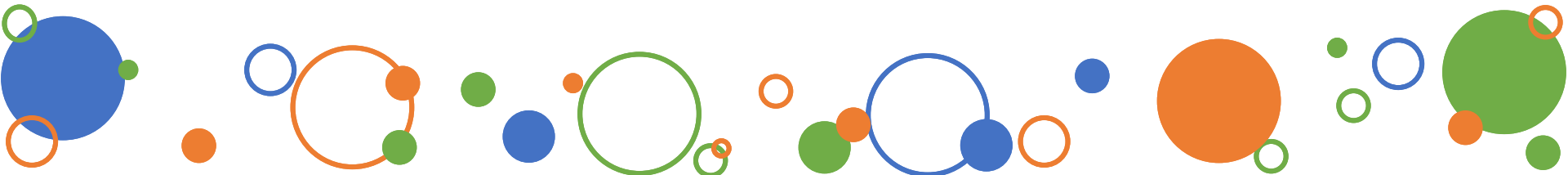
DiCA

Data Intelligence & Computing Art Laboratory

算法分析

刘权辉

2024




四川大学
SICHUAN UNIVERSITY
1896


DiCA
Data Intelligence & Computing Art Laboratory

第六章：分支限界法

学习要点

- 理解分支限界法的剪枝搜索策略
- 掌握分支限界法的算法框架
  - 队列式(FIFO)分支限界法
  - 优先队列式分支限界法
- 通过应用范例学习分支限界法的设计策略
  - 装载问题；
  - 0-1背包问题；
  - 单源点最短路径问题；
  - 最大团问题。

引言：回溯法

- 回溯法在问题的解空间树中，按深度优先策略，从根结点出发搜索解空间树。算法搜索至解空间树的任意一点时，先判断该结点是否包含问题的解。如果肯定不包含，则跳过对该结点为根的子树的搜索，逐层向其祖先结点回溯；否则，进入该子树，继续按深度优先策略搜索。

- 求解目标：所有可行解（满足某些约束条件）
- 搜索方式：深度优先搜索
- 不撞南墙不回头（回溯）

分支限界法的基本思想

- 分支限界法在问题的解空间树中，按广度优先策略或最小耗费优先，从根结点出发搜索解空间树。每个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有子结点。在这些子结点中，导致不可行解或导致非最优解的子结点被舍弃，其余子结点被加入活结点表中。接着，不断取出活结点，一直持续到找到所需的解或活结点表为空时为止。

- 解空间树：一般和回溯法的解空间树相同
- 求解目标：一个可行解或者某个最优解
- 搜索方式：广度优先搜索或最小耗费优先搜索

分支限界法的基本思想

口 常见分支限界法（活结点表中选择下一扩展节点方式）

① 队列（FIFO）分支限界法

- 按FIFO原则从活结点表中选取节点做判断，

② 优先队列分支限界法

- 按优先队列的优先级策略从活结点表中选取节点做判断，
- 实现时一般用最大（最小）堆实现。

DiCA

Data Intelligence & Computing Art Laboratory

6.1 装载问题

装载问题

口 问题描述：

- 有一批共n个集装箱要装上2艘载重量分别为c_1和c_2的轮船，其中集装箱 i 的重量为w_i，且：

$$\sum_{i=1}^{n} w_i \le c_1 + c_2$$

- 装载问题要求确定是否有一个合理的装载方案可将这n个集装箱装上这2艘轮船。如果有，找出一种装载方案。

口 分析：

- 容易证明，如果一个给定装载问题有解，则采用下面的策略可得到最优装载方案：

  1)首先将第一艘轮船尽可能装满；

  2)将剩余的集装箱装上第二艘轮船。

分析：

- 将第一艘轮船尽可能装满等价于选取全体集装箱的一个子集，使该子集中集装箱重量之和最接近c_1。
- 由此可知，装载问题等价于以下特殊的0-1背包问题：

$$\max \sum_{i=1}^{n} w_i x_i$$

$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \le c_1$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

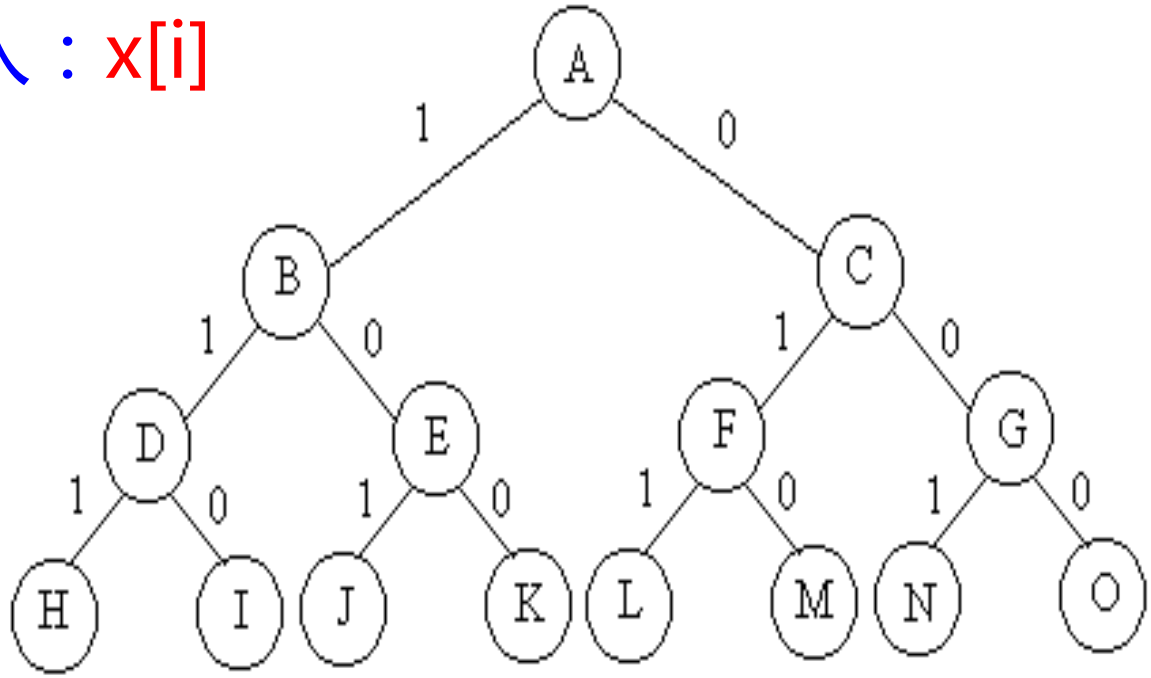
注意：

- 用分支限界法设计解装载问题时，算法复杂度受限于活结点队列的时间和空间复杂度。

分析：

- 解空间：子集树
- 可行性约束函数(选择当前元素)：$\sum_{i=1}^{n} w_i x_i \le c_1$
- 当前扩展节点载重量：Ew；当前最优载重量bestw；

  货物 i 是否装入：x[i]



□ FIFO队列分支限界法实现：

```
template < class Type > Type MaxLoading ( Type  w[] , Type c , int n )
{       //返回最优载重量 ,初始化
        Queue < Type > Q ;                          //活结点队列
        Q . Add(-1) ;                               //同层结点尾部标志
        int i = 1 ;                                 //当前扩展结点所在层
        Type Ew = 0 ,  bestw = 0 ;                  //扩展结点载重量,当前最优载重
        while(true) {                          //搜索子集空间树
            if(Ew+w[i]≤c)                                  //x[i] = 1
                EnQueue(Q, Ew+w[i], bestw, i, n) ;    //入队
            EnQueue(Q, Ew, bestw, i, n) ;              //x[i] = 0，右节点始终满足条件
            Q.Delete(Ew) ;                             //取下一扩展结点
            if(Ew = = -1) {                            //同层结点尾部
                if(Q.IsEmpty())    return bestw ;
                Q.Add(-1) ;                            //同层结点尾部标志
                Q.Delete(Ew) ;                         //取下一扩展结点
                i++ ;  }                               //进入下一层
         }
}
```

检查左节点

□ FIFO队列分支限界法实现：

```
template < class Type > Type MaxLoading ( Type  w[] , Type c , int n )
{       //返回最优载重量 ,初始化
        Queue < Type >  Q ;                          //活结点队列
        Q . Add(-1) ;                                //同层结点尾部标志
        int i = 1 ;                                  //当前扩展结点所处的层
        Type Ew = 0 ,  bestw = 0 ;  //扩展结点所相应的载重量,当前最优载重量
        while(true) {                          //搜索子集空间树
             if(Ew+w[i]≤c)                           //x[i] = 1
                  EnQueue(Q, Ew+w[i], bestw, i, n) ;
             EnQueue(Q, Ew, bestw, i, n) ;          //x[i] = 0，右节点始终满足条件
             Q.Delete(Ew) ;                         //取下一扩展结点
             if(Ew = = -1) {                        //同层结点尾部
                  if(Q.IsEmpty())   return bestw ;
                  Q.Add(-1) ;                       //同层结点尾部标志
                  Q.Delete(Ew) ;                    //取下一扩展结点
                  i++ ;  }                          //进入下一层
         }
}
```

右节点表示不用装入，不需检查

装载问题

□ FIFO队列分支限界法实现：

```
template < class Type > Type MaxLoading ( Type  w[] , Type c , int n )
{       //返回最优载重量 ,初始化
        Queue < Type >  Q ;                          //活结点队列
        Q . Add(-1) ;                                //同层结点尾部标
        int i = 1 ;                                  //当前扩展结点层
        Type Ew = 0 ,  bestw = 0 ;  //扩展结点所相应的载重
        while(true) {                          //搜索子集空间树
            if(Ew+w[i]≤c)                            //x[i] =
                EnQueue(Q, Ew+w[i], bestw, i+1);
            EnQueue(Q, Ew, bestw, i+1);              //x[i] = 0，右节点始终满足条件
            Q.Delete(Ew) ;                           //取下一扩展结点
            if(Ew = = -1) {                          //同层结点尾部
                if(Q.IsEmpty())   return bestw ;
                Q.Add(-1) ;                          //同层结点尾部标志
                Q.Delete(Ew) ;                       //取下一扩展结点
                i++ ;  }                             //进入下一层
         }
}
```

队列中存在每一层的结束标志，所以取元素时活结点队列一定不为空。只有当取出-1时，再行判断是否为空

装载问题

```
// i:解空间树的层数编号，初始为1
// Q:活结点队列
// bestw:当前最优解，初始为0
// EnQueue():活结点入队算法

template < class Type >
void EnQueue ( Queue < Type > & Q , Type wt , Type& bestw ,
int i , int n )
 {        //将活结点加入到活结点队列Q中
          //可行叶结点
         if ( i = = n ) {
                  if ( wt > bestw ) bestw = wt ;
          }
         else Q.Add(wt) ; //非叶结点
 }
```

到达叶节点，更新最优值

非叶节点，加入活结点队列

□ FIFO队列分支限界法：

时间复杂度：

- 当使用FIFO队列时，因为解空间树的节点为2^n个，所以时间和空间复杂度均为$O(2^n)$

□ FIFO队列分支限界法优化：

剪枝优化：提前检查右子树，提前更新左子树

- 设 **bestw** 是**当前最优解**；Ew是当前扩展结点相应的重量； **r是剩余集装箱的重量。**
- 分析易得：检查右子树，即使加上剩余集装箱重量，最终重量小于当前最优时，无需继续搜索子节点。即当 **Ew+r≤bestw** 时，可将其右子树剪去
- 另外，左子树表示装入当前货物，且只有搜索到叶节点时才会更新bestw。因此初始时，因为bestw=0,r>0,总有Ew+r>bestw。此时只需每次进入左子树时更新bestw，即可。

装载问题：FIFO队列分支限界法优化实现：

```
template < class Type >  Type MaxLoading(Type w[],Type c,int n) { //返回最优载重
    Queue < Type >  Q；                          //活结点队列
    Q.Add(-1)；                                  //同层结点尾部标志
    int i = 1 ；                                 //当前扩展结点所处的层
    Type Ew = 0，bestw = 0， r=0；               //扩展结点载重，当前最优载重 ，剩余集装箱重
    for(int j=2; j<=n; j++)     r += w[j]；              //初始化 r
    while（ true ） {                                    //搜索子集空间树
        Type wt = Ew+ w[i];
        if（ wt<=c ）{            //x[i]=1 检查左儿子结点
            if(wt>bestw)          bestw = wt;
            if(i<n)                   Q.Add(wt)；          }
        if（ Ew+r>bestw && i<n ）        Q.Add(Ew)；        //x[i]=0检查右儿子节点
        Q.Delete(Ew);                                        //取下一扩展结点
        if(Ew==-1) {                                         //同层结点尾部
            if( Q.IsEmpty())          return bestw ；
            Q.Add(-1)；                                  //同层结点尾部标志
            Q.Delete(Ew)；                               //取下一扩展结点
            i++ ；                                       //进入下一层
            r-=w[i];                                     //更新剩余重量
    } }
}
```

提前更新bestw，以便后续右子树剪枝

装载问题：FIFO队列分支限界法优化实现：

```
template < class Type >  Type MaxLoading(Type w[],Type c,int n) {     //返回最优载重
    Queue < Type >  Q ;                                  //活结点队列
    Q.Add(-1) ;                                          //同层结点尾部标志
    int i = 1 ;                                          //当前扩展结点所处的层
    Type Ew = 0 , bestw = 0 ,  r=0 ;                     //扩展结点载重，当前最优载重 ，剩余集装箱重
    for(int j=2; j<=n; j++)  {  r += w[j] ;   }          //初始化 r
    while ( true ) {                                     //搜索子集空间树
        Type wt = Ew+ w[i];
        if ( wt<=c ) {                                   //x[i]=1 检查左儿子结点
            if(wt>bestw)          bestw = wt;
            if(i<n)                  Q.Add(wt) ;  }
        if ( Ew+r>bestw && i<n )      Q.Add(Ew) ;        //x[i]=0检查右儿子节点
        Q.Delete(Ew);                                    //取下一扩展结点
        if(Ew==-1) {                                     //同层结点尾部
            if( Q.IsEmpty())        return bestw ;
            Q.Add(-1) ;                                  //同层结点尾部标志
            Q.Delete(Ew) ;                               //取下一扩展结点
            i++ ;                                        //进入下一层
            r-=w[i];                                     //更新剩余重量
     } }   }
```

右子树剪枝：若条件不满足，则将右子树剪去


四川大学
1896
SICHUAN UNIVERSITY


DiCA
Data Intelligence & Computing Art Laboratory

6.2 0-1背包问题

0-1背包问题

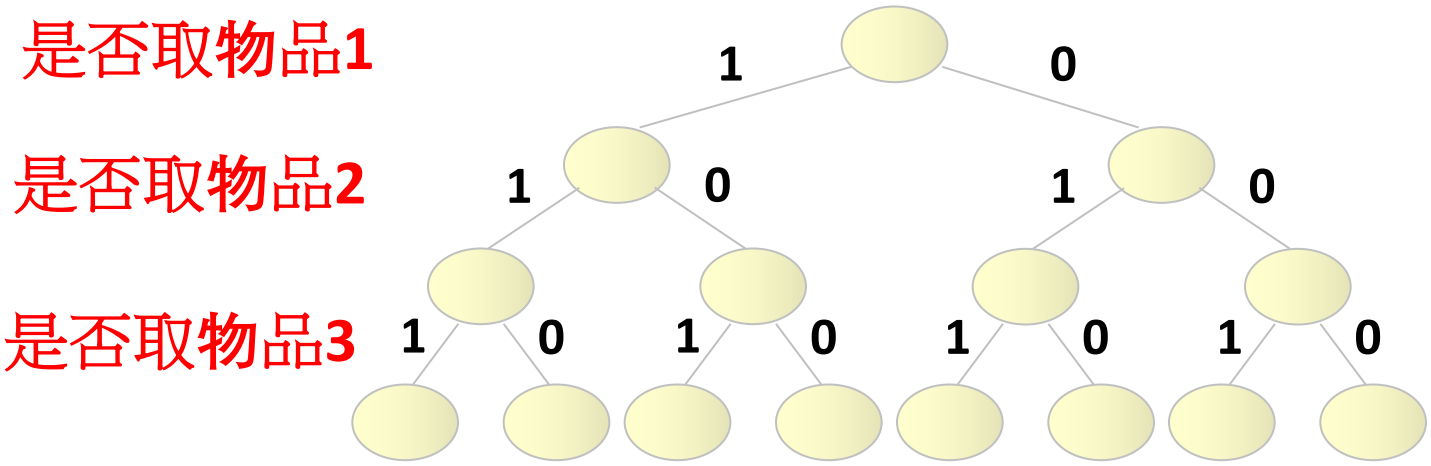
- **问题描述**：略
- **分析：**
  - 0-1背包问题也是一个子集选取问题（子集树）。
  - 例：有3个不可分割的商品，其重量与价值分别如下图示。若背包容量为10公斤（C=10），请利用分支限界法策略找出最佳解。

Item	重量 (W_i)	价值(P)	价值/重量
1	4	$40	$10
2	7	$42	$6
3	5	$25	$5

0-1背包问题

口 分析：

- 首先将货物按单位价值降序排列
- 三个商品的0-1背包问题的子集树如下：



0-1背包问题

口 分析：

- 首先将货物按单位价值降序排列,
- 三个商品的0-1背包问题的子集树,
- 采用优先队列的分支限界法，将背包可能的最大价值定义为优先队列的优先值：

$$ub = v + (W - w) * (v_i/w_i)$$

其中：

1）ub 是可能的背包最大价值，

2）v, w 是当前背包的价值和重量,

3）w_i 是商品i的重量, v_i 是商品i的价值，

4）W 是背包容量。

0-1背包问题

Item	重量 (w_i)	价值(V)	价值/重量
1	4	$40	$10
2	7	$42	$6
3	5	$25	$5

ub=v+(W-w)(v_i/w_i) ；

容量：W=10

0

w=0,v=0
ub=100

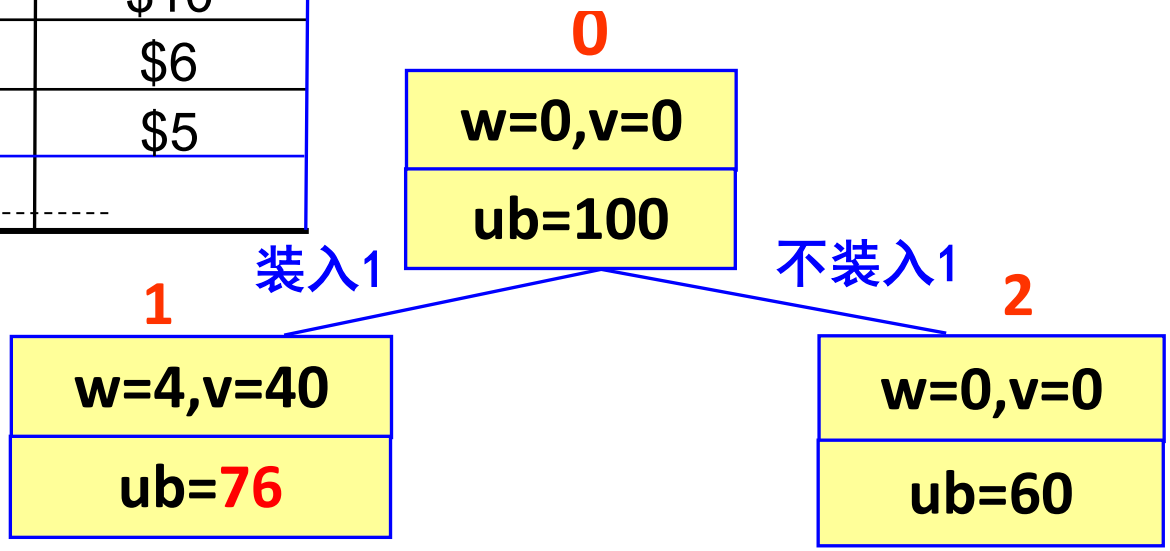
0-1背包问题

$ub=v+(W-w)(v_i/w_i)$ ；

$W=10$

Item	重量 (w_i)	价值(V)	价值/重量
1	4	$40	$10
2	7	$42	$6
3	5	$25	$5

是否取商品1



0-1背包问题

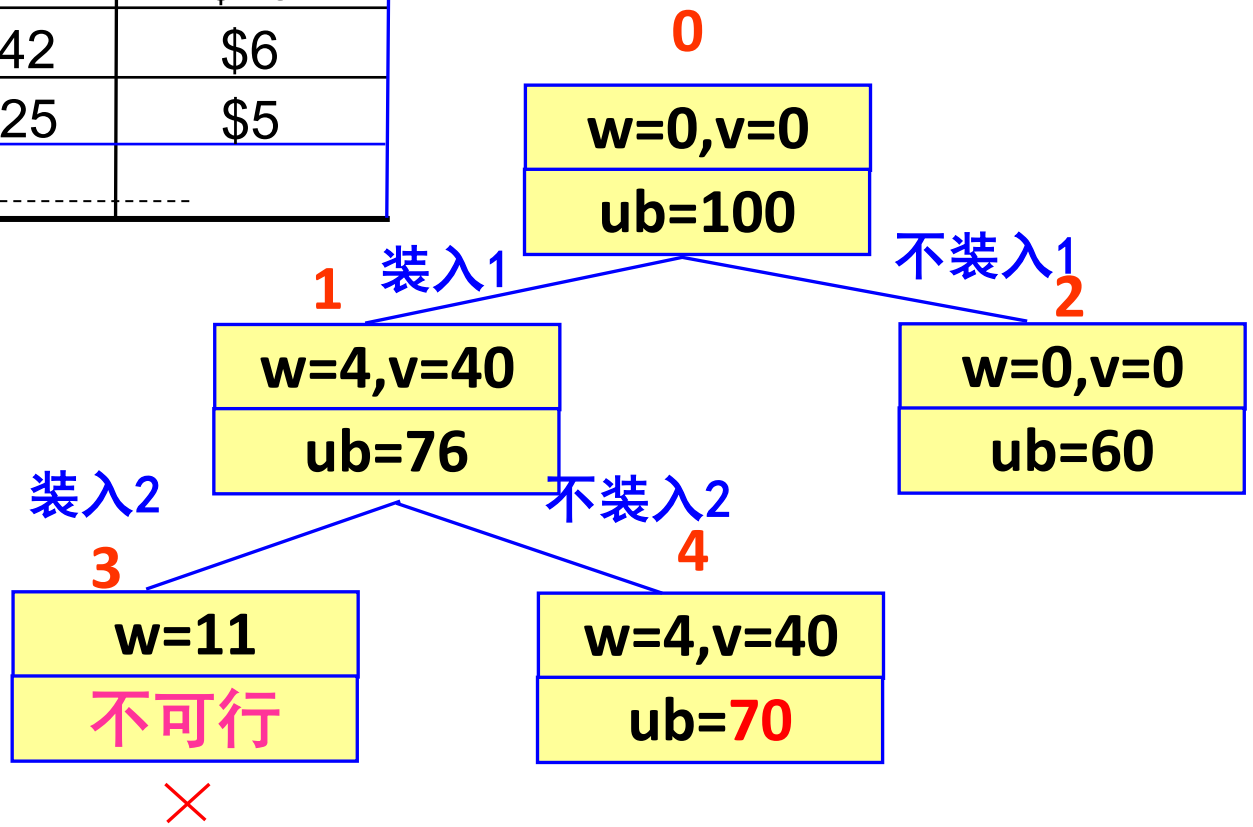
$ub = v + (W - w)(v_i / w_i)$ ；

$W = 10$

Item	重量 (w_i)	价值(V)	价值/重量
1	4	$40	$10
2	7	$42	$6
3	5	$25	$5

是否取商品1

是否取商品2



0-1背包问题

$ub = v + (W - w)(v_i / w_i)$ ；

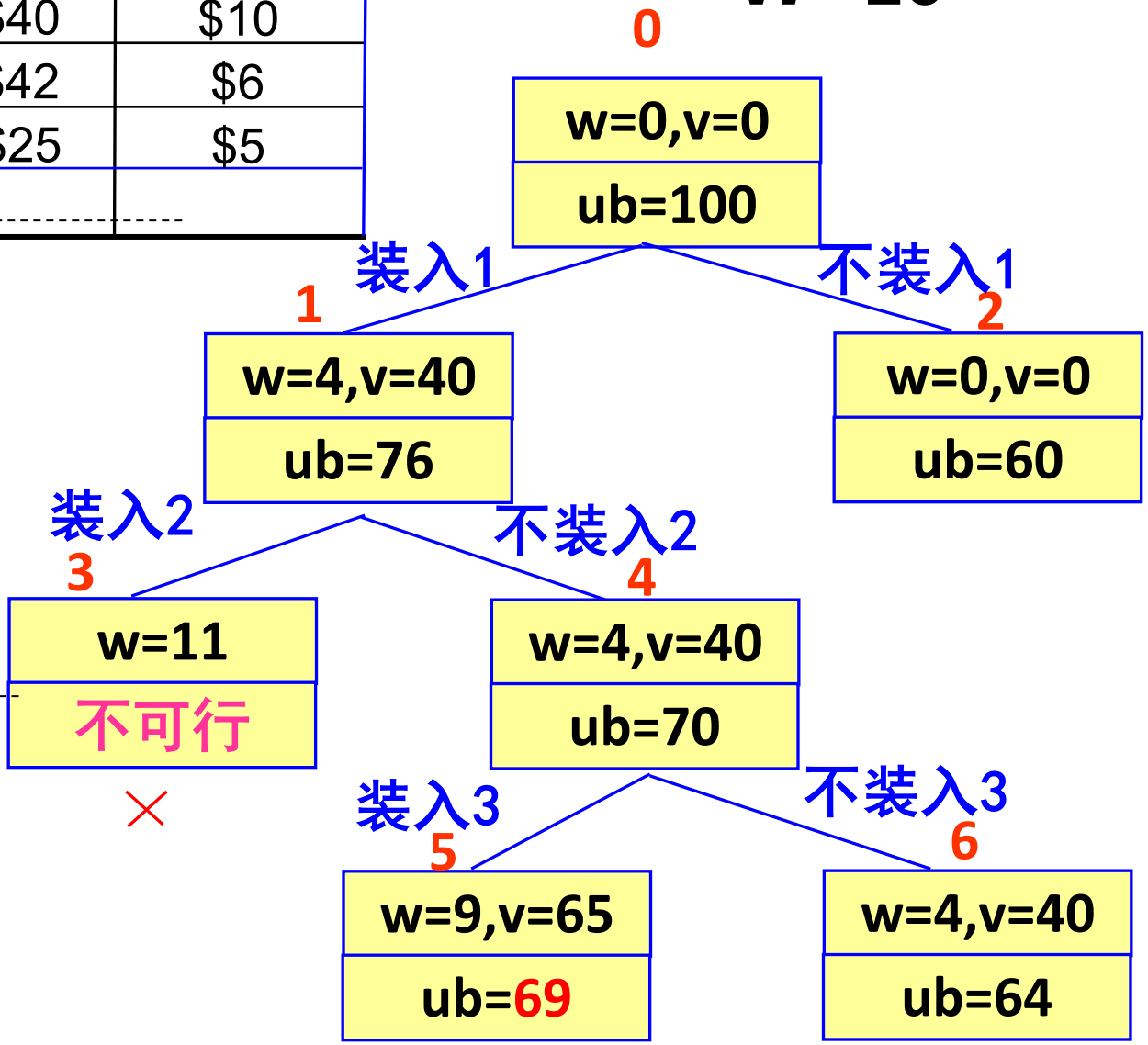
W=10

Item	重量 (w_i)	价值(V)	价值/重量
1	4	$40	$10
2	7	$42	$6
3	5	$25	$5

是否取商品1

是否取商品2

是否取商品3

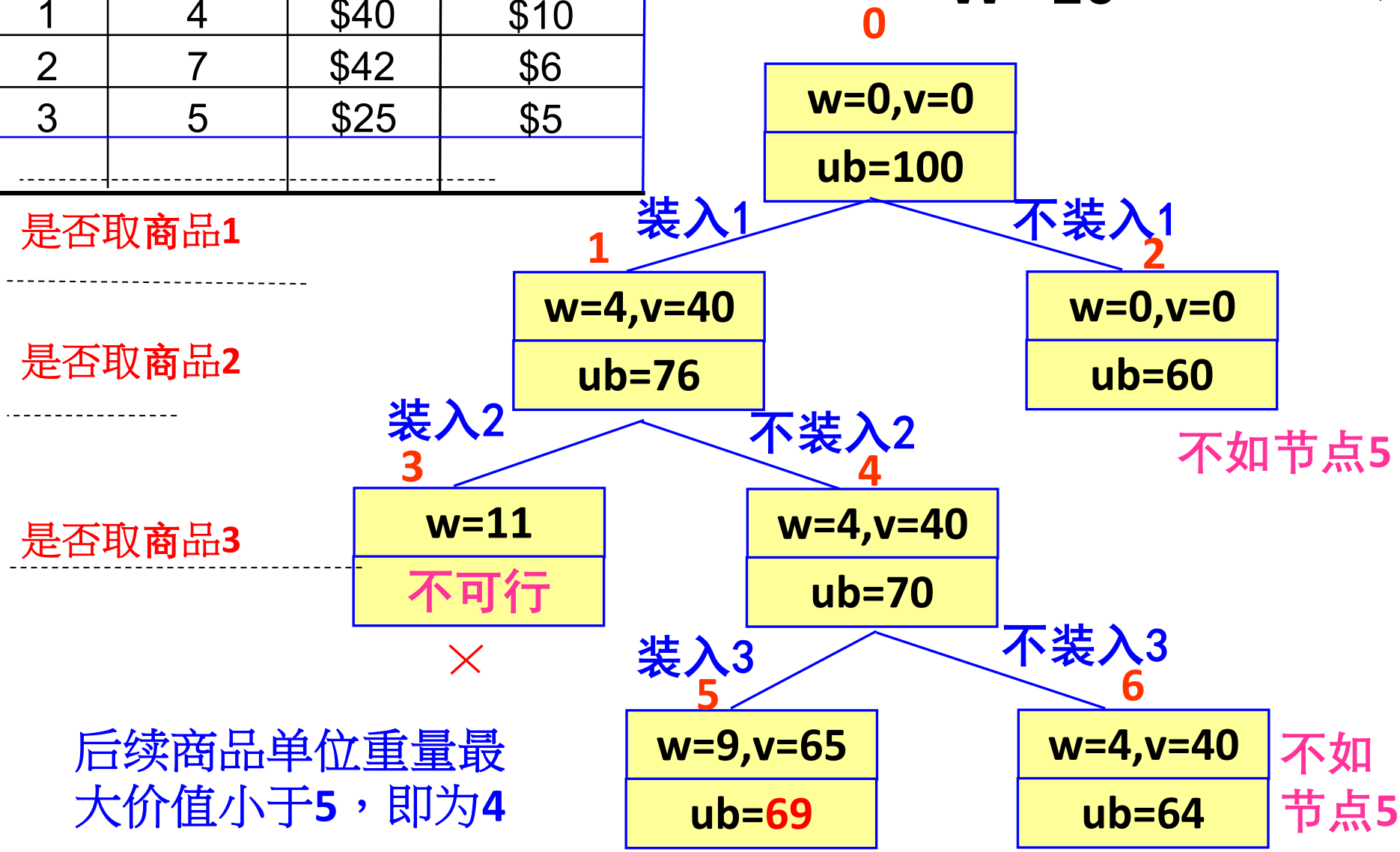


0-1背包问题

$ub = v + (W - w)(v_i / w_i)$ ；

$W = 10$

Item	重量 (w_i)	价值(V)	价值/重量
1	4	$40	$10
2	7	$42	$6
3	5	$25	$5

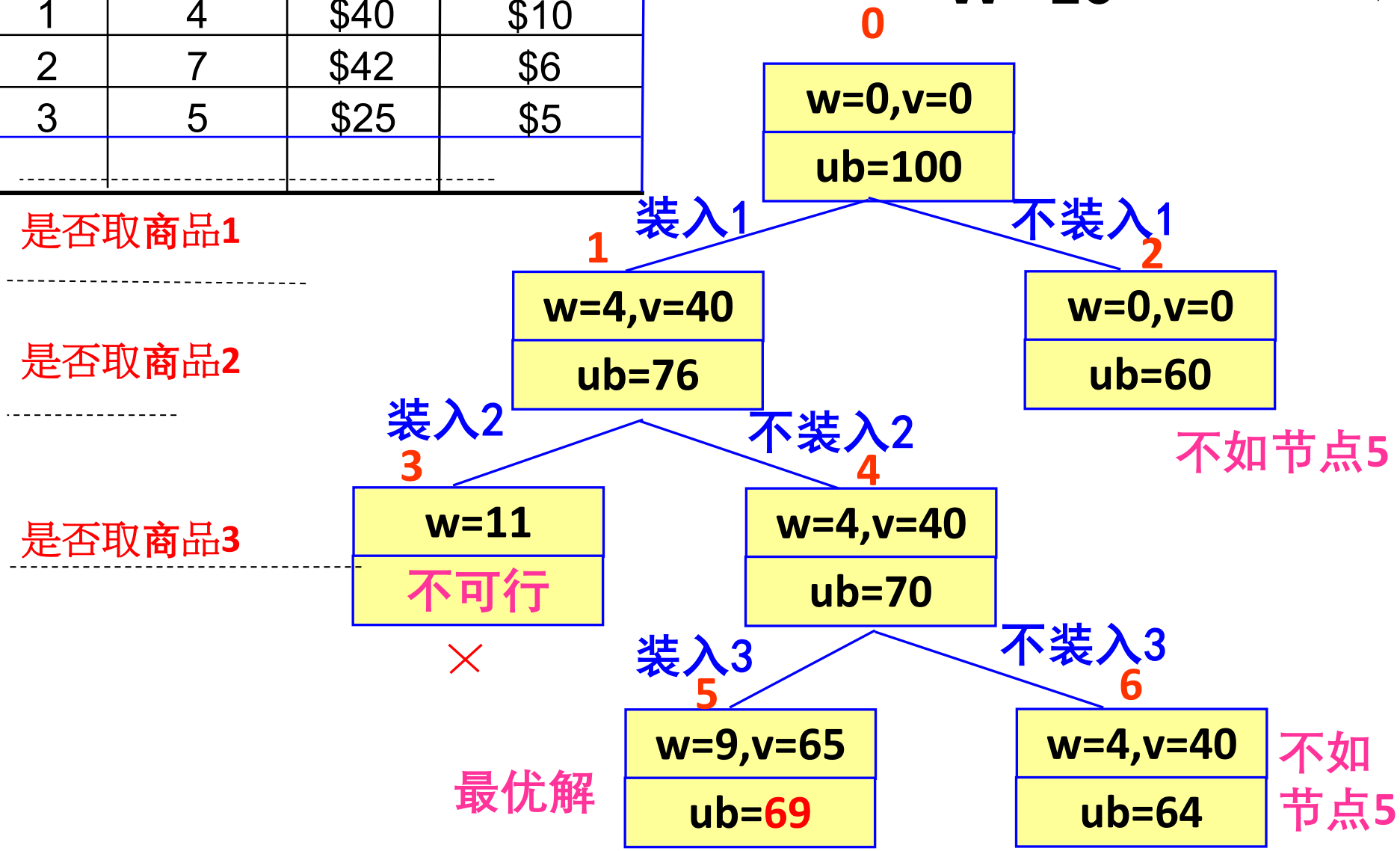


0-1背包问题

$ub = v + (W - w)(v_i / w_i)$ ；

W=10

Item	重量 (w_i)	价值(V)	价值/重量
1	4	$40	$10
2	7	$42	$6
3	5	$25	$5

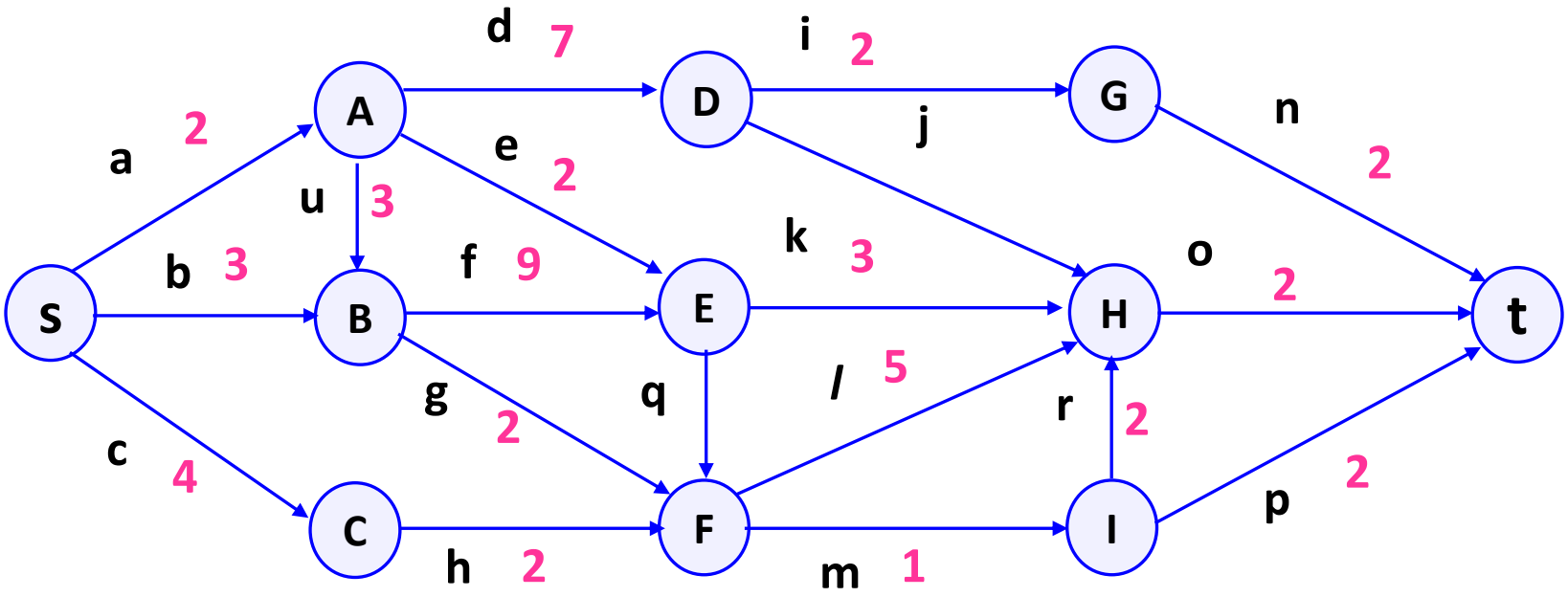



四川大学
SICHUAN UNIVERSITY
1896
DiCA
Data Intelligence & Computing Art Laboratory

6.3 单源点最短路径

问题描述：

- 例：在下图所给的有向图G中，每一边都有一个非负边权。要求图G的从源顶点s到目标顶点t之间的最短路径



口分析：

- 解空间树：m叉树
- 采用优先队列的分支限界法，优先策略为从根点到当前节点的路径长度。采用最小堆实现。

单源点最短路径

算法过程：

- 算法从图G的源顶点s和空优先队列开始；
- 结点s被扩展后，它的儿子结点被依次插入堆中。此后，算法从堆中取出具有最小当前路长的结点作为当前扩展结点，并依次检查与当前扩展结点有边的所有顶点；
- 如果从当前扩展结点i到顶点j有边可达，且从源出发，途经顶点i再到顶点j的所有路径长度小于当前最优路径长度，则将该顶点作为活结点插入到活结点优先队列中；
- 这个结点的扩展过程一直继续到活结点优先队列为空时为止。

单源点最短路径

算法过程：

- 算法从图G的源顶点s和空优先队列开始；
- 结点s被扩展后，它的儿子结点被依次插入堆中。此后，算法从堆中取出具有最小当前路长的结点作为当前扩展结点，并依次检查与当前扩展结点有边的所有顶点；
- 如果从当前扩展结点i到顶点j有边可达，且从源出发，途经顶点i再到顶点j的所有路径长度小于当前最优路径长度，则将该顶点作为活结点插入到活结点优先队列中；
- 这个结点的扩展过程一直继续到活结点优先队列为空时为止。

单源点最短路径

算法过程：

- 算法从图G的源顶点s和空优先队列开始；
- 结点s被扩展后，它的儿子结点被依次插入堆中。此后，算法从堆中取出具有最小当前路长的结点作为当前扩展结点，并依次检查与当前扩展结点有边的所有顶点；
- 如果从当前扩展结点i到顶点j有边可达，且从源出发，途经顶点i再到顶点j的所有路径长度小于当前最优路径长度，则将该顶点作为活结点插入到活结点优先队列中；
- 这个结点的扩展过程一直继续到活结点优先队列为空时为止。

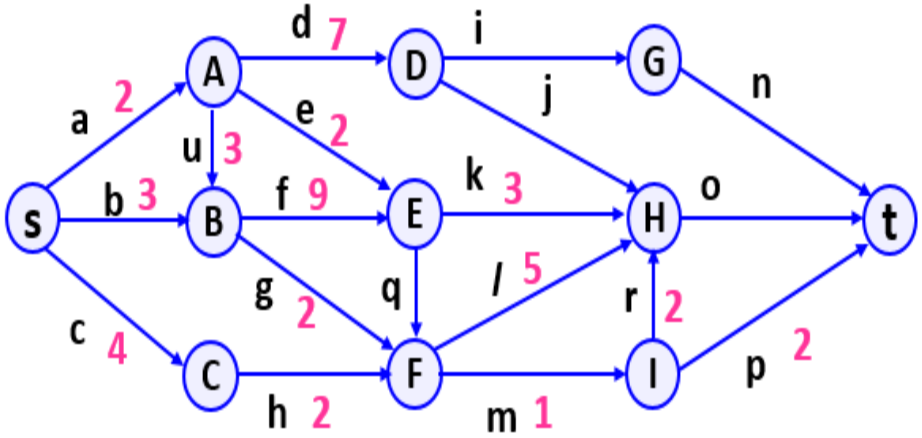
单源点最短路径

算法过程：

- 算法从图G的源顶点s和空优先队列开始；
- 结点s被扩展后，它的儿子结点被依次插入堆中。此后，算法从堆中取出具有最小当前路长的结点作为当前扩展结点，并依次检查与当前扩展结点有边的所有顶点；
- 如果从当前扩展结点i到顶点j有边可达，且从源出发，途经顶点i再到顶点j的所有路径长度小于当前最优路径长度，则将该顶点作为活结点插入到活结点优先队列中；
- 这个结点的扩展过程一直继续到活结点优先队列为空时为止。

单源点最短路径

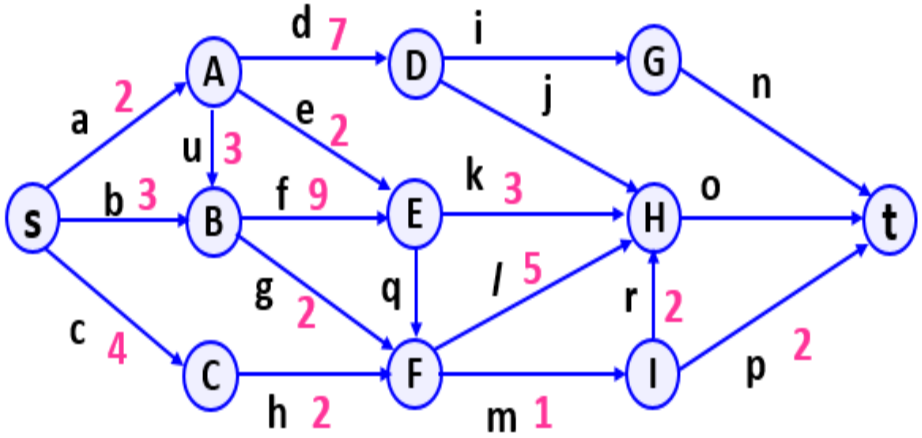
- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

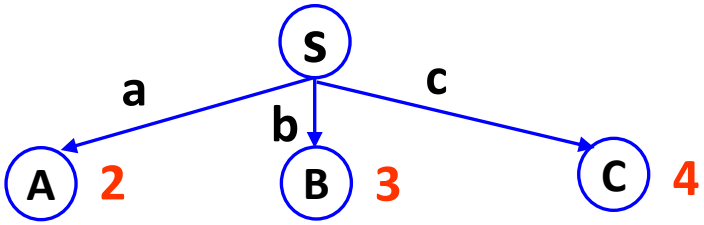


s

单源点最短路径

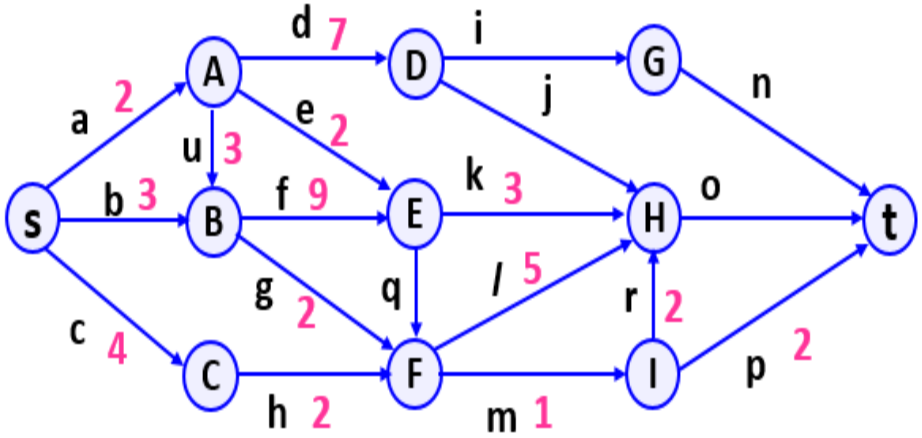
- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

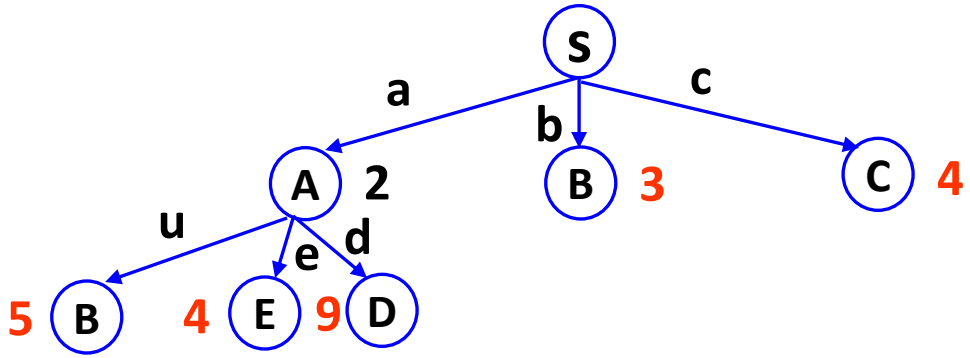




单源点最短路径

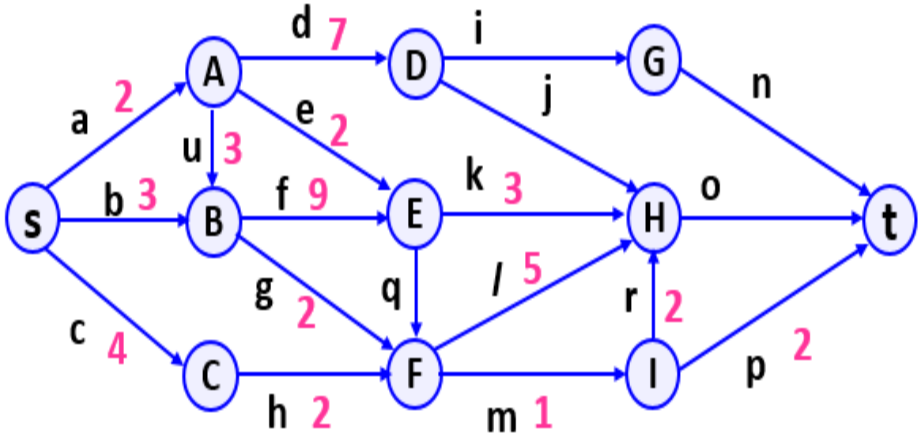
- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

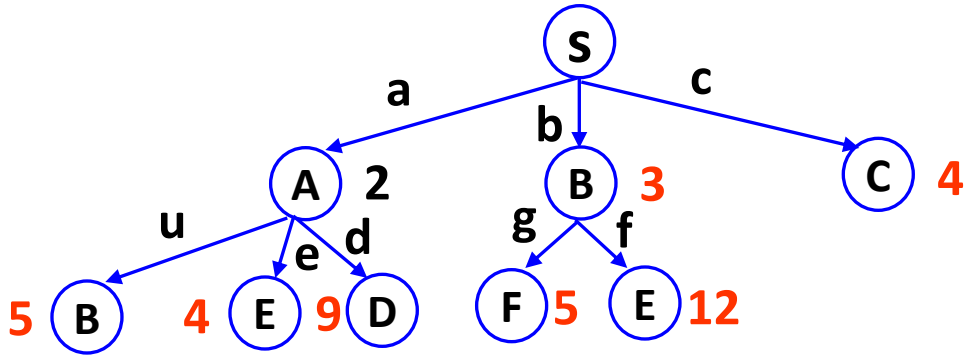




单源点最短路径

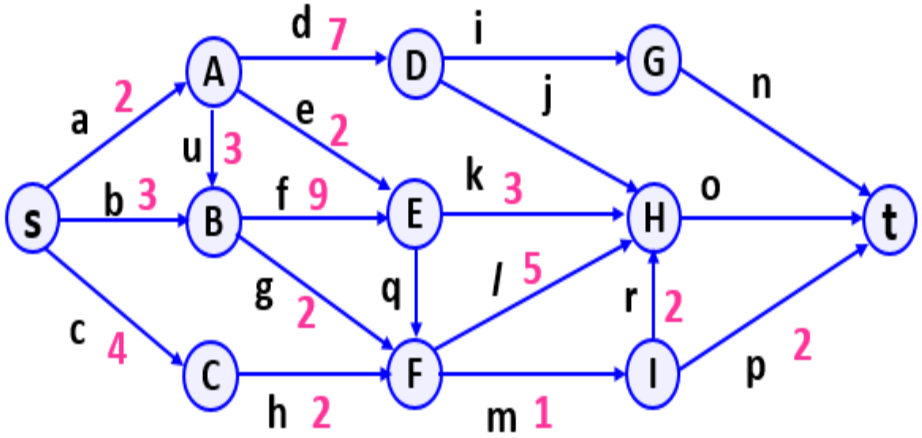
- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

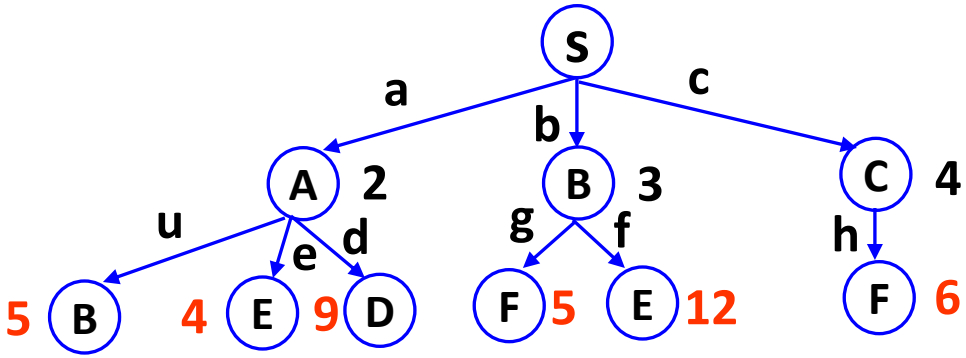




单源点最短路径

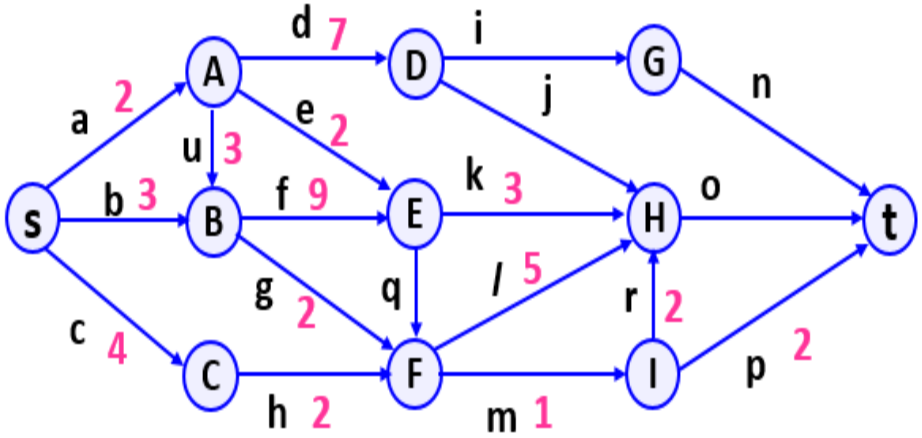
- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

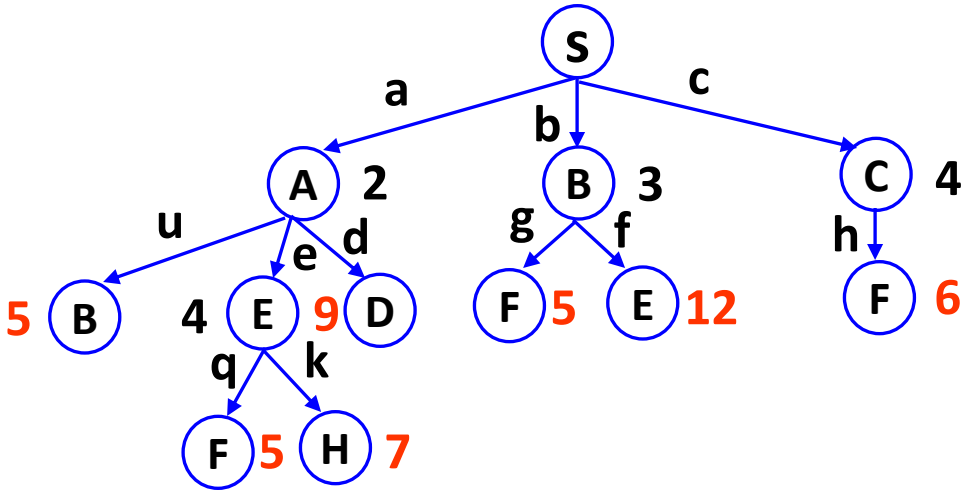




单源点最短路径

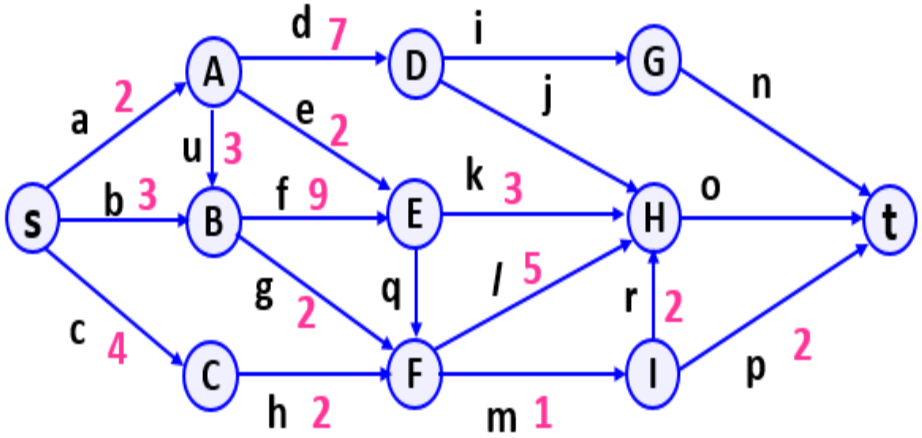
- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

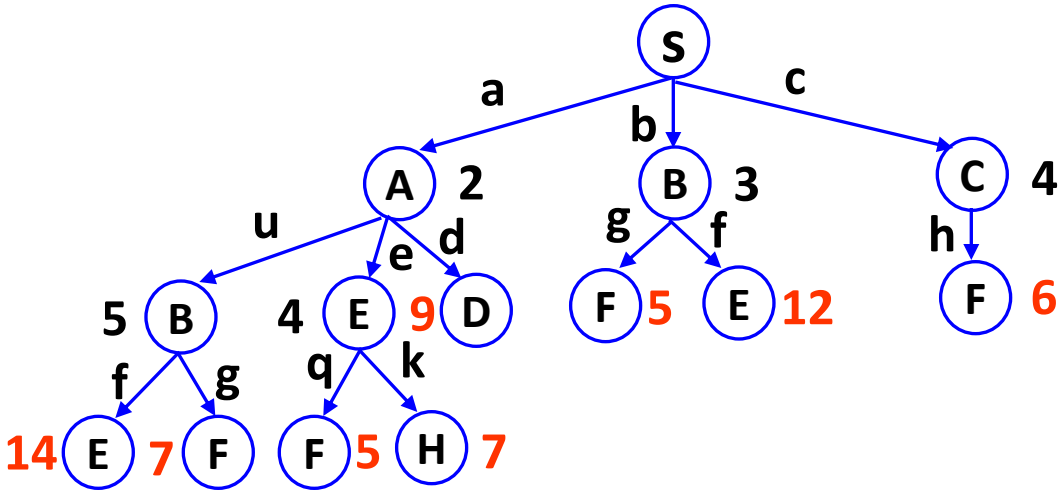




单源点最短路径

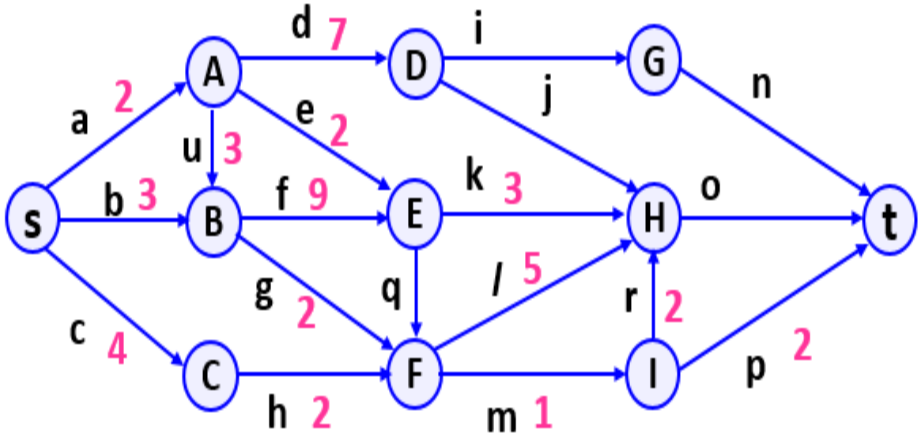
- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

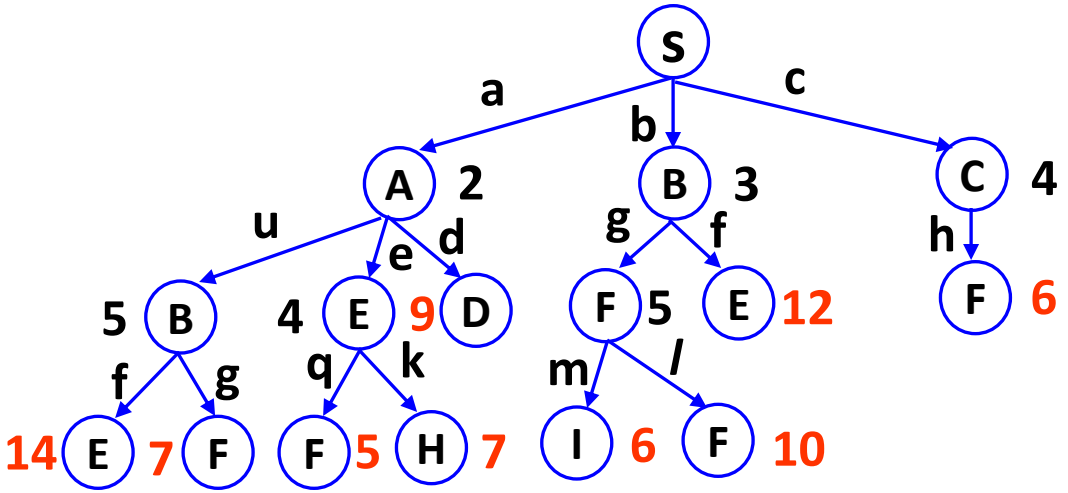




单源点最短路径

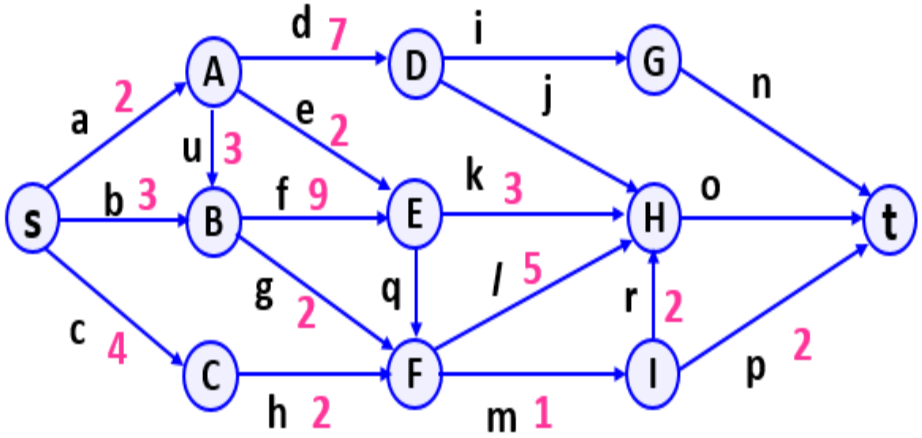
- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

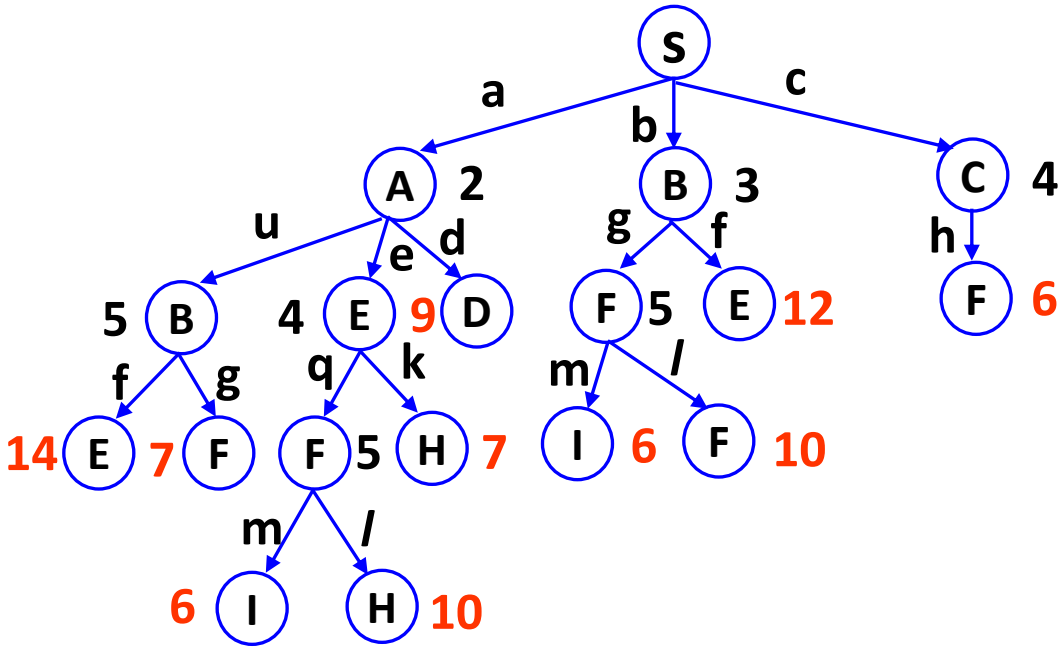




单源点最短路径

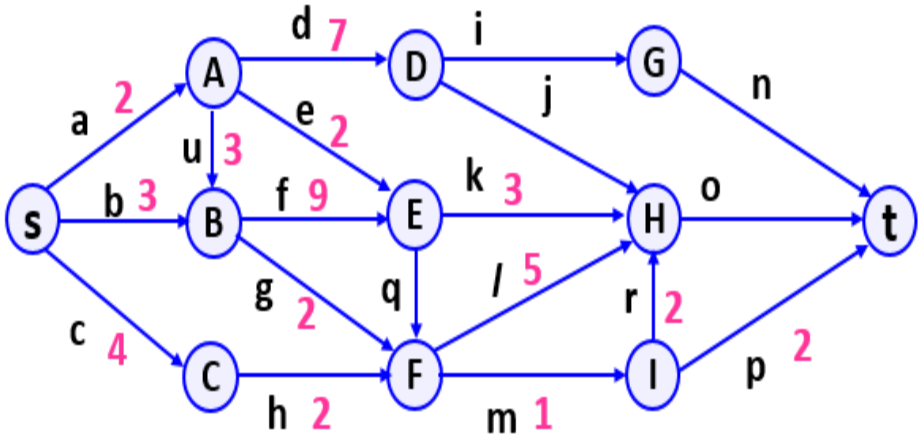
- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

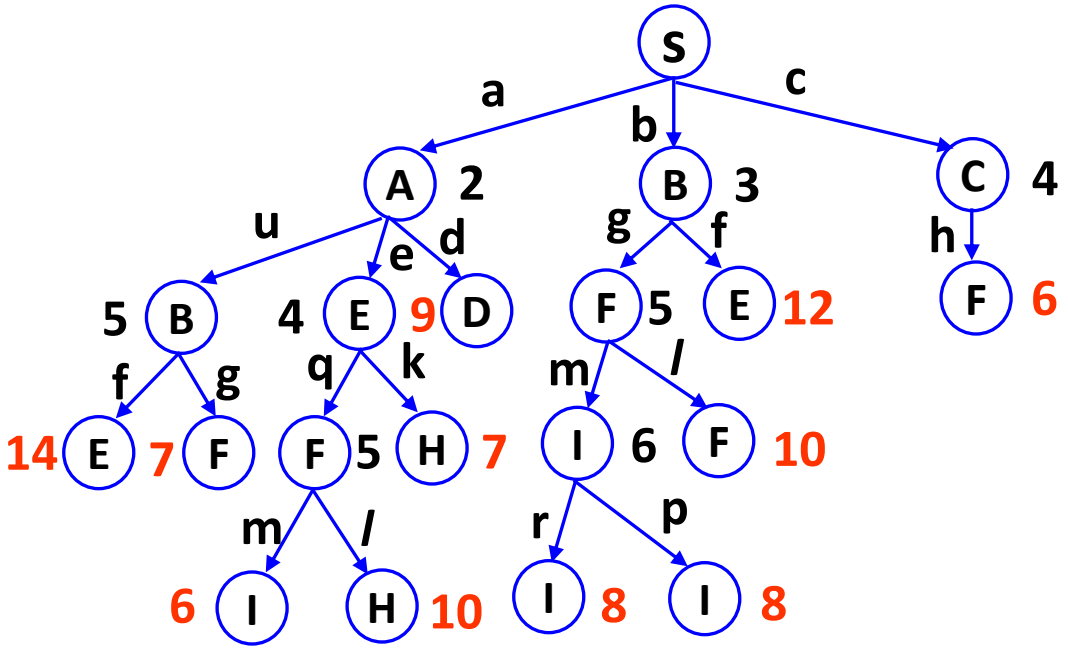




单源点最短路径

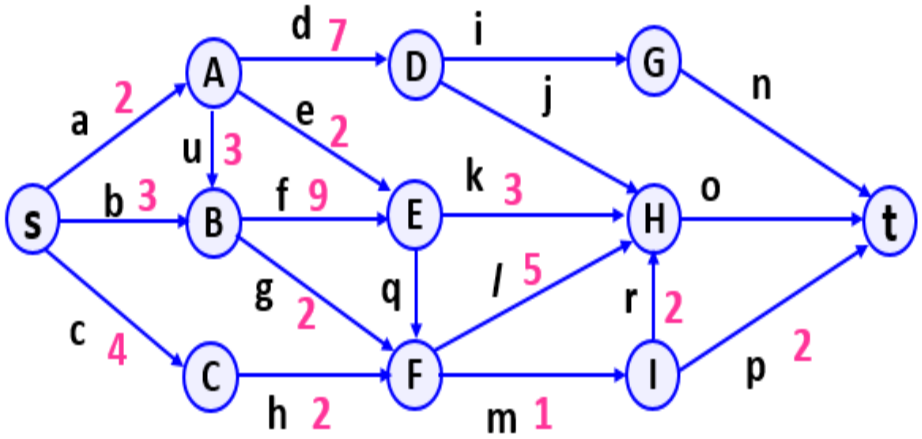
- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

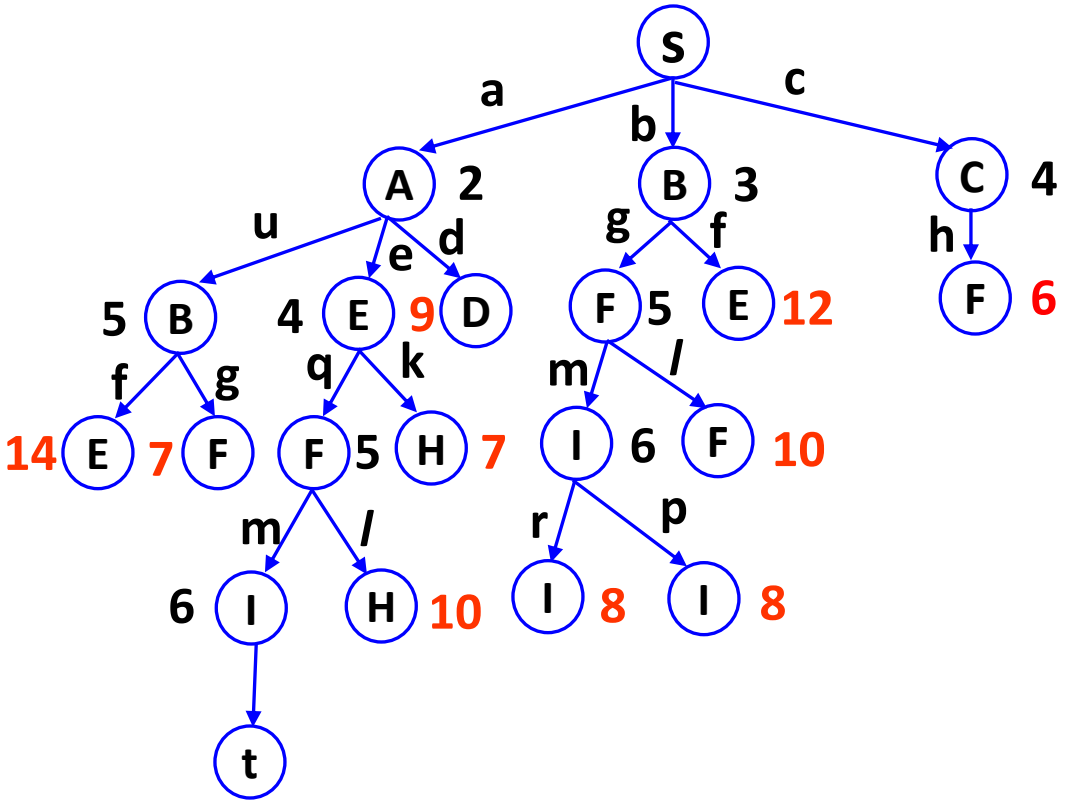




单源点最短路径

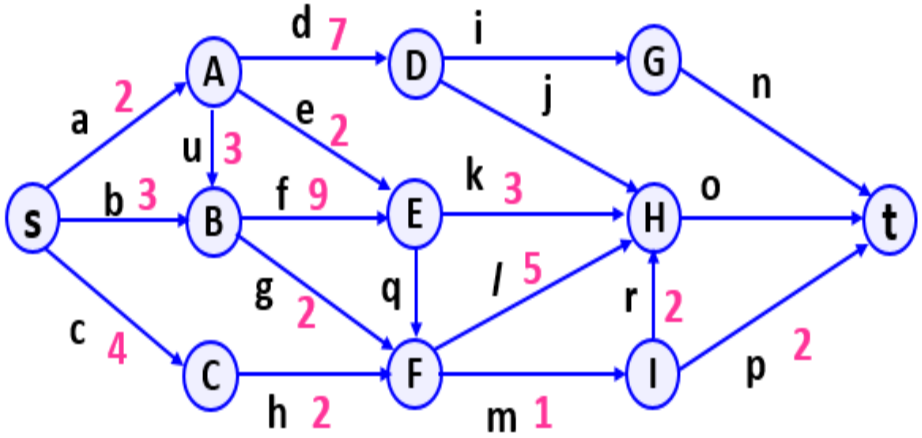
- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

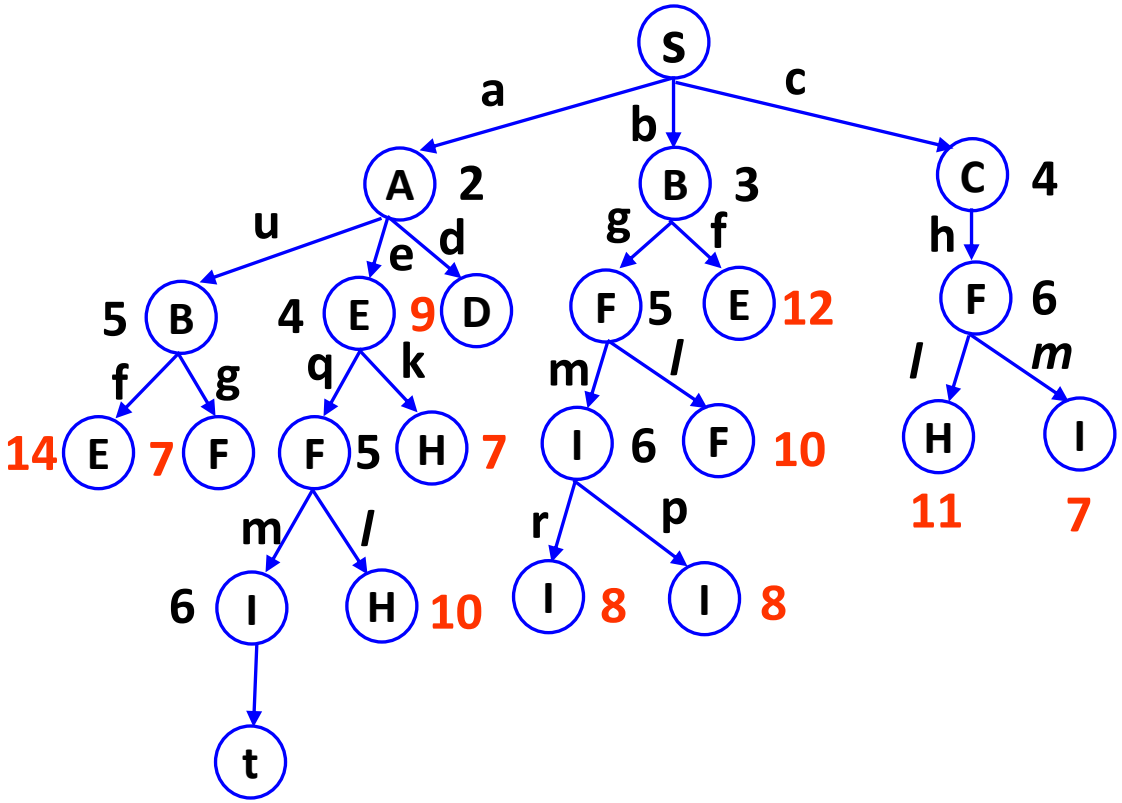




单源点最短路径

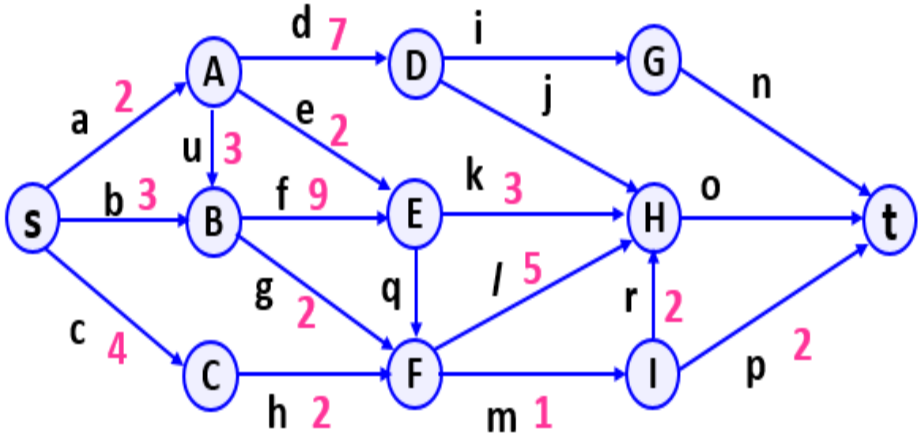
- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

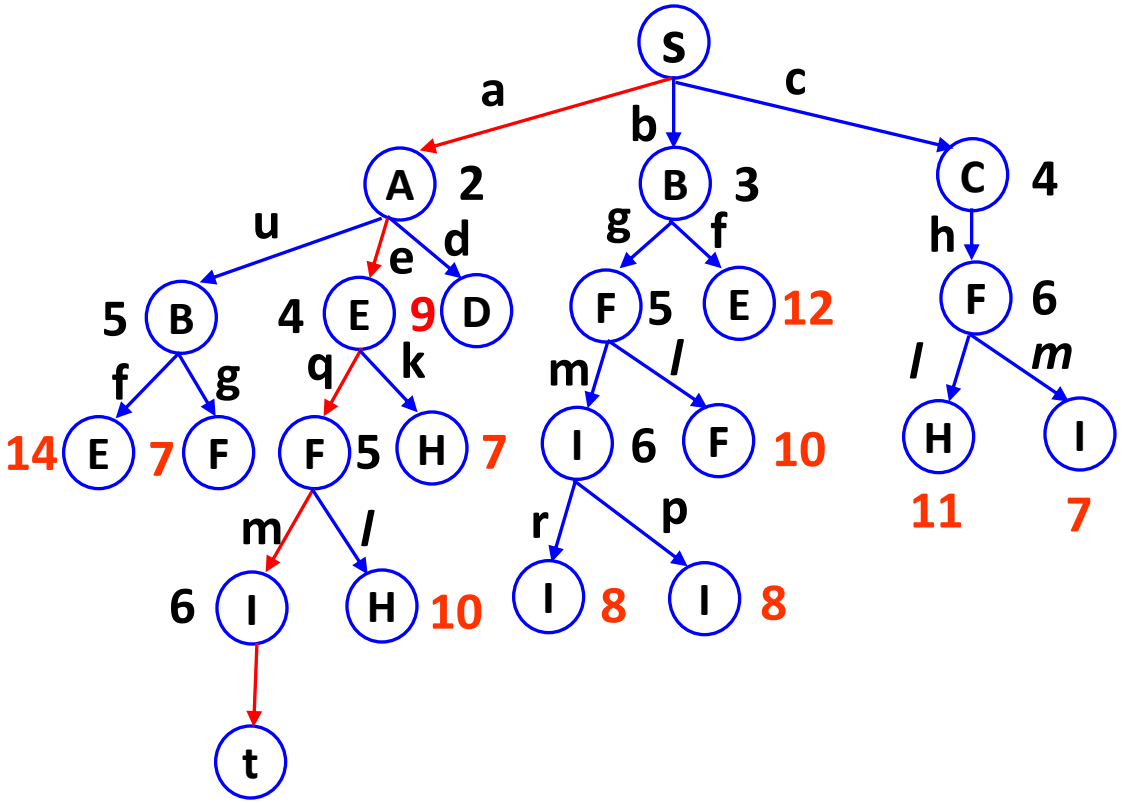




单源点最短路径

- **算法过程**：解空间树是一个m叉树，其中数字代表源点到当前点的路径长度，红色代表当前节点为带扩展的活结点

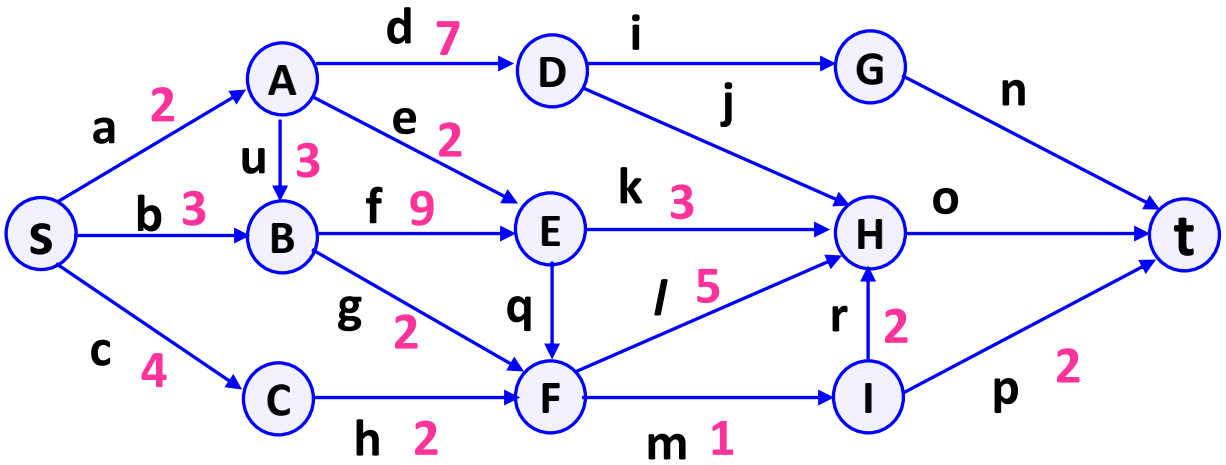




单源点最短路径

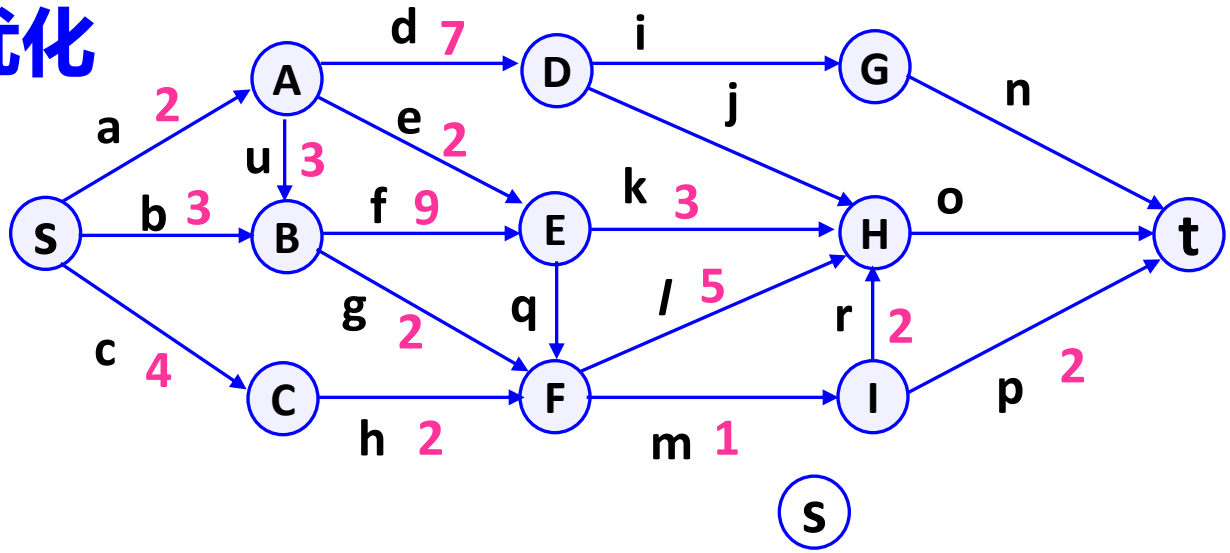
剪枝优化：

- 易知，当前点对应的路径长度是以该结点为根的子树中所有结点所对应的路长的一个下界，
- 一旦发现一个节点的下界不小于当前找到的最短路径，则剪去以该节点为根的子树
- 从源顶点s出发，2条不同路径到达图G的同一顶点。因此可以将路长较长的结点为根的子树剪去。
- 以如下图G为例，算法过程见下页



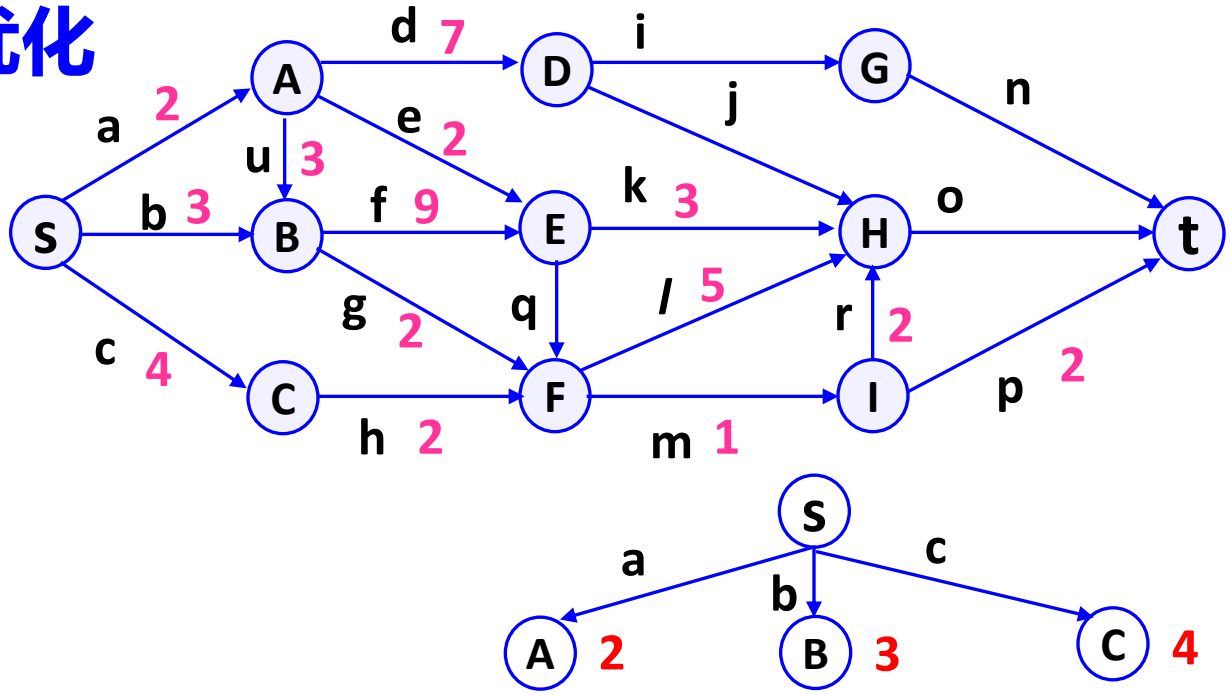
单源点最短路径

剪枝优化

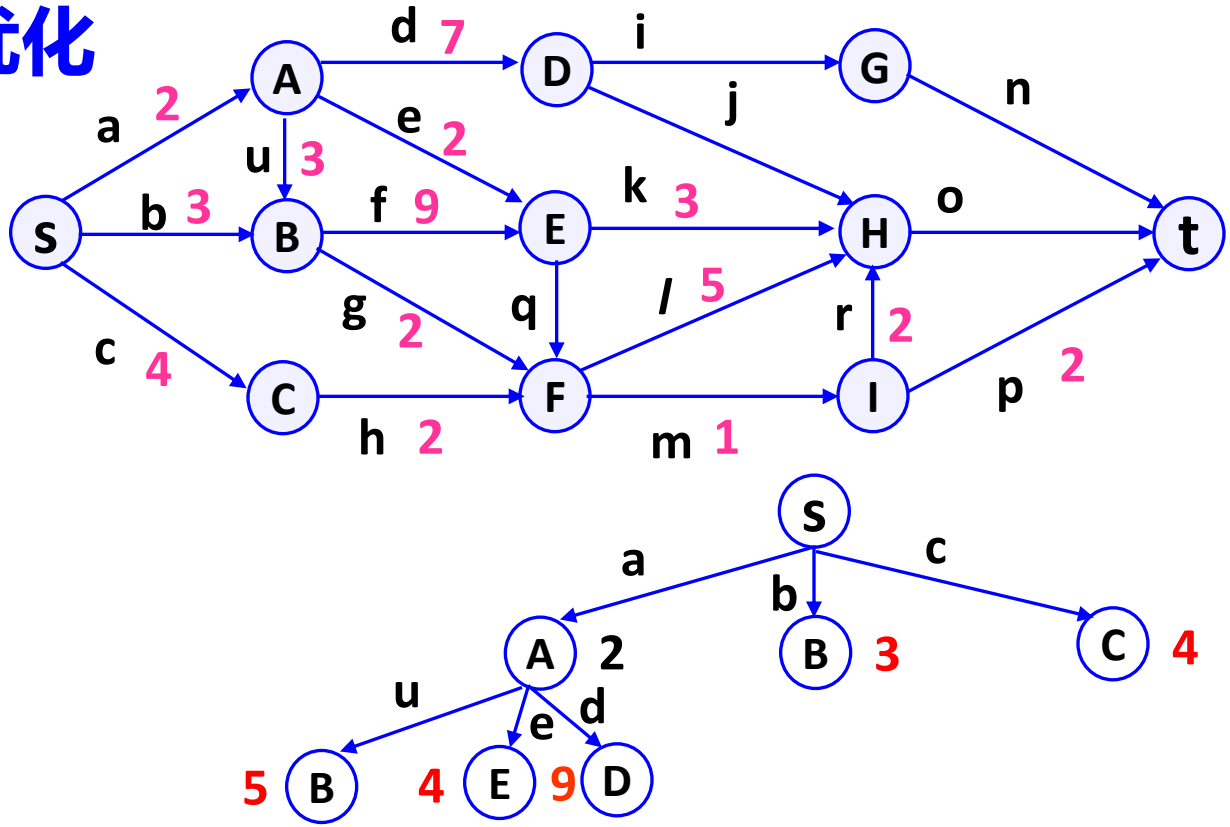


单源点最短路径

口 剪枝优化

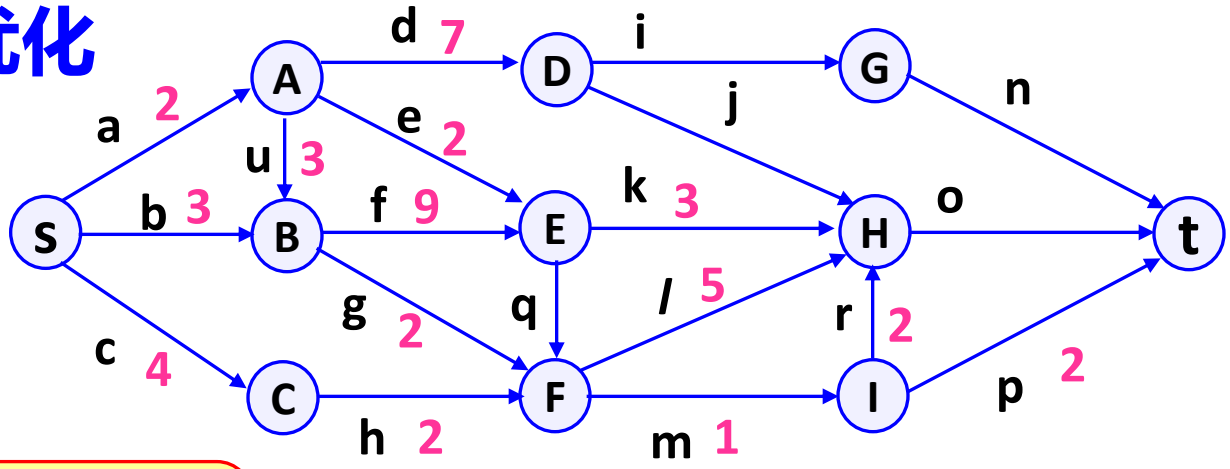


剪枝优化

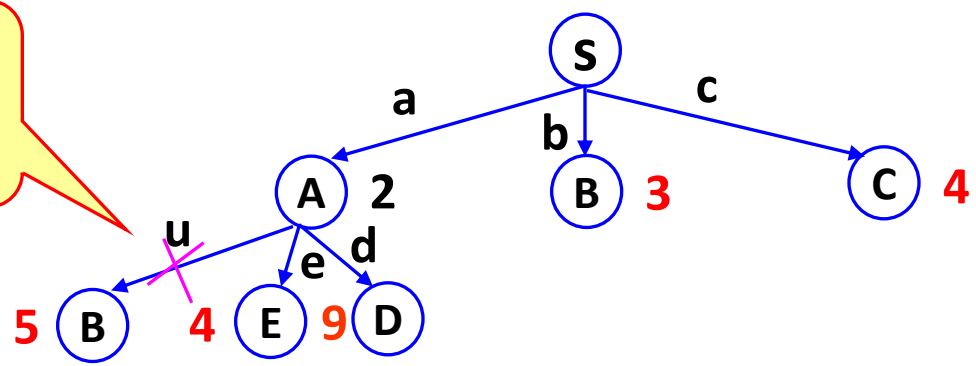

S
A
B
C
D
E
F
G
H
I
t
a 2
b 3
c 4
d 7
e 2
u 3
f 9
g 2
h 2
i
j
k 3
l 5
m 1
n
o
p 2
q
r 2
S
a
b
c
A 2
B 3
C 4
u
e
d
5 B
4 E
9 D

单源点最短路径

剪枝优化

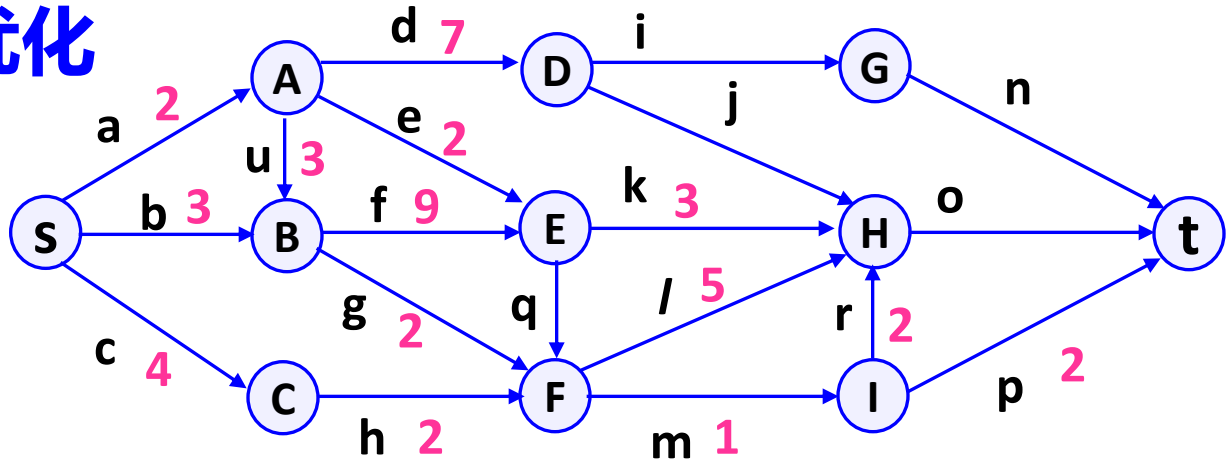


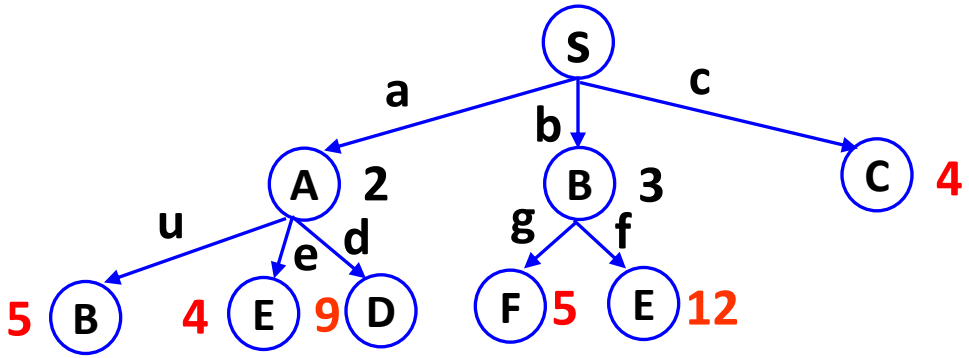
剪去路长较长的子树



单源点最短路径

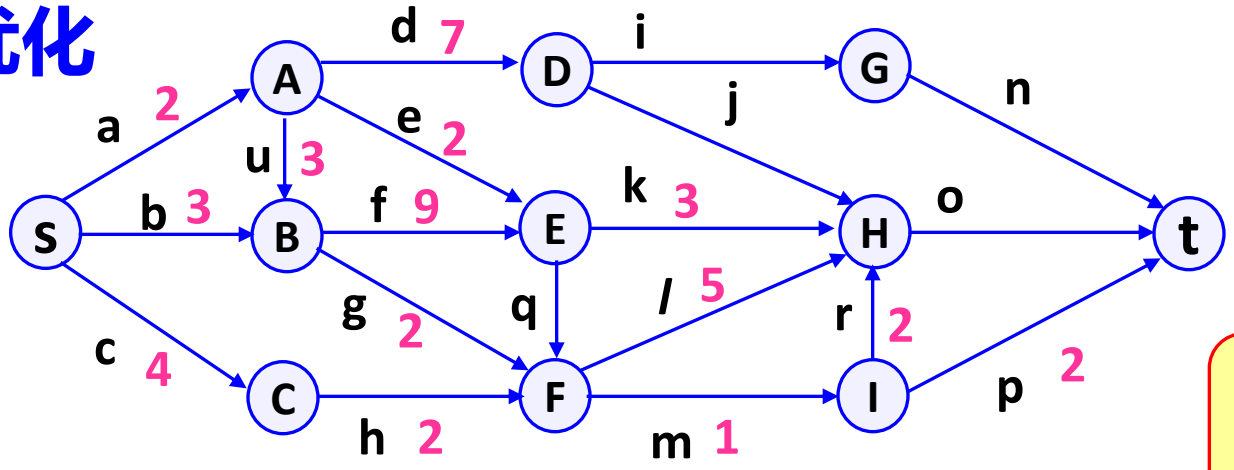
剪枝优化

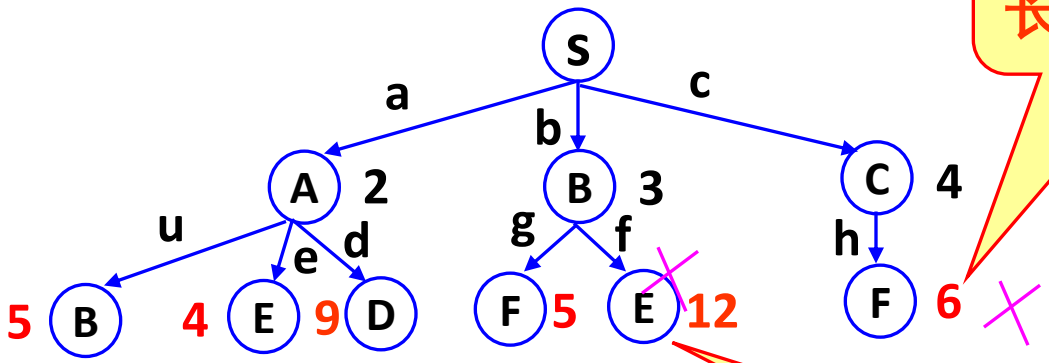




单源点最短路径

剪枝优化



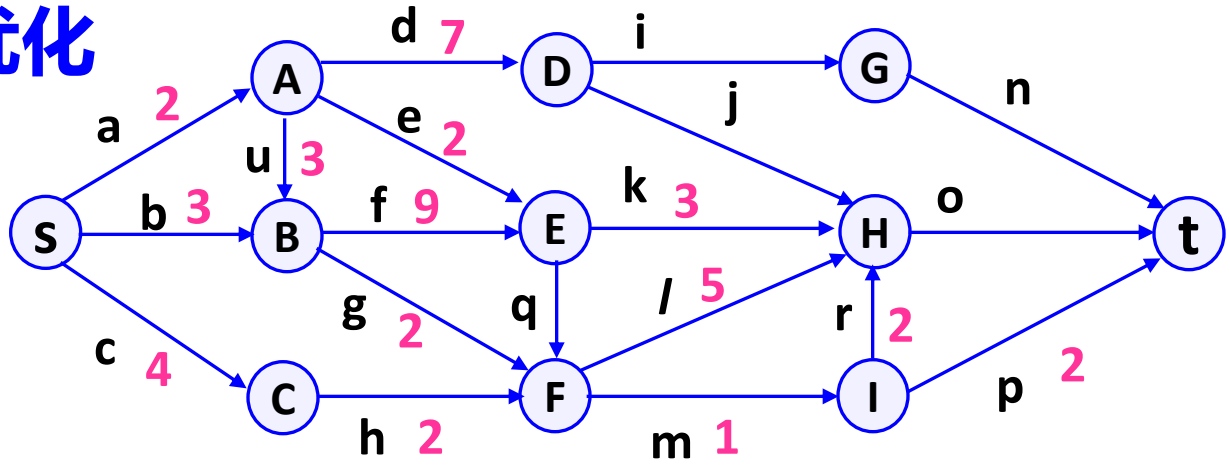


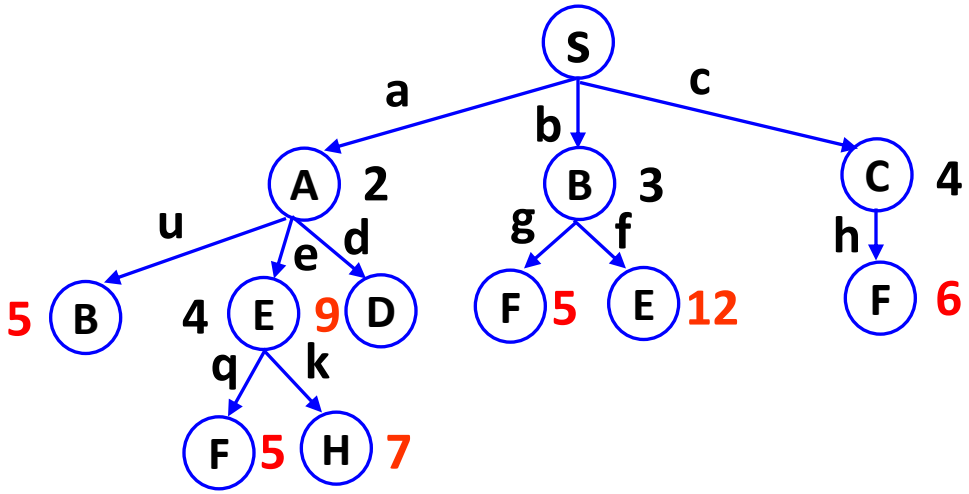
剪去路长较长的子树

剪去路长较长的子树

单源点最短路径

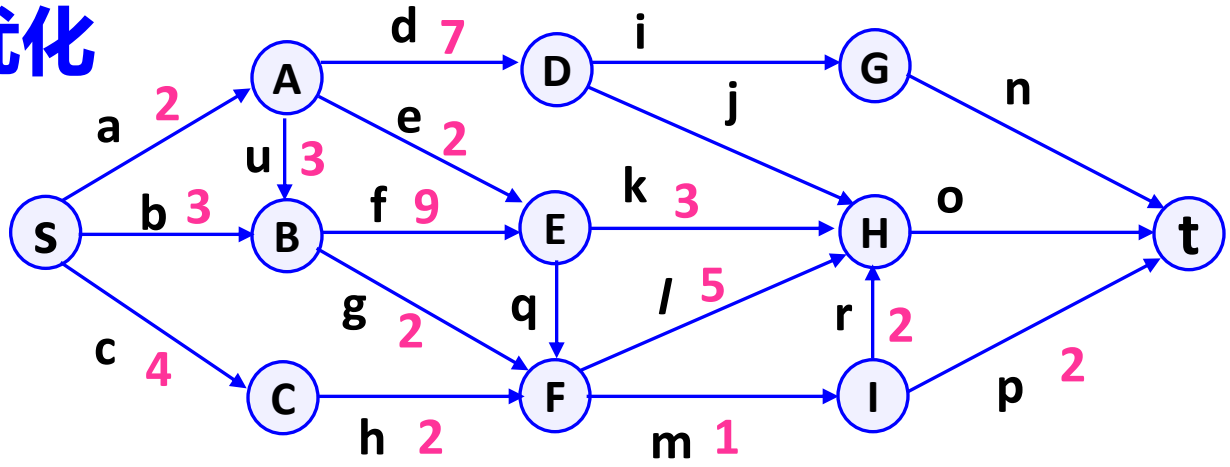
剪枝优化

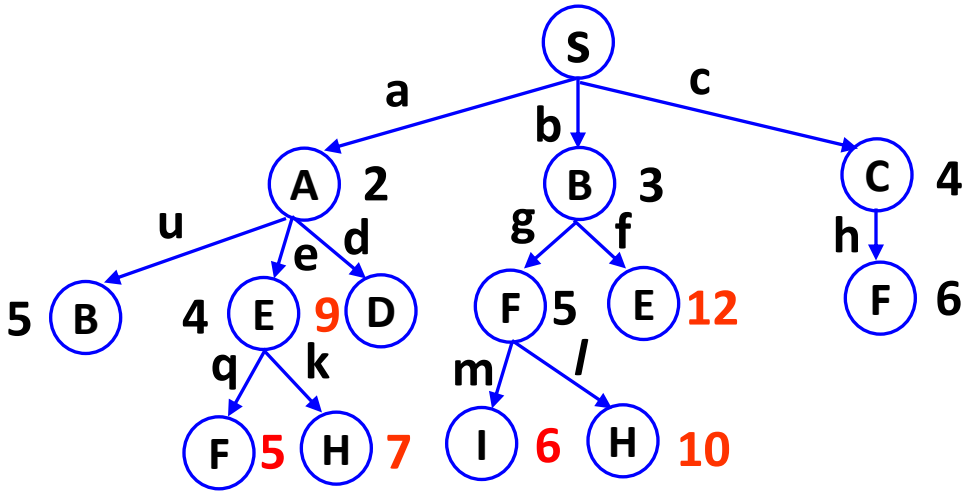




单源点最短路径

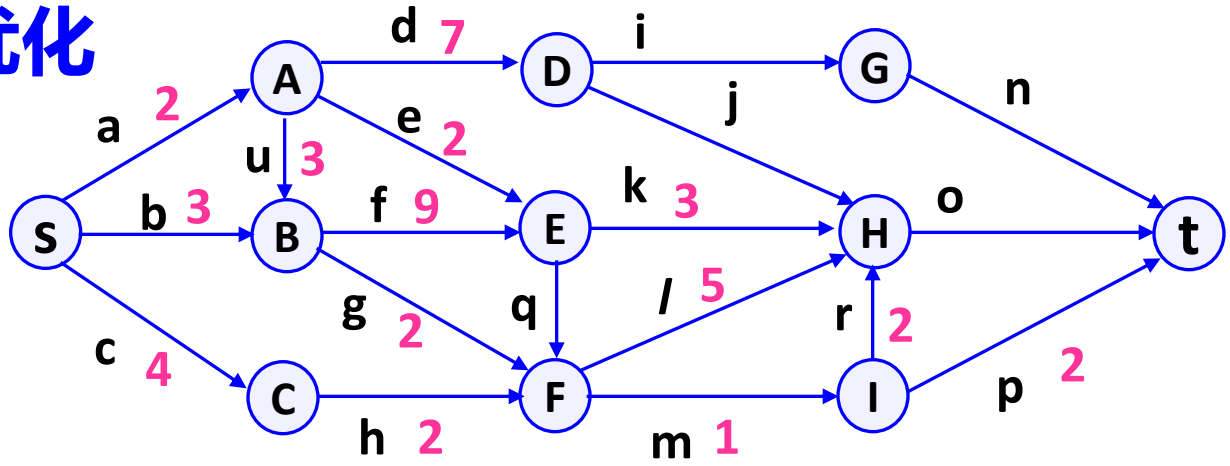
剪枝优化

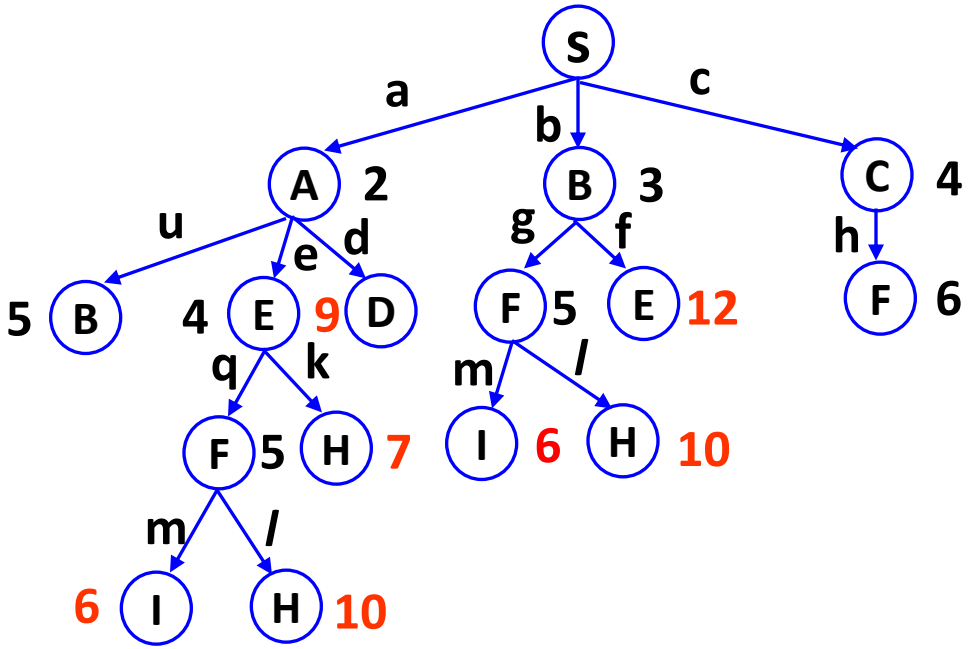




单源点最短路径

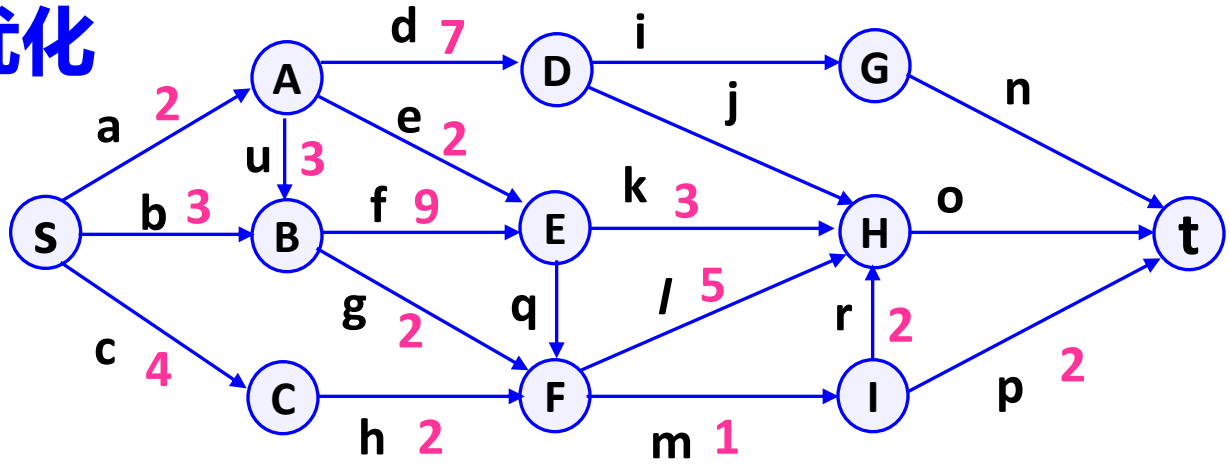
剪枝优化

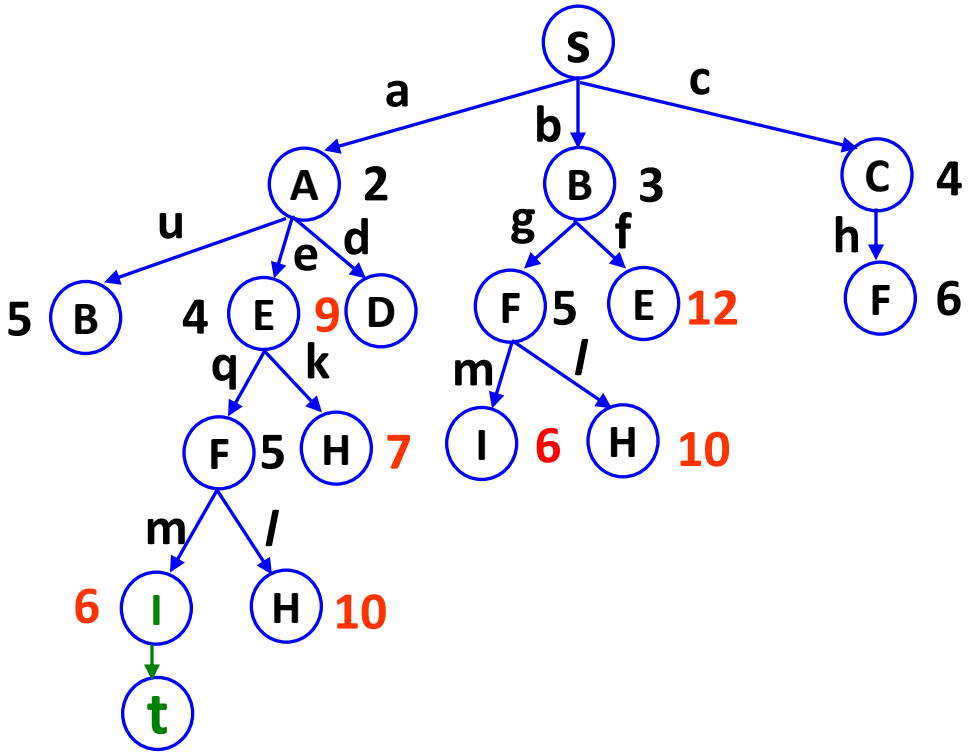




单源点最短路径

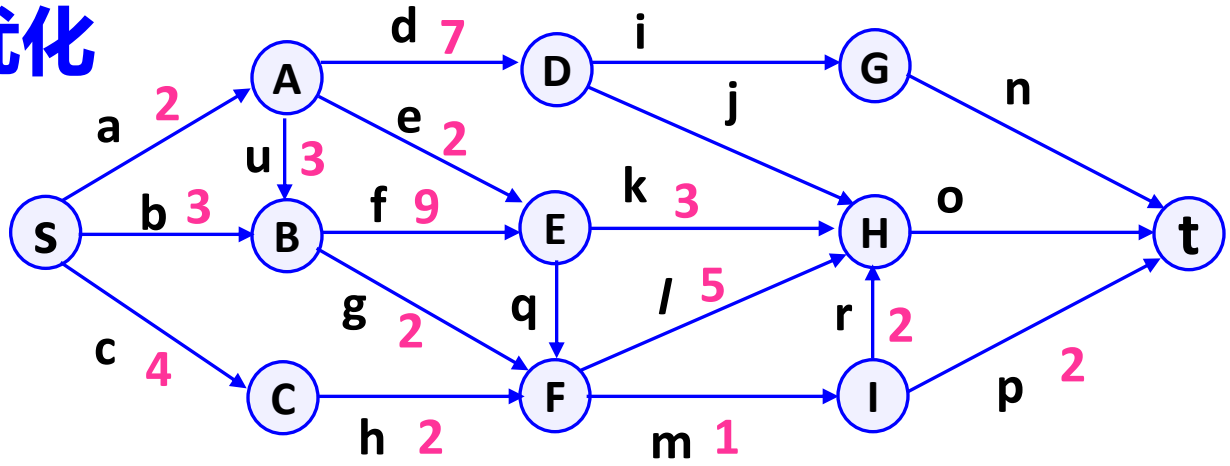
剪枝优化

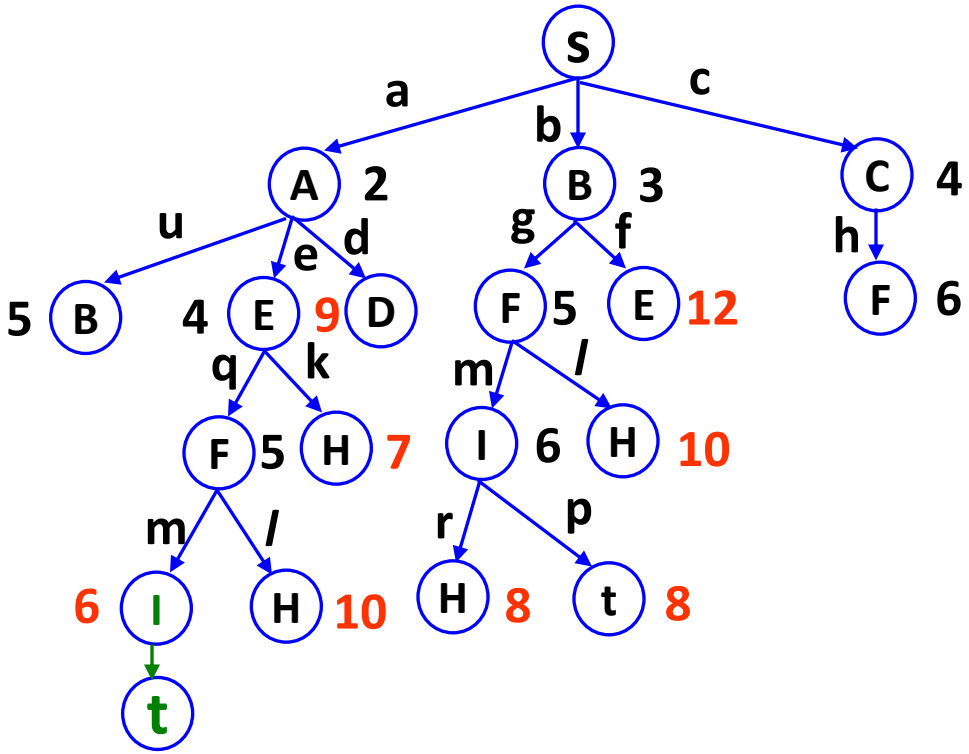




单源点最短路径

剪枝优化





6.4 最大团问题

最大团问题

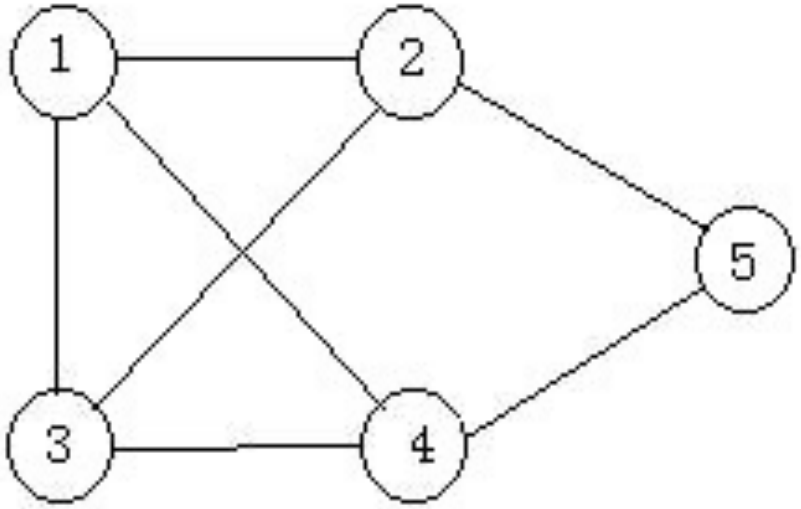
问题描述

- 给定一个无向图 G=(V, E), 如果 U 是 V 的子集，且对任意 u,v ∈U, (u,v)一定是边，且有(u,v)∈E ，则称 U 是 G 的一个完全子图（U是完全子图的必要条件：U为完全图）；
- G 的完全子图 U 是 G 的一个团当且仅当U 不包含在G 的更大的完全子图中;
- G 的最大团是指G 中所含顶点数最多的团——最大完全子图。

最大团问题

口例：下图所示无向图 G 中：

- 子集{1，2} 是 G 的大小为2的完全子图；
- {1，2 ，5} 不是完全子图（因为它当中的1 ，5 之间没有边）
- {1，2，3} 是 G 的大小为3的完全子图，它包含了{1 ，2} ，所以，{1 ，2} 这个完全子图不是团，{1 ，2 ，3} 和{1 ，3 ，4} 是G 的最大团。

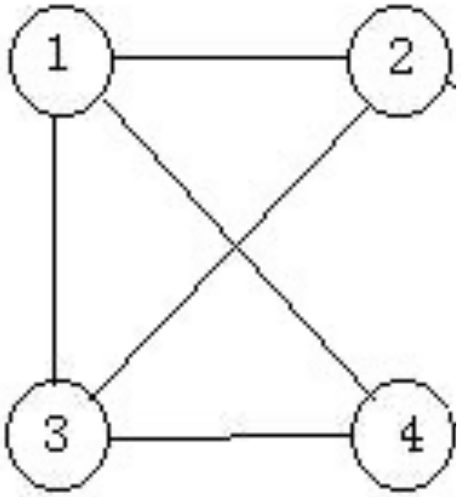


无向图G，顶点数n=5

口 分析

- 将图G的n个顶点依次编号1,2,…,n, 并将图用邻接矩阵表示;
- 定义：int g[n][n]; 用二维数组g存储图的邻接矩阵, 图G的邻接矩阵可表示为数组g：

```
g={ 0, 1, 1, 1,
    1, 0, 1, 0,
    1, 1, 0, 1,
    1, 0, 1, 0, };
```



分析

- 我们用 x_i=0 表示不选第i号顶点，x_i=1表示选第i号顶点，其中，i=1,2,…, n 。则此问题的解可表示为n元组（$x_1, x_2, \ldots, x_n$ ），$x_i \in \{0,1\}$, $1 \le i \le n$；
- 因此，可将最大团问题的视为图G顶点集V的子集选取问题，解空间树为一颗子集树;
- 在不受约束的情况下，所有可能的解共有 2^n 个（为解空间树的叶子数目），解空间可构造成一棵深度为 n+1 的满二叉树。

口分析

- 第k层即代表是第k个节点；
- 与回溯法的区别是，既应用约束函数约束可行性，也利用上界函数剪枝;
- 约束函数：当前备选的节点x_k加入后，原图仍够成一个完全子图——任意两个点间存在边，即x_k与$x_1,\ldots,x_{k-1}$均存在边;
- 上界函数：定义当前节点的最大团节点数上界为优先队列的优先级，这样可以始终最先扩展具有最大上界的节点。

实现

定义活节点类

```
class CliqueNode  {
      friend class Clique ;
      public :  operator int  ()  const  { return un ; }
      private :  int cn,          //当前团的顶点数
                    un,           //当前团最大顶点数的上界
                  level ;       //结点在子集空间树中所处层
     bbnode* ptr ;        //指向活结点在子集树中结点指针
}    ;
```

实现

- 定义最大图类

```
class Clique {
    friend void main (void) ;
    public: int BBMaxClique (int [] ) ;
    private: void AddLiveNode (MaxHeap < CliqueNode >
&H,
    int cn, int un, int level, bbnode E[ ], bool ch) ;
    int **a,          //图G的邻接矩阵
        n ;           //图G的顶点数
} ;
```

最大团问题

算法思想

- 用变量un表示与该结点相应的团的顶点数，level表示结点在子集空间树中所处的层次，用 un+n-level+1 作为顶点数上界的值;
- 子集树的根结点是初始扩展结点，对于这个特殊的扩展结点，其un的值为0;
- 算法在扩展内部结点时，首先考察其左儿子结点，在左儿子结点处，将顶点i加入到当前团中，并检查该顶点与当前团中其它顶点之间是否有边相连;当顶点i与当前团中所有顶点之间都有边相连，则相应的左儿子结点是可行结点，将它加入到子集树中并插入活结点优先队列，否则就不是可行结点;
- 接着继续考察当前扩展结点的右儿子结点。当un>bestn时，右子树中可能含有最优解，此时将右儿子结点加入到子集树中并插入到活结点优先队列中。

算法思想

- 用变量un表示与该结点相应的团的顶点数，level表示结点在子集空间树中所处的层次，用 un+n-level+1 作为顶点数上界的值;
- 子集树的根结点是初始扩展结点，对于这个特殊的扩展结点，其un的值为0;
- 算法在扩展内部结点时，首先考察其左儿子结点，在左儿子结点处，将顶点i加入到当前团中，并检查该顶点与当前团中其它顶点之间是否有边相连;当顶点i与当前团中所有顶点之间都有边相连，则相应的左儿子结点是可行结点，将它加入到子集树中并插入活结点优先队列，否则就不是可行结点;
- 接着继续考察当前扩展结点的右儿子结点。当un>bestn时，右子树中可能含有最优解，此时将右儿子结点加入到子集树中并插入到活结点优先队列中。

最大团问题

算法思想

- 用变量un表示与该结点相应的团的顶点数，level表示结点在子集空间树中所处的层次，用 un+n-level+1 作为顶点数上界的值；
- 子集树的根结点是初始扩展结点，对于这个特殊的扩展结点，其un的值为0；
- 算法在扩展内部结点时，首先考察其左儿子结点，在左儿子结点处，将顶点i加入到当前团中，并检查该顶点与当前团中其它顶点之间是否有边相连;当顶点i与当前团中所有顶点之间都有边相连，则相应的左儿子结点是可行结点，将它加入到子集树中并插入活结点优先队列，否则就不是可行结点；
- 接着继续考察当前扩展结点的右儿子结点。当un>bestn时，右子树中可能含有最优解，此时将右儿子结点加入到子集树中并插入到活结点优先队列中。

最大团问题

口 算法思想

- 用变量un表示与该结点相应的团的顶点数，level表示结点在子集空间树中所处的层次，用 un+n-level+1 作为顶点数上界的值;
- 子集树的根结点是初始扩展结点，对于这个特殊的扩展结点，其un的值为0;
- 算法在扩展内部结点时，首先考察其左儿子结点，在左儿子结点处，将顶点i加入到当前团中，并检查该顶点与当前团中其它顶点之间是否有边相连;当顶点i与当前团中所有顶点之间都有边相连，则相应的左儿子结点是可行结点，将它加入到子集树中并插入活结点优先队列，否则就不是可行结点;
- 接着继续考察当前扩展结点的右儿子结点。当un>bestn时，右子树中可能含有最优解，此时将右儿子结点加入到子集树中并插入到活结点优先队列中。

实现

```
//解最大团问题的优先队列式分支限界法
int Clique::BBMaxClique(int bextx[])
  {
        //定义最大堆的容量为1000
        MaxHeap < CliqueNode >  H （1000）  ;
        bbnode *E = 0 ;     //初始化
        int i = 1,                  // 子空间树的level
            cn = 0,                 // 当前团的顶点数
            bestn = 0 ;

        //搜索子集空间树
```

```
while (i != n+1) {                    //非叶结点
    bool OK = true;
    bbnode *B = E;                    //是否满足约束条件

    for (int j = i-1; j > 0; B = B->parent, j--)
      if (B->Lchild && a[i][j]==0) { OK = false; break; }

    if (OK) {                         //左儿子结点为可行结点
        if (cn+1 > bestn) bestn = cn+1;
        AddLiveNode (H, cn+1, cn+n-i+1, i+1, E, true); }

    if (cn+n-i >= bestn)              //右子树可能含最优解
        AddLiveNode (H, cn, cn+n-i, i+1, E, false);

    CliqueNode N;
    H.DeleteMax (N);                  //堆非空，取下一扩展结点

    E = N.ptr;
    cn = N.cn;
    i = N.level;
}
```


SICHUAN UNIVERSITY
1896


DiCA
Data Intelligence & Computing Art Laboratory

End