算法分析与设计

分枝限界法

主要内容

- 分枝限界法的基本思想
- 分枝限界法的求解实例

分枝限界法和回溯法

- 分枝限界法类似于回溯法，是在问题的解空间树上搜索问题解的算法

分枝限界法和回溯法

- 不同点
  - 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，而分枝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，而分枝限界法以广度优先或以最小耗费优先的方式搜索解空间树

分枝限界法的基本思想

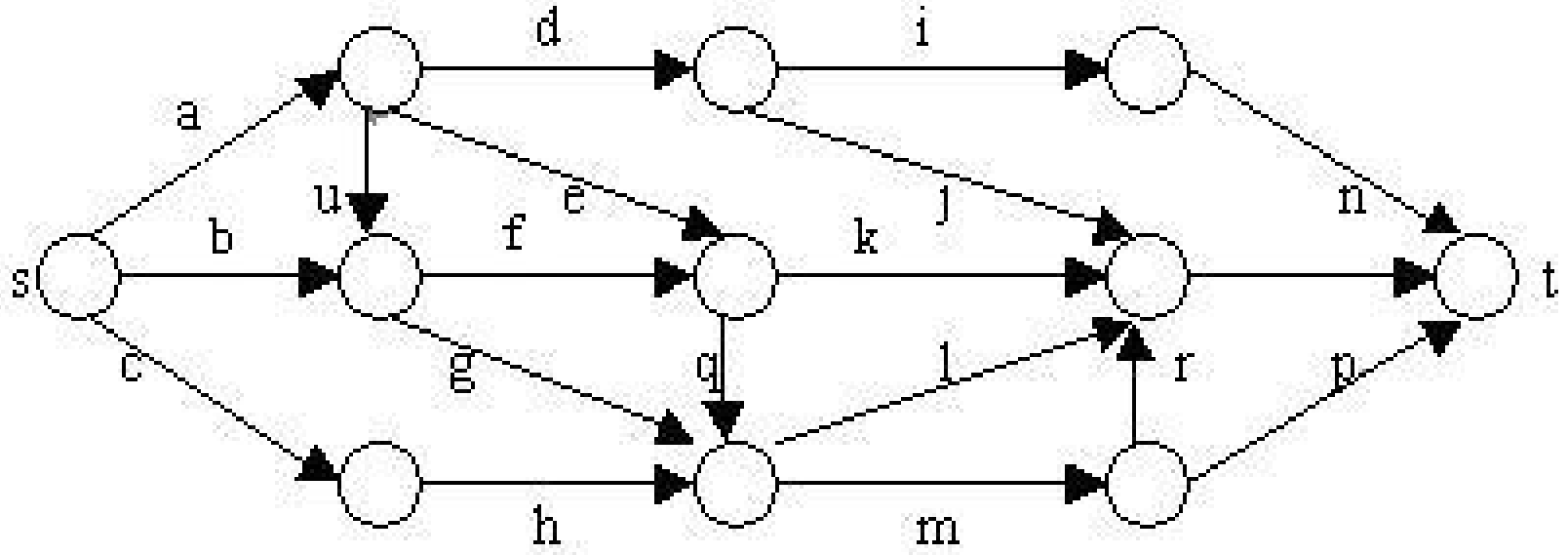
- 分枝限界法常以广度优先或以最小耗费（最大效益）优先的方式搜索问题的解空间树
  - 在分枝限界法中，每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点。在这些儿子结点中，导致不可行解或导致非最优解的儿子结点被舍弃，其余儿子结点被加入活结点表中
  - 此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。这个过程一直持续到找到所需的解或活结点表为空时为止

常见的两种分枝限界法

- 队列式(FIFO)分枝限界法
  - 按照队列先进先出（FIFO）原则选取下一个节点为扩展节点
- 优先队列式分枝限界法
  - 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
  - 应用优先队列式分枝限界法求解具体问题时，应该根据具体问题的特点确定选用最大优先队列或者最小优先队列表示解空间的活结点表

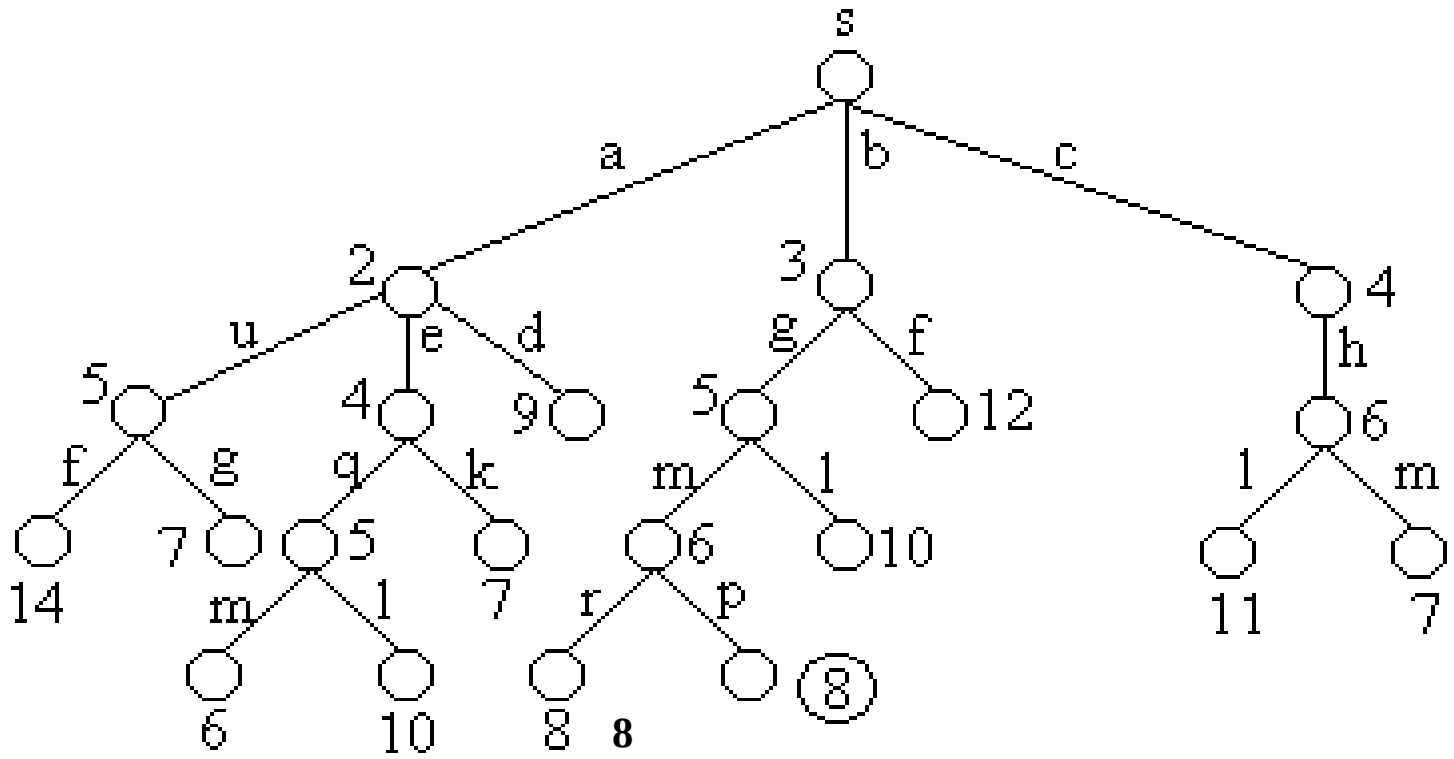
单源最短路径问题

- 问题描述
  - 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枝限界法解有向图G的单源最短路径问题产生的解空间树
  - 每一个结点旁的数字表示该结点所对应的当前路长

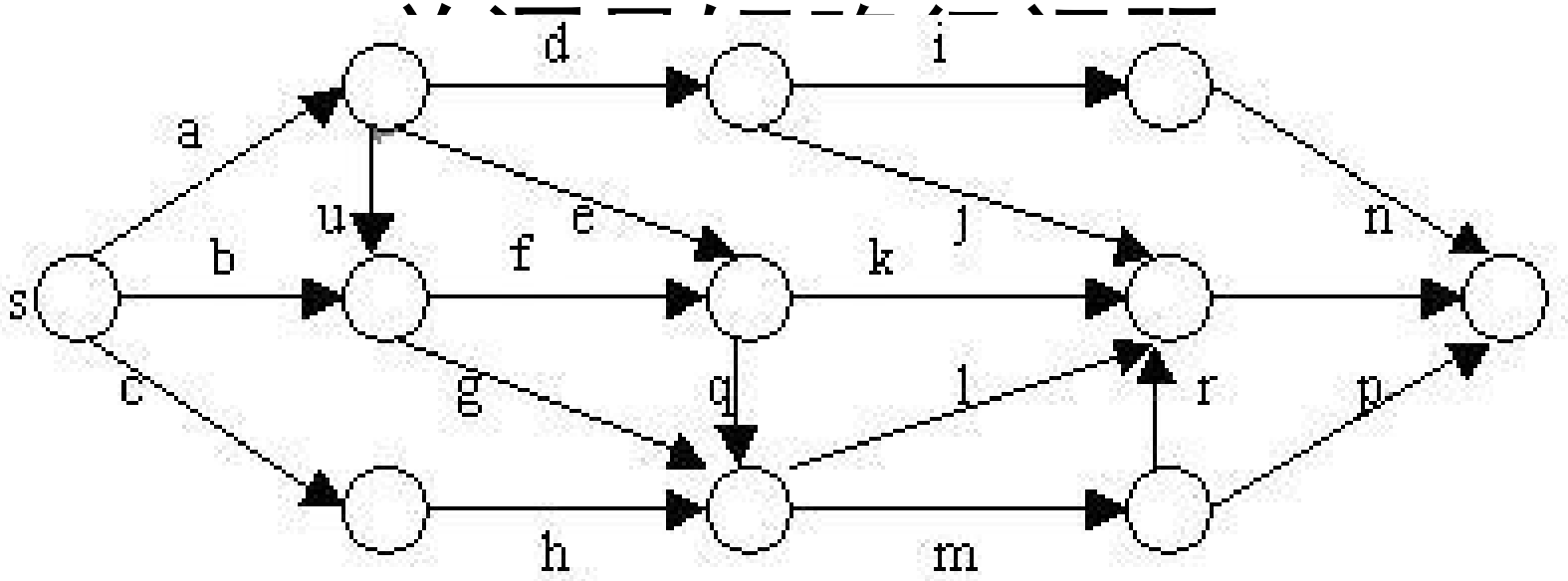


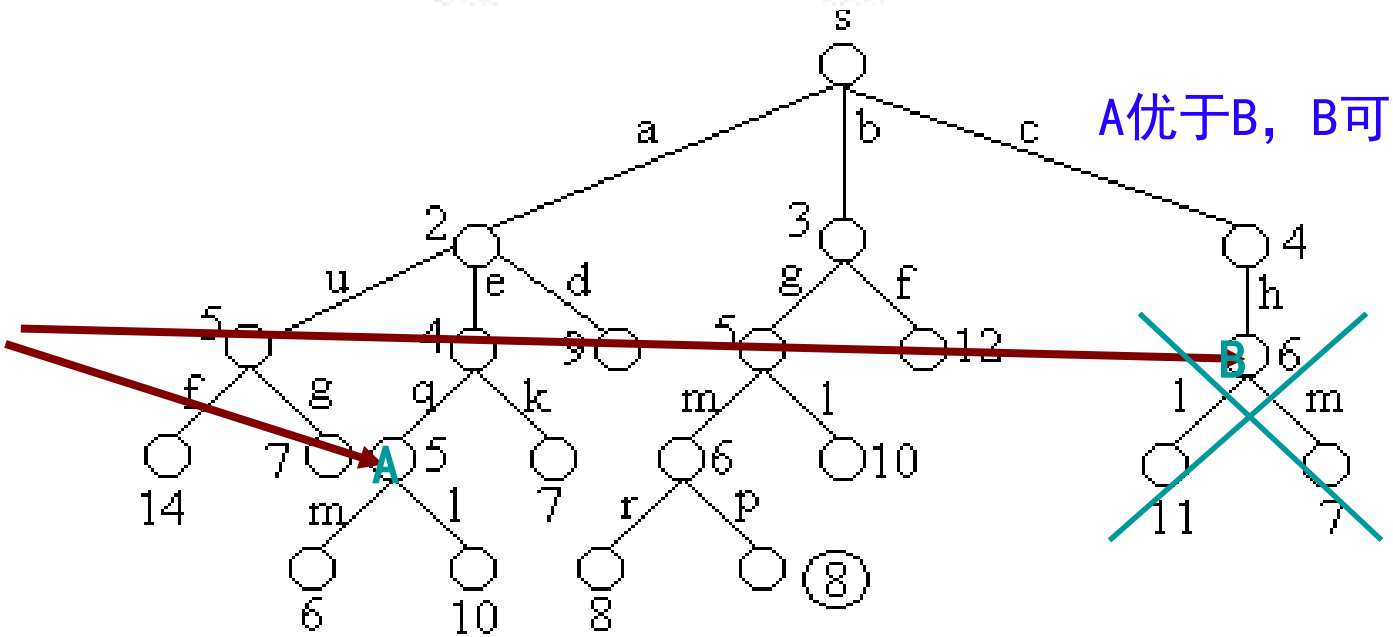
单源最短路径问题

- 算法思想
  - 解单源最短路径问题的优先队列式分枝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
  - 结点的扩展过程一直继续到活结点优先队列为空时为止

单源最短路径问题

- 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





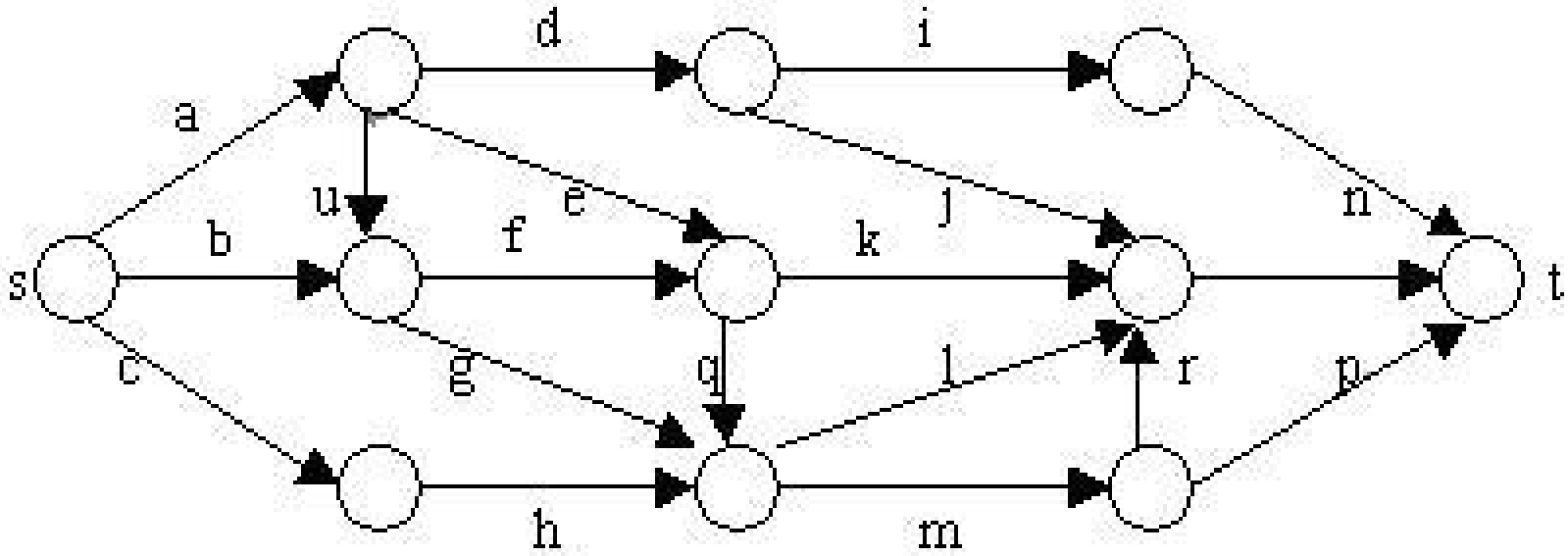
A优于B，B可剪枝

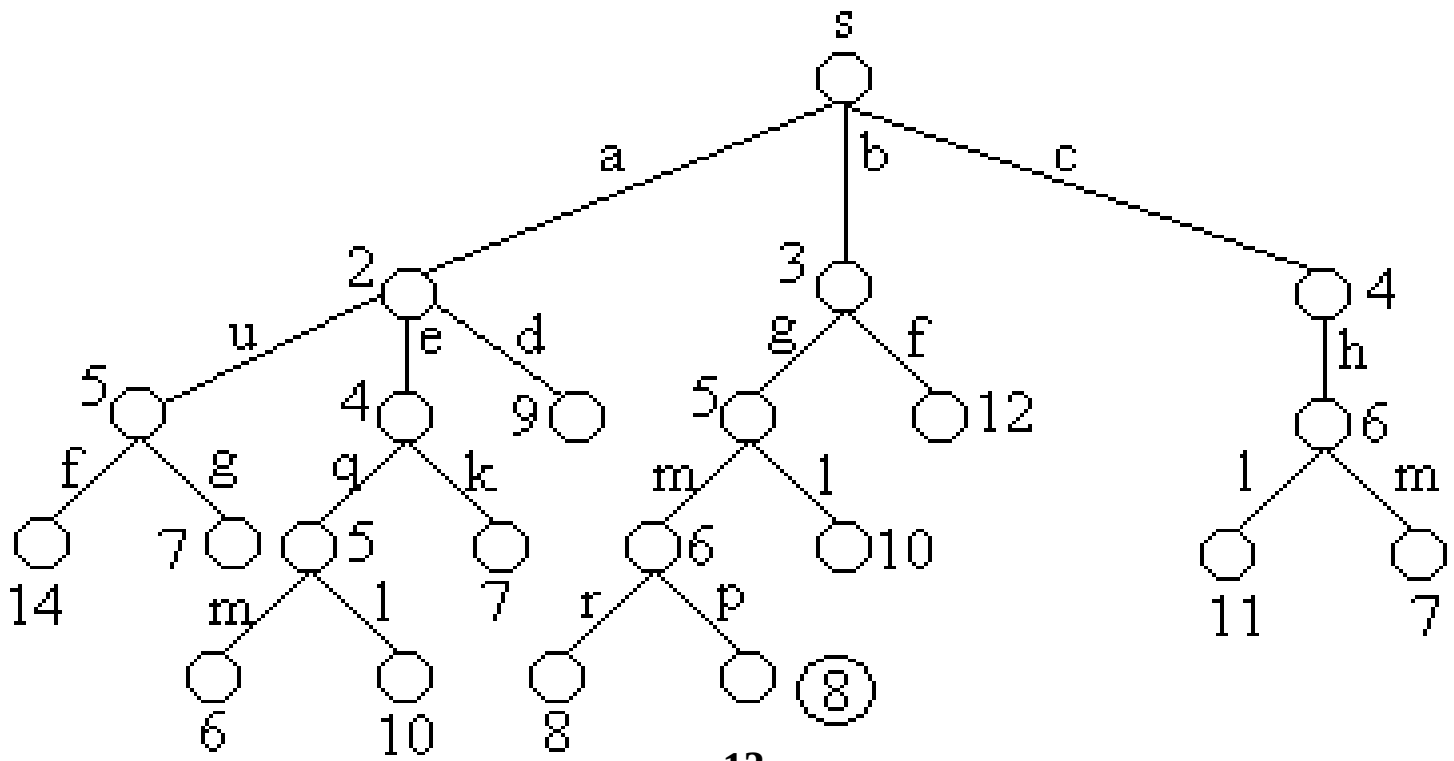
经过不同的路径到达相同的顶点

单源最短路径问题

```
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源点到*j*的路径长


d
i
a
u
e
j
n
b
f
k
s
t
c
g
q
l
r
p
h
m


s
a
b
c
2
3
4
u
e
d
g
f
h
5
4
9
5
12
6
f
g
q
k
m
l
l
m
14
7
5
7
6
10
11
7
m
l
r
p
6
10
8
8

单源最短路径问题

- Dijakstra算法和分枝限法在解决该问题的异同
  - Dijakstra算法：每一步的选择为当前步的最优
  - 队列式分枝限界法的搜索解空间树的方式类似于解空间树的宽度优先搜索，不同的是队列式分枝限界法不搜索以不可行结点（已经被判定不能导致可行解或不能导致最优解的结点）为根的子树。按照规则，这样的结点不被列入活结点表

单源最短路径问题

- Dijakstra算法和分枝限法在解决该问题的异同
  - 优先队列式分枝限界法的搜索方式是根据活结点的优先级确定下一个扩展结点。结点的优先级常用一个与该结点有关的数值p来表示。最大优先队列规定p值较大的结点点的优先级较高。在算法实现时通常用一个最大堆来实现最大优先队列，体现最大效益优先的原则。类似地，最小优先队列规定p值较小的结点的优先级较高。在算法实现时，常用一个最小堆来实现，体现最小优先的原则。采用优先队列式分枝定界算法解决具体问题时，应根据问题的特点选用最大优先或最小优先队列，确定各个结点点的p值

装载问题

- 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些集装箱装上这2艘轮船。如果有，找出一种装载方案

装载问题

- 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。
  - 首先将第一艘轮船尽可能装满
  - 将剩余的集装箱装上第二艘轮船

装载问题

- 队列式分枝限界法
  - 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

装载问题

- 队列式分枝限界法
  - 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

装载问题

- 队列式分枝限界法

```
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
}
```

装载问题

- 算法的改进
  - 结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，因为此时若要船装最多集装箱，就应该把此箱装上船
  - 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

装载问题

●算法的改进

提前更新bestw

```
// 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);
}
```

右儿子剪枝

```
// 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
```

装载问题

- 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
```

装载问题

- 构造最优解
  - 找到最优值后，可以根据parent回溯到根结点，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;
}
```

装载问题

- 优先队列式分枝限界法
  - 解装载问题的优先队列式分枝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
  - 在优先队列式分枝限界法中，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

0-1背包问题

- 问题描述
  - 给定n种物品和一个背包。物品i的重量是Wi，其价值为Vi，背包的容量为C。应如何选择装入背包的物品，使得装入背包中物品的总价值最大?
  - 限制：在选择装入背包的物品时，对每种物品i只有2种选择，即装入背包或不装入背包。不能将物品i装入背包多次，也不能只装入部分的物品i

0-1背包问题

- 队列式分枝定界法
  - 用队列存储活动结点表
  - 解空间为子集树

0-1背包问题

- 优先队列式分枝定界法的一种方案
  - 首先，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；节点的优先级为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

0-1背包问题

- 上界函数

```
// n表示物品总数，cleft为剩余空间
while (i <= n && w[i] <= cleft)
{
        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);

    //  取下一个扩展结点（略）}
```

分支限界搜索过程

旅行售货员问题

- 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）最短（或最小）。

旅行售货员问题

- 解空间树：排列树

- 队列式分枝定界法
- 优先队列式分枝定界法