一、算法设计分析

算法设计分析.

1. 算法是由若干条指令组成的有穷序列.

(1) 输入

(2) 输出

(3) 确定性

(4) 有限性

2. 渐近复杂性

$T(n) = 3n^3 + 2n^2 + n + 1$

$\tilde{T}(n) = n^3$

3. 渐近记号 O

4. 套用公式法：求解如下形式的递归式的方法

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

分治法的时间复杂性所满足的递归关系，即一个规模为 n 的问题被分为规模均为 n/b 的 a 个子问题，递归地求解这 a 个子问题，然后通过对这 a 个子问题的解的综合，得到原问题的解。

$f(n)$：除了递归求解部分外，其它部分时间复杂性.

3种情况，都是 $f(n)$ 与 $n^{\log_b a}$ 作比较.

(1) 如果 $n^{\log_b a} > f(n)$，则 $T(n) = \theta(n^{\log_b a})$

(2) 如果 $n^{\log_b a} = f(n)$，则 $T(n) = \theta(n^{\log_b a} \cdot \log n)$ 或 $T(n) = \theta(f(n) \cdot \log n)$

(3) 如果 $n^{\log_b a} < f(n)$，则 $T(n) = f(n)$

例：$T(n)=9T(n/3)+n$　　谁大就取谁，相等无所谓.

$f(n)=n$，$a=9$　$b=3$，$n^{\log_b a}=n^2$，此时 $f(n)<n^{\log_b a}$

则 $T(n)=\theta(n^{\log_b a})=\theta(n^2)$

二、分治法

二、分治法

1. 分治法基本思想
2. 最大子段和问题
3. Strassen 矩阵乘法
4. 大整数的乘法
5. 线性时间选择
6. 循环赛日程表问题

1. 分治法的基本思想是将一个规模为n的问题分解为K个规模为较小的子问题，这些子问题互相独立且与原问题相同。递归地求解这些子问题，然后利用子问题的解合并出原问题的解。

(1) 分治算法的设计

①分解(Divide)：将整个问题划分为多个子问题

②递归求解(Conquer)：求解每个子问题

③合并(Combine)：合并子问题的解，形成原始问题的解。

2. 最大子段和问题

给定由n个整数组成的序列 $a_1、a_2 \dots a_n$，求该序列的子段和的最大值，当所有整数均为负整数时定义其最大子段和为 0.

例如当 $(a_1、a_2、a_3、a_4、a_5、a_6) = (-2, 11, -4, 13, -5, -2)$ 时，最大子段和为 20

(1) 设计：

如果将所给的序列 $a[1:n]$ 分为长度相等的两段 $a[1:n/2]$ 和 $a[n/2+1:n]$，分别求出这两段的最大子段和。

$a[1:n]$的最大子段和有3种情况：

① $a[1:n]$的最大子段和与$a[1:n/2]$的最大子段和相同.

② $a[1:n]$的最大子段和与$a[n/2+1:n]$的最大子段和相同

③ $a[1:n]$的最大子段和为$\sum_{k=i}^{j} a_k$，且$1\le i\le n/2$，$n/2+1\le j\le n$.

其中①和②两种可递归求得.

对于③，~~$a[n/2]$与$a[n/2+1]$~~ 可以在$a[1:n/2]$计算S_{1max}，在$a[n/2+1:n]$计算S_{2max}，

则S_1+S_2为③的最优解。

```
int MaxSubSum(int *a, int left, int right){
    int sum=0;
    if(left==right)
        sum=a[left]>0?a[left]:0;
    int center=(left+right)/2;
    int leftsum=MaxSubSum(a,left,center);
    int rightsum=MaxSubSum(a,center+1,right);
    int s1=0;  int lefts=0;
    for(int i=center; i>=left; i--){
        lefts+=a[i];
        if(lefts>s1)
            s1=lefts;
    }
    int s2=0;  int rights=0;
    for(int i=center+1; i<=right; i++){
        rights+=a[i];
        if(rights>s2)
            s2=rights;
    }
    sum=s1+s2;
    if(sum<leftsum)  sum=leftsum;
    if(sum<rightsum)  sum=rightsum;
    return sum;
}
```

(2) 算法时间复杂性分析

分治法三阶段.

本算法 Divide：$\theta(1)$

Conquer：$2T(n/2)$

Combine：$\theta(n)$

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

3. Strassen 矩阵乘法

$$T(n)=\begin{cases}\theta(1) & n=2\\ 7T(n/2)+\theta(n^2) & n>2\end{cases}$$

$$T(n)=\theta(n^{\log_2 7})=O(n^{2.81})$$

4. 大整数乘法

(1) 将每两个一位数的乘法或加法看作一步运算，计算 $X*Y$ 的时间复杂性为 $\theta(n^2)$，通过分治法为 $\theta(n^{\log_2 3})$　　$1011=10\times 2^2+11$　　$1234=12\times 10^2+34$

$X=$ | $\frac{n}{2}$位 | $\frac{n}{2}$位 |　　$Y=$ | $\frac{n}{2}$位 | $\frac{n}{2}$位 |

| A | B |　　| C | D |

$$XY=(A2^{\frac{n}{2}}+B)(C2^{\frac{n}{2}}+D)$$

$$=AC2^n+(AD+BC)2^{\frac{n}{2}}+BD \quad \text{4次乘法}$$

改进 $=AC2^n+((A-B)(D-C)+AC+BD)2^{\frac{n}{2}}+BD$　3次乘法

算法：

1. 计算 A-B 和 D-C
2. 计算 n/2 位乘法 AC、BD、(A-B)(D-C)；
3. 计算 (A-B)(D-C)+AC+BD
4. AC 左移 n 位，((A-B)(D-C)+AC+BD) 左移 n/2 位，计算 XY

(2)分析

划分：$\theta(1)$

递归：$3T(n/2)$

合并：$\theta(n)$

$$T_n = \begin{cases} \theta(1) & n=1 \\ 3T(\frac{n}{2})+\theta(n) & n>1 \end{cases}$$

用套用公式法求此递归方程解的渐进阶得：

$$T(n)=\theta(n^{\log_2 3})=O(n^{1.59})$$

5、线性时间选择

给定线性序集中的n个元素和一个整数k（$1\le k\le n$），要求找出这n个元素中第k小的元素。

设数据在a数组中，其左界为p，右界为r，算法如下：

```
Type Select(Type a[], int p, int r, int k){
    if(r-p < 75)
```

6、循环赛日程表

设有 $n=2^k$ 个运动员要进行网球循环赛，现设计一个满足以下要求的比赛日程表：

① 每个选手必须与其它 n-1 个选手各赛一次

② 每个选手一天只能赛一次

③ 循环赛一共进行 n-1 天

按此要求，可将比赛日程表设计成有 n 行和 n-1 列的一个表，在表中第 i 行和第 j 列处填入第 i 个选手在第 j 天所遇到的选手。

选手\天数	1	2	3
1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

(1)设计

将所有选手对分为两组，n个选手的比赛日程表就可通过 n/2 个选手设计的比赛日程决定，递归地用这种一分为二的策略对选手进行分割，直到只剩下2个选手时，只让这两个选手进行比赛就可以了。

```
void fun(int n){
    int i, j;
    if(n<=0) return;
    if(n>2){
        fun(n/2);
        for(i=1; i<= n/2; i++)
            for(j=n/2+1; j<=n; j++)
                matrix[i][j] = matrix[i][j-n/2] + n/2;

        for(i=n/2+1; i<=n; i++)
            for(j=1; j<= n/2; j++)
                matrix[i][j] = matrix[i-n/2][j+n/2];

        for(i=n/2+1; i<=n; i++)
            for(j=n/2+1; j<=n; j++)
                matrix[i][j] = matrix[i-n/2][j-n/2];
```

三、动态规划法

三、动态规划法

1.基本思想

2.最长公共子序列问题

3.矩阵连乘最佳计算次序问题

4.最大子段和问题

5. 0-1 背包问题

1.基本思想

将要求解的问题一层一层地分解成一级一级、规模逐渐缩小的子问题，直到可以直接求解其解的子问题为止。所有子问题按层次关系构成一颗子问题树。树根是原问题，原问题的解依赖于子问题树中所有子问题的解.

子问题往往不是相互独立的.

设计一个动态规划算法步骤：

(1) 分析最优解的性质，并刻画其结构特征.

(2) 递归地定义最优值（每个解都有一个值）

(3) 根据递归方程分解子问题，直到不能分为止。

(4) 自底向上的方式计算最优值，并记录构造最优解的所需信息

(5) 根据计算最优值时得到的信息，构造一个最优解.

2、最长公共子序列

LCS长度递归方程：

$$c[i][j]=\begin{cases}0 & 当 i=0 或 j=0 \\ c[i-1][j-1]+1 & i,j>0 且 x_i=y_i \\ \max(c[i][j-1],\ c[i-1][j]) & i,j>0，且 x_i\neq y_i\end{cases}$$

(1)构造LCS

基本思想：

1、从b[m][n]开始按指针搜索

2、若b[i][j]="↖"，则 $x_i=y_j$ 是LCS的一个元素

3、如此找到的序列是X与Y的LCS的反序。

$T(n)=O(mn)$

3、矩阵连乘最佳计算次序问题

4、最大子段和。

$$b[j]=\begin{cases}0 & j=0 \\ \max\{b[j-1]+a[j],\ a[j]\} & 1\le j\le n\end{cases}$$

```
int besti=0 , bestj=0
int MaxSum (int n, int *a)
{  int sum=0, b=0;
   for(int j=1; j<=n; j++)
   { if(b>0)  b+=a[j];
     else{
          b=a[j];
          i=j;
     }
   if(b>sum){
      sum=b;
      besti=i;
      bestj=j;
   }
return sum;
```

四、贪心法

四. 贪心法

1. 贪心基本思想
2. 活动安排问题
3. 最优装载问题
4. 背包问题

1. 贪心基本思想

求解组合(最)优化问题的贪心算法。

每一步都在一组选择中做出在当前看来最好的选择，希望通过做出局部优化选择达到全局优化选择。但贪心算法不一定总产生优化解，所以一个贪心算法是否产生优化解，需要严格证明。

贪心法求解的问题必须具有最优子结构和贪心选择性

贪心无法解决 01 背包问题

2. 活动安排问题（选择、调度问题）

(1) 算法设计

为了选择最多的相容活动，每次选 f_i 最小的活动，使剩余的可安排时间极大化，以便接纳尽可能多的相容活动。

结束时间

(2) 算法描述

```
void GreedySelector(int n, Type s[], Type f[], bool A[]){
    A[1] = true;          选择活动1
    int j = 1;            j用来记录最近一次加入到A中的活动
    for(int i=2; i<=n; i++){
        if(s[i] >= f[j]){     找到一个相容活动
            A[i] = true;      选择活动i
            j = i;
        }
        else
            A[i] = false;     活动i不相容，不选择活动i
    }
}
```

(2) 复杂度分析

当活动按结束时间已经排序 $T(n)=\theta(n)$

当活动按结束时间未排序 $T(n)=\theta(n)+\theta(n\log n)=\theta(n\log n)$

3. 最优装载问题

(1) 思想:

采用重量最轻者先装的贪心选择策略。由此可产生装载问题的最优解.

(2) 描述

假设集装箱已按重量递增的次序排序.

```
void Loading(int x[], Type w[], Type c, int n){
    for(int i=1; i<=n; i++)      数组元素x[i]=0 表示不装入集装箱i
        x[i]=0;
    for(int i=1; i<=n && w[i]<=c; i++){
        x[i]=1;
        c-=w[i];
    }
}
```

4. 背包问题

(1) 算法

选择单位重量价值 vi/wi 最大的物品。每次从物品集合中选择单位重量价值最大的物品，如果其重量小于背包容量，就可以把它装入，并将背包价值增加该物品的价值，同时背包容量减去该物品的重量。

(2)

```
void Knapsack(int n, float c, float v[], float w[], float x[], float &value){
    // 假设已将各种物品依其单位重量的价值 vi/wi 从大到小排序
    float value=0;                          最大价值
    for(int i=1; i<=n; i++)     x[i]=0;     初始化 x为零向量
    for(int i=1; i<=n && w[i]<=c ; i++){    物品 i 能够全部装入时循环
        x[i]=1;                             装入物品 i
        c -=w[i];                           减少背包中能装入的余下重量
        value += v[i];                      累计总价值
    }
    if(i<=n){                               i<=n / c>0
        x[i]= c/w[i];                       将物品 i 的一部分装入
        value += x[i] * v[i];               累计总价值.
    }
}
```

五、回溯法

五、回溯法

1. 基本思想
2. N后问题
3. 图的M着色问题
4. 批处理作业调度.

1.基本思想

(1) 回溯法是一个既带有系统性又带有跳跃性的搜索法。它在包含问题所有解的解空间树中,按照深度优先的策略,从根出发进行搜索。搜索每到达解空间树的一个结点,总是先判断以该结点为根的子树是否肯定不包含问题的解。

如果肯定不包含,则跳过对该子树的系统搜索,一层一层地向它的祖先回溯,直到遇上一个还有未被搜索过的儿子结点,才转向该结点的一个未曾搜索过的儿子结点,继续搜索,否则进入该子树,继续按深度优先的策略进行搜索。

回溯法在用来求问题的所有解(或最优解)时,要回溯到根,且根的所有儿子都已被搜索过才结束;而在用来求问题的任一解时,只要搜索到问题的一个解就可结束.

(2) 搜索解空间树

可行解、最优解.

(3) 子集树与排列树

当所给的问题是从n个元素的集合S中找出满足某种性质的子集时,相应的解空间树称为子集树。

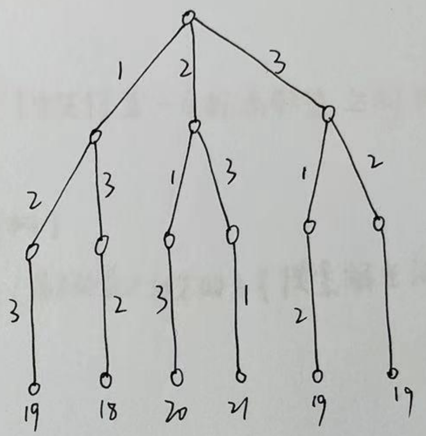
```
void Backtrack (int t)
{ if(t>n)
      Output(x);      //已搜索至树叶
  else
    for(int i=0; i<=n; i++)
    { x[t]=i          //在当前扩展结点处 x[t] 取值可选 i
      if( Constraint (t) && Bound(t) )
            Backtrack(t+1);
    }
}
```

2、N后问题

n后问题等价于：任何两个皇后不能在同行、同列、同一斜线上

3. 批处理作业调度

tji	机器1	机器2
作业1	2	1
作业2	3	1
作业3	2	3



这三作业的6种调度方案：

1,2,3　　1,3,2　　2,1,3

2,3,1　　3,1,2　　3,2,1

其完成时间：

19(3+6+10)　　18(3+7+8)　　20(4+6+10)

19(5+6+8)　　19(5+6+8)

最优解 (1,3,2)　　最优值18.

六、分支限界法

六、分支限界法.

1.基本思想

2.单源最短路径问题

1. 分支限界法的求解目标则是找出T中使得某一目标函数值达到极小或极大的解,即问题在某种意义下的最优解.

(1) 回溯法以深度优先搜索解空间树T

分支限界法以广度优先或最小耗费(最大效益)优先的方式搜索解空间树T.

(1) 队列式分支限界法

(2) 优先队列分支限界法.

↓

大根堆/小根堆