1. **经过图中所有顶点的简单路径**

哈密顿路径是一种经过图中所有顶点恰好一次的路径。哈密顿路径在图中不一定存在，且寻找哈密顿路径是一个NP-完全问题，意味着没有已知的多项式时间算法可以解决所有情况‌1。

**算法实现**

**深度优先搜索（DFS）**‌：可以从任一顶点开始，使用深度优先遍历算法尝试构建哈密顿路径。如果在遍历过程中发现无法继续，则回溯并尝试下一个顶点作为起点。如果所有顶点都尝试过且未能找到路径，则说明图中不存在哈密顿路径‌。

‌**回溯法**‌：使用邻接矩阵存储图的信息，通过递归的方式尝试所有可能的路径组合，直到找到一条经过所有顶点的路径或者确认不存在这样的路径‌。

#include <iostream>

#include <vector>

using namespace std;

const int MaxSize = 100;

class Graph

{

public:

Graph(int n, int e);

void path\_len\_n(int u);

private:

void dfs\_path\_len\_n(int u, vector<int>path, bool visited[]);

int edge[MaxSize][MaxSize];

int vertexNum, edgeNum;

};

Graph::Graph(int n, int e) {

vertexNum = n;

edgeNum = e;

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

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

edge[i][j] = 0;

}

}

pair<int, int> data[15] = { {0,1},{1,2},{0,5},{2,5},{2,3},{5,7},{5,6},{6,7},{5,4},{6,4},{4,8},{3,8} };

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

int u = data[i].first, v = data[i].second;

edge[u][v] = edge[v][u] = 1;

}

}

void Graph::path\_len\_n(int u) {

bool visited[MaxSize];

memset(visited, 0, sizeof(visited));

vector<int> path;

path.push\_back(u);

visited[u] = 1;

dfs\_path\_len\_n(u, path, visited);

}

void Graph::dfs\_path\_len\_n(int u, vector<int>path, bool visited[]) {

**if(path.size()==vertexNum){**

cout<<"[";

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

cout<<path[i];

if(i!=path.size()-1) cout<<", ";

}

cout<<"]"<<endl;

return;

}

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

if(edge[u][i]&&!visited[i]){

**visited[i]=1;**

path.push\_back(i);

dfs\_path\_len\_n(i,path,visited);

**visited[i]=0;**

path.pop\_back();

}

}

}

int main() {

int start;

cin >> start;

Graph g(9, 12);

g.path\_len\_n(start);

}

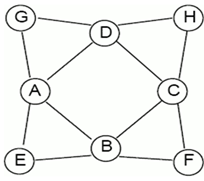
|  |  |
| --- | --- |
| **2** | **图的遍历及连通性 - C++** |

略

参见PPT调用DFS次数的实现

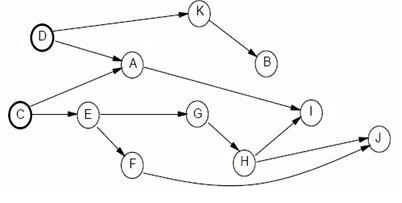
简答题

1. **图-dfs遍历序列和bfs遍历序列**



**DFS:ABCDGHFE**

**BFS:ABDEGCFH**



**DFS:AIBCEFJGHDK**

**BFS:AIBCEFGJHDK**

**2.无向图是否为树**

**设计算法，判断邻接矩阵表示的无向图是否为树。**

**判断n个结点是否有n-1条边**

**如果是，继续判断是否是连通的，即调用一次dfs全部访问完结点**

bool MGraph<DataType> :: is\_tree(){

    int visited[MaxSize];

     if (edgeNum != vertexNum-1)

  return false;  
    for (i = 0; i < vertexNum; i++)

  visited[i] = 0;

   dfs(visited, 0);  
    for (i = 0; i < vertexNum; i++)

  if (visited[i] == 0)

          return false;  
  return true;

}

void MGraph<DataType> :: dfs(int[] visited,int v)

{

      visited[v] = 1;

      for (int j = 0; j < vertexNum; j++)

           if (edge[v][j] == 1 && visited[j] == 0)

                dfs(visited, j );

}

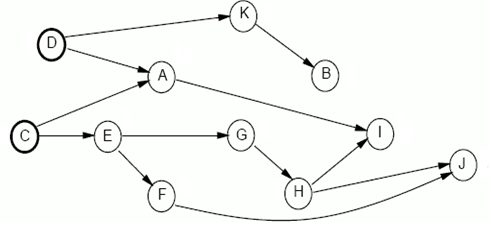
|  |  |
| --- | --- |
| **3** | **用prim算法求最小生成树** |



|  |  |
| --- | --- |
| **4** | **Dijkstra算法求解单源点最短路径** |



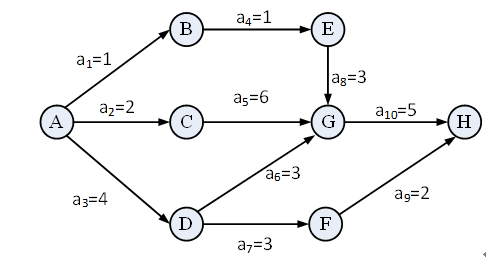
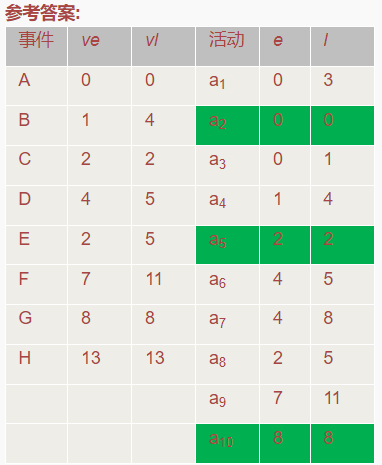
|  |  |
| --- | --- |
| **5** | **求解拓扑序列** |



深度优先拓扑序列：DKBCEGHFJAI

广度优先拓扑序列：CDEAKFGBHIJ

|  |  |
| --- | --- |
| **6** | **求出关键路径及关键活动** |



关键活动a2,a5,a10

关键路径：ACGH

选择题

1. C

2. B

3. C

4. B|E