简答题、应用题、算法设计题

一、

**时间、空间复杂性：**

当给出合法输入时，为了得到输出，该算法所需要的时间和空间

算法复杂度分为时间复杂度和空间复杂度。其作用： 时间复杂度是度量算法执行的时间长短；而空间复杂度是度量算法所需存储空间的大小。

**动态规划算法思想，可能与分治算法思想比较**

动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干子问题，先求解子问题，然后从这些子问题的解得到原问题的解。与分治法不同的是，适合于用动态规划法求解的问题，经分解得到的子问题往往不是相互独立的。若用分治法来解这类问题，则分解得到的子问题数目太多，以至于最后解决原问题需要耗费指数时间。然而，不同子问题的数目常常有多项式量级。在用分治法求解时，有些子问题被重复计算了许多次。如果我们能够保存已经解决的子问题的答案，而在需要 时再找出已求解的答案，这样就可以避免大量重复计算，从而得到多项式的时间算法

**动态规划（Dynamic planning）**

通过组合子问题的解而解决整个问题。

将待求解问题分解成若干个子问题，先求解子问题的解，然后从子问题的解得到原问题的解。适用于子问题不独立的情况，即各子问题包含公共的子问题。保存已求解子问题的答案，在需要时再找出以求得的答案，可以避免大量重复计算。步骤：1、找出最优解的性质，刻画其结构特征2、递归的定义最优值3、以自底向上的方式计算最优值4、根据计算最优值时得到的信息，构造最优解

**贪心算法（greedy algorithm）**

贪心算法总是作出在当前看来最好的选择。并不从整体最优考虑，它所作出的选择只是在某种意义上的局部最优选择。虽然贪心算法不能对所有问题都得到整体最优解，但对许多问题它能产生整体最优解。

**分治法(Divide and Conquer)**

将一个规模为n的问题分解为k个规模较小的子问题，这些子问题互相独立且与原问题相同。对这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。分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之

**随机算法：**

分为两类，第一类称为Las Vegas算法，它建立的那些随机算法总是或者给出正确的解，或者无解。而另一类Monte Carlo算法与之相反，总是给出解，但是偶尔可能会产生非正确的解。然而，可以通过多次运行原算法，并且满足每次运行时的随机选择都相互独立，使产生非正确解的概率可以减到任意小。

**近似算法评价标准：**

近似算法的性能分析包括时间复杂度分析、空间复杂度分析和近似精度分析，其中时间（空间）复杂度的分析同精确复杂度相同。近似精度分析是近似算法特有的，它主要用于刻画近似算法给出的近似解相比于问题优化解的优劣程度。目前，存在三种刻画近似精度的度量，即近似比、相对误差界和1+ε近似。

**递推关系，求时间空间复杂度**

**公式推导**

****

**矩阵乘积、背包问题、最小生成树**

**三、**

**6.50.设计一个分治算法，判定两棵给定的二叉树T1和T2是否相同**

int EqualBtr(bitreptr t1, bitreptr t2)

{

if(t1 == NULL && t2 == NULL)

return 1;

if((t1==NULL&&t2!=NULL)||t1!=NULL&&t2==NULL)||

(t1->data!=t2->data))

return 0;

hl = EqualBtr (t1->Lchild, t2->Lchild);

hr = EqualBtr (t1->Rchild, t2->Rchild);

if(hl==1&&hr==1)

return 1;

else

return 0;

}

**6.51：设计一个分治算法，求给定二叉树的高度**

void Btdepth(bitreptr t, int k, int &h){

//指针t指向二叉树的根结点，k, h的初值均设为0

if (t! = NULL)

{

k++; //表示结点的层次

if (k>h)

h = k;

Btdepth(t->Lchild,k,h);

Btdepth(t->Rchild,k,h);

}

}

**13.9:设计一个回溯算法，生成1,2,3,…,n的全排列**

输入：数字1,2,3,...,n

输出：全排列

1.for k=1 to n

2. c[k] 0

3.end for

4.k 1

5.while k>=1

6. While c[k]<= n-1

7 c[k] c[k]+1

8 if c为合法着色 then output c 退出两层while循环

9 else if c为部分解 then k k+1

10 end while

11.c[k] 0

12.k k-1 {回溯}

13.end while

**14.13.判断AB=C**

Public static boolean product(double [][]a,double [][]b,

double [][]c,int n){

cnd=new Random();

double []x=new double [n+1];

double []y=new double [n+1];

double []z=new double [n+1];

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

x[i]=cnd.random(2);

if(x[i]==0) x[i]=-1;

}

mult(b,x,y,n);

mult(a,y,z,n);

mult(c,x,y,n);

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

if(y[i]!=z[i]) return false;

else

return ture;

}

}

**14.14.AB=1**

Public static boolean product(double [][]a,double [][]b,int n){

cnd=new Random();

double []x=new double [n+1];

double []y=new double [n+1];

double []z=new double [n+1];

Double []e=new double [n+1];

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

x[i]=cnd.random(2);

if(x[i]==0) x[i]=-1;

}

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

for(int j=1;j<=n;j++)

if(i==j) e[i][j]=1

else e[i][j]=0;

}

mult(b,x,y,n);

mult(a,y,z,n);

mult(e,x,y,n);

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

if(y[i]!=z[i]) return false;

else

return ture;

}

}