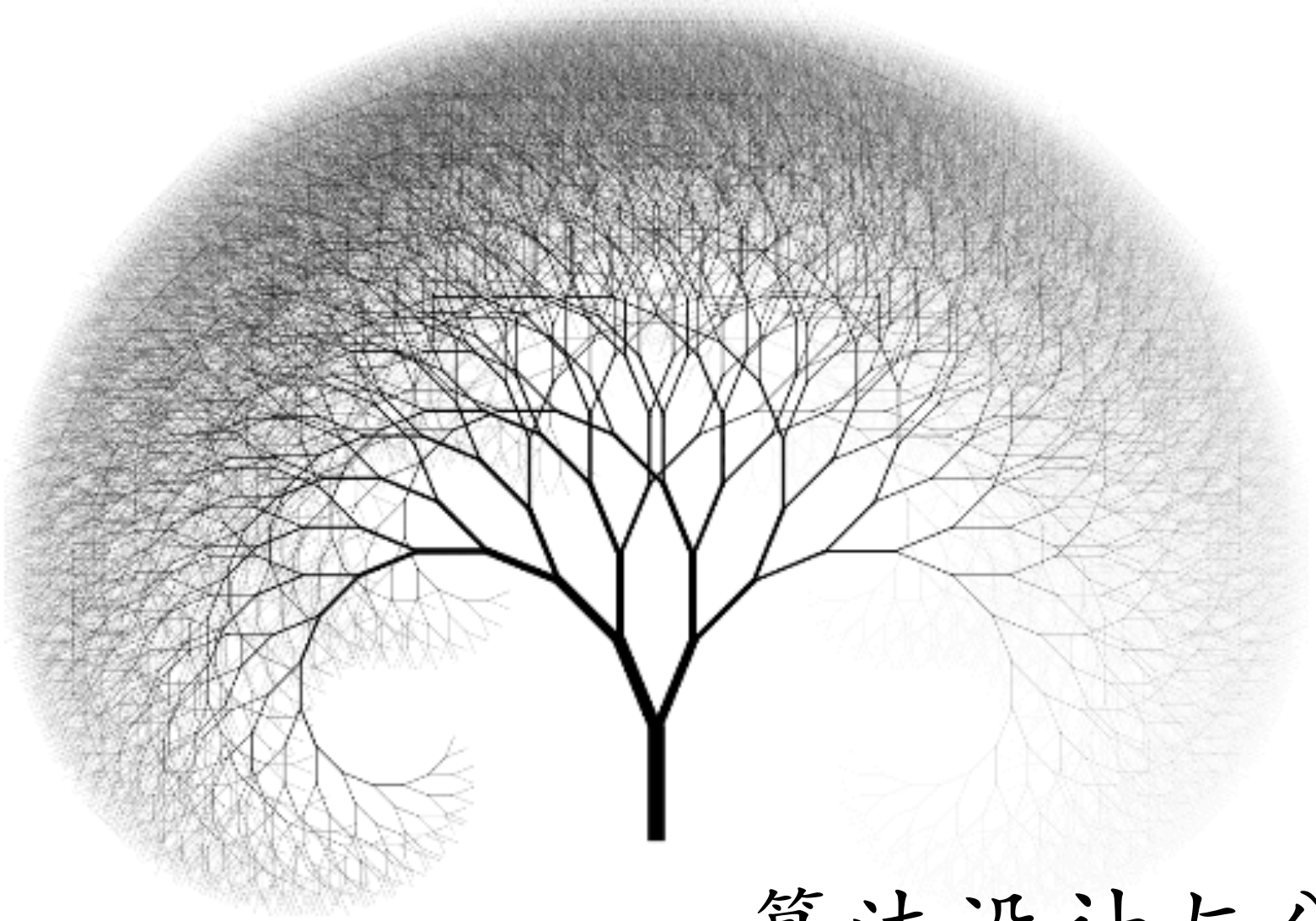


算法设计与分析

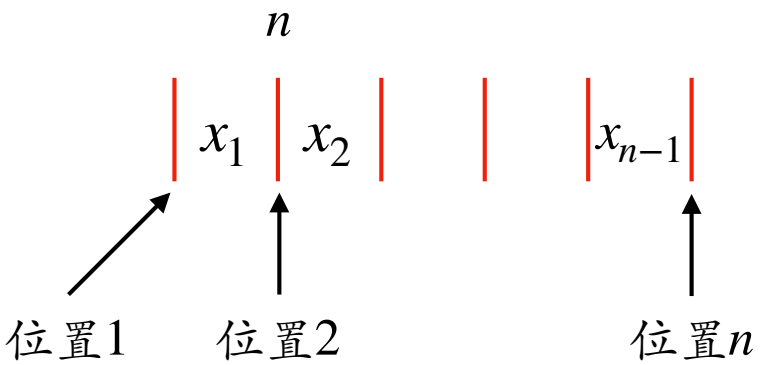
刘 安
苏州大学 计算机科学与技术学院
http://web.suda.edu.cn/anliu/

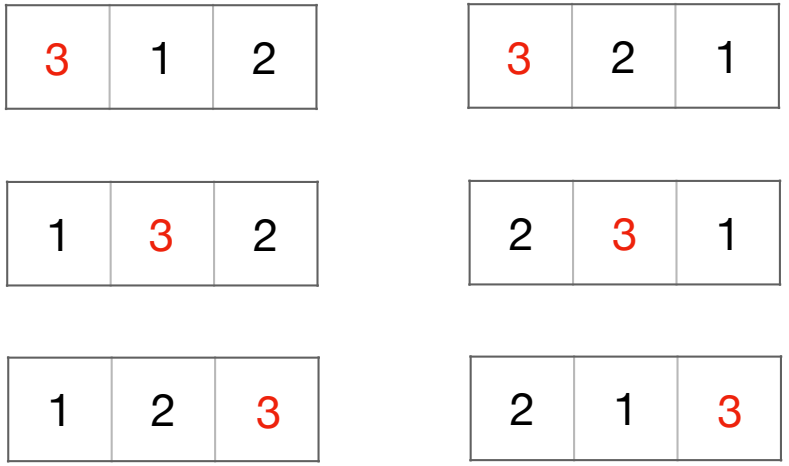
排列

Permutation

构造所有排列

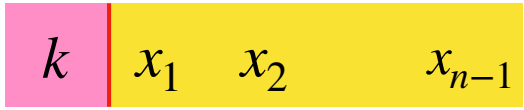
- 问题：构造集合$[n]=\{1,2,\cdots,n\}$的所有排列
- 归纳假设：可以构造集合$[n-1]=\{1,2,\cdots,n-1\}$的所有排列
- 归纳步骤：对于$[n-1]$的每一个排列$x_1,x_2,\cdots,x_{n-1}$，将n放在如下n个不同的位置上，即可得到$[n]$的所有排列
- 基本情况：集合[1]的排列

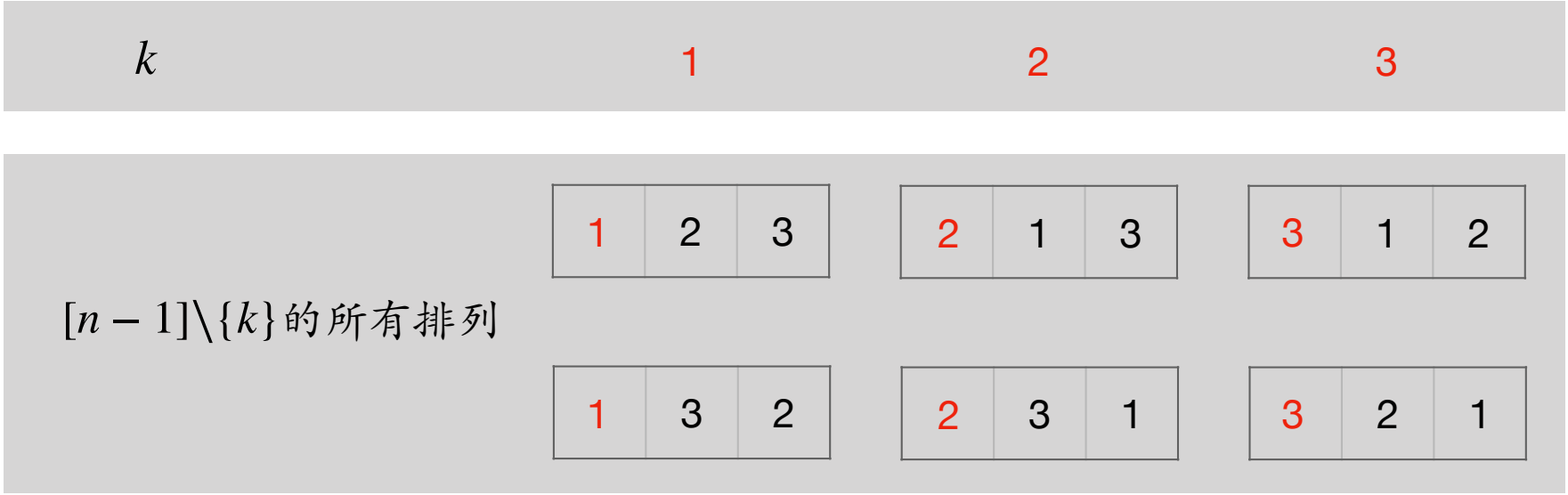




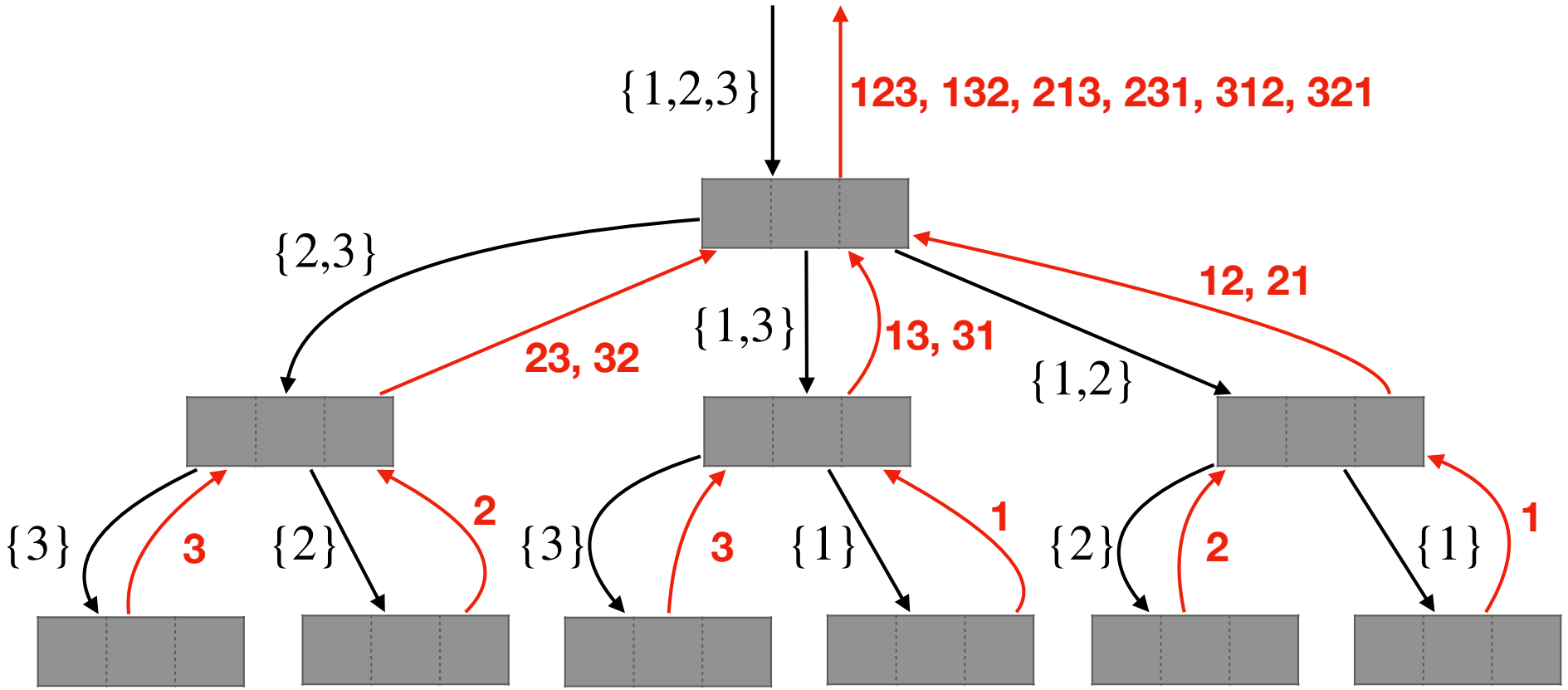
构造所有排列

- 问题：构造集合$[n]=\{1,2,\cdots,n\}$的所有排列
- 归纳假设：可以构造包含$n-1$个元素的集合S的所有排列
- 归纳步骤：对于每一个$k\in[n]$，对于$[n]\backslash\{k\}$的每一个排列$x_1,x_2,\cdots,x_{n-1}$，将k放在$x_1,x_2,\cdots,x_{n-1}$的前面，即可得到$[n]$的所有排列
- 基本情况：包含1个元素的集合

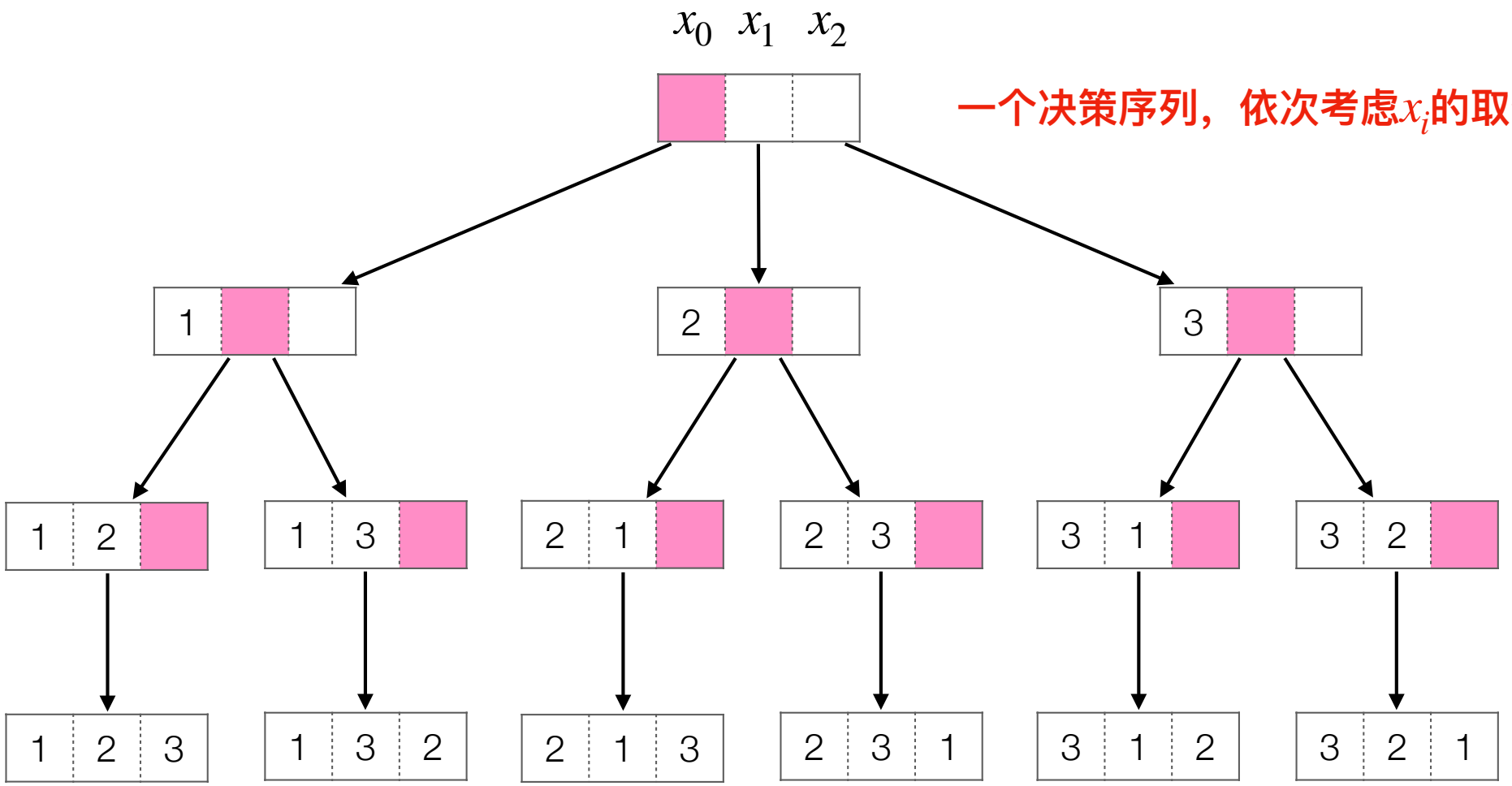




递归调用树

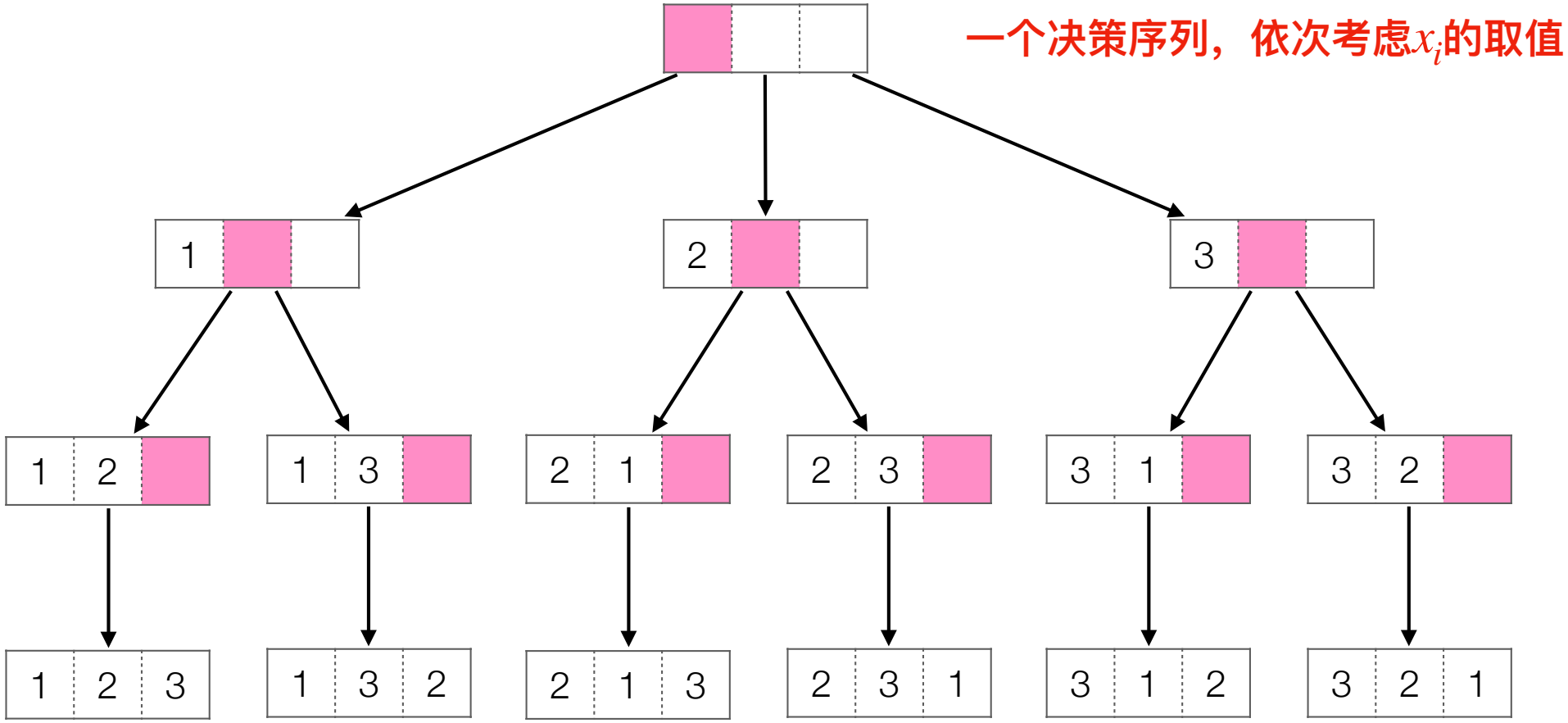

{1,2,3}
123, 132, 213, 231, 312, 321
{2,3}
{1,3}
{1,2}
23, 32
13, 31
12, 21
{3}
3
{2}
2
{3}
3
{1}
1
{2}
2
{1}
1

乘法原理与排列生成


x_0 x_1 x_2
一个决策序列，依次考虑x_i的取值
1
2
3
1 2
1 3
2 1
2 3
3 1
3 2
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

解空间图

- 问题的解是一个序列$X = [x_0, x_1, \cdots]$，其中x_i从有限大小的集合P_i中取值
- 部分解 (partial solution)：$[x_0, x_1, \cdots, x_i]$
- 解的扩展：在部分解末尾$[x_0, x_1, \cdots, x_i]$增加一个元素，成为$[x_0, x_1, \cdots, x_{i+1}]$
- 解的检测：检测$[x_0, x_1, \cdots, x_i]$是否是一个可行解 (feasible solution)



构造所有排列

- 调用perm(0)得到所有排列，其中$C_0 = \{1,2,\cdots,n\}$
- 解空间图上的一次深度优先搜索

procedure perm(i)

Require: global $X = [x_0, x_1, \cdots]$, C_i $(i = 0,1,\cdots)$

01 if X is a valid permutation: //检测是否是可行解

02 process it //处理解

03 else: //否则扩展部分解

04 for each $x \in C_i$: // 遍历可用于扩展的元素

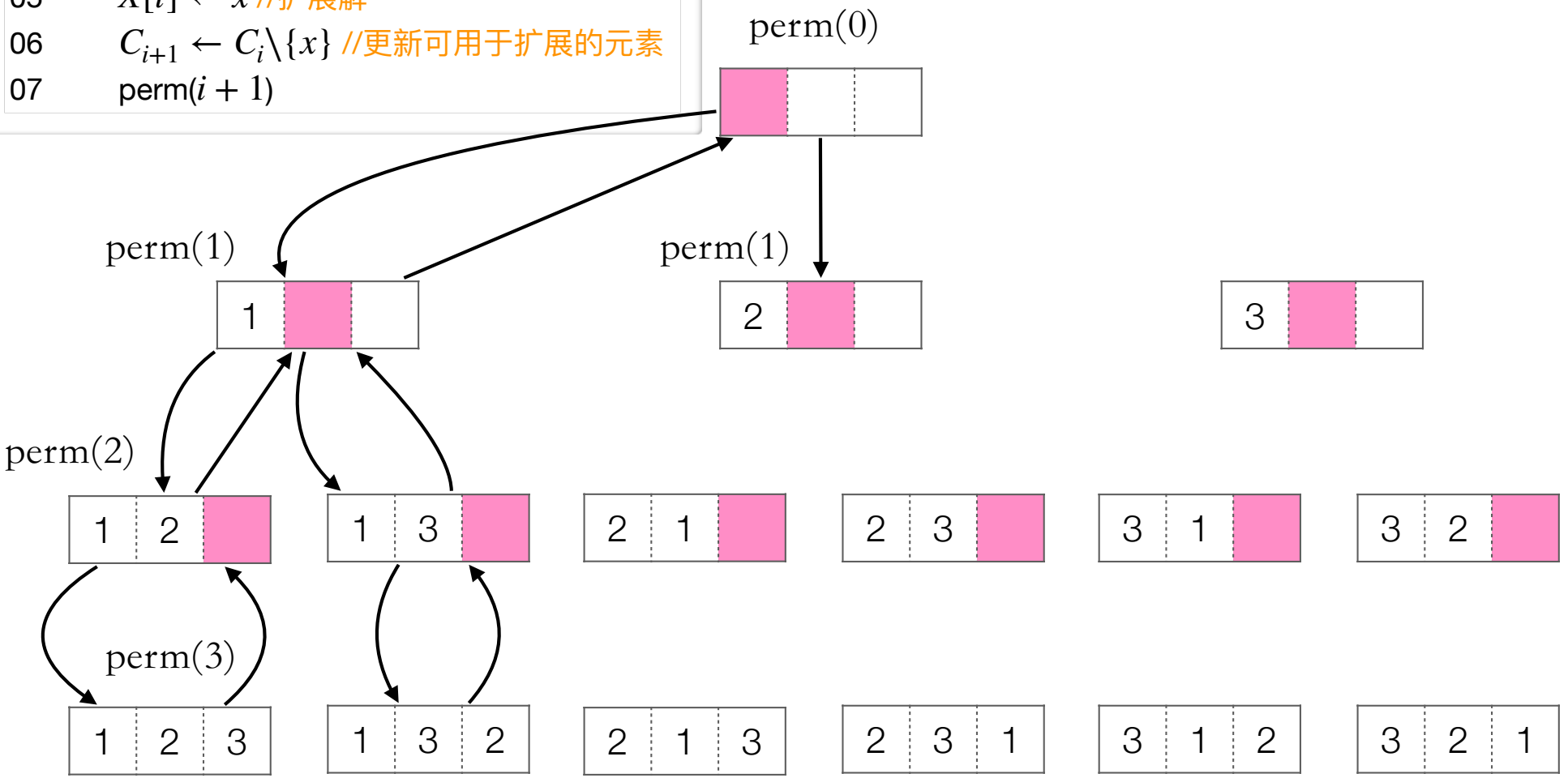
05 $X[i] \leftarrow x$ //扩展解

06 $C_{i+1} \leftarrow C_i \backslash \{x\}$ //更新可用于扩展的元素

07 perm($i + 1$)

解空间图上的深搜

```
procedure perm(i)
01 if X is a valid permutation: //检测是否是可行解
02     process it //处理解
03 else: //否则扩展部分解
04     for each x ∈ C_i: // 遍历可用于扩展的元素
05         X[i] ← x //扩展解
06         C_{i+1} ← C_i\{x} //更新可用于扩展的元素
07         perm(i + 1)
```



Python实现

```
def perm_v3(X, S, i, n):
    if i == n:
        print(X)
    else:
        for e in S:
            X[i] = e
            perm_v3(X, S - set([e]), i + 1, n)

def test_perm_v3(n):
    S = set(list(range(1, n + 1)))
    X = [None] * len(S)
    perm_v3(X, S, 0, n)
```

Python实现

```
def perm_v4(X, i, n):
    if i == n:
        print(X)
    else:
        for j in range(i, n):
            X[i], X[j] = X[j], X[i]
            perm_v4(X, i + 1, n)
            X[i], X[j] = X[j], X[i]

def test_perm_v4(n):
    X = list(range(1, n + 1))
    perm_v4(X, 0, n)
```

回溯

Backtracking

穷举法和回溯

- 穷举 (exhaustive search, brute-force search)
  - 枚举所有可能的情况并逐一检查，从而找到答案
- 目前主流CPU在1秒内可以进行100万次计算
  - 20个元素的子集数、10个元素的排列数
- 回溯：一种系统的穷举方法
  - 问题的解是一个序列$X = [x_0, x_1, \cdots]$，其中x_i从有限集合P_i中取值
  - 回溯法依次考虑$x_0, x_1, \cdots$的取值（扩展解）
  - 每当确定x_i的取值后，需要判断$[x_0, x_1, \cdots, x_i]$是否是可行的完整解
    - 是：输出解，统计解的数量，更新最优解，等等
    - 否：是否可以继续扩展解
      - 是：扩展解$[x_0, x_1, \cdots, x_i] \to [x_0, x_1, \cdots, x_{i+1}]$
      - 否：回溯（剪枝，prune），考虑x_{i-1}的另一个取值

回溯算法的基本框架

Algorithm 1 A general backtracking algorithm

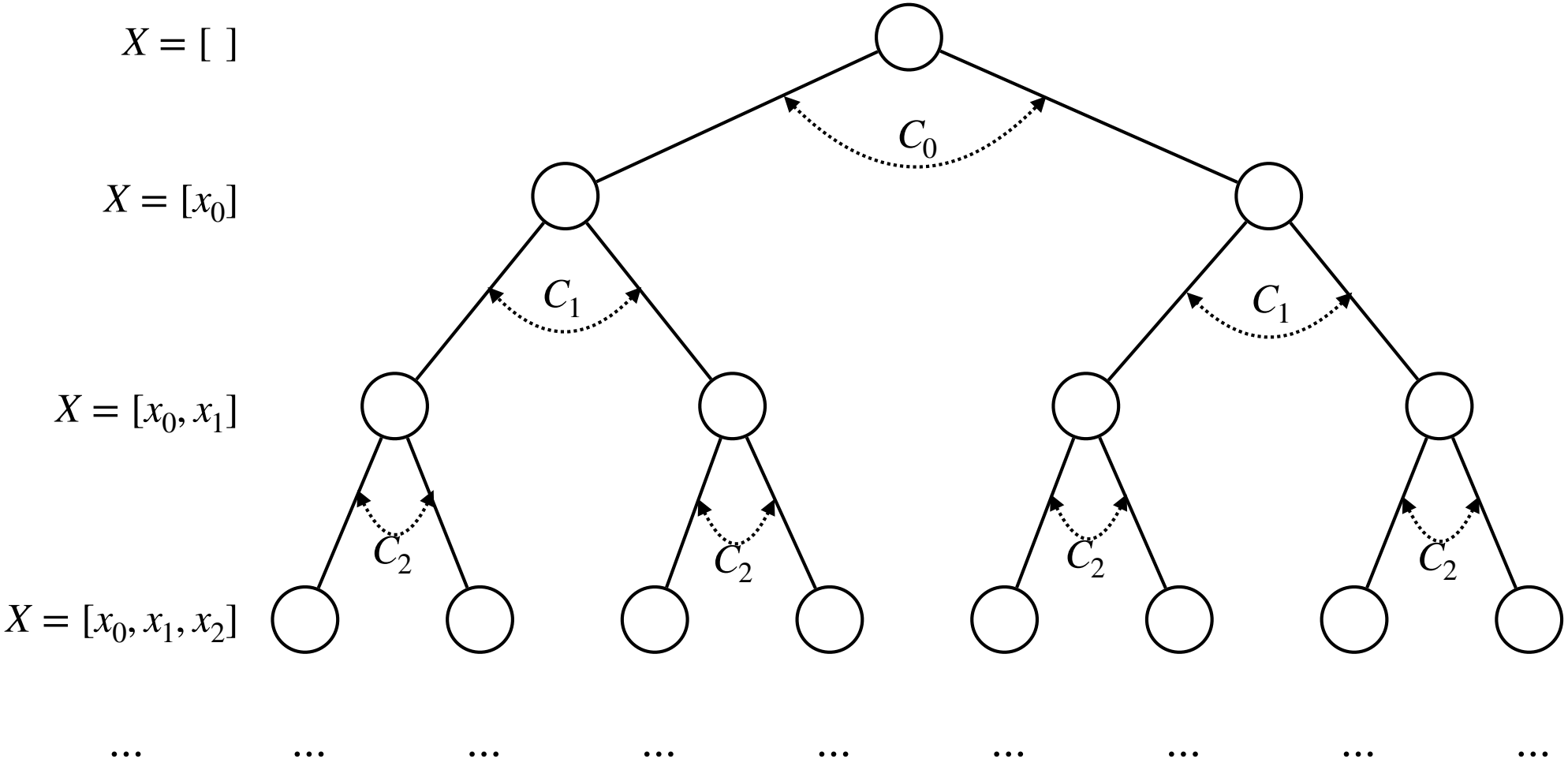
Require: **global** $X = (x_0, x_1, \cdots)$, $C_k (k = 0, 1, \cdots)$

```
1:  procedure BACKTRACK(k)
2:      if (x_0, x_1, ..., x_{k-1}) is a feasible solution then
3:          process it
4:      end if
5:      compute C_k
6:      for all x ∈ C_k do
7:          x_k ← x
8:          BACKTRACK(k + 1)
9:      end for
10: end procedure

11: BACKTRACK(0)                    ▷ Invoke the algorithm with k = 0
```

解空间树

- 从解空间树的根节点开始，按照深度优先的方式来搜索整棵树



背包问题

Knapsack

背包问题

输入：物品价值 $p[0..n-1]$，重量 $w[0..n-1]$，背包容量 M

问题：最大化 $P=\sum_{i=0}^{n-1} p_i x_i$，其中 $(x_0,\cdots,x_{n-1})\in\{0,1\}^n$，$\sum_{i=0}^{n-1} w_i x_i \le M$

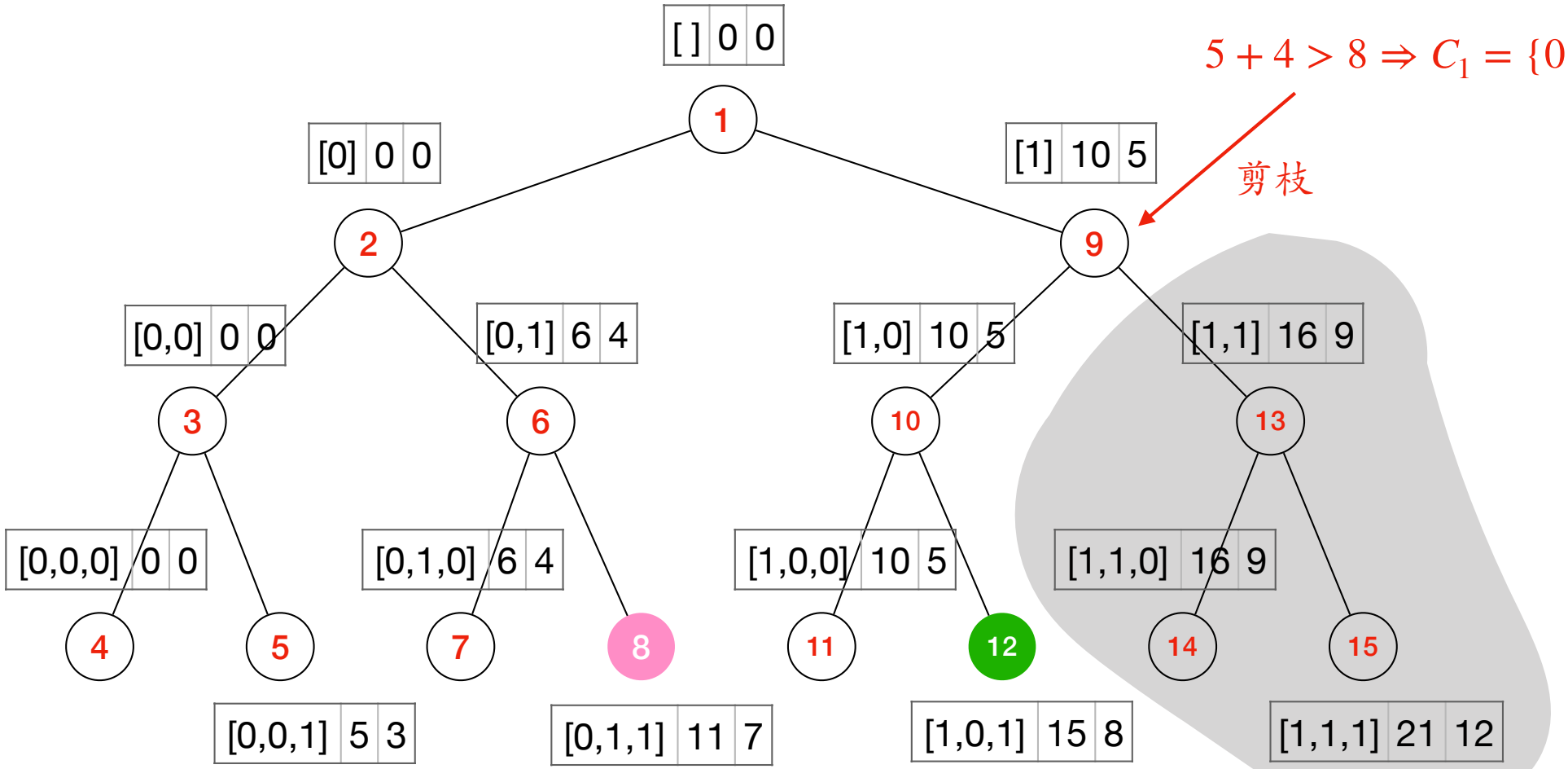
输出：P的最大值

- 当背包容量 $M=7$时，$P=11$
- 当背包容量 $M=8$时，$P=15$

- 回溯求解的一种思路
  - x_i的取值集合$C_i=\{0,1\}$
  - 是否是完整解：$i=n$

物品	0	1	2
重量	5	4	3
价值	10	6	5

背包问题的解空间树



给定部分解$(x_0, x_1, \cdots, x_{k-1})$，考虑$x_k$的可选值集合$C_k$

- 令$W=\sum_{i=0}^{k-1} w_i x_i$，有$C_k=\begin{cases}\{1,0\} & \text{if } W+w_k \le M;\\ \{0\} & \text{otherwise.}\end{cases}$

物品	0	1	2
重量	5	4	3
价值	10	6	5

回溯算法中进行剪枝

Algorithm 3 Backtracking with simple pruning for knapsack problem

Require: **global** $X = (x_0, x_1, \cdots)$, $C_k(k = 0, 1, \cdots)$, $OptP$, $OptX$

1: **procedure** KNAPSACK2(k,$CurW$)
2: **if** $k = n$ **then**
3: PROCESS2()
4: **end if**
5: **if** $k = n$ **then**
6: $C_k \leftarrow \emptyset$
7: **else if** $CurW + w_k \leq M$ **then**
8: $C_k \leftarrow \{1, 0\}$
9: **else**
10: $C_k \leftarrow \{0\}$
11: **end if**
12: **for all** $x \in C_k$ **do**
13: $x_k \leftarrow x$
14: KNAPSACK2($k + 1$, $CurW + w_k x_k$)
15: **end for**
16: **end procedure**

17: **procedure** PROCESS2()
18: **if** $\sum_{i=0}^{n-1} p_i x_i > OptP$ **then**
19: $OptP \leftarrow \sum_{i=0}^{n-1} p_i x_i$
20: $OptX \leftarrow (x_0, \cdots, x_{n-1})$
21: **end if**
22: **end procedure**

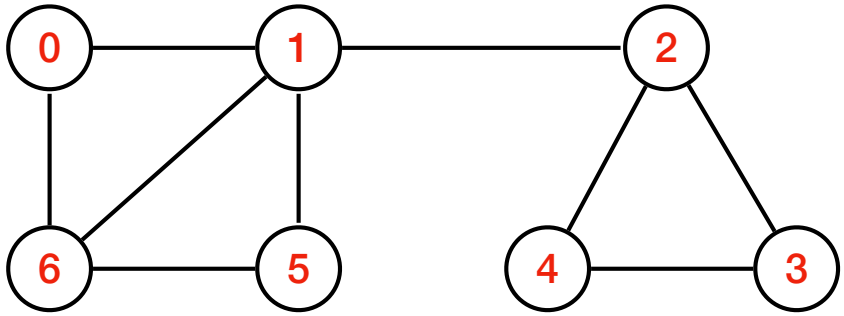
23: KNAPSACK2(0, 0) ▷ Invoke the algorithm with $k = 0$ and $CurW = 0$

团问题

Clique

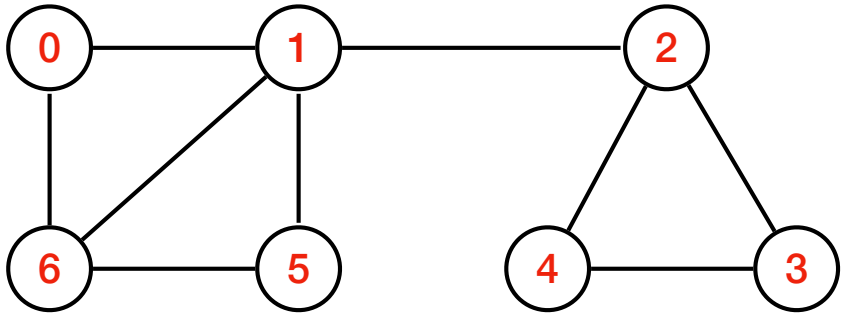
团的相关概念

- 团（clique）：无向图$G=(V,E)$中的团是一个顶点子集$S \subseteq V$，其中每一对顶点之间都由E中的一条边来连接，即对于$\forall x,y \in S$，有$(x,y) \in E$
  - $\varnothing$；$\forall x \in V, \{x\}$；$\forall (x,y) \in E, \{x,y\}$；$\cdots$
- 极大团（maximal clique）：如果一个团S不是任何其他团的真子集，那么称S是无向图G的一个极大团
  - $\{0, 1, 6\}, \{1, 2\}, \{1, 5, 6\}, \{1, 3, 4\}$
- 最大团（maximum clique）：顶点数最多的极大团
  - $\{0, 1, 6\}, \{1, 5, 6\}, \{1, 3, 4\}$



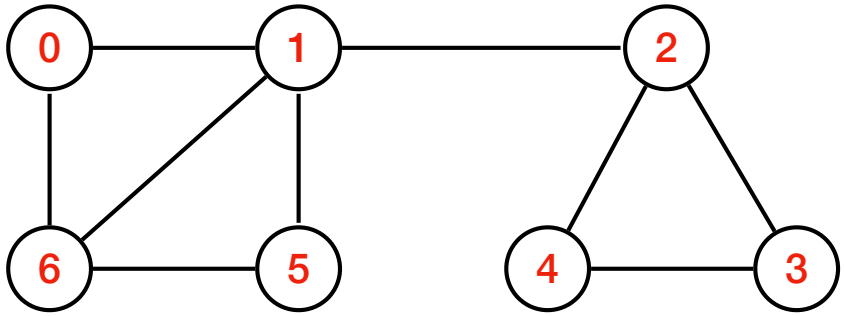
枚举所有团

- 考虑x_k的取值
  - 已知部分解$[x_0, x_1, \cdots, x_{k-1}]$，并且$S_{k-1} = \{x_0, x_1, \cdots, x_{k-1}\}$是一个团
  - 定义$C_k = \{v \in V \backslash S_{k-1} : (v, x) \in E \text{ for each } x \in S_{k-1}\}$
    - 或者根据C_{k-1}来定义C_k
      - $C_k = \{v \in C_{k-1} \backslash \{x_{k-1}\} : (v, x_{k-1}) \in E\}$
- 上述做法的问题：一个大小为k的团被枚举$k!$次



枚举所有团

- 考虑x_k的取值
  - 已知部分解$[x_0, x_1, \cdots, x_{k-1}]$，并且$S_{k-1} = \{x_0, x_1, \cdots, x_{k-1}\}$是一个团
  - 为顶点规定一种全序关系，比如$v_0 < v_1 < \cdots < v_{n-1}$
  - 定义$C_k = \{v \in C_{k-1} : (v, x_{k-1}) \in E \text{ and } v > x_{k-1}\}$
    - 令$A_v = \{u \in V : (u, v) \in E\}$，即$v$的邻接点
    - 令$B_v = \{u \in V : u > v\}$，即比$v$大的点
    - 那么，$C_k = A_{x_{k-1}} \cap B_{x_{k-1}} \cap C_{k-1}$



枚举所有团的回溯算法

Algorithm 4 A backtracking algorithm that generates all cliques

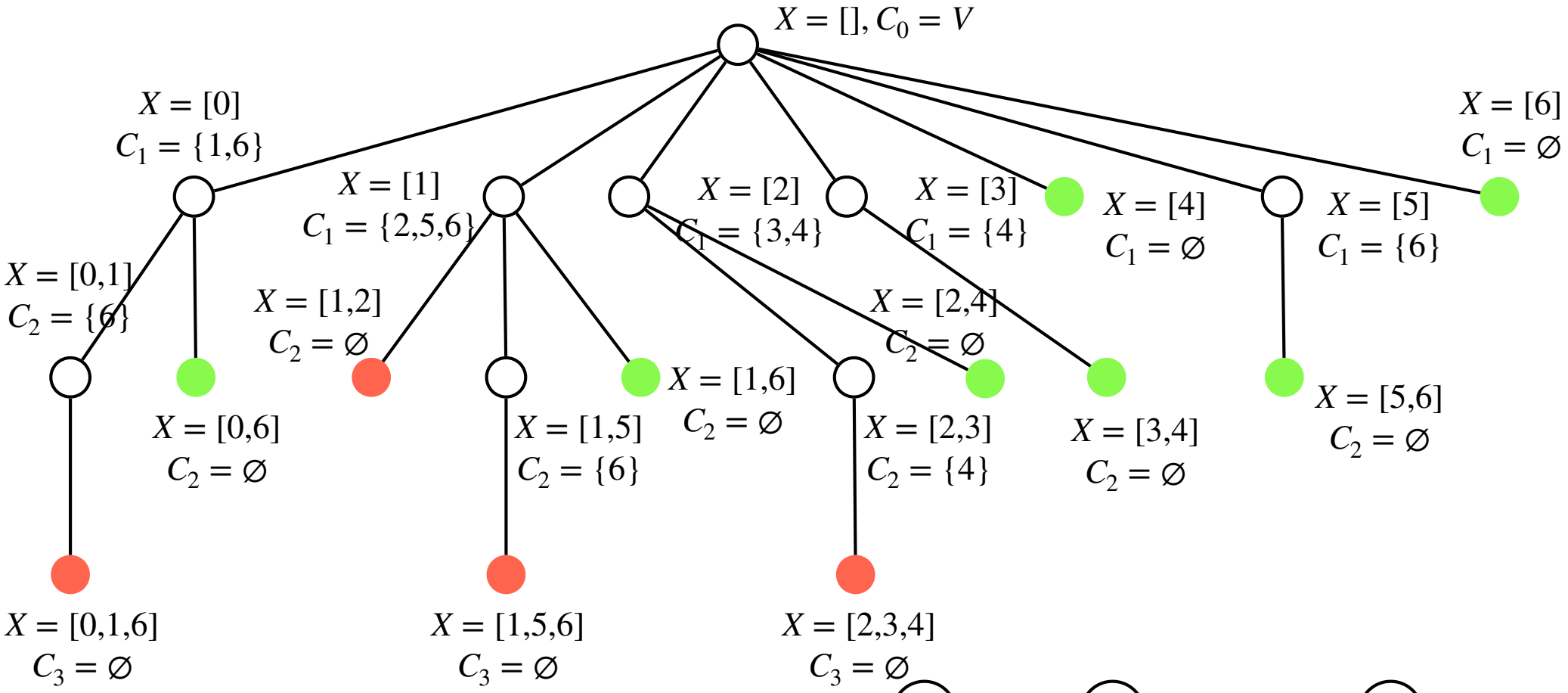
Require: **global** $X = (x_0, x_1, \cdots)$, $C_k (k = 0, 1, \cdots)$

```
1: procedure AllCliques(k)
2:     if k = 0 then
3:         output([])
4:     else
5:         output([x_0, ..., x_{k-1}])
6:     end if
7:     if k = 0 then
8:         C_k ← V
9:     else
10:        C_k ← A_{x_{k-1}} ∩ B_{x_{k-1}} ∩ C_{k-1}
11:    end if
12:    for all x ∈ C_k do
13:        x_k ← x
14:        AllCliques(k + 1)
15:    end for
16: end procedure

17: AllCliques(0)                    ▷ Invoke the algorithm with k = 0
```

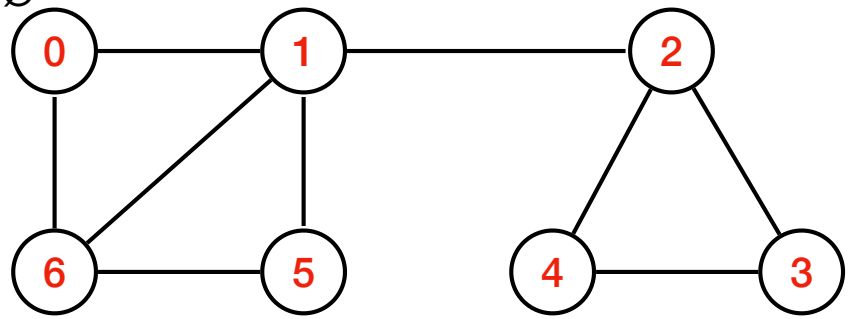
枚举所有团

- $C_k = \{v \in C_{k-1} : (v, x_{k-1}) \in E \text{ and } v > x_{k-1}\}$



团：所有节点

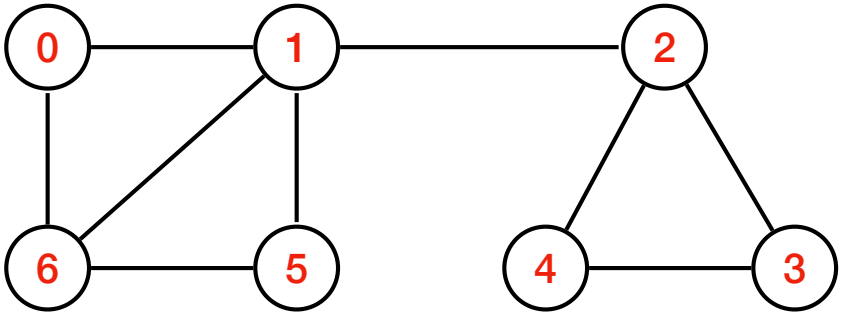
极大团：红色节点

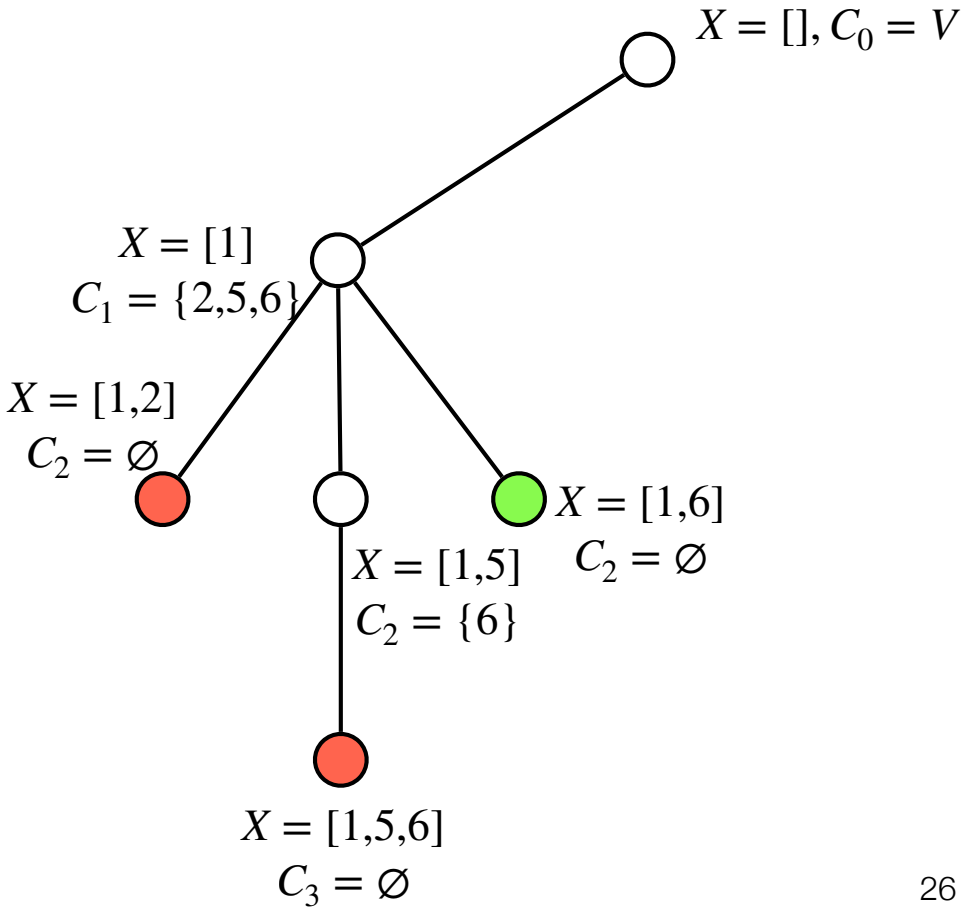


枚举所有极大团

- 非叶节点一定不是极大团
- 叶节点不一定是极大团
- 考虑部分解$[x_0, x_1, \cdots, x_{k-1}]$，$S_{k-1} = \{x_0, x_1, \cdots, x_{k-1}\}$是一个团
  - 如果存在v，其与x_i $(0 \le i \le k-1)$均有边相连，那么S_{k-1}不是极大团
  - 否则，S_{k-1}是极大团
  - 因此只需检查$N_k = \bigcap_{x \in S_{k-1}} A_x$是否为空

$$N_k = \begin{cases} V & \text{if } k = 0; \\ N_{k-1} \cap A_{x_{k-1}} & \text{otherwise.} \end{cases}$$





枚举所有极大团的回溯算法

Algorithm 5 A backtracking algorithm that generates all maximal cliques

Require: **global** $X = (x_0, x_1, \cdots)$, $C_k (k = 0, 1, \cdots)$

1: **procedure** AllMaxCliques(k)
2: **if** $k = 0$ **then**
3: $N_k \leftarrow V$
4: **else**
5: $N_k \leftarrow A_{x_{k-1}} \cap N_{k-1}$
6: **end if**
7: **if** $N_k = \emptyset$ **then** ▷ Find a maximal clique
8: output($[x_0, \cdots, x_{k-1}]$)
9: **end if**
10: **if** $k = 0$ **then**
11: $C_k \leftarrow V$
12: **else**
13: $C_k \leftarrow A_{x_{k-1}} \cap B_{x_{k-1}} \cap C_{k-1}$
14: **end if**
15: **for all** $x \in C_k$ **do**
16: $x_k \leftarrow x$
17: AllMaxCliques($k + 1$)
18: **end for**
19: **end procedure**

20: AllMaxCliques(0) ▷ Invoke the algorithm with $k = 0$

集合覆盖问题

Exact Cover

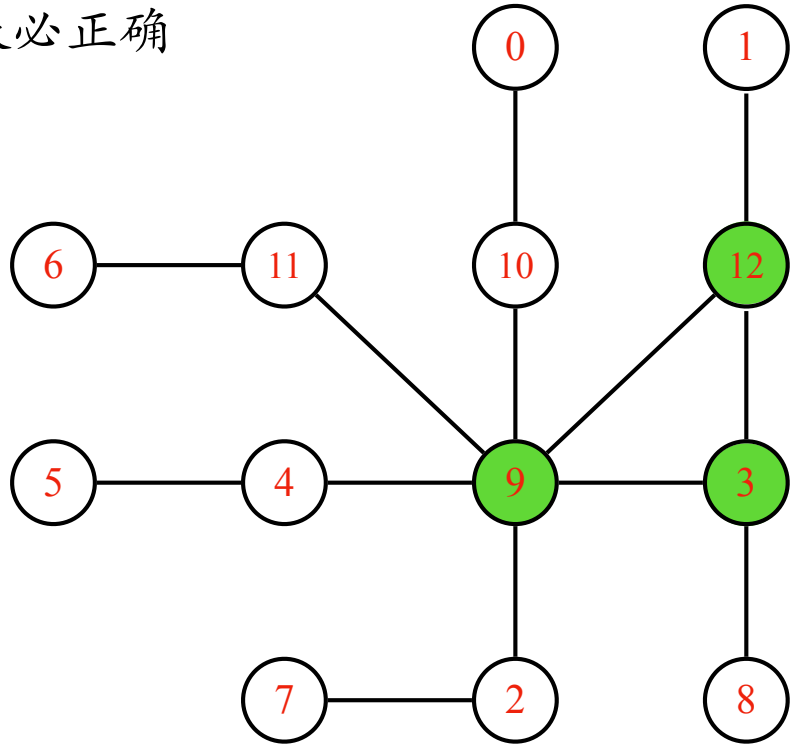
集合覆盖问题

- 给定集合$R = \{0, \cdots, n-1\}$和一个R的子集族S，判断S是否包含R的一个覆盖，即是否存在一个集合$S' = \{S_{x_0}, S_{x_1}, \cdots, S_{x_{k-1}}\} \subseteq S$，使得$R$中的每个元素仅出现在$S'$的一个成员中

j	S_j
0	{0, 1, 3}
1	{0, 1, 5}
2	{0, 2, 4}
3	{0, 2, 5}
4	{0, 3, 6}
5	{1, 2, 4}
6	{1, 2, 6}
7	{1, 3, 5}
8	{1, 4, 6}
9	{1}
10	{2, 5, 6}
11	{3, 4, 5}
12	{3, 4, 6}

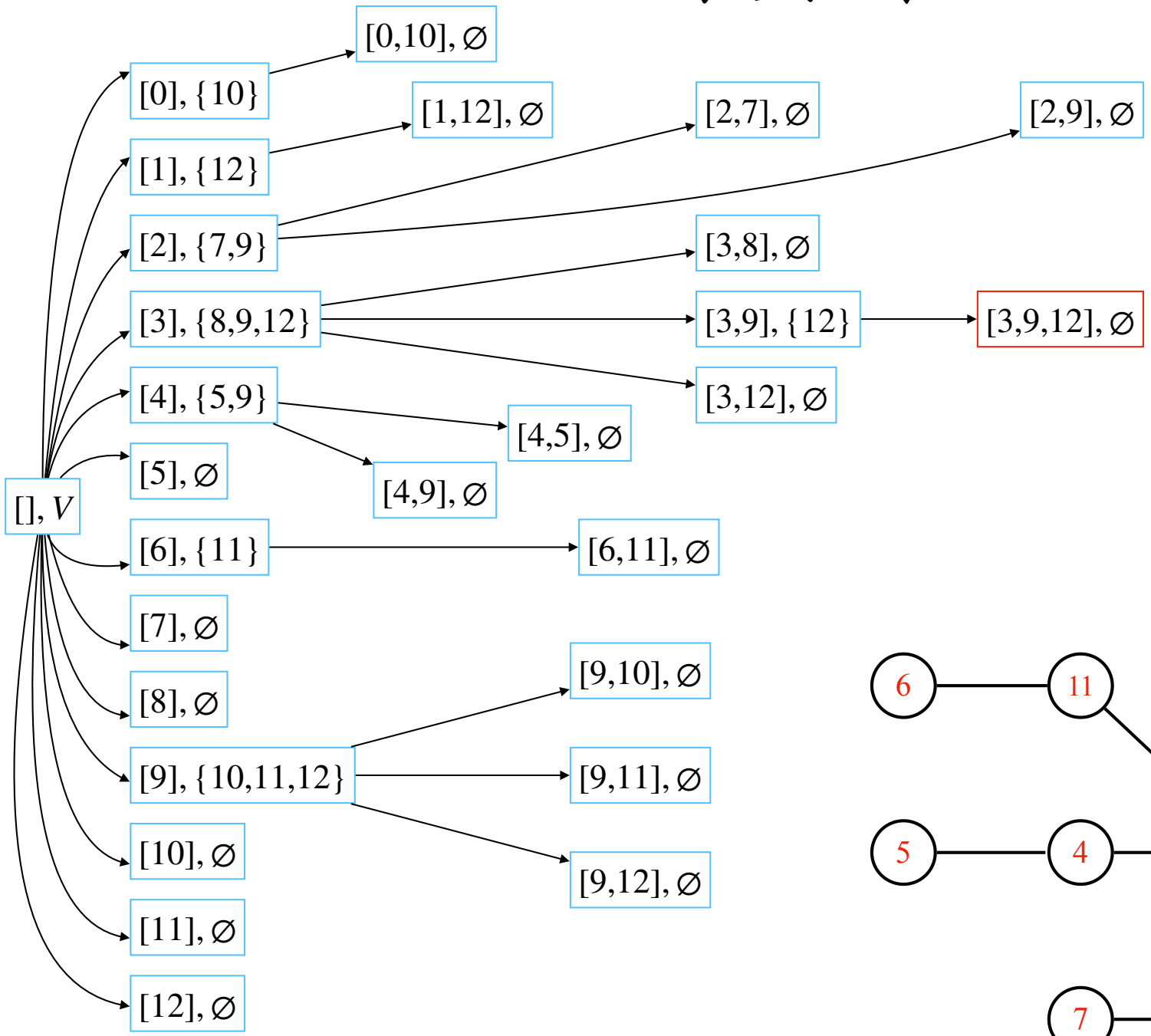
集合覆盖问题与团问题

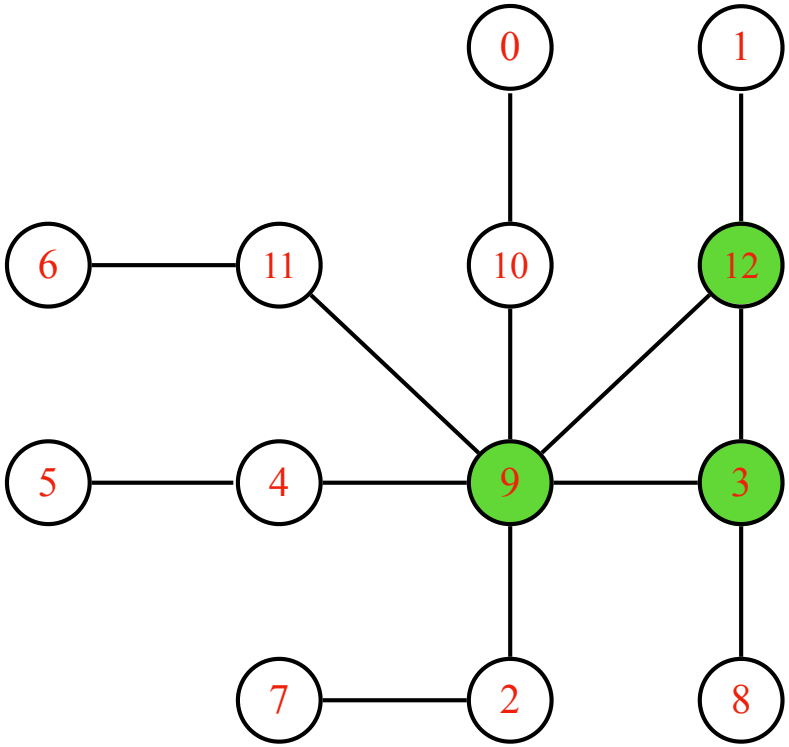
- 给定$R=\{0,\cdots,n-1\}$和R的子集族$S=\{S_0,\cdots,S_{m-1}\}$，如下构造无向图G
  - 顶点集合$V=\{0,1,\cdots,m-1\}$
  - $(i,j)\in E$当且仅当$S_i\cap S_j=\varnothing$
- 命题：如果S'是R的一个覆盖，那么
  - S'对应的顶点集合必然是G中的一个极大团
- 注意，上述命题的逆命题未必正确
- 求解集合覆盖问题
  - 枚举所有的极大团S'
  - 检查S'是否满足条件



j	S_j
0	{0, 1, 3}
1	{0, 1, 5}
2	{0, 2, 4}
3	{0, 2, 5}
4	{0, 3, 6}
5	{1, 2, 4}
6	{1, 2, 6}
7	{1, 3, 5}
8	{1, 4, 6}
9	{1}
10	{2, 5, 6}
11	{3, 4, 5}
12	{3, 4, 6}

运行实例


[], V
[0], {10}
[1], {12}
[2], {7,9}
[3], {8,9,12}
[4], {5,9}
[5], ∅
[6], {11}
[7], ∅
[8], ∅
[9], {10,11,12}
[10], ∅
[11], ∅
[12], ∅
[0,10], ∅
[1,12], ∅
[2,7], ∅
[2,9], ∅
[3,8], ∅
[3,9], {12}
[3,12], ∅
[3,9,12], ∅
[4,5], ∅
[4,9], ∅
[6,11], ∅
[9,10], ∅
[9,11], ∅
[9,12], ∅


0
1
6
11
10
12
5
4
9
3
7
2
8

枚举所有团

- 回顾：$C_k = \{v \in C_{k-1} : (v, x_{k-1}) \in E \text{ and } v > x_{k-1}\}$
- 计算v的邻接点：$A_v = \{u \in V : S_v \cap S_u = \varnothing\}$
- 计算比v大的点：$B_v = \{u \in V : S_u > S_v\}$
  - 理论上可以使用任何全序关系，比如集合的ID信息
- 没有完全利用顶点的语义信息：一个顶点表示一个集合
  - 顶点是否相邻：对应的集合是否相交
  - 极大团未必对应集合覆盖：需要考虑顶点（对应的集合）对覆盖的贡献
    - 哪些顶点可以覆盖元素0
    - 哪些顶点可以覆盖元素1
    - …

集合的字典序

- $\overset{lex}{>}$：集合的字典序 (lexicographic order)
- 给定$T \subseteq S = \{0,1,\cdots,n-1\}$
  - T的特征向量：$\chi(T) = [x_{n-1}, \cdots, x_0]$，其中$x_i = \begin{cases} 1 & \text{if } n-1-i \in T; \\ 0 & \text{if } n-1-i \notin T. \end{cases}$
    - 比如$n = 7$时，$T = \{0,3,6\}$的特征向量是[1,0,0,1,0,0,1]
  - T的排名：$\text{rank}(T) = \sum_{i=0}^{n-1} x_i 2^i$
    - 比如$n = 7$时，$T = \{0,3,6\}$的排名是$2^6 + 2^3 + 2^0 = 73$
  - $S_u \overset{lex}{>} S_v \Leftrightarrow \text{rank}(S_u) > \text{rank}(S_v)$

集合的划分

- $R = \{0, \cdots, n-1\}$的子集族$S = \{S_0, \cdots, S_{m-1}\}$的一个划分：$H_0, \cdots, H_{n-1}$
  - $H_i = \{v \in V : S_v \cap \{0, \cdots, i\} = \{i\}\}, 0 \le i \le n-1$

	j	S_j	$\chi(S_j)$	$\text{rank}(S_j)$
H_0	0	{0, 1, 3}	[1, 1, 0, 1, 0, 0, 0]	104
	1	{0, 1, 5}	[1, 1, 0, 0, 0, 1, 0]	98
	2	{0, 2, 4}	[1, 0, 1, 0, 1, 0, 0]	84
	3	{0, 2, 5}	[1, 0, 1, 0, 0, 1, 0]	82
	4	{0, 3, 6}	[1, 0, 0, 1, 0, 0, 1]	73
H_1	5	{1, 2, 4}	[0, 1, 1, 0, 1, 0, 0]	52
	6	{1, 2, 6}	[0, 1, 1, 0, 0, 0, 1]	49
	7	{1, 3, 5}	[0, 1, 0, 1, 0, 1, 0]	44
	8	{1, 4, 6}	[0, 1, 0, 0, 1, 0, 1]	37
	9	{1}	[0, 1, 0, 0, 0, 0, 0]	32
H_2	10	{2, 5, 6}	[0, 0, 1, 0, 0, 1, 1]	19
H_3	11	{3, 4, 5}	[0, 0, 0, 1, 1, 1, 0]	14
	12	{3, 4, 6}	[0, 0, 0, 1, 1, 0, 1]	13

如果$S_x \in H_i, S_y \in H_j, i < j$，

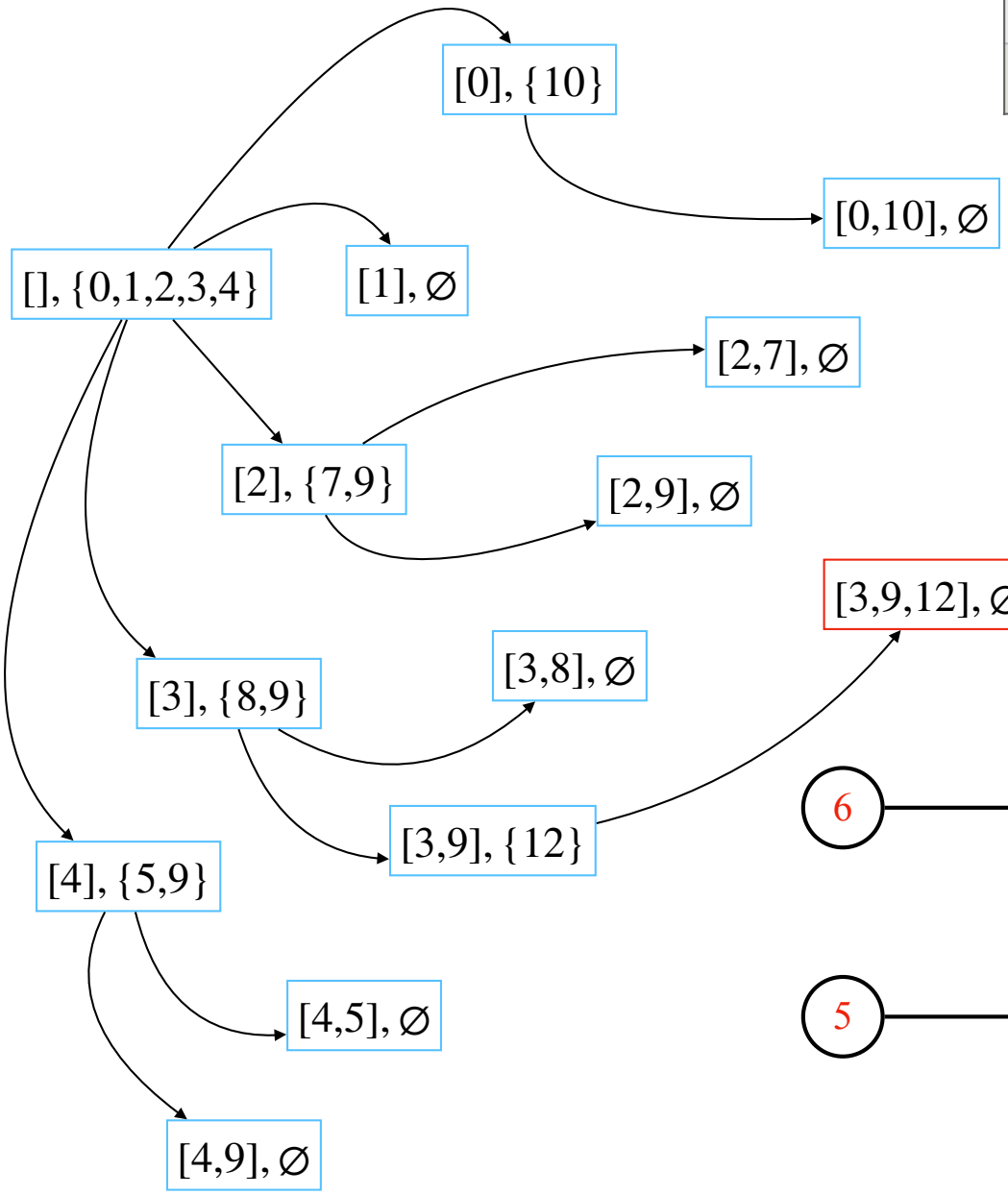
那么$\text{rank}(S_x) > \text{rank}(S_y)$

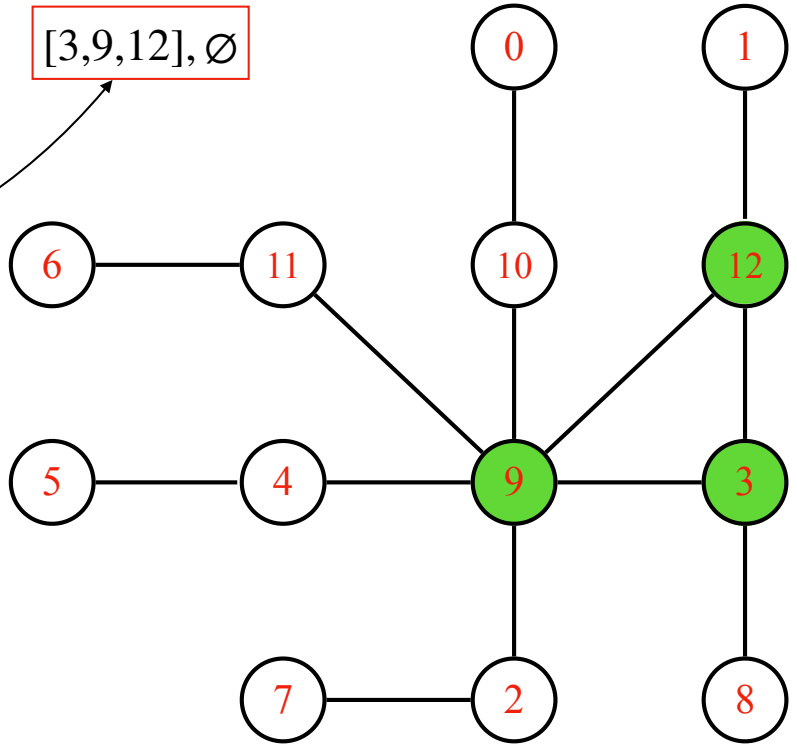
枚举所有团

- 如果$[x_0, \cdots, x_{k-1}]$不是R的一个覆盖
  - 那么$U_k = R \backslash (\bigcup_{i=0}^{k-1} S_{x_i})$必然非空，令$r$是$U_k$中的最小元素
- 如果$[x_0, \cdots, x_{k-1}]$可以扩展为一个可行解$[x_0, \cdots, x_{k-1}, x_k, \cdots]$
  - 那么必然要从H_r中选择某个元素
- 所以，第k步的决策C_k可以如下计算
  - 计算v的邻接点：$A_v = \{u \in V : S_v \cap S_u = \varnothing\}$
  - 计算比v大的点：$B_v = \{u \in V : S_u \overset{lex}{>} S_v\}$
  - 计算$C'_k = A_{x_{k-1}} \cap B_{x_{k-1}} \cap C'_{k-1}$
  - 计算$C_k = C'_k \cap H_r$

运行实例

i	0	1	2	3	4	5	6
H_i	{0,1,2,3,4}	{5,6,7,8,9}	{10}	{11,12}	∅	∅	∅





j	S_j
0	{0, 1, 3}
1	{0, 1, 5}
2	{0, 2, 4}
3	{0, 2, 5}
4	{0, 3, 6}
5	{1, 2, 4}
6	{1, 2, 6}
7	{1, 3, 5}
8	{1, 4, 6}
9	{1}
10	{2, 5, 6}
11	{3, 4, 5}
12	{3, 4, 6}

求解覆盖问题的回溯算法

Algorithm 6 Main procedure for exact cover problem

```
1: procedure MAIN(n, S)
2:     m ← |S|
3:     sort S in decreasing lexicographic order
4:     for i = 0 to m − 1 do
5:         A_i ← {j : S_i ∩ S_j = ∅}
6:     end for
7:     for i = 0 to m − 1 do
8:         B_i ← {i + 1, i + 2, ··· , m − 1}
9:     end for
10:    for i = 0 to n − 1 do
11:        H_i ← {j : S_j ∩ {0, ··· , i} = {i}}
12:    end for
13:    H_n ← ∅
14:    EXACTCOVER(0,0)
15: end procedure
```

Algorithm 7 ExactCover procedure for exact cover problem

Require: **global** X, A, B, C, C', H, S, U

procedure ExactCover(k, r')
  if $k = 0$ **then**
    $U_0 \leftarrow \{0, \cdots, n-1\}$
    $r \leftarrow 0$
  else
    $U_k \leftarrow U_{k-1} \setminus S_{x_{k-1}}$
    $r \leftarrow r'$
    while $r \notin U_k$ **and** $r < n$ **do**
      $r \leftarrow r + 1$
    end while
  end if
  if $r = n$ **then**
    output($[x_0, \cdots, x_{k-1}]$)
  end if
  if $k = 0$ **then**
    $C'_0 \leftarrow \{0, 1, \cdots, m-1\}$
  else
    $C'_k \leftarrow A_{x_{k-1}} \cap B_{x_{k-1}} \cap C'_{k-1}$
  end if
  $C_k \leftarrow C'_k \cap H_r$
  for all $x \in C_k$ **do**
    $x_k \leftarrow x$
    ExactCover($k + 1, r$)
  end for
end procedure

限界函数

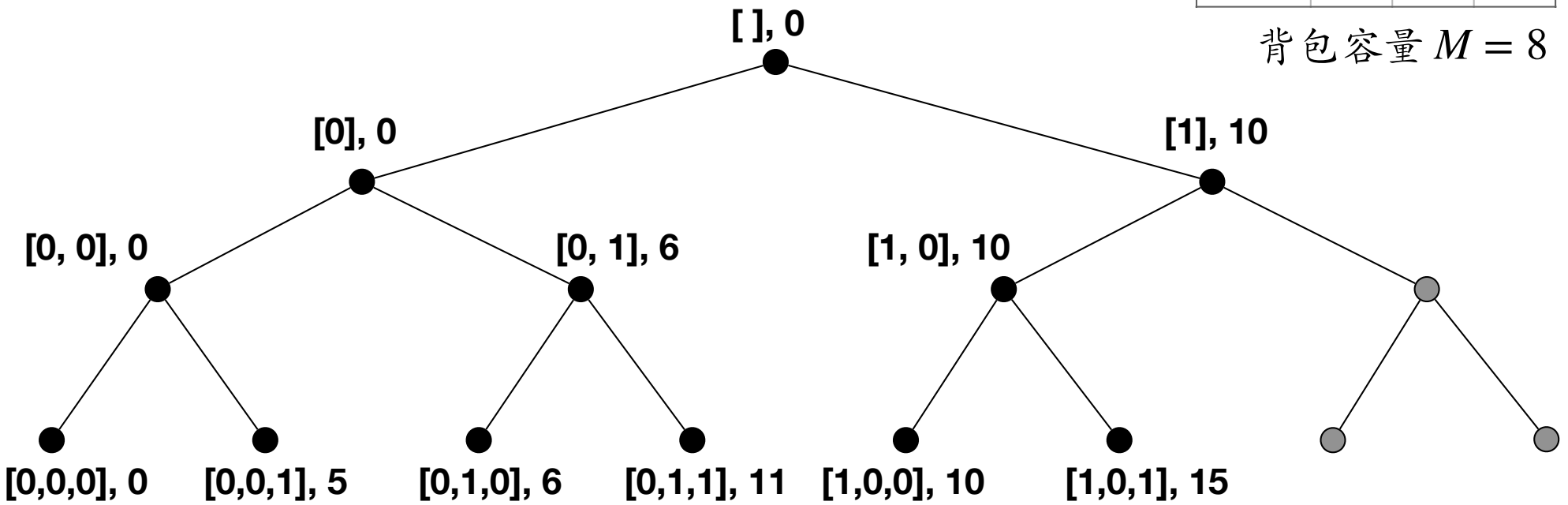
Bounding Function

限界函数

- $X = [x_0, x_1, \cdots, x_{k-1}]$是解空间树上的某个节点，profit($X$)是$X$的收益
- $P(X)$：从X扩展得到的可行解的最大收益
  - 比如，$P([0]) = 11$，$P([1]) = 15$，$P([])$即为背包问题的最优解
- $P(X)$的精确值可以通过遍历以X为根的子树得到
- 能否快速地得到$P(X)$的近似值，比如一个上界B(X)
- 剪枝：$P(X) \leq \mathrm{B}(X) \leq OptP$，其中$OptP$是当前最优解

物品	0	1	2
重量	5	4	3
价值	10	6	5

背包容量 $M = 8$



限界函数

- $X = [x_0, x_1, \cdots, x_{k-1}]$是解空间树上的某个节点，profit($X$)是$X$的收益
- $P(X)$：从X扩展得到的可行解的最大收益
- 限界函数B(X)是对$P(X)$的近似
  - 容易计算
  - 尽可能接近$P(X)$
- 包含限界函数的回溯
- 对于每一个$x \in C_k$
  - 需要检查剪枝条件

Algorithm 8 A backtracking algorithm incorporating a bounding function

Require: **global** X, $OptP$, $OptX$, $C_k (k = 0, 1, \cdots)$

```
1:  procedure BACKTRACK(k)
2:      if [x_0, x_1, ..., x_{k-1}] is a feasible solution then
3:          P ← profit([x_0, x_1, ..., x_{k-1}])
4:          if P > OptP then
5:              OptP ← P
6:              OptX ← [x_0, x_1, ..., x_{k-1}]
7:          end if
8:      end if
9:      compute C_k
10:     B ← B([x_0, x_1, ..., x_{k-1}])
11:     for all x ∈ C_k do
12:         if B ≤ OptP then                    ▷ OptP may increase
13:             return
14:         end if
15:         x_k ← x
16:         BACKTRACK(k + 1)
17:     end for
18: end procedure

19: BACKTRACK(0)                                ▷ Invoke the algorithm with k = 0
```

背包问题

Knapsack

背包问题

- 给定物品价值$p[0..n-1]$，重量$w[0..n-1]$，背包容量M
  - 在满足$\sum_{i=0}^{n-1} w_i x_i \le M$的情况下，最大化$P=\sum_{i=0}^{n-1} p_i x_i$
    - 01背包问题：$(x_0,\cdots,x_{n-1})\in\{0,1\}^n$
    - 分数背包问题：x_i是一个有理数且$0\le x_i\le 1$
- 01背包问题是NP难问题，不存在多项式时间的算法
- 分数背包问题存在$O(n\log n)$的算法
  - 贪心
  - 将物品按照单位重量的价值从大到小排序，然后依次放入物品
    - 如果剩余空间不足以放入整个物品，按照剩余空间放入部分物品
- 利用分数背包算法设计限界函数

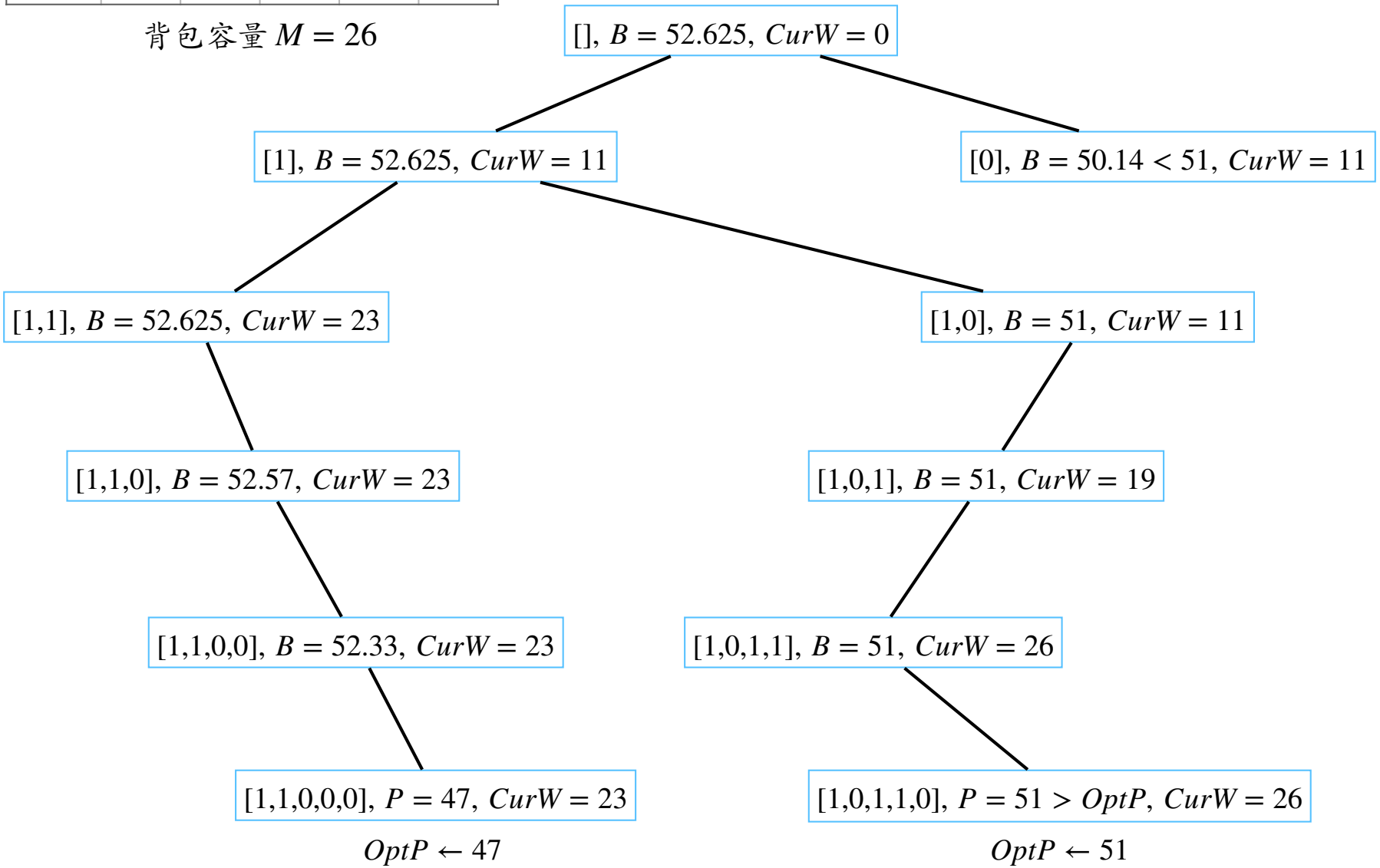
背包问题的限界函数

- 利用分数背包算法设计限界函数
  - RKNAP$(p_0, \cdots, p_{n-1}, w_0, \cdots, w_{n-1}, M)$返回分数背包问题的最优解
- 对于$X = [x_0, x_1, \cdots, x_{k-1}]$，定义限界函数B$(X)$如下
  - $\mathrm{B}(X) = \sum_{i=0}^{k-1} p_i x_i + \mathrm{RKNAP}(p_k, \cdots, p_{n-1}, w_k, \cdots, w_{n-1}, M - \sum_{i=0}^{k-1} w_i x_i)$

运行实例

物品	0	1	2	3	4
重量	11	12	8	7	9
价值	23	24	15	13	16

背包容量 $M = 26$



$OptP \leftarrow 47$

$OptP \leftarrow 51$

剪枝效果 - 解空间树的大小

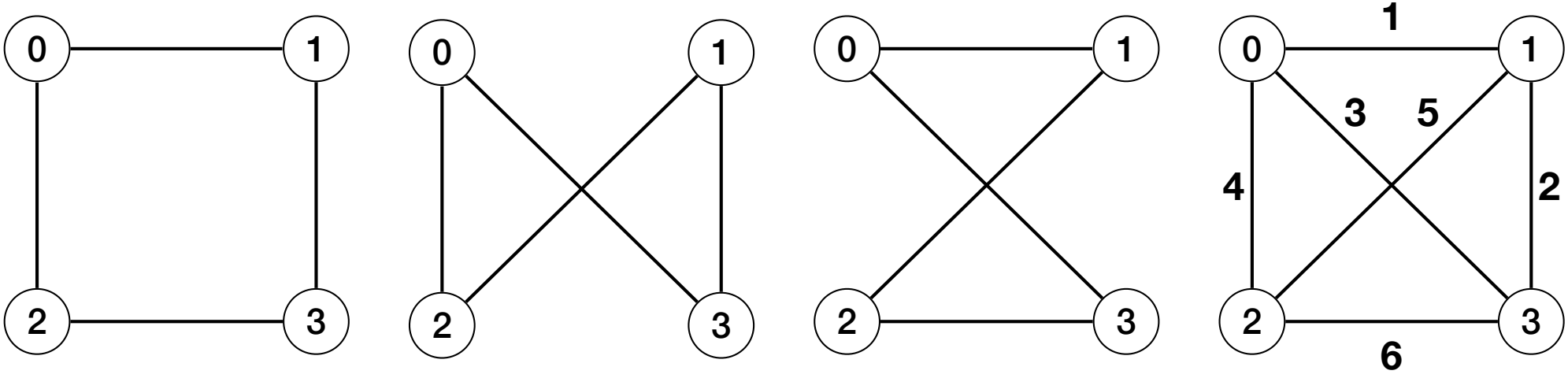
n	无剪枝	背包容量剪枝	分数背包剪枝
8	511	332	52
8	511	312	78
8	511	333	72
12	8191	4598	109
12	8191	4737	93
12	8191	5079	164
16	131071	73639	192
16	131071	72302	58
16	131071	76512	168
20	2097151	1173522	299
20	2097151	1164523	104
20	2097151	1257745	416
24	33554431	19491410	693
24	33554431	18953093	180
24	33554431	17853054	278

旅行商问题

The Traveling Salesman Problem

旅行商问题

- 给定n个顶点的完全图$G = (V, E)$，代价函数$\text{cost} : E \to \mathbb{Z}^+$，找到最小代价的哈密顿圈$X$
  - 恰好经过每个顶点一次的圈
  - $\text{cost}(X) = \sum_{e \in E(X)} \text{cost}(e)$
- 一个哈密顿圈可以看成是顶点集合V的排列
- 包含n个顶点的完全图G有$\frac{1}{2}(n-1)!$个不同的哈密顿圈



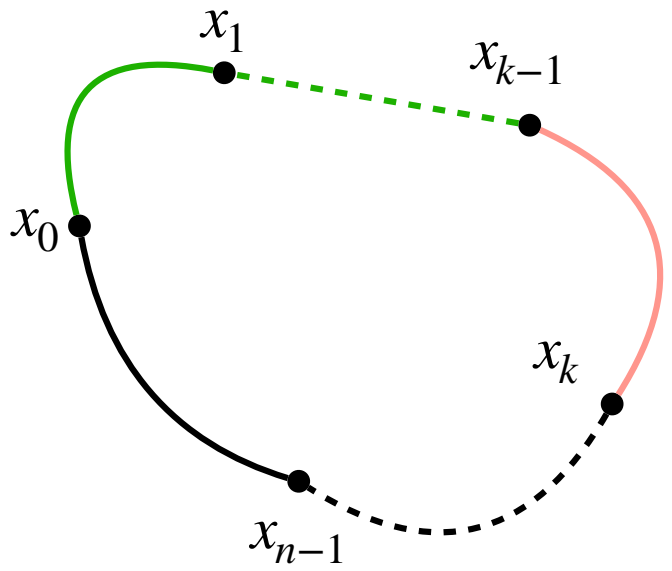
$cost(X_1) = 13$　　$cost(X_2) = 14$　　$cost(X_3) = 15$

基于最小值的限界函数

- 旅行商问题是最小化问题，所以限界函数应该提供一个下界
  - 当$P(X) \geq \mathrm{B}(X) \geq OptP$时，剪枝
- 考虑解$X = [x_0, x_1, \cdots, x_{k-1}]$，从其扩展的解是$[x_0, x_1, \cdots, x_{n-1}]$
- 令$Y = V \backslash \{x_0, x_1, \cdots, x_{k-1}\}$，那么$Y = \{x_k, \cdots, x_{n-1}\}$
- 令$X' = [x_0, \cdots, x_{n-1}]$是从$[x_0, \cdots, x_{k-1}]$扩展而来的最小代价的哈密顿圈，那么
  - $\mathrm{cost}(X') \geq \sum_{i=0}^{k-1} \mathrm{cost}(x_i, x_{i+1}) + \mathrm{b}(x_{k-1}, Y) + \sum_{y \in Y} \mathrm{b}(y, Y \cup \{x_0\})$
  - 其中，$\mathrm{b}(x, W) = \min\{\mathrm{cost}(x, y) : y \in W\}$

基于最小值的限界函数

- 令$Y = V \backslash \{x_0, x_1, \cdots, x_{k-1}\}$，那么$Y = \{x_k, \cdots, x_{n-1}\}$
- 令$X' = [x_0, \cdots, x_{n-1}]$是从$X = [x_0, \cdots, x_{k-1}]$扩展而来的最小代价的哈密顿圈
  - $\text{cost}(X') \geq \sum_{i=0}^{k-1} \text{cost}(x_i, x_{i+1}) + \text{b}(x_{k-1}, Y) + \sum_{y \in Y} \text{b}(y, Y \cup \{x_0\})$
  - 其中，$\text{b}(x, W) = \min\{\text{cost}(x, y) : y \in W\}$
- 证明
  - 令$x_n = x_0$，那么$\text{cost}(X') = \sum_{i=0}^{n-1} \text{cost}(x_i, x_{i+1})$
  - 已经确定的边：绿色部分
  - 边(x_{k-1}, x_k)：$x_k \in Y \Rightarrow \text{cost}(x_{k-1}, x_k) \geq \text{b}(x_{k-1}, Y)$
  - 剩余的边：对于$k \leq i \leq n-1$，$\text{cost}(x_i, x_{i+1}) \geq \text{b}(x_i, Y \cup \{x_0\})$

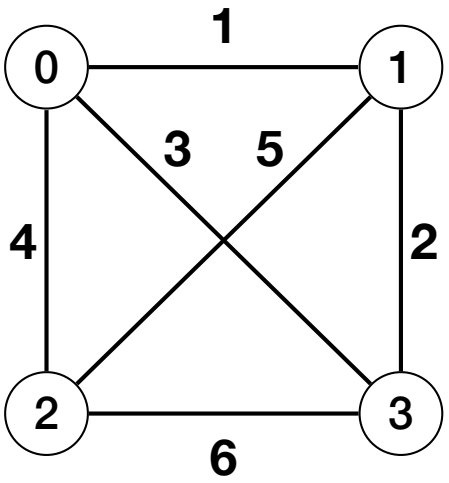


基于最小值的限界函数

- 令$Y = V \backslash \{x_0, x_1, \cdots, x_{k-1}\}$，那么$Y = \{x_k, \cdots, x_{n-1}\}$
- 令$X' = [x_0, \cdots, x_{n-1}]$是从$X = [x_0, \cdots, x_{k-1}]$扩展而来的最小代价的哈密顿圈
  - $\text{cost}(X') \geq \sum_{i=0}^{k-1} \text{cost}(x_i, x_{i+1}) + \text{b}(x_{k-1}, Y) + \sum_{y \in Y} \text{b}(y, Y \cup \{x_0\})$
  - 其中，$\text{b}(x, W) = \min\{\text{cost}(x, y) : y \in W\}$
- 限界函数$\text{B}(X)$
  - 当$k \leq n-1$时，$\text{B}(X) = \sum_{i=0}^{k-1} \text{cost}(x_i, x_{i+1}) + \text{b}(x_{k-1}, Y) + \sum_{y \in Y} \text{b}(y, Y \cup \{x_0\})$
  - 当$k = n$时，$\text{B}(X) = \sum_{i=0}^{k-1} \text{cost}(x_i, x_{i+1}) + \text{cost}(x_{n-1}, x_0)$
- $\text{B}(X)$的计算复杂度：$O(n^2)$

基于简化矩阵的限界函数

- 简化矩阵：元素都是非负整数；每一行至少包含1个0；每一列至少包含1个0
- 将一个整数矩阵转换成简化矩阵
  - 每一行减去该行的最小值
  - 每一列减去该列的最小值
- 矩阵M的简化值Reduce(M)：简化过程中减去的值的和
- 令$M[i,j] = \text{cost}(i,j)$
  - 右图包含的三个哈密顿圈的代价分别是13，14和15



∞	1	4	3		∞	0	3	2	1		∞	0	0	1
1	∞	5	2		0	∞	4	1	1		0	∞	1	0
4	5	∞	6		0	1	∞	2	4		0	1	∞	1
3	2	6	∞		1	0	4	∞	2		1	0	2	∞
				Reduce(M) = 12							0	0	3	1

基于简化矩阵的限界函数

- 命题：完全图G的任意哈密顿圈的代价至少是Reduce(M)
- 证明
  - 令$X=[x_0,\cdots,x_{n-1}]$是图G的任意一个哈密顿圈且$x_n=x_0$
  - $\text{cost}(X)=M[x_0,x_1]+M[x_1,x_2]+\cdots+M[x_{n-1},x_n]$ ← M中每一行以及每一列只有一个值参与计算
  - 令$r_i=\min\{M[i,j]:0\le j\le n-1\}$，$c_j=\min\{M[i,j]-r_i:0\le i\le n-1\}$
  - $r_{x_i}+c_{x_{i+1}}\le r_{x_i}+c_{x_{i+1}}+M'[x_i,x_{i+1}]=M[x_i,x_{i+1}]$
  - 所以，$\text{cost}(X)=\sum_{i=0}^{n-1}M[x_i,x_{i+1}]\ge\sum_{i=0}^{n-1}r_i+\sum_{j=0}^{n-1}c_j=\text{Reduce}(M)$

∞	1	4	3
1	∞	5	2
4	5	∞	6
3	2	6	∞

Reduce(M) = 12

∞	0	3	2	1
0	∞	4	1	1
0	1	∞	2	4
1	0	4	∞	2

0	0	3	1
∞	0	0	1
0	∞	1	0
0	1	∞	1
1	0	2	∞

基于简化矩阵的限界函数

- 令$X=[x_0,x_1,\cdots,x_{k-1}]$，对矩阵$M$做如下操作得到矩阵$M'$
  - 当$k\le n-1$时，$M[x_{k-1},0]\leftarrow\infty$
  - 删除M的$x_0,x_1,\cdots,x_{k-2}$对应的行
  - 删除M的$x_1,x_2,\cdots,x_{k-1}$对应的列
- 令$X'=[x_0,\cdots,x_{n-1}]$是从$[x_0,\cdots,x_{k-1}]$扩展而来的最小代价的哈密顿圈，那么
  - $\text{cost}(X')\ge\sum_{i=0}^{k-2}M(x_i,x_{i+1})+\text{Reduce}(M')$

$\text{Reduce}(M)=12$

M

∞	1	4	3
1	∞	5	2
4	5	∞	6
3	2	6	∞

$X=[x_0,x_1]$

M'

∞	1	4	3
∞	∞	5	2
4	5	∞	6
3	2	6	∞

$\text{Reduce}(M')=11$

0		3	0	
∞	0	0	1	
∞	∞	0	0	2
0	1	∞	2	4
0	0	0	∞	3

基于简化矩阵的限界函数

- 令$X'=[x_0,\cdots,x_{n-1}]$是从$[x_0,\cdots,x_{k-1}]$扩展而来的最小代价的哈密顿圈，那么
  - $\text{cost}(X')\geq\sum_{i=0}^{k-2}M(x_i,x_{i+1})+\text{Reduce}(M')$　　限界函数的计算复杂度：$O(n^2)$
- 证明　　还需确定剩余$n-k$个顶点的顺序
  - M'的大小：$(n-k+1)\times(n-k+1)$；构成哈密顿圈还需$n-k+1$条边
  - 因为$M[x_{k-1},0]\leftarrow\infty$，所以不会选择边$(x_{k-1},x_0)$。另外，新加入的边的起点来自$\{x_{k-1},x_k,\cdots,x_{n-1}\}$，终点来自$\{x_k,\cdots,x_{n-1},x_0\}$。所以计算$\text{cost}(X')$时，$M'$中每一行以及每一列只有一个值参与
  - 与前一命题证明过程类似，加入的$n-k+1$条边的代价最少是$\text{Reduce}(M')$

M

∞	1	4	3
1	∞	5	2
4	5	∞	6
3	2	6	∞

$X=[x_0,x_1]$ →

M'

∞	1	4	3
∞	∞	5	2
4	5	∞	6
3	2	6	∞

→

0		3	0	
∞	0	0	1	
∞	∞	0	0	2
0	1	∞	2	4
0	0	0	∞	3

剪枝效果 - 解空间树的大小

n	无剪枝	最小值剪枝	简化矩阵剪枝
5	65	45	18
10	986410	5,199	1,287
15	236,975,164,805	1,538,773	53,486
20	$\approx 3.3 \times 10^{17}$	64,259,127	1,326,640