ACM 竞赛

常用算法模板(系列一)

目　录

三角形面积计算

```
//已知三条边和外接圆半径，公式为 s = a*b*c/(4*R)
double GetArea(double a, double b, double c, double R)
{
return a*b*c/4/R;
}
//已知三条边和内接圆半径，公式为 s = pr
double GetArea(double a, double b, double c, double r)
{
return r*(a+b+c)/2;
}

//已知三角形三条边，求面积
double GetArea(doule a, double b, double c)
{
double p = (a+b+c)/2;
return sqrt(p*(p-a)*(p-b)*(p-c));
}

//已知道三角形三个顶点的坐标

struct Point
{
double x, y;
Point(double a = 0, double b = 0)
{
     x = a; y = b;
}
};
double GetArea(Point p1, Point p2, Point p3)
{
double t =
-p2.x*p1.y+p3.x*p1.y+p1.x*p2.y-p3.x*p2.y-p1.x*p3.y+p2.x*p3.y;
if(t < 0) t = -t;
return t/2;
}
```

字典树模板

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

#define BASE_LETTER 'a'
#define MAX_TREE 35000
#define MAX_BRANCH 26

struct
{

	int next[MAX_BRANCH]; //记录分支的位置
	int c[MAX_BRANCH]; //查看分支的个数

	int flag; //是否存在以该结点为终止结点的东东，可以更改为任意的
属性
}trie[MAX_TREE];

int now;

void init()
{
now = 0;
memset(&trie[now], 0, sizeof(trie[now]));
	now ++;
}

int add ()
{
	memset(&trie[now], 0, sizeof(trie[now]));
	return now++;
}

int insert( char *str)
{

	int pre = 0, addr;
	while( *str != 0 )
	{
		addr = *str - BASE_LETTER;
		if( !trie[pre].next[addr] )
			trie[pre].next[addr] = add();
```

```
                trie[pre].c[addr]++;
                pre = trie[pre].next[addr];
                str ++;
        }

        trie[pre].flag = 1;

        return pre;
}

int search( char *str )
{

        int pre = 0, addr;
        while( *str != 0 )
        {
                addr = *str - BASE_LETTER;
                if ( !trie[pre].next[addr] )
                        return 0;
                pre = trie[pre].next[addr];
                str ++;
        }
        if( !trie[pre].flag )
                return 0;

        return pre;
}
```

pku2001 题，源代码：

```
void check( char *str )
{
int pre = 0, addr;
while(*str != 0)
{
     addr = *str - BASE_LETTER;
                if( trie[pre].c[addr] == 1)
     {
       printf("%c\n", *str);
       return;
     }

     printf("%c", *str);
                pre = trie[pre].next[addr];
```

```
                str ++;
}
printf("\n");
}

char input[1001][25];

int main()
{
int i = 0,j;
init();
while(scanf("%s", input[i]) != EOF)
{
     getchar();
     insert(input[i]);
     i++;
}

for(j = 0; j < i; j ++)
{
     printf("%s ", input[j]);
     check(input[j]);
}
return 0;
}
```

求线段所在直线

```
//*****************************线段所在的直线
struct Line
{
double a, b, c;
};
struct Point
{
double x, y;
}
Line GetLine(Point p1, Point p2)
{
//ax+by+c = 0 返回直线的参数
Line line;
line.a = p2.y - p1.y;
line.b = p1.x - p2.x;
line.c = p2.x*p1.y - p1.x*p2.y;
```

```
return line;
}
```

求外接圆

```
//***************已知三角形三个顶点坐标，求外接圆的半径和坐标
********************
struct Point
{
double x, y;
Point(double a = 0, double b = 0)
{
     x = a; y = b;
}
};
struct TCircle
{
double r;
Point p;
}

double distance(Point p1, Point p2)
{
return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}
double GetArea(doule a, double b, double c)
{
double p = (a+b+c)/2;
return sqrt(p*(p-a)*(p-b)*(p-c));
}
TCircle GetTCircle(Point p1, Point p2, Point p3)
{
double a, b, c;
double xa,ya, xb, yb, xc, yc, c1, c2;
TCircle tc;

a = distance(p1, p2);
b = distance(p2, p3);
c = distance(p3, p1);

//求半径
tc.r = a*b*c/4/GetArea(a, b, c);
```

```
//求坐标
xa = p1.x; ya = p1.b;
xb = p2.x; yb = p2.b;
xc = p3.x; yc = p3.b;

c1 = (xa*xa + ya*ya - xb*xb - yb*yb)/2;
c2 = (xa*xa + ya*ya - xc*xc - yc*yc)/2;

tc.p.x = (c1*(ya-yc) - c2*(ya-yb))/((xa-xb)*(ya-yc) - (xa-xc)*(ya-yb));
tc.p.y = (c1*(xa-xc) - c2*(xa-xb))/((ya-yb)*(xa-xc) - (ya-yc)*(xa-xb));

return tc;
}
```

求内接圆

```
struct Point
{
double x, y;
Point(double a = 0, double b = 0)
{
     x = a; y = b;
}
};
struct TCircle
{
double r;
Point p;
}

double distance(Point p1, Point p2)
{
return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}
double GetArea(doule a, double b, double c)
{
double p = (a+b+c)/2;
return sqrt(p*(p-a)*(p-b)*(p-c));
}
TCircle GetTCircle(Point p1, Point p2, Point p3)
{
double a, b, c;
double xa,ya, xb, yb, xc, yc, c1, c2, f1, f2;
double A,B,C;
```

```
TCircle tc;

a = distance(p1, p2);
b = distance(p3, p2);
c = distance(p3, p1);

//求半径
tc.r = 2*GetArea(a, b, c)/(a+b+c);

//求坐标
A = acos((b*b+c*c-a*a)/(2*b*c));
B = acos((a*a+c*c-b*b)/(2*a*c));
C = acos((a*a+b*b-c*c)/(2*a*b));

p = sin(A/2); p2 = sin(B/2); p3 = sin(C/2);

xb = p1.x; yb = p1.b;
xc = p2.x; yc = p2.b;
xa = p3.x; ya = p3.b;

f1 = ( (tc.r/p2)*(tc.r/p2) - (tc.r/p)*(tc.r/p) + xa*xa - xb*xb + ya*ya
- yb*yb)/2;
f2 = ( (tc.r/p3)*(tc.r/p3) - (tc.r/p)*(tc.r/p) + xa*xa - xc*xc + ya*ya
- yc*yc)/2;

tc.p.x = (f1*(ya-yc) - f2*(ya-yb))/((xa-xb)*(ya-yc)-(xa-xc)*(ya-yb));
tc.p.y = (f1*(xa-xc) - f2*(xa-xb))/((ya-yb)*(xa-xc)-(ya-yc)*(xa-xb));

return tc;
}
```

判断点是否在直线上

```
//***************判断点是否在直线上**********************
//判断点 p 是否在直线[p1,p2]
struct Point
{
double x,y;
};
bool isPointOnSegment(Point p1, Point p2, Point p0)
{
//叉积是否为 0，判断是否在同一直线上
if((p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y) != 0)
   return false;
```

```
//判断是否在线段上
if((p0.x > p1.x && p0.x > p2.x) || (p0.x < p1.x && p0.x < p2.x))
   return false;
if((p0.y > p1.y && p0.y > p1.y) || (p0.y < p1.y && p0.y < p2.y))
   return false;
return true;
}
```

简单多边形面积计算公式

```
struct Point
{
double x, y;
Point(double a = 0, double b = 0)
{
     x = a; y = b;
}
};


Point pp[10];

double GetArea(Point *pp, int n)
{//n 为点的个数，pp 中记录的是点的坐标

int i = 1;
double t = 0;

for(; i <= n-1; i++)
     t += pp[i-1].x*pp[i].y - pp[i].x*pp[i-1].y;
t += pp[n-1].x*pp[0].y - pp[0].x*pp[n-1].y;

if(t < 0) t = -t;
return t/2;
}
```

stein 算法求最大共约数

```
int gcd(int a,int b)
{
if (a == 0) return b;
if (b == 0) return a;
if (a % 2 == 0 && b % 2 == 0) return 2 * gcd(a/2,b/2);
```

```
else if (a % 2 == 0) return gcd(a/2,b);
else if (b % 2 == 0) return gcd(a,b/2);
else return gcd(abs(a-b),min(a,b));
}
```

最长递增子序列模板——o(nlogn 算法实现)

```
#include <stdio.h>

#define MAX 40000
int array[MAX], B[MAX];

int main()
{
int count,i,n,left,mid,right,Blen=0,num;
scanf("%d",&count); //case 的个数

while(count--)
{
     scanf("%d",&n); //每组成员的数量
               Blen = 0;
     for(i=1;i<=n;i++)
       scanf("%d",&array[i]); //读入每个成员

     for(i=1;i<=n;i++)
     {
                        num = array[i];
                        left = 1;
                        right = Blen;
                        while(left<=right)
                        {
                                mid = (left+right)/2;
                                if(B[mid]<num)
                                        left = mid+1;
                                else
                                        right = mid-1;
                        }

                        B[left] = num;
                        if(Blen<left)
                                Blen++;
     }
     printf("%d\n",Blen);//输出结果
}
```

```
return 1;
}
```

判断图中同一直线的点的最大数量

```
#include <iostream>
#include <cstdio>
#include <memory>
using namespace std;

#define MAX 1010 //最大点的个数
struct point
{
int x,y;
}num[MAX];
int used[MAX][MAX*2]; //条件中点的左边不会大于1000,just equal MAX
int countN[MAX][MAX*2];

#define abs(a) (a>0?a:(-a))
int GCD(int x, int y)
{
int temp;
if(x < y)
{
     temp = x; x = y; y = temp;
}
while(y != 0)
{
     temp = y;
     y = x % y;
     x = temp;
}
return x;
}


int main()
{
int n,i,j;
int a,b,d,ans;
while(scanf("%d", &n)==1)
{
     //inite
     ans = 1;
```

```
    memset(used, 0, sizeof(used));
    memset(countN, 0, sizeof(countN));

    //read
    for(i = 0; i < n; i++)
      scanf("%d%d", &num[i].x, &num[i].y);

    for(i = 0; i < n-1; i++)
    {
      for(j = i+1; j < n; j++)
      {
        b = num[j].y-num[i].y;
        a = num[j].x-num[i].x;
        if(a < 0) //这样可以让（2，3）（－2，－3）等价
        {a = -a; b = -b;}
        d = GCD(a,abs(b));
        a /= d;
        b /= d; b += 1000;//条件中点的左边不会大于 1000
        if(used[a][b] != i+1)
        {
          used[a][b] = i+1;
          countN[a][b] = 1;
        }
        else
        {
          countN[a][b]++;
          if(ans < countN[a][b])
            ans = countN[a][b];
        }
      }//for
    }//for
    printf("%d\n", ans+1);
}
return 0;
}
```

公因数和公倍数

```
int GCD(int x, int y)
{
int temp;
if(x < y)
{
    temp = x; x = y; y = temp;
```

```
}
while(y != 0)
{
     temp = y;
     y = x % y;
     x = temp;
}
return x;
}

int beishu(int x, int y)
{
return x * y / GCD(x,y);
}
```

已知先序中序求后序

```
#include <iostream>
#include <string>
using namespace std;

string post;
void fun(string pre, string mid)
{
if(pre == "" || mid == "") return;
int i = mid.find(pre[0]);
fun(pre.substr(1,i), mid.substr(0,i));
fun(pre.substr(i+1, (int)pre.length()-i-1), mid.substr(i+1,
(int)mid.length()-i-1));
post += pre[0];
}

int main()
{
string pre, mid;
while(cin >> pre)
{
     cin >> mid;
     post.erase();
     fun(pre, mid);
     cout << post << endl;
}
return 0;
}
```

深度优先搜索模板

```
int t; //t 用来标识要搜索的元素
int count; //count 用来标识搜索元素的个数
int data[m][n]; //data 用来存储数据的数组

//注意，数组默认是按照 1……n 存储，即没有第 0 行

//下面是 4 个方向的搜索,
void search(int x, int y)
{
data[x][y] = *; //搜索过进行标记
if(x-1 >= 1 && data[x-1][y] == t)
{
     count++;
     search(x-1,y);
}
if(x+1 <= n && data[x+1][y] == t)
{
     count++;
     search(x+1,y);
}
if(y-1 >= 1 && data[x][y-1] == t)
{
     count++;
     search(x,y-1);
}
if(y+1 <= n && data[x][y+1] == t)
{
     count++;
     search(x,y+1);
}
}

//下面是 8 个方向的搜索
void search(int x, int y)
{
data[x][y] = *; //搜索过进行标记
if(x-1 >= 1)
{
     if(data[x-1][y] == t)
     {
       count++;
       search(x-1,y);
```

```
        }
        if(y-1 >= 1 && data[x-1][y-1] == t)
        {
          count++;
          search(x-1,y-1);
        }
        if(y+1 <= n && data[x-1][y+1] == t)
        {
          count++;
          search(x-1,y+1);
        }
    }
    if(x+1 <= n)
    {
        if(data[x+1][y] == t)
        {
          count++;
          search(x+1,y);
        }
        if(y-1 >= 1 && data[x+1][y-1] == t)
        {
          count++;
          search(x+1,y-1);
        }
        if(y+1 <= n && data[x+1][y+1] == t)
        {
          count++;
          search(x+1,y+1);
        }
    }
    if(y-1 >= 1 && data[x][y-1] == t)
    {
        count++;
        search(x,y-1);
    }
    if(y+1 <= n && data[x][y+1] == t)
    {
        count++;
        search(x,y+1);
    }
    }
```

匈牙利算法——二部图匹配 BFS 实现

```
//匈牙利算法实现
#define MAX 310                         //二部图一侧顶点的最大个数
int n,m;                                //二分图的两个集合分别含有 n 和 m 个元素。

bool map[MAX][MAX];                     //map 存储邻接矩阵。


int Bipartite()
{
    int i, j, x, ans;       //n 为最大匹配数

    int q[MAX], prev[MAX], qs, qe;
    //q 是 BFS 用的队列，prev 是用来记录交错链的，同时也用来记录右边的点是否被找过

    int vm1[MAX], vm2[MAX];
    //vm1,vm2 分别表示两边的点与另一边的哪个点相匹配

    ans = 0;
    memset(vm1, -1, sizeof(vm1));
    memset(vm2, -1, sizeof(vm2)); //初始化所有点为未被匹配的状态

    for( i = 0; i < n; i++ )
{
        if(vm1[i] != -1)continue; //对于左边每一个未被匹配的点进行一次 BFS 找交错链

        for( j = 0; j < m; j++ ) prev[j] = -2; //每次 BFS 时初始化右边的点

        qs = qe = 0; //初始化 BFS 的队列

        //下面这部分代码从初始的那个点开始，先把它能找的的右边的点放入队列
        for( j = 0; j < m; j++ )
    {
      if( map[i][j] )
      {
        prev[j] = -1;
        q[qe++] = j;
      }
```

```
    }

            while( qs < qe )
    { //BFS
                    x = q[qs];
                    if( vm2[x] == -1 ) break;
                    //如果找到一个未被匹配的点，则结束，找到了一条
交错链
                    qs++;
                    //下面这部分是扩展结点的代码
                    for( j = 0; j < m; j++ )
      {
        if( prev[j] == -2 && map[vm2[x]][j] )
        {
          //如果该右边点是一个已经被匹配的点，则 vm2[x]是与该点相匹配
的左边点
          //从该左边点出发，寻找其他可以找到的右边点
          prev[j] = x;
          q[qe++] = j;
        }
      }
            }
            if( qs == qe ) continue; //没有找到交错链

            //更改交错链上匹配状态
            while( prev[x] > -1 )
    {
                    vm1[vm2[prev[x]]] = x;
                    vm2[x] = vm2[prev[x]];
                    x = prev[x];
            }
            vm2[x] = i;
            vm1[i] = x;

            //匹配的边数加一
            ans++;
      }
      return ans;
}
```

带输出路径的 prime 算法

```
#include <iostream>
#include <memory>
using namespace std;

const int MAX = 110;
int data[MAX][MAX];
int lowcost[MAX];
int adjvex[MAX];

int main()
{
int n;
cin >> n;
int i,j;

for(i = 0 ;i < n; i++)
     for(j = 0; j < n; j++)
       cin >> data[i][j];

//prim
for(i = 1; i < n; i++)
{
     lowcost[i] = data[0][i];
     adjvex[i] = 0;
}
for(i = 1; i < n; i++)
{
     int min = 1<<25, choose;
     for(j = 1; j < n; j++)
     {
       if(lowcost[j] && lowcost[j] < min)
       {
         min = lowcost[j];
         choose = j;
       }
     }
     printf("<%d %d> %d\n", adjvex[choose]+1, choose+1,
lowcost[choose]);
     lowcost[j] = 0;

     for(j = 1; j < n; j++)
     {
```

```
        if(lowcost[j] && lowcost[j] > data[choose][j])
        {
          lowcost[j] = data[choose][j];
          adjvex[j] = choose;
        }
      }
}
return 0;
}
```

prime 模板

```
#include <iostream>
#include <memory>
#include <cmath>
using namespace std;

int const MAX = 110;
int dis[MAX][MAX];
int lowcost[MAX];

int main()
{
int n;
int i,j;

while(cin >> n)
{
for(i = 0; i < n; i++)
      for(j = 0; j < n; j++)
        cin >> dis[i][j];

//下面是 prim 算法部分，ans 是计算所有路径的和
lowcost[0] = 0;
for(i = 1; i < n; i++)
      lowcost[i] = dis[0][i];

int ans = 0;
for(i = 1; i < n; i++)
{
      double min = (1<<30);
      int choose;
      for(j = 1; j < n; j++)
      {
```

```
        if(lowcost[j] != 0 && lowcost[j] < min)
        {
          min = lowcost[j];
          choose = j;
        }
      }

      ans += lowcost[choose];
      lowcost[choose] = 0;
      for(j = 1; j < n; j++)
      {
        if(lowcost[j] != 0 && lowcost[j] > dis[choose][j])
          lowcost[j] = dis[choose][j];
      }
}
cout << ans << endl;
}
return 0;
}
```

kruskal 模板

```
#include <iostream>
#include <memory>
#include <algorithm>
using namespace std;

const int MAX = 1010; //节点个数
const int MAXEDGE = 15010; //边个数
bool used[MAXEDGE]; //标记边是否用过

struct node
{
int begin, end, dis;
}data[MAXEDGE];

class UFSet
{
private:
int parent[MAX+1];
int size;
public:
UFSet(int s = MAX);
int Find(int x);
```

```
void Union(int root1, int root2);
};

UFSet::UFSet(int s)
{
size = s+1;
memset(parent, -1, sizeof(int)*size);
}
void UFSet::Union(int root1, int root2)
{
int temp = parent[root1] + parent[root2];
if(parent[root1] <= parent[root2])
{
     parent[root2] = root1;
     parent[root1] = temp;
}
else
{
     parent[root1] = root2;
     parent[root2] = temp;
}
}
int UFSet::Find(int x)
{
int p = x;
while(parent[p] > 0)
     p = parent[p];

int t = x;
while(t != p)
{
     t = parent[x];
     parent[x] = p;
     x = t;
}
return p;
}

bool cmp(node a, node b)
{
return (a.dis < b.dis);
}
int main()
{
int n, m;
```

```
scanf("%d%d", &n, &m);
int i,j;

for(i = 0; i < m; i++)
     scanf("%d%d%d", &data[i].begin, &data[i].end, &data[i].dis);

//最小生成树
UFSet ufs(n);
sort(data, data+m, cmp);

int root1, root2;
int total = 0;
for(i = 0; i < m; i++)
{
     root1 = ufs.Find(data[i].begin);
     root2 = ufs.Find(data[i].end);
     if(root1 == root2) continue;
     ufs.Union(root1, root2);
     used[i] = true;
     total++;
     if(total == n-1) break;
}
printf("%d\n%d\n", data[i].dis, n-1);
for(j = 0; j <= i; j++)
     if(used[j])
       printf("%d %d\n", data[j].begin, data[j].end);

return 0;
}
```

dijsktra

```
#include <iostream>
#include <memory>
using namespace std;

const int maxint = 9999999;
const int maxn = 1010;

int data[maxn][maxn], lowcost[maxn]; //data 存放点点之间的距离，lowcost 存放点到 start 的距离,从 0 开始存放
bool used[maxn];//标记点是否被选中
int n; //顶点的个数
```

```
void disktra(int start)//初始点是 start 的 dij 算法
{
int i,j;

memset(used, 0, sizeof(used));

//inite
for(i = 0; i < n; i++)
     lowcost[i] = data[start][i];

used[start] = true;
lowcost[start] = 0;

for(i = 0; i < n-1; i++)
{
     //choose min
     int tempmin = maxint;
     int choose;
     for(j = 0; j < n; j++)
     {
       if(!used[j] && tempmin > lowcost[j])
       {
         choose = j;
         tempmin = lowcost[j];
       }
     }
     used[choose] = true;

     //updata others
     for(j = 0; j < n; j++)
     {
       if(!used[j] && data[choose][j] < maxint &&
lowcost[choose]+data[choose][j] < lowcost[j])
         lowcost[j] = lowcost[choose]+data[choose][j];
     }
}
}
```

并查集模板

```
#include <iostream>
#include <memory>
using namespace std;
```

```
const int MAX = 5005;

class UFSet
{
private:
int parent[MAX+1];
int size;
public:
UFSet(int s = MAX);//初始化
void Union(int root1, int root2);//合并，注意参数为根节点
int Find(int i );//返回根节点
int SetNum();//返回集合的个数
};
UFSet::UFSet(int s)
{
size = s+1;
memset(parent, -1, sizeof(int)*size );
}

void UFSet::Union( int root1, int root2)
{
int temp = parent[root1]+parent[root2];
if(parent[root1]<parent[root2])
{
     parent[root2]=root1;
     parent[root1]=temp;
}
else
{
     parent[root1]=root2;
     parent[root2]=temp;
}
}

int UFSet::Find(int i)
{
int j;
for(j = i; parent[j]>=0; j = parent[j]);

while(i!=j)
{
     int temp = parent[i];
     parent[i] = j; i = temp;
}
return j;
```

```
}
int UFSet::SetNum()
{
int totalNum = 0,i;
for(i = 0; i < size; i++)
     if(parent[i] < 0)
       totalNum++;
return totalNum;
}
```

高精度模板

```
#include <string>
#include <algorithm>
using namespace std;

// ----------------- 非负数计算部分：f1~f14
-------------------------------------
string operator+(string x, string y); // x、y 都必须非负
string operator-(string x, string y); // x、y 都必须非负（结果可能为负）
string operator*(string x, string y); // x、y 非负
string operator*(string s, int a); // s,a 非负，且 a 必须小于 2*10^8.

string MSDiv(string x, int y, int &res); // 多精度除以 int，x 非负，y
为正
string operator/(string s, int a); // 调用 MSDiv
int       operator%(string s, int a); // 调用 MSDiv
string MMDiv(string x, string y, string &res); // 多精度除以多精度，x
非负，y 为正
string operator/(string x, string y); // 调用 MMDiv
string operator%(string x, string y); // 调用 MMDiv

string HPower(string s, int a); // s,a 必须非负！
string HSqrt(string s); // 开平方取整，s 非负！

string Head_zero_remover(string num); // 除了开头可能有'0'外，num 必须
是非负数。
bool Less(string x, string y); // 非负数之间的“小于”

// ----------------- 以下是负数支持：f15~f19
-------------------------------------
string operator-(string s); // 取负
string SAdd(string x, string y);
string SMinus(string x, string y);
```

```
string SMul(string x, string y);
string SMul(string s, int a); // 同样，a 的绝对值不能超过 2*10^8

// ----------------- f1 () -------------------------
string operator+(string x, string y)
{
if(x.size() < y.size()) // 预处理，保证 x 的实际长度>=y
x.swap(y);
y.insert(y.begin(), x.size()-y.size(), '0'); // y 开头补 0 到和 x 一样长

string sum(x.size(), -1); // 初始大小：x.size()

int carry=0;
for(int i=x.size()-1; i >= 0; --i)
{
carry += x[i]+y[i]-2*'0';
sum[i] = carry%10+'0';
carry /= 10;
}

if(carry > 0) // 还有进位 1
return string("1") += sum; // 给开头添加一个“1”
return sum;
}


// ----------------- f2 (need: f13, f14) --------------------
string operator-(string x, string y)
{
bool neg = false; // 结果为负标志
if(Less(x, y))
{
x.swap(y); // 如果 x<y，交换
neg = true; // 结果标记为负
}
string diff(x.size(), -1); // 差(结果)

y.insert(y.begin(), x.size()-y.size(), '0');

int carry=0;
for(int i=x.size()-1; i >= 0; --i)
if(x[i] >= y[i]+carry) // 本位够减
{
     diff[i] = x[i]-y[i]-carry+'0';
     carry = 0;
```

```
}
else // 需要借位
{
     diff[i] = 10+x[i]-y[i]-carry+'0';
     carry=1;
}

if(neg)
return string("-") += Head_zero_remover(diff);

return Head_zero_remover(diff);
}


// ----------------- f3 (need f1, f4) --------------------------
string operator*(string x, string y)
{
string prod="0"; //初值：0
for(int i=y.size()-1; i >= 0; --i)
{
               string p_sum = x * (y[i]-'0'); // p_sum：部分积
if(p_sum != "0") // 保证后面加 0 后也符合 UAdd 的要求!
     p_sum.insert(p_sum.end(), y.size()-1-i, '0');
prod = prod + p_sum;
}
return prod;
}


// ----------------- f4 () --------------------------
string operator*(string s, int a)
{
if(s == "0" || a == 0) // 以免后面特殊处理!
return "0";

string prod(s.size(), -1); // 先申请 s.size()位

int carry=0;
for(int i=s.size()-1; i >= 0; --i)
{
carry += (s[i]-'0')*a;
prod[i] = carry%10+'0';
carry /= 10;
}
while(carry>0)
```

```
{
prod.insert(prod.begin(), carry%10+'0');
carry /= 10;
}

return prod;
}


// ----------------- f5 (need f13) -------------------------
string MSDiv(string x, int y, int &res)
{
string quot(x.size(), 0);

res=0;
for(int i=0; i<x.size(); ++i)
{
res = 10*res+x[i]-'0';
quot[i] = res/y+'0'; // 整除结果为商
res %= y; // 取余保留
}
return Head_zero_remover(quot);
}


// ----------------- f6 (need f5, f13) -------------------------
string operator/(string s, int a)
{
int res;
return MSDiv(s, a, res);
}


// ----------------- f7 (need f5, f13) -------------------------
int operator%(string s, int a)
{
int res;
MSDiv(s, a, res);
return Head_zero_remover(res);
}


// ----------------- f8 (need f2, f13, f14) -------------------------
string MMDiv(string x, string y, string &res)
{
```

```
string quot(x.size(), '0'); // 初始化成全'0'

res = ""; // 初始为空，每次下移一个字符
for(int i=0; i<x.size(); ++i)
{
res += x[i]; // 等价 res = res*10+x[i];（注意：不是加）
while( ! Less(res, y)) // 余数大于等于除数时...
{
     res = res - y; // 余数减去除数
     ++quot[i]; // 商对应位加 1
}
}
return Head_zero_remover(quot);
}


// ----------------- f9 (need f2, f8, f13, f14)
--------------------------
string operator/(string x, string y)
{
return MMDiv(x, y, string());
}


// ----------------- f10 (need f2, f8, f13, f14)
--------------------------
string operator%(string x, string y)
{
string res;
MMDiv(x, y, res);
return res;
}


// ----------------- f11 (need f1, f3, f4) -------------------------
string HPower(string s, int a) // 最多做 2*ln(a)次大数乘法
{
string power="1";
while(a>0)
{
if(a%2 == 1)
     power = power * s;
a /= 2;
s = s * s;
}
```

```
return power;
}

// ----------------- f12 (need f2, f4, f13, f14)
--------------------------
string HSqrt(string s) // 手工开平方。若要返回余数，return 前的 res 就
是!
{
string sqroot((s.size()+1)/2, -1);

string res = s.substr(0, 2-s.size()%2); // 奇位取前 1，偶位取前 2
string div="0"; // 占一位置
for(int i=0; i<sqroot.size(); ++i)
{
for(int quot=9; ; --quot)
{
     div[div.size()-1] = quot+'0'; // 末位试商，从'9'到'0'
     string p_prod = div*quot;
     if( ! Less(res, p_prod) ) // p_prod <= res
     {
       sqroot[i] = quot+'0'; // 将结果追加!
       div = sqroot.substr(0, i+1)*20;
       res = res - p_prod;

       string next2 = s.substr((i+1)*2-s.size()%2, 2);
       if(res == "0")
         res = next2; // 取后 2 位
       else
         res += next2; // 下移 2 位，追加；即 res = res*100+next2
       break;
     }
}
}
return sqroot;
}


// ----------------- f13 () --------------------------
bool Less(string x, string y)
{
return x.size()<y.size() || x.size() == y.size() && x < y;
}


// ----------------- f14 () --------------------------
```

```
string Head_zero_remover(string num) // 化简“003”等数
{
if(num[0] != '0')
return num;
int pos=num.find_first_not_of('0');
if(pos == string::npos) // 全0
return "0";
return num.substr(pos, num.size()-pos);
}


//////////////////////////////////////////////////////////////////////
//////////
//              以下是负数支持!
//////////////////////////////////////////////////////////////////////
//////////

// ----------------- f15 () -------------------------
string operator-(string s)
{
if(s[0] == '-')
return s.substr(1, s.size()-1);
if(s == "0")
return "0";
return string("-") += s;
}


// ----------------- f16 (need f1, f2, f13, f14, f15) ----------
string SAdd(string x, string y)
{
if(x[0] == '-' && y[0] == '-')
return -(-x + -y);
if(x[0] == '-')
return y - -x;
if(y[0] == '-')
return x - -y;
return x + y;
}


// ----------------- f17 (need f1, f2, f13, f14, f15) ---------
string SMinus(string x, string y)
{
if(x[0] == '-' && y[0] == '-')
```

```
return -y - -x;
if(x[0] == '-')
return -(-x + y);
if(y[0] == '-')
return x + -y;
return x - y;
}


// ----------------- f18 (need f1, f3, f4, f15)
-------------------------
string SMul(string x, string y)
{
if(x[0] == '-' && y[0] == '-')
return (-x)*(-y);
if(x[0] == '-')
return -((-x)*y);
if(y[0] == '-')
return -(x*(-y));
       return x * y;
}


// ----------------- f19 (need f4, f15)-------------------------
string SMul(string s, int a)
{
if(s[0] == '-' && a<0)
return (-s)*(-a);
if(s[0] == '-')
return -((-s)*a);
if(a<0)
return -(s*(-a));
return s * a;
}
```