# 优先队列与分支限界法

（根据王晓东《计算机算法设计与分析》（第 2 版）第 6 章改写，资料内容较多，只讲一少部分，其余供自学和讨论）

目录：



## 1．基本思想：队列式(FIFO)分支限界法与优先队列式分支限界法

分支限界法与回溯法

（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。

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

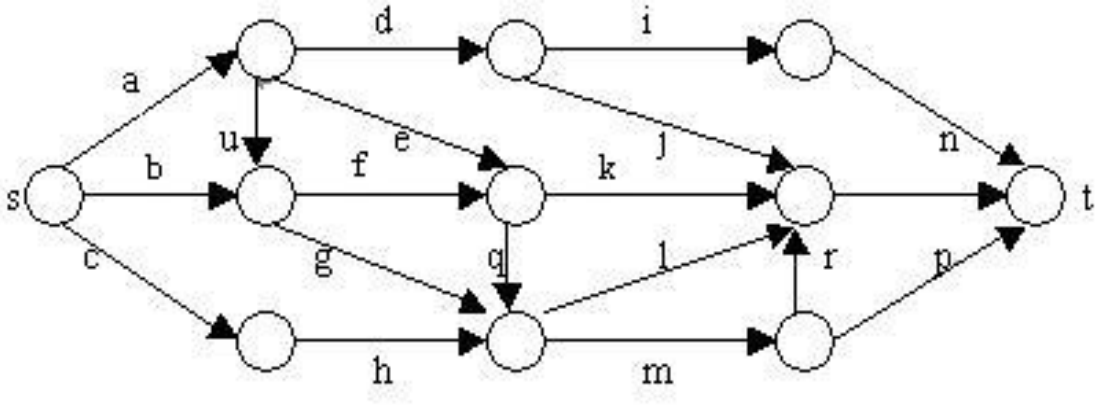
（2）优先队列式分支限界法

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

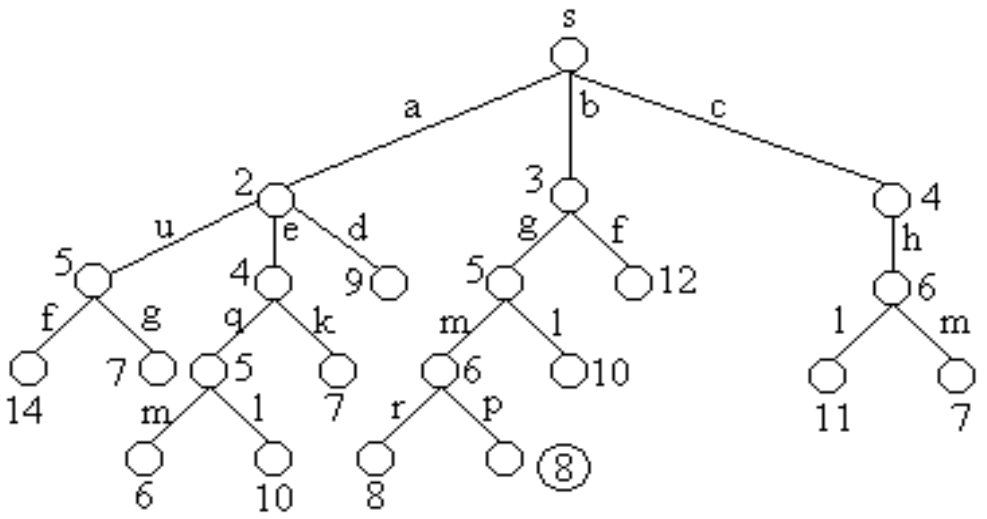
## 2．单源最短路径问题

2.1 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。




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。




2.2 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。

2.3. 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。

```
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
 }
}
```

## 3. 装载问题

3.1 问题描述

有一批共个集装箱要装上 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。

3.2 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。

```
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
       }
  }
```

3.3 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，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。

```
// 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);
// 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
}
```

3.4 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
```

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;
}
```

3.5 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。

## 4．布线问题

4.1 算法思想

解此问题的队列式分支限界法从起始位置 a 开始将它作为第一个扩展结点。与该扩展结点相邻并且可达的方格成为可行结点被加入到活结点队列中，并且将这些方格标记为 1，即从起始方格 a 到这些方格的距离为 1。

接着，算法从活结点队列中取出队首结点作为下一个扩展结点，并将与当前扩展结点相邻且未标记过的方格标记为 2，并存入活结点队列。这个过程一直继续到算法搜索到目标方格 b 或活结点队列为空时为止。即加入剪枝的广度优先搜索。

```
Position offset[4];
  offset[0].row = 0; offset[0].col = 1; // 右
  offset[1].row = 1; offset[1].col = 0; // 下
  offset[2].row = 0; offset[2].col = -1; // 左
  offset[3].row = -1; offset[3].col = 0; // 上
for (int i = 0; i <= m+1; i++)
    grid[0][i] = grid[n+1][i] = 1; // 顶部和底部
  for (int i = 0; i <= n+1; i++)
    grid[i][0] = grid[i][m+1] = 1; // 左翼和右翼
for (int i = 0; i < NumOfNbrs; i++)
  { nbr.row = here.row + offset[i].row;
   nbr.col = here.col + offset[i].col;
    if (grid[nbr.row][nbr.col] == 0)
     {    // 该方格未标记
       grid[nbr.row][nbr.col] = grid[here.row][here.col] + 1;
       if ((nbr.row == finish.row) && (nbr.col == finish.col)) break; //
完成布线
         Q.Add(nbr);}
```

```
    }
```

找到目标位置后，可以通过回溯方法找到这条最短路径

## 5．0-1 背包问题

5.1 算法的思想

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。

5.2 上界函数

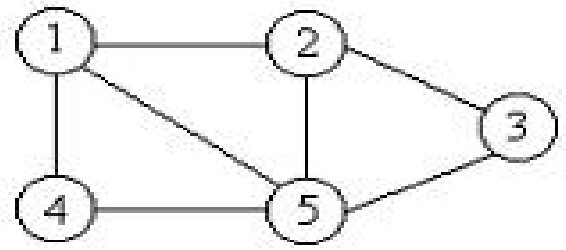
```
while (i <= n && w[i] <= cleft)       // n 表示物品总数，cleft 为剩余空间
    { cleft -= w[i];                          //w[i]表示 i 所占空间
      b += p[i];                              //p[i]表示 i 的价值
      i++;
     }
if (i <= n) b += p[i] / w[i] * cleft;    // 装填剩余容量装满背包
return b;                             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

## 6．最大团问题

6.1　问题描述

给定无向图 G=(V，E)。如果 U⊆V，且对任意 u，v∈U 有(u，v)∈E，则称 U 是 G 的完全子图。G 的完全子图 U 是 G 的团当且仅当 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。

下图 G 中，子集{1，2}是 G 的大小为 2 的完全子图。这个完全子图不是团，因为它被 G 的更大的完全子图{1，2，5}包含。{1，2，5}是 G 的最大团。{1，4，5}和{2，3，5}也是 G 的最大团。




6.2 上界函数

用变量 cliqueSize 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 cliqueSize +n-level+1 作为顶点数上界 upperSize 的值。

在此优先队列式分支限界法中，upperSize 实际上也是优先队列中元素的优先级。算法总是从活结点优先队列中抽取具有最大 upperSize 值的元素作为下一个扩展元素。

6.3　算法思想

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 cliqueSize 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 upperSize>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。

算法的 while 循环的终止条件是遇到子集树中的一个叶结点(即 n+1 层结点)成为当前扩展结点。

对于子集树中的叶结点，有 upperSize＝cliqueSize。此时活结点优先队列中剩余结点的 upperSize 值均不超过当前扩展结点的 upperSize 值，从而进一步搜索不可能得到更大的团，此时算法已找到一个最优解。

## 7．旅行售货员问题

7.1．问题描述

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可以组织成一棵树，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。

7.2 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：

(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。

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 v 给出。

## 8．电路板排列问题

算法开始时，将排列树的根结点置为当前扩展结点。在 do-while 循环体内算法依次从活结点优先队列中取出具有最小 cd 值的结点作为当前扩展结点，并加以扩展。

首先考虑 s=n-1 的情形，当前扩展结点是排列树中的一个叶结点的父结点。x 表示相应于该叶结点的电路板排列。计算出与 x 相应的密度并在必要时更新当前最优值和相应的当前最优解。

当 s<n-1 时，算法依次产生当前扩展结点的所有儿子结点。对于当前扩展结点的每一个儿子结点 node，计算出其相应的密度 node.cd。当 node.cd<bestd 时，将该儿子结点 N 插入到活结点优先队列中。

```
do {// 结点扩展
    if (E.s == n - 1)
     {// 仅一个儿子结点
```

```
int ld = 0; // 最后一块电路板的密度
for (int j = 1; j <= m; j++)
   ld += B[E.x[n]][j];
if (ld < bestd)
  {// 密度更小的电路板排列
   delete [] bestx;
   bestx = E.x;
   bestd = max(ld, E.cd);
   }
 else
{// 产生当前扩展结点的所有儿子结点
 for (int i = E.s + 1; i <= n; i++) {
BoardNode N;
N.now = new int [m+1];
for (int j = 1; j <= m; j++)
   // 新插入的电路板
   N.now[j] = E.now[j] + B[E.x[i]][j];
int ld = 0; // 新插入电路板的密度
for (int j = 1; j <= m; j++)
   if (N.now[j] > 0 && total[j] != N.now[j]) ld++;
N.cd = max(ld, E.cd);
if (N.cd < bestd) {// 可能产生更好的叶结点
   N.x = new int [n+1]; N.s = E.s + 1;
   for (int j = 1; j <= n; j++) N.x[j] = E.x[j];
   N.x[N.s] = E.x[i]; N.x[i] = E.x[N.s]; H.Insert(N);}
else delete [] N.now;}
delete [] E.x;}
```

## 9．批处理作业调度问题

9.1 问题的描述

给定 n 个作业的集合 J={J1,J2,…,Jn}。每一个作业 Ji 都有 2 项任务要分别在 2 台机器上完成。每一个作业必须先由机器 1 处理，然后再由机器 2 处理。作业 Ji 需要机器 j 的处理时间为 tji，i=1,2,…,n；j=1,2。对于一个确定的作业调度，设是 Fji 是作业 i 在机器 j 上完成处理的时间。则所有作业在机器 2 上完成处理的时间和

$$f=\sum_{i=1}^{n}F_{2i}$$

称为该作业调度的完成时间和。批处理作业调度问题要求对于给定的 n 个作业，制定最佳作业调度

方案，使其完成时间和达到最小

9.2 限界函数

在结点 E 处相应子树中叶结点完成时间和的下界是

$$f \geq \sum_{i \in M} F_{2i} + \max\{S_1, S_2\}$$

注意到如果选择 Pk，使 t1pk 在 k>=r+1 时依非减序排列，S1 则取得极小值。同理如果选择 Pk 使 t2pk 依非减序排列，则 S2 取得极小值

$$f \geq \sum_{i \in M} F_{2i} + \max\{\hat{S}_1, \hat{S}_2\}$$

9.3 算法描述

算法的 while 循环完成对排列树内部结点的有序扩展。在 while 循环体内算法依次从活结点优先队列中取出具有最小 bb 值（完成时间和下界）的结点作为当前扩展结点，并加以扩展。

首先考虑 E.s=n 的情形，当前扩展结点 E 是排列树中的叶结点。E.sf2 是相应于该叶结点的完成时间和。当 E.sf2 < bestc 时更新当前最优值 bestc 和相应的当前最优解 bestx。

当 E.s<n 时，算法依次产生当前扩展结点 E 的所有儿子结点。对于当前扩展结点的每一个儿子结点 node，计算出其相应的完成时间和的下界 bb。当 bb < bestc 时，将该儿子结点插入到活结点优先队列中。而当 bb≥ bestc 时，可将结点 node 舍去。

```
while (E.s <= n )
  { if (E.s == n ) {// 叶结点
       if (E.sf2 < bestc) {
         bestc = E.sf2;
         for (int i = 0; i < n; i++)
           bestx[i] = E.x[i];}
       delete [] E.x;}
     else {// 产生当前扩展结点的儿子结点
       for (int i = E.s; i < n; i++) {
            Swap(E.x[E.s],E.x[i]);
            int f1,f2;
            int bb=Bound(E,f1,f2,y);
            if (bb < bestc ) {
               MinHeapNode N;
               N.NewNode(E,f1,f2,bb,n);
               H.Insert(N);}
            Swap(E.x[E.s],E.x[i]);
            }
       delete [] E.x;
     } // 完成结点扩展
```

**（ 李学武 ， 2007-1-15 修改 ）**