



Exercise 1

Harry Potter's Test

<https://pintia.cn/problem-sets/15/problems/716>

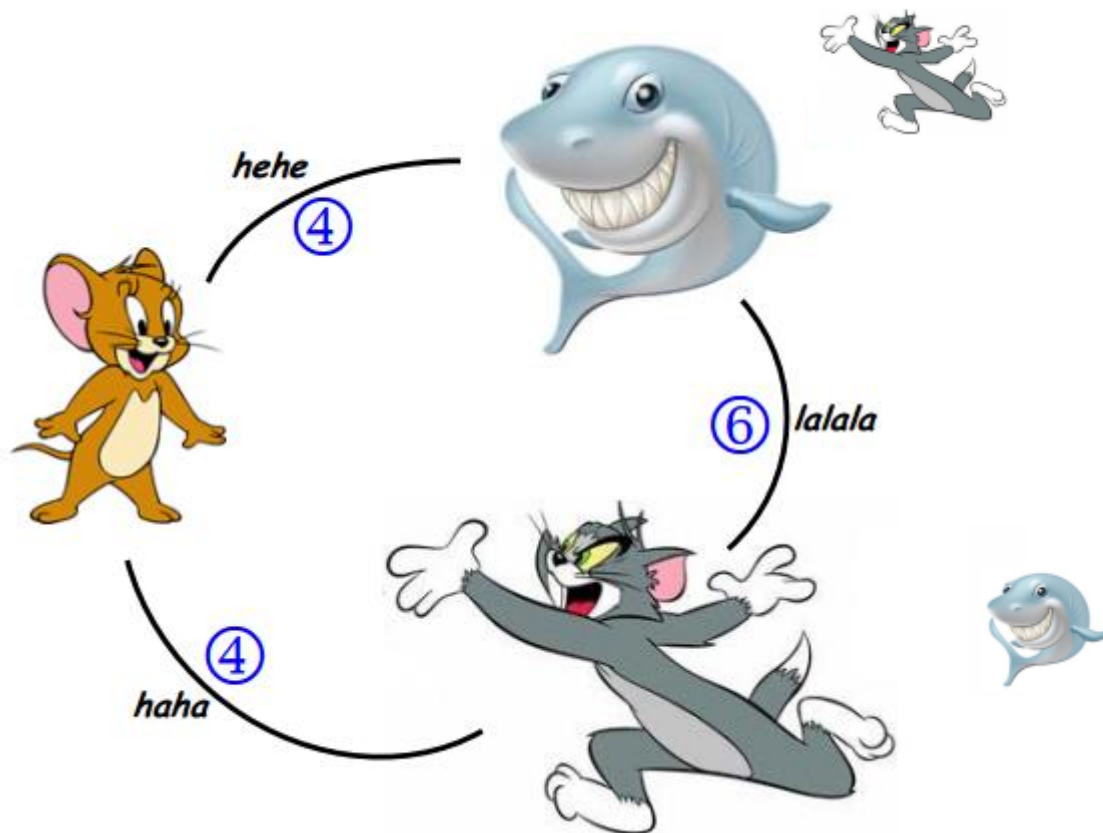
*Slides adapted from material by
Profs. Chen Yue(Zhejiang University)*

问题描述

哈利·波特要考试了，他需要你的帮助。这门课学的是用魔咒将一种动物变成另一种动物的本事。例如将猫变成老鼠的魔咒是haha，将老鼠变成鱼的魔咒是hehe等等。反方向变化的魔咒就是简单地将原来的魔咒倒过来念，例如ahah可以将老鼠变成猫。另外，如果想把猫变成鱼，可以通过念一个直接魔咒lalala，也可以将猫变老鼠、老鼠变鱼的魔咒连起来念：hahahehe。

现在哈利·波特的手里有一本教材，里面列出了所有的变形魔咒和能变的动物。老师允许他自己带一只动物去考场，要考察他把这只动物变成任意一只指定动物的本事。于是他来问你：带什么动物去可以让最难变的那种动物（即该动物变为哈利·波特自己带去的动物所需要的魔咒最长）需要的魔咒最短？例如：如果只有猫、鼠、鱼，则显然哈利·波特应该带鼠去，因为鼠变成另外两种动物都只需要念4个字符；而如果带猫去，则至少需要念6个字符才能把猫变成鱼；同理，带鱼去也不是最好的选择。

题意理解



输入输出格式

- 输入格式:
- 输入说明: 输入第1行给出两个正整数 $N(\leq 100)$ 和 M , 其中 N 是考试涉及的动物总数, M 是用于直接变形的魔咒条数。为简单起见, 我们将动物按 $1\sim N$ 编号。随后 M 行, 每行给出了3个正整数, 分别是两种动物的编号、以及它们之间变形需要的魔咒的长度(≤ 100), 数字之间用空格分隔。
- 输出格式:
- 输出哈利·波特应该带去考场的动物的编号、以及最长的变形魔咒的长度, 中间以空格分隔。如果只带1只动物是不可能完成所有变形要求的, 则输出0。如果有若干只动物都可以备选, 则输出编号最小的那只。

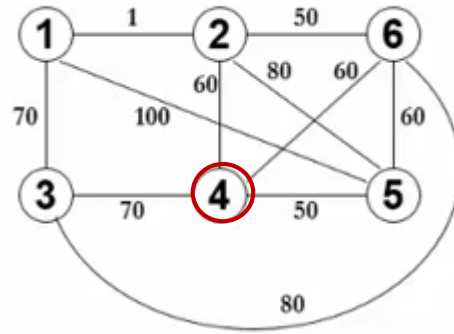
输入输出示例

输入样例:

6 11
3 4 70
1 2 1
5 4 50
2 6 50
5 6 60
1 3 70
4 6 60
3 6 80
5 1 100
2 4 60
5 2 80

输出样例:

4 70



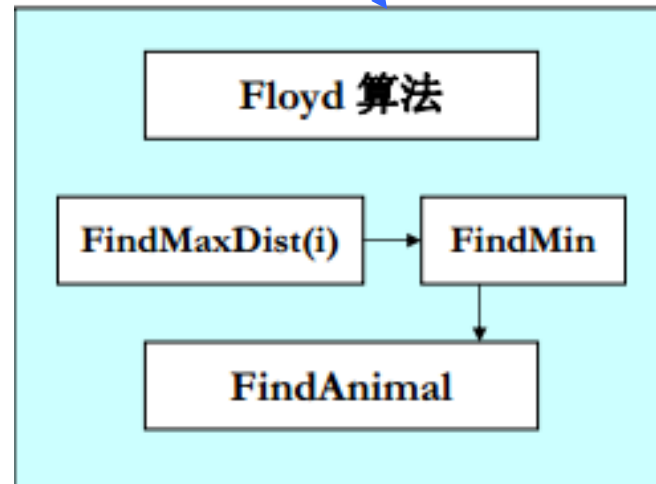
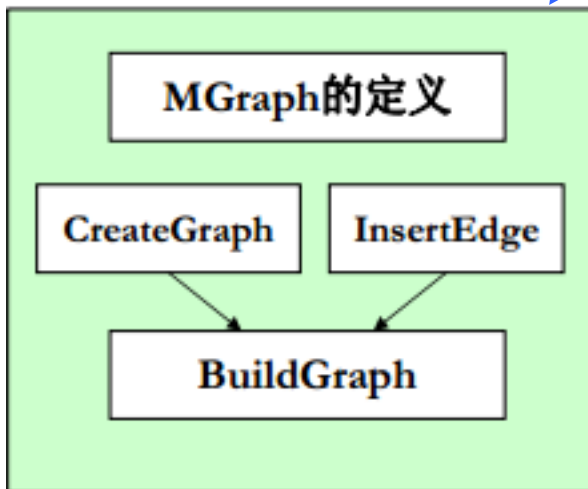
任意两顶点间最短路径 —— Floyd算法

$$D = \begin{bmatrix} \infty & 1 & 70 & 61 & 81 & 51 \\ 1 & \infty & 71 & 60 & 80 & 50 \\ 70 & 71 & \infty & 70 & 120 & 80 \\ 61 & 60 & 70 & \infty & 50 & 60 \\ 81 & 80 & 120 & 50 & \infty & 60 \\ 51 & 50 & 80 & 60 & 60 & \infty \end{bmatrix}$$

程序框架

```
int main()  
{  
    读入图;  
    分析图;  
    return 0;  
}
```

```
int main()  
{  
    MGraph G = BuildGraph();  
    FindAnimal( G );  
    return 0;  
}
```



核心伪代码

```
void FindAnimal( MGraph Graph )  
{
```

```
    Floyd( Graph, D );
```

FindMin: 从每个动物i的最短距离的最大值中，找到最小值
MinDist，以及对应的动物Animal

```
    printf("%d %d\n", Animal, MinDist);
```

```
}
```

核心伪代码

```
WeightType FindMaxDist( WeightType D[][MaxVertexNum],  
Vertex i, int N )  
{  
    WeightType MaxDist;  
    Vertex j;  
    MaxDist = 0;  
    for( j=0; j<N; j++ ) /* 找出i到其他动物j的最长距离 */  
        if (      && D[i][j]>MaxDist )  
            MaxDist = D[i][j];  
    return MaxDist;  
}
```


相关定义

```
#define MaxVertexNum 100 /* 最大顶点数设为100 */
#define INFINITY 65535 /* ∞设为双字节无符号整数的最大值65535*/
typedef int Vertex; /* 用顶点下标表示顶点,为整型 */
typedef int WeightType; /* 边的权值设为整型 */

/* 边的定义 */
typedef struct ENode *PtrToENode;
struct ENode{
    Vertex V1, V2; /* 有向边<v1, v2> */
    WeightType Weight; /* 权重 */
};

typedef PtrToENode Edge;
/* 图结点的定义 */
typedef struct GNode *PtrToGNode;
struct GNode{
    int Nv; /* 顶点数 */
    int Ne; /* 边数 */
    WeightType G[MaxVertexNum][MaxVertexNum]; /* 邻接矩阵 */
};

typedef PtrToGNode MGraph; /* 以邻接矩阵存储的图类型 */
```

相关函数

```
MGraph CreateGraph( int VertexNum )
{ /* 初始化一个有VertexNum个顶点但没有边的图 */
    Vertex V, W;
    MGraph Graph;
    Graph = (MGraph)malloc(sizeof(struct GNode)); /* 建立图 */
    Graph->Nv = VertexNum;
    Graph->Ne = 0;
    /* 初始化邻接矩阵 */
    /* 注意：这里默认顶点编号从0开始，到(Graph->Nv - 1) */
    for (V=0; V<Graph->Nv; V++)
        for (W=0; W<Graph->Nv; W++)
            Graph->G[V][W] = INFINITY;
    return Graph;
}

void InsertEdge( MGraph Graph, Edge E )
{
    Graph->G[E->V1][E->V2] = E->Weight; /* 插入边 <v1, v2> */
    /* 若是无向图，还要插入边<v2, v1> */
    Graph->G[E->V2][E->V1] = E->Weight;
}
```

相关函数


```
MGraph BuildGraph()  
{  
    MGraph Graph;  
    Edge E;  
    int Nv, i;  
    scanf("%d", &Nv); /* 读入顶点个数 */  
    Graph = CreateGraph(Nv); /* 初始化有Nv个顶点但没有边的图 */  
    scanf("%d", &(Graph->Ne)); /* 读入边数 */  
    if ( Graph->Ne != 0 ) { /* 如果有边 */  
        E = (Edge)malloc(sizeof(struct ENode)); /* 建立边结点 */  
        /* 读入边, 格式为"起点 终点 权重", 插入邻接矩阵 */  
        for (i=0; i<Graph->Ne; i++) {  
            scanf("%d %d %d", &E->V1, &E->V2, &E->Weight);  
            E->V1--; E->V2--;  
            InsertEdge( Graph, E );  
        }  
    }  
    return Graph;  
}
```

相关函数

```
void Floyd( MGraph Graph, WeightType D[][MaxVertexNum] )
{
    Vertex i, j, k;
    /* 初始化 */
    for ( i=0; i<Graph->Nv; i++ )
        for( j=0; j<Graph->Nv; j++ ) {
            D[i][j] = Graph->G[i][j];
        }
    for( k=0; k<Graph->Nv; k++ )
        for( i=0; i<Graph->Nv; i++ )
            for( j=0; j<Graph->Nv; j++ )
                if( D[i][k] + D[k][j] < D[i][j] ) {
                    D[i][j] = D[i][k] + D[k][j];
                }
}
```

完整代码

完整代码实现



Data Structures

To be continued...