6.3 装载问题

1. 问题描述

子集选取问题

- 有一批共n个集装箱要装上2艘载重量分别为C1和C2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。

特殊的0-1背包

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

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

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

6.3 装载问题

1. 问题描述

- 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。

6.3 装载问题

2. 队列式分支限界法-算法实现

时间复杂度是多少？

```
Maxloading(w[],c,n)
{InitQueue(Q）; Add(Q,-1); // 初始化队列，同层尾结点表示
   i=1; Ew=0 ;    // 扩展结点层次，扩展结点对应的载重量
   bestw=0 ;          //当前最优载重量
  while (true) {          // 搜索子集树
if (Ew + w[i] <= c) //检查左儿子结点， x[i] = 1
      EnQueue(Q, Ew + w[i], bestw, i, n);// 右儿子结点总是可行的
   EnQueue(Q, Ew, bestw, i, n);   // x[i] = 0
   Q_Delete(Ew);              // 取下一扩展结点
   if (Ew == -1) {              // 同层结点尾部
      if (QIsEmpty()) return bestw;
       Add(Q，-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    } }
```

6.3 装载问题

2. 队列式分支限界法

- 在算法的while循环中，首先检测当前扩展结点的左儿子结点是否为可行结点。如果是则将其加入到活结点队列中。然后将其右儿子结点加入到活结点队列中(右儿子结点一定是可行结点)。2个儿子结点都产生后，当前扩展结点被舍弃。

- 技巧：活结点队列中的队首元素被取出作为当前扩展结点，由于队列中每一层结点之后都有一个尾部标记-1，故在取队首元素时，活结点队列一定不空。当取出的元素是-1时，再判断当前队列是否为空。如果队列非空，则将尾部标记-1加入活结点队列，算法开始处理下一层的活结点。

6.3 装载问题

3. 算法的改进

- 节点的左子树表示将此集装箱装上船，右子树表示不将此集装箱装上船。设bestw是当前最优解；Ew是当前扩展结点所相应的重量；r是剩余集装箱的重量。

- 则当Ew+r≤bestw时，可将其右子树剪去。

- 为了确保右子树成功剪枝，应该在算法每一次进入左子树的时候更新bestw的值。

6.3 装载问题

3. 算法的改进——while（）内

```
// 检查左儿子结点
Type wt = Ew + w[i];
        // 左儿子结点的重量
    if (wt <= c) {
            // 可行结点
        if (wt > bestw) bestw = wt;
        // 加入活结点队列
        if (i < n) Add(Q，wt);
}
```

提前更新bestw

右儿子剪枝

```
// 检查右儿子结点
if (Ew + r > bestw && i < n)
Add(Q，Ew);     // 可能含最优解
Q_Delete(Q,Ew); // 取下一扩展结点
if (Ew == -1) {          // 同层结点尾部
 if (QIsEmpty()) return bestw;
 Add(Q，-1); // 同层结点尾部标志
Q_Delete(Q,Ew);
i++;
r-=w[i];
```

6.3 装载问题

4. 构造最优解

- 为了在算法结束后能方便地构造出与最优值相应的最优解，算法必须存储相应子集树中从活结点到根结点的路径。为此目的，可在每个结点处设置指向其父结点的指针，并设置左、右儿子标志。

```
class QNode
{QNode *parent;  // 指向父结点的指针
    bool LChild;        // 左儿子标志
    Type weight;        // 结点所相应的载重量
}
```

6.3 装载问题

4. 构造最优解

- 找到最优值后，可以根据parent回溯到根节点，找到最优解。

```
for (int j = n - 1; j > 0; j--) {
    bestx[j] = bestE->LChild;
    bestE = bestE->parent;
}
Return bestw;
```

6.3 装载问题

5. 优先队列式分支限界法

- 装载问题的优先队列式分支限界法用**最大优先队列**存储活结点表。活结点x在优先队列中的优先级定义为**从根结点到结点x的路径所相应的载重量再加上剩余集装箱的重量之和**。
- 优先队列中优先级**最大**的活结点成为下一个扩展结点。以结点x为根的子树中所有结点相应的路径的载重量**不超过**它的优先级。子集树中叶结点所相应的载重量与其优先级相同。
- 在优先队列式分支限界法中，**一旦有一个叶结点成为当前扩展结点，则可以断言该叶结点所相应的解即为最优解。**此时可终止算法。

6.4 旅行售货员问题（TSP）

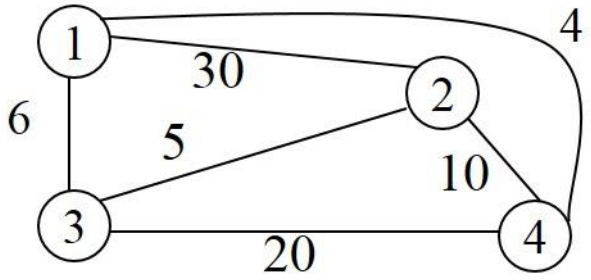
- 某售货员要到若干城市去推销商品，已知各城市之间的路程(或旅费)。他要选定一条从驻地出发，经过每个城市一次，最后**回到驻地**的路线，使总的**路程(或总旅费)最小**。

- 路线是一个带权图。图中各边的**费用（权）为正数**。图的一条周游路线是包括V中的每个顶点在内的**一条回路**。周游路线的费用是这条路线上所有边的费用之和。

- 旅行售货员问题的解空间树，从树的根结点到任一叶结点的路径定义了图的一条周游路线。旅行售货员问题要在图G中找出费用最小的周游路线。

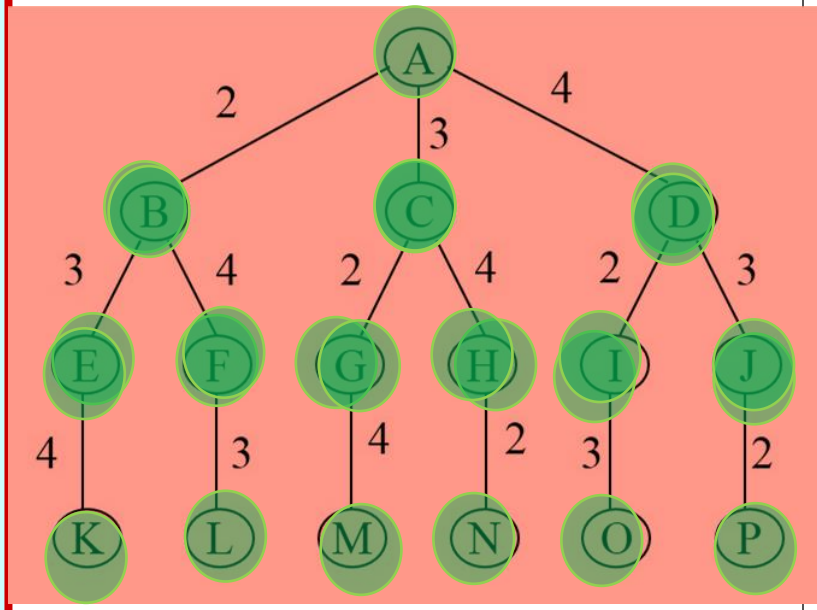
实例

2 TSP问题的两种分支定界法



队列式分支限界法：

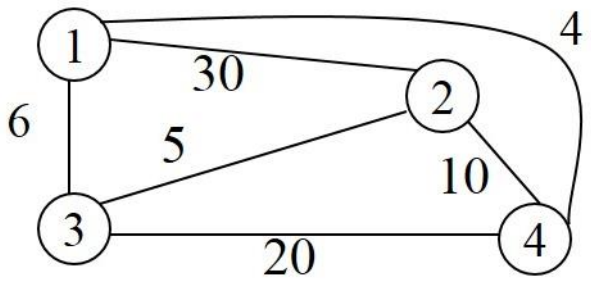
- [A] B, C, D => B,C,D
- [B, C, D] E, F => E,F
- [C, D, E, F] G, H => G,H
- [D, E, F, G, H] I, J => I,J
- [E, F, G, H, I, J] K(59)
  [1,2,3,4]
- [F, G, H, I, J] L(66)
- [G, H, I, J] M(25) [1, 3, 2, 4]
- [H, I, J] 1-3-4(26)
- [I, J]O => O(25)
- [J]P => P(59)



两个实

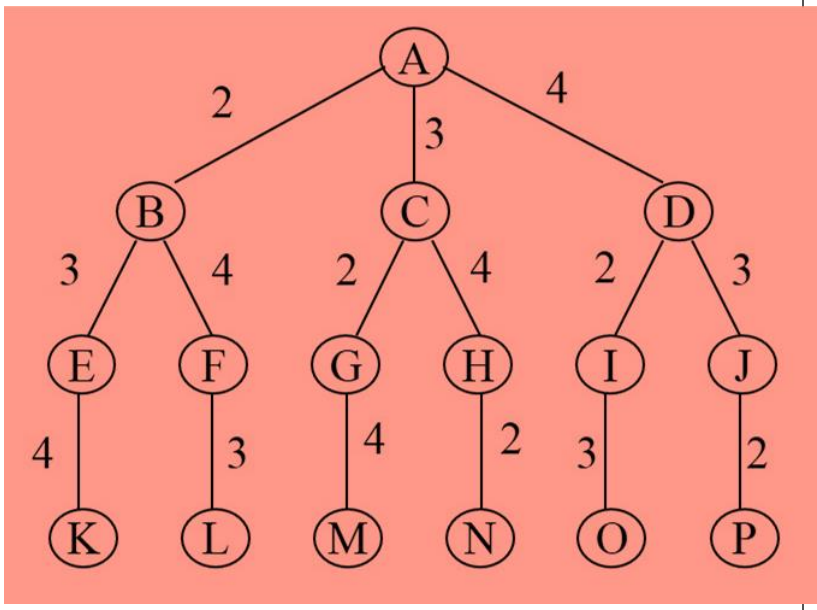
2 TSP问题的

优先级的选取：
根到该结点的路径长度

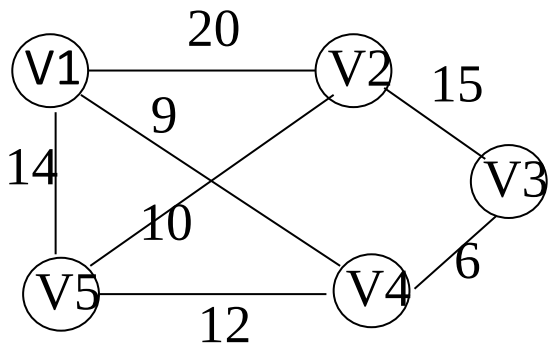


优先队列式分支限界法：

- [A] B, C, D => B(30), C(6), D(4)
- [D, B, C] I, J => I(14), J(24)
- [C, B, I, J] G, H => G(11), H(26)
- [G, B, I, J, H] M => M(25)
  [1, 3, 2, 4]
- [I, J, H, B] O => O(25)
- [J, H, B] P => P(59)
- [H, B] B, H 限界掉



TSP问题



课堂习题：
从V1出发回到V1，采用优先队列写出求解过程。

6.4 旅行售货员问题

2. 算法描述

算法开始时创建一个最小堆，用于表示活结点优先队列。堆中每个结点的子树费用的下界lcost值是优先队列的优先级。接着算法计算出图中每个顶点的最小费用出边并用minout记录。如果所给的有向图中某个顶点没有出边，则该图不可能有回路，算法即告结束。如果每个顶点都有出边，则根据计算出的minout作算法初始化。

算法的while循环体完成对排列树内部结点的扩展。对于当前扩展结点，算法分2种情况进行处理：

6.4 旅行售货员问题

2. 算法描述

1、首先考虑s=n-2的情形，此时当前扩展结点是排列树中某个叶结点的父结点。如果该叶结点相应一条可行回路且费用小于当前最小费用，则将该叶结点插入到优先队列中，否则舍去该叶结点。

2、当s<n-2时，算法依次产生当前扩展结点的所有儿子结点。由于当前扩展结点所相应的路径是x[0:s]，其可行儿子结点是从剩余顶点x[s+1:n-1]中选取的顶点x[i]，且(x[s]，x[i])是所给有向图G中的一条边。对于当前扩展结点的每一个可行儿子结点，计算出其前缀(x[0:s]，x[i])的费用cc和相应的下界lcost。当lcost<bestc时，将这个可行儿子结点插入到活结点优先队列中。

6.4 旅行售货员问题

2. 算法描述

- **算法中while循环的终止条件是排列树的一个叶结点成为当前扩展结点。**
  - 当s=n-1时，已找到的回路前缀是x[0:n-1]，它已包含图G的所有n个顶点。
  - 因此，当s=n-1时，相应的扩展结点表示一个叶结点。此时该叶结点所相应的回路的费用等于cc和lcost的值。
  - 剩余的活结点的lcost值不小于已找到的回路的费用。它们都不可能导致费用更小的回路。
  - 因此已找到的叶结点所相应的回路是一个最小费用旅行售货员回路，算法可以结束。
- **算法结束时返回找到的最小费用，相应的最优解由数组X给出。**

- 本章结束！