Đường đi và Liên thông

Nội dung

- Đường đi
- Tính liên thông của đồ thị
- Bài tập

Đường đi

Input: Đồ thị g, hai đỉnh s, d

Output: Một đường đi từ s đến d trên đồ thị g, nếu có.

Dùng mảng parent[] để lưu vết của đường đi.

Action:

+ Duyệt đồ thị xuất phát từ đỉnh s:

Trong quá trình duyệt, mỗi khi từ đỉnh v thăm đỉnh kề w thì ta lưu parent[w] = v.

+ Nếu đỉnh d được thăm thì có đường đi từ s đến d. Ngược lại thì không có đường đi từ s đến d.

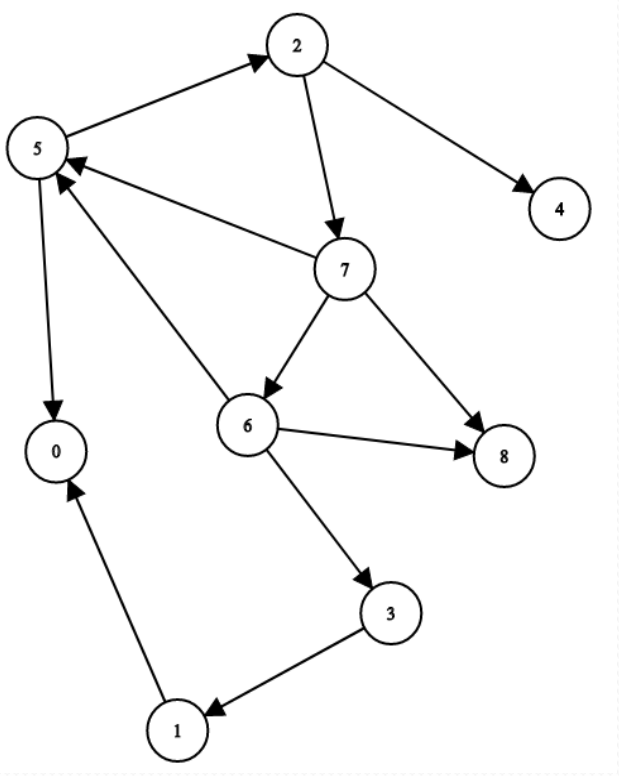
+ Đường đi từ s đến d được xác định qua mảng parent[] như sau:

d <= v1 = parent[d] <= v2 = parent[v1] <= … <= s = parent[vk]

Ví dụ

Tìm đường đi từ đỉnh 6 đến đỉnh 0.

Duyệt theo chiều sâu xuất phát từ 6:



vistied

true	**true**	**true**	**true**	**true**	**true**	**true**	**true**	**true**
0	1	2	3	4	5	6	7	8

parent

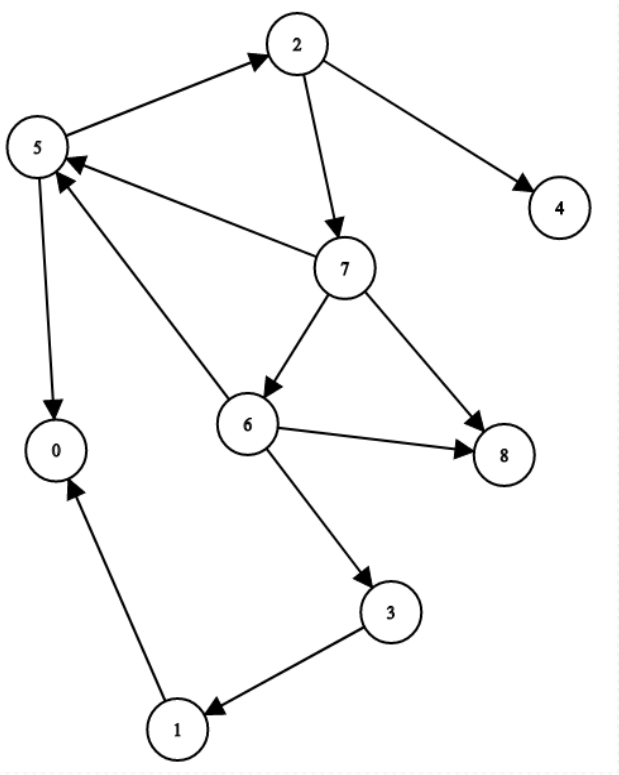
1	**3**	**5**	**6**	**2**	**6**	**-1**	**2**	**7**
0	1	2	3	4	5	6	7	8

0 ← parent[0] = 1 ← parent[1] = 3 ← parent[3] = 6

Ví dụ

Tìm đường đi từ đỉnh 6 đến đỉnh 0.

Duyệt theo chiều rộng xuất phát từ 6:



vistied

true	**true**	**true**	**true**	**true**	**true**	**true**	**true**	**true**
0	1	2	3	4	5	6	7	8

parent

5	**3**	**5**	**6**	**2**	**6**	**-1**	**2**	**6**
0	1	2	3	4	5	6	7	8

0 ← parent[0] = 5 ← parent[5] = 6

Nhận xét

- Dùng thuật toán duyệt đồ thị xuất phát từ đỉnh s có thể tìm được đường đi từ đỉnh s đến các đỉnh của đồ thị (nếu có).
- Nếu dùng thuật toán duyệt đồ thị theo chiều rộng sẽ tìm được đường đi qua ít cạnh nhất (nếu có).

Cài đặt

```
void path(AdjMatrixGraph g, int s, bool vistied[], int
parent[]){
  for(int i = 0; i < g.numVertices; i++){
       visted[i] = false;
       parent[i] = -1;
  }
  DFSPath(g, s, visted, parent);
}
```

```
void DFSPath(AdjMatrixGraph g, int v, bool visted[], int
parent[]){
  visted[v] = true;
  for(w = 0; w < g.numVertices; w++)
    if(g.adjMatrix[v][w] == 1 && !vistied[w]){
       parent[w] = v;
       DFSPath(g, w, visted, parent);
    }
}
```

```
void printPath(int s, int d, int parent[])
{
        int z = d;
        while(z != s)
        {
                cout << z << “<==”;
                z = parent[z];
        }
        cout << s << endl;
}
```

Tính liên thông

- Thuật toán kiểm tra đồ thị vô hướng liên thông

Input: Đồ thị vô hướng g cần kiểm tra tính liên thông

Output: True nếu đồ thị liên thông, False nếu ngược lại

Action:

Duyệt đồ thị xuất phát đỉnh bất kỳ của đồ thị

Nếu tất cả các đỉnh được thăm thì trả về True

Ngược lại thì trả về False

Cài đặt

```
bool isConnected(AdjMatrixGraph g)
{
  bool visited[g.numVertices] = {false};
  DFSConnected(g, 0, visited);
  for(int i = 0; i < g.numVertices; i++)
      if (!visted[i]) return false;
  return true;
}
```

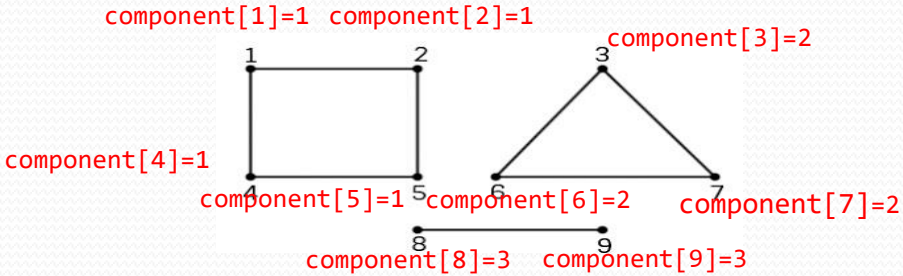
```
void DFSConnected(AdjMatrixGraph g, int v, bool
visited[]){
  visited[v] = true;
  for(int w = 0; w < g.numVertices; w++)
      if(g.adjMatrix[v][w] == 1 && !vistied[w]){
            DFSConnected(g, w, visited);
      }
}
```

Đồ thị có hướng

Input: Đồ thị có hướng g cần kiểm tra tính liên thông mạnh
Output: True nếu đồ thị liên thông mạnh, False nếu ngược lại
Action:
Duyệt đồ thị g xuất phát từ đỉnh v0 bất kỳ
Nếu có đỉnh chưa thăm thì trả về false
Ngược lại
    Duyệt xuất phát đỉnh v0 trên đồ thị g' ngược với đồ thị g
    Nếu trên đồ thị g' có đỉnh chưa thăm thì trả về false
    Ngược lại thì trả về true

Các thành phần liên thông

- Dùng 1 mảng component[] để lưu thành phần liên thông của từng đỉnh. component[i]=k có nghĩa đỉnh i thuộc thành phần liên thông thứ k.



Thuật toán

- Input: đồ thị g
- Output: các thành phần liên thông của đồ thị g
- Action:
- k = 0
- Duyệt lần lượt các đỉnh thứ i của đồ thị g:
  - Nếu đỉnh i chưa thăm thì:
    - Tăng k lên 1
    - Duyệt đồ thị xuất phát từ i: mỗi khi thăm đỉnh v thì gán component[v]=k

Cài đặt

```
int connectedComponents(AdjMatrixGraph graph, int component[]){
  bool visited[MAX_VERTICES] = { false };
  int k = 0;
  for (int i = 0; i < graph.numVertices; i++){
    if (!visited[i]){
      k++;
      DFSConnectedComponents(graph, i, visited, component, k);
    }
  }
  return k;
}
```

Cài đặt

```
void DFSConnectedComponents(AdjMatrixGraph graph, int v, bool
visited[], int component[], int k){
  visited[v] = true; component[v] = k;
  for (int w = 0; w < graph.numVertices; w++){
    if (graph.adjMatrix[v][w]==1 && !visited[w])
      DFSConnectedComponents(graph, w, visited, component, k);
  }
}
```

Cài đặt

```
void printConnectedComponents(int component[], int n, int k){
 cout << "Num of connected components is: " << k << endl;
 for(int i = 1; i <= k; i++){
   cout << "Component[" << i << "]: ";
   for(int j = 0; j < n; j++)
    if (component[j]==i)
      cout << j << " ";
    cout << endl;
   }
   cout << endl;
}
```

Bài tập

1. Tìm đường đi từ đỉnh x đến đỉnh y qua các cạnh có trọng số >=M
2. Tìm đường đi từ đỉnh x đến đỉnh y không qua đỉnh z.
3. Cài đặt thuật toán kiểm tra đồ thị có hướng liên thông mạnh.
4. Trong đồ thị "biết nhà" hãy chỉ một cách hỏi nhà để người a tìm được đến nhà người b mà số người cần đến hỏi là ít nhất.
5. Trong đồ thị tiếp xúc gần, biết vo là người Fo, cho k là một số nguyên dương. Liệt kê tất cả những người F_i với o<= i <=k bằng cách duyệt theo chiều sâu và theo chiều rộng.

Bài 1

- Hướng dẫn: khi duyệt đồ thị, từ v qua w nếu trọng số cạnh (v, w)>=M

```
void DFSPathM(AdjMatrixGraph g, int v, bool visited[],
int parent[]){
  visited[v] = true;
  for(int w = 0; w <g.numVertices; w++)
    if(g.adjMatric[v][w]!=VC && !visited[w] &&
g.adjMatrix[v][w]>=M){
      parent[w] = v;
      DFSPathM(g, w, visited, parent);
  }
}
```

Bài 2

- Hướng dẫn: đánh dấu đỉnh z đã thăm trước khi duyệt đồ thị để tìm đường đi.

Tổng kết

- Một đồ thị vô hướng có 1 hoặc nhiều thành phần liên thông.
- Thuật toán tìm đường đi bằng cách duyệt chỉ tìm được 1 đường đi.
- Muốn tìm tất cả các đường đi phải dùng thuật toán vét cạn.