第6章 分支限界

1. 任务分派问题分析
2. 分支限界一般步骤
3. 装载问题
4. 最大团问题
5. 旅行售货员问题(TSP)
6.批处理作业调度

- 例：分配问题：
- 分配问题：设有n个人，每个人都可以完成n种不同的任务，但所需时间不同。如果只需一人去完成每一项工作，则应如何分配n个人并使完成所有n项工作的总时间为最小。

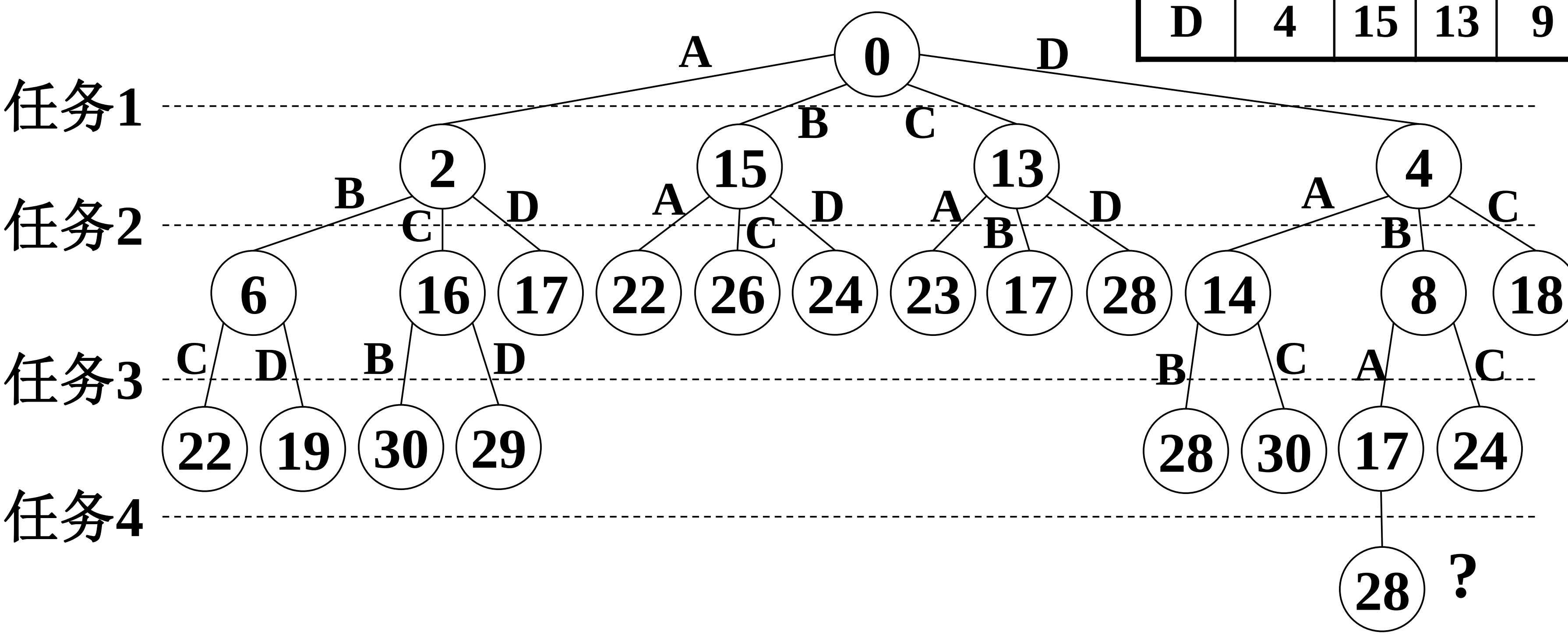
观察: 任务分配问题

右表是不同人完成不同任务所需时间

找出总时间最少的分配方案

任务 / 人	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

任务1:4最少时间: 2, 4, 9, 7



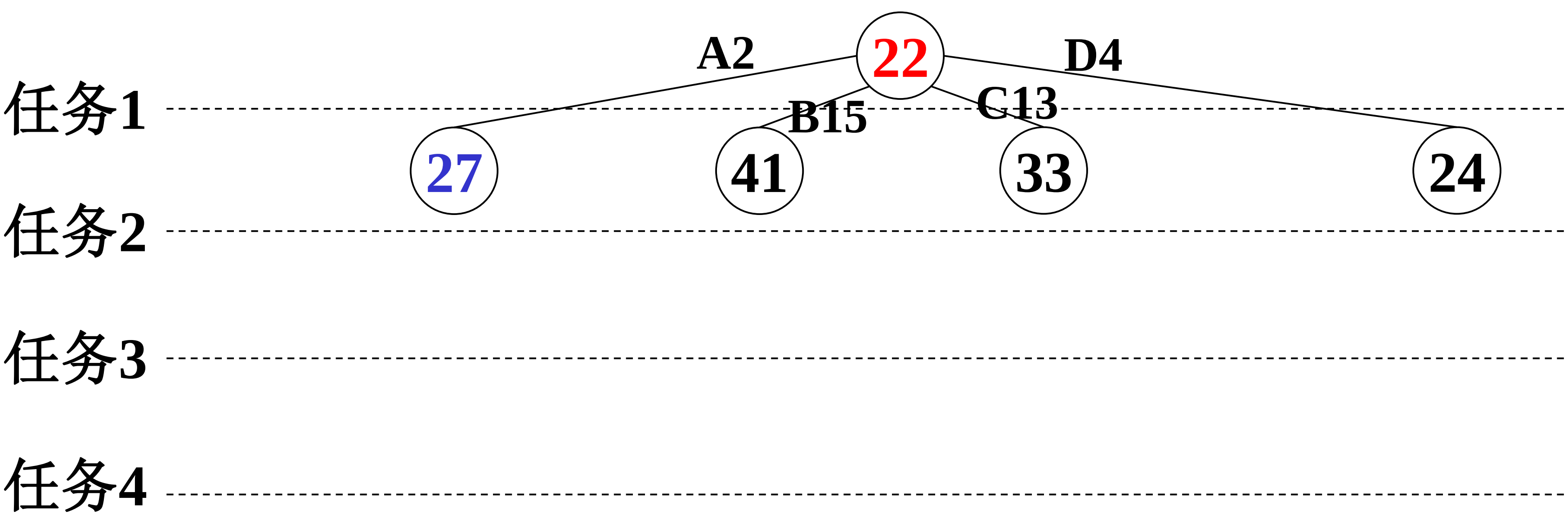
任务分配: 时间下界

以(当前时间+剩余最少时间)为关键值扩展新的节点

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9



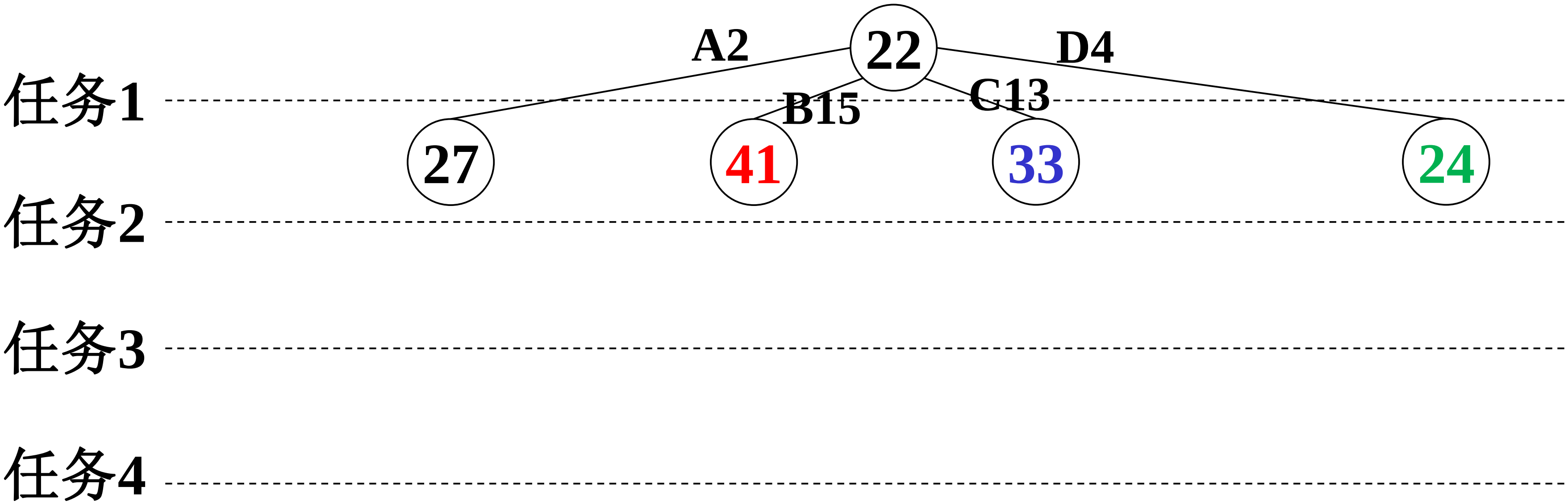
任务分配: 时间下界

以(当前时间+剩余最少时间)为关键值扩展新的节点

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9



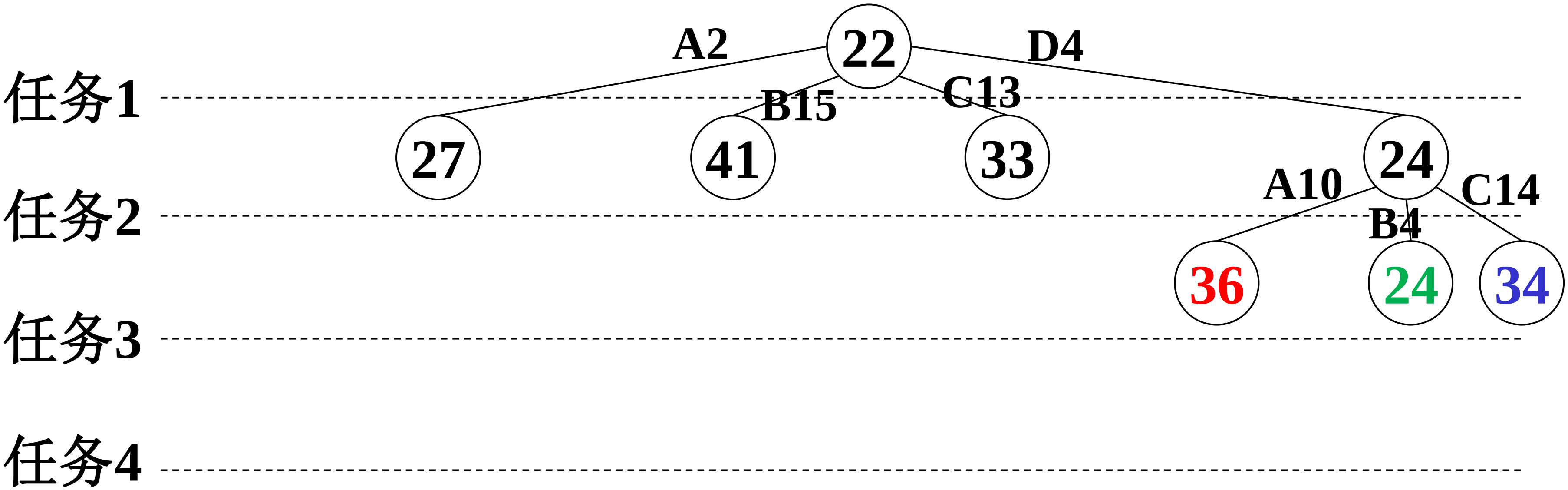
任务分配: 时间下界

以(当前时间+剩余最少时间)为关键值扩展新的节点

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

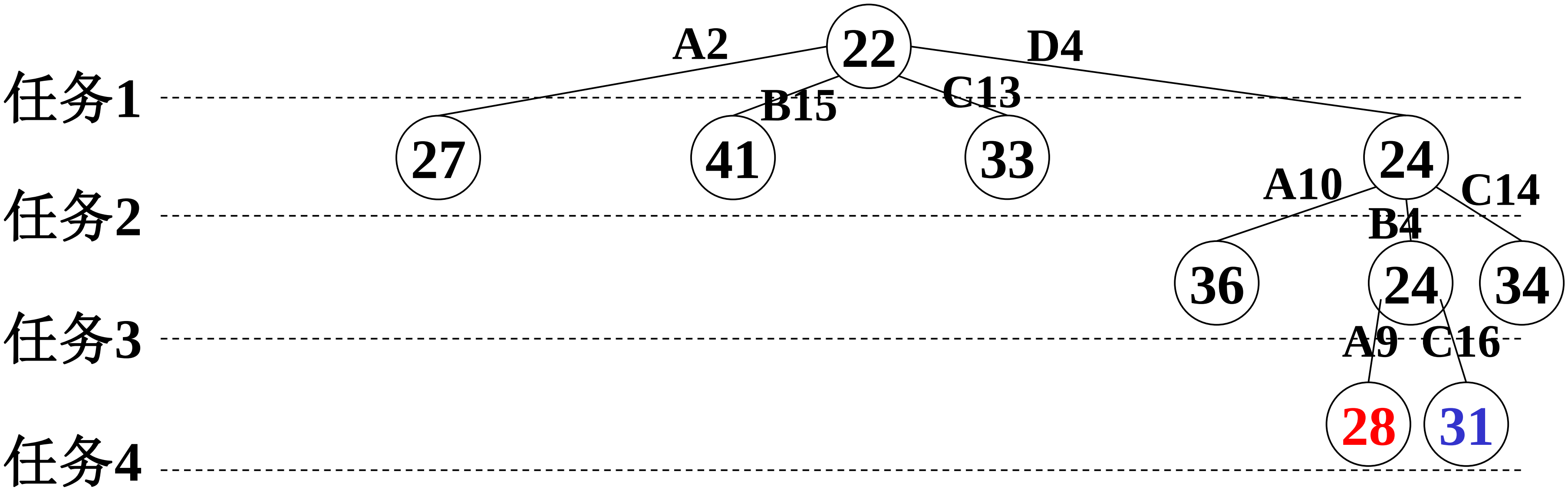


任务分配: 时间下界

以(当前时间+剩余最少时间)为关键值扩展新的节点

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9



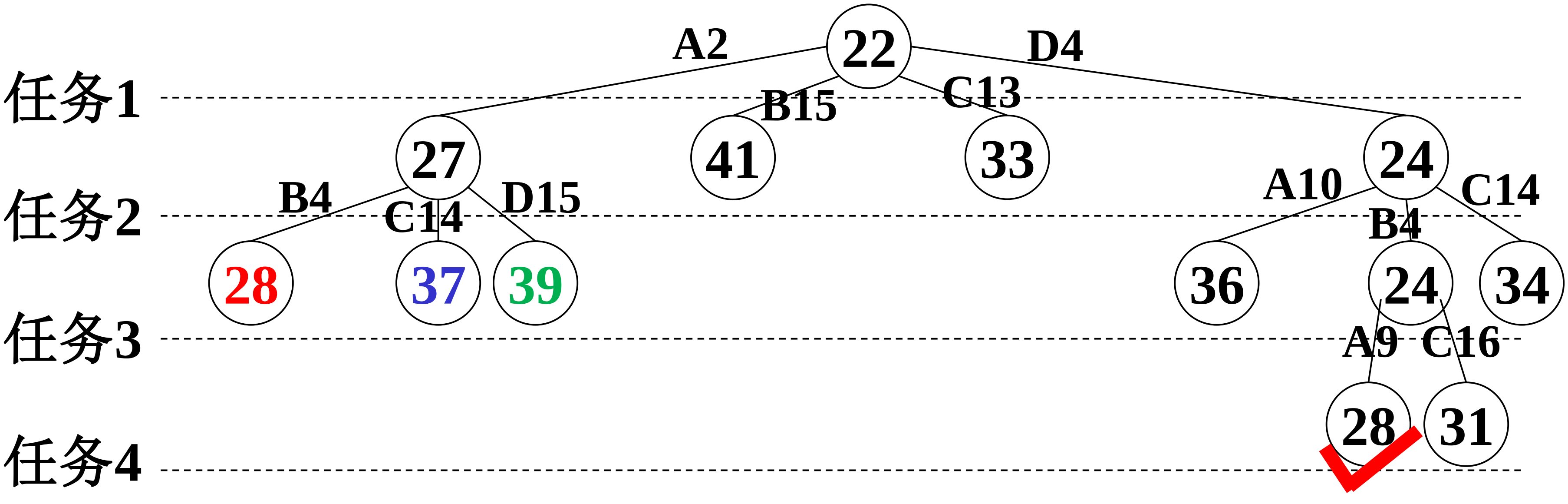
任务分配: 时间下界

以(当前时间+剩余**最少时间**)为关键值扩展新的节点

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9



任务分配: 下界与上界

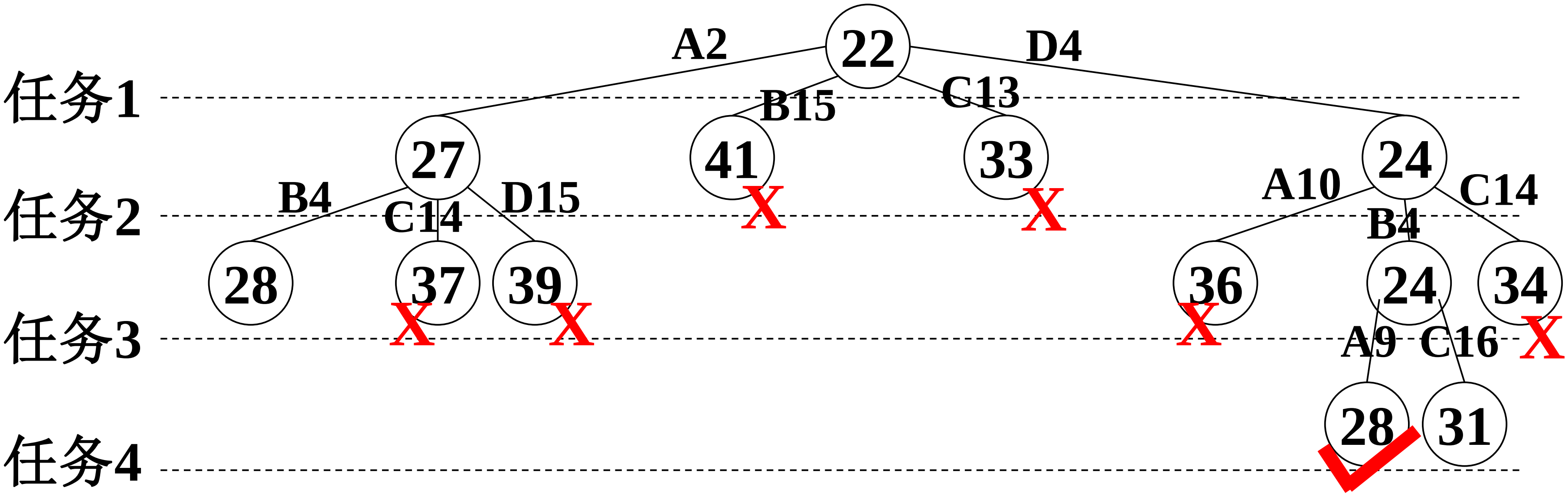
	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9

下界: 当前时间+剩余最少时间

关键值 = 下界

上界: 任意选取 31 = 2+4+16+11

初始 bestt = 上界



第6章 分支限界

1. 任务分派问题分析
2. 分支限界一般步骤
3. 装载问题
4. 最大团问题
5. 旅行售货员问题(TSP)
6.批处理作业调度

分支限界的一般过程

1. 初始根节点是活节点(灰色), 其它节点都是白色

2. while(true)

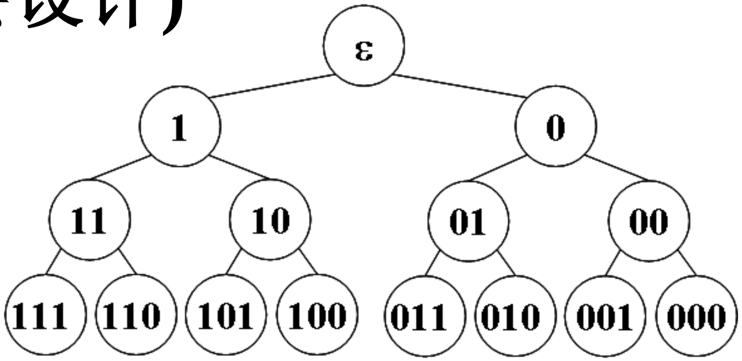
3.　取一个活节点, 设为扩展节点(红色)

4.　将扩展节点的所有孩子设为活节点(灰色),

5.　扩展节点设为死节点(黑色)

取活节点: 先进先出队列 或 优先队列

结束: 队列空 或 到达叶节点(需要设计)



Graph traversal 图遍历

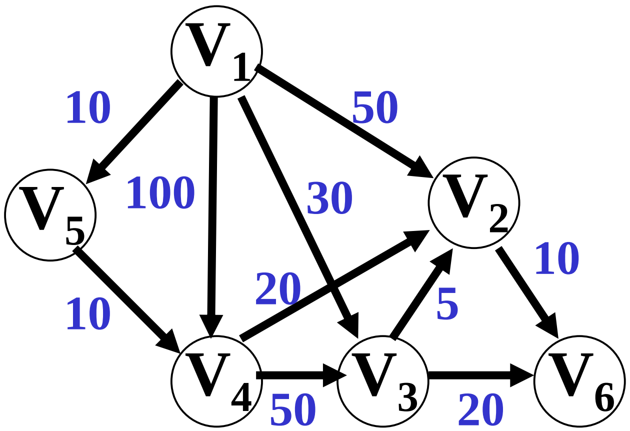
- 有序遍历图的所有边和节点
- 深度优先搜索 **Depth-first search(DFS)**
- 广度优先搜索 **Breadth-first search(BFS)**

Dijkstra: 优先队列或先进先出

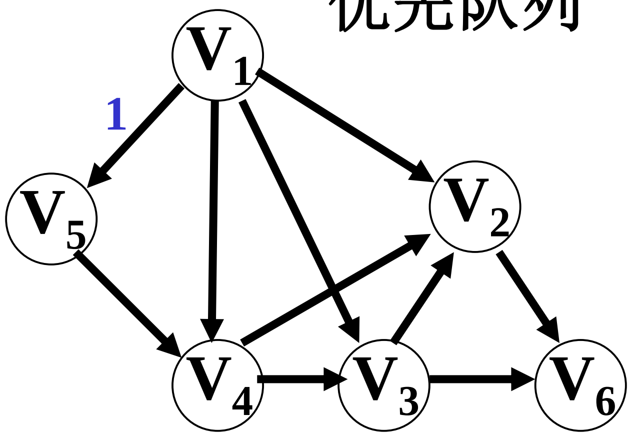
- 输入: G=(V,E,w,s), w权, s起点;
- 输出: δ(s,·)

1. 初始d[s]=0, 其它d[u]=INF,
2. S,Q空, Q.add(s,0),
3. 当Q非空 //Q是优先队列
4. Q.delete(u), 若u∈S, continue(),
5. 将u添加到S中,
6. ∀v∈adj[u], 松弛(u,v),
7. 若d[v]改变, Q.add(v,d[v])

松弛(u,v): 若d[v]>d[u]+w[u,v], 则d[v]=d[u]+w[u,v]



优先队列

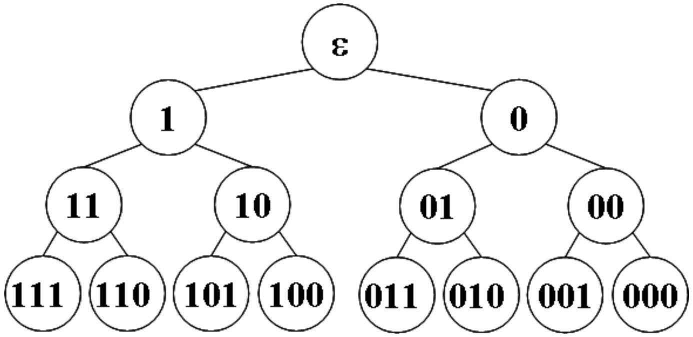


先进先出

第6章 分支限界

1. 任务分派问题分析
2. 分支限界一般步骤
3. 装载问题
4. 最大团问题
5. 旅行售货员问题(TSP)
6.批处理作业调度

回溯: 装载问题w[1:n],c



backtrack(t)

1. 若t>n, 判断 记录 返回

2. r-=w[t]

3. 若 cw + w[t] ≤ c, 则 //左分支, 上界

4. cw+=w[t], backtrack(t+1), cw-=w[t],

5. 若 cw + r > bestw, 则 //右分支, 下界

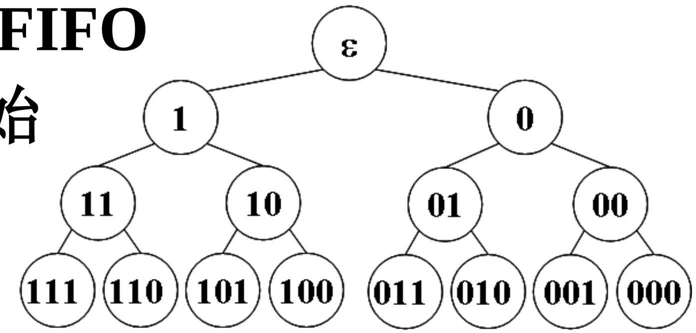
6. backtrack(t+1)

7. r+=w[t]

时间: 回溯 $O(2^n)$, 动态规划 O(nC), 但可能$C=O(2^n)$

空间: 回溯 O(n), 动态规划O(C),

分支限界FIFO: 装载问题w[1:n],c



1. Q.add(-1), //放入层分隔标记, FIFO
2. t=1, cw=0, bestw=0, r=$\text{sum}_{i=2}^{n}$ w[i], //初始
3. while(true),
4. wt = cw + w[t],
5. 若 wt ≤ c, 若wt>bestw, 则 bestw=wt, //更新bestw
6. 若t<n, Q.add(wt) //左分支
7. 若 cw+r>bestw 且 t<n, Q.add(cw), //右分支
8. Q.delete(cw) // 取下一个节点的cw
9. 若cw==-1, //搜完一层的处理
10. 若Q.isempty(), 则返回bestw, //循环出口
11. Q.add(-1), Q.delete(cw), r-=w[++t]

说明: Q.add不是严格记号, 只表示是队列和要存储的内容

问题: 不使用分层标记该怎么做? 记录层号

分支限界FIFO:不使用层分隔标记

1. t=1, cw=0, bestw=0, r=$\text{sum}_{i=2}^{n}$ w[i], //装载问题, 初始

2. while(true)   //FIFO

3. | wt = cw + w[t], r1=r-w[t+1];

4. | 若 wt ≤ c, 若wt>bestw, 则 bestw=wt,   //提前更新

5. | | 若t<n, Q.add(t+1,wt,r1)   //左分支

6. | 若 cw+r>bestw 且 i<n, Q.add(t+1,cw,r1),   //右分支

7. | 若Q空, 返回

8. | Q.delete(t,cw,r)   // 取下一个节点数据

为什么要记录 t, cw, r? 计算最优值需要记住x[1:t]吗?

分支限界

回溯: 以深度优先方式搜索解空间

分支限界: 以广度优先或最小耗费方式搜索解空间

不考虑剪枝, 时间复杂度都是 **O(|E|)**,

空间复杂度差别太大: **O(h) vs O(|E|)**

如果需要搜索整个解空间, 分支限界没有任何优势?

一般用优先队列. 如何使用?

上界, 下界, 关键值. 到达第**n**层就结束. 装载问题?

上界:**c**, 下界:**bestw**, 关键值:当前重量+剩余重量

一旦取到最大关键值的**r=0**(叶节点), 则结束搜索.

维护: 上界, 下界, 关键值, 记录信息

分支限界: 装载问题,优先队列

上界c, 下界bestw, 关键值cw+r, 记录(t,cw,r), 最大堆

1. t=0, cw=0, bestw=0, r=$\text{sum}_{i=1}^{n}$ w[i],　　//初始

2. 当t< n+1　//t=n+1时结束

3. |　wt = cw + w[t], r1=r-w[t];

4. |　若 wt ≤ c, 若wt>bestw, 则 bestw=wt,

5. |　　　　　　Q.add(wt+r1; t+1,wt,r1)

6. |　若 cw+r>bestw, Q.add(cw+r1; t+1,cw,r1),

7. |　Q.delete(; t,cw,r)　//取最大关键值活节点

第6章 分支限界

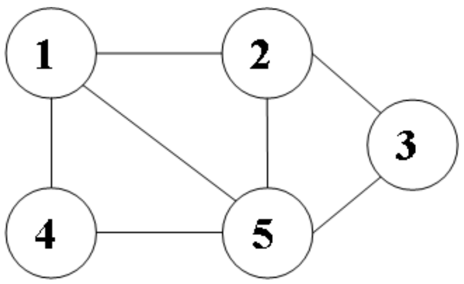
1. 任务分派问题分析
2. 分支限界一般步骤
3. 装载问题
4. 最大团问题
5. 旅行售货员问题(TSP)
6.批处理作业调度

回溯:最大团问题

- 无向图G=(V,E). G的完全子图称为团:

  即$U \subseteq V$满足$\forall u,v \in U$, 都有$(u,v) \in E$

- 最大团: 顶点数最多的团.
- 解空间结构: 子集树



bestn:

目前最大团顶点数

cn:

当前团顶点数

backtrack(t)

1. 若 **t > n**, 则判断,记录**bestn,x**, 返回

2. 若**v_t与x[1:t-1]中已有的cn个点都相连**, 则

3. **x[t]=1, cn++, backtrack(t+1), cn--, x[t]=0**

4. 若**cn+n-t ≥ bestn**, 则

5. **x[t]=0, backtrace(t+1)**

分支限界:最大团问题

上界无, 下界bestn, 关键值cn+n-t, 记录(t,cn,x)

1. t=1, cn=0, bestn=0, x[1:n]=0,　　//初始

2. 当 t < n+1　　　　　　//当t=n+1时搜索结束

3. |　若vt与x[1:t-1]中已有的cn个点都相连, 则,

4. |　|　x[t]=1, Q.add(cn+n-t+1; t+1,cn+1,x), x[t]=0

5. |　|　若cn+1>bestn, bestn=cn+1 //提前更新最优值

6. |　若cn+n-t ≥ bestn, 则Q.add(cn+n-t; t+1,cn,x),

7. |　Q.delete(; t,cn,x)　　　//取最大关键值活节点

分支限界:任务分派

上界: 31=2+4+16+9, 初始最优值bestt

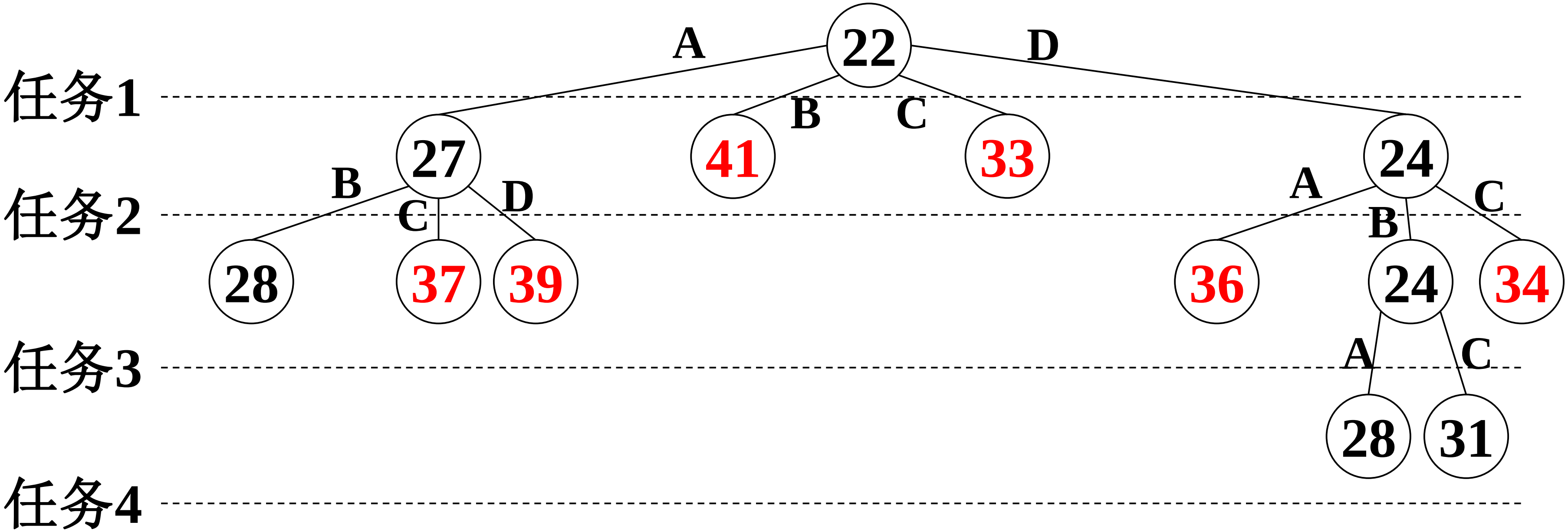
下界 = 关键值 = ct + r(当前+剩余最少时间)

记录(当前层号t, 当前时间ct, 当前选择x)

当有关键值相同时, 取层号大的优先

r设计越精确, 计算剪枝函数时间花销越多

任务 / 人	1	2	3	4
A	2	10	9	7
B	15	4	14	8
C	13	14	16	11
D	4	15	13	9



比较回溯法与分支限界法

剪枝函数可以共用

都可以及时更新最优值

回溯优点: 存储少

分支限界优点: 可以自由选择活节点作为扩展节点

回溯可以用来搜索所有最优解

分支限界可以用来搜索一个最优解

第6章 分支限界

1. 任务分派问题分析
2. 分支限界一般步骤
3. 装载问题
4. 最大团问题
5. 旅行售货员问题(TSP)
6. 批处理作业调度

旅行售货员问题(TSP)

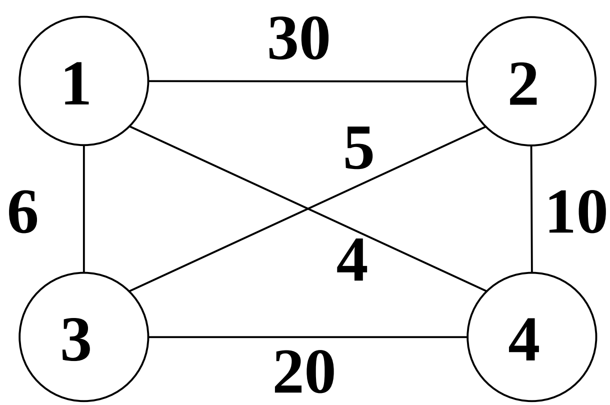
某售货员要到若干城市推销商品

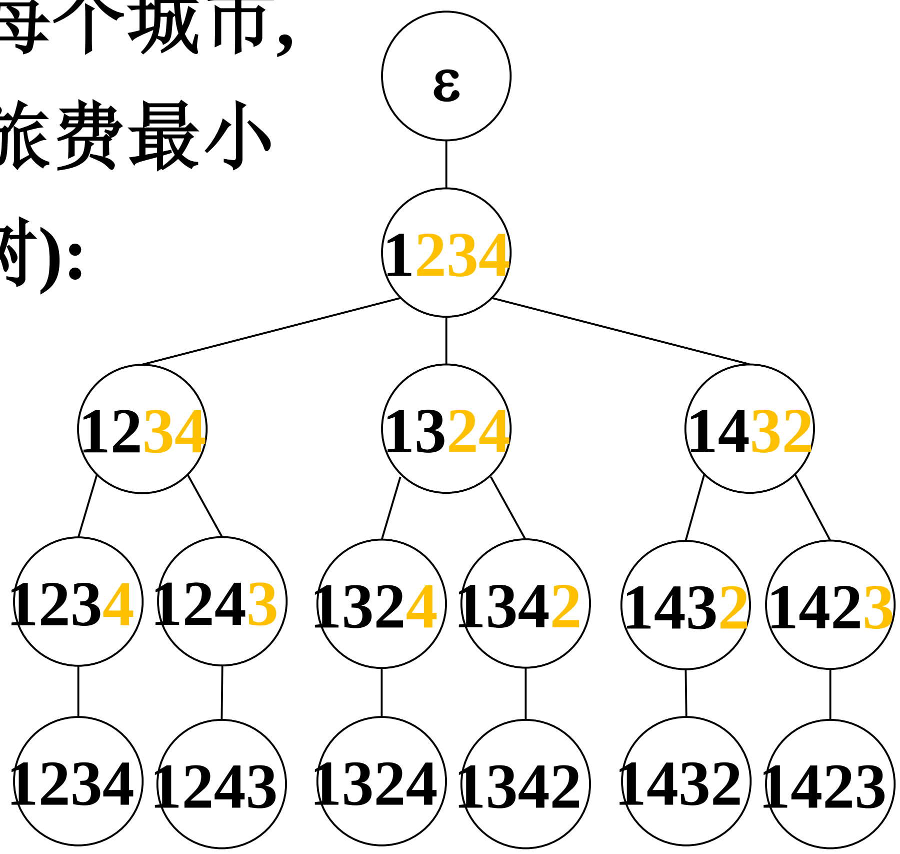
已知各城市间的旅费

选择一条从驻地出发, 经过每个城市,

最后回到驻地的路线, 使总旅费最小

解空间和解空间结构(排列树):





回溯: TSP

backtrack(t)

1. 若 t > n, 则判断(记录bestc, bestx), 返回

2. 对 j = t : n

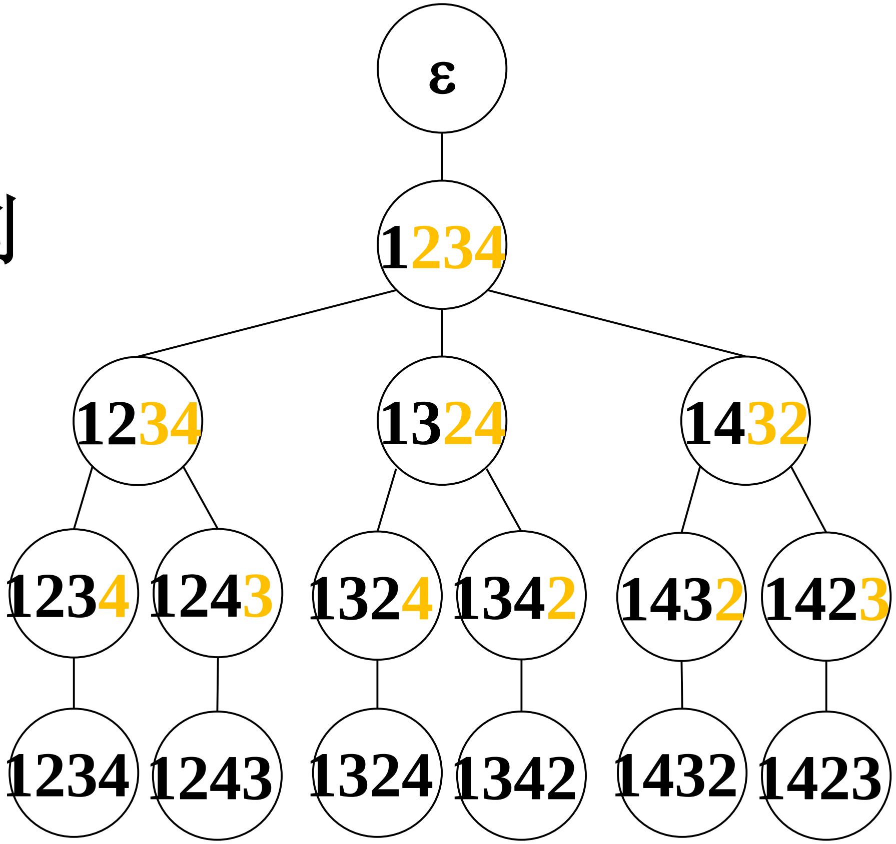
3.   交换x[t],x[j],

4.   若x[1:t]费用<bestc, 则

5.   backtrace(t+1)

6.   交换x[t],x[j]

- **初始x[i]=i, bestc=INF,**
- **backtrack(2)**



TSP分支限界设计

设目前在第t层

x[0:t]: 已经选定的x[0]到x[t]的路径

x[t+1:n]: 剩余顶点

cc: x[0:t]的费用

bestc: 目前最佳费用 (上界)

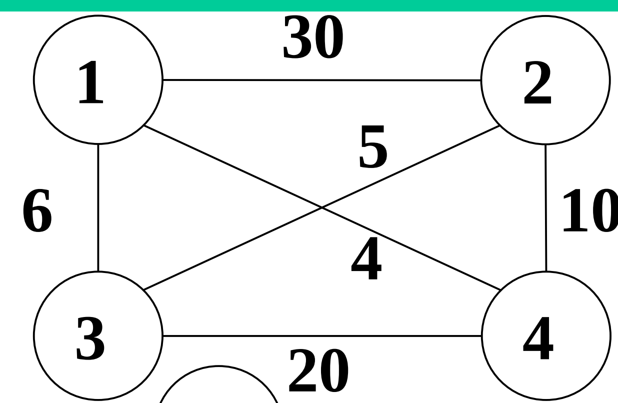
minout[t]: 顶点t的出边的最小权

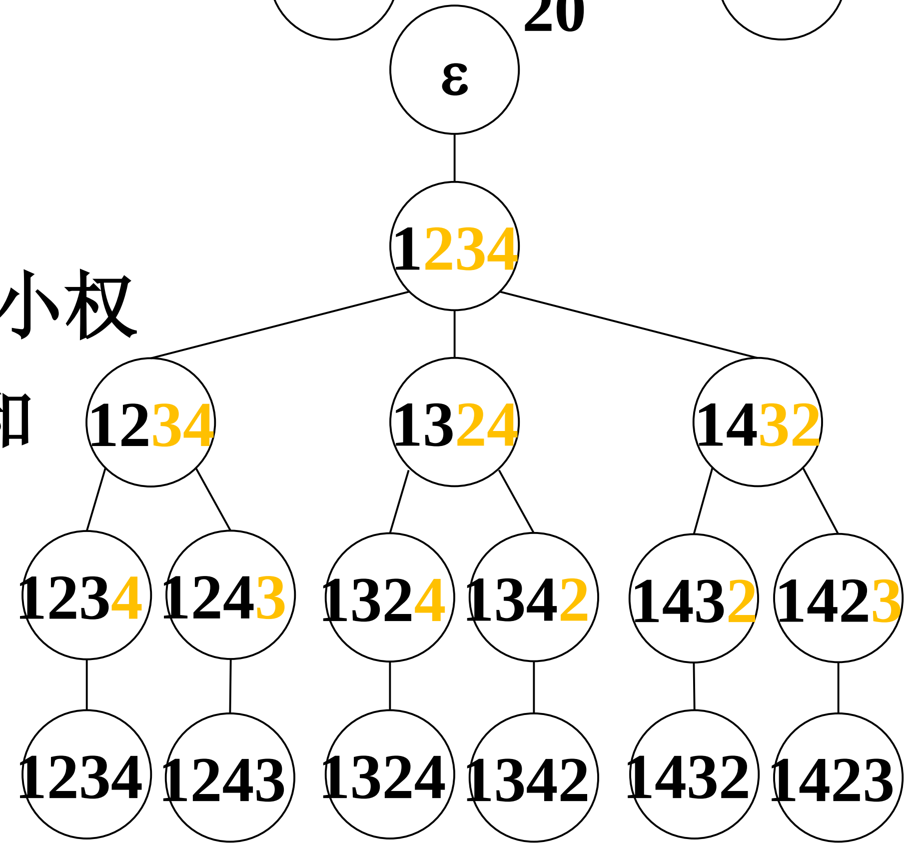
rcost = x[t:n]的minout[t]之和

关键值: cc + rcost

下界 = 关键值

记录(t, cc, rcost, x)





分支限界: TSP

1. t=0,cc=0,bestc=INF, rcost=$\text{sum}_{k=1}^{n-1}$ minout[k], x[0]=1

2. 当 t < n-1, //当i=n-1时搜索结束，优先队列，最小堆

3. | 若 t = n-2 且可行(能回到1),

4. | | 更新bestc, Q.add(cc; t+1,cc,0,x), continue()

5. | 对 j = t : n-1,

6. | | 交换x[t],x[j]

7. | | 若x[t-1],x[t]无边continue; 计算cc, rcost

8. | | 若cc+rcost≤bestc, 则Q.add(cc+rcost; t+1, cc,rcost,x)

9. | | 交换x[t],x[j]

10. | | Q.delete(; t,cc,rcost,x) //取最小关键值活节点

TSP程序

```
template<class Type>
Class MinHeapNode  {
   friend Traveling <Type>;
   public:
        operator Type ( ) const {return lcost;}
   private:
      Type lcost,           //子树费用的下界
               cc,          //当前费用
               rcost;       //x[t:n-1]中顶点最小出边费用和
       int   t,             //根节点到当前节点的路径为x[0:t]
};
```

TSP程序: 最小出边费用计算

```
template<class Type>
Type Traveling <Type>::BBTSP(int v[])
{   //解TSP问题的优先队列式分支限界法
    //定义最小堆容量为1000
    MinHeap<MinHeapNode<Type>>H(1000);
    Type *MinOut = new Type[n+1];
    //计算MinOut[i]=顶点i的最小出边费用
    Type MinSum = 0;          //最小出边费用和
    for(int i=1; i<=n;i++) {
        Type Min = NoEdge;
        for(int j=1;j<=n;j++)
            if(a[i][j]!=NoEdge&&(a[i][j]<Min||Min==NoEdge))Min=a[i][j]
        if(Min==NoEdge)return NoEdge;  //无回路
        MinOut[i]=Min; MinSum+=Min;
    } /*for*/  …
```

TSP程序: 初始化

```
template<class Type>
Type Traveling <Type>::BBTSP(int v[])
{  …  //初始化
    MinHeapNode <Type> E;
    E.x=new int[n];
    for(int i=1;i<n;i++)E.x[i]=i+1;
    E.t=0; E.cc=0; E.rcost=MinSum;
    Type bestc=NoEdge;    …
```

//初始: t=0, x[0]=1, x[1:n-1]=(2:n), cc=0,

//   bestc=INF,   rcost=$\text{sum}_{k=1}^{n-1}$ minout[k]

TSP程序: 处理叶子的父结点

2. 当 t < n-1,
   //当i=n-1时搜索结束
3. 若 t = n-2且可行(能回1),
4. 则 更新bestc,
   Q.add(cc; n-1,cc,0,x),
   continue()

```
template<class Type>
Type Traveling <Type>::BBTSP(int v[])
{  … //搜索排列空间树
   while(E.t<n-1){  //非叶结点
     if(E.t==n-2){  //当前扩展结点是叶结点的父结点
     //加两条边形成回路, 回路是否优于当前最优解
      if( a[E.x[n-2]][E.x[n-1]] != NoEdge &&
          a[E.x[n-1]][1] != NoEdge &&
          (E.cc+a[E.x[n-2]][E.x[n-1]]+a[E.x[n-1]][1]<bestc || bestc=NoEdge) )
       {  //费用更小的回路
          bestc = E.cc+a[E.x[n-2]][E.x[n-1]]+a[E.x[n-1]][1];
          E.cc=bestc; E.lcost=bestc; E.t++; H.Insert(E);}
       else delete [] E.x; //舍弃扩展结点
     else { //产生当前扩展结点的儿子结点 …
```

TSP程序: 扩展当前结点

```
template<class Type>
Type Traveling <Type>::BBTSP(int v[])
{  ... //搜索排列空间树
   else { //产生当前扩展结点的儿子结点
     for (int i=E.t+1;i<n;i++)
     if ( a[E.x[E.t]][E.x[i]]!=NoEdge){ //可行
       Type cc=E.cc+a[E.x[E.t]][E.x[i]];
       Type rcost=E.rcost-MinOut[E.x[E.t]];
       Type b=cc+rcost;  //下界
       if ( b<bestc || bestc==NoEdge ) { //限界条件, 插入最小堆
         MinHeapNode <Type> N; N.x = new int [n];
         for(int j=0;j<n;j++) N.x[j]=E.x[j];
         N.x[E.t+1]=E.t[i]; N.x[i]=E.x[E.t+1];
         N.cc=cc; N.t=E.t+1; N.lcost=b; N.rcost=rcost; H.Insert(N);}
     } delete [ ] E.x} //else //完成当前结点扩展...
```

6. 对 j = t : n-1,
7. 交换x[t],x[j];
   若不可行continue;
   计算cc, rcost;
8. 若cc+rcost≤bestc,
   则Q.add(cc+rcost; t+1, cc,rcost,x)
9. 交换x[t],x[j]

TSP程序: 优先队列式分支限界

```
template<class Type>
Type Traveling <Type>::BBTSP(int v[])
{    …
      } delete [ ] E.x} //else  //完成当前结点扩展
    try { H.DeleteMin(E);} //取下一扩展结点
    catch(OutOfBounds){break;} //空堆
    } //while结束
  if (bestc==NoEdge) return NoEdge; //无回路
  for(int i=0;i<n;i++)v[i+1]=E.x[i]; //取最优解
  while(true) { //释放最小堆中所有结点
     delete [] E.x;
     try { H.DeleteMin(E);}
     catch(OutOfBounds){break;}
     }
  return bestc; }// Traveling结束
```

10. Q.delete(; t,cc,rcost,x)
//取最大关键值活节点

第6章 分支限界

1. 任务分派问题分析
2. 分支限界一般步骤
3. 装载问题
4. 最大团问题
5. 旅行售货员问题(TSP)
6.批处理作业调度

批处理作业调度

✓给定n个作业的集合$\{J_1, J_2, \ldots, J_n\}$。每个作业必须先由机器1处理，然后由机器2处理。作业J_i需要机器j的处理时间为t_{ji}。

✓对于一个确定的作业调度，设F_{ji}是作业i在机器j上完成处理的时间。所有作业在机器2上完成处理的时间和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。

批处理作业调度

t_{ji}	机器1	机器2
作业1	2	1
作业2	3	1
作业3	2	3

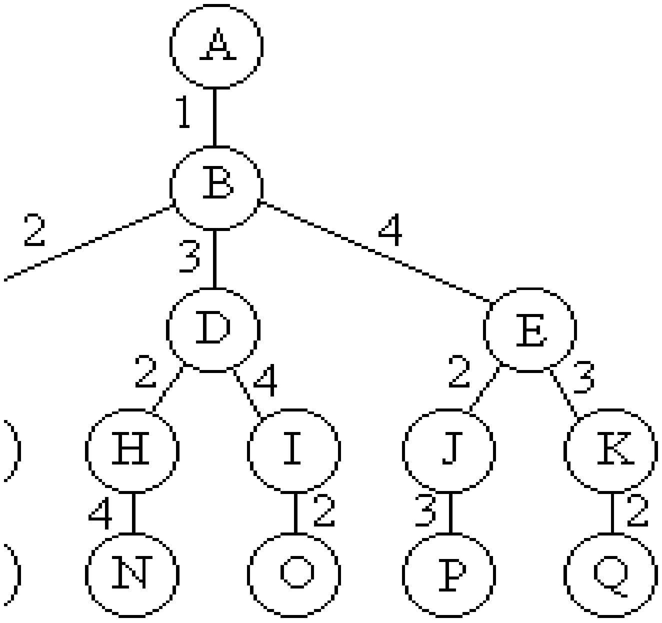
这**3**个作业的**6**种可能的调度方案是**1,2,3；1,3,2；2,1,3；2,3,1；3,1,2；3,2,1；**

它们所相应的完成时间和分别是**19，18，20，21，19，19**。

最佳调度方案是**1,3,2**，其完成时间和为**18**。

批处理作业调度

- 解空间：排列树




```
class Flowshop {
 friend Flow(int**, int, int []);
void Backtrack(int i);
  int  **M,//作业所需处理时间
  *x,     // 当前作业调度
  *bestx, //当前最优作业调度
  *f2,    // 机器2完成处理时间
  f1,    // 机器1完成处理时间
  f,     // 完成时间和
  bestf,    // 当前最优值
  n;   // 作业数};
```

批处理作业调度

```
void Flowshop::Backtrack(int i)
{
   if (i > n) {
      for (int j = 1; j <= n; j++)
        bestx[j] = x[j];
        bestf = f;
      }
   else
```

```
    for (int j = i; j <= n; j++) {
        f1+=M[x[j]][1];
        f2[i]=((f2[i-1]>f1)?f2[i-1]:f1)+M[x[j]][2];
        f+=f2[i];
        if (f < bestf) {
            Swap(x[i], x[j]);
            Backtrack(i+1);
            Swap(x[i], x[j]);
          }
          f1- =M[x[j]][1];
          f- =f2[i];
     }
}
```

6.9 批处理作业调度问题

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。

6.9 批处理作业调度问题

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t_{1pk}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t_{2pk}依非减序排列，则S2取得极小值。

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

这可以作为优先队列式分支限界法中的限界函数。

6.9 批处理作业调度问题

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。

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。

6.9 批处理作业调度问题

3. 算法描述

当E. sf2<bestc时，更新当前最优值bestc和相应的最优解bestx

```
      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;}
```

6.9 批处理作业调度问题

3. 算法描述

```
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
```

当bb<bestc时，将儿子结点插入到活结点优先队列中

本章作业

在解最大团问题的优先队列式分支限界法中, 当前扩展节点满足cn+n-t≥bestn的右儿子节点被插入到优先队列中. 如果将这个条件改为满足cn+n-t>bestn右儿子节点插入优先队列仍能满足算法正确性吗? 为什么?

本章小节

1. 任务分派问题分析
2. 分支限界一般步骤
3. 装载问题
4. 最大团问题
5. 旅行售货员问题(TSP)