第六章分支限界法

本章主要知识点

- 6.1　分支限界法的基本思想
- 6.2　装载问题
- 6.3　布线问题
- 6.4　0-1背包问题
- 6.5　最大团问题
- 6.6　旅行售货员问题
- 6.7　小结

6.1 分支限界法的基本思想

1. 分支限界法与回溯法的不同

（1）**求解目标**：

回溯法求解目标是找出解空间树中满足约束条件**所有解**

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**一个最优解**）

（2）**搜索方式的不同**：

回溯法以深度优先方式搜索解空间树

分支限界法则以广度优先或以最小耗费优先方式搜索解空间树

6.1 分支限界法的基本思想

2. 分支限界法基本思想

（1）让根结点成为当前扩展结点。

（2）当前扩展结点一次性产生其所有儿子结点。

（3）舍弃导致不可行解或非最优解的儿子结点，其余儿子结点加入**活结点表**

（4）从活结点表中取下一结点成为当前扩展结点，

（5）如果找到所需的解或活结点表为空，算法结束，否则重复步骤(2)(3)(4)。

从活结点表中取下一个扩展结点：广度优先、最小耗费（最大效益）优先

6.1 分支限界法的基本思想

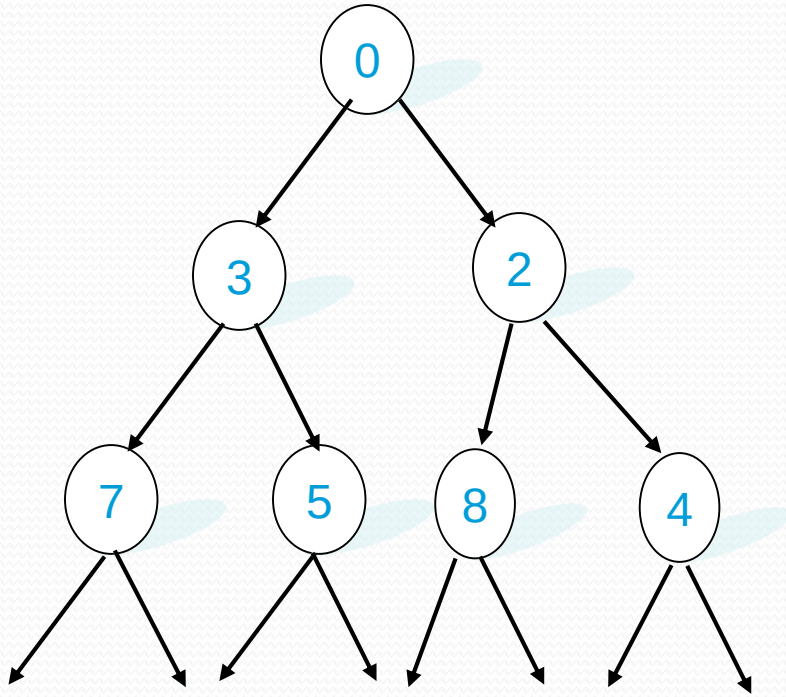
广度优先和最小耗费优先的区别

③	②	⑦	⑤	⑧	④	

广度优先的活结点表(队列）

②	③	④	⑤	⑦	⑧	

最小耗费优先的活结点表（优先队列）



6.1 分支限界法的基本思想

3. 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）原则选取下一个结点为扩展结点。

（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结点成为当前扩展结点。（优先队列的数据结构？）

6.2 装载问题

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 $\sum_{i=1}^{n} w_i \le c_1 + c_2$

采用下面的策略可得到最优装载方案。

(1) 将第一艘轮船尽可能装满；

(2) 将剩余集装箱装上第二艘轮船；

6.2 装载问题

1. 队列式分支限界法

（1）检测当前扩展结点的左儿子结点，可行入队

（2）将其右儿子结点加入队列(右儿子结点一定可行)

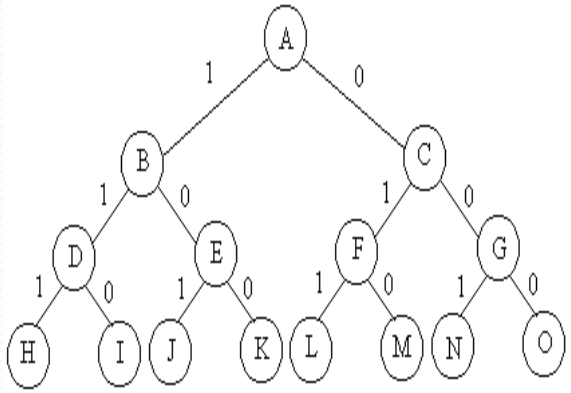
（3）舍弃当前扩展结点。

（4）每层结点之后都加一个尾部标记-1，将活结点分层

只考虑求最大载重量，不考虑构造最优解

船载重量：C=10

集装箱：W1=3，W2=4，W3=5



				7	3	4	0	-1	i=3
				12 | 7	8 | 3	9 | 4	5 | 0		i=3

bestw=0
bestw=7
bestw=8
bestw=9

思考：

（1）以Ew为已在船集装箱重量，w[i]为第i个集装箱的重量，

在考虑装入第i个集装箱时， 左分支约束条件是什么？

（2）最优值bestw 是如何求得的？

（3）叶子结点是否要进入队列（活结点表）？

（4）算法在什么条件下结束？

装载问题队列式分支限界法算法

```
Ew=0, Q.Add(-1) ;i=1; bestw=0;
//Ew为当前结点对应的在船的集装箱重量
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
}
```

```
void EnQueue(Queue<Type> &Q, Type wt,
Type &bestw,  int i,  int n)
 {  if(i==n){
            if(wt>bestw)bestw=wt;
            }//叶子结点不入队列
    else Q.add(wt);
}
```

算法中右分支不剪枝，效率差

如何改进？

6.2 装载问题

2. 算法的改进（右子树加入剪枝条件）

- 策略：当Ew+r≤bestw时，可将其右子树剪去。

  bestw:当前最优解；Ew:当前扩展结点所相应重量； r:剩余集装箱的重量和

- 算法每一次进入左子树的时候更新bestw的值，确保右子树有效剪枝，不要等待i=n时才去更新。

2. 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;
      if (i < n) Q.Add(wt); // 加入活结点队列
    }
// 检查右儿子结点
    if (Ew + r > bestw && i < n)
      Q.Add(Ew);     // 可能含最优解
```

提前更新bestw

右子树剪枝

入队列函数 EnQueue( ) 就可以去掉了

6.2 装载问题

3. 构造最优解

结点信息？

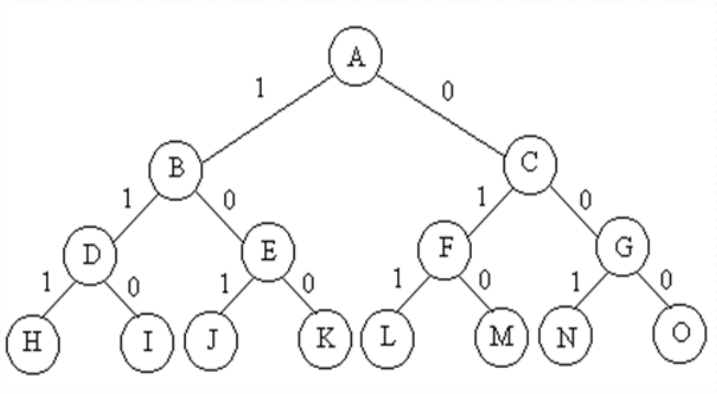
```
class QNode
 { QNode *parent;  // 指向父结点的指针
    bool LChild;        // 左儿子标志
    Type weight;  // 结点所相应的载重量
}
```

思考： **可不可以把路径信息直接存放在结点中？**

6.2 装载问题

记住最优值结点，根据parent可回溯到根节点，找到最优解

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



C=10, W1=3, W2=4, W3=5

解空间树中L结点构造出最优值9

根据构造最优解的算法，最优解bestx[1..3]={0,1,1}

6.2 装载问题

4. 优先队列式分支限界法

- 活结点x的优先级：**根到结点x的路径相应的载重量+剩余集装箱重量之和**
- 优先队列中优先级最高的活结点成为下一个扩展结点。
- 用最大优先队列存储活结点表（大顶堆）
- **以结点x为根的子树中所有结点相应的路径的载重量不会超过x的优先级。**

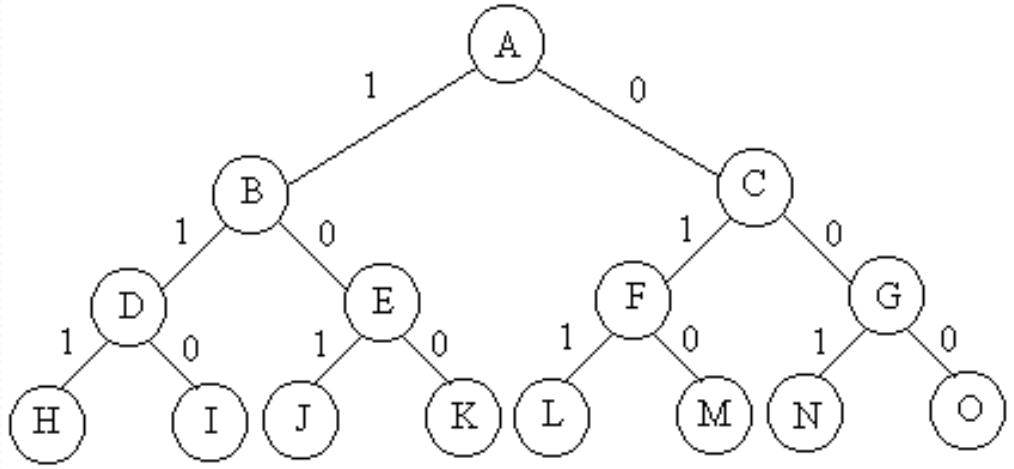
  叶结点所相应的载重量与其优先级相同。
- 一旦优先队列中有一个叶结点成为当前扩展结点，则可以断言该叶结点所相应的载重量即为最优值。此时即可终止算法的搜索过程。

（注意：算法中扩展出的叶子结点要进队列）

6.2 装载问题

模拟优先队列（堆）

n=3, C=10，W1=3，W2=4，W3=5



3	0				载重				
1	0				路径				
12 2	9 2				优先级 下一层				

7	0	3			载重				
11	0	10			路径				
12 3	9 2	8 3			优先级 下一层				

0	3	7		载重					
0	10	110		路径					
9 2	8 3	7 4		优先级 下一层					

4	3	7		载重					
01	10	110		路径					
9 3	8 3	7 4		优先级 下一层					

9	3	7	载重	
011	10	110	路径	
9 4	8 3	7 4	优先级 下一层	

```
Templete<class Type >
Type MaxLoading(Type w[], Type c, int n, int bestx[]) { //优先队列分支限界法，返回最大装载量Ew，最优解bestx
MaxHeap<HeapNode<Type>> H(1000);  //定义最大堆的容量为1000
Type  *r=new Type [n+1];
r[n]=0;
for(int j=n-1;j>0;j--)  r[j]=r[j+1]+w[j+1];
int i=1;
bbnode *E=0;
Type Ew=0;
while (i!=n+1){   //搜索子集树
      if (Ew+w[i] <= c)  AddLiveNode(H, E, Ew+w[i]+r[i], true,  i+1);
      AddLiveNode(H, E, Ew+r[i],  false,  i+1);
      HeapNode<Type> N;
      H.DeleteMax(N);
      i=N.level;
      E=N.ptr;
      Ew=N.uweight-r[i-1];
 }
for（int j=n;j>0;j--）{//构造最优解
        bestx[j]=E->lchild;  E=E->parent;}
return Ew;
}
```

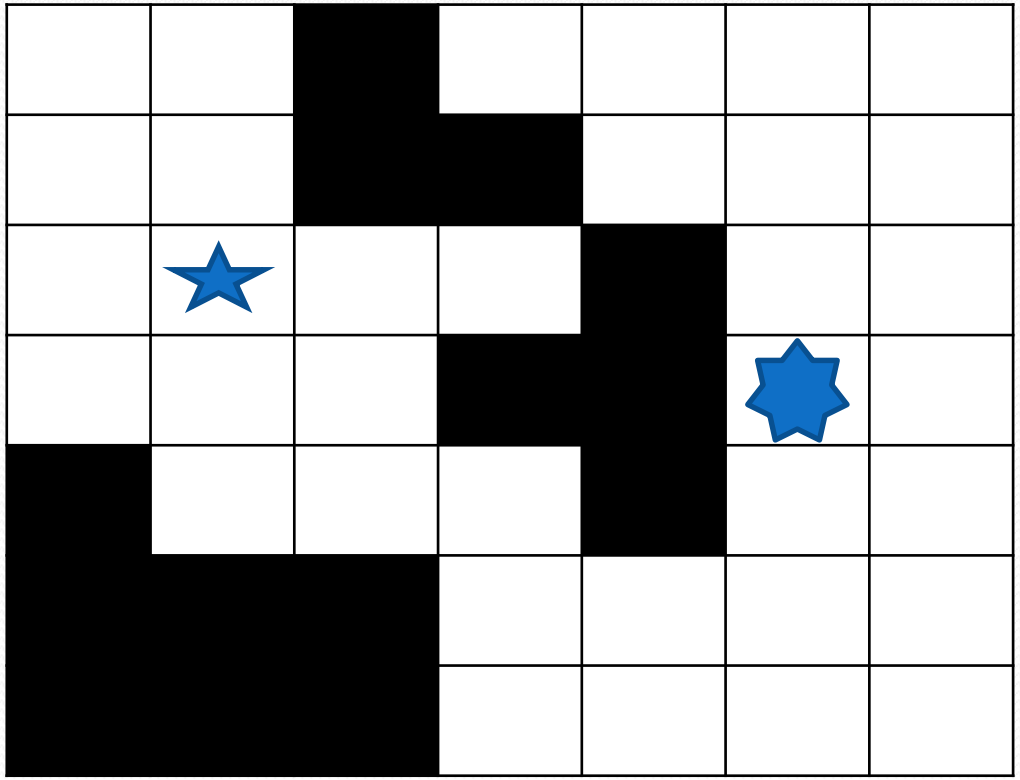
总结

- 1、回溯法与分支限界法的区别：求解目标、搜索方式
- 2、分支限界法的基本思想：　　五个步骤
- 3、分支限界法的分类：　　队列与优先队列，同活结点表有关
- 4、装载问题的分支限界法：　　队列式与优先队列式，重点内容包括算法设计，剪枝条件，算法结束条件，优先队列式的优先级的定义,构造最优解。

THE END

6.3 布线问题

	1	
1	0	1
	1	



-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	3	2	-1	∞	∞	∞	∞	-1
-1	2	1	-1	-1	∞	∞	∞	-1
-1	1	0	1	2	-1	∞	∞	-1
-1	2	1	2	=1	-1	9	∞	-1
-1	-1	2	3	4	-1	8	∞	-1
-1	-1	-1	-1	5	6	7	8	-1
-1	-1	-1	-1	6	7	8	∞	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

6.3 布线问题

算法的思想: 队列式分治限界法

行号	3	2	3	4	3	
列号	2	2	3	2	1	
路径长度	0	1	1	1	1	

剪枝策略：新扩展的该位置路径长度小于该位置已记录的值

位置偏移量

```
Position offset[4];
   offset[0].row = -1; offset[0].col = 0; // 上
   offset[1].row = 1; offset[1].col = 0; // 下
   offset[2].row = 0; offset[2].col = -1; // 左
   offset[3].row = 0; offset[3].col = 1; // 右
```

边界设置

```
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
```

6.3 布线问题

```
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] > grid[here.row][here.col] +1)
            {    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);
            }
  }
```

6.4 0-1背包问题

- **算法的思想**

- 先进行预处理：将各物品依其单位重量价值从大到小排列。
- 优先队列的优先级：已装物品价值+后面物品装满剩余容量的价值
- 算法：

（1）先检查当前扩展结点的左儿子结点。如果该左儿子结点是可行结点，则将它加入活结点优先队列中，如优于当前最优值，则更新当前最优值。

（2）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（优先级大于当前最优值）才将它加入活结点优先队列。

（3）从优先队列中取下一个活结点成为扩展结点，继续扩展。

（4）当叶节点为扩展结点时即为问题的最优值，算法结束。

6.4 0-1背包问题

b=背包已有物品价值；cleft=背包剩余容量，从第i~n物品为剩余的物品，用剩余物品装满剩余背包容量的背包的贪心算法（主要代码如下）

```
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为上界函数值
```

上界函数bound（int i）的计算---贪心算法中的背包问题

6.4 0-1背包问题

- **优先队列中的结点信息包含：**

  当前背包价值、重量；

  结点的优先级（价值上界）；

  左孩子标志；

  父结点地址；

  本结点对应的下一个要装入背包的物品编号；

6.4 0-1背包问题

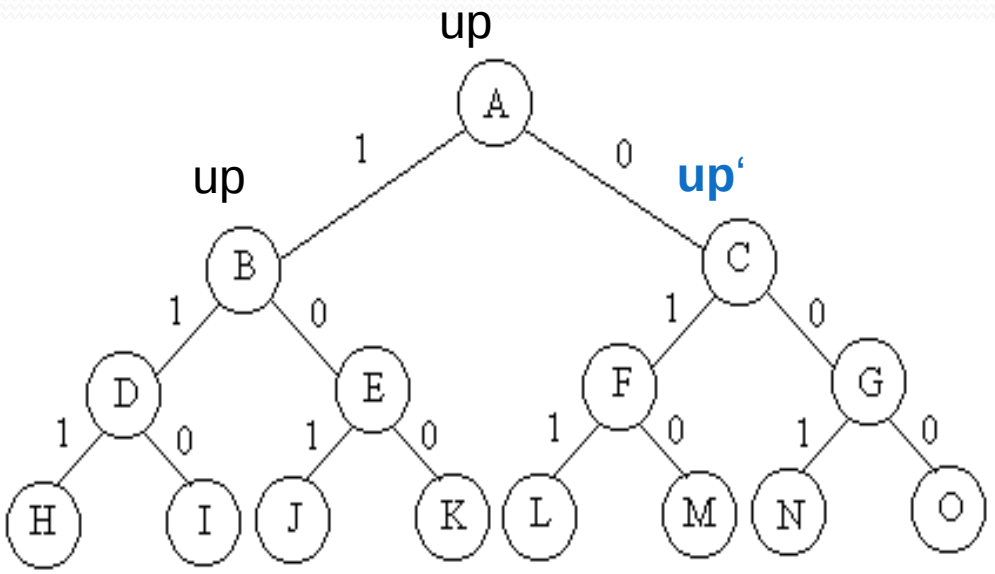
搜索算法的主代码

```
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）；

}
```

注意优先级up的变化规律



6.4 0-1背包问题

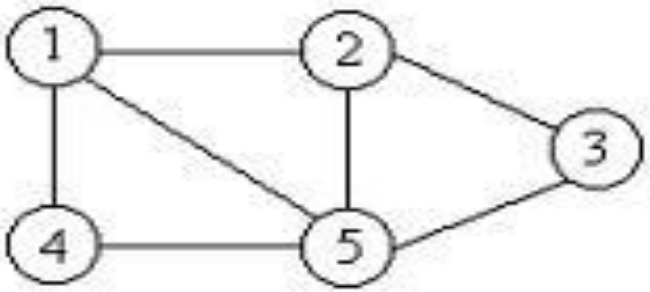
思考题：

如果优先级设定为当前为止的背包内物品价值，算法是否可以在一个叶子结点成为一个扩展结点时算法结束？

6.5 最大团问题

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。
- 子集{1，2}是G的大小为2的完全子图。但不是团，因为它被G的更大的完全子图{1，2，5}包含。{1，2，5}、{1，4，5}、{2，3，5}是G的最大团。

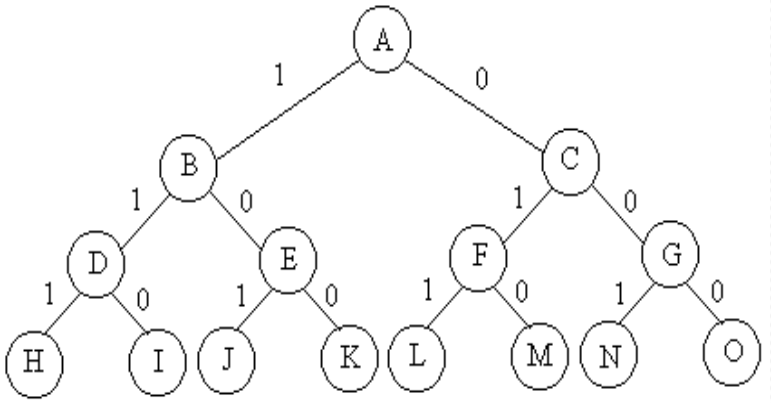


6.5 最大团问题

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 +剩余定点数作为顶点数上界upperSize的值 它是优先队列中元素的优先级。

- **解空间树：**



6.5 最大团问题

3. 算法思想

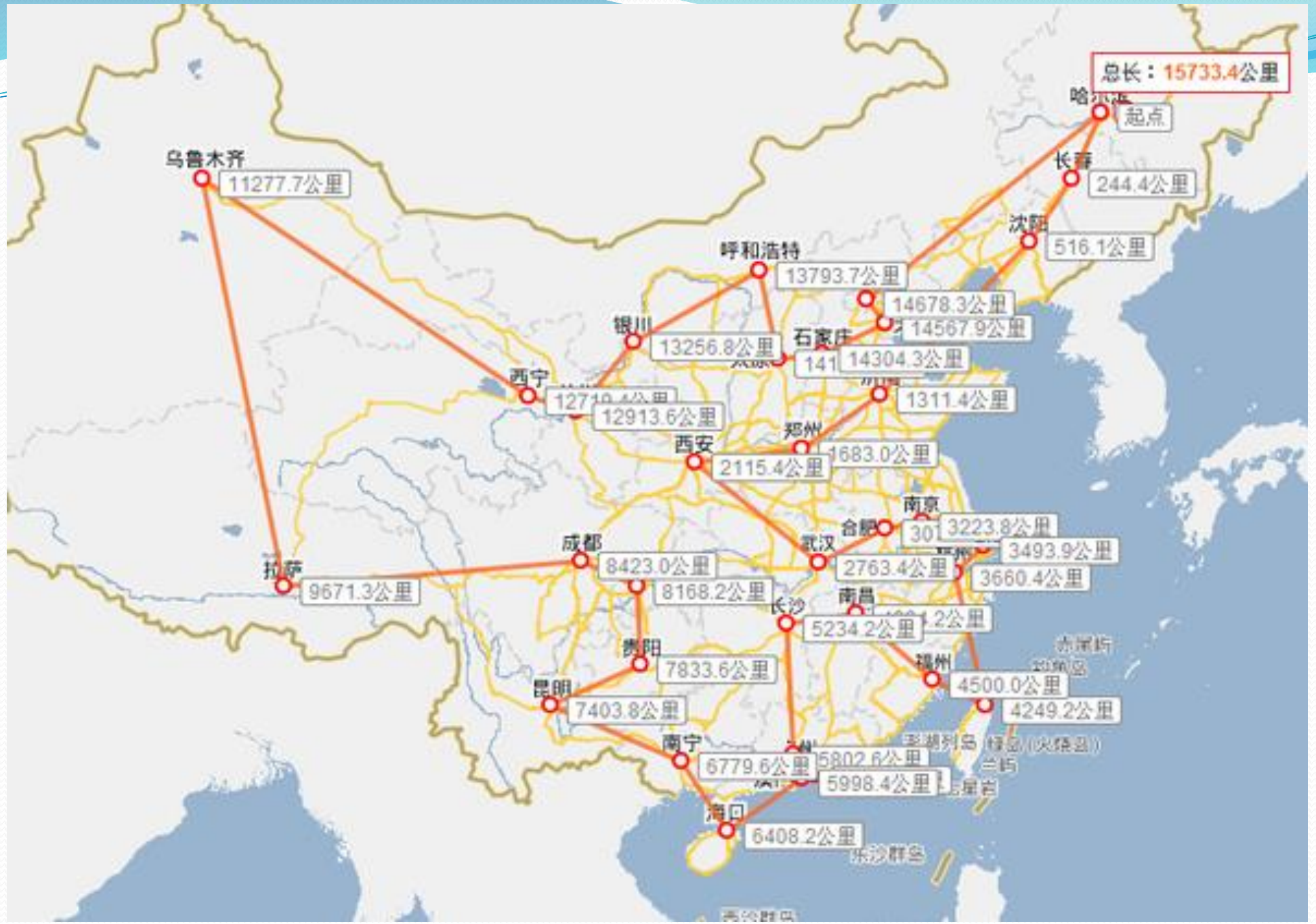
- （1）子集树的根结点是初始扩展结点，其cliqueSize的值为0。
- （2）在扩展内部结点时，首先考察其左儿子结点。在左儿子结点处，将顶点i加入到当前团中，并检查该顶点与当前团中其他顶点之间是否都有边相连。有则将它插入活结点优先队列，否则就不是可行结点。
- （3）接着继续考察当前扩展结点的右儿子结点。当upperSize>bestn时，右子树中可能含有最优解，此时将右儿子结点插入到活结点优先队列中。
- （4）从优先队列中取下一个活结点，重复（2）（3），直到一个叶子结点成为扩展结点，找到最优解，算法结束
- **代码见教材**

THE END

6.6 旅行售货员问题

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。


总长：15733.4公里
起点
乌鲁木齐
11277.7公里
长春
244.4公里
沈阳
516.1公里
呼和浩特
13793.7公里
14678.3公里
14567.9公里
银川
13256.8公里
石家庄
14304.3公里
1311.4公里
西宁
12913.6公里
郑州
1683.0公里
西安
2115.4公里
南京
3223.8公里
3493.9公里
3660.4公里
成都
8423.0公里
8168.2公里
2763.4公里
拉萨
9671.3公里
长沙
5234.2公里
贵阳
7833.6公里
福州
4500.0公里
4249.2公里
昆明
7403.8公里
南宁
6779.6公里
5998.4公里
海口
6408.2公里

6.6 旅行售货员问题

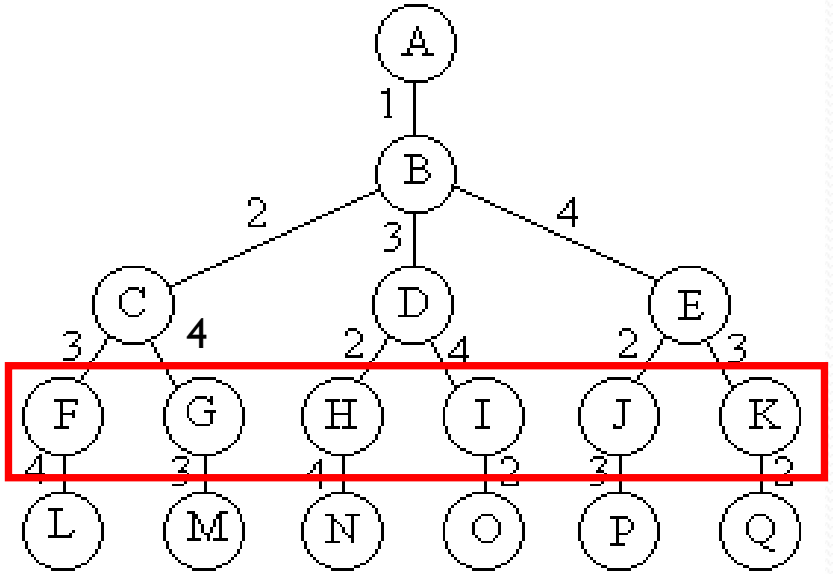
2.解空间树是排列树

（1）指定一个城市作为出发城市

（2）第n-1层结点看做叶子结点

n=4 TSP问题

X[1]
X[2]
X[3]
X[4]



6.6 旅行售货员问题

3. 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。
- 堆中每个活结点的优先级为：cc+lcost。cc为出发城市到当前城市的路程（或费用），lcost 是当前顶点(当前城市）最小出边+剩余顶点（城市）最小出边和（禁忌除外）。
- 每次从优先队列中取出一个活结点成为扩展结点（s层结点），
- **当s=n-2时**，扩展出的结点是排列树中某个叶子结点的父结点。如该叶结点相应一条可行回路且费用小于当前最优解bestc，则将该结点插入到优先队列中，否则舍去该结点

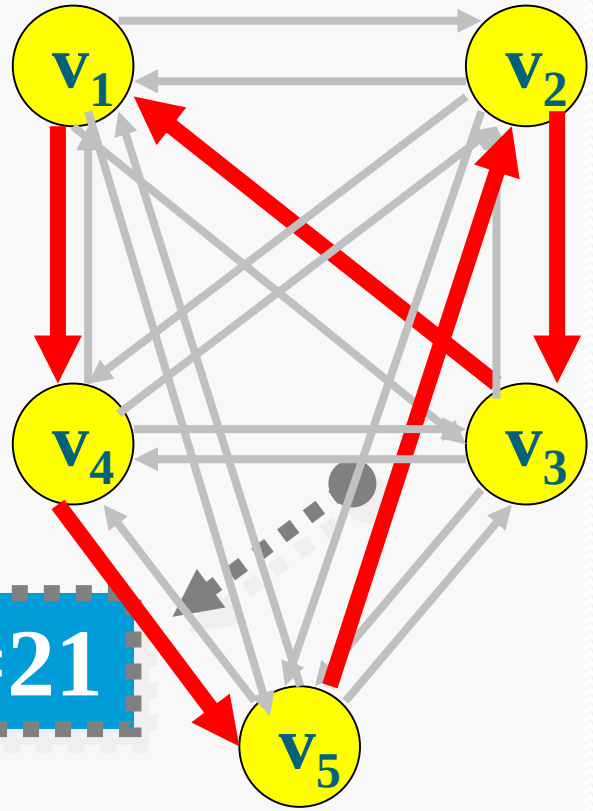
- **当s<n-2时**，产生当前扩展结点的所有儿子结点。计算可行儿子结点的优先级cc+lcost及相关信息。当cc+lcost<bestc时，将这个可行儿子结点插入到活结点优先队列中。
- 该扩展过程一直持续到优先队列中取出的活结点是一个叶子结点为止。
- **最小出边（禁忌除外）的解释**

对于刚扩展出的顶点，其前已选的所有顶点是禁忌的（不能选）。

对于未扩展出的顶点，其前已选的（顶点1除外）的顶点是禁忌的

算法演示

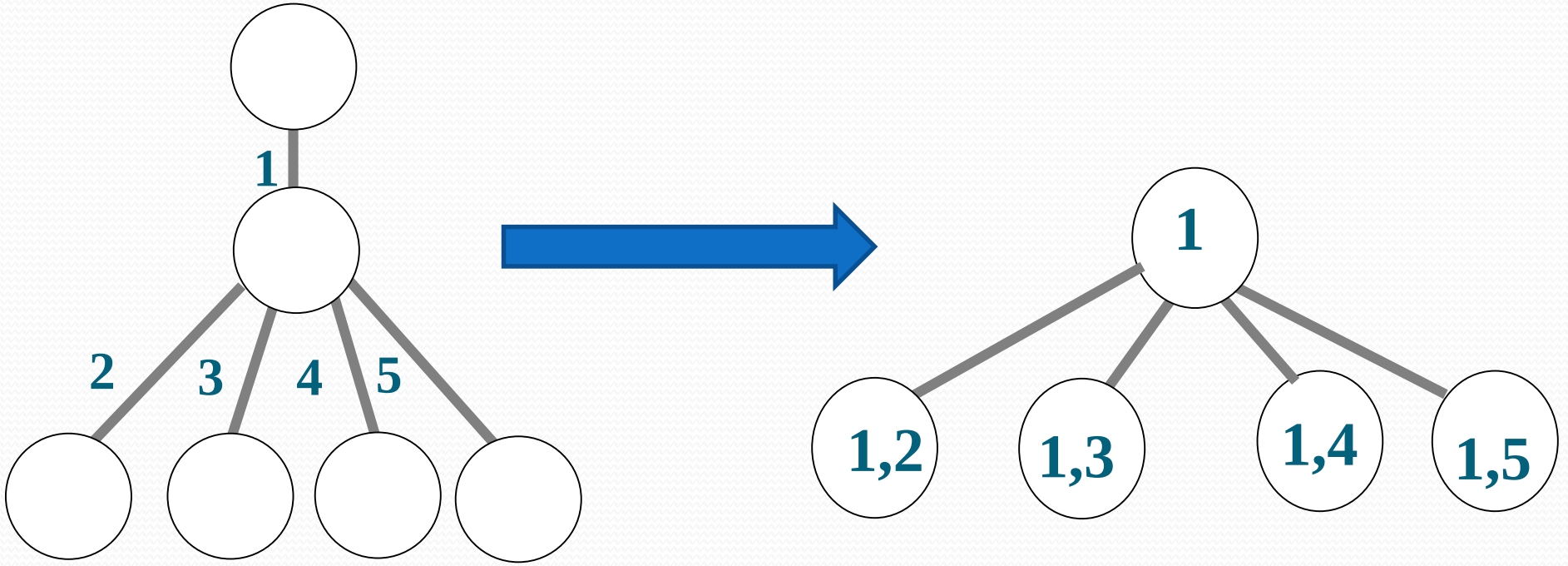
0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0

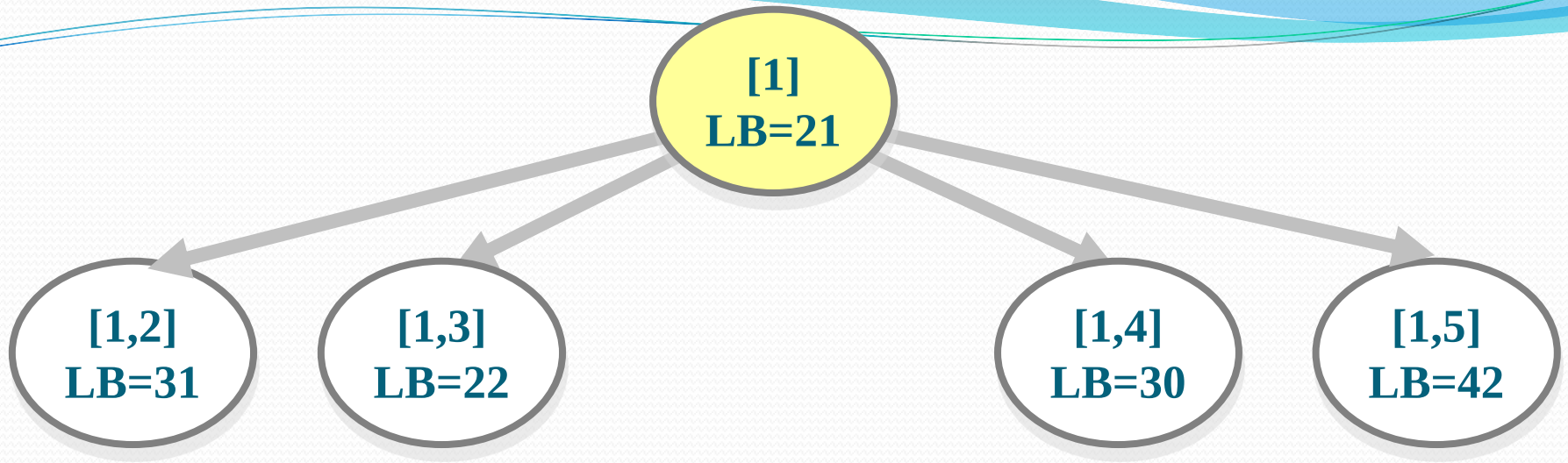


最小出边和之成本下界LB=21

LB=当前路径长度+当前扩展出的顶点及其后顶点可选最小出边和。

为动画演示状态空间改变为如图所示

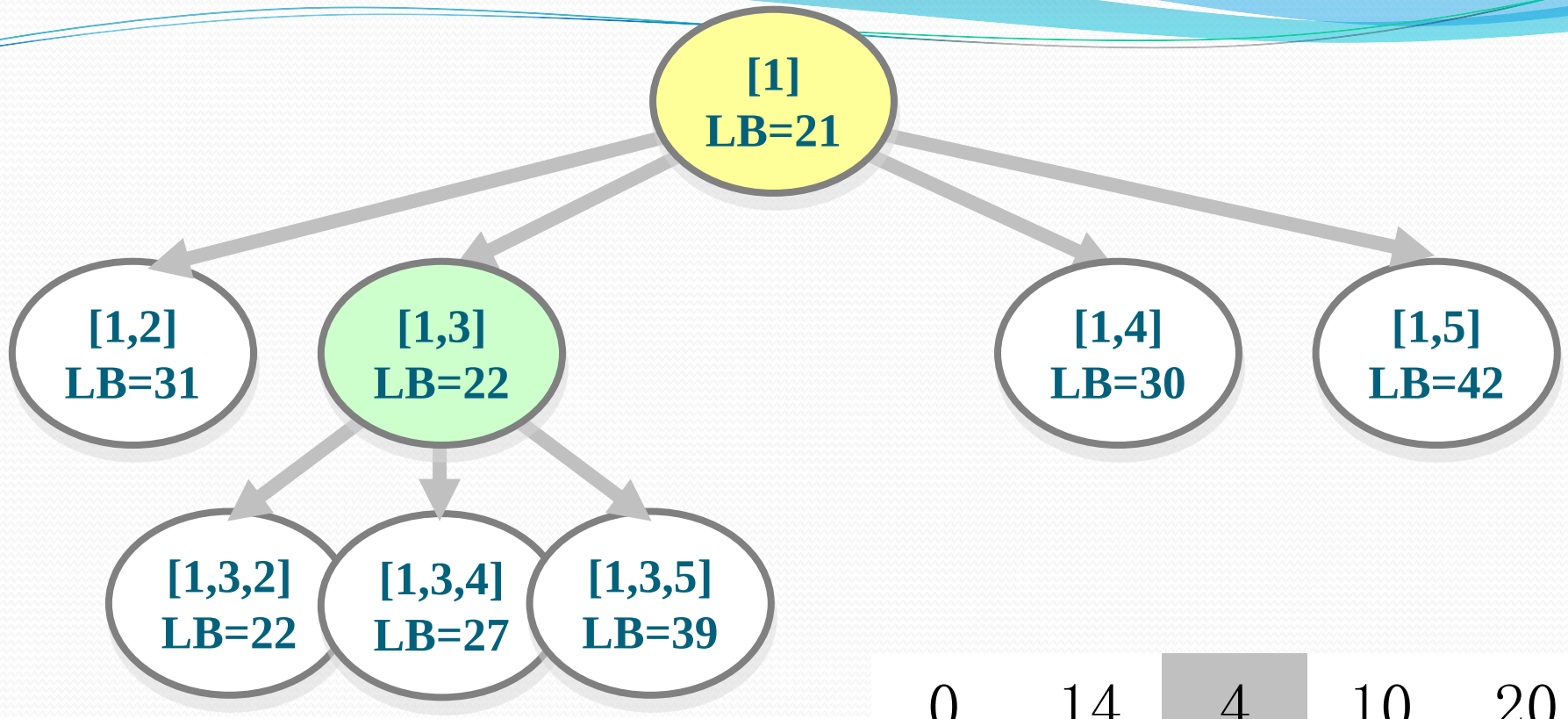




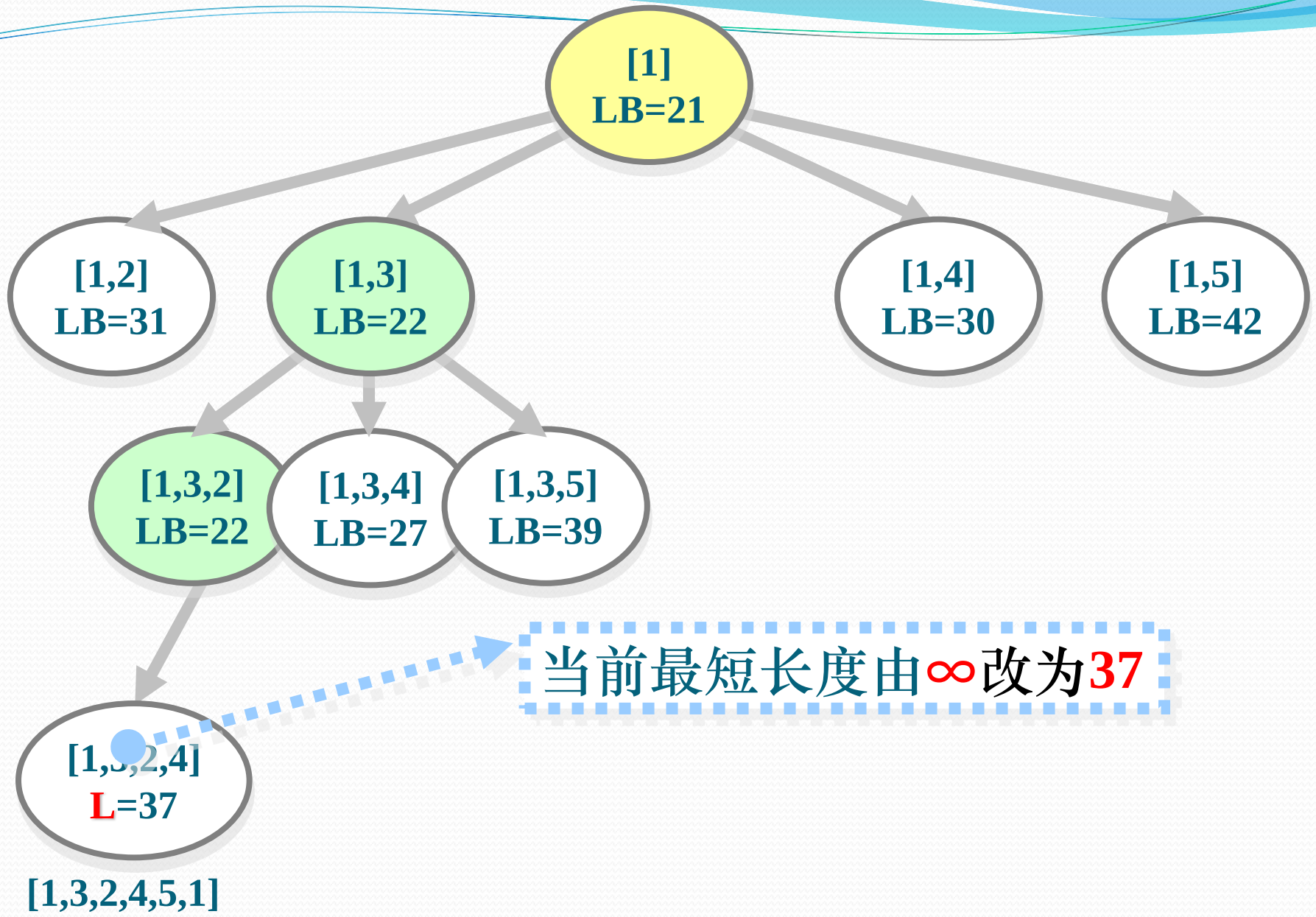
第一个孩子结点的优先级

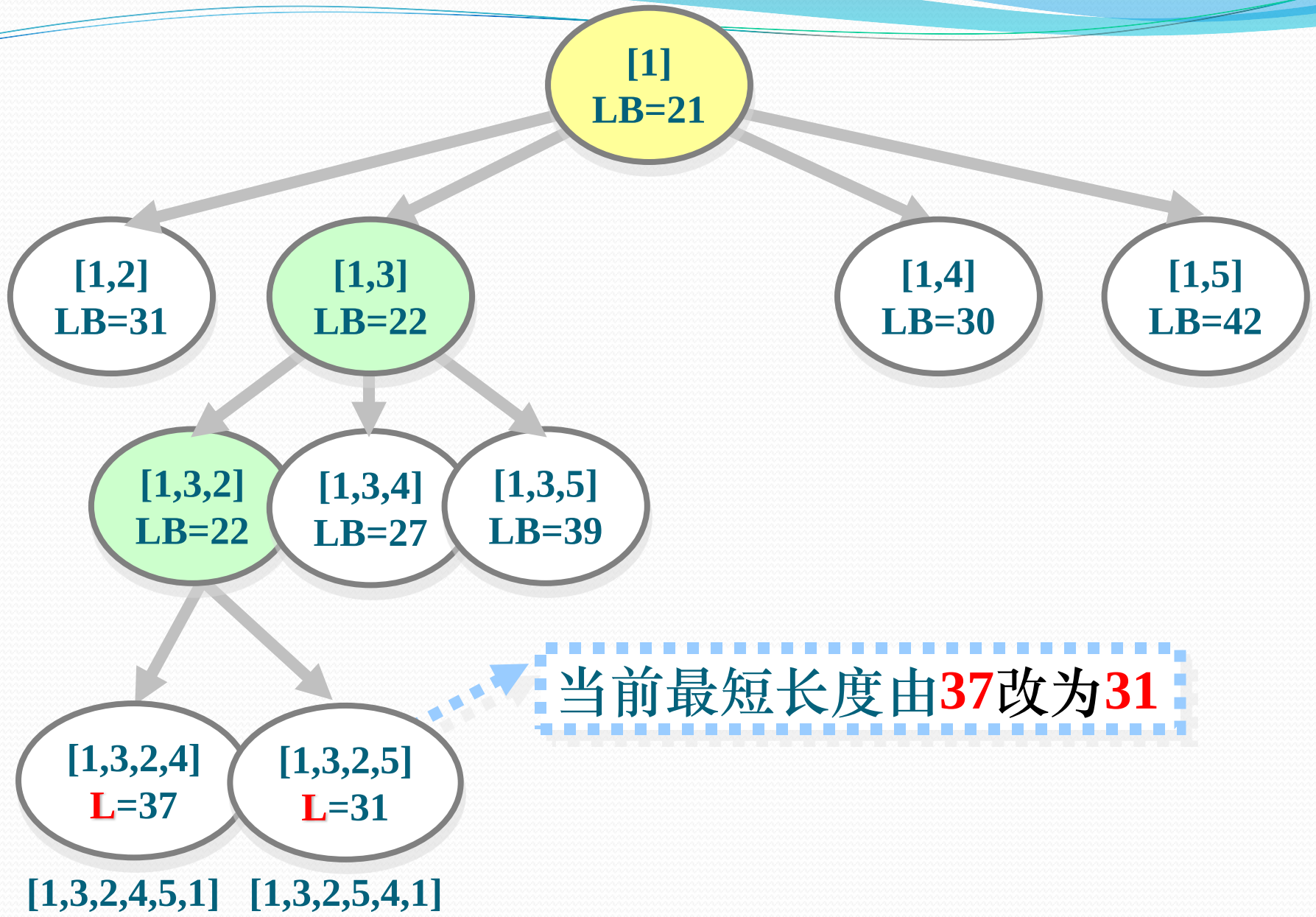
LB=cc+ lcost=14+7(2号城市最小出边)+4(3号城市最小出边)+2(4号城市最小出边)+4(5号城市最小出边)=31

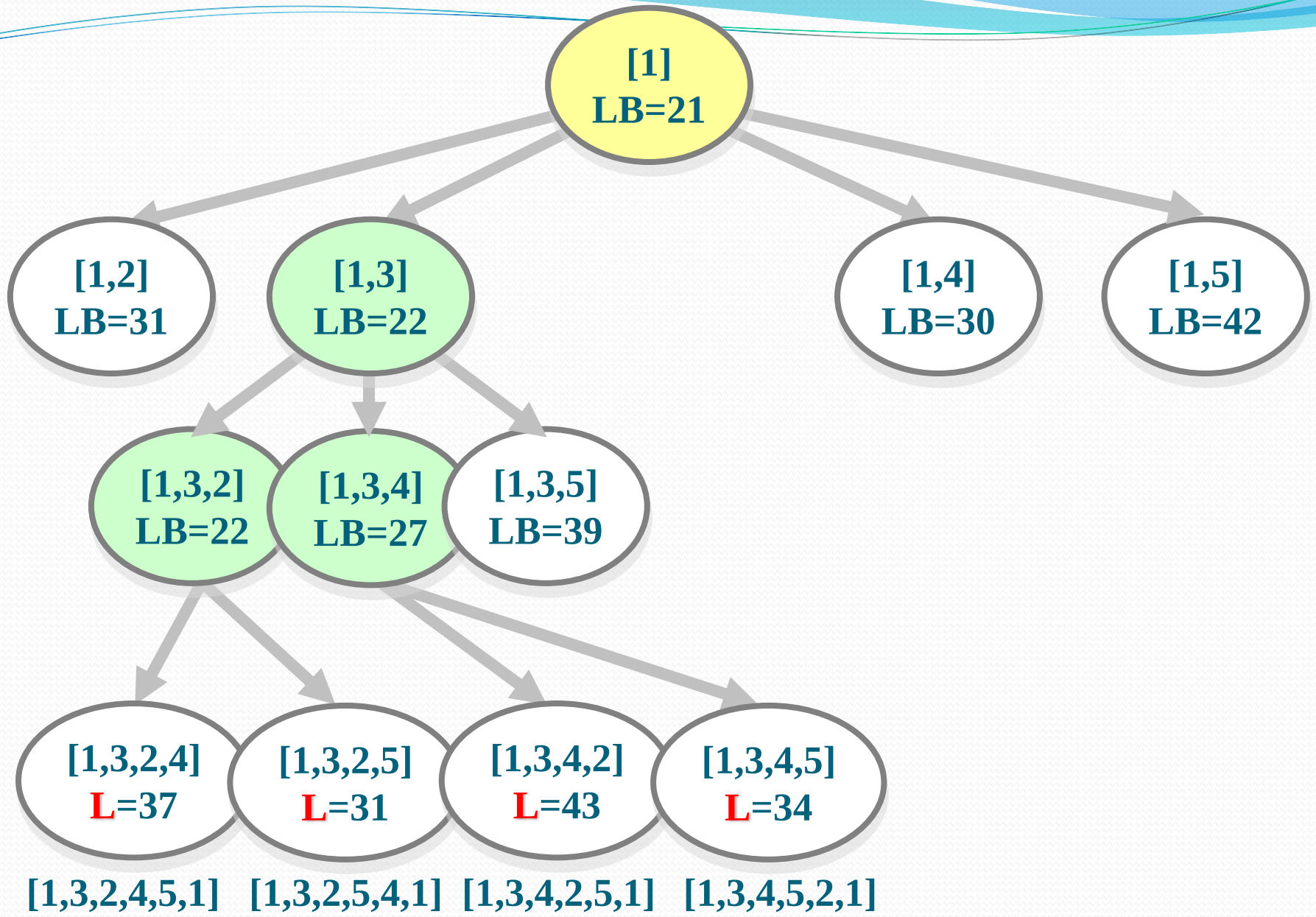
0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0

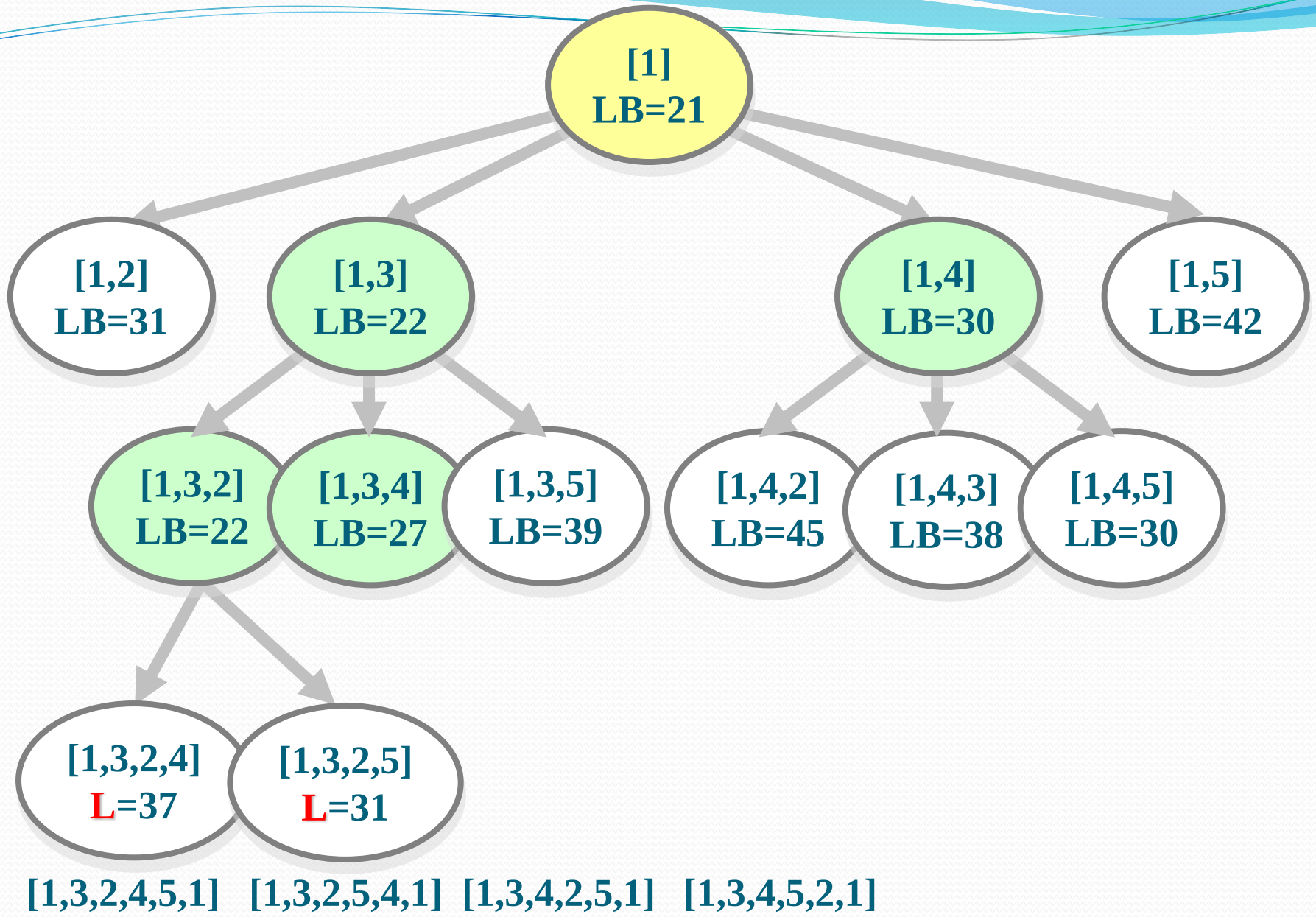


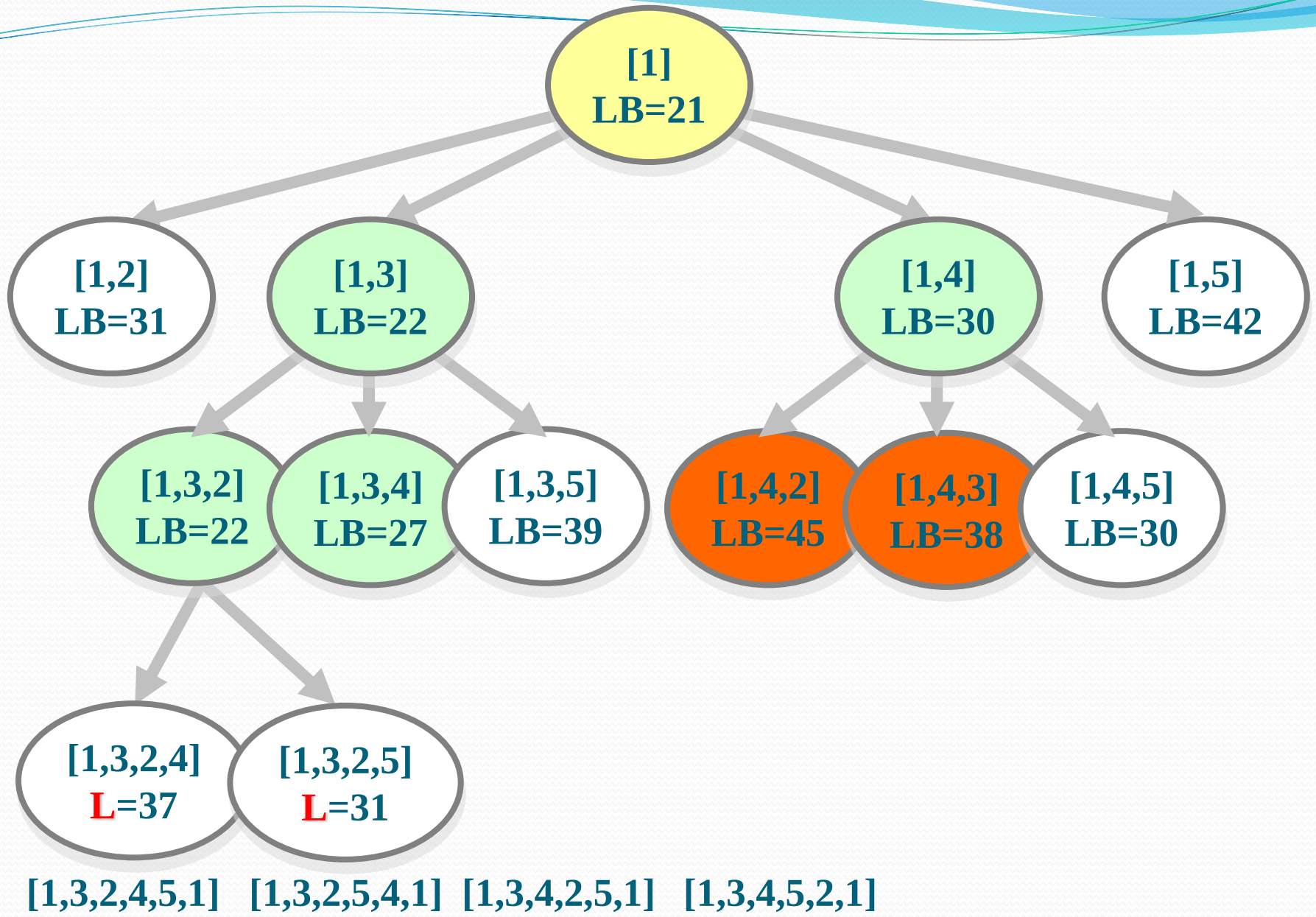
0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0

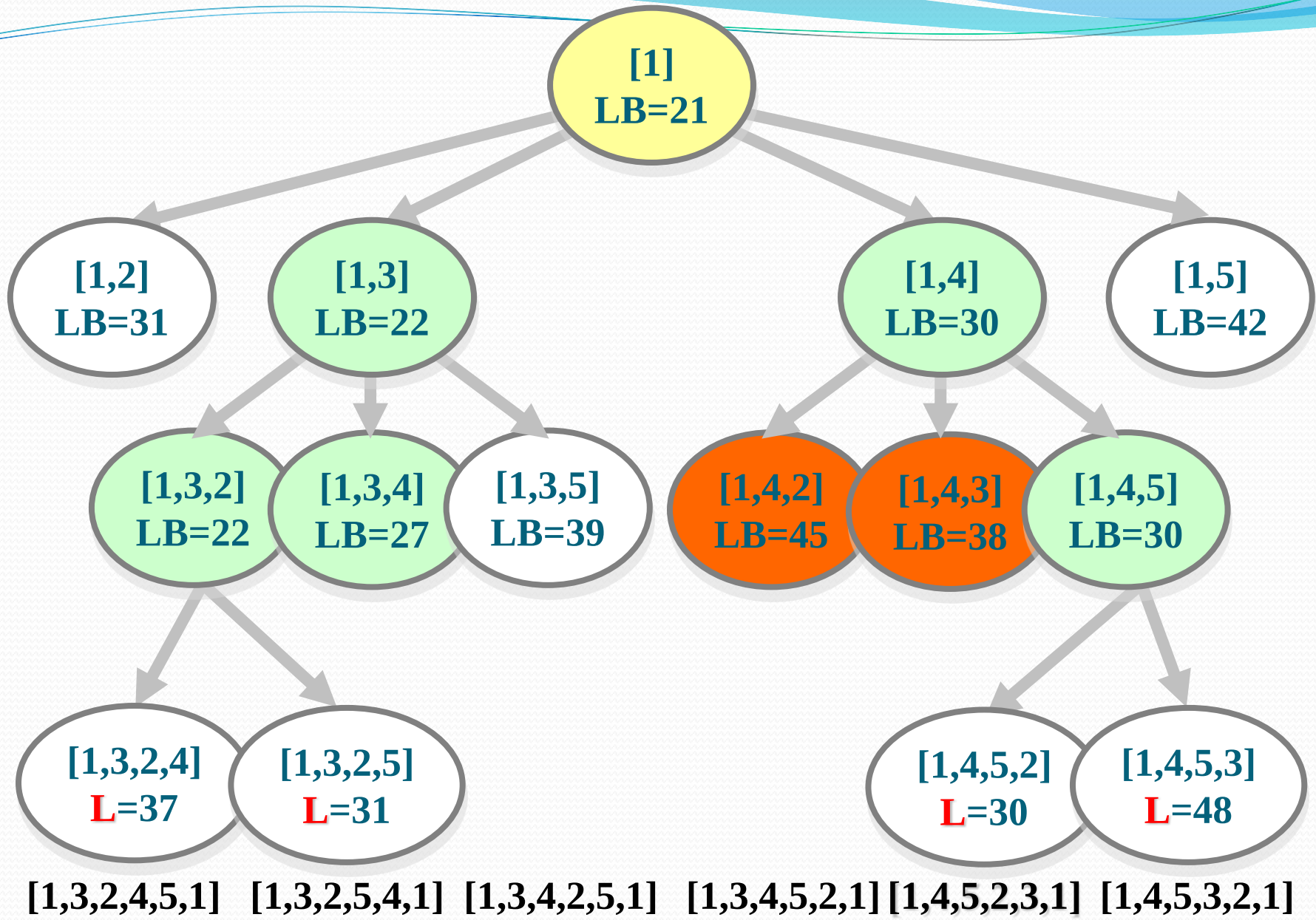

[1]
LB=21
[1,2]
LB=31
[1,3]
LB=22
[1,4]
LB=30
[1,5]
LB=42
[1,3,2]
LB=22
[1,3,4]
LB=27
[1,3,5]
LB=39
当前最短长度由∞改为37
[1,3,2,4]
L=37
[1,3,2,4,5,1]

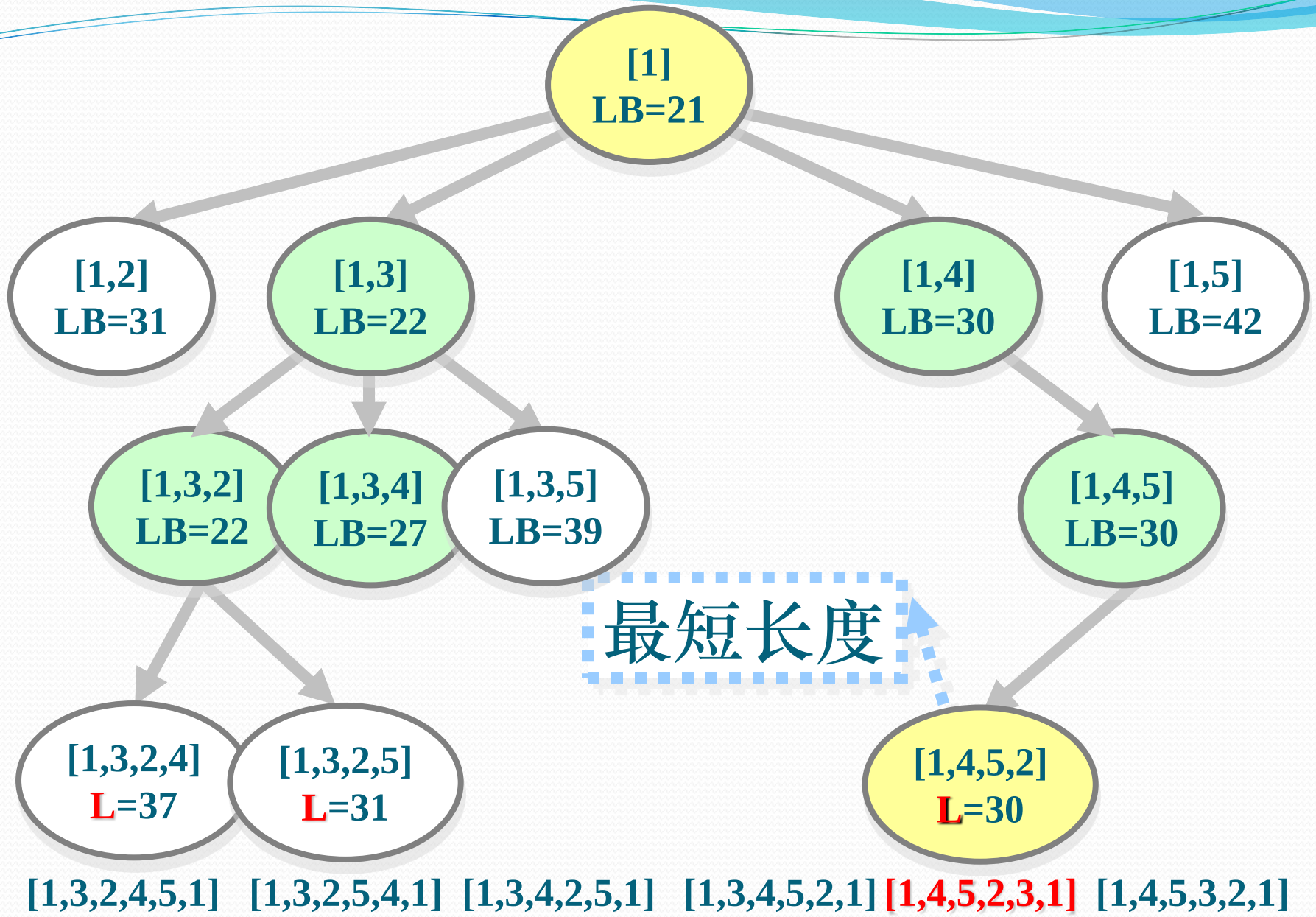

[1]
LB=21
[1,2]
LB=31
[1,3]
LB=22
[1,4]
LB=30
[1,5]
LB=42
[1,3,2]
LB=22
[1,3,4]
LB=27
[1,3,5]
LB=39
[1,3,2,4]
L=37
[1,3,2,5]
L=31
[1,3,2,4,5,1]
[1,3,2,5,4,1]
当前最短长度由37改为31


[1]
LB=21
[1,2]
LB=31
[1,3]
LB=22
[1,4]
LB=30
[1,5]
LB=42
[1,3,2]
LB=22
[1,3,4]
LB=27
[1,3,5]
LB=39
[1,3,2,4]
L=37
[1,3,2,5]
L=31
[1,3,4,2]
L=43
[1,3,4,5]
L=34
[1,3,2,4,5,1]
[1,3,2,5,4,1]
[1,3,4,2,5,1]
[1,3,4,5,2,1]


[1]
LB=21
[1,2]
LB=31
[1,3]
LB=22
[1,4]
LB=30
[1,5]
LB=42
[1,3,2]
LB=22
[1,3,4]
LB=27
[1,3,5]
LB=39
[1,4,2]
LB=45
[1,4,3]
LB=38
[1,4,5]
LB=30
[1,3,2,4]
L=37
[1,3,2,5]
L=31
[1,3,2,4,5,1]
[1,3,2,5,4,1]
[1,3,4,2,5,1]
[1,3,4,5,2,1]


[1]
LB=21
[1,2]
LB=31
[1,3]
LB=22
[1,4]
LB=30
[1,5]
LB=42
[1,3,2]
LB=22
[1,3,4]
LB=27
[1,3,5]
LB=39
[1,4,2]
LB=45
[1,4,3]
LB=38
[1,4,5]
LB=30
[1,3,2,4]
L=37
[1,3,2,5]
L=31
[1,3,2,4,5,1]
[1,3,2,5,4,1]
[1,3,4,2,5,1]
[1,3,4,5,2,1]


[1]
LB=21
[1,2]
LB=31
[1,3]
LB=22
[1,4]
LB=30
[1,5]
LB=42
[1,3,2]
LB=22
[1,3,4]
LB=27
[1,3,5]
LB=39
[1,4,2]
LB=45
[1,4,3]
LB=38
[1,4,5]
LB=30
[1,3,2,4]
L=37
[1,3,2,5]
L=31
[1,4,5,2]
L=30
[1,4,5,3]
L=48
[1,3,2,4,5,1]
[1,3,2,5,4,1]
[1,3,4,2,5,1]
[1,3,4,5,2,1]
[1,4,5,2,3,1]
[1,4,5,3,2,1]


[1]
LB=21
[1,2]
LB=31
[1,3]
LB=22
[1,4]
LB=30
[1,5]
LB=42
[1,3,2]
LB=22
[1,3,4]
LB=27
[1,3,5]
LB=39
[1,4,5]
LB=30
最短长度
[1,3,2,4]
L=37
[1,3,2,5]
L=31
[1,4,5,2]
L=30
[1,3,2,4,5,1]
[1,3,2,5,4,1]
[1,3,4,2,5,1]
[1,3,4,5,2,1]
[1,4,5,2,3,1]
[1,4,5,3,2,1]

6.7 小结

1. 队列式：活结点表是一个队列，新扩展出的满足条件的活结点追加在队尾。这里需要加入层次标志（如-1），或记录下层编号在活结点中。
2. 优先队列式：活结点表是优先队列，一般使用堆（大顶堆或小顶堆）存储，优点是只需要O（logn）时间复杂性完成插入或者删除（取堆顶结点，即优先级最高的结点）
3. 构造解方法， 活结点通过记录其父节点地址，及左孩子标志去，在找到最优值时，回溯方法找到最优解。**也可在扩展出的结点中记录构造的解，如问题规模较大时，应考虑压缩存储。**

6.7 小结

4.分支限界法的剪枝方法：

（1）对于子集树，左右分支剪枝策略不同，

（2）对于排列树，n叉树， 剪枝策略是相同的。

5. 算法的结束控制：

（1）队列式分支限界， 活结点表为空

（2）优先队列式，叶子结点成为扩展结点(在确认后面的活结点不存在更好的解)或队列为空。通常叶子结点加入优先队列中。