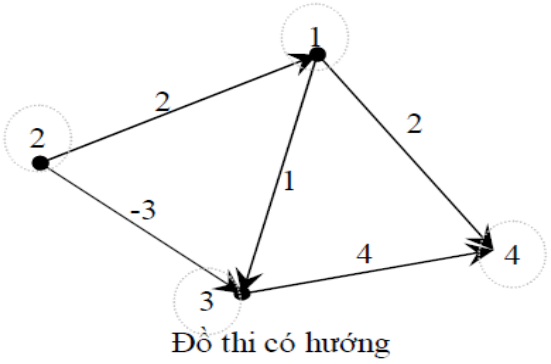
HƯỚNG DẪN THỰC HÀNH

TÌM ĐƯỜNG ĐI TRÊN ĐỒ THỊ

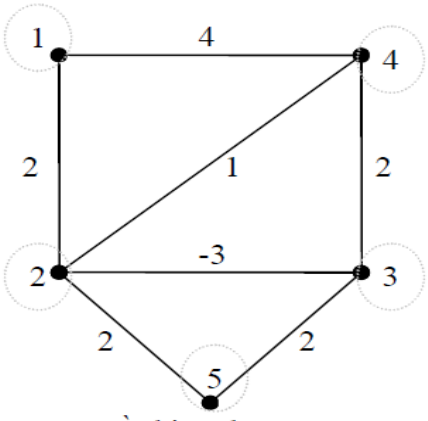
I. MA TRẬN KỀ

1. LÝ THUYẾT

Trong lý thuyết đồ thị, người ta thường dùng ma trận kề để biểu diễn một đồ thị. Một giá trị a[i,j] trong ma trận (dòng i cột j của ma trận A) ứng số cạnh/trọng số của cạnh giữa hai đỉnh i và j trong đồ thị. Nếu giữa 2 đỉnh của đồ thị không có cung thì phần tử a[i, j] =0



Đồ thi có hướng



Đồ thi vô hướng

Theo định nghĩa trên thì đồ thị có hướng ở trên sẽ có ma trận kề là

	1	**2**	**3**	**4**
1	0	0	1	2
2	2	0	-3	0
3	0	0	0	4
4	0	0	0	0

Đối với đồ thị vô hướng, ta sẽ có giá trị a[*i*,*j*] = a[*j*,*i*] (xem như có hai cung có hướng từ *i* → *j* và từ *j* → *i* trong đồ thị). Ma trận kề của đồ thị vô hướng ở trên là:

	1	**2**	**3**	**4**	**5**
1	0	2	0	4	0
2	2	0	-3	1	2
3	0	-3	0	2	2
4	4	1	2	0	0
5	0	2	2	0	0

Nhận xét:

- Đường chéo chính trong đồ thị có giá trị 0 (do trong đồ thị không có khuyên).
- Ma trận kề của đồ thị vô hướng đối xứng qua đường chéo chính vì a[*i*,*j*] = a[*j*,*i*].

2. CÀI ĐẶT

Lưu trữ trên tập tin

Thông thường ma trận kề được lưu trữ trên tập tin và được chương trình đọc lên để thực hiện các thuật toán trên đồ thị tương ứng.

Ví dụ: đồ thị có hướng ở trên sẽ được lưu trong tập tin input.txt như sau:

```
4
0   0   1   2
2   0   -3  0
0   0   0   4
0   0   0   0
```

Tổ chức lớp

Xây dựng lớp Matrix cài đặt **toán tử** >> và << (cho việc nhập/xuất ma trận) cùng các phương thức cần thiết khác.

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

class Matrix
{
public:
    int size;   // kích thước của ma trận
    int** weights; // Ma trận trọng số

    Matrix(void);       // Hàm khởi tạo mặc định
    Matrix(int size);   // Hàm khởi tạo ma trận với kích thước xác định

    ~Matrix(void);      // Hàm hủy ma trận

    // Nhập giá trị cho ma trận(bàn phím hoặc file)
    friend istream& operator >>(istream& inDevice, Matrix& a);
    // Xuất giá trị ma trận(màn hình hoặc file)
    friend ostream& operator <<(ostream& outDevice, Matrix& a);

    // Các phương thức khác
    //...

};
```

- ***Đọc/ghi ma trận kề***

Đọc và ghi ma trận kề được thực hiện thông qua toán tử >> và <<. Dưới đây là đoạn mã tham khảo cho các toán tử:

```
istream& operator >>(istream& inDevice, Matrix& a)
{
    int temp;
    for(int i = 0; i < a.size; i++){
        for(int j = 0; j < a.size; j++){
            a.weights[i][j] = 0;
            if(inDevice != NULL){
                inDevice >> temp;
                a.weights[i][j] = temp;
            }
        }
    }
    return inDevice;
}

ostream& operator <<(ostream& outDevice, Matrix& a)
{
    for(int i = 0; i < a.size; i++){
        for(int j = 0; j < a.size; j++){
            outDevice << (int)a.weights[i][j] << " ";
        }
        outDevice << endl;
    }
    return outDevice;
}
```

Để nhập giá trị cho ma trận M bằng bàn phím và xuất kết quả vừa nhập ra màn hình, ta thực hiện như sau:

```
int _tmain(int argc, _TCHAR* argv[])
{
    // Tạo một ma trận có kích thước là 3 x 3.
    int n = 3;
    Matrix M(n);
    // Nhập giá trị cho ma trận bằng bàn phím bằng toán tử >>
    cout << "Nhap gia tri cua ma tran" << endl;
    cin >> M;

    // Xuất ma trận vừa nhập ra màn hình bằng toán tử <<
    cout << "Ma tran vua nhap la: " <<endl;
    cout << M;

    return 0;
}
```

Trong trường hợp giá trị của ma trận được lưu trong tập tin với nội dung sau:

```
3                       // kích thước của ma trận
0  4      1             // Giá trị trọng số
4  0      2
1  2      0
```

Thì việc đọc ma trận được thực hiện như sau, trong đó "input.txt" là tên của tập tin cần đọc:

```
// Mở tập tin cần đọc
ifstream inputStream("input.txt");
// Đọc kích thước của ma trận
int n;
inputStream >> n;
// Khởi tạo ma trận
Matrix M(n);
// Đọc các trọng số
inputStream >> M;
```

Và để kết xuất giá trị của ma trận xuống tập tin, ta làm như sau:

```
ofstream outputStream("output.txt");
outputStream << M;
```

Lưu ý: *Trên đây chỉ là giới thiệu một cách để anh/chị đọc ghi ma trận (thông qua toán tử << và >>) và không bắt buộc phải tuân theo. Anh/chị có thể sử dụng cách khác quen thuộc với mình, chẳng hạn sử dụng các hàm như printf, scanf, fprintf, fscanf…để đọc/ghi ma trận.*

Kiểm tra tính hợp lệ của ma trận

Sau khi đã lấy dữ liệu từ tập tin, chúng ta cần kiểm tra dữ liệu này có hợp lệ không? Tùy thuộc vào loại đồ thị mà ta có cách kiểm tra ma trận kề phù hợp. Chẳng hạn với đồ thị vô hướng đơn không khuyên thì ta kiểm tra các giá trị a[i][i] xem có giá trị nào khác 0 hay không hoặc a[i][j] xem có giá trị nào > 1 hay không, nếu có thì ma trận kề không hợp hợp lệ.

Như vậy, ta có thể xây dựng một lớp đối tượng Graph để thể hiện đồ thị. Trong đó, cài đặt phương thức kiểm tra tính hợp lệ của ma trận kề.

```
class Graph
{
    //lưu trữ ma trận kề của đồ thị
    Matrix adjacentMatrix;

    //các phương thức khác
    //...

    //Kiểm tra tính hợp lệ của ma trận kề
    bool IsAdjacentMatValid();
};
```

II. TÌM KIẾM THEO CHIỀU SÂU - DFS

1. THUẬT TOÁN

Dưới đây là chi tiết thuật toán DFS, tìm đường đi từ đỉnh s đến đỉnh g.

1. Tại thời điểm khởi động tất cả các đỉnh chưa viếng thăm ($nhan[i] = false$).
2. Khởi tạo đường đi rỗng.
3. Viếng thăm đỉnh s

Việc viếng thăm một đỉnh v được thực hiện như sau:

Nếu $v == g$ → đã tìm thấy đường đi

Ngược lại

Đánh dấu đã viếng thăm v ($nhan[v] = true$)

Viếng thăm các đỉnh kề với v mà chưa được viếng thăm.

2. HƯỚNG DẪN CÀI ĐẶT

Như đã đề cập trong phần thuật toán, trước tiên chúng ta cần cài đặt hàm ViengTham một đỉnh bất kì trong lớp Graph.

Để lưu vết đường đi trong quá trình duyệt, sử dụng mảng một chiều ($dinhTruoc$). Trong đó phần tử thứ i cho biết đỉnh đứng trước đỉnh i trong đường

đi là đỉnh nào. Ví dụ, $dinhtruoc[2] = 4$, nghĩa là trước khi đi qua đỉnh 2 thì đi qua đỉnh 4.

Dưới đây là một giới thiệu cài đặt:

- ***Hàm viếng thăm một đỉnh.***

```
bool Graph:: ViengTham(int *nhan, int v, int g, int *dinhTruoc){

  if(v == g) return true;

  // Đánh dấu đã viếng thăm v
  nhan[v] = ...;

  // Duyệt qua các đỉnh kề với đỉnh v mà chưa viếng thăm
  for (int i = 0; i < soDinh; i++){
        if(... && ...){

              // Cập nhật đỉnh trước đỉnh i
              dinhTruoc[i] = ...;

              // Viếng thăm đỉnh i (lưu ý thoát sớm khi tìm thấy
đường đi)
              ViengTham(nhan, i, g);
        }
  }

}
```

Lưu ý rằng, sinh viên có thể dùng ngăn xếp để khử đệ quy cho hàm ViengTham

- ***Hàm tìm đường đi theo DFS.***

```
void Graph:: DFS(int *nhan, int v, int g){

  // Khởi tạo nhan
  int* nhan = NULL;
  ...

  // Khởi tạo đường đi
  int* dinhTruoc = new int[soDinh];
  for (int i = 0; i < soDinh; i++)
  {
        dinhTruoc[i] = -1; // không có đỉnh trước.
  }
  // Viếng thăm đỉnh s
  ...
}
```

III. TÌM KIẾM THEO CHIỀU RỘNG – BFS

1. THUẬT TOÁN

Dưới đây là chi tiết thuật toán BFS, tìm đường đi từ đỉnh s đến đỉnh g.

```
1  Khởi tạo các đỉnh chưa được viếng thăm
2  Khởi tạo đường đi rỗng
3  Khởi tạo hàng đợi queue
4  Thêm s vào hàng đợi queue
5  Trong khi queue ≠ ∅, thực hiện{
       v = Dequeue (queue)
       Đánh dấu viếng thăm v
       Với những đỉnh i kề với v mà chưa được viếng thăm{
            Lưu vết đường đi cho đỉnh  i
            Thêm i vào hàng đợi queue
       }
       }
```

2. HƯỚNG DẪN CÀI ĐẶT

Sinh viên tham khảo phần DFS và tự cài đặt.

IV. THÔNG TIN ĐƯỜNG ĐI TỪ MẢNG LƯU VẾT ĐƯỜNG ĐI.

Bắt đầu từ phần tử thứ g, xác định đỉnh trước đỉnh g (tạm gọi là x)

Tiếp tục với phần tử thứ x, xác định đỉnh trước đỉnh x (tạm gọi là y).

....

Dừng khi gặp đỉnh bắt đầu s hoặc không tìm thấy đỉnh trước tương ứng (-1)