Noi-CSP/J 复习笔记

# 1.初赛的试卷分析，分值分布，难度分布

总目标分数 80

43、44道小题来组成

试卷试题分布：

第一大题

单项选择题 15道题 每题2分 <最简单的一类，最容易拿分的一类>

第二大题

阅读程序题目<1,2,3> 一共40分

第一题比较简单， 代码长度比较短

<考一点 指针，位运算>

第二题，第三题 难度比较高，认真对待

<二分，递归，搜索，动态规划，指针，位运算>

判断题+选择题的形式 判断题一般一个1.5分 个别的2分 选择题一般一个3分 个别的4分

第三大题

程序填空题

两道题 给你题目描述，代码功能，给你残缺代码，让你选择空缺部分的代码

每道题 5个空，每个空3分共计30分

阅读程序，程序填空 -- 编码能力，阅读代码的能力

# 2.Day 1

## 进制转换题：

进制转换-<二进制，八进制，十进制，十六进制> 选择题，阅读程序

二进制<->八进制 100011.10110 每三位二进制表示一个八进制数字

二进制<->十六进制 100011.10110 每四位。。。。。。。十六进制

有m进制的数字 要转化成 n进制的 怎么办？--> 中间使用十进制来作为过度

1. 把m进制的数字转化成十进制的（按位权值相乘后相加）

2. 把十进制的数字转成n进制

进制转换模板，见第6段

## 位运算题：

位操作/位运算<需要先将数字转成二进制>

~ << >> & | ^

~:单目运算符按位取反 0-1 1-0

<<: 左移操作 x<<1 === x\*2

>>: 右移操作 x>>1 === x/2

& : 按位与操作 5 & 6 ==4 同时为1结果为1 否则结果为0

| : 按位或操作 5|6 == 7 同时为0结果为0 否则结果为1

^ : 按位异或操作 5^6 ==3 相同为0，不同为1

## 位运算常见的应用：

1. 判断奇数偶数

bool fun\_(int a){

    if(a&1){

        return true;

    }else{

        return false;

    }

}

2. 交换变量

void Swap(int &a,int &b){

    if(a!=b){

        a^=b;

        b^=a;

        a^=b;

    }

}

3. 变换符号

int sr(int a){

    return (~a+1);

}

4. 取绝对值1

int Abs(int a){

    int i=a>>31;// i就是 数字a的符号位 一个int占4个字节 一个字节8个

    return i==0?a:(~a+1);

}

5. 取绝对值2

int Abs2(int a){

    int i=a>>31;

    return ((a^i)-i);

}

6. 大小写转换

char fun\_2(char a){

    return char(a^' ');

}

7. 去重操作

（1，2，3，4，1，2，3，4，5）找出只出现一次的数字

int d[9]={1,2,3,4,1,2,3,4,5};

int fun\_find(int d,int n){

    int ans=0;

    for(int i=1;i<=n;i++){

        ans^=d[i];

    }

    return ans;

}

## 按位异或公式：

几个常用的性质

1. x^x==0

2. x^y==y^x

3. (x^y)^z==x^(y^z)

4. x^y^x=y

## 原码、反码、补码：

先了解原码、反码、补码分别是什么。以整数为例，假定长度为8位

**原码**

原码比较容易理解，举两个例子：

【+100】原 = 01100100

【+10】原=00001010

【-100】原=11100100

【-10】原=10001010

第一位表示的是正负符号，0是正，1是负，后面的位数表示的是数值的二进制，如果位数不够补0代替

0 1100100=【+100】原

第一位0表示正，后面100的二进制为：1100100，所以+100的原码是01100100

0 0001010= 【+10】原

第一位0表示正，10的二进制是1010，上面假定了长度为8，所有在1010前面补上3个0，所以+10的原码是00001010

负数原码同理，第一位变成1。

**反码**

对于正数，反码与原码相同，

对于负数，符号位不变，其数值位的**1变0，0变1**。

计算反码一般要把原码先求出来。

两个例子：

【+100】原 = 01100100

【+100】反 = 01100100

【-100】原=11100100

【-100】反=10011011

样例详解：

【+100】原 = 01100100

【+100】反 = 01100100

对于正数，反码与原码相同，所以+100的反码跟原码相同，都为01100100

【-100】原=11100100

【-100】反=10011011

对于负数，第一位(符号位)不变，其余位1变0，0变1，所以-100的反码为10011011

**补码**

对于正数，补码与原码相同，

对于负数，符号位不变，其数值位**反码后，在最低位加1**。

计算补码一般要把反码先求出来。

两个例子：

【+100】原 = 01100100

【+100】反 = 01100100

【+100】补 = 01100100

【-100】原=11100100

【-100】反=10011011

【-100】补 =10011100

样例详解：

【+100】原 = 01100100

【+100】反 = 01100100

【+100】补 = 01100100

对于正数，补码与原码相同，所以+100的补码为01100100

【-100】原=11100100

【-100】反=10011011

【-100】补 =10011100

对于负数，符号位不变，其数值位反码后，在最低位加1，意思就是，-100的反码数值部分(0011011)加1，可以转换成十进制计算（也可以直接用逢二进一直接计算，更简单），即：0011011(前面0的忽略)，转换成10进制为27，27加1等于28，28转换成二进制为0011100（由于假定位数为8，补上两个0），得出-100的补码为10011100。

## 位运算操作：

| **功能** | **示例** | **位运算** |
| --- | --- | --- |
| 去掉最后一位 | (101101->10110) | x >> 1 |
| 在最后加一个0 | (101101->1011010) | x < < 1 |
| 在最后加一个1 | (101101->1011011) | x < < 1+1 |
| 把最后一位变成1 | (101100->101101) | x | 1 |
| 把最后一位变成0 | (101101->101100) | x | 1-1 |
| 最后一位取反 | (101101->101100) | x ^ 1 |
| 把右数第k位变成1 | (101001->101101,k=3) | x | (1 < < (k-1)) |
| 把右数第k位变成0 | (101101->101001,k=3) | x & ~ (1 < < (k-1)) |
| 右数第k位取反 | (101001->101101,k=3) | x ^ (1 < < (k-1)) |
| 取末三位 | (1101101->101) | x & 7 |
| 取末k位 | (1101101->1101,k=5) | x & ((1 < < k)-1) |
| 取右数第k位 | (1101101->1,k=4) | x >> (k-1) & 1 |
| 把末k位变成1 | (101001->101111,k=4) | x | (1 < < k-1) |
| 末k位取反 | (101001->100110,k=4) | x ^ (1 < < k-1) |
| 把右边连续的1变成0 | (100101111->100100000) | x & (x+1) |
| 把右起第一个0变成1 | (100101111->100111111) | x | (x+1) |
| 把右边连续的0变成1 | (11011000->11011111) | x | (x-1) |
| 取右边连续的1 | (100101111->1111) | (x ^ (x+1)) >> 1 |
| 去掉右起第一个1的左边 | (100101000->1000) | x & (x ^ (x-1)) |
| 消去最右边的1 | - | x&(x - 1) |

# 3.Day 2

## 冒泡排序：

**冒泡排序**是一种简单的排序算法。它重复地遍历要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。遍历数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。

以下是冒泡排序的基本步骤：

* 比较相邻的元素。如果第一个比第二个大（升序），就交换他们两个。
* 对每一对相邻元素做同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。
* 针对所有的元素重复以上的步骤，除了最后一个。
* 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

void bubblesort(int a[],int n){

    for(int i=1;i<=n;i++){

        bool re=true;

        for(int j=1;j<n-i+1;j++){

            if(a[j]>a[j+1]){

                swap(a[j],a[j+1]);

                re=false;

            }

        }

        if(re) break;

    }

}

## 选择排序：

**选择排序**是一种简单直观的排序算法。它的工作原理是每一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，直到全部待排序的数据元素排完。

以下是选择排序的基本步骤：

* 在未排序序列中找到最小（大）元素，存放到排序序列的起始位置。
* 再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。
* 重复第二步，直到所有元素均排序完毕。

void selectsort(int a[],int len){

    for(int i=1;i<=len;i++){

        int maxn=i;

        for(int j=i+1;j<=len;j++){

            if(a[j]<a[maxn]) maxn=j;

        }

        if(maxn!=i) swap(a[i],a[maxn]);

    }

}

## 插入排序：

## **插入排序**是一种简单直观的排序算法。它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。

以下是插入排序的基本步骤：

* 从第一个元素开始，该元素可以认为已经被排序。
* 取出下一个元素，在已经排序的元素序列中从后向前扫描。
* 如果被扫描的元素（已排序）大于新元素，将该元素后移一位。
* 重复步骤3，直到找到已排序的元素小于或者等于新元素的位置。
* 将新元素插入到该位置后。
* 重复步骤2~5。

如果比较操作的代价比交换操作大的话，可以进行优化，创建一个临时变量存储要排序的数，然后将前面比它大的数向后移动，最后再把它放到正确的位置。

void insertsort(int a[],int len){

    for(int i=1;i<=len;i++){

        for(int j=i;j>=1;j--){

            if(a[j]<a[j-1]) swap(a[j],a[j-1]);

            else break;

        }

    }

}

## 快速排序(扩展) ：

思路：假设我们有个数字序列是这样的

[6 1 2 7 9 3 4 5 10 8]

1. 现在序列中寻找一个基准值

2. 我们将比6小的数字放在他的左边，比6大的数字放在他的右边

1. 从右向左找第一个小于6的数字，从左往右找第一个大于6的数字

2. 交换他们两个的位置

快速排序是一种非常高效的排序算法，它的基本思想是通过一趟排序将待排记录分割成独立的两部分，其中一部分记录的关键字均比另一部分的关键字小，然后分别对这两部分记录继续进行排序，以达到整个序列有序。在这段代码中，temp是基准值，i和j是左右指针，通过不断地交换和移动指针，最后达到排序的目的。当i和j相遇时，基准值归位，然后对基准值左右两边的子序列递归进行快速排序。这就是快速排序的基本过程。

## 排列（在乎顺序）：

* **全排列**：有n个人全部来排队，队长度为n，问题就是一共有多少种排队的方式？
* 在第一个位置上 我们选人的时候有 8种选择
* 在第二个位置上 我们选人的时候有 7种选择
* 在第三个位置上 我们选人的时候有 6种选择
* …
* 在第八个位置上 我们选人的时候有 1种选择
* ans=8765432\*1 <乘法原理>
* **部分排列**：有n个人选择其中m个人来排队，一共有多少种排队方式？
* 在第一个位置上 我们选人的时候有 8种选择
* 在第二个位置上 我们选人的时候有 7种选择
* 在第三个位置上 我们选人的时候有 6种选择
* 在第四个位置上 我们选人的时候有 5种选择
* 在第五个位置上 我们选人的时候有 4种选择
* ans=87654
* **阶乘**：8！= 8765432\*1
* A(n,m)=n\*(n-1)(n-2)(n-3)…1/ (n-m)(n-m-1)(n-m-2)…\*1 = n! / (n-m)!

## 组合（不在乎顺序）由已知推未知：

* **组合**：有n个人 我从中选取m个 有多少种选法？
* C(n,m)=A(n,m)= n!/(n-m)!
* 每一种选法有多少种顺序？ 假设是mm种排队方法

A(n,m)/mm==C(n,m)

mm=m!

C(n,m)=A(n,m)/m!

=n!/(n-m)!/m!

=n!/((n-m)!\*m!)

## 鸽巢原理（抽屉原理）：

* **第一抽屉原理**：如果把多于n个物体，放入n个盒子中，那么至少有一个盒子里面包含了两个及以上的物体
* **第二抽屉原理**：如果把多于nm个物体，放入n个盒子中，那么至少有一个盒子里面包含了不少于m+1个物体。如果把mn-1个物体， 放入n个盒子中，那么至少有一个盒子里的物体数量至多为m-1个
* **第三抽屉原理**：如果把无数个物体， 放入n个盒子中，那么至少有一个盒子里面包含了无数个物体

例题：

* 在13个人中，至少有两个人他们的生日是同一个月份。 正确
* 假设有n对已婚夫妇，为了保证有一对夫妇被选出，至少要从这2\*n个人里选n+1个人。 正确
* 我现在有5种颜色不同的袜子，各15只，混装在一个箱子里，至少要从箱子里取几只袜子，就能保证至少有三双袜子(袜子颜色相同就算是一双，不分左右) 10只

## 6. 容斥原理：

* **容斥原理**：在不考虑重叠的情况下，把包含于某内容的所有对象的数目先计算出来，然后再把重复计算的去掉。这种计数方式 我们就叫容斥原理。

AUB=A+B-A^B

AUBUC=A+B+C-AB-AC-BC+AB^C

# 4.Day 3

## 计数排序

基数排序是桶排序的简化版，用数组模拟来完成，用数组下标来表示数字，下标所指向的元素表示当前位置是否出现了数字

示例：

3 1 2 5 4 8

a[3]++;

a[1]++;

a[2]++;

a[5]++;

a[4]++;

a[8]++;

a 0 1 2 3 4 5 6 7 8 9 10

0 1 1 1 1 1 0 0 1 0 0

输出的时候遍历a数组，如果a[i]==1 则输出 i

void countingsort(int a[],int len){

    int b[100001]={};

    for(int i=1;i<=len;i++){

        b[a[i]]++;

    }

    int j=1;

    for(int i=1;i<10001;i++){

        while(b[i]!=0){

            a[j]=i;

            j++;

            b[i]--;

        }

    }

}

## 桶排序：

桶排序也叫箱排序 将大问题化小

思路：

* 将数组分到有限的桶里
* 每个桶再个别排序（有可能再使用别的排序算法或者是以递归的方式继续使用桶排序）
* 最后再依次把各个桶里的记录
* 列出来得到有序序列

void buckesort(int a[],int len){

    int bucker[11][11]={};

    for(int i=1;i<=len;i++){

        int index=a[i]/10;

        bucker[index][0]++;

        bucker[index][bucker[index][0]]=a[i];

    }

    for(int i=0;i<=10;i++){

        quicksort(bucker[i],1,bucker[i][0]);

    }

    int i=1;

    for(int j=0;j<=10;j++){

        for(int k=1;k<=bucker[j][0];k++){

            a[i++]=bucker[j][k];

        }

    }

}

## 希尔排序：

希尔排序是插入排序的升级版，简单的插入排序很循规蹈矩，不管数组怎么样分布，他都会一点点进行比较，如果遇见[54321]这种倒叙排列，我们数组末端的1要回到首位是非常费劲的，需要移动n-1次。希尔排序在此基础之上采用跳跃式分组的策略，通过某个增量将数组元素划分为若干组，然后分组进行插入排序，随后逐步缩小增量，继续按组进行插入排序，直到增量为。

增量的选择 len= n/2

void shellsort(int a[],int len){

    for(int gap=len/2;gap>0;gap/=2){

        for(int i=gap+1;i<=len;i++){

            int j=i-gap;

            while(j>=1&&a[j]>a[j+gap]){

                swap(a[j],a[j+gap]);

                j-=gap;

            }

        }

    }

}

# 基数排序(扩展)：

是一种非比较的整数排序算法，原理里将整数按位切割成不同的数字，然后各个位进行比较

有两种方式：

* 最低位优先法(LSD)：从最低位向最高位依次进行排序
* 最高位优先法(MSD): 从最高位向最地位依次进行排序

void radixsort(int a[],int len){

    for(int tail=1;tail<=100;tail\*=10){

        int radix[10][11]={};

        for(int i=1;i<=len;i++){

            int index=a[i]/tail%10;

            radix[index][0]++;

            radix[index][radix[index][0]]=a[i];

        }

        int i=1;

        for(int j=0;j<10;j++){

            for(int k=1;k<=radix[j][0];k++){

                a[i++]=radix[j][k];

            }

        }

    }

}

## 堆排序：

思路：

堆排序实际上是利用堆的性质来进行排序的，堆：本质是是一颗完全二叉树

满足以下两个性质：

* 堆的每一个父节点都大于（小于）其子节点--要大于就都大于 要小于你就都小于
* 堆的每个左子树和右子树也是一个堆

堆的分类：

* 大根堆（大顶堆）：堆的每个父节点都大于其子节点
* 小根堆（小顶堆）：堆的每个子节点都小于其子节点

堆的存储：

* 一般堆的存储我们用数组来实现，i节点的两个子节点分别是，i\*2 和i\*2+1

我们继续分析 ，如果我们把一组数字放在堆里 ，堆的第一个元素，最大值、最小值这样的话 我们就可以在排序的时候，直接将第一个元素和最后一个元素进行交换，然后从第一个元素开始向下调整至第n-1个元素。

堆排序的步骤 可以分为三步：

* 建堆（升序大堆，降序小堆）
* 交换数据
* 向下调整

void adjust(int a[],int end,int parent){

    int child=2\*parent;

    if(child+1<=end&&a[child]<a[child+1]){

        child++;

    }

    if(a[child]>a[parent]){

        swap(a[child],a[parent]);

        if(child<=end/2) adjust(a,end,child);

    }

}

void heapsort(int a[],int len){

    for(int i=len;i>1;i--){

        for(int j=i/2;j>=1;j--){

            adjust(a,i,j);// 把当前完全二叉树改造成堆

        }

        swap(a[1],a[i]);

    }

}

## 归并排序：

思路：经典的分之策略，二分的思想

将待排序的序列，一分两半，继续分，一直分，直到只剩一个元素

然后再将分开的序列一点点合并回来

举例：

* 8 4 5 7 1 3 6 2
* 8 4 5 7 1 3 6 2
* 8 4 5 7 1 3 6 2
* 8 4 5 7 1 3 6 2
* 4 8 5 7 1 3 2 6
* 4 5 7 8 1 2 3 6
* 1 2 3 4 5 6 7 8

递归算法

void mergesort(int a[],int st,int ed){

    if(st>=ed) return;

    int mid=(st+ed)/2;

    mergesort(a,st,mid); // 不断递归往下分

    mergesort(a,mid+1,ed);

    int i=st; // 进行合并操作

    int j=mid+1;

    int k=st;// b数组的下标

    int b[100]={};// 备胎

    while(i<=mid&&j<=ed){

        if(a[i]<=a[j]){

            b[k]=a[i];

            i++;

            k++;

        }else{

            b[k]=a[j];

            j++;

            k++;

        }

    }

    while(i<=mid){

        b[k]=a[i];

        i++;

        k++;

    }

    while(j<=ed){

        b[k]=a[j];

        j++;

        k++;

    }

    for(int i=st;i<=ed;i++){

        a[i]=b[i];

    }

}

# 5.Day 4

### 线性数据结构

* 栈
* 队列

### 树

**逻辑结构**

树（树状图–一种特殊的图）是一种树形的数据结构，我们假设有n个节点，那么应该有n-1条边，我们把它称之为树，是因为它看起来像一颗倒挂的树。

**树的特点：**

1. 每个元素我们称之为节点
2. 每个节点有0个或者多个子节点
3. 没有父节点的节点我们称之为根节点
4. 每一个非根节点都有一个父节点
5. 节点和节点之间的连线我们称之为边
6. 如果树有n个节点，那么应该有n-1条边
7. 除了根节点之外其余数据元素被分为m个互不相交的集合，其中每一个集合本身也是一棵树，我们称之为子树
8. 单个节点也是一棵树，树根就是该节点本身
9. 空集合也是树，我们称之为空树，其中没有节点
10. 树中的从任一节点出发到达其他节点的路径唯一

**树还可以这样来定义：**

看作是一个特殊的集合

**树的相关术语：**

1. 节点的度：一个节点含有的子树个数，称为该节点的度
2. 树的度：一棵树中，最大的节点的度我们称之为树的度
3. 叶节点/终端节点：度为0的点
4. 分支节点/非终端节点：度不为0的点
5. 双亲节点：如果一个节点含有子节点，那么这个节点我们知之为其子节点的双亲节点/父节点
6. 孩子节点/子节点：一个节点含有的子树的根节点，称之为该节点的子节点/孩子节点
7. 兄弟节点：具有相同的双亲节点的节点互称为兄弟节点
8. 节点的层次：从根节点开始，根节点为第1层，根节点的子节点为第2层，依次类推
9. 树的高度/深度：树中节点的最大层次
10. 堂兄弟节点：双亲节点在同一层的节点我们称之为堂兄弟节点
11. 节点的祖先：从根节点到该节点所经过的分支上的所有节点
12. 子孙：以某节点为根的子树中的任一节点称为该节点的子孙
13. 森林：m棵互不相交的树的集合称为森林

**树的分类：**

1. 无序树：树中任意节点的子节点之间没有顺序关系，这种我们称之为无序树
2. 有序树：树中任意节点的子节点之间有左右顺序关系，这种我们称之为有序树
3. 二叉树：度小于等于2的树，也就是每个节点的子树个数小于等于2
4. 完全二叉树：一颗二叉树至多只有最下面两层的节

进制转换代码模板

这段C++代码是用于进制转换的。它接收三个输入变量，分别是：

n：这是一个整数，表示输入数字的当前进制。

a：这是一个字符串，表示要转换的数字。这个数字是以n进制表示的。

m：这是一个整数，表示目标进制，即我们希望将数字a转换为m进制。

这段代码首先将输入的数字（以字符串形式）转换为十进制，然后再将其转换为目标进制。如果目标进制大于10，那么它会使用字母A-F来表示10-15。

最后，它会打印出转换后的数字。如果输入的数字是0，那么它会直接输出0。如果输入的数字包含小写字母，那么它会首先将其转换为大写字母。这是因为在这段代码中，字母A-Z被用来表示10-35。所以，这段代码可以处理最高到36进制的转换

D

#include<bits/stdc++.h>

using namespace std;

char a[10001];

int b[10001];

int n,m;

char w[7]="ABCDEF";

int main(){

    cin >> n >> a >> m;//  用一个字符串来表示数字

    int len=strlen(a);

    if(a[0]=='0'&&len==1){

        cout << '0' << endl;

        return 0;

    }

    for(int i=0;i<len;i++){

        if(a[i]>='a'&&a[i]<='z'){

            a[i]=a[i]-32;

        }

        if(a[i]>='0'&&a[i]<='9'){

            b[i]=a[i]-48;

        }else if(a[i]>='A'&&a[i]<='Z'){

            b[i]=a[i]-55;

        }

    }

    int ans=0,cnt=0; // 转为十进制,并且存在ans中

    for(int i=len-1;i>=0;i--){

        ans=ans+b[i]\*(pow(n,cnt));

        cnt++;

    }

    int cnt1=1; // 将十进制的数字ans转化为m进制,并倒叙存储在b数组中

    while(ans!=0){

        int r=ans%m;

        b[cnt1++]=r;//

        ans=ans/m;

    }

    for(int i=cnt1-1;i>=1;i--){//倒序输出

        if(b[i]<10){

            cout << b[i];

        }else{

            int k=b[i]-10;

            cout << w[k];

        }

    }

    return 0;

}