đưa đỉnh adj vào Stack s;
                }
        }
}
```

c. Hàm duyệt đồ thị theo chiều rộng

```
// Duyệt đồ thị theo chiều rộng
void BFS(Graph g, int start)
{
        Đánh dấu đỉnh start đã được xét;
        Khởi tạo hàng đợi q;
        Đưa đỉnh start vào hàng đợi q;

        int curr, adj;
        while (Hàng đợi q khác rỗng)        // Còn đỉnh chưa xét?
        {
                Lấy đỉnh đầu tiên curr từ hàng đợi;
                Xuất thông tin đỉnh curr;
                while (true)
                {
                        Tìm đỉnh adj kề với curr và chưa xét;
                        if (Không tim thấy adj như vậy) break;
                        else
                        {
                                Đánh dấu đỉnh adj đã được xét;
                                Đưa đỉnh adj vào hàng đợi;
                        }
                }
        }
}
```

Bài 4. Kiểm tra chương trình và xem kết quả. Viết hàm trong tập tin menu.h (C++)

VI. Bài tập

Tính liên thông của đồ thị

Cho một đồ thị G=(V,E). Viết chương trình:

- Kiểm tra xem đồ thị G có liên thông hay không?
- Nếu không, đếm số thành phần liên thông của đồ thị G.
- Liệt kê các thành phần (đồ thị con) liên thông của đồ thị G

Gợi ý:

- Dùng 1 trong các phương pháp duyệt DFS hoặc BFS: Mỗi khi không thể đi đến một đỉnh khác, ta tăng số thành phần liên thông lên 1. Tìm một đỉnh khác chưa được xét và duyệt tiếp.
- Hoặc sử dụng thuật toán tìm cây bao trùm: Đồ thị G là liên thông khi và chỉ khi G có cây bao trùm.

=== HẾT ===

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2

LAB 5 – ĐỒ THỊ VÀ BIỂU DIỄN ĐỒ THỊ BẰNG DANH SÁCH KỀ (4 TIẾT)

I. Mục tiêu

Sau khi thực hành, sinh viên cần:

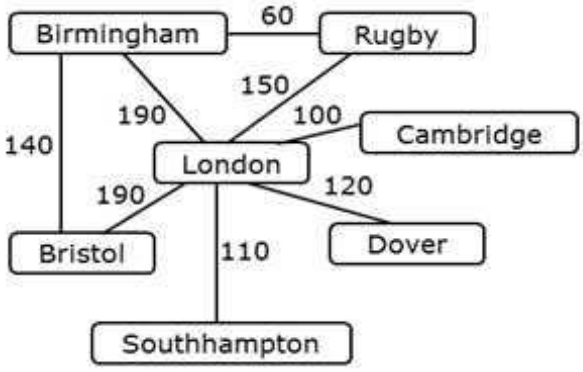
- Nắm vững định nghĩa đồ thị, các khái niệm liên quan đến đồ thị.
- Cài đặt được kiểu dữ liệu đồ thị và các thao tác, phép toán trên đồ thị được biểu diễn bởi danh sách kề.
- Vận dụng kiến thức đã học để giải một số bài toán thực tế.

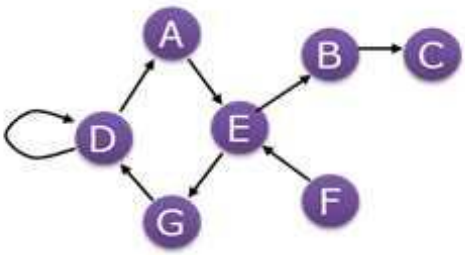
II. Yêu cầu

- Sinh viên phải hoàn thành ***4 bài*** thuộc mục V. Mỗi bài tập tạo một project, xóa các thư mục debug của project này. Sau đó chép các project vào thư mục: Lab5_CTK40_HoTen_MSSV_Nhom#. Nén thư mục , đặt tên tập tin nén theo dạng sau: Lab5_CTK40_HoTen_MSSV_Nhom#.rar.
  Ví dụ: Lab5_CTK40_NguyenVanA_161111_Nhom4.rar.
- Sinh viên sẽ nộp bài Lab qua mạng tại phòng lab theo hướng dẫn của giáo viên.

III. Ôn tập lý thuyết

1. *Đồ thị và các định nghĩa*





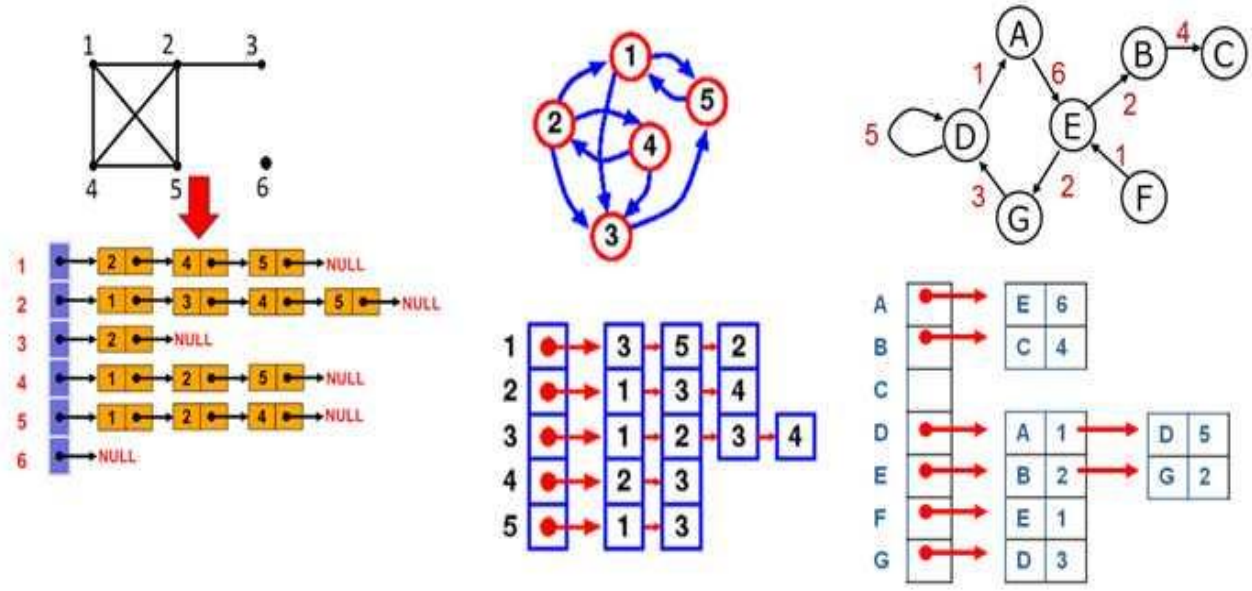
Một đồ thị G được định nghĩa là G = (V, E). Trong đó:

- V(G) là tập hữu hạn các đỉnh và khác rỗng. E(G) là tập các cung (cạnh) = Tập các cặp đỉnh (u, v) mà u, v ϵ V.
- Các đỉnh còn gọi là các nút (node) hay điểm (point)
- Mỗi cung nối giữa hai đỉnh v, w có thể ký hiệu là cặp (v,w). Hai đỉnh có thể trùng nhau. Nếu cặp (v,w) có thứ tự thì gọi là cung có thứ tự hay cạnh có hướng. Ngược lại ta nói là cung không có thứ tự hay cạnh vô hướng.
- Đỉnh kề: Hai đỉnh được gọi là kề nhau nếu chúng được nối với nhau bởi một cạnh. Trong trường hợp này, hai nút đỉnh được gọi là hàng xóm (neighbors) của nhau.
- Đường đi là một dãy tuần tự các đỉnh v_1, v_2, …, v_n sao cho (v_i, v_{i+1}) là một cung trên đồ thị. Đỉnh v_1 được gọi là đỉnh đầu, đỉnh v_n được gọi là đỉnh cuối. Độ dài đường đi là số cạnh trên đường đi.
- Đỉnh X gọi là có thể đi đến được từ đỉnh Y nếu tồn tại một đường đi từ đỉnh Y đến đỉnh X.
- Đường đi đơn là đường đi mà mọi đỉnh trên đó đều khác nhau, ngoại trừ đỉnh đầu và cuối có thể trùng nhau.

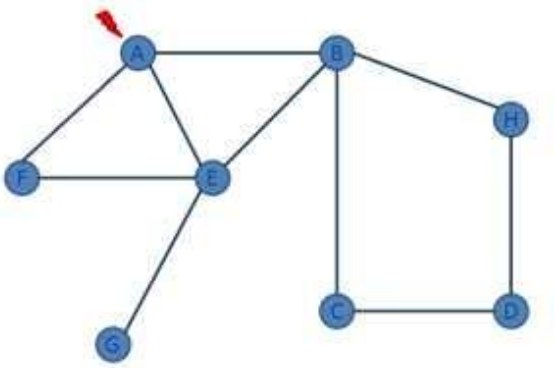
- Chu trình là đường đi có đỉnh đầu trùng với đỉnh cuối. Chu trình đơn là một đường đi đơn có đỉnh đầu và đỉnh cuối trùng nhau và có độ dài ít nhất là 1.
- Cấp của đồ thị là số đỉnh của đồ thị. Kích thước của đồ thị là số cạnh của đồ thị. Đồ thị rỗng là đồ thị có kích thước bằng 0
- Bậc của một đỉnh là số cạnh nối với đỉnh đó
- Trong nhiều ứng dụng, ta thường kết hợp các giá trị (value) hoặc nhãn (label) cho các cạnh hoặc đỉnh. Khi đó, ta gọi là đồ thị có trọng số.
- Nhãn có thể có kiểu tùy ý và dùng để biểu diễn tên, chi phí, khoảng cách, …
- Đồ thị có hướng là đồ thị mà các cạnh của nó là có hướng. Nghĩa là các cặp đỉnh (v,w) có phân biệt thứ tự.
- Đồ thị vô hướng là đồ thị mà các cặp đỉnh tương ứng với các cạnh của nó không phân biệt thứ tự.

2. *Biểu diễn đồ thị*

Biểu diễn bằng danh sách kề



3. *Các phương pháp duyệt đồ thị*



Kết quả: A B C D H E F G

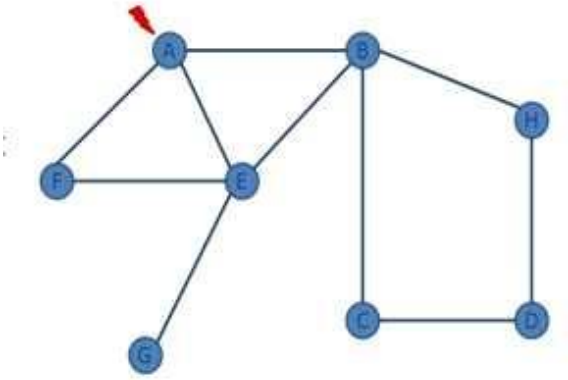
Duyệt đồ thị là một thủ tục có hệ thống để khám phá đồ thị bằng cách kiểm tra tất cả các đỉnh và các cạnh của nó.

Có hai cách duyệt đồ thị phổ biến

- Duyệt đồ thị theo chiều sâu (Depth First Search - DFS)
- Duyệt đồ thị theo chiều rộng (Breadth First Search -BFS)

Cách 1: Duyệt đồ thị theo chiều sâu

- B1. Khởi gán tất cả các đỉnh đều chưa được duyệt
- B2. Xuất phát từ một đỉnh **v** bất kỳ của đồ thị, đánh dấu đỉnh v đã được duyệt
- B3. Xử lý đỉnh v
- B4. Với mỗi đỉnh w chưa được duyệt và kề với v, ta thực hiện đệ quy quá trình trên cho w.



Kết quả: A B E F C H G D

Cách 2: Duyệt đồ thị theo chiều rộng

- B1. Khởi gán tất cả các đỉnh đều chưa được duyệt
- B2. Xuất phát từ một đỉnh **v** bất kỳ của đồ thị, đánh dấu đỉnh v đã được duyệt. Đưa v vào hàng đợi.
- B3. Lấy x ra khỏi hàng đợi. Xử lý đỉnh x.
- B4. Với mỗi đỉnh w chưa được duyệt và kề với x, ta đánh dấu w đã được duyệt. Đưa w vào hàng đợi.
- B5. Quay lại B3.

IV. Hướng dẫn thực hành

1. *Tạo dự án*

Sinh viên có thể chọn cài đặt kiểu dữ liệu đồ thị theo ngôn ngữ C++.

- Tạo dự án mới, đặt tên là Lab5_GraphAsList
- Trong thư mục Header Files, tạo ra các files sau:
  - menu.h: Định nghĩa các thao tác thực thi của chương trình
  - common.h: Định nghĩa các hằng số và kiểu dữ liệu đồ thị
  - graph.h: Định nghĩa các thao tác trên đồ thị
- Trong thư mục Source Files, tạo tập tin: program.cpp

2. *Định nghĩa kiểu dữ liệu đồ thị (Biểu diễn bằng danh sách kề)*

- Nhấp đôi chuột vào file common.h, định nghĩa các hằng số và kiểu dữ liệu như sau:

```
#define TAB         '\t'   // Khoảng TAB
#define EOL         '\n'   // Xuống dòng
#define UPPER       100    // Số ptu tối đa
#define ZERO        0      // Giá trị 0
#define MAX         30     // Số đỉnh tối đa
#define INF         1000   // Vô cùng
```

```
#define YES         1      // Đã xét
#define NO          0      // Chưa xét
#define NULLDATA    -1     // Giả giá trị rỗng

// Định nghĩa các kiểu dữ liệu
typedef char LabelType;
typedef int CostType;

// Định nghĩa cấu trúc một cạnh
struct Edge
{
	int Marked; // Trạng thái
	char Target; // Đỉnh cuối
	CostType Weight; // Trọng số
	Edge* Next; // Cạnh tiếp
};

// Định nghĩa cấu trúc của một đỉnh
struct Vertex
{
	LabelType Label; // Nhãn của đỉnh
	int Visited; // Trạng thái
	Edge* EdgeList; // DS cạnh kề
};

struct Path // Một đoạn đường đi
{
	CostType Length; // Độ dài đđi
	int Parent; // Đỉnh trước
};
typedef Path* PathPtr;

typedef Edge* EdgePtr;
typedef Vertex* VertexPtr;

// Định nghĩa kiểu dữ liệu đồ thị
struct GraphADT
{
	bool Directed; // ĐT có hướng?
	int NumVertices; // Số đỉnh
	int NumEdges; // Số cạnh
	VertexPtr Vertices[MAX]; // DS cac đỉnh kề
};
```

- Trong tập tin program.cpp, nhập đoạn mã sau:

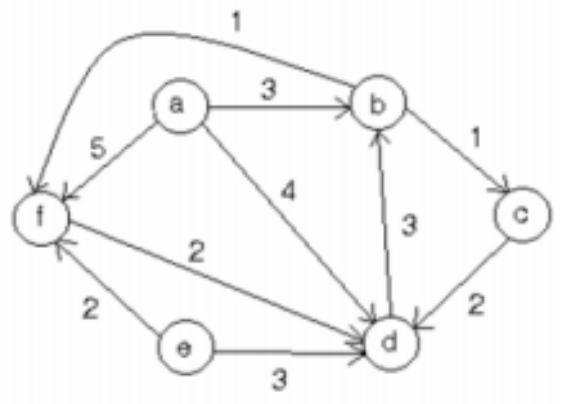
```
#include <iostream>
#include <conio.h>
#include <fstream>
#include <stack> // Dùng cho Stack trong thư viện có sẵn
#include <queue> // Dùng cho Queue trong thư viện có sẵn

using namespace std;

#include "common.h"
#include "graph.h"

void main()
{
	_getch();
}
```

Vào menu Build > chọn Build Solution để biên dịch chương trình, kiểm tra lỗi. Nếu có lỗi, kiểm tra lại mã nguồn bạn đã viết. Nếu chương trình không có lỗi, ta sẽ cài đặt các thao tác, phép toán trên đồ thị.

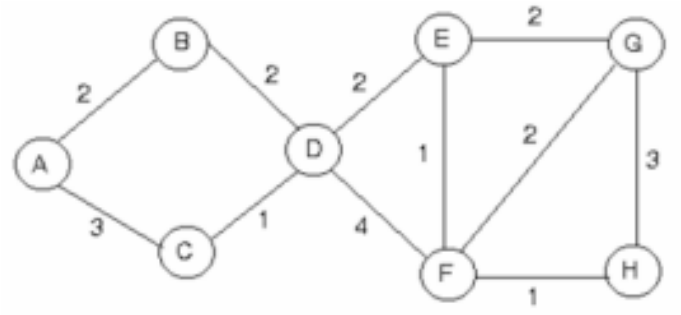
V. Bài tập thực hành (có HD)

Bài 1. Với mỗi đồ thị sau, hãy tạo các tập tin (sử dụng chương trình notepad, wordpad, …) lưu trữ chúng theo định dạng:

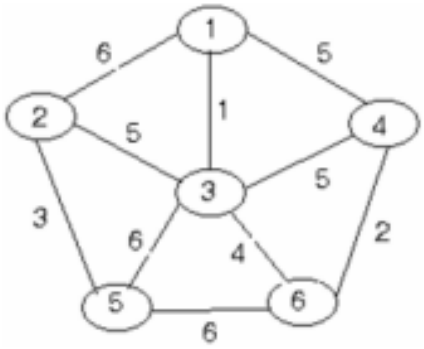
- Dòng đầu: Ghi số N (số đỉnh của đồ thị)
- Dòng thứ 2: Ghi số M (số cạnh của đồ thị)
- Dòng thứ 3: Ghi số 0 (nếu đồ thị vô hướng) hoặc 1 (nếu đồ thị có hướng)
- N dòng kế tiếp, mỗi dòng ghi nhãn của 1 đỉnh
- N dòng tiếp theo, mỗi dòng ghi N giá trị biểu diễn cho ma trận kề hay ma trận trọng số (gồm N dòng, N cột). Quy ước, nếu giữa hai đỉnh v và w có cạnh nối thì ghi giá trị trọng số của cạnh đó. Nếu không có cạnh nối thì ghi 0 (nếu v = w) hoặc 1000 (=INF nếu v khác w).



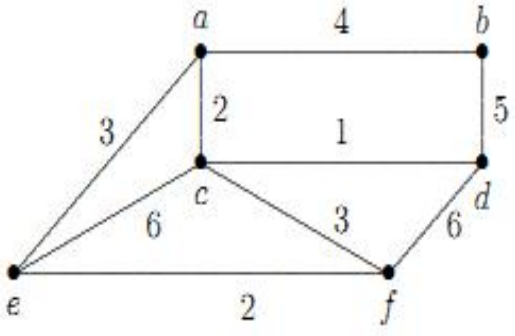
a)



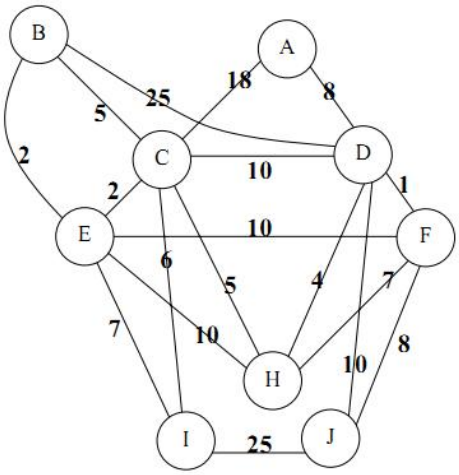
b)



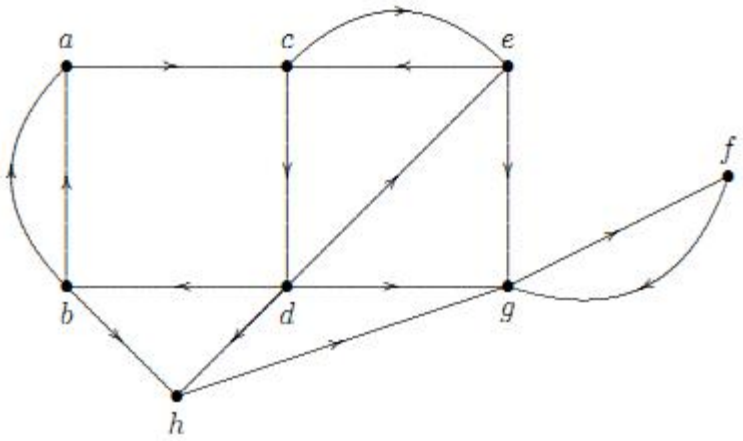
c)



d)



e)



f)

Bài 2. Cài đặt các thao tác cơ bản trên kiểu dữ liệu đồ thị – Viết hàm trong tập tin graph.h (C++)

a. Tạo một đỉnh có nhãn lab

```
// Tạo một đỉnh có nhãn lab
VertexPtr CreateVertex(LabelType lab)
{
	Khai báo và khởi tạo một đỉnh v có kiểu VertexPtr
	Gán nhãn lab cho đỉnh v
	Gán đỉnh v chưa xét
	Gán danh sách đỉnh liên kết từ v = NULL
	Trả về đỉnh v
}


// Tạo một cạnh chỉ đến đỉnh có nhãn label, trọng số w
EdgePtr CreateEdge(char lab, CostType w)
{
	Khai báo và khởi tạo biến động e có kiểu EdgePtr
	Lưu nhãn đỉnh đích lab vào e
	Lưu trọng số w vào e
	Đánh dấu e chưa xét
	Gán liên kết kế tiếp từ e = NULL
	Trả về cạnh e
}
```

b. Hiển thị thông tin của một đỉnh

```
//Tìm vị trí đỉnh v trong đồ thị g
int FindIndexVertex(Graph g, char v)
{
	for (int i = 0; i < Số đỉnh của đồ thị g; i++)
	{
		if (Nếu đỉnh thứ i có nhãn = v)
			Trả về i;
	}
	Trả về -1;
}
// Xuất danh sách các cạnh của 1 đỉnh được trỏ bởi ds
void XuatDSCanh(EdgePtr ds)
{
	while (ds khác rỗng)
	{
		Xuất ra màn hình Target và Weight;
```

```
                Đến liên kết kế tiếp;
        }
}

// Hiển thị thông tin đỉnh v trong đồ thị g
void Xuat1Dinh(char v, Graph g)
{
        Khai báo biến s và dùng hàm FindIndexVertex để đến đỉnh v
        if (không tìm thấy v)
                cout << “Khong co dinh “ << v << ”trong do thi”;
        else
        {
                Xuất nhãn đỉnh của s;
                Xuất danh sách các cạnh của s
        }
}
```

c. *Hàm khởi tạo một đồ thị*

```
// Khởi tạo một đồ thị
// directed = true: Đồ thị có hướng
Graph InitGraph(bool directed)
{
        Khai báo và khởi tạo đồ thị g
        Khởi tạo số cạnh của g = 0
        Khởi tạo số đỉnh của g = 0
        Gán đồ thị có hướng hay không có hướng cho đồ thị g
        // Khởi tạo danh sách đỉnh
        for (int i = 0; i < Số đỉnh của đồ thị; i++)
        {
                g->Vertices[i] = NULL;
        }
        return g;
}
```

d. *Hàm thiết lập lại trạng thái của các đỉnh trong đồ thị*

```
// Thiết lập lại trạng thái của các đỉnh
void ResetFlags(Graph &g)
{
        for (int i = 0; i < Số đỉnh của đồ thị; i++)
                g->Vertices[i]->Visited = NO; // Thiết lập trường Visited của đỉnh i = NO
}
```

e. *Hàm kiểm tra 2 đỉnh có kề nhau hay không (2 đỉnh được nối với nhau bởi ít nhất một cạnh)*

```
//Tìm vị trí đỉnh
int FindIndexVertex(Graph g, char vertex)
{
        for (int i = 0; i < Số đỉnh của đồ thị; i++)
        {
                if (nhãn của đỉnh thứ I = vertex)
                        return i;
        }
        return -1;
}

// Tìm cạnh nối 2 đỉnh start và end
EdgePtr FindEdge(Graph g, char start, char end)
{
        // Bắt đầu tìm từ cạnh kề đầu tiên với start
        Tìm đỉnh đầu start bằng hàm FindIndexVertex
        Khai báo và gán danh sách cạnh
        // Tìm cạnh với đỉnh cuối là end
        while (danh sách cạnh khác NULL) // Duyệt qua các cạnh kề
```

```
	{
		if (nhãn của danh sách cạnh = end)
			break; // Nếu tìm thấy thì dừng.
		Đi đến cạnh kế tiếp; // Nếu ko thì sang cạnh tiếp
	}
	return e;//trả về null
}

// Kiểm tra hai đỉnh u, v có kề nhau (có cạnh nối giữa chúng) hay không?
int IsConnected(Graph g, int start, int end)
{
	Khai báo biến e có kiểu EdgePtr;
	Gán biến e= FindEdge(g, start, end);
	return (e != NULL);//true false
}
```

f. Hàm thêm một đỉnh có nhãn lab vào đồ thị

```
//Thêm đỉnh v vào đồ thị g
void AddVertex(Graph &g, VertexPtr v)
{
	g->Vertices[g->NumVertices] = v;//thêm vào cuối
	g->NumVertices++;//tăng số đỉnh
}
```

g. Hàm thêm một cạnh nối giữa hai đỉnh của đồ thị

```
// Thêm một cạnh e vào cạnh v (nối vào danh sách v)
void AddEdge(EdgePtr &v, EdgePtr e)
{

	if (v == NULL)
		v = e;
	else
	{
		AddEdge(v->Next, e);

	}
}
// Thêm một cạnh chỉ đến đỉnh có nhãn lab với trọng số w vào danh sách của đỉnh v

void AddEdge(Graph &g, char v, char lab, CostType w)
{

	if (v == NULL)
		v = lab;
	else
	{

		if (g->Directed == false)//Nếu không có hướng
		{
			EdgePtr h, k;
			h = CreateEdge(lab, w);
			k = CreateEdge(v, w);
			AddEdge(g->Vertices[FindIndexVertex(g, v)]->EdgeList, h);
			AddEdge(g->Vertices[FindIndexVertex(g, lab)]->EdgeList, k);
			g->NumEdges = g->NumEdges + 2; //Do thêm 2 cạnh
		}
		else//neu co huong
		{
			EdgePtr a;
			a = CreateEdge(lab, w);
			AddEdge(g->Vertices[FindIndexVertex(g, v)]->EdgeList, a);
			g->NumEdges = g->NumEdges + 1;
		}
```

```
	}
}
```

h. Hàm lưu đồ thị xuống file

```
// Lưu đồ thị xuống file
void WriteGraph(Graph g, char* filename)
{
	ofstream out(filename); // Khai báo biến và mở tập tin để ghi
	out << g->NumVertices << '\n'; // Lưu số đỉnh
	out << g->NumEdges << '\n'; // Lưu số cạnh
	out << g->Directed << '\n'; // Lưu loại đồ thị
	// Lưu tên các đỉnh
	for (int i = 0; i < g->NumVertices; i++)
		out << g->Vertices[i]->Label << '\n';
	// Lưu ma trận kề
	for (int i = 0; i < g->NumVertices; i++)
	{
		for (int j = 0; j < g->NumVertices; j++)
		{
			if (i == j)
				out << "0" << "\t";
			else
				if (IsConnected_2(g, g->Vertices[i]->Label, g->Vertices[j]-
>Label))
				{
					EdgePtr temp = FindEdge(g, g->Vertices[i]->Label, g-
>Vertices[j]->Label);
					out << g->Vertices[j]->EdgeList->Weight << TAB;
				}
				else
				{
					out << "1000" << TAB;
				}
		}
		if (i != g->NumVertices - 1)
			out << "\n";
	}
	out.close();// Đóng tập tin
}
```

i. Hàm tạo đồ thị từ dữ liệu được lưu trong file

```
// Đọc dữ liệu từ file để tạo đồ thị
int OpenGraph(Graph &g, char* filename)
{
	ifstream in(filename);
	if (in.is_open())
	{
		int n = 0, m = 0, w = 0;
		bool d = false;
		char lab;
		in >> n;	//doc so dinh
		in >> m;	//doc so canh
		in >> d;	//doc loai do thi
		g = InitGraph(d);
		for (int i = 0; i < n; i++)
		{
			in >> lab;
			VertexPtr v = CreateVertex(lab);
			AddVertex(g, v);
		}
		lab = ' ';
		int indexVertex;
```

```
		for (int i = 0; i < m; i++)
		{
			char dinh, dich;
			int trongSo;
			in >> dinh;
			in >> dich;
			in >> trongSo;
			if (lab != dinh)
			{
				lab = dinh;
				indexVertex = FindIndexVertex(g, dinh);
			}

			if (trongSo == NULL)
			{
				trongSo = 1;
			}
			EdgePtr e = CreateEdge(dich, trongSo);
			AddEdge(g->Vertices[indexVertex]->EdgeList, e);
			g->NumEdges++;
		}
		in.close();
		return 1;
	}
	else
	{
		cout << "\nLoi doc file nhap lai ten file\n";
		system("pause");
		return 0;
	}
}
```

j. Hàm tìm đỉnh đầu tiên chưa được xét và kề với đỉnh hiện tại (curr)

```
// Tìm đỉnh đầu tiên kề với cur mà chưa xét
char FindFirstAdjacentVertex(Graph g, char cur)
{
	// Duyệt qua mọi đỉnh
	for (int i = 0; i < g->NumVertices; i++)
	{
		if (g->Vertices[i]->Label == cur)
		{
			EdgePtr p = g->Vertices[i]->EdgeList;
			while (p != NULL)
			{
				for (int j = 0; j < g->NumVertices; j++)
					if (g->Vertices[j]->Label == p->Target&&g->Vertices[j]-
>Visited == NO)
						return p->Target; // Thỏa đk -> tìm thấy
				p = p->Next;
			}
		}
	}
	return NULLDATA; // Không tìm thấy

}
```

Bài 3. Các phương pháp duyệt đồ thị.

a. Hàm duyệt đồ thị theo chiều sâu (dạng lặp)

```
// Duyệt đồ thị theo chiều sâu (Depth First Search)
// Dạng lặp, sử dụng Stack
void DFS(Graph g, char start)
{
	int index = FindIndexVertex(g, start);
```

```
        g->Vertices[index]->Visited = YES;
        cout << start << TAB;
        stack<char> st;
        st.push(start);
        char cur, adj;
        while (!st.empty())
        {
                cur = st.top();
                adj = TimDinhDauTien(g, cur);
                if (adj == NULLDATA)
                        st.pop();
                else
                {
                        int index = FindIndexVertex(g, adj);
                        g->Vertices[index]->Visited = YES;
                        cout << adj << TAB;
                        st.push(adj);
                }
        }
}
```

b. Hàm duyệt đồ thị theo chiều rộng

```
// Duyệt đồ thị theo chiều rộng
// Dạng lặp, sử dụng Queue

void BFS(Graph g, char start)
{
        int index = FindIndexVertex(g, start);
        g->Vertices[index]->Visited = YES;
        queue<char> que;
        que.push(start);
        char cur, adj;
        cur = que.front();
        cout << cur << TAB;
        while (!que.empty())
        {
                adj = TimDinhDauTien(g, cur);
                if (adj == NULLDATA)
                {
                        que.pop();
                        if (que.size() != 0)
                                cur = que.front();
                        else
                                break;
                }
                else
                {
                        index = FindIndexVertex(g, adj);
                        g->Vertices[index]->Visited = YES;
                        cout << adj << TAB;
                        que.push(adj);
                }
        }
        ResetFlags(g);
}
```

Bài 4. Kiểm tra chương trình và xem kết quả: Hoàn thiện hàm main và viết hàm trong tập tin menu.h (C++)

VI. Bài tập

Bài 1. Đồ thị biểu diễn bằng danh sách cạnh

- Tìm hiểu cách biểu diễn đồ thị bằng danh sách cạnh (Soạn slide thuyết trình)
- Cài đặt chương trình theo yêu cầu tương tự như V ở trên.

Bài 2. Đồ thị biểu diễn bằng ma trận liên thuộc

- Tìm hiểu cách biểu diễn đồ thị bằng ma trận liên thuộc (Soạn slide thuyết trình)
- Cài đặt chương trình theo yêu cầu tương tự như V ở trên.

=== HẾT ===

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2

LAB 6 – MỘT SỐ BÀI TOÁN TRÊN ĐỒ THỊ (4 TIẾT)

I. Mục tiêu

Sau khi thực hành, sinh viên cần:

- Nắm vững định nghĩa đồ thị, các khái niệm liên quan đến đồ thị.
- Cài đặt được kiểu dữ liệu đồ thị và các thao tác, phép toán trên đồ thị được biểu diễn bởi ma trận kề.
- Vận dụng kiến thức đã học để giải một số bài toán thực tế.

II. Yêu cầu

- Sinh viên phải hoàn thành cả ***3 bài*** thuộc mục V. Mỗi bài tạo một project, xóa các thư mục debug của project này. Sau đó chép các project vào thư mục: Lab6_CTK40_HoTen_MSSV_Nhom#. Nén thư mục , đặt tên tập tin nén theo dạng sau: Lab6_CTK40_HoTen_MSSV_Nhom#.rar.

Ví dụ: Lab6_CTK40_NguyenVanA_161111_Nhom4.rar.

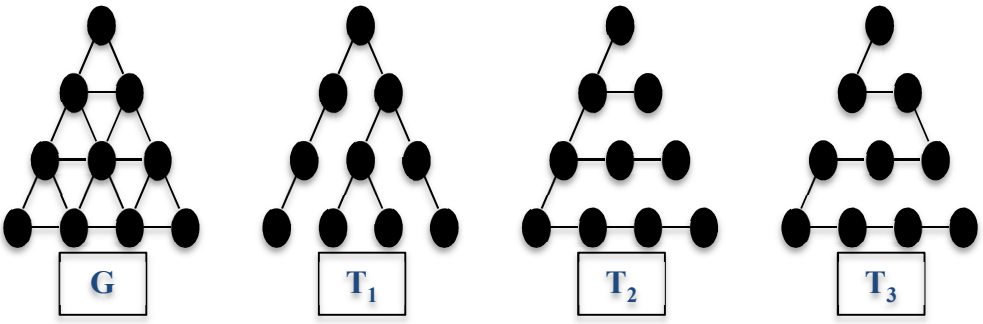
- Sinh viên sẽ nộp bài Lab qua mạng tại phòng lab theo hướng dẫn của giáo viên.

III. Ôn tập lý thuyết

1. Bài toán tìm cây bao trùm tối thiểu

Phát biểu bài toán:

- Cho đồ thị vô hướng G=(V,E)
- Tìm đồ thị T=(V,F) trong đó F là tập con của E sao cho:
  - (1) T liên thông
  - (2) T không có chu trình
  - (3) Tổng độ dài các cạnh trong T là nhỏ nhất



- Nếu T chỉ thỏa (1), (2) thì T gọi là cây bao trùm
- Nếu thỏa (1), (2), (3) thì T gọi là cây bao trùm tối thiểu.
- Một số tài liệu sử dụng thuật ngữ: Cây khung

Các thuật toán tìm cây bao trùm tối thiểu

Thuật toán Prim	**Thuật toán Kruskal**
▪ Input: G = (V, E) ▪ Output: T = (V, F) nhỏ nhất ▪ U: Tập các đỉnh chưa được chọn (xét) ▪ F: Tập các cạnh được chọn ▪ Khởi tạo U = Ø, F = Ø	▪ Input: G = (V, E) ▪ Output: T = (V, F) nhỏ nhất ▪ Khởi tạo cây T không có cạnh nào (F = Ø), chỉ gồm n đỉnh ▪ Sắp xếp các cạnh của G tăng dần theo trọng số

▪ Chọn một đỉnh v bất kỳ làm gốc của cây bao trùm
▪ Đưa v vào U
▪ Trong khi U khác V
  ▪ Chọn cạnh (u, v) nhỏ nhất sao cho u ∈ U, v ∈ V – U
  ▪ Thêm v vào U
  ▪ Thêm (u, v) vào F
▪ Kết thúc: T=(V, F) là cây bao trùm tối thiểu
	▪ Lần lượt xét từng cạnh (u,v) từ trọng số nhỏ nhất đến lớn nhất
	▪ Thêm cạnh (u, v) vào F nếu không tạo thành chu trình
	▪ Lặp lại bước trên cho đến khi đủ n-1 cạnh hoặc mọi cạnh còn lại đều tạo thành chu trình.
	▪ Kết thúc: T=(V, F) là cây bao trùm tối thiểu

2. Bài toán tìm đường đi ngắn nhất

❖ Phát biểu bài toán
- Cho đồ thị G = (V, E)
- Mỗi cạnh được gán một giá trị không âm, gọi là trọng số (hoặc giá, chi phí) của cạnh
- Cho trước một đỉnh s_0, gọi là đỉnh nguồn
- Vấn đề: Tìm đường đi ngắn nhất từ đỉnh s_0 đến các đỉnh còn lại sao cho tổng chi phí trên đường đi là nhỏ nhất.

❖ Input
- s_0: Đỉnh nguồn
- G: Đồ thị được biểu diễn bởi ma trận kề hoặc danh sách kề

❖ Output
- L: Mảng biểu diễn độ dài đường đi ngắn nhất từ s_0 đến các đỉnh #
- T: Mảng lưu vết đường đi từ s_0 đến các đỉnh còn lại

Thuật toán Dijkstra	**Thuật toán Floyd**
❖ Tìm đường đi từ 1 đỉnh tới các đỉnh còn lại trong đồ thị ❖ Ký hiệu ▪ Giả sử G có n đỉnh, trọng số được lưu trong ma trận C ▪ C[i,j] là chi phí của cung (i, j). C[i,j] = **ε** nếu i không kề j ▪ L[w]: lưu độ dài đường đi từ s_0 đến đỉnh w ▪ T[w]: lưu đỉnh trước w trên đường đi ▪ M[w]: cho biết đỉnh w đã được xét hay chưa ▪ L, T và M sẽ được cập nhật lại sau mỗi bước của thuật toán ▪ S: là tập các đỉnh đã được chọn ▪ U: là tập các đỉnh chưa được chọn ❖ Ý tưởng ▪ Khởi đầu: S = {s_0}, L[w] = C[s_0, w], T[w] = w	❖ Tìm đường đi ngắn nhất giữa mọi cặp đỉnh ❖ Ký hiệu: ▪ L[i, j]: Độ dài đường đi ngắn nhất từ đỉnh i đến đỉnh j ▪ Ban đầu, L được khởi tạo giống với C: L[i,j] = C[i,j] ▪ T[i, j]: Lưu lại đường đi từ i đến j ▪ T[i, j] = k: k là đỉnh trước đỉnh j trong đường đi từ i → j ❖ Ý tưởng: ▪ Tại lần lặp thứ k sẽ xác đỉnh khoảng cách ngắn nhất giữa hai đỉnh i và j theo công thức: $L_k[i, j] = \min(L_{k-1}[i, j], L_{k-1}[i, k] + L_{k-1}[k, j])$ ▪ Để tìm đường đi từ v đến một đỉnh w, ta sử dụng mảng T để lần ngược các đỉnh trước k trên đường đi.

▪ Tại mỗi bước, ta chọn một đỉnh w trong U mà khoảng cách L[w] từ đỉnh nguồn tới w là nhỏ nhất. ▪ Loại w khỏi U và thêm w vào tập S. ▪ Với mỗi đỉnh u thuộc U, tính lại độ dài đường đi ngắn nhất từ s_0 đến u. L(u) = Min(L(u), L(w) + C[w, u]) ▪ Kết thúc: S = tập các đỉnh của G ▪ Để tìm đường đi từ s_0 đến một đỉnh w, ta sử dụng mảng T để lần ngược các đỉnh trước w trên đường đi.	

IV. Hướng dẫn thực hành

1. Tạo dự án

Sinh viên chọn cài đặt kiểu dữ liệu đồ thị theo ngôn ngữ C++.

Sử dụng dự án đã tạo trong bài Lab4

V. Bài tập thực hành

Bài 1. Cây bao trùm tối thiểu

a. Thuật toán Prim

```
// Tìm cây bao trùm nhỏ nhất theo thuật toán Prim
// Kết quả được lưu trong mảng tree. Chỉ số mảng i
// là đỉnh nguồn, tree[i].Parent là đỉnh cuối và
// tree[i].Length là trọng số của cạnh được chọn.
void Prim(Graph g, Path tree[MAX])
{
	// Khởi tạo cây ban đầu có tất cả các cạnh
	// xuất phát từ đỉnh đầu tiên (đỉnh 0)
	for (int i=1; i< Số đỉnh của g; i++)
	{
		tree[i].Length = g.Cost[0][i];
		tree[i].Parent = 0;
	}

	CostType min;	// Lưu cạnh có trọng số bé nhất
	int minVertex;	// Lưu đỉnh cuối của cạnh đó

	// Tìm n-1 cạnh cho cây bao trùm
	for (int i=1; i< Số đỉnh của g; i++)
	{
		min = INF;		// Giả sử min = vô cùng
		minVertex = 1;

		// Duyệt qua các cạnh để tìm cạnh min
		for (int j=2; j< Số đỉnh của g; j++)
		{
			if (Đỉnh j chưa được xét &&
				tree[j].Length < min)
			{
				min = tree[j].Length;
				minVertex = j;
			}
		}
		// Đánh dấu đỉnh cuối của cạnh min là đã xét
		g.Vertices[minVertex].Visited = YES;
```

```
            // Cập nhật lại trọng số của các cạnh
            // với đỉnh nguồn bây giờ là minVertex
            for (int j=0; j< Số đỉnh của g; j++)
            {
               if (Đỉnh j chưa được xét && Chi phí từ
                  đỉnh minVertex đến j < tree[j].Length)
               {
                  tree[j].Length = g.Cost[minVertex][j];
                  tree[j].Parent = minVertex;
               }
            }
      }
}
// Xuất danh sách cạnh được chọn làm cây bao trùm
// theo thuật toán Prim
void PrintPrimMST(Graph g, Path tree[MAX])
{
      cout << endl << "Cay bao trum gom cac canh sau:";
      CostType sum = 0; // Lưu tổng trọng số

      // Duyệt qua các đỉnh để tìm cạnh nối với đỉnh đó
      // và nằm trong cây bao trùm tối thiểu
      for (int i=1; i<g.NumVertices; i++)
      {
            Cập nhật tổng trọng số;       // sum
            Xuất nhãn của đỉnh đầu, nhãn của đỉnh cuối và
            chiều dài (trọng số) của cạnh nối 2 đỉnh đó;
      }
      cout << endl << "Cay bao trum ngan nhat co chieu dai
                        : " << sum;
}
```

b. *Thuật toán Kruskal*

```
// Duyệt ma trận kề và lấy ra danh sách các cạnh của
// đồ thị. Mỗi cạnh lưu đỉnh đầu, cuối và trọng số.
int AdjMatrix2EdgeList(Graph g, Edge edgeList[UPPER])
{
      int count = 0;                        // Lưu số cạnh
      for (int i=0; i<g.NumVertices; i++) // Duyệt nửa dưới
            for (int j=0; j<i; j++)       // của ma trận kề
                  if (Có cạnh nối 2 đỉnh I và J)
                  {
                        Tạo cạnh v;
                        Gán đỉnh nguồn của v = i;
                        Gán đỉnh đích của v = j;
                        Gán trọng số của v = chi phi đi từ
                        đỉnh I đến đỉnh J;
                        Đánh dấu cạnh v chưa được xét;
                        Đưa cạnh v vào danh sách edgeList;
                        Tăng số cạnh (count) lên 1;
                  }
      return count;
}

// Sắp xếp danh sách cạnh tăng dần theo trọng số của cạnh
void QSortEdges(Edge edgeList[MAX], int d, int c)
{
      int i = d, j = c; // d = đầu, c = cuối

      // Giá trị ở giữa mảng
      CostType mid = edgeList[(d + c) / 2].Weight;
```

```
	// tiến hành tách mảng thành 2 phần
	while (i <= j)
	{
		// Tìm các ptu đứng sai vị trí trong mảng
		while (edgeList[i].Weight < mid) i++;
		while (edgeList[j].Weight > mid) j--;

		// Nếu có 2 ptu sai vị trí -> hoán vị chúng
		if (i <= j)
		{
			Edge temp = edgeList[i];
			edgeList[i] = edgeList[j];
			edgeList[j] = temp;
			i++;
			j--;
		}
	}
	// Sắp xếp mảng con bên phải mid
	if (i < c) QSortEdges(edgeList, i, c);

	// Sắp xếp mảng con bên trái mid
	if (d < j) QSortEdges(edgeList, d, j);
}

// Tìm nút gốc của cây chứa đỉnh x
int Find(int leader[MAX], int x)
{
	// chừng nào chưa tìm thấy gốc thì
	while (x != leader[x])
		x = leader[x];	// Chuyển đến nút cha
	return x;
}

// Hợp nhất 2 cây bằng cách nối thêm cạnh e
bool Union(int leader[MAX], Edge e)
{
	int x = Tìm nút gốc của cây chứa đỉnh e.Source;
	int y = Tìm nút gốc của cây chứa đỉnh e.Target;

	// Nếu trùng gốc => không thêm cạnh
	if (x == y)
		return false;
	else if (x < y)		// Nhập chung cây y vào cây x
		leader[y] = x;	// hay cây chứa y có gốc là x
	else
		leader[x] = y;	// Nhập chung cây x vào y
	return true;
}

// Thuật toán Kruskal tìm cây bao trùm tối thiểu.
// Kết quả được lưu trong mảng tree. Chỉ số mảng i
// là đỉnh nguồn, tree[i].Parent là đỉnh cuối và
// tree[i].Length là trọng số của cạnh được chọn.
void Kruskal(Graph g, Edge tree[UPPER])
{
	// Tạo ra danh sách các cạnh từ MT kề
	int ne = AdjMatrix2EdgeList(g, tree);

	// Sắp xếp các cạnh tăng dần theo trọng số
	QSortEdges(tree, 0, ne-1);
	// Khởi tạo đỉnh gốc của các cây con
	int leader[MAX];
	for (int i=0; i<g.NumVertices+1; i++)
		leader[i] = i;
```

```
	// Duyệt các cạnh tăng dần theo trọng số
	int count = 0;
	for (int i=0; i<ne; i++)
	{
		// Nếu có thể ghép nó vào cây bao trùm
		if (Union(leader, tree[i]))
		{
			tree[i].Marked = YES;	// Đánh dấu chon
			count++;		// Tăng biến đếm

			// Nếu đã chọn đủ n-1 cạnh cho cây
			if (count == g.NumVertices - 1)
				break;		// thì dừng
		}
	}
}
// Xuất danh sách các cạnh được chọn để tạo cây bao
// trùm nhỏ nhất theo thuật toán Kruskal
void PrintKruskalMST(Graph g, Edge tree[UPPER])
{
	cout << endl << "Cay bao trum gom cac canh sau:";
	CostType sum = 0;	// Lưu tổng chiều dài cây

	// Duyệt qua các cạnh
	for (int i=0; i<g.NumEdges; i++)
	{
		if (Cạnh I được chọn)
		{
			Xuất nhãn của đỉnh nguồn, nhãn của đỉnh
			đích và trọng số của cạnh nối 2 đỉnh đó;
			Cập nhật tổng chiều dài cây bao trùm;
		}
	}
	cout << endl << "Tong chieu dai cay bao trum la "
		<< sum;
}
```

Bài 2. Tìm đường đi ngắn nhất

a. Tìm đường đi ngắn nhất từ 1 đỉnh tới các đỉnh còn lại (thuật toán Dijkstra)

```
// Thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh nguồn
// source đến tất cả các đỉnh còn lại. Độ dài đường đi được
// lưu trong mảng labels, đường đi lưu trong mảng path.
void Dijkstra(Graph g, int source, Path road[MAX])
{
	CostType min;	// Lưu độ dài đường đi ngắn nhất
	int counter,	// Đếm số đỉnh đã xét
		minVertex,	// Lưu đỉnh có nhãn nhỏ nhất
		curr;	// Đỉnh hiện tại đang xét

	// Khởi gán giá trị cho các đoạn đường đi
	for (int i=0; i<Số đỉnh của g; i++)
	{
		road[i].Length = g.Cost[source][i];
		road[i].Parent = source;
	}

	Đánh dấu đỉnh source đã được xét;
	Gán nhãn cho đỉnh source = 0;
	counter = 1;		// Có 1 đỉnh đã xét
	curr = source;		// Bắt đầu từ nguồn

	// Trong khi chưa xét hết các đỉnh
```

```
	while (counter < Số đỉnh của g - 1)
	{
		min = INF;		// Giả sử min = vô cùng
		minVertex = curr; // Đỉnh min = đỉnh hiện tại

		// Duyệt qua từng đỉnh để kiểm tra
		for (int i=0; i< Số đỉnh của g; i++)
		{
			if (Đỉnh i chưa được xet)
			{
				// Gán lại nhãn cho các đỉnh
				if (road[i].Length >
						road[curr].Length + g.Cost[curr][i])
				{
					road[i].Length =
						road[curr].Length + g.Cost[curr][i];
					road[i].Parent = curr;
				}
				// Tìm đỉnh có nhãn nhỏ nhất
				if (min > road[i].Length)
				{
					min = road[i].Length;
					minVertex = i;
				}
			}
		}
		curr = minVertex; // Xét đỉnh có nhãn nhỏ nhất
		Đánh dấu đỉnh curr là đã xét;
		Tăng số đỉnh đã xét lên 1;
	}
}

// In ra đường đi từ đỉnh nguồn tới đỉnh đích target
void PrintPath(Graph g, Path road[MAX], int target)
{
	// Nếu chưa gặp đỉnh nguồn
	if (road[target].Parent != target)
		// thì tìm đường từ nguồn tới đỉnh trước target
		PrintPath(g, road, road[target].Parent);

	// Sau đó in ra nhãn của đỉnh trên đường đi
	cout << " --> " << g.Vertices[target].Label;
}
```

b. Tìm đường đi ngắn nhất giữa mọi cặp đỉnh (thuật toán Floyd)

```
// Tìm đđi ngắn nhất giữa mọi cặp đỉnh theo thuật toán Floyd
void Floyd(Graph g, Path route[MAX][MAX])
{
	int i, j, k;

	// Khởi tạo chiều dài đường đi giữa các cặp đỉnh
	for (i=0; i< Số đỉnh của g; i++)
		for (j=0; j< Số đỉnh của g; j++)
		{
			route[i][j].Length = g.Cost[i][j];
			route[i][j].Parent = i;
		}

	// Tính toán lại đường đi giữa các cặp đỉnh
	for (k=0; k< Số đỉnh của g; k++)
		for (i=0; i< Số đỉnh của g; i++)
			for (j=0; j< Số đỉnh của g; j++)
				// Nếu đường đi từ i->j qua đỉnh k
```

```
                        // ngắn hơn đường đi trực tiếp i->j
                        if (route[i][j].Length >
                  route[i][k].Length + route[k][j].Length)
      {
      // thì cập nhật lại độ dài đường đi giữa 2 đỉnh i & j
      route[i][j].Length=route[i][k].Length+route[k][j].Length;
      // Cập nhật đỉnh trung gian
      route[i][j].Parent = route[k][j].Parent;
      }
}

// Xuất đường đi ngắn nhất từ đỉnh source đến đỉnh target
// Ma trận road lưu độ dài đường đi ngắn nhất và đường đi
void FloydPath(Graph g, Path route[MAX][MAX],
                  int source, int target)
{
      if (route[source][target].Parent != target)
            FloydPath(g, route, source,
                        route[source][target].Parent);

      // Sau đó in ra nhãn của đỉnh trên đường đi
      cout << " --> " << g.Vertices[target].Label;
}
```

Bài 3. Kiểm tra chương trình và xem kết quả

```
#include <iostream>
#include <fstream>
#include <conio.h>

using namespace std;
#include "common.h"
#include "stack.h"//#include <stack>
#include "queue.h"//#include <queue>
#include "graph.h"

void main()
{
      Graph g = InitGraph(false);

      // ================================================
      // … Đoạn này xem trong bài Lab 3 …
      // ================================================

      cout << endl << endl << "========================";
      cout << endl << "Tim duong di ngan nhat theo thuat
                        toan Dijkstra" << endl;
      ResetFlags(g);

      Path road[MAX];
      Dijkstra(g, 0, road);

      for (int i=1; i<g.NumVertices; i++) {
            if (road[i].Length == INF)
            cout << endl << "Khong co duong di tu dinh "
                  << g.Vertices[0].Label << " den dinh "
                  << g.Vertices[i].Label;
            else
            {
            cout << endl << "Duong di ngan nhat tu dinh "
                  << g.Vertices[0].Label << " den dinh "
                  << g.Vertices[i].Label << " la " << endl;
            PrintPath(g, road, i);
            cout<<" : do dai = " << road[i].Length << endl;
```

```
		}
	}

	cout << endl << "=========================";
	cout << endl << "Tim duong di ngan nhat theo thuat
			toan Floyd" << endl;
	ResetFlags(g);

	Path route[MAX][MAX];
	Floyd(g, route);

	for (int i=1; i<g.NumVertices; i++)  {
		if (route[0][i].Length == INF)
		cout << endl << "Khong co duong di tu dinh "
			<< g.Vertices[0].Label << " den dinh "
			<< g.Vertices[i].Label;
		else
		{
		cout << endl << "Duong di ngan nhat tu dinh "
			<< g.Vertices[0].Label << " den dinh "
			<< g.Vertices[i].Label << " la " << endl;
		FloydPath(g, route, 0, i);
		cout<<" : do dai = "<<route[0][i].Length<<endl;
		}
	}

	cout << endl << "=========================";
	cout << endl << "Tim bao dong chuyen tiep" << endl;
	ResetFlags(g);
	Warshall(g, route);

	for (int i=0; i<g.NumVertices; i++) {
		for (int j=0; j<g.NumVertices; j++) {
			if (i == j) continue;
			cout << endl << g.Vertices[i].Label <<
				" --> " << g.Vertices[j].Label;
			if (route[i][j].Length == 0)
				cout << " : Khong co duong di";
			else
				cout << " : Co duong di";
		}
	}

	cout << endl << endl << "=========================";
	cout << endl << "Tim cay bao trum ngan nhat theo
			thuat toan Prim" << endl;
	ResetFlags(g);
	Prim(g, road);
	PrintPrimMST(g, road);


	cout << endl << endl << "=========================";
	cout << endl << "Tim cay bao trum ngan nhat theo
			thuat toan Kruskal" << endl;
	ResetFlags(g);
	Edge mst[UPPER];
	Kruskal(g, mst);
	PrintKruskalMST(g, mst);


	_getch();
}
```

VI. Bài tập

Bài 1. Mạng truyền thông

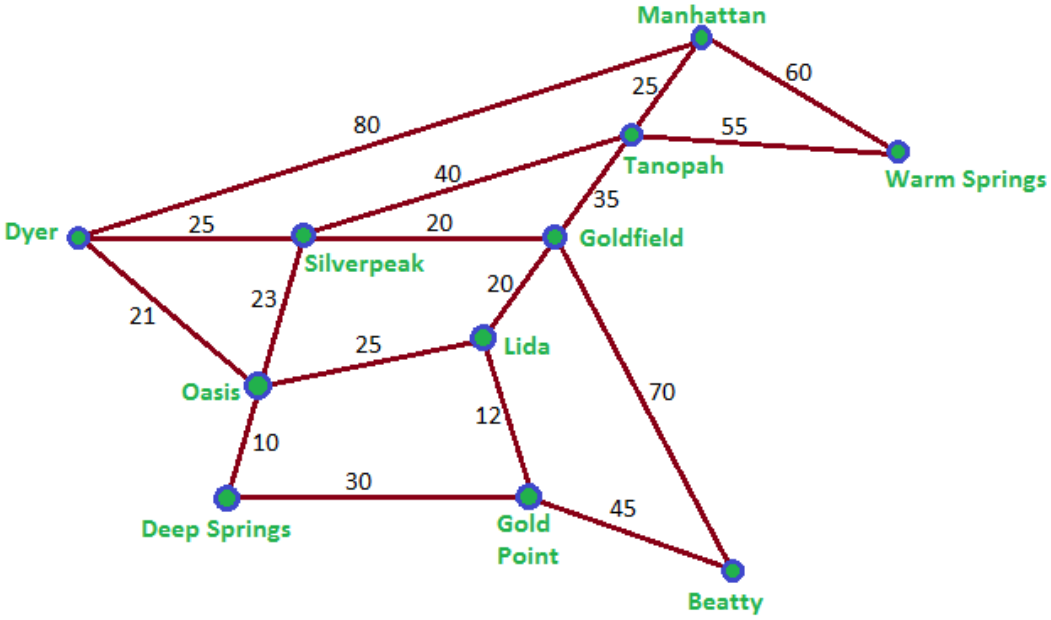
Một công ty lập kế hoạch xây dựng một mạng truyền thông nối năm trung tâm máy tính với nhau. Bất kỳ hai trung tâm nào cũng có thể được nối kết với nhau bằng đường điện thoại. Cần phải kết nối như thế nào để đảm bảo giữa hai trung tâm máy tính bất kỳ luôn có đường truyền thông sao cho tổng số tiền thuê bao của toàn mạng là tối thiểu? Phí thuê bao phải trả hàng tháng đối với các đường truyền thông được cho trong bảng sau:

Từ trung tâm	Đến trung tâm	Phí thuê bao
San Francisco	New York	$2000
San Francisco	Chicago	$1200
San Francisco	Denver	$900
San Francisco	Atlanta	$2200
Chicago	Denver	$1300

Từ trung tâm	Đến trung tâm	Phí thuê bao
Chicago	New York	$1000
Chicago	Atlanta	$700
New York	Denver	$1600
New York	Atlanta	$800
Denver	Atlanta	$1400

Bài 2. Ông Ngâu, Bà Ngâu.

Hẳn các bạn đã biết ngày "ông Ngâu bà Ngâu" hàng năm, đó là một ngày đầy mưa và nước mắt. Tuy nhiên, một ngày trước đó, nhà Trời cho phép 2 "ông bà" được đoàn tụ. Trong vũ trụ vùng thiên hà nơi ông Ngâu bà Ngâu ngự trị có ***N*** hành tinh đánh số từ 1 đến ***N***, ông ở hành tinh Adam (có số hiệu là S) và bà ở hành tinh Eva (có số hiệu là T). Họ cần tìm đến gặp nhau.



N hành tinh được nối với nhau bởi một hệ thống cầu vồng. Hai hành tinh bất kỳ có thể không có hoặc có duy nhất một cầu vồng (hai chiều) nối giữa chúng. Họ luôn đi tới mục tiêu theo con đường ngắn nhất. Họ đi với tốc độ không đổi và nhanh hơn tốc độ ánh sáng. Điểm gặp mặt của họ chỉ có thể là tại một hành tinh thứ 3 nào đó.

Yêu cầu: Hãy tìm một hành tinh sao cho ông Ngâu và bà Ngâu cùng đến đó một lúc và thời gian đến là sớm nhất. Biết rằng, hai ngươời có thể cùng đi qua một hành tinh nếu như họ đến hành tinh đó vào những thời điểm khác nhau.

Dữ liệu được cho trong file ongbangau.inp có cấu trúc như sau:

Dòng đầu là 4 số ***N M S T*** ($N \le 100$, $1 \le S \ne T \le N$), M là số cầu vồng. M dòng tiếp, mỗi dòng gồm hai số ***I J L*** thể hiện có cầu vồng nối giữa hai hành tinh ***I , J*** và cầu vồng đó có độ dài là ***L*** ($1 \le I \ne J \le N$, $0 < L \le 200$).

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2

LAB 7 – MỘT SỐ PHƯƠNG PHÁP THIẾT KẾ THUẬT TOÁN (4 TIẾT)

I. Mục tiêu

Sau khi thực hành, sinh viên cần:

- Nắm vững một số phương pháp thiết kế thuật toán: Chia để trị, quay lui, tham lam, quy hoạch động
- Minh họa được bài toán liên quan liên quan đến các phương pháp thiết kế.
- Vận dụng kiến thức đã học để giải một số bài toán thực tế.

II. Yêu cầu

- Sinh viên phải hoàn thành tối thiểu ***2 bài*** (bài 1 và 1 bài tùy chọn thuộc mục IV). Mỗi bài tạo một project, xóa các thư mục debug của project này. Sau đó chép cả 2 project vào thư mục: Lab7_CTK40_HoTen_MSSV_Nhom#. Nén thư mục, đặt tên tập tin nén theo dạng sau: Lab7_CTK40_HoTen_MSSV_Nhom#.rar.

Ví dụ: Lab7_CTK40_NguyenVanA_161111_Nhom4.rar.

- Sinh viên sẽ nộp bài Lab qua mạng tại phòng lab theo hướng dẫn của giáo viên.

III. Ôn tập lý thuyết

1. Phương pháp chia để trị :

Mô hình :

Nếu gọi D&C(ℜ) - Với ℜ là miền dữ liệu - là hàm thể hiện cách giải bài toán theo phương pháp chia để trị thì ta có thể viết :

```
void D&C(ℜ)
{
    If (ℜ đủ nhỏ)
      giải bài toán;
    Else
    {
      Chia  ℜ thành ℜ1, …, ℜm;
      for (i = 1; i <=m; i++)
            D&C(ℜi);
      Tổng hợp kết quả;
    }
}
```

2. Phương pháp quay lui

Mô hình :

Với n là số bước cần phải thực hiện, k là số khả năng mà x_i có thể chọn lựa, Try(i) là bước thử thứ i để xác định x_i

```
Try(i) ≡
    for ( j = 1 → k)
      If ( xi chấp nhận được khả năng j)
      {
            Xác định xi theo khả năng j;
            Ghi nhận trạng thái mới;
            if( i < n)
                  Try(i+1);
            else
                  Ghi nhận nghiệm;
            Trả lại trạng thái cũ cho bài
    toán;
            }
```

3. Phương pháp tham lam:

Mô hình :

```
Input A[1..n]
Output  S //lời giải;
greedy (A,n) ≡
S = ∅;
while ( A ≠ ∅)
{
      x= Chọn(A);
      A = A-{x}
      if( S∪ {x} chấp nhận  được )
```

4. Phương pháp quy hoạch động:

- Phương pháp quy hoạch động dựa vào một nguyên lý, gọi là nguyên lý tối ưu (The principle of optimality) của Bellman :*" Nếu lời giải của bài toán là tối ưu thì lời giải của các bài toán con cũng tối ưu".*
- Trong thuật toán quy hoạch động thường dùng các thao tác :

+ Xây dựng một hàm quy hoạch động (hoặc phương trình quy hoạch động).

+ Lập bảng lưu lại các giá trị của hàm.

S = S∪ {x}; } return S;	+ Truy xuất lời giải tối ưu của bài toán từ bảng lưu.

IV. Bài tập thực hành

Bài 1: (Chia để trị)

Bài toán Chia thưởng

Bài 2: (Quay lui)

Ngựa đi tuần

Bài 3 : (Phương pháp tham lam)

Bài toán tô màu

Bài 4: (Phương pháp quy hoạch động)

Thuật toán Floy xác định đường đi ngắn nhất giữa các cặp đỉnh

V. Bài tập

Bài 1. Mạng truyền thông

Bài 2. Ông Ngâu, Bà Ngâu.

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2

LAB 8 – BẢNG BĂM ĐỊA CHỈ MỞ

I. Mục tiêu

Sau khi thực hành, sinh viên cần:

- Nắm vững các phương pháp băm, hàm băm, phương pháp giải quyết va chạm (đụng độ khóa).
- Cài đặt được kiểu dữ liệu bảng băm địa chỉ mở và các thao tác, phép toán trên bảng băm.
- Vận dụng kiến thức đã học để giải một số bài toán thực tế.

II. Yêu cầu

- Sinh viên phải hoàn thành bài 1, bài 2 và bài 3 thuộc mục V, 3 bài này tạo chung một project. Xóa các thư mục debug của project này, sau đó chép các project vào thư mục: Lab8_CTK40_HoTen_MSSV_Nhom#. Nén thư mục, đặt tên tập tin nén theo dạng sau: Lab8_CTK40_HoTen_MSSV_Nhom#.rar.

Ví dụ: Lab6_CTK40_NguyenVanA_161111_Nhom4.rar.

- Sinh viên sẽ nộp bài Lab theo hướng dẫn của giáo viên.

III. Ôn tập lý thuyết

1. *Giới thiệu bảng băm*

Bảng băm là cấu trúc dữ liệu để cài đặt kiểu dữ liệu từ điển.

Kiểu dữ liệu từ điển là một tập các đối tượng dữ liệu được thao tác với 3 phép toán cơ bản:

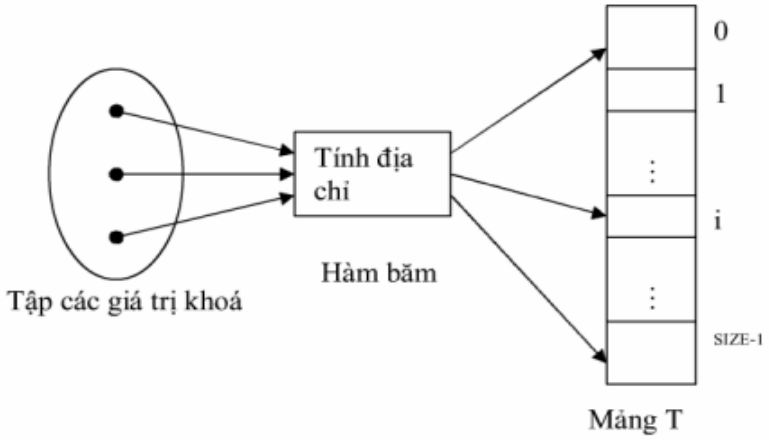
- Tìm kiếm
- Chèn
- Loại bỏ (xóa)

Dạng đơn giản nhất của bảng băm là một mảng các mẫu tin.

Mỗi mẫu tin có một trường dữ liệu đặc biệt gọi là khóa (key)

2. *Phương pháp băm*

Cho dữ liệu D có khóa K, tìm được chỉ số i sao cho 0 <= I <= SIZE-1. Khi đó, dữ liệu D sẽ được lưu trong thành phần thứ i của mảng (hay bảng băm).



Cần tìm một hàm Hash: index = Hash(key)

Giả sử K là tập giá trị khóa. Khi đó, hàm Hash là một ánh xạ từ tập giá trị khóa (K) vào tập các số nguyên {0, 1, …, SIZE-1}

Hash: K → {0, 1, …, SIZE-1}

Hash được gọi là hàm băm. Hash(key) được gọi là giá trị băm của khóa key. Dữ liệu sẽ được lưu tại vị trí Hash(key)

3. *Các hàm băm*

Trường hợp khóa là số nguyên

- Phương pháp chia

$$Hash(k) = k \% SIZE$$

Trong đó, SIZE là kích thước của bảng băm (thường là một số nguyên tố)

- Phương pháp nhân

$$hash(k) = \lfloor (\alpha k - \lfloor \alpha k \rfloor).SIZE \rfloor$$

Trong đó: $\lfloor \alpha k \rfloor$ là phần nguyên của x. Thông thường, α = 0.61803399

Trường hợp khóa là chuỗi

Bước 1: Chuyển xâu ký tự thành số nguyên

$$\text{"NOTE"} \rightarrow \text{'N'}.128^3 + \text{'O'}.128^2 + \text{'T'}.128 + \text{'E'} =$$

$$78.128^3 + 79.128^2 + 84.128 + 69$$

$$\text{"NOTE"} \rightarrow 78.37^3 + 79.37^2 + 84.37 + 69 =$$

$$((78.37 + 79).37 + 84).37 + 69$$

Bước 2: Áp dụng phương pháp chia (hoặc nhân) để tính giá trị băm

4. *Phương pháp giải quyết va chạm (đụng độ khóa)*

- Mỗi phần tử của mảng lưu trữ một dữ liệu
- Khi cần chèn một dữ liệu mới với khóa k, ta
  - Tìm giá trị băm của khóa k (chính là Hash(k))
  - Nếu tại vị trí Hash(k) trống, chèn dữ liệu vào bảng băm.
  - Nếu tại vị trí Hash(k) đã chứa dữ liệu, tiến hành thăm dò một vị trí trống để lưu dữ liệu.
- Phương pháp tiến hành thăm dò vị trí trống được gọi là phương pháp định địa chỉ mở.
- Giải pháp tổng quát:
  - Tìm chỉ số i = Hash(k) với k là khóa
  - Lần lượt xem xét các vị trí $i_0, i_1, i_2, \ldots, i_m, \ldots$ trong đó $i_0 = i$.
  - Dãy các vị trí $i_0, i_1, i_2, \ldots, i_m, \ldots$ được gọi là dãy thăm dò

Thăm dò tuyến tính

$$\begin{cases} i = hash(k) \\ i_m = (i + m) \% SIZE \end{cases}$$

Thăm dò bình phương

$$\begin{cases} i = hash(k) \\ i_m = (i + m^2) \% SIZE \end{cases}$$

Băm kép

$$\begin{cases} i = hash(k) \\ i_m = (i + m.h_2(k)) \% SIZE \end{cases}$$

IV. Hướng dẫn thực hành

1. *Tạo dự án*

Sinh viên có thể chọn cài đặt kiểu dữ liệu bảng băm theo ngôn ngữ C# hoặc C++. Các phần sau minh họa cách cài đặt, hướng dẫn bằng mã giả trên ngôn ngữ C++ (bên phải).

- Tạo dự án mới, đặt tên là Lab6_CTK38_HoTen_MSSV_Nhom#
- Trong thư mục Header Files, tạo ra 4 files sau:
  - datatype.h: Định nghĩa các hằng số và kiểu dữ liệu bảng băm
  - hashfunc.h: Định nghĩa các hàm băm
  - hashtable.h: Định nghĩa các thao tác trên bảng băm
  - utility.h: Định nghĩa các hàm tiện ích
- Trong thư mục Source Files, tạo tập tin: program.cpp

2. *Định nghĩa kiểu dữ liệu bảng băm (Cài đặt theo phương pháp định địa chỉ mở)*

- Nhấp đôi chuột vào file datatype.h, định nghĩa các hằng số và kiểu dữ liệu như sau:

```
//
============================
======================
// Định nghĩa hằng số

#define           EIGHT
      8
#define           PRIME
      7
#define           MAXSIZE
      1000
#define           PHI
      0.61803399
#define           EOL
      '\r\n'
#define           TAB
      '\t'

//
============================
======================
// Định nghĩa kiểu dữ liệu

typedef           int
      KeyType;
typedef           int
      DataType;

// Trạng thái của mỗi vị trí
trong bảng băm
enum  ItemState
{
      Empty,            //
Rỗng
      Active,           //
Đang có dữ liệu
      Deleted           // Dữ
liệu đã bị xóa
};

// Phương pháp thăm dò
enum  Probing
{
      Linear,           //
Thăm dò tuyến tính
      Quadratic,        //
Thăm dò bình phương
      Double            //
Băm kép
};

// Kiểu dữ liệu thể hiện một
phần tử của bảng băm
struct      HashEntry
{
      KeyType     Key;
      // Trường khóa
      DataType    Value;
      // Giá trị thực
      ItemState   Status;
      // Trạng thái
};

typedef HashEntry *EntryPtr;

// Kiểu dữ liệu bảng băm
struct
      OpenAddressHashTable
{
      int         Count;
            // Số ptử thực tế
      int         Capacity;
            // Số ptử tối đa
      EntryPtr*   Elements;
            // DSách phần tử
      Probing
      ProbingMethod;    //
Ppháp thăm dò
};

// Kiểu dữ liệu bảng băm
(dùng con trỏ)
typedef  OpenAddressHashTable
*HashTable;
```

- Trong tập tin program.cpp, nhập đoạn mã sau:

```
#include <iostream>
#include <conio.h>
#include <fstream>
```

```
using namespace std;

#include "utility.h"
#include "datatype.h"
#include "hashfunc.h"
#include "hashtable.h"
```

```
void main()
{
	_getch();
}
```

Vào menu Build > chọn Build Solution để biên dịch chương trình, kiểm tra lỗi. Nếu có lỗi, kiểm tra lại mã nguồn bạn đã viết. Nếu chương trình không có lỗi, ta sẽ cài đặt các thao tác, phép toán trên bảng băm.

V. Bài tập thực hành

Bài 1. Tạo các hàm tiện ích

a. Hàm kiểm tra số nguyên tố

```
// Hàm kiểm tra số n có phải là số nguyên tố
int IsPrime(unsigned int n)
{
	int sn = (int)sqrt((double)n);
	for (int i=2; i<sn; i++)
	{
		if (n % i == 0)
			return 0;
	}
	return 1;
}
```

b. Tìm số nguyên tố lớn hơn và gần nhất với N

```
// Tìm số nguyên tố lớn hơn và gần nhất với N
unsigned int FindNearestPrime(unsigned int n)
{
	if (n % 2 == 0) n++;
	while (!IsPrime(n))
		n += 2;
	return n;
}
```

Bài 2. Xây dựng các hàm băm và hàm thăm dò

a. Các hàm băm

```
// Hàm băm một giá trị số nguyên
unsigned int Hash(unsigned int
value,
                        unsigned
int tableSize)
{
      return value % tableSize;
}

// Hàm băm giá trị chuỗi
unsigned int Hash(const char
*value,
                        unsigned
int tableSize)
{
      int ind = 0;
      unsigned int sum = 0;

      while (value[ind] != NULL)
      {
            sum = (sum * 37) +
value[ind];
            sum = sum % tableSize;
            ind++;
      }
      return sum;
}

// Hàm băm thứ cấp, dùng cho băm
kép
unsigned int Hash2(unsigned int
value)
{
      return EIGHT - value % EIGHT;
}

// Hàm băm thứ cấp, dùng cho băm
kép
unsigned int Hash2(const char
*value)
{
      int ind = 0;
      unsigned int sum = 0;

      while (value[ind] != NULL)
      {
            sum = (sum * 37) +
value[ind];
            sum = sum % EIGHT;
            ind++;
      }
      return EIGHT - sum;
}

// Hàm băm thứ cấp, dùng cho băm
kép
unsigned int Hash3(unsigned int
value)
{
      return 1 + value % PRIME;
}
```

b. Hàm thăm dò

```
// Hàm thăm dò tuyến tính
unsigned int LinearProbing(unsigned
int curr,
            unsigned int step,
unsigned int tableSize)
{
      return (curr + step) %
tableSize;
}

// Hàm thăm dò bình phương
unsigned int SquareProbing(unsigned
int curr,
            unsigned int step,
unsigned int tableSize)
{
      return (curr + step * step) %
tableSize;
}

// Hàm thăm dò theo phương pháp băm
kép
unsigned int DoubleProbing(unsigned
int curr,
            unsigned int step,
KeyType key,
            unsigned int tableSize)
{
      unsigned int h2 = Hash2(key);
      return (curr + step * h2) %
tableSize;
}
```

Bài 3. Cài đặt các thao tác trên bảng băm

```
// Khai báo nguyên mẫu của hàm thay đổi kích thước bảng băm
void Resize(HashTable &ht, unsigned int newSize);

// Lưu dữ liệu vào bảng băm
void Update(EntryPtr item, KeyType k, DataType d)
{
	item->Key = k;
	item->Value = d;
	item->Status = Active;
}

// Tạo một phần tử mới cho bảng băm
EntryPtr NewEntry(KeyType k, DataType d)
{
	EntryPtr entry = new HashEntry;
	Update(entry, k, d);
	return entry;
}

// Khởi tạo một bảng băm rỗng
HashTable Initialize(unsigned int maxSize,

Probing probMethod)
{
	maxSize = FindNearestPrime(maxSize);

	HashTable table = new OpenAddressHashTable;
	table->Count = 0;
	table->Capacity = maxSize;
	table->ProbingMethod = probMethod;
	table->Elements = new EntryPtr[maxSize];

	for (int i=0; i<maxSize; i++)
		table->Elements[i] = NULL;
	return table;
}

// Kiểm tra bảng băm đã ở mức độ đầy 2/3
bool IsFull(HashTable ht)
{
	return (ht->Count / ht->Capacity * 1.0) > 0.66;
}

// Thăm dò
unsigned int DoProbing(HashTable ht, KeyType key,

unsigned int currPos,

unsigned int step)
{
	switch (ht->ProbingMethod)
	{
		case Linear:
			return LinearProbing(currPos, step, ht->Capacity);
		case Quadratic:
			return SquareProbing(currPos, step, ht->Capacity);
		case Double:
			return DoubleProbing(currPos, step, key, ht->Capacity);
	}
}

// Hàm tìm vị trí của phần tử chứa khóa key.
// Trả về chỉ số của phần tử cuối cùng được kiểm tra.
unsigned int Search(HashTable ht, KeyType key)
{
	unsigned int pos = Hash(key, ht->Capacity),
		index = pos, m = 1;

	while (ht->Elements[index] != NULL &&
		ht->Elements[index]->Key != key)
	{
		index = DoProbing(ht, key, pos, m);
		m++;
	}
	return index;
}

// Kiểm tra phần tử thứ pos trong bảng băm
// có chứa dữ liệu
bool IsActive(HashTable ht, unsigned int index)
{
	return (ht->Elements[index] != NULL) &&
		(ht->Elements[index]->Status == Active);
}

// Kiểm tra bảng băm có chứa dữ liệu với khóa key
bool Contains(HashTable ht, KeyType key)
{
	unsigned int index = Search(ht, key);
	return IsActive(ht, index);
}
```

```
// Thêm một phần tử mới vào bảng
băm
bool Insert(HashTable &ht, KeyType
key, DataType item)
{
	unsigned int index =
Search(ht, key);
	if (IsActive(ht, index))
		return false;
	else
	{
		ht->Count++;
		ht->Elements[index] =
NewEntry(key, item);

		if (IsFull(ht))
			Resize(ht, ht-
>Capacity * 2);

		return true;
	}
}

// Thay đổi giá trị của phần tử có
khóa key
bool Update(HashTable ht, KeyType
key, DataType item)
{
	unsigned int index =
Search(ht, key);
	if (IsActive(ht, index))
		return false;
	else
	{
		Update(ht-
>Elements[index], key, item);
		return true;
	}
}

// Thay đổi kích thước của bảng băm
void Resize(HashTable &ht, unsigned
int newSize)
{
	HashTable table =
Initialize(newSize, ht-
>ProbingMethod);

	for (int i=0; i<ht->Capacity;
i++)
		if (IsActive(ht, i))
		{
			Insert(table, ht-
>Elements[i]->Key, ht->Elements[i]-
>Value);
			delete ht-
>Elements[i];
		}
	delete ht;
	ht = table;
}

// Loại bỏ một phần tử có khóa key
void Remove(HashTable ht, KeyType
key)
{
	unsigned int index =
Search(ht, key);
	if (IsActive(ht, index))
	{
		//delete ht-
>Elements[index];
		//ht->Elements[index] =
NULL;
		ht->Elements[index]-
>Status = Deleted;
		ht->Count--;
	}
}

// Xuất nội dung của một phần tử
trong bảng băm
void PrintHashEntry(EntryPtr entry)
{
	cout << entry->Value;
}

// Xuất nội dung toàn bộ bảng băm
void PrintHashTable(HashTable ht)
{
	cout << endl << "Dung luong :
" << ht->Capacity;
	cout << endl << "So phan tu :
" << ht->Count << endl;
	cout << endl << "Du lieu cua
bang bam ";
	cout << endl << "INDEX" <<
TAB
			<< "KEY"
<< TAB
			<< "DATA";

	for (int i=0; i<ht->Capacity;
i++)
	{
		if (IsActive(ht, i))
		{
			cout << endl << i
<< TAB
				<< ht-
>Elements[i]->Key << TAB;

			PrintHashEntry(ht-
>Elements[i]);
		}
		else
			cout << endl << i
<< TAB
				<< '-' <<
TAB << '-';
	}
	cout << endl << endl;
}
```

Bài 4. Kiểm tra chương trình và xem kết quả

```
#include <iostream>
#include <fstream>
#include <conio.h>
#include <stdio.h>
#include <math.h>
#include <string>

using namespace std;

#include "utility.h"
#include "datatype.h"
#include "hashfunc.h"
#include "hashtable.h"

void main()
{
	HashTable h = Initialize(11, Quadratic);
	//KeyType key[] = { "monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday" };
	DataType data[] = {32, 15, 25, 44, 36, 21, 534, 702,
			105, 523, 959, 699, 821, 883, 842,
			686, 658, 4, 20, 382, 570, 344, 125,
			200, 217, 175, 153, 201};
	int n = 28;  // So phan tu cua mang data

	for (int i=0; i<n; i++)
	{
		cout << endl << "Sau khi chen them : "
			<< data[i] << endl;

		Insert(h, data[i], data[i]);
		PrintHashTable(h);
		_getch();
	}
	for (int i=0; i<n; i++)
	{
		cout << endl << "Sau khi xoa : "
			<< data[i] << endl;

		Remove(h, data[i]);
		PrintHashTable(h);
		_getch();
	}
	_getch();
}
```

Bài 5: Cài đặt kiểu dữ liệu bảng băm địa chỉ mở dùng để lưu trữ dữ liệu với khóa có kiểu chuỗi.

VI. Bài tập làm thêm

Bài 1. Thống kê từ trong tập tin văn bản

Các văn bản được lưu trữ thành từng dòng trong các file văn bản, mỗi dòng có chiều dài không quá 127 ký tự. Hãy đề xuất cấu trúc dữ liệu thích hợp để lưu trữ trong bộ nhớ trong của máy tính tên từ và số lần xuất hiện của từ đó trong tập tin văn bản. Với cấu trúc dữ liệu này, hãy trình bày thuật toán và cài đặt chương trình thực hiện việc thống kê xem các từ trong file văn bản xuất hiện với tần suất như thế nào? Cho biết văn bản có bao nhiêu từ, bao nhiêu tên từ?

Bài 2. Từ điển Anh – Việt

Cài đặt kiểu dữ liệu từ điển Anh-Việt bằng bảng băm.

=== HẾT ===