西北大学

算法设计与分析

第六章 分支限界法

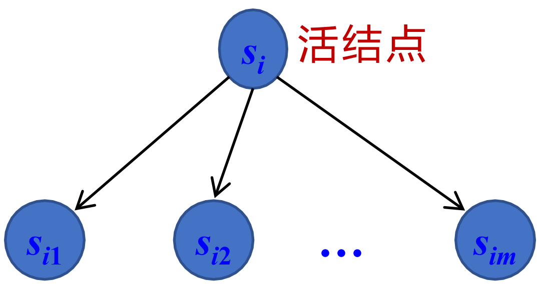
本 章 内 容

- 分支限界法概述
- 分支限界法求解0/1背包问题
- 分支限界法求解图的单源最短路径
- 分支限界法小结

6 分支限界法概述①定义②设计思想③算法性能

➤分支限界法类似于回溯法，也是一种在问题的解空间树上搜索问题解的算法。

➤分支限界法对解空间的搜索采用广度优先搜索策略，回溯法采用深度优先搜索策略。



扩展产生所有的子结点

Q：求最优解时，如何选择子结点进行再扩展？

A：设计一个**限界函数**，计算限界函数值，选择一个最有利的子结点作为扩展结点，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。

➤分支限界法与回溯法的求解目标不同。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**。

6

分支限界法概述①定义②设计思想③算法性能

方法	解空间搜索方式	存储结点的数据结构	结点存储特性	常用应用
回溯法	深度优先	栈	活结点的所有可行子结点被遍历后才从栈中出栈	找出满足条件的所有解
分枝限界法	广度优先	队列，优先队列（堆（大根or小根））	每个结点只有一次成为活结点的机会	找出满足条件一个解或者特定意义的最优解

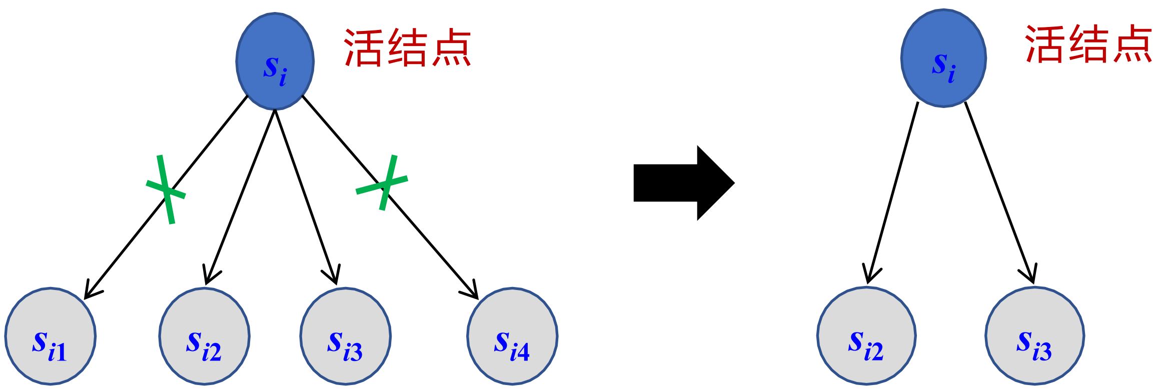
分支限界法的设计思想

- 1.设计合适的限界函数
- 2. 组织活结点表
- 3.确定最优解的解向量X

分支限界法概述①定义②设计思想③算法性能

1.设计合适的限界函数

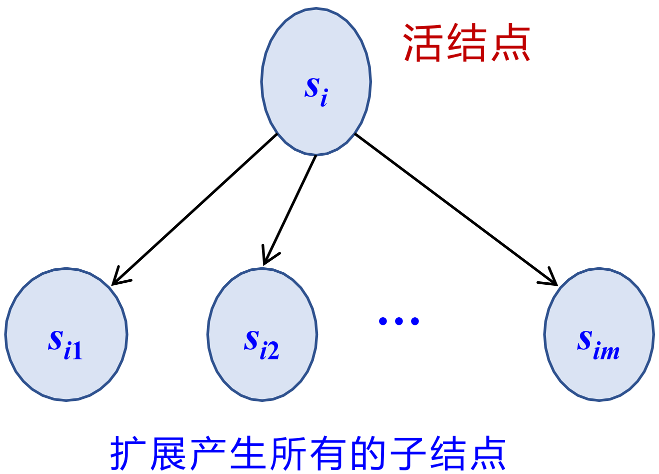
在搜索解空间树时，每个活结点可能有很多孩子结点，其中有些孩子结点搜索下去是可能产生问题解或最优解。通过设计好的限界函数在扩展时删除不必要的孩子结点，从而提高搜索效率。



通过高效的限界函数进行剪枝，使得从s_i出发的搜索效率提高一倍。

2.组织活结点表

分支限界法是对解空间树进行广度优先搜索，需要队列将待扩展活结点存储组织。



- 普通队列式分支限界法
- 优先队列式分支限界法

2.组织活结点表

普通队列式分支限界法

队列式分支限界法将活结点表组织成一个队列，并按照队列先进先出（FIFO）原则选取下一个结点为扩展结点。步骤如下：

①将根结点加入活结点队列。

②从活结点队中取出队头结点，作为当前扩展结点。

③对当前扩展结点，先从左到右地产生它的所有孩子结点，用约束条件检查，把所有满足约束条件的孩子结点加入活结点队列。

④重复步骤②和③，直到找到一个解或活结点队列为空为止。

2.组织活结点表

优先队列式分支限界法

优先队列式分支限界法的主要特点是将活结点表组成一个优先队列，并选取优先级最高的活结点成为当前扩展结点。步骤如下：

①计算起始结点（根结点）的优先级并加入优先队列（与特定问题相关的信息的函数值决定优先级）。

②从优先队列中取出优先级最高的结点(队头）作为当前扩展结点，使搜索朝着解空间树上可能有最优解的分枝推进，以便尽快地找出一个最优解。

③对当前扩展结点，先从左到右地产生它的所有孩子结点，然后用约束条件检查，对所有满足约束条件的孩子结点计算优先级并加入优先队列（进队）。

④重复步骤②和③，直到找到一个解或优先队列为空为止。

2. 组织活结点表

优先队列利用堆来实现。堆可以看作一棵完全二叉树的顺序存储结构。在这棵完全二叉树中：

- 若树非空，且根结点的值均大于其左右孩子结点的值，则称之为大根堆；且其左、右子树分别也是大根堆。
- 若树非空，且根结点的值均小于其左、右孩子结点的值，则称之为小根堆；且其左、右子树分别也是小根堆。

优先队列的两个基本操作：

出队：堆顶出队，最后一个元素代替堆顶位置，重新调整成堆；

进队：新进队元素放在堆末尾之后，重新调整成堆；

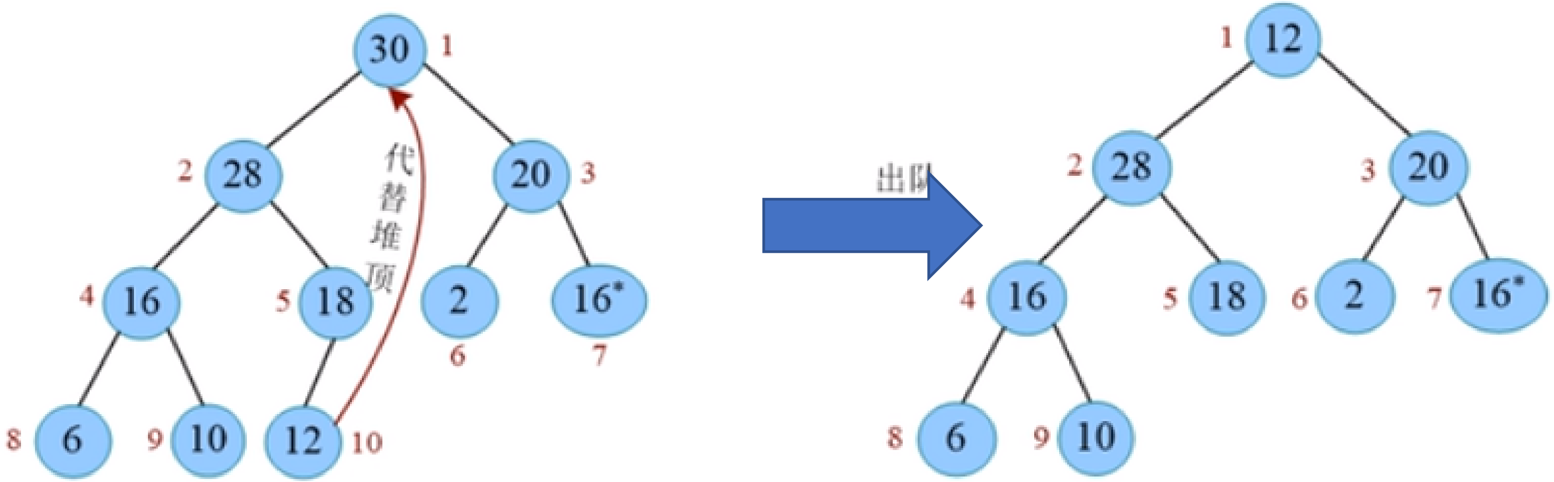
2.组织活结点表

优先队列出队操作（以大根堆为例）

- 出队：堆顶出队，最后一个元素代替堆顶位置。除了堆顶外，其他结点均满足大根堆定义，只需将堆顶执行“下沉”操作即可调整为堆。
- “下沉”：堆顶与其左、右孩子进行比较，若满足堆定义，则不需调整。若不满足，则与其值较大的孩子进行交换，交换后，继续向下调整，直到满足大根堆定义为止。

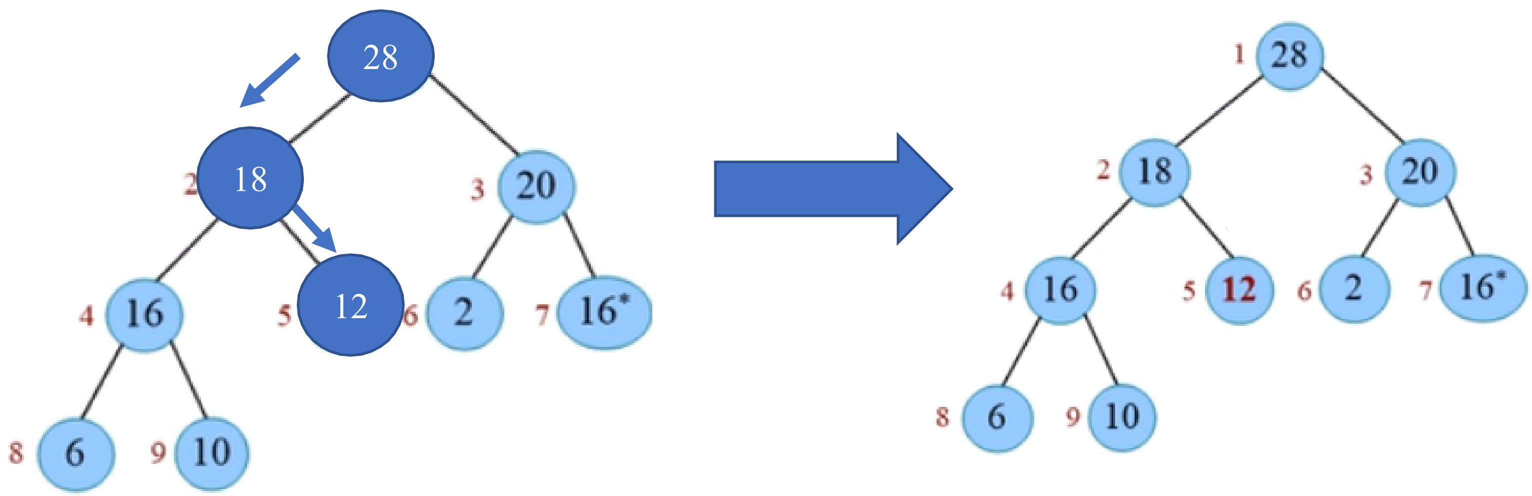
2.组织活结点表

优先队列出队操作示例：队头(堆顶）元素30出队,最后一个元素12代替堆顶，进行“下沉”调整。



2.组织活结点表

优先队列出队操作示例：队头(堆顶）元素30出队，最后一个元素12代替堆顶，进行“下沉”调整。



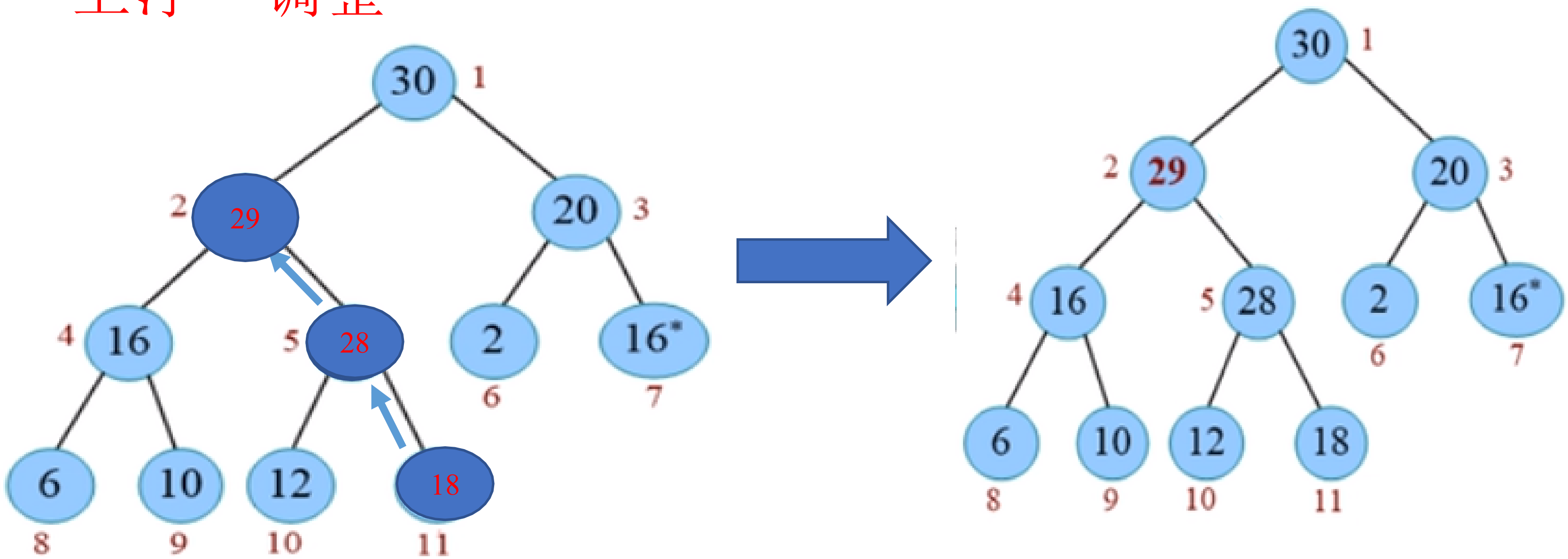
2.组织活结点表

优先队列进队操作（以大根堆为例）

- 进队：进队操作后，除了新进队的元素外，其他结点都满足大根堆的定义，需要将新元素执行“上浮”操作，调整成堆。
- “上浮”：新进队元素与其双亲比较，如果值小于其双亲，则满足堆定义，无需调整。如果其值比双亲大，则与双亲交换，交换到新位置后，继续向上比较、调整，直到满足大根堆定义为止。

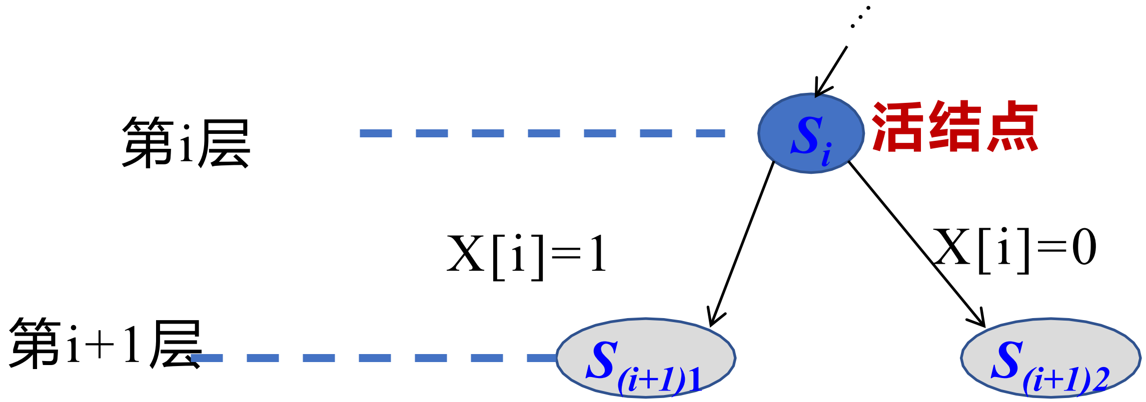
2.组织活结点表

优先队列进队操作示例：新元素29进队后，将其放在最后;进行“上浮”调整



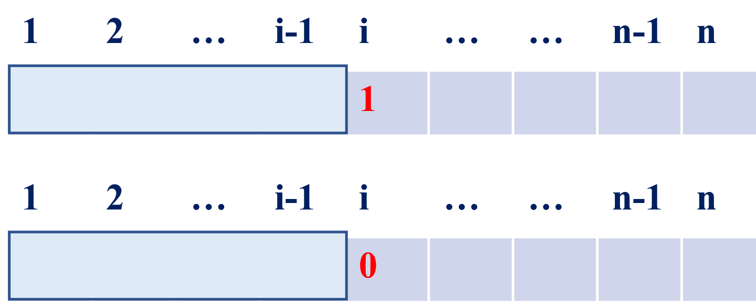
3.确定最优解的解向量X

以子集树为例



解空间树中状态$S_{(i+1)1}$对应的解向量X：

解空间树中状态$S_{(i+1)2}$对应的解向量X：



这两个子结点均要进入队列保存，此时每个结点对应的状态即X如何存储？

3.确定最优解的解向量X

两种方法：

①对每个扩展结点保存从根结点到该结点的“路径” 。

➢每个结点带有一个可能的解向量。这种做法比较浪费空间，但实现起来简单。（0-1背包问题为例）

②在搜索过程中构建搜索经过的树结构。

➢每个结点记录其双亲的信息，当找到最优解时，通过双亲信息找到对应的最优解向量。（单源最短路径为例）

6 分支限界法概述①定义②设计思想③算法性能

一般情况下，在问题的解向量$X=(x_1, x_2, \cdots, x_n)$中，分量$x_i$（$1 \leqslant i \leqslant n$）的取值范围为某个有限集合$S_i=(s_{i1}, s_{i2}, \cdots, s_{ir})$。

问题的解空间由笛卡尔积$S_1 \times S_2 \times \cdots \times S_n$构成：

- 第1层根结点有$|S_1|$棵子树
- 第2层有$|S_1|$个结点，第2层的每个结点有$|S_2|$棵子树，第3层有$|S_1| \times |S_2|$个结点…第$n+1$层有$|S_1| \times |S_2| \times \ldots \times |S_n|$个结点，它们都是叶子结点，代表问题的所有可能解。以子集树为例，$|S_i|=2$，其时间性能：2^n。

在最坏情况下，时间复杂性是**指数阶。通过设计限界函数和约束进行剪枝，提高算法效率。**

分支限界法示例

0-1背包问题

6 分支限界法求解0/1背包问题

【问题描述】有n个重量分别为$\{w_1, w_2, \cdots, w_n\}$的物品，它们的价值分别为$\{v_1, v_2, \cdots, v_n\}$，给定一个容量为$W$的背包。

设计从这些物品中选取一部分物品放入该背包的方案，每个物品要么选中要么不选中，要求选中的物品不仅能够放到背包中，而且重量和不超过W具有**最大**的价值。

6 分支限界法求解0/1背包问题

假设一个0/1背包问题是，n=3，重量为w=（16，15，15），价值为v=（45，25，25），背包限重为W=30，

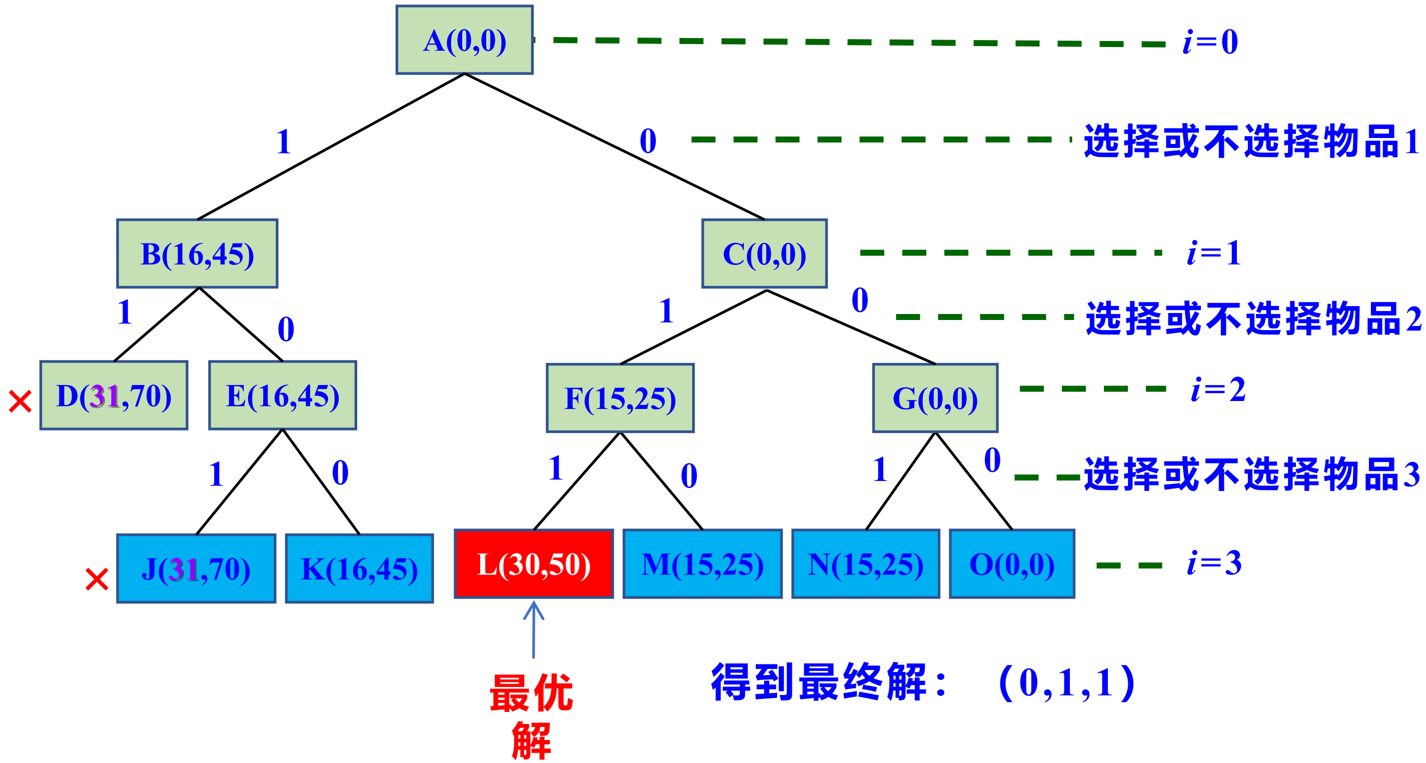
解向量为x=（x_1，x_2，x_3）。

编号	1	2	3
重量	16	15	15
价值	45	25	25

6 分支限界法求解0/1背包问题

编号	1	2	3
重量	16	15	15
价值	45	25	25

首先不考虑限界问题，用FIFO表示队列（实际上对应层次遍历）。初始时，队列为空。



得到最终解：（0,1,1）

6 分支限界法求解0/1背包问题

采用STL的queue<NodeType>容器qu作为队列，队列中的结点类型声明如下：

```
struct NodeType                //队列中的结点类型
{  int no;                     //结点编号，从1开始
   int i;                      //当前结点在搜索空间中的层次
   int w;                      //当前结点的总重量
   int v;                      //当前结点的总价值
   int x[MAXN];                //当前结点包含的解向量
   double ub;                  //上界
};
```

6 分支限界法求解0/1背包问题

设计限界函数

为了简便，设根结点为第0层，然后各层依次递增，显然i=n时表示是叶子结点层。由于该问题是求装入背包的最大价值，属求最大值问题，采用上界设计方式。

6 分支限界法求解0/1背包问题

设计限界函数

对于第i层的某个结点e，用e.w表示结点e时已装入的总重量，用e.v表示已装入的总价值。E.i=i即对应表示结点e的状态是在确定解向量的e.X[E.i]分量;计算当前结点e对应的状态所能达到的价值上界ub。

- 如果所有剩余的物品都能装入背包，
  那么价值的上界e.ub=e.v+ (v[i+1]+…+v[n])
- 如果所有剩余的物品不能全部装入背包，
  那么价值的上界e.ub=e.v+ (v[i+1]+…+v[k])+(物品k+1装入的部分重量)×物品k+1的单位重量价值。

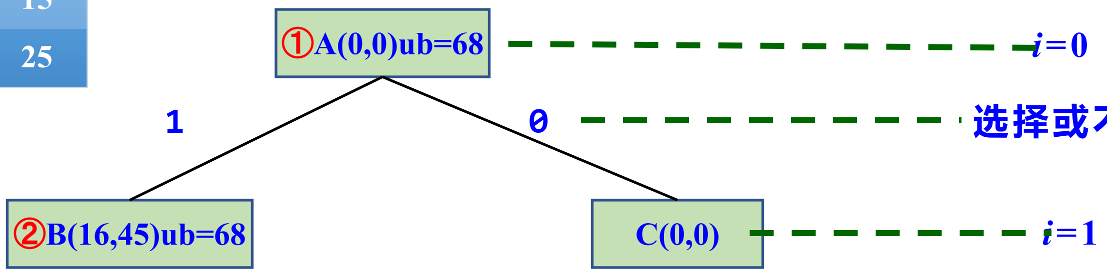
注：为了计算出来的上界更加紧凑，需要物品按单位重量价值递减有序。

6 分支限界法求解0/1背包问题——普通队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25

按单位重量价值递减存放

A



结点*A*出队，*A*的层次i=0，w=0，v=0,A.ub=A.v+剩余物品最大价值（从物品1~物品3）=0+45+ (30-16)×25/15 =68:

因为w+w[1]=16<30,所以不剪左支，生成左子B结点

B.w=A.w+w[1]=16; B.v=A.v+v[1]=45;B.i=A.i+1=1(B.i为B结点的层号。

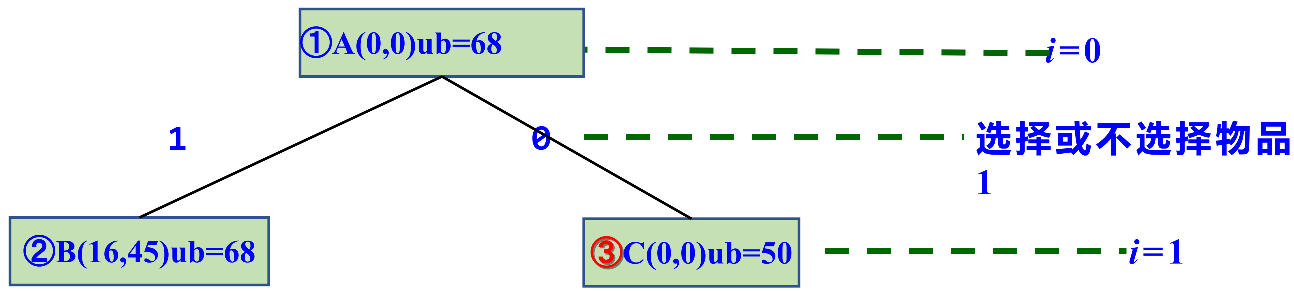
B.ub= B.v(45) + (30-16)×25/15 = 68（采用取整运算）

第2件开始之后的物品的最大价值。此时：可选物品2的一部分,即30-16，对应的价值

B

6 分支限界法求解0/1背包问题——普通队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



根结点A的层次i=0，w=0，v=0:看是否扩展右子。

右子C.i=A.i+1;C.w=A.w=0;C.v=A.v=0，计算C.ub

C.ub=C.v+ $\boxed{25+25}$ =50 因为C.ub>maxv(-9999)，所以C结点要扩展，放入活结点表（队列）

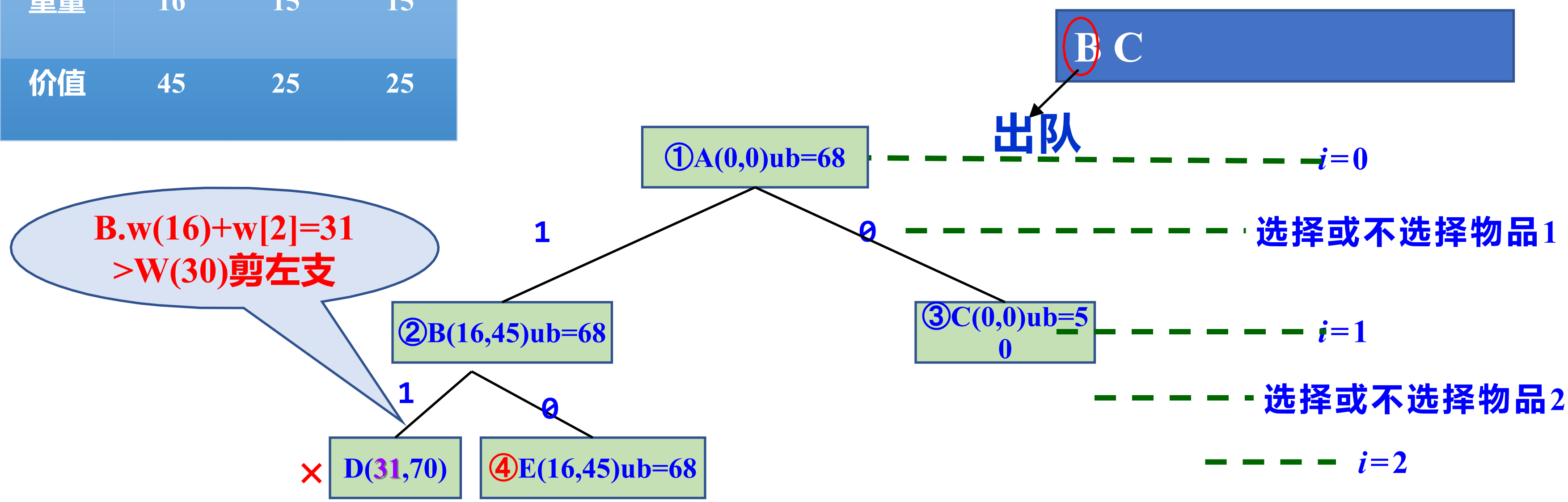
物品1未选，C.w=0

剩余物品即物品2，物品3。因为重量均为15，所以剩余物品全选的最大价值为25+25=50

B C

6 分支限界法求解0/1背包问题——普通队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



结点B的层次B.i=1，B.w=16，B.v=45:看是否扩展右子E

E.i=B.i+1=2; E.w=$B.w$; E.v=B.v

E.ub=45 + (30-16)×25/15 = 68（采用取整运算）用于限界.

E.ub(68)>maxv(-9999),E进队

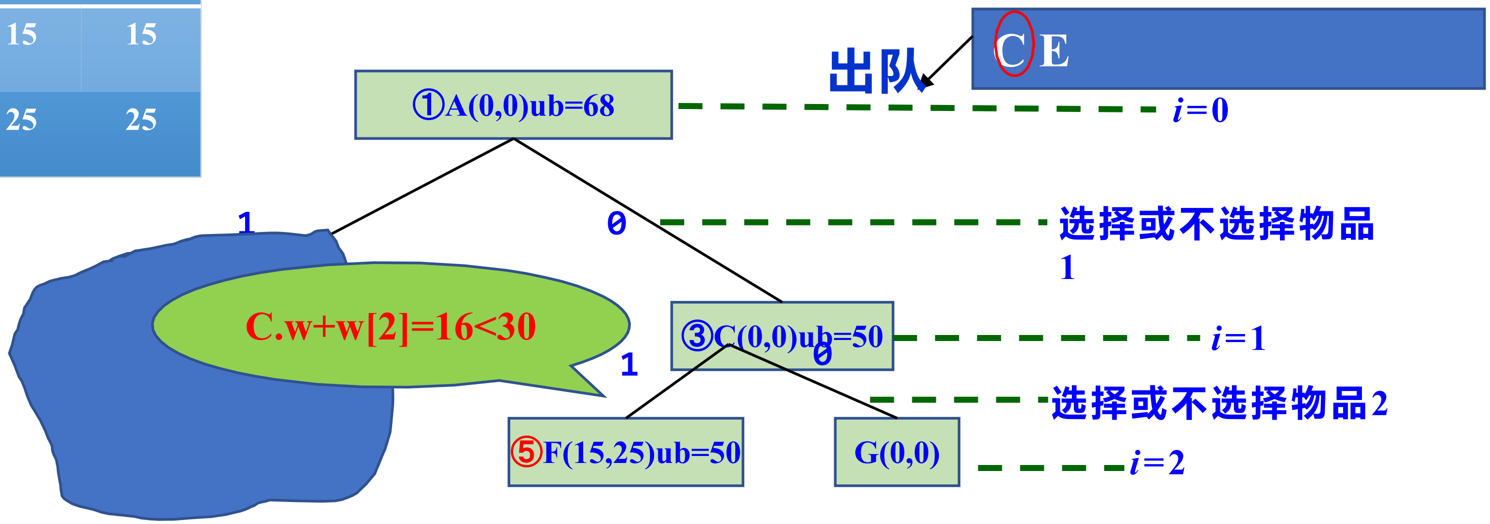
B.v

可选物品3的一部分，即30-16，对应的价值

C E

6 分支限界法求解0/1背包问题——普通队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



结点C的层次C.i=1，C.w=0，C.v=0:看是否扩展左子F

因为C.w+w[2]=15<30，所以扩展出左子F

F.i=C.i+1=2；F.w=C.w+w[2]=15；F.v=C.v+v[2]=25

F.ub=25 + 25 = 50（采用取整运算）用于限界.

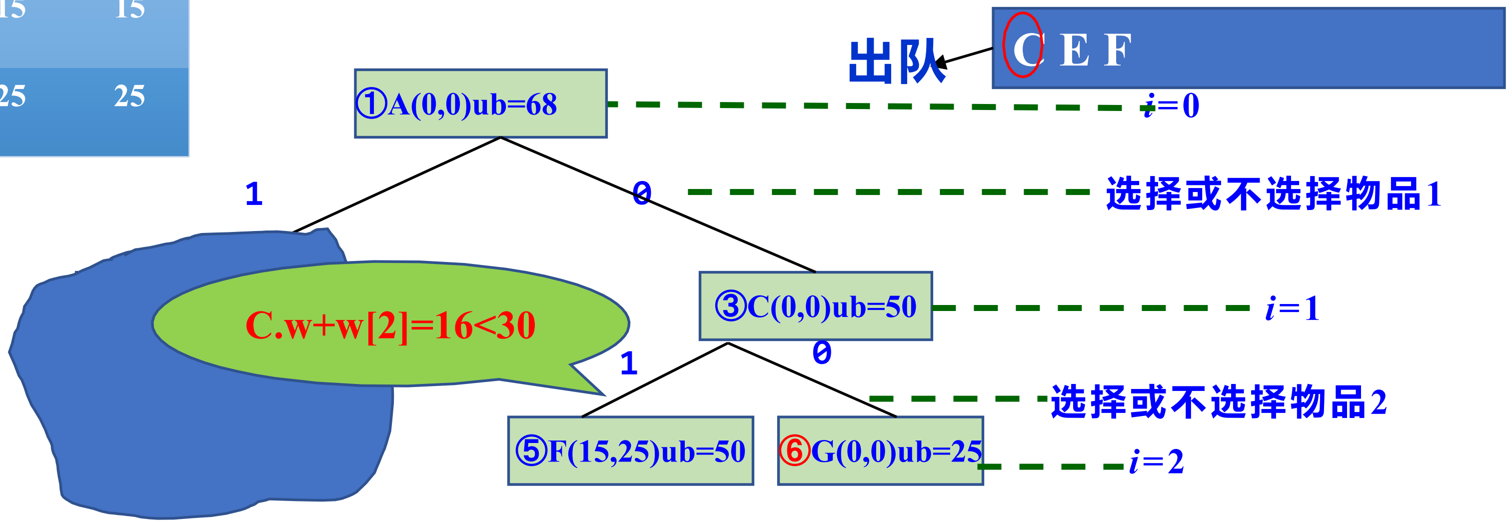
F.v

剩余物品最大价值（只剩物品3了，且物品3可以全部装入，价值为25）

E F

6 分支限界法求解0/1背包问题——普通队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



结点C的层次C.i=1，C.w=0，C.v=0:看是否扩展右子G

G.i=G.i+1=2；G.w=C.w=0；G.v=C.v=0

G.ub=0 + 25 = 25。用于限界.

G.v

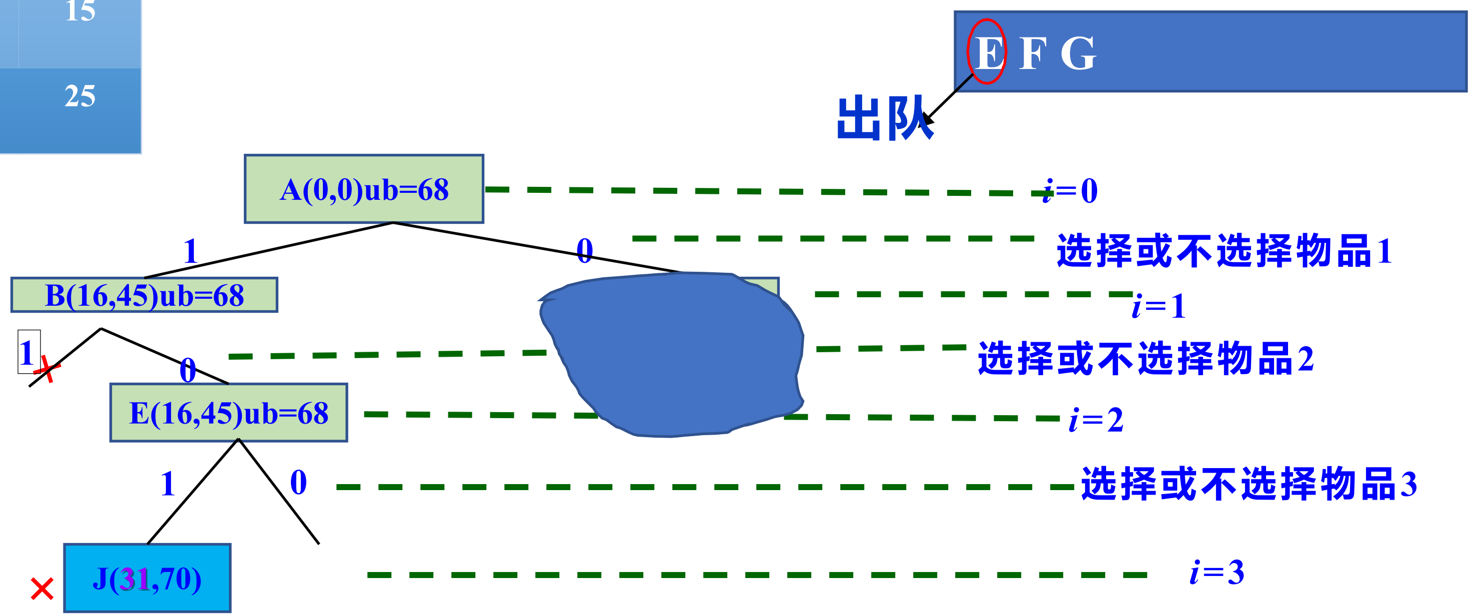
剩余物品最大价值（只剩物品3了，且物品3可以全部装入，价值为25）

G.ub(25)>maxv(-9999),G进队

E F G

6 分支限界法求解0/1背包问题——普通队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25

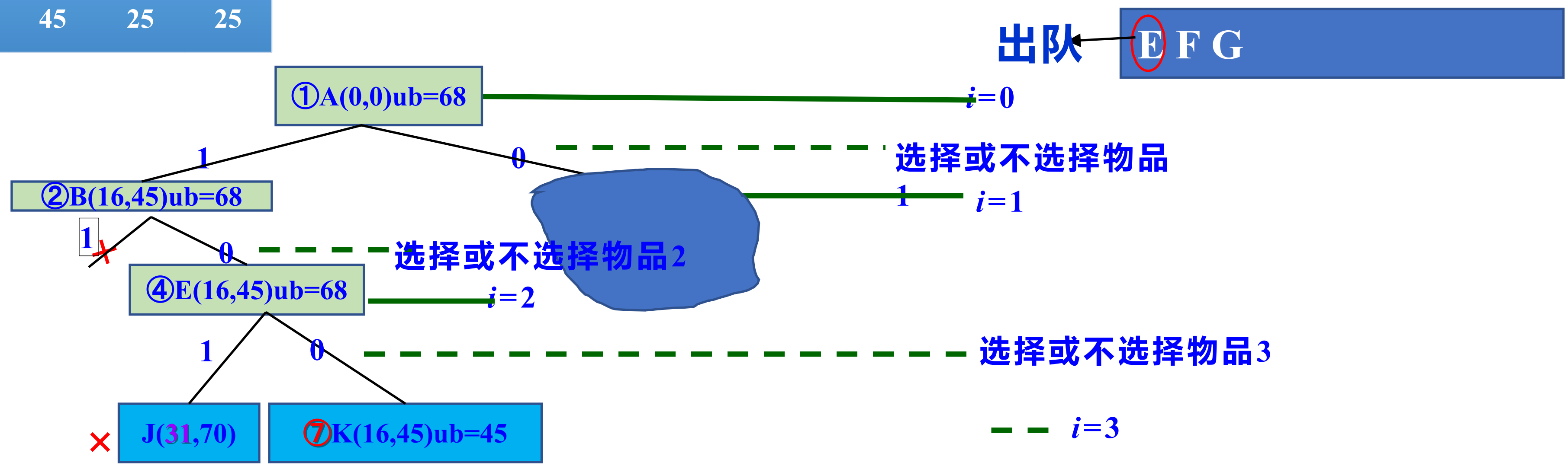


结点E的层次E.i=2，E.w=16，E.v=45:看是否扩展左子J

E.w+w[3]=16+15=31>30,所以不扩展J，剪掉E的左支。

6 分支限界法求解0/1背包问题——普通队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



结点E的层次E.*i*=2，E.*w*=16，E.*v*=45:看是否扩展右子K

K.i=E.i+1=3; K.*w*=*E.w=16;* K.*v*=E.*v=45*

一共3件物品，所以此值为0

K.ub=K.v+物品4~n的最大价值= 45。用于限界.

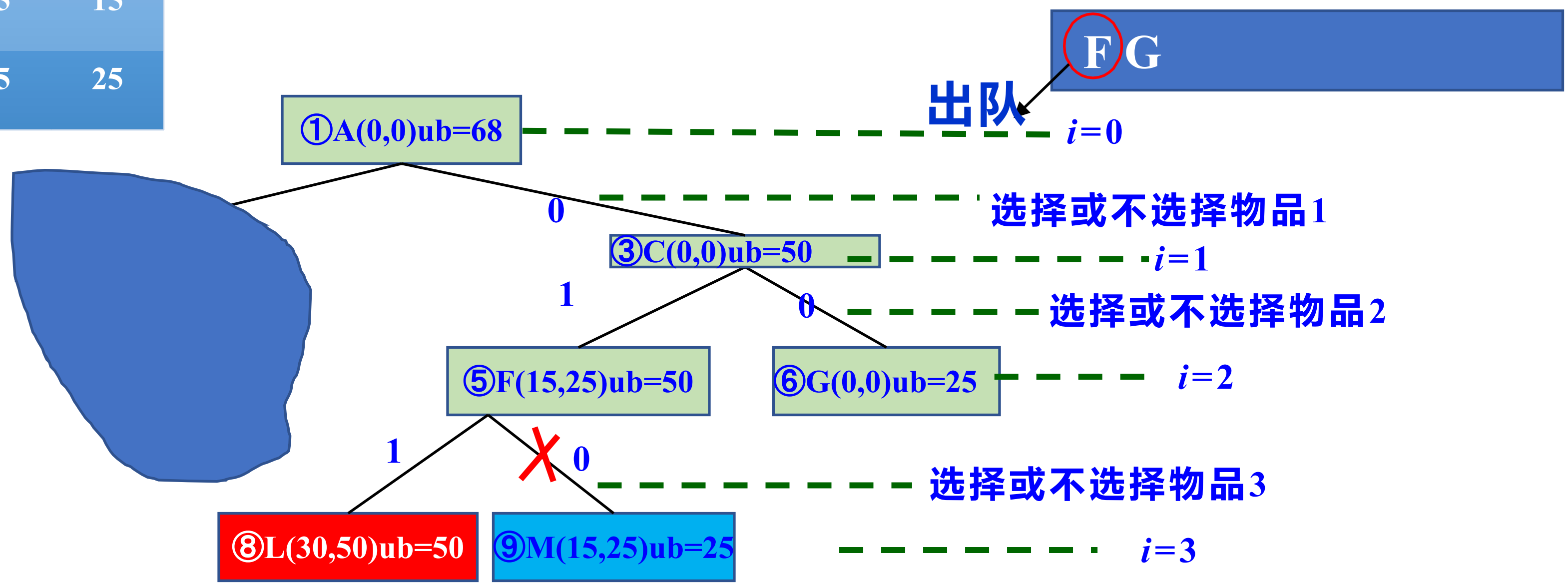
F G

因为K.ub>maxv(-9999)

所以K“扩展”，K进队（在进队处理时，因为K.i==n(3)，已经到达叶子结点,此时，K.v>maxv,maxv=K.v=45,bestx=(1,0,0),K没有真正进队）。

6 分支限界法求解0/1背包问题——普通队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



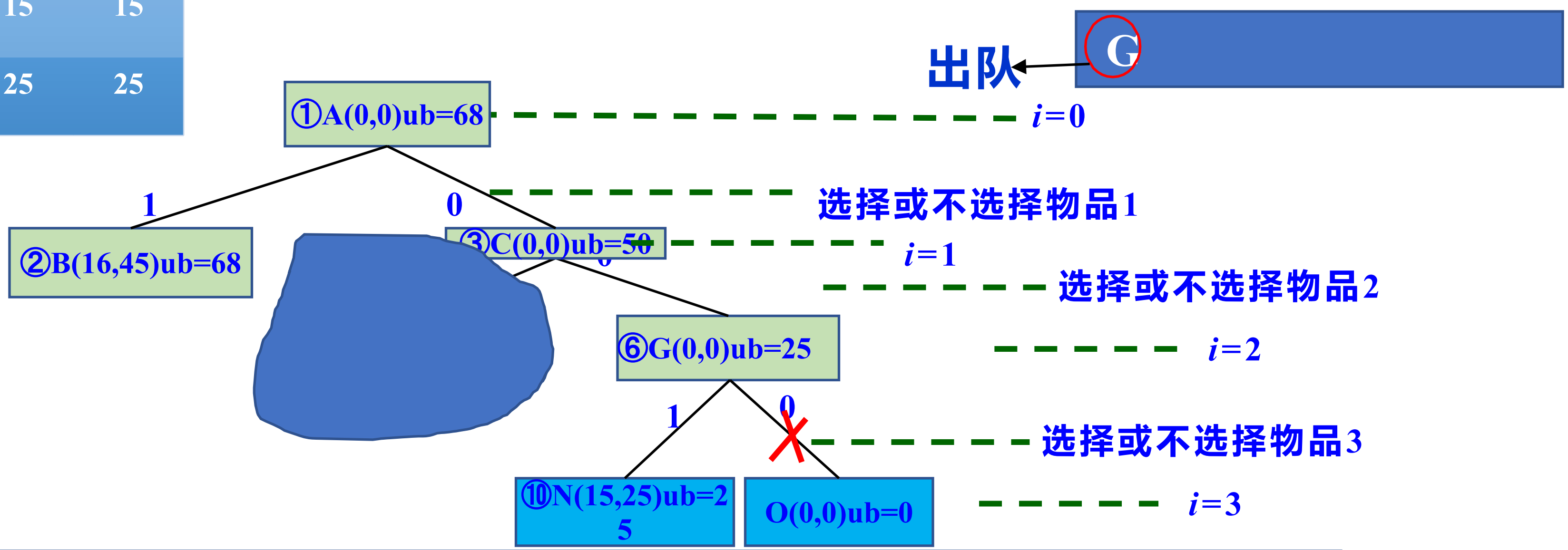
同理，计算L结点的各值，L.ub值计算同K结点，L进队处理也同K结点，L并未真正进队。

因为L.ub>maxv（45）所以L“扩展”，L“进队”（在进队处理时，因为L.i==n(3),此时，L.v>maxv,maxv=L.v=50,bestx=(0,1,1),L没有真正进队）。

计算M点各值，M.ub值计算同K结点，M.ub=25<maxv(50),所以，M被剪枝。

6 分支限界法求解0/1背包问题——普通队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



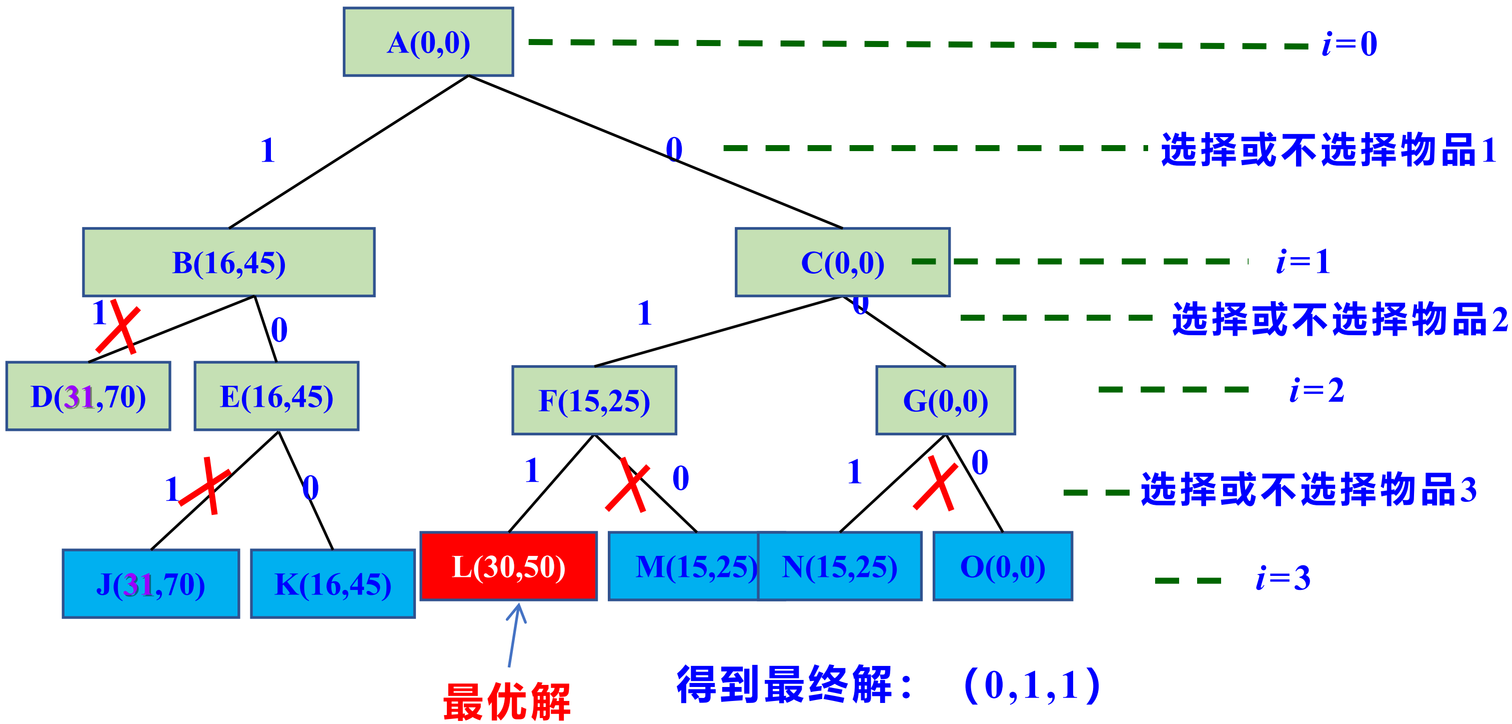
同理，计算N结点的各值，N.ub值计算同K结点，N进队处理也同K结点，N并未真正进队。

因为N.w=G.w+w[3]=15<30,所以N结点扩展，N“进队”（在进队处理时，因为N.i==n(3)，到达叶子结点,此时，N.v(25)<maxv(50),所以maxv维持原值=50,bestx=(0,1,1)不变,N没有真正进队）。

计算O点各值，O.ub值计算同K结点，）O.ub<maxv(50),所以，O被剪枝。

6 分支限界法求解0/1背包问题——普通队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



得到最终解：（0,1,1）

6 分支限界法求解0/1背包问题——普通队列式

普通队列式分支限界法求解0-1背包问题【算法描述】

求结点*e*的上界*e*.ub的算法如下：

```
void bound(NodeType &e)                     //计算分枝结点e的上界
{  int i=e.i+1;                             //考虑结点e的余下物品
   int sumw=e.w;                            //求已装入的总重量
   double sumv=e.v;                         //求已装入的总价值
   while ((sumw+w[i]<=W) && i<=n)
   {  sumw+=w[i];                           //计算背包已装入载重
      sumv+=v[i];                           //计算背包已装入价值
      i++;
   }

   if (i<=n)                                //余下物品只能部分装入
      e.ub=sumv+(W-sumw)*v[i]/w[i];
   else                                     //余下物品全部可以装入
      e.ub=sumv;
}
```

6 分支限界法求解0/1背包问题——普通队列式

普通队列式分支限界法求解0-1背包问题【算法描述】

```
//问题表示
int n=3,W=30;
int w[]={0,16,15,15};            //重量，下标0不用
int v[]={0,45,25,25};       //价值，下标0不用
//求解结果表示
int maxv=-9999;             //存放最大价值,初始为最小值
int bestx[MAXN];            //存放最优解,全局变量
int total=1;                //解空间中结点数累计,全局变量
struct NodeType             //队列中的结点类型
{   int no;                 //结点编号
    int i;              //当前结点在搜索空间中的层次
    int w;              //当前结点的总重量
    int v;              //当前结点的总价值
    int x[MAXN];            //当前结点包含的解向量
    double ub;              //上界
};
```

分支限界法求解0/1背包问题——普通队列式

普通队列式分支限界法求解0-1背包问题【算法描述】

```
void EnQueue(NodeType e,queue<NodeType> &qu)
//结点e进队qu
{  if (e.i==n)                          //到达叶子结点
   {  if (e.v>maxv)                     //找到更大价值的解
      {  maxv=e.v;
         for (int j=1;j<=n;j++)
           bestx[j]=e.x[j];
      }
   }
   else qu.push(e);                     //非叶子结点进队
}
```

注意!

在结点进队时判断是否为叶子结点

- **叶子结点对应一个解,叶子结点没有真正进队，作寻找最优解和构造输出解的操作**
- **叶子结点不再扩展**

6 分支限界法求解0/1背包问题——普通队列式

普通队列式分支限界法求解0-1背包问题【算法描述】

```
void bfs()                          //求0/1背包的最优解
{   int j;
    NodeType e,e1,e2;                    //定义3个结点
    queue<NodeType> qu;             //定义一个队列

    e.i=0;                      //根结点置初值，其层次计为0
    e.w=0; e.v=0;
    e.no=total++;
    for (j=1;j<=n;j++)
      e.x[j]=0;
    bound(e);                       //求根结点的上界
    qu.push(e);                     //根结点进队
```

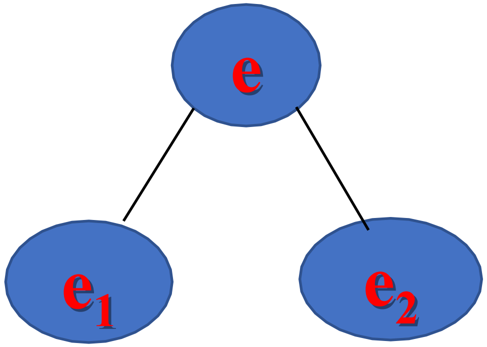
6 分支限界法求解0/1背包问题——普通队列式

普通队列式分支限界法求解0-1背包问题【算法描述】

```
while (!qu.empty())                              //队不空循环
  {   e=qu.front(); qu.pop();                          //出队结点e
      if (e.w+w[e.i+1]<=W)           //剪枝：检查左孩子结点
      {   e1.no=total++; e1.i=e.i+1;                   //建立左孩子结点
          e1.w=e.w+w[e1.i];e1.v=e.v+v[e1.i];
          for (j=1;j<=n;j++) e1.x[j]=e.x[j];                //复制解向量
           e1.x[e1.i]=1;
          bound(e1); EnQueue(e1,qu); }//求左孩子结点的上界,左孩子结点进队操作
          e2.no=total++;     e2.i=e.i+1;//建立右孩子结点
          e2.w=e.w; e2.v=e.v;
          for (j=1;j<=n;j++) e2.x[j]=e.x[j];                //复制解向量
          e2.x[e2.i]=0;
          bound(e2);                             //求右孩子结点的上界
      if (e2.ub>maxv)EnQueue(e2,qu);} }//若右孩子结点可行,则进队,否则被剪枝
```

6 分支限界法求解0/1背包问题——普通队列式

结点e → e1,e2，剪枝



- 左孩子e1：

e.w+w[e.i+1]<=W(约束），则进队

- 右孩子e2：

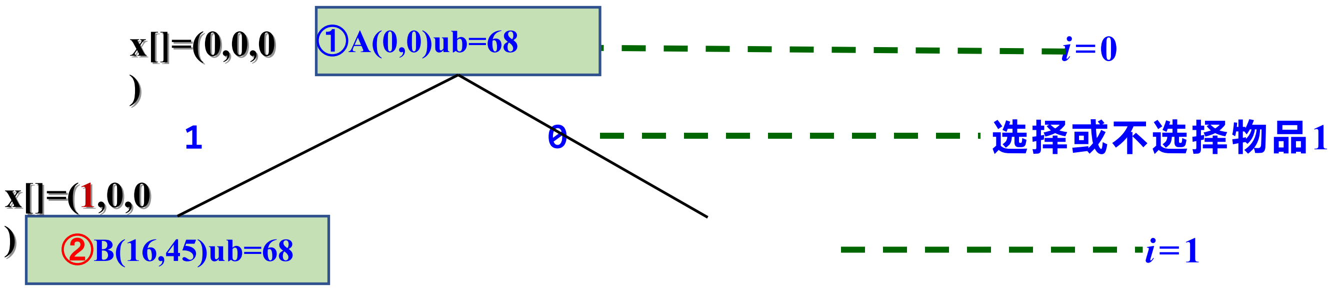
e2.ub>maxv（限界），则进队

6 分支限界法求解0/1背包问题——优先队列式

- 采用优先队列式分支限界法求解就是将一般的队列改为优先队列，但必须设计限界函数，因为优先级是以限界函数值为基础的。
- 限界函数的设计方法与前面的相同。这里用大根堆表示活结点表，取优先级为活结点所获得的价值。

6 分支限界法求解0/1背包问题——优先队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



优先队列

A（0，0）

结点*A出队*，*A*的层次i=0，w=0，v=0,A.ub=A.v+剩余物品最大价值（从物品1~物品3）=0+45+ (30-16)×25/15 =68:

因为w+w[1]=16<30,所以不剪左支，生成左子B结点

B.w=A.w+w[1]=16; B.v=A.v+v[1]=25;B.i=A.i+1=1(B.i为B结点的层号。

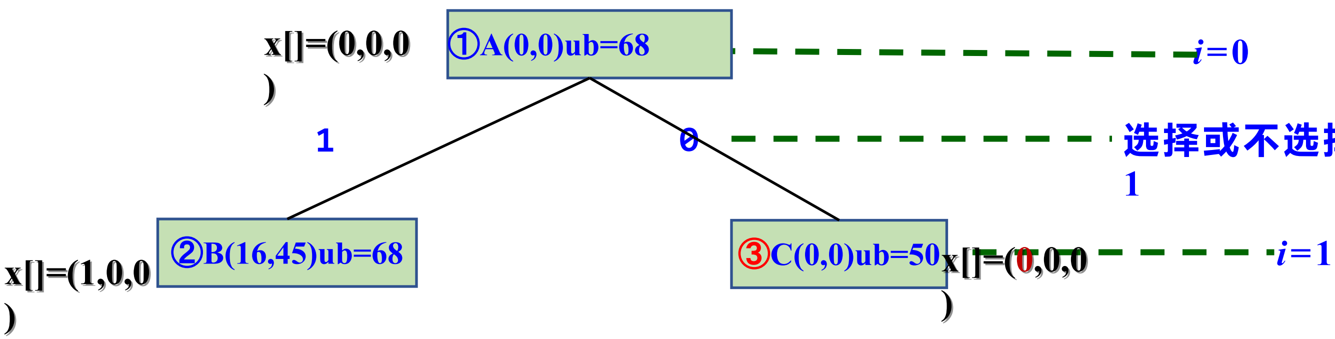
B.ub= B.v(45) + (30-16)×25/15 = 68（采用取整运算）

优先队列

B(16,45)

6 分支限界法求解0/1背包问题——优先队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



优先队列

B(16,45)

根结点A的层次i=0，w=0，v=0:看是否扩展右子。

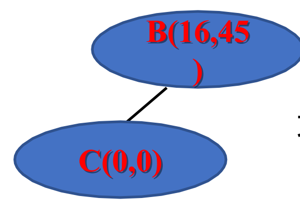
右子C.i=A.i+1;C.w=A.w=0;C.v=A.v=0，计算C.ub

C.ub=C.v+ $\boxed{25+25=}$50 因为C.ub>maxv(-9999)，所以C结点要扩展，放入活结点表（队列）

B(16,45) C(0,0)

物品1未选，C.w=0

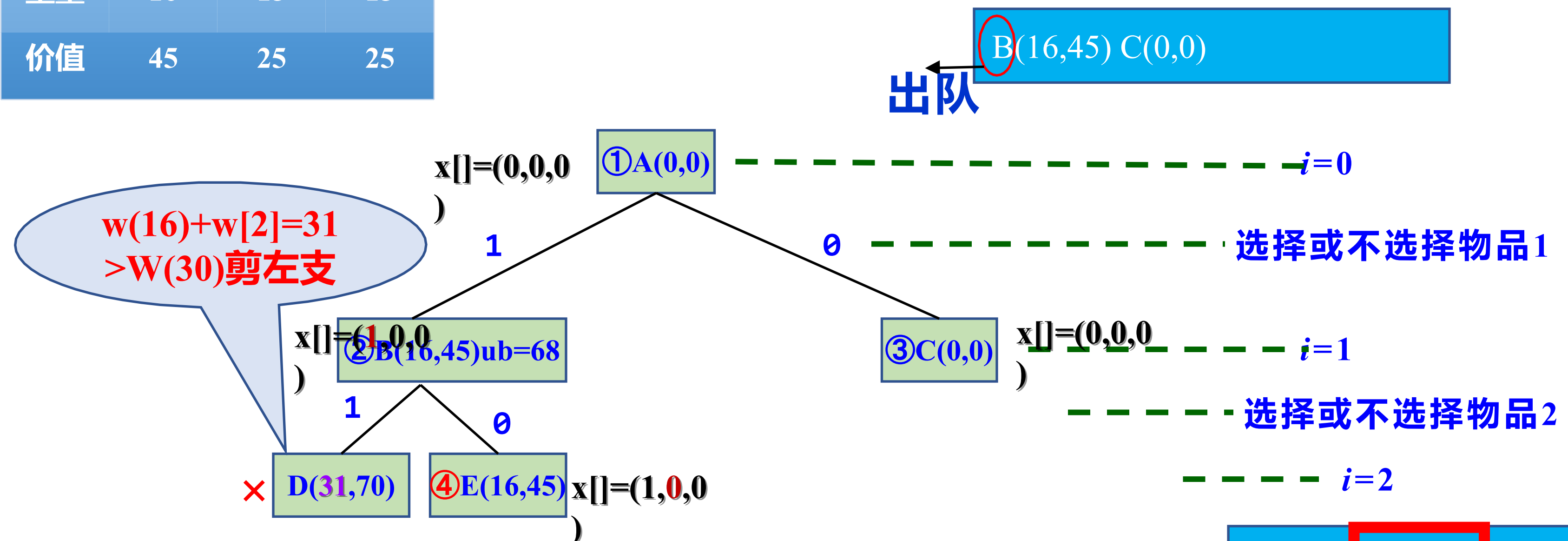
剩余物品即物品2，物品3。因为重量均为15，所以剩余物品全选的最大价值为25+25=50



堆调整

6 分支限界法求解0/1背包问题——优先队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



结点B的层次B.i=1， B.w=16， B.v=45:看是否扩展左子E

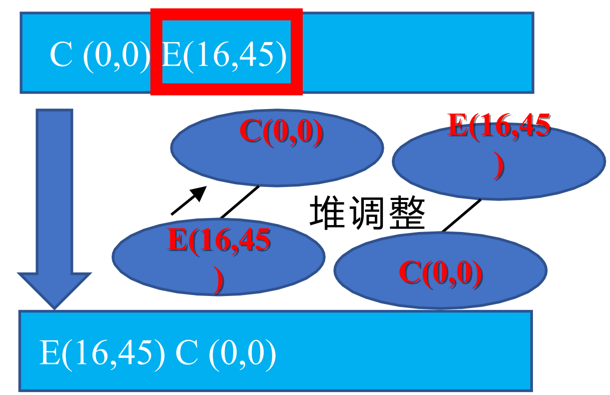
E.i=B.i+1=2; E.w=$B.w$; E.v=B.v

E.ub=45 + (30-16)×25/15 = 68（采用取整运算）用于限界.

B.v

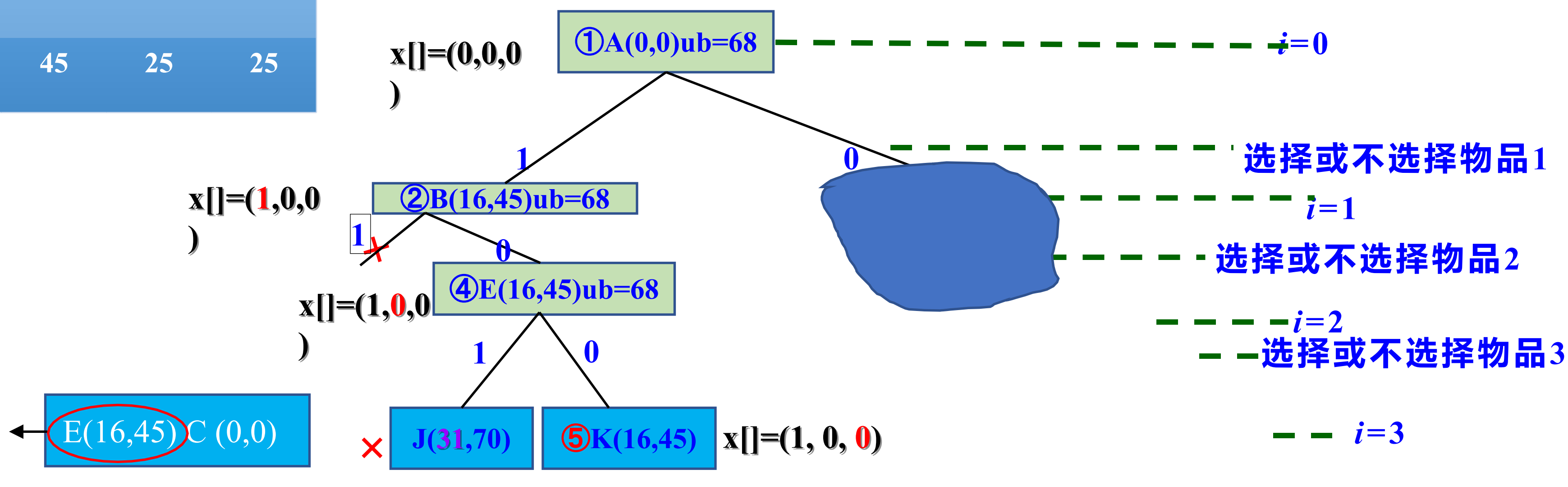
可选物品2的一部分，即30-16，对应的价值

E.ub(68)>maxv(-9999),E进队



6 分支限界法求解0/1背包问题——优先队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



结点E的层次E.i=2，E.w=16，E.v=45:看是否扩展右子K

结点E的层次E.i=2，E.w=16，E.v=45:看是否扩展左子J

K.i=3，K.w=16，K.v=45

K.ub=K.v+物品4~n的最大价值=45，不用限界

E.w+w[3]=16+15=31>30，所以不扩展，剪掉E的左支。

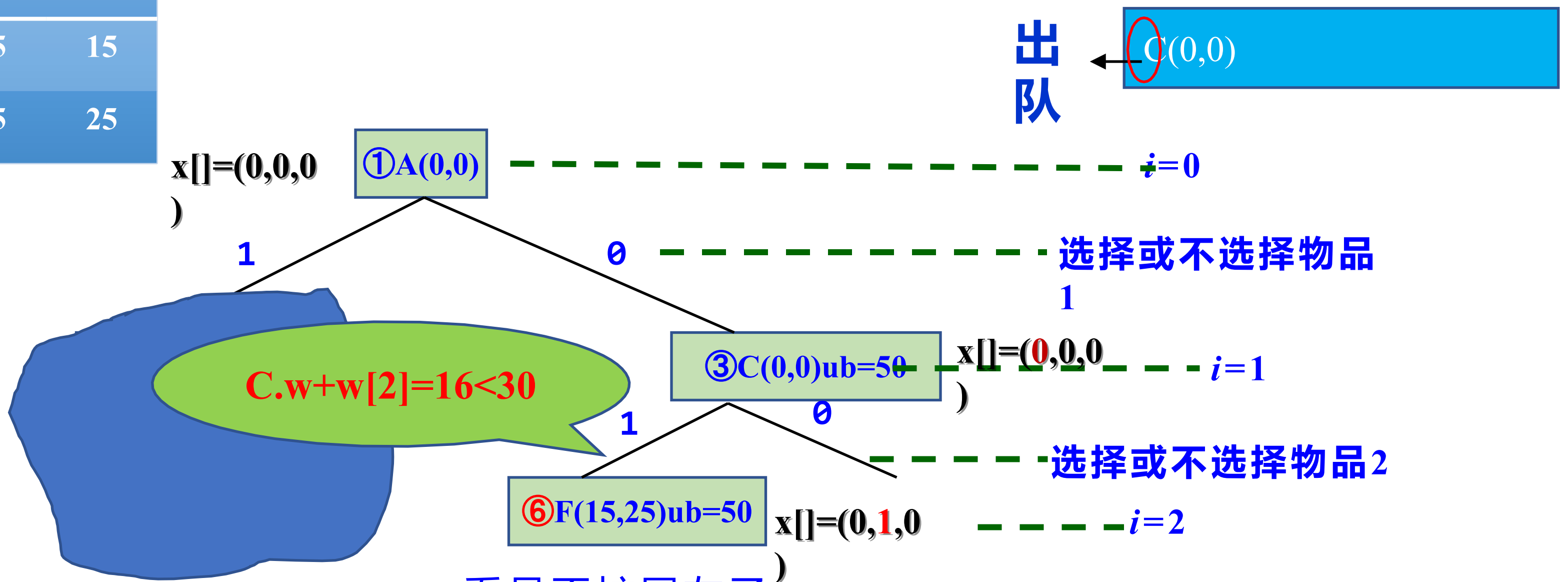
C (0,0)

因为K.ub>maxv(-9999)

所以K“扩展”，K“进队”（在进队处理时，因为K.i==n(3)，已经到达叶子结点,此时，K.v>maxv,maxv=K.v=45,bestx=(1,0,0),K没有真正进队）。

6 分支限界法求解0/1背包问题——优先队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



结点C的层次C.i=1，C.w=0，C.v=0:看是否扩展左子F

因为C.w+w[2]=16<30，所以扩展出左子F

F.i=C.i+1=2; F.w=C.w+w[2]=15; F.v=C.v+v[2]=25

F(15,25)

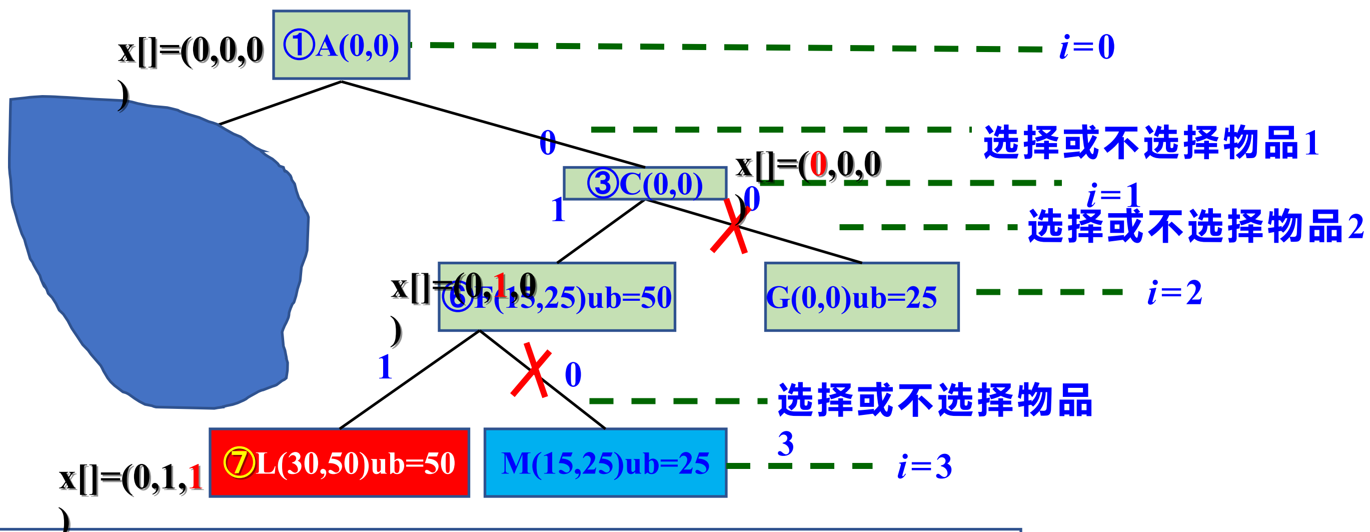
F.ub=25 + 25 = 50（采用取整运算）用于限界.

F.v

剩余物品最大价值（只剩物品3了，且物品3可以全部装入，价值为25）

6 分支限界法求解0/1背包问题——优先队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



同理，计算L结点的各值，L.ub值计算同K结点，L进队处理也同K结点，L并未真正进队。

因为L.ub>maxv（45）所以L“扩展”，L“进队”（在进队处理时，因为L.i==n(3),

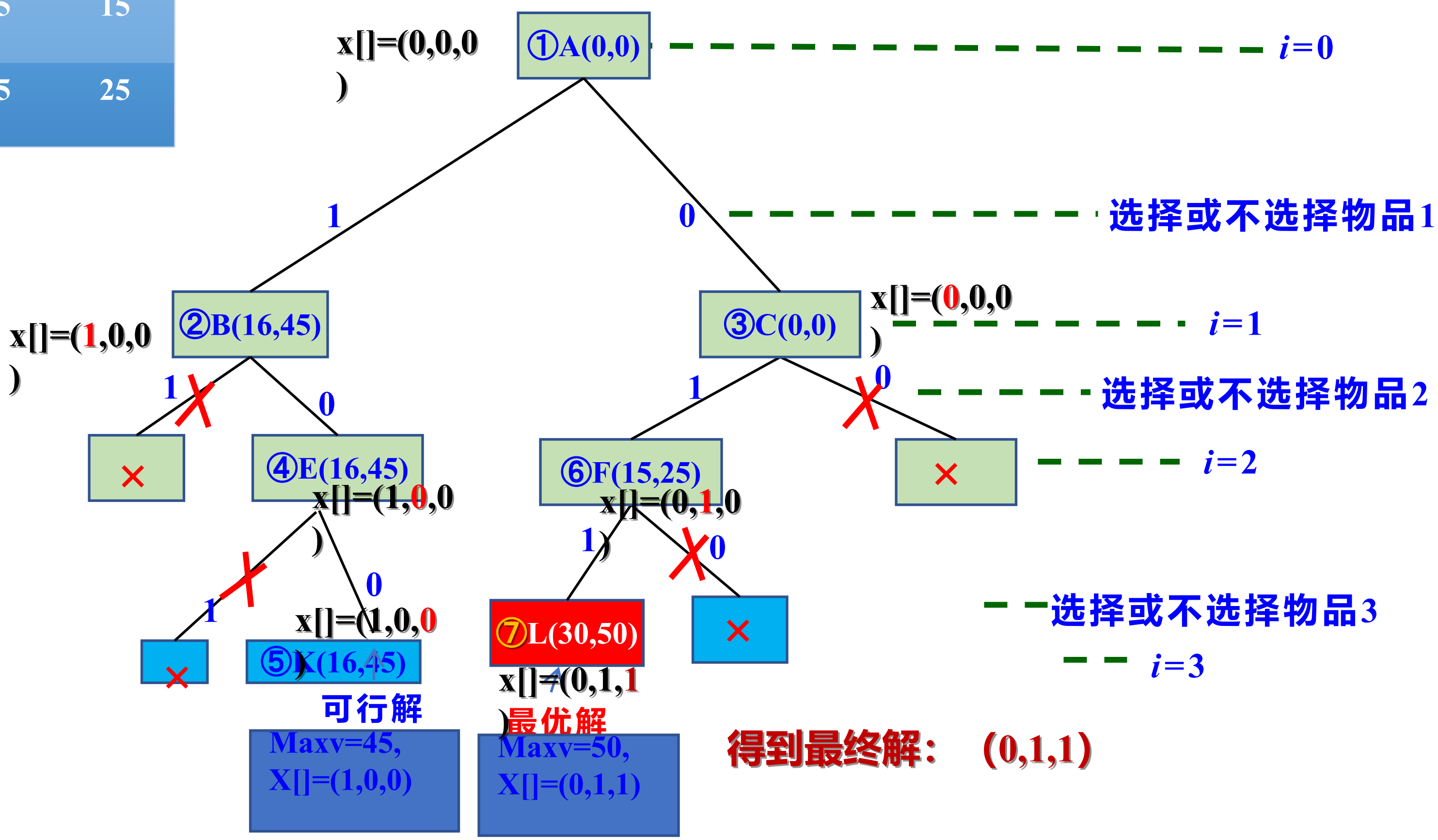
此时，L.v>maxv,maxv=L.v=50,bestx=(0,1,1),L没有真正进队）。

计算M点各值，M.ub值计算同K结点，M.ub=25<maxv(50),所以，M被剪枝。

6

分支限界法求解0/1背包问题——优先队列式

编号	1	2	3
重量	16	15	15
价值	45	25	25



得到最终解：（0,1,1）

6 分支限界法求解0/1背包问题——优先队列式

优先队列式分支限界法求解0-1背包问题【算法描述】

```
struct NodeType          //队列中的结点类型
{   int no;              //结点编号
    int i;               //当前结点在搜索空间中的层次
    int w;               //当前结点的总重量
    int v;               //当前结点的总价值
    int x[MAXN];         //当前结点包含的解向量
    double ub;                //上界

    bool operator<(const NodeType &s) const //重载<关系函数
    {
        return ub<s.ub;       //ub越大越优先出队
    }
};
```

6 分支限界法求解0/1背包问题——优先队列式

优先队列式分支限界法求解0-1背包问题【算法描述】

```
void bfs()                              //求0/1背包的最优解
{  int j;
   NodeType e,e1,e2;                    //定义3个结点
   priority_queue<NodeType> qu;//定义一个优先队列（大根堆）
   e.i=0;                               //根结点置初值，其层次计为0
   e.w=0; e.v=0;
   e.no=total++;
   for (j=1;j<=n;j++)
      e.x[j]=0;
   bound(e);                            //求根结点的上界
   qu.push(e);                          //根结点进队
```

分支限界法求解0/1背包问题——优先队列式

优先队列式分支限界法求解0-1背包问题【算法描述】

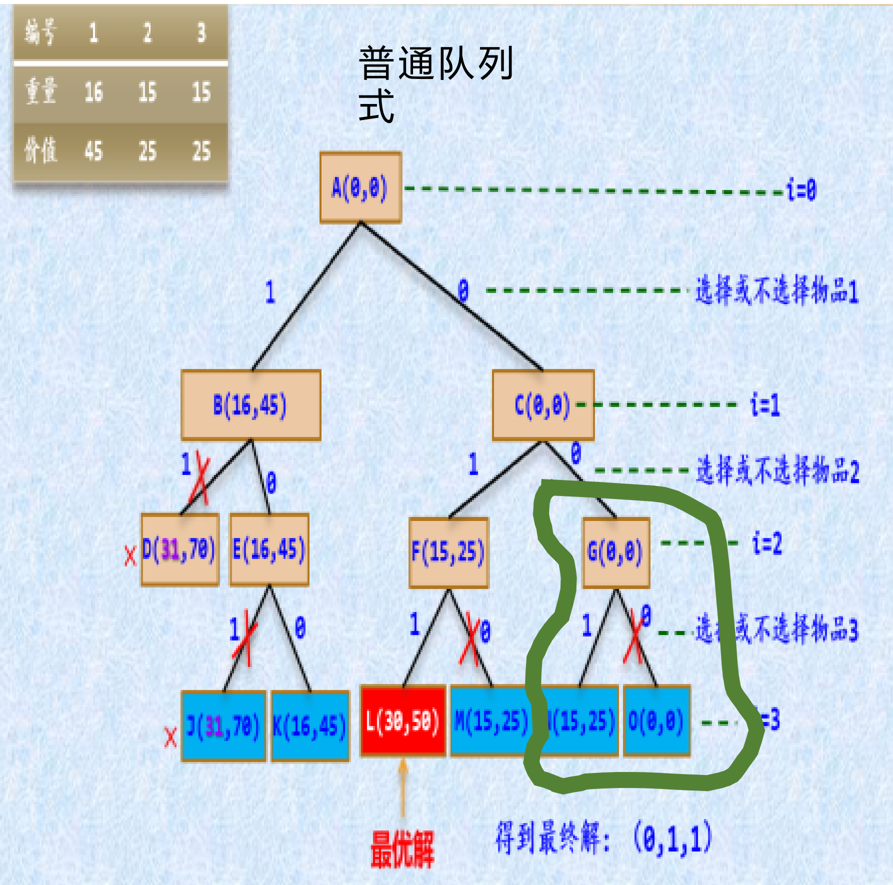
```
while (!qu.empty())                    //队不空循环
{  e=qu.top(); qu.pop();               //出队结点e
   if (e.w+w[e.i+1]<=W)                //剪枝：检查左孩子结点
   {  e1.no=total++;
      e1.i=e.i+1; e1.w=e.w+w[e1.i]; e1.v=e.v+v[e1.i]; //建立左孩子结点
      for (j=1;j<=n;j++) e1.x[j]=e.x[j];    //复制解向量
      e1.x[e1.i]=1;bound(e1);                    //求左孩子结点的上界
      qu.push(e1);                      //左孩子结点进队操作
   }
   e2.no=total++; e2.i=e.i+1; e2.w=e.w; e2.v=e.v;//建立右孩子结
   for (j=1;j<=n;j++) e2.x[j]=e.x[j];       //复制解向量
   e2.x[e2.i]=0;
   bound(e2);                          //求右孩子结点的上界
   if (e2.ub>maxv)                     //若右孩子结点的上界大于maxv则不剪枝
      qu.push(e2);}}
```

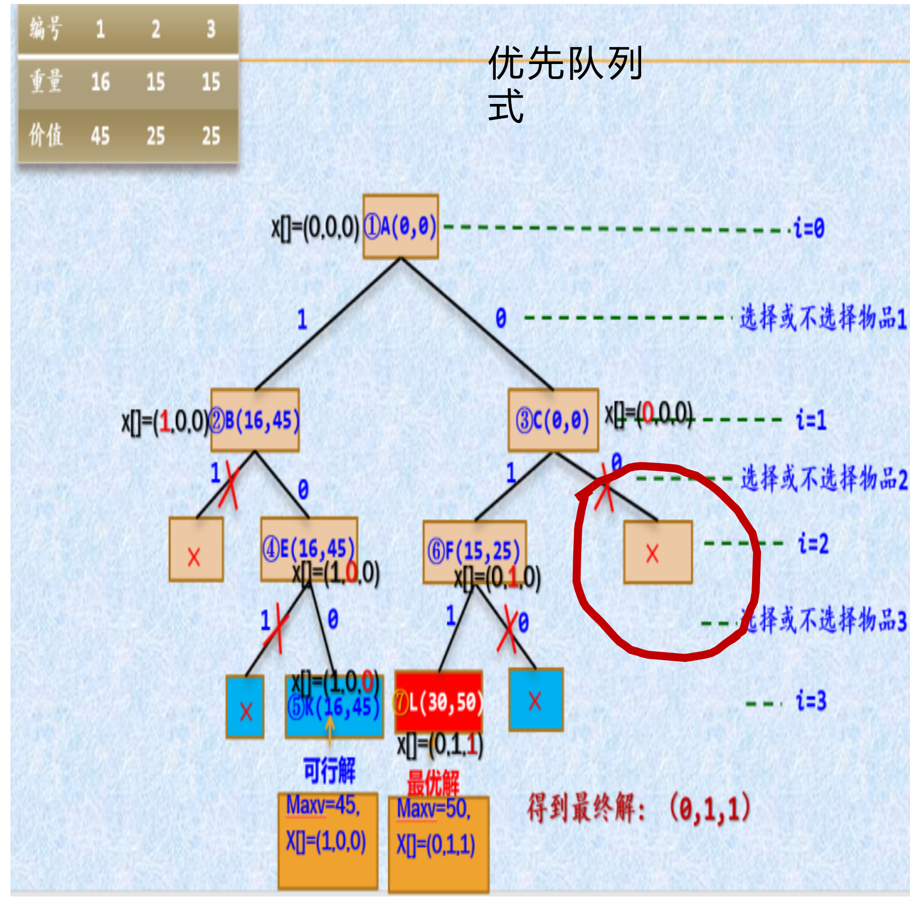
不是进普通队列，是进优先队列

6 分支限界法求解0/1背包问题——优先队列式

分支限界法求解0/1背包问题

——普通队列式与优先队列式比较





分支限界法求解0/1背包问题——优先队列式

【算法分析】无论采用队列式分支限界法还是优先队列式分支限界法求解0/1背包问题，最坏情况下要搜索整个解空间树，所以最坏时间和空间复杂度均为$O(2^n)$，其中n为物品个数。

分支限界法示例

图的单源最短路径

6 分支限界法求解图的单源最短路径

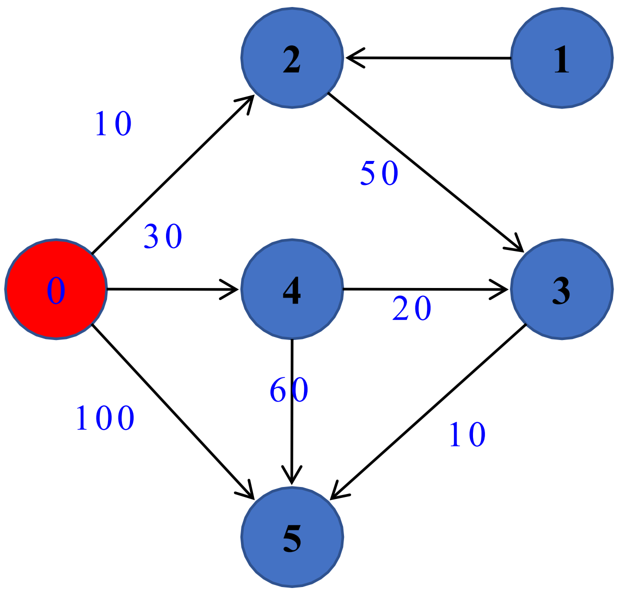
【问题描述】给定一个带权有向图G=（V，E），其中每条边的权是一个正整数。另外，还给定V中的一个顶点v，称为源点。

计算从源点到其他所有顶点的最短路径长度。

这里的长度是指路上各边权之和。

6 分支限界法求解图的单源最短路径

求取从顶点0出发到其余顶点的最短路径



带权有向图G

$$\begin{bmatrix} 0 & \infty & 10 & \infty & 30 & 100 \\ \infty & 0 & 4 & \infty & \infty & \infty \\ \infty & \infty & 0 & 50 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty & 10 \\ \infty & \infty & \infty & 20 & 0 & 60 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

带权有向图的邻接矩阵

6 分支限界法求解图的单源最短路径

解空间树中结点设计：即队列结点类型声明如下：

```
struct NodeType             //队列结点类型
{   int vno;                //顶点编号
    int length;                  //路径长度
};
```

- 此例中，结点中并未设计记录路径path的解向量，采用第二种方法设计解向量：即用prev数组存放最短路径，prev[i]表示源点v到顶点i的最短路径中顶点i的前驱顶点。
- 用dist数组存放源点v出发的最短路径长度，dist[i]表示源点v到顶点i的最短路径长度，初始时所有dist[i]值为∞。

6 分支限界法求解图的单源最短路径

分支限界求解单源最短路径问题相关定义

```
#define INF 0x3f3f3f3f  //表示∞
#define MAXN 51
//问题表示
int n;                 //图顶点个数
int a[MAXN][MAXN];//图的邻接矩阵
int v;                 //源点

//求解结果表示
int dist[MAXN];    //dist[i]源点到顶点i的最短路径长度
int prev[MAXN];    //prev[i]表示源点到j的最短路径中顶点j的前驱顶点

struct NodeType    //队列结点类型
{  int vno;          //顶点编号
   int length;           //路径长度
};
```

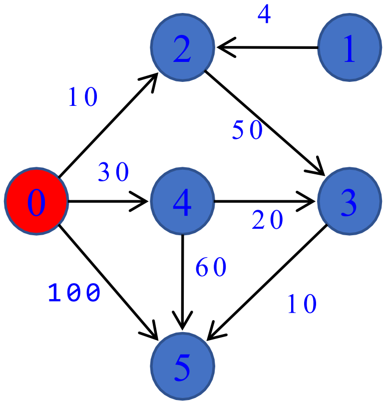
6 分支限界法求解图的单源最短路径

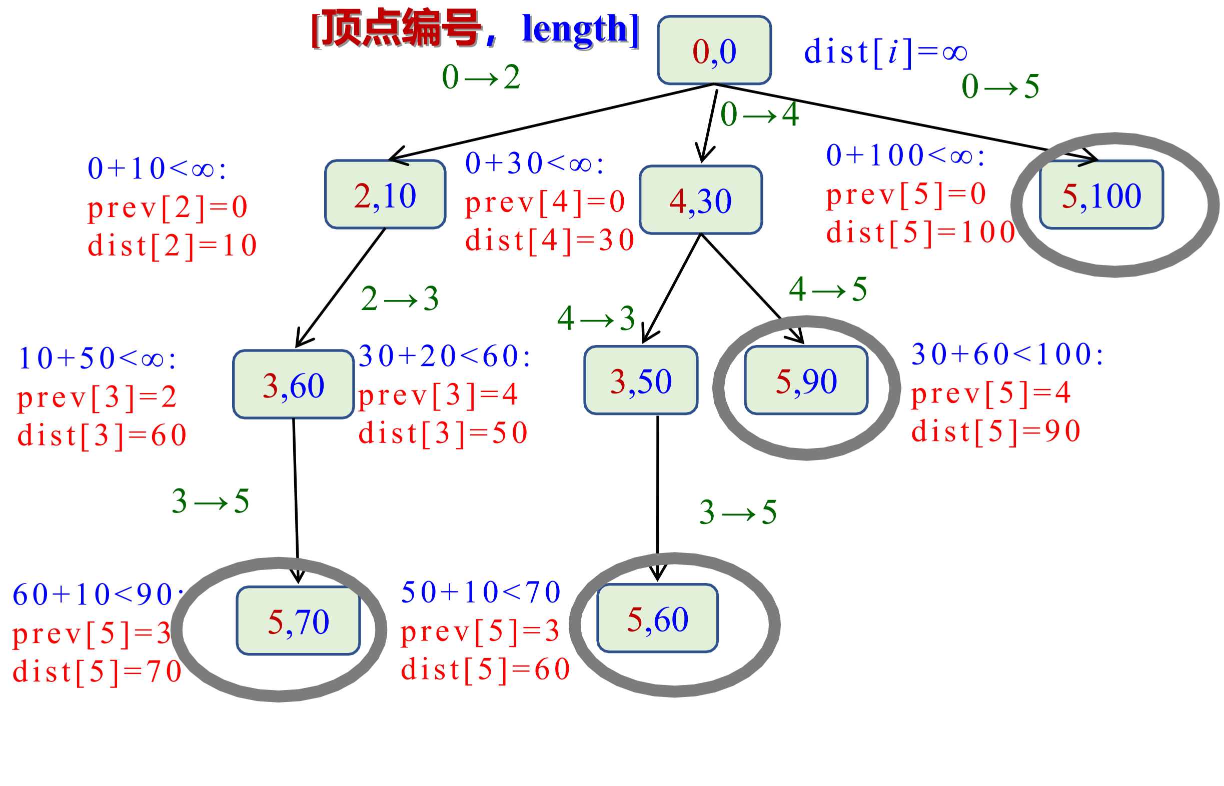
普通队列式分支限界法求解单源最短路径问题【算法描述】

```
void bfs(int v)                          //求解算法
{  NodeType e,e1;
   queue<NodeType> qu;
   e.vno=v; e.length=0;//建立源点结点e（根结点）
   qu.push(e);                           //源点结点e进队
   dist[v]=0;
  while(!qu.empty())                     //队列不空循环
   {  e=qu.front(); qu.pop();            //出队列结点e
      for (int j=0; j<n; j++)
      {  if(a[e.vno][j]<INF && e.length+a[e.vno][j]<dist[j])
         {//e.vno到顶点j有边并且路径长度更短
                 dist[j]=e.length+a[e.vno][j];
                 prev[j]=e.vno;
                 e1.vno=j;               //建立相邻顶点j的结点e1
                 e1.length=dist[j];
                 qu.push(e1);            //结点e1进队
     }}}}
```

与出队的顶点有边相连

6 分支限界法求解图的单源最短路径—普通队列式



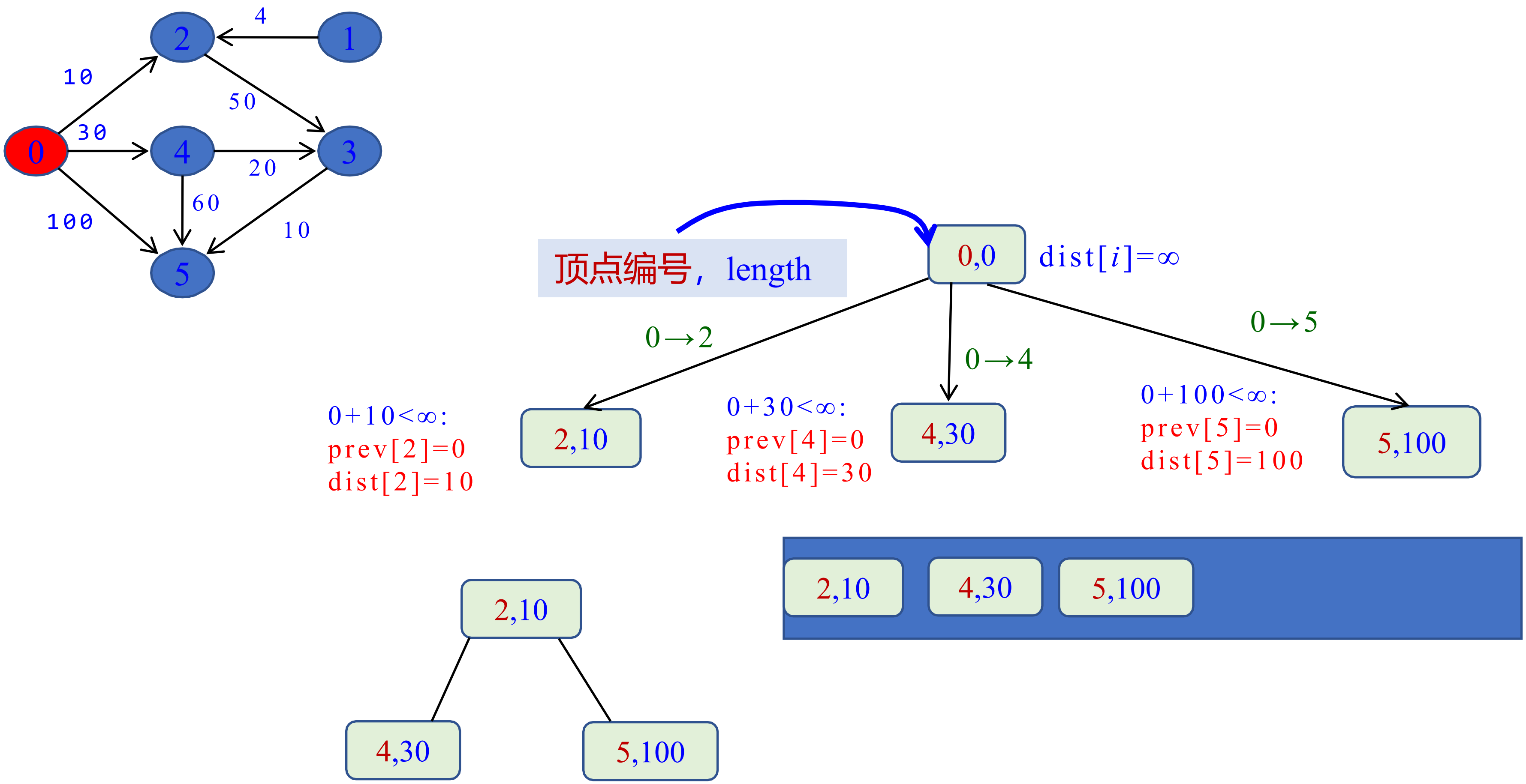


dist[1]=∞， prev[1]=*； dist[2]=10， prev[2]=0
dist[3]=50， prev[3]=4 ； dist[4]=30， prev[4]=0
dist[5]=60， prev[5]=3

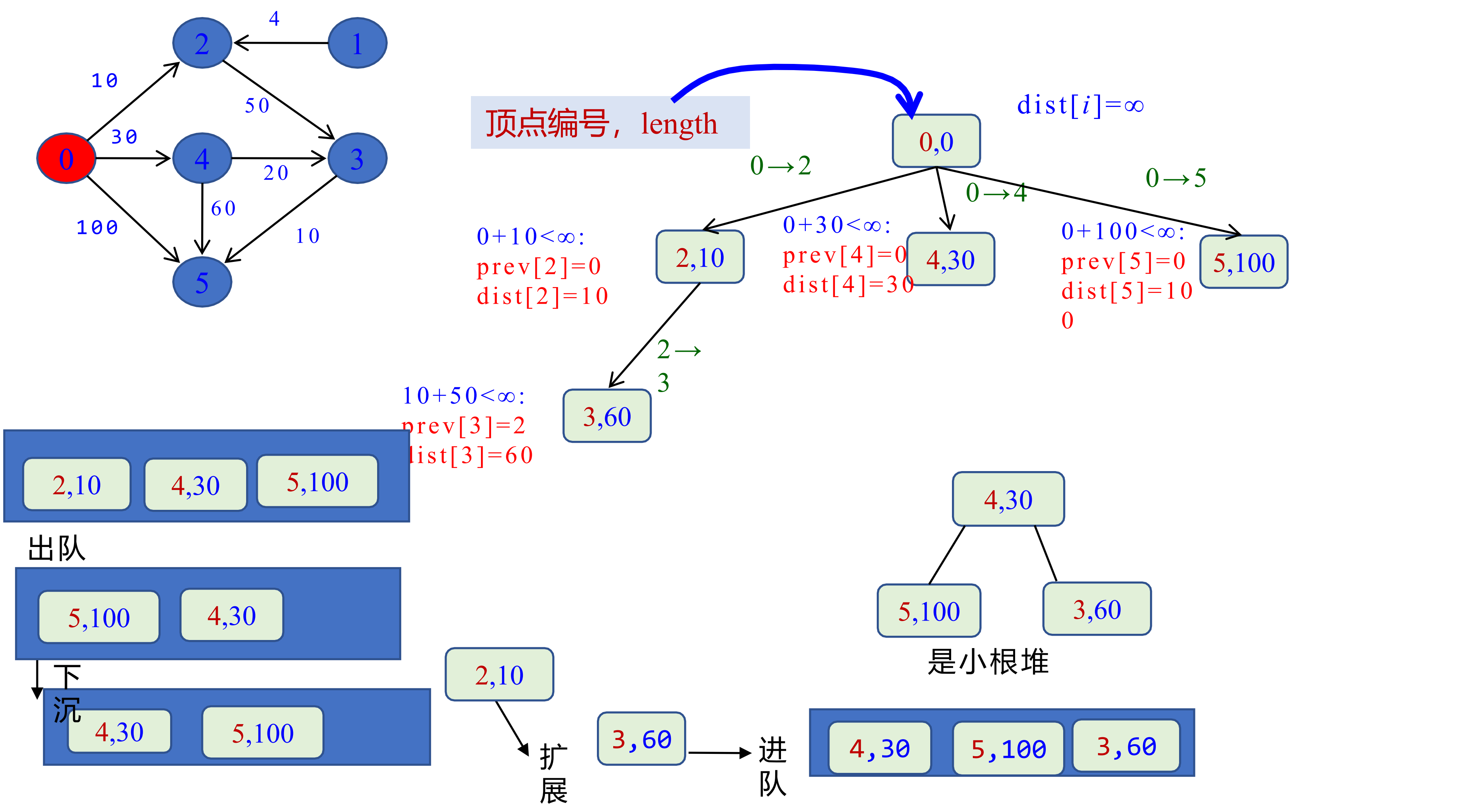
顶点0出发到其余顶点的的最短路径

6

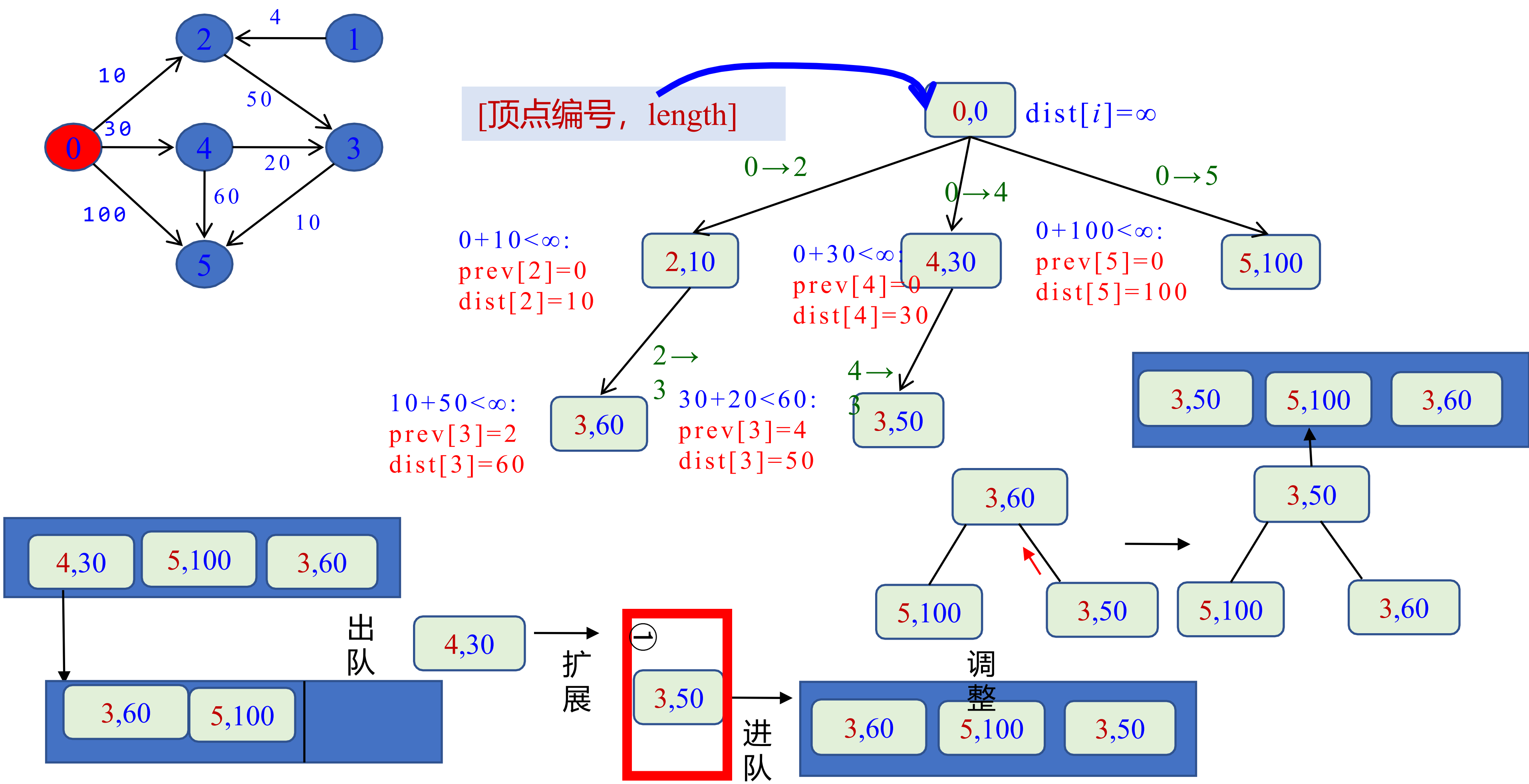
分支限界法求解图的单源最短路径—优先队列式



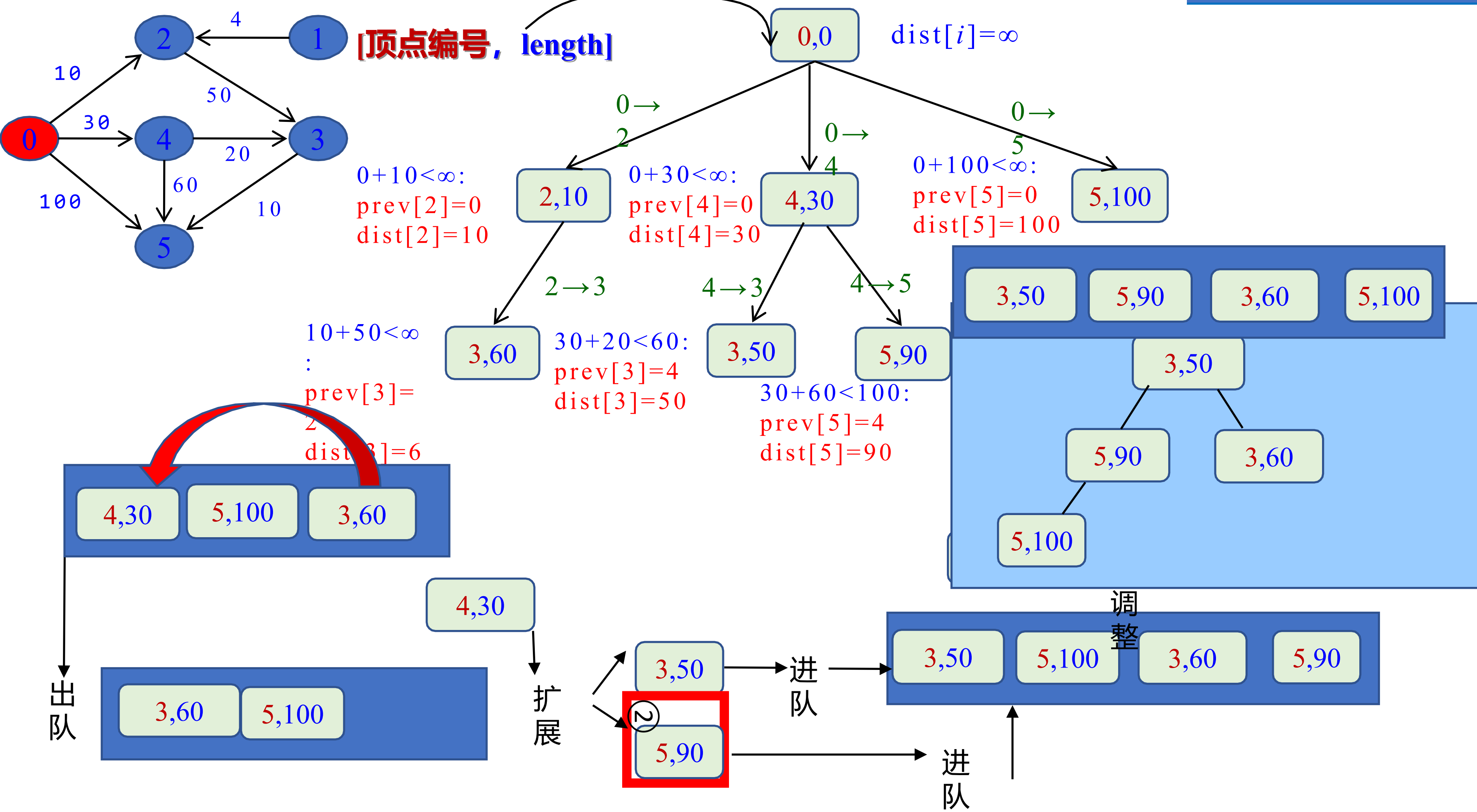
6 分支限界法求解图的单源最短路径—优先队列式



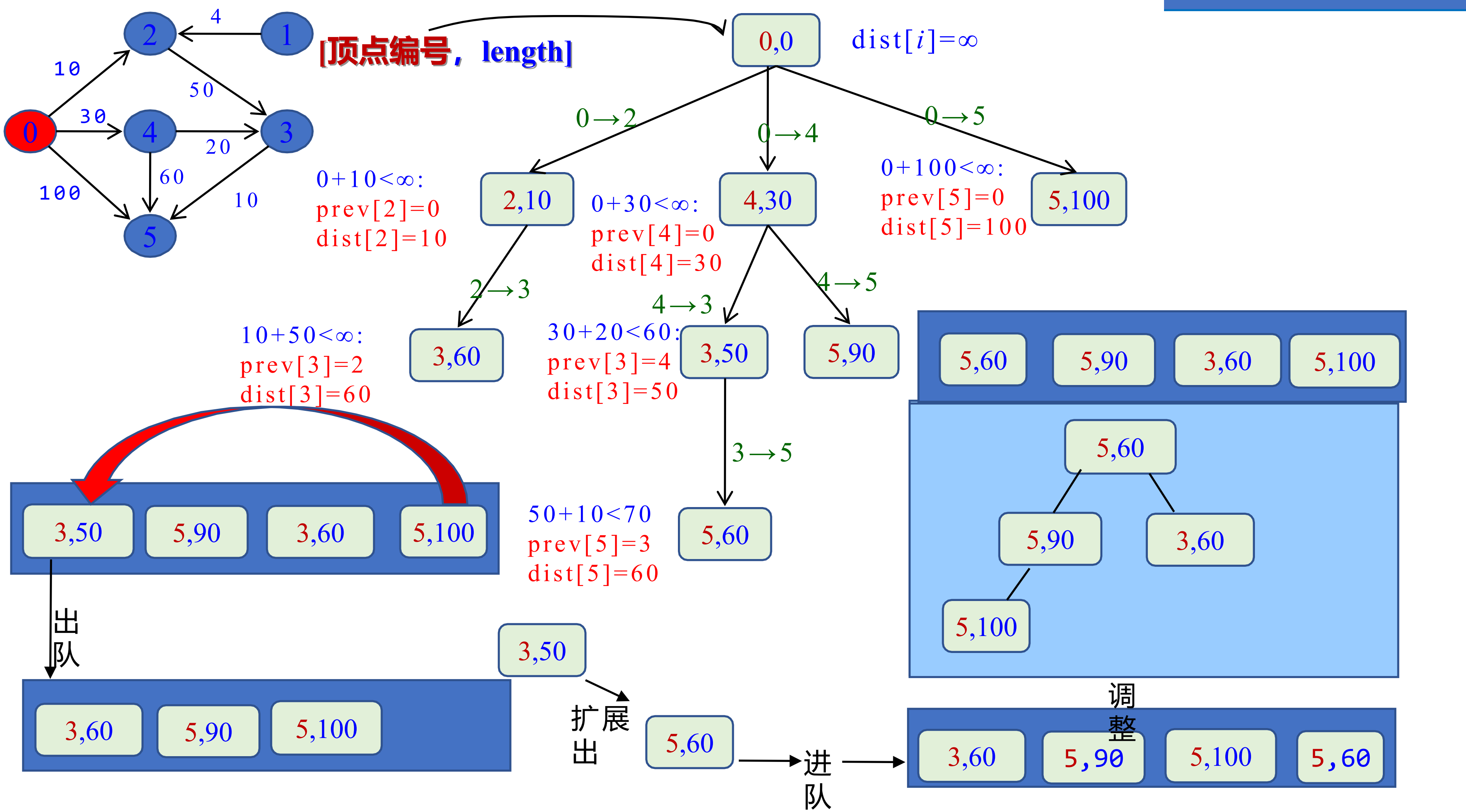
6 分支限界法求解图的单源最短路径—优先队列式



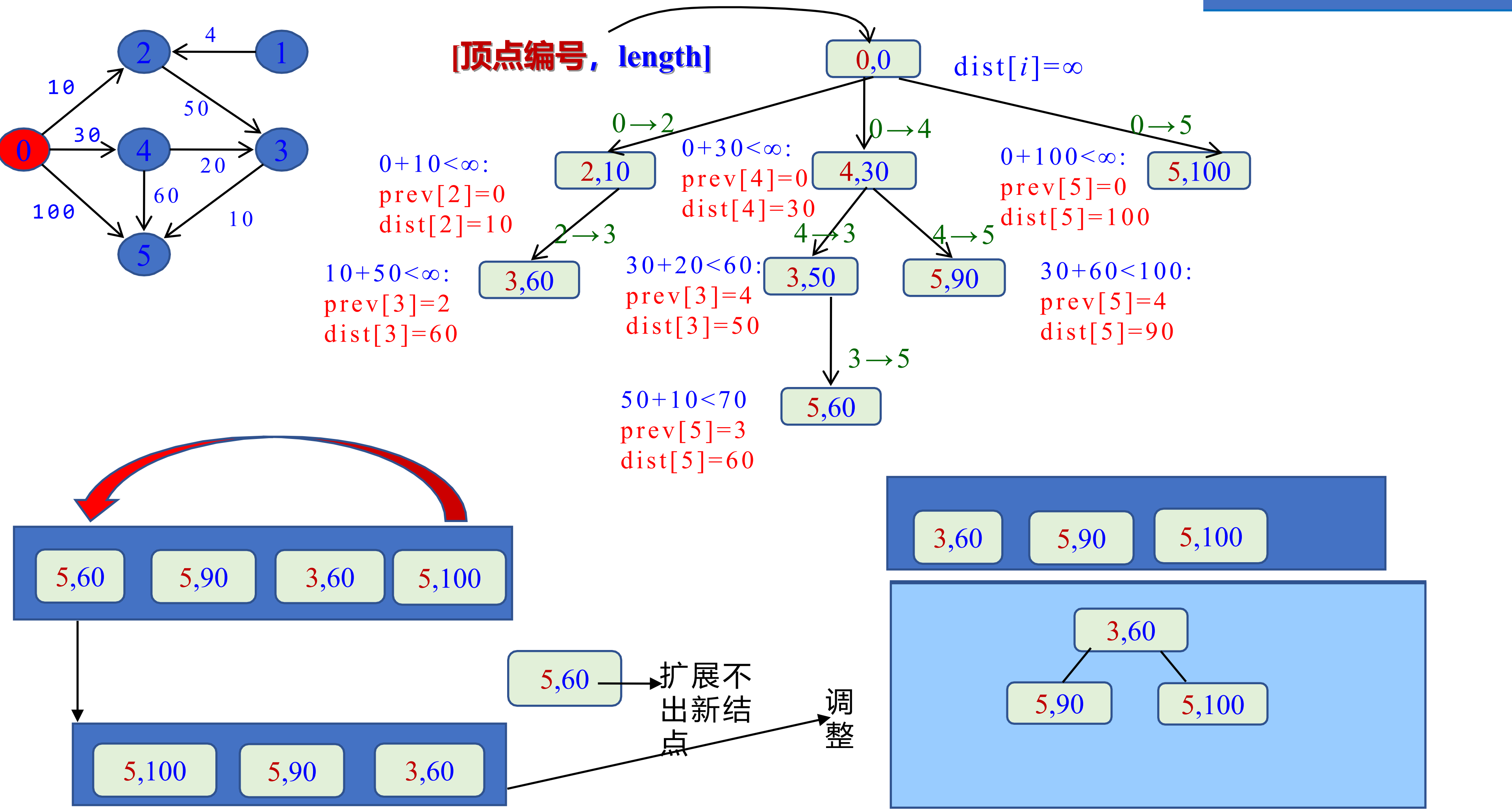
6 分支限界法求解图的单源最短路径—优先队列式



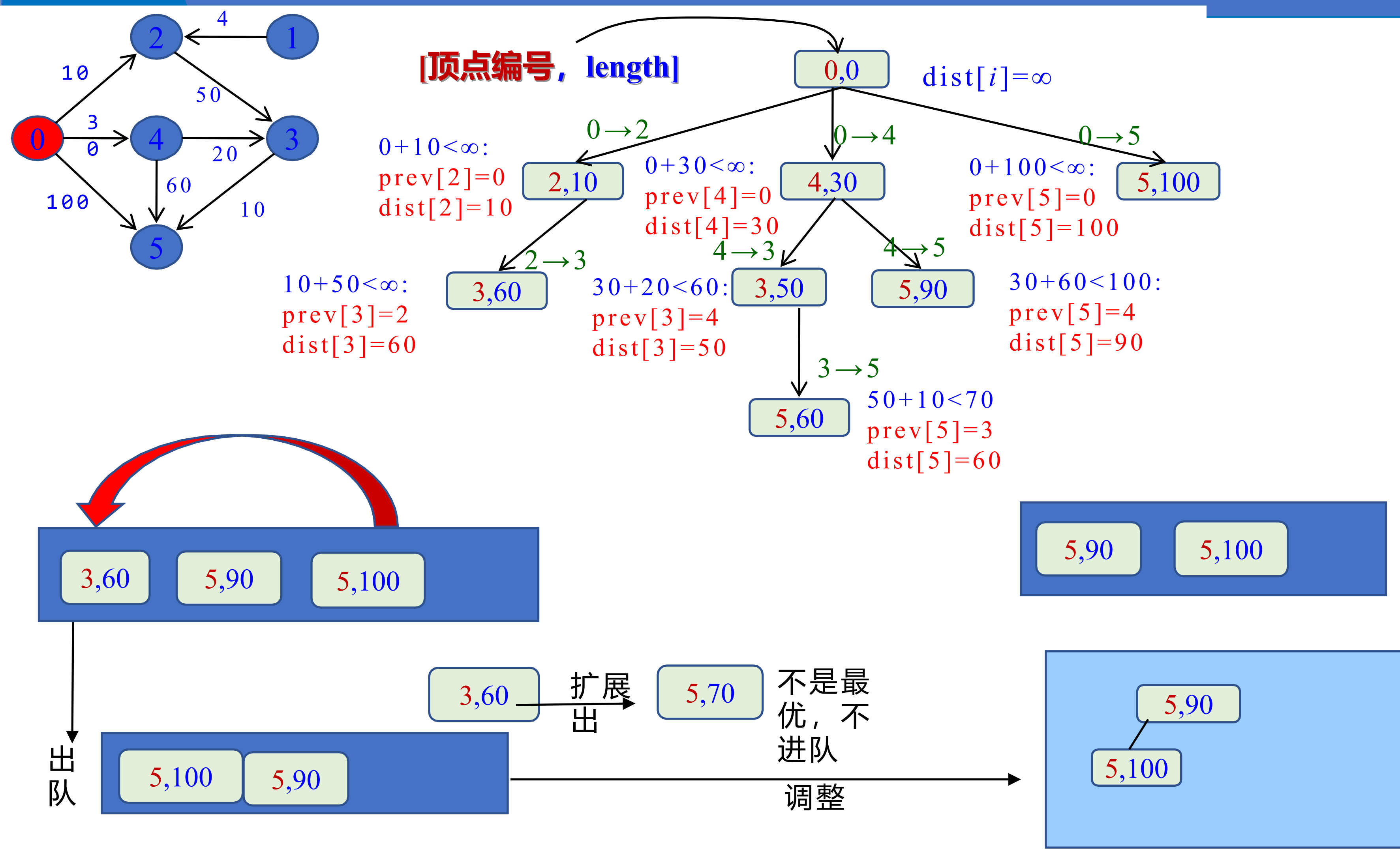
6 分支限界法求解图的单源最短路径—优先队列式



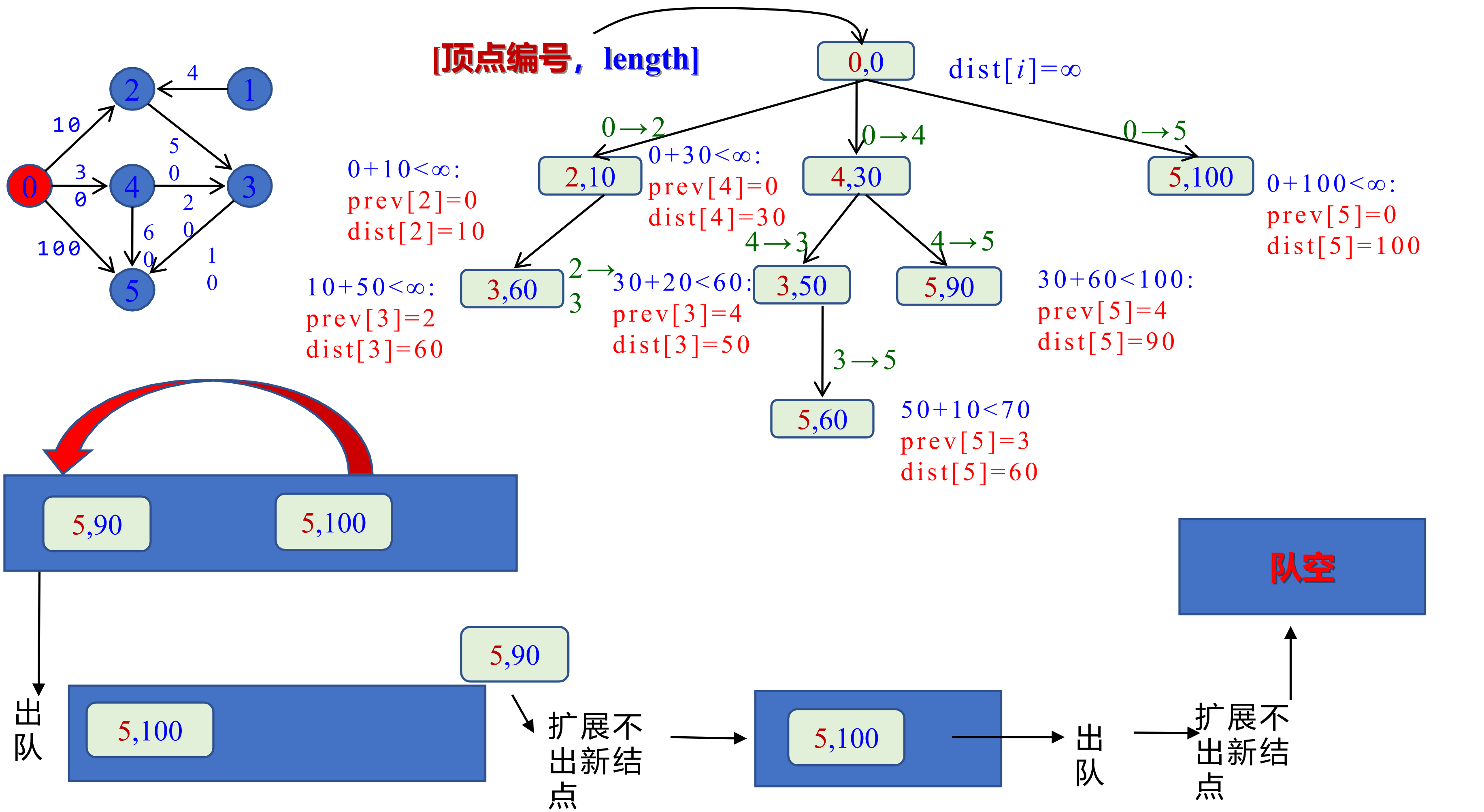
6 分支限界法求解图的单源最短路径—优先队列式



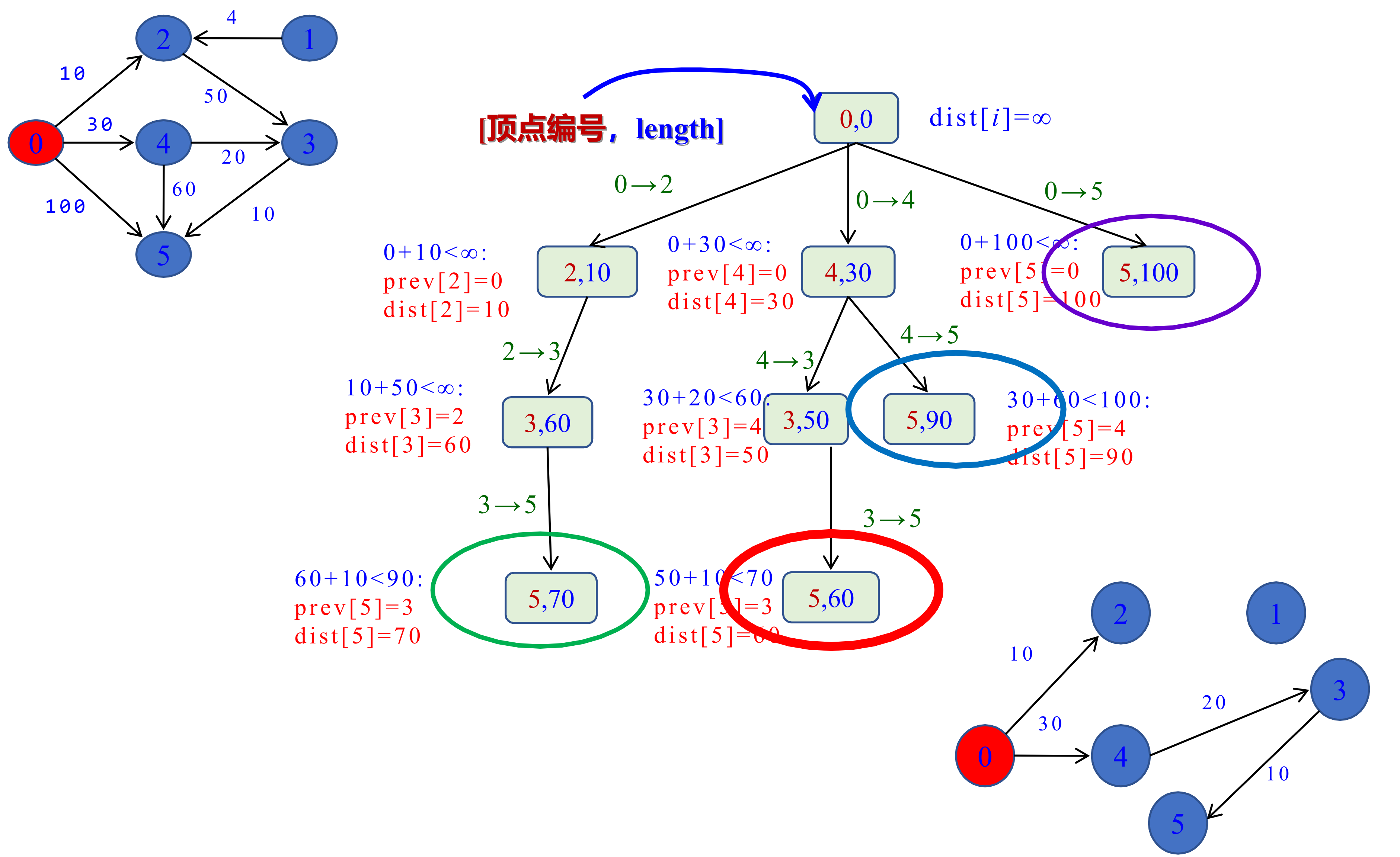
6 分支限界法求解图的单源最短路径—优先队列式



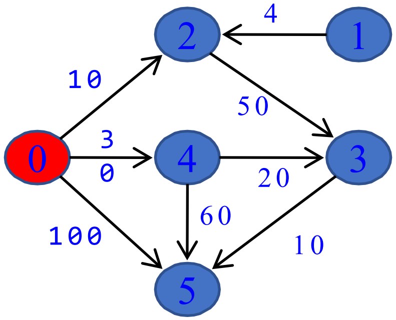
6 分支限界法求解图的单源最短路径—优先队列式

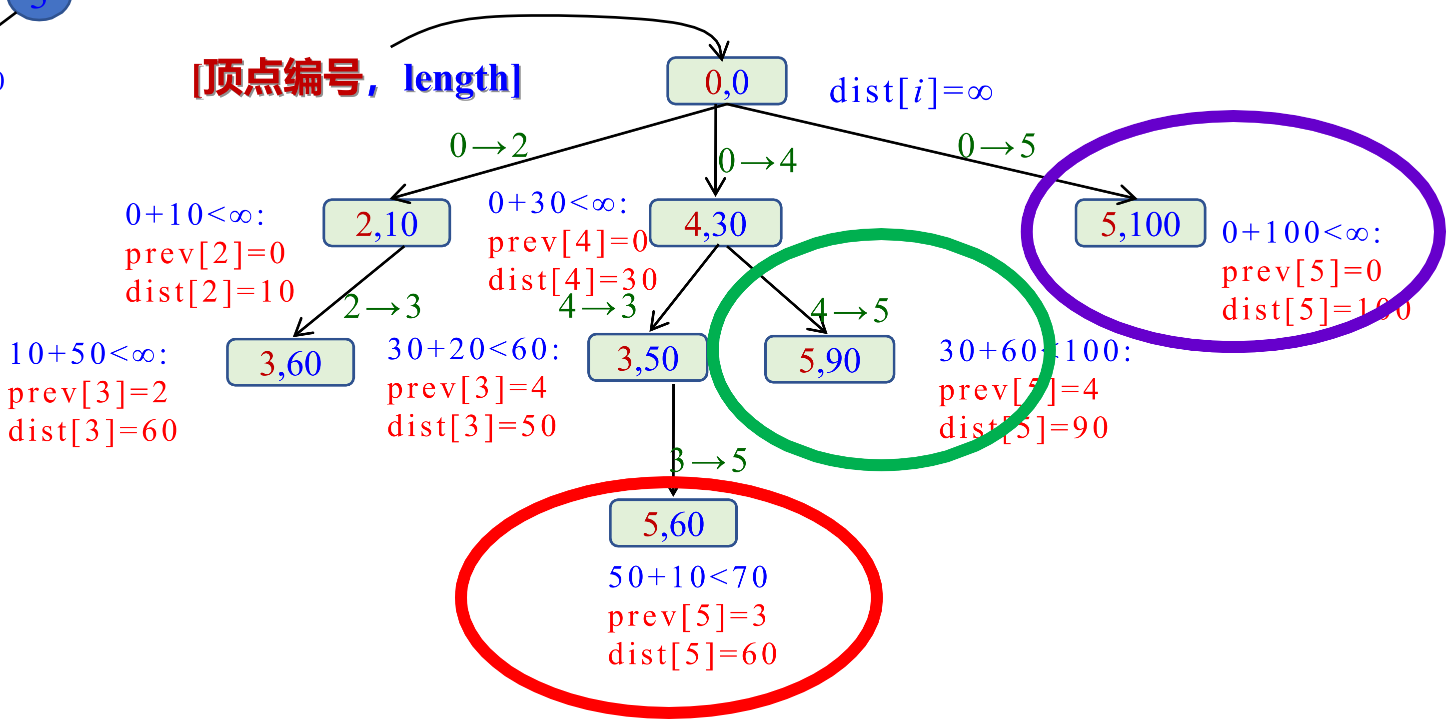


6 分支限界法求解图的单源最短路径—普通队列式



6 分支限界法求解图的单源最短路径—优先队列式





6 分支限界法求解图的单源最短路径—优先队列式

采用STL的priority_queue<NodeType>容器作为优先队列（小根堆），优先队列结点类型与前面的相同，添加比较重载函数，即按结点的length成员值越小越优先出队，为此设计NodeType结构体的比较重载函数如下：

```
bool operator<(const NodeType & node) const
{
    return length>node.length;  //length越小越优先出队
}
```

分支限界法求解图的单源最短路径—优先队列式

优先队列式分支限界法求解单源最短路径问题【算法描述】

```
void bfs(int v)                    //求解算法
{  NodeType e,e1;
   priority_queue<NodeType> pqu;   //定义优先队列
   e.vno=v;                        //建立源点结点e
   e.length=0;
   pqu.push(e);                    //源点结点e进队
   dist[v]=0;
   while(!pqu.empty())             //队列不空循环
   {  e=pqu.top(); pqu.pop();      //出队列结点e
      for (int j=0; j<n; j++)
      {  if(a[e.vno][j]<INF && e.length+a[e.vno][j]<dist[j])
         {  //剪枝：e.vno到顶点j有边并且路径长度更短
            dist[j]=e.length+a[e.vno][j];
            prev[j]=e.vno;
            e1.vno=j;              //建立相邻顶点j的结点e1
            e1.length=dist[j];
            pqu.push(e1);          //结点e1进队
}}}}
```

6 分支限界法小结

- **分支限界法与回溯法的主要区别。**
- **分支限界法对问题的解空间树进行搜索的过程**
  - 一次性产生当前扩展结点的所有孩子结点；
  - 在产生的孩子结点中，抛弃那些不可能产生可行解(约束）或最优解的结点（限界）；
  - 将其余的孩子结点加入活结点表；
  - 从活结点表中选择下一个活结点作为新的扩展结点。
  - 重复上述步骤，直到找到一个解或优先队列为空为止。
- **分支限界法的关键问题**
  - 如何确定合适的限界函数。
  - 如何组织待处理活结点表（普通队列和优先队列）。
  - 如何确定最优解中的各个分量。