问题

- 如何设计一个活动安排，保证资源使用最充分？即资源空闲时间最少。
  - 设有 n 个活动的集合S={1,2,…,n}，其中每个活动都要求使用同一资源，如学校礼堂等，而在**同一时间内只有一个活动能使用这一资源**；
  - 每个活动都有使用的起始时间b_i和结束时间e_i 。若$b_i \geq e_j$或$b_j \geq e_i$（即一个活动结束后另一个才开始），则活动 i 与活动 j 相容；

i	1	2	3	4	5	6	7	8	9
b[i]	1	2	0	5	4	5	7	9	11
e[i]	3	5	5	7	9	9	10	12	15

问题求解与实践
——最短路径问题

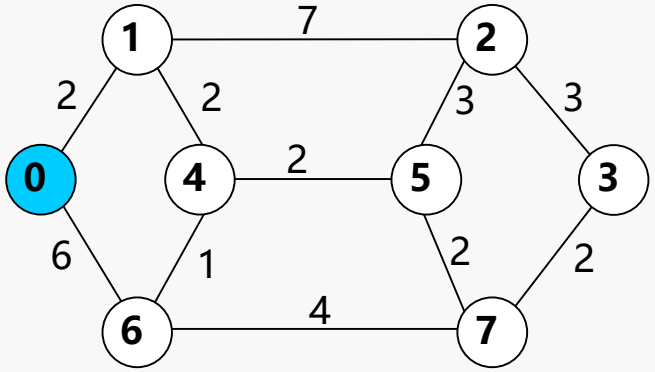
主讲教师：陈雨亭、沈艳艳

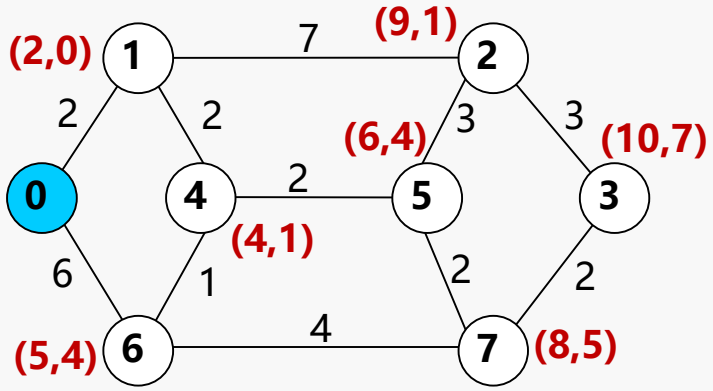
图的最短路径问题

- 在带权图中，两个顶点之间的路径长度不是指路径上边数的总和，而是指路径上各边的权值总和
- 最短路径问题：一般是求带权图中两个顶点间长度最短的路径
- 求最短路的算法是E. W. Dijkstra于1959年提出来的，这是至今公认的求最短路径的最好方法，一般称它为Dijkstra算法

图的最短路径问题

- 习惯上称路径的开始顶点为**源点**，路径的最后一个顶点为**终点**；
- 为了讨论方便，设顶点集V={0，1，…，n-1}，并假定所有边上的权值均是表示长度的非负实数；
- Dijkstra算法是逐步求出**源点**到其余各顶点的最短路径，从而求得**源点**到**终点**的最短路径（如求图中0->3的最短路径的过程）

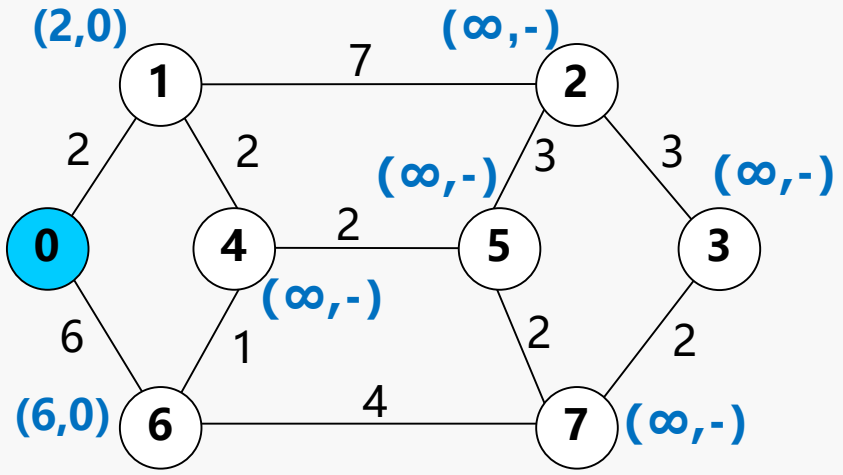




Dijkstra算法示例

设G =<V, E,W>是带权的图，其中V是顶点集合，E是边集合，W是权值集合，W(i, j)为从结点 i 到结点 j 的直连距离

第1步，把V分成两部分 S（已找到最短路径）和 T（其他结点）。初始时 S ={0}，T=V-S，且对任何T中的点X，设0到X最短距离D(X)=W(A, X)

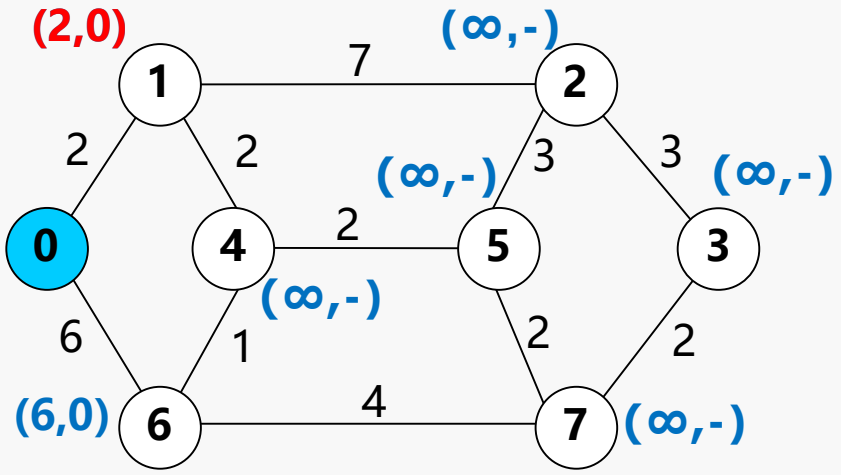


W = {
∞,2,∞,∞,∞,∞,6,∞,
2, ∞,7, ∞,2,∞,∞,∞,
∞, 7,∞, 3,∞,3,∞,∞,
∞,∞,3, ∞,∞,∞,∞,2,
∞, 2,∞,∞,∞, 2,1,∞,
∞,∞, 3,∞, 2,∞,∞,2,
6, ∞,∞,∞, 1,∞,∞,4,
∞,∞,∞, 2,∞,2, 4,∞ }

S ={0}，T={1,2,3,4,5,6,7}

Dijkstra算法示例

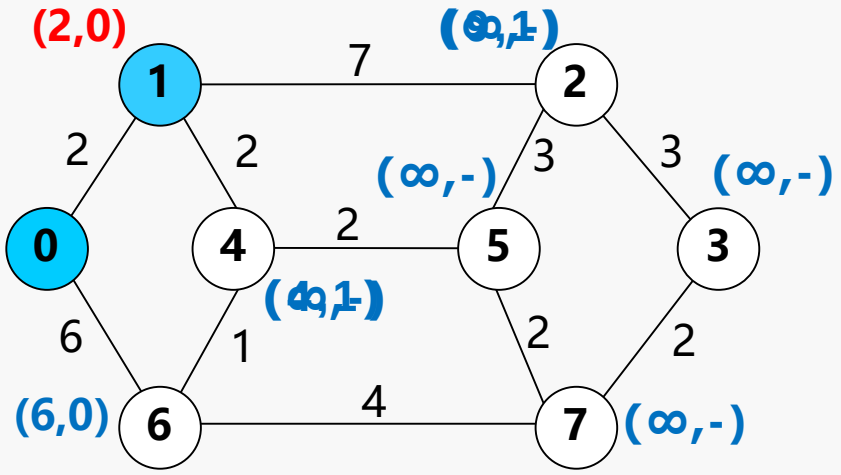
第2步，寻找T中的点Y使D(Y)最小，把Y加入S，再对所有仍在T中的结点X执行下面语句：D(X) =Min[D(X)，D(Y) + W(Y, X)]



S ={0}，T={1,2,3,4,5,6,7}

Dijkstra算法示例

第2步，寻找T中的点Y使D(Y)最小，把Y加入S，再对所有仍在T中的结点X执行下面语句：D(X) =Min[D(X), D(Y) + W(Y, X)]

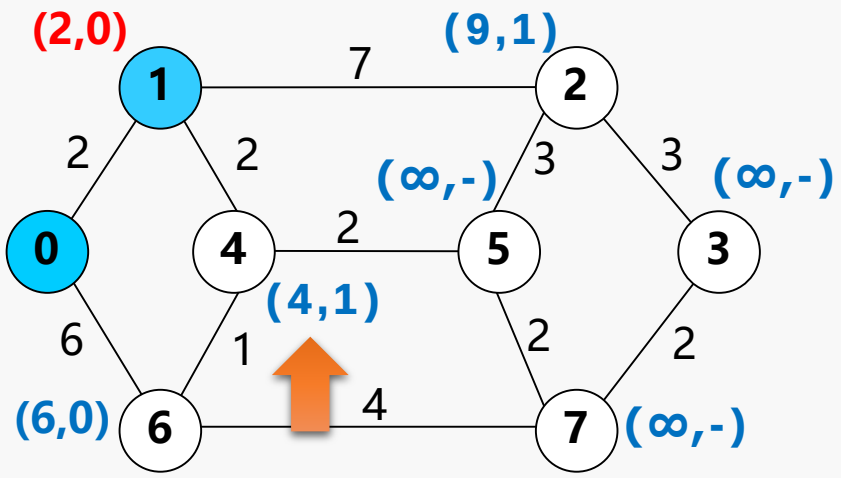


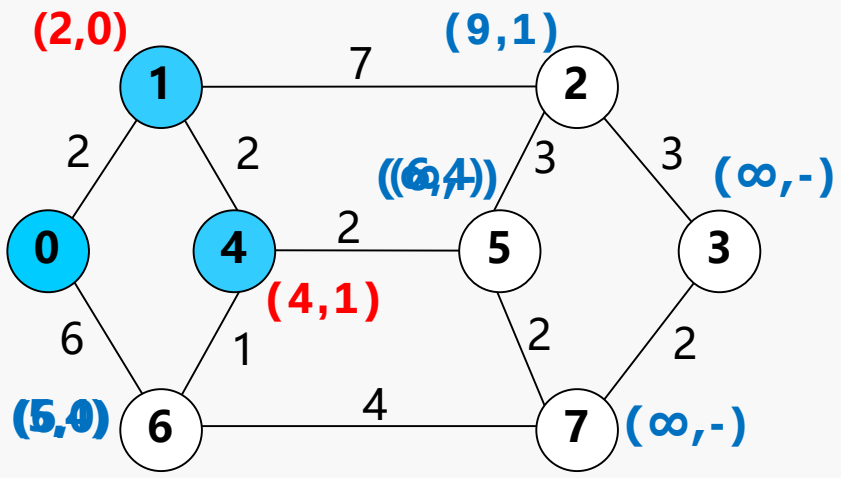
为什么0->1的最短距离是2？
有其他更短的路径吗?

S ={0,1}，T={2,3,4,5,6,7}

Dijkstra算法示例

第3步，重复步骤 2，直到所有结点都在S中为止



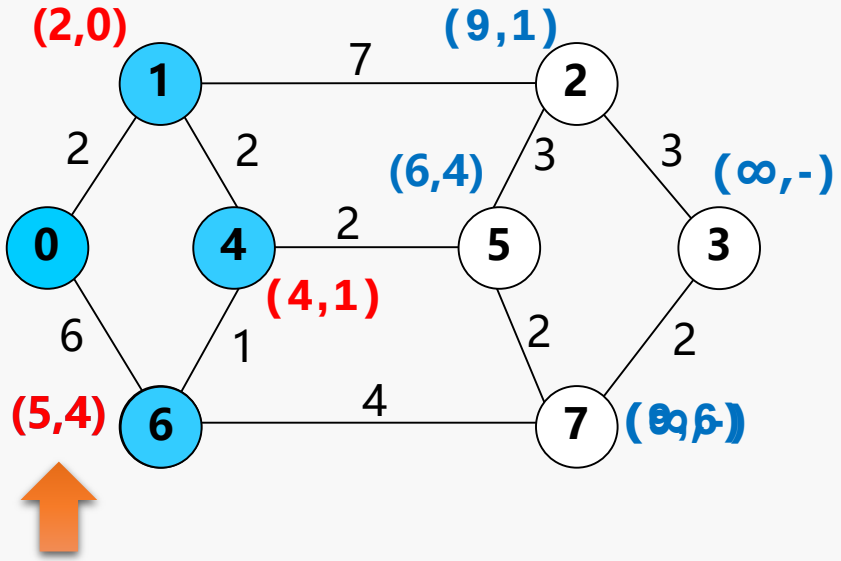


S ={0,1}，T={2,3,4,5,6,7}

S ={0,1,4}，T={2,3,5,6,7}

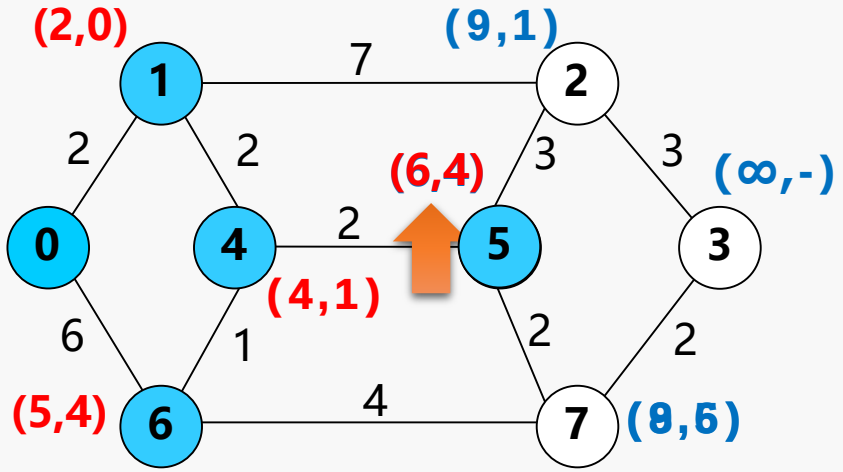
Dijkstra算法示例

继续重复步骤 2



Dijkstra算法示例

继续重复步骤 2

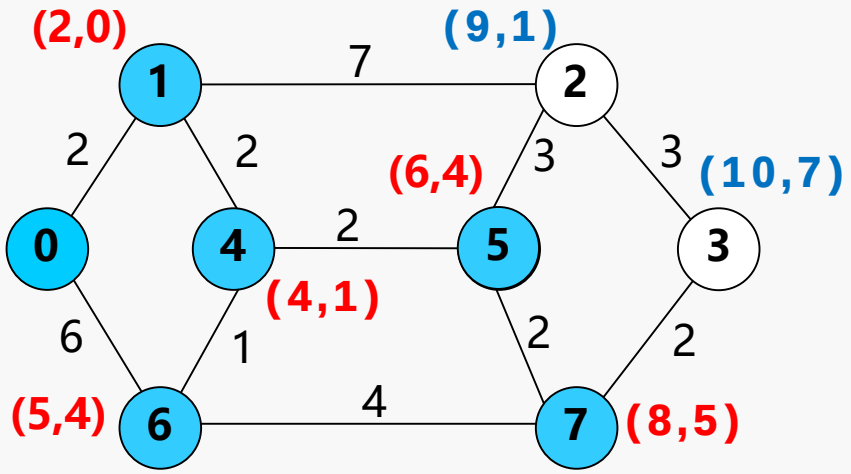


到顶点 2 的最短距离需要修改吗?

S ={0,1,4,6,5}, T ={2,3,7}

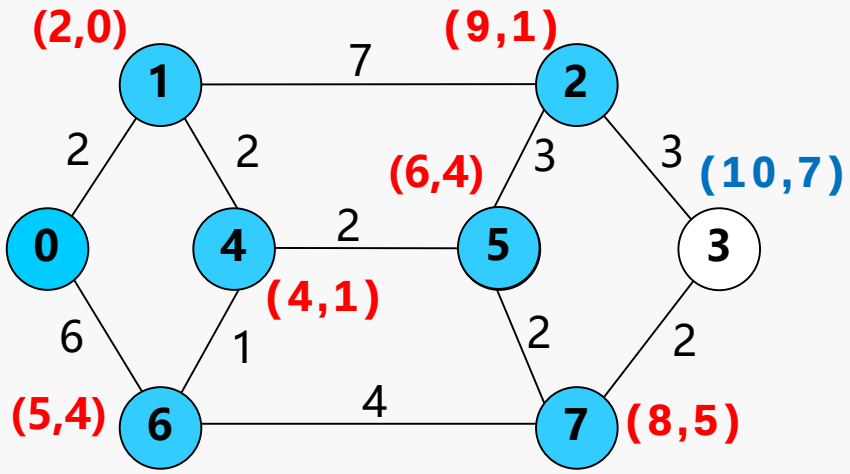
Dijkstra算法示例

继续重复步骤 2



S ={0,1,4,6,5,7}，T={2,3}

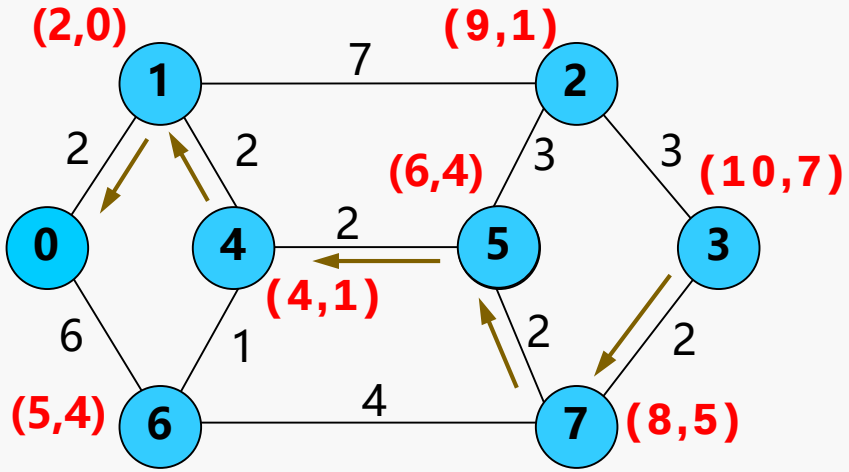
添加顶点7，修改到3的距离



S ={0,1,4,6,5,7,2}，T={3}

添加顶点2

Dijkstra算法示例



S ={0,1,4,6,5,7,2,3}，T={}

添加顶点3，完成!

从终点开始得到0->3最短路径:

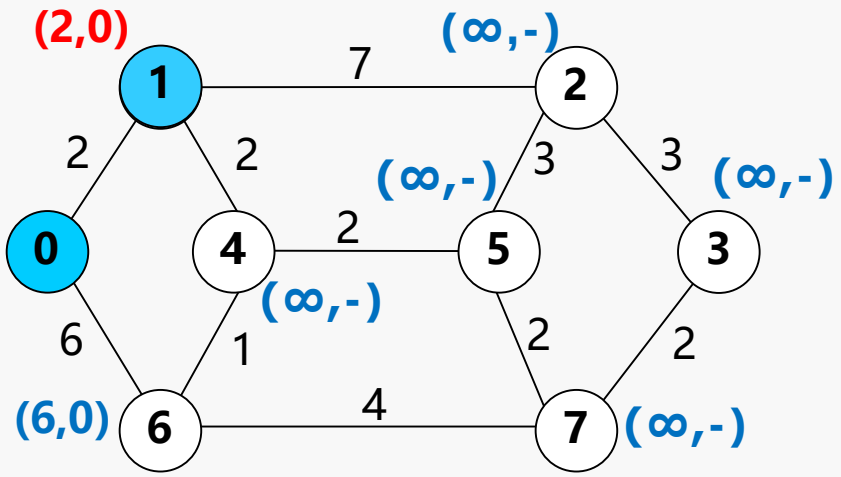
0 1 4 5 7 3

Dijkstra算法程序实现要点

待解问题1：寻找 T 中的点 Y 使其到 **源点** 距离最小

解决方案：将 T 中与 S 中的点**相邻接的顶点**放入优先队列PQueue，对优先队列执行出队操作就得到顶点 Y

优先队列：是指顶点出队列时，首先**找到其中距离源点最近的元素**，再**将这个元素交换到头部**，最后**将队头元素出队列**



队列操作：

1 6
出队 ⇩
6

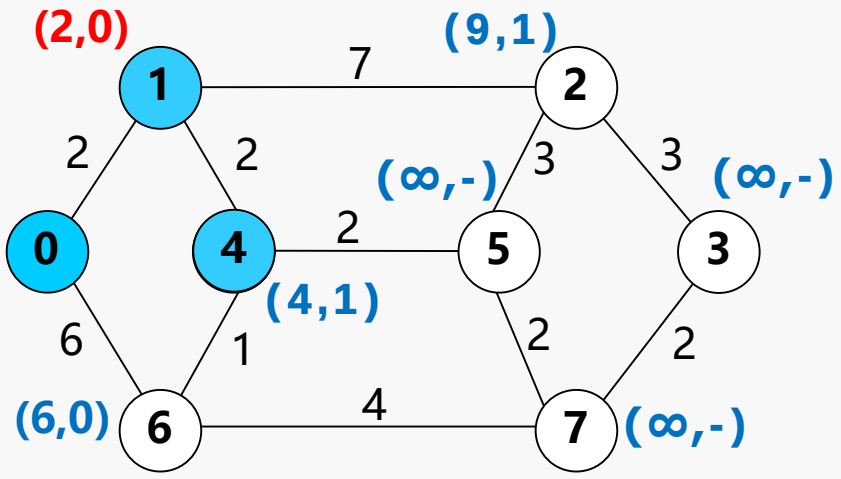
S ={0}，T={1,2,3,4,5,6,7} → S ={0,1}，T={2,3,4,5,6,7}

Dijkstra算法程序实现要点

待解问题1：寻找 T 中的点 Y 使其到 **源点** 距离最小

解决方案：将 T 中与 S 中的点相邻接的顶点放入优先队列PQueue，对优先队列执行出队操作就得到顶点 Y

优先队列：是指顶点出队列时，首先**找到其中距离源点最近的元素**，再**将这个元素交换到头部**，最后**将队头元素出队列**



队列操作：

6 2 4

只需入队顶点 Y 的属于 T 的邻接点

出队 2 6

S ={0,1}，T={2,3,4,5,6,7} → S ={0,1,4}，T={2,3,5,6,7}

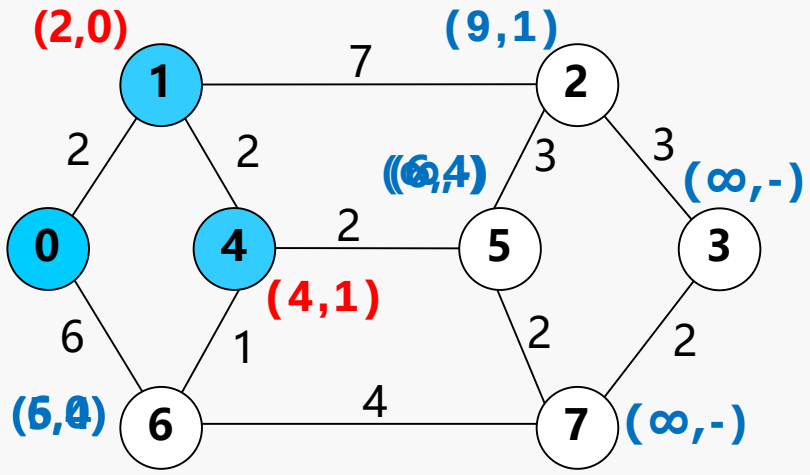
Dijkstra算法程序实现要点

待解问题2：找到点 Y 后，Y 将影响 T 中其他顶点到**源点**的距离，如何修改?

解决方案：扫描 T 中与 Y 点相邻的顶点（记作X），计算

$$D(X) = \mathrm{Min}[\, D(X),\ D(Y) + W(Y, X)\,]$$

设 D(X) 为 X 到 源点 的最新距离



原：D(6)=6

新：D(6)=D(4)+W(4,6)=4+1=5 < 6

原：D(5)= ∞

新：D(5)=D(4)+W(4,5)=4+2=6 < ∞

S ={0,1,4}，T={2,3,5,6,7}

Dijkstra算法程序实现要点

- 用结构体定义顶点数组d[], 结构体为：

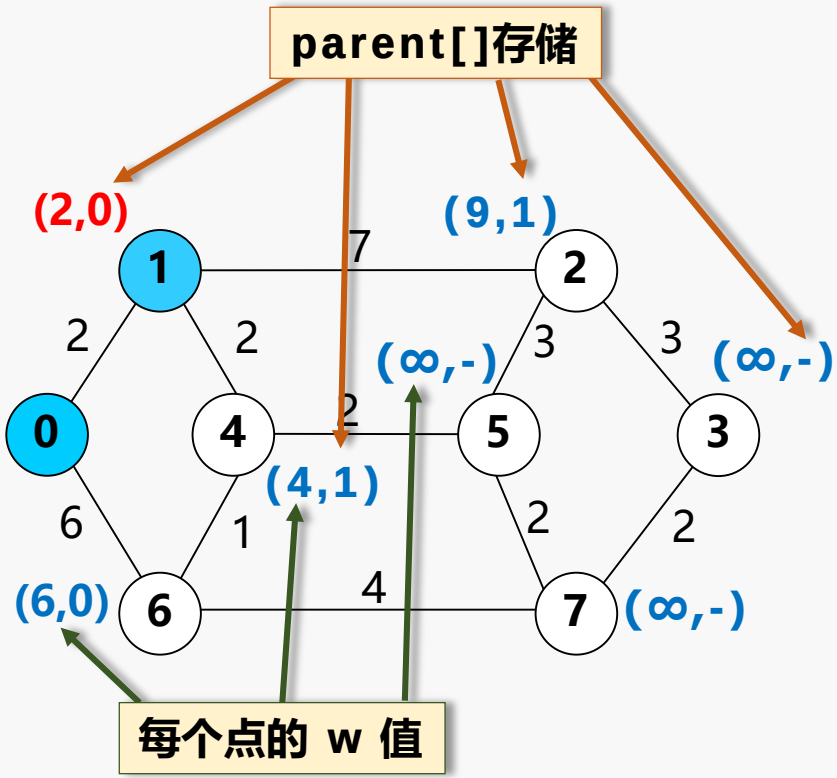
```
struct node
{
    int nodeID;     //顶点的序号
    int w;          //源点到该点的估算距离
};
```

- 用数组 parent[] 存储某个顶点在最短路径上的前驱

  例如 parent[1] 为 0 表示 1 号顶点的前驱为 0 号点

- 用数组 visited[] 表示某个顶点是否已找到最短路径

  若visited[5]为 false 则表示 5 号顶点尚未找到最短路径



S ={0,1}，T={2,3,4,5,6,7}

visited[0]、visited[1]为 true，其他为 false

Dijkstra算法程序实现概要

```
d[0].w = 0;                          // 源点到源点最短路权值为0
将 d[0] 放入pqueue;                  // 0号点入队列
while ( pqueue不为空 ) {             // 算法核心，队列空说明操作完成
    顶点从 pqueue 出队放入cd;        // 取最小估算距离顶点出队
    int u = cd.nodeID;               // 取顶点编号
    visited[u] = true;               // 顶点加入 S 集合
    p = G[u].firstedge;              // p为u的第一个邻接点的指针
    while (p != NULL)                // 遍历相邻顶点，更新估算距离，邻接点入队列
    {
        int v = p->adjvex;           // 取邻接顶点编号
        if (!visited[v] && d[v].w > d[u].w + p->weight)
        {
            d[v].w = d[u].w + p->weight;      // 更新估算距离
            parent[v] = u;    // 顶点 v 前驱为 u
            将顶点 d[v] 放入pqueue;
        }
        p = p->next;                 // 取下一个邻接点指针
    }
}
```

问题求解与实践

——挖金矿问题

主讲教师：陈雨亭、沈艳艳

挖金矿问题

- 从前有个国家，所有的国民都很诚实、正义
- 某天他们在自己的国家发现了十座金矿，并且这十座金矿在地图上排成一条直线，国王知道这个消息后非常高兴，他希望能够把这些金子都挖出来造福国民
- 首先他把这些金矿按照在地图上的位置从西至东进行编号，依次为0、1、2、3、4、5、6、7、8、9，然后他命令他的手下去对每一座金矿进行勘测，以便知道挖取每一座金矿需要多少人力以及每座金矿能够挖出多少金子，接着动员国民来挖金子

挖金矿问题

- 已知1：挖每一座金矿需要的人数是固定的，多一个人少一个人都不行。国王知道每个金矿各需要多少人手，金矿i需要的人数为peopleNeeded[i]
- 已知2：每一座金矿所挖出来的金子数是固定的，当第i座金矿有peopleNeeded[i]人去挖的话，就一定能恰好挖出gold[i]个金子。否则一个金子都挖不出来
- 已知3：开采一座金矿的人完成开采工作后，他们不会再次去开采其它金矿，因此一个人最多只能使用一次

挖金矿问题

- 已知4：国王在全国范围内仅招募到了10000名愿意为了国家去挖金子的人，这些人可能不够把所有的金子都挖出来，但是国王希望挖到的金子越多越好
- 已知5：这个国家的每一个人都很老实（包括国王），不会私吞任何金子，也不会弄虚作假，不会说谎话
- 已知6：国王只想知道最多可以挖出多少金子，而不用关心哪些金矿挖哪些金矿不挖

一个具体的运行结果

```
please enter the number of total people:
10000↙
please enter the number of gold mine:
10↙
please enter the number of gold each of gold mine has:
800 1500 6700 5800 5678 1200 500 900 7000 8888↙
please enter the number of people each of gold mine needs:
50 70 800 760 800 120 30 49 1000 1500↙
the max number of gold:38966
```

挖金矿问题

- 国王来到了第9个金矿的所在地，他的臣子告诉他，如果要挖取第9个金矿，需要1500个人，第9个金矿可以挖出8888个金子。
- 然后国王叫来甲、乙两个大臣，请甲大臣求出：若挖取第9个金矿，则用剩下的8500人，去挖取剩下9个金矿，最多能挖取多少金子。请乙大臣求出：若不挖取第9个金矿，则用10000人，去挖取剩下9个的金矿，最多能挖取多少金子。国王心想：“如此，若甲大臣求得，用剩下的人去挖剩下的金矿最多挖出x个金子，则所有金矿最多能挖出x+8888个金子。若乙大臣求得，用剩下的人去挖剩下的金矿最多挖出y个金子，则所有金矿最多能挖出y个金子。而我只需取x+8888和y中的较大值。”

挖金矿问题

- 然后呢?
- 两个大臣又各找来了两个人...

挖金矿问题

- 最后，当被问到给你z个人和仅有第0座金矿时最多能挖出多少金子时，就不再需要别人的帮助了
- 因为你知道，如果z大于等于挖取第0座金矿所需要的人数的话，那么挖出来的最多金子数就是第0座金矿能够挖出来的金子数，如果这z个人不够开采第0座金矿，那么能挖出来的最多金子数就是0，因为这唯一的金矿不够人力去开采
- 让我们为这些不需要别人的帮助就可以准确地得出答案的人们鼓掌吧，这就是传说中的底层劳动人民!

挖金矿问题

- 解决挖金矿问题的核心思想—动态规划
- 什么是动态规划?
- 动态规划的基本概念和性质?

什么是动态规划

- 动态规划算法通常用于求解具有某种最优性质的问题
- 在这类问题中，可能会有许多可行解。每一个解都对应于一个值，我们希望找到具有最优值的解
- 动态规划算法其基本思想也是将待求解问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解

什么是动态规划

- 但如果只是单纯进行问题分解的话，得到的子问题数目太多，有些子问题被重复计算了很多次
- 如果我们能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，这样就可以避免大量的重复计算，节省时间
- 我们可以用一个表来记录所有已解的子问题的答案。不管该子问题以后是否被用到，只要它被计算过，就将其结果填入表中。这就是动态规划法的基本思路

回顾挖金矿问题

- **子问题**：国王需要根据两个大臣的答案以及第9座金矿的信息才能判断出最多能够开采出多少金子。为了解决自己面临的问题，他需要给别人制造另外两个问题，这两个问题就是子问题
- **最优子结构**：国王相信，只要他的两个大臣能够回答出正确的答案（对于考虑能够开采出的金子数，最多的也就是最优的同时也就是正确的），再加上他的聪明的判断就一定能得到最终的正确答案。我们把这种子问题最优时母问题通过优化选择后一定最优的情况叫做“最优子结构”
- **重叠子问题**：实际上国王也好，大臣也好，所有人面对的都是：给你一定数量的人，给你一定数量的金矿，让你求出能够开采出来的最多金子数。在这个过程中，会有很多情况他们所面对的问题的人数和金矿数是一样的，也就是他们面对的问题是一样的。即：不是每次遇到的问题都是新问题，有些子问题出现多次。这就是“重叠子问题”

回顾挖金矿问题

- **边界**：想想如果不存在前面我们提到的那些底层劳动者的话这个问题能解决吗？永远都不可能！我们把这种子问题在一定时候就不再需要提出子问题的情况叫做边界，没有边界就会出现死循环
- **子问题独立**：当国王的两个大臣在思考他们自己的问题时他们是不会关心对方是如何计算怎样开采金矿的，因为他们知道，国王只会选择两个人中的一个作为最后方案，另一个人的方案并不会得到实施，因此一个人的决定对另一个人的决定是没有影响的。我们把这种一个母问题在对子问题选择时，当前被选择的子问题两两互不影响的情况叫做“子问题独立”

回顾挖金矿问题

- 如何求解?
  - 递归：对于每个金矿选择挖还是不挖，也就是要有$2^{10}=1024$种情况
  - 动态规划：共有10000个人和10个金矿，一共有10000*10个问题
- 递归的计算量是按指数级增长的，而动态规划的方法是按多项式级增长的
- **备忘录方法**(动态规划的变形)
  - 当我们在求解过程中，其实有很多子问题是不会出现的，我们没必要求出所有的子问题的最优解
  - 我们可以为每个子问题建立一个记录项，在第一次计算时做记录，以后直接读取

能使用动态规划求解的问题的性质

- **最优化原理**：如果问题的最优解所包含的子问题的解也是最优的，就称该问题具有最优子结构，即满足最优化原理。如：挖金矿问题中，国王能计算出所有金矿最多能挖出多少金子，建立在大臣能正确计算出剩下的金矿最多能挖出多少金子的基础上
- **无后效性**：即某阶段状态一旦确定，就不受这个状态以后决策的影响。也就是说，某状态以后的过程不会影响以前的状态，只与当前状态有关。如：若确定要对第9座金矿进行挖取时，第8个金矿是否挖取对其没有影响
- **重叠子问题**：即每次产生的子问题不总是新问题，有些子问题可能会反复出现多次。动态规划算法正是利用了该性质，从而获得较高的运算效率。（该性质并不是动态规划适用的必要条件，但是如果没有这条性质，动态规划算法同其他算法相比就不具备优势）。如：在挖金矿问题中，一个people和mineNum的组合，可能在不同的子问题中会出现多次

问题求解与实践
——0-1背包问题

主讲教师：陈雨亭、沈艳艳

问题描述

- 小明要远行，他有一个容量为15kg的背包，另外有4个物品，物品的质量和价值如下表所示：

物品	A1	A2	A3	A4
质量(kg)	3	4	5	6
价值	4	5	6	7

- 小明希望能用他的背包带走的物品总价值最大，你能告诉他应该怎么做吗？这就是一个0-1背包问题

问题分析

- 假设A1不放进包中，在剩余物品中选择，包中能放的物品最大价值是y
- 如果A1放进去，用剩余的容积可以放的剩余物品的最大价值是x，那么此时包中物品的总价值就是x+4
- 这时，我们只需取y和x+4的最大值
- 至此，你有没有发现，这个问题和我们前面所讲得挖金矿例子很像，只是这里要求我们给出选择方案，实际上它们都是0-1背包问题

物品	A1	A2	A3	A4
质量(kg)	3	4	5	6
价值	4	5	6	7

问题分析

- 一般的0-1背包问题
- 给定n种物品和一背包。物品i的重量是w_i，其价值是v_i，背包的容量是c。

  问：应如何选择装入背包中的物品，使得背包中物品的总价值最大?

问题分析

- 如果我们把物品i装入背包和不装入背包分别用$x_i=1$和$x_i=0$表示，那么每个物品是否装入背包中可以用一个n元向量$(x_1,x_2,\ldots\ldots,x_n)$表示，则装入背包的物品的总重量可以用 $\sum_{i=1}^{n} w_i x_i$ 表示，总价值可以用 $\sum_{i=1}^{n} v_i x_i$ 表示。那么这个问题就转换为这样一个问题：怎么在满足 $\sum_{i=1}^{n} w_i x_i \le c$ 的条件下，使得 $\sum_{i=1}^{n} v_i x_i$ 达到最大，即：

$$\max \sum_{i=1}^{n} v_i x_i \quad \begin{cases} \sum_{i=1}^{n} w_i x_i \le c \\ x_i \in \{0,1\}, 1 \le i \le n \end{cases}$$

问题分析

- 由上面的例子，我们可以发现0-1背包问题具有最优子结构性质
- 我们设$(y_1,y_2,\ldots\ldots,y_n)$是所给0-1背包问题的一个最优解，则$(y_2,y_3,\ldots\ldots,y_n)$是下面子问题的一个最优解：

$$\max\sum_{i=2}^{n}v_i x_i \qquad \begin{cases}\sum_{i=2}^{n}w_i x_i \le c-w_1 y_1\\ x_i\in\{0,1\},2\le i\le n\end{cases}$$

问题分析

- 如何证明?
- 我们假设$(z_2, z_3, \ldots\ldots, z_n)$ 是上面子问题的一个最优解，而(y_2,y_3,……,y_n)不是它的最优解。
- 那么 $\sum_{i=2}^{n} v_i z_i > \sum_{i=2}^{n} v_i y_i$ 且 $w_1 y_1 + \sum_{i=2}^{n} w_i z_i \le c$ 。因此有：

$$v_1 y_1 + \sum_{i=2}^{n} v_i z_i > \sum_{i=2}^{n} v_i y_i \quad \text{且} \quad w_1 y_1 + \sum_{i=2}^{n} w_i z_i \le c$$

- 这说明(y_1,z_2,……,z_n)是0-1背包问题的一个更优解，而 (y_1,y_2,……,y_n)不是最优解，产生矛盾。

问题分析

- 设所给的0-1背包问题的子问题

$$\max \sum_{k=i}^{n} v_k x_k \quad \begin{cases} \sum_{k=i}^{n} w_k x_k \le j \\ x_k \in \{0,1\}, i \le k \le n \end{cases}$$

的最优值是m(i,j)，即m(i,j)是背包容量为j，可选择物品为i，i+1，……，n时0-1背包问题的最优值

- 由0-1背包问题的最优子结构性质，可以建立计算m(i,j)的递归式如下：

$$m(i, j) = \begin{cases} \max\{m(i+1, j), m(i+1, j-w_i) + v_i\} & j \ge w_i \\ m(i+1, j) & 0 \le j \le w_i \end{cases}$$

其中

$$m(n, j) = \begin{cases} v_n & j \ge w_n \\ 0 & 0 \le j \le w_n \end{cases}$$

问题分析

- 有了这个递推关系，我们可以很快的计算出问题的最优值，但是这里要求给出最优方案，所以我们需要进行下一步，构造最优解
- 如何构造最优解呢?
- 由上面的递推关系，我们可知，若m(i,j)=m(i+1,j)，表示物品i没有装入背包时子问题最优，这时我们记$x_i=0$
- 若m(i,j)=m(i+1,j-w_i)+v_i表示物品i装入背包时子问题是最优解，这时我们记$x_i=1$
- 最后得到一个n元向量X(x_1,x_2,……,x_n)，就是问题的最优解

0-1背包问题关键代码

```
// 寻找每个子问题的最优解
// c[i][j]表示背包容量为j，可选择物品为i,i+1,……,n时的最优值，0-1背包问题的最优值为c[1][m]
void knapsack()
{
        int jMax=min(w[n]-1,m); //背包可承受最大重量和物体n的重量-1中较小数
        int i,j;
        //初始化c[n][j]
        //当j<jMax时，c[n][j]为0；
        //例：假设jMax = w[n]-1，即背包可以容纳物体n，则在j<=jMax时，c[n][j]=0，w[n]<=j<=m时，
c[n][j]=v[n]
        for(j=0;j<=jMax;j++)   {       c[n][j]=0;      }
        for(j=w[n];j<=m;j++)   {       c[n][j]=v[n];   }

        for(i=n-1;i>1;i--)          {
                jMax=min(w[i]-1,m);
                for(j=0;j<=jMax;j++) c[i][j]=c[i+1][j];
                for(j=w[i];j<=m;j++) c[i][j]=max(c[i+1][j],c[i+1][j-w[i]]+v[i]);
        }
        c[1][m]=c[2][m];
        if(m>=w[1])c[1][m]=max(c[1][m],c[2][m-w[1]]+v[1]);
}
```

0-1背包问题关键代码

```
/*
构造最优解
*/
void traceBack(int *x)
{
        int i,j;
        j=m;
        for(i=1;i<n;i++)
        {
                if(c[i][j]==c[i+1][j]) x[i]=0;
                else { x[i]=1;j-=w[i];}
        }
        x[n]=(c[n][j])?1:0;
}
```

问题求解与实践
——最长公共子序列问题

主讲教师：陈雨亭、沈艳艳

问题描述

- 一个给定序列的子序列是在该序列中删去若干元素后得到的序列
- 给定两个序列X和Y，当另一序列Z既是X的子序列又是Y的子序列时，称Z是序列X和Y的公共子序列
- **最长公共子序列**: 公共子序列中长度最长的子序列

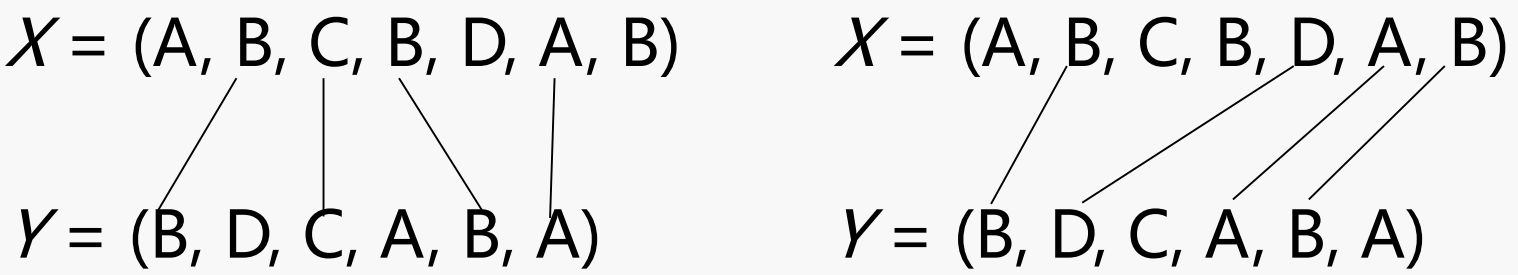
问题描述

- **最长公共子序列问题**
  - 给定两个序列X={x_1,x_2,…,x_m}和Y={y_1,y_2,…, y_n}，找出X和Y的一个最长公共子序列
- 设 X = (A, B, C, B, D, A, B)
- X的**子序列**
  - 所有X的子集(集合中元素按原来在X中的顺序排列)
  - 如：(A, B, D), (B, C, D, B), 等等

问题描述

X = (A, B, C, B, D, A, B)

Y = (B, D, C, A, B, A)



- (B, C, B, A)和(B, D, A, B)都是X和Y的最长公共子序列(长度为4)
- 但是,(B, C, A)就不是X和Y的最长公共子序列

可否用穷举法？

- 对于每一个X_m的子序列,验证它是否是Y_n的子序列
- X_m有2^m个子序列
- 每个子序列需要o(n)的时间来验证它是否是Y_n的子序列
- 从Y_n的第一个字母开始扫描下去,如果不是则从第二个开始
- 运行时间: $o(n2^m)$

问题分析

步骤1

- 给定一个序列$X_m = (x_1, x_2, \ldots, x_m)$,我们定义$X_m$的第$i$个前缀为:

$$X_i = (\ x_1, x_2, \ldots, x_i\ )\quad i = 0, 1, 2, \ldots, m$$

$c[i, j]$为序列$X_i = (x_1, x_2, \ldots, x_i)$和$Y_j = (y_1, y_2, \ldots, y_j)$的最长公共子序列的长度.

步骤2 **最优子结构性质：**

设序列$X_m=\{x_1,x_2,\ldots,x_m\}$和$Y_n=\{y_1,y_2,\ldots,y_n\}$的一个最长公共子序列为$Z_k=\{z_1,z_2,\ldots,z_k\}$，则

1.若$x_m=y_n$，则$z_k=x_m=y_n$，且Z_{k-1}是X_{m-1}和Y_{n-1}的最长公共子序列。

2.若$x_m\neq y_n$，且$z_k\neq x_m$，则Z_k是X_{m-1}和Y_n的最长公共子序列。

3.若$x_m\neq y_n$，且$z_k\neq y_n$，则Z_k是X_m和Y_{n-1}的最长公共子序列。

问题分析

- 我们用f[i][j]记录序列X_i和Y_j的最长公共子序列的长度。其中X_i={ $x_1,x_2,\ldots\ldots,x_i$}，Y_j={ $y_1,y_2,\ldots\ldots,y_j$}。当i=0或者j=0时，X_i和Y_j的最长公共子序列为空，所以此时f[i][j]=0。所以我们可以建立如下的递推关系：

$$f[i][j]=\begin{cases}0 & i=0,j=0\\ f[i-1][j-1]+1 & i,j>0;x_i=y_j\\ \max\{f[i][j-1],f[i-1][j]\} & i,j>0;x_i\neq y_j\end{cases}$$

问题分析

◆我们用数组f[i][j]记录序列X_i和Y_j的最长公共子序列的长度，因为我们需要求序列的最长公共子序列，因此需要在计算最优值的同时记录最优值是由哪些子问题得到的，我们引入数组b[i][j]用来记录f[i][j]是通过哪个子问题的解得到的。

◆假设f[i][j]=f[i-1][j-1]+1 ，即此时有$x_i=y_j$，我们用b[i][j]=’ ↖’ ，来表示。

◆f[i][j]=f[i-1][j]，即此时有$x_i \neq y_j$且f[i-1][j]>f[i][j-1]，我们用b[i][j]=’ ↑’ ，来表示。

◆f[i][j]=f[i][j-1]，即此时有$x_i \neq y_j$且f[i][j-1]>f[i-1][j]，我们用b[i][j]=’ ←’ ，来表示。

◆那么当计算出问题的最优值，即序列X和序列Y的最长公共子序列时，即可得一个记录每一个子问题的最优解是怎样得来的二维数组b，而问题的最优值，即最长公共子序列的长度记录在f[m][n]中。

问题分析

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

关键代码

```
//根据递推关系，求最优值，并记录相关信息。
//f[i][j]记录序列Xi和Yj的最长公共子序列的长度
//b[i][j]用来记录f[i][j]是通过哪个子问题的解得到的
void LCSLength(char *x, char *y, int m, int n, int f[][MAXLEN], int b[][MAXLEN])
{
        int i, j;
        for (i = 1; i <= m; i++)  f[i][0] = 0;
        for (j = 1; j <= n; j++)   f[0][j] = 0;
        for (i = 1; i <= m; i++) {
                for (j = 1; j <= n; j++) {
                        if (x[i-1] == y[j-1]) {
                                f[i][j] = f[i - 1][j - 1] + 1;
                                b[i][j] = 0;
                        }
                        else if (f[i - 1][j] >= f[i][j - 1]) {
                                f[i][j] = f[i - 1][j];
                                b[i][j] = 1;
                        }
                        else {
                                f[i][j] = f[i][j - 1];
                                b[i][j] = -1;
                        }
                }
        }
}
```