第6章　分支限界法
(Branch and Bound)

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)单源最短路径问题；(2)装载问题；(3)0-1背包问题；

(4)最大团问题；(5)旅行售货员问题；(6)批处理作业调度问题

6.1 分支限界法的基本思想

分支限界法与回溯法

- 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。

- 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。

6.1 分支限界法的基本思想

- 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。

6.1 分支限界法的基本思想

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。

数据结构：队列

（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。

数据结构：堆

最大优先队列：最大堆，体现最大效益优先

最小优先队列：最小堆，体现最小费用优先

队列式(FIFO)分支限界法

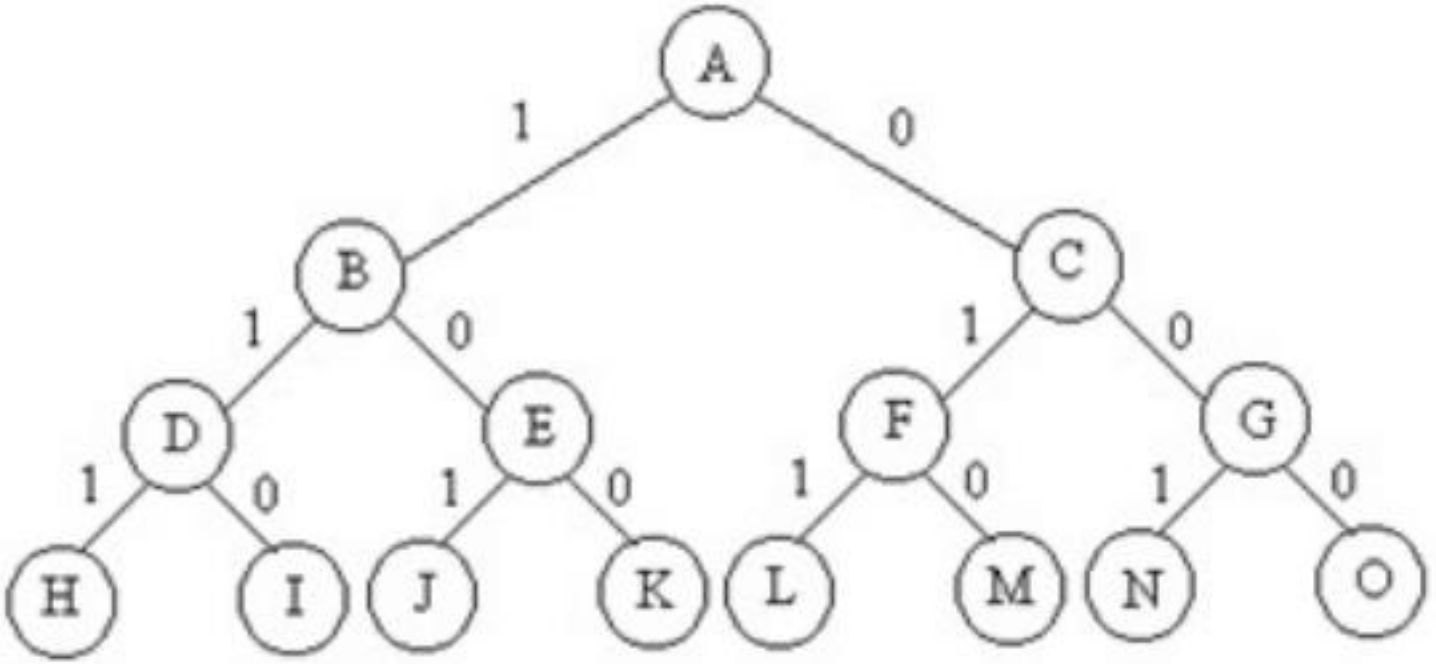
- 搜索策略：一开始，根结点是唯一的活结点，根结点入队。从活结点队中取出根结点后，作为当前扩展结点。对当前扩展结点，先从左到右地产生它的所有儿子，用约束条件检查，把所有满足约束函数的儿子加入活结点队列中。再从活结点表中取出队首结点为当前扩展结点，……，直到找到一个解或活结点队列为空为止。

优先队列式分支限界法

- 搜索策略：对每一活结点计算一个优先级（某些信息的函数值），并根据这些优先级；从当前活结点表中优先选择一个优先级最高（最有利）的结点作为扩展结点，使搜索朝着解空间树上有最优解的分支推进，以便尽快地找出一个最优解。再从活结点表中下一个优先级别最高的结点为当前扩展结点，……，直到找到一个解或活结点队列为空为止。

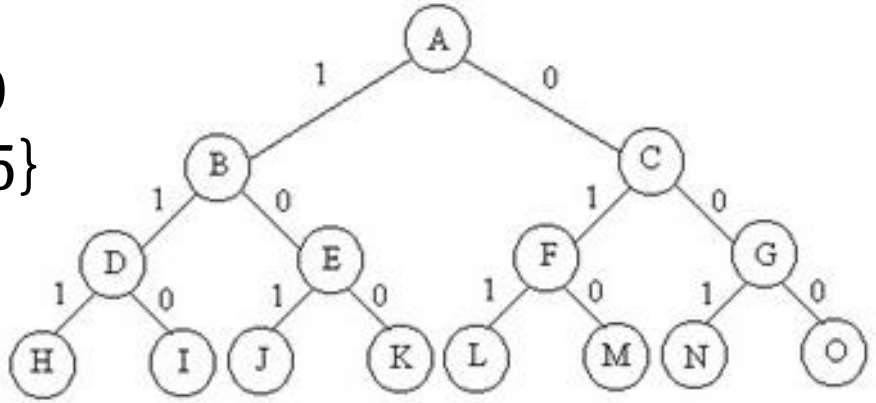
分支限界法搜索应用举例

- 0-1背包问题，当n=3时，w={16, 15, 15}，p={45, 25, 25}， c=30



分支限界法搜索应用举例

0-1背包问题，当n=3时， c=30
w={16, 15, 15}， p={45, 25, 25}

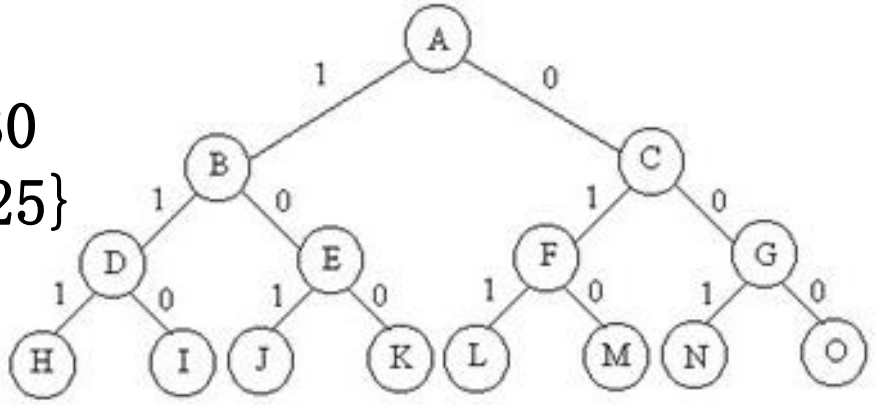


- 队列式分支限界法(处理法则:先进先出)：{}—>{A}—>{B,C}—>{C,D,E}(***D是不可行解，舍弃***)—>{C,E}—>{E,F,G}—>{F,G,J,K}(***J是不可行解，舍弃***)—>{F,G,K}—>{G,K,L,M}—>{K,L,M,N,O}—>{}

解空间广度搜索

分支限界法搜索应用举例

0-1背包问题，当n=3时， c=30
w={16, 15, 15}， p={45, 25, 25}



- 优先队列式分支限界法(处理法则:价值大者优先)：{}—>{A}—>{B(45),C(0)}—>{C(0)，D(x)，E(45)}—>{C(0)，E(45)}—>{C(0)，J(x)，K(45)}—>{C}—>{F(25)，G(0)}—>{G(0)，L(50)，M(25)}—>{G(0)，M(25)}—>{G}—>{N，O}—>{O}—>{}

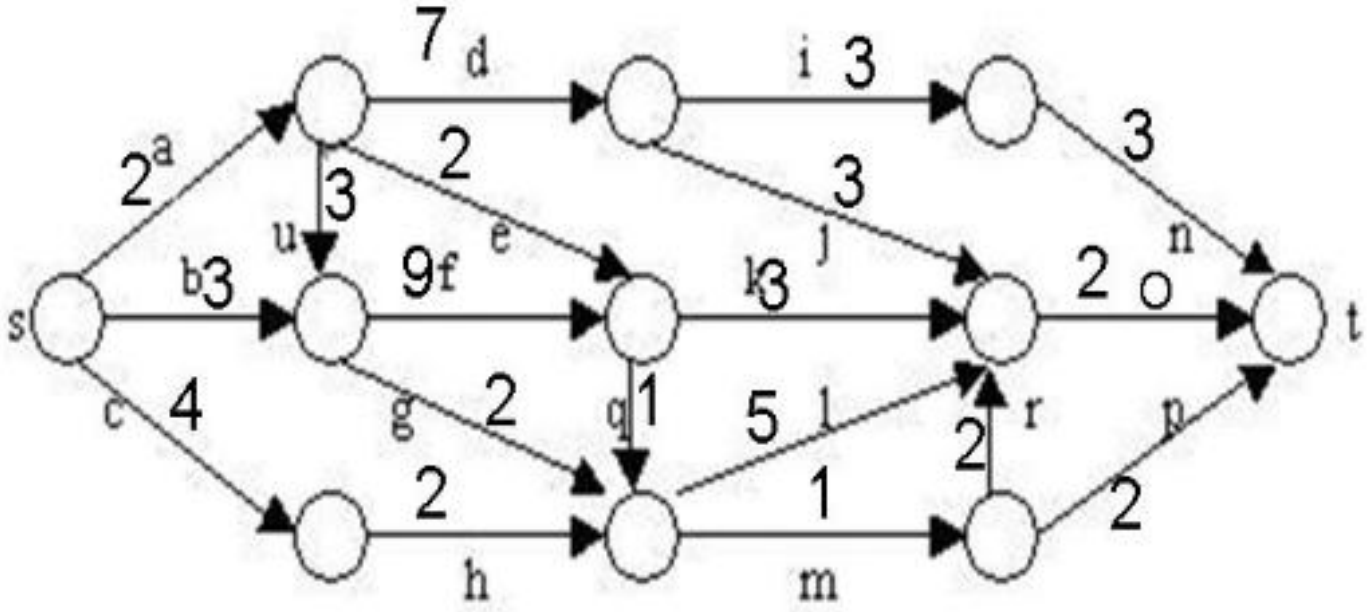
算法步骤

1. 定义解空间
2. 确定易于搜索的**解空间结构**（二叉树。。）
3. 确定结点的**数据结构**
4. 确定**约束条件**与**限界条件**用于剪枝
5. 确定活结点组织方法（队列/优先队列（**优先级函数**））
6. 确定答案节点识别方法

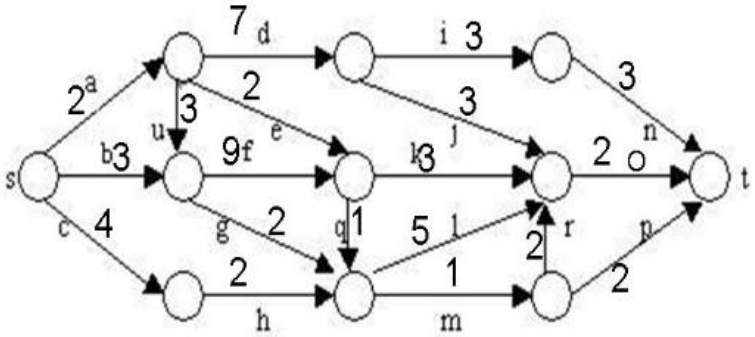
6.2 单源最短路径问题

1. 问题描述

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权值。要求图G的从源顶点s到目标顶点t之间的最短路径。

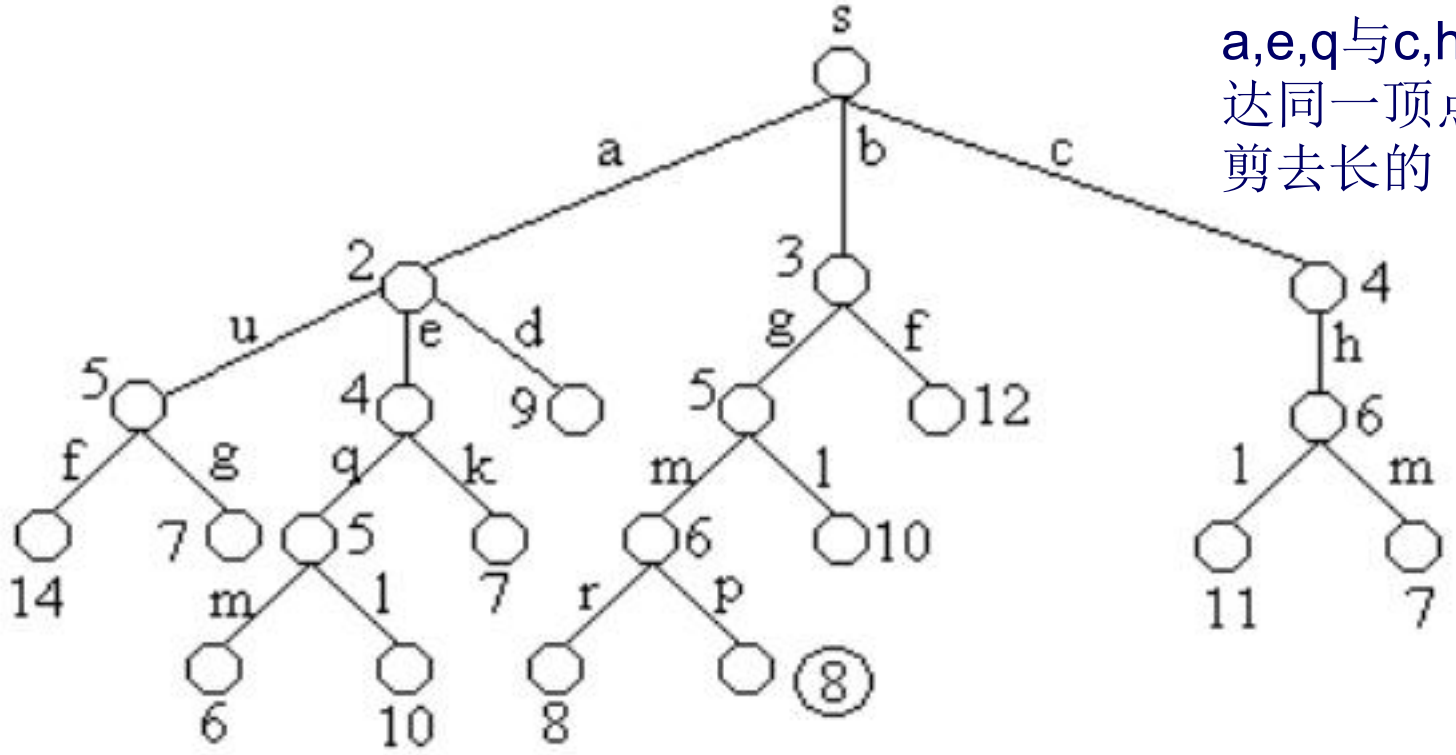


6.2 单源最短路径



1. 问题描述

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。



a,e,q与c,h到达同一顶点，剪去长的

6.2 单源最短路径问题

2. 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。

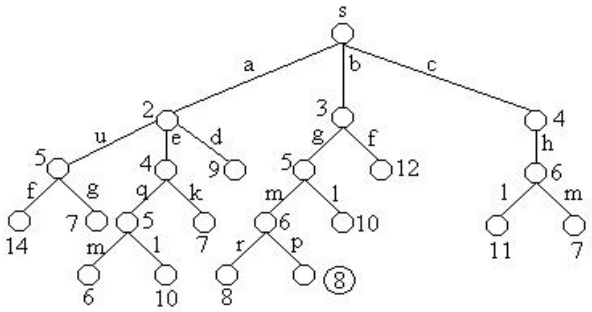
- 算法从图G的源顶点s和空优先队列开始。结点s被扩展后，它的儿子结点被依次插入堆中。此后，算法从堆中取出具有最小当前路长的结点作为当前扩展结点，并依次检查与当前扩展结点相邻的所有顶点。如果从当前扩展结点i到顶点j有边可达，且从源出发，途经顶点i再到顶点j的所相应的路径的长度小于当前最优路径长度，则将该顶点作为活结点插入到活结点优先队列中。这个结点的扩展过程一直继续到活结点优先队列为空时为止。

6.2 单源最短路径问题

3. 剪枝策略

- 在算法扩展结点的过程中，一旦发现一个结点的下界不小于当前找到的最短路长，则算法剪去以该结点为根的子树。
- 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。

6.2 单源最短路径问题



一次性生成所有儿子节点

```
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

6.3 装载问题

1. 问题描述

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Wi，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个集装箱装上这2艘轮船。如果有，找出一种装载方案。

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

- 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。

- 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

2. 队列式分支限界法

```
while (true) {
    // 检查左儿子结点
    if (Ew + w[i] <= c) // 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 (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      } }
```

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。

- 另外，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

```
// 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) Q.Add(wt);
}
```

提前更新bestw

```
// 检查右儿子结点
if (Ew + r > bestw && i < n)
        Q.Add(Ew);     // 可能含最优解
Q.Delete(Ew);     // 取下一扩展结点
```

右儿子剪枝

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。

```
// 构造当前最优解
for (int j = n - 1; j > 0; j--) {
    bestx[j] = bestE->LChild;
    bestE = bestE->parent;
}
```

```
template<class Type>
void EnQueue(Queue<QNode<Type> *> &Q, Type wt, int i, int n, Type bestw, QNode<Type>*E,
             QNode<Type> * &bestE, int bestx[], bool ch) {       // 将活结点加入到活结点队列Q中
  if (i == n) {                              // 可行叶结点
    if (wt == bestw) {
      bestE = E;                             // 当前最优载重量
      bestx[n] = ch;
    }
    return;
  }
  // 非叶结点
  QNode<Type> *b;
  b = new QNode<Type>;
  b->weight = wt;
  b->parent = E;
  b->LChild = ch;
  Q.Add(b);
}
```

6.3 装载问题

5. 优先队列式分支限界法

- 解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。 （比剩余节点可能最好的还要好）

6.3 装载问题

树结点数据结构：

```
template <class Type>class HeapNode;
class bbnode {
   friend void AddLiveNode(MaxHeap<HeapNode<int>> &, bbnode *, int, bool, int);
   friend int MaxLoading(int*, int, int, int*);
   friend class AdjacencyGraph;
private:
   bbnode  *parent;                          // 指向父结点的指针
   bool  LChild;                             // 左儿子结点标志
};
```

6.3 装载问题

堆结点数据结构：

```
template<class Type>
class HeapNode {
   friend void AddLiveNode(MaxHeap<HeapNode<Type>> &, bbnode*, Type, bool, int);
   friend Type MaxLoading(Type*, Type, int, int*);
public:
   operator Type () const {  return uweight;  }
private:
   bbnode  *ptr;                          // 指向活结点在子集树中相应结点的指针
   Type  uweight;                         // 活结点优先级(上界)
   int  level;                            // 活结点在子集树中所处的层序号
};
```

6.3 装载问题

AddLiveNode:将新产生的节点加入优先队列中

```
template<class Type>
// 将活结点加入到表示活结点优先队列的最大堆H中
void AddLiveNode(MaxHeap<HeapNode<Type>>&H, bbnode *E, Type wt, bool ch, int lev) {
   bbnode  *b = new bbnode;
   b->parent = E;
   b->LChild = ch;
   HeapNode<Type> N;
   N.uweight = wt;
   N.level = lev;
   N.ptr = b;
   H.Insert(N);
}
```

6.3装载问题

MaxLoading:搜索

```
// 搜索子集空间树
while (i != n+1) {                                    // 非叶结点
   // 检查当前扩展结点的儿子结点
   if (Ew+w[i] <= c)                                  // 左儿子结点为可行结点
      AddLiveNode(H, E, Ew+w[i]+r[i], true, i+1);
   AddLiveNode(H, E, Ew+r[i], false, i+1);            // 右儿子结点
   HeapNode<Type> N;                                  // 取下一扩展结点
   H.DeleteMax(N);                                    // 非空
   i = N.level;
   E = N.ptr;
   Ew = N.uweight-r[i-1];
}
```