

8.13 课堂笔记 34

初赛练习

NOIP 提高组 2016 年真题解析

多选

1:

GPRS: 中文名称为通用无线分组业务, 属于无线通用技术。

以太网: 是一种计算机局域网技术。它规定了包括物理层的连线、电子信号和介质访问层协议的内容。

2:

网卡: 网卡是一块被设计用来允许计算机在计算机网络上进行通讯的计算机硬件。它使得用户可以通过电缆或无线相互连接。

光驱: 电脑用来读写光碟内容的机器。

显卡: 控制显示器的正确显示, 是连接显示器和个人计算机主板的重要组件。

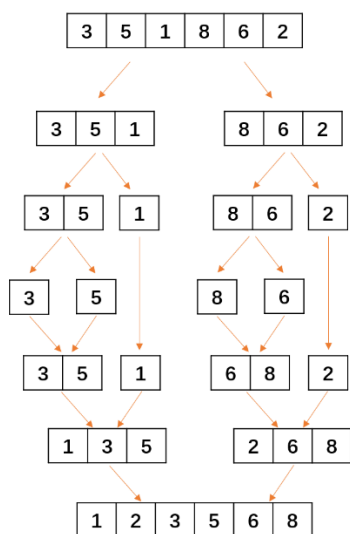
3:

选择排序: 对未排好序的数据进行循环遍历, 每一轮遍历找出最值的元素, 将其放到已排好序的序列一端。

插入排序: 将一个新的元素插入到已经排好序的序列中。类似于我们打扑克时不断摸牌插入手牌的过程。

冒泡排序: 对序列中的元素两两进行排序, 在一次次遍历过程中, 较小的元素会像水中的气泡一样慢慢上浮。

归并排序: 分治算法的一种典型应用。它的基本思想是将原本的序列拆分成一个个小的片段, 先对小的片段进行排序, 然后片段之间组合, 再进行排序。实际上就是从局部有序到整体有序的过程。

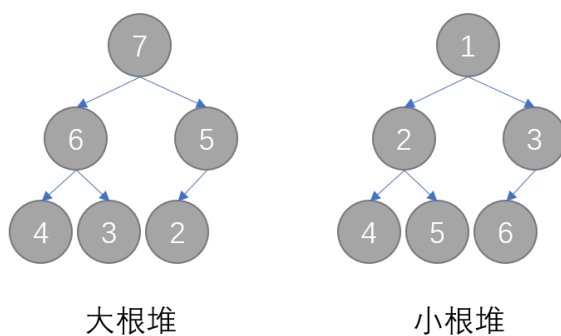


快速排序：也是分治算法的典型应用。它的基本思想是将数组划分为大于区、等于区和小于区。在遍历的过程中，两个指针的移动代表着大于区和小于区的边界的变化。

堆排序：利用堆这种数据结构进行排序的算法。

大根堆：每个节点都比它的子节点大的完全二叉树。

小根堆：每个节点都比它的子节点小的完全二叉树。

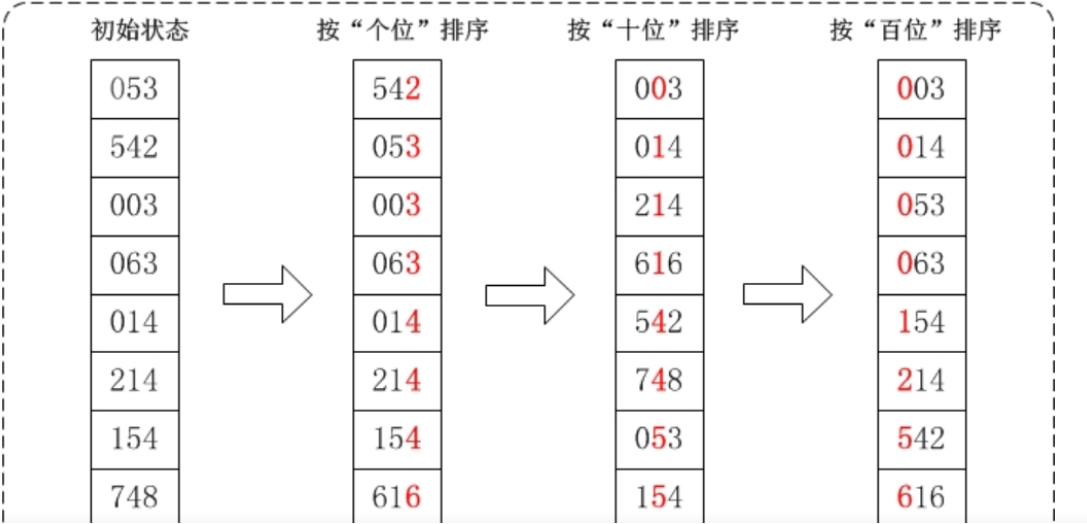


希尔排序：插入排序的一种，又称“缩小增量排序”。最后一次增量为 1。

基数排序 (Radix Sort)是桶排序的扩展，它的基本思想是：将整数按位数切割成不同的数字，然后按每个位数分别比较。

具体做法是：将所有待比较数值统一为同样的数位长度，数位较短的数前面补零。然后，从最低位开始，依次进行一次排序。这样从最低位排序一直到最高位排序完成以后，数列就变成一个有序序列。

通过基数排序对数组{53, 3, 542, 748, 14, 214, 154, 63, 616}，它的示意图如下：



NOIP 提高组 2016 年真题答案

第二十二届全国青少年信息学奥林匹克联赛初赛 提高组参考答案

一、单项选择题（共 15 题，每题 1.5 分，共计 22.5 分）

1	2	3	4	5	6	7	8
D	A	B	B	B	B	B	B
9	10	11	12	13	14	15	
B	D	B	A	C	C	A	

二、不定项选择题（共 5 题，每题 1.5 分，共计 7.5 分；每题有一个或多个正确选项，没有部分分）

1	2	3	4	5
ABC	A	AB	A	ABD

三、问题求解（共 2 题，每题 5 分，共计 10 分；每题全部答对得 5 分，没有部分分）

- 55
- 3

四、阅读程序写结果（共 4 题，每题 8 分，共计 32 分）

- 6,5,4,3,2,1,
- YES,NO,YES,
- 5
- 2 5

五、完善程序（共计 28 分，以下各程序填空可能还有一些等价的写法，由各省赛区组织本省专家审定及上机验证，可以不上报 CCF NOI 科学委员会复核）

		Pascal 语言	C++语言	C 语言	分值
1	(1)	i<=j			2
	(2)	next[rank[i]]:=rank[i+1]	next[rank[i]]=rank[i+1]		3
	(3)	higher:=height[next[i]]-height[i]	higher=height[next[i]]-height[i]		3
	(4)	shorter<higher			3
	(5)	previous[next[i]]:=previous[i]	previous[next[i]]=previous[i]		3
2	(1)	dist[1]:=0	dist[1]=0		2
	(2)	dist[x]+weight[j]<dist[point[j]]			3
	(3)	visit[x]:=0	visit[x]=0		3
	(4)	dist[x]+weight[j]=dist[point[j]]	dist[x]+weight[j]==dist[point[j]]		3
	(5)	visit[point[j]]:=1	visit[point[j]]=1		3

复赛练习

1. P2679

思路：

本题物品数量很小($N \leq 100$)，而背包大小很大($W \leq 10^9$)，所以这里要尽量减小可以使用的背包大小。如果直接使用 01 背包的方法做，空间复杂度过高。

每个物品的价格的极差为 3，记所有物品的最小价格为 minv ，那么可能的价格有：

$\text{minv}, \text{minv}+1, \text{minv}+2, \text{minv}+3$ 。

如果每个物品价格很高，占用的背包空间就会很大，空间复杂度就很大。这里可以把每个物品的价格都减去 minv ，使得所有物品的价格只可能为 $0, 1, 2, 3$ 。购买这样处理后的物品最多 100 个，能花的钱最多为所有物品价格的加和 sumv ，这个加和一定小于等于 300。在这种情况下，就可以认为背包大小为所有物品处理后价格的加和 sumv ，这个值不超过 300。

而选择的商品的总花费仍受到总钱数 W 的限制。假设购买价格处理后商品的花费（即背包大小）为 j ，买了 k 个商品。这 k 个商品每个商品的基础花费为 minv ，多出的价格部分即为处理后的商品，通过 j 元购买，总花费可以认为是 $k * \text{minv} + j$ ，这个总花费不能超过 W 元，所以有 $k * \text{minv} + j \leq W$ 。这里涉及到了购买商品的数量，所以在状态定义时得包含商品数量这一项。

1. 状态定义

集合：购买商品的方案

限制：前几个商品中选择，最多花费，购买商品的数量

属性：重要度加和

条件：最大

统计量：重要度加和

状态定义： $\text{dp}[i][j][k]$ ：处理价格后，最多花费 j 元在前 i 个物品中选择物品购买，选了 k 个物品能获得的最大重要度加和。

2. 状态转移方程

记 $p[i]$ 为第 i 物品的价格， $c[i]$ 为第 i 物品的重要程度。

集合：最多花费 j 元在前 i 个物品中选择物品购买，选了 k 个物品的方案

分割集合：第 i 物品是否选择

如果不选择第 i 物品，或无法选择第 i 物品，那么接下来需要用 j 元钱在前 $i-1$ 个物品中购买 k 个物品。 $\text{dp}[i][j][k] = \text{dp}[i-1][j][k]$

如果第 i 物品满足 $j \geq p[i]$ 且 k 与 j 满足 $k * \text{minv} + j \leq W$ ，此时可以选择第 i 物品，共选择 k 个物品。如果选择第 i 物品，那么接下来需要用 $j - p[i]$ 的元钱在前 $i-1$ 个物品中购买 $k-1$ 个物品。 $\text{dp}[i][j][k] = \text{dp}[i-1][j-p[i]][k-1] + c[i]$

以上两种情况取最大值。

3. 获得结果

对于所有可能的 k 与 j 构成的数对，都会产生一个最大花费钱数（背包大小），在该值的限制下可以得到一个可以购买商品的极大价值。

应该遍历所有的 k 与 j ，求在前 n 个物品中选择物品，选择 k 个物品， j 元购买价格处理后的商品，能获得的最大价值。即 $\text{dp}[n][j][k]$ 的最大值。

这也相当于求 dp 数组中所有元素的最大值，这就是本题的结果。

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  #define N 105
4  #define M 305
5  #define INF 0x3f3f3f3f
6  int p[N], c[N]; //p[i]: 第i物品的价格 c[i]: 第i物品的重要程度
7  int dp[N][M][N]; //dp[i][j][k]: 处理价格后, 最多j元在前i个物品中选择物品购买, 选了k个物品能获得的最大重要度
8  int n, w, minv = INF, sumv, ans; //sumv: 处理后背包大小
9  int main()
10 {
11     cin >> n >> w;
12     for(int i = 1; i <= n; ++i)
13     {
14         cin >> p[i] >> c[i];
15         minv = min(minv, p[i]);
16     }
17     for(int i = 1; i <= n; ++i)
18     {
19         //更新sumv, 降低空间复杂度
20         p[i] -= minv;
21         sumv += p[i];
22     }
23     for(int i = 1; i <= n; i++)
24     {
25         for(int j = 0; j <= sumv; j++)
26         {
27             for(int k = 1; k <= n; k++)
28             {
29                 if((long long)minv*k+j <= w && j >= p[i])
30                     dp[i][j][k] = max(dp[i-1][j][k], dp[i-1][j-p[i]][k-1]+c[i]);
31                 else
32                     dp[i][j][k] = dp[i-1][j][k];
33             }
34         }
35     }
36     //找最大值
37     for(int j = 0; j <= sumv; j++)
38         for(int k = 1; k <= n; k++)
39             ans = max(ans, dp[n][j][k]);
40     cout << ans;
41     return 0;
42 }
```

2. P1291

我们假设我们手中已有了 x 个球星的瓶盖，则要想拿到这 x 个之外的瓶盖的概率为 $1/(n-x)$ ，就需要买 $n \cdot 1/(n-x)$ 次。

将所有的期望加起来然后化简，得到 $n/1 + n/2 + n/3 + \dots + n/n$

```

/*
期望为  $n/1 + n/2 + n/3 + \dots + n/n$ 
*/
#include<bits/stdc++.h>
using namespace std;
long long n,k,fenmu=1,fenzi;
long long fenmuTemp, fenziTemp1, fenziTemp2, fenziTemp3;

long long gcd(long long x, long long y){
    if(y==0){
        return x;
    }
    else{
        gcd(y,x%y);
    }
}

int main(){
    cin >> n;
    fenzi = n;
    for(int i=2;i<=n;i++){
        fenmuTemp = fenmu*i;
        fenziTemp1 = fenzi*i;
        fenziTemp2 = n*fenmu; // 分子分母通分才能相加
        fenziTemp3 = fenziTemp1 + fenziTemp2;
        k=gcd(fenziTemp3, fenmuTemp);
        fenziTemp3 /= k;
        fenmuTemp /=k;
        fenmu = fenmuTemp;
        fenzi = fenziTemp3;
    }
    if(fenzi%fenmu==0){
        cout << fenzi/fenmu;
    }
    else{
        long long num=0, num1=0, templ=fenzi/fenmu, temp2=fenmu;
        //求整数的位数
        while(templ>0){
            num++;
            templ /= 10;
        }
        // 求分母的位数, 最终要使分子、分母与分数线对其输出
        while(temp2>0){
            num1++;
            temp2 /= 10;
        }
        for(int i=1;i<=num;i++){
            cout<<" ";
        }
        cout << fenzi%fenmu << endl << fenzi/fenmu;
        for(int i=1;i<=num1;i++){
            cout<<"-"; //减号的个数等于分母的位数
        }
        cout<<endl;
        for(int i=1;i<=num;i++){
            cout<<" ";
        }
        cout<<fenmu<<endl;
    }
    return 0;
}

```

3. P1618

先把三个数求出来，然后判断是不是在 100-999 范围内，并且判断是不是每个数字出现一次即可

```

#include<bits/stdc++.h>
using namespace std;
int a, b, c;
int A, B, C;
int vis[10];
bool flag, flag2;

int main() {
    cin >> a >> b >> c;
    for (int k=1; k<=1000/c; k++) {
        //求出三个数
        A = k*a;
        B = k*b;
        C = k*c;
        if (B>999 || C>999) {
            break;
        }
        //将三个数进行分解，判断是否有重复数字
        for (int i=1; i<=3; i++) {
            vis[A%10]++;
            A/=10;
        }
        for (int i=1; i<=3; i++) {
            vis[B%10]++;
            B/=10;
        }
        for (int i=1; i<=3; i++) {
            vis[C%10]++;
            C/=10;
        }
        for (int i=1; i<=9; i++) {
            if (vis[i]!=1) {
                flag=1;
                break;
            }
        }
        memset(vis, 0, sizeof(vis)); //每次标记数组清零
        if (!flag) {
            cout << k*a << " " << k*b << " " << k*c << endl;
            flag2 = 1;
        }
        flag = 0;
    }
    if (!flag2) {
        cout << "No!!!";
    }
    return 0;
}

```

4. P1627

用一个标记数组记录每个数与所给中位数的相对大小（大于记为 1，小于记为 -1），

然后以那个指定的中位数的位置向两边分别遍历一遍，记录标记数组的累加和。

满足条件的子序列分四种情况：

- 1、从中位数向左遍历，标记数组累加和为 0
- 2、从中位数向右遍历，标记数组累加和为 0
- 3、中位数在子序列中间位置，标记数组累加和为 0
- 4、中位数单独成一个子序列


```

#include <bits/stdc++.h>
using namespace std;
const int maxn=1e5+6;
int a[100010], vis[100010], sum[110000];
int n, b;
int pos, s, cnt;
int main() {
    cin >> n >> b;
    for (int i=1; i<=n; i++)
    {
        cin >> a[i];
        if (a[i]==b)
        {
            pos=i; //找中位数下标
        }
        else if (a[i]<b)
        {
            vis[i]=-1; //小于中位数标记为-1
        }
        else {
            vis[i]=1; //大于中位数标记为1
        }
    }
    for (int i=pos-1; i>=1; i--)
    {
        s+=vis[i];
        if (s==0)
        {
            cnt++; //1、从中位数向左遍历，标记数组累加和为0
        }
        sum[maxn+s]++;
    }
    s=0;
    for (int i=pos+1; i<=n; i++)
    {
        s+=vis[i];
        if (s==0)
        {
            cnt++; //2、从中位数向右遍历，标记数组累加和为0
        }
    }
    cnt+=sum[maxn-s]; //3、中位数在子序列中间位置，标记数组累加和为0
    cnt++; //4、中位数单独成一个子序列
    cout << cnt << endl;
    return 0;
}

```

5. P3719

四种情况：

- 1、如果是字符 **a**，那么直接计数
- 2、如果是左括号，那么把左括号当作起点，重新计数并累加
- 3、如果是 **|**，那么取 **|** 左边和右边的最大值，右边从 **|** 开始重新计数
- 4、如果是右括号，表示此单位的计数结束

提问：

- 1、什么时候需要递归？
遇到起点的时候，即遇到左括号或者 **|**
- 2、什么时候需要返回？

遇到终点的时候，即遇到右括号或者 **|** 或者输入结束

```

#include<bits/stdc++.h>
using namespace std;
int diGui(int cnt){
    char ch;
    while(cin >> ch){
        //1、如果是字符a，那么直接计数
        if(ch=='a'){
            cnt++;
        }
        //2、如果是左括号，那么把左括号当作起点，重新计数并累加
        if(ch=='('){
            cnt += diGui(0);
        }
        //3、如果是|，那么取|左边和右边的最大值，右边从|开始重新计数
        if(ch=='|'){
            return max(cnt, diGui(0));
        }
        //4、如果是右括号，表示此单位的计数结束
        if(ch==')'){
            return cnt;
        }
    }
    return cnt;
}
int main(){
    cout<<diGui(0);
    return 0;
}

```

6. P3332

P3223:

不考虑老师相邻问题: $(n+2)! * C(m, n+3) * m!$

如果老师相邻: $2 * (n+1)! * C(m, n+2) * m!$

$$\begin{aligned}
 \text{排队方案数} &= \frac{(n+1)! * (n+2)! * (n*n+3*n+2*m)}{(n+3-m)!} \\
 &= (n+1)! * (n*n+3*n+2*m) * ((n+4-m)*(n+5-m)* \dots * (n+2))
 \end{aligned}$$