

# qhfh 算法思维训练 题解

registerGen

## Contents

<b>1</b>	<b>字符串变换</b>	<b>3</b>
1.1	题目描述 . . . . .	3
1.2	输入格式 . . . . .	3
1.3	输出格式 . . . . .	3
1.4	输入输出样例 . . . . .	3
1.5	说明/提示 . . . . .	3
1.5.1	数据范围 . . . . .	3
1.6	题解 . . . . .	4
1.7	代码 . . . . .	4
<b>2</b>	<b>最大子方阵</b>	<b>5</b>
2.1	题目描述 . . . . .	5
2.2	输入格式 . . . . .	5
2.3	输出格式 . . . . .	5
2.4	输入输出样例 . . . . .	5
2.5	说明/提示 . . . . .	6
2.5.1	样例 1 解释 . . . . .	6
2.5.2	数据范围 . . . . .	6
2.6	题解 . . . . .	6
2.7	代码 . . . . .	7
<b>3</b>	<b>摘苹果</b>	<b>9</b>
3.1	题目描述 . . . . .	9
3.2	输入格式 . . . . .	9
3.3	输出格式 . . . . .	9
3.4	输入输出样例 . . . . .	9

3.5	说明/提示 . . . . .	10
3.5.1	数据范围 . . . . .	10
3.6	题解 . . . . .	10
3.7	代码 . . . . .	11

## 1 字符串变换

### 1.1 题目描述

给定一个由字符 **0**, **1**, **2** 组成的字符串  $s$ 。如果 **0** 和 **1** 相邻或 **1** 和 **2** 相邻, 则我们可以交换它们的位置, 如  $01 \rightarrow 10$ ,  $10 \rightarrow 01$ ,  $12 \rightarrow 21$  等。

对于  $s$ , 只可以进行上述操作, 且可以操作任意的次数, 请问可以得到的字典序最小的字符串是什么?

### 1.2 输入格式

一行, 一个字符串  $s$ 。

### 1.3 输出格式

一行, 表示答案。

### 1.4 输入输出样例

输入 #1	输出 #1
100210	001120
输入 #2	输出 #2
11222121	11112222

### 1.5 说明/提示

#### 1.5.1 数据范围

对于 30% 的数据,  $|s| \leq 10$ 。

对于 50% 的数据,  $|s| \leq 100$ 。

对于 70% 的数据,  $|s| \leq 10^3$ 。

对于 100% 的数据,  $1 \leq |s| \leq 10^5$ ,  $\Sigma \subseteq \{0, 1, 2\}$ 。

$|s|$  表示字符串  $s$  的长度,  $\Sigma$  表示字符集。

## 1.6 题解

由于 0, 2 的相对位置不会改变, 所以只需将所有的 1 移到  $s$  的第一个 2 前面即可。

时间复杂度  $O(n)$ 。

## 1.7 代码

```
1 #include<cstdio>
2 #include<algorithm>
3 #include<cstring>
4
5 const int N=1e5;
6
7 int n;
8 char s[N+10];
9
10 int main(){
11     scanf("%s",s+1);
12     n=int(strlen(s+1));
13     int pos=n+1,cnt=0; // pos 表示第一个 2 的位置, cnt 表
        示 1 的个数
14     for(int i=1;i<=n;i++)
15         if(s[i]=='1')cnt++;
16     for(int i=1;i<=n;i++)
17         if(s[i]=='2'){pos=i;break;}
18     for(int i=1;i<pos;i++)
19         if(s[i]!='1')putchar(s[i]);
20     for(int i=1;i<=cnt;i++)
21         putchar('1');
22     for(int i=pos;i<=n;i++)
23         if(s[i]!='1')putchar(s[i]);
24     puts("");
25     return 0;
26 }
```

## 2 最大子方阵

### 2.1 题目描述

给定  $n \times m$  的矩形的点方阵，每个元素只可能为 1 或 0，例如：

```
0111
1000
0110
```

1 表示这个位置上站着一位同学，0 表示这个位置上没有人。现在你有指挥的权利，具体为可以交换任意的两行，并且这个权利可以使用无数次，那么请问，你可以折腾指挥出一个最大的，全部由同学构成的子方阵吗？

“子方阵”是指这样的点集：对于其中的任意一个点  $(x, y)$ ， $1 \leq a \leq x \leq b \leq n, 1 \leq c \leq y \leq d \leq m$ 。子方阵的大小是指其中 1 的个数。

例如，对于上述方阵，当  $a = 1, b = 2, c = 2, d = 3$  时，子方阵为：

```
11
00
```

### 2.2 输入格式

第一行包含 2 个整数  $n$  和  $m$ 。

接下来  $n$  行，每行  $m$  个字符，表示这一行中同学的分布情况。

### 2.3 输出格式

一行一个整数，表示答案。

### 2.4 输入输出样例

输入 #1	输出 #1
3 4 0111 1000 0110	4

## 2.5 说明/提示

### 2.5.1 样例 1 解释

我们可以交换原始方阵的 2, 3 行, 获得:

```
0111
0110
1000
```

则当  $a = 1, b = 2, c = 2, d = 3$  时, 对应的子方阵即为最大子方阵。

### 2.5.2 数据范围

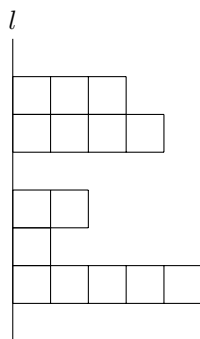
对于 50% 的数据,  $n, m \leq 100$ 。

对于 100% 的数据,  $1 \leq n, m \leq 5 \times 10^3$ 。

## 2.6 题解

首先, 这个数据范围是允许我们枚举子方阵的一条边的, 考虑枚举子方阵的左边。

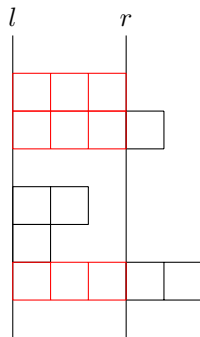
那么, 我们可以“利用”的 1 的范围如下图:



可以发现, 第  $i$  行可“利用”的 1 的个数为  $(i, l)$  往右的连续的 1 的个数 (包括它自己), 记为  $b_{i,l}$ 。

然后我们发现貌似还是不太好算, 所以再枚举子方阵的右边。

此时, 由于行是可以随便换的, 所以我们可以“利用”的 1 的范围如下图的红色部分:



为了方便计算，我们按 1 的个数从大到小排个序再枚举。

现在的问题就是，如何高效计算  $b_{i,j}$ ？

```

1 for(int i=1;i<=n;i++){ // 对于每一行
2     int cnt=0; // counter
3     for(int j=m;j;j--){ // 倒着枚举
4         if(a[i][j]==0)cnt=0; // 如果这个格子为 0，则 b[i][j] 就
           为 0
5         else cnt++; // 如果这个格子为 1，
           则 b[i][j] = b[i][j+1] + 1 (即 cnt + 1)
6         b[i][j]=cnt;
7     }
8 }
```

时间复杂度  $\mathcal{O}(nm \log n)$ 。但待排序的元素均在  $[0, m]$  内，所以排序可以使用桶排，时间复杂度  $\mathcal{O}(nm)$ 。

## 2.7 代码

```

1 #include<cstdio>
2 #include<algorithm>
3
4 const int N=5000;
5
6 int n,m;
7 bool a[N+10][N+10];
8 int b[N+10][N+10];
9 int tmp[N+10];
```

```
10 int buc[N+10];
11
12 int main(){
13     scanf("%d%d",&n,&m);
14     for(int i=1;i<=n;i++)
15         for(int j=1;j<=m;j++)
16             scanf("%1d",&a[i][j]);
17     // 预处理 b[i][j]
18     for(int i=1;i<=n;i++){
19         int cnt=0;
20         for(int j=m;j>=1;j--){
21             if(a[i][j]==0)cnt=0;
22             else cnt++;
23             b[i][j]=cnt;
24         }
25     }
26     int ans=0;
27     for(int j=1;j<=m;j++){
28         // tmp[i] 为 b[i][j] 排好序后的数组
29         for(int i=1;i<=n;i++)tmp[i]=b[i][j];
30         // 桶排
31         for(int i=0;i<=m;i++)buc[i]=0;
32         for(int i=n;i>=1;i--)buc[tmp[i]]++;
33         int tot=0;
34         for(int i=m;i>=0;i--){
35             for(int k=1;k<=buc[i];k++)
36                 tmp[++tot]=i;
37         }
38         int res=0;
39         for(int i=1;i<=n;i++){
40             res=max(res,i*tmp[i]); // 计算全部由同学构成的子方
41                                     阵大小。如果不理解这句话可以画画图。
42         }
43         ans=max(ans,res);
44     }
45     printf("%d\n",ans);
46 }
```



## 3 摘苹果

### 3.1 题目描述

果园中有  $n$  棵苹果树，它们排成了一排，每棵树上有  $c_i$  个苹果。现在，小玉站在第一棵树下，有  $W$  点能量，同时能量上限也为  $W$ 。

对于第  $i$  棵树，小玉可以摘  $[0, c_i]$  间的任意数量的苹果。每摘一个苹果，小玉的能量将会被消耗  $cost_i$ ，但是能量上限将会增加  $B$ 。

小玉只能按着顺序从第 1 棵树移动到第 2 棵树，从第 2 棵树到第 3 棵树……每次他到达下一棵树时，他都会恢复  $X$  点能量（当然，不能超过当时的上限）。这样一直进行下去，请问最后小玉最多可以摘到多少苹果？注意，小玉不可以出现负能量的状态。

### 3.2 输入格式

第一行 4 个整数  $n, W, B, X$ 。

第二行  $n$  个整数  $c_i$ 。

第三行  $n$  个整数  $cost_i$ 。

### 3.3 输出格式

一行一个整数表示答案。

### 3.4 输入输出样例

输入 #1	输出 #1
2 10 7 11 2 10 6 1	11
输入 #2	输出 #2
5 1 4 6 3 4 6 5 1 3 0 10 2 9	10

### 3.5 说明/提示

#### 3.5.1 数据范围

对于 40% 的数据,  $1 \leq n \leq 100$ ,  $1 \leq \sum_{i=1}^n c_i \leq 100$ 。

对于 100% 的数据,  $1 \leq n \leq 10^3$ ,  $1 \leq \sum_{i=1}^n c_i \leq 10^4$ ,  $0 \leq c_i \leq 10^4$ ,  
 $0 \leq W, B, X, cost_i \leq 10^9$ 。

### 3.6 题解

首先, 容易想到如下的状态设计: 设  $f_{i,j}$  表示只摘前  $i$  棵树上的苹果, 且当前能量为  $j$  时能摘到的最多的苹果。

很快你会发现这很不现实—— $W \leq 10^9$ , 我们不能开下这么大的数组!

考虑从  $\sum_{i=1}^n c_i \leq 10^4$  入手, 改变状态如下:

设  $f_{i,j}$  表示前  $i$  棵树, 摘了  $j$  个苹果的——等等, 摘了  $j$  个苹果的什么呀?

考察题目的变量, 你会发先有三个: 摘了的苹果树的个数, 摘到的苹果的个数, 能量。

dp 第  $\times$  定理: 一般的, 设计状态时要把题目中所有变量包含进状态里。

那么, 我们可以得到最终的状态定义: 设  $f_{i,j}$  表示前  $i$  棵树, 摘了  $j$  个苹果的最大能量 (当然是能量越多越好了)。

那么, 最终的答案就是使得  $f_{n,i} \geq 0$  的最大的  $i$ 。

有了状态定义, 转移方程和初始化就很简单了:

$$f_{i,j} = \max_{k \in [0, c_i]} \{ \min \{ f_{i-1, j-k} - k \cdot cost_i + X, W + jB \} \} \quad \text{if } f_{i-1, j-k} - k \cdot cost_i \geq 0$$

初始化:

$$f_{i,j} = \begin{cases} W & i = 0 \wedge j = 0 \\ -\infty & \text{otherwise} \end{cases}$$

开一个  $10^3 \times 10^4$  的 `long long` 数组可能会爆空间 (我懒得算), 可以滚动数组优化空间。

$$\text{时间复杂度 } \mathcal{O} \left( \left( \sum_{i=1}^n c_i \right)^2 \right)。$$

### 3.7 代码

```
1 #include<cstdio>
2 #include<algorithm>
3 #include<cstring>
4
5 typedef long long ll;
6
7 const int N=1e3;
8 const int M=1e4;
9 const ll inf=0x3f3f3f3f3f3f3fLL;
10
11 int n,W,B,X,c[N+10],cost[N+10],sumc;
12 ll f[2][M+10];
13
14 int main(){
15     scanf("%d%d%d%d",&n,&W,&B,&X);
16     for(int i=1;i<=n;i++)
17         scanf("%d",c+i),sumc+=c[i];
18     for(int i=1;i<=n;i++)
19         scanf("%d",cost+i);
20     memset(f,~0x3f,sizeof(f));
21     f[0][0]=W;
22     for(int i=1;i<=n;i++)
23         for(int j=0;j<=sumc;j++)
24             for(int k=0;k<=std::min(c[i],j);k++){
25                 ll tmp=f[(i&1)^1][j-k]-1LL*k*cost[i];
26                 if(tmp<0)continue;
27                 tmp+=X;
28                 if(tmp>W+1LL*j*B)tmp=W+1LL*j*B;
29                 f[i&1][j]=std::max(f[(i&1)^1][j],tmp);
30             }
31     for(int i=sumc;i>0;i--)
32         if(f[n&1][i]>=0){
33             printf("%d\n",i);
```

```
34     return 0;  
35 }  
36 return 0;  
37 }
```