算法设计与分析

主讲教师：邵蓥侠

联系方式：shaoyx@bupt.edu.cn

个人主页：https://shaoyx.github.io/

第6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

（1）队列式(FIFO)分支限界法

（2）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。

1

与回溯法的区别?

理解
分支限界法的算法策略

分支限界法

分支限界法与回溯法的区别：

（1）**求解目标：** 回溯法的求解目标是找出解空间树中满足约束条件的所有解，而分支限界法的求解目标则是找出满足约束条件的一个解，或是在满足约束条件的解中找出在某种意义下的最优解。

（2）**搜索方式的不同：** 回溯法以深度优先的方式搜索解空间树，而分支限界法则以广度优先或以最小耗费优先的方式搜索解空间树。

基本思想

分支限界法常以广度优先或以最小耗费（最大效益）优先的方式搜索问题的解空间树。

在分支限界法中，每一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儿子结点。在这些儿子结点中，导致不可行解或导致非最优解的儿子结点被舍弃，其余儿子结点被加入活结点表中。

此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。这个过程一直持续到找到所需的解或活结点表为空时为止。

常见的两种分支限界法

（1）队列式(FIFO)分支限界法

按照队列先进先出（FIFO）原则选取下一个节点为扩展节点。

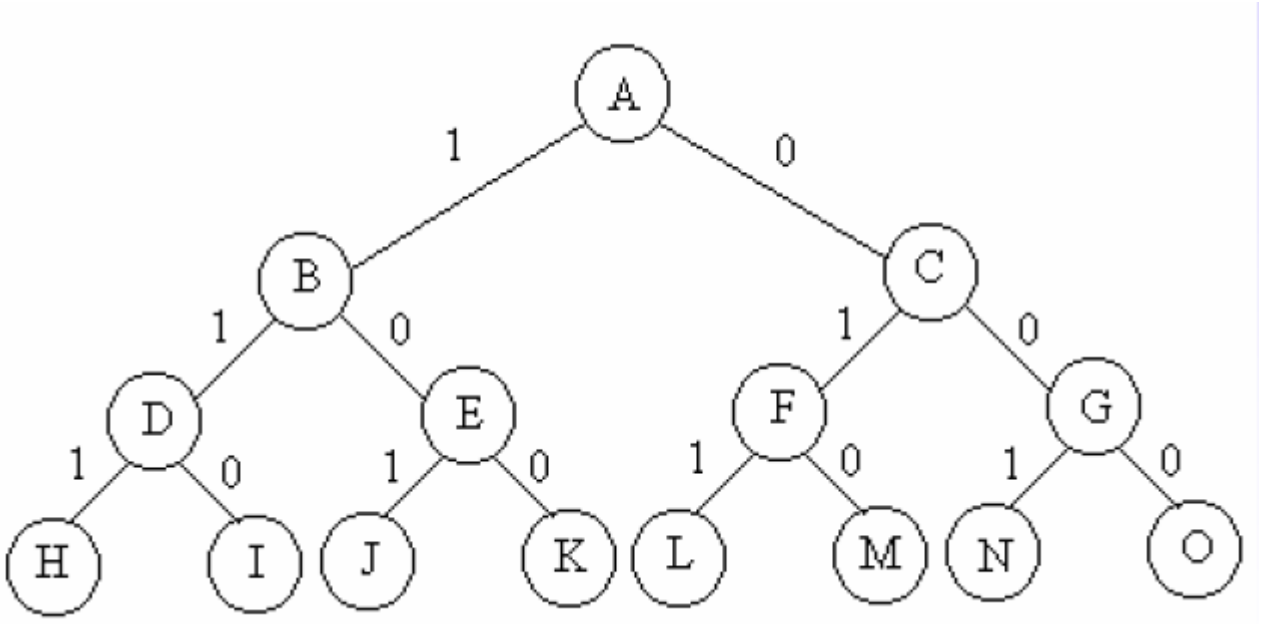
（2）优先队列式分支限界法

按照优先队列中规定的优先级选取优先级最高的节点成为当前扩展节点。

问题的解空间

与回溯法相同，一般用**解空间树**（Solution Space Trees,也称状态空间树）的方式组织。

例如，对于n=3时的0-1背包问题，可用完全二叉树表示其解空间，如下图所示。



对解空间树的动态搜索过程

分支限界法首先确定一个合理的限界函数，并根据限界函数确定目标函数的界[*down*, *up*] 。然后，按照广度优先策略遍历问题的解空间树，在分支结点上，依次搜索该结点的所有孩子结点，分别估算这些孩子结点的目标函数的可能取值，如果某孩子结点的目标函数可能取得的值超出目标函数的界，则将其丢弃，因为从这个结点生成的解不会比目前已经得到的解更好；否则，将其加入待处理结点表（以下简称活结点表）中。依次从活结点表中选取使目标函数的值取得极值的结点成为当前扩展结点，重复上述过程，直到找到最优解。

对解空间树的动态搜索过程

随着这个遍历过程的不断深入，活结点表中所估算的目标函数的界越来越接近问题的最优解。**当搜索到一个叶子结点时**，如果该结点的目标函数值是活结点表中的极值（对于最小化问题，是极小值；对于最大化问题，是极大值），则该叶子结点对应的解就是问题的最优解；**否则**，根据这个叶子结点调整目标函数的界（对于最小化问题，调整上界；对于最大化问题，调整下界），依次考察活结点表中的结点，将超出目标函数界的结点丢弃，然后从活结点表中选取使目标函数取得极值的结点继续进行扩展。

0/1背包问题

例：0/1背包问题。假设有4个物品，其重量分别为(4, 7, 5, 3)，价值分别为(40, 42, 25, 12)，背包容量W=10。首先，将给定物品按单位重量价值从大到小排序，结果如下：

物品	重量(w)	价值(v)	价值/重量(v/w)
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

0/1背包问题

应用贪心法求得近似解为(1, 0, 0, 0)，获得的价值为40，这可以作为0/1背包问题的下界。

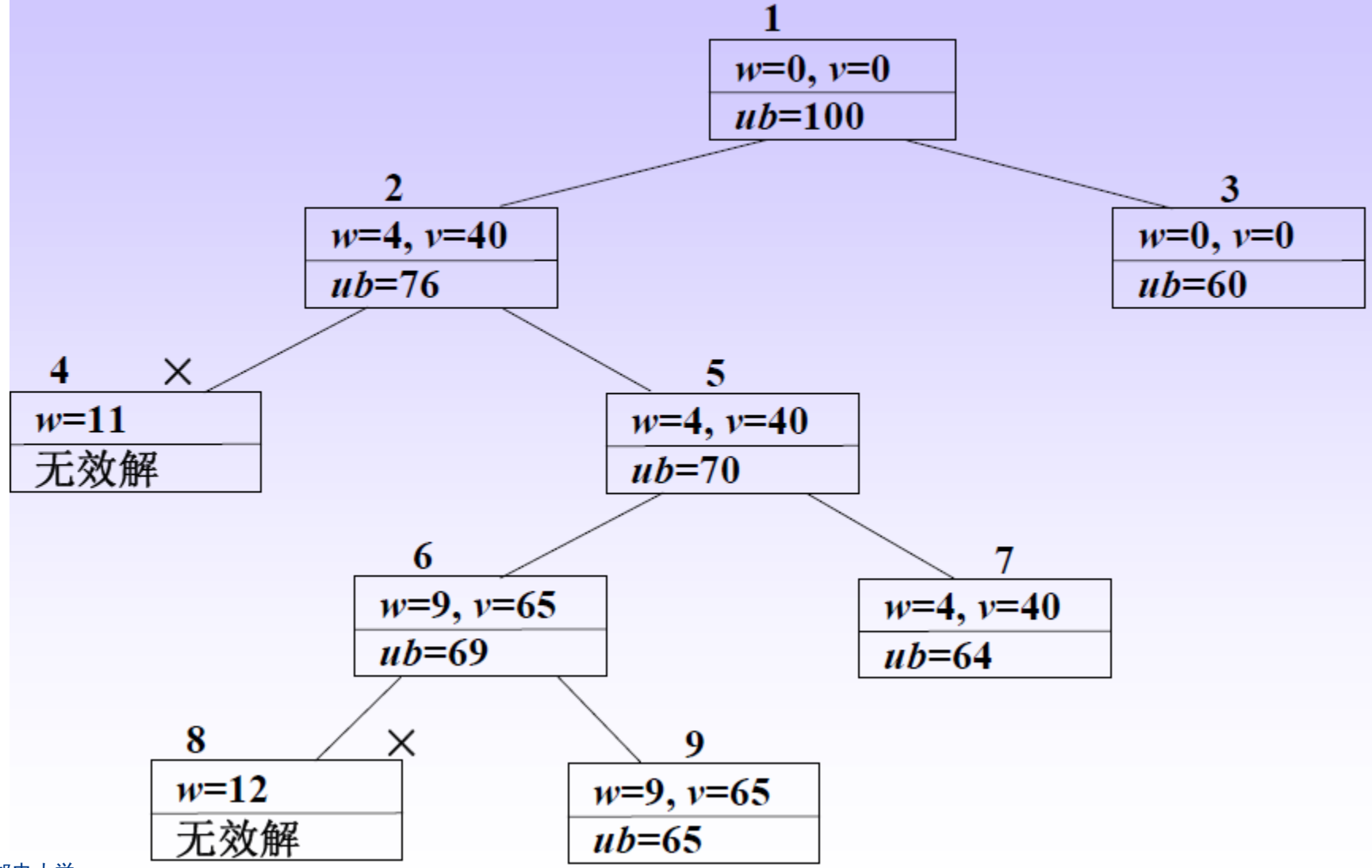
如何求得0/1背包问题的一个合理的上界呢？考虑最好情况，背包中装入的全部是第1个物品且可以将背包装满，则可以得到一个非常简单的上界的计算方法：$ub=W\times(v_1/w_1)=10\times10=100$。于是，得到了目标函数的界[40, 100]。

限界函数为：

$$ub = v + (W - w) \times (v_{i+1} / w_{i+1})$$

优先队列式分支界限法求解0/1背包问题

4个物品的重量分别为(4, 7, 5, 3)，价值分别为(40, 42, 25, 12)，背包容量W=10。



优先队列式分支限界法求解0/1背包问题，其搜索空间如上页图所示，具体的搜索过程如下：

（1）在根结点1，没有将任何物品装入背包，因此，背包的重量和获得的价值均为0，根据限界函数计算结点1的目标函数值为10×10=100；

（2）在结点2，将物品1装入背包，因此，背包的重量为4，获得的价值为40，目标函数值为40 + (10-4)×6=76，将结点2加入待处理结点活结点表中；在结点3，没有将物品1装入背包，因此，背包的重量和获得的价值仍为0，目标函数值为10×6＝60，将结点3加入活结点表中；

（3）在活结点表中选取目标函数值取得极大的结点2优先进行搜索；

（4）在结点4，将物品2装入背包，因此，背包的重量为11，不满足约束条件，将结点4丢弃；在结点5，没有将物品2装入背包，因此，背包的重量和获得的价值与结点2相同，目标函数值为40 + (10-4)×5=70，将结点5加入活结点表中；

（5）在活结点表中选取目标函数值取得极大的结点5优先进行搜索；

（6）在结点6，将物品3装入背包，因此，背包的重量为9，获得的价值为65，目标函数值为65 + (10-9)×4=69，将结点6加入活结点表中；在结点7，没有将物品3装入背包，因此，背包的重量和获得的价值与结点5相同，目标函数值为40 + (10-4)×4＝64，将结点6加入活结点表中；

（7）在活结点表中选取目标函数值取得极大的结点6优先进行搜索；

（8）在结点8，将物品4装入背包，因此，背包的重量为12，不满足约束条件，将结点8丢弃；在结点9，没有将物品4装入背包，因此，背包的重量和获得的价值与结点6相同，目标函数值为65；

（9）由于结点9是叶子结点，同时结点9的目标函数值是活结点表中的极大值，所以，结点9对应的解即是问题的最优解，搜索结束。

优先队列式分支界限法求解最大化问题的一般过程

1. 根据限界函数确定目标函数的界[down, up];
2. 将待处理结点活结点表初始化为空;
3. 对根结点的每个孩子结点x执行下列操作
   3.1 估算结点x的目标函数值value;
   3.2 若(value>=down)，则将结点x加入活结点表中;
4. 循环直到某个叶子结点的目标函数值在活结点表中最大
   4.1 i=活结点表中值最大的结点;
   4.2 对结点i的每个孩子结点x执行下列操作
      4.2.1 估算结点x的目标函数值value;
      4.2.2 若(value>=down)，则将结点x加入活结点表中;
      4.2.3 若(结点x是叶子结点且结点x的value值在活结点表中最大)，
            则将结点x对应的解输出，算法结束;
      4.2.4 若(结点x是叶子结点但结点x的value值在活结点表中不是最大)，
            则令down=value，并且将活结点表中所有小于value的结点删除;

0-1背包问题的算法思想

首先，要对输入数据进行预处理，将各物品依其单位重量价值从大到小进行排列。

在下面描述的优先队列分支限界法中，节点的优先级由已装袋的物品价值加上剩下的最大单位重量价值的物品装满剩余容量的价值和。

算法首先检查当前扩展结点的左儿子结点的可行性。如果该左儿子结点是可行结点，则将它加入到子集树和活结点优先队列中。当前扩展结点的右儿子结点一定是可行结点，仅当右儿子结点满足上界约束时才将它加入子集树和活结点优先队列。当扩展到叶节点时为问题的最优值。

上界函数

```
while (i <= n && w[i] <= cleft)
 // n表示物品总数，cleft为剩余空间
    {
      cleft -= w[i];                    //w[i]表示i所占空间
      b += p[i];                        //p[i]表示i的价值
      i++;
    }
if (i <= n) b += p[i] / w[i] * cleft;   // 装填剩余容量装满背包
return b;                               //b为上界函数
```

0-1背包问题

```
while (i != n+1) {// 非叶结点
   // 检查当前扩展结点的左儿子结点
   Typew wt = cw + w[i];
   if (wt <= c) {// 左儿子结点为可行结点
     if (cp+p[i] > bestp) bestp = cp+p[i];
     AddLiveNode(up, cp+p[i], cw+w[i], true, i+1);}
     up = Bound(i+1);
   // 检查当前扩展结点的右儿子结点
   if (up >= bestp) // 右子树可能含最优解
       AddLiveNode(up, cp, cw, false, i+1);

     //  取下一个扩展节点（略）
}
```

分支限界搜索过程

应用分支限界法的关键问题

（1）如何确定合适的限界函数

（2）如何组织待处理结点表

（3）如何确定最优解中的各个分量

分支限界法对问题的解空间树中结点的处理是跳跃式的，回溯也不是单纯地沿着双亲结点一层一层向上回溯，因此，当搜索到某个叶子结点且该叶子结点的目标函数值在活结点表中最大时（假设求解最大化问题），求得了问题的最优值，但是，却无法求得该叶子结点对应的最优解中的各个分量。这个问题可以用**如下方法①或②**解决：

① 对每个扩展结点保存该结点到根结点的路径；

② 在搜索过程中构建搜索经过的树结构，在求得最优解时，从叶子结点不断回溯到根结点，以确定最优解中的各个分量。

方法①确定0/1背包问题最优解的各分量

例如，在0/1背包问题中，为了对每个扩展结点保存该结点到根结点的路径，将部分解$(x_1, \ldots, x_i)$和该部分解的目标函数值都存储在待处理结点活结点表中，在搜索过程中活结点表的状态如下图所示。

(1)76	(0)60	

(a) 扩展根结点后活结点表状态

(0)60	(1,0)70	

(b) 扩展结点2后活结点表状态

(0)60	(1,0,1)69	(1,0,0)64

(c) 扩展结点5后活结点表状态

(0)60	(1,0,0)64	(1,0,1,0)65

(d) 扩展结点6后活结点表状态，最优解为(1,0,1,0) 65

方法②确定0/1背包问题最优解的各分量

在0/1背包问题中，为了在搜索过程中构建搜索经过的树结构，设一个表ST，在活结点表中取出最小值结点进行扩充时，将最小值结点存储到表ST中，活结点表和表ST的数据结构为(物品*i*-1的选择结果，<物品*i*，物品*i*的选择结果>*ub*)，在搜索过程中活结点表和表ST的状态如下图所示。

PT	(0,<1,1>76)	(0,<1,0>60)	
ST			

(a) 扩展根结点后

PT	(0,<1,0>60)	(1,<2,0>70)	
ST	(0,<1,1>76)		

(b) 扩展结点2后

PT	(0,<1,0>60)	(0,<3,1>69)	(0,<3,0>64)
ST	(0,<1,1>76)	(1,<2,0>70)	

(c) 扩展结点5后

PT	(0,<1,0>60)	(0,<3,0>64)	(1,<4,0>65)
ST	(0,<1,1>76)	(1,<2,0>70)	(0,<3,1>69)

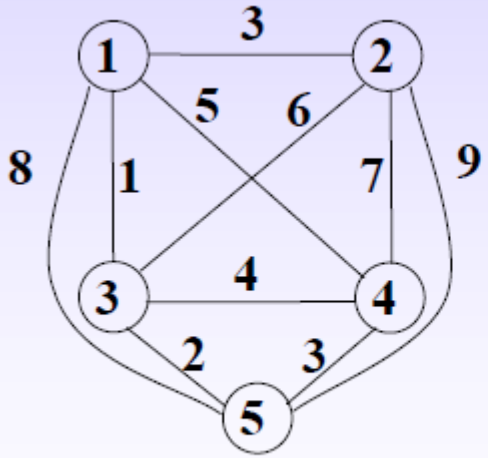
(d) 扩展结点6后，最优解为(1,0,1,0)65

分支限界法的时间性能

分支限界法和回溯法实际上都属于穷举法，当然不能指望它有很好的最坏时间复杂性，遍历具有指数阶个结点的解空间树，在最坏情况下，时间复杂性肯定为指数阶。

与回溯法不同的是，分支限界法首先扩展解空间树中的上层结点，并采用**限界函数**，有利于实行大范围剪枝，同时，根据限界函数不断调整搜索方向，选择最有可能取得最优解的子树优先进行搜索。所以，如果选择了结点的合理扩展顺序以及设计了一个好的限界函数，分支界限法可以快速得到问题的解。

TSP问题



(a) 一个无向图

$$C=\begin{pmatrix} \infty & 3 & 1 & 5 & 8 \\ 3 & \infty & 6 & 7 & 9 \\ 1 & 6 & \infty & 4 & 2 \\ 5 & 7 & 4 & \infty & 3 \\ 8 & 9 & 2 & 3 & \infty \end{pmatrix}$$

(b) 无向图的代价矩阵

TSP问题是指旅行家要旅行n个城市，要求各个城市经历且仅经历一次然后回到出发城市，并要求所走的路程最短。

TSP问题

采用贪心法求得近似解为1→3→5→4→2→1，其路径长度为1+2+3+7+3=16，这可以作为TSP问题的上界。

把矩阵中每一行最小的元素相加，可以得到一个简单的下界，其路径长度为1+3+1+3+2=10，但是还有一个信息量更大的下界：考虑一个TSP问题的完整解，在每条路径上，每个城市都有两条邻接边，一条是进入这个城市的，另一条是离开这个城市的，那么，如果把矩阵中每一行最小的两个元素相加再除以2，如果图中所有的代价都是整数，再对这个结果向上取整，就得到了一个合理的下界。

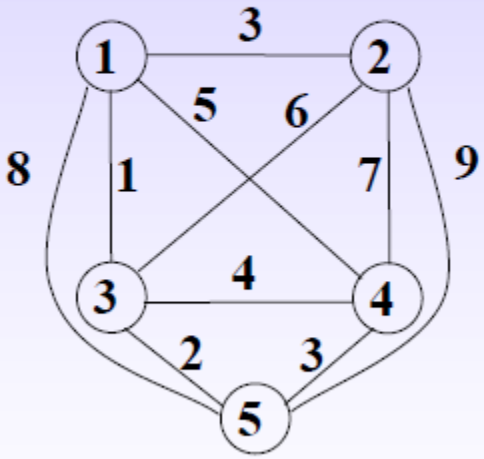
$$lb=((1+3)+(3+6)+(1+2)+(3+4)+(2+3))/2=14$$

于是，得到了目标函数的界[14, 16]。
需要强调的是，这个解并不是一个合法的选择（可能没有构成哈密顿回路），它仅仅给出了一个参考下界。

部分解的目标函数值的计算方法

$$lb=\left(\sum_{i=1}^{k-1}2c[r_i][r_{i+1}]+\sum_{r_i\in U}r_i\text{行不在路径上的最小元 素}+\sum_{r_j\notin U}r_j\text{行最小的两个元素}\right)/2$$

例如，以下无向图中，如果部分解包含边(1, 4)，则该部分解的下界是*lb*=((1+5)+(3+6)+(1+2)+(3+5)+(2+3))/2=16。

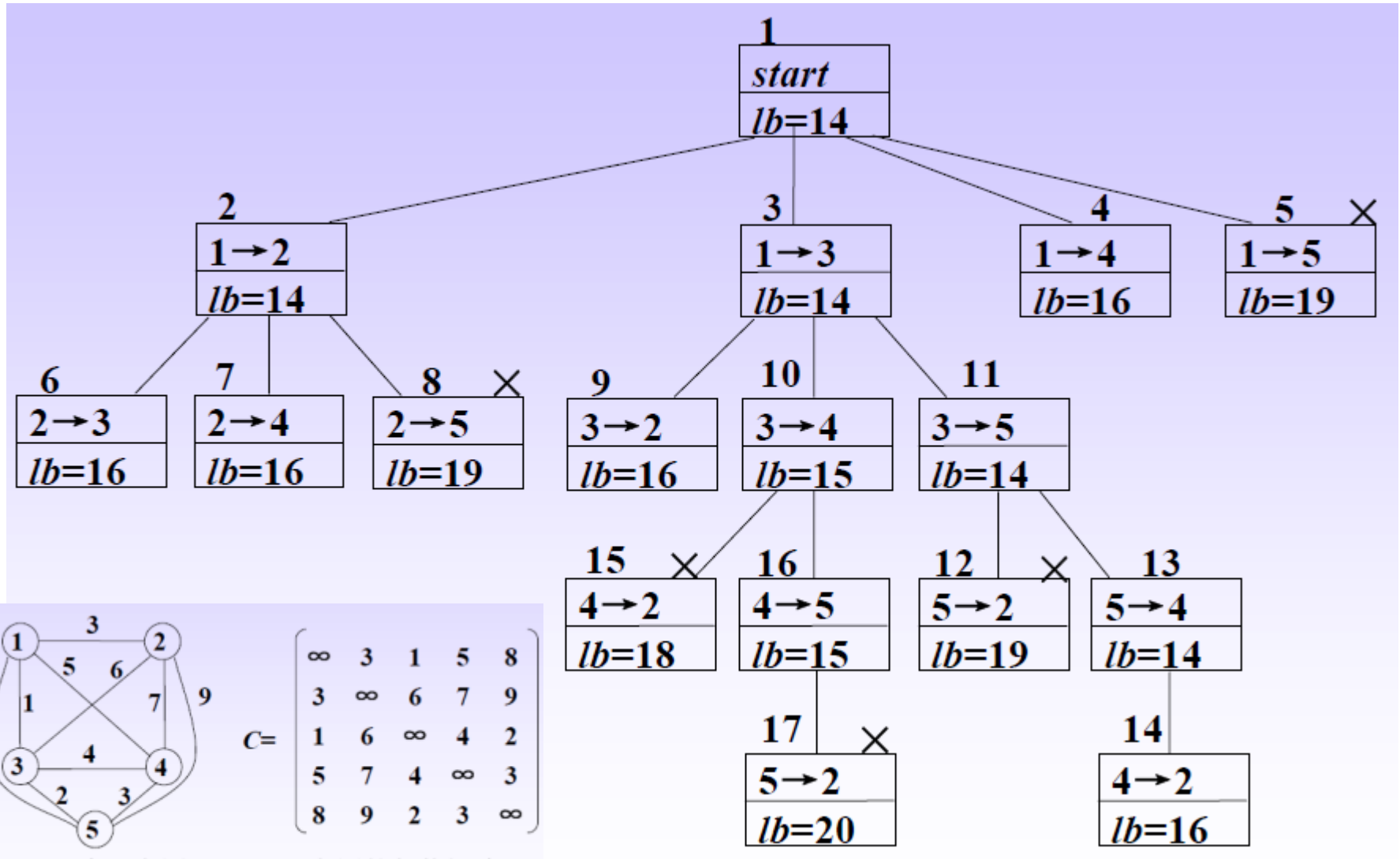


(a) 一个无向图

$$C=\begin{bmatrix}\infty & 3 & 1 & 5 & 8\\ 3 & \infty & 6 & 7 & 9\\ 1 & 6 & \infty & 4 & 2\\ 5 & 7 & 4 & \infty & 3\\ 8 & 9 & 2 & 3 & \infty\end{bmatrix}$$

(b) 无向图的代价矩阵

分支限界法求解TSP问题示例



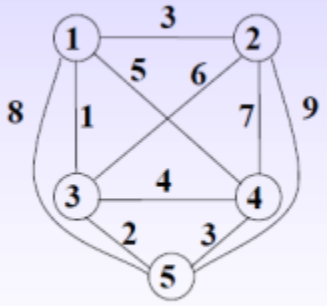
(a) 一个无向图

$$C=\begin{pmatrix} \infty & 3 & 1 & 5 & 8 \\ 3 & \infty & 6 & 7 & 9 \\ 1 & 6 & \infty & 4 & 2 \\ 5 & 7 & 4 & \infty & 3 \\ 8 & 9 & 2 & 3 & \infty \end{pmatrix}$$

(b) 无向图的代价矩阵

（1）在根结点1，根据限界函数计算目标函数的值为 $lb=((1+3)+(3+6)+(1+2)+(3+4)+(2+3))/2=14$；

（2）在结点2，从城市1到城市2，路径长度为3，目标函数的值为$((1+3)+(3+6)+(1+2)+(3+4)+(2+3))/2=14$，将结点2加入待处理结点活结点表中；在结点3，从城市1到城市3，路径长度为1，目标函数的值为$((1+3)+(3+6)+(1+2)+(3+4)+(2+3))/2=14$，将结点3加入活结点表中；在结点4，从城市1到城市4，路径长度为5，目标函数的值为$((1+5)+(3+6)+(1+2)+(3+5)+(2+3))/2=16$，将结点4加入活结点表中；在结点5，从城市1到城市5，路径长度为8，目标函数的值为$((1+8)+(3+6)+(1+2)+(3+5)+(2+8))/2=19$，超出目标函数的界，将结点5丢弃；



(a) 一个无向图

$$C=\begin{pmatrix} \infty & 3 & 1 & 5 & 8 \\ 3 & \infty & 6 & 7 & 9 \\ 1 & 6 & \infty & 4 & 2 \\ 5 & 7 & 4 & \infty & 3 \\ 8 & 9 & 2 & 3 & \infty \end{pmatrix}$$

(b) 无向图的代价矩阵

（3）在活结点表中选取目标函数值极小的结点2优先进行搜索；
（4）在结点6，从城市2到城市3，目标函数值为((1+3)+(3+6)+(1+6)+(3+4)+(2+3))/2=16，将结点6加入活结点表中；在结点7，从城市2到城市4，目标函数值为((1+3)+(3+7)+(1+2)+(3+7)+(2+3))/2=16，将结点7加入活结点表中；在结点8，从城市2到城市5，目标函数值为((1+3)+(3+9)+(1+2)+(3+4)+(2+9))/2=19，超出目标函数的界，将结点8丢弃；
（5）在活结点表中选取目标函数值极小的结点3优先进行搜索；
（6）在结点9，从城市3到城市2，目标函数值为((1+3)+(3+6)+(1+6)+(3+4)+(2+3))/2=16，将结点9加入活结点表中；在结点10，从城市3到城市4，目标函数值为((1+3)+(3+6)+(1+4)+(3+4)+(2+3))/2=15，将结点10加入活结点表中；在结点11，从城市3到城市5，目标函数值为((1+3)+(3+6)+(1+2)+(3+4)+(2+3))/2=14，将结点11加入活结点表中；

（7）在活结点表中选取目标函数值极小的结点11优先进行搜索；

（8）在结点12，从城市5到城市2，目标函数值为((1+3)+(3+9)+(1+2)+(3+4)+(2+9))/2=19，超出目标函数的界，将结点12丢弃；在结点13，从城市5到城市4，目标函数值为((1+3)+(3+6)+(1+2)+(3+4)+(2+3))/2=14，将结点13

加入活结点表中；

（9）在活结点表中选取目标函数值极小的结点13优先进行搜索；

（10）在结点14，从城市4到城市2，目标函数值为((1+3)+(3+7)+(1+2)+(3+7)+(2+3))/2=16，最后从城市2回到城市1，目标函数值为((1+3)+(3+7)+(1+2)+(3+7)+(2+3))/2=16，由于结点14为叶子结点，得到一个可行解，其路径长度为16；

（11）在活结点表中选取目标函数值极小的结点10优先进行搜索；

（12）在结点15，从城市4到城市2，目标函数的值为((1+3)+(3+7)+(1+4)+(7+4)+(2+3))/2=18，超出目标函数的界，将结点15丢弃；在结点16，从城市4到城市5，目标函数值为((1+3)+(3+6)+(1+4)+(3+4)+(2+3))/2=15，将结点16加入活结点表中；

（13）在活结点表中选取目标函数值极小的结点16优先进行搜索；

（14）在结点17，从城市5到城市2，目标函数的值为((1+3)+(3+9)+(1+4)+(3+4)+(9+3))/2=20，超出目标函数的界，将结点17丢弃；

（15）活结点表中目标函数值均为16，且有一个是叶子结点14，所以，结点14对应的解1→3→5→4→2→1即是TSP问题的最优解，搜索过程结束。

(1, 2)14	(1, 3)14	(1, 4)16

(a) 扩展根结点后的状态

(1, 3)14	(1, 4)16	(1, 2, 3)16	(1, 2, 4)16

(b) 扩展结点2后的状态

(1, 4)16	(1, 2, 3)16	(1, 2, 4)16	(1, 3, 2)16	(1, 3, 4)15	(1, 3, 5)14

(c) 扩展结点3后的状态

(1, 4)16	(1, 2, 3)16	(1, 2, 4)16	(1, 3, 2)16	(1, 3, 4)15	(1, 3, 5, 4)14

(d) 扩展结点11后的状态

(1, 4)16	(1, 2, 3)16	(1, 2, 4)16	(1, 3, 2)16	(1, 3, 4)15	(1, 3, 5, 4, 2)16

(e) 扩展结点13后的状态

(1, 4)16	(1, 2, 3)16	(1, 2, 4)16	(1, 3, 2)16	(1, 3, 5, 4, 2)16	(1, 3, 4, 5)15

(f) 扩展结点10后的状态

(1, 4)16	(1, 2, 3)16	(1, 2, 4)16	(1, 3, 2)16	(1, 3, 5, 4, 2)16	(1, 3, 4, 5)15

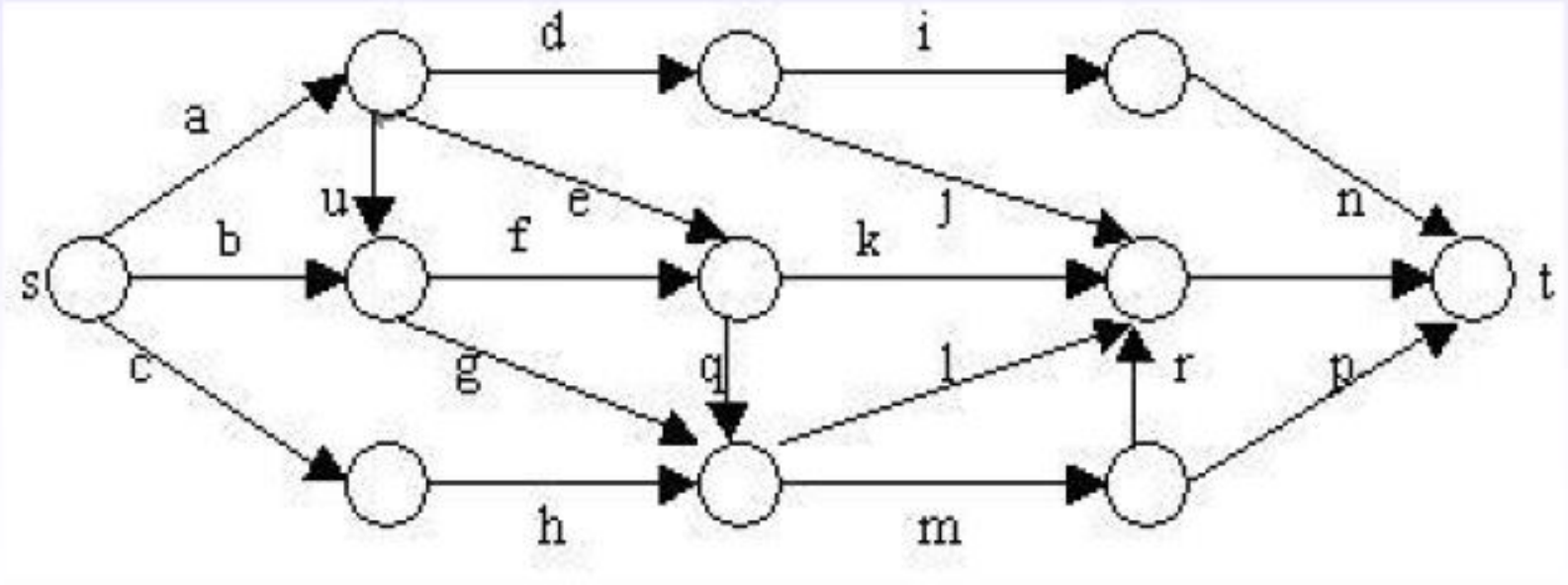
(g) 扩展结点16后的状态，最优解为1→3→5→4→2→1

TSP问题的算法描述

1. 根据限界函数计算目标函数的下界down；采用贪心法得到上界up；
2. 将待处理结点活结点表初始化为空；
3. for (i=1; i<=n; i++)
   x[i]=0;
4. k=1; x[1]=1; //从顶点1出发求解TSP问题
5. while (k>=1)
   5.1 i=k+1;
   5.2 x[i]=1;
   5.3 while (x[i]<=n)
      5.3.1 如果路径上顶点不重复，则
         5.3.1.1 计算目标函数值lb；
         5.3.1.2 if (lb<=up) 将路径上的顶点和lb值存储在活结点表中；
      5.3.2 x[i]=x[i]+1;
   5.4 若i=n且叶子结点的目标函数值在活结点表中最小
      则将该叶子结点对应的最优解输出；
   5.5否则，若i=n，则在活结点表中取叶子结点的目标函数值最小的结点lb，
      令up=lb，将活结点表中目标函数值lb超出up的结点删除；
   5.6 k=活结点表中lb最小的路径上顶点个数；

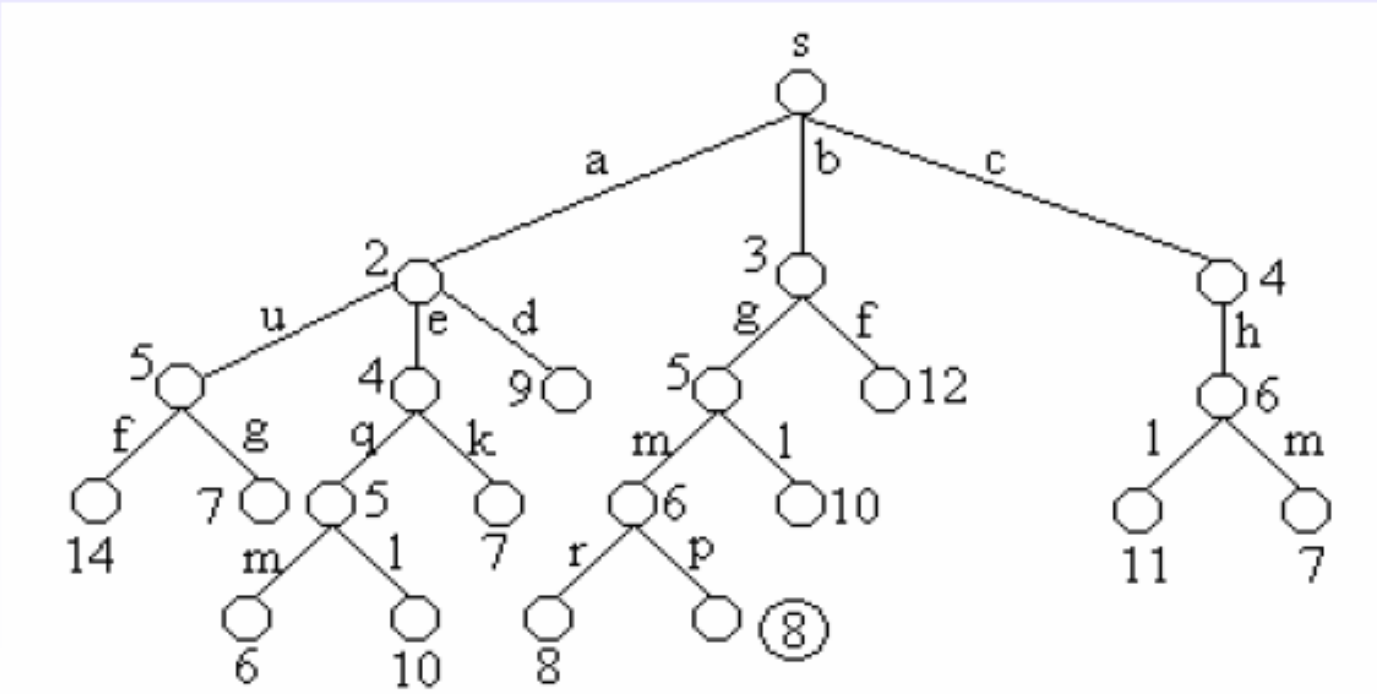
单源最短路径问题

下面以一个例子来说明单源最短路径问题：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。



单源最短路径问题

下图是用优先队列式分支限界法解有向图G的单源最短路径问题产生的解空间树。其中，每一个结点旁边的数字表示该结点所对应的当前路长。



单源最短路径问题 -- 算法思想

解单源最短路径问题的优先队列式分支限界法用一极小堆来存储活结点表。其优先级是结点所对应的当前路长。

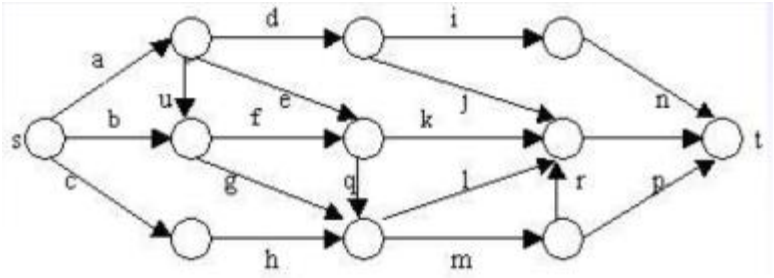
算法从图G的源顶点s和空优先队列开始。结点s被扩展后，它的儿子结点被依次插入堆中。此后，算法从堆中取出具有最小当前路长的结点作为当前扩展结点，并依次检查与当前扩展结点相邻的所有顶点。**如果从当前扩展结点i到顶点j有边可达，且从源出发，途经顶点i再到顶点j的所相应的路径的长度小于当前最优路径长度，则将该顶点作为活结点插入到活结点优先队列中。**这个结点的扩展过程一直继续到活结点优先队列为空时为止。

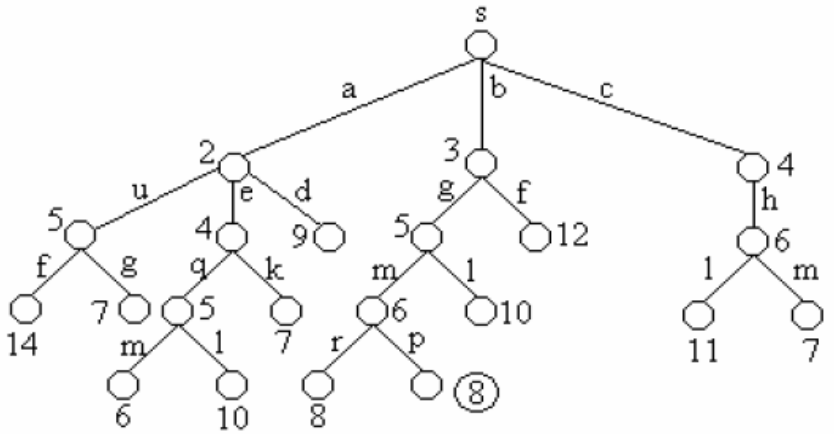
剪枝策略

由于图G中各边的权均非负，所以结点所对应的当前路长也是解空间树中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。

在算法中，**利用结点间的控制关系进行剪枝**。从源顶点s出发，2条不同路径到达图G的同一顶点。由于两条路径的路长不同，因此可以将路长长的路径所对应的树中的结点为根的子树剪去。

例如在上例中，从源顶点s出发，经过边a，e，q(路长为5)和经过边c，h(路长为6)的2条路径到达图G的同一顶点。在该问题的解空间树中，这2条路径相应于解空间树的2个不同的结点A和B。由于结点A所相应的路长小于结点B所相应的路长，因此以结点A为根的子树中所包含的从s到t的路长小于以结点B为根的子树中所包含的从s到t的路长。因而可以将以结点B为根的子树剪去。在这种情况下，称结点A控制了结点B。





算法实现

```
while (true) {
    for (int j = 1; j <= n; j++)
      if ((c[E.i][j]<inf)&&(E.length+c[E.i][j]<dist[j])) {
        // 顶点i到顶点j可达，且满足控制约束
        dist[j]=E.length+c[E.i][j];
        prev[j]=E.i;
        // 加入活结点优先队列
        MinHeapNode<Type> N;
        N.i=j;
        N.length=dist[j];
        H.Insert(N);}
    try {H.DeleteMin(E);}        // 取下一扩展结点
    catch (OutOfBounds) {break;} // 优先队列空
    }
}
```

顶点i和j间有边，且此路径长小于原先从原点到j的路径长