作业和练习

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(logn)　　B. O(n)　　C. O(nlogn)　　D. $O(n^2)$

算法分析概论-2

2. 求整数n(n≥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(logn)　　B. O(n)　　C. O(nlogn)　　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(m2)　　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(A)。

```
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

计步法-1

执行该语句所需步数 该语句的执行次数 该语句的总步数。

LINEAR-SEARCH	s/e	Frequency	Total steps
1. 将answer赋值为NOT-FOUND。	1	1	1
2. 对每个索引值i，按顺序从1取到n：	1	n+1	n+1
A. 如果A[i]=x，那么将answer赋值为i.	1	0~n	0~n
3. 返回answer的值作为输出。	1	1	1
Total			n+3~2n+3

计步法-2

BETTER-LINEAR-SEARCH	s/e	Frequency	Total steps
Int BLS(T a[], T&x, int n)			
{			
int i;	1	1	1
for (int i=0;i<n && a[i]!=x; i++)	1	1	1
if (i==n) return NOT-FOUND;	1	1	1
return i;	1	1	1
}			
Total			4

最好情况

计步法-3

BETTER-LINEAR-SEARCH	s/e	Frequency	Total steps
Int BLS(T a[], T&x, int n)			
{			
int i;	1	1	1
for (int i=0;i<n && a[i]!=x; i++)	1	n+1	n+1
if (i==n) return NOT-FOUND;	1	1	1
return i;	1	1	1
}			
Total			n+4

最坏情况

计步法-4

一条或多条执行时间是常数的语句可称为“1步”

SUMMATION	s/e	Frequency	Total steps
T Sum(T a[], int n)	0	0	0
{			
T tsum=0;	1	1	1
for (int i=0; i<n; i++)	1	n+1	n+1
tsum +=a[i]	1	n	n
return tsum	1	1	1
}			
Total			2n+3

计步法-5

Matrix Addition	s/e	Frequency	Total steps
`Void Add(T**a …)`			
`{`			
`for (int i=0; i<m; i++`	1	$m+1$	$m+1$
`for (int j=0;j<n; j++)`	1	$m\cdot(n+1)$	$m\cdot(n+1)$
`c[i][j]=a[i][j]+b[i][j];`	1	$m\cdot n$	$m\cdot n$
`}`			
Total			$2m\cdot n+2m+1$

计步法-6

分析以下函数执行的乘法次数,该函数计算两个nxn矩阵的乘法.

```
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<n; i++)
          for (int j=0; j<n; j++) {
               T sum = 0;
               for (int k = 0; k < n; k++)
                    sum := a[i][k] * b[k][j];
               c[i][j] = sum; }
}
```

1 次

n次

n次

n次

共n^3次乘法

计步法-7

以下函数计算一个mxn矩阵和nxp矩阵的乘法，分析执行的乘法次数。

```
template <class T>
void Mult (T **a, T **b, T **c, int m, int n, int p)
{// Multiply the mxn matrix a and the nxp matrix 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];
            c[i][j] = sum;
        }
}
```

1次

n次

p次

m次

共mpn次乘法

计步法-8

函数minmax找出数组a[0:n-1]中的最大和最小元素,试分析元素间比较次数.

```
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;
        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.

n<1时不比较

n-1次比较

n-1次比较

共计2(n-1)次比较

计步法-9

下面算法也计算A[0:N]中最大最小元素,试分析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)
{// 寻找a[0:n-1]中的最小元素和最大元素.
 // 如果数组中的元素数目小于1，返回 false.
    if (n < 1) return false;
    Min = Max = 0; //初始化
    for (int i = 1; i < n; i++){
        if (a[Min] > a[i]) Min = i;
        else if (a[Max] < a[i]) Max = i;}
    return true;
}
```

n<1时不比较

最好情况比较n-1次；

最坏情况比较2(n-1)次

计步法-10

以下是另一个顺序查找算法,试分析其最坏情形元素比较次数.

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

计步法-11

使用计算步数的方法分析下面算法的渐近复杂度

```
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<n; i++)
        for (int j=0; j<n; j++) {
            T sum = 0;
            for (int k = 0; k < n; k++)
                sum += a[i][k]*b[k][j];
                c[i][j] = sum; }
}
```

s/e	Frequency	Total steps
0	0	$\Theta(0)$
0	0	$\Theta(0)$
1	$\Theta(n)$	$\Theta(n)$
1	$\Theta(n^2)$	$\Theta(n^2)$
1	$\Theta(n^2)$	$\Theta(n^2)$
1	$\Theta(n^3)$	$\Theta(n^3)$
1	$\Theta(n^3)$	$\Theta(n^3)$
1	$\Theta(n^2)$	$\Theta(n^2)$
Total:		$\Theta(n^3)$

计步法-12

使用计算步数的方法分析下面算法的渐近复杂度

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

s/e	Frequency	Total steps
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:		$\Theta(n)$

计步法-12

使用计算步数的方法分析下面算法的渐近复杂度

```
template <class T>
void Rank ( T a[], int n, int r[])
{//Rank the n elements a[0:n-1].
    for (int i=0; i<n; i++)
        r[i] = 0; // initialize
// compare all element pairs
    for (int i=1; i<n; i++)
        for (int j=0; j<i; j++)
            if (a[j] <= a[i]) r[i]++;
            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:		$\Theta(n^2)$

代入法

用归纳法证明

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

$$\Rightarrow T(n) \le cn\lceil \log_2 n \rceil$$

证明 (1) n=1时，成立。

(2)令⌈X⌉表示X向上取整。设$2^k < n \le 2^{k+1}$，则有$\lceil \log n \rceil = k+1$；又因为$2^{k-1} < n/2 \le 2^k$，即$2^{k-1} < \lceil n/2 \rceil \le 2^k$，所以$\lceil \log \lceil n/2 \rceil \rceil = k$；$\lceil \log n \rceil = \lceil \log \lceil n/2 \rceil \rceil + 1$。令$n_1 = \lfloor n/2 \rfloor$, $n_2 = \lceil n/2 \rceil$。

用迭代方法求解下面棋盘覆盖问题的递归式

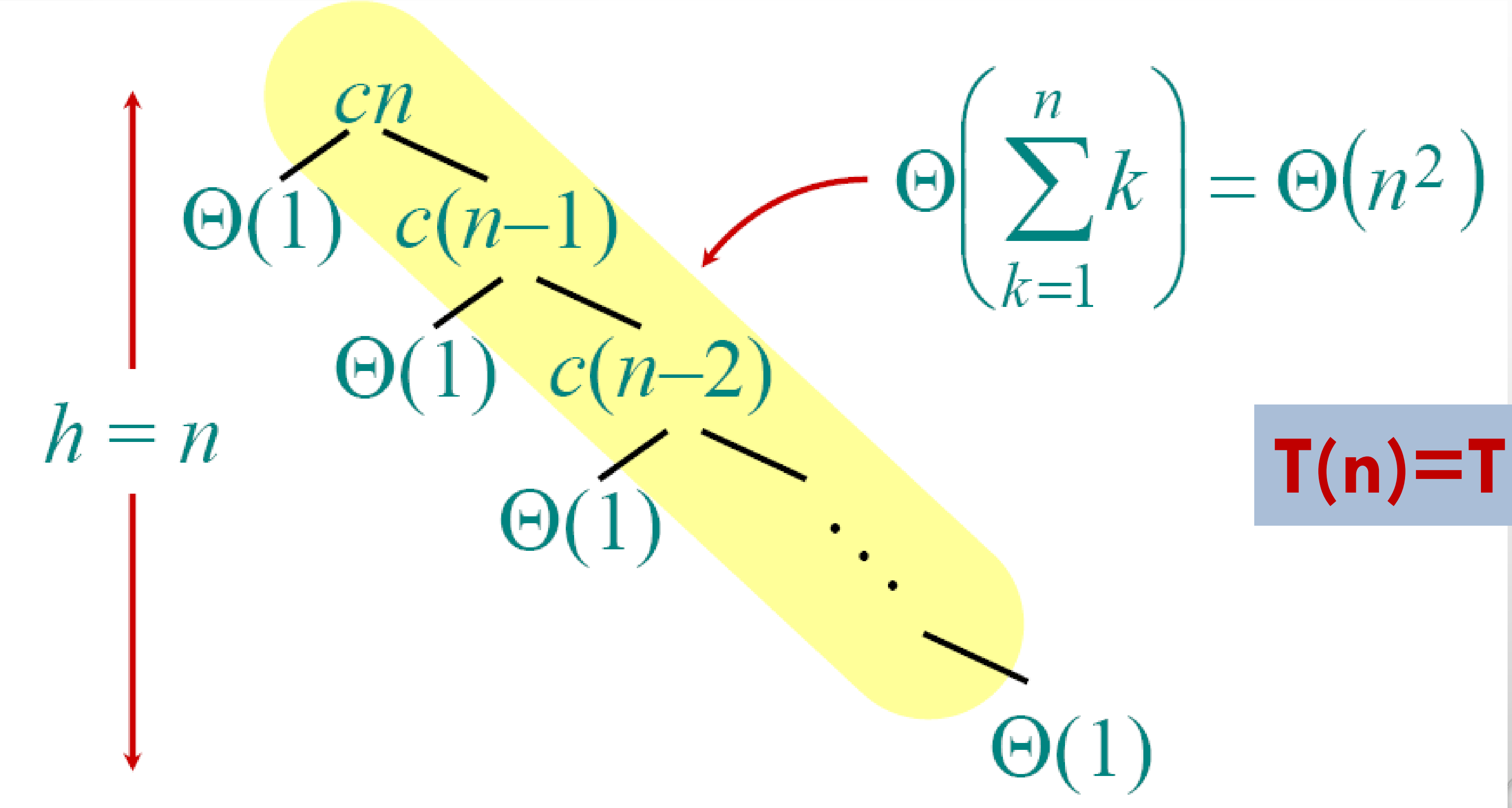
$$t(k) = \begin{cases} d & k = 0 \\ 4t(k-1) + c & k > 0 \end{cases}$$

cn

递归树-1

展开递归树：T(n)=T(0)+T(n-1)+cn, 并做渐近分析。

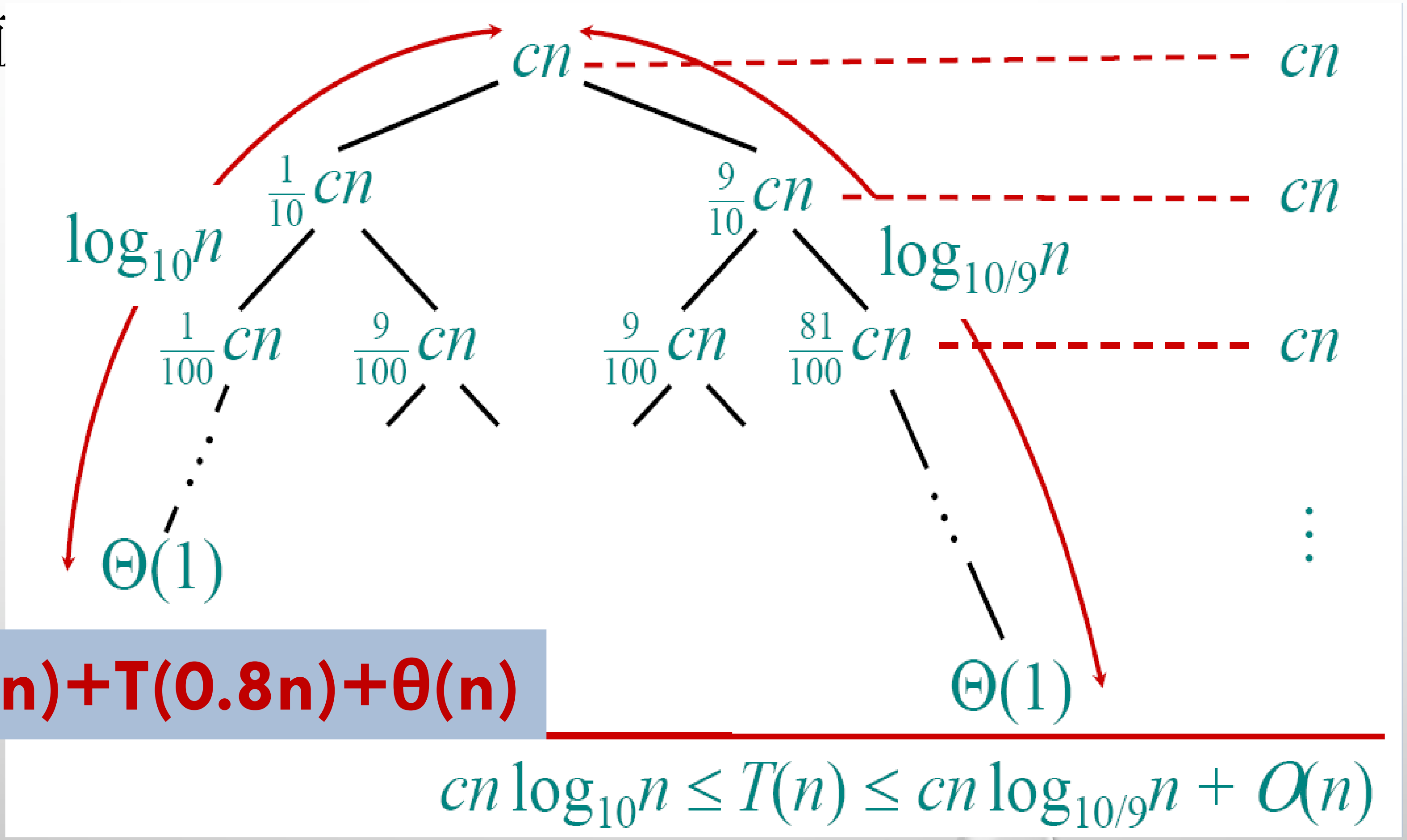
解：$T(n)=nT(0)+c[1+\ldots+(n-1)+n] = \Theta(n^2)$,



T(n)=T(2)+T(n−2)+cn

递归树-2

- 展开T(n)=T(0.1n)+ T(0.9n)+Θ(n)的递归树并计算递归树的深度和T(n)的渐近值



T(n)=T(0.2n)+T(0.8n)+θ(n)

主方法-1

应用master方法求解以下递归方程：

1) $T(n)=2T(n/2)+\Theta(n^{1/2})$
2) $T(n)=10T(n/3)+11n$
3) $T(n)= 10T(n/3)+11n^5$
4) $T(n)= 27T(n/3)+11n^3$
5) $T(n)= 64T(n/4)+10n^3\log^2 n$
6) $T(n)=9T(n/2)+\Theta(n^2 2^n)$
7) $T(n)=3T(n/8)+\theta(n^2 2^n \log n)$
8) $T(n)= 128T(n/2)+6n$
9) $T(n)= 128T(n/2)+\theta(n^8)$
10) $T(n)= 128T(n/2)+2^n/n$
11) $T(n)= 128T(n/2)+ \log^3 n$

主方法-2

Case 2的一般形式：

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

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

解：1) $a=b=2$, $n^{\log_b a}=n$, $f(n)=cn^{1/2}=o(n^{1-1/2})$。 $\therefore$ case 1, $T(n)=\theta(n)$;

2) $a=10$, $b=3$, $n^{\log_b a}>n^2$. $f(n)=cn= o(n^{2-1})$. $\therefore$ case 1, $T(n)= \theta(n^{\log_3 10})$;

3) $a=10$, $b=3$, $n^{\log_b a}<n^3$. $f(n)=cn^5= \omega(n^{5-2})$. 又因为当n充分大时 $af(n/b)=an^5/b^5<0.05n^5$ $\therefore$ case 3, $T(n)= \theta(n^5)$;

4) $a=27$, $b=3$, $n^{\log_b a} = n^3$. $f(n)=cn^3$. $\therefore$ case 2, $T(n)= \theta(n^3\log n)$;

5) $a=64$, $b=4$, $n^{\log_b a} = n^3$. $f(n)= cn^3\log^2 n$. $\therefore$ $T(n)= \theta(n^3\log^3 n)$; (case 2的一般情况）

6) case 3:$a=9$,$b=2$, $f(n)=n^2 2^n$,任取c, $T(n)=\theta(n^2 2^n)$

7) case 3: $a=3$,$b=8$, $f(n)= n^2 2^n \log n$,任取c, $T(n)=\theta(n^2 2^n \log n)$

8) case 1:$a=128$,$b=2$,$f(n)=6n$, $T(n)=\theta(n^7)$

9) case3,logba=7,$f(n)=n8=\omega(n^7+\varepsilon)$, $8(n/2)^8<cn^8$,$T(n)=\theta(n^8)$

10) case 3,$T(n)= \theta(2^n/n)$

11) case 1,$T(n)= \theta(n^7)$

时间复杂度

- 最好、最坏、平均
- 计步法
- 递归树法
- 主方法

主方法

主定理　令$a\geq1$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$T(n)=\Theta(f(n))$。

算法

- 递归算法：归并排序
- 分治法：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

连续背包问题-1

- 考虑$0 \le x_i \le 1$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-2

- 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=80;
- i=3, p=30+20(80/100)=46.

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连续背包问题的最优解。

连续背包问题-3

证明: 设$x=(x_1..., x_n)$为贪心法产生的解;则它有形式 $(1,1,..., x_j,0...0)$, 其中 $0<x_j<1$;设 $y=(y_1..., y_n)$是优化解;

- 设k是$x_i \neq y_i$的最小下标. 则$k \le j$且$y_k<x_k$.
- 将y_k增加到x_k,并从$\sum_{k<i\le n} y_i w_i$减去$(x_k-y_k)w_k$ (无论什么方式)得到$(z_k, z_{k+1},..., z_n)$, 其中$z_k=x_k$.
- 下面证明$(y_1... y_{k-1}, z_k, z_{k+1},..., z_n)$仍是优化解:
- 证明$\sum_{k\le i\le n} y_i p_i \le \sum_{k\le i\le n} z_i p_i$.
- 将$(y_i-z_i)p_i$改写成$(y_i-z_i)w_i p_i/w_i$. 利用$p_i/w_i \le p_k/w_k (i>k)$和$\sum_{k<i\le n} (y_i-z_i)w_i=(x_k-y_k)w_k$可得到上述不等式.

0/1背包问题-1

- 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$\boxed{1/(K+1)*100\%}$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)=$\begin{cases} \max\{c(i-1,y), c(i-1,y-w_i)+v_i\} & y \geq w_i \\ c(i-1,y) & 0 \leq y < w_i \end{cases}$，元组法。

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

- 分治限界：优先队列

0/1背包问题-2：贪心算法和动态规划

- 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,

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

- $c(1,y) = \begin{cases} v_1 & y \geq w_1 \\ 0 & 0 \leq y < w_1 \end{cases}$

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

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

0/1背包问题3: 贪心算法和1-优化

- n=8, w=[16, 20, 4, 15, 25, 10, 5, 8], p=[100, 200, 50, 90, 175, 50, 20, 60], c=70. 请用1-优化法求解。
- 解：物品的收益率依次为[6.25, 10, 12.5, 6, 7, 5, 4, 7.5]. 排序后物品顺序为[3, 2, 8, 5, 1, 4, 6, 7]. 对应的物品重量为w’=[4, 20, 8, 25, 16, 15, 10, 5], 效益p’= [50, 200, 60, 175, 100, 90, 50, 20].

k=0时计算结果为: x =(0, 1, 1, 0, 1, 1, 0, 1),得到的效益值为535。

k=1时计算结果为:

1. 先放物品2, 3, 5, 6, 8, 得到的效益值仍为535, 贪心解同上;
2. 先放物品1, 得到效益值520, 贪心解为(1, 1, 1, 1, 0, 0, 1, 1);
3. 先放物品4, 得到的效益值为520, 贪心解为(1, 1, 1, 1, 0, 0, 1, 1);
4. 先放物品7, 得到效益值为505, 贪心解为(0, 1, 1, 0, 1, 1, 1);
5. 所以1-优化法得到的解为(0, 1, 1, 0, 1, 1, 0, 1), 效益值为535.

0/1

$$f(i,y)=\begin{cases}\max\{f(i+1,y),\ f(i+1,y-w_i)+v_i\} & y \ge w_i \\ f(i+1,y) & 0 \le y < w_i\end{cases}$$

- 例：

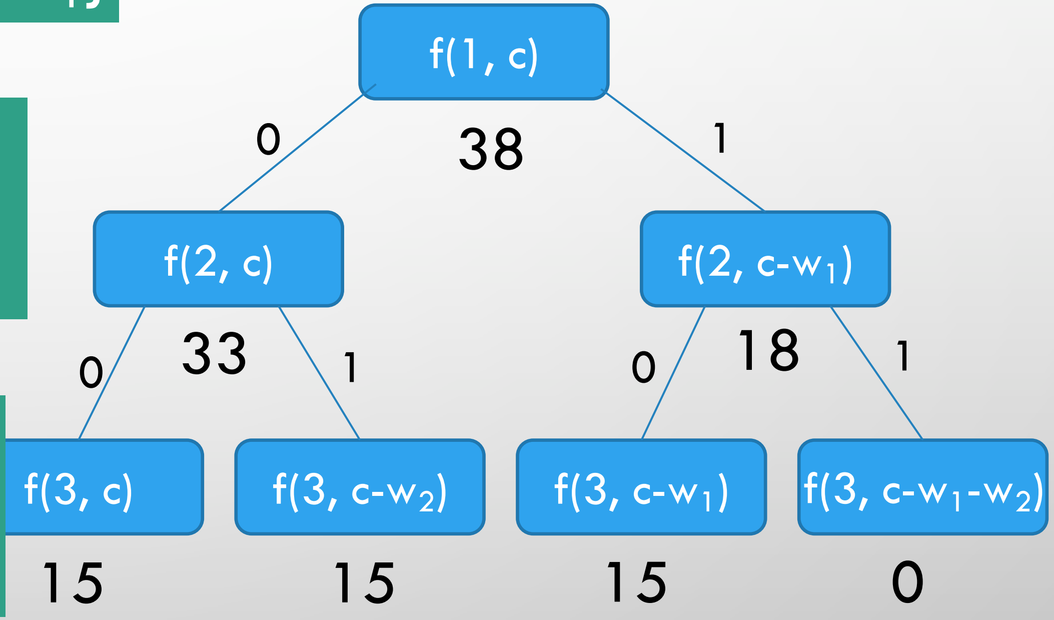
$$f(2,y)=\begin{cases}\max\{f(3,y),\ f(3,y-w_2)+18\} & y \ge 14 \\ f(3,y) & 0 \le y < 14\end{cases}$$

- 解：　最优值为38.

$$f(1,c)=\max\{f(2,c),\ f(2,c-w_1)+v_1\}$$

$$f(n,y)=\begin{cases}v_n & y \ge w_n \\ 0 & 0 \le y < w_n\end{cases}$$

$$f(3,y)=\begin{cases}15 & y \ge 10 \\ 0 & 0 \le y < 10\end{cases}$$



0/1背包问题5: DP法的递归求解

n=5, p=[6,3,5,4,6],w=[2,2,6,5,4]，c=10

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

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

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

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

0/1背包问题6: 元组法

写出背包问题实例n=5, p=[6,3,5,4,6],w=[2,2,6,5,4]，c=10的元组法求解过程。

- P(1)=[(0,0),(6,2)]; (w_2,v_2)=(3,2), Q2={(3,2),(9, 4)}
- P(2)=[(0,0),(3,2),(9,4)]; (w_3,v_3)=(5,6), Q3={(5,6),(8,8)}
- P(3)=[(0,0),(3,2),(5,6),(8,8)]; (w_4,v_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,v_5)=(6,4), Q5={(6,4),(10,9)}
- P(5)=[(0,0),(3,2),(4,5),(5,6),(7,7),(8,8),(9,11)];
- 回溯：P(5)中最大收益为11. Q4中包含(9,11), 选择物品4. (9,11)-(W_4,V_4)=(5,6), Q3中包含(5,6), 选择物品3. (5,6)-(W_3,V_3)=(0,0), 背包没有空间了, 所以最优解为(0,0,1,1,0).

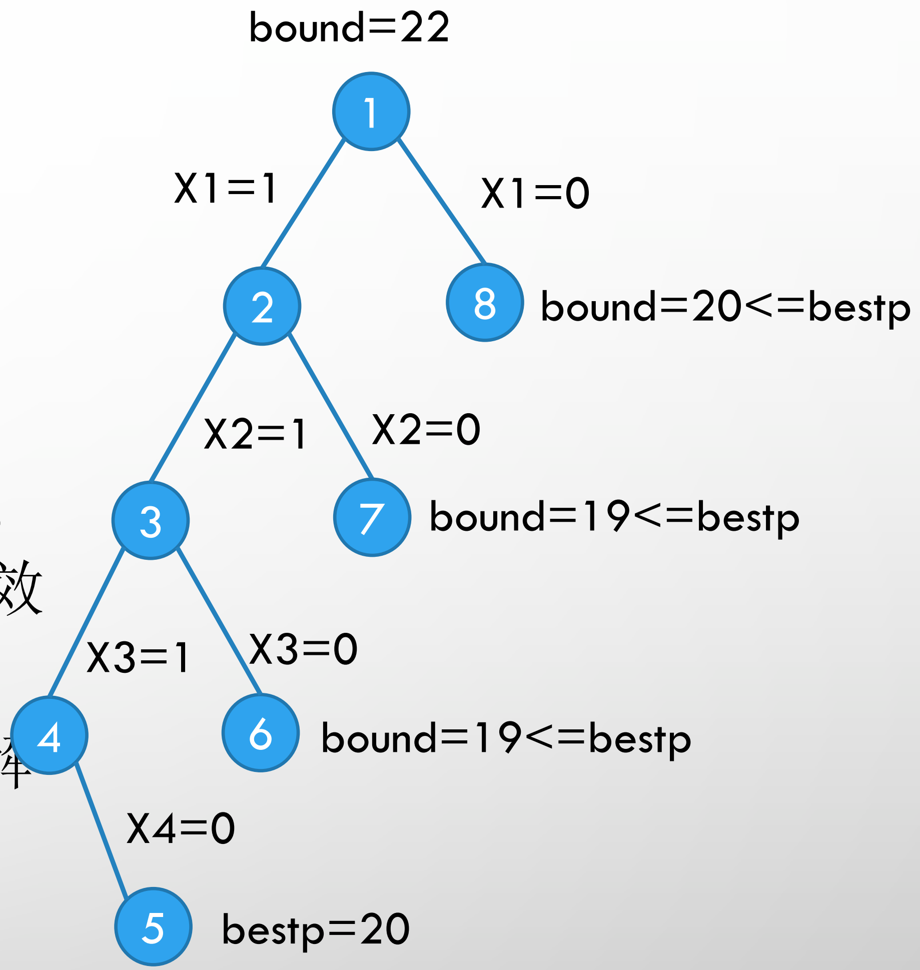
0/1背包问题7: DP法的时间复杂度

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$O(\min\{2^n, n\sum_{1\le i\le n} v_i, nc\})$.

- 每次产生一个表P(i)时的计算时间和表的长度成正比，共产生n个表. 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$\min\{1+\sum_{1\le i\le n} v_j, c\}$.
- 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. ∴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$\boxed{O(2^n)}$. ∴存在输入使算法最坏情形为$2^n$量级.
- 所以总时间不超过$O(\min\{2^n, n\sum_{1\le i\le n} v_i, nc\})$.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。

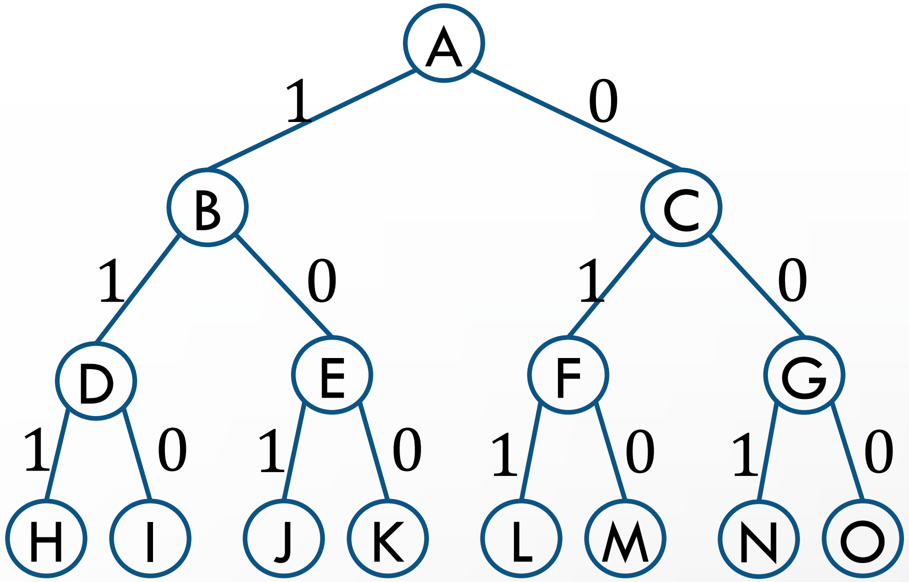
0/1背包问题8: 回溯法例

- 考察一个背包例子: n=4, c=7, p=[9, 10, 7, 4], w=[3, 5, 2, 1]. 这些对象的收益密度为[3, 2, 3.5, 4].
- 按密度排列为: (4,3,1,2), w'=[1,2,3,5], p'=[4,7,9,10]
- 限界方法1：cp+r<=bestp,则停止生成右子树。cp为当前已得到的效益值，r为尚未考虑的物品的效益值之和。
- 限界方法2：定义bound＝cp+对其余物品的贪心解效益值。如bound<＝bestp则停止产生右子树
- 展开的部分状态空间树见右图。
- 在结点5处得到bestp=20; 结点6处bound =19,故限界掉；类似,结点7, 8也被限界掉.



限界条件：bound<=bestp

0/1背包问题9: 分支限界法例



- n=3, w=(20, 15, 15), v=(40, 25, 25), c=30
- 优先队列：按价值率优先。

扩展结点	活结点	队列(可行结点)	可行解(叶结点)	解值
A	B,C	B→C		
B	D,E(D死结点)	E→C		
E	J,K(J死结点)	C	K	40
C	F,G	F→G		
F	L,M	G	L	50(最优)
G	N,O	Φ		

∴ 最优解为L, 即(0,1,1); 最优值为50

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 ($x_1, x_2, \ldots, x_n$) ($1\leq i \leq n$), 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：

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

贪心算法1：最小平均完成时间的任务调度问题

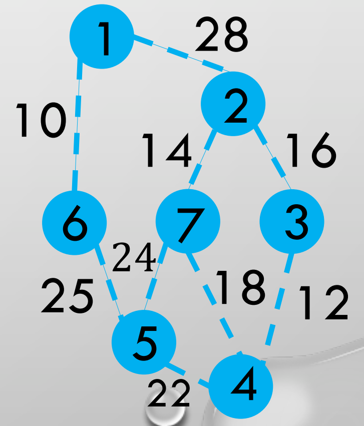
- 已知n个任务的执行序列。假设任务i需要t_i个时间单位。若任务完成的顺序为1, 2, ..., n, 则任务i完成的时间为$c_i = \sum_{j=1}^{i} t_j$。任务的平均完成时间(Averge Completion Time, ACT)为$\frac{1}{n}\sum_{i=1}^{n} c_i$.
  1） 考虑有四个任务的情况，每个任务所需时间分别是(4, 2, 8, 1)，若任务的顺序为1，2，3，4，则ACT是多少？
  2） 若任务顺序为2，1，4，3，则ACT是多少？
  3) 创建具有最小ACT的任务序列的贪心算法分n步来构造该任务序列，在每一步中，从剩下的任务里选择时间最小的任务。对于1)，利用这种策略获得的任务顺序为4,2,1,3，这种顺序的ACT是多少？写一个程序伪代码实现1)中的策略，程序的复杂性应为O(nlogn)，试证明之。
  4) 证明利用3)中的贪心算法获得的任务顺序具有最小的ACT。

贪心算法2：最小平均完成时间的任务调度问题

- 设在某任务顺序中，$i>j$而$t_i \le t_j$, 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'，则所得的调度的平均完成时间ACT'最小。
- 假设$i>j$, $t_i<t_j$(逆序)
- $nACT=(nt_1+\cdots+(n-j+1)t_j+\ldots+(n-i+1)t_i+\ldots)$
- $nACT'=(nt_1+\cdots+(n-j+1)t_i+\ldots+(n-i+1)t_j+\ldots)$
- $nACT - nACT' = (i-j)t_j-(i-j)t_i=(i-j)(t_j-t_i)>0$
- 所以, ACT'<ACT.
- 又：每消除一个逆序ACT值减小, 所以当无逆序时, 也即任务按执行时间从小到大排列时ACT值最小.

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) 即具有最大项点数目的完备子图。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

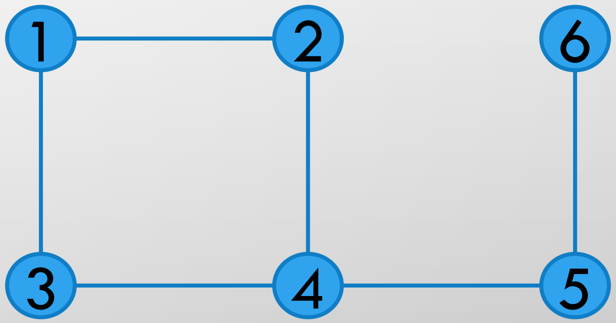
- 无向图的一个最大完全子图(包含意义下)称为一个集团, 集团的节点数称为集团的尺寸(size), 求图的最大集团. 这是NP难度问题. 本题要求给出一个贪心算法.
- 可考虑使用以下启发式: 如果一个节点的度数较大则它有可能在一个最大集团中.
- 算法首先将图的节点按度数排序然后从每个节点。

1. for j←1 to n do
2. { S←{v_j};
3.   for i←1 to n do {
4.   检查S∪{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？

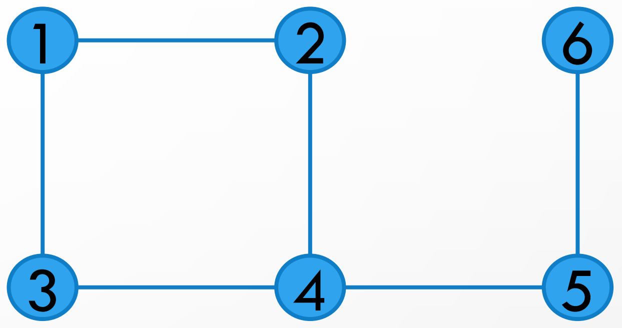


贪心算法6：无向图的着色方案

解 1) 令i为节点号, c为颜色标号; 算法的伪代码如右：

(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。

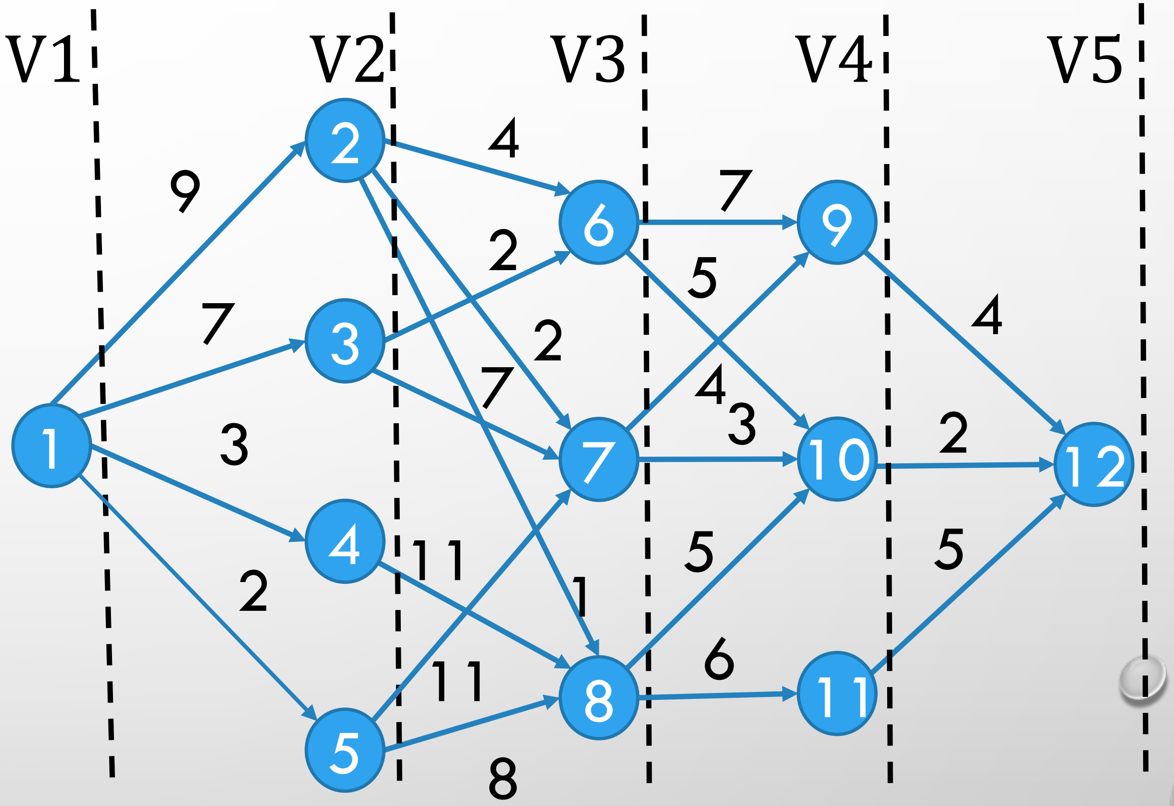
(3)否，该启发式算法不总是产生一个最优的着色方案。在上图中如果按1,5,4,2,3,6的顺序运行上述算法,则得到一个使用3种颜色 的着色方案。



1. for(i = 1; i≤n; i++)
2. for(c = 1; c≤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1：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。



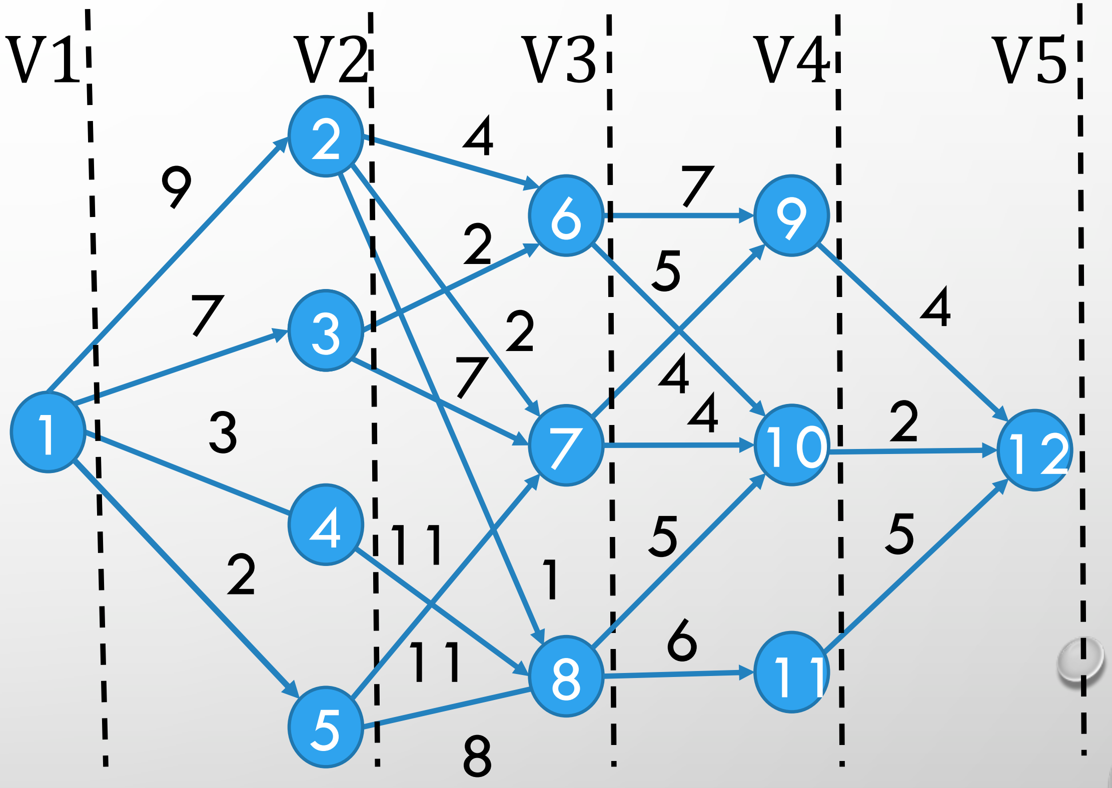
解: 设B(i)为i的前驱节点集合，有

$$d(1)=0;$$

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

动态规划2：多段图


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


V1
V2
V3
V4
V5
1
2
3
4
5
6
7
8
9
10
11
12
9
7
3
2
4
2
2
7
11
1
11
8
7
5
4
4
5
6
4
2
5

动态规划3：多段图

V1: d(1)=0

V2: d(2)=w(1,2)=9; d(3)=w(1,3)=7

d(4)=w(1,4)=3; d(5)=~~w(1,5)=2~~

V3: d(6)=min{~~9+w(2,6),~~ 7+w(3,6)}=9

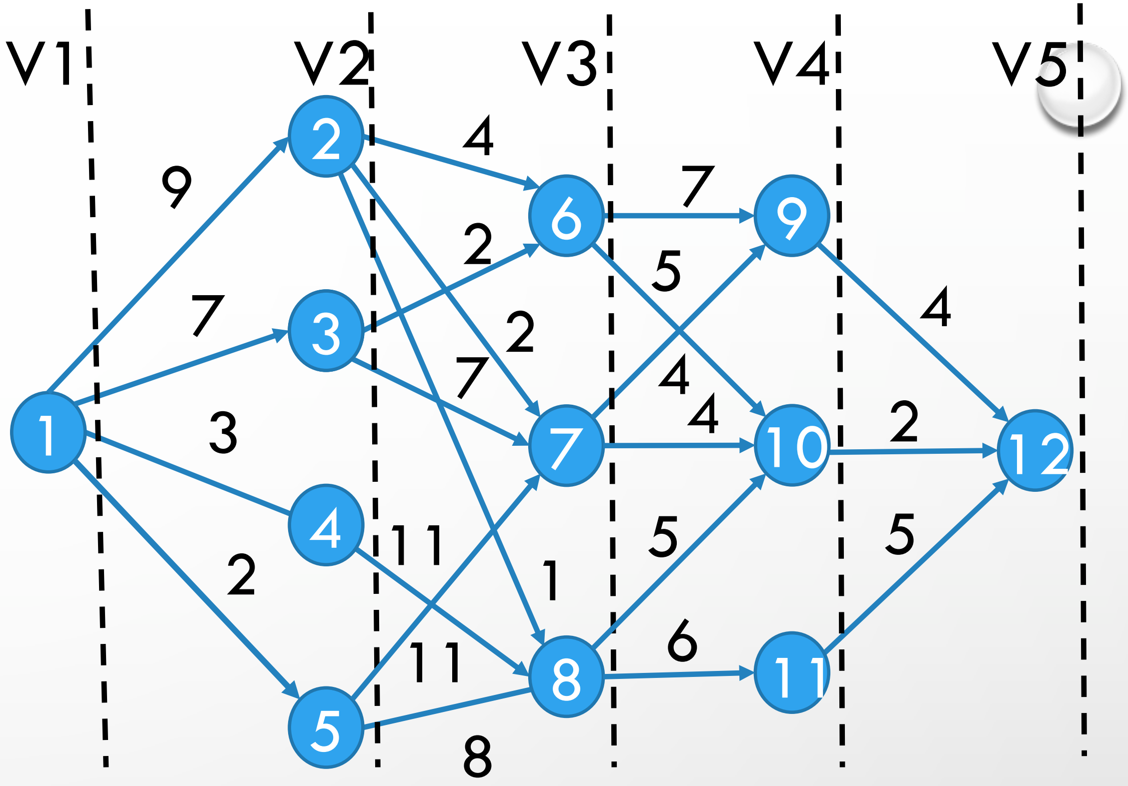
d(7)=min(9+w(2,7), 7+w(3,7),
~~2+w(5,7)~~)}=11

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

V4: d(9)=min{~~9+w(6,9),~~11+w(7,9)}=15

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

d(11)=10+w(8,11)=16



V5: d(12)=min{15+w(9,12),

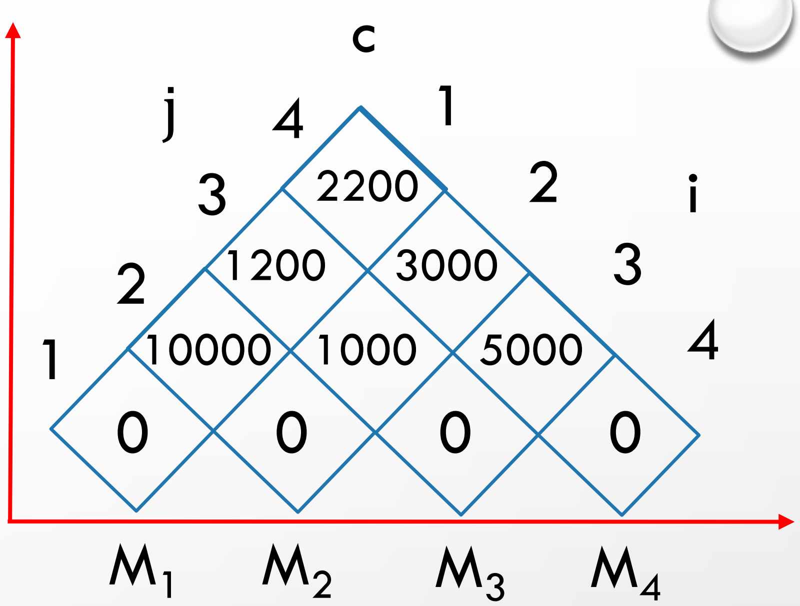
14+w(10,12), 16+w(11,12)}=16

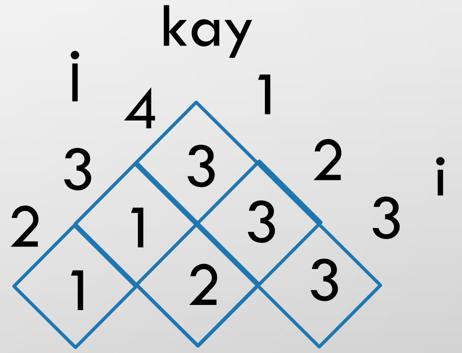
最优值为16;回溯最优解1, 3, 6, 10, 12.

动态规划4：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:

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





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

动态规划5：拓扑排序任务

- 假设一个项目的n个任务已按拓扑顺序排好, 编号为1到n; 任务1先执行, 接下来是任务2, 等等. 又假定每个任务有2中方式完成: 任务i按第一种方式需花费成本C(i,1), 时间T(i,1); 按第2种方式需成本C(i,2), 时间T(i,2).
- 令cost(i, j)=任务1到i能在j时间内完成的最小成本, 列出cost(i, j)满足的动态规划递归关系式。
- 设n＝3, T=(2,1,4; 3,2,1), C=(1,5,2; 2,3,4), t=8, 试计算cost(3,8)和优化的完成任务方案。

动态规划6：拓扑排序任务

约定如果在时间j内无法安排前i个任务，cost(i,j)=∞。令

$$cost(0, j)=\begin{cases}\infty & 当 j\le 0;\\ 0 & j>0.\end{cases}$$

不失一般性，假定T(1,1) <T(1,2)，我们有递归关系：

$$cost(1,j)=\begin{cases}\infty & j<T(1,1)\\ C(1,1) & T(1,1)\le j<T(1,2)\\ \min\{C(1,1), C(1,2)\} & T(1,2)\le j\end{cases}$$

则

$$cost(i,j)= \min\{cost(i-1,j-T(i,1))+C(i,1),$$
$$cost(i-1,j-T(i,2))+C(i,2)\} \quad i>0$$

动态规划7：拓扑排序任务

实例： n＝3, t=(2,1,4; 3,2,1), c=(1,5,2; 2,3,4), t=8，求cost(3, 8).

- cost(1,j)= ∞ j<2
- ＝1   j≥2
- cost(2,j)= ∞   j<3
- ＝6   3≤j<4
- ＝4   j≥4
- cost(3,j)= ∞ j<4
- cost(3,j)=10   4≤j<5
- cost(3,j)=8   5≤ j<8
- cost(3,j)=6   j≥8

动态规划8：子集和数问题

- 子集和数问题:设$S=\{s_1, s_2, \ldots, s_n\}$为n个正数的集合, 试找出满足以下条件的和数最大的子集J: $\sum_{i=1}^{n} s_i \ge c$, $\sum_{i \in J} s_i \le c$. c是任意给定的常数.该问题是NP-难度问题, 试用动态规划法设计一算法.
  1. 列出递归关系
  2. 举例说明算法的执行过程。

动态规划9：子集和数问题

- 解 （1）设$f(i, y)=\sum_{k\in J} s_k$, 其中J为{i, …, n}，约束为y的最大子集和数问题的解，则有以下的递归关系：

$$f(n, y)=\begin{cases} s_n & y \geq s_n \\ 0 & y < s_n \end{cases}$$

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

动态规划9：子集和数问题

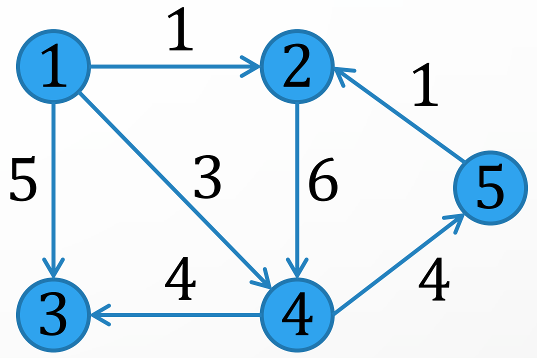
例 S = [20,18,15], C= 34，

- P(3)={(0,0)(15,15)}, s_2=18, Q2={(18,18),(33,33)}
- P(2)={(0,0),(15,15),(18,18),(33,33)}, s_1=20, Q1={(20,20)}
- P(1)={(0,0),(15,15),(18,18),(20,20),(33,33)}.

最优值33. 回溯：Q2中首次出现(33,33), X2=1. (33,33)-(18,18) =(15,15), 首次出现在P(3)中， X3=1. 最优解为(0,1,1).

动态规划10：最短路径问题

求DP法求右图中各点间的最短路



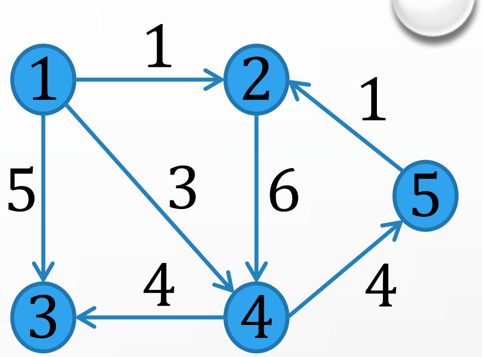
$$A=\begin{bmatrix}\infty & 1 & 5 & 3 & \infty\\ \infty & \infty & \infty & 6 & \infty\\ \infty & \infty & \infty & \infty & \infty\\ \infty & \infty & 4 & \infty & 4\\ \infty & 1 & \infty & \infty & \infty\end{bmatrix}$$

- 解：先写出有向图对应的矩阵A
- $c(k)=(c(i,j,k))=(\min\{c(i, j, k-1), c(i, k, k-1)+c(k, j,k-1)\}(i\neq j))$
- $k=0$时
- $k=1$时：$i\in\phi, j\in\{2,3,4\}$,

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

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

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



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

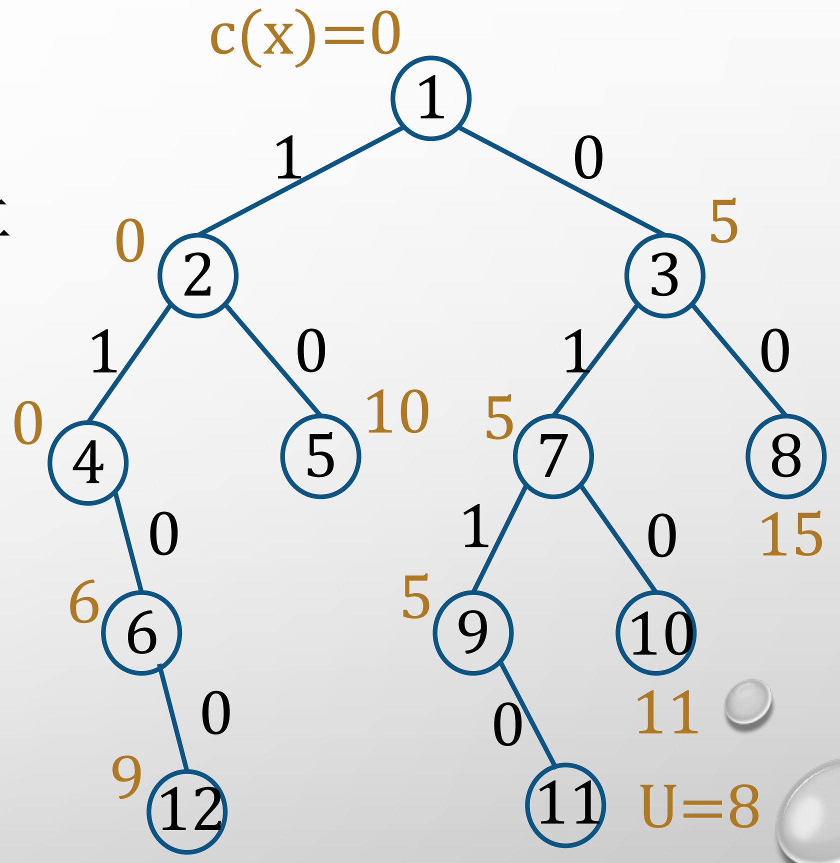
$$C^{(2)} = \begin{bmatrix} 0 & 1 & 5 & 3 & \infty \\ \infty & 0 & \infty & 6 & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & 4 & 0 & 4 \\ \infty & 1 & \infty & 7 & 0 \end{bmatrix} \quad C^{(3)} = \begin{bmatrix} 0 & 1 & 5 & 3 & \infty \\ \infty & 0 & \infty & 6 & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & 4 & 0 & 4 \\ \infty & 1 & \infty & 7 & 0 \end{bmatrix} \quad C^{(4)} = \begin{bmatrix} 0 & 1 & 5 & 3 & 7 \\ \infty & 0 & 10 & 6 & 10 \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & 4 & 0 & 4 \\ \infty & 1 & 11 & 7 & 0 \end{bmatrix}$$

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

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

分枝限界法1：最小罚款额作业调度

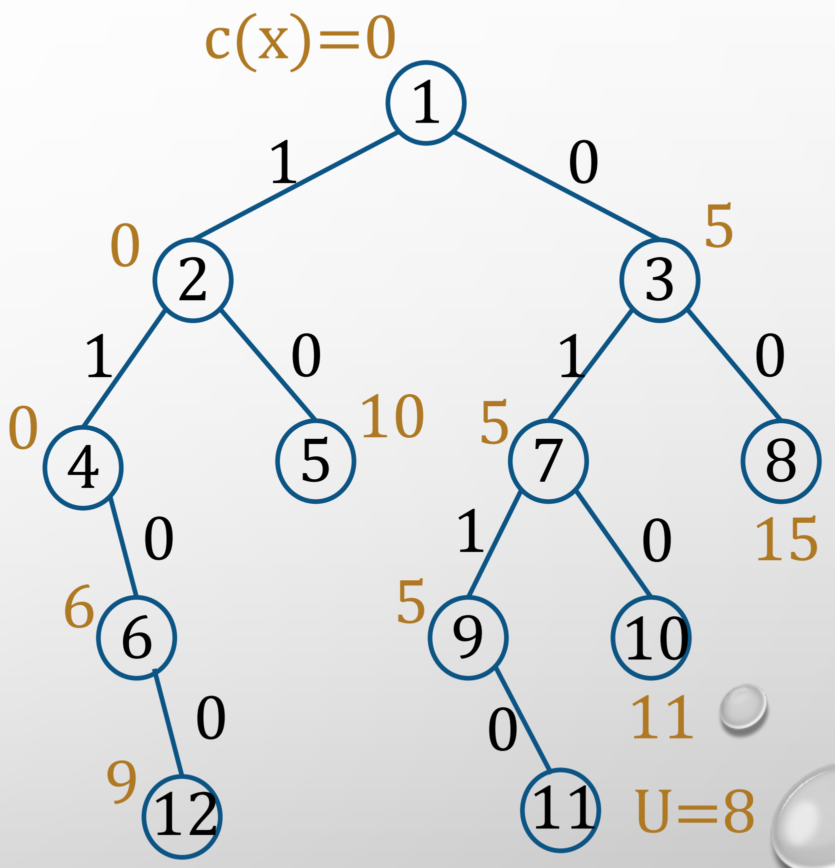
- 令三元组(p_i, d_i, t_i)表示一个作业的罚款额、截止期和执行时间. 试用LC-分枝限界法求解以下最小罚款额调度问题的实例: 4个作业的罚款额、截止期和执行时间分别为(5,1,1), (10,3,2), (6,2,1), (3,1,1)。
  1) 写出所使用的限界条件;
  2) 画出LC分枝-限界过程中展开的部分状态空间树，并给出优化解和优化值。



限界条件为$c(X) \geq U$. 优化解为(0,1,1,0), 优化值为8.

分枝限界法2：最小罚款额作业调度

1. 限界条件：设X=(x(1),...x(k))为状态空间树的节点，下界ĉ(X)估计为展开到x时已产生的罚款额$\sum_{1\le j\le k}(1-x(j))p_j$，令U为当前获得的最优成本值，则限界条件为ĉ(X)≥U；另一个限界条件是解的可行性，即，作业子集中的作业必须是可调度的。
2. 已知4个作业，表示它们的三元组(PI,DI,TI)分别为：(5,1,1), (10,3,2), (6,2,1), (3,1,1).。使用LC分枝－限界法得到的部分状态空间树为
3. 优化解为：（0，1，1，0），优化值为8。
4. 使用回溯法求解上述带截止期的调度问题，自己完成。



分治法-1

叙述分治法算法的思想并用归并排序算法说明。

- 把大问题分成两个或多个更小的子问题；
- 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。
- 把这些子问题的解组合成原始大问题的解。

分治法2：归并排序

Merge-Sort A[1..n]:

1. if n=1, done.
2. $p=\lfloor n/2 \rfloor$
3. Merge-Sort a[1..p]
4. Merge-Sort a[p+1..n]
5. merge the 2 sorted lists.

把大问题分成两个或多个更小的子问题；

分别解决每个子问题

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 $l[i]\le r[j]$
5. a[k]=l[i]
6. i=i+1
7. else a[k]=r[j]
8. j=j+1

分治3：快速排序-A

叙述快速排序算法的过程（最好用伪代码），并分析其最好最坏和平均时间复杂度；

```
template<class type>
void Quicksort (type a[], int p, int r)
{
    if (p<r) {
      int q=partition(a,p,r);
      Quicksort (a,p,q-1); //对左半段排序
      Quicksort (a,q+1,r); //对右半段排序
      }
}
```

```
void partition (int s[], int l, int r)  {
    if (l < r)   {
        int i = l, j = r, x = s[l];
        while (i < j)
        {   while(i < j && s[j] >= x)
                j--;
            if(i < j)
                s[i++] = s[j];
            while(i < j && s[i] < x)
                i++;
            if(i < j)
                s[j--] = s[i];
    }
```

分治法4：快速排序-B

- $\boxed{\text{最好最坏}}$时间复杂度
  - ➢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。

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

  - ➢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

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

分治法5：快速排序-C

- 平均时间复杂度
  - 定理：n个元素的快速排序的平均复杂度是Θ(nlogn).
  - 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

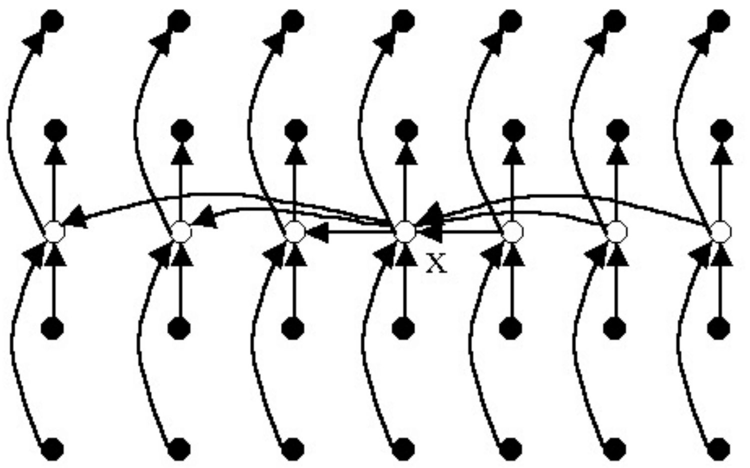
$$T(n)=\frac{1}{n}\sum_{s=0}^{n-1}[T(s)+T(n-s-1)]+cn=\frac{2}{n}\sum_{s=0}^{n-1}T(s)+cn$$

$$\begin{cases} nT(n)=2\sum_{s=0}^{n-1}T(s)+cn^2 \\ (n-1)T(n-1)=2\sum_{s=0}^{n-2}T(s)+c(n-1)^2 \end{cases} \Longrightarrow nT(n)-(n-1)T(n-1)=2T(n-1)+O(n)$$

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

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

分治法6:快排的支点选择-D



- 对于快速排序算法，试设计一种能够在O(n)时间内选择第k小元素作为支点的算法。
  1. 根据一个整数r，将数组a的n个元素分成⌊n/r⌋个组，每组都有r个元素(r≥5).
  2. 如果剩下不足r个元素，个数是n除以r的余数，它们不作为选择支点元素的候选。
  3. 对每组的r个元素进行排序，寻找在中间位置上的元素。
  4. 递归地使用选择算法，在所有的⌊n/r⌋个中间元素中选择中间元素作为支点元素。
- 证明当r=5，且a中所有元素都不同，那么当n≥24时，有max{| left |, | right | }≤3n/ 4。(略)

分治法7：金块问题

- 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？

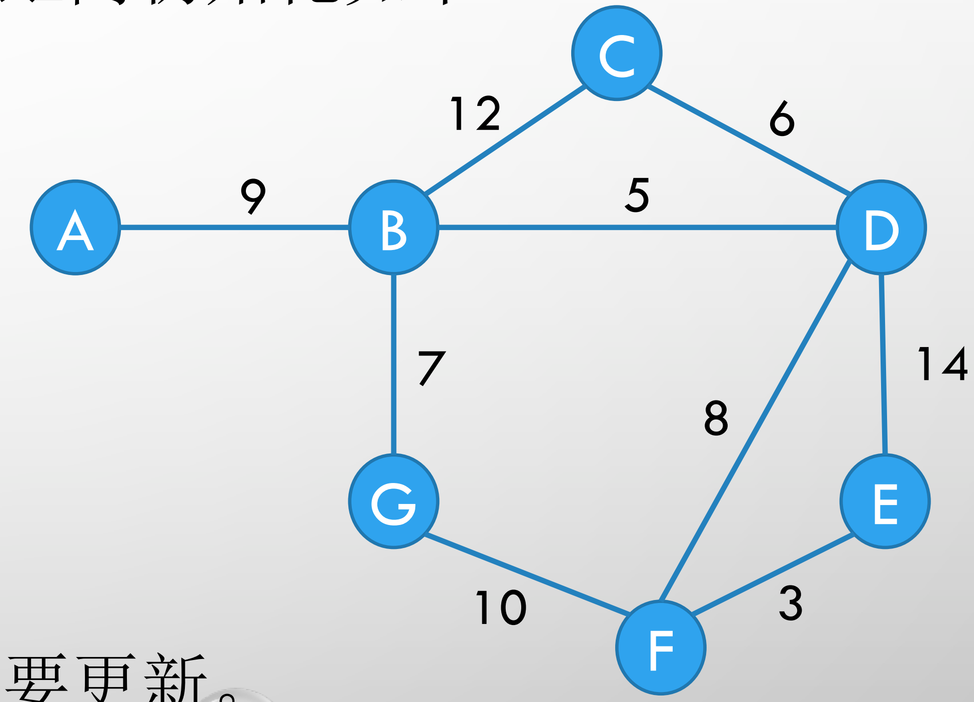
```
Max-Min(a[0,n-1], max, min)
if n=1 max←min←a[0], return;
if n=2 {if a[0]≥a[1] max←a[0], min←a[1]
        else max←a[1], min←a[0];}
   else  m←n/2
        max-min(a[0,m-1],max1,min1),
        max-min(a[m,n-1],max2,min2),
        max←max(max1,max2),
        min←min(min1,min2),
        return.
```

贪心算法1：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∅，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=∅,

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。

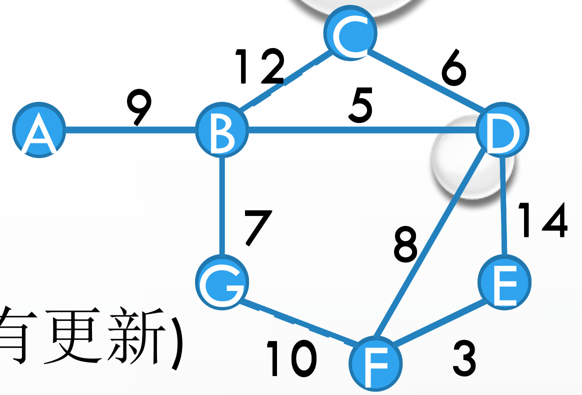
节点	A	B	C	D	E	F	G
离源点距离	0	9	∞	∞	∞	∞	∞

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。

贪心算法2：构造哈夫曼编码

➢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.

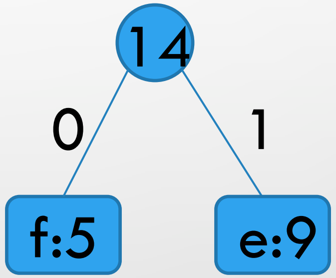
	a	b	c	d	e	f
频率	45	13	12	16	9	5
变长码	0	101	100	111	1101	1100

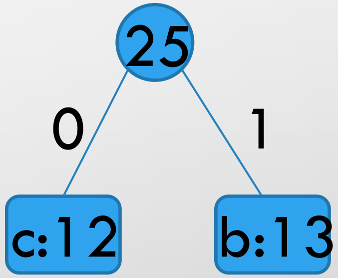
➢i=2, {a:45, b:13, c:12, d:16, z1:14}

c. freq=12最小， x2=c;

b. freq=13最小， y2=b;

z2.freq=12+13=25.





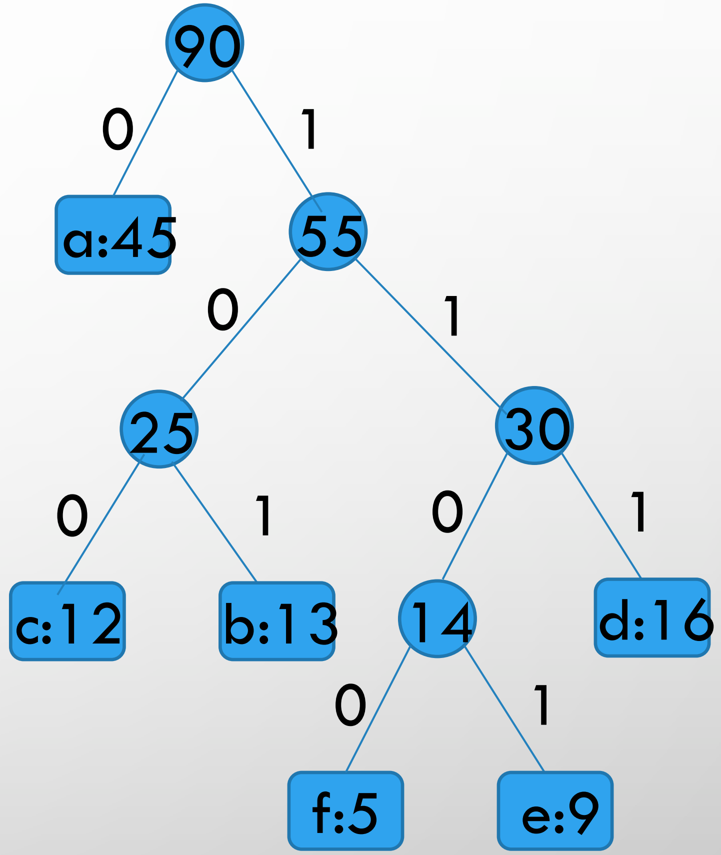
贪心算法3：构造哈夫曼编

	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.

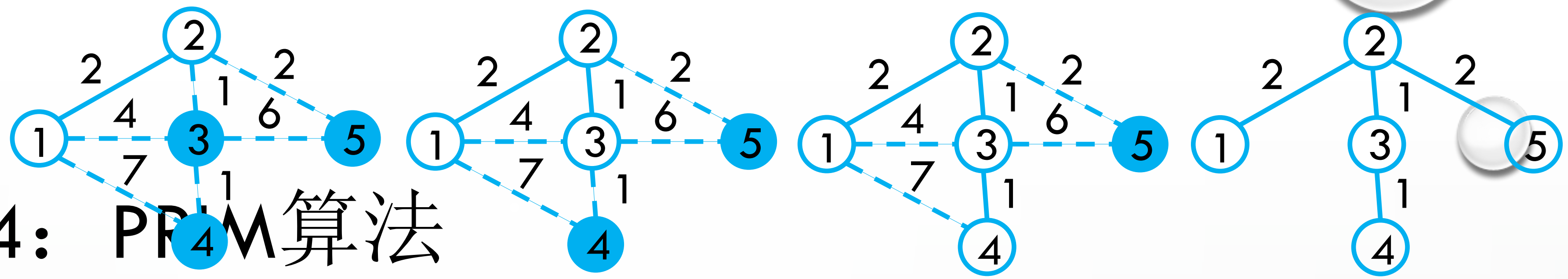


哈夫曼算法以自底向上的方式构造表示最优前缀码的二叉树T。

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。

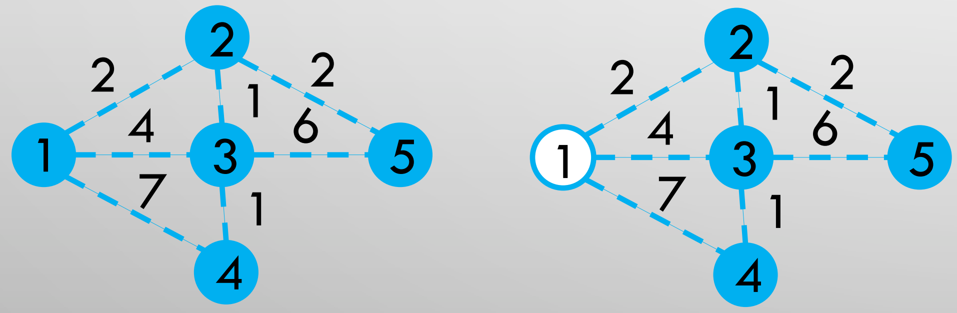
贪心算法4：PRIM算法



- 初始时所有的点都是蓝点, min[1]=0, min[2,3,4,5]=∞,权值之和mst=0.
- min[1]=0最小. 找到蓝点1. 将1变为白点, 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;

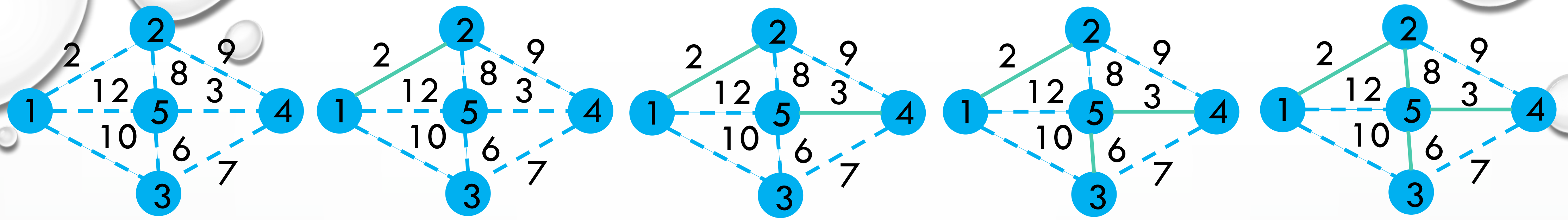
  min[4]=w(1,4)=7.



- min[2]最小. 找到蓝点2. 将2变为白点, 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. 找到蓝点3. 将3变为白点, 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min[5]的值.
- 最后两轮循环将点4,5以及边w(2,5), w(3,4)添加进最小生成树。
- 最后权值之和mst=6.



克鲁斯卡尔算法

- 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。而且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。

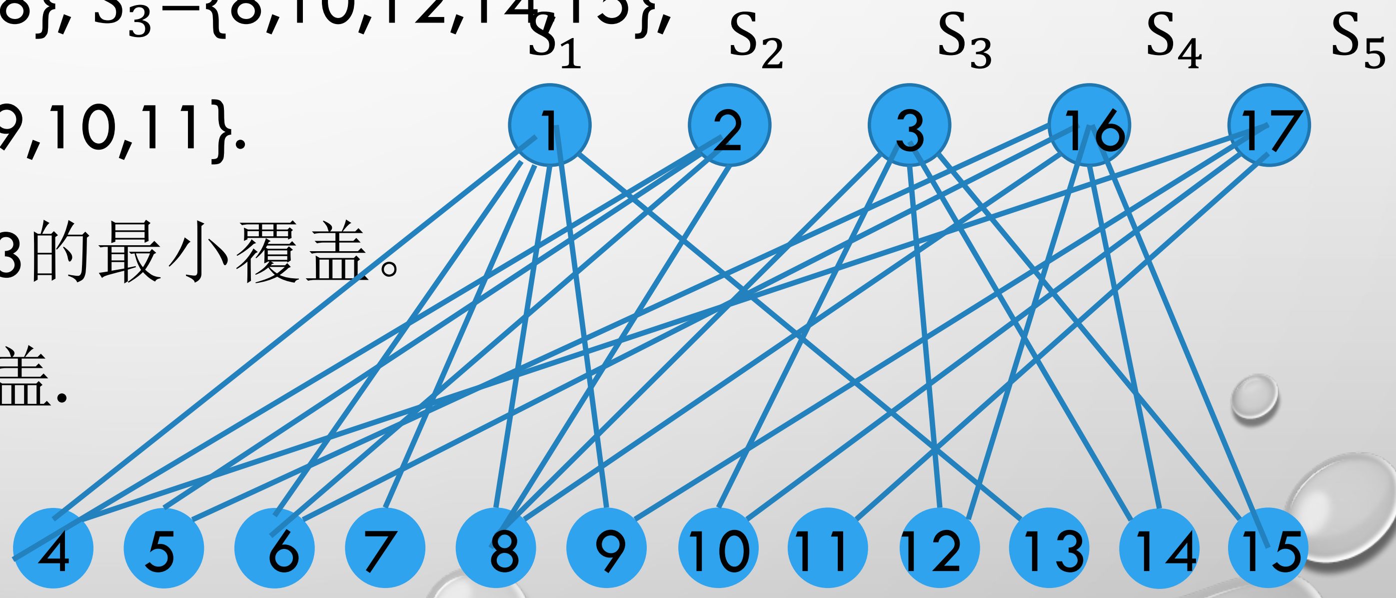
1. 选择边(1, 2)，将这条边加入到生成树中，并且将它的两个顶点1, 2合并成一个集合。现在有4个集合{(1, 2), {3}, {4}, {5}}, 生成树中有一条边(1, 2).
2. 选择边(4, 5)，将其加入到生成树中，并且将它的两个顶点4, 5合并成一个集合。现在有3个集合{{1, 2}, {3}, {4, 5}}, 生成树中有两条边(1,2), (4,5).
3. 选择边(3, 5}, 将这条边加入到生成树中，并且将它的两个顶点3, 5所在的两个集合合并成一个集合。现有2个集合{{1, 2},{3, 4, 5}}, 生成树中有3条边{(1, 2), (4, 5), (3, 5)}.
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.

拓扑排序的贪心算法

- 从空集开始，每步产生拓扑排序序列中的一个顶点V。顶点V需要满足贪心策略。
- 贪心策略：从当前尚不在拓扑排序序列的顶点中选择一顶点V, 其所有前驱节点U都在已产生的拓扑序列中(或无前驱顶点)，并将V加入到拓扑序列中.
- 用减节点入度的方法确定V:入度变成0的顶点为要加到拓扑序列中的顶点

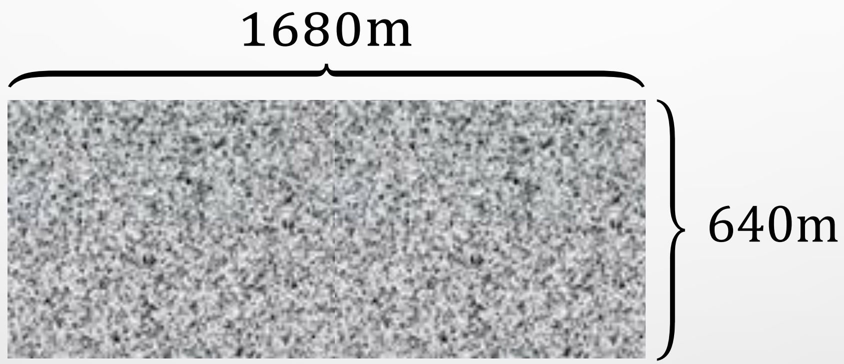
二分图最小覆盖例

- A={1,2,3,16,17}, B={4,5,6,7,8,9,10,11,12,13,14,15}=U.
- F={S_1, ..., S_5}满足$\cup_{I=1}^{5} S_I = U$
- S_1={4,6,7,8,9,13}, S_2={4,5,6,8}, S_3={8,10,12,14,15},
  S_4={5,6,8,12,14,15}, S_5={4,9,10,11}.
- S'={S_1, S_4, S_5}是一个大小为3的最小覆盖。
- 即{1,16,17}是偶图的最小覆盖.
- NP难问题！



实例1

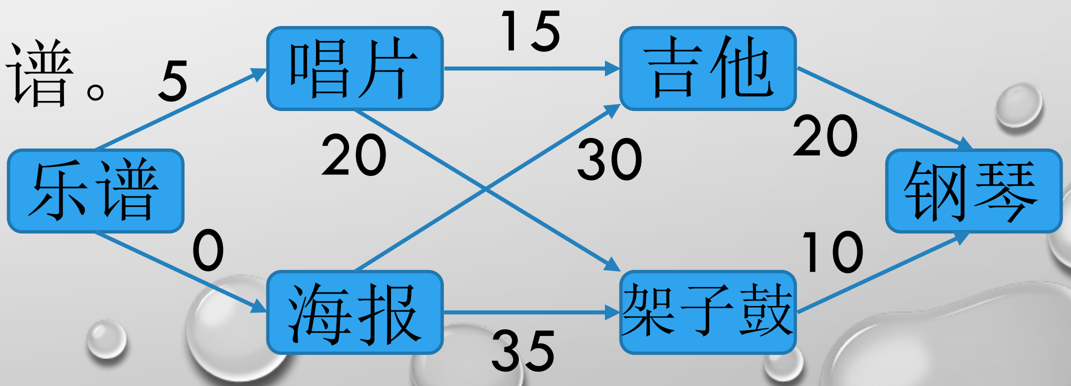
- 假设你是农场主，有一小块土地：



要将这块地均匀地分成方块，且分出的方块要尽可能大。

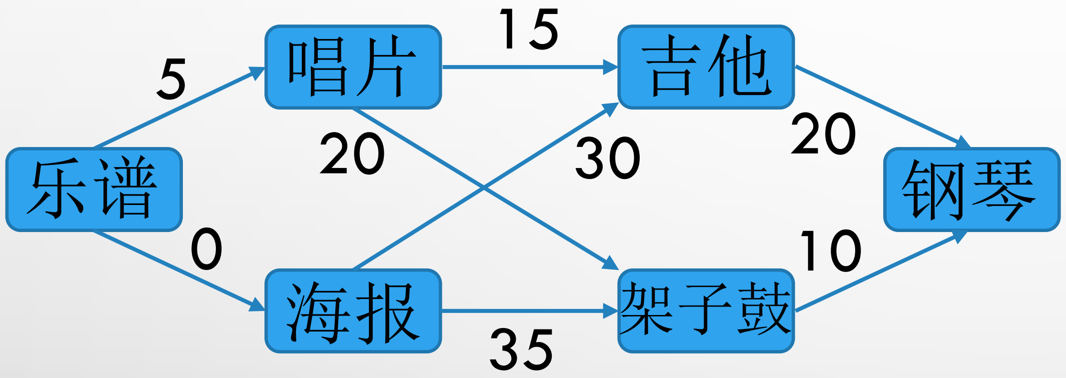
实例2

- Rama有一本乐谱，想用乐谱换钢琴。
- 经过调查发现，Beethoven有钢琴，但是他想换吉他或架子鼓。
- Amy有吉他和架子鼓，如果让她换，只能用她一直想要的一张黑胶唱片或者乐队Destroyer的海报。
- Alex有这张海报，他喜欢Rama的乐谱。
- Rama如何花最少的钱换到钢琴。



实例3

- 最短路，Dijkstra算法，源点为乐谱。



- 需要创建三个散列表，分别存储图、开销和父节点。还需要一个数组存储已经处理过的节点。

父节点	节点	开销
乐谱	唱片	5
	海报	0
海报	吉他	30
	架子鼓	35
唱片	吉他	20
	架子鼓	25
吉他	钢琴	40
架子鼓	钢琴	35