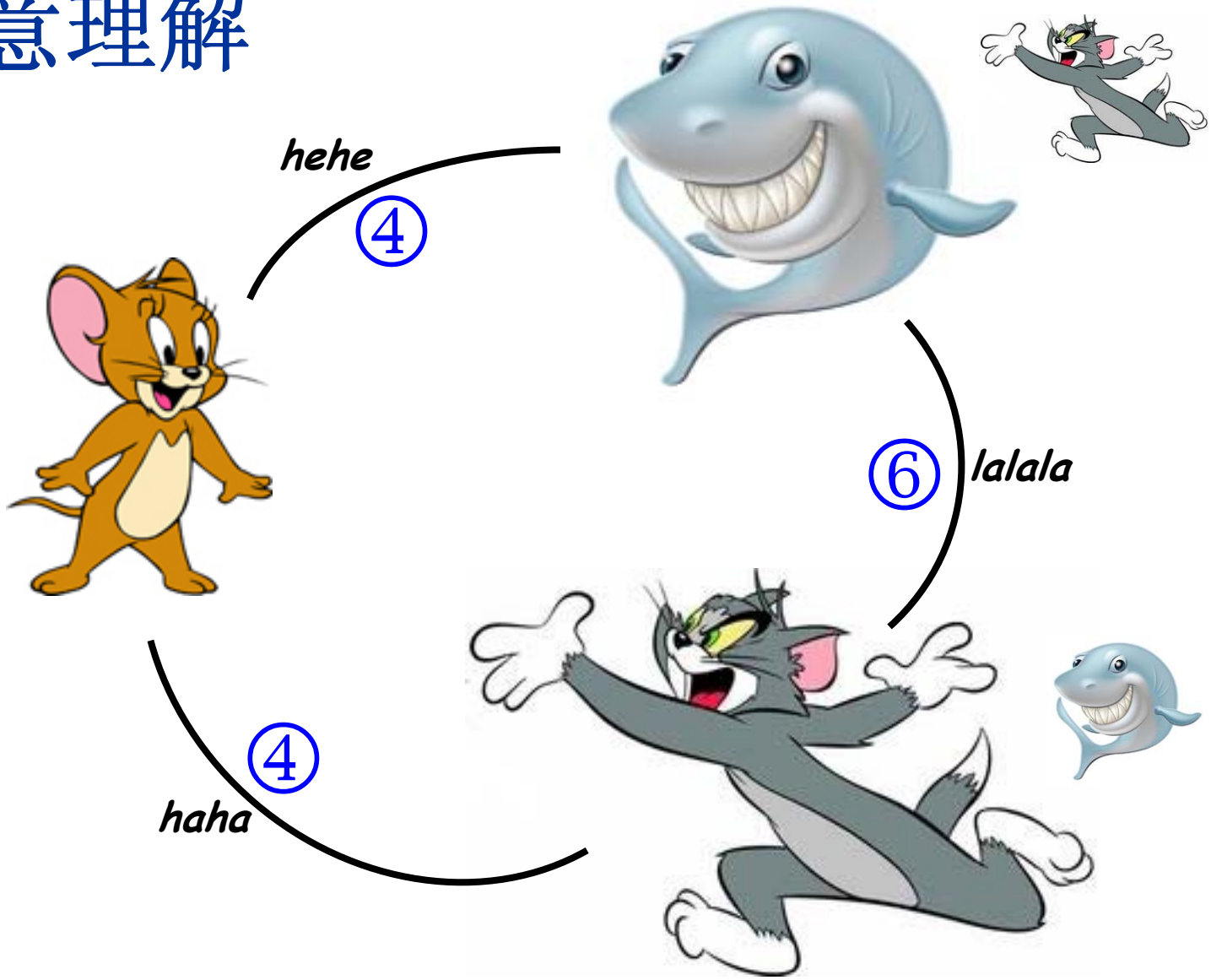
小白专场：
哈利·波特的考试

浙江大学 陈 越

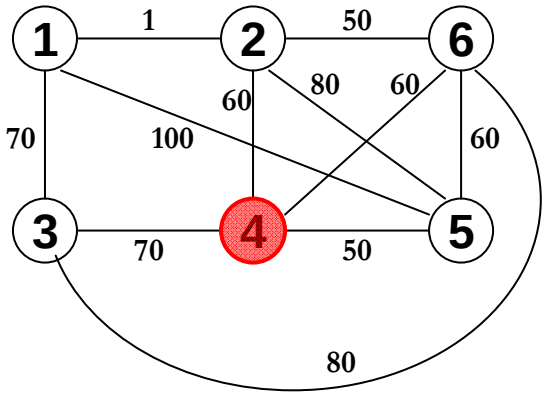
题意理解



题意理解

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
```

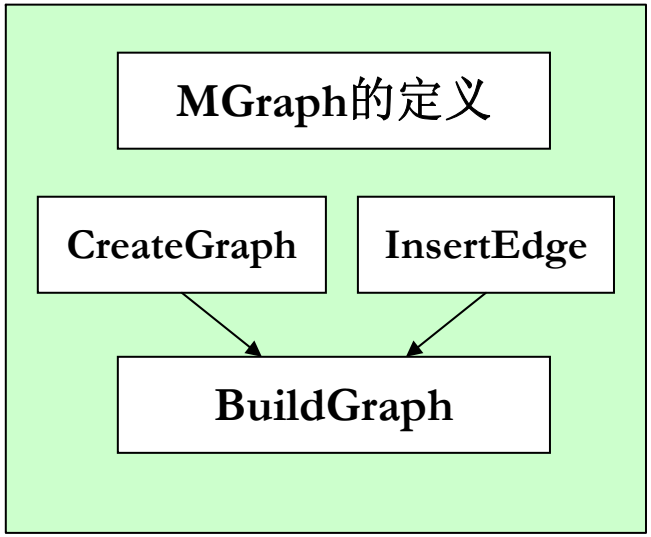
任意两顶点间最短路径 —— Floyd算法

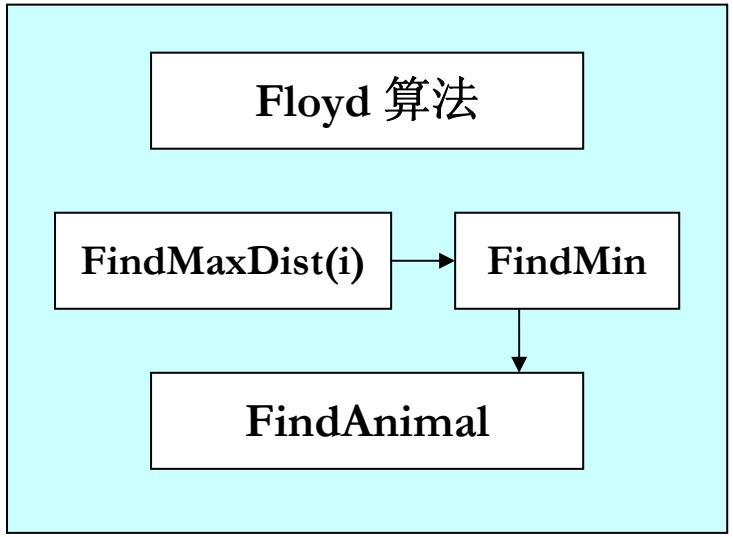
$$D=\begin{bmatrix} \infty & 1 & 70 & 61 & 81 & 51 \\ 1 & \infty & 71 & 60 & 80 & 50 \\ 70 & 71 & \infty & 70 & 120 & 80 \\ 61 & 60 & 70 & \infty & 50 & 60 \\ 81 & 80 & 120 & 50 & \infty & 60 \\ 51 & 50 & 80 & 60 & 60 & \infty \end{bmatrix}$$

程序框架搭建

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

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





选择动物

```
void FindAnimal( MGraph Graph )
{  WeightType D[MaxVertexNum][MaxVertexNum], MaxDist, MinDist;
   Vertex Animal, i;

   Floyd( Graph, D );

   MinDist = INFINITY;
   for ( i=0; i<Graph->Nv; i++ ) {
      MaxDist = FindMaxDist( D, i, Graph->Nv );
      if ( MaxDist == INFINITY ) { /* 说明有从i无法变出的动物 */
         printf("0\n");
         return;
      }
      if ( MinDist > MaxDist ) { /* 找到最长距离更小的动物 */
         MinDist = MaxDist;  Animal = i+1; /* 更新距离，记录编号 */
      }
   }
   printf("%d %d\n", Animal, MinDist);
}
```

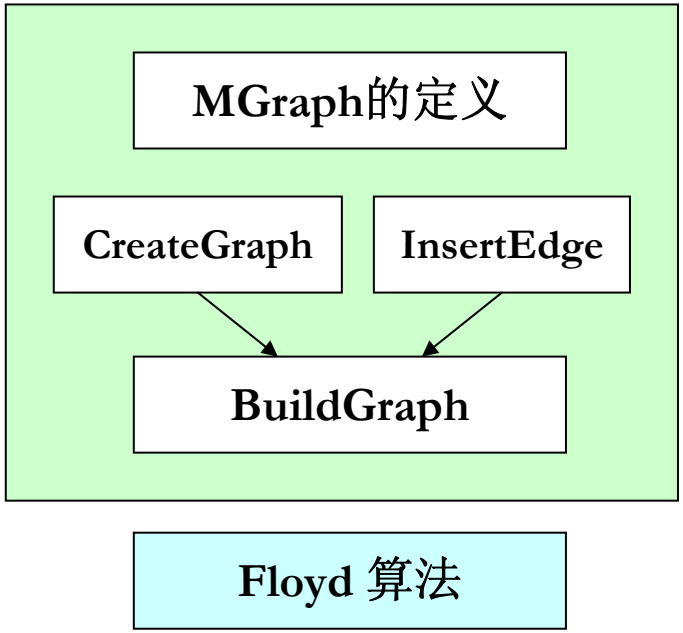
选择动物

```
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 ( i!=j && D[i][j]>MaxDist )
         MaxDist = D[i][j];

   return MaxDist;
}
```

模块的引用与裁剪




```
/* 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 */
  DataType Data[MaxVertexNum];      /* 存顶点的数据 */
  /* 注意：很多情况下，顶点无数据，此时Data[]可以不用出现 */
};
typedef PtrToGNode MGraph; /* 以邻接矩阵存储的图类型 */
```

```
#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 */
typedef char DataType;   /* 顶点存储的数据类型设为字符型 */
```

模块的引用与裁剪

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

模块的引用与裁剪

```
MGraph BuildGraph()
{
    MGraph Graph;
    Edge E;
    Vertex V;
    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--; /* 起始编号从0开始 */
            InsertEdge( Graph, E );
        }
    }

    /* 如果顶点有数据的话，读入数据 */
    for (V=0; V<Graph->Nv; V++)
        scanf(" %c", &(Graph->Data[V]));

    return Graph;
}
```

模块的引用与裁剪

```
void Floyd( MGraph Graph, WeightType D[][MaxVertexNum],
            Vertex path[][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];
         path[i][j] = -1;
      }

   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];
               if ( i==j && D[i][j]<0 ) /* 若发现负值圈 */
                  return false; /* 不能正确解决，返回错误标记 */
               path[i][j] = k;
            }

   return true; /* 算法执行完毕，返回正确标记 */
}
```