常见面试算法

1. 选择排序的算法实现

void select\_sort(int a[],int n)

{

int i,k,j,temp;

for(i=0;i<n-1;i++) //n个数排序，循环n-1趟

{

k=i;

for(j=i;j<n;j++) //查找最小的数的下标

{

if(a[k]>a[j])

k = j;

}

if(i!=k) //交换

{

temp = a[i];

a[i]=a[k];

a[k]=temp;

}

}

}

2. 冒泡排序的算法实现

void bubble\_sort(int a[],int n)

{

int change = 0,temp,i,j;

for(i=0;i<n-1;i++)

{

change = 0;

//如果有交换change = 1;

for(j=0;j<n-1-i;j++)

{

if(a[j]>a[j+1])//相邻两个元素比较

{

temp = a[j];

a[j]=a[j+1];

a[j+1]=temp;

change = 1;

}

}

if(change == 0)

{

printf("cishu= %d\n",i+1);

break;

}

}

if(i==n-1)

{

printf("cishu=%d\n",i);

}

}

3. 快速排序的算法实现

void swap(int v[],int i,int j)

{

int temp;

temp = v[i];

v[i]=v[j];

v[j]=temp;

}

void qsort(int v[],int left,int right)

{

int high = right;

int last=0;

int low=left;

if(left>=right)//出口

return;

while(left<right)

{

if(v[left]>v[right])

{

swap(v,left,right);

last = ~last;

}

if(last==0)

right--;

else

left++;

}

qsort(v,low,left-1);

qsort(v,left+1,high);

}

4.判断单链表是不是有环（circle）

解题思路：类似操场跑步，一个跑的快，一个跑的慢，如果是“有环”，两人终究会“相遇”。

// 这是笔试时需要写的最主要函数链表创建函数可以不实现

int isCircle(node \*head)

{

 node \*p=head, \*q=head; //定义两个指针 p,q

 while( p->next && q->next)

{

   p = p->next; //p每次向后移动一个节点

   if (NULL == (q=q->next->next)) return 0; //q每次向后移动二个节点

  if (p == q) return 1; //相遇则有环，返回1

 }

 return 0;

}

5. 删除串中指定的字符（做此题时，千万不要开辟新空间，否则面试官可能认为你不适合做嵌入式开发）

**void delChar(char \*str, char c)**

**{**

**int i, j=0;**

**for(i=0; str; i++)**

**{**

**if(str!=c) str[j++]=str;**

**str[j] = '\0';**

**}**

**}**

6. 编码实现字符串转整型的函数（实现函数atoi的功能），据说是神州数码笔试题。如将字符串 ”+123”?123, ”-0123”?-123, “123CS45”?123, “123.45CS”?123, “CS123.45”?0

**int str2int(const char \*str)**

{

**// 字符串转整型函数**

**int i=0, sign=1, value = 0;**

**if(str==NULL) return 0; // 空串直接返回 0**

**if(str[0]=='-' || str[0]=='+')**

**{ // 判断是否存在符号位，确定正数、负数**

**i = 1;**

**sign = (str[0]=='-' ? -1 : 1);**

**}**

**//如果是数字，则继续转换**

**for(; str[i]!='\0' && str[i]>='0' && str[i]<='9'; i++)**

**value = value \* 10 + (str[i] - '0');**

**return sign \* value;**

**}**

7. 时针分针一天24小时重合几次

基本思路：表面上有60个小格，每小格代表一分钟，时针每分钟走1/12小格（一小时60分钟，移动5小格），分针每分钟走1小格，从第一次重合到第二次重合分针比时针多走一圈即60小格，所以

60/（1-1/12）=720/11

每隔720/11分才重合一次（而并不是每小时重合一次）

一天24小时共1440分钟里有22个720/11，如果说算上0点和24点，那也是重合23次而已，但我觉得0点应该算到前一天的24点头上，所以每一天循环下来重合22次啊

8. 说是有一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前十个词。（类似于大数据分析）

基本思路：先用哈希，统计每个词出现的次数，然后在用在N个数中找出前K大个数的方法找出出现次数最多的前10个词。

递归经典算法

1、递归实现判断一个字符串是不是回文字串（如：abcdedbca就是回文字串，判断一个面试者对递归理解的简单程序）

**int find(char \*str, int n) {**

**if(n<=1)**

**{**

**return 1;**

**}**

**else if(str[0]==str[n-1])**

**{**

**return find(str+1, n-2); //切记，return不能丢**

**}**

**else  return 0;**

**}**

非递归实现回文字串的算法

**int isPalindromeStr(char \*str)**

**{**

**int len = strlen(str);**

**char \*start = str;**

**char \*end = str+len-1;**

**while(start<end)**

**{**

**if(\*start!=\*end)**

**return 0; //不是回文**

**start ++;**

**end--;**

**}**

**if(start>=end)**

**return 1; // 是回文字串**

**}**

2. 递归将一个整数输出：如654321，输出1,2,3,4,5,6，代码实现如下：

**void Reverse(int\* pResult,int iOrigin)**

**{**

**if(iOrigin/10==0)**

**{**

**pResult[0]=iOrigin%10 ;**

**return ;**

**}else**

**{**

**pResult[0]=iOrigin%10 ;**

        Reverse(pResult+1,iOrigin/10) ;

    }

}

**int** main()

{

**int** a=654321;

**int** b[10] ;

    Reverse(b,a) ;

**for**(**int** i=0;i<6;i++)

    {

        cout<<b[i]<<',' ;

    }

    cout<<endl ;

}

3. 将一个整型数转化为二进制，递归实现，直接打印

**void binary(int numbers)**

**{**

**int remainder;**

**if (numbers <= 1)**

**{**

**cout << numbers;**

**return;**

**}**

**remainder = numbers % 2;**

**binary(numbers >> 1);**

**cout << remainder;**

**}**

**int main()**

**{**

**int number;**

**cout << "请输入一个正整数： ";**

**cin >> number;**

**if (number < 0)**

**cout << "输入的数据错误！\n";**

**else{**

**cout << number << "转化为二进制位：";**

**binary(number);**

**}**

**return 0;**

**}**

4. 将一个整型数转化为二进制字符串，递归实现

**void binary(int numbers,char \*str,int \*p\_n)**

**{**

**int remainder;**

**if (numbers <= 1) //递归出口**

**{**

**str[\*p\_n]=numbers+48;**

**(\*p\_n)++;**

**return;**

**}**

**remainder = numbers % 2;**

**binary(numbers/2,str,p\_n);**

**str[\*p\_n]=remainder+48;**

**(\*p\_n)++;**

**}**

**int main()**

**{**

**int number;**

**int n=0;**

**char str[32]={0};**

**printf( "请输入一个正整数： ");**

**scanf("%d",&number) ;**

**if (number < 0)**

**printf( "输入的数据错误！\n");**

**else{**

**printf("%d转化为二进制位:",number);**

**binary(number,str,&n);**

**puts(str);**

**}**

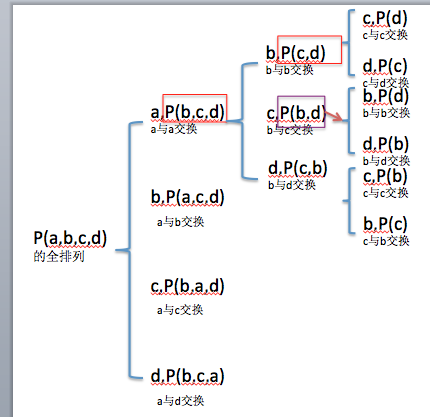
**return 0;**

**}**

5.用递归实现全排列

如abc的全排列是：abc acb bac bca cab cba

基本思路：将多项的全排列以此分解成少项的全排列，一项一项递减。



**void swap(char \*a,char \*b)**

**{**

**char tmp;**

**tmp = \*a;**

**\*a = \*b;**

**\*b = tmp;**

**}**

**void Perm(char list[],int index,int len)**

**{**

**int i=0 ;**

**char tmp ;**

**if(index==len)**

**{//Print**

**for(i=0;i<len;i++)**

**{**

**printf("%c,",list[i]);**

**}**

**printf("\n");**

**}**

**else**

**{**

**for(i=index;i<len;i++)**

**{**

**swap(&list[index],&list[i] );**

**Perm(list,index+1,len) ;**

**swap(&list[index],&list[i] );**

**}**

**}**

**}**

**int main()**

**{**

**char tmp[]={'a','b','c','d'} ;**

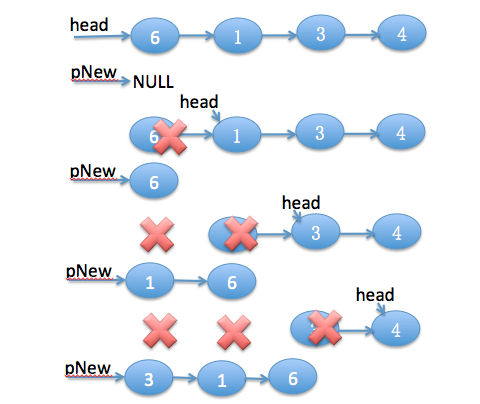
**Perm(tmp,0,4) ;**

**return 0 ;**

**}**

6 .单链表的逆转（倒置），用递归及非递归实现

//链表的倒置，非递归实现,将节点一个一个摘下来，前插入的方式插入到新链表上



**linklist link\_reverse(linklist head)**

**{**

**linklist pNew = NULL;**

**//入参检查**

**if(head == NULL)**

**return NULL;**

**linklist p = head;**

**while(head!=NULL)**

**{**

**p=head;**

**head = head->next;**

**p->next = pNew;**

**pNew = p;**

**}**

**return pNew;**

**}**

//链表的倒置，递归实现

**linklist link\_reverse\_digui(linklist node)**

**{**

**if (node.next == null)**

**return node;**

**var prevNode = Reverse(node.next);**

**var temp = node.next;**

**temp.next = node;**

**node.next = null;**

**return prevNode;**

**}**

7.

/\*二分/折半查找算法及其各种二分查找变种算法\*/

**/\*原始版本点二分查找算法(循环版本)**

**参数:**

**arr: 待查找的数组**

**n: 数组的最大下标**

**key: 要查找的关键字**

**返回值:成功找到返回下标，否则返回-1**

**\*/**

**int binarySearch\_for(int arr[],int n,int key)**

**{**

**int low=0;**

**int high=n;**

**int mid=0;**

**while(low<=high)**

**{**

**mid=low+((high-low)>>1);//** 若使用(low+high)/2求中间位置容易溢出, 特别是数组元素比较多的时候

**if(key==arr[mid])**

**return mid;**

**else if(key<arr[mid])**

**high=mid-1;**

**else**

**low=mid+1;**

**}**

**return -1;**

**}**

**/\*原始版本点二分查找算法(递归版本)**

**参数:**

**arr: 待查找的数组**

**low: 查找的最小下标**

**high:查找的最大下标**

**key: 要查找的关键字**

**返回值:成功找到返回下标，否则返回-1**

**\*/**

**int binarySearch(int arr[],int low,int high,int key)**

**{**

**int mid=low+((high-low)>>1);**

**if(low>high)**

**return -1;**

**if(key==arr[mid])**

**return mid;**

**else if(key<arr[mid])**

**return binarySearch(arr,low,mid-1,key);**

**else**

**return binarySearch(arr,mid+1,high,key);**

**}**

**/\***

**要点**

**每次查找范围一定要缩小，如果另mid = l 或 mid = r时，查找范围为长度为1时会陷入死循环。**

**如果写成 l < r , 当查找范围长度为1时，会导致找不到key。**

**写成mid = l + (r - l)/2，可防止整数溢出。**

**使用ans保存已经符合条件的最后的mid，最后返回即可。**

**\*/**

**/\*查找等于关键字的最小位置\*/**

**int binarySearchMin(int arr[],int n,int key)**

**{**

**int low=0;**

**int high=n;**

**int mid=0;**

**int ans=-1;**

**while(low<=high)**

**{**

**mid=low+((high-low)>>1);**

**if(key<=arr[mid])**

**{**

**ans=mid;**

**high=mid-1;**

**}**

**else**

**low=mid+1;**

**}**

**if(arr[ans]==key)**

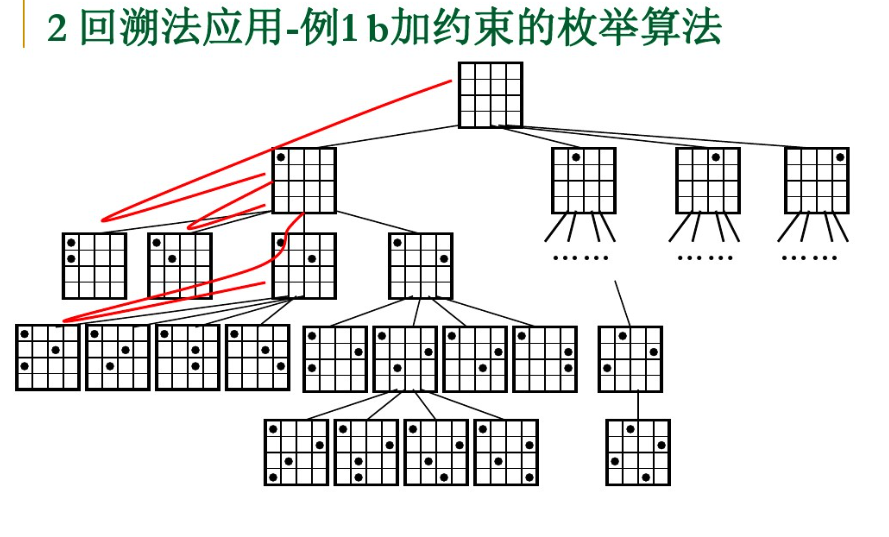
**return ans;**

**else**

**return -1;**

**}**

8. 八皇后问题



/\*

1,要求皇后之间互相冲突，每行只能有一个皇后

2,用queen[]数组来存贮每个皇后的列位置

\*/

**#define MAX 8**

**int sum=0; //满足要求的解个数**

**int queen[MAX];//记录每行皇后的列标**

**/\*输出满足八皇后要求的皇后位置\*/**

**void printOut()**

**{**

**for(int i=0;i<MAX;i++)**

**{**

**for(int j=0;j<MAX;j++)**

**{**

**if(j==queen[i])**

**printf("Q");**

**else**

**printf("0");**

**}**

**printf("\n");**

**}**

**printf("\n");**

**return;**

**}**

**/\*判断该位置放放入皇后是否满足要求，满足要求返回1,否则返回**0\*/

**int isValid(int n)**

**{**

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

**{**

**if(queen[i]==queen[n])**

**return 0;**

**if(abs(queen[i]-queen[n])==(n-i))**

**return 0;**

**}**

**return 1;**

**}**

/\*求解八皇后的过程\*/

**void placeQueen(int i)**

**{**

**for(int j=0;j<MAX;j++)**

**{**

**if(i==MAX)**

**{**

**sum++;**

**printf("第%d组解:\n",sum);**

**printOut();**

**return;**

**}**

**queen[i]=j; //放入皇后的位置**

**if(isValid(i)) //判断该位置是否符合要求**

**placeQueen(i+1); //符合要求则继续放入下一行皇后的位置**

**}**

**}**

**int main()**

**{**

**placeQueen(0);**

**printf("共%d组解\n",sum);**

**return 0;**

**}**

9. 大整数乘法

**int multi(int num1,int num2)**

**{**

**int \*result=NULL,\*pt1,\*pt2;**

**int size1=0,size2=0;**

**int n1=num1;**

**int n2=num2;**

**/\*计算第一个数字的位数\*/**

**for(;n1;n1/=10)**

**size1++;**

**/\*计算第二个数字的位数\*/**

**for(;n2;n2/=10)**

**size2++;**

**/\*将每一个数字的每一位当作数组的一个元素，存储在数组中,需**要**三个数组，分别存贮乘数和结果\*/**

**result=malloc((size1+size2)\*4);**

**assert(result!=NULL);**

**memset(result,0,(size1+size2)\*4);**

**pt1=malloc(size1\*4);**

**assert(pt1!=NULL);**

**memset(pt1,0,size1\*4);**

**pt2=malloc(size2\*4);**

**assert(pt2!=NULL);**

**memset(pt2,0,size2\*4);**

**n1=num1;**

**for(int i=0;i<size1;i++)**

**{**

**pt1[i]=n1%10;**

**n1/=10;**

**}**

**n2=num2;**

**for(int i=0;i<size2;i++)**

**{**

**pt2[i]=n2%10;**

**n2/=10;**

**}**

**/\*乘法实现**

**1,先按照乘法规则将数组中的每一位对应相乘法**

**2,对结果中点数组进行进位处理**

**\*/**

**for(int i=0;i<size1;i++)**

**{**

**int k=i;**

**for(int j=0;j<size2;j++)**

**{**

**result[k++] +=pt1[i]\*pt2[j];**

**}**

**}**

**for(int i=0;i<size1+size2;i++)**

**{**

**if(result[i]>=10)**

**{**

**result[i+1] +=result[i]/10;**

**result[i]=result[i]%10;**

**}**

**}**

**/\*逆序输出数组的每一个元素即为两个数字相乘的结果\*/**

**for(int i=size1+size2-1;i>=0;i--)**

**{**

**printf("%d",result[i]);**

**}**

**free(result);**

**free(pt1);**

**free(pt2);**

**return 0;**

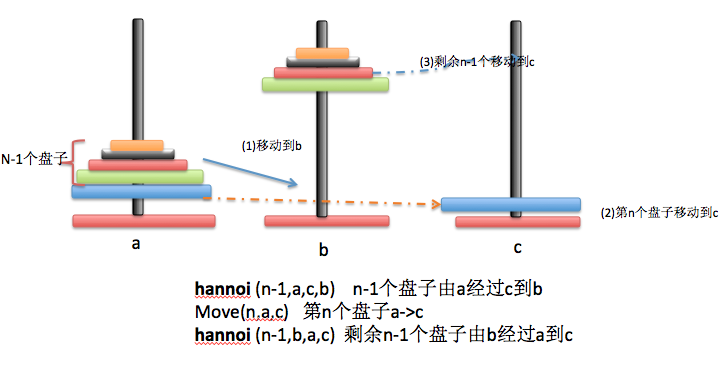
**}**

10.汉诺塔算法

汉诺塔是根据一个传说形成的一个问题。汉诺塔（又称河内塔）问题是源于印度一个古老传说的益智玩具。大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着64片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。

如果柱子标为a,b,c 要由a搬到c，在只有一个盘子时，直接a->c,当有2个盘子时，b当作辅助柱。a->b a->c b->c

当有n个盘子时：



#define NUMOFDISKS 10

/\* 汉诺塔问题求解

1,假设盘子数量为1时，则直接将盘子从a移动到c

2,当盘子数量为n时，先将n-1个盘子从a移动到b

3,将第n个盘子从a移动到c

4,最后将n-1个盘子从b移动到c

\*/

**void move(int n,char x,char y)**

**{**

**printf("move %d from %c to %c\n",n,x,y);**

**return;**

}

**/\*将盘子从a移动到c,其中b作为辅助柱子\*/**

**void hannoi(int n,char a,char b,char c)**

**{**

**if(n==1)**

**move(1,a,c);**

**else**

**{**

**hannoi(n-1,a,c,b);**

**move(n,a,c);**

**hannoi(n-1,b,a,c);**

**}**

**}**

**int main()**

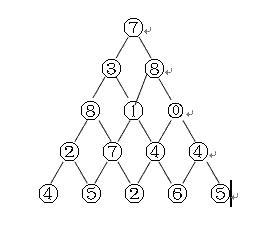
**{**

**hannoi(NUMOFDISKS,'a','b','c');**

**return 0;**

**}**

1. 假设淘宝一天有5亿条成交数据，求出销量最高的100个商品并给出算法的时间复杂度。
2. 给一列无序数组，求出中位数并给出算法的时间复杂度。
3. 输入一个整型数组，求出子数组和的最大值，并给出算法的时间复杂度。
4. 给出10W条人和人之间的朋友关系，求出这些朋友关系中有多少个朋友圈（如A-B、B-C、D-E、E-F，这4对关系中存在两个朋友圈），并给出算法的时间复杂度。
5. 如图所示的数字三角形，从顶部出发，在每一结点可以选择向左走或得向右走，一直走到底层，要求找出一条路径，使路径上的值的和最大。给出算法的时间复杂度。



1. 有一个很长二进制串，求出除以3的余数是多少，给出算法的时间复杂度。
2. 链表中倒数第k个结点（剑指offer）
3. 数组中超过一半的数字(剑指offer)