数据结构与算法

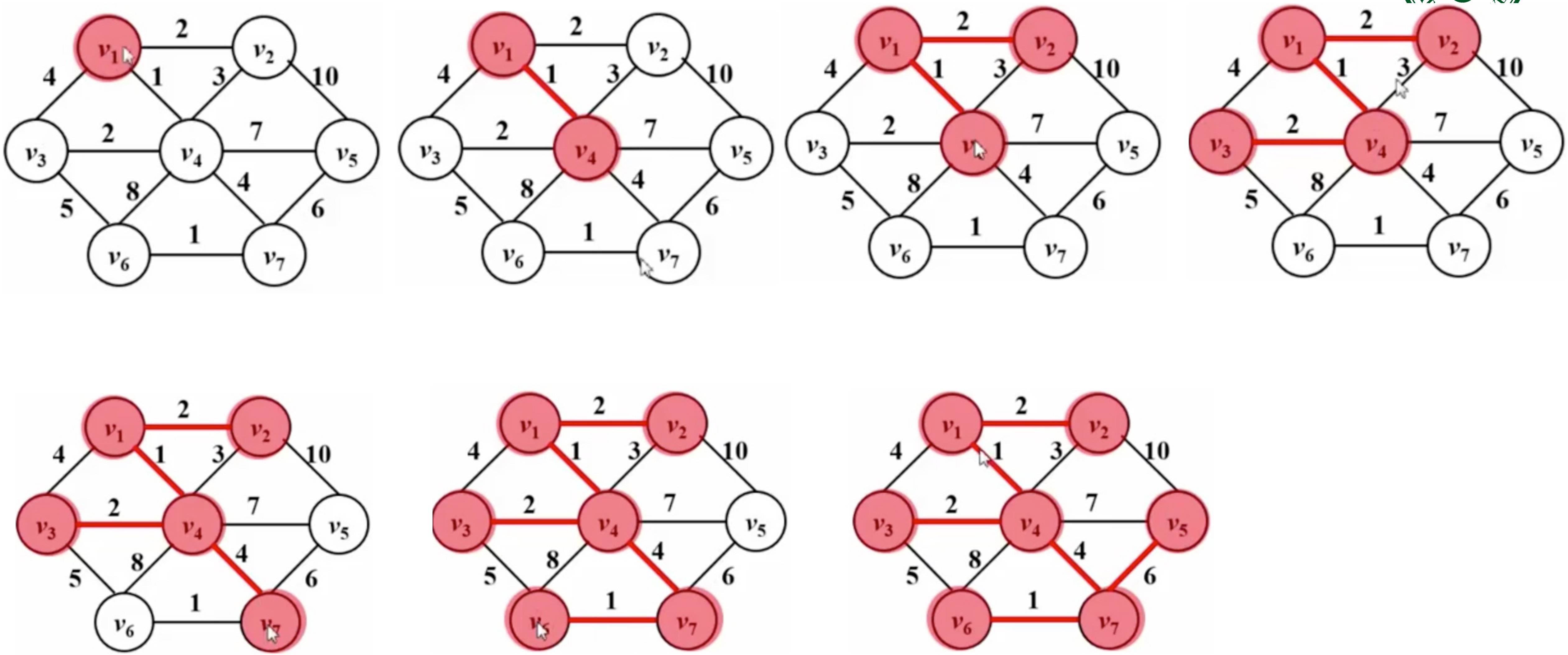
图

彭振辉
中山大学人工智能学院
2023秋季学期

本章大纲

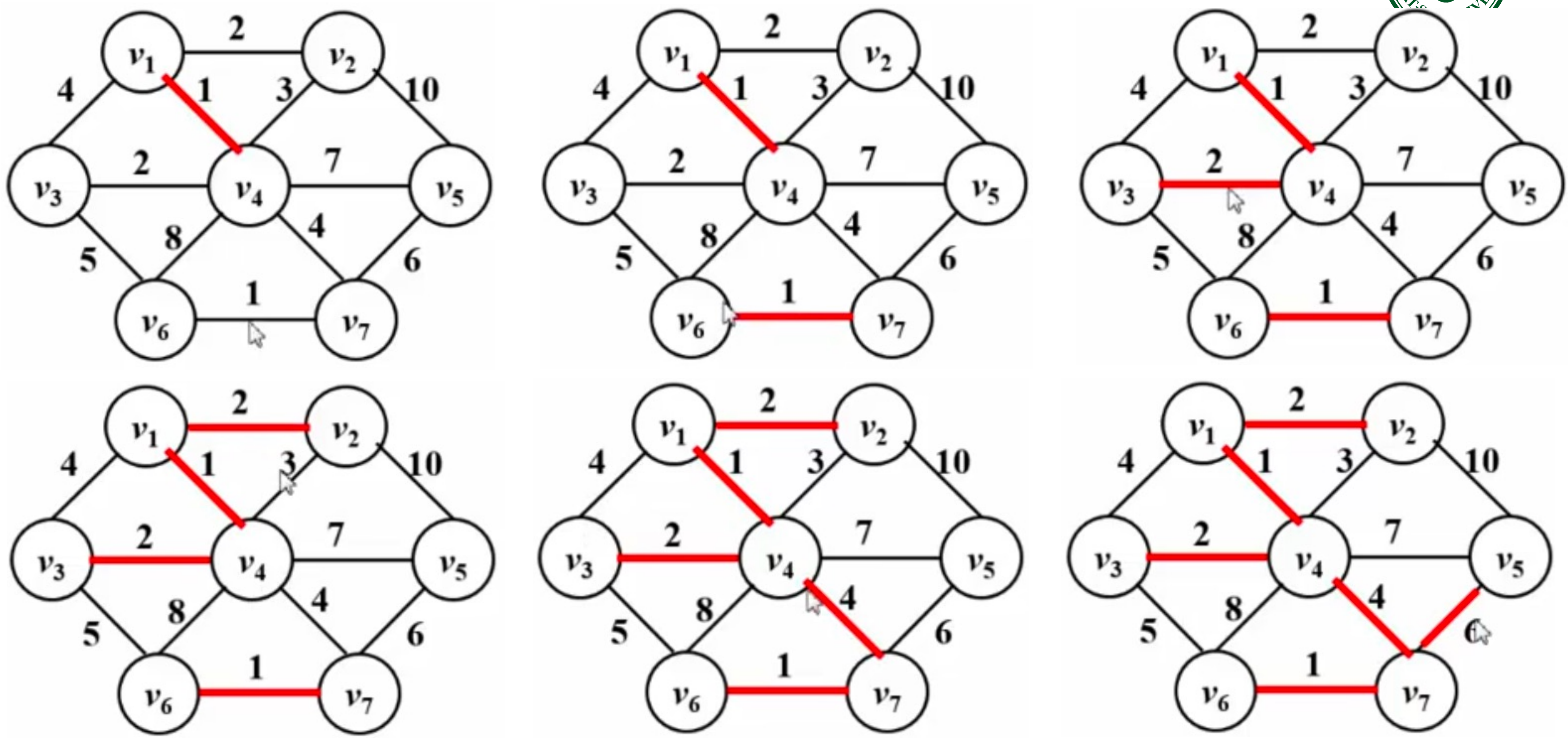
1. 图的定义与表示
2. 图的遍历
3. 图的构建
4. 最短路径问题
5. 最小生成树问题
6. **拓扑排序**
7. **应用实例**

上节回顾：Prim算法：让一棵小树长大



是不是有点像Dijkstra算法……

上节回顾：Kruskal算法：将森林合并成树



上节回顾：拓扑排序

- 随时将入度变为0的顶点放到一个容器里

```
void TopSort()
{    for ( 图中每个顶点 V )
     if ( Indegree[V]==0 )
          Enqueue( V, Q );

     while ( !IsEmpty(Q) ){
        V = Dequeue( Q );
        输出V，或者记录V的输出序号;
        cnt++;
        for ( V 的每个邻接点 W )
           if ( --Indegree[W]==0 )
              Enqueue( W, Q );
     }
     if ( cnt != |V|)
        Error( “图中有回路” );
}
```

$T = O(|V| + |E|)$

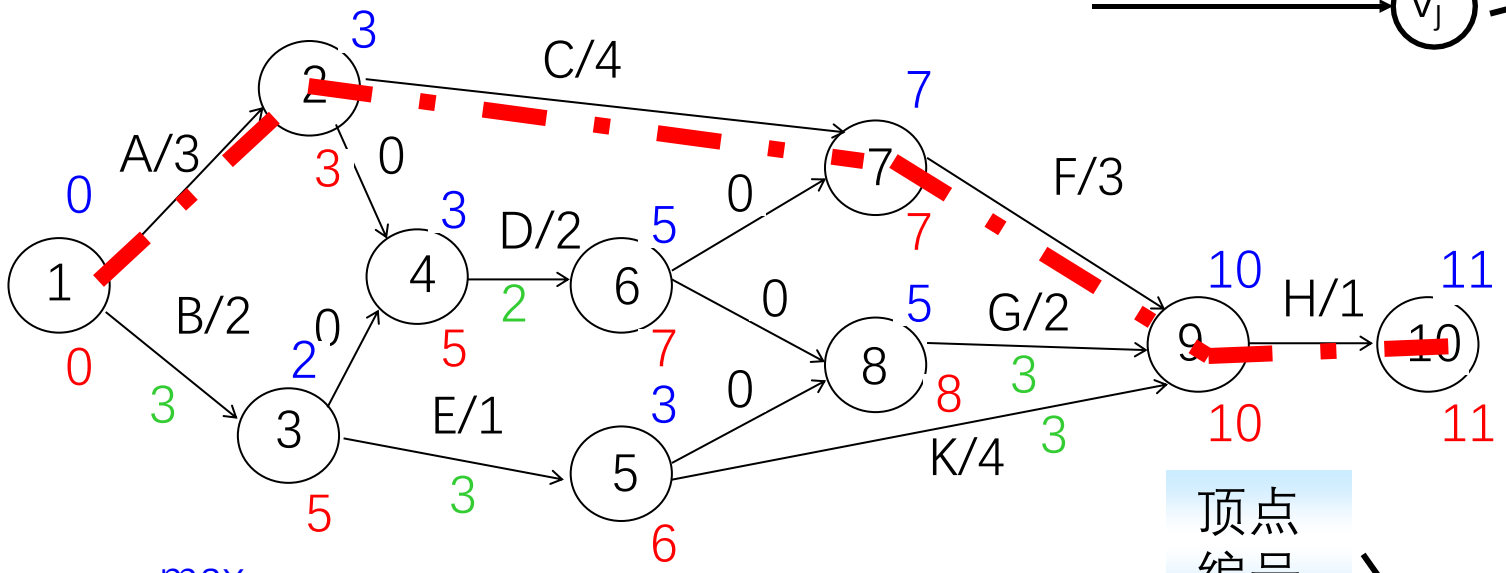
思考：此算法可以用来检测有向图是否DAG?

上节回顾：关键路径问题

由绝对不允许延误的活动组成的路径

练习：求下图的关键路径，其中， A/3代表活动A需要3个时间单位

关键路径计算过程：

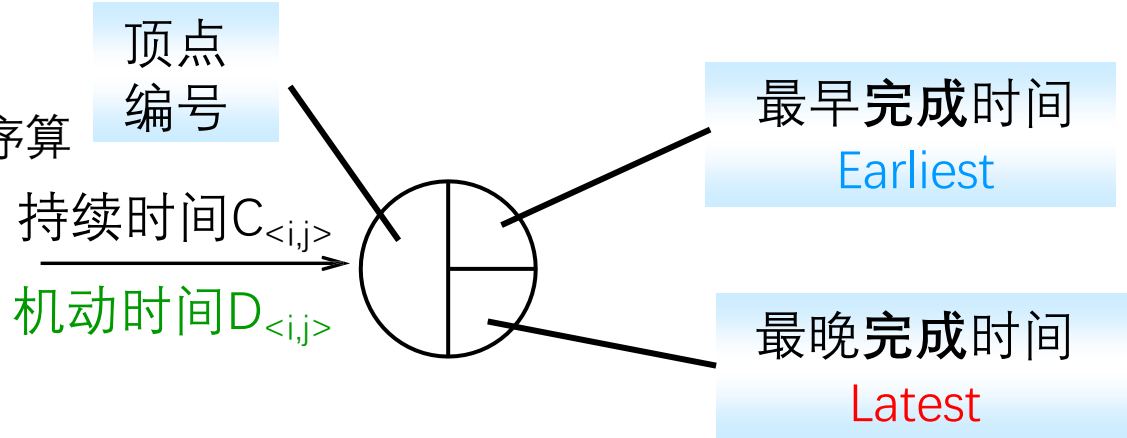


a_i 表示活动 → V_j 表示活动a_i **到此结束**

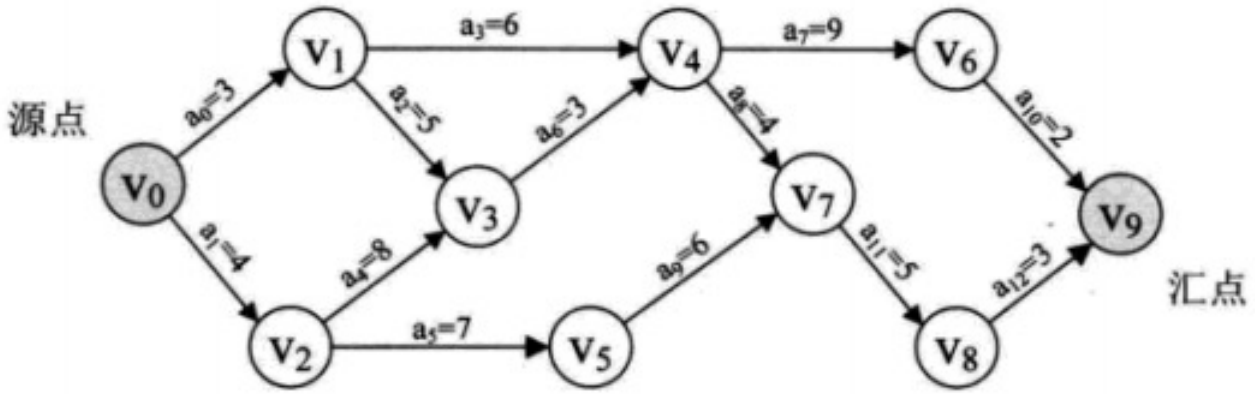
Earliest[j] = $\max_{<i,j>\in E}$ { Earliest[i] + $C_{<i,j>}$}; 按拓扑排序的顺序算

Latest[i] = $\min_{<i,j>\in E}$ { Latest[j] - $C_{<i,j>}$ }; 按逆拓扑排序的顺序算

$D_{<i,j>}$= Latest[j] - Earliest[i] - $C_{<i,j>}$



关键路径算法



下标	in	data	firstedge	adjvex	weight	next
0	0	V_0	→	2	4	→ 1 3 ∧
1	1	V_1	→	4	6	→ 3 5 ∧
2	1	V_2	→	5	7	→ 3 8 ∧
3	2	V_3	→	4	3	∧
4	2	V_4	→	7	4	→ 6 9 ∧
5	1	V_5	→	7	6	∧
6	1	V_6	→	9	2	∧
7	2	V_7	→	8	5	∧
8	1	V_8	→	9	3	∧
9	2	V_9	∧			

首先定义全局变量：

```
int *etv,*ltv;      /* 事件最早发生时间和最迟发生时间数组 */
int *stack2;        /* 用于存储拓扑序列的栈 */
int top2;           /* 用于 stack2 的指针 */
```

其中stack2用来存储拓扑序列，
已便后面求关键路径时使用

```
int *etv,*ltv;      /* 事件最早发生时间和最迟发生时间数组 */
int *stack2;        /* 用于存储拓扑序列的栈 */
int top2;           /* 用于 stack2 的指针 */
```

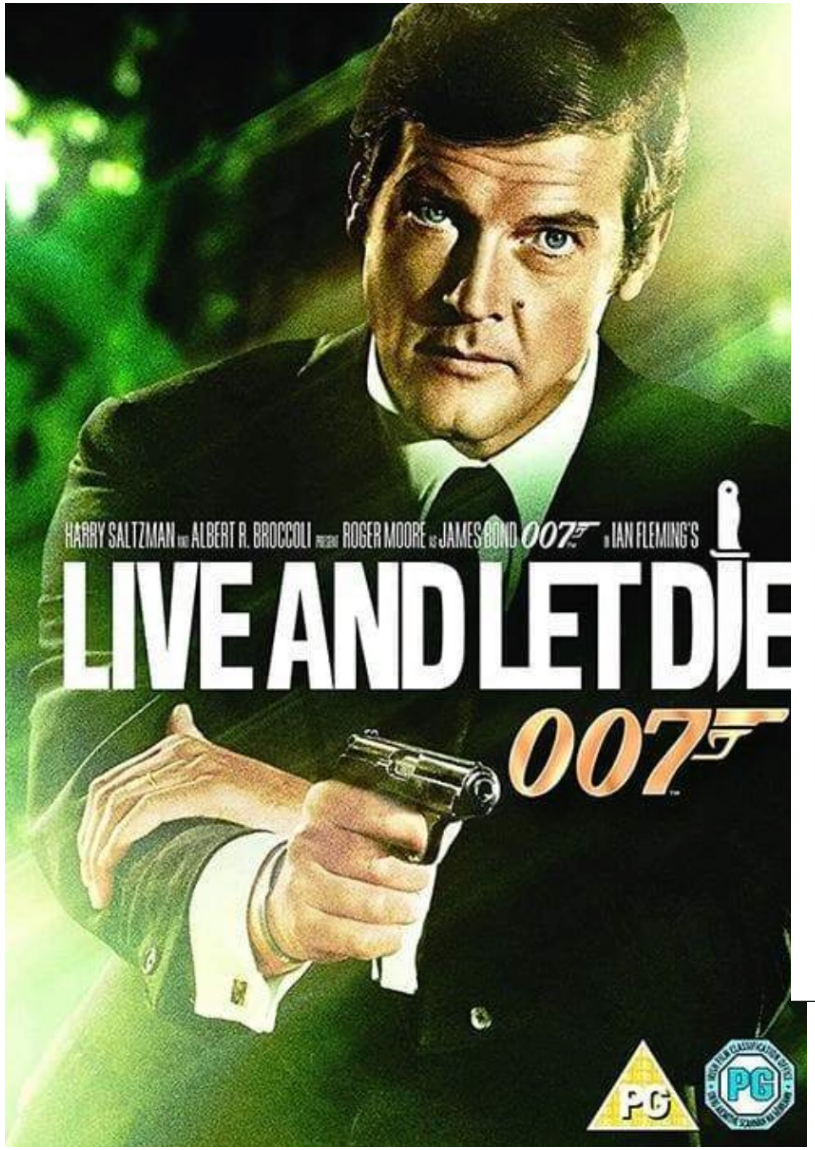
```
/* 拓扑排序，用于关键路径计算 */
Status TopologicalSort(GraphAdjList GL)
{
    EdgeNode *e;
    int i,k,gettop;
    int top=0;          /* 用于栈指针下标 */
    int count=0;        /* 用于统计输出顶点的个数 */
    int *stack;         /* 建栈将入度为 0 的顶点入栈 */
    stack=(int *)malloc(GL->numVertexes * sizeof(int) );
    for(i = 0; i<GL->numVertexes; i++)
        if(0 == GL->adjList[i].in)
            stack[++top]=i;
    top2=0;             /* 初始化为 0 */
    etv=(int *)malloc(GL->numVertexes*sizeof(int));/*事件最早发生时间*/
    for(i=0; i<GL->numVertexes; i++)
        etv[i]=0;       /* 初始化为 0 */
    stack2=(int *)malloc(GL->numVertexes*sizeof(int));/*初始化*/
    while(top!=0)
    {
        gettop=stack[top--];
        count++;
        stack2[++top2]=gettop; /* 将弹出的顶点序号压入拓扑序列的栈 */

        for(e = GL->adjList[gettop].firstedge; e; e = e->next)
        {
            k=e->adjvex;
            if( !(--GL->adjList[k].in) )
                stack[++top]=k;
            if((etv[gettop]+e->weight)>etv[k])/*求各顶点事件最早发生时间值*/
                etv[k] = etv[gettop] + e->weight;
```

关键路径算法

```
/* 求关键路径，GL 为有向网，输出 GL 的各项关键活动 */
void CriticalPath(GraphAdjList GL)
{
    EdgeNode *e;
    int i,gettop,k,j;
    int ete,lte;        /* 声明活动最早发生时间和最迟发生时间变量 */
    TopologicalSort(GL);     /* 求拓扑序列，计算数组 etv 和 stack2 的值 */
    ltv=(int *)malloc(GL->numVertexes*sizeof(int));/*事件最晚发生时间*/
    for(i=0; i<GL->numVertexes; i++)
        ltv[i]=etv[GL->numVertexes-1];  /* 初始化 ltv */
    while(top2!=0)                      /* 计算 ltv */
    {
        gettop=stack2[top2--];          /* 将拓扑序列出栈，后进先出 */
        for(e = GL->adjList[gettop].firstedge; e; e = e->next)
        { /* 求各顶点事件的最迟发生时间 ltv 值 */
            k=e->adjvex;
            if(ltv[k]-e->weight<ltv[gettop])/*求各顶点事件最晚发生时间 ltv*/
                ltv[gettop] = ltv[k] - e->weight;
        }
    }
    for(j=0; j<GL->numVertexes; j++) /* 求 ete, lte 和关键活动 */
    {
        for(e = GL->adjList[j].firstedge; e; e = e->next)
        {
            k=e->adjvex;
            ete = etv[j];           /* 活动最早发生时间 */
            lte = ltv[k] - e->weight;/* 活动最迟发生时间 */
            if(ete == lte)          /* 两者相等即在关键路径上 */
                printf("<v%d,v%d> length: %d , ",
                GL->adjList[j].data,GL->adjList[k].data,e->weight);
        }
    }
}
```

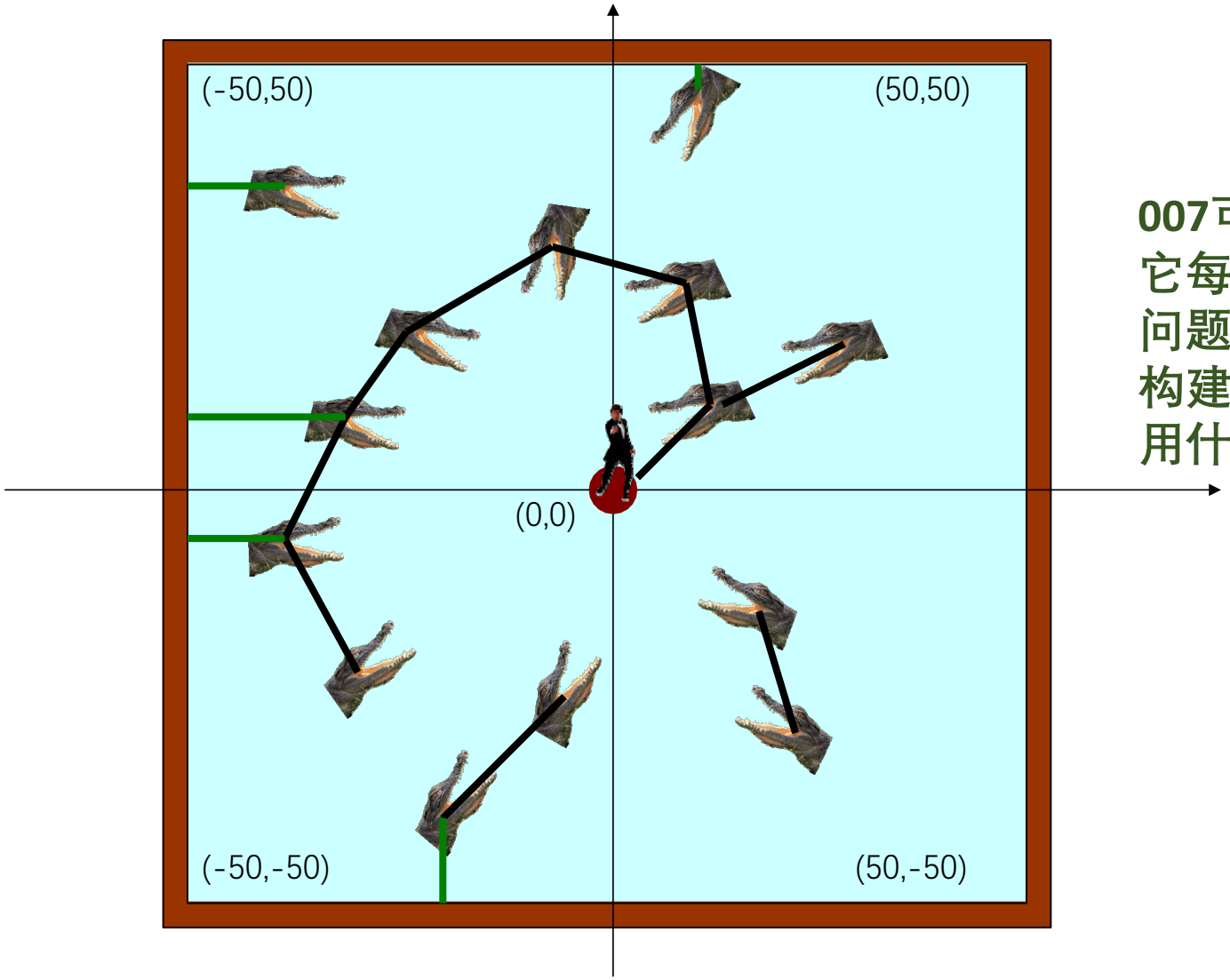
9.7 应用实例1：拯救007



https://www.youtube.com/watch?v=-LrbTd69iwI

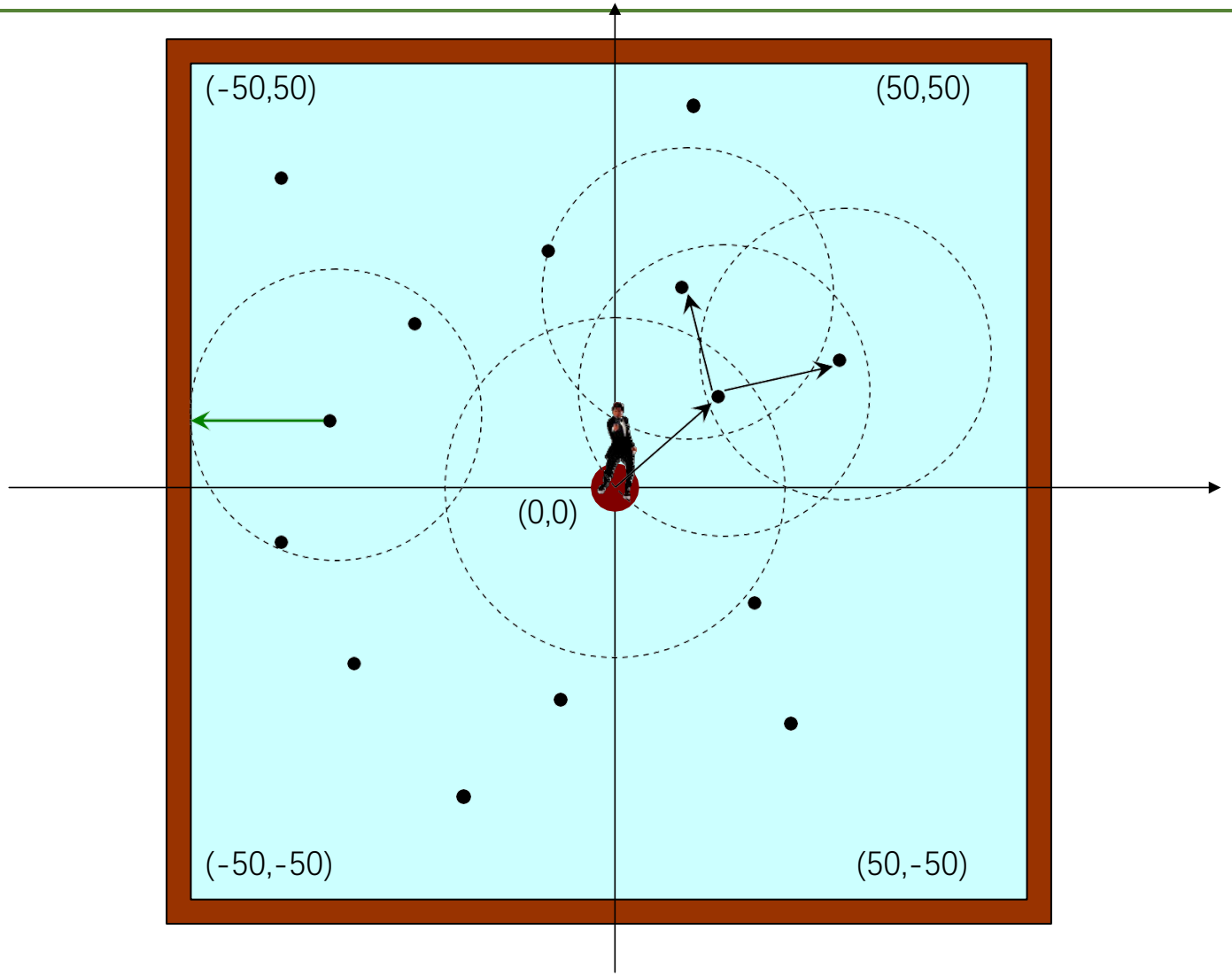
https://007store.com/products/live-and-let-die-dvd

9.7 应用实例1：拯救007



007可以踩着鳄鱼的头跳跃，但它每次有一个最长跳跃距离。
问题：他能逃到岸上去吗? (如何构建图? 顶点和边分别是什么? 采用什么算法求解?)

拯救007



总体算法

```
void ListComponents ( Graph G )
{ for ( each V in G )
        if ( !visited[V] ) {
            DFS( V );
        }
}
```

图的深度优先遍历算法(能处理非联通的情况)

```
void Save007 ( Graph G )
{ for ( each V in G ) {
        if (!visited[V]   && FirstJump(V)   ) {
             answer =  DFS( V );
            if (answer==YES) break;
        }
    }
    if (answer==YES) output(  “Yes”  );
    else output(  “No”  );
}
```

FirstJump(V): 原点到V的距离是否小于第一个阈值

从原点能跳到的点(可能有多个)开始

DFS算法

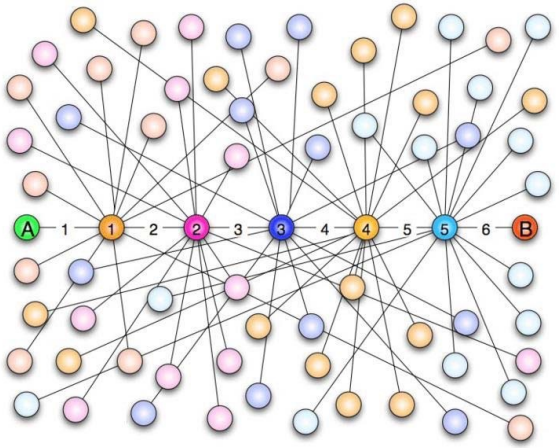
```
void DFS ( Vertex V )
{ visited[V] = true;
   for ( V 的每个邻接点 W )
        if ( !visited[W] )
             DFS(W);
}
```

从一个点开始的深度优先遍历算法

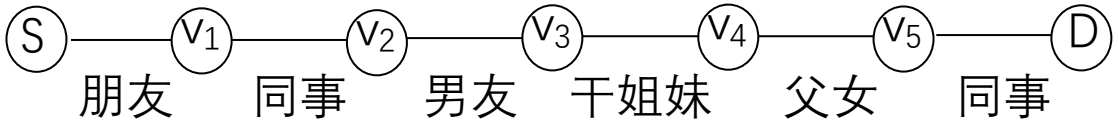
```
int DFS ( Vertex V )
{ visited[V] = true;
   if ( IsSafe(V) ) answer = YES;        对该问题的定制化处理
   else {
      for ( each W in G )
         if (!visited[W]  && Jump(V,W) ){     Jump(V, W): 能否从V跳到W
              answer = DFS(W);
              if (answer==YES) break;
         }
   }
   return answer;
}
```

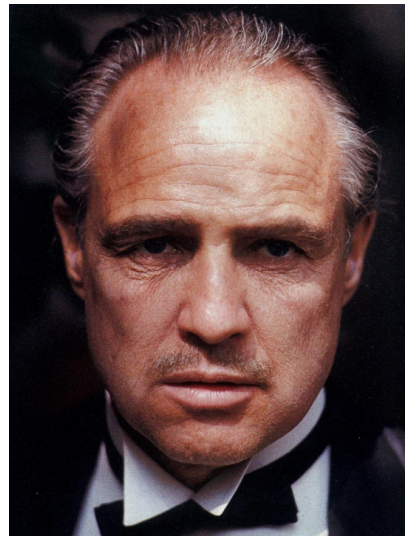
实例2：六度空间

- 你和任何一个陌生人之间所间隔的人不会超过六个



  - 法兰克福的一位土耳其烤肉店老板(S)，找到了他和他最喜欢的影星马龙·白兰度(D)的关联：





https://baike.baidu.com/item/马龙·白兰度/2487100

- **给定社交网络图，请对每个节点计算符合“六度空间”理论的结点占结点总数的百分比**

实例2：六度空间

- 如何验证六度空间理论?
  - “六度空间理论”表示成图论中的最短距离问题：用一个无向图G来表示N个人的人际关系网络。则该图大约有|E| ≈ 150×N/2条边，并假定边上的权值都是1
  - “六度空间理论”：在人际关系网络图G中，任意两个顶点之间都有一条最短距离不超过6的路径

❖ **“150定律”**

每个人可以与大约150人左右建立比较合适的人际关系。

- 方法1：采用迪杰斯特拉（Dijkstra）算法
  - 假设: N ≈ 10亿 (10^9),
  - 验证一个人 O(|E|log|V|) ≈ 150×10^9/2×log10^9 ≈ 75×10^9×30 ≈ 2250G
  - 对于每秒万亿次运算速度的计算机来说，几秒钟可以验证一个顶点，每天可以验证近万人
  - 但是，这个是无权图哎，有更简单的方法

算法思路

- 方法2：对每个节点，进行**广度优先搜索**
- 搜索过程中累计访问的节点数
- 需要记录“层”数，仅计算6层以内的节点数**（思考：如何记录层数呢?）**

```
void SDS()
{
    for ( each V in G ) {
        count = BFS(V);
        Output(count/N);
    }
}
```

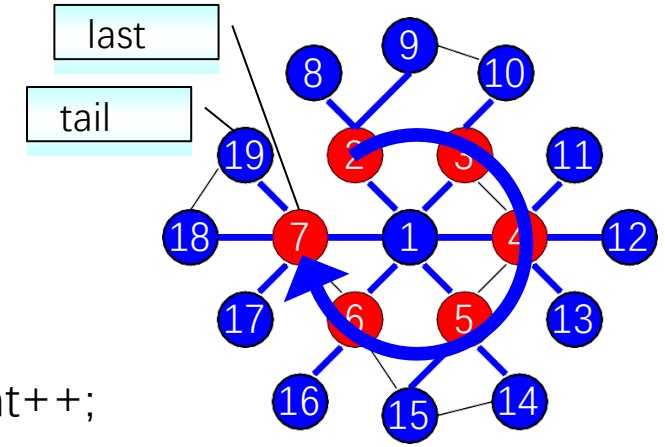
```
int BFS ( Vertex V )
{ visited[V] = true;    count = 1;
Enqueue(V, Q);
while(!IsEmpty(Q)){
        V = Dequeue(Q);
        for ( V 的每个邻接点 W )
            if ( !visited[W] ) {
                visited[W] = true;
                Enqueue(W, Q);   count++;
            }
      }
    return count;
}
```

给每个V再存一个变量表示层数? 这需要 O(n) 的空间，有无更好的办法?

算法思路

```
int BFS ( Vertex V )
{ visited[V] = true; count = 1;
   level = 0; last = V;
   Enqueue(V, Q);
   while(!IsEmpty(Q)){
      V = Dequeue(Q);
      for ( V 的每个邻接点 W )
         if ( !visited[W] ) {
            visited[W] = true;
            Enqueue(W, Q); count++;
            tail = W;
         }
      if ( V == last ) {
         level++; last = tail;
      }
      if ( level == 6 ) break;
   }
   return count;
}
```

一个 level 变量：当前顶点所在的层数；
一个 last 变量：当层访问的最后一个节点是谁



思考：时间复杂度是多少?

用邻接表表示图 O(|E|+|V|)

用广度优先搜索（BFS）方法

- 对图G进行“6层”遍历，可以算出所有路径长度不超过6的顶点数
  - 假设: N ≈ 10亿 (10^9)
  - 验证时间：O(|E|+|V|) ≈ 75×10^9 ≈ 100G
  - 对于每秒万亿次运算速度的计算机来说，一秒钟可以验证数个顶点，每天可以验证数万人
  - 空间复杂性：采用邻接表存储，大体上会有数百个G的存储量

验证六度空间理论的代码

```
void  SixDegree_BFS( MGraph  G , VertexType  Start )
{  /* 计算距离不超过SIX的结点百分比，  存于G->adjlist[Start].percent  */
   struct QElementType { VertexType v; int layer } qe;
   VertexType  v;    EdgeNode *edge;    LinkQueue Q;
    long int VisitCount = 1;  /* 记录路径长度<=SIX的顶点数 */
    Initialize( &Q );    /* 置空的队列Q */
    G->adjlist[Start].Visited = 1;
    qe.v = Start; qe.layer = 0;  /* 起点算0层 */
    AddQ( &Q, qe );              /* qe入队列 */
    while ( !IsEmptyQ(&Q) ) {    /* 队列非空循环 */
       qe = DeleteQ(&Q);  v = qe.v;
       for( edge=G->adjlist[v].FirstEdge; edge; edge=edge->Next )
           if ( !G->adjlist[edge->AdjV].Visited ) /* 若是v的尚未访问的邻接顶点 */
           {   G->adjlist[edge->AdjV].Visited = 1;  /* 将其记为六度以内的顶点 */
               VisitCount++ ;       /* 增加路径长度<=SIX的顶点数 */
               if( ++qe.layer < SIX ) /* 仅将六度以内的顶点再进队 */
               {   qe.v = edge->AdjV;   AddQ(&Q, qe);}
               qe.layer--;  /* 恢复qe的层数 */
           } /* 结束if，  for */
    } /* 结束while循环 */
    DestroyQueue( Q );
    G->adjlist[Start].percent = (float) 100.0 * VisitCount / G->n;
}
```

- 用邻接表存储

```
typedef unsigned long  VertexType;   /* 顶点用无符号长整数表示 */
typedef  struct  Vnode{      /* 顶点表结点 */
  char   Visited;          /* 顶点域，这里用于标记该结点是否已经访问 */
  float  percent;          /* 用于记录距离不超过SIX的结点百分比 */
  EdgeNode  *FirstEdge;      /* 边表头指针 */
} VertexNode;
```

实例3：旅游规划

问题描述：

有了一张自驾旅游路线图，你会知道城市间的高速公路长度、以及该公路要收取的过路费。现在需要你写一个程序，帮助前来咨询的游客找一条出发地和目的地之间的最短路径。如果有若干条路径都是最短的，那么需要输出最便宜的一条路径。

输入格式:

输入说明：输入数据的第1行给出4个正整数N、M、S、D，其中N（2≤N≤500）是城市的个数，顺便假设城市的编号为0~(N−1)；M是高速公路的条数；S是出发地的城市编号；D是目的地的城市编号。随后的M行中，每行给出一条高速公路的信息，分别是：城市1、城市2、高速公路长度、收费额，中间用空格分开，数字均为整数且不超过500。输入保证解的存在。

输出格式:

在一行里输出路径的长度和收费总额，数字间以空格分隔，输出结尾不能有多余空格。

输入样例:

```
4 5 0 3
0 1 1 20
1 3 2 30
0 3 4 10
0 2 2 20
2 3 1 20
```

输出样例:

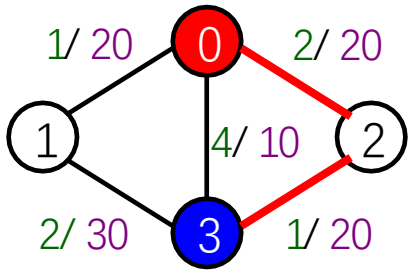
```
3 40
```

题意理解

- 城市为结点
- 公路为边
  - 权重1：距离
  - 权重2：收费
- 单源最短路
  - Dijkstra – 距离
  - 等距离时按收费更新

Sample Input:

```
4 5 0 3
0 1 1 20
1 3 2 30
0 3 4 10
0 2 2 20
2 3 1 20
```



核心算法

```
void Dijkstra( Vertex s )
{ while (1) {
      V = 未收录顶点中dist最小者;
      if ( 这样的V不存在 )
        break;
      collected[V] = true;
      for ( V 的每个邻接点 W )
        if ( collected[W] == false )
          if ( dist[V]+E<V,W> < dist[W] ) {
               dist[W] = dist[V] + E<V,W> ;
               path[W] = V;
               cost[W] = cost[V] + C<V,W> ;
          }
          else if ( (dist[V]+E<V,W> == dist [w] )
                    && (cost[V]+C<V,W> < cost[W]) ){
              cost[W] = cost[V] + C<V,W> ;
          }
    }
}
```

其他类似问题

- 要求最短路径有多少条
  - count[s] = 1; //记录从源点到当前点的最短路径数量
  - 如果找到更短路（ dist[V]+$E_{<V,W>}$ < dist[W] ）： count[W]=count[V];
  - 如果找到等长路（ dist[V]+$E_{<V,W>}$ == dist [W] ）： **count[W]+=count[V];**

- 要求边数最少的最短路径
  - count[s] = 0; // 跟上一页算法基本一致，只是count[W]的更新不一样
  - 如果找到更短路：count[W]=count[V]+1;
  - 如果找到等长路：count[W]=count[V]+1;

实例4：哈利·波特的考试

输入格式: 输入第1行给出两个正整数N (≤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~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: 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。

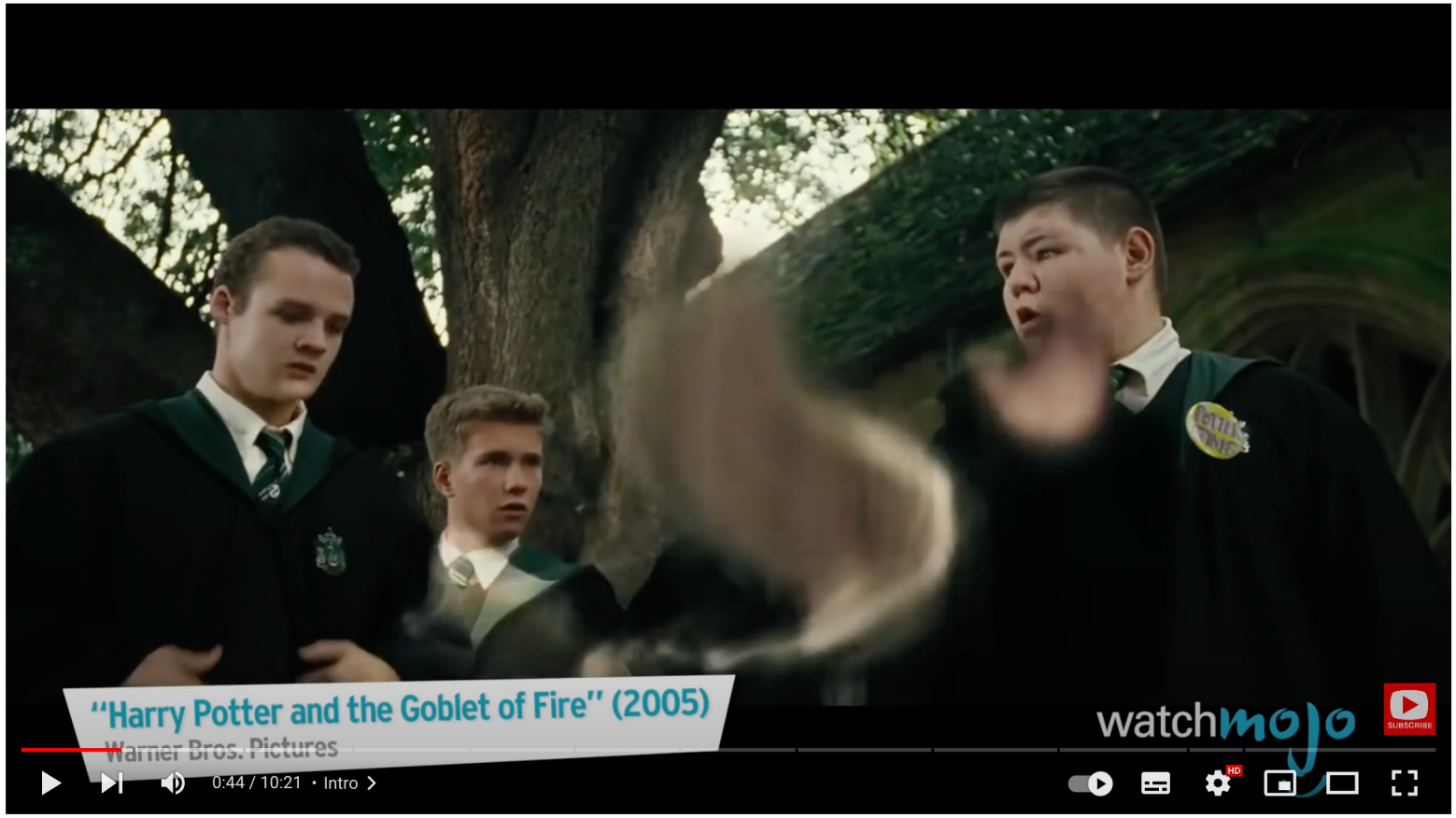
输入样例:

```
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
```

应用：哈利·波特的考试



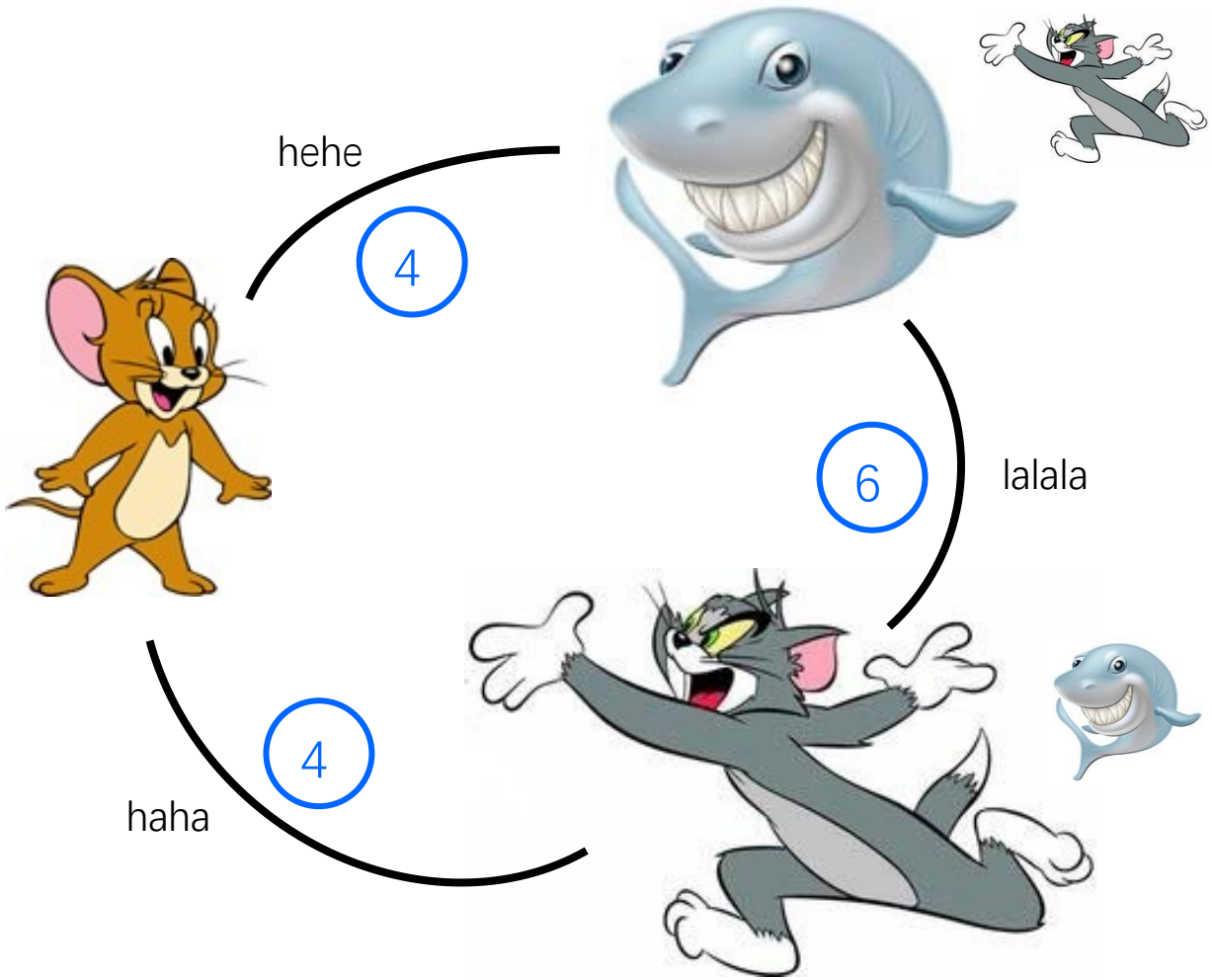
Top 10 Harry Potter Spells

WatchMojoUK
64.4万位订阅者
订阅
1.3万
分享
剪辑
保存

92万次观看 4年前
Top 10 Amazing Harry Potter Spells

https://www.youtube.com/watch?v=j_Ys8IWzyH0

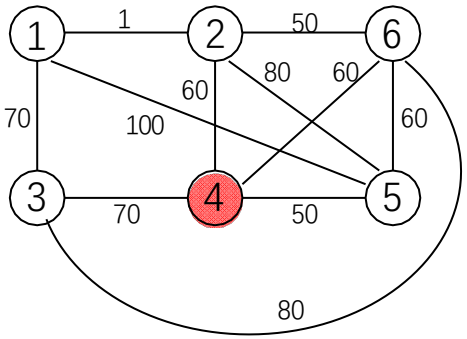
题意理解



题意理解

输入样例:

```
6 1
  1
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

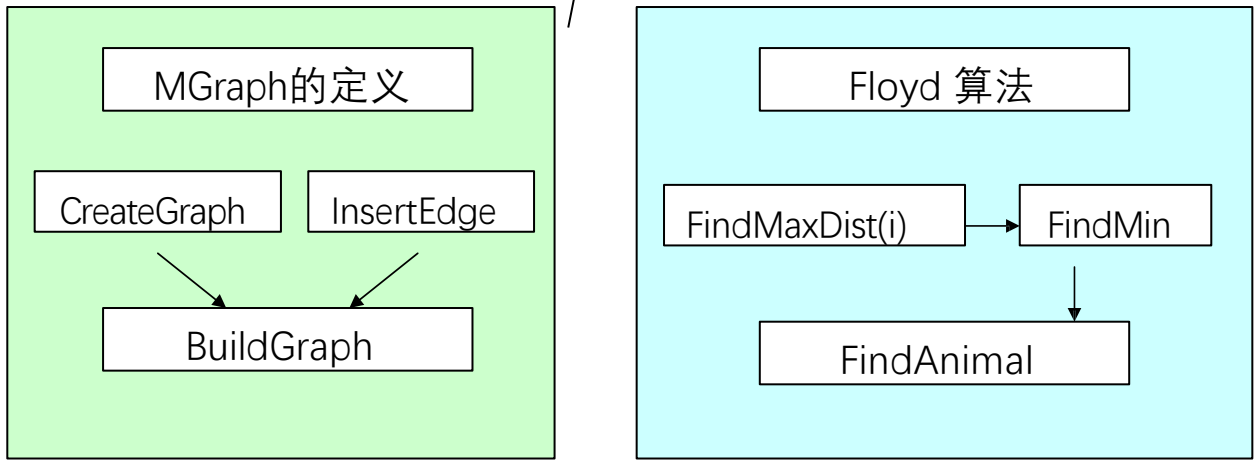
任意两顶点间最短路径 —— Floyd算法

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

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);
}
```

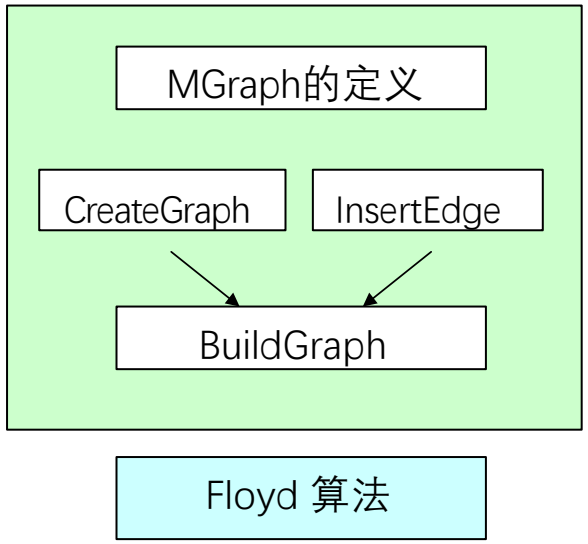
选择动物

```
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 */
}
```

实例5：散列逆问题

题目简介

- 给定一个大小为N的哈希表，我们可以定义一个哈希函数**H(x)=x%N**。假设使用线性探测来解决冲突，我们可以很容易地得到哈希表的状态。**这道题需要根据给定的哈希表的状态来重构输入序列**，如果有多种选择，则选择较小的数排前面。

输入规范

- 每个输入文件包含一个测试用例。对于每个测试用例，第一行包含一个**正整数N(≤1000)，这是哈希表的大小**。下一行包含N个整数，用空格分隔。**一个负整数表示哈希表中的一个空单元格**。保证所有非负整数在表中是不同的。

输出规范

- 对于每个测试用例，打印一行包含输入序列的代码，数字之间用空格隔开。注意，每一行的末尾必须没有额外的空间。

输入输出样例

Sample Input:

```
11
33 1 13 12 34 38 27 22 32 -1 21
```

Sample Output:

```
1 13 12 21 33 34 38 27 22 32
```

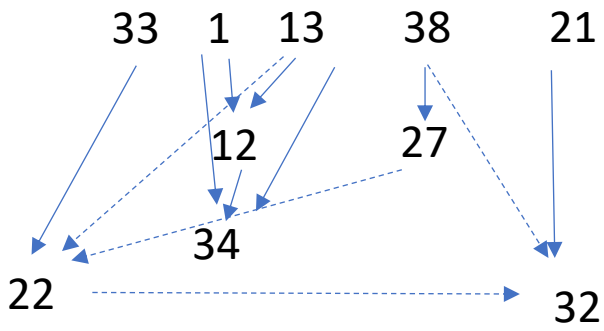
题意理解

- 已知 H(x) = x % N 以及用线性探测解决冲突问题
- 先给出散列映射的结果，反求输入顺序
  - **当元素 x 被映射到 H(x) 位置，发现这个位置已经有 y 了，则 y 一定是在 x 之前被输入的**

题意理解

H(x) = x % 11

下标	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
H[ ]	33	1	13	12	34	38	27	22	32		21



1 13 12 21 33 34 38 27 22 32

拓扑排序！