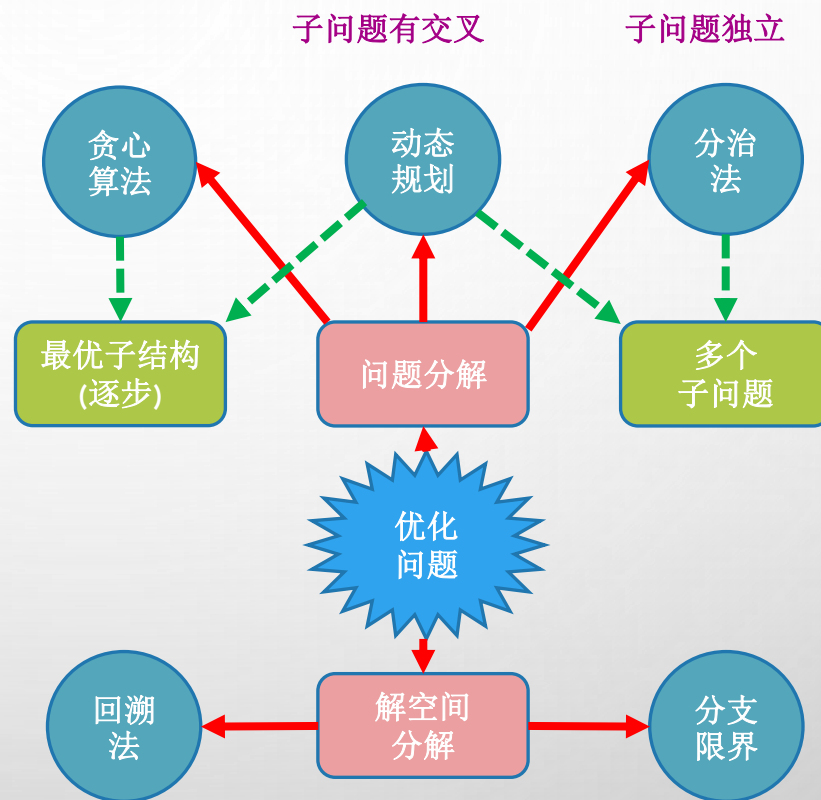


算法设计与分析

本课程内容

- 算法的基本概念
- 算法复杂度分析
- 分治法
- 贪心算法
- 动态规划
- 回溯法
- 分支限界法



算法分析概述

算法分析概论-1

1. 设 n 是描述问题规模的非负整数, 下面程序片段的时间复杂度是(A).

```
x=2;  
while (x<n/2)  
    x=2*x;
```

A. $O(\log n)$

B. $O(n)$

C. $O(n \log n)$

D. $O(n^2)$

算法分析概论-2

2. 求整数 $n(n \geq 0)$ 阶乘的算法如下，其时间复杂度是(**B**).

```
int fact(int n)
{if (n<=1) return 1;
return n*fact(n-1);}
```

A. $O(\log n)$

B. $O(n)$

C. $O(n \log n)$

D. $O(n^2)$

算法分析概论-3

3. 下列程序段的时间复杂度是(C).

```
count=0;
```

```
for (k=1; k<=n; k*=2)
```

```
    for (j=1; j<=n; j++) count++;
```

A. $O(\log n)$

B. $O(n)$

C. $O(n \log n)$

D. $O(n^2)$

算法分析概论-4

4. 算法的计算量的大小称为计算的(B).

A. 效率

B. 复杂性

C. 现实性

D. 难度

5. 计算机算法指的是(C), 它必须具备(B)这三个特性。

(1)A. 计算方法

B. 排序方法

C. 解决问题的步骤

D. 调度方法

(2)A. 可执行性、可移植性、可扩充性

B. 可执行性、确定性、有穷性

C. 确定性、有穷性、稳定性

D. 易读性、稳定性、安全性

算法分析概论-5

6. 算法的时间复杂度取决于(C).

A. 问题的规模

B. 待处理数据的初态

C. A和B

7. 在汉诺塔递归中，假设碟子的个数为 n ，则时间复杂度为(**C**).

A. $O(n)$

B. $O(n^2)$

C. $O(2^n)$

D. $O(\sqrt{n})$

8. 设计一个“好”的算法应该达到的目标是(**B、D**)。

A. 可行的

B. 健壮的

C. 无二义性

D. 可读性好

算法分析概论-6

9. 计算算法的时间复杂度是属于一种(**B**)。

A. 事前统计的方法

B. 事前分析估算的方法

C. 事后统计的方法

D. 事后分析估算的方法

10. 算法分析的目的是(**C**)。

A. 找出数据结构的合理性

B. 研究算法中的输入和输出的关系

C. 分析算法的效率以求改进

D. 分析算法的易懂性和文档性

算法分析概论-7

11. 下面程序的算法复杂性为(**B**)。

```
for (int i=0; i<m; i++)
```

```
    for (int j=0; j<n; j++)
```

```
        a[i][j]=i*j;
```

A. $O(n^2)$

B. $O(m*n)$

C. $O(m^2)$

D. $O(m+n)$

算法分析概论-8

12. 在下列算法中，“ $x=x*2$ ”的执行次数是(A)。

```
int suanfa1 (int n)
{ int i, j, x=1;
  for (i=0; i<n; i++)
    for (j=i; j<n; j++) x=x*2;
  return x; }
```

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} 1 = \sum_{i=0}^{n-1} (n - i - 1)$$

A. $n(n+1)/2$

B. $n \log_2 n$

C. n^2

D. $n(n-1)/2$

算法分析概论-9

13. 执行下列算法suanfa2(1000), 输出结果是(C)。

```
void suanfa2 (int n)
{ int i=1;
  while (i<=n) i*=2;
  printf("%d", i); }
```

- A. 2000 B. 512 C. 1024 D. 2^{1000}

算法分析概论-10

14. 当 n 足够大时下述函数中渐进时间最小的是(**B**)。

A. $T(n) = n \log_2 n - 1000 \log_2 n$

B. $T(n) = n \log_2 3 - 1000 \log_2 n$

C. $T(n) = n^2 - 1000 \log_2 n$

D. $T(n) = 2n \log_2 n - 1000 \log_2 n$

15. 下列函数中渐近时间复杂度最小的是(**A**)。

A. $T(n) = \log_2 n + 5000n$

B. $T(n) = n^2 - 8000n$

C. $T(n) = n^3 + 5000n$

D. $T(n) = 2n \log_2 n - 1000n$

算法分析概论-11

16. 下面算法时间复杂度是(D)。

```
int suanfa3 (int n)
{ int i=1, s=1;
  while(s<n) s+=++i;
  return i; }
```

A. $O(n)$

B. $O(2^n)$

C. $O(\log_2 n)$

D. $O(\sqrt{n})$

算法分析概论-12

17. 某算法的时间复杂度为 $O(n^2)$ ，表明该算法的(C)。

A. 问题的规模是 n^2

B. 执行时间等于 n^2

C. 执行时间与 n^2 成正比

D. 问题规模与 n^2 成正比

18. 当输入非法错误时，一个“好”的算法回进行适当处理，而不会产生难以理解的输出结果，这称为算法的(B)。

A. 可读性

B. 健壮性

C. 正确性

D. 有穷性

算法概述简答题

1. 简述算法复杂性分析的方法及用途。分别给出两种以上原因解释程序员为什么会对算法的时间复杂度和空间复杂度感兴趣。

方法：解析方法和实验测量方法。从解析的角度又分为时间复杂度分析和空间复杂度分析。

用途：根据算法复杂度分析的结果可以区分出算法的快慢和适用范围，根据算法对空间和时间的要求设计满足合适的算法。

算法概述简答题

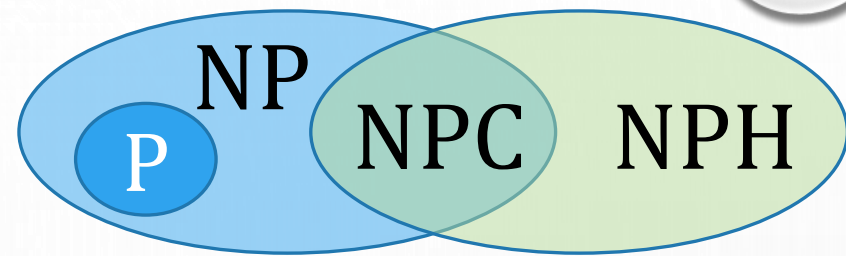
2. 算法 A 的计算时间 $f(n)$ 满足递归关系式： $f(n)=2f(n-2)+1, n>0; f(0)=1$ 。
请使用渐进符号分别表示 $f(n)$ 。

$$f(n)=O(2^n)$$

3. 算法 B 的计算时间 $g(n)=2g(n/2)+\log n, g(1)=c$ 。请使用渐进符号分别表示 $g(n)$ 。

$$g(n)=O(n)。$$

P、NP、NPC、NPH



4. 什么是NP难(NP hard)问题？NP hard问题一定是NP问题吗？

P类问题是存在多项式时间算法的问题，

NP问题是能在多项式时间内验证得出一个正确解的问题，P问题是NP问题的子集，因为存在多项式时间解法的问题，总能在多项式时间内验证他。

NPC问题首先是一个NP问题，同时所有的NP问题都可以约化成它。换句话说，只要解决了这个问题，那么所有的NP问题都解决了。所以它是NP问题的子集。

NPH问题满足NPC问题定义的第二条但不一定要满足第一条。就是说，NP-hard问题要比NPC问题的范围广。NPH问题没有限定属于NP，即所有的NP问题都能约化到它，但是它不一定是一个NP问题。所有的NPC问题都可以在多项式时间内转化为它。

算法复杂性分析

分析下面用伪代码描述的算法的复杂性，写出分析过程。 $w(n,n)$ ， $P(n)$ ， $Q(n)$ ， $c(n,n)$ 为算法中使用的数组并已初始化。(7分)

```
for m = 2 to n do {
```

```
    for i = 0 to n-m do{
```

```
        j = i + m
```

```
        w(i, j) = w(i, j-1) + P(i) + Q(j)
```

```
        c(i, j) = mini < k ≤ j { c(i, k-1) + c(k, j) } + w(i, j)
```

```
    }
```

```
}
```

$\min_{i < k \leq j} \{ c(i, k-1) + c(k, j) \}$ 的执行时间为 $O(j-i)=O(m)$;

内层for-循环的执行时间为 $O(m(n-m))$;

总的执行时间 $T(n)=O(\sum_{2 \leq m \leq n} m(n-m))=O(n^3)$

最坏情况复杂度分析

设 A 是 n 个实数的数组，考虑下面的递归算法：

XYZ($A[1 \dots n]$)

if $n=1$, then return $A[1]$

else $\text{Temp} \leftarrow \text{XYZ}(A[1 \dots n-1])$

 if $\text{Temp} < A[n]$

 then return Temp

 else return $A[n]$

$$T(n) = T(n-1) + O(1)$$

以 A 中元素的比较作为基本运算，列出算法 XYZ 最坏情况下的时间复杂度 $T(n)$ 的递推方程，并解出 $T(n)$. (8分)

矩阵乘法的时间复杂度

使用计步法分析下面算法的渐近复杂度/执行的乘法次数。

```
template <class T>
void Mult ( T **a, T **b, T **c, int n)
{ // Multiply the nxn matrices a and b to get c
  for (int i=0; i<m; i++)
    for (int j=0; j<p; j++) {
      T sum = 0;
      for (int k = 0; k < n; k++)
        sum += a[i][k]*b[k][j]; ← 1次
      c[i][j] = sum; }}
```

共 mpn 次乘法

s/e	Freq.	Total
0	0	$\Theta(0)$
0	0	$\Theta(0)$
1	$\Theta(m)$	$\Theta(m)$
1	$\Theta(mp)$	$\Theta(mp)$
1	$\Theta(mp)$	$\Theta(mp)$
1	$\Theta(mpn)$	$\Theta(mpn)$
1	$\Theta(mpn)$	$\Theta(n^3)$
1	$\Theta(mp)$	$\Theta(mp)$
Total:		$\Theta(mpn)$

最大最小问题

使用计步法分析下面算法的渐近复杂度/元素见的比较次数。

```
template <class T>
```

```
bool MinMax(T a[], int n, int & Min, int & Max)
```

```
{//Locate min and max elements in a [0:n-1].
```

```
//Return false if less than one element.
```

```
if (n<1) return false;           $n < 1$  时不比较
```

```
Min = Max = 0; // initial guess
```

```
for (int i=1; i< n; i++) {
```

```
    if (a[Min] > a[i]) Min = i;     $n-1$  次比较
```

```
    if (a[Max] < a[i]) Max = i; }  $n-1$  次比较
```

```
return true; }
```

共计 $2(n-1)$ 次比较

s/e	Freq.	Total
1	1	$\Theta(1)$
1	1	$\Theta(1)$
1	$\Theta(n)$	$\Theta(n)$
1	$\Theta(n)$	$\Theta(n)$
1	$\Theta(n)$	$\Theta(n)$
1	1	$\Theta(1)$
Total:		$\Theta(n)$

最大最小问题——小改进

试分析下面改进的算法在最好和最坏情形使用的元素比较次数

```
template <class T>
```

```
bool MinMax(T a[], int n, int & Min, int & Max)
```

```
{//Locate min and max elements in a [0:n-1].
```

```
//Return false if less than one element.
```

```
if (n<1) return false;
```

```
Min = Max = 0; // initial guess
```

```
for (int i=1; i< n; i++) {
```

```
    if (a[Min] > a[i]) Min = i;
```

```
    else if (a[Max] < a[i]) Max = i; }
```

```
return true; }
```

当输入数组已排好序时, 算法要做 $2(n-1)$ 次比较. 当输入为逆序时, 算法要做 $n-1$ 次比较.

判断, 1步, $n<1$ 时不比较

初始化, 1步

判断+i自增1, n 次比较 $n-1$ 次;

比较 $n-1$ 次,

最好情况比较0次, 最坏情况比较 $n-1$ 次

返回, 1步

顺序查找

试分析顺序查找算法中元素比较次数.

```
1. template <class T>
2. int SequentialSearch (T a[ ], const T & x, int n)
3. { // Search the unordered list a{0:n-1} for x.
4.   // Return position if found; return -1 otherwise.
5.   a[n] = x; // assume extra position available
6.   int i;
7.   for (i=0; a[i] != x; i++);
8.   if (i == n) return -1;
9.   return i; }
```

最好情况 $a[0] = x$, 比较1次;
最坏情况 x 不在数组中, 比较 $n+1$ 次;
当仅考虑成功查找时, 平均情况比较 $(n+1)/2$ 次。

RANK

试用计步法分析下面算法的渐进时间复杂度。

```
1. template <class T>
2. void Rank ( T a[], int n, int r[])
3. { //Rank the n elements a[0:n-1].
4.     for (int i=0; i<n; i++)
5.         r[i] = 0; // initialize
6.     // compare all element pairs
7.     for (int i=1; i<n; i++)
8.         for (int j=0; j<i; j++)
9.             if (a[j] <= a[i]) r[i]++;
10.            else r[j]++; }
```

s/e	Frequency	Total steps
1	$\Theta(n)$	$\Theta(n)$
1	$\Theta(n)$	$\Theta(n)$
1	$\Theta(n)$	$\Theta(n)$
1	$\Theta(\sum_{i=1}^{n-1} i)$	$\Theta(n^2)$
1	$\Theta(n^2)$	$\Theta(n^2)$
1	$\Omega(0), O(n^2)$	$\Omega(0), O(n^2)$
total:		$\Theta(n^2)$

插入排序

分析插入排序算法在最好、最坏和平均情形所用的关键字比较次数。

`insertionsort` (`a`, `n`)

`for` `j` ← 2 `to` `n`

`do` `key` ← `a[j]`

`i` ← `j`-1

`while` `i`>0 and `a[i]`>`key`

`do` `a[i+1]` ← `a[i]`

`i` ← `i`-1

`a[i+1]`=`key`

最好情况：输入数据已经排好序，比较 $n-1$ 次；

最坏情况：输入数据按倒序排列，比较

$\sum_{j=2}^n (j-1) = \Theta(n^2)$ 次；

平均情况：比较 $\sum_{j=2}^n [(j-1)/2] = \Theta(n^2)$ 次。

代入法

假定 $T(n)$ 满足以下递归式:

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{cn}_{\text{combine}} & \text{otherwise} \end{cases}$$

用归纳法 $T(n) \leq cn\lceil\log_2 n\rceil$ 。

分析: 设 $2^k < n \leq 2^{k+1}$, 有 $\lceil\log n\rceil = k+1$; 又因为 $2^{k-1} < n/2 \leq 2^k$, 有 $2^{k-1} < \lfloor n/2 \rfloor \leq 2^k$, $\lceil\log \lfloor n/2 \rfloor\rceil = k$; $\lceil\log n\rceil = \lceil\log \lfloor n/2 \rfloor\rceil + 1$ 。

证明 当 $n=1$ 时, 命题成立。假设不等式对 $1, 2, \dots, n-1$ 都成立, 则对于 n , 有

$$\begin{aligned} T(n) &= T(n_1) + T(n_2) + cn \quad (\text{记 } n_1 = \lfloor n/2 \rfloor, n_2 = \lfloor n/2 \rfloor) \\ &\leq cn_1 \lceil\log n_1\rceil + cn_2 \lceil\log n_2\rceil + cn \quad (\text{由 } n_1 \leq n_2 \text{ 得 } \lceil\log n_1\rceil \leq \lceil\log n_2\rceil) \\ &\leq cn \lceil\log n_2\rceil + cn = cn(\lceil\log n_2\rceil + 1) \quad (\lceil\log n\rceil = \lceil\log \lfloor n/2 \rfloor\rceil + 1) \\ &= cn \lceil\log n\rceil \end{aligned}$$

代入法

假定 $T(n)$ 满足以下递归式:

$$\begin{cases} T(n)=c & n=1, \\ T(n)=4T(n/2)+n & n>1 \end{cases}$$

式中 c 为正的常数。用归纳法证明 $T(n) \leq O(n^2)$ 。

证明 不妨设 $T(n) \leq c_1n^2 - c_2n$, c_1 和 c_2 为待定常数。

当 $n = 1$ 时, 可取 c_1 足够大, 使得 $T(1) = c \leq c_1 - c_2$, 命题成立。假设 $T(k) \leq c_1k^2 - c_2k$ 对 $k < n$ 都成立, 那么

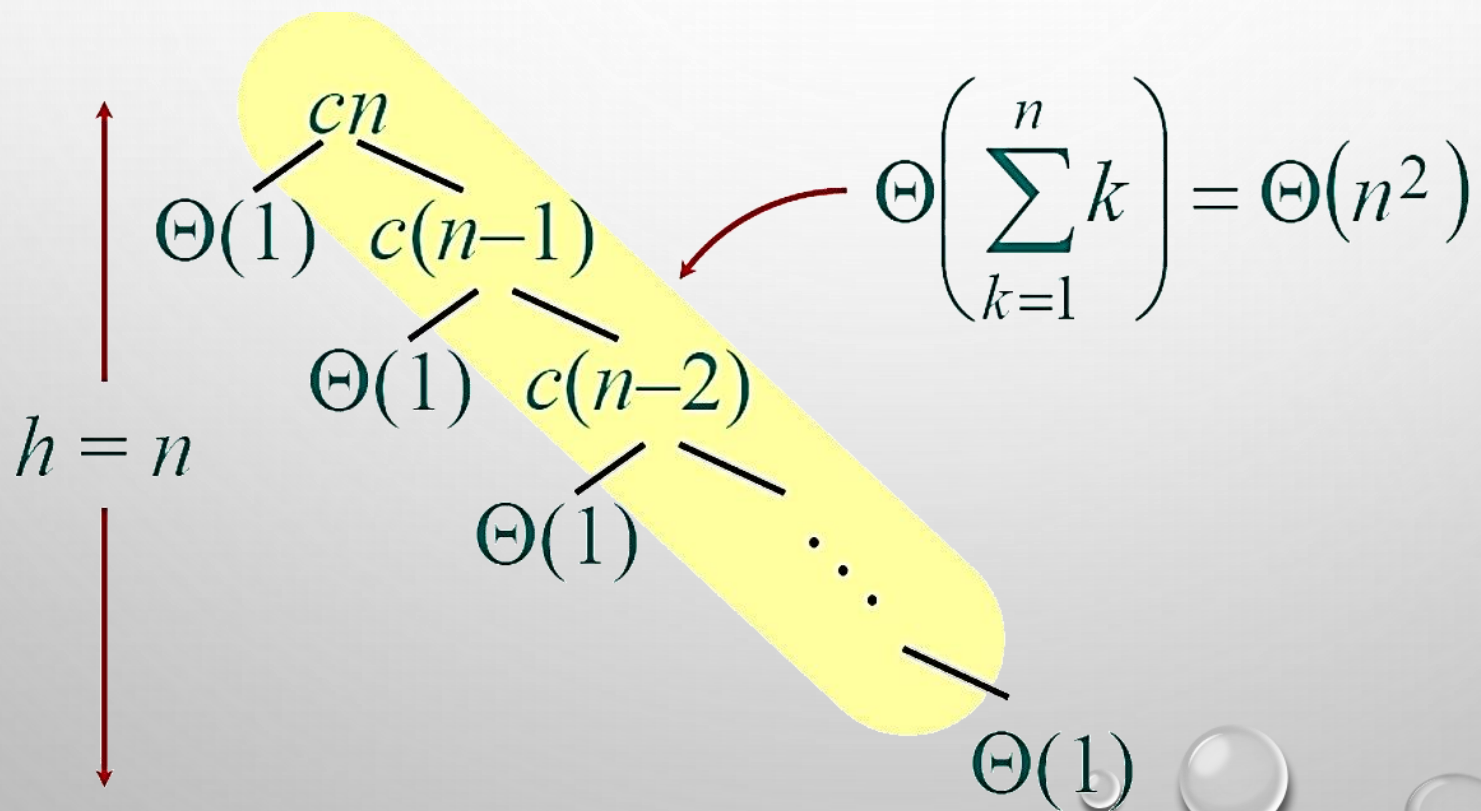
$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n \leq 4\left[c_1\left(\frac{n}{2}\right)^2 - c_2\left(\frac{n}{2}\right)\right] + n \\ &= c_1n^2 - c_2n - (c_2n - n) \\ &\leq c_1n^2 - c_2n \quad (\forall c_2 > 1) \end{aligned}$$

递归树-1

$$T(n) = T(a) + T(n-a) + cn$$

展开递归树： $T(n) = T(0) + T(n-1) + cn$, 并做渐近分析。

解： $T(n) = nT(0) + c[1 + \dots + (n-1) + n] = \Theta(n^2)$,

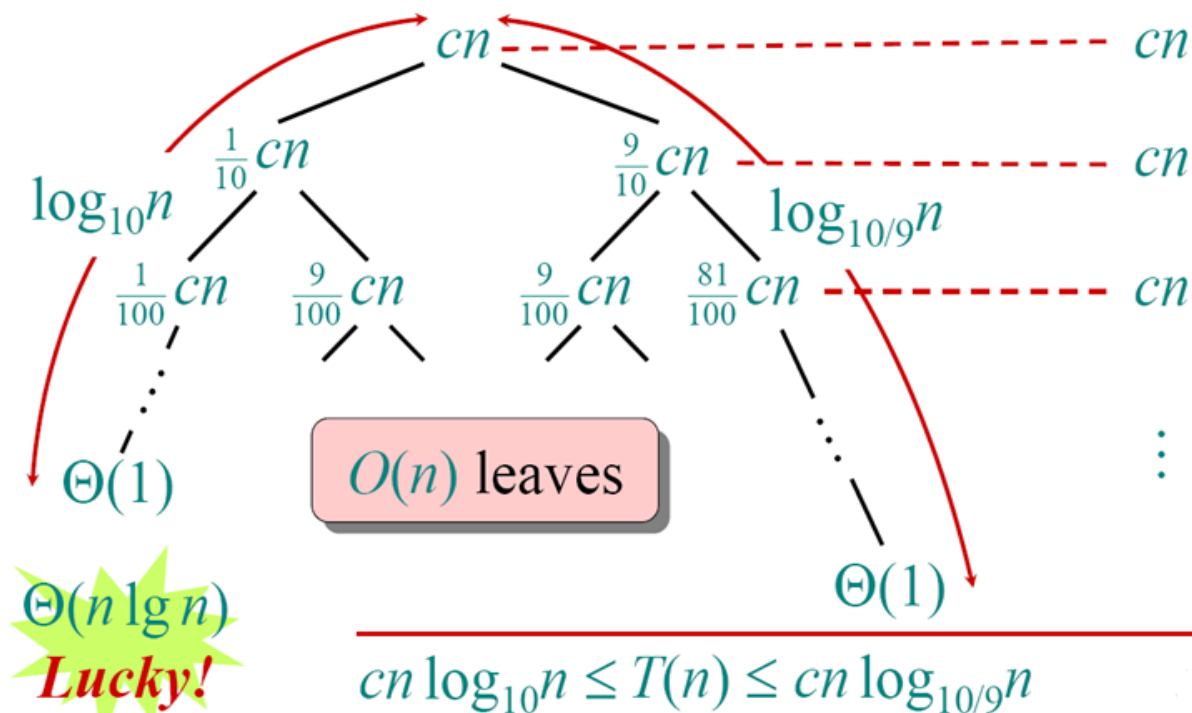


递归树-2

$$T(n) = T(\alpha n) + T(\beta n) + \theta(n)$$

展开 $T(n) = T(0.1n) + T(0.9n) + \Theta(n)$ 的递归树，计算递归树的深度和 $T(n)$ 的渐近值。(10分)

Analysis of “almost-best” case



主方法

主定理 令 $a \geq 1$ 和 $b > 1$ 是常数, $f(n)$ 是一个函数,

$$T(n) = aT(n/b) + f(n)$$

是定义在非负整数上的递归式, 其中 n/b 为 $\lfloor n/b \rfloor$ 或 $\lceil n/b \rceil$ 。那么 $T(n)$ 有如下渐进界:

1. 若对某个常数 $\varepsilon > 0$ 有 $f(n) = O(n^{\log_b a - \varepsilon})$, 则 $T(n) = \Theta(n^{\log_b a})$.
2. 若 $f(n) = \Theta(n^{\log_b a})$, 则 $T(n) = \Theta(n^{\log_b a} \lg n)$ 。
3. 若对某个常数 $\varepsilon > 0$ 有 $f(n) = \Omega(n^{\log_b a + \varepsilon})$, 且对某个常数 $c < 1$ 和所有足够大的 n 有 $af(n/b) \leq cf(n)$, 则 $T(n) = \Theta(f(n))$ 。

主方法 case 1

1) $T(n) = 2T(n/2) + \Theta(n^{1/2})$

$a=b=2, n^{\log_b a} = n$, 取 $\varepsilon = 1/2$,

$f(n) = cn^{1/2} = O(n^{1-1/2})$ 。

\therefore case 1, $T(n) = \Theta(n)$

2) $T(n) = 10T(n/3) + 11n$

$a=10, b=3, n^{\log_b a} > n^2$. 取 $\varepsilon = 1$,

$f(n) = cn = O(n^{2-1})$.

\therefore case 1, $T(n) = \Theta(n^{\log_3 10})$

3) $T(n) = 128T(n/2) + 6n$

$a=128, b=2, n^{\log_b a} = n^7$, 取 $\varepsilon = 3$,

$f(n) = 6n = O(n^{7-3})$

\therefore case 1, $T(n) = \Theta(n^7)$

4) $T(n) = 4T(n/2) + n^{1.9}$

$a=4, b=2, n^{\log_b a} = n^2$, 取 $\varepsilon = 0.03$,

$f(n) = n^{1.9} = O(n^{2-0.03})$,

case 1. $T(n) = \Theta(n^2)$

主方法 case 2

$$1) T(n) = 27T(n/3) + 11n^3$$

$$a=27, b=3, n^{\log_b a} = n^3.$$

$$f(n) = cn^3.$$

$$\therefore \text{case 2, } T(n) = \theta(n^3 \log n);$$

$$2) T(n) = 4T(n/2) + n^2$$

$$a=4, b=2, n^{\log_b a} = n^2 = \Theta(f(n)).$$

$$\text{case 2, } T(n) = \Theta(n^3 \log n).$$

Case 2的一般形式:

$$f(n) = \Theta(n^{\log_b a} \lg s n), \quad s \geq 0 \text{ 为某一常数.}$$

$$\text{Solution: } T(n) = \Theta(n^{\log_b a} \lg s + 1n)$$

$$3) T(n) = 64T(n/4) + 10n^3 \log^2 n$$

$$a=64, b=4, n^{\log_b a} = n^3.$$

$$f(n) = cn^3 \log^2 n.$$

$$\text{解1: } f(n) = \Omega(n^{\log_b a})$$

找不到 $\varepsilon > 0$, s.t. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ 。

不符合 case3, 不能用主定理求解。

$$\text{解2: case 2的一般情况,}$$

$$T(n) = \theta(n^3 \log^2 n)$$

主方法 case 3

1) $T(n) = 10T(n/3) + 11n^5$

$a=10, b=3, n^{\log_b a} < n^3$. 取 $\varepsilon=2$,

$f(n) = 11n^5 = \Omega(n^{5-2})$.

又因为当 $c=0.5$ 时, 有

$af(n/b) = an^5/b^5 < 0.5 \times 11n^5 = cf(n)$

$\therefore \text{case 3, } T(n) = \Theta(n^5)$

2) $T(n) = 9T(n/2) + \Theta(n^2 2^n)$

$a=9, b=2, f(n) = n^2 2^n$,

任取 $c, T(n) = \Theta(n^2 2^n)$

3) $T(n) = 3T(n/8) + \Theta(n^2 2^n \log n)$

$a=3, b=8, f(n) = n^2 2^n \log n$,

任取 $c, T(n) = \Theta(n^2 2^n \log n)$

4) $T(n) = 128T(n/2) + \Theta(n^8)$

$\log_b a = 7, f(n) = n^8 = \Omega(n^{7+\varepsilon})$,

$8(n/2)^8 < cn^8, T(n) = \Theta(n^8)$

5) $T(n) = 128T(n/2) + 2^n/n$

$T(n) = \Theta(2^n/n)$

算法篇

本课程的涉及到的算法

- 贪心算法
- 分治法
- 动态规划
- 回溯法
- 分支限界法



本课程的涉及到的算法

- 递归和分治：归并排序、快速排序、寻找第k小、中间的中间、金块问题、棋盘覆盖、循环赛日程、最接近点对等
- 贪心算法：货箱装船问题、活动安排问题、最短路径问题(Dijkstra算法、Bellman-Ford算法)、最小生成树问题(PRIM算法、Kruskal算法)、哈夫曼编码问题、拓扑排序问题、偶图覆盖问题等
- 动态规划：斐波那契数、多段图最短路径、矩阵乘法链、All-Pair最短路(Floyd-Warshall算法)、不交叉网子集等
- 回溯法：8-皇后问题、货箱装船问题、图的m着色问题、最大团问题、子集和数问题、多段图问题等
- 分支限界法：旅行商问题(哈密顿回路)、作业调度问题等

递归和分治

递归和分治

- 归并排序
- 快速排序
- 寻找第k小
- 中间的中间
- 金块问题(最大最小问题)
- 最接近点对问题
- 二分搜索算法
- 大整数的乘法
- 矩阵乘法
- 棋盘覆盖
- 循环赛日程
- 最接近点对等

分治法的思想

叙述分治法算法的思想并用归并排序算法说明。

思想：

1. 把大问题分成两个或多个更小的子问题；
2. 分别解决每个子问题。如果子问题的规模不够小，则再划分成两个或多个更小的子问题。如此递归地进行下去，直到问题规模足够小，很容易求出其解为止。
3. 把这些子问题的解组合成原始大问题的解。

归并排序

Merge-Sort $A[1..n]$:

1. if $n=1$, done.

2. $p=\lfloor n/2 \rfloor$

把大问题分成
两个或多个更
小的子问题;

3. Merge-Sort $a[1..p]$

分别解决每
个子问题

4. Merge-Sort $a[p+1..n]$

5. Merge $L[1..p]$ 和 $R[1..q]$.

把这些子问题的解组
合成原始大问题的解。

Merge $L[1..p]$ 和 $R[1..q]$:

1. $l[p+1]=\infty; r[q+1]=\infty$

2. $i=1; j=1$

3. for $k=1$ to $p+q$

4. if $l[i] \leq r[j]$

5. $a[k]=l[i]$

6. $i=i+1$

7. else $a[k]=r[j]$

8. $j=j+1$

分治法的基本思想和适用条件

叙述分治法设计算法的基本思想和适用条件。

思想：将要求解的较大规模的问题分割成 k 个更小规模的子问题；这 k 个子问题分别求解。如果子问题的规模仍然不够小，则再划分为 k 个子问题，如此递归的进行下去，直到问题规模足够小，很容易求出其解为止；将求出的小规模的问题的解合并为一个更大规模的问题的解，自底向上逐步求出原来问题的解

适用条件：该问题的规模缩小到一定的程度就可以容易地解决；该问题可以分解为若干个规模较小的相同问题；分解出的子问题的解可以合并为原问题的解；分解出的各个子问题是相互独立的

金块问题(最大最小问题)

- 问题：有若干金块试用一台天平找出其中最轻和最重的金块.
- 相当于在 n 个数中找出最大和最小的数.
- 如果是12枚个金块，其中有一块金块重量不一样，多少次可以找出来？

最大最小问题 (金块问题)

以下伪代码是用分治法设计的求解最大-最小问题的算法:

```
Max-Min(A[1,n],max,min)
if n=1 max←min←a[1],return;
if n=2 比较a[1]和a[2]得到max和min
else m←n/2
    Max-Min(A[1,m],max1,min1),
    Max-Min(A[m+1,n],max2,min2),
    max←max(max1,max2),
    min←min(min1,min2),
    return.
```

试分析在 $n=2^k$ 时算法所用的关键字比较次数.

最大最小问题 复杂度分析

设 $c(n)$ 为使用分治法所需要的比较次数。因为 $n=2^k$ ，有：

$$c(n) = \begin{cases} 0 & n=1 \\ 1 & n=2 \\ 2c(n/2)+2 & n>2 \end{cases}$$

迭代展开可得

$$c(n) = 2^{k-1}c(2) + \sum_{1 \leq i \leq k-1} 2^i = (3n/2) - 2$$

或者：

$$a = 2, b = 2, \log_b a = 1, n^1 = n, f(n) = 2 = \Theta(1).$$

根据主定理， $c(n) = \Theta(n)$ 。

快速排序算法

QuickSort (A, p, r)

 if $p < r$

 then $q \leftarrow \text{Partition}(A, p, r)$

 QuickSort (A, p, $q-1$)

 QuickSort (A, $q+1$, r)

1. **partition** (A, p, r)

2. $x \leftarrow A[p]$

3. $i \leftarrow p$

4. **for** $j \leftarrow p+1$ to q

5. **do if** $A[j] \leq x$

6. **then** exchange $A[i] \leftrightarrow A[j]$

7. $i \leftarrow i+1$

8. exchange $A[p] \leftrightarrow A[i]$

9. **return** i

快速排序时间复杂度分析：最好、最坏

- 最坏情况发生在划分过程产生的两个区域分别包含 $n-1$ 个元素和1个元素的时候。

$$T_{\max}(n) = \begin{cases} O(1) & n \leq 1 \\ T(n-1) + O(n) = O(n^2) & n > 1 \end{cases}$$

- 最好情况下，每次划分所取的基准都恰好为中值，即每次划分都产生两个大小为 $n/2$ 的区域，此时

$$T_{\min}(n) = \begin{cases} O(1) & n \leq 1 \\ 2T(n/2) + O(n) = O(n \log n) & n > 1 \end{cases}$$

快速排序时间复杂度分析：平均

命题： n 个元素的快速排序的平均复杂度是 $\Theta(n \log n)$.

证明： 每轮快速排序将数组分成两部分。用 s 表示左数据段元素个数，则右数据段元素个数为 $n - s - 1$ 。 S 取 0 到 $n - 1$ 中任何数的概率为 $1/n$ 。分割数组元素所需时间为 cn ，所以

$$T(n) = \frac{1}{n} \sum_{s=0}^{n-1} [T(s) + T(n-s-1)] + cn = \frac{2}{n} \sum_{s=0}^{n-1} T(s) + cn$$

由 $nT(n) - (n-1)T(n-1) = 2T(n-1) + O(n)$ 得

$$T(n) = \left(1 + \frac{1}{n}\right) T(n-1) + O(n)$$

$$\frac{\sum_{s=0}^{n-1} T(s)}{\sum_{s=0}^{n-1} T(n-s-1)}$$

用归纳法可以证明， $T_{\text{avg}}(n) = \Theta(n \log n)$ 。

中间的中寻找第k小做快速排序的支点

1. 将 n 个元素分成5个一组，找到每组的中位数。 $\leftarrow \Theta(n)$
 2. 重复步骤1直到找到支点元素 x 。 $\leftarrow T(n/5)$
 3. 用支点元素将数组分为两部分， $i = \text{rank}(x)$ 。 $\leftarrow \Theta(n)$
 4. **if** $k=i$ **then return** x
 5. **elseif** $k < i$
 6. **then** 在左半部分寻找第 k 小元素
 7. **else** 在右半部分寻找第 $(k-i)$ 小元素
- } $T(3n/4)$

$$T(n) \leq T(n/5) + T(3n/4) + \Theta(n)$$

中间的中寻找第k小的时间复杂度

- 求解递归式: $T(n) \leq T(n/5) + T(3n/4) + \Theta(n)$
- 代入法: 假设 $T(n) \leq cn$,

$$T(n) \leq T(n/5) + T(3n/4) + \Theta(n)$$

$$\leq cn/5 + 3cn/4 + \Theta(n)$$

$$= 19cn/20 + \Theta(n)$$

$$= cn - (cn/20 - \Theta(n))$$

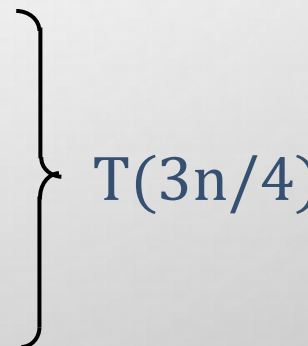
$$\leq cn \text{ (取足够大的 } c \text{)}$$

用第K小做支点的快速排序时间复杂度

- 在快速排序算法中，如果我们先调用最坏情形时间复杂度 $O(n)$ 的选择算法找出第 $n/2$ 小元素并以此为支点(pivot)对要排序的数组进行分划，则改进后的快速排序算法的最坏情形时间复杂度 $T(n)$ 是什么？假定 $n=2^k$ ，列出 $T(n)$ 满足的递归方程并分析.
- 解答：用第 $n/2$ 小的元素做支点，把数组分成均等的两半，同时，分化的最坏情形时间代价为 $O(n)=cn$ ，则改进快速排序算法的最坏时间复杂度为 $T(n)=2T(n/2)+cn$ 。符合主定理case2， $T(n)=\Theta(n\log n)$ 。
假定 $n=2^k$ ， $T(n)=2T(n/2)+cn=2^kT(1)+k\cdot cn=\Theta(n+n\log n)=\Theta(n\log n)$ 。

寻找第K大元素

设计一个分治算法来找出 n 个元素中的第 k 大元素，请写出算法的伪代码并分析算法的最好及最坏时间复杂度。

1. 将 n 个元素分成5个一组，找到每组的中位数。 $\leftarrow \Theta(n)$
 2. 重复步骤1直到找到支点元素 x 。 $\leftarrow T(n/5)$
 3. 用支点元素将数组分为两部分， $i = \text{rank}(x)$. $\leftarrow \Theta(n)$
 4. **if** $k=i$ **then return** x // 左大右小
 5. **elseif** $k < i$
 6. **then** 在左半部分寻找第 k 大元素
 7. **else** 在右半部分寻找第 $(k-i)$ 大元素
- 

$T(3n/4)$

$$T(n) \leq T(n/5) + T(3n/4) + \Theta(n)$$

中间的中寻找第 k 大

1. 将n个元素分成5个一组，找到每组的中位数。 $\leftarrow \Theta(n)$
 2. 重复步骤1直到找到支点元素x。 $\leftarrow T(n/5)$
 3. 用支点元素将数组分为两部分， $i = \text{rank}(x)$ 。 $\leftarrow \Theta(n)$
 4. **if** $k=i$ **then return** x // 左大右小
 5. **elseif** $k < i$
 6. **then** 在左半部分寻找第 k 大元素
 7. **else** 在右半部分寻找第 (k-i) 大元素
- } $T(3n/4)$

$$T(n/5) + \Theta(n) \leq T(n) \leq T(n/5) + T(3n/4) + \Theta(n)$$

最好情况，x=第k大

最坏情况，代入法可证明 $T(n)=\Theta(n)$

求逆序对个数

令 a_1, \dots, a_n 为 n 个元素的一个序列。元素 a_i 和 a_j 是逆序的，当且仅当 $a_i > a_j$ ($i < j$)。在元素序列中，具有逆序关系的元素对 (a_i, a_j) 的个数被称为逆序数。

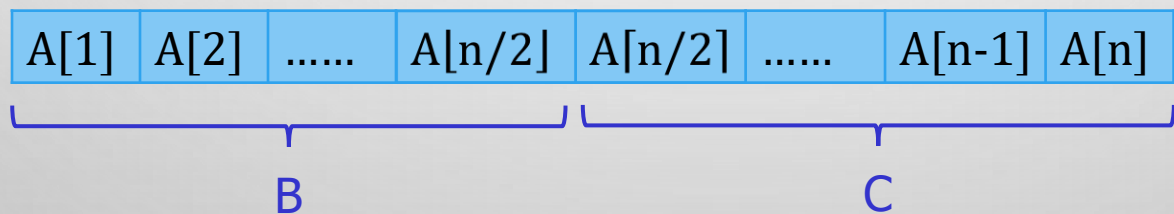
- 1) 序列 6, 2, 3, 1 的逆序数是多少？ $3+1+1=5$ 个
- 2) 在一个 n 个元素的序列中，最大的逆序数是多少？倒序，共有 $(n-1)n/2$ 个
- 3) 试用分治法设计一个计算给定排列的逆序总数的算法，并分析该算法的时间复杂度。

提示：可在 $O(n)$ 时间内算出 2 段排好序的子序列之间的逆序数。例如序列 (1, 2, 4, 8, 3, 5, 6, 7) 中，(1, 2, 4, 8) 为排在左边的子序列，(3, 5, 6, 7) 为排在右边的子序列，它们之间的逆序数可用一个线性时间的算法算出。

求逆序对个数的递归关系式

- $f(i, j)$ 为 i 到 j 号元素中的逆序对个数。
- $s(i, j, k)$ 表示以 k 为分割点，第一个元素在 i 到 k 中，第二个元素在 $k + 1$ 到 j 中形成的逆序对数。

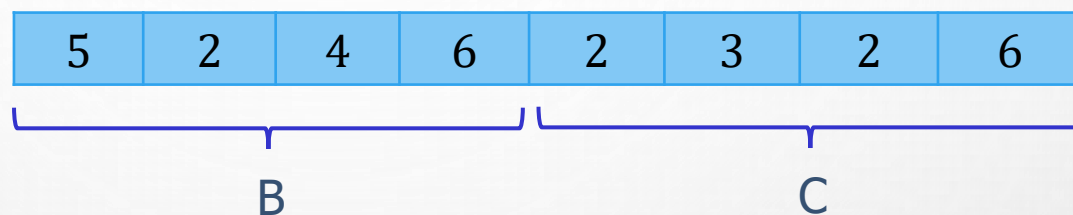
- 递归关系式：
$$\begin{cases} f(i, i) = 0; \\ f(i, i+1) = f(i, i) + f(i+1, i+1) + s(i, i+1, i); \\ f(i, j) = f(i, k) + f(k+1, j) + s(i, j, k) \end{cases}$$
- 如何来统计序列 B 和 C 之间的“逆序对”呢？



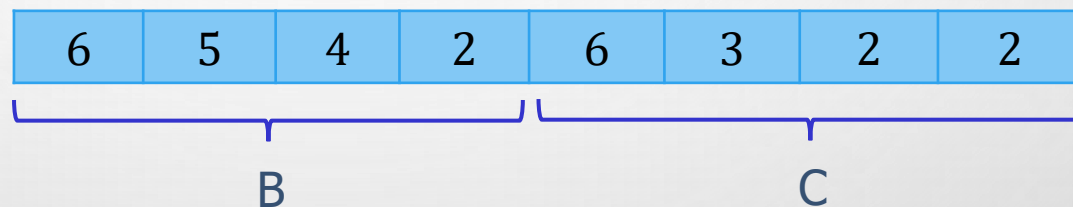
求B、C之间的逆序对个数

- 在递归的求解 B、C 两个序列中的逆序对的个数以后，如果对 B、C 两个序列当中的元素进行排列的话，要统计 B 和 C 两个序列之间的“逆序对”是非常容易的。排序前：

5	2	4	6	2	3	2	6
---	---	---	---	---	---	---	---



- 排序后:



- 在 B 数组当中，首先，B 中的 6，5，4 都与 C 数组当中的 3，2，2 都构成了“逆序对”，而 2 不会构成逆序对，因此 B、C 两个数组之间构成的逆序对的个数为 $3+3+3=9$ 。

求逆序对个数的伪代码

1. 将输入排列 $A[1, n]$ 分成两个子排列 $A[1, n/2]$ 和 $A[n/2+1, n]$;
2. 递归对这两个子排列应用该算法, 得到逆序数 $n1$ 和 $n2$;
3. 对这两个子排列排序;
4. 使用线性时间算法计算排序后两个子排列间的逆序数, 设其为 $n3$;
5. $n1 + n2 + n3$ 即为原始的输入排列的逆序数;

求逆序对个数的时间复杂度

- 虽然对 B、C 两个序列当中的元素进行了排序，使得序列 A 当中某一部分元素的顺序被打乱了，但由于求解过程是递归定义的，在排序之前 B、C 两个序列各自其中的元素之间的“逆序对”个数已经被统计过了，并且排不排序对统计 B、C 两个数组之间的“逆序对”的个数不会产生影响。所以排序过程对整个问题的求解的正确性是没有任何影响的。
- B、C 之间的逆序对个数可以在线性时间完成，参照归并排序的合并过程。
- 排序的过程可在递归求解子问题时完成，算法的时间复杂度为 $T(n) = 2T(n/2) + O(n\log n)$ 。根据主定理 case 3, $T(n) = O(n\log n)$ 。

最接近点对问题

给定平面上 n 个点的集合 S ，找其中的一对点，使得在 n 个点组成的所有点对中，该点对间的距离最小。试给出求解该问题的算法并做复杂度分析。

算法复杂度分析：

$$T(n) = \begin{cases} O(1) & n < 4 \\ 2T(n/2) + O(n \log n) & n \geq 4 \end{cases}$$

1. $m = S$ 中各点 x 间坐标的中位数；构造 S_1 和 S_2 ；

2. $d_1 = \text{cpair2}(S_1)$; $d_2 = \text{cpair2}(S_2)$;
 $dm = \min(d_1, d_2)$;

3. P_i 是 S_i 中距垂直分割线 ℓ 的距离在 dm 之内的所有点组成的集合, $i = 1, 2$;

4. 设 P_1 和 P_2 中的点已按 y 坐标值排好序；

5. 扫描 P_1 ，对于 P_1 中的每个点检查 P_2 中与其距离在 dm 之内的所有点(最多6个)。当 x 中的扫描指针逐次向上移动时， y 中的扫描指针可在宽为 $2dm$ 的区间内移动。 dl 是按这种扫描方式找到的点对间的最小距离；

6. $d = \min(dm, dl)$;

背包问题

0/1 背包问题

- 贪心算法：把物品按密度降序排列，每次选密度最大的. K-优化算法是上述密度贪心算法的改进，改进后其误差可控制在 $1/(K+1)*100\%$ 之内. 例如3-优化算法的误差为25%.

- 动态规划：递归式 $c(1,y)=\begin{cases} v_1 & y \geq w_1 \\ 0 & 0 \leq y < w_1 \end{cases}$;

$$c(i,y)=\begin{cases} \max\{c(i-1,y), c(i-1,y-w_i) + v_i\} & y \geq w_i \\ c(i-1,y) & 0 \leq y < w_i \end{cases}, \text{ 元组法。}$$

- 回溯法：把物品按密度降序排列，约束函数： $\sum_{i=0}^n w_i x_i \leq c$
- 分治限界：优先队列

0/1 背包问题DP法的递归公式

问题描述：给定 $c > 0$, $w_i > 0$ ($1 \leq i \leq n$), 要求找出一个 n 元0-1向量 (x_1, x_2, \dots, x_n) ($1 \leq i \leq n$), 使得 $\sum_{i=1}^n w_i x_i \leq c$, 且 $\sum_{i=1}^n v_i x_i$ 达到最大.

整数规划问题:
$$\begin{cases} \max \sum_{i=1}^n v_i x_i \\ \text{s.t. } \sum_{i=1}^n w_i x_i \leq c \end{cases}, \text{ 其中 } x_i = \begin{cases} 0 & \text{如果不选择第} i \text{件珍宝} \\ 1 & \text{如果选择第} i \text{件珍宝} \end{cases}$$

递归公式：设 $m(i, j)$ 是背包容量为 j , 可选择物品为 $i, i+1, \dots, n$ 时0-1背包问题的最优值, 有

$$m(i, j) = \begin{cases} \max\{m(i+1, j), m(i+1, j-w_i) + v_i\} & j \geq w_i \\ m(i+1, j) & 0 \leq j < w_i \end{cases}$$

$$m(n, j) = \begin{cases} v_n & j \geq w_n \\ 0 & 0 \leq j < w_n \end{cases}$$

0/1 背包问题DP法的时间复杂度

证明：当重量和效益值均为整数时动态规划算法的时间复杂度为 $O(\min\{2^n, n\sum_{1 \leq i \leq n} v_i, nc\})$.

- 每次产生一个表 $P(i)$ 时的计算时间和表的长度成正比，因为表 $P(i)$ 按 w_i 和 v_i 的增序排列, w_i 和 v_i 都是整数, 表 $P(i)$ 的长度不超过 $\min\{1 + \sum_{1 \leq i \leq n} v_i, c\}$. 共产生 n 个表，总时间为 $O(\min\{n\sum_{1 \leq i \leq n} v_i, nc\})$.
- 计算 Q 需 $O(|P(i-1)|)$ 的时间, 合并 $P(i-1)$ 和 Q 需要 $O(|P(i-1)| + |Q|) = O(2|P(i-1)|)$ 的时间. \therefore 计算 $P(i)$ 需 $O(2^{n-i+1})$ 时间. 计算所有 $P(i)$ 时所需要的总时间 $O(2^n)$.
 \therefore 存在输入使算法最坏情形为 2^n 量级.
- 所以总时间不超过 $O(\min\{2^n, n\sum_{1 \leq i \leq n} v_i, nc\})$. 其中, 2^n 是考虑到每次新产生的表顶多是前一个表长度的2倍。

0/1 背包问题

(1) 用1-优化法求解以下0/1背包问题的实例：

$$n = 5, w = [16, 20, 5, 15, 10], p = [32, 20, 15, 30, 15], c = 40.$$

要求写出计算过程.

(2) 分析1-优化法的时间复杂度.

0/1 背包问题的贪心解法

$K=0$ 时，即一般的密度贪心法。 $n = 5$, $w = [16, 20, 5, 15, 10]$, $p = [32, 20, 15, 30, 15]$, $c = 40$ 。

i	1	2	3	4	5
v_i	32	20	15	30	15
w_i	16	20	5	15	10
v_i/w_i	2	1	3	2	1.5

贪心解为 (1,0,1,1,0)，效益值为77。

0/1 背包问题的1-优化解法

$n = 5, w = [16, 20, 5, 15, 10], p = [32, 20, 15, 30, 15], c = 40$ 。当前解为 $(1, 0, 1, 1, 0)$ ，效益值为77。

K=1时, 先挑出一个物品放入袋里, 需要考虑2种可能。

➤ $x_2=1$: 对其余物品做密度贪心法, 得到解 $(0, 1, 1, 1, 0)$, 效益值65。

➤ $x_5=1$: 对其余物品做密度贪心法, 得到解 $(0, 0, 1, 1, 1)$, 效益值60。

➤ 三个效益值中取最大的, 得到k-优化解 $(1, 0, 1, 1, 0)$, k-优化值为77。

0/1 背包问题的1-优化解法的时间复杂度

1-优化算法的时间复杂度：

- 需要测试的子集数目为 $O(n)$;
- 每一个子集做贪心法需要时间 $O(n)$;
- 因此当 $k=1$ 时，总的时间开销为 $O(n^2)$.

0/1 背包问题的动态规划解法

设 $g(i, x)$ 表示物品 $1, \dots, i$, 背包容量 x 的 0/1 背包问题的优化效益值。

(1) 试写出 $g(i, x)$ 满足的动态规划递归关系式

(2) 就以下实例, $n = 5$, $c = 10$, $w = (6, 3, 5, 4, 6)$, $p = (2, 2, 6, 5, 4)$, 用元组法计算, 并回溯求出优化的物品装法。

(3) 就以下实例, $n = 4$, $c = 20$, $w = (10, 15, 6, 9)$, $p = (2, 5, 8, 1)$, 用元组法计算, 并回溯求出优化的物品装法。

0/1 背包问题的动态规划递归关系式

(1) $g(i, x)$ 满足的动态规划递归关系式:

$$g(i, x) = \begin{cases} \max\{g(i-1, x), g(i-1, x-w_i) + p_i\} & x \geq w_i \\ g(i-1, x) & 0 \leq x < w_i \end{cases}$$

$$g(1, x) = \begin{cases} p_1 & x \geq w_1 \\ 0 & 0 \leq x < w_1 \end{cases}$$

0/1 背包问题DP法的递归求解

$n=5, w=[2,2,6,5,4], p=[6,3,5,4,6], c=10$

$$f(5, y) = \begin{cases} 6, & y \geq 4 \\ 0, & y < 4 \end{cases} \quad P[5]=[(0,0),(4,6)]$$

$$f(4, y) = \begin{cases} \max(f(5, y), f(5, y - 5) + 4) & y \geq w_4 \\ f(5, y), & y < w_4 \end{cases} = \begin{cases} 6, & 4 \leq y < 5 \\ 0, & y < 4 \\ 6, & 5 \leq y < 9 \\ 10, & y \geq 9 \end{cases}$$

$$= \begin{cases} 0, & y < 4 \\ 6, & 4 \leq y < 9 \\ 10, & y \geq 9 \end{cases} \quad P[4]=[(0,0),(4,6),(9,10)]$$

0/1 背包问题DP法的递归求解

$$\bullet f(3, y) = \begin{cases} \max(f(4, y), f(4, y - 6) + 5), & y \geq w_3 \\ f(4, y), & y < w_3 \end{cases} = \begin{cases} 0, & y < 4 \\ 6, & 4 \leq y < 6 \\ 6, & 6 \leq y < 9 \\ 10, & 9 \leq y < 10 \\ 11, & y \geq 10 \end{cases}$$

$$= \begin{cases} 0, & y < 4 \\ 6, & 4 \leq y < 9 \\ 10, & 9 \leq y < 10 \\ 11, & y \geq 10 \end{cases}$$

$$P[3] = [(0, 0), (4, 6), (9, 10), (10, 11)]$$

$$\bullet f(2, y) = \begin{cases} \max(f(3, y), f(3, y - 2) + 3), & y \geq w_2 \\ f(3, y), & y < w_2 \end{cases} = \begin{cases} 0, & y < 2 \\ 3, & 2 \leq y < 4 \\ 6, & 4 \leq y < 6 \\ 9, & 6 \leq y < 9 \\ 10, & 9 \leq y < 10 \\ 11, & y = 10 \end{cases}$$

$$P[2] = [(0, 0), (2, 3), (4, 6), (6, 9), (9, 10), (10, 11)]$$

0/1 背包问题的DP元组法

(2) $n = 5$, $w = [6, 3, 5, 4, 6]$, $p = [2, 2, 6, 5, 4]$, $c = 10$, 元组法。

- $P(1)=[(0,0),(6,2)]$; $(w_2,p_2)=(3,2)$, $Q2=\{(3,2),(9,4)\}$
- $P(2)=[(0,0),(3,2),(9,4)]$; $(w_3,p_3)=(5,6)$, $Q3=\{(5,6),(8,8)\}$
- $P(3)=[(0,0),(3,2),(5,6),(8,8)]$; $(w_4,p_4)=(4,5)$, $Q4=\{(4,5),(7,7),(9,11)\}$
- $P(4)=[(0,0),(3,2),(4,5),(5,6),(7,7),(8,8),(9,11)]$;
 $(w_5,p_5)=(6,4)$,
- 放第5个物品，得到 $(10,9)$ ；不放，得到 $(9,11)$ 。选择不放物品5。
- 回溯： $Q4$ 中包含 $(9,11)$ ，选择物品4. $(9,11)-(w_4,p_4)=(5,6)$, $Q3$ 中包含 $(5,6)$ ，选择物品3. $(5,6)-(w_3,p_3)=(0,0)$ ，背包没有空间了，所以最优解为 $(0,0,1,1,0)$.

0/1 背包问题的DP元组法

(3) $n = 4$, $c = 20$, $w = (10, 15, 6, 9)$, $p = (2, 5, 8, 1)$, 元组法。

$P(1)=[(0,0),(10,2)];$

$(w_2,p_2)=(15,5), Q2=\{(15,5)\}$

$P(2)=[(0,0),(10,2),(15,5)];$

$(w_3,p_3)=(6,8), Q3=\{(6,8),(16,10)\}$

$P(3)=[(0,0),(6,8),(16,10)];$

放第4个物品，得到(15, 9)；不放，得到(16, 10)。选择不放物品4。

回溯：Q3中包含(16,10), 选择物品3. $(16,10)-(w_3,p_3)=(10,2)$, P1中包含(10,2), 选择物品1. 所以最优解为(1,0,1,0)，最优值为10.

0/1/2背包问题的动态规划法

0/1/2背包问题：具有权重 (w_1, w_2, \dots, w_n) 及效益值 (p_1, p_2, \dots, p_n) 的 n 个物品，放入到容量为 c 的背包中，使得放入背包中的物品效益值最大，

$\max(\sum_{1 \leq i \leq n} x_i p_i)$ ，并满足约束条件 $\sum_{1 \leq i \leq n} x_i w_i \leq c, x_i \in \{0, 1, 2\}, 1 \leq i \leq n$ 。

1. 证明0/1/2背包问题满足最优子结构性质。
2. 定义最优值函数。
3. 给出用动态规划算法求解最优值的递归方程。
4. 用元组法求解该背包问题的动态规划解时，对每个 i (物品从 i 到 n) 需要构造一个元组集合 P_i ，试用形式化的方法描述集合 P_i 及相应的集合 Q 。
5. 给出用元组法求解该问题的一般过程。

多重背包问题的最有子结构性质

命题： 0/1/2背包问题满足最优子结构性质。即对于问题 (w_1, w_2, \dots, w_n) 及 (p_1, p_2, \dots, p_n) 的 n 个物品放入到容量为 c 的背包中的最优解 (x_1, x_2, \dots, x_n) ($x_i \in \{0, 1, 2\}$)， $(x_1, x_2, \dots, x_{n-1})$ 是子问题 $(w_1, w_2, \dots, w_{n-1})$ 及 $(p_1, p_2, \dots, p_{n-1})$ 的 $n-1$ 个物品放入到容量为 $c - x_n w_n$ 的背包中的最优解。

证明： 首先， $\sum_{1 \leq i \leq n-1} x_i w_i \leq c - x_n w_n$ ，即 $(x_1, x_2, \dots, x_{n-1})$ 是子问题的可行解。

反证法，假设0/1/2背包问题不满足最优子结构性质，即 $(x_1, x_2, \dots, x_{n-1})$ 不是子问题的最优解，最优解为 $(y_1, y_2, \dots, y_{n-1})$ ，有 $\sum_{1 \leq i \leq n-1} y_i p_i > \sum_{1 \leq i \leq n-1} x_i p_i$ ， $\sum_{1 \leq i \leq n-1} y_i p_i + x_n w_n > \sum_{1 \leq i \leq n-1} x_i p_i + x_n w_n$ 。并且 $\sum_{1 \leq i \leq n-1} y_i w_i + x_n w_n \leq c$ 。这与假设 (x_1, x_2, \dots, x_n) 是该问题的最优解矛盾。

多重背包问题递归公式

- 多重背包问题可以转化成 0/1 背包问题来求解：因为第 i 件物品最多选 m_i 件，于是可以把第 i 种物品转化为 m_i 件体积和价值相同的物品，然后再来求解这个 0/1 背包问题。
- 设 $f(i, y)$ 为物品从 i 到 n ，背包容量为 y 时最优装包方案对应的效益值。对于第 i 种物品，我们有 k 种选择，即可以选择 $0, 1, 2, \dots, m_i$ 个第 i 种物品。递推表达式为：

$$f(i, y) = \max_k \{ f(i + 1, y - w_i \cdot k) + p_i \cdot k \mid 0 \leq k \leq m_i \text{ 并且 } 0 \leq k \cdot w_i \leq c \}$$

- 边界条件：
$$f(n, y) = \begin{cases} m_n p_n, & y \geq m_n w_n \\ \lfloor y / w_n \rfloor p_n, & w_n \leq y < m_n w_n \\ 0, & y < w_n \end{cases}$$

多重背包问题DP算法

```
for (int i = 0; i < n; i++){ // 考虑第i个物品
    if ( $m_i \cdot w_i \geq c$ ) {
        for (int j =  $w_i$ ; j  $\leq y$ ; j++) {
             $f[j] = \max(f[j], f[j - w_i] + p_i);$  } }
    //  $m_i \cdot w_i < c$ , 需要在  $f[j - w_i \cdot k] + v_i \cdot k$  中找到最大值,  $0 \leq k \leq m_i$ 
    else {
        for (int j =  $w_i$ ; j  $\leq y$ ; j++) {
            int k = 1;
            while ( $k \leq m_i \ \&\& \ j \geq w_i \cdot k$ ) {
                 $f[j] = \max(f[j], f[j - w_i \cdot k] + p_i \cdot k);$ 
                k++; } } }
```

多重背包问题元组法

- 每个 i (物品从 i 到 n) 需要构造一个元组集合 P_i :

$$P(i) = \{(\sum_j k_j w_j, \sum_j k_j p_j) \mid 0 \leq k_j \leq 2, i \leq j \leq n \}$$

相应的集合 $Q(i-1)$ 为

$$Q(i-1) = \{(\sum_j k_j w_j + k_{i-1} w_{i-1}, \sum_j k_j p_j + k_{i-1} p_{i-1}) \mid 0 \leq k_j \leq 2, 1 \leq k_{i-1} \leq 2, i \leq j \leq n \}$$

多重背包问题元组法的一般过程

1. $P(n) = \{(0, 0), (w_n, p_n), (2w_n, 2p_n)\};$

$$Q(n-1) = \{(w_{n-1}, p_{n-1}), (w_n + w_{n-1}, p_n + p_{n-1}), (w_n + 2w_{n-1}, p_n + 2p_{n-1}), \\ (2w_n + 2w_{n-1}, p_n + p_{n-1}), (2w_n + 2w_{n-1}, 2p_n + 2p_{n-1})\};$$

如果有选项超过包的体积，从集合中删除。

2. $P(n-1) = P(n) \cup Q(n-1)$ ，删除并集中的被支配数对和重复数对。

3. $Q(i) = \{(\sum_j k_j w_j + k_i w_i, \sum_j k_j p_j + k_i p_i) \mid 0 \leq k_j \leq 2, 1 \leq k_{i-1} \leq 2, i+1 \leq j \leq n\};$

$$P(i) = P(i+1) \cup Q(i) = \{(\sum_j k_j w_j, \sum_j k_j p_j) \mid 0 \leq k_j \leq 2, i \leq j \leq n\}, \quad 2 \leq i \leq n。$$

4. 考虑 (w_1, p_1) 和 $(2w_1, 2p_1)$ ，根据 $P(2)$ 得到最优值并回溯出最优解。

二维0/1背包问题的动态规划法

二维 0/1 背包问题：给定 n 种物品和一载重量为 C ，容积为 V 的背包。物品 i 的重量是 $w[i]$ ，体积是 $v[i]$ ，价值为 $p[i]$ 。如何选择装入背包中的物品，满足总重量不超过 C 且总容积不超过 V ，使得装入背包中物品的总价值最大？在选择装入背包的物品时，对每种物品 i 只有两种选择，即装入背包或者不装入背包。不能将物品 i 装入背包多次，也不能只装入部分的物品 i 。用动态规划算法求解该问题，要求：

- 1) 证明二维 0/1 背包问题满足最优子结构性质。
- 2) 定义二维 0/1 背包问题的最优值函数。
- 3) 推导上述最优值函数的递归方程。

二维 0/1 背包问题的最优子结构性质

- 最优值函数 $f(i, j, k) = \max\{f(i + 1, j, k), f(i + 1, j - w[i], k - v[i]) + p[i]\}$ 。
- 设 (y_1, y_2, \dots, y_n) 是 $f(1, C, V)$ 的一个最优解。
则 (y_2, \dots, y_n) 是子问题 $f(2, C - w_1 y_1, V - v_1 y_1)$ 的一个最优解。
- 否则, 设 (z_2, \dots, z_n) 是 $f(2, C - w_1 y_1, V - v_1 y_1)$ 的一个最优解,
则 (y_1, z_2, \dots, z_n) 是 $f(1, C, V)$ 的一个更优解。
与假设矛盾。
- 二维0/1背包问题满足最优子结构性质, 其最优解可以由其子问题的最优解来构造。

二维 0/1 背包问题的最优值函数和递归关系式

- 最优值函数 $f(i, j, k)$: 背包容量为 j , 容积为 k , 可选物品为前 i 个物品时二维 0/1 背包问题的最优值。

- 递归关系式:

$$f(i, j, k) = \max\{f(i + 1, j, k), f(i + 1, j - w[i], k - v[i]) + p[i]\}$$

- 按此递归式计算出的 $f(1, c, d)$ 为最优值。

0/1 背包问题的回溯法和分支限界法

对以下0/1背包问题的实例: $n=5$, $c=10$, $w=[2,2,6,5,4]$, $p=[6,3,5,4,6]$

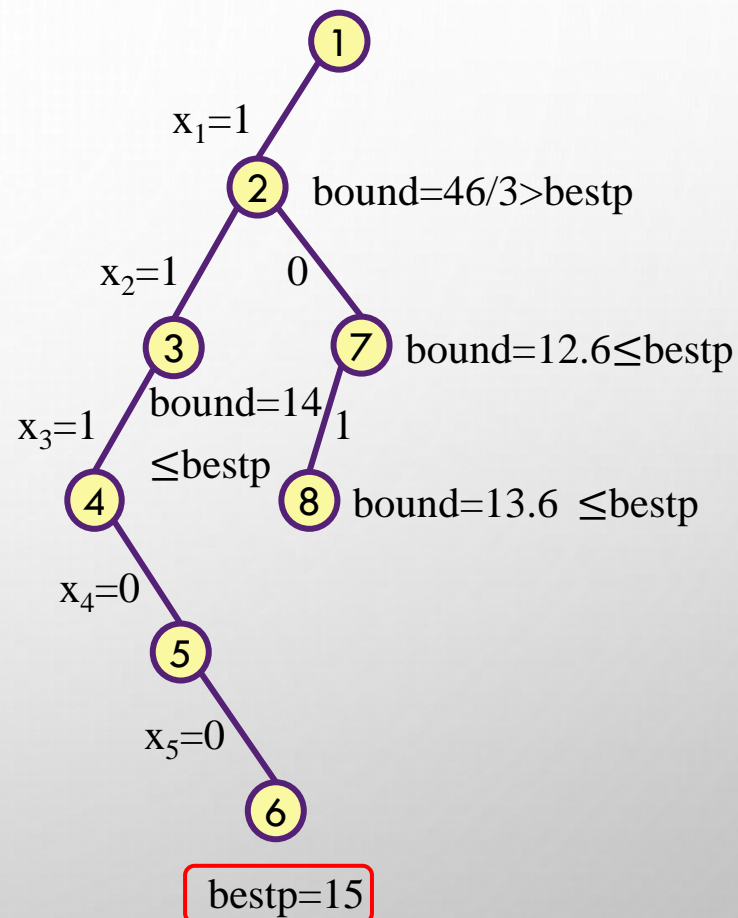
1. 用回溯法求解, 写出限界条件, 并画出展开的状态空间树, 求出优化解和优化值;
2. 用LC-检索的分枝-限界法求解, 写出限界条件, 画出以“下标增量”表示展开的状态空间树, 求出优化解和优化值。

0/1 背包问题的回溯法状态空间树

$n=5$, $c=10$, $w=[2,2,6,5,4]$, $p=[6,3,5,4,6]$ 。

按密度排序, $w'=[2,2,4,6,5]$, $p'=[6,3,6,5,4]$

- $\text{bound}(x) = \text{cp} + r$;
- $\text{cp} = x_1p_1 + \dots + x_{k-1}p_{k-1}$;
- $r =$ 剩余待选物品的连续背包问题的贪心值。
- **左枝限界**: $\text{cw} + w_k > c$
- **右枝限界**: $\text{bound} \leq \text{bestp}$
- 最优解为 $x' = [1,1,1,0,0]$; 即 $x = [1,1,0,0,1]$ 。
- 对应的最优值为15。

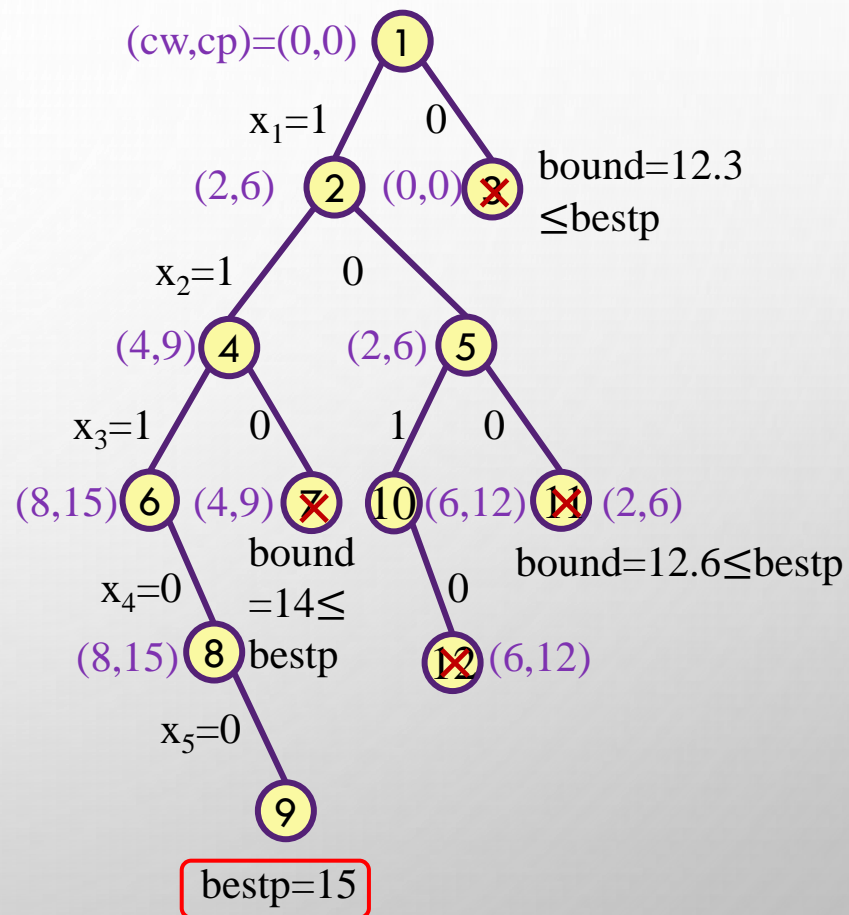


0/1 背包问题的LC分支限界法状态空间树

$n=5$, $c=10$, $w=[2,2,6,5,4]$, $p=[6,3,5,4,6]$ 。

按密度排序, $w'=[2,2,4,6,5]$, $p'=[6,3,6,5,4]$

- $\text{bound}(x) = \text{cp} + r$;
- $\text{cp} = x_1 p_1 + \dots + x_{k-1} p_{k-1}$;
- r = 剩余待选物品的连续背包问题的贪心值。
- **左枝限界**: $\text{cw} + w_k > c$
- **右枝限界**: $\text{bound} \leq \text{bestp}$
- 最优解为 $x' = [1,1,1,0,0]$; 即 $x = [1,1,0,0,1]$ 。
- 对应的最优值为15。



连续背包问题的贪心法

- 考虑 $0 \leq x_i \leq 1$ 而不是 $x_i \in \{0,1\}$ 的连续背包问题。一种可行的贪心策略是：按价值密度非递减的顺序检查物品，若剩余容量能容下正在考察的物品，将其装入；否则，往背包中装入此物品的一部分。
 1. 对于 $n = 3$, $w = [100, 10, 10]$, $p = [20, 15, 15]$ 及 $c = 105$, 上述装入方法获得的结果是什么？
 2. 写出伪代码。
 3. 证明这种贪心算法总能获得最优解。

连续背包问题求解和伪代码

- $n=3$, $w=[100,10,10]$,
 $p=[20,15,15]$, $c=105$ 。
- 求密度并排序: $w=[10,10,100]$,
 $p=[15, 15, 20]$
- $i=1$, $p=15$, $c=95$;
- $i=2$, $p=30$, $c=85$;
- $i=3$, $p=30+20\cdot(85/100)=47$ 。
- 最优解为 $(0.85, 1, 1)$, 最优值为47。

密度贪心法的伪代码:

1. 将物品按密度从大到小排序
2. $p=0$
3. for ($i=0$; $i<n$; $i++$)
4. if (物品 i 可装入到背包内)
5. $p=p+p_i$; $c=c-w_i$;
6. else $p=p+p_i \cdot (c/w_i)$

反证法: 假设得到的解不是最优解, 说明同样的空间下, 还有密度更大的物品没有装入。这与算法矛盾。所以本贪心算法总能获得最优解。

连续背包问题最优性证明

证明: 假设物品已按价值密度非递减的顺序排列, $x=(x_1, \dots, x_n)$ 为贪心法产生的解, 则它有形式 $(1, 1, \dots, x_j, 0 \dots 0)$, 其中 $0 < x_j < 1$; 设 $y=(y_1, \dots, y_n)$ 是优化解;

设 k 是 $x_i \neq y_i$ 的最小下标, 则 $k \leq j$ 且 $y_k = 0$ ($x_k = 1$)。如果这样的 k 不存在, 则两组解是同样的, 因此贪心法得到的解是最优的。

令 $y_k = x_k$, 并在 $\sum_{k < i \leq n} y_i w_i$ 中减去 $(x_k - y_k)w_k$ (无论什么方式) 得到 $(z_k, z_{k+1}, \dots, z_n)$, 其中 $z_k = x_k$ 。

下面证明 $(y_1 \dots y_{k-1}, z_k, z_{k+1}, \dots, z_n)$ 仍是优化解:

证明 $\sum_{k \leq i \leq n} y_i p_i \leq \sum_{k \leq i \leq n} z_i p_i$ 。

将 $(y_i - z_i)p_i$ 改写成 $(y_i - z_i)w_i p_i / w_i$. 利用 $p_i / w_i \leq p_k / w_k$ ($i > k$) 和 $\sum_{k < i \leq n} (y_i - z_i)w_i = (x_k - y_k)w_k$ 可得到上述不等式。

贪心算法

本课程涉及到的算法

- 货箱装船问题
- 活动安排问题
- 最短路径问题(Dijkstra算法、Bellman-Ford算法)
- 最小生成树问题(Prim算法、Kruskal算法)
- 哈夫曼编码问题
- 拓扑排序问题
- 偶图覆盖问题
- 等等

作业调度问题

假定有 n 个作业，在一台处理机上执行。每个作业 i 对应一个 3 元组 (p_i, d_i, t_i) ，其中 t_i 为该作业要求的处理时间， d_i 为截止期， p_i 为该作业未在截止期之前完成引起的罚款。求 n 个作业的一个子集 J ， J 中的作业都能在截止期之前完成，且使不在 J 中的作业罚款总额最小。试用最大罚款优先的贪心策略设计一个启发式算法：

1. 写出算法伪代码；
2. 举例说明上述贪心策略能否得到最优解。

作业调度问题贪心算法伪代码

1. 将物品按罚款从大到小排序。
2. for ($i=1; i < n; i++$)
3. if (作业 i 可以在截止期前执行)
4. $x_i = 1$ (装入)
5. else $x_i = 0$ (舍弃);

➤ 该贪心得不了不一定能得到最优解。

➤ 反例：三个作业对应的三元组 (p_i, d_i, t_i) 分别是 $(6, 4, 3)$ 、 $(5, 4, 2)$ 、 $(4, 4, 2)$ 。
贪心解为 $(1, 0, 0)$ ，罚款额为 9；最优解为 $(0, 1, 1)$ ，罚款额为 6。

最小平均完成时间的任务调度问题

已知 n 个任务的执行序列。假设任务 i 需要 t_i 个时间单位。若任务完成的顺序为 $1, 2, \dots, n$, 则任务 i 完成的时间为 $c_i = \sum_{j=1}^i t_j$ 。任务的平均完成时间(Average Completion Time, ACT)为 $(\sum_{i=1}^n c_i)/n$ 。

- 1) 考虑有四个任务的情况, 每个任务所需时间分别是 $(4, 2, 8, 1)$, 若任务的顺序为 $1, 2, 3, 4$ 的ACT是多少? 若任务顺序为 $2, 1, 4, 3$ 的ACT是多少? 利用最小ACT的任务优先的贪心策略获得的任务顺序为 $4, 2, 1, 3$, 这种顺序的ACT是多少?
- 2) 写一个最小ACT任务优先的贪心算法伪代码, 并证明程序的复杂性为 $O(n \log n)$ 。
- 3) 证明利用3)中的贪心算法获得的任务顺序具有最小的ACT。

最小ACT的任务调度问题的最优性

证明： 设在某任务顺序中， $i > j$ 而 $t_i < t_j$ ，则交换作业 i 、 j 的顺序得到一个新的执行顺序。设原顺序的平均完成时间为 ACT ，改变后的平均完成时间为 ACT' 。由 $ACT = (c_1 + c_2 + \dots + c_n)/n$ ， $c_i = t_1 + t_2 + \dots + t_i$ ，有

$$nACT = (nt_1 + \dots + (n-j+1)t_j + \dots + (n-i+1)t_i + \dots)$$

$$nACT' = (nt_1 + \dots + (n-j+1)t_i + \dots + (n-i+1)t_j + \dots)$$

$$nACT - nACT' = (i-j)t_j - (i-j)t_i = (i-j)(t_j - t_i) > 0$$

即 $ACT' < ACT$ 。说明交换后所得的平均完成时间 ACT' 更小。

同理，每消除一个逆序 ACT 值减小。直到无逆序时，即按最小执行时间优先得到的任务顺序执行任务时，平均完成时间 ACT 值达到最小。

货箱装载问题

设有 n 个集装箱,集装箱大小一样, 第 i 个集装箱的重量为 $w_i (1 \leq i \leq n)$, 设船的载重量为 c .试设计一种装船的方法使得装入的集装箱数目最多.

数学描述: 给定 $c > 0$, $w_i > 0 (1 \leq i \leq n)$, 要求找出一个 n 元0-1向量 $(x_1, x_2, \dots, x_n) (1 \leq i \leq n)$, 使得 $\sum_{i=1}^n w_i x_i \leq c$, 且 $\sum_{i=1}^n x_i$ 达到最大.

整数规划问题:

$$\begin{cases} \max \sum_{i=1}^n x_i \\ \text{s.t.} \sum_{i=1}^n w_i x_i \leq c \end{cases}, \text{ 其中 } x_i = \begin{cases} 0 & \text{如果货厢} i \text{不装船} \\ 1 & \text{如果货箱} i \text{装船} \end{cases}$$

货箱装载问题的贪心属性

贪心选择性质：设集装箱已依其重量从小到大排序, $x=(x_1, x_2, \dots, x_n)$ 是装载问题的贪心解, $y=(y_1, \dots, y_n)$ 是任意一个可行解。则 $\sum_{i=1}^n x_i \geq \sum_{i=1}^n y_i$ 。

证明：由算法知, 存在 $k=\max\{i|x_i=1\} \{ 1 \leq i \leq n\}$, 使得 $x_i=1$ 当 $i \leq k$, 且 $x_i=0$ 当 $i > k$ 。即 $x=(x_1, x_2, \dots, x_n)=(1, \dots, 1, 0, \dots, 0)$ 。由算法知,

$$\sum_{i=1}^k w_i x_i \leq c, \quad \left(\sum_{i=1}^k w_i x_i\right) + w_{k+1} > c。$$

设 $p: 1 \leq p \leq k$ 是满足 $x_i \neq y_i$ 的最小的下标, 则 $y_p = 0$ 。如果存在一个 q ($k < q \leq n$) 使得 $y_q = 1$ 。则令 $y_p = 1, y_q = 0$ ($w_p \leq w_q$), 产生一个新的可行解 z , 满足 $\sum_{i=1}^n y_i = \sum_{i=1}^n z_i$ 。如果不存在这样的 q , 则命题成立。继续上述操作, 直到找不到这样的 q 为止。

$$\sum_{i=1}^n y_i = \sum_{i=1}^n z_i \leq \sum_{i=1}^n x_i, \quad \text{命题成立。}$$

最短路径问题DIJKSTRA算法

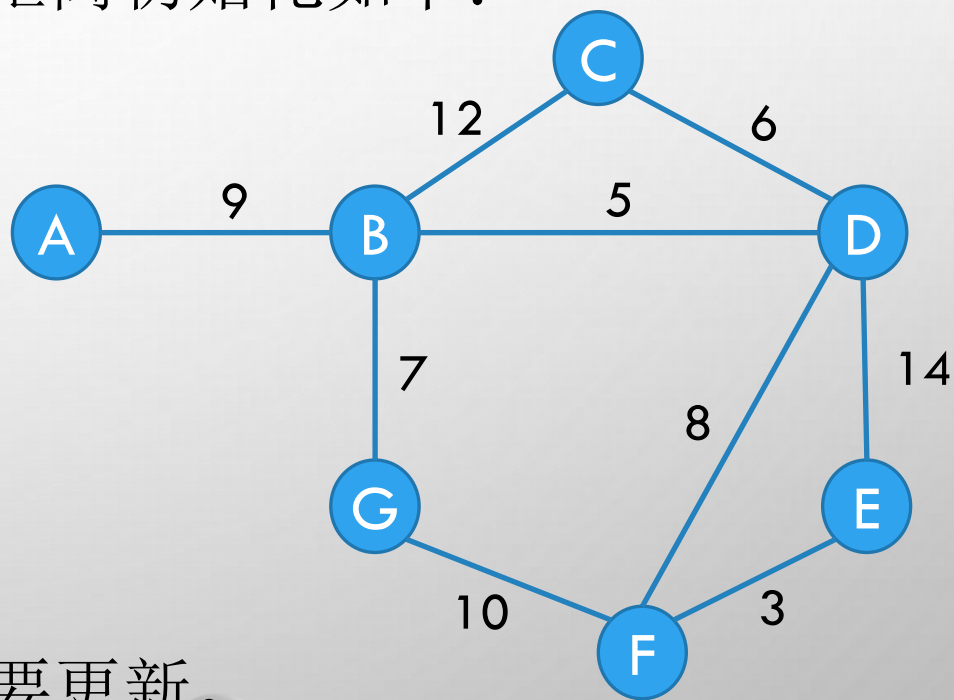
用DIJKSTRA算法计算从节点A到所有其他节点的最短路径。

1. S初始化为 \emptyset ，所有节点到源点A的距离初始化如下：

节点	A	B	C	D	E	F	G
离源点距离	0	∞	∞	∞	∞	∞	∞

2. 在所有不属于S的节点里面选取离源点距离最短的节点，即源点本身，将源点A加入到S，此时 $S=\{A\}$ 。

每个在V-S里的节点到源点的距离也需要更新。



1. $S=\emptyset$,

节点	A	B	C	D	E	F	G
离源点距离	0	∞	∞	∞	∞	∞	∞

2. $S=\{A\}$, 更新各节点到源点的距离。

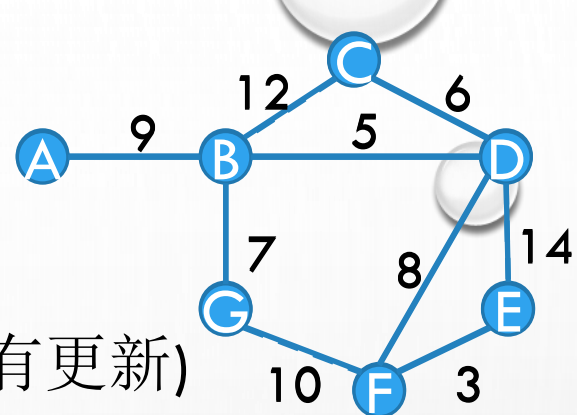
节点	A	B	C	D	E	F	G
离源点距离	0	9	∞	∞	∞	∞	∞

3. $S=\{A, B\}$,

节点	A	B	C	D	E	F	G
离源点距离	0	9	21	14	∞	∞	16

4. $S=\{A, B, D\}$,

节点	A	B	C	D	E	F	G
离源点距离	0	9	20	14	28	22	16



5. $S=\{A, B, D, G\}$ (没有更新)

节点	A	B	C	D	E	F	G
离源点距离	0	9	20	14	28	22	16

6. $S=\{A, B, D, G, C\}$ (没有更新)

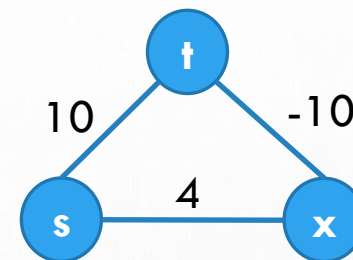
节点	A	B	C	D	E	F	G
离源点距离	0	9	20	14	28	22	16

7. $S=\{A, B, D, G, C, F\}$

节点	A	B	C	D	E	F	G
离源点距离	0	9	20	14	25	22	16

8. $S=\{A, B, D, G, C, F, E\}$, 算法结束。

最短路径问题：遍历点 → 遍历边



- DIJKSTRA 算法对节点划分, 遍历节点。
- 对有负权重边的图, 节点到源点的最短路径不能确定, 不能采用对节点进行划分的方法。
- **换个想法：** 把针对节点进行降距改为对每条边进行降距。
- 依据: 每次降距都是针对每一条边进行, 因为降距的核心就是从一条新的边获得更短的路径。
- 结束准则: 当一轮降距结束时(即每条边都考察一次后), 如果没有任何距离被降低, 则说明最短路径已全部找出; 如果有距离降低, 则需要再来一轮降距。
- 最多需要 V 轮降距操作。

• 最短路径问题Bellman-Ford算法

- Bellman-Ford-Shortest-Paths (g, s)

1. $d[s] \leftarrow 0$ $O(1)$

2. for $v \in V - \{s\}$ do

3. $d[v] \leftarrow \infty$ $O(1)$

4. for $i \leftarrow 1$ to $V - 1$ do $\left. \begin{matrix} V \text{次} \\ E \text{次} \end{matrix} \right\} O(VE)$

5. for edge $(u, v) \in E$ do

6. if $d[v] > d[u] + w(u, v)$ then

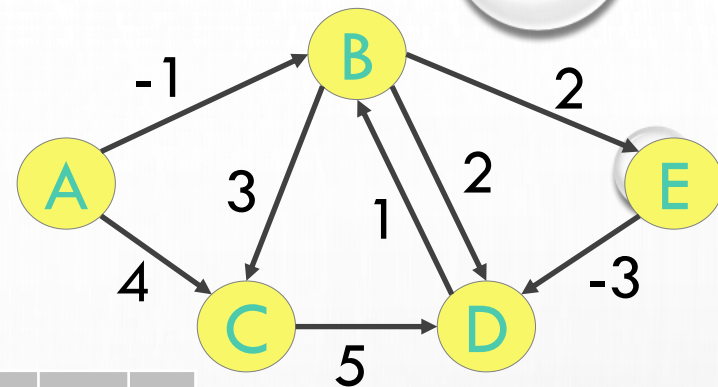
7. $d[v] \leftarrow d[u] + w(u, v)$

//对所有的边进行 $V-1$ 次降距操作

//对每条边进行考察，顺序任意

//降距操作

最短路径问题Bellman-Ford算法



	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
	0	∞	∞	∞	∞	0	-1	2	∞	∞	0	-1	2	-2	1	0	-1	2	-2	1
BE(2)					∞					1					1					1
DB(1)		∞					-1					-1					-1			
BD(2)				∞					1					-2					-2	
AB(-1)		-1					-1					-1					-1			
AC(4)			4					2					2					2		
DC(5)			4					2					2					2		
BC(3)			2					2					2					2		
ED(-3)				∞					-2					-2					-2	
	0	-1	2	∞	∞	0	-1	2	-2	1	0	-1	2	-2	1	0	-1	2	-2	1

- ✓ $V=5$ 个节点,
- ✓ $V-1=4$ 。最多需要做4次循环。
- ✓ 对8条边遍历。
- ✓ 如果
 $d[v] > d[u] + w(u,v)$
 那么
 $d[v] = d[u] + w(u,v)$

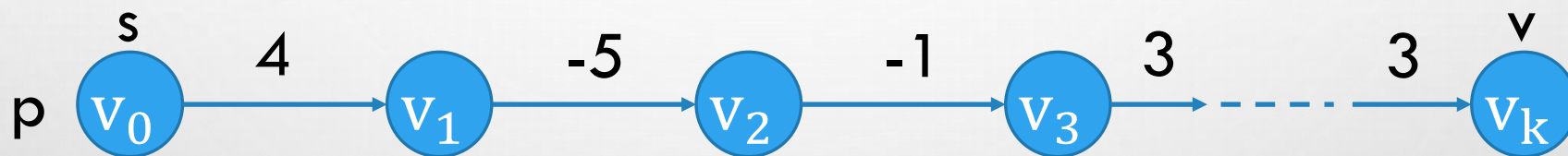
最短路径问题2: Bellman-Ford算法的正确性

算法得到的
路径

真正的
最短路径

➤ **定理** 如果图 $G=(V, E)$ 不包括负循环, 则在Bellman-Ford算法终止后,
 $\forall v \in v, d[v] = \delta(s, v)$.

➤ **证明** 设 $v \in V$ 为任意节点, 考虑从源点 s 到任意节点 v 的最短路径 p :



- 由于 p 是最短路径, 我们有 $\delta(s, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i)$.
- 在算法开始时, 第1行的初始化将把源点 s 的距离设置为0, 即 $d[v_0] = 0 = \delta(s, v_0)$, 并且 $d[v_0]$ 在算法的执行过程中保持不变。

最短路径问题2: Bellman-Ford算法的正确性

- 在第1轮降距循环中, 由于所有的边都考虑了一遍, 因此, 边 (s, v_1) 一定会被考虑到, 从而导致 v_1 的最短距离被更新为真正最短路径长度, 即 $d[v_1] = \delta(s, v_1)$.
- 在第2轮降距循环中, 由于所有的边又轮转考虑了一遍, 因此, 边 (v_1, v_2) 一定会被考虑到, 从而导致 v_2 的最短距离被更新为真正最短路径长度 $d[v_2] = \delta(s, v_1) + w(v_1, v_2)$, 即 $d[v_2] = \delta(s, v_2)$.
-
- 在第 k 轮降距循环中, 由于所有的边又轮转考虑了一遍, 因此, 边 (v_{k-1}, v_k) 一定会被考虑到, 从而导致 v_k 的最短距离被更新为真正最短路径长度 $d[v_k] = \delta(s, v_{k-1}) + w(v_{k-1}, v_k)$, 即
$$d[v_k] = \delta(s, v_k) = \delta(s, v).$$

最短路径问题2: Bellman-Ford算法的正确性

算法得到的
路径

真正的
最短路径

➤ **定理** 如果图 $G=(V, E)$ 不包括负循环，则在Bellman-Ford算法终止后， $\forall v \in V, d[v] = \delta(s, v)$.

由于图 G 不包括负循环，路径 p 必定是一条简单路径(不包括循环的路径)，如果不是简单路径，我们可以将路径上的循环去掉，结果仍然是一条最短路径。因此路径 p 最长只有 $v-1$ 条边，而算法循环的遍数是 $v-1$ 遍。因此，算法结束后，最短路径 p 将被发现，即对于任意节点 v ，Bellman-Ford算法确实能将源点到节点 v 的最短路径求出。

最短路径问题2：负循环检查

- **Bellman-Ford 算法虽然能应对负权重，但却不能求解有负循环的所有最短路径。**
- 如果有负循环存在，节点之间可能不存在最短路径。
- 我们可以侦测一个图是否存在负循环。
- 观察：如果没有负环路，最短路径包含的边的条数最多只有 $v-1$ 条，因此，在 $v-1$ 轮降距操作后，所有最短路径都已经找出。换言之，如果存在负环路，则某些“最短路径”包含的边的条数将超过 $v-1$ 条(实际上为无穷)，因此，在 $v-1$ 轮降距操作后，仍然存在 $d[v] > d[u] + w(u, v)$ 的情况。

最短路径问题2：负循环检查

➤推论 如果任意 $d[v]$ 经过 $v-1$ 次循环后还是不收敛(即仍可降低), 则该图存在一个负循环。

➤根据推论, 对Bellman-Ford算法进行修改:

Bellman-Ford-Shortest-Paths-ii (g, s)

1. $d[s] \leftarrow 0$
2. for $v \in V - \{s\}$ do $d[v] \leftarrow \infty$
3. for $i \leftarrow 1$ to $V - 1$ do
4. for edge $(u, v) \in E$ do
5. if $d[v] > d[u] + w(u, v)$ then $d[v] \leftarrow d[u] + w(u, v)$
6. for edge $(u, v) \in E$ do
7. if $d[v] > d[u] + w(u, v)$ then 报告负环路存在

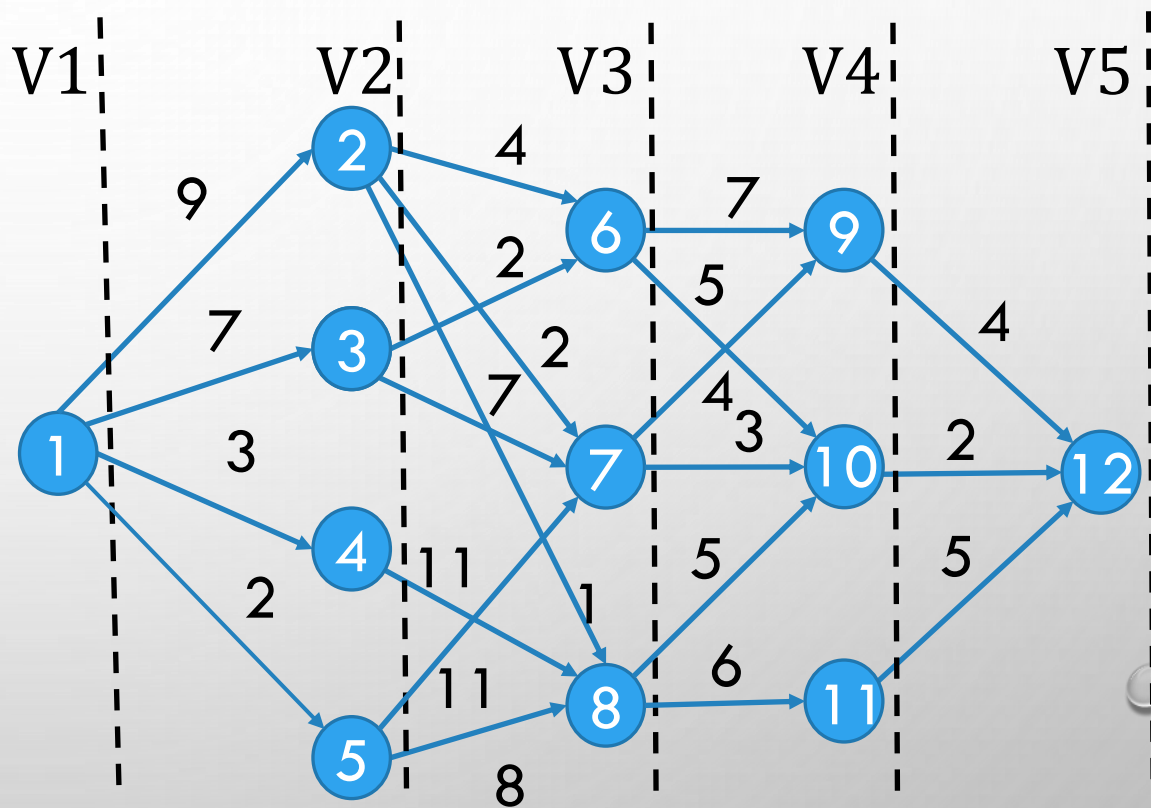
最短路径问题3：多段图

- 设 $d(i)$ 表示为源点1到节点 i 的最短路长度, $w(i,j)$ 为边 (i,j) 的权重.
 1. 试写出 $d(i)$ 满足的动态规划递归关系式;
 2. 如右图, 求解, 并回溯求出优化的路径。

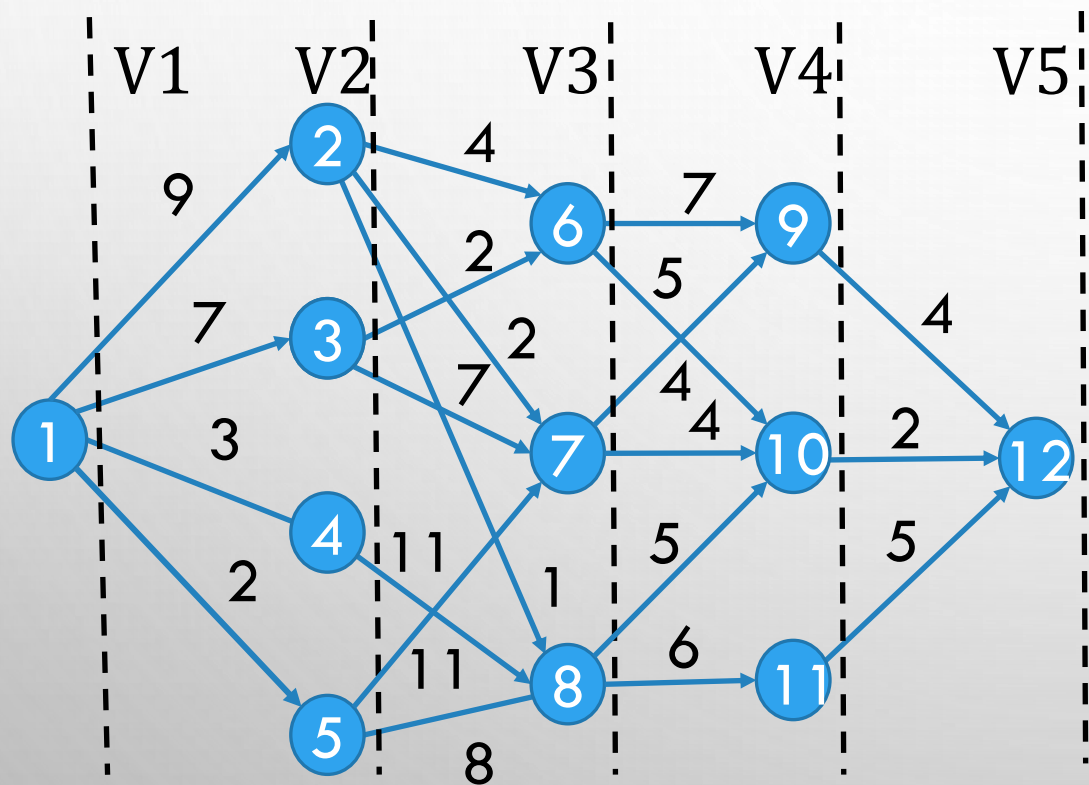
解: 设 $B(i)$ 为 i 的前驱节点集合, 有

$$d(1)=0;$$

$$d(i)=\min_{j \in B(i)} \{d(j)+w(j,i)\}$$



最短路径问题3：多段图



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

3	2	Λ
3	7	
4	11	Λ
5	8	Λ
7	4	Λ
7	4	
8	5	Λ
10	2	
11	5	Λ

最短路径问题3：多段图

● $v_1: d(1)=0$

$v_2: d(2)=w(1,2)=9; d(3)=w(1,3)=7$

$d(4)=w(1,4)=3; d(5)=\underline{w(1,5)=2}$

$v_3: d(6)=\min\{\underline{9+w(2,6)}, 7+w(3,6)\}=9$

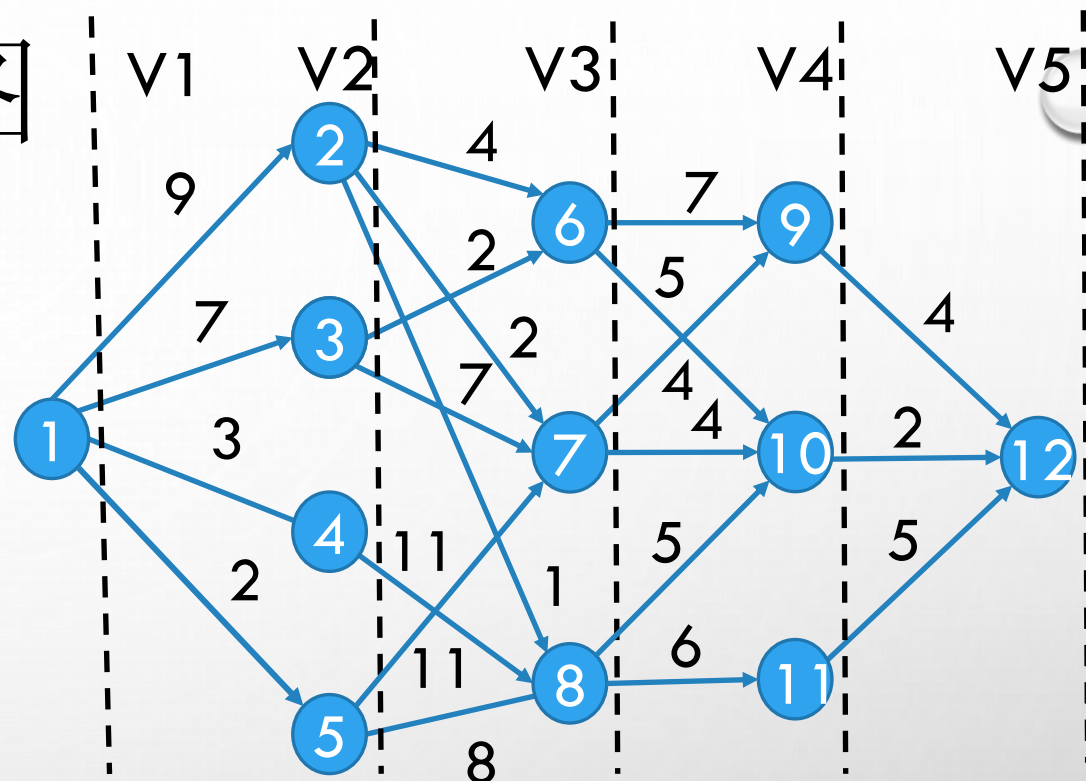
$d(7)=\min(9+w(2,7), 7+w(3,7),$
 $\underline{2+w(5,7)})=11$

$d(8)=\min\{9+w(2,8), 3+w(4,8),$
 $\underline{2+w(5,8)}\}=10$

$v_4: d(9)=\min\{9+w(6,9), \underline{11+w(7,9)}\}=15$

$d(10)=\min\{9+w(6,10), 11+w(7,10),$
 $10+w(8,10)\}=14$

$d(11)=\underline{10+w(8,11)}=16$



$v_5: d(12)=\min\{15+w(9,12),$

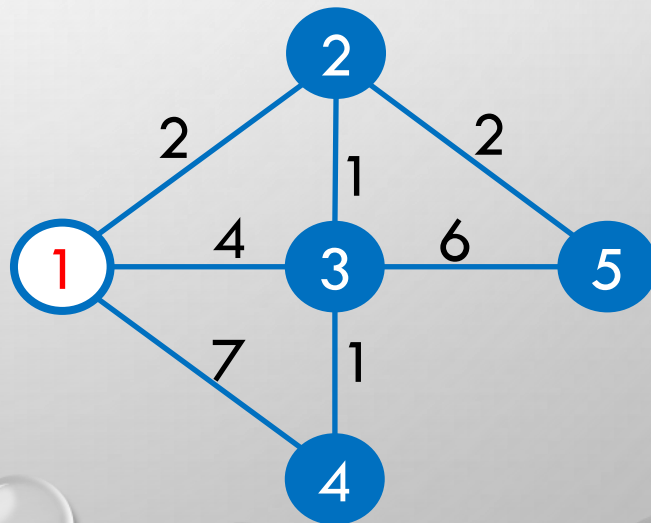
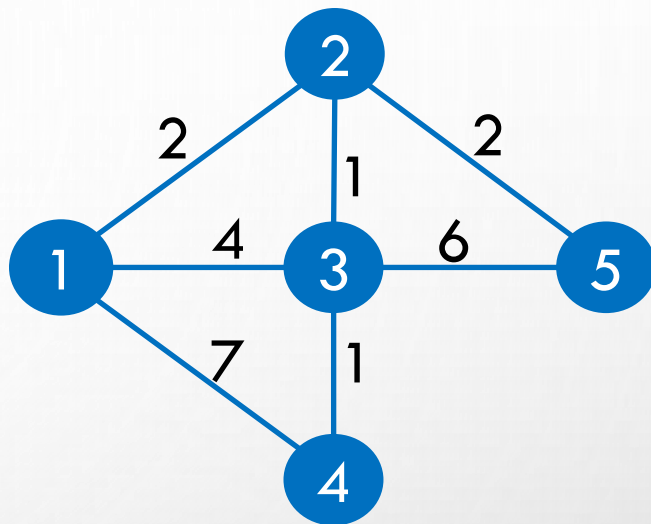
$\underline{14+w(10,12)}, 16+w(11,12)\}=16$

最优值为16;回溯最优解1, 3, 6, 10, 12.

最小生成树1： Prim算法

➤ 初始时 $S=\emptyset$, 权值之和 $MST=0$. 所有的点都是蓝点, $\min[1]=0$, $\min[2,3,4,5]=\infty$, $\min[1]=0$ 最小. 找到蓝点1. 将1变为白点。

节点	1	2	3	4	5
最小边权	0	∞	∞	∞	∞



最小生成树1： Prim算法

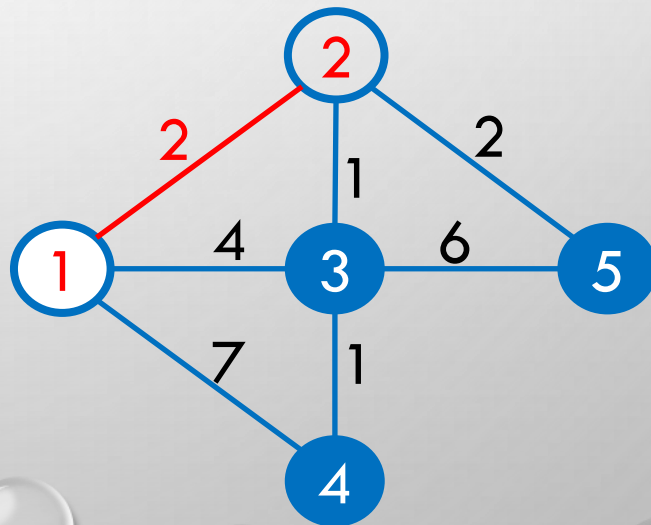
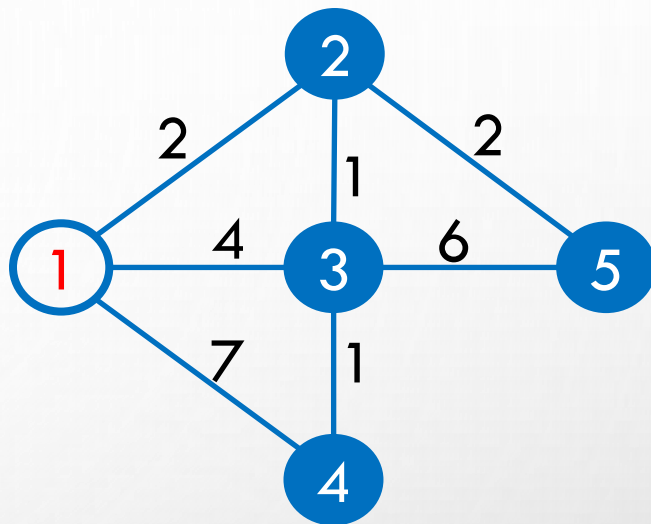
节点	1	2	3	4	5
最小边权	0	∞	∞	∞	∞

- 接着枚举与1相连的所有蓝点并修改它们与白点相连的最小边权。

$\min[2]=w(1,2)=2$; $\min[3]=w(1,3)=4$;
 $\min[4]=w(1,4)=7$ 。

节点	1	2	3	4	5
最小边权	0	2	4	7	∞

- $\min[2]$ 最小, 将2变为白点, 并标记(1,2)边。



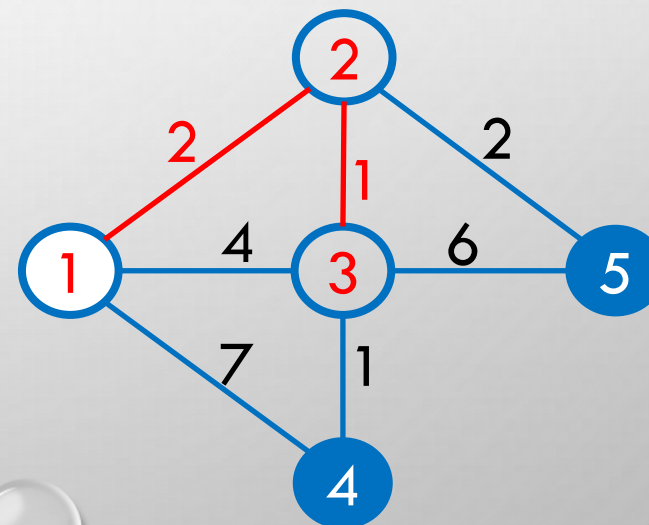
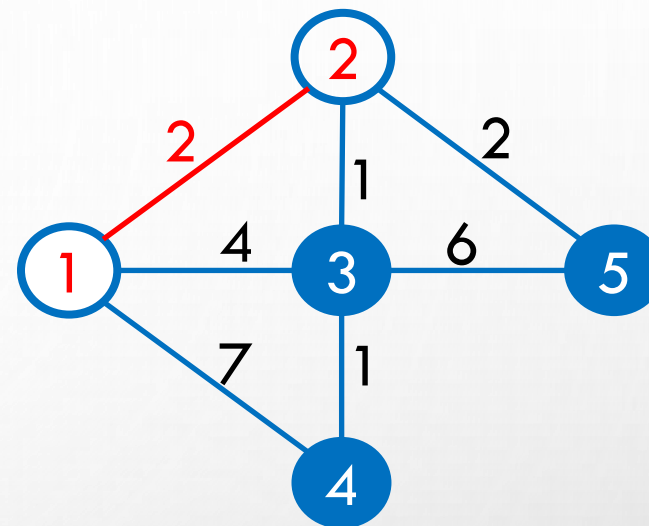
最小生成树1：Prim算法

节点	1	2	3	4	5
最小边权	0	2	4	7	∞

➤接着枚举与2相连的所有蓝点并修改它们与白点相连的最小边权.

$\min[3]=w(2,3)=1$; $\min[5]=w(2,5)=2$. $\min[3]$ 最小. 将3变为白点, 并标记(2,3)边。

节点	1	2	3	4	5
最小边权	0	2	1	7	2



最小生成树1： Prim算法

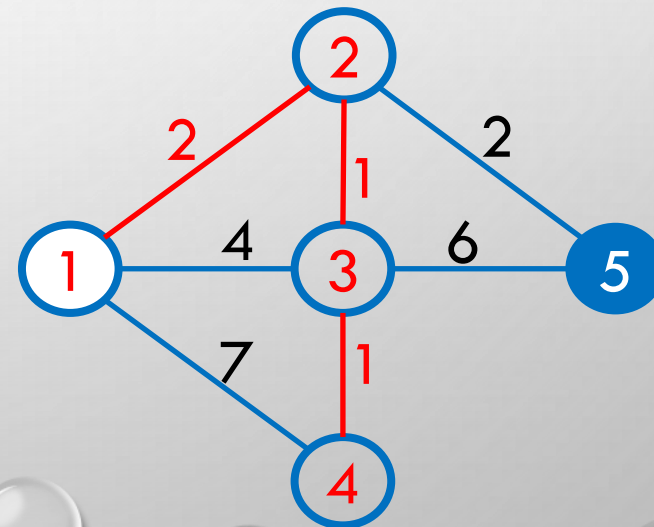
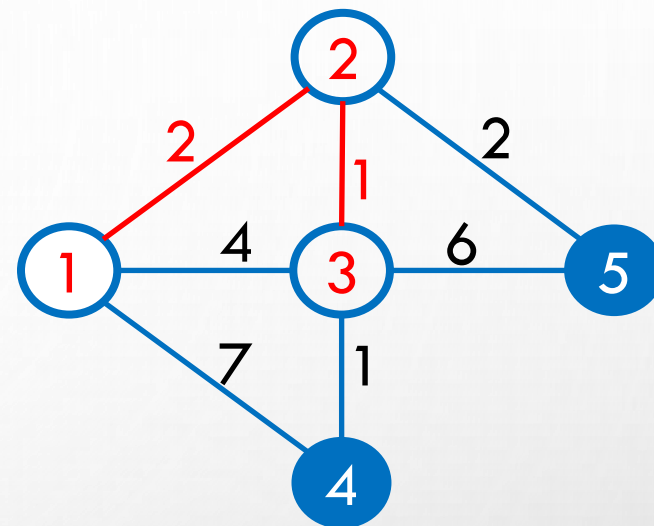
节点	1	2	3	4	5
最小边权	0	2	1	7	2

➤接着枚举与3相连的所有蓝点并修改它们与白点相连的最小边权.

$\min[4]=w(3,4)=1$;

$\min[5]=w(3,5)=6 > 2$, 所以不修改5的边权. $\min[4]$ 最小. 将4变为白点, 并标记(3,4)边。

节点	1	2	3	4	5
最小边权	0	2	1	1	2

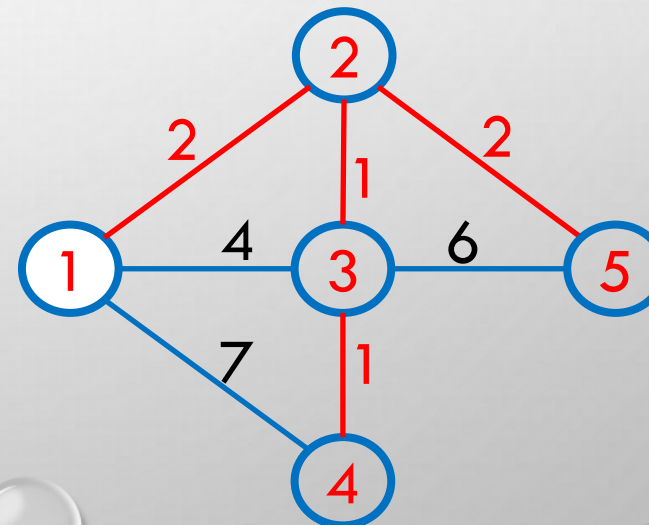
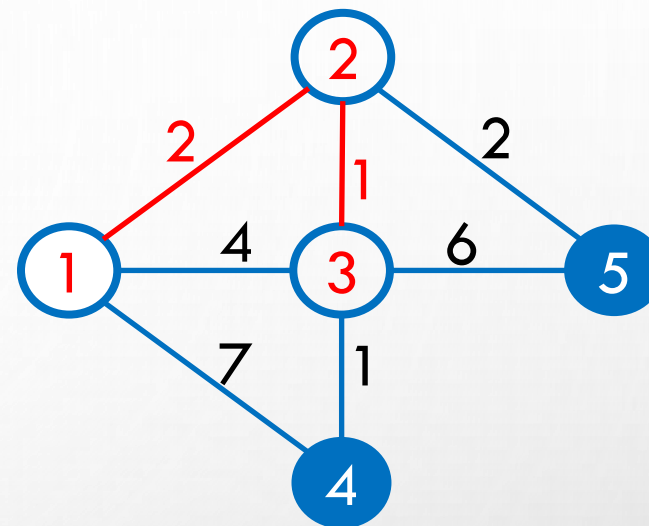


最小生成树1： Prim算法

节点	1	2	3	4	5
最小边权	0	2	1	1	2

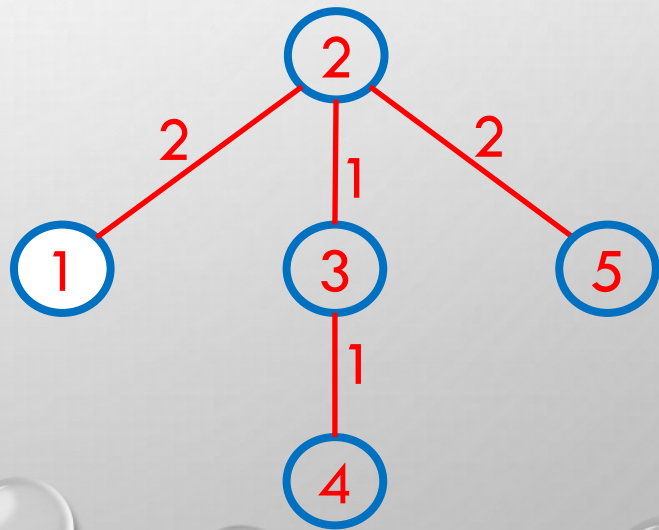
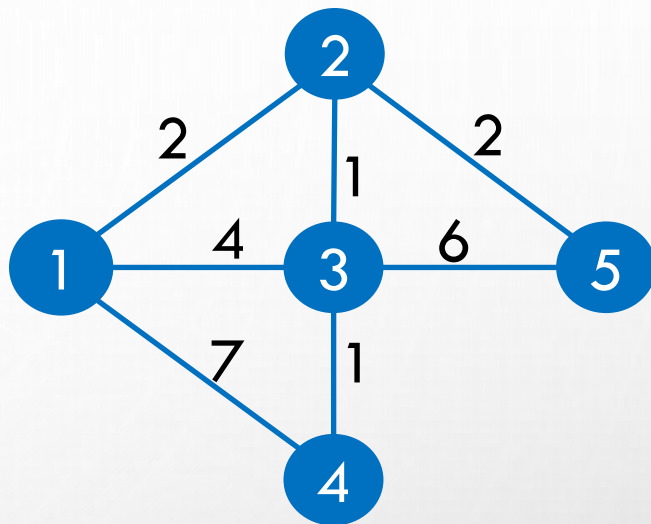
➤接着枚举与4相连的所有蓝点并修改它们与白点相连的最小边权.4没有相连的蓝点,不需修改边权.此时,唯一剩余的蓝点5,边权为 $w(2,5)=2$,将5变为白点,并标记(2,5)边。

节点	1	2	3	4	5
最小边权	0	2	1	1	2



最小生成树1： Prim算法

- 右下图为Prim算法生成的最小生成树。
- n 个顶点的图的最小生成树有 $n-1$ 条边。
- 权值之和 $MST=6$ 。
- 从所有不在 S 中的点中，选一个距离 u 最近的点，时间复杂度为 $O(n)$ ；选取点后，更新相关边权重，时间复杂度为 $O(n)$ 。要对 $n-1$ 个结点进行以上操作，所以总时间复杂度为 $O(n^2)$ 。
- 使用不同的数据结构，Prim算法有和Dijkstra算法相同的表现。



最小生成树1: Prim算法

• MST-Prim(G, w, r)

1. for each $u \in G.V$

2. $u:key = \infty$

//将根结点以外的结点的key值设置为 ∞

3. $u:\pi = \text{nil}$

//每个结点的父结点设置为nil

4. $s:key = 0$

//根结点的key值设置为0

5. $Q = G.V$

//对最小优先队列q进行初始化

6. while $Q \neq \phi$

7. $u = \text{extract_min}(Q)$

//找出连接v-q和q的轻边的一个端点u, 接着

8. for each $v \in G.\text{adj}[u]$

9. if $v \in Q$ and $w(u, v) < v.key$

} //将每个与u邻接但却不在树中的结点更新,
 将u从Q中删除, 并加入V-Q中, 即 $(v, v.\pi) \rightarrow A$

10. $v.\pi = u$

11. $v.key = w(u, v)$

//decrease-key操作

最小生成树1: Prim算法

• MST-Prim(G, w, r)

1. for each $u \in G.V$
2. $u.key = \infty$
3. $u.\pi = \text{nil}$
4. $s.key = 0$
5. $Q = G.V$
6. while $Q \neq \phi$
7. $u = \text{EXTRACT_MIN}(Q)$
8. for each $v \in G.\text{adj}[u]$
9. if $v \in Q$ and $w(u, v) < v.key$
10. $v.\pi = u$
11. $v.key = w(u, v)$ $O(\lg V)$

$O(V)$

$O(V \lg V)$

总执行次数为 $O(E)$

➤ 在while循环的每遍循环之前，我们有：

1. $A = \{(v, v.\pi) \mid v \in V - \{r\} - Q\}$
2. 已经加入到最小生成树的结点为集合 $V - Q$
3. 对于所有的结点 $v \in Q$ ，如果 $v.\pi \neq \text{NIL}$ ，则 $v.key < \infty$ 并且 $v.key$ 是连接结点 v 和最小生成树中某个结点的轻边 $(v, v.\pi)$ 的权重。

➤ $T = O(V) \cdot T_{\text{EXTRACT-MIN}} + O(E) \cdot T_{\text{DECREASE-KEY}}$
 $= O(V \lg V + E \lg V)$
 $= O(E \lg V)$ (二叉堆)

最小生成树1: Prim算法的时间复杂度

- $T = O(V) \cdot T_{\text{EXTRACT-MIN}} + O(E) \cdot T_{\text{DECREASE-KEY}}$

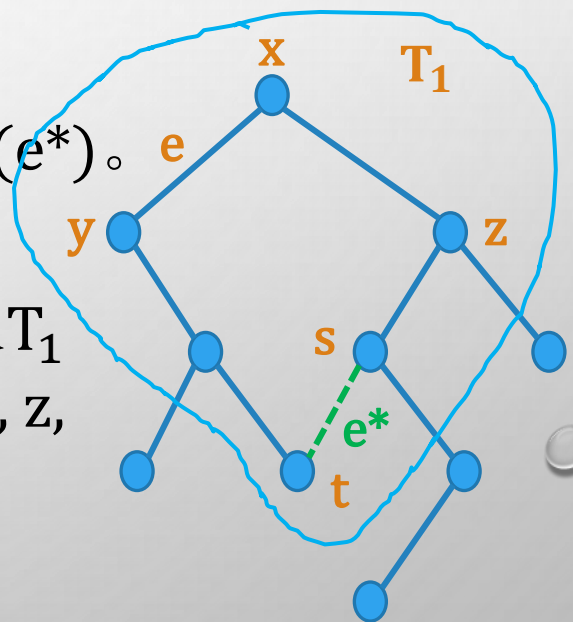
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$	$O(1)$	$O(E + V \lg V)$
	amortized	amortized	worst case

最小生成树1: Prim算法的正确性

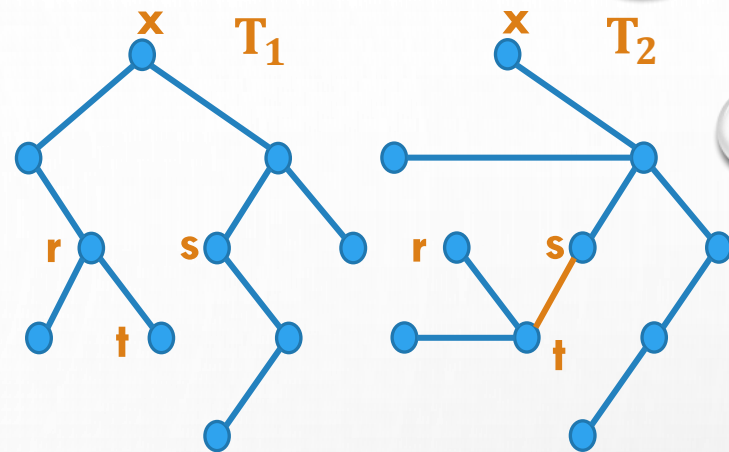
引理 将算法生成的MST记为 T_1 , e^* 是图 G 中任意一条不属于 T_1 的边。将 e^* 添入 T_1 后形成回路, 则 e^* 比回路中任意一条属于 T_1 的边都长。

证明 反证法。

- 假设 T_1 中存在至少一条边 e' , s.t. $w(e') > w(e^*)$ 。
- 令 $e = xy \in T_1$ 为回路中最长的边, 有 $w(e) \geq w(e') > w(e^*)$ 。
- 不妨设 x 先于 y 被添加到 T_1 中。
- 由于 e 是回路中最长的边, 根据Prim算法, 在 x 被加入 T_1 后, 会优先选择回路中 e 以外的边加入 T_1 。那么路径 x, z, s, t, \dots, y 会被选进 T_1 。而边 e 不会被选入 T_1 。
- 这与 e 为 T_1 中的边矛盾, 则假设错误, 原命题成立。



最小生成树1：Prim算法的正确性

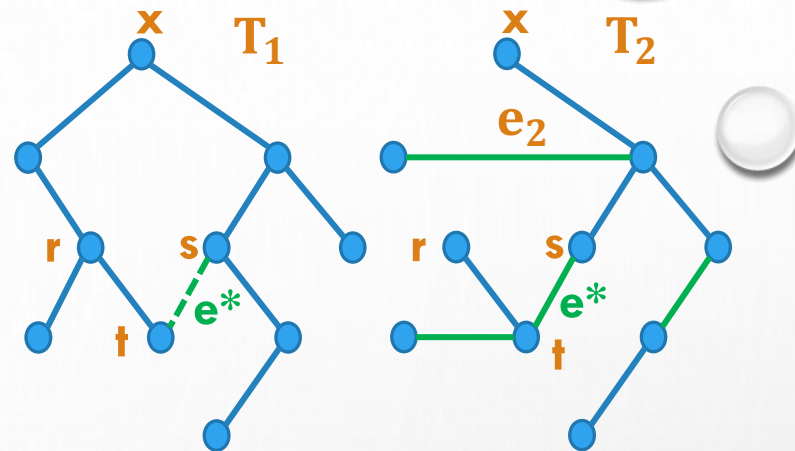


定理 Prim算法生成的MST，记为 T_1 ，是最小生成树。

证明 反证法。假设 T_1 不是最小生成树。

- 假设存在 T_2 ，为与 T_1 具有相同边数最多的实际MST，则 $w(T_2) < w(T_1)$ 。
- 假定 $e^*=st$ 为在 T_2 中且不在 T_1 中的权值最小的边。
- 将 e^* 加入 T_1 中，形成环路。
- 根据引理，则环路中任意一条 T_1 边的权值都比 e^* 小。

最小生成树1：Prim算法的正确性



定理 Prim算法生成的MST，记为 T_1 ，是最小生成树。

证明 反证法。假设 T_1 不是最小生成树。

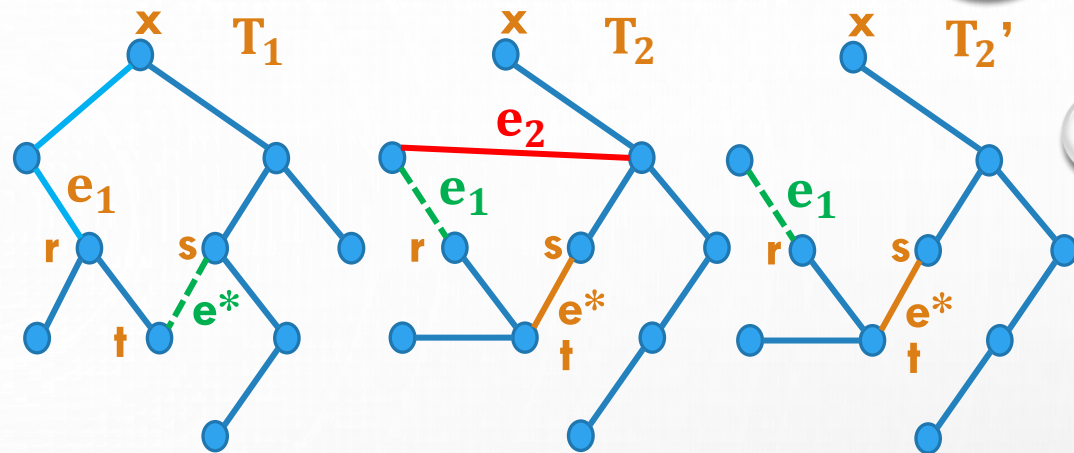
- 假设存在 T_2 ，为与 T_1 具有相同边数最多的实际MST，则 $w(T_2) < w(T_1)$ 。
- 假定 $e^*=st$ 为在 T_2 中且不在 T_1 中的权值最小的边。
- 将 e^* 加入 T_1 中，形成环路。
- 根据引理，则环路中任意一条 T_1 边的权值都比 e^* 小。

Prim算法的正确性

定理 Prim算法生成的MST，记为 T_1 ，是最小生成树。

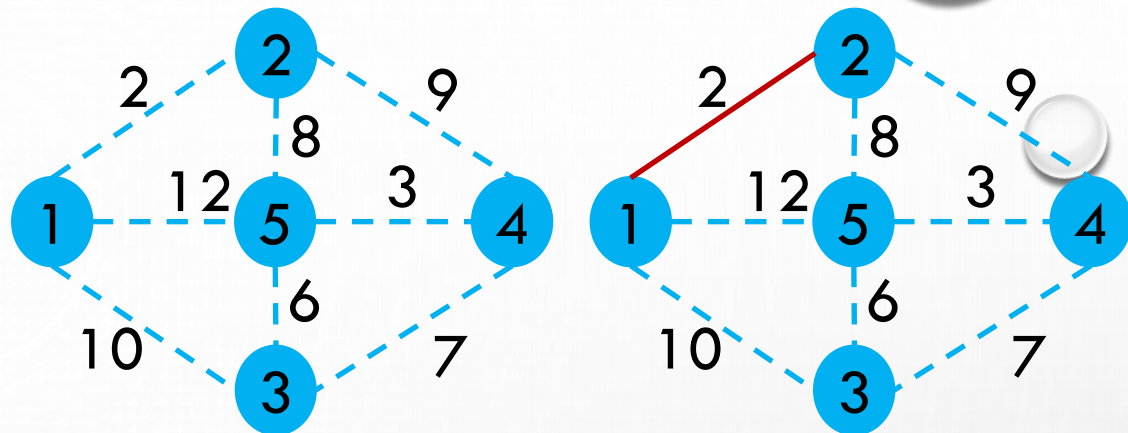
证明 反证法。

- 假设存在 T_2 ，为与 T_1 具有相同边数最多的实际MST，则 $w(T_2) < w(T_1)$ 。
- 假定 $e^*=st$ 为在 T_2 中且不在 T_1 中的权值最小的边。将 e^* 加入 T_1 中，形成环路。
- 根据引理，则环路中任意一条 T_1 边的权值都比 e^* 小。



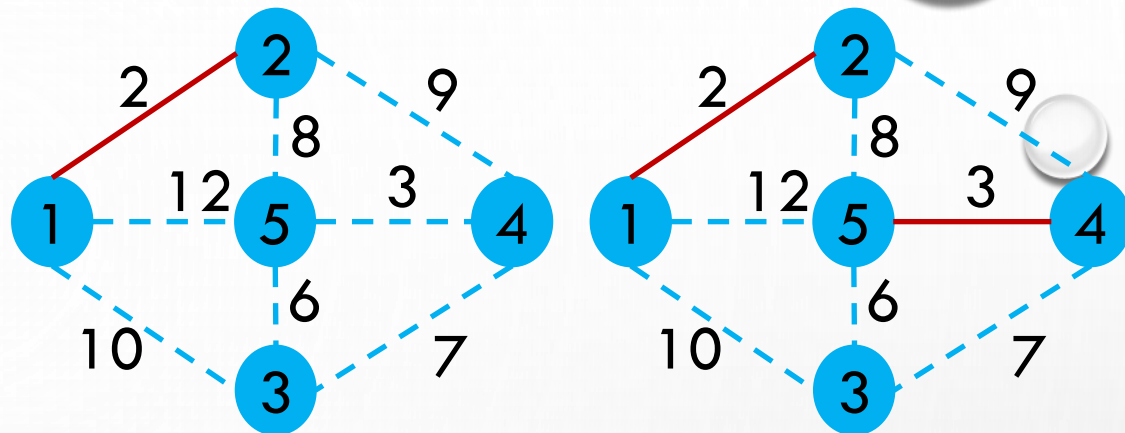
- 设 E_1 是在环路上且不在 T_2 中，且权值最小的边。将 E_1 加入 T_2 中构成回路。
- 在回路中找一条在 T_2 中但不在 T_1 中的边 E_2 并删除之，得 T_2' 。
 - 1) $W(E_1) < W(E_2)$, 则 $W(T_2') < W(T_2)$, 与 T_2 是一棵最小生成树矛盾；
 - 2) $W(E_1) \geq W(E_2)$, 因为 $W(E^*) > W(E_1)$, 与 E^* 是在 T_2 中不在 T_1 中的最小权值的边矛盾。
- 所以，PRIM算法执行过程是正确的。

最小生成树2: Kruskal算法



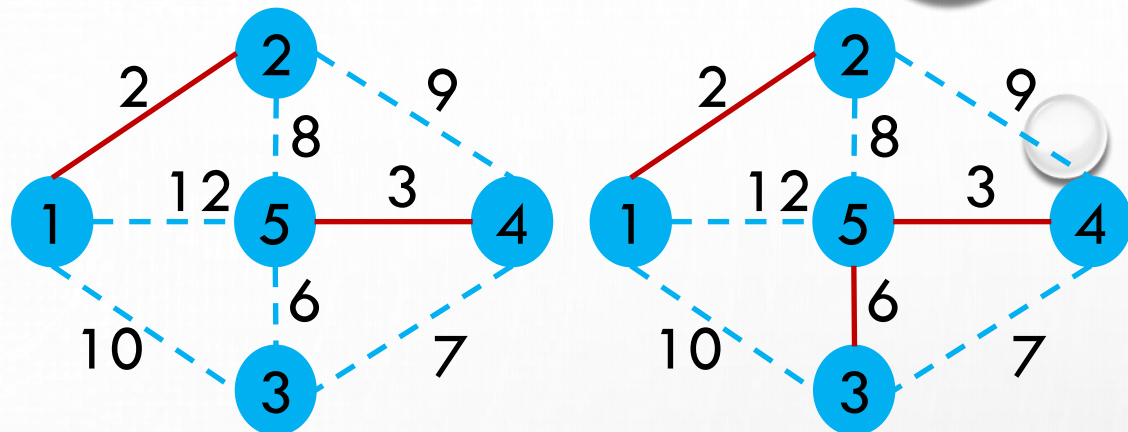
- 以右上图为例。
- 算法开始时，认为每一个点都是孤立的，分属于 n 个独立的集合。
- 每次选择一条最小的边，要求这条边的两个顶点分属于两个不同的集合。
将选取的这条边加入最小生成树，并且合并集合。
 1. 选择边(1, 2)，将这条边加入到生成树中，并且将它的两个顶点1, 2合并成一个集合。现在有4个集合 $\{(1, 2), \{3\}, \{4\}, \{5\}\}$ ，生成树中有一条边(1, 2)。

最小生成树2: Kruskal算法

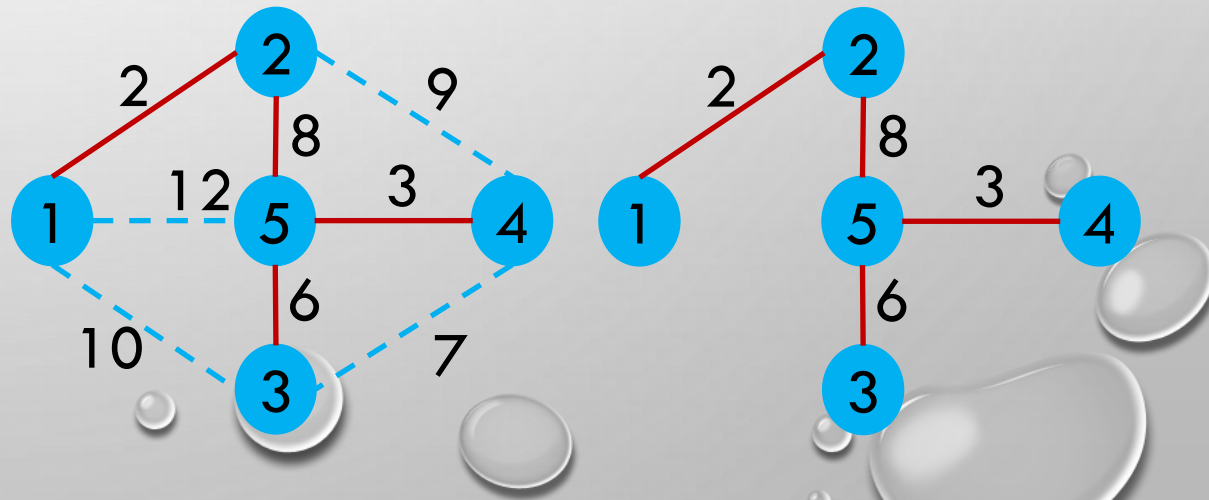


- 算法开始时，认为每一个点都是孤立的，分属于 n 个独立的集合。
- 每次选择一条最小的边，要求这条边的两个顶点分属于两个不同的集合。
将选取的这条边加入最小生成树，并且合并集合。
- 以右上图为例。
 1. 选择边(1, 2)，将这条边加入到生成树中，并且将它的两个顶点1, 2合并成一个集合。现在有4个集合 $\{(1, 2), \{3\}, \{4\}, \{5\}\}$ ，生成树中有一条边(1, 2)。
 2. 选择边(4, 5)，将其加入到生成树中，并且将它的两个顶点4, 5合并成一个集合。现在有3个集合 $\{(1, 2), \{3\}, \{4, 5\}\}$ ，生成树中有两条边(1, 2), (4, 5)。

最小生成树2: Kruskal算法



3. 选择边 $(3, 5)$, 将这条边加入到生成树中, 并且将它的两个顶点 $3, 5$ 所在的两个集合合并成一个集合。现有2个集合 $\{\{1, 2\}, \{3, 4, 5\}\}$, 生成树中有3条边 $\{(1, 2), (4, 5), (3, 5)\}$.
4. 选择边 $(2, 5)$, 将这条边加入到生成树中, 并且将它的两个顶点 $2, 5$ 所在的两个集合合并成一个集合。现有1个集合 $\{\{1, 2, 3, 4, 5\}\}$, 生成树中有4条边 $\{(1, 2), (4, 5), (3, 5), (2, 5)\}$.
5. 算法结束, 最小生成树权值为19.



哈夫曼编码

➤ $i=1$, $\{a:45, b:13, c:12, d:16, e:9, f:5\}$

$f.freq=5$ 最小, $x1=f$;

$e.freq=9$ 最小, $y1=e$;

$z1.freq=5+9=14$.

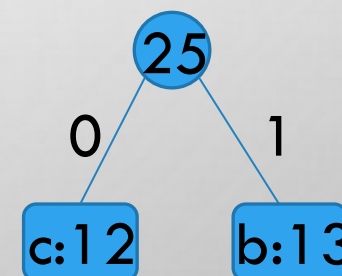
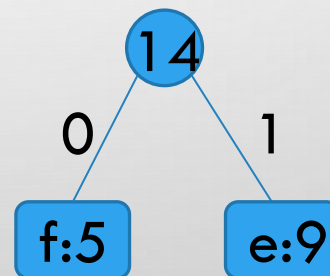
	a	b	c	d	e	f
频率	45	13	12	16	9	5
变长码	0	101	100	111	1101	1100

➤ $i=2$, $\{a:45, b:13, c:12, d:16, z1:14\}$

$c.freq=12$ 最小, $x2=c$;

$b.freq=13$ 最小, $y2=b$;

$z2.freq=12+13=25$.



哈夫曼编码

	a	b	c	d	e	f
频率	45	13	12	16	9	5
变长码	0	101	100	111	1101	1100

➤ $i=3$, $\{a:45, d:16, z1:14, z2:25\}$

$z1.freq=14$, $x3=z1$; $d.freq=16$, $y3=d$;

$z3.freq=14+16=30$.

➤ $i=4$, $\{a:45, z2:25, z3:30\}$

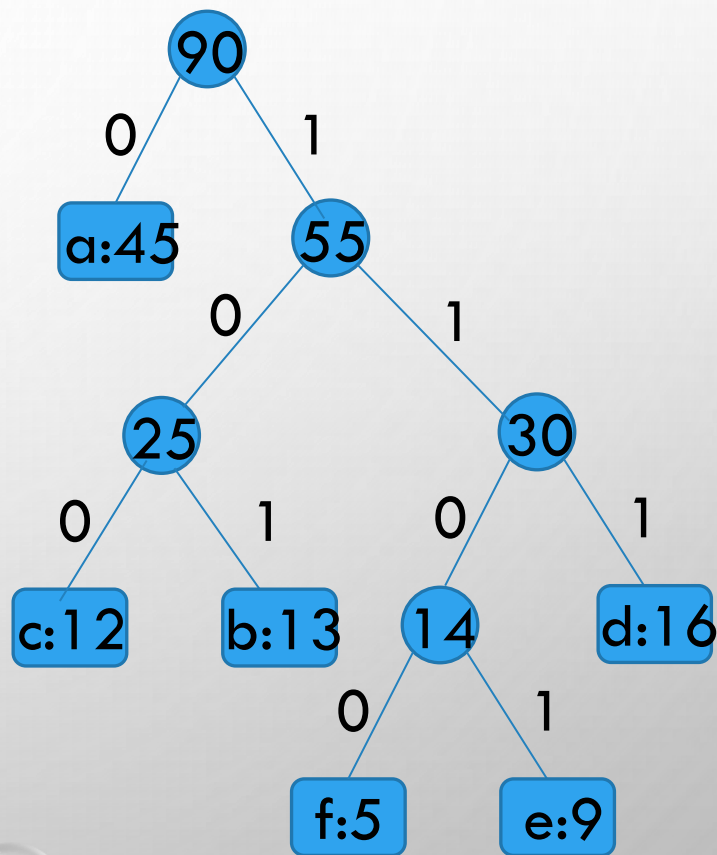
$z2.freq=25$, $x4=z2$; $z3.freq=30$, $y4=z3$;

$z4.freq=25+30=55$.

➤ $i=5$, $\{a:45, z4:55\}$

$a.freq=45$, $x5=a$; $z4.freq=55$, $y5=z4$;

$z5.freq=45+55=90$.



哈夫曼编码

	a	b	c	d	e	f
频率	45	13	12	16	9	5
变长码	0	101	100	111	1101	1100

➤ $l=3$, $\{A:45, D:16, Z1:14, Z2:25\}$

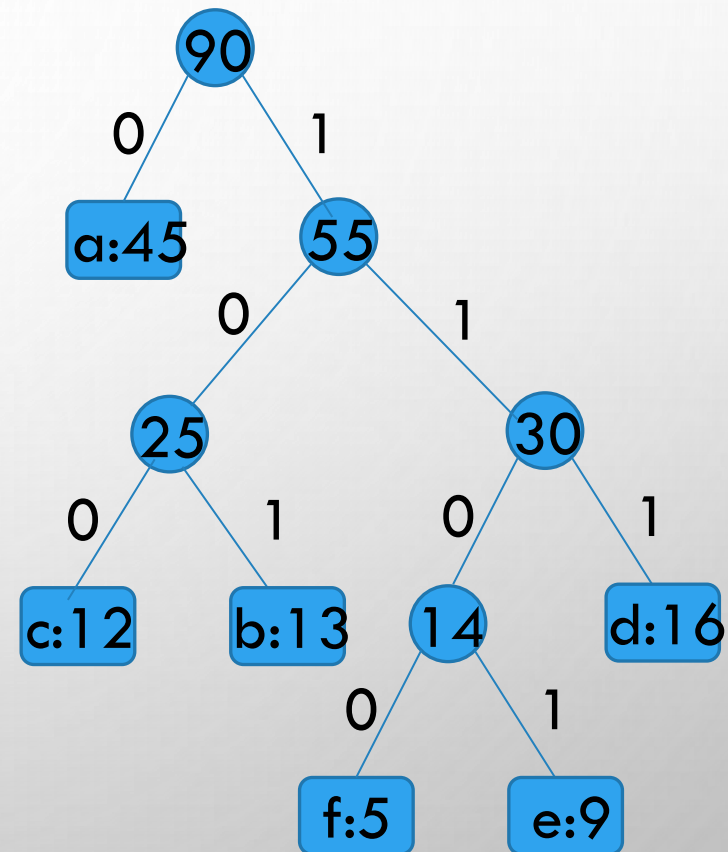
$Z1.FREQ=14$, $X3=Z1$; $D.FREQ=16$, $Y3=D$;

$Z3.FREQ=14+16=30$.

哈夫曼算法以自底向上的方式构造表示最优前缀码的二叉树T。

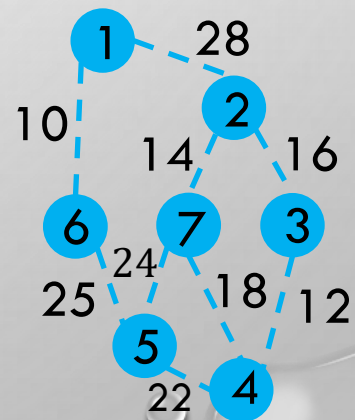
算法以 $|C|$ 个叶结点开始，执行 $|C|-1$ 次的“合并”运算后产生最终所要求的树T。

哈夫曼算法的贪心选择性质：合并出现频率最低的两个字符。



贪心算法3：最大完备子图问题

- 令 G 为无向图， S 为 G 中顶点的子集，当且仅当 S 中的任意两个顶点都有一条边相连时， S 为完备子图(团, clique)，完备子图的大小即 S 中的顶点数目。最大完备子图(最大团maximum clique)即具有最大顶点数目的完备子图。在图中寻找最大完备子图的问题(即最大完备子图问题)是一个NP—复杂问题。
 - 1) 给出最大完备子图问题的一种可行的贪婪算法及其伪代码。
 - 2) 给出一个能用1)中的启发式算法求解最大完备子图的图例，以及不能用该算法求解的一个图例。
 - 3) 可以使用以下贪心策略：如果一个节点的度数较大，则它有可能在一个最大集团中；写出伪代码算法；
 - 4) 对右图运行你的算法；举出反例。



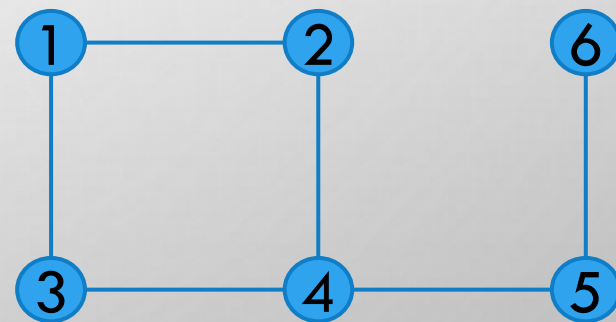
贪心算法4：最大完备子图问题

- 无向图的一个最大完全子图(包含意义下)称为一个集团, 集团的节点数称为集团的尺寸(size), 求图的最大集团. 这是NP难度问题. 本题要求给出一个贪心算法.
- 可考虑使用以下启发式: 如果一个节点的度数较大则它有可能在一个最大集团中.
- 算法首先将图的节点按度数排序然后从每个节点.

1. for $j \leftarrow 1$ to n do
 2. { $S \leftarrow \{v_j\}$;
 3. for $i \leftarrow 1$ to n do {
 4. 检查 $S \cup \{v_i\}$ 是否构成完全子图;
 5. 如构成完全子图, 将 v_i 加入到 S 中, 否则舍弃 v_i ;
 6. 设该集团的节点数为 d_j ;
 7. }/*找出包含 v_j 的集团*/
 8. }
 9. 从 d_j 中找出最大者, 输出对应的集团;
- 例如对书中图13.12(a)中的图, 上述算法输出最大集团; 反例也能找到。

贪心算法5：无向图的着色方案

- 无向图的一种着色方案指将颜色标号赋给图的顶点，使得任意两个有边连接的顶点的颜色标号均不同。求使用最少数目的不同颜色标号的着色方案称为图着色问题，这是一NP难度问题。试按“标号小的颜色优先”的贪心策略设计一图着色问题的启发式算法。要求：
 1. 写出算法的伪代码；
 2. 就下面的图从节点1开始运行你设计的算法并在图上标出所得到的着色方案；
 3. 上述贪心策略能保证得到最优解吗？

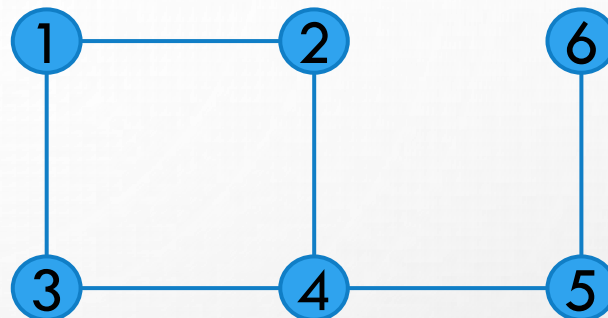


贪心算法6： 无向图的着色方案

解 1) 令 i 为节点号, c 为颜色标号; 算法的伪代码如下:

(2) 算法对上图各节点的着色如图中各节点旁标记的数字所示, 共使用了2种颜色。

(3) 否, 该启发式算法不总是产生一个最优的着色方案。在上图中如果按1,5,4,2,3,6的顺序运行上述算法, 则得到一个使用3种颜色的着色方案。



1. for($i = 1; i \leq n; i++$)
2. for($c = 1; c \leq n; c++$)
3. if no vertex adjacent to i has color c then color i with c ;
4. break; // exit for (c)
5. // continue for (c)
6. // continue for (i)

动态规划

动态规划

- 斐波那契数
- 多段图最短路径
- 矩阵乘法链
- All-Pair最短路(Floyd-Warshall算法)
- 不交叉网子集
- 等等

子集和数问题

设 s_i ($i = 1, 2, \dots, n$) 和 M 为任意给定的正数。 J 是下标 $\{1, 2, \dots, n\}$ 的一个子集， $S_J = \sum_{i \in J} s_i$ 为子集 J 的和数。最大子集和数问题指：求满足 $S_J \leq M$ 且使 S_J 最大的子集 J 。试用动态规划法设计求解最大子集和数问题的算法，要求：

(1) 列出动态规划的递归关系；

(2) 就实例： $n = 5, M = 12, s_1 = 8, s_2 = 2, s_3 = 7, s_4 = 9, s_5 = 5$ 执行算法并写出回溯（traceback）求解过程。

子集和数问题的DP递归关系

设 $f(i, y) = \sum_{k \in J} s_k$, 其中 J 为 $\{i, \dots, n\}$, 约束为 y 的最大子集和数问题的解。
有以下的递归关系:

$$f(n, y) = \begin{cases} s_n & y \geq s_n \\ 0 & y < s_n \end{cases}$$
$$f(i, y) = \begin{cases} \max\{f(i+1, y), f(i+1, y - s_i) + s_i\} & y \geq s_i \\ f(i+1, y) & 0 \leq y < s_i \end{cases}$$

子集和数问题的DP求解

$n = 5, M = 12, s_1 = 8, s_2 = 2, s_3 = 7, s_4 = 9, s_5 = 5。$

$P(5) = \{0, 5\},$

$s_4 = 9, Q4 = \{9\}$

$P(4) = \{0, 5, 9\},$

$s_3 = 7, Q3 = \{7, 12\}$

$12 = M,$ 算法结束。

回溯：12 出现在 Q3 中， $3 \in J$ ； $12 - s_3 = 5$ ，5 出现在 P(5) 中， $5 \in J$ ； $5 - s_5 = 0$ ，回溯结束。 $J = \{3, 5\}, S_J = M = 12。$

矩阵乘法链

设一个矩阵乘法链的行列数为 $r=(10,20,40,1,90)$

1. 写出解矩阵乘法链问题的动态规划递归关系式；
2. 用动态规划算法给出求解过程、优化的乘法顺序和优化的乘法数。

$$c(i, j) = \begin{cases} 0 & \text{if } j = i \\ r_i r_{i+1} r_{i+2} & \text{if } j = i+1 \\ \min_{i \leq k < j} \{c(i, k) + c(k+1, j) + r_i r_{k+1} r_{j+1}\} & \text{if } j > i+1 \end{cases}$$

矩阵乘法链DP算法

$r=(10,20,40,1,90)$

$C_{12}=8000, C_{23}=800, C_{34}=3600,$

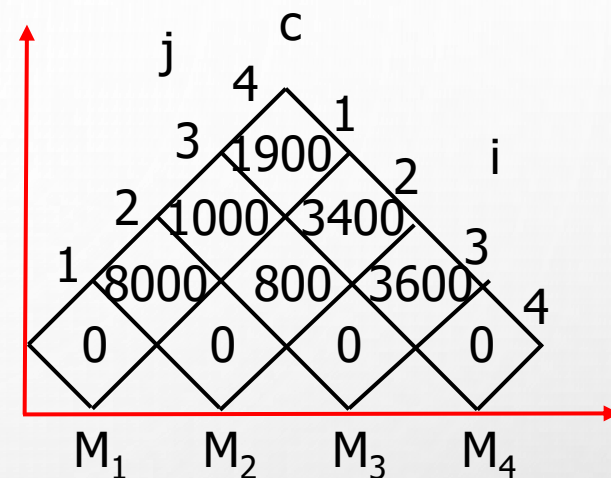
$C_{13}=C_{11}+C_{23}+200=1000;$

$C_{24}=C_{23}+C_{44}+2600=3400;$

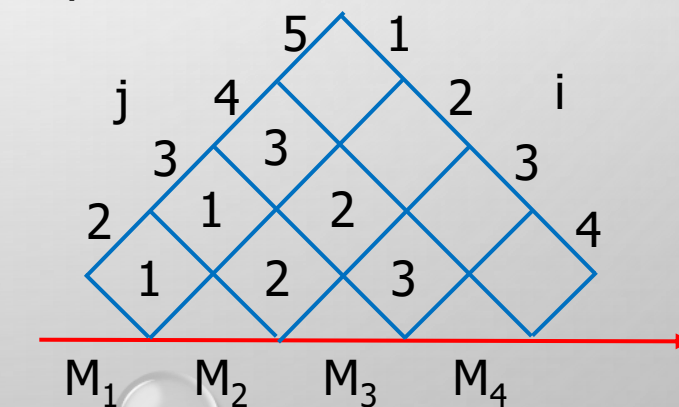
$C_{14}=C_{13}+C_{44}+900=1900;$

回溯：乘法数为1900，优化的乘法顺序为

$$(M_1*(M_2*M_3))*M_4$$



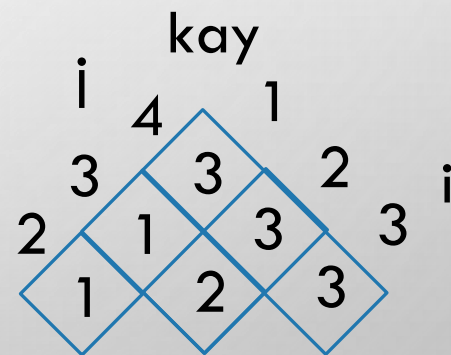
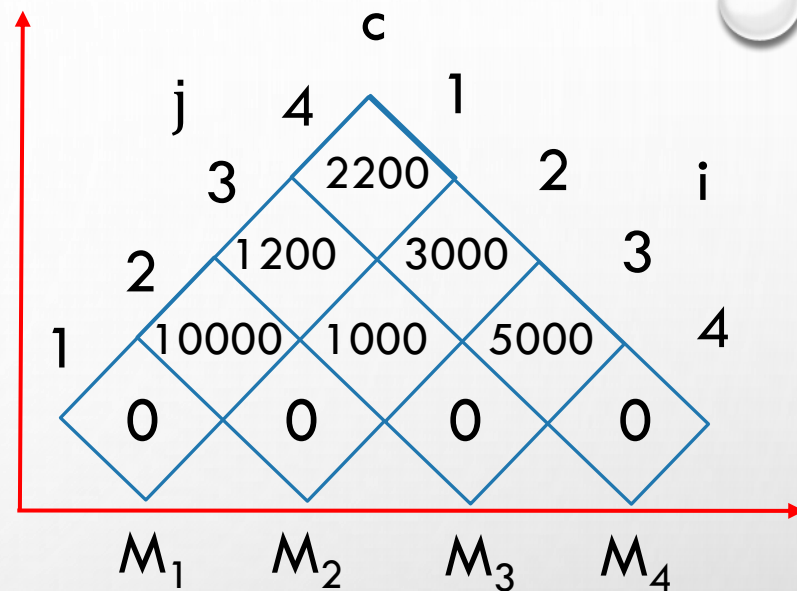
key: 达到最小值的 k , 即断开位置



矩阵乘法链

- 已知 $R=(10,20,50,1,100)$, 给出优化的乘法顺序和元素乘法数目.
- 设 $c(i,j)$ 为计算 $M(i,j)$ 所需乘法次数的最小值 (优化值), 根据优化原理, 优化值之间满足:

$$c(i, j) = \begin{cases} 0 & \text{if } j = i \\ r_i r_{i+1} r_{i+2} & \text{if } j = i + 1 \\ \min_{i \leq k < j} \{c(i, k) + c(k + 1, j) + r_i r_{k+1} r_{j+1}\} & \text{if } j > i + 1 \end{cases}$$



$$(M_1 * (M_2 * M_3)) * M_4$$

编辑距离问题 (公共子序列问题)

编辑距离问题：给定两个字符串 $S = s_1s_2\cdots s_n$ 和 $T = t_1t_2\cdots t_m$ ，编辑距离是指由一个转成另一个所需的最少编辑操作次数。许可的编辑操作包括将一个字符替换成另一个字符，插入一个字符，删除一个字符。一般来说，编辑距离越小，两个串的相似度越大。定义 $c(i, j)$ 是子串 $S_i = s_1s_2\cdots s_i$ 和 $T_j = t_1t_2\cdots t_m$ 的编辑距离，假设替换、插入及删除操作的费用是相同的：

1. 推导计算两个字符串之间编辑距离 $c(i, j)$ 的递归方程。
2. 写出利用编辑距离 $c(i, j)$ 输出两个序列公共子序列的伪代码。

最长公共子序列问题的递归关系

- $c(i, j)$ = 序列 S_i 和 T_j 的最长公共子序列的长度。
- 边界值：当 $i = 0$ 或 $j = 0$ 时， S_i 和 T_j 的最长公共子序列是 \emptyset ，此时 $c(i, j) = 0$ 。
- 当 $i, j > 0$ 时：
 - 如果 $S_i = T_j$ ，有 $c(i, j) = c(i - 1, j - 1) + 1$ 。
 - 如果 $S_i \neq T_j$ ，有 $c(i, j) = \max\{c(i, j - 1), c(i - 1, j)\}$ 。
- 递归关系式如下：

$$c(i, j) = \begin{cases} 0 & i, j = 0 \\ c(i - 1, j - 1) + 1 & i, j > 0; S_i = T_j \\ \max\{c(i, j - 1), c(i - 1, j)\} & i, j > 0; S_i \neq T_j \end{cases}$$

最长公共子序列算法

```
void LCSLENGTH (int m, int n, char *s,
    char *t, int **c, int **b) {
    int i, j;
    for (i = 1; i <= m; i++) c[i][0] = 0;
    for (i = 1; i <= n; i++) c[0][i] = 0;
    for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++) {
            if (s[i]==t[j]) {
                c[i][j]=c[i-1][j-1]+1; b[i][j]=1;}
            else if (c[i-1][j]>=c[i][j-1]) {
                c[i][j]=c[i-1][j]; b[i][j]=2;}
            else { c[i][j]=c[i][j-1]; b[i][j]=3; } } }
```

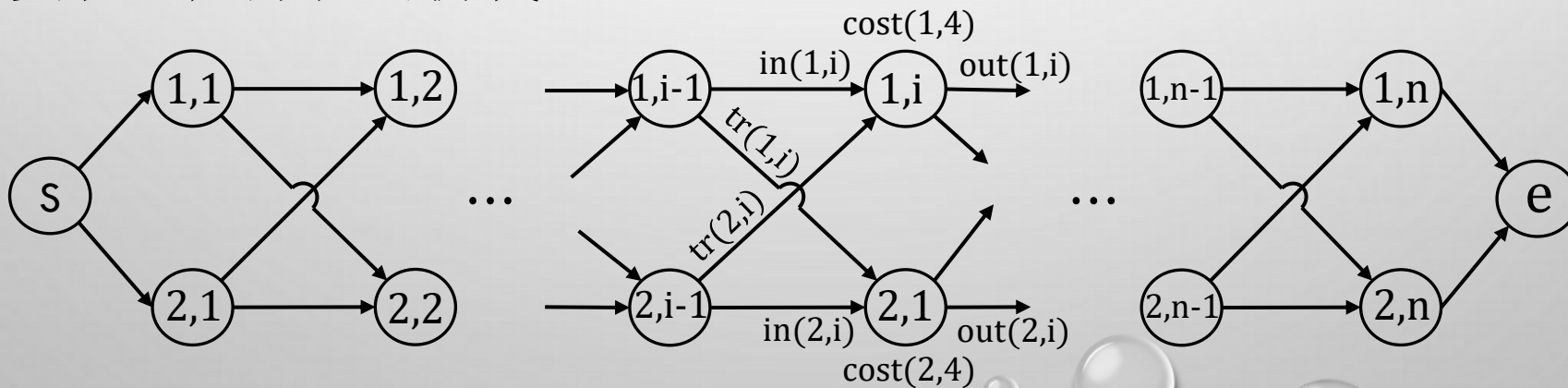
构造最长公共子序列

```
void LCS(int i, int j, char *s, int **b)
{
    if (i ==0 || j==0) return;
    if (b[i][j]== 1)
    {
        lcs(i-1, j-1, s, b);
        cout<<s[i];
    }
    else if (b[i][j]== 2) lcs(i-1, j, s, b);
    else lcs(i, j-1, s, b);
}
```

装配线调度问题

有 2 个装配线，每一个装配线上有 n 个装配站， $\text{site}[i, j]$ 表示第 i 个装配线上的第 j 个装配站。两个装配线上相同位置的转配站有相同的功能。在装配站 $\text{site}[i, j]$ 上花费时间为 $\text{cost}[i, j]$ 。进入和退出装配线 i, j 的时间分别为 $\text{in}[i, j]$ 和 $\text{out}[i, j]$ 。从一个装配站到相同装配线的下一个的时间忽略不计，到不同的装配站的时间为 $\text{transfer}[i][j]$ 。利用动态规划算法给出时间最少的装配方案。

1. 定义最优值函数并给出最优值的递归方程。
2. 写出实现递归方程的伪代码。



装配线调度问题递归方程

- 定义 $f(i, j)$ 为从初始状态到第 i 条装配线第 j 个装配站所用的最少时间。
- 由于从一个装配站到相同装配线的下一个的时间忽略不计，因此只需考虑从初始状态进入装配线的时间及退出装配线的时间，故对所有 $in[i, j]$ 中 j 只能为1，对于所有 $out[i, j]$ ， j 只能为 n

- 递归关系方程为：

$$\begin{cases} f(1, j) = \min\{f(1, j-1) + \text{cost}(1, j), f(2, j-1) + \text{trans}(2, j) + \text{cost}(1, j)\} \\ f(2, j) = \min\{f(2, j-1) + \text{cost}(2, j), f(1, j-1) + \text{trans}(1, j) + \text{cost}(2, j)\} \end{cases}$$

- 边界条件为：
$$\begin{cases} f(1, 1) = in(1, 1) + \text{cost}(1, 1) \\ f(2, 1) = in(2, 1) + \text{cost}(2, 1) \end{cases}$$

$$f = \min\{f(1, n) + out(1, n), f(2, n) + out(2, n)\}$$

· 装配线调度问题伪代码

$f(1, 1) = \text{in}(1) + \text{cost}(1, 1)$

$f(2, 1) = \text{in}(2) + \text{cost}(2, 1)$

for $j=2$ to n do

$f(1, j) = \min\{f(1, j-1) + \text{cost}(1, j), f(2, j-1) + \text{trans}(2, j) + \text{cost}(1, j)\}$

$f(2, j) = \min\{f(2, j-1) + \text{cost}(2, j), f(1, j-1) + \text{trans}(1, j) + \text{cost}(2, j)\}$

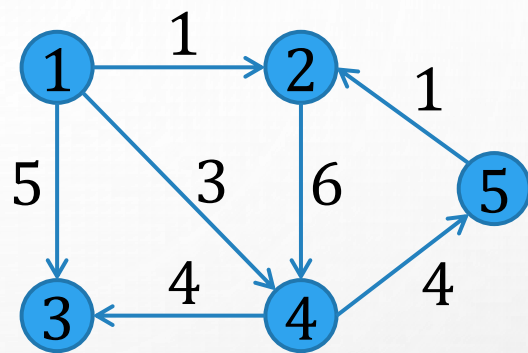
end for

return $\min\{f(1, n) + \text{out}(1), f(2, n) + \text{out}(2)\}$

动态规划求最短路径问题

求DP法求右图中各点间的最短路

- 解：先写出有向图对应的矩阵A



$$A = \begin{bmatrix} \infty & 1 & 5 & 3 & \infty \\ \infty & \infty & \infty & 6 & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 4 & \infty & 4 \\ \infty & 1 & \infty & \infty & \infty \end{bmatrix}$$

- $c(k) = (c(i, j, k)) = (\min\{c(i, j, k-1), c(i, k, k-1) + c(k, j, k-1)\} \mid i \neq j)$

- $k=0$ 时

- $k=1$ 时： $i \in \emptyset, j \in \{2, 3, 4\}$,

$$C^{(1)} = C^{(0)}$$

$$C^{(0)} = \begin{bmatrix} 0 & 1 & 5 & 3 & \infty \\ \infty & 0 & \infty & 6 & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & 4 & 0 & 4 \\ \infty & 1 & \infty & \infty & 0 \end{bmatrix} \quad C^{(1)} = \begin{bmatrix} 0 & 1 & 5 & 3 & \infty \\ \infty & 0 & \infty & 6 & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & 4 & 0 & 4 \\ \infty & 1 & \infty & \infty & 0 \end{bmatrix}$$

- $k=2$ 时: $i \in \{1,5\}, j \in \{4\}$

- $c(1,4,2) = \min\{c(1,4,1), c(1,2,1)+c(2,4,1)\} = \min\{3, 1+6\} = 3$

- $c(5,4,2) = \min\{c(5,4,1), c(5,2,1)+c(2,4,1)\} = \min\{\infty, 1+6\} = 7$

- $k=3$ 时: $i \in \{1,4\}, j \in \emptyset, C^{(3)}=C^{(2)}$.

- $k=4$ 时: $i \in \{1,2,5\}, j \in \{3,5\}$

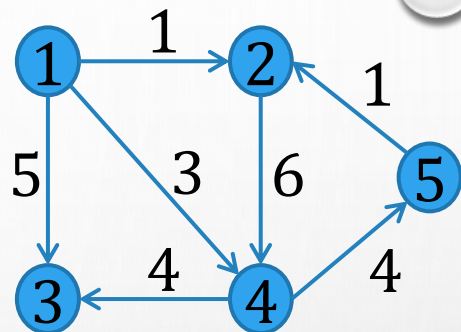
- $c(1,3,4) = \min\{c(1,3,4), c(1,4,3)+c(4,3,3)\} = \min\{5, 3+4\} = 5$

- $c(1,5,4) = \min\{c(1,5,4), c(1,4,3)+c(4,5,3)\} = \min\{\infty, 3+4\} = 7$

- $c(2,3,4) = \min\{c(2,3,3), c(2,4,3)+c(4,3,3)\} = \min\{\infty, 6+4\} = 10$

- $c(2,5,4) = \min\{c(2,5,3), c(2,4,3)+c(4,5,3)\} = \min\{\infty, 6+4\} = 10$

- $c(5,3,4) = \min\{c(5,3,3), c(5,4,3)+c(4,3,3)\} = \min\{\infty, 7+4\} = 11$



$$\begin{bmatrix} \infty & 1 & 5 & 3 & \infty \\ \infty & \infty & \infty & 6 & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 4 & \infty & 4 \\ \infty & 1 & \infty & \infty & \infty \end{bmatrix} = A$$

$$C^{(2)} = \begin{bmatrix} 0 & 1 & 5 & 3 & \infty \\ \infty & 0 & \infty & 6 & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & 4 & 0 & 4 \\ \infty & 1 & \infty & 7 & 0 \end{bmatrix}$$

$$C^{(3)} = \begin{bmatrix} 0 & 1 & 5 & 3 & \infty \\ \infty & 0 & \infty & 6 & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & 4 & 0 & 4 \\ \infty & 1 & \infty & 7 & 0 \end{bmatrix}$$

$$C^{(4)} = \begin{bmatrix} 0 & 1 & 5 & 3 & 7 \\ \infty & 0 & 10 & 6 & 10 \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & 4 & 0 & 4 \\ \infty & 1 & 11 & 7 & 0 \end{bmatrix}$$

- $k=5$ 时: $i \in \{1,2,4\}, j \in \{2,3,4\}$
- $c(1,2,5) = \min\{c(1,2,4), c(1,5,4) + c(5,2,4)\} = \min\{1, 7+1\} = 1$
- $c(1,3,5) = \min\{c(1,3,4), c(1,5,4) + c(5,3,4)\} = \min\{5, 7+11\} = 5$
- $c(1,4,5) = \min\{c(1,4,4), c(1,5,4) + c(5,4,4)\} = \min\{3, 7+7\} = 3$
- $c(2,3,5) = \min\{c(2,3,4), c(2,5,4) + c(5,3,4)\} = \min\{10, 1+11\} = 10$
- $c(2,4,5) = \min\{c(2,4,4), c(2,5,4) + c(5,4,4)\} = \min\{6, 1+7\} = 6$
- $c(4,2,5) = \min\{c(4,2,4), c(4,5,4) + c(5,2,4)\} = \min\{\infty, 4+1\} = 5$
- $c(4,3,5) = \min\{c(4,3,4), c(4,5,4) + c(5,3,4)\} = \min\{4, 4+11\} = 4$

$$C^{(4)} = \begin{bmatrix} 0 & 1 & 5 & 3 & 7 \\ \infty & 0 & 10 & 6 & 10 \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & 4 & 0 & 4 \\ \infty & 1 & 11 & 7 & 0 \end{bmatrix}$$

$$C^{(5)} = \begin{bmatrix} 0 & 1 & 5 & 3 & 7 \\ \infty & 0 & 10 & 6 & 10 \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \textcolor{red}{5} & 4 & 0 & 4 \\ \infty & 1 & 11 & 7 & 0 \end{bmatrix}$$

回溯和分支限界

回溯和分支限界

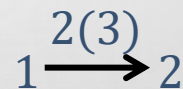
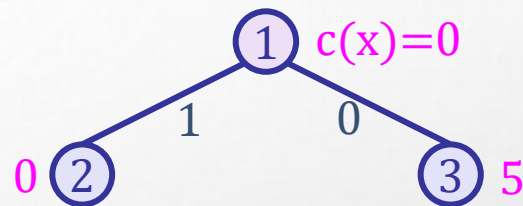
- 回溯法：
 - 8-皇后问题
 - 货箱装船问题
 - 图的 m 着色问题
 - 最大团问题
 - 子集和数问题
 - 多段图问题等
- 分支限界法：
 - 旅行商问题(哈密顿回路)
 - 作业调度问题等

最小罚款额作业调度

- 令三元组 (p_i, d_i, t_i) 表示一个作业的罚款额、截止期和执行时间. 试用LC-分枝限界法求解以下最小罚款额调度问题的实例: 4个作业的罚款额、截止期和执行时间分别为 $(5,1,1)$, $(10,3,2)$, $(6,2,1)$, $(3,1,1)$ 。
 - 1) 写出所使用的限界条件;
 - 2) 画出LC分枝-限界过程中展开的部分状态空间树, 并给出优化解和优化值。

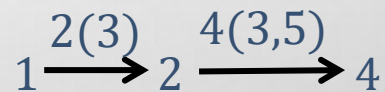
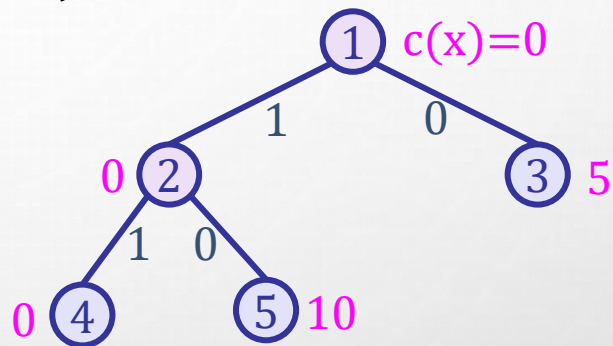
最小罚款额作业调度

- 一台机器，4 个作业：(5,1,1), (10,3,2), (6,2,1), (3,1,1)。
- $(p_i, d_i, t_i) = (\text{罚款额}, \text{截止期}, \text{需要的处理时间})$ 。
- 令 $c(x) =$ 当前结点 x 处已发生成本值。
- u 为当前可行解成本。
- 限界条件：
 - 显性约束：截止期 d_i ；(解的可行性)
 - 隐性约束： $c(x) \geq u$ 。(解的最优性)
- 初始值： $c(x) = 0$ ； $u = \infty$ 。



最小罚款额作业调度

- 一台机器，4 个作业：(5,1,1), (10,3,2), (6,2,1), (3,1,1)。
- $(p_i, d_i, t_i) = (\text{罚款额}, \text{截止期}, \text{需要的处理时间})$ 。
- 令 $c(x) =$ 当前结点 x 处已发生成本值。
- u 为当前可行解成本。
- 限界条件：
 - 显性约束：截止期 d_i ;
 - 隐性约束： $c(x) \geq u$ 。
- 初始值： $c(x) = 0$; $u = \infty$ 。



最小罚款额作业调度

- 一台机器，4 个作业：(5,1,1), (10,3,2), (6,2,1), (3,1,1)。

- $(p_i, d_i, t_i) = (\text{罚款额}, \text{截止期}, \text{需要的处理时间})$ 。

- 令 $c(x) =$ 当前结点 x 处已发生成本值。

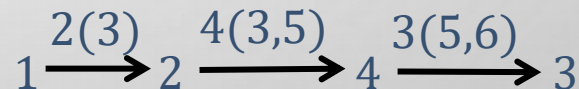
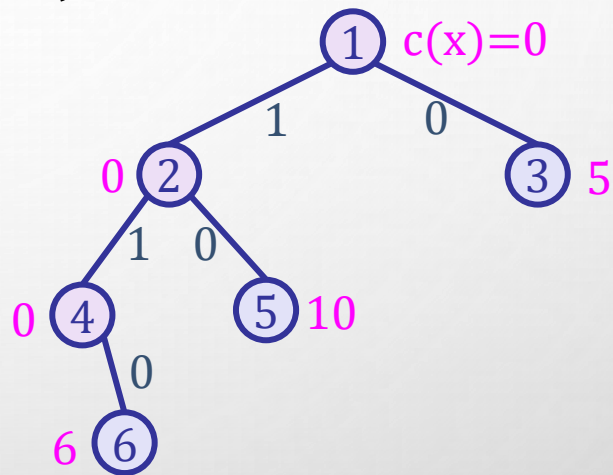
- u 为当前可行解成本。

- 限界条件：

- 显性约束：截止期 d_i ;

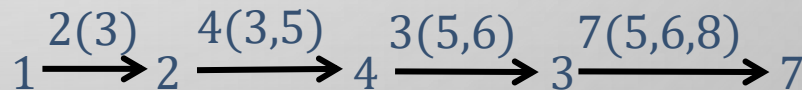
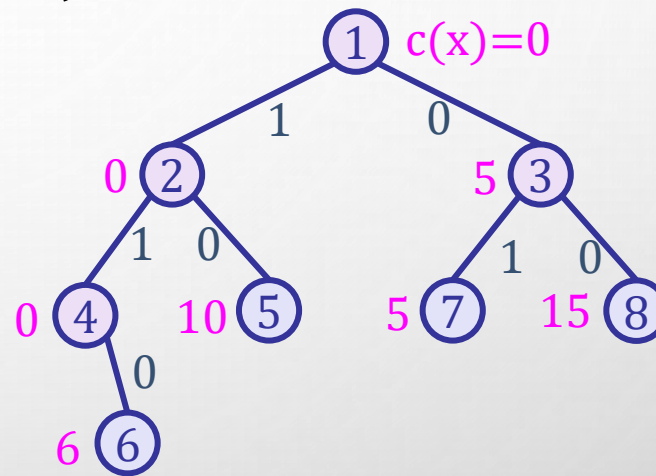
- 隐性约束： $c(x) \geq u$ 。

- 初始值： $c(x) = 0$; $u = \infty$ 。



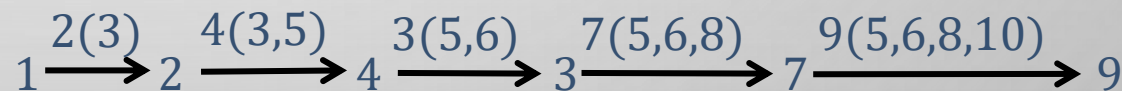
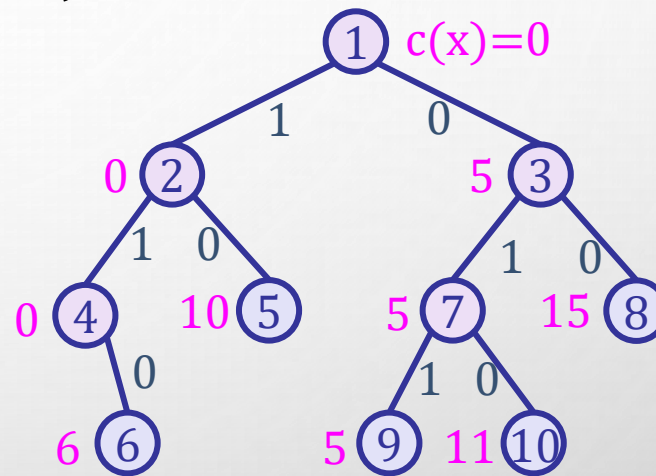
最小罚款额作业调度

- 一台机器，4 个作业：(5,1,1), (10,3,2), (6,2,1), (3,1,1)。
- $(p_i, d_i, t_i) = (\text{罚款额}, \text{截止期}, \text{需要的处理时间})$ 。
- 令 $c(x) =$ 当前结点 x 处已发生成本值。
- u 为当前可行解成本。
- 限界条件：
 - 显性约束：截止期 d_i ;
 - 隐性约束： $c(x) \geq u$ 。
- 初始值： $c(x) = 0$; $u = \infty$ 。



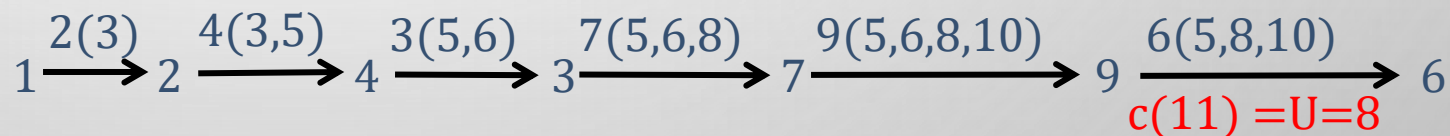
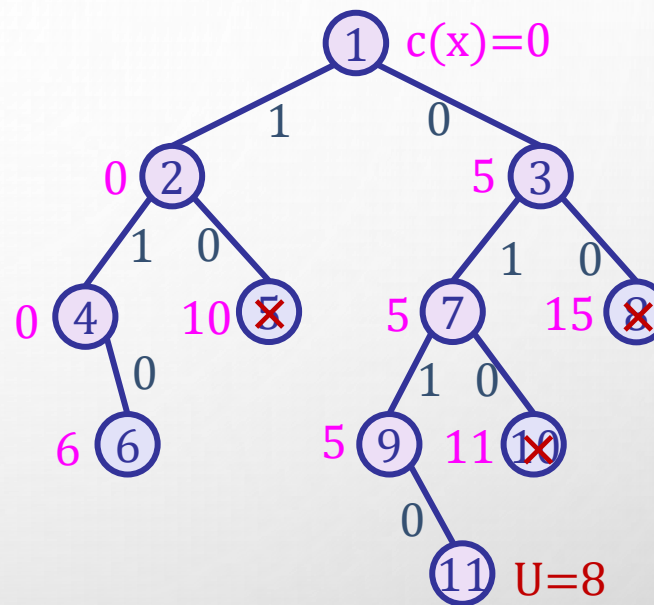
最小罚款额作业调度

- 一台机器，4 个作业：(5,1,1), (10,3,2), (6,2,1), (3,1,1)。
- $(p_i, d_i, t_i) = (\text{罚款额}, \text{截止期}, \text{需要的处理时间})$ 。
- 令 $c(x) =$ 当前结点 x 处已发生成本值。
- u 为当前可行解成本。
- 限界条件：
 - 显性约束：截止期 d_i ;
 - 隐性约束： $c(x) \geq u$ 。
- 初始值： $c(x) = 0$; $u = \infty$ 。



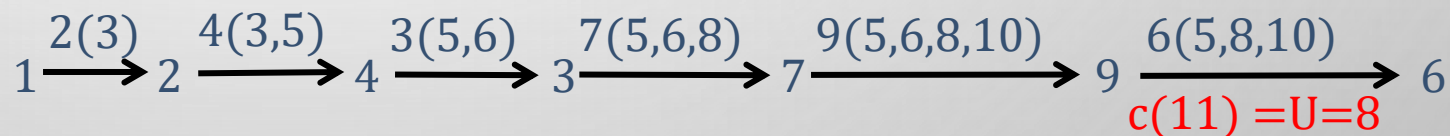
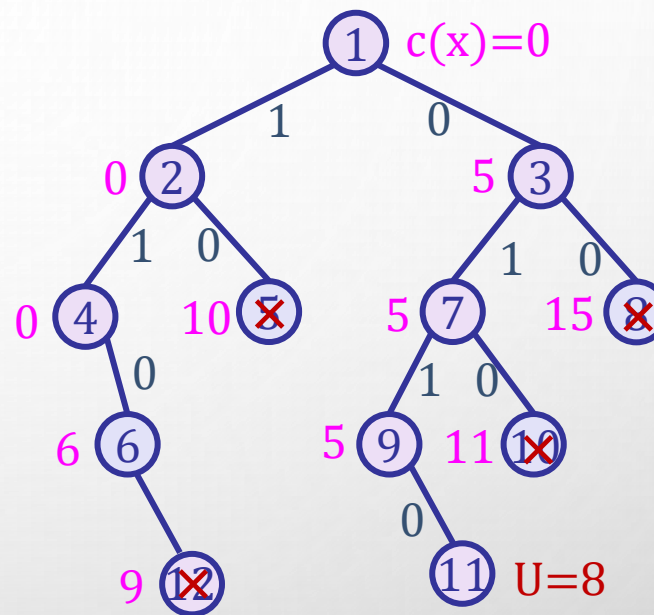
最小罚款额作业调度

- 限界条件：
 - 显性约束：截止期 d_i ;
 - 隐性约束： $c(x) \geq u$ 。
- 初始值： $c(x) = 0$; $u = \infty$ 。
- 优化解(0, 1, 1, 0)，优化值 = 8。



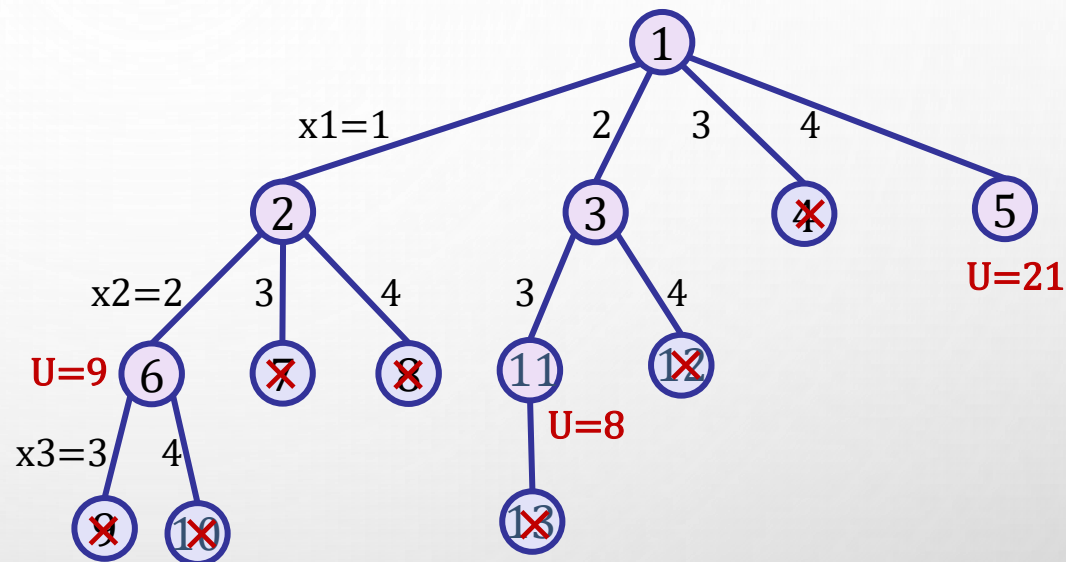
最小罚款额作业调度

- 限界条件：
 - 显性约束：截止期 d_i ;
 - 隐性约束： $c(x) \geq u$ 。
- 初始值： $c(x) = 0$; $u = \infty$ 。
- 优化解(0, 1, 1, 0)，优化值 = 8。



最小罚款额作业调度

- 限界条件：
 - 显性约束：截止期 d_i ;
 - 隐性约束： $c(x) \geq u$ 。
- 初始值： $c(x) = 0$; $u = \infty$ 。
- 优化解 $(0, 1, 1, 0)$;
- 优化值 $= 8$ 。



$$1 \xrightarrow[U(5)=21]{2(3,4)} 2 \xrightarrow{6(3,4,7)} 6 \xrightarrow[U(6)=9]{3(4,7)} 3 \xrightarrow{11} 11 \quad U(11)=8$$

最小罚款额作业调度

- 令三元组 (p_i, d_i, t_i) 表示一个作业的罚款额、截止期和执行时间。试用 LC – 分枝限界法求解以下最小罚款额调度问题的实例：4 个作业的罚款额、截止期和执行时间分别为 $(8, 2, 1)$ ， $(4, 2, 1)$ ， $(9, 3, 2)$ ， $(2, 1, 1)$ 。
 - 1) 写出所使用的限界条件；
 - 2) 画出 LC 分枝–限界过程中展开的部分状态空间树，并给出优化解和优化值。

最小罚款额作业调度

• 4个作业: $(8,2,1), (4,2,1), (9,3,2), (2,1,1)$ 。

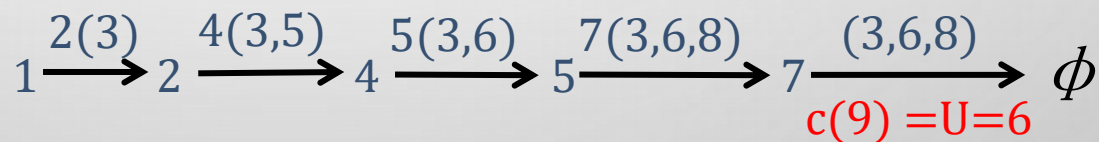
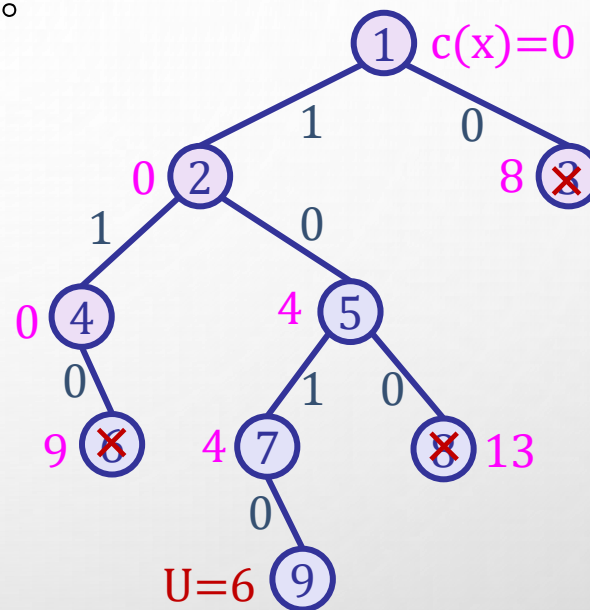
• 限界条件:

• 显性约束: 截止期 d_i ;

• 隐性约束: $c(x) \geq u$ 。

• 初始值: $c(x) = 0$; $u = \infty$ 。

• 优化解 $(1, 0, 1, 0)$, 优化值 = 6。



货箱装船问题

已知船的载重量为 c ，货箱的重量为 w_i ， $i = 1, \dots, n$ 。分别用回溯法和分支限界法找出使装载量最大的装船方案。要求：

1. 给出在两种方法中使用的限界条件；
2. 就实例 $n = 4$ ， $w = [10, 13, 7, 9]$ ， $c = 20$ 画出算法展开的部分状态空间树。

货箱装船问题限界条件

限界方法1：设 $cw = \sum_{0 < j < i} w_j x_j$ 为当前已装的重量。如果 $cw + w_i > c$ ，则杀死该(左)子节点。

限界方法2：设 $bestw$ 为当前最优装箱重量，初始值为 $-\infty$ 。 r 为未装货箱的连续背包问题的贪心值。 $cw + r$ 为该节点对应的子问题的优化值的上界。

两种限界同时使用，即

限界条件：

显性约束： $cw + w_i > c$ ；

隐性约束： $cw + r < bestw$ 。

货箱装船问题的回溯法状态空间树

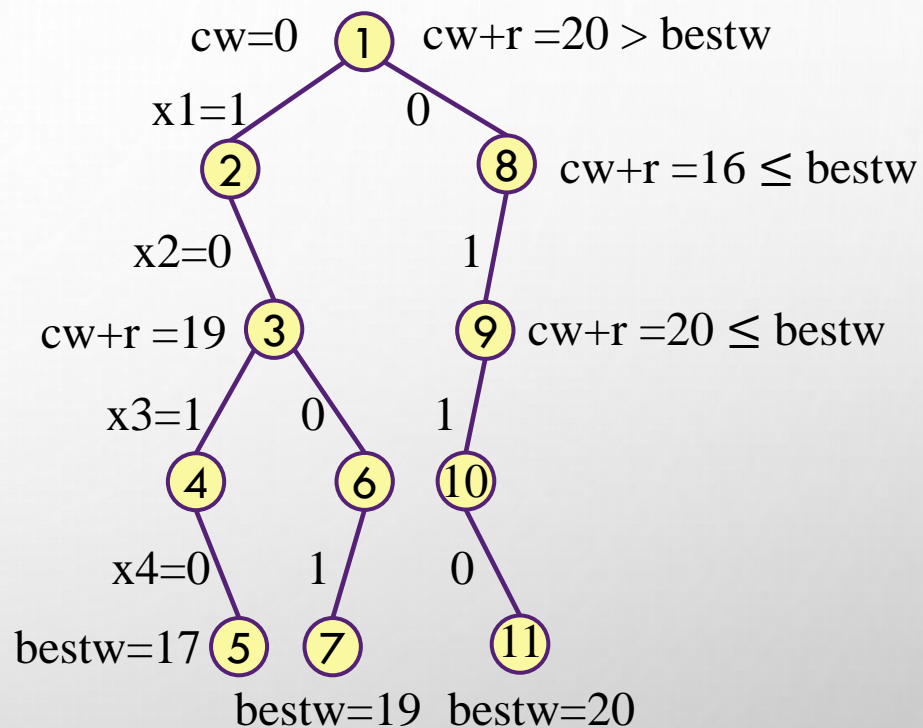
- $n = 4$, $w = [10, 13, 7, 9]$, $c = 20$ 。

限界条件:

显性约束: $cw + w_i > c$;

隐性约束: $cw + r \leq bestw$ 。

- 回溯法状态空间树见右图。
- 最优解为(0, 1, 1, 0), 最优值为20。
- 如果算法中增加一条判断: 当 $bestw = c$ 时返回结果, 则节点 8 和 9 处的判断不用做。



货箱装船问题的分支限界法状态空间树

- $n = 4$, $w = [10, 13, 7, 9]$, $c = 20$ 。

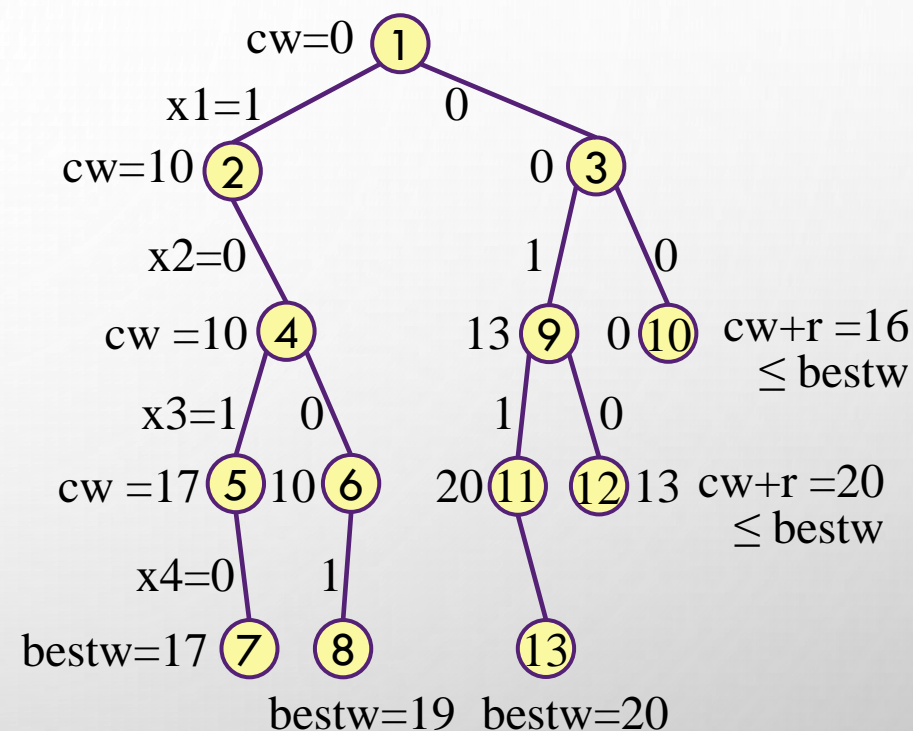
限界条件:

显性约束: $cw + w_i > c$;

隐性约束: $cw + r \leq bestw$ 。

启发式: cw 大的节点先展开。

- 分支限界法状态空间树见右图。
- 最优解为 $(0, 1, 1, 0)$, 最优值为20。
- 如果算法中增加一条判断: 当 $bestw = c$ 时返回结果, 则不用展开节点 13, 且节点 10、12 处的判断不用做。



最大团问题(Max Clique)

给定具有 n 个节点的无向加权图 G ，利用回溯法找出图 G 中的最大团。要求：

1. 写出回溯法求解该问题的两种限界函数。
2. 写出利用上述限界条件从图 G 中获取最大团的伪代码。

最大团问题的回溯法

限界条件:

左支限界: 当前点到集合中某个点没有边;

右支限界: $cn + n - i \leq bestn$

伪代码:

1. 初始时最大团为一个空集, 依次考虑每个顶点。
2. 查看当前顶点是否到集合中各点均有边 (是否可以成团):

- a) 可以成团, 将该顶点加入团;
- b) 如果不行, 直接舍弃;

3. 递归判断下一顶点。
4. 如果剩余未考虑的顶点数加上团中顶点数 ($cn + n - i$) 不大于当前解的顶点数 ($bestn$), 回溯。
5. 当搜索到一个叶结点时, 停止搜索, 更新最优解和最优值。

子集和数问题

已知 $n + 1$ 个正数 w_i ($1 \leq i \leq n$) 和 M 。要求找出 $\{w_i\}$ 的所有子集，使得子集内元素之和等于 M 。用 n 元组的方法表示解向量，要求：

1. 给出回溯方法求解该问题的两种限界方法。
2. 给出利用上述限界条件的回溯法伪代码。

子集和数问题的限界条件和伪代码

限界条件：用定长元组的方法表示解向量，即 (x_1, x_2, \dots, x_n) ， $x_i \in \{0, 1\}$ 。
将 w_i 按非减序排列。

1. 左枝限界：

$$\sum_{1 \leq i \leq k} w_i x_i + w_{k+1} > M$$

2. 右枝限界：

$$\sum_{1 \leq i \leq k} w_i x_i + \sum_{k+1 \leq i \leq n} w_i < M$$

设 $s + r \geq M$ ；

$$s = w_1 x_1 + \dots + w_{k-1} x_{k-1}, r = w_k + \dots + w_n。$$

如果 $s + w_k + w_{k+1} < M$ ，展开左子节点：

$$x_k \leftarrow 1, s \leftarrow s + w_k, r \leftarrow r - w_k, k \leftarrow k + 1$$

如果 $s + w_k + w_{k+1} > M$ ，则 $r \leftarrow r - w_k$ ，并展开右子节点：

如果 $s + r < M$ 或 $s + w_{k+1} > M$ ，停止展开右子节点并回溯；

否则， $x_k \leftarrow 0, r \leftarrow r - w_k, k \leftarrow k + 1$ ；

回溯： $k \leftarrow k - 1$ (回到父节点)。