分支限界

Youtube频道： **hwdong**

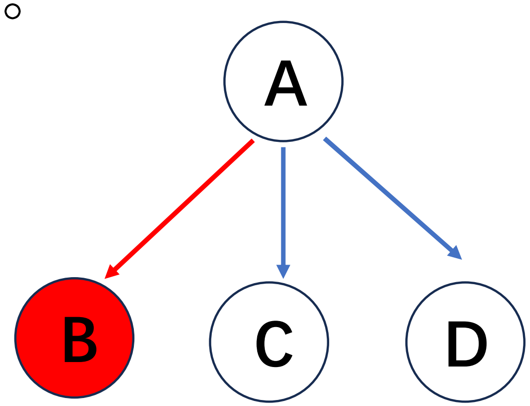
博客: **hwdong-net**.github.io

分支限界

- **分支限界**（Branch and Bound）是一种**启发式搜索**技术，通过构造搜索树并利用界限估计**剪枝**，结合**深度优先(回溯法)**或**广度优先搜索**策略，来求解优化问题并找到**最优解**。
- 分支限界主要用于求解**组合优化问题**，即需要找到最优解的问题。对于像**全排列**这种不涉及优化目标的组合问题，分支限界就不适用了，因为它没有最优解的概念。

分支限界

- 分支限界的基本思想是：每当我们尝试选择一个**分支（子问题）**时，通过计算某个“**界限**”来**估计此分支解的质量**。如果该分支的**界限**表明它不可能比已有的最优解更好，就可以提前终止这个分支的搜索（即**剪枝**）。



最大值问题

当前最优值：9

分支状态B的界限：8

分支限界

- 它的核心思想是 **分支**（将问题分解为子问题）和 **限界**（估算子问题的上下界，提前**剪枝**）。

分支限界：深度优先搜索(回溯法)

不适合大规模问题!

```
DFS(S, V):     //S:=当前状态，V:= 启发式解的价值
   if S是目标状态：   更新V, return
   for(S的后续状态N):
       b = bound(N) //估算N状态的价值的界
       if b不优于V:
           continue    //停止该状态N的探索
       else:
           DFS(N, V): //从N状态继续探索
```

分支限界：广度优先搜索

```
BFS() :
    初始化V为无穷大(最小值问题)或无穷小（最大值问题)

    初始化状态队列
    while 队列不空
        出队一个节点N
        if N表示一个可行解x,     //如到达叶子状态
            if f(x)优于V:   则V = f(x)
        else
            for 每个分支Ni ,
                if B(Ni)优于V:
                    Ni入队
```

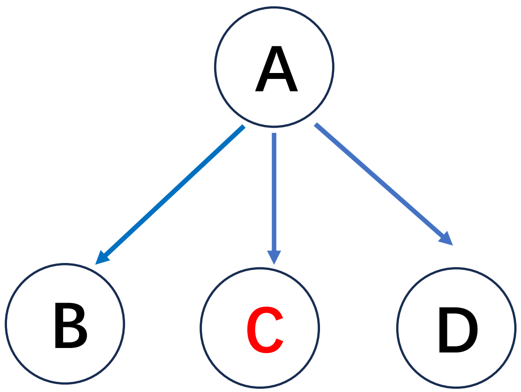
直到遇到叶子节点才更新！太迟了！

分支限界：广度优先搜索

```
BFS() :
    用（贪婪法）得到的可行解初始化V
    初始化状态队列
    while 队列不空
        出队一个节点N
        if N表示一个可行解x,    //如到达叶子状态
            if f(x)优于V:   则V = f(x)
        else
            for 每个分支Ni ,
                if B(Ni)优于V:
                    Ni入队
```

更好的方法：优先队列

- 分支限界通常借助于**优先队列**，先扩展“最有希望”的节点（如**界限最优**的节点）。



最大值问题

当前最优值：9

分支状态B、**C**、D的界限分别为：11，15，13

分支限界算法步骤

1. **初始化**：
   - 创建**优先队列**（按界限排序）。
   - 计算**根节点**的界限并加入队列。
   - 设定**当前最优解**为无穷小。
2. **循环搜索**（直到队列为空）：
   - 取出**界限最优**的节点。
   - 若该节点的界限 ≤ 当前最优解，则剪枝，跳过。
   - 若该节点是可行解且优于当前最优解，则更新最优解。
   - 生成**子节点**，计算界限，若界限大于当前最优解，则入队。
3. **终止**：队列为空，返回最优解。

```
branch_and_bound(problem):
    priority_queue = PriorityQueue() # 初始化优先队列
    initial_node =    create_initial_node(problem)   # 根节点
    priority_queue.put(initial_node)
    # 初始化最优解
    best_solution = None
    best_value = -infinity  # 或者最适合的问题的初值

   while not priority_queue.empty():
       # 从优先队列中取出当前最优的节点
       current_node = priority_queue.get()
```

```python
# 如果当前节点的界限大于当前已知最优解，则跳过该节点
if current_node.bound <= best_value:
    continue

# 如果当前节点是目标解且比最优解更好，则更新最优解
if is_solution(current_node):
    if current_node.value > best_value:
        best_solution = current_node.solution
        best_value = current_node.value
```

```
# 扩展当前节点的子节点
for child_node in generate_children(current_node):
    # 计算子节点的界限
    child_node.bound = compute_bound(child_node)

    # 如果子节点的界限表示它不可能得到更优解，则剪枝
    if child_node.bound > best_value:
        priority_queue.put(child_node)

return best_solution
```

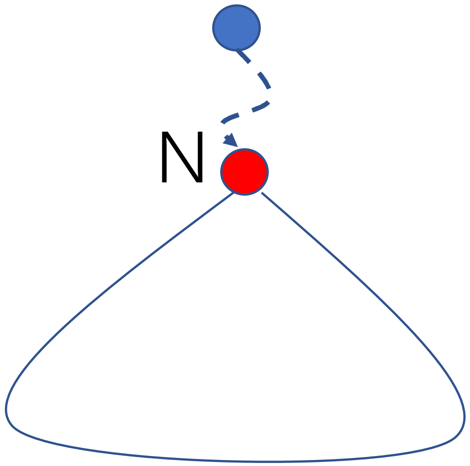
0-1背包问题

- 给定一组物品，每个物品有重量和价值，背包有固定容量，求选择哪些物品放入背包，使得背包中的物品总价值最大，且总重量不超过背包容量。

背包问题：估算状态的限界

按单位重量价值排序:

$$v_1/w_1 >= v_2/w_2 >= \cdots >= v_n/w_n$$



$$w = (4,7,5,3)\ , v = (40,42,25,12)\ , W = 10$$

估算限界

$$ub = cv + (W - cw) * v_{i+1}/w_{i+1}$$

已装物品价值

已装物品重量

剩余物品最大单位重量价值

背包问题：估算状态的限界

```
ub = cv ; wt = cw; j=i+1
while j < =n and wt + w[j] <= W:
        ub+= v[j]
        wt += w[j]
        j += 1

//最后一物品的分数重量的价值
if j <= n and wt < W :
    ub += (W - wt) * float(v[j])/ w[j]
```

背包问题：估算当前最优价值

- 1）当前解的最优价值作为最优价值V

缺点：必须等到找到可行解，才能更新V; 不适合广度优先

- 2）已装填物品的最大价值作为最优价值V。优点：随时更新

- w = (4,7,5,3) ,v = (40,42,25,12) ,W = 10

cw=0 cv=0
ub=100

V = 0

ub = 0+(10-0)* v_1/w_1 =
10*10=100

- $w = (4,7,5,3)$, $v = (40,42,25,12)$, $W = 10$

$V = 40$

cw=0 cv=0
ub=100

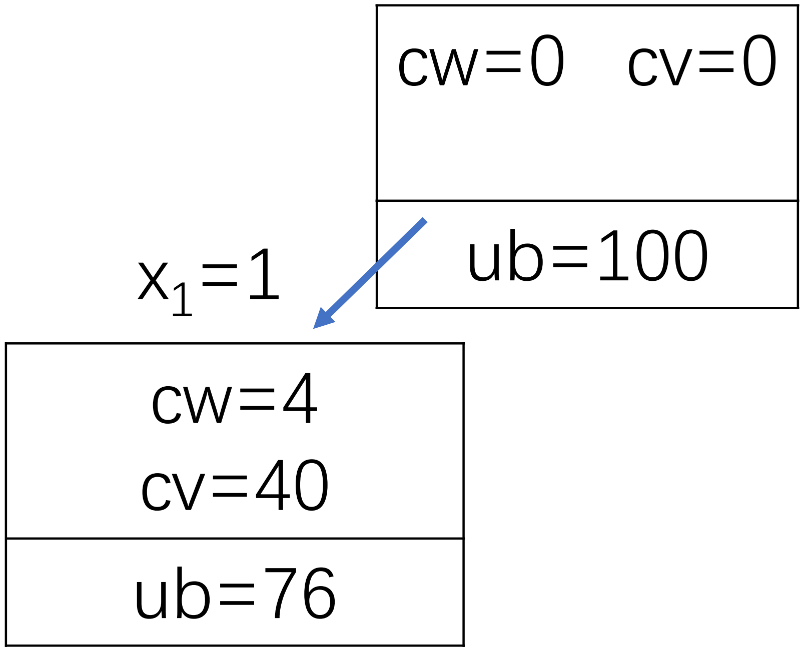
$x_1=1$

cw=4 cv=40
ub=76

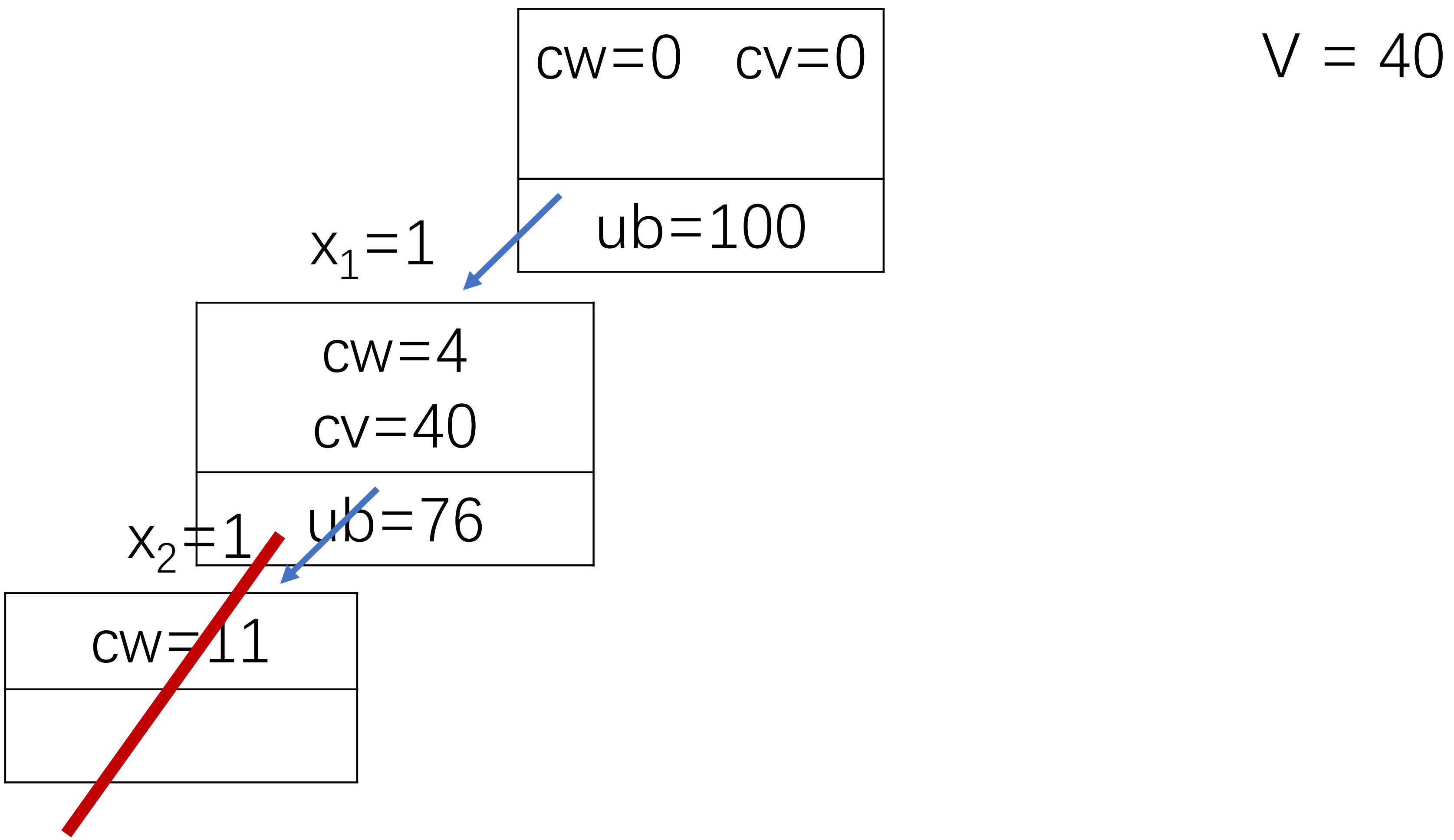
$$ub = 40+(10-4)* v_2/w_2 = 40+6*6=76$$

- $w = (4,7,5,3)$, $v = (40,42,25,12)$, $W = 10$

$V = 40$

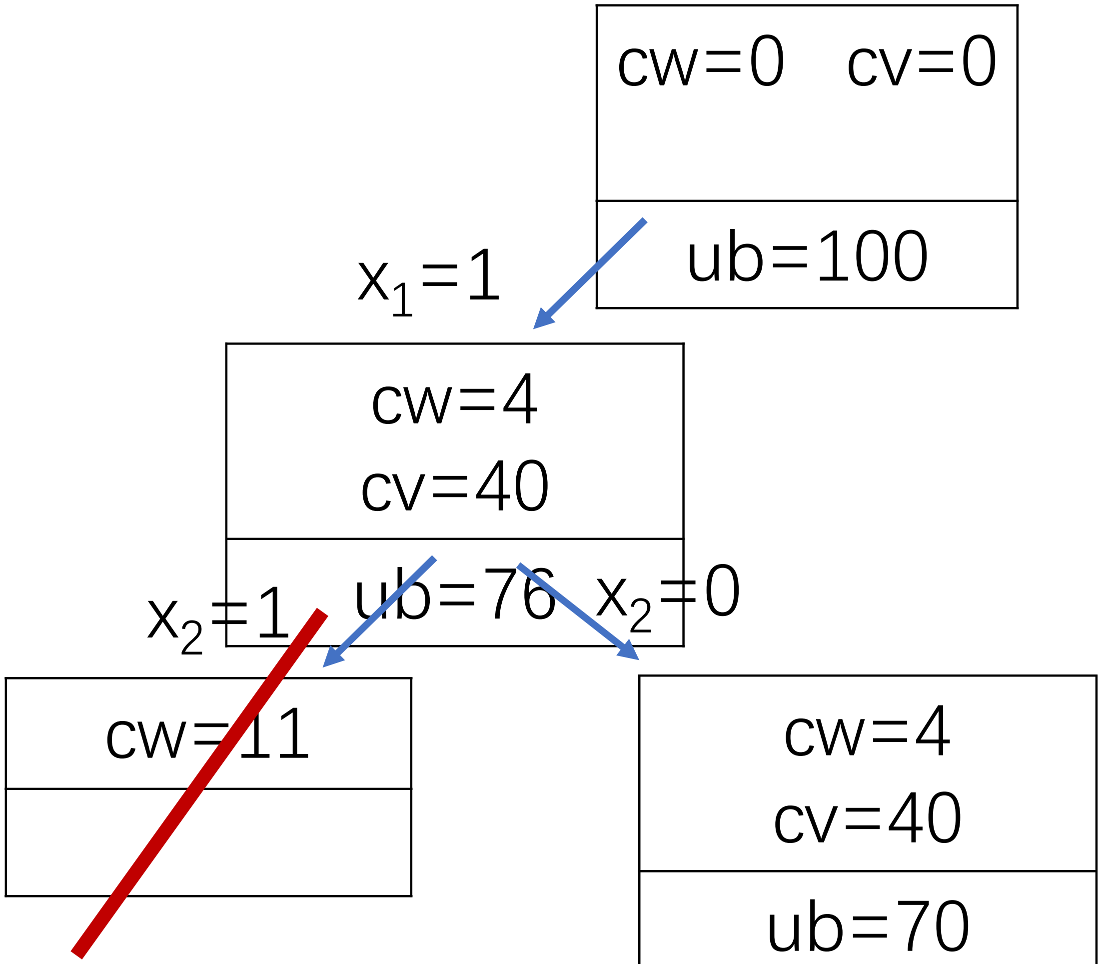


- $w = (4,7,5,3)$,$v = (40,42,25,12)$,$W = 10$



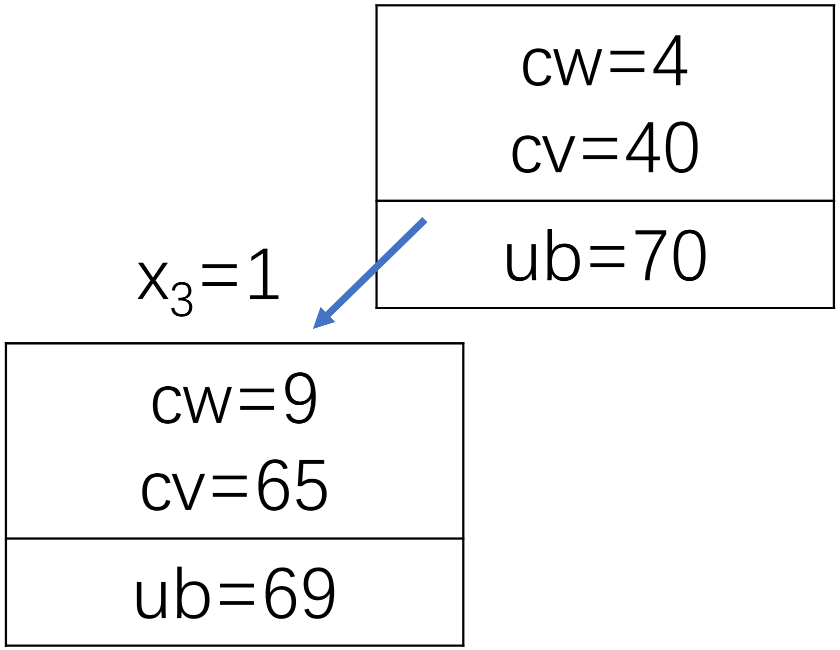
- $w = (4,7,5,3)$, $v = (40,42,25,12)$, $W = 10$

$V = 40$



$$ub = 40+(10-4)* v_3/w_3 = 40+6*5=70$$

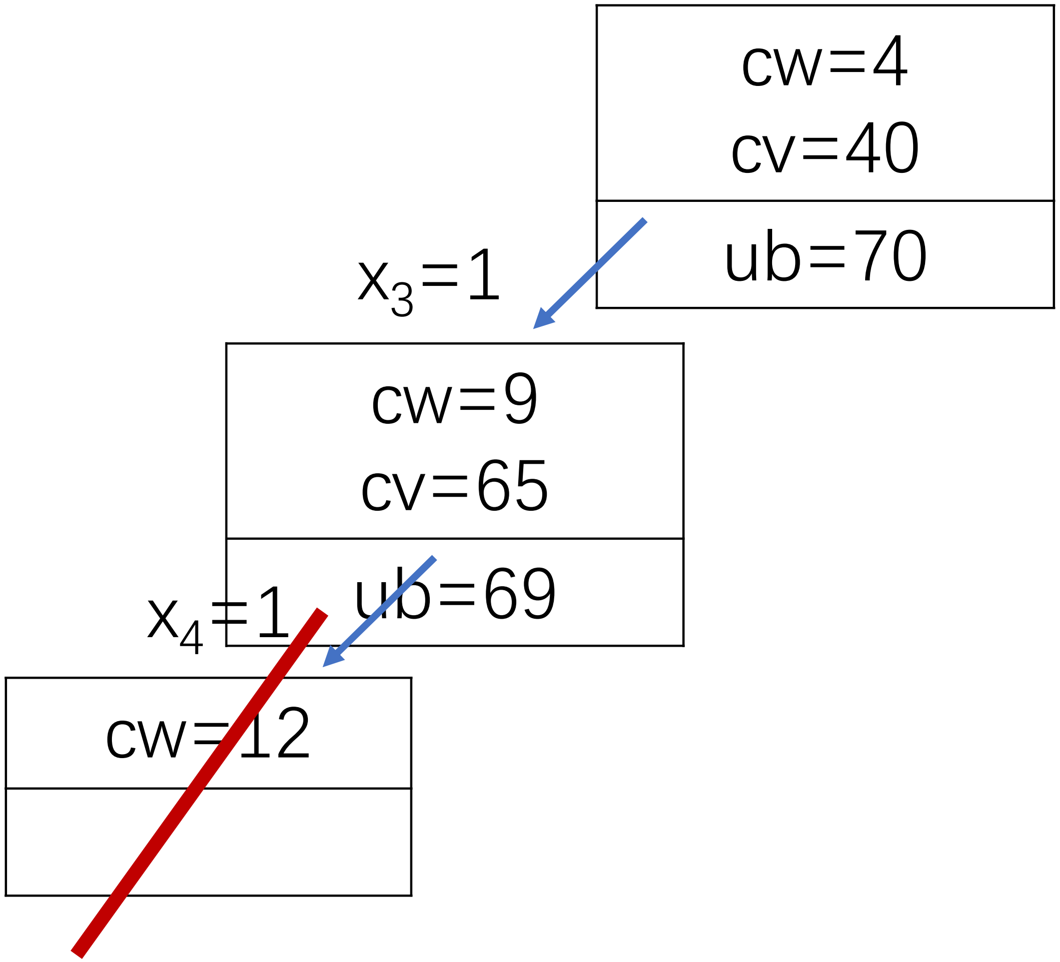
- $w = (4,7,5,3)$,$v = (40,42,25,12)$,$W = 10$



V = 65

$ub = 65+(10-9)* v_4/w_4 = 65+1*4=$
69

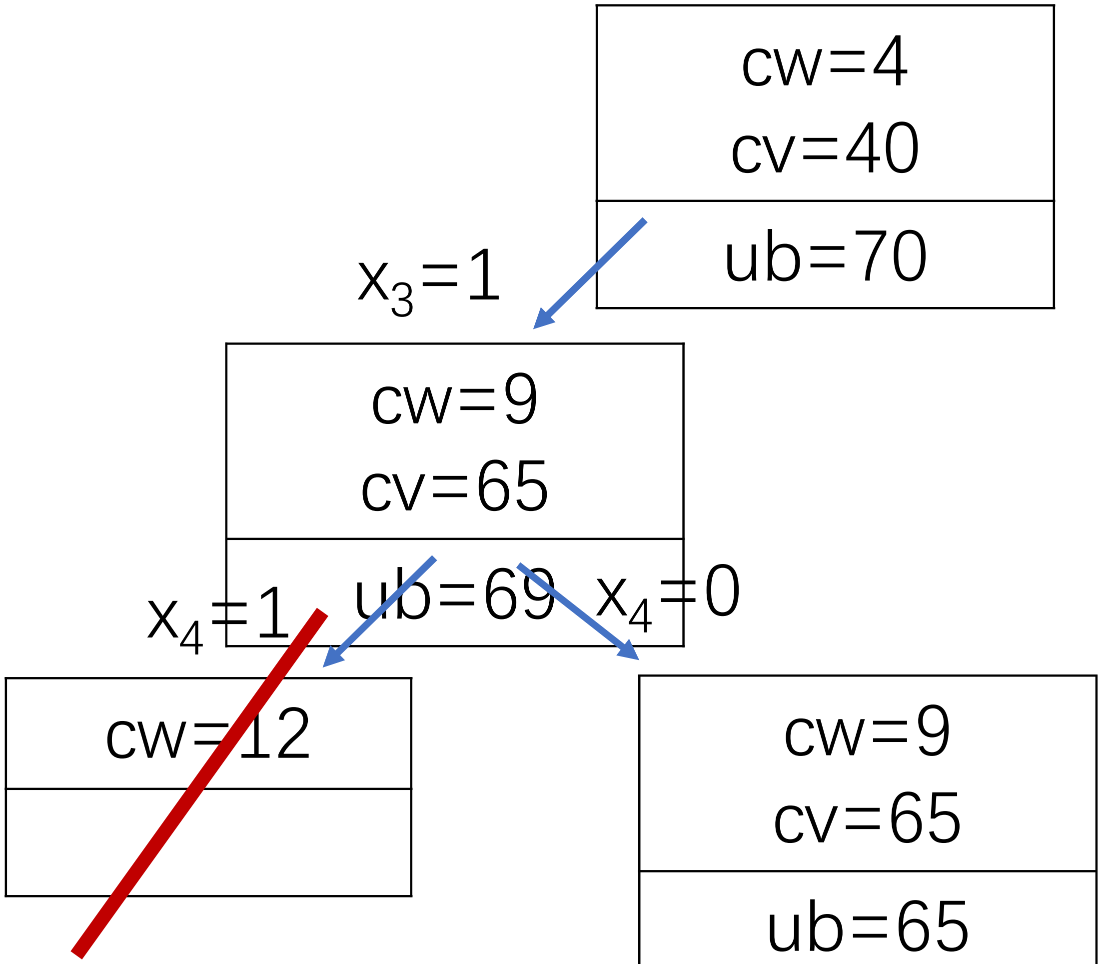
- $w = (4,7,5,3)$,$v = (40,42,25,12)$,$W = 10$



$V = 65$

$ub = 40+(10-4)* v_4/w_4 = 40+6*4=$
64

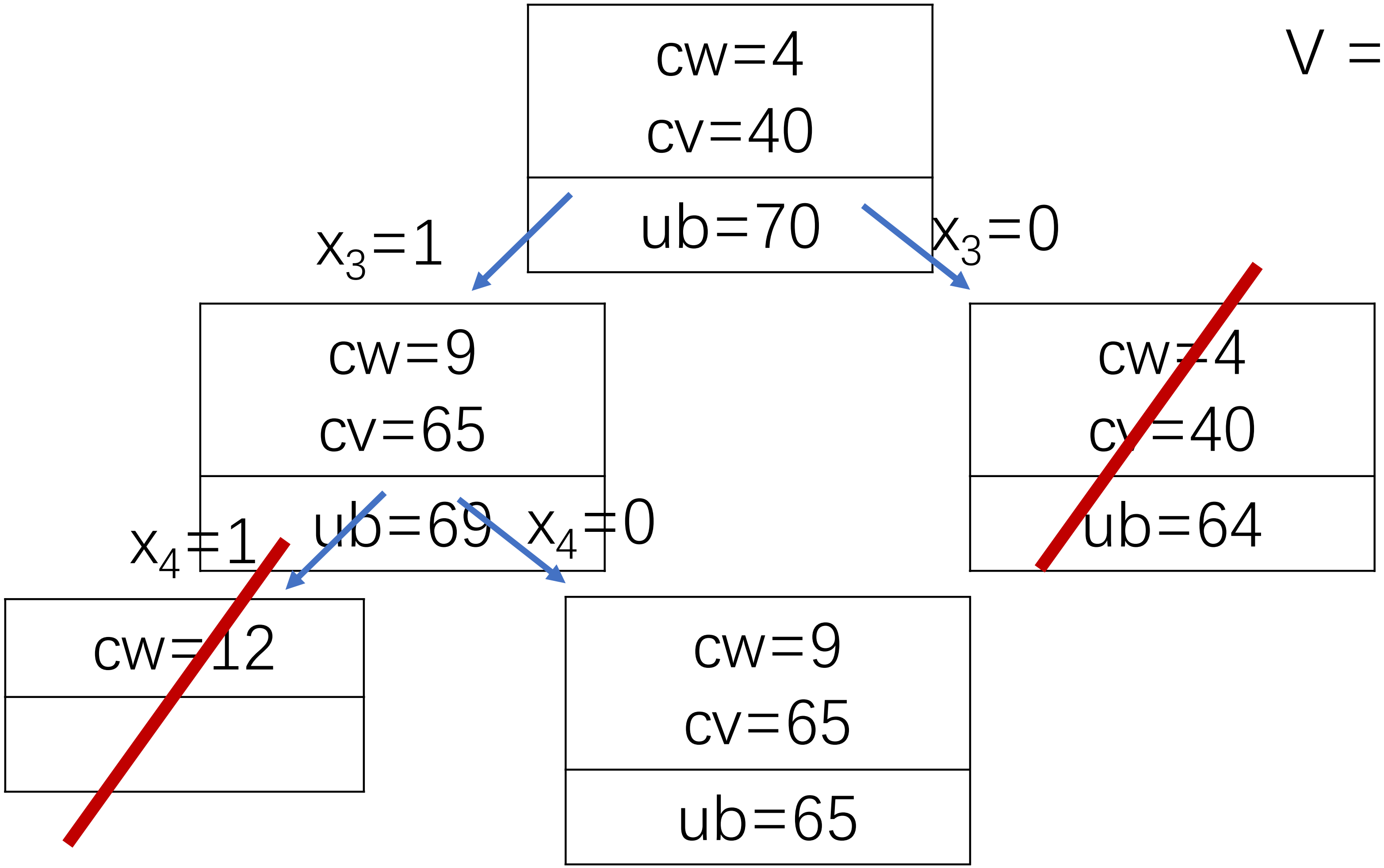
- w = (4,7,5,3) ,v = (40,42,25,12) ,W = 10



V = 65

- w = (4,7,5,3) ,v = (40,42,25,12) ,W = 10

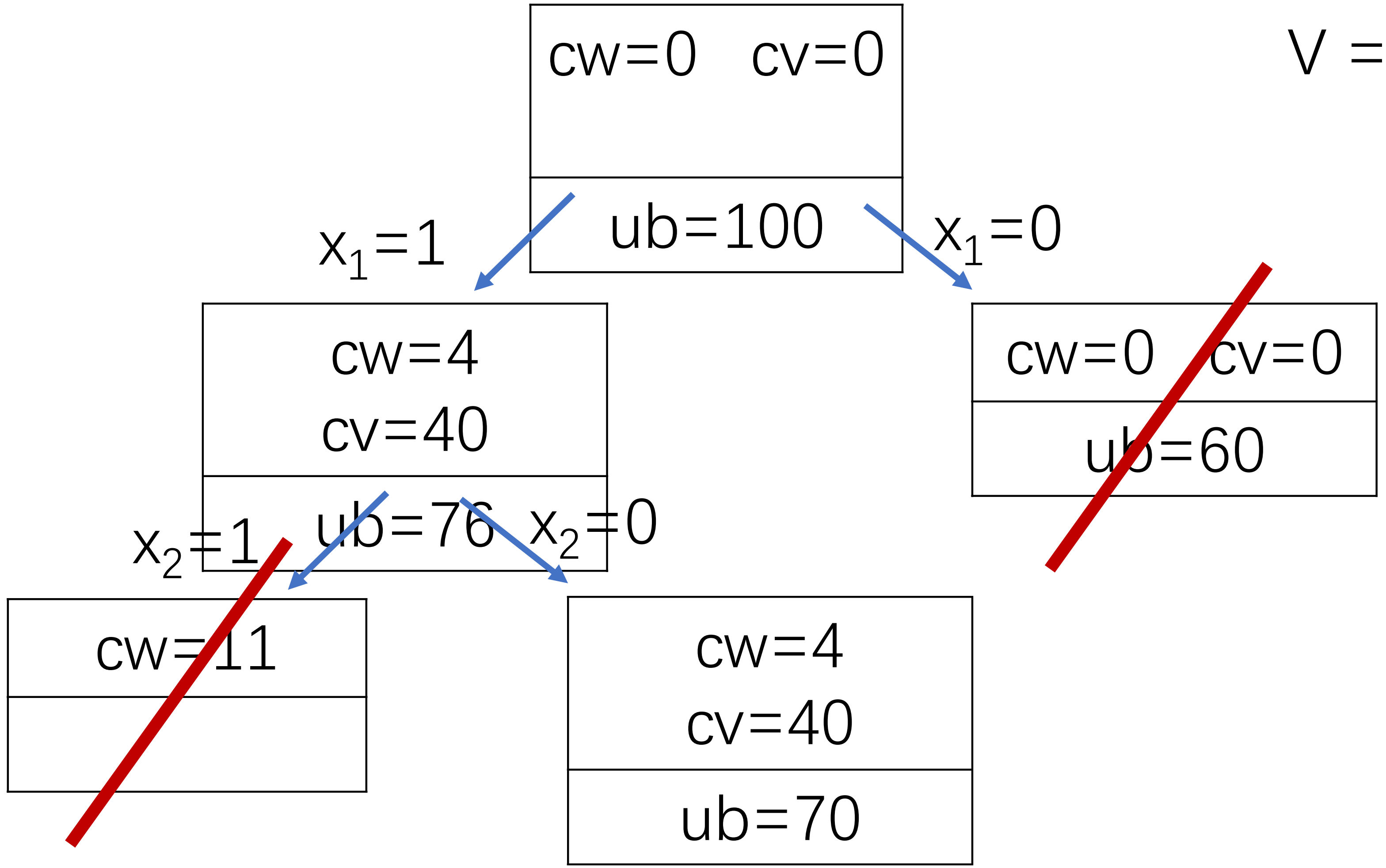
V = 65



ub = 40+(10-4)* v_4/w_4 = 40+6*4=
64

- $w = (4,7,5,3)$,$v = (40,42,25,12)$,$W = 10$

$V = 65$



$$ub = 0+(10-0)* v_2/w_2 = 0+10*6=60$$

```
struct Node {
    level: int        // 当前处理的物品层级
    value: int        // 当前总价值
    weight: int       // 当前总重量
    bound: float      // 上界
}
```

```
knapsack(items, capacity):
    sort items by value/weight descending
    initialize priority queue pq
    max_value = 0
    root = Node(level=-1, value=0, weight=0,
                bound=calculate_bound(0,0,-1))
    pq.push(root)
```

```
while pq not empty:
    u = pq.pop()
    if u.bound ≤ max_value:
        continue
    if u.level == n-1:
        update max_value if needed
        continue
```

```
// 生成子节点
  next_level = u.level + 1
  // 左子节点（选物品）
  left = Node(level=next_level, value=u.value +
                                    items[next_level].value,
              weight=u.weight + items[next_level].weight)
  if left.weight ≤ capacity:
      left.bound = calculate_bound(left.value, left.weight,
                                   next_level)
      if left.bound > max_value:
          pq.push(left)
```

```
    // 右子节点（不选物品）
    right = Node(level=next_level, value=u.value, weight=u.weight)
    right.bound = calculate_bound(right.value, right.weight,
                                  next_level)
    if right.bound > max_value:
        pq.push(right)
return max_value
```

状态的界

```
bound(cv,cw,iW, w[], v[]) :
  if (cw+w[i] > W) return 0;
  T bound_value = cv;
  int j = i, n = w.size();
  T weight = cw;
  while j < n && weight + w[j] <= W:
     weight += w[j];
     bound_value += v[j]; j = j+1;

   if j<n:
      bound_value += (W - weight) * v[j]/w[j];
   return bound_value;
```

State

```
struct{
    Array path; //
    T cv,cw;
};
```

分支限界法求解旅行商问题（TSP）

旅行商问题（TSP）是一个典型的**组合优化问题**，描述如下：

- 给定 n 个城市和一个 $n \times n$ 的距离矩阵 $D[i][j]$，表示城市 i 到城市 j 的路径距离。
- 旅行商从某个城市出发，访问所有城市**恰好一次**，然后回到起点。
- **目标**：找到一条总距离最短的回路（Hamiltonian Cycle）。

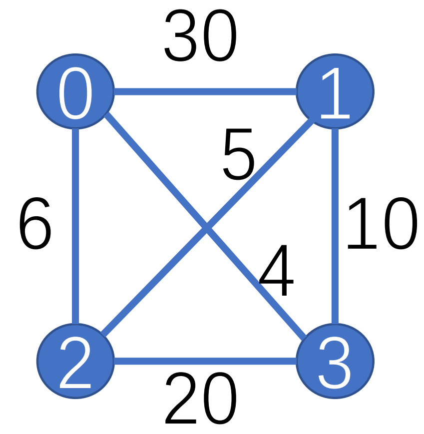
- 由于 TSP 的解空间规模为 O(n!)，使用暴力搜索会导致指数级时间复杂度，因此可以使用 分支限界法 来优化求解过程。

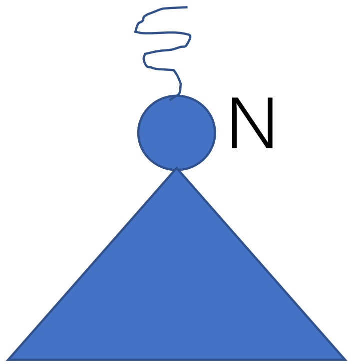
分支限界法求解 TSP 的基本思想

1. **构建搜索树**:
   - 根节点代表**未访问任何城市**的状态。
   - 每个子节点表示选择了某个城市后形成的部分路径。
2. **分支策略（子节点扩展）**：
   - 从当前城市扩展到**尚未访问的城市**，形成新的路径。
3. **界限计算（Bound）**：
   - 计算当前部分路径的**下界**（Lower Bound），即从该状态出发，最乐观情况下的最小旅行成本。
   - 如果某个分支的下界已经大于当前最优解，则剪枝（prune）。
4. **剪枝**:
   - 只有可能带来更优解的分支才继续搜索，否则丢弃该分支，减少搜索空间。

TSP-分支限界

- 这是一个最小化问题，
- 设当前最优解价值是B，
- lb(N)是状态N子树的所有可能价值的最小值

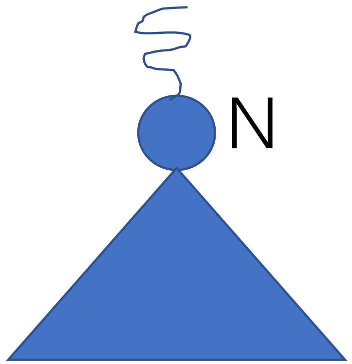


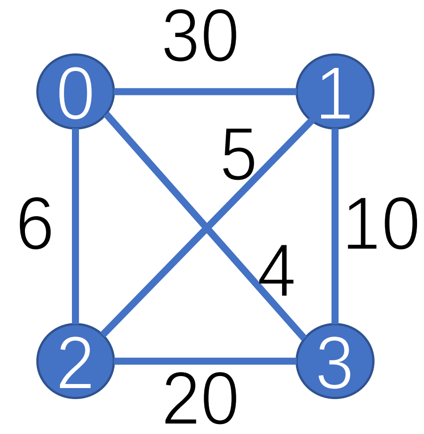


TSP-分支限界

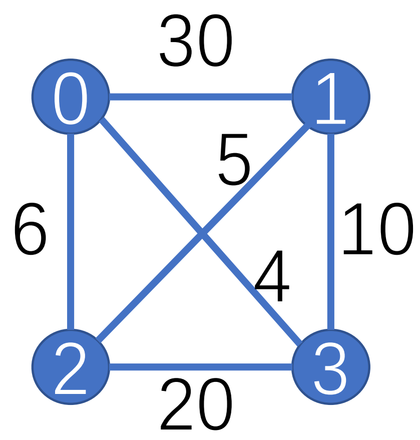
- 这是一个最小化问题，
- 设当前最优解价值是B,
- lb(N)是状态N子树的所有可能价值的最小值
- if lb(N) >B:

  舍弃N子树(不可能产生更小的B)
- else:可探索N

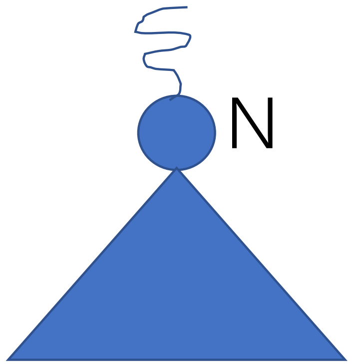




TSP-分支限界

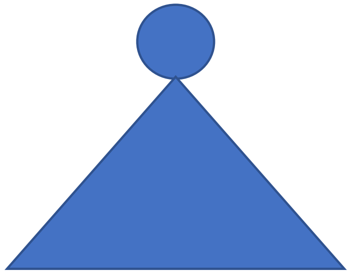


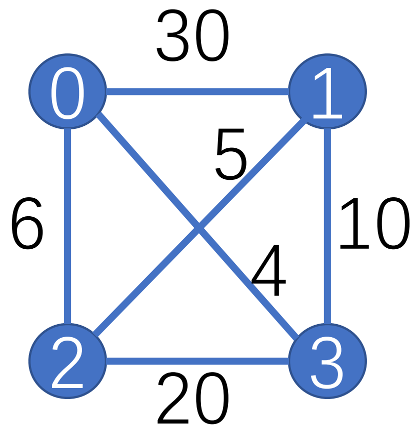
- 对任一状态N，
  其代表子树的最小的界是多少？



初始状态的(下)界

- 初始状态从顶点v_0出发
- 设路径为(v_0, v_1, v_2, v_3)





$w(v_0v_1)+ w(v_1v_2)+ w(v_2v_3)+ w(v_3v_0)$的值最小不小于?

$w(v_0v_1)+ w(v_3v_0)>= v_0$关联的2个最小边权之和

$w(v_0v_1)+ w(v_1v_0)>= v_1$关联的2个最小边权之和

$w(v_1v_2)+ w(v_2v_3)>= v_2$关联的2个最小边权之和

$w(v_2v_3)+ w(v_3v_0)>= v_3$关联的2个最小边权之和

初始状态的(下)界

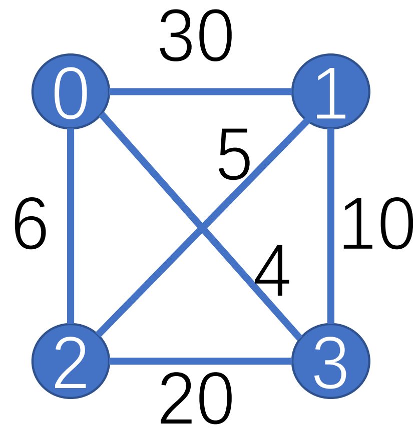
$2(w(v_0v_1)+ w(v_1v_2)+ w(v_2v_3)+ w(v_3v_0))>=$

(v_0关联的2个最小边权之和+

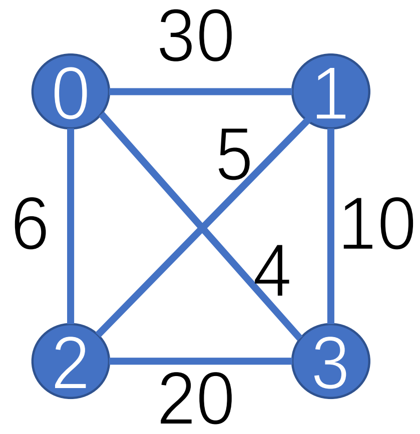
v_1关联的2个最小边权之和+

v_2关联的2个最小边权之和+

v_3关联的2个最小边权之和)



初始状态的(下)界



$w(v_0v_1)+ w(v_1v_2)+ w(v_2v_3)+ w(v_3v_0)>=$

(v_0关联的2个最小边权之和+

v_1关联的2个最小边权之和+

v_2关联的2个最小边权之和+

v_3关联的2个最小边权之和)/2

lb = [(4+6)+(5+10)+(6+5)+(4+10)]/2 = 50/2=25

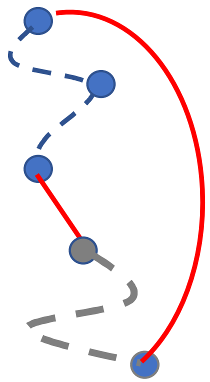
某个状态N的下界lb?

- 设N = (v_0,v_1,⋯,v_k)已经从顶点v_0到达了v_k,其

$$\text{lb}= \sum_{i=0}^{k-1} w(v_i, v_{i+1})$$

$$+[\sum_{i=0,k} \text{不在路径上的最小边权}$$

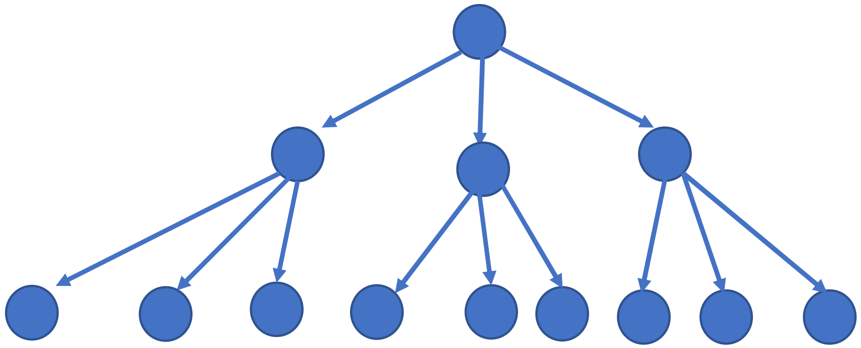
$$+\sum_{i=k+1}^{n}\text{不在路径上的顶点}v_i\text{关联的2个最小边权之和}]/2$$



当前的最优价值V=？

$V = \infty$

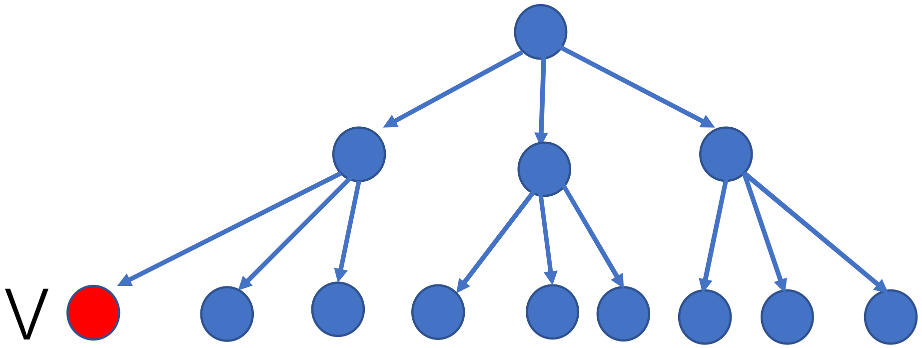
- 如果$V = \infty$，采用广度优先搜索



当前的最优价值V=?

V= ∞

- 如果V=∞，采用广度优先搜索

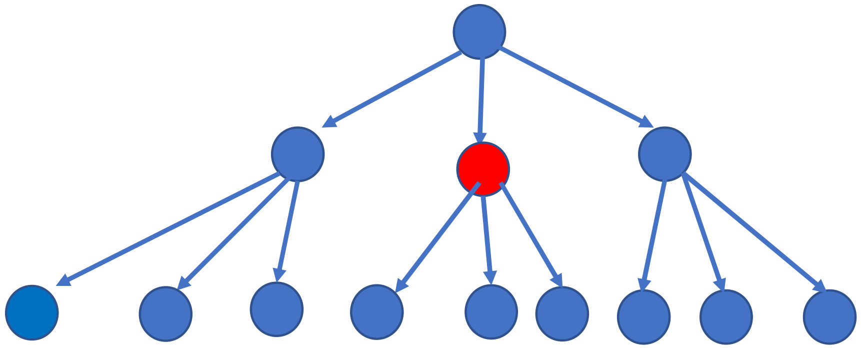


- 需要到达叶子结点层，才能开始用限界剪枝!
- 限界剪枝意义不大了！如何解决?

当前的最优价值V=？

V= ∞

- 如果V=∞，采用广度优先搜索

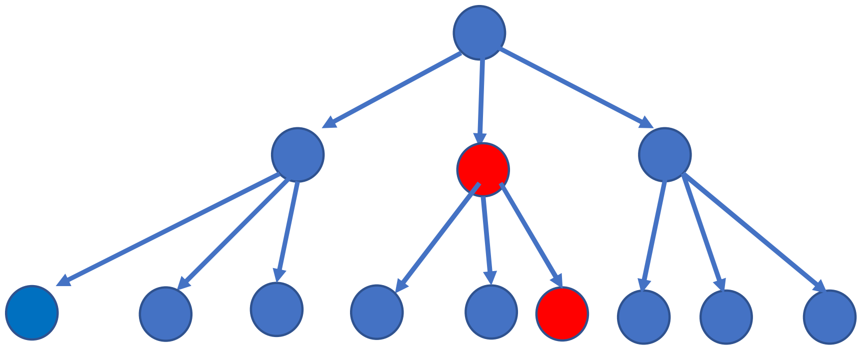


- 深度和广度搜索结合：优先队列，限界小的状态先探索

当前的最优价值V=？

V= ∞

- 如果V=∞，采用广度优先搜索

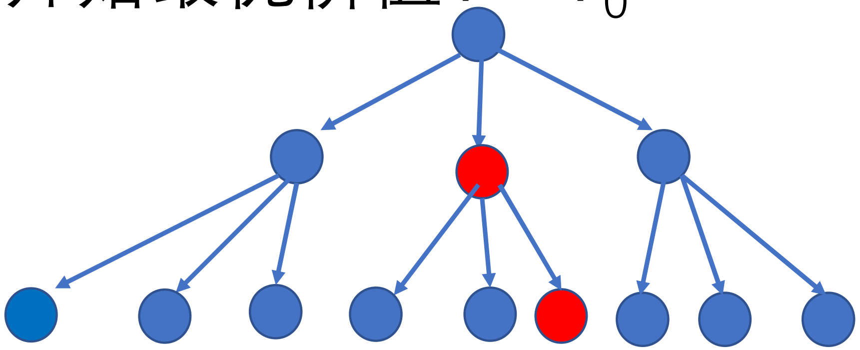


- 深度和广度搜索结合：优先队列，限界小的状态先探索

当前的最优价值V=?

$V=\infty$

- 一开始采用其他算法得到一个可行解，其价值V_0作为开始最优价值$V=V_0$

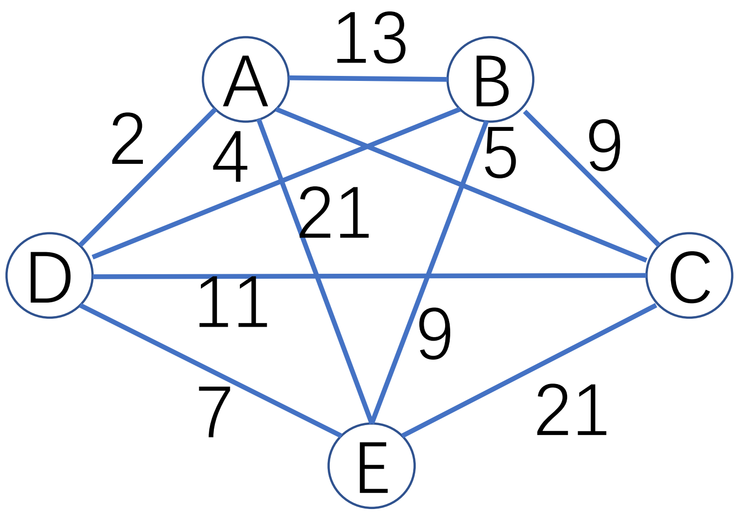


- 再广度优先搜索或者基于优先队列的广度与深度结合搜索

最近邻方法 Nearest Neighbor Method

- 随机选择出发城市
- while 还有未访问城市

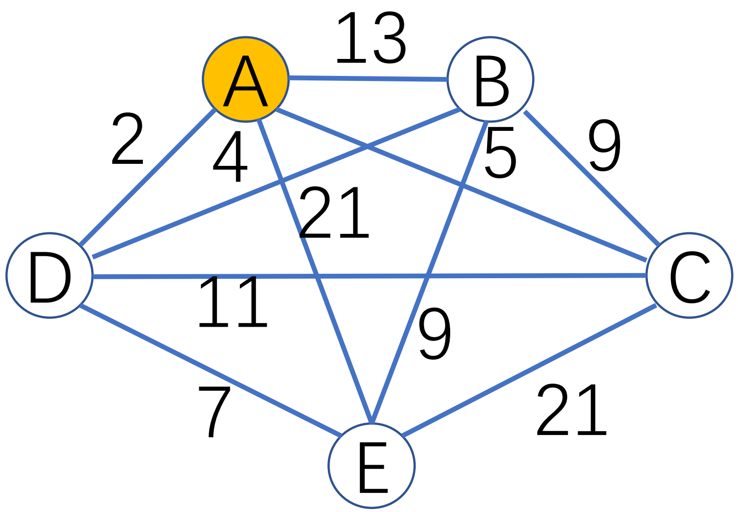
  走到距出发城市最近的未访问城市



最近邻方法 Nearest Neighbor Method

- 随机选择出发城市
- while 还有未访问城市

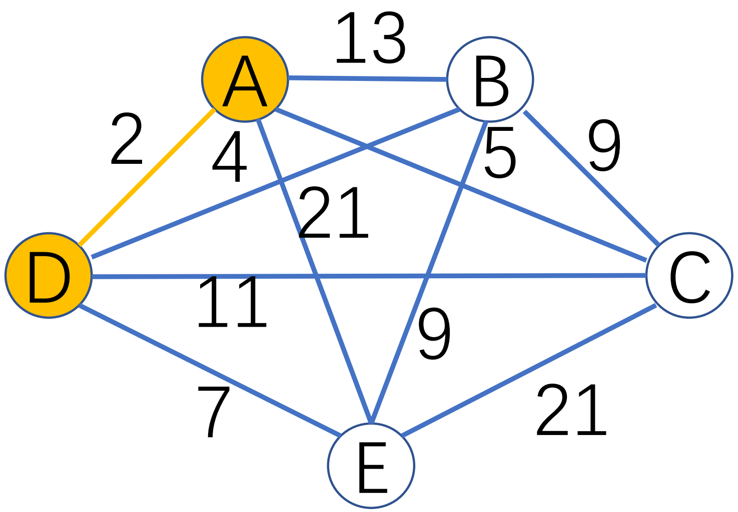
    走到距出发城市最近的未访问城市



最近邻方法 Nearest Neighbor Method

- 随机选择出发城市
- while 还有未访问城市

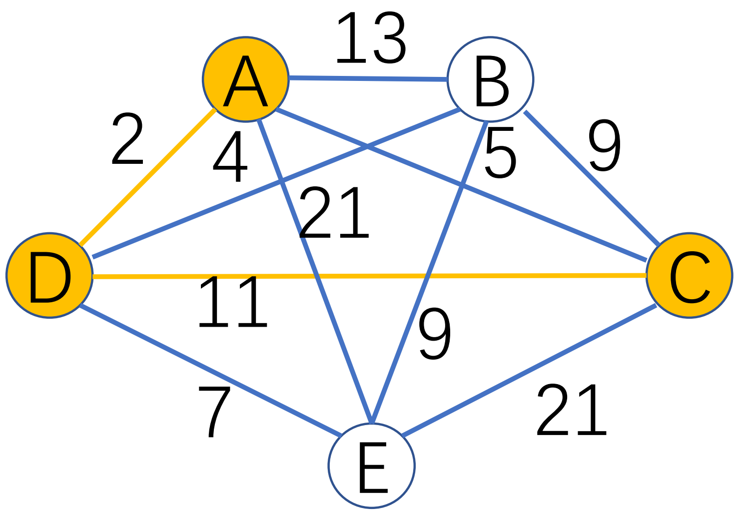
    走到距出发城市最近的未访问城市



最近邻方法 Nearest Neighbor Method

- 随机选择出发城市
- while 还有未访问城市

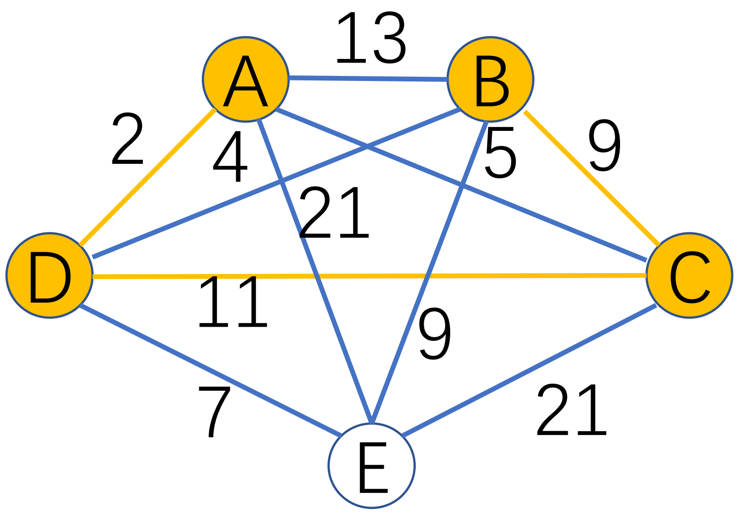
  走到距出发城市最近的未访问城市



最近邻方法 Nearest Neighbor Method

- 随机选择出发城市
- while 还有未访问城市

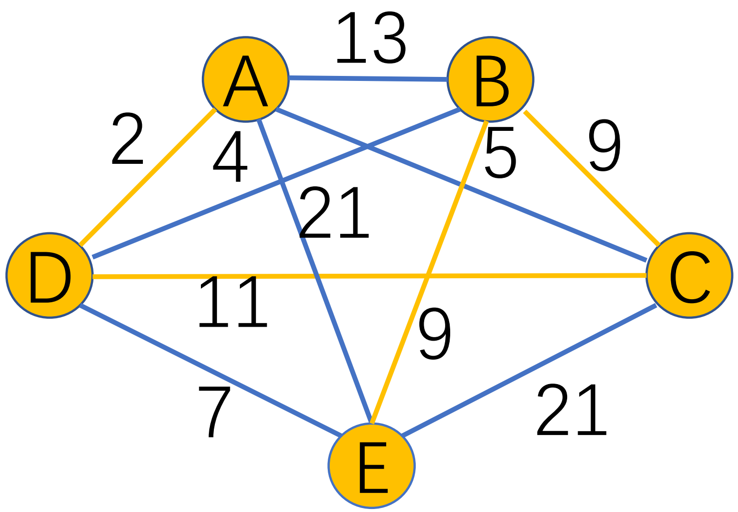
    走到距出发城市最近的未访问城市



最近邻方法 Nearest Neighbor Method

- 随机选择出发城市
- while 还有未访问城市

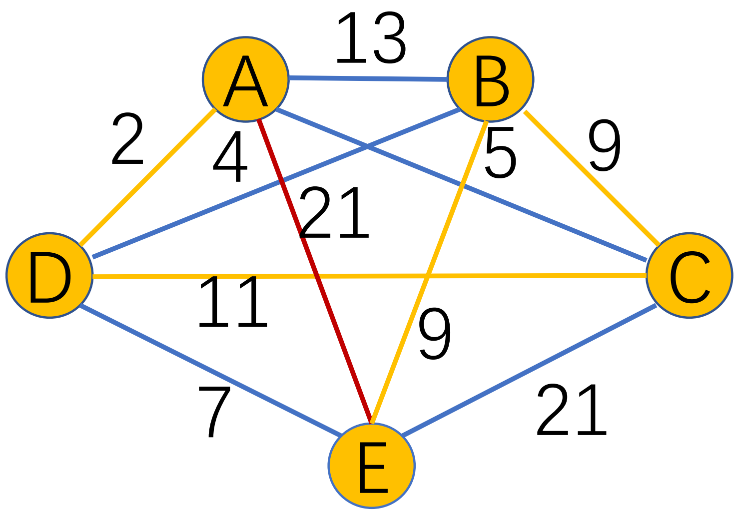
  走到距出发城市最近的未访问城市



最近邻方法 Nearest Neighbor Method

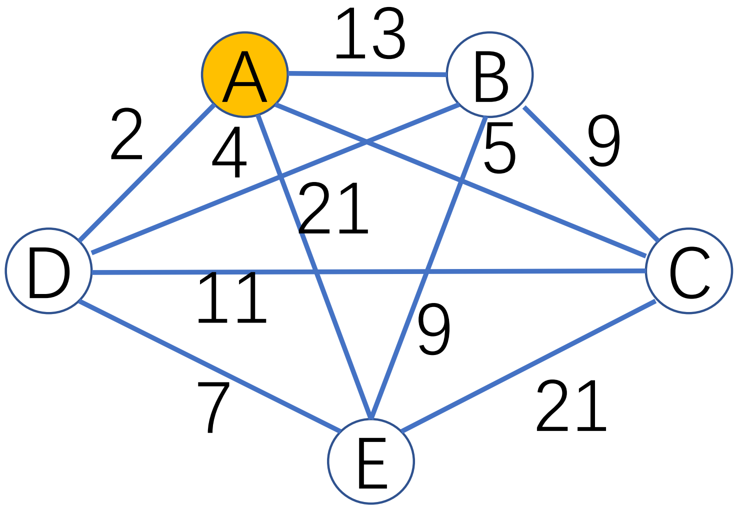
- 随机选择出发城市
- while 还有未访问城市

  走到距出发城市最近的未访问城市



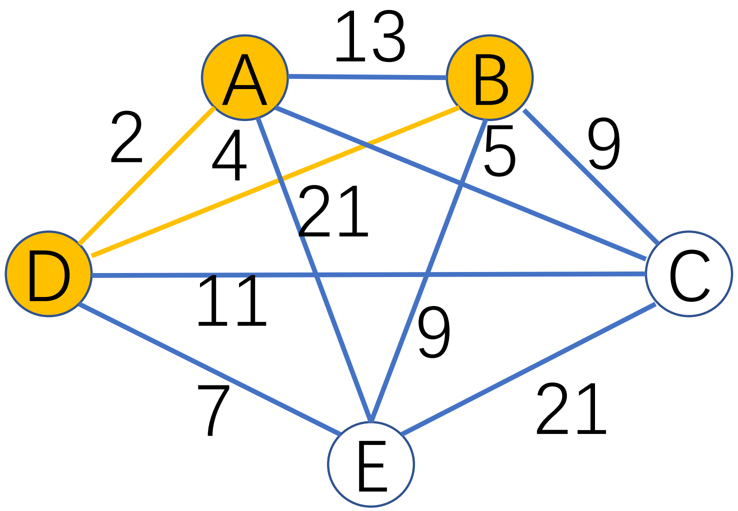
贪婪法 Greedy

- 总是选择距离当前城市最近的未访问城市



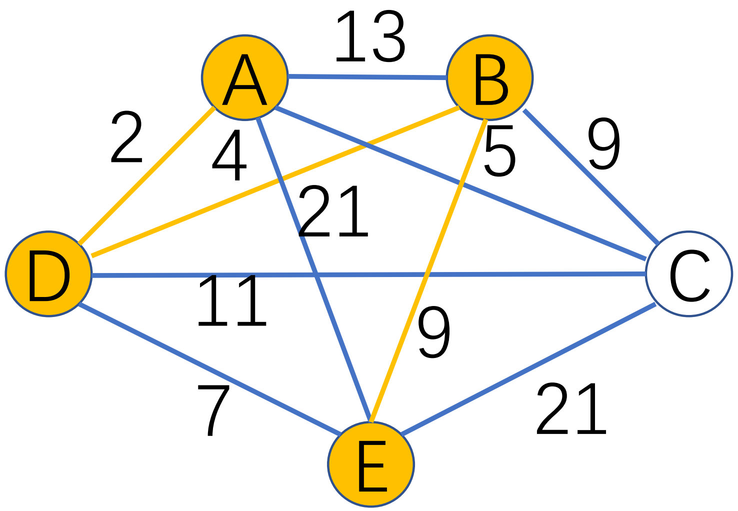
贪婪法 Greedy

- 总是选择距离当前城市最近的未访问城市



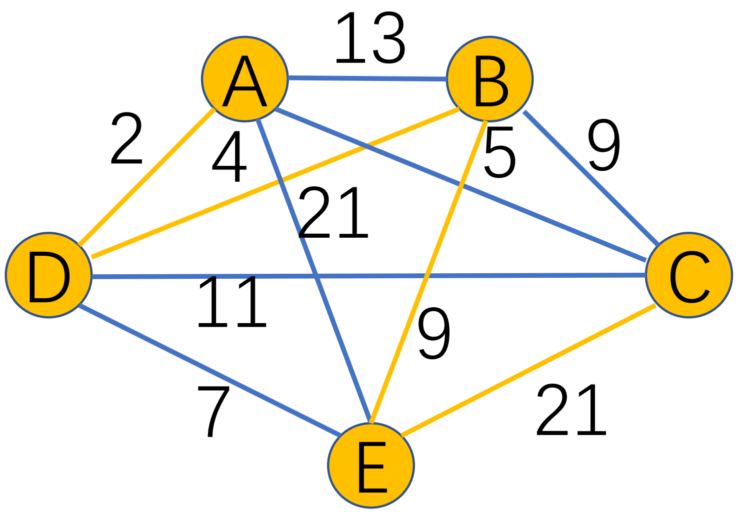
贪婪法 Greedy

- 总是选择距离当前城市最近的未访问城市



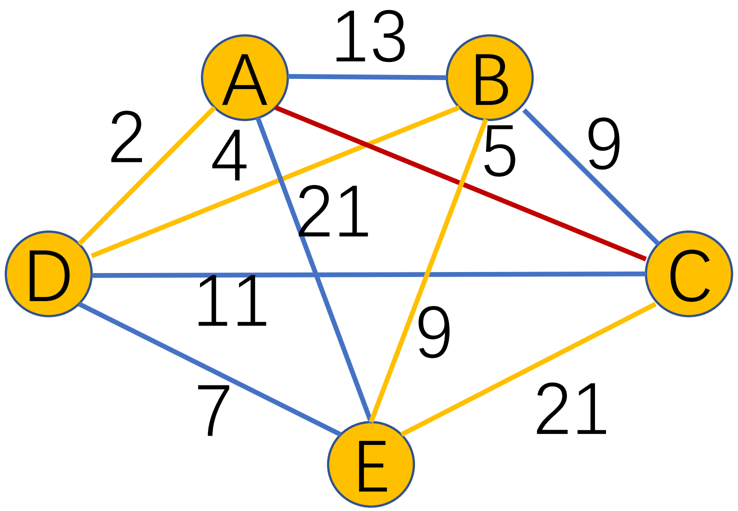
贪婪法 Greedy

- 总是选择距离当前城市最近的未访问城市



贪婪法 Greedy

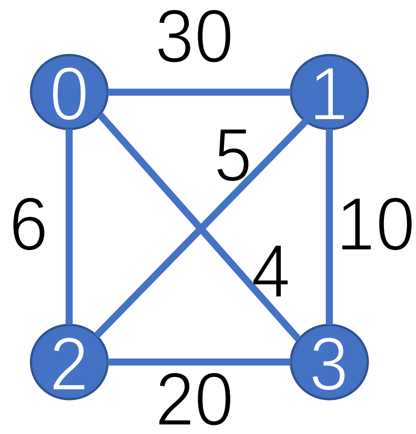
- 总是选择距离当前城市最近的未访问城市



TSP用贪婪法得到一个可行解

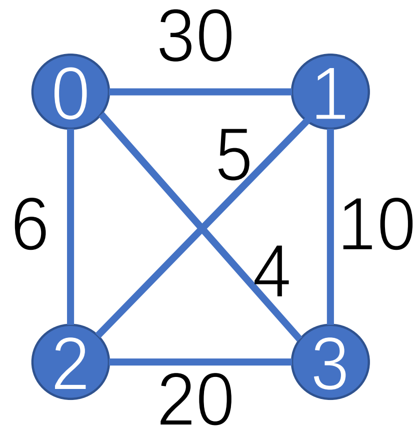
- 0->3->1->2->0
- 4+10+5+6 = 25
- 因此，可行解的价值是25，
- 作为最小值问题的上界V = 25，

其他解的价值只有低于V，才能更优。

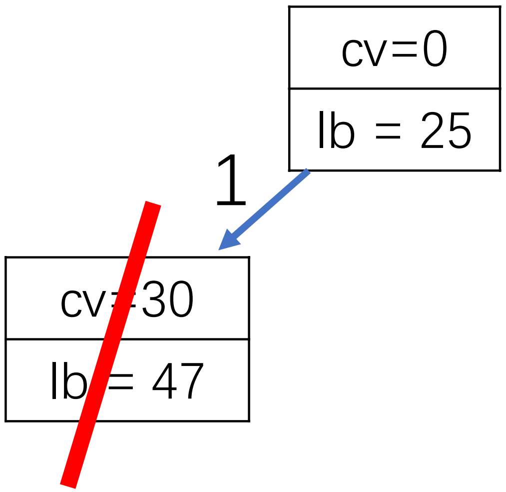


cv=0
lb = 25

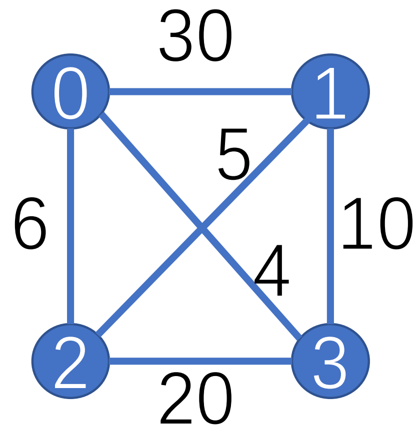
V=25



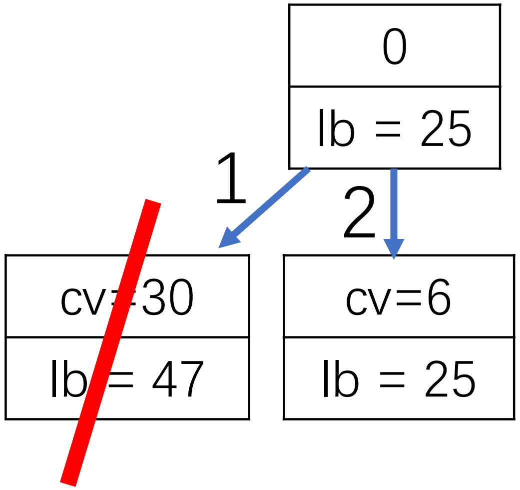
$$lb = [(4+6)+(5+10)+(6+5)+(4+10)]/2 = 50/2=25$$



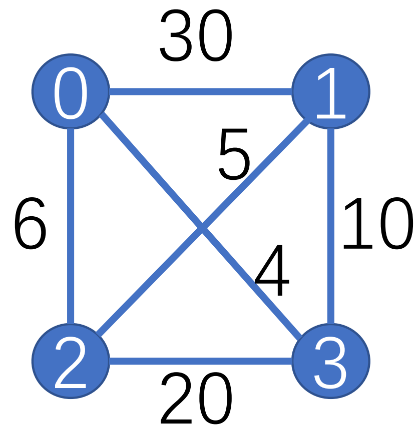
V=25



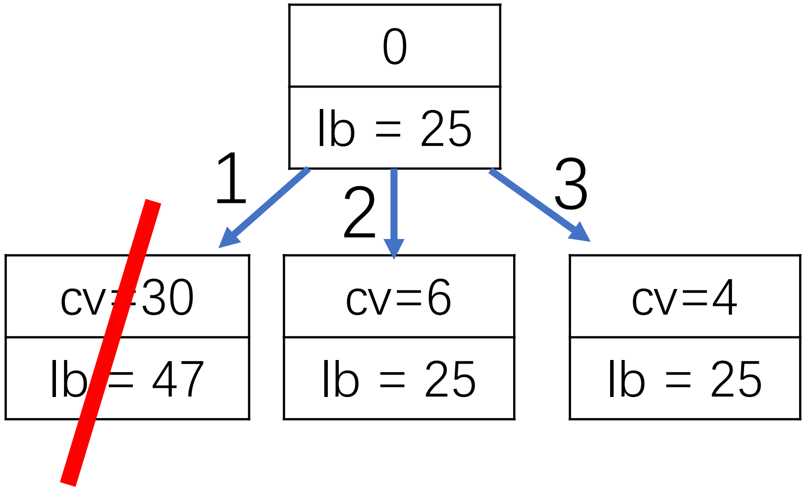
$$lb = 30+[(4+5)+(5+6)+(4+10)]/2 = 30+34/2=47$$



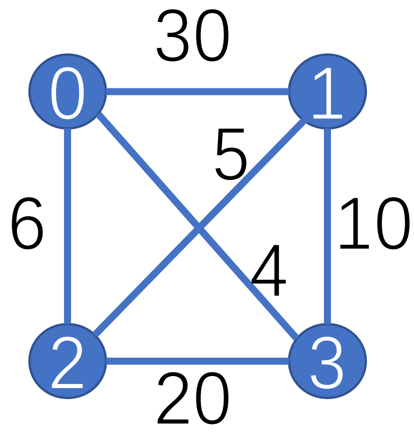
V=25



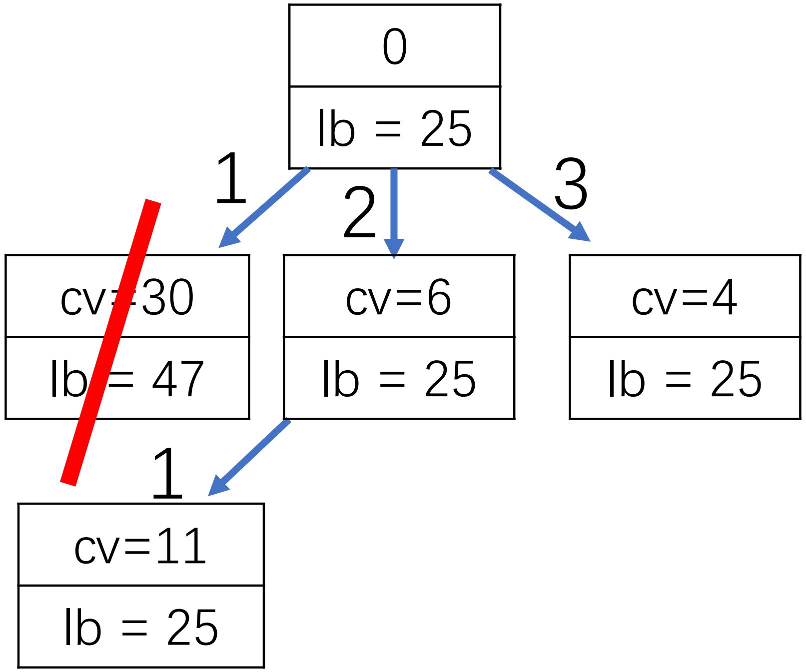
$$lb = 6+[(4+5)+(5+10)+(4+10)]/2 = 6+19=25$$



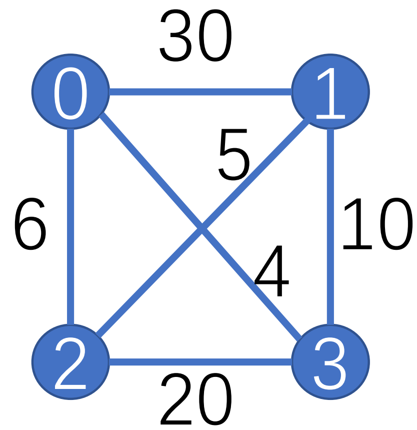
V=25



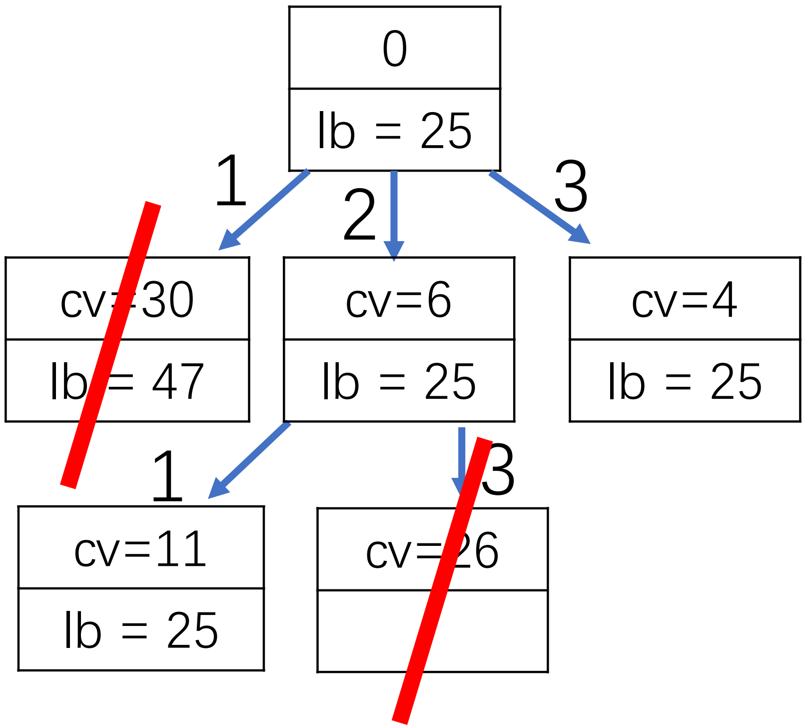
$$lb = 4+[(6+10)+(5+10)+(6+5)]/2 = 4+21=25$$



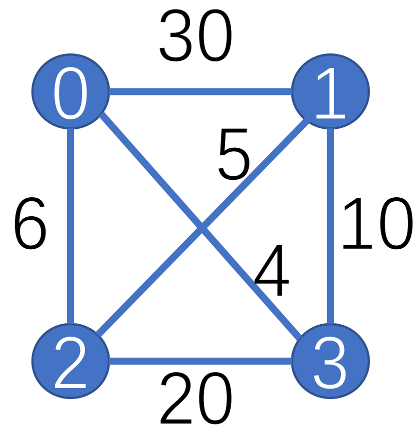
V=25

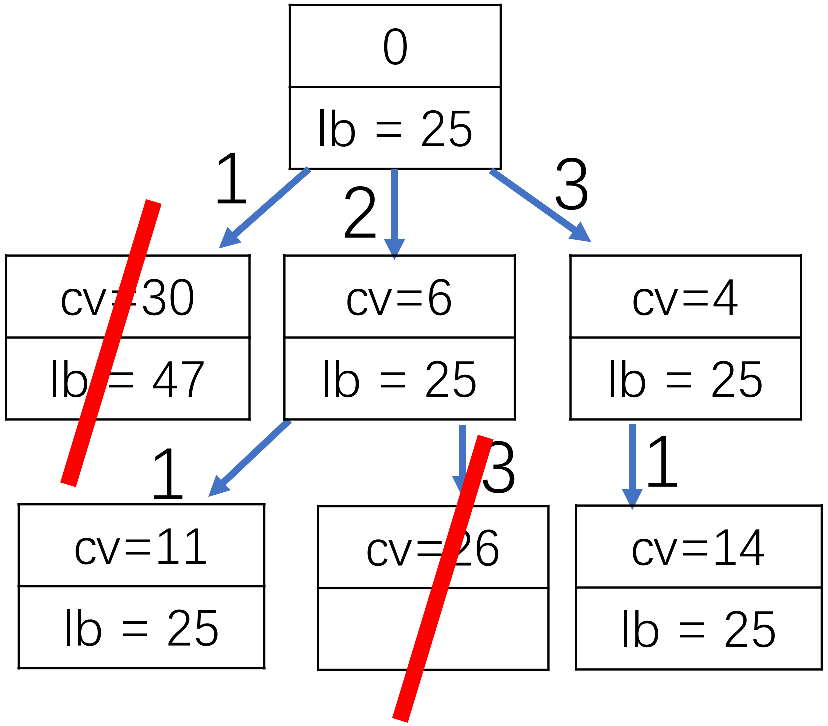


$$lb = 11+[(4+10)+(4+10)]/2 = 11+14=25$$

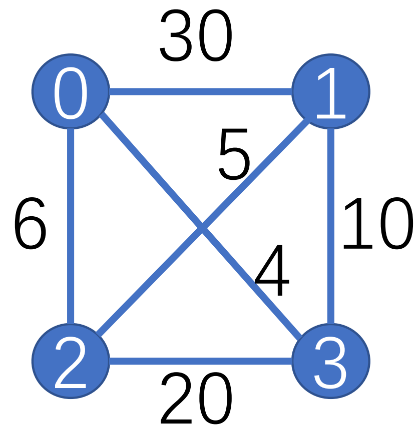

0
lb = 25
1
2
3
cv=30
lb = 47
cv=6
lb = 25
cv=4
lb = 25
1
3
cv=11
lb = 25
cv=26

V=25

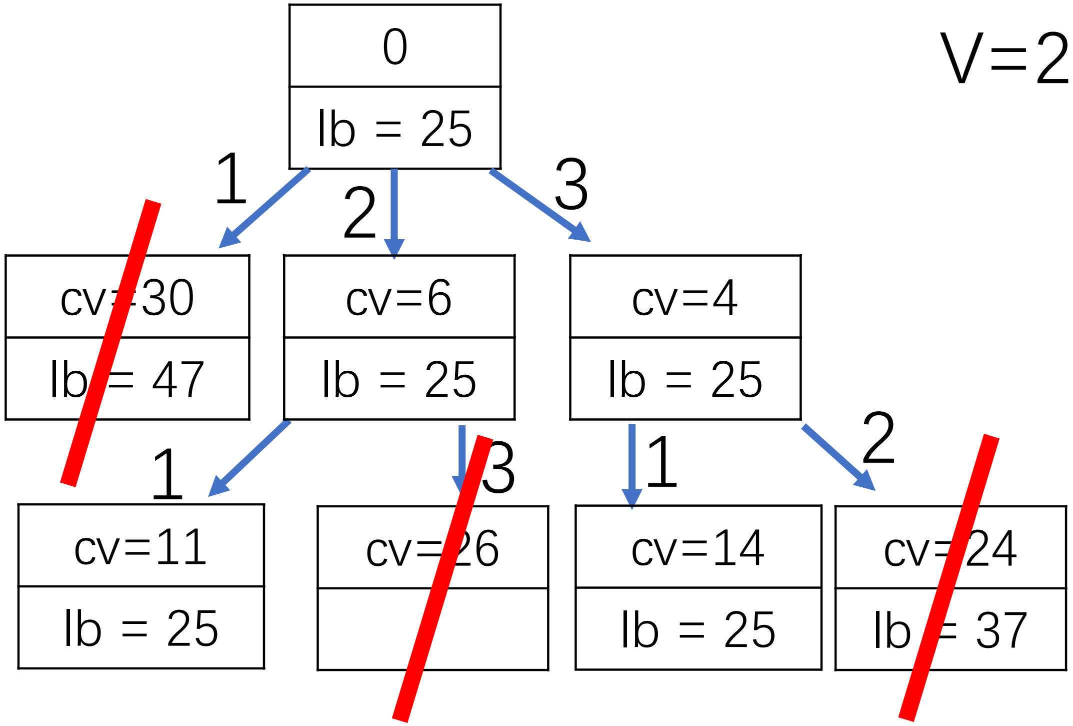

30
0
1
5
6
10
4
2
3
20



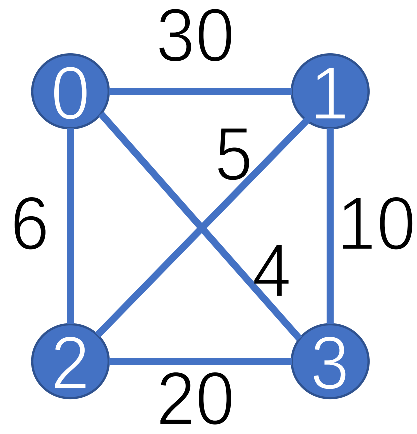
B=25



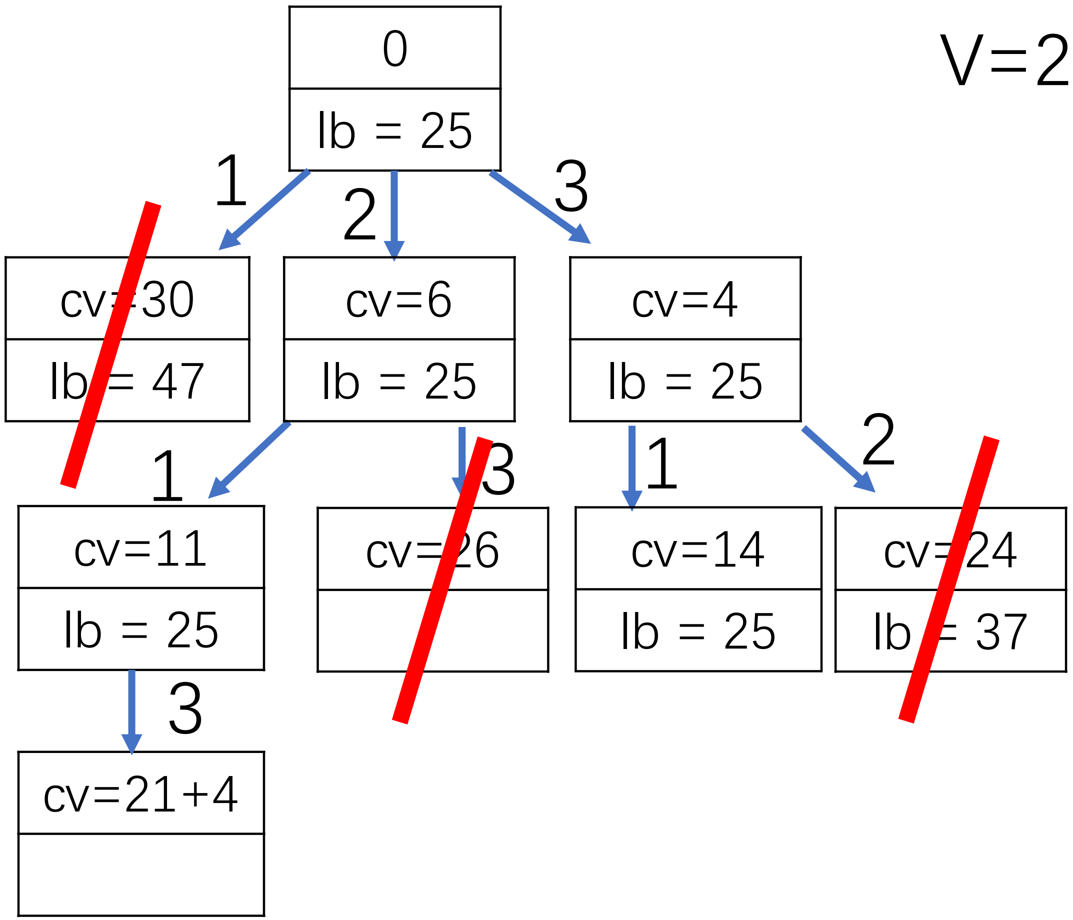
$$lb = 14+[(6+5)+(5+6)]/2 = 14+11=25$$



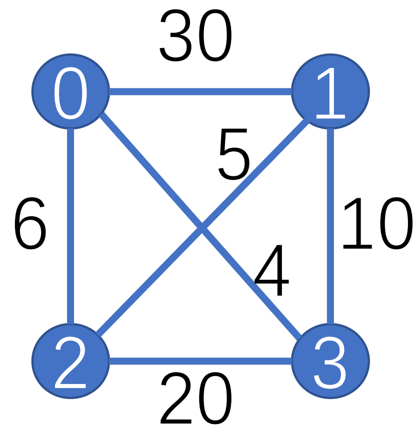
V=25

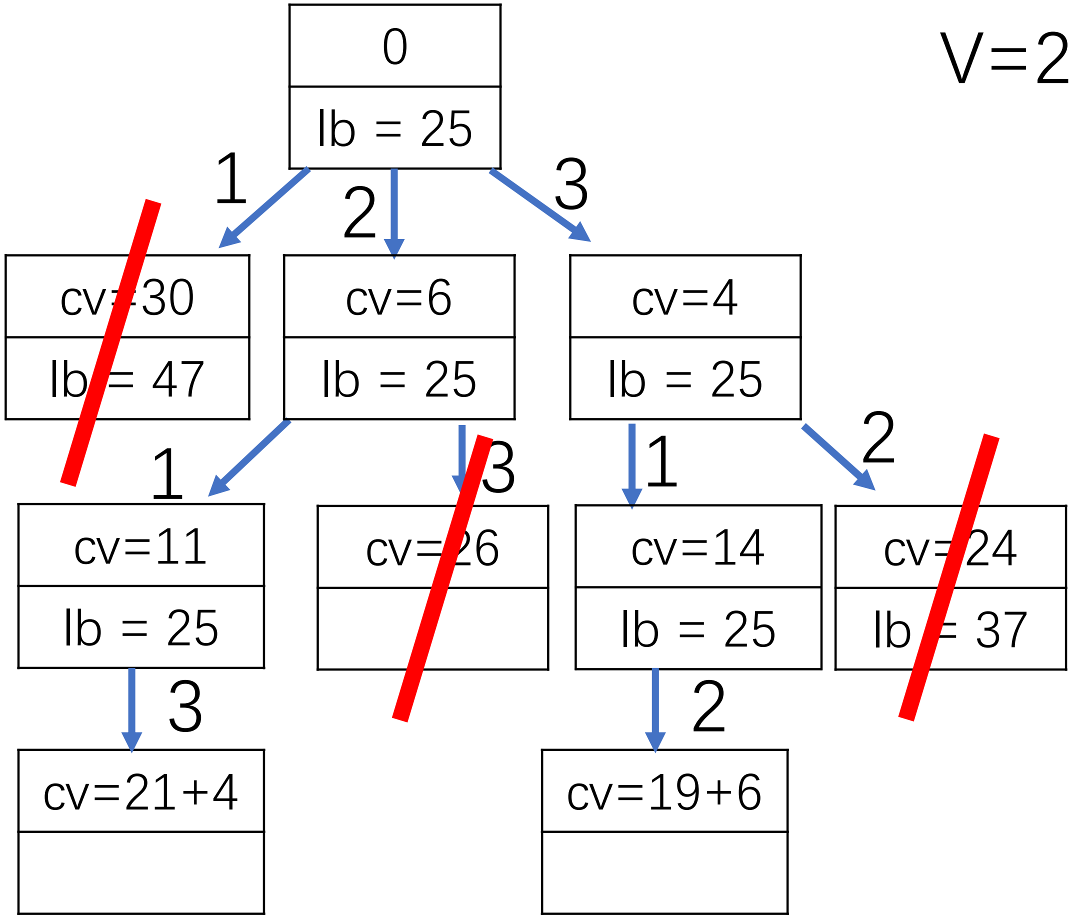


$$lb = 24+[(6+5)+(5+10)]/2 = 24+13=37$$


0
lb = 25
1
2
3
cv=30
lb = 47
cv=6
lb = 25
cv=4
lb = 25
1
3
1
2
cv=11
lb = 25
cv=26
cv=14
lb = 25
cv=24
lb = 37
3
cv=21+4

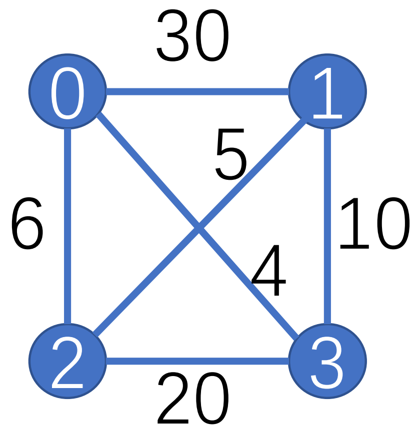
V=25


30
0
1
5
6
10
4
2
3
20

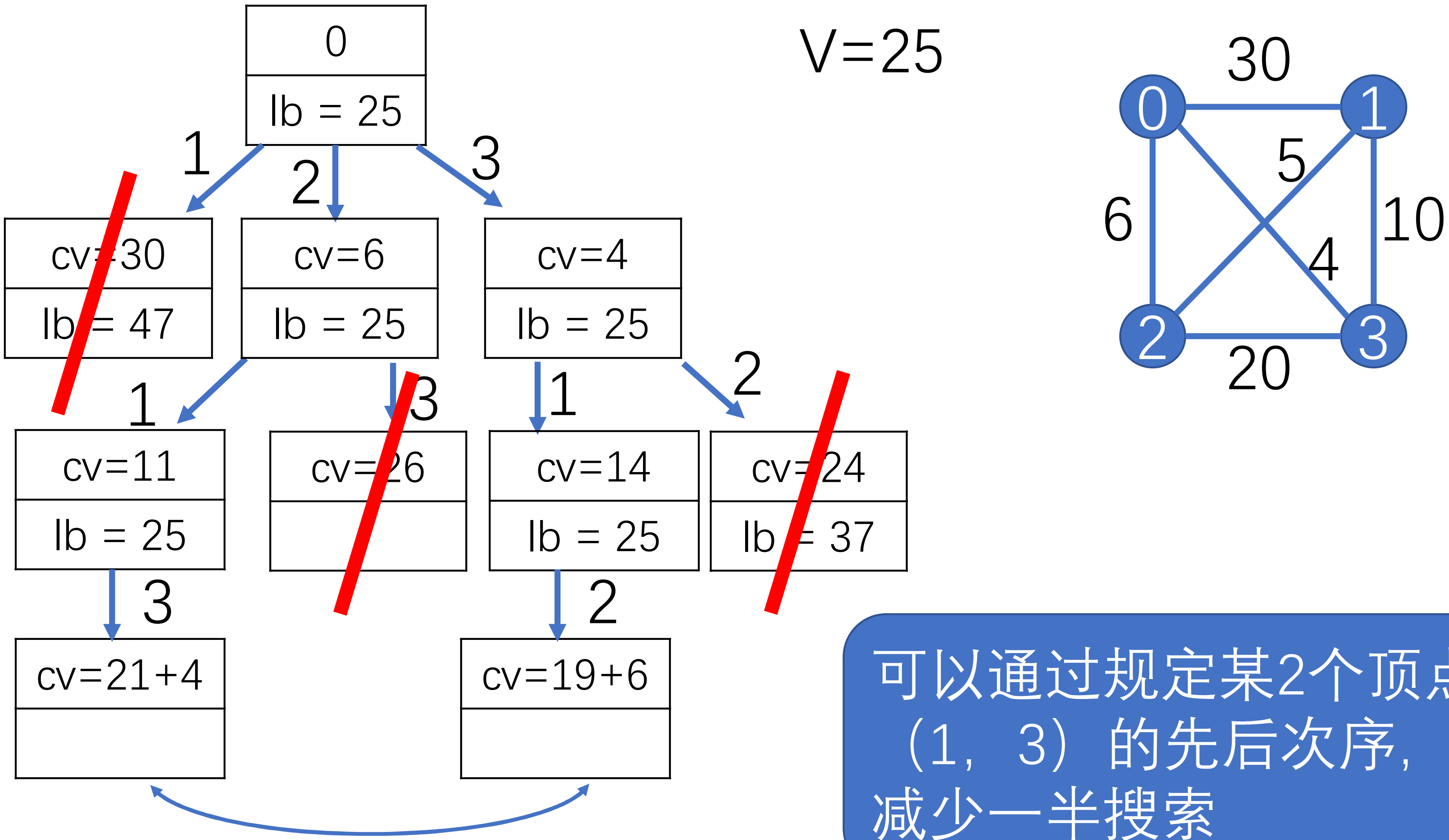


0
lb = 25
1
2
3
cv=30
lb = 47
cv=6
lb = 25
cv=4
lb = 25
1
3
1
2
cv=11
lb = 25
cv=26
cv=14
lb = 25
cv=24
lb = 37
3
2
cv=21+4
cv=19+6

V=25

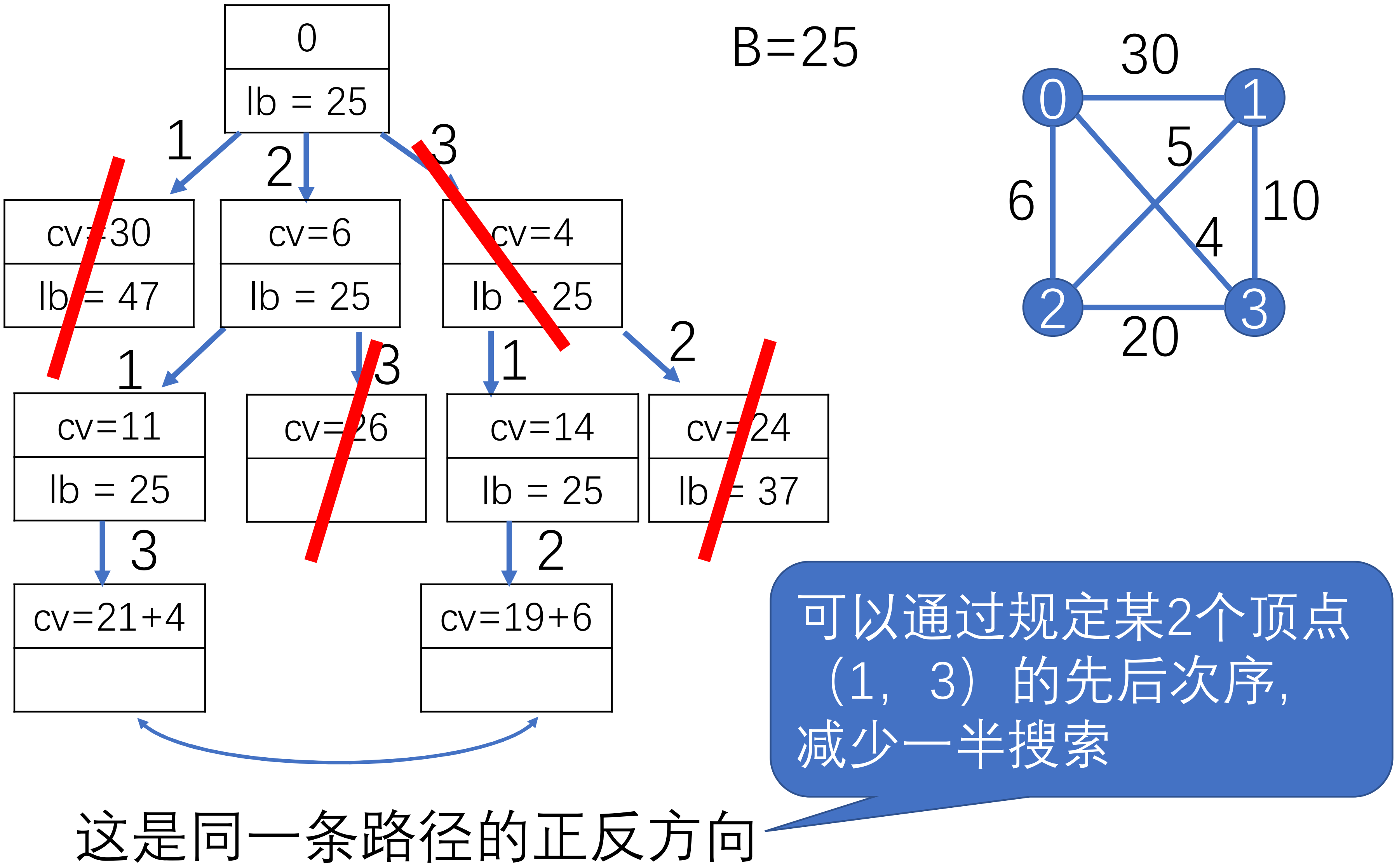


30
0
1
5
6
10
4
2
3
20



可以通过规定某2个顶点（1，3）的先后次序，减少一半搜索

这是同一条路径的正反方向



B=25
0
lb = 25
1
2
3
cv=30
lb = 47
cv=6
lb = 25
cv=4
lb = 25
1
3
1
2
cv=11
lb = 25
cv=26
cv=14
lb = 25
cv=24
lb = 37
3
2
cv=21+4
cv=19+6
30
5
6
10
4
20
0
1
2
3
可以通过规定某2个顶点（1，3）的先后次序，减少一半搜索
这是同一条路径的正反方向

贪婪法：1-2-4-3-1， 价值 V = 10+25+30+15= 80

1

lb = 75

2 3 4

cv=10

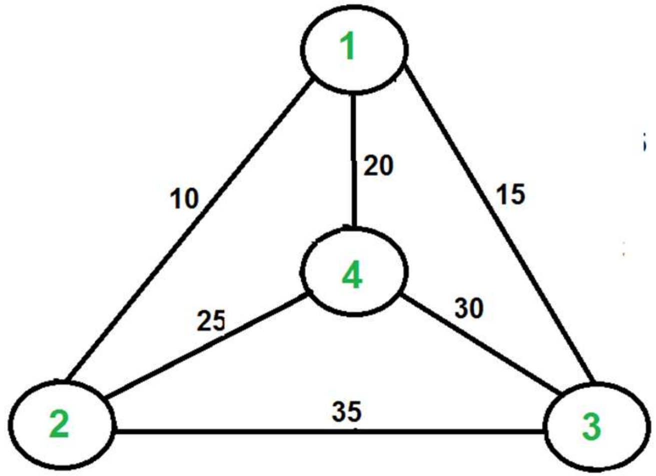
lb = 75

cv=6

lb = 75

cv=20

lb = 77.5



lb(1): [(10+15)+(10+25)+(30+15)+(20+25)]/2 = 150/2=75

lb(1-2): 10+[(15+25)+(30+15)+(20+25)]/2 = 75

lb(1-3): 15+[(10+30)+(10+25)+(20+25)]/2 = 75

lb(1-4): 20+[(10+25)+(10+25)+(15+30)]/2 =77.5

贪婪法：1-2-4-3-1，价值 V = 10+25+30+15= 80

1

lb = 75

2 3 4

cv=10

lb = 75

cv=6

lb = 75

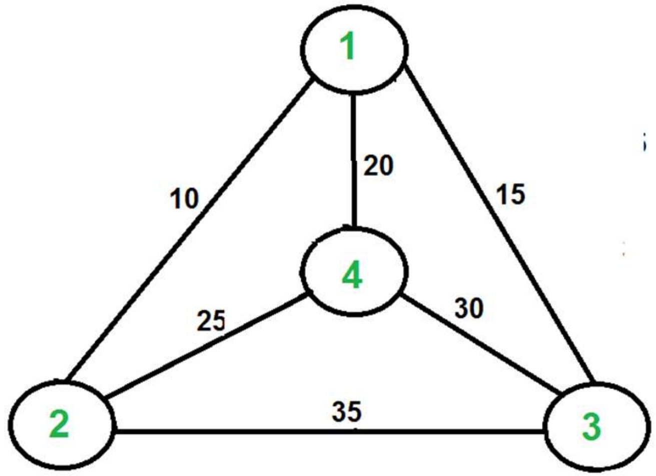
cv=20

lb = 77.5

3

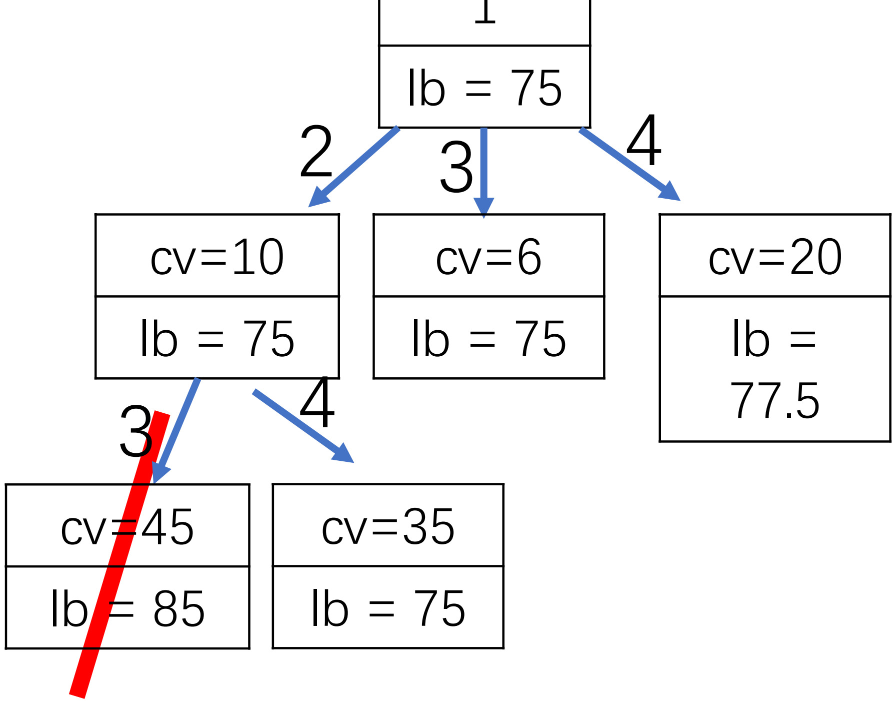
~~cv=45~~

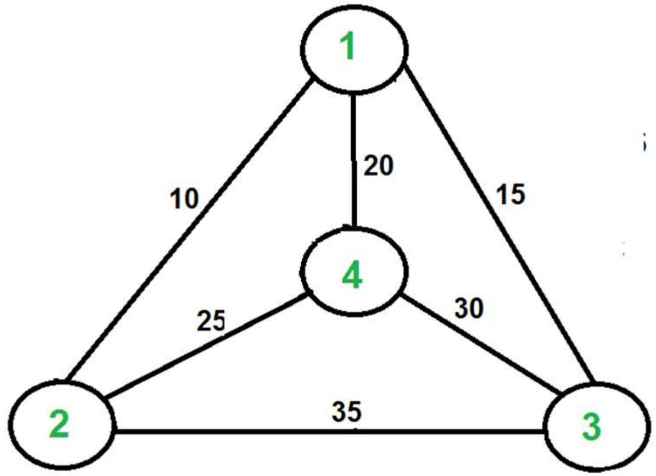
~~lb = 85~~



lb(1-2-3): 45+[(15+15)+(20+30)]/2 =45+40=85

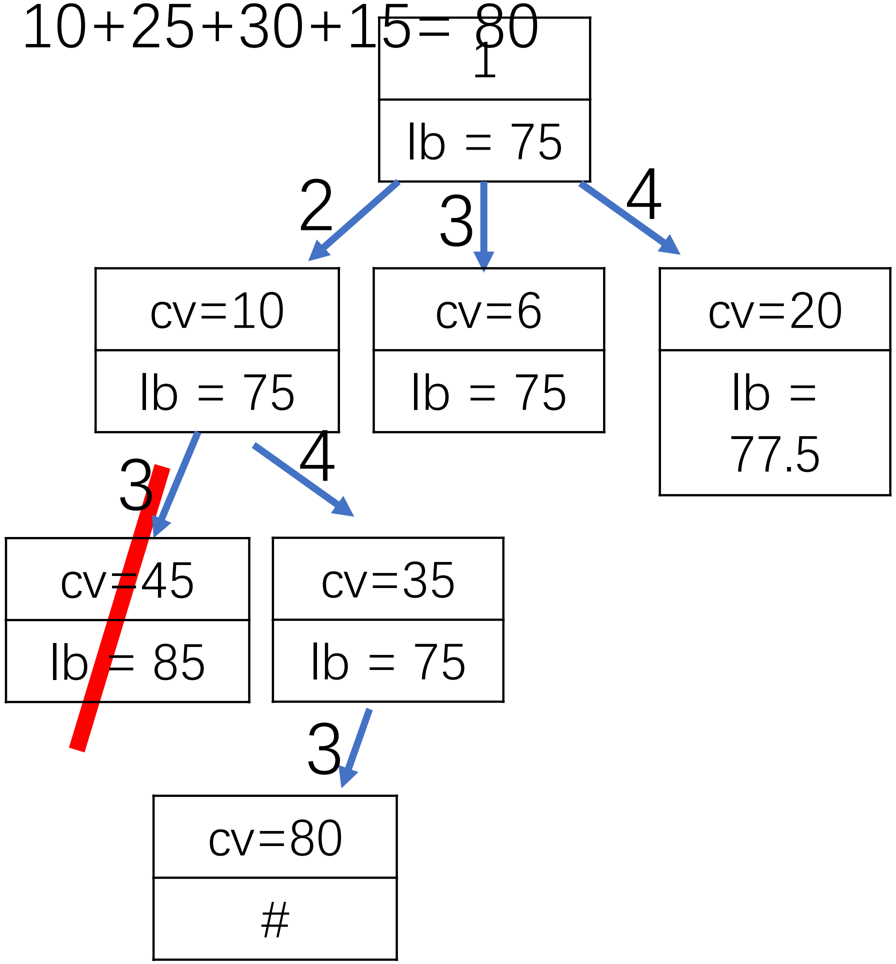
贪婪法：1-2-4-3-1，价值 V = 10+25+30+15= 80

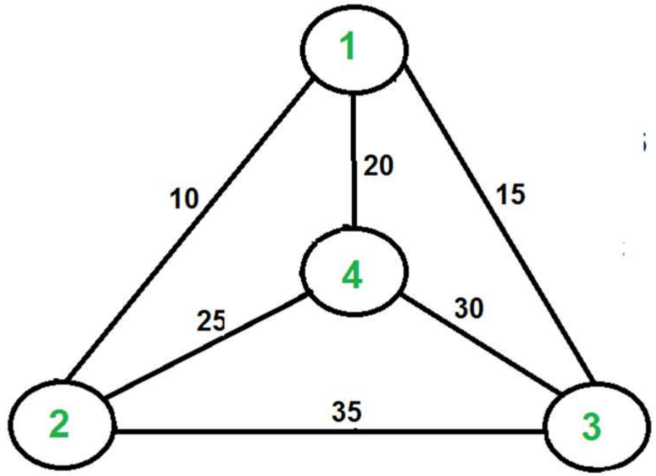




$$lb(1-2-4): 35+[(15+20)+(15+30)]/2 = 35+40=75$$

贪婪法：1-2-4-3-1， 价值 V = 10+25+30+15= 80





内容目录

- 全排列
- 回溯法通用框架

关注我

https://hwdong-net.github.io

Youtube频道:hwdong