# 数据结构 A
# 第八章第一次作业参考答案

题 目 范 围 ：图
授 课 老 师 ：何璐璐
助 教 ：王思怡（撰写人），赵守玺
联 系 邮 箱 ：2021302111095@whu. edu. cn

## 目录

# 1. 单选题

**题目1**

在一个无向图中，所有顶点的度之和等于边数的 _____ 倍

A.1/2

B.1

C.2

D.4

**答案：C**

**解析：**每条边连接两个顶点，因此每条边对应顶点的度数为2，因此所有顶点的度数之和等于所有边数的2倍。

**题目2**

采用邻接表存储的图的广度优先遍历算法类似于二叉树的 _____ 算法

A.先序遍历

B.中序遍历

C.后序遍历

D.层次遍历

**答案：D**

**解析：**广度优先搜索时会依次访问同一表结点的所有节点，所以类似于依次访问同一层的所有节点，也就是类似于层次遍历。

**题目3**

以下关于广度优先遍历的叙述正确的是 _____

A.广度优先遍历不适合有向图

B.对任何有向图调用一次广度优先遍历算法便可访问所有的顶点

C.对一个强连通图调用一次广度优先遍历算法便可访问所有的顶点

D.对任何非强连通图需要多次调用广度优先遍历算法才可访问所有的顶点

**答案：C**

**解析：**

定义：强连通图（Strongly Connected Graph）是指在有向图$G$中，如果对于每一对$v_i$、$v_j$，$v_i \neq v_j$，从$v_i$到$v_j$和从$v_j$到$v_i$都存在路径，则称$G$是强连通图；广度优先遍历是从一个顶点开始，逐层访问未访问的邻接点

A. 错误。广度优先遍历（BFS）适用于有向图和无向图，且在有向图中可以帮助确定从一个顶点可以到达哪些其他顶点。

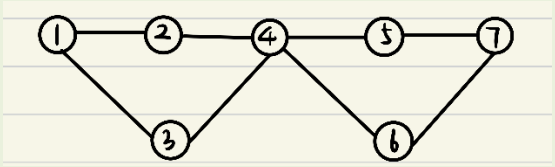
B. 错误。如果有向图不是强连通的，则一次广度优先遍历可能无法访问所有顶点。例如，如果图中存在不可达的孤立子图，那么从主连通组件开始的遍历不会覆盖这些孤立顶点。

C. 正确。强连通图意味着图中任何一个顶点都可以到达其他所有顶点。在这种情况下，从任何一个顶点开始的广度优先遍历都能访问图中的所有顶点。

D. 错误。可以单次调用，如1->2->3->4，不是强连通图，但从1开始遍历就可以访问所有顶点。

**题目4**

对于图所示的无向图，从顶点1开始进行深度优先遍历，可得到顶点访问序列是_____




A.1243576

B.1243567

C.1234576

D.1234567

**答案：A**

**解析：**图示是助教本人手绘，因为我的平台显示貌似有点怪. . 解题参考书籍即可。

**题目5**

对某个带权连通图构造最小生成树，以下说法正确的是 _____

一，该图的所有最小生成树的总代价一定是唯一的

二，其所有权值最小的边一定会出现在所有的最小生成树中

三，用Prim算法从不同顶点开始构造的所有最小生成树一定相同

四，使用Prim算法和Kruskal算法得到的最小生成树总不相同

A.一

B.二

C.一，三

D.二，四

**答案： A**

**解析：**

一、正确。最小生成树的定义是在一个带权连通图中找到一棵包括所有顶点的树，使得其所有边的权重之和最小。不同的最小生成树的结构可能不同，尤其是在权重相同的边选择上，但总权重一定是相同的，因为它们都代表了该图的最小权重和。

二、错误。最小的边有可能出现在所有最小生成树中，但这不是必然的。例如，如果有几条边具有相同的最小权重，且它们形成了一个环，那么可能只选择这些最小边中的一部分来构造最小生成树，以避免形成环路。

三、错误。 Prim算法从不同的顶点开始可能会得到不同的最小生成树，特别是当有多条边具有相同权重时。选择的起始顶点可能影响哪条边被首先选中，尤其在存在权重相等的边时，最终可能构造出结构不同的最小生成树。

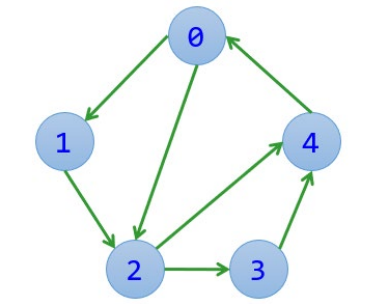
四、错误。 Prim算法和Kruskal算法都旨在找到最小生成树，这两种算法可能得到结构上不同的树，但这并不是必然的，它们完全有可能得到结构相同的最小生成树。

故选A。

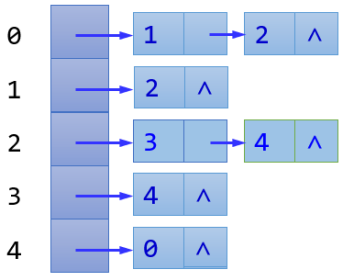
# 2. 简答题

**题目：图的存储和遍历**

画出下图的邻接表存储结构，并在该邻接表上调用dfs算法找出从顶点0到顶点4的所有路径（按顺序输出）。




**答案：**




**路径1：0，1，2，3，4**

**路径2：0，1，2，4**

**路径3：0，2，3，4**

**路径4：0，2，4**

# 3. 编程题

## 3.1. 图的遍历及连通性

【问题描述】

根据输入的图的邻接矩阵A，判断此图的连通分量的个数。

【输入形式】

第一行为图的结点个数n，之后的n行为邻接矩阵的内容，每行n个数表示。其中A[i][j]=1表示两个结点邻接，而A[i][j]=0表示两个结点无邻接关系。

【输出形式】

输出此图连通分量的个数。

【样例输入】

5
0 1 1 0 0
1 0 1 0 0
1 1 0 0 0
0 0 0 0 1
0 0 0 1 0

【样例输出】

2

【样例说明】

邻接矩阵中对角线上的元素都用0表示。(单个独立结点，即与其它结点都没有边连接，也算一个连通分量)

【评分标准】

要求必须使用图的广度或者深度优先遍历算法，否则不得分。

**参考答案：**

```
#include<stdio.h>
int count=0;
//计算连通分量个数
void caculate(int num[][50],int visit[],int t[],int n)
{
 int i,j,k,first,last,s[50]={0};
 first=-1;
 last=0;
 while(first!=last){
  //遍历队列
  i=t[++first];
  if(visit[i]==0){
   for(j=1;j<=n;j++){
    if(num[i][j]==1&&visit[j]!=1&&s[j]!=1){
     t[++last]=j;
     s[j]=1;
    }
   }
  }
  //遍历过的点把 visit 置为 1
  visit[i]=1;
 }
}

void main()
{
 int num[50][50]={0},visit[50]={0},t[50]={0},i,j,n;
 char c;
 //使用一个 char 读入换行符
 scanf("%d",&n);
 c=getchar();
 for(i=1;i<=n;i++){
  for(j=1;j<=n;j++){
   scanf("%d",&num[i][j]);
   c=getchar();
  }
 }
 i=1;
 while(i<=n){
  if(visit[i]==0){
   //如果 i 还没有被其他连通分量找过，它就是一个新的连通分量起点
   t[0]=i;
   //那么队列起始就作为 i，然后连通分量个数++
   caculate(num,visit,t,n);
```

```
45.    count++;
46.   }
47.   i++;
48. }
49. //最后输出连通分量个数
50. printf("%d",count);
51.}
```

**解析：**对于每个点，一定属于某一个连通分量，那么就可以从这个点出发进行广度优先搜索/深度优先搜索，能找到的所有点就是与它处在同一个连通分量里的。那么我们把找到的点都标记一下，说明它已经属于当前连通分量，不用再重新入队了。每个连通分量会造成一次广度优先搜索/深度优先搜索，记录下这样搜索的次数即可。

## 3.2. 求最小生成树的权值之和

【问题描述】

已知含有n个顶点的带权连通无向图，采用邻接矩阵存储，邻接矩阵以三元组的形式给出，**只给出不包括主对角线元素在内的下三角形部分的元素，且不包括不相邻的顶点对**。求该连通图的最小生成树的权值

【输入形式】

第一行给出结点个数n和三元组的个数count，以下每行给出一个三元组，数之间用空格隔开。**（注意这里顶点的序号是从1到n，而不是0到n-1，程序里要小心！）**

【输出形式】

最小生成树的权值

【样例输入】

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

【样例输出】

18

【样例说明】

权值是正整数，可能很大，但不需要考虑整型溢出问题

【评分标准】

在程序里写好注释，最好注明自己使用的是prim算法还是kruskal算法，方便评分

**参考答案：**

```c
#include <stdio.h>
#define M 100

int main()
{
 int G[M][M],ver[M],n,count,i,j;
 int u,v,value,m,right = 0;

    //输入点的个数和边的条数
 scanf("%d %d",&n,&count);
 for(i = 1;i <= n;i++){
  for(j = 1;j <= n;j++)
   G[i][j] = 0;
 }
 for(i = 1;i <= count;i++){
  scanf("%d %d %d",&u,&v,&value);
  G[u][v] = value;
  G[v][u] = value;
 }

   //以 prim 算法为例
 ver[1] = 1;
 m = 1;
 while(m < n){
  u = 0,v = 0;
  value = 32767;

  //u,v 用来找出当前还未统计到、不在同一侧的权值最小的边
  for(i = 1;i <= m;i++){
   for(j = 1;j <= n;j++){
    if(G[ver[i]][j] < value && G[ver[i]][j] != 0){
     value = G[ver[i]][j];
     u = ver[i];
     v = j;
    }
   }
  }

  right += G[u][v];
  G[u][v] = 0;
  G[v][u] = 0;
  for(i = 1;i <= m;i++){
   G[ver[i]][v] = 0;
   G[v][ver[i]] = 0;
```

```
45.  }
46.  ver[++m] = v;
47. }
48.
49. printf("%d",right);
50. return 0;
51.}
```

**解析：**最小生成树问题，可参考书上Prim或Kruskal算法代码。Prim需要从一个点出发，从已知的最小权值边扩充整棵树到所有点；Kruskal需要先对所有边进行排序，从小到大确认边的两边不属于同一连通分量，直到找到n−1条边。

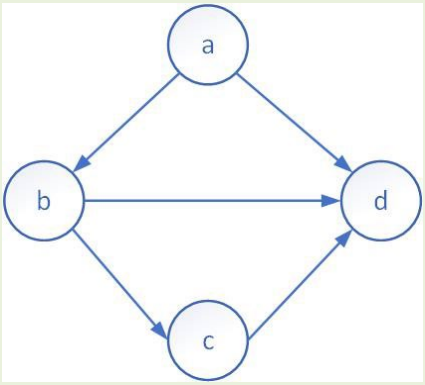
## 3.3. 2023考研408试题编程题

【问题描述】

已知优先图 G 采用邻接矩阵存储是，其定义如下：

```
typedef struct{
 int numberVertices, numEgges;
char VerticesList[maxV];
int edge[maxV][maxV];
 } MGraph;
```

将图中出度大于入度的顶点成为 K 顶点，如图，




a 和 b 都是 K 顶点，设计算法 int printVertices(MGraph G)对给定任意非空有向图 G，输出 G 中所有 K 顶点的算法，并返回 K 顶点的个数。

【输入形式】

```
please input the vertices such as '{a,b,c,d}':
{a,b,c,d}
please input the edge pair,such as '(a,b),(c,d)'
(a,b),(a,d),(b,d),(b,c),(c,d)
```

【输出形式】

```
K vertices: a b
Number of K vertices: 2
```

【样例输入】

{a,b,c,d}

(a,b),(a,d),(b,d),(b,c),(c,d)

【样例输出】

K vertices: a b

Number of K vertices: 2

【样例说明】

顶点以字母表示，边以数对表示。

注意：输出 G 中所有 K 顶点时，请保证这些节点的相对次序与第一行输入这些节点的相对次序一致

【评分标准】

正确输出给100分，错误输出给0分。

**参考答案：**

```
#include <iostream>
#include <vector>
#include <string>
#include <regex>
using namespace std;

// 图的最大顶点数
const int MAX_VERTICES = 40;

// 图的类表示
class MGraph {
private:
 int numberVertices;
 std::vector<char> verticesList;
 int edges[MAX_VERTICES][MAX_VERTICES];

public:
 MGraph() : numberVertices(0) {
  for (int i = 0; i < MAX_VERTICES; ++i) {
   for (int j = 0; j < MAX_VERTICES; ++j) {
    edges[i][j] = 0;
   }
  }
 }
```

```
25. // 设置顶点
26. void setVerticesList(const std::vector<char>& vList) {
27.  verticesList = vList;
28.  numberVertices = vList.size();
29. }
30.
31. // 添加边
32. void addEdge(char from, char to) {
33.  int fromIndex = getIndex(from);
34.  int toIndex = getIndex(to);
35.
36.  if (fromIndex != -1 && toIndex != -1) {
37.   edges[fromIndex][toIndex] = 1;
38.  }
39. }
40.
41. // 获取顶点索引
42. int getIndex(char v) {
43.  for (int i = 0; i < numberVertices; ++i) {
44.   if (verticesList[i] == v) {
45.    return i;
46.   }
47.  }
48.  return -1;
49. }
50.
51. // 打印出度大于入度的顶点，并返回它们的数量
52. int printKVertices() {
53.  int countKVertices = 0;
54.  for (int i = 0; i < numberVertices; ++i) {
55.   int outDegree = 0, inDegree = 0;
56.   for (int j = 0; j < numberVertices; ++j) {
57.    if (edges[i][j] == 1) outDegree++;
58.    if (edges[j][i] == 1) inDegree++;
59.   }
60.   if (outDegree > inDegree) {
61.    std::cout << verticesList[i] << " ";
62.    countKVertices++;
63.   }
64.  }
65.  std::cout << std::endl;
66.  return countKVertices;
67. }
68.};
69.
70.int main() {
```

```
71. MGraph g;
72. // 设置顶点列表
73. vector<char> vertices;
74. //std::cout << "please input the vertices such as '{a,b,c,d}':" << endl;
75. string aLine;
76. getline(cin,aLine);
77.
78. //匹配顶点
79. regex vertex_regex("([a-z])");
80. smatch ver_match;
81.
82. while (regex_search(aLine,ver_match,vertex_regex)) {
83.  char c = ver_match[1].str().at(0);
84.  vertices.push_back(c);
85.  aLine = ver_match.suffix().str();
86. }
87. g.setVerticesList(vertices);
88.
89. // 根据提供的图添加边
90. //cout << "please input the edge pair,such as '(a,b),(c,d)'"<< endl;
91. getline(cin, aLine);
92.
93. //设置正则表达式
94. regex edge_pair_pattern(R"(\(([a-z]),([a-z])\))");
95. smatch match;
96. while (regex_search(aLine, match, edge_pair_pattern)) {
97.
98.  //提取顶点
99.  char v1 = match[1].str().at(0);
100.       char v2= match[2].str().at(0);
101.
102.       //将顶点加入到图中
103.       g.addEdge(v1, v2);
104.
105.       aLine = match.suffix().str();
106.      }
107.
108.      // 打印K 顶点并返回数量
109.      std::cout << "K vertices: ";
110.      int kVerticesCount = g.printKVertices();
111.      std::cout << "Number of K vertices: " << kVerticesCount << std::en
    dl;
112.
113.      return 0;
114.     }
```

**解析：**本题总体而言需要统计每个点的入度和出度，建议使用stl里的map或者答案中提供的正则表达式进行匹配。只要一个点的出度大于入度就要输出，所以在读入边时，分别将前面点的出度和后面点的入度+1，最后重新遍历一次。但需要注意“输出 G 中所有 K 顶点时，请保证这些节点的相对次序与第一行输入这些节点的相对次序一致”，所以要按每个点出现的顺序遍历。