**要求代码和实验报告规范，在算法思想中：对实验涉及的数据结构进行有效设计和分析；对算法进行分析并给出时间、空间复杂度的结论；清晰表达实验思路、出现的问题及解决方法。**

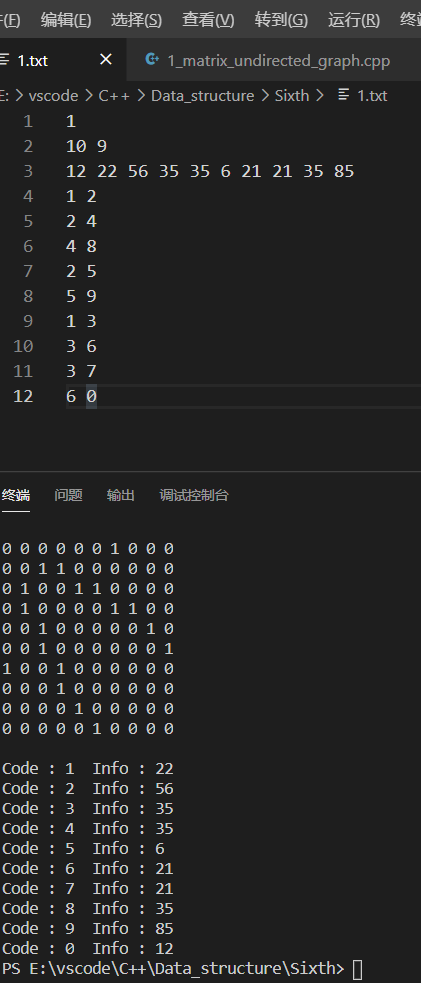
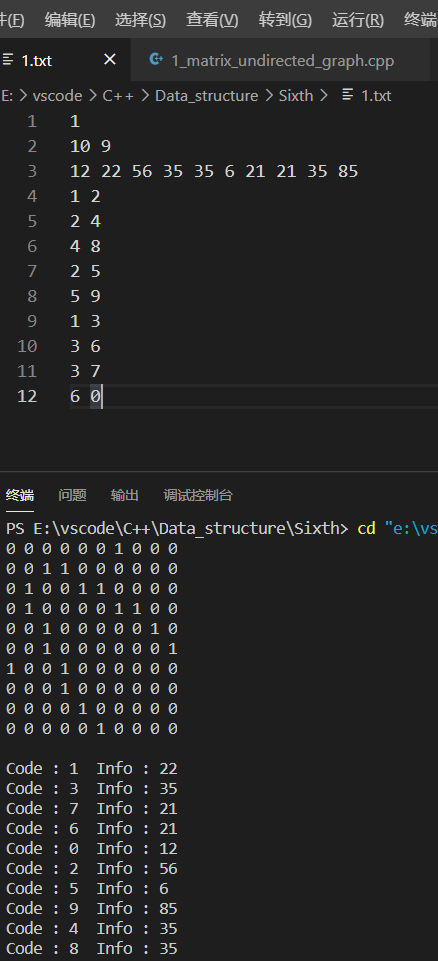
一、调试成功程序及说明

1、

题目：图的深度优先和广度优先遍历

算法思想：深度优先遍历利用栈结构实现，广度优先遍历利用队列结构进行实现。

运行结果：



结果分析：运行结果正确，实现了图的深度优先遍历和广度优先遍历。

附源程序。

void DFSTraverse(MGraph G, int v) //v记录当前遍历到的结点

{

bool flag[G.vexnum];

stack <int> S;

S.push(v); //从v开始进行遍历

int now;

while(!S.empty())

{

now = S.top();

VisitVertex(G.vexs[now]);

flag[now] = true;

S.pop();

for(int j = 0; j < G.vexnum; j++)

{

if(G.arcs[now][j] == 1 && !flag[j]) //找到一个连接的点，并且没有访问过

{

S.push(j);

}

}

}

return ;

}

//进行深度优先遍历

void BFSTraverse(MGraph G, int v)

{

bool flag[G.vexnum];

queue <int> Q;

int now;

VisitVertex(G.vexs[v]);

flag[v] = true;

Q.push(v);

while(!Q.empty())

{

now = Q.front();

Q.pop();

for(int j = 0; j <G.vexnum; j++)

{

if(G.arcs[now][j] == 1 && !flag[j])

{

VisitVertex(G.vexs[j]);

flag[j] = true;

Q.push(j);

}

}

}

return;

}

//广度优先遍历

2、

题目：朴素的实现单源最短路的最短路径算法

算法思想：最短路，没什么创新

运行结果：



结果分析：运行结果正确，实现了Dijkstra算法，达到了预期效果

附源程序。

#include <iostream>

#include <fstream>

using namespace std;

#define INF 1e9+7

int G[1005][1005];

int n, m, s;

int path[1005], dis[1005];

bool flag[1005];

void read()

{

fstream file("2.txt",ios::in);

cin >> n >> m >> s;

int u, v, w;

for(int i = 1; i <= n; i++)

{

for(int j = 1; j <= n; j++)

{

G[i][j] = INF;

}

}

for(int i = 1; i <= m; i++)

{

cin >> u >> v >> w;

G[u][v] = w;

}

file.close();

return ;

}

void showGraph()

{

for(int i = 1; i <= n; i++)

{

for(int j = 1; j <= n; j++)

{

cout << G[i][j]<< " ";

}

cout << " " <<endl;

}

return ;

}

void Dijkstra(int s)

{

for(int i = 1; i <= n; i++)

{

path[i] = s;

dis[i] = G[s][i];

if(i == s)

{

dis[i] = 0; //初始点到自己的距离为0

}

}

flag[s] = true;

int min\_index;

for(int cnt = 2; cnt <= n; cnt ++) //循环n-1次,把所有的点都拓展进来

{

min\_index = -1;

for(int i = 1; i <= n; i++) //选出路径最短的点

{

if(!flag[i])

{

if(min\_index == -1)

{

min\_index = i;

}

else if(dis[i] < dis[min\_index])

{

min\_index = i;

}

}

}

flag[min\_index] = true;

for(int i = 1; i <= n; i++)

{

if(!flag[i])

{

if(dis[i] > G[min\_index][i] + dis[min\_index])

{

dis[i] = G[min\_index][i] + dis[min\_index];

path[i] = min\_index;

}

}

}

}

return ;

}

void print()

{

for(int i = 1; i <= n; i++)

{

cout << dis[i] << " ";

}

return ;

}

int main()

{

read();

Dijkstra(s);

print();

return 0;

}

3、

题目：问题描述 ：Alice和Bob正在玩井字棋游戏。 井字棋游戏的规则很简单：两人轮流往3\*3的棋盘中放棋子，Alice放的是“X”，Bob放的是“O”，Alice执先。当同一种棋子占据一行、一列或一条对角线的三个格子时，游戏结束，该种棋子的持有者获胜。当棋盘被填满的时候，游戏结束，双方平手。   
　　Alice设计了一种对棋局评分的方法：   
　　- 对于Alice已经获胜的局面，评估得分为(棋盘上的空格子数+1)；   
　　- 对于Bob已经获胜的局面，评估得分为 -(棋盘上的空格子数+1)；   
　　- 对于平局的局面，评估得分为0；

算法思想：利用深度优先搜索，对棋局的状态进行搜索，确定出当前状态的最大得分，选出落子方案。

运行结果：



结果分析：运行结果正确，通过测试

附源程序。

#include <iostream>

#include <fstream>

#include <cmath>

using namespace std;

int T;

int cheese[5][5];

void read()

{

for(int i = 1; i <= 3; i++)

{

for(int j = 1; j <= 3; j++)

{

cin >> cheese[i][j];

}

}

}

//读入棋盘

int space()

{

int sum = 0;

for(int i = 1; i <= 3; i++)

{

for(int j = 1; j <= 3; j++)

{

if(cheese[i][j] == 0)

{

sum ++;

}

}

}

return sum;

}

//计算空格数

int simple()

{

int sum = 0;

for(int i = 1; i <= 3; i++)

{

if(cheese[i][1] == 0 || cheese[i][2] == 0 || cheese[i][3] == 0 )

{

continue;

}

else

{

sum = cheese[i][1] + cheese[i][2] + cheese[i][3];

}

if(sum == 6)

{

return 2; //Bob

}

else if(sum == 3)

{

return 1; //Alice

}

else

{

sum = 0;

}

}

for(int i = 1; i <= 3; i++)

{

if(cheese[1][i] == 0 || cheese[2][i] == 0 || cheese[3][i] == 0 )

{

continue;

}

else

{

sum = cheese[1][i] + cheese[2][i] + cheese[3][i];

}

if(sum == 6)

{

return 2; //Bob

}

else if(sum == 3)

{

return 1; //Alice

}

else

{

sum = 0;

}

}

if(cheese[1][1] != 0 && cheese[2][2] != 0 && cheese[3][3] !=0)

{

sum = cheese[1][1] + cheese[2][2] + cheese[3][3];

if(sum == 6)

{

return 2; //Bob

}

else if(sum == 3)

{

return 1; //Alice

}

else

{

sum = 0;

}

}

if(cheese[1][3] != 0 && cheese[2][2] != 0 && cheese[3][1] !=0)

{

sum = cheese[1][3] + cheese[2][2] + cheese[3][1];

if(sum == 6)

{

return 2; //Bob

}

else if(sum == 3)

{

return 1; //Alice

}

else

{

sum = 0;

}

}

return 0; // 当前无人胜出

}

//判断当前的局面有没有人赢了

int caculate()

{

int si = simple();

int sn = space();

if(si == 0)

{

return 0;

}

else if(si == 1)

{

return sn+1;

}

else if(si == 2)

{

return (sn + 1)\*(-1);

}

else

{

return 0;

}

}

//计算棋局分数

int space\_num = 0;

int win\_num = 0;

int score = 0;

int dfs(int id) //当前到谁走，1为AIice ，2 为Bob

{

if(space() == 0)

{

return 0;

}

int max\_score = -10;

int min\_score = 10;

for(int i = 1; i <= 3; i++)

{

for(int j = 1; j <= 3; j++ )

{

if(cheese[i][j] == 0)

{

cheese[i][j] = id;

if(simple()) //有人赢了

{

score = caculate();

cheese[i][j] = 0;

return score > 0 ? max(max\_score,score) : min(min\_score,score);

}

if(id == 1)

{

max\_score = max(max\_score,dfs(2));

}

else

{

min\_score = min(min\_score,dfs(1));

}

cheese[i][j] = 0;//回溯

}

}

}

return id == 1 ? max\_score : min\_score;

}

int main()

{

cin >> T;

for(int i = 1; i <= T; i++)

{

read();

space\_num = space();

win\_num = simple();

if(win\_num == 1)

{

cout << space\_num + 1 << endl;

}

else if(win\_num == 2)

{

space\_num ++;

space\_num\*=-1;

cout << space\_num << endl;

}

else //当前状态不满足的时候

{

cout << dfs(1) << endl;

}

}

return 0;

}

4、

题目：问题描述：小刘承包了很多片麦田，为了灌溉这些麦田，小刘在第一个麦田挖了一口很深的水井，所有的麦田都从这口井来引水灌溉。 为了灌溉，小刘需要建立一些水渠，以连接水井和麦田，小刘也可以利用部分麦田作为“中转站”，利用水渠连接不同的麦田，这样只要一片麦田能被灌溉，则与其连接的麦田也能被灌溉。现在小刘知道哪些麦田之间可以建设水渠和建设每个水渠所需要的费用（注意不是所有麦田之间都可以建立水渠）。请问灌溉所有麦田最少需要多少费用来修建水渠。

输入格式：输入的第一行包含两个正整数n, m，分别表示麦田的片数和小刘可以建立的水渠的数量。麦田使用1, 2, 3, ……依次标号。 接下来m行，每行包含三个整数ai, bi, ci，表示第ai片麦田与第bi片麦田之间可以建立一条水渠，所需要的费用为ci。

输出格式：输出一个整数，表示灌溉所有麦田所需要的最小费用，及水渠连接说明。

**问题分析：**这个问题可以用最小生成树算法实现。

算法思想：最小生成树的模板题目，直接利用prim算法，计算一下所有的边的权值即可。

运行结果：



结果分析：运行结果正确，通过csp模拟考试测试

附源程序。

#include <iostream>

#include <string.h>

using namespace std;

#define INF 22222222

int G[1005][1005];

int n,m;

void read()

{

cin >> n >> m;

for(int i = 1; i <= n; i++)

{

for(int j = 1; j <= n; j++)

{

G[i][j] = INF;

}

}

int u,v,w;

for(int i = 1; i <= m; i++)

{

cin >> u >> v >> w;

G[u][v] = w;

G[v][u] = w;

}

}

struct node

{

int lowcost; //最小的权

int from; //从哪个点来的

};

struct node help[1005];

int prim()

{

long long sum = 0;

help[1].lowcost = -2;

for(int i = 2; i <= n; i++)

{

help[i].from = 1;

help[i].lowcost = G[1][i];

}

int min\_index; //当前最小lowcost的下标

for(int i = 2; i <= n; i++)

{

min\_index = -1;

for(int j = 2; j <= n; j++)

{

if(help[j].lowcost != -2 && help[j].lowcost != INF)

{

if(min\_index == -1)

{

min\_index = j;

}

else if(help[min\_index].lowcost > help[j].lowcost)

{

min\_index = j;

}

else

{

continue;

}

}

}

help[min\_index].lowcost = -2;

sum += G[min\_index][help[min\_index].from];

for(int k = 2; k <= n; k++)

{

if(help[k].lowcost == -2)

{

continue;

}

else

{

if(G[min\_index][k] < help[k].lowcost)//更新最短的路径

{

help[k].lowcost = G[min\_index][k];

help[k].from = min\_index;

}

}

}

}

return sum;

}

int main()

{

read();

cout << prim();

return 0;

}

5、

题目：

算法思想：

运行结果：

结果分析：

附源程序。

......

二、未调试成功程序及说明

1、

题目：

算法思想：

错误原因：

附源程序。

2、

题目：

算法思想：

错误原因：

附源程序。

......

三、代码行数及小结

