# 算法重点

1. 基础知识

1.程序=算法+数据结构

2.算法是解决某一特定问题的一组有穷指令的序列

3.常见的时间复杂度：

O(n)：遍历、扫描全部输入

O(nlogn)：许多分治算法

O(n2 )：两层循环

O(n3) ：三层循环

O(2n ) ：一个集合的所有子集

O(n!) ：一个集合中的元素的所有组合

押题：

1.算法的复杂性有 时间 复杂性和 空间 复杂性之分。

2.衡量一个算法好坏的标准是（ C ）。  
A 运行速度快 B 占用空间少 C 时间复杂度低 D 代码短

3.算法的五个重要特性

确定性、可行性、输入、输出、有限性

22.算法是由若干条指令组成的有穷序列，且要满足输入、 输出 、确定性和 有限性 四条性质。

1. 递归
2. 递归：直接递归和间接递归
3. 折半查找

int bsearch(int b[],int x,int L,int R)

{

int mid;

if(L>R) return (-1);

mid=(L+R)/2;

if(x==b[mid])

return mid;//正好位于中间位置找到

else if(x<b[mid])

return bsearch(b,x,L,mid-1);//小于中间数，后一半就不用看了

else

return bsearch(b,x,mid+1,R);//大于，舍前一半

}

1. 一般地,一个递归模型是由递归边界和递归体两部分组成,前者确定递归到何时结束,后者确定递归求解时的递推关系。
2. Fabanoci（斐波那契数列）

long Fib(int n)

{

if(n==0||n==1) return n;

else return Fib(n-1)+Fib(n-2);//当前数等于之前两个数的和

}

时间复杂度O(n)=2n

1. 汉诺塔基本思想：n个盘子的汉诺塔问题可递归表示为，首先把上边的n-1个盘子从A柱借助C移到B柱，然后把最下边的一个盘子从A柱移到C柱，最后把移到B柱的n-1个盘子再借助A移到C柱。

void Hanoi(int n,char a,char b,char c)

{

if(n==1)

printf("\t将第%d个盘片从%c移动到%c\n",n,a,c);

else

{

Hanoi(n-1,a,c,b);//把n-1个盘从A借助C移动到B

printf("\t将第%d个盘片从%c移动到%c\n",n,a,c)；

Hanoi(n-1,b,a,c);//把n-1个盘片从B借助A移动到C

}

}

时间复杂度O(n)=2n  
6. 辗转相除法

int GCD(int m,int n)

{

if(n==0) return m;

else return GCD(n,m mod n);

}

1. 递归查找线性表最大元素

int max(int a[],int n)

{

int m;

if(n==0) return a[n];

else{

m=max(a,n-1);

if m>a[n]

return m;

else return a[n]

}

}

押题：

9. 实现循环赛日程表利用的算法是（ A ）。

A、分治策略 B、动态规划法 C、贪心法 D、回溯法

1. 分治算法
2. 分治法的三个步骤：划分、解决、合并。
3. 分治法能解决的问题具有最优子结构性质。各个子问题最好独立，否则最好用动态规划。
4. 分治法体现了一种平衡的思想
5. 合并排序

void MergeSort(Type a[],int left,int right)

{

if(left<right)//两个元素才可开始执行

{

int i=(left+right)/2;//找出中点

mergeSort(a,left,i);

mergeSort(a,i+1,right);

merge(a,b,left,i,right);//合并

copy(a,b,left,right);//复制

}

}

时间复杂度O(nlog2n)

1. 快速排序

Quicksort(A[l...r])

//用Quicksort对子数组排序

//输入：数组A[0..n-1]中的子数组

//输出：非降序的子数组A[l...r]

if (l<r){

s  Partition(A[l...r]);

Quicksort(A[l...s-1]);

Quicksort(A[s+1...r]);

}

//以第一个元素作为中轴，划分数组

//输入：数组A[l..r]，l,r为左右下标

//输出：A[l..r]的一个分区，返回划分点位置

pA[l];

il; jr+1;

reapeat

reapeat jj-1 until A[j] ≤p；

reapeat ii+1 until A[i] ≥p；

swap(A[i],A[j]);

until i ≥ j

swap(A[l],A[j]);

return j;

时间复杂度O(n2)

押题：

1. 给定已按升序排好序的n个元素a[0:n-1]，现要在这n个元素中找出一特定元素x，返回其在数组中的位置，如果未找到返回-1。

写出二分搜索的算法，并分析其时间复杂度。

template<class Type>

int BinarySearch(Type a[], const Type& x, int n)

{//在a[0:n]中搜索x，找到x时返回其在数组中的位置，否则返回-1

Int left=0; int right=n-1;

While (left<=right){

int middle=(left+right)/2;

if (x==a[middle]) return middle;

if (x>a[middle]) left=middle+1;

else right=middle-1;

}

Return -1;

}

时间复杂性为O(log2n)

2. 利用分治算法写出合并排序的算法，并分析其时间复杂度

1. void MergeSort(Type a[], int left, int right)

{

if (left<right) {//至少有2个元素

int i=(left+right)/2; //取中点

mergeSort(a, left, i);

mergeSort(a, i+1, right);

merge(a, b, left, i, right); //合并到数组b

copy(a, b, left, right); //复制回数组a

}

}

算法在最坏情况下的时间复杂度为O(nlogn)。

1、二分搜索算法是利用（ A ）实现的算法。

A、分治策略   B、动态规划法   C、贪心法    D、回溯法

29、使用分治法求解不需要满足的条件是（ A ）。  
A 子问题必须是一样的  
B 子问题不能够重复  
C 子问题的解可以合并  
D 原问题和子问题使用相同的方法解

34．实现合并排序利用的算法是（ A ）。

A、分治策略 B、动态规划法 C、贪心法 D、回溯法

41．实现大整数的乘法是利用的算法（ C ）。

A、贪心法 B、动态规划法 C、分治策略 D、回溯法

52. 一个问题可用动态规划算法或贪心算法求解的关键特征是问题的（ B ）。

A、重叠子问题 B、最优子结构性质 C、贪心选择性质 D、定义最优解

55. 实现最长公共子序列利用的算法是（ B ）。

A、分治策略 B、动态规划法 C、贪心法 D、回溯法

4.矩阵连乘问题的算法可由 动态规划 设计实现。

7、从分治法的一般设计模式可以看出，用它设计出的程序一般是 递归算法 。

8、问题的 最优子结构性质 是该问题可用动态规划算法或贪心算法求解的关键特征。

21. 动态规划算法的基本思想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。

1. 大整数乘积算法是用 分治法 来设计的。

27.快速排序算法是基于 分治策略 的一种排序算法。

35.快速排序算法的性能取决于 划分的对称性 。

1. 贪心算法

1.贪心策略：在每个局部阶段，都做出一个看上去最优的决策(即某种意义下的、或某个标准下的局部最优解)，并期望通过每次所做的局部最优选择产生出一个全局最优解。

2.贪心算法设计要素：最优子结构性质和贪心选择性质

3.背包问题可以用贪心算法求解，而0-1背包问题却不能用贪心算法求解。

4.背包问题贪心解法：先装满收益最高的，然后次高的，依次到装不下。

5.最优装载问题：有一批集装箱要装上一艘载重量为c的轮船。其中集装箱i的重量为wi。最优装载问题要求确定在装载体积不受限制的情况下，将尽可能多的集装箱装上轮船。

template<class Type>

void Loading(int x[],Type w[],Type c,int n)

{

int \*t=new int[n+1];

Sort(w,t,n);

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

x[i]=0;

for(int i=1;i<=n&&w[t[i]]<=c;i++)

{

x[t[i]]=1;//从最轻的开始装

c-=w[t[i]];//装完之后剩余空间减少

}

}

1. 哈夫曼树算法

template<class T>

BinaryTree<int>HuffmanTree(T f[],int n)

{

Huffman<T>\*W=new Huffman<T>[n+1];

BinaryTree<int>z,zero;

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

{

z.MakeTree(i,zero,zero);

W[i].weight=f[i];

W[i].tree=z;

}//数组变成一个最小堆

MinHeap<Huffman<T>>Q(1);

Q.Initialize(w,n,n);//将堆中的树合并

Huffman<T>x,y;

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

{

Q.DeleteMin(x);

Q.DeleteMin(y);

z.MakeTree(0,x.tree,y.tree);

x.weight+=y.weight;

x.tree=z;

Q.Insert(x);

Q.DeleteMin(x);//最后得到的树

Q.deactivate();

delete[] w;

return x.tree;

}

}

押题：

14．哈弗曼编码的贪心算法所需的计算时间为（ B ）。

A、O（n2n） B、O（nlogn） C、O（2n） D、O（n）

18.下面是贪心算法的基本要素的是（ C ）。

A、重叠子问题 B、构造最优解 C、贪心选择性质 D、定义最优解

21、下面关于NP问题说法正确的是（ B ）  
A NP问题都是不可能解决的问题  
B P类问题包含在NP类问题中  
C NP完全问题是P类问题的子集  
D NP类问题包含在P类问题中

24. （D ）是贪心算法与动态规划算法的共同点。

A、重叠子问题 B、构造最优解 C、贪心选择性质 D、最优子结构性质

30、下面问题（ B ）不能使用贪心法解决。  
A 单源最短路径问题 B N皇后问题   
C 最小花费生成树问题 D 背包问题  
31、下列算法中不能解决0/1背包问题的是（ A ）  
A 贪心法 B 动态规划 C 回溯法 D 分支限界法  
40、背包问题的贪心算法所需的计算时间为（ B ）

A、O（n2n）     B、O（nlogn）    C、O（2n）      D、O（n）

44．贪心算法与动态规划算法的主要区别是（ B ）。

A、最优子结构 B、贪心选择性质 C、构造最优解 D、定义最优解

53．采用贪心算法的最优装载问题的主要计算量在于将集装箱依其重量从小到大排序，故算法的时间复杂度为 ( B ) 。

A、O（n2n） B、O（nlogn） C、O（2n） D、O（n）

1.背包问题的贪心算法

void Knapsack(int n,float M,float v[],float w[],float x[])

{

Sort(n,v,w);

int i;

for (i=1;i<=n;i++) x[i]=0;

float c=M;

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

if (w[i]>c) break;

x[i]=1;

c - =w[i];

}

if (i<=n) x[i]=c/w[i];

}

4.贪心算法求活动安排问题

template<class Type>

void **GreedySelector**(int n, Type s[], Type f[], bool A[])

{

A[1]=true;

int j=1;

for (int i=2;i<=n;i++) {

if (s[i]>=f[j])

{ A[i]=true;

j=i;

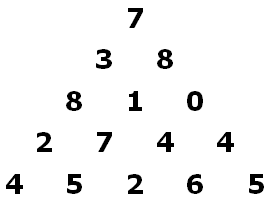
}

else A[i]=false;

}

}

1. 动态规划
2. 数字三角形（最短路径)



int MaxSum(int r,int j)

{

if(r==N)//判断是否走完

return D[r][j];

if(aMaxSum[r+1][j]==-1)//如果MaxSum(r+1,j)没计算过

aMaxSum[r+1][j]=MaxSum(r+1,j);

if(aMaxSum[r+1][j+1]==-1)//如果MaxSum(r+1,j+1)没计算过

aMaxSum[r+1][j+1]=MaxSum(r+1,j+1);

if(aMaxSum[r+1][j]>aMaxSum[r+1][j+1])

return aMaxSum[r+1][j]+D[r][j];

return aMaxSum[r+1][j+1]+D[r][j];

}

2.0-1背包问题

设n个物品的重量存储在数组w[n]中，价值存储在数组v[n]中，背包容量为C，数组V[n+1][C+1]存放迭代结果，其中V[i][j]表示前i个物品装入容量为j的背包中获得的最大价值，数组x[n]存储装入背包的物品

int KnapSack(int n,int w[],int v[])

{

for(i=0;i<=n;i++)//初始化第0列

V[i][0]=0;

for(j=0;j<=C;k++)//初始化第0行

V[0][j]=0;

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

for(j=1;j<=C;j++)

if(j<w[i]) V[i][j]=V[i-1][j];

else V[i][j]=max(V[i-1][j],V[i-1][j-w[i]=v[i]);

j=C;//求装入背包的物品

for(i=n;i>0;i--)

{

if(V[i][j]>V[i-1][j])

{

x[i]=1;

j=j-w[i];

}

else x[i]=0;

}

return V[n][C];//返回背包获取的最大价值

}

3.最长公共子序列

int CommonOrder(int m, int n, int x[ ], int y[ ], int z[ ])

{

for (j=0; j<=n; j++) //初始化第0行

L[0][j]=0;

for (i=0; j<=m; i++) //初始化第0列

L[i][0]=0;

for (i=1; i<=m; i++)

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

if (x[i]= =y[j]) { L[i][j]=L[i-1][j-1]+1; S[i][j]=1; }

else if (L[i][j-1]>=L[i-1][j]) { L[i][j]=L[i][j-1]; S[i][j]=2; }

else {L[i][j]=L[i-1][j]; S[i][j]=3; }

i=m; j=n; k=L[m][n];

for (i>0 && j>0)

{

if (S[i][j]= =1) { z[k]=x[i]; k--; i--; j--; }

else if (S[i][j]= =2) j--;

else i--;

}

return L[m][n];

}

1. 平均比较次数：中pi是记录ri的查找概率，ci是在二叉查找树中查找ri的比较次数。

押题：

13.备忘录方法是那种算法的变形。（ B ）

A、分治法 B、动态规划法 C、贪心法 D、回溯法

16．最长公共子序列算法利用的算法是（ B ）。

A、分支界限法 B、动态规划法 C、贪心法 D、回溯法

23.下列哪一种算法不是随机化算法（ C ）

A. 蒙特卡罗算法B. 拉斯维加斯算法C.动态规划算法D.舍伍德算法

35．下列是动态规划算法基本要素的是（ D ）。

A、定义最优解 B、构造最优解 C、算出最优解 D、子问题重叠性质

36．下列算法中通常以自底向下的方式求解最优解的是（ B ）。

A、分治法 B、动态规划法 C、贪心法 D、回溯法

45. 实现最大子段和利用的算法是（ B ）。

A、分治策略 B、动态规划法 C、贪心法 D、回溯法

28.动态规划算法的两个基本要素是 最优子结构 性质和 重叠子问题 性质 。

21. 动态规划算法的基本思想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。

2.最大子段和: 动态规划算法

int MaxSum(int n, int a[])

{

int sum=0, b=0； //sum存储当前最大的b[j], b存储b[j]

for(int j=1； j<=n； j++) {

if (b>0) b+= a[j] ；

else b=a[i]; ； //一旦某个区段和为负，则从下一个位置累和

if(b>sum) sum=b;

}

return sum；

2设计动态规划算法的主要步骤为：

1. 找出最优解的性质，并刻划其结构特征。
2. 递归地定义最优值。
3. 以自底向上的方式计算出最优值。

（4）根据计算最优值时得到的信息，构造最优解。

1. 分治法与动态规划法的相同点与不同点分别是什么？

两者的相同点：将待求解的问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。

两者的不同点是：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互相独立的。而用分治法求解的问题，经分解得到的子问题往往是互相独立的。

1. 回溯
2. 回溯和分枝限界法是比较常用的对候选解进行系统检查两种方法。
3. 常用的剪枝函数：用约束函数在扩展结点处剪去不满足约束的子树；用限界函数剪去得不到最优解的子树。
4. 具有限界函数的深度优先生成法称为回溯法。
5. 回溯法算法框架：采用树的非递归深度优先遍历算法，可将回溯法表示为一个非递归迭代过程。

void iterativeBacktrack()

{

int t=1;

while(t>0){

if(f(n,t)<=g(n,t))

for(int i=f(n,t);i<=g(n,t);i++){

x[t]=h(i);

if(constraint(t)&&bound(t)){

if(solution(t)) output(x);//输出最优解

else t++;//否则搜索下一节点

}

}

else t--;//回溯到上一节点

}

}

1. 回溯三要素：解空间、约束条件、状态树
2. 装载问题：

public static void Backtrack(int i)

{

if(i>=n)//到达叶节点

{

if(cw>bestw)//更新当前最优解

{

for(int j=0;j<n;j++) bestx[j]=x[j];//更新

bestw=cw;

return;

}

r-=w[i];//剩余货箱的重量和

if(cw+w[i]<=c)//搜索左子树，之前的一件可以装进来

{

x[i]=1;cw+=w[i];//更新当前重量

Backtrack(i+1);//搜索下一结点

cw-=w[i];//回溯

}

if(cw+r>bestw)//当前+剩余有可能大于最优解，搜索右子树

{

x[i]=0;Backtrack(i+1);

}

r+=w[i];//更新

}

}

1. 作业调度

int Flow(int \*\* M, int n, int bestx[ ] )

{ int ub = 32767;

Flowshop X;

X.x = new int [n+ 1]; //当前调度

X.f2 = new int[n+ l]; //机器2完成处理时间

X.t = t; //各作业所需处理时间

X.n= n; //作业数

X.bestx = bestx; //当前最优调度

X. bestf = ub; //当前最优调度时间

X.f1 = 0; //机器1完成处理时间

X.f = 0; //时间和

for(int i = 0;i<= n; i++)

X.f2[i] = 0,X.x[i]=i;

X. Backtrack( 1 );

delete [ ] X. x;

delete [ ] X. f2;

return X. bestf ;}

时间复杂度O(n!)

押题：

5. 回溯法解旅行售货员问题时的解空间树是（ A ）。

A、子集树 B、排列树 C、深度优先生成树 D、广度优先生成树

19.回溯法的效率不依赖于下列哪些因素（ D ）

A.满足显约束的值的个数 B. 计算约束函数的时间

C. 计算限界函数的时间 D. 确定解空间的时间

20.下面哪种函数是回溯法中为避免无效搜索采取的策略（ B ）

A．递归函数 B.剪枝函数 C。随机数函数 D.搜索函数

32、回溯法搜索状态空间树是按照（ C ）的顺序。  
A 中序遍历 B 广度优先遍历 C 深度优先遍历 D 层次优先遍历

15、使用回溯法进行状态空间树裁剪分支时一般有两个标准：约束条件和目标函数的界，N皇后问题和0/1背包问题正好是两种不同的类型，其中同时使用约束条件和目标函数的界进行裁剪的是 0/1背包问题 ，只使用约束条件进行裁剪的是 N皇后问题 。

30.回溯法是一种既带有 系统性 又带有 跳跃性 的搜索算法。

33．回溯法搜索解空间树时，常用的两种剪枝函数为 约束函数 和 限界函数 。

5.用回溯法搜索子集树的算法为：

void **backtrack** (int t)

{

if (t>n) output(x);

else

for (int i=0;i<=1;i++) {

x[t]=i;

if (constraint(t)&&bound(t)) backtrack(t+1);

}

}

9. 回溯法中常见的两类典型的解空间树是子集树和排列树。

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。这类子集树通常有2n个叶结点，遍历子集树需O(2n)计算时间 。

当所给的问题是确定n个元素满足某种性质的排列时，相应的解空间树称为排列树。这类排列树通常有n!个叶结点。遍历排列树需要O(n!)计算时间。

//定义一个节点类

struct Node

{

int number;

vector<int>x;//保存当前解

};

//定义一个Queen的类

class Queen

{

friend int nQueen(int);

public:

bool Place(Node q,int n);

void Research();

int n;//皇后个数

int \*bestx;//最优解

};

//判断是否能够放置的函数

bool Queen::Place(Node q,int n)

{

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

if((abs(n-j)==abs(q.x[j]-q.x[n]))||(q.x[j]==q.x[n])) return false;

return true;

}

void Queen::Research()

{

queue<Node>Q;//活节点队列

Node sign;

sign.number=-1;

Q.push(sign);//同层节点尾部标志

int t=1;//当前节点所处的层

Node Ew;//当前扩展节点

Ew.number=0;

//搜索子集空间树

while(1)

{

//检查所有的孩子节点

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

{

//把当前扩展节点的值赋给下一个节点

Node q;

q.number=t;

q.x.push\_back(0);//第一个位置为0

for(int i=1; i<t; i++) q.x.push\_back(Ew.x[i]);

q.x.push\_back(k);

if(Place(q,t))

Q.push(q);

}

//取下一扩展节点,取出，赋值给Ew

Ew=Q.front();

Q.pop();

if(Ew.number==-1)

{

//同层节点尾部标记

t++;//进入下一层

Q.push(sign);//增加标记

//继续往下去下一个节点

Ew=Q.front();

Q.pop();

}

if(Ew.number==n) //找到最后一层的节点

{

for(int i=0; i<=n; i++) bestx[i]=Ew.x[i];

break;

}

}

}

int nQueen(int n)

{

Queen X;

X.n=n;

X.bestx=new int[n+1];

for(int i=0; i<=n; i++) X.bestx[i]=0;

X.Research();

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

{

cout<<X.bestx[i]<<" ";

}

}

1. 分支限界
2. 分类：队列式和优先队列式
3. 关键问题：如何确定合适的限界函数；如何组织待处理结点表；如何确定最优解中的各个分量 。

押题：

15．分支限界法解最大团问题时，活结点表的组织形式是（ B ）。

A、最小堆 B、最大堆 C、栈 D、数组

26. 分支限界法解旅行售货员问题时，活结点表的组织形式是（ A ）。

A、最小堆 B、最大堆 C、栈 D、数组

37．采用广度优先策略搜索的算法是（ A ）。

A、分支界限法 B、动态规划法 C、贪心法 D、回溯法

43．采用最大效益优先搜索方式的算法是（ A ）。

A、分支界限法 B、动态规划法 C、贪心法 D、回溯法

46.优先队列式分支限界法选取扩展结点的原则是（ C ）。

A、先进先出 B、后进先出 C、结点的优先级 D、随机

48、广度优先是（ A ）的一搜索方式。

A、分支界限法      B、动态规划法    C、贪心法    D、回溯法

32．分支限界法是一种既带有 系统性 又带有 跳跃性 的搜索算法。

1. 分支限界法与回溯法的相同点与不同点分别是什么？

相同点：都是一种在问题的解空间树T中搜索问题解的算法。

不同点：

（1）求解目标不同；

1. 搜索方式不同；
2. 对扩展结点的扩展方式不同；

（4）存储空间的要求不同。

7. 用分支限界法设计算法的步骤是：

(1)针对所给问题，定义问题的解空间（对解进行编码）；分

(2)确定易于搜索的解空间结构（按树或图组织解） ；

(3)以广度优先或以最小耗费（最大收益）优先的方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。

8. 常见的两种分支限界法的算法框架

（1）队列式(FIFO)分支限界法：按照队列先进先出（FIFO）原则选取下一个节点为扩展节点。 （2）优先队列式分支限界法：按照优先队列中规定的优先级选取优先级最高的节点成为当前扩展节点。

10. 分支限界法的搜索策略是：

在扩展结点处，先生成其所有的儿子结点（分支），然后再从当前的活结点表中选择下一个扩展结点。为了有效地选择下一扩展结点，加速搜索的进程，在每一个活结点处，计算一个函数值（限界），并根据函数值，从当前活结点表中选择一个最有利的结点作为扩展结点，使搜索朝着解空间上有最优解的分支推进，以便尽快地找出一个最优解。