

算法1-1 模拟与高精度

1-1-1 P1065 作业分配问题(模拟)

题目描述

🔼 收起

我们现在要利用 m 台机器加工 n 个工件，每个工件都有 m 道工序，每道工序都在不同的指定的机器上完成。每个工件的每道工序都有指定的加工时间。

每个工件的每个工序称为一个操作，我们用记号 $j-k$ 表示一个操作，其中 j 为1到 n 中的某个数字，为工件号； k 为1到 m 中的某个数字，为工序号，例如 $2-4$ 表示第2个工件第4道工序的这个操作。在本题中，我们还给定对于各操作的一个安排顺序。

例如，当 $n = 3, m = 2$ 时， $1-1, 1-2, 2-1, 3-1, 3-2, 2-2$ 就是一个给定的安排顺序，即先安排第1个工件的第1个工序，再安排第1个工件的第2个工序，然后安排第2个工件的第1个工序，等等。

一方面，每个操作的安排都要满足以下的两个约束条件。

1. 对同一个工件，每道工序必须在它前面的工序完成后才能开始；
2. 同一时刻每一台机器至多只能加工一个工件。

另一方面，在安排后面的操作时，不能改动前面已安排的操作的工作状态。

由于同一工件都是按工序的顺序安排的，因此，只按原顺序给出工件号，仍可得到同样的安排顺序，于是，在输入数据中，我们将这个安排顺序简写为“112332”。

还要注意，“安排顺序”只要求按照给定的顺序安排每个操作。不一定是各机器上的实际操作顺序。在具体实施时，有可能排在后面的某个操作比前面的某个操作先完成。

例如，取 $n = 3, m = 2$ ，已知数据如下（机器号/加工时间）：

工件号	工序1	工序2
1	1/3	2/2
2	1/2	2/5
3	2/2	1/4

则对于安排顺序“112332”，下图中的两个实施方案都是正确的。但所需要的总时间分别是10与12。

则对于安排顺序“112332”，下图中的两个实施方案都是正确的。但所需要的总时间分别是10与12。



当一个操作插入到某台机器的某个空档时（机器上最后的尚未安排操作的部分也可以看作一个空档），可以靠前插入，也可以靠后或居中插入。为了使问题简单一些，我们约定：在保证约束条件（1）（2）的条件下，尽量靠前插入。并且，我们还约定，如果有多个空档可以插入，就在保证约束条件（1）（2）的条件下，插入到最前面的一个空档。于是，在这些约定下，上例中的方案一是正确的，而方案二是不正确的。

显然，在这些约定下，对于给定的安排顺序，符合该安排顺序的实施方案是唯一的，请你计算出该方案完成全部任务所需的总时间。

输入格式

第1行为两个正整数 m, n ，用一个空格隔开。（其中 $m(< 20)$ 表示机器数， $n(< 20)$ 表示工件数）

第2行： $m \times n$ 个用空格隔开的数，为给定的安排顺序。

接下来的 $2n$ 行，每行都是用空格隔开的 m 个正整数，每个数不超过20。

其中前 n 行依次表示每个工件的每个工序所使用的机器号，第1个数为第1个工序的机器号，第2个数为第2个工序机器号，等等。

后 n 行依次表示每个工件的每个工序的加工时间。

可以保证，以上各数据都是正确的，不必检验。

输出格式

1个正整数，为最少的加工时间。

代码：

```
#include<stdio.h>
```

```
int m,n,ans,wk[25][25],t[25][25],a[405],xu[25],gong[25];
_Bool b[25][10007];
_Bool check(int s,int num,int lst)
{
    for(int i=1;i<=lst;i++)
        if(b[num][s+i])
            return 0;
    return 1;
}
int max(int a,int b){
    if (a>b) return a;
    else return b;
}
int main (){
    scanf("%d %d",&m,&n);
    for(int i=1;i<=m*n;i++)
        scanf("%d",&a[i]);
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            scanf("%d",&wk[i][j]);
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            scanf("%d",&t[i][j]);
}
for(int i=1;i<=n*m;i++)//枚举每个安排
{
    int x,y,z;//a[i] 工件号,x工序,y机器号,z开始时间
    x=++xu[a[i]];
    y=wk[a[i]][x];
    for(int j=gong[a[i]];;j++)
        if(check(j,y,t[a[i]][x]))
        {
            z=j;
            break;
        }
    for(int j=1;j<=t[a[i]][x];j++)
        b[y][z+j]=1;
    gong[a[i]]=z+t[a[i]][x];
}
for(int i=1;i<=n;i++)
    ans=max(ans,gong[i]);
printf("%d\n",ans);
return 0;
}
```

首先第一步就是处理输入：首先输入的是工件数量和工序数量
输入了确定的一个安排的顺序，存进数组a
然后输入每个工件使用的时候的机器的序号，存进wk数组
最后是每个工件每个工序所需要的时间，存进t数组
现在开始枚举各种安排了。

xu数组存的是当前工件的所在的工序数

a[i]表示现在讨论的工件,x是当前工序,y是工序所对应的使用机器,z是该工件工序的开始时间

b数组就是存储了对应机器时间的数,当某个工件某个工序结束的时候,机器所到的时间时间就会来到m时间,那么b[n][0]-b[n][m]会设置为1

比如说第一个工件的第一个工序需要时间2,机器1,这个时候,b[1][0]-b[1][1]-

b[1][2]就设置成1,表示这个时间段机器呗占用了

gong数组就是存储了每个工件工序完成了之后所到的时间数

```
for(int j=gong[a[i]];;j++)
    if(check(j,y,t[a[i]][x]))
    {
        z=j;
        break;
    }
}
```

这个for循环就是检查一下开始的时间,

先令j为上一个工序结束的时间(第一个默认为0)

调用check函数,检查一下可以开始的时间,如果check函数说可以啦,那就返回这个值

```
_Bool check(int s,int num,int lst)
{
    for(int i=1;i<=lst;i++)
        if(b[num][s+i])
            return 0;
    return 1;
}
```

s传进来的是目前打算确定的开始的时间,lst就是当前工序的耗时,num就是使用的机器序号!

就看,如果在这个时间段内该机器有被占用的情况,就是b[num][占用的时间序列],就返回不行(0),如果跳出for循环了,那就是可以了,返回1(可以)

```
for(int j=1;j<=t[a[i]][x];j++)
    b[y][z+j]=1;
```

这个循环就是确定占用了!

```
gong[a[i]]=z+t[a[i]][x];
```

确定过了这个工序,这个工件所需要的时间

```
for(int i=1;i<=n;i++)
    ans=max(ans,gong[i]);
```

找到gong的最大值

刷后有感:

1. 模拟的问题需要我们把问题简单地抽象为数学模型,然后对这个数学模型进行处理,这道题就是对一定的工件处理顺序,求出最小的时间,这就是一个类似于贪心的算法,尽量让他们的时间最小

2. 题目需要认真地读

3. 保存的思想,我们可以使用多个数组保存我们暂时求出来的一些值,在这里我们保存了工序的进程,保存了机器的使用时间,保存了暂时的工序时间

1-1-2 P1591 阶乘求和(高精度)

题目描述

[展开](#)

求 $n!$ 中某个数码出现的次数。

输入格式

第一行为 t ($t \leq 10$)，表示数据组数。接下来 t 行，每行一个正整数 n ($n \leq 1000$) 和数码 a 。

输出格式

对于每组数据，输出一个整数，表示 $n!$ 中 a 出现的次数。

输入输出样例

输入 #1

[复制](#)

输出 #1

[复制](#)

```
2
5 2
7 0
```

```
1
2
```

```
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<iostream>
#include<cmath>
#include<algorithm>
using namespace std;
struct number{
    int x[1000];
    int len;
    number(int a=0){
        if(a==0){
            memset(x,0,sizeof(x));
            len=0;
        }
        if(a==1){
            x[0]=1;
            len=1;
        }
    }
    void print(){
        for(int i=len-1;i>=0;i--)
            printf("%d",x[i]);
        printf("\n");
    }
};
number chengfa(number a,int n){
    number res;
    res.len=a.len;
    for(int i=0;i<res.len;i++){
```

```
        res.x[i]+=a.x[i]*n;
        res.x[i+1]+=res.x[i]/10;
        res.x[i]%=10;
    }
    while(res.x[res.len]){
        res.x[res.len+1]+=res.x[res.len]/10;
        res.x[res.len]%=10;
        res.len++;
    }
    return res;
}
number jiafa(number ans,number a){
    number res;
    res.len=max(ans.len,a.len);
    for(int i=0;i<res.len;i++){
        res.x[i]+=ans.x[i]+a.x[i];
        res.x[i+1]+=res.x[i]/10;
        res.x[i]%=10;
    }
    while(res.x[res.len]){
        res.x[res.len+1]+=res.x[res.len]/10;
        res.x[res.len]%=10;
        res.len++;
    }
    return res;
}
number getans(int n){//S=1!+2!+3!+i+n!
    number ans;
    for(int i=1;i<=n;i++){
        number res=1;
        for(int j=1;j<=i;j++)
            res=chengfa(res,j);
        ans=jiafa(ans,res);
    }
    return ans;
}
int main(){
    int i,j,k,m,n;
    scanf("%d",&n);
    number ans;
    ans=getans(n);
    ans.print();
    return 0;
}
```

1-1-3 P5143 攀爬者

这道题还是非常的简单, 首先先关键字排序, 将 z 按照由高到低的顺序排进去, 这个每个点就是一个结构体, 结构体里面存的是 x, y, z 的坐标
然后用 sqrt 函数计算就可以啦

题目背景

[\[\] 展开](#)

HKE考完GDOI之后跟他的神犇小伙伴们一起去爬山。

题目描述

他在地形图上标记了 N 个点, 每个点 P_i 都有一个坐标 (x_i, y_i, z_i) 。所有点对中, 高度值 z 不会相等。HKE准备从最低的点爬到最高的点, 他的攀爬满足以下条件:

- (1) 经过他标记的每一个点;
- (2) 从第二个点开始, 他经过的每一个点高度 z 都比上一个点高;
- (3) HKE会飞, 他从一个点 P_i 爬到 P_j 的距离为两个点的欧几里得距离。即,
$$\sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2 + (Z_i - Z_j)^2}$$

现在, HKE希望你能求出他攀爬的总距离。

输入格式

第一行, 一个整数 N 表示地图上的点数。

接下来 N 行, 三个整数 x_i, y_i, z_i 表示第 i 个点的坐标。

输出格式

一个实数, 表示HKE需要攀爬的总距离 (保留三位小数)

```
#include<cstdio>
#include<iostream>
#include<algorithm>
#include<cmath>
using namespace std;
struct weizhi
{
    int x,y,z;
}a[100005];
bool cmp(struct weizhi a,struct weizhi b)
{
    if(a.z<b.z)
```

```

        return 1;
    }
    else
    {
        return 0;
    }
}
int main()
{
    int n;
    double ans;
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i].x>>a[i].y>>a[i].z;
    }
    sort(a+1,a+n+1,cmp);
    for(int i=1;i<=n-1;i++)
    {
        ans+=sqrt((a[i+1].x-a[i].x)*(a[i+1].x-a[i].x)+(a[i+1].y-
a[i].y)*(a[i+1].y-a[i].y)+(a[i+1].z-a[i].z)*(a[i+1].z-a[i].z));
    }
    printf("%.3lf",ans);
    return 0;
}

```

1-1-4 P1518 两只塔姆沃斯牛

牛在地图里以固定的方式游荡。每分钟，它们可以向前移动或是转弯。如果前方无障碍（地图边沿也是障碍），它们会按照原来的方向前进一步。否则它们会用这一分钟顺时针转 90 度。同时，它们不会离开地图。

Farmer John 深知牛的移动方法，他也这么移动。

每次（每分钟）Farmer John 和两头牛的移动是同时的。如果他们在移动的时候穿过对方，但是没有在同一格相遇，我们不认为他们相遇了。当他们在某分钟末在某格子相遇，那么追捕结束。

读入十行表示地图。每行都只包含 10 个字符，表示的含义和上面所说的相同。保证地图中只有一个 **F** 和一个 **C**。**F** 和 **C** 一开始不会处于同一个格子中。

计算 Farmer John 需要多少分钟来抓住他的牛，假设牛和 Farmer John 一开始的行动方向都是正北（即上）。如果 John 和牛永远不会相遇，输出 0。

输入格式

输入共十行，每行 10 个字符，表示如上文描述的地图。

输出格式

输出一个数字，表示 John 需要多少时间才能抓住牛们。如果 John 无法抓住牛，则输出 0。

题目描述

[展开](#)

两只牛奔跑到了森林里。Farmer John 开始用他的专家技术追捕这两头牛。你的任务是模拟他们的行为（牛和 John）。

追击在 10×10 的平面网格内进行。一个格子可以是：一个障碍物，两头牛（它们总在一起），或者 Farmer John。两头牛和 Farmer John 可以在同一个格子内（当他们相遇时），但是他们都不能进入有障碍的格子。

一个格子可以是：

-  空地；
-  障碍物；
-  两头牛；
-  Farmer John。

这里有一个地图的例子：

```
*...*.....
.....*....
...*...*...
.....
...*.F....
*...*....
...*.....
..C.....*
...*...*...
.*...*.....
```

```
#include<cstdio>
#include<iostream>
#include<algorithm>
#include<cmath>
using namespace std;
typedef long long ll;
typedef pair<ll ,ll > P;
#define INF 0xf3f3f3f
const int Max=10000+10;
int dx[4]= {-1,0,1,0};
int dy[4]= {0,1,0,-1};
bool m[21][21],vis[21][21][4][21][21][4];
int fx,fy,cx,cy,ft,ct;
void dfs(int step) {
    if(fx==cx&&fy==cy) {
        printf("%d\n",step);
        return ;
    }
    if(vis[fx][fy][ft][cx][cy][ct]) {
        printf("0\n");
        return ;
    }
}
```

```

    }
    vis[fx][fy][ft][cx][cy][ct]=1;

    int tx=fx+dx[ft],ty=fy+dy[ft];
    if(!m[tx][ty])
        fx=tx,fy=ty;
    else
        ft=(ft+1)%4;
    tx=cx+dx[ct],ty=cy+dy[ct];
    if(!m[tx][ty])
        cx=tx,cy=ty;
    else
        ct=(ct+1)%4;
    dfs(step+1);
}

int main() {
    memset(m,0,sizeof m);//0 可以走 1有障碍
    memset(vis,0,sizeof vis);//1 标记
    char c;
    for(int i=0;i<=11;i++)
        m[0][i]=1,m[11][i]=1,m[i][0]=1,m[i][11]=1;
    for(int i=1; i<=10; i++)
        for(int j=1; j<=10; j++) {
            cin>>c;
            if(c=='C')
                cx=i,cy=j;
            if(c=='F')
                fx=i,fy=j;
            if(c=='*')
                m[i][j]=1;
        }
    ft=0,ct=0;
    dfs(0);

    return 0;
}

```

首先我们先来看看每个数组的作用,m这个数组存储的是障碍物的情况,vis这个数组存储的是有没有形成闭环的情况(真没有想到形成闭环就是找不到了),闭环是指:这个情况已经重复出现第二遍了六维数组,分别对应着两个动物的xy坐标和朝向

问:出现闭环时,会不会是一开始的情况呢?

首先来看看main函数,main函数首先确定边界,将四周射程障碍(边界扩增的方法)

然后导入地图数据

接着设置一开始的方向,全部为0

导入数据,第零次:其实不导入也可以.就开始搜索了.

(其实这个也算不上搜索)

先判断:如果追到了,输出当前的分钟数

追不到,就输出0,如果都不对,那就先把当前的状态设置为1

然后进行移动

判断有无撞到障碍物,撞到了就改变方向,没撞到就往前走,好了,递归调用,分钟数加1

1-1-5 P2670 扫雷

题目描述

[展开](#)

扫雷游戏是一款十分经典的单机小游戏。在 n 行 m 列的雷区中有一些格子含有地雷（称之为地雷格），其他格子不含地雷（称之为非地雷格）。玩家翻开一个非地雷格时，该格将会出现一个数字——提示周围格子中有多少个是地雷格。游戏的目标是在不翻出任何地雷格的条件下，找出所有的非地雷格。

现在给出 n 行 m 列的雷区中的地雷分布，要求计算出每个非地雷格周围的地雷格数。

注：一个格子的周围格子包括其上、下、左、右、左上、右上、左下、右下八个方向上与之直接相邻的格子。

输入格式

第一行是用一个空格隔开的两个整数 n 和 m ，分别表示雷区的行数和列数。

接下来 n 行，每行 m 个字符，描述了雷区中的地雷分布情况。字符 $*$ 表示相应格子是地雷格，字符 $?$ 表示相应格子是非地雷格。相邻字符之间无分隔符。

输出格式

输出文件包含 n 行，每行 m 个字符，描述整个雷区。用 $*$ 表示地雷格，用周围的地雷个数表示非地雷格。相邻字符之间无分隔符。

```
#include <iostream>
#include <cstdio>
#define N 103
void search(int i,int j);
char a[N][N];
int n,m;
int main() {
    scanf("%d %d",&n,&m);
    getchar();
    for (int i=0;i<n;i++) {
        for (int j=0;j<m;j++)
            scanf("%c",&a[i][j]);
        getchar();
    }
    for (int i=0;i<n;i++) {
        for (int j=0;j<m;j++) {
            if (a[i][j]=='*')
                printf("*");
            else
                search(i,j);
            if (j+1==m) printf("\n");
        }
    }
```

```

    }
    return 0;
}
void search(int i,int j)    {
    int flag=0;
    for (int di=-1;di<2;di++)
        for (int dj=-1;dj<2;dj++)
        {
            int x=i+di;
            int y=j+dj;
            if ((di!=0||dj!=0)&& x>=0&&x<n&&y>=0&&y<m)
                if (a[x][y]=='*')
                    flag++;
        }
    printf("%d",flag);
}

```

这道题很简单2333,就把原来的的数据存进去,search函数就是寻找周围的地雷的数量的函数,传参也是传位置

getchar();//这个语句就是为了吸收回车

```

for (int i=0;i<n;i++)    {
    for (int j=0;j<m;j++)    {
        if (a[i][j]=='*')
            printf("*");
        else
            search(i,j);
        if (j+1==m) printf("\n");
    }
}

```

地雷:*

遇到不是的就search一下输出周围8个空格*的个数

1-1-6 P1024 乒乓球

题目背景

[展开](#)

国际乒联现在主席沙拉拉自从上任以来就立志于推行一系列改革，以推动乒乓球运动在全球的普及。其中11分制改革引起了很大的争议，有一部分球员因为无法适应新规则只能选择退役。华华就是其中一位，他退役之后走上了乒乓球研究工作，意图弄明白11分制和21分制对选手的不同影响。在开展他的研究之前，他首先需要对他多年比赛的统计数据进行一些分析，所以需要你的帮忙。

题目描述

华华通过以下方式进行分析，首先将比赛每个球的胜负列成一张表，然后分别计算在11分制和21分制下，双方的比赛结果（截至记录末尾）。

比如现在有这么一份记录，（其中W表示华华获得一分，L表示华华对手获得一分）：

WWWWWWWWWWWWWWWWWWWWWWLW

在11分制下，此时比赛的结果是华华第一局11比0获胜，第二局11比0获胜，正在进行第三局，当前比分1比1。而在21分制下，此时比赛结果是华华第一局21比0获胜，正在进行第二局，比分2比1。如果一局比赛刚开始，则此时比分为0比0。直到分差大于或者等于2，才一局结束。

你的程序就是要对于一系列比赛信息的输入（WL形式），输出正确的结果。

输入格式

每个输入文件包含若干行字符串，字符串有大写的W、L和E组成。其中E表示比赛信息结束，程序应该忽略E之后的所有内容。

输出格式

输出由两部分组成，每部分有若干行，每一行对应一局比赛的比分（按比赛信息输入顺序）。其中第一部分是11分制下的结果，第二部分是21分制下的结果，两部分之间由一个空行分隔。

输入输出样例

输入 #1

复制

```
WWWWWWWWWWWWWWWWWWWWWWLW
WWLWE
```

输出 #1

复制

```
11:0
11:0
1:1

21:0
2:1
```

```
#include<cmath>
#include<iostream>
using namespace std;
main() {
    char c;
```

```

int a[10000][2], b=0, d=0, i, l=0, p=0, w=0;
while(1) {
    cin>>c;
    if(c=='E')
        break;
    if(c=='W') {
        w++;
        b++;
    } else if(c=='L') {
        l++;
        d++;
    }
    if((w>=11 || l>=11) && abs(w-l)>=2) {
        cout<<w<<':'<<l<<endl;
        w=0;
        l=0;
    }
    if((b>=21 || d>=21) && abs(b-d)>=2) {
        a[++p][0]=b;
        a[p][1]=d;
        b=0;
        d=0;
    }
}
cout<<w<<':'<<l<<"\n\n";
for(i=1; i<=p; i++)
    cout<<a[i][0]<<':'<<a[i][1]<<endl;
cout<<b<<':'<<d;
}

```

首先存c的数据,如果是赢:就是胜场加一,如果是输:输场加一,wl代表11局制,bd代表21局制,

```

if((w>=11 || l>=11) && abs(w-l)>=2) {
    cout<<w<<':'<<l<<endl;
    w=0;
    l=0;
}
if((b>=21 || d>=21) && abs(b-d)>=2) {
    a[++p][0]=b;
    a[p][1]=d;
    b=0;
    d=0;
}

```

一局结束,输出结果

```

cout<<w<<':'<<l<<"\n\n";
for(i=1; i<=p; i++)
    cout<<a[i][0]<<':'<<a[i][1]<<endl;
cout<<b<<':'<<d;

```

最后的处理完了,就输出剩下的胜场和负场

1-1-7 P1303 A*B Problem

题目描述

[展开](#)

求两数的积。

输入格式

两行，两个整数。

输出格式

一行一个整数表示乘积。

输入输出样例

输入 #1

[复制](#)

输出 #1

[复制](#)

```
1
2
```

```
2
```

说明/提示

每个数字不超过 10^{2000} ，需用高精。

在这里复习一下高精度乘法的做法

```
#include<stdio.h>
```

```
#include<string.h>
```

```
int main() {
    char number1[1500], number2[1500];
    scanf("%s %s", number1, number2);
    int n = strlen(number1), m = strlen(number2);
    int a[n], b[m];
```

//首先输入两个字符串,将字符串存进去,求出字符串位数

```
    int i, j;
    for (i = 0, j = n - 1; i < n; i++, j--) {
        a[i] = number1[j] - '0';
    }
    for (i = 0, j = m - 1; i < m; i++, j--) {
        b[i] = number2[j] - '0';
    }
```

//将字符数组转化成整数的数组

```
    int c[3000];
    for (i = 0; i < 3000; i++) {
        c[i] = 0;
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            c[i + j] += a[i] * b[j];
        }
    }
```

```
    }  
}  
//逐位的进行乘法运算  
for (i = 0; i < n + m; i++) {  
    if (c[i] >= 10) {  
        c[i + 1] += c[i] / 10;  
        c[i] %= 10;  
    }  
}  
//进位  
for (j = 2999; j > 0; j--) {  
    if (c[j] != 0)  
        break;  
}  
for (i = j; i >= 0; i--) {  
    printf("%d", c[i]);  
}  
printf("\n");  
return 0;  
}
```

1-1-8 P1601 A+B Problem(高精)

题目背景

[展开](#)

无

题目描述

高精度加法,相当于 $a+b$ problem, 不用考虑负数.

输入格式

分两行输入。 $a, b \leq 10^{500}$

输出格式

输出只有一行, 代表 $a + b$ 的值

输入输出样例

输入 #1

[复制](#)

输出 #1

[复制](#)

```
1  
1
```

```
2
```

```
#include<stdio.h>  
#include<string.h>  
int a[1500], b[1500];  
int main() {
```

```
char number1[1500], number2[1500];
scanf("%s %s", number1, number2);
int n = strlen(number1), m = strlen(number2);
//首先输入两个字符串,将字符串存进去,求出字符串位数
int i, j;
for (i = 0, j = n - 1; i < n; i++, j--) {
    a[i] = number1[j] - '0';
}
for (i = 0, j = m - 1; i < m; i++, j--) {
    b[i] = number2[j] - '0';
}
//将字符数组转化成整数的数组
int c[3000];
for (i = 0; i < 3000; i++) {
    c[i] = 0;
}
if (m > n) {
    int a;
    a = m;
    m = n;
    n = a;
}
for (i = 0; i < n; i++) {
    c[i] += a[i] + b[i];
}
//逐位的进行乘法运算
for (i = 0; i < n; i++) {
    if (c[i] >= 10) {
        c[i + 1] += c[i] / 10;
        c[i] %= 10;
    }
}
//进位
for (j = 2999; j > 0; j--) {
    if (c[j] != 0)
        break;
}
for (i = j; i >= 0; i--) {
    printf("%d", c[i]);
}
printf("\n");
return 0;
}
```

对于数组的处理:用memset也是可以的,我在这用全局变量使其默认为0!

1-1-9 P1009 阶乘之和

题目描述

[展开](#)

用高精度计算出 $S = 1! + 2! + 3! + \dots + n!$ ($n \leq 50$)

其中“!”表示阶乘，例如： $5! = 5 \times 4 \times 3 \times 2 \times 1$ 。

输入格式

一个正整数 N 。

输出格式

一个正整数 S ，表示计算结果。

输入输出样例

输入 #1

[复制](#)

输出 #1

[复制](#)

3

9

```
#include<stdio.h>
#define maxn 10000
int a[maxn],sum[maxn];
int main()
{
    int n,k,i,j,t;
    while(scanf("%d",&n)!=EOF){
        sum[0]=0;
        k=1,t=1;
        for(int e=1;e<=n;e++){
            a[0]=1;
            for(i=1;i<=e;i++){
                for(j=0;j<k;j++){
                    a[j]=a[j]*i;//
                }
                for(j=0;j<k;j++){
                    if(a[j]>=10){
                        a[j+1]+=a[j]/10;
                        a[j]=a[j]%10;
                        if(j==k-1) k++;//
                    }
                }
            }
        }
        for(int p=0;p<k;p++){
            sum[p]=sum[p]+a[p];
        }
        for(int p=0;p<t;p++){
```

```
        if(sum[p+1]>=10){
            t++;
        }
        if(sum[p]>=10){
            sum[p+1]+=sum[p]/10;
            sum[p]=sum[p]%10;
        }
    }
    for(int q=1;q<k;q++){
        a[q]=0;
    }
}
for(i=k-1;i>=0;i--){
    printf("%d",sum[i]);
}
printf("\n");
}
```

这个阶乘的方法就是加法和乘法之间的结合

这个是高精度乘低精度的代码

```
for(i=1;i<=e;i++){
    for(j=0;j<k;j++){
        a[j]=a[j]*i;//
    }
    for(j=0;j<k;j++){
        if(a[j]>=10){
            a[j+1]+=a[j]/10;
            a[j]=a[j]%10;
            if(j==k-1) k++;
        }
    }
}
```

这里k是阶乘的位数,t是总和的位数

其实还可以这样:

从最后往前遍历,到了第一个非零数就标记位数

1-1-10 P4924 魔法少女芙兰朵露-斯卡雷特

题目描述

[展开](#)

Scarlet最近学会了一个数组魔法，她会在 $n * n$ 二维数组上将一个奇数阶方阵按照顺时针或者逆时针旋转 90° ,

首先，Scarlet会把1到 n^2 的正整数按照从左往右，从上至下的顺序填入初始的二维数组中，然后她会施放一些简易的魔法。

Scarlet既不会什么分块特技，也不会什么Splay套Splay，她现在提供给你她的魔法执行顺序，想让你来告诉她魔法按次执行完毕后的二维数组。

输入格式

第一行两个整数 n, m ，表示方阵大小和魔法施放次数。

接下来 m 行，每行4个整数 x, y, r, z ，表示在这次魔法中，Scarlet会把以第 x 行第 y 列为中心的 $2r + 1$ 阶矩阵按照某种时针方向旋转，其中 $z = 0$ 表示顺时针， $z = 1$ 表示逆时针。

输出格式

输出 n 行，每行 n 个用空格隔开的数，表示最终所得的矩阵

```
#include<stdio.h>
int N,M,x,y,r,z,tmp=1;
int base[505][505]={};
int ans[505][505]={};
int main()
{
    scanf("%d%d",&N,&M);
    for(int i=1;i<=N;++i)
    {
        for(int j=1;j<=N;++j)
        {
            ans[i][j]=tmp++;
        }
    }
    for(int T=1;T<=M;++T)
    {
        scanf("%d%d%d%d",&x,&y,&r,&z);

        if(z==0)
        for(int i=x-r;i<=x+r;++i)
        for(int j=y-r;j<=y+r;++j)
        {
            int ti=i-(x-r)+1,tj=j-(y-r)+1;
            base[tj][r+r+2-ti]=ans[i][j];
        }
    }
}
```

```

else
for(int i=x-r;i<=x+r;++i)
for(int j=y-r;j<=y+r;++j)
{
    int ti=i-(x-r)+1,tj=j-(y-r)+1;
    base[r+r+2-tj][ti]=ans[i][j];
}

for(int i=x-r;i<=x+r;++i)
for(int j=y-r;j<=y+r;++j)
ans[i][j]=base[i-x+r+1][j-y+r+1];
}
for(int i=1;i<=N;++i)
{
    for(int j=1;j<=N;++j)
    {
        printf("%d ",ans[i][j]);
    }
    printf("\n");
}
return 0;
}

```

N方阵大小,M是操作次数

x,y坐标,r魔法范围,z:顺时针逆时针

当然,进行旋转也可以使用简单的旋转操作算法也是可以的

```

if(z==0){
    for(int i=x-r;i<=x+r;i++){
        for(int j=y-r;j<=y+r;j++){
            a[x-y+j][x+y-i]=b[i][j];
        }
    }
}
else{
    for(int i=x-r;i<=x+r;i++){
        for(int j=y-r;j<=y+r;j++){
            a[x+y-j][y-x+i]=b[i][j];
        }
    }
}
}

```

1-1-11 P1563 玩具谜题

题目描述

[收起](#)

小南有一套可爱的玩具小人, 它们各有不同的职业。

有一天, 这些玩具小人把小南的眼镜藏了起来。小南发现玩具小人们围成了一个圈, 它们有的面朝圈内, 有的面朝圈外。如下图:



这时 *singer* 告诉小南一个谜题: “眼镜藏在我左数第3个玩具小人的右数第1个玩具小人的左数第2个玩具小人那里。”

小南发现, 这个谜题中玩具小人的朝向非常关键, 因为朝内和朝外的玩具小人的左右方向是相反的: 面朝圈内的玩具小人, 它的左边是顺时针方向, 右边是逆时针方向; 而面向圈外的玩具小人, 它的左边是逆时针方向, 右边是顺时针方向。

小南一边艰难地辨认着玩具小人, 一边数着:

singer 朝内, 左数第3个是 *archer*。

archer 朝外, 右数第1个是 *thinker*。

thinker 朝外, 左数第2个是 *writer*。

所以眼镜藏在 *writer* 这里!

虽然成功找回了眼镜, 但小南并没有放心。如果下次有更多的玩具小人藏他的眼镜, 或是谜题的长度更长, 他可能就无法找到眼镜了。所以小南希望你写程序帮他解决类似的谜题。这样的谜题具体可以描述为:

有 n 个玩具小人围成一圈, 已知它们的职业和朝向。现在第1个玩具小人告诉小南一个包含 m 条指令的谜题, 其中第 z 条指令形如 “左数/右数第 s_z 个玩具小人”。你需要输出依次数完这些指令后, 到达的玩具小人的职业。

输入格式

输入的第一行包含两个正整数 n, m , 表示玩具小人的个数和指令的条数。

接下来 n 行, 每行包含一个整数和一个字符串, 以逆时针为顺序给出每个玩具小人的朝向和职业。其中 0 表示朝向圈内, 1 表示朝向圈外。保证不会出现其他的数。字符串长度不超过 10 且仅由小写字母构成, 字符串不为空, 并且字符串两两不同。整数和字符串之间用一个空格隔开。

接下来 m 行, 其中第 i 行包含两个整数 a_i, s_i , 表示第 i 条指令。若 $a_i = 0$, 表示向左数 s_i 个人; 若 $a_i = 1$, 表示向右数 s_i 个人。保证 a_i 不会出现其他的数, $1 \leq s_i < n$ 。

输出格式

输出一个字符串, 表示从第一个读入的小人开始, 依次数完 m 条指令后到达的小人的职业。

输入输出样例

输入 #1

[复制](#)

输出 #1

[复制](#)

```
7 3
0 singer
0 reader
0 mengbier
1 thinker
1 archer
0 writer
1 mogician
0 3
1 1
0 2
```

```
writer
```

```
#include<stdio.h>
```

```
int main(){
    int n,m,i,j,t,s,a[100005];
    //a存储朝向
    char b[100005][15];
    //b存储名字
    scanf("%d%d",&n,&m);
    //mn存储个数和步数
    for(i=1;i<=n;i++){
        scanf("%d",&a[i]);
        scanf("%s",b[i]);
    }
    //存数据
    int sum=1;
    //首先是第一个
    for(i=1;i<=m;i++){
        scanf("%d%d",&t,&s);
        //存储顺时针还是逆时针以及步数
        j=t;j+=a[sum];//这个是确定
        if(j%2==0){
            sum-=s;//往左走
            if(sum<0)
                sum+=n;//如果小于0的话,加一个n
        }
        else sum+=s;//往右走
        if(sum>n)
            sum-=n;//如果大于n的话,减一个n
    }
    printf("%s\n",b[sum]);
    return 0;
}
```

1-1-12 P5515 灵梦的计算器(迷途之家OI比赛)

这道题有点难

题目背景

[🔗 收起](#)

注: 该背景部分改编自 [disangan233](#) 中考前买计算器的真实事件。

博丽 灵梦 (Hakurei Reimu) 在成功抢回八云 紫 (Yakumo Yukari) 用隙间偷走的香火钱后, 她和依神 紫苑 (Yorigami Shion) 去香霖堂买东西啦!

灵梦想买一个计算器来计算神社的香火钱, 但是因为香霖堂的东西太贵了, 她选择使用河童重工网络 (Kawashiro Nitori's Network,KNN) 网购一个 Casio 计算器。

但出人意料的是, 灵梦使用 KNN 买回来的 Casio 是个假货, 最多只能显示整数部分 (即向下取整)。

灵梦很苦恼, 因为这个计算器可能会导致一些特别大的误差。所以灵梦想让拥有外界的式神(指电脑)的你帮她解决一个问题。

题目描述

灵梦得到了3个实数 n , a , b ($4 \leq n \leq 5, 5 \leq a, b \leq 10$), 她成功地计算了 $n^a + n^b$, 得到了一个只显示整数部分的结果。

灵梦想知道, 若存在一个实数 n' ($n' \geq 0$), 使得 $n'^a + n'^b$ 的结果在计算器上与 $n^a + n^b$ 的结果显示出来完全一致时, n' 的变化范围, 即 n' 的最大值与最小值之差。

如果你不知道如何计算 n^k , 请使用 `cmath` 库的 `pow()` 函数, `pow(n,k)` 的结果即为 n^k 的结果。

为了提高本题的难度，灵梦给你设置了 T 组询问。而为了在某种程度上减少你的输入和输出量，我们采用以下的代码来生成询问(代码来自河童重工)：

```
namespace Mker
{
// Powered By Kawashiro_Nitori
// Made In Gensokyo, Nihon
#define uint unsigned int
uint sd;int op;
inline void init() {scanf("%u %d", &sd, &op);}
inline uint uint_rand()
{
    sd ^= sd << 13;
    sd ^= sd >> 7;
    sd ^= sd << 11;
    return sd;
}
inline double get_n()
{
    double x = (double) (uint_rand() % 100000) / 100000;
    return x + 4;
}
inline double get_k()
{
    double x = (double) (uint_rand() % 100000) / 100000;
    return (x + 1) * 5;
}
inline void read(double &n,double &a, double &b)
{
    n = get_n(); a = get_k();
    if (op) b = a;
    else b = get_k();
}
}
```

在调用 `Mker::init()` 函数之后，你第 i 次调用 `Mker::read(n,a,b)` 函数后得到的便是第 i 次询问的 n_i, a_i 和 b_i 。

为了减少你的输出量，令第 i 次询问的答案为 s_i ，你只需要输出 $\sum_{i=1}^T s_i$ 。如果你的答案与标准答案的绝对误差在 10^{-2} 以内，你的答案则被视为是正确答案。

本题数据的生成采用时间复杂度远远劣于普通算法的高 (da) 精 (bao) 度 (li) 算法来保证精度，本题数据保证单次询问的误差小于 10^{-10} ，所以本题的SPJ范围对于正解来说是完全足够的。

为了让你更好地做题，这里给出了关于 op 的说明：

- 当 $op = 1$ 时，有 $a = b$ ，否则无特殊限定。

输入格式

输入共一行，包含 3 个正整数 $T, seed, op$ ，含义见题目描述。

输出格式

输出共一行，输出题目描述中要求输出的答案。

```
#include<cstdio>
#include<cmath>
using namespace std;
int t;
double n,a,b;
namespace Mker
{
// Powered By Kawashiro_Nitori
// Made In Gensokyo, Nihon
#define uint unsigned int
uint sd;int op;
inline void init() {scanf("%u %d", &sd, &op);}
inline uint uint_rand()
{
    sd ^= sd << 13;
    sd ^= sd >> 7;
```

```

        sd ^= sd << 11;
        return sd;
    }
    inline double get_n()
    {
        double x = (double) (uint_rand() % 100000) / 100000;
        return x + 4;
    }
    inline double get_k()
    {
        double x = (double) (uint_rand() % 100000) / 100000;
        return (x + 1) * 5;
    }
    inline void read(double &n, double &a, double &b)
    {
        n = get_n(); a = get_k();
        if (op) b = a;
        else b = get_k();
    }
}
int main()
{
    scanf("%d", &t);
    Mker::init();
    double ans=0;
    for (int i=1; i<=t; i++)
    {
        Mker::read(n, a, b);
        ans+=1/(a*pow(n, a-1)+b*pow(n, b-1));
    }
    printf("%.3lf", ans);
    return 0;
}

```

既然只用迭代一次，那么我们可以直接得到 n_0, n_1 的公式：

$$n_0 = n - \frac{f(n) - k}{f'(n)}$$

$$n_1 = n - \frac{f(n) - (k + 1)}{f'(n)}$$

$$n_1 - n_0 = \frac{1}{f'(n)} = \frac{1}{an^{a-1} + bn^{b-1}}$$

那么单次询问答案就为

我们已经知道与十分接近，因此我们可以把图像画出来，把段看成直线。

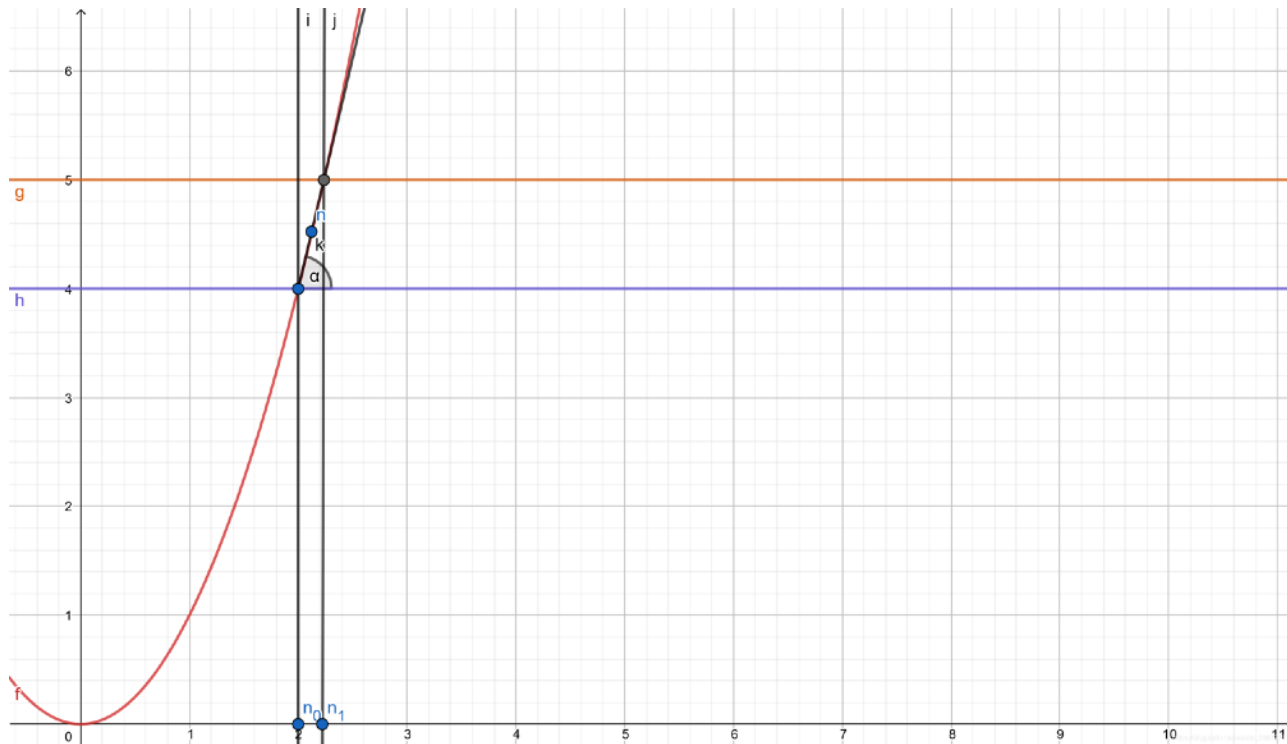
为什么可以看成直线呢？

如果你是高中生，应该在数学课上学定积分时学过“化曲为直”的思想，当时为了方便计算“曲边梯形”的面积，我们在很小的一个区间内把曲线看成直线从而转换为普通的梯形求其面积。

当然上述“化曲为直”的依据是区间无限小，在这里，我们的区间只能说是很小，但并不是无限小。

那么在本题中，“化曲为直”依据是什么呢？

精度。



题目的精度要求不高，因此可以粗略地把曲线看成直线。（当然即使这样，这种方法还是比较冒险，我个人不建议用）

那么由三角函数知识，在图中的直角三角形内，

$$n_1 - n_0 = (f(n_1) - f(n_0)) \times \tan \alpha = (f(n_1) - f(n_0)) \times f'(n) = \frac{1}{an^{a-1} + bn^{b-1}}。$$

1-1-13 P1067 多项式输出

题目描述

[展开](#)

一元 n 次多项式可用如下的表达式表示：

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0, a_n \neq 0$$

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0, a_n \neq 0$$

其中， $a_i x^i$ 称为 i 次项， a_i 称为 i 次项的系数。给出一个一元多项式各项的次数和系数，请按照如下规定的格式要求输出该多项式：

1. 多项式中自变量为 x ，从左到右按照次数递减顺序给出多项式。
2. 多项式中只包含系数不为0的项。
3. 如果多项式 n 次项系数为正，则多项式开头不出现“+”号，如果多项式 n 次项系数为负，则多项式以“-”号开头。
- 4.

对于不是最高次的项，以“+”号或者“-”号连接此项与前一项，分别表示此项系数为正或者系数为负。紧跟一个正整数，表示此项系数的绝对值（如果一个高于0次的项，其系数的绝对值为1，则无需输出1）。如果 x 的指数大于1，则接下来紧跟的指数部分的形式为“ x^b ”，其中 b 为 x 的指数；如果 x 的指数为1，则接下来紧跟的指数部分形式为“ x ”；如果 x 的指数为0，则仅需输出系数即可。

- 5.

多项式中，多项式的开头、结尾不含多余的空格。

输入格式

输入共有2行

第一行1个整数， n ，表示一元多项式的次数。

第二行有 $n+1$ 个整数，其中第 i 个整数表示第 $n-i+1$ 次项的系数，每两个整数之间用空格隔开。

输出格式

输出共1行，按题目所述格式输出多项式。

模拟题,注意一下特判!

一个是第一个

一个是倒数第二项,分成5种情况

倒数第一项,3种情况

普通项,5种情况

```
#include<stdio.h>
int xishu[100];
int main (){
    int N;
    scanf("%d",&N);
```

```
    for(int i=0;i<N+1;i++){
        scanf("%d",&xishu[i]);
    }
    if(xishu[0]==1) printf("x^%d",N);
    else if(xishu[0]!=-1) printf("%dx^%d",xishu[0],N);
    else printf("-x^%d",N);
    for(int i=1;i<N-1;i++){
        if(xishu[i]>1){
            printf("+%dx^%d",xishu[i],N-i);
        }
        else if(xishu[i]==1){
            printf("+x^%d",N-i);
        }
        else if(xishu[i]==-1){
            printf("-x^%d",N-i);
        }
        else if(xishu[i]<-1){
            printf("-%dx^%d",-xishu[i],N-i);
        }
        else{
        }
    }
    if(xishu[N-1]>0) {
        if(xishu[N-1]==1) printf("+x");
        else printf("+%dx",xishu[N-1]);
    }
    else if(xishu[N-1]==0) ;
    else {
        if(xishu[N-1]==-1) printf("-x");
        else printf("-%dx",-xishu[N-1]);
    }
    if(xishu[N]>0) printf("+%d",xishu[N]);
    else if(xishu[N]==0) ;
    else printf("-%d",-xishu[N]);
    printf("\n");
}
```

就是特判要小心！

1-1-14 P1098 字符串的展开

题目描述

[展开](#)

在初赛普及组的“阅读程序写结果”的问题中，我们曾给出一个字符串展开的例子：如果在输入的字符串中，含有类似于“d-h”或者“4-8”的字串，我们就把它当作一种简写，输出时，用连续递增的字母或数字串替代其中的减号，即，将上面两个子串分别输出为“defgh”和“45678”。在本题中，我们通过增加一些参数的设置，使字符串的展开更为灵活。具体约定如下：

(1) 遇到下面的情况需要做字符串的展开：在输入的字符串中，出现了减号“-”，减号两侧同为小写字母或同为数字，且按照ASCII码的顺序，减号右边的字符严格大于左边的字符。

(2) 参数 p_1 ：展开方式。 $p_1 = 1$ 时，对于字母子串，填充小写字母； $p_1 = 2$ 时，对于字母子串，填充大写字母。这两种情况下数字子串的填充方式相同。 $p_1 = 3$ 时，不论是字母子串还是数字子串，都用与要填充的字母个数相同的星号“*”来填充。

(3) 参数 p_2 ：填充字符的重复个数。 $p_2 = k$ 表示同一个字符要连续填充 k 个。例如，当 $p_2 = 3$ 时，子串“d-h”应扩展为“deeefffgggh”。减号两边的字符不变。

(4) 参数 p_3 ：是否改为逆序： $p_3 = 1$ 表示维持原来顺序， $p_3 = 2$ 表示采用逆序输出，注意这时候仍然不包括减号两端的字符。例如当 $p_1 = 1$ 、 $p_2 = 2$ 、 $p_3 = 2$ 时，子串“d-h”应扩展为“dggffeeh”。

(5) 如果减号右边的字符恰好是左边字符的后继，只删除中间的减号，例如：“d-e”应输出为“de”，“3-4”应输出为“34”。如果减号右边的字符按照ASCII码的顺序小于或等于左边字符，输出时，要保留中间的减号，例如：“d-d”应输出为“d-d”，“3-1”应输出为“3-1”。

输入格式

共两行。

第1行为用空格隔开的3个正整数，依次表示参数 p_1, p_2, p_3 。

第2行为一行字符串，仅由数字、小写字母和减号“-”组成。行首和行末均无空格。

输出格式

共一行，为展开后的字符串。

tolower和toupper函数懒得写了

```
#include<string.h>
#include<stdio.h>
char num[10000];
int main()
{
    int p1,p2,p3;
    while(scanf("%d%d%d",&p1,&p2,&p3))
        //输入p1,p2,p3
    {
        scanf("%s",num);
        int l=strlen(num);
        //找到长度
```

```

for(int i=0; i<l; ++i)
    //每个每个得遍历
{
    if(i==0||i==l-1)
    {
        printf("%c",num[i]);
        continue;
        //不是减号,放它一马
    }
    if(num[i]=='-')
    {
        //是减号

if(num[i-1]>='a'&&num[i-1]<='z'&&num[i+1]>='a'&&num[i+1]<='z' ||
num[i-1]>='A'&&num[i-1]<='Z'&&num[i+1]>='A'&&num[i+1]<='Z')//如果是
大写字母小写字母就进入
    {
        if(num[i-1]<num[i+1])//减号左右是左小右大,就进入
        {
            if(p3==1)//p3为1,正序
            for(int j=num[i-1]-'0'+1;
j<=num[i+1]-'0'-1; ++j)
            {
                for(int k=1; k<=p2; ++k)
                {
                    if(p1==1)

printf("%c",tolower(j+'0')); //输出小写字母
                    //
                    else if(p1==2)

printf("%c",toupper(j+'0')); //输出大写字母
                    else
                        printf("*");
                    //如果p1==3,那就输出*
                }
            }
        }
        else
            for(int j=num[i+1]-'0'-1;
j>=num[i-1]-'0'+1; --j)
            {
                for(int k=1; k<=p2; ++k)
                {
                    if(p1==1)

printf("%c",tolower(j+'0'));
                    else if(p1==2)

printf("%c",toupper(j+'0'));
                }
            }
        }
    }
}

```

```

        else
            printf("*");
    }
}
}
else{
    printf("-");
    continue;
    //如果不满足的话,那就是直接输出为-即可
}
}

else
if(num[i-1]>='0'&&num[i-1]<='9'&&num[i+1]>='0'&&num[i+1]<='9')
{ //如果是数字那就进入
    if(num[i-1]<num[i+1]) //左边的数字比右边的小
    {
        if(p3==1)
            for(int j=num[i-1]-'0'+1;
j<=num[i+1]-'0'-1; ++j)
            {
                for(int k=1; k<=p2; ++k)
                {
                    if(p1==1||p1==2)
                        printf("%c",j+'0');
                    //无论如何就打印数字
                    else if(p1==3)
                        printf("*");
                    //p1为3的时候输出*
                }
            }
        else if(p3==2)
            for(int j=num[i+1]-'0'-1;
j>=num[i-1]-'0'+1; --j)
            {
                for(int k=1; k<=p2; ++k)
                {
                    if(p1==1||p1==2)
                        printf("%c",j+'0');
                    else if(p1==3)
                        printf("*");
                }
            }
    }
}
else
{
    printf("-");
    continue;
}
}
}

```

```
        else{
            printf("-");
            continue;
        }
    }
    else
        printf("%c", num[i]);
}
printf("\n\n");
}
return 0;
}
```