ACM Template

UESTC_Jungle

Last build at October 23, 2018

Contents

1 Datastructure

1.1 Fenwick

```
/* Fenwick Tree (Binary Indexed Tree), by Abreto <m@abreto.net>. */
#include <cstring>

using namespace std;

template <class T = int, int MAXN = 100001>
struct fenwick {
  static inline int lowbit(int x) {
    return (x&(-x));
  }
  int N;
  T f[MAXN]; /* 1=based. */
  fenwick(void):N(MAXN) {
    init();
  }
  fenwick(int n):N(n) {
    init();
  }
  void init(void) {
    memset(f,0,sizeof(f));
  }
  void upd(int i, T dx) {
    while(i <= N) {
      f[i] += dx;
      i += lowbit(i);
    }
  }
  T sum(int i) {
    T ret = 0;
    while(i) {
      ret += f[i];
      i -= lowbit(i);
    }
    return ret;
  }
};
```

```
/* Fenwick Tree (Binary Indexed Tree), by Abreto <m@abreto.net>. */

#define MAXN 100001
#define LOWBIT(x)   ((x)&(-(x)))

int N;
int fen[MAXN];

void update(int i, int dx) {
  while(i <= N) {
    fen[i] += dx;
    i += LOWBIT(i);
  }
}

int sum(int i) {
  int s = 0;
  while(i > 0) {
    s += fen[i];
    i -= LOWBIT(i);
  }
  return s;
}
```

1.2 BST in pb_ds

```
/* Red-Black tree via pb_ds. */
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

int main() {
  ordered_set<int>  s;
  s.insert(1);
  s.insert(3);
  cout << s.order_of_key(2) << endl; // the number of elements in the s less than 2
  cout << *s.find_by_order(0) << endl; // print the 0-th smallest number in s(0-based
      )
}
```

1.3 Segment Tree

```
/* Segment tree (Interval tree, range tree), by Abreto <m@abreto.net>. */

template <int STMAX = 1000000>
struct segment_tree {
  struct node_t {
    static inline node_t merge(node_t n1, node_t n2) {
      node_t ans;
      ans.l = n1.l;
      ans.r = n2.r;
      /* merge n1 and n2 to ans. */
      return ans;
    }

    /* Data field */
    int l,r;
  } nodes[(STMAX+1)<<2];

  struct lazy_t {
    int marked; /* Optional */
    /* lazy mark. */

    lazy_t(void) {
      clear();
    }
    void clear(void) {
      marked=0;
    }
  } marks[(STMAX+1)<<2];

  inline void maintain_leaf(int o, int idx) {
    nodes[o].l = nodes[o].r = idx;
    /* Operations to single elements ... */
  }
  inline void maintain(int o) {
    nodes[o] = node_t::merge(nodes[o<<1], nodes[o<<1|1]);
  }

  /* Usage: build(1,1,n); */
  void build(int o, int l, int r) { /* [l,r] */
```

```
    if( r <= l ) {
      maintain_leaf(o, l);
    } else {
      int mid = l+r>>1;
      build(o<<1, l, mid);
      build(o<<1|1, mid+1, r);
      maintain(o);
    }
  }

  /* Modify all elements in [l,r] */
  void mark(lazy_t act, int o) {
    /* do something .. */
    marks[o].marked = 1;
  }

  /* Pass cached updates. */
  void pushdown(int o) {
    if( marks[o].marked ) {
      mark(marks[o], o<<1);
      mark(marks[o], o<<1|1);
      marks[o].clear();
    }
  }

  /* Do act on all elements in [L,R] */
  void upd(int L, int R, lazy_t act, int o, int l, int r) {
    if( L <= l && r <= R ) {
      mark(act, o);
    } else if (L <= R) {
      int mid = (l+r)>>1;
      pushdown(o);
      if( L <= mid ) upd(L, R, act, o<<1, l, mid);
      if( R > mid ) upd(L, R, act, o<<1|1, mid+1, r);
      maintain(o);
    }
  }

  node_t qry(int L, int R, int o, int l, int r) {
    if(L <= l && r <= R)
      return nodes[o];
    else if (L <= R) {
      int mid = (l+r)>>1;
      pushdown(o);
      if(R <= mid) return qry(L,R,o<<1,l,mid);
      if(L > mid) return qry(L,R,o<<1|1,mid+1,r);
      return node_t::merge(qry(L,R,o<<1,l,mid),qry(L,R,o<<1|1,mid+1,r));
    }
  }

  int N;

  segment_tree(void):N(STMAX) {}
  segment_tree(int n):N(n) {}
  void build(int n) {
    N = n;
    build(1,1,N);
  }
  void update(int L, int R, lazy_t act) {
    upd(L,R,act,1,1,N);
  }
  node_t query(int L, int R) {
    return qry(L,R,1,1,N);
  }
```

```
};
```

```
/* Segment tree (Interval tree, range tree), by Abreto <m@abreto.net>. */

#define MAXN    1000001

typedef struct {
  int l,r;
  /* Data field */
} node_t;

node_t merge(node_t n1, node_t n2) {
  node_t ans;
  ans.l = n1.l;
  ans.r = n2.r;
  /* merge n1 and n2 to ans. */
  return ans;
}

typedef struct {
  int marked; /* Optional */
  /* lazy mark. */
} lazy_t;

int A[MAXN];
node_t nodes[MAXN<<2];
lazy_t marks[MAXN<<2];

void maintain_leaf(int o, int idx) {
  nodes[o].l = nodes[o].r = idx;
  /* Operations to single elements ... */
}
void maintain(int o) {
  nodes[o] = merge(nodes[o<<1], nodes[o<<1|1]);
}

/* Usage: build(1,1,n); */
void build(int o, int l, int r) { /* [l,r] */
  if( r <= l ) {
    maintain_leaf(o, l);
  } else {
    int mid = l+r>>1;
    build(o<<1, l, mid);
    build(o<<1|1, mid+1, r);
    maintain(o);
  }
  marks[o].marked = 0;
}

/* Modify all elements in [l,r] */
void mark(lazy_t act, int o) {
  /* do something .. */
  marks[o].marked = 1;
}

/* Pass cached updates. */
void pushdown(int o) {
  if( marks[o].marked ) {
    mark(marks[o], o<<1);
    mark(marks[o], o<<1|1);
    marks[o].marked = 0;
  }
}

```

```
/* **DISCARDED** */
/* Set A[p]=v. Usage: modify(p, v, 1, 1, n);
void modify(int p, int v, int o, int l, int r)
{
    if( r - l < 2 )
    {
        maintain_leaf(o, v);
    } else {
        int mid = (l+r)/2;
        pushdown(o);
        if( p <= mid ) modify(p, v, o*2, l, mid);
        else modify(p, v, o*2+1, mid, r);
        maintain(o);
    }
}*/

/* Do act on all elements in [L,R] */
void update(int L, int R, lazy_t act, int o, int l, int r) {
  if( L <= l && r <= R ) {
    mark(act, o);
  } else if (L <= R) {
    int mid = (l+r)>>1;
    pushdown(o);
    if( L <= mid ) update(L, R, act, o<<1, l, mid);
    if( R > mid ) update(L, R, act, o<<1|1, mid+1, r);
    maintain(o);
  }
}
```

1.4 Sparse Table

```
/* RMQ with Sparse Table, by Abreto <m@abreto.net>. */

int min(int a, int b) {
  return (a<b)?a:b;
}

#define MAXN    100001
#define MAXLOG  32

int N;
int A[MAXN];    /* indexed from 0. */
int st[MAXN][MAXLOG];

void st_init() {
  int i = 0, j = 0, t = 0;
  for(i = 0; i < N; ++i) st[i][0] = A[i];
  for(j = 1; (t=(1<<j)) <= N; ++j)
    for(i = 0; (i+t-1) < N; ++i)
      st[i][j] = min(st[i][j-1], st[i+(t>>1)][j-1]);
  /* st(i,j) = min(st(i,j-1), st(i+2^(j-1),j-1)). */
}

int st_query(int l, int r) {
  int k = 0;
  while((1<<(k+1)) <= (r-l+1)) k++;
  return min(st[l][k], st[r-(1<<k)+1][k]);
}
```

1.5 Treap

```
#include <bits/stdc++.h>

using namespace std;

#define MAXN  (2560000)

int __treap_mem[MAXN];
void init_treap_mem(void) {
  for( int i = 1 ; i < MAXN ; i++ )
    __treap_mem[i-1] = i;
}
int alloc_address(void) {
  int ret = __treap_mem[0];
  __treap_mem[0] = __treap_mem[ret];
  return ret;
}
void free_address(int p) {
  __treap_mem[p] = __treap_mem[0];
  __treap_mem[0] = p;
}

typedef int key_t;
typedef int val_t;
struct treap {
  key_t x;
  val_t v;
  int r;    /* random priority */
  int eq, s;  /* number of equal ones, size of subtree (include root itself) */
  treap *fa;  /* point to its father */
  treap *ch[2]; /* 0 for left child, 1 for right child. */

  treap(void);
  inline void maintain(void); /* update s */
  inline void set_child(int d, treap *child);
  inline int which(void); /* determine which child this is of its father */
  inline int cmp(key_t ox); /* determine which child to insert ox */
  treap *rotate(void);  /* rotate this to its father, return this */
} treap_nodes[MAXN];

treap *new_treap(key_t x, val_t v, treap *f) {
  treap *ret = treap_nodes + alloc_address();
  ret->x = x;
  ret->v = v;
  ret->eq = ret->s = 1;
  ret->fa = f;
  ret->ch[0] = ret->ch[1] = NULL;
}
void free_treap(treap *p) {
  free_address( p - treap_nodes );
}
void treap_clear(void) {
  init_treap_mem();
}

treap::treap(void) {
  r = rand();
  eq = s = 0;
  fa = ch[0] = ch[1] = NULL;
}
inline void treap::maintain(void) {
  s = eq;
  for( int i = 0 ; i < 2 ; i++ )
    if( ch[i] )
      s += ch[i]->s;
```

```
}
inline void treap::set_child(int d, treap *child) {
  ch[d] = child;
  maintain();
  if( child ) child->fa = this;
}
inline int treap::which(void) {
  if( NULL == fa ) return -1; /* this is not a child */
  else return ( fa->ch[1] == this );
}
inline int treap::cmp(key_t ox) {
  if( ox == x ) return -1;  /* equal */
  else return ( ox > x );   /* left less, right more */
}
treap *treap::rotate(void) {
  if ( NULL == fa ) return this;  /* no father, already global root. */
  int d = which();
  fa->set_child(d, ch[d^1]);
  set_child(d^1, fa);
  return this;
}

// --- deprecated |
typedef int T;

struct node {
  T v;  /* value of this node */
  int r;  /* random priority */
  int eq; /* the number of equal things */
  int s;  /* the size of subtree rooted at this */
  node *ch[2];  /* 0 for left child, 1 for right child. */
  node(void) {
    r = rand();
    ch[0] = ch[1] = NULL;
  }
  /* return where to insert x */
  int cmp(T x) {
    if(v == x) return -1;
    else return (x < v) ? 0 : 1;
  }
  /* return 1 if this node is prior to other */
  int pri(node *o) {
    return (r > (o->r));
  }
  /* maintain the s field */
  void maintain(void) {
    s = eq;
    if(NULL != ch[0]) s += ch[0]->s;
    if(NULL != ch[1]) s += ch[1]->s;
  }
};

/* move o to ch[d] of o->ch[d^1] */
void rotate(node *&o, int d) {
  node *k = o->ch[d^1];
  o->ch[d^1] = k->ch[d];
  o->maintain();
  k->ch[d] = o;
  k->maintain();
  o = k;
}
```

1.6 Splay

```
/* splay, by Abreto<m@abreto.net>. */

#ifndef NULL
#define NULL 0
#endif

struct node {
  node *f, *ch[2];
  int sz;
  node(node *fa = NULL, node *lc = NULL, node *rc = NULL) {
    f = fa;
    ch[0] = lc;
    ch[1] = rc;
    maintain();
  }
  inline int szof(const int d) const {
    return ch[d] ? ch[d]->sz : 0;
  }
  inline void maintain(void) {
    sz = szof(0) + szof(1) + 1;
  }
  inline int which(void) {
    if (NULL == f) return 0;
    return (f->ch[1] == this);  /* f[which()] == this */
  }
  inline node *setf(node *fa, int d = 0) {
    f = fa;
    if (f) {
      f->ch[d] = this;
      f->maintain();
    }
    return f;
  }
  inline node *setc(node *son, int d = 0) {
    ch[d] = son;
    if (son) son->f = this;
    maintain();
    return this;
  }
  /* rotate this to its fater, return this. */
  inline node *rotate(void) {
    if (f != NULL) {
      node *ff = f->f;
      int d = which(), fd = f->which();
      setc(f->setc(ch[d ^ 1], d), d ^ 1);
      setf(ff, fd);
    }
    return this;
  }
  /* splay this to child of target */
  inline node *splay(node * const target = NULL) {
    while (f != target) {
      if (target != f->f) {
        ( (which() == f->which()) ? f : this )->rotate();
      }
      rotate();
    }
    return this;
  }
  /* 0-based rank */
  inline node *get_k_th(unsigned k) {
    node *p = this;
```

```
    int rank;
    while (k != (rank = (p->szof(0)))) {
      if (k < rank) {
        p = p->ch[0];
      } else {
        k -= (rank + 1);
        p = p->ch[1];
      }
    }
    return p->splay(f);
  }
};
```

```
/* HDU 3487 - Play with Chain, by Abreto<m@abreto.net>. */
#include <bits/stdc++.h>

using namespace std;

#define MAXN    300300

int n, m;

#define LC(p)   ch[p][0]
#define RC(p)   ch[p][1]
#define TARGET(p) LC(RC(p))

int nodes;
int val[MAXN], ch[MAXN][2], fa[MAXN], sz[MAXN];
int rev[MAXN];

inline int new_node(int v, int f) {
  int p = (++nodes);
  val[p] = v;
  fa[p] = f;
  ch[p][0] = ch[p][1] = rev[p] = 0;
  sz[p] = 1;
  return p;
}
inline void maintain(int p) {
  if (p) {
    sz[p] = sz[LC(p)] + sz[RC(p)] + 1;
  }
}
inline void make_child(int f, int d, int p) { /* make p the d-th ch of f */
  ch[f][d] = p;
  if(p) fa[p] = f;
}
inline void myrev(int p) {
  if (p) {
    rev[p] ^= 1;
    swap(LC(p), RC(p));
  }
}
inline void pushdown(int p) {
  if(p && rev[p]) {
    if(LC(p)) myrev(LC(p));
    if(RC(p)) myrev(RC(p));
    rev[p] = 0;
  }
}
int build(int f = 0, int l = 0, int r = n+1) {
  if(r < l) return 0;
  if(l == r) return new_node(l, f);
  int mid = l+r>>1;
```

```
52    int p = new_node(mid, f);
53    LC(p) = build(p, l, mid-1);
54    RC(p) = build(p, mid+1, r);
55    maintain(p);
56    return p;
57  }
58  inline int which(int p) {  /* return 1 if p is a right child or 0 if p is a left
        child. */
59    return (RC(fa[p]) == p);
60  }
61  inline int rotate(int p) { /* rotate p to its father. [!] make sure p is not global
        root. */
62    int f = fa[p], ff = fa[f];
63    if(0 == f) return p;  /* p is global root */
64    pushdown(f);
65    pushdown(p);
66    int d = which(p), df = which(f);
67    make_child(f, d, ch[p][d^1]);
68    make_child(p, d^1, f);
69    maintain(f);
70    maintain(p);
71    fa[p] = ff;
72    if(ff) ch[ff][df] = p;
73    return p;
74  }
75  inline int splay(int p, int fr) { /* splay p to the son of fr, return p. */
76    pushdown(p);
77    while(fa[p] != fr) {
78      int f = fa[p], dp = which(p);
79      if(fa[f] == fr) {
80        return rotate(p);
81      } else {
82        int df = which(f);
83        if(dp == df) {
84          rotate(f);
85        } else {
86          rotate(p);
87        }
88        rotate(p);
89      }
90    }
91    return p;
92  }
93  inline int get_k_th(int root, int k) {
94    int p = root;
95    int rank;
96    while(k != (rank = (sz[LC(p)] + 1))) {
97      pushdown(p);
98      if(k < rank) p = LC(p);
99      else {
100       k -= rank;
101       p = RC(p);
102     }
103   }
104   return splay(p, fa[root]);
105 }
106 inline int merge(int left, int right) {
107   pushdown(left);
108   if(RC(left)) left = get_k_th(left, sz[left]);
109   RC(left) = right;
110   maintain(left);
111   fa[right] = left;
112   return left;
113 }
```

```
114 inline int split(int root, int d) { /* split ch[root][d], return the root of splited
        out. */
115   pushdown(root);
116   int child = ch[root][d];
117   ch[root][d] = 0;
118   maintain(root);
119   fa[child] = 0;
120   return child;
121 }
122 inline int concat(int root, int d, int p) { /* make p be ch[root][d], return root */
123   pushdown(root);
124   ch[root][d] = p;
125   fa[p] = root;
126   maintain(root);
127   return root;
128 }
129
130 void myclear(void) {
131   nodes = 0;
132 }
133
134 int ans[MAXN];
135 void inorder(int p, int &pos) {
136   if(0 == p) return;
137   pushdown(p);
138   inorder(LC(p), pos);
139   if( (0 < val[p]) && (val[p] < n+1) ) ans[pos++] = val[p];
140   inorder(RC(p), pos);
141 }
142
143 void handle() {
144   int i;
145   int root;
146   myclear();
147   root = build(0);
148   while(m--) {
149     char command[8];
150     int a, b, c;
151     int tar;
152     scanf("%s%d%d", command, &a, &b);
153     if('C' == command[0]) {
154       scanf("%d", &c);
155       root = get_k_th(root, a);
156       RC(root) = get_k_th(RC(root), b-a+2);
157       tar = split(RC(root), 0);
158       maintain(root);
159       root = get_k_th(root, c+1);
160       RC(root) = get_k_th(RC(root), 1);
161       RC(root) = concat(RC(root), 0, tar);
162       maintain(root);
163     } else {
164       root = get_k_th(root, a);
165       RC(root) = get_k_th(RC(root), b-a+2);
166       myrev(TARGET(root));
167     }
168   }
169   int pos = 0;
170   inorder(root, pos);
171   for(i = 0; i < n; i++) printf("%s%d", i ? " ":"", ans[i]);
172   puts("");
173 }
174
175 int main(void) {
176   while( scanf("%d%d", &n, &m) && (n > 0) && (m > 0) )
```

```
177     handle();
178   return 0;
179 }
```

2 Dynamic Programming

2.1 LIS $O(n \log n)$

```
 1
 2 int top = 0;
 3 for( int i=1; i<=n; i++ ) {
 4   if( ap[i] > dp[top] ) { // 如果大于 ”模拟栈” 的栈顶元素直接 入栈 长度加 1
 5     top++;
 6     dp[top] = ap[i];
 7     continue;
 8   }
 9   int m = ap[i];
10   // lower_bound 前闭后开 返回不小于 m 的最小值的位置
11   pos = lower_bound(dp,dp+top,m)-dp; // 注意减去dp
12   if( dp[pos] > ap[i])
13     dp[pos] = ap[i];
14 }
```

2.2 LCS $O(n \log n)$

总的来说，就是把 LCS 转化成 LIS，然后用 LIS 的 $\mathcal{O}(N \log N)$ 算法来求解。
实现如下：（引用）
假设有两个序列 $s_1[1 \ldots 6] = abcadc$, $s_2[1 \ldots 7] = cabedab$.
记录 s_1 中每个元素在 s_2 中出现的位置, 再将位置按降序排列, 则上面的例子可表示为：
$loc(a) = \{6, 2\}$, $loc(b) = \{7, 3\}$, $loc(c) = \{1\}$, $loc(d) = \{5\}$. (倒着扫一遍 s_2 即可把位置扔进 vector).
将 s_1 中每个元素的位置按 s_1 中元素的顺序排列成一个序列 $s_3 = \{6, 2, 7, 3, 1, 6, 2, 5, 1\}$.
在对 s_3 求 LIS 得到的值即为求 LCS 的答案。

2.3 Improved by quadrilateral inequality

```
 1 /*
 2  * 四边形不等式
 3  *
 4  * 如果 dp(i,j) 满足 dp(i,j)<=dp(i,j+1)<=dp(i+1,j+1)
 5  * 那么决策 s(i,j) 满足 s(i,j)<=s(i,j+1)<=s(i+1,j+1)
 6  * 可以变形为:
 7  *      s(i-1,j) <= s(i,j) <= s(i,j+1)  // dp方向：i增j减
 8  *  或
 9  *      s(i,j-1) <= s(i,j) <= s(i+1,j)  // dp方向：区间长度L增
10  */
11 #include <bits/stdc++.h>
12
13 using namespace std;
14
15 #define MAXN    1024
16 #define inf     (0x3fffffff)
17
18 int n, m;
19 int v[MAXN];
20 int s[MAXN];
21 int w[MAXN][MAXN];
22 int dp[MAXN][MAXN];
23 int c[MAXN][MAXN];
24
```

```
int wa(void) {
  int i, j, k;
  for(i = 1; i <= n; ++i) {
    scanf("%d", v+i);
    s[i] = v[i] + s[i-1];
  }
  for(i = 1; i <= n; ++i) {
    w[i][i] = 0;
    for(j = i+1; j <= n; ++j)
      w[i][j] = w[i][j-1] + v[j] * (s[j-1] - s[i-1]);
  }
  /* doing dp */
  for(i = 1; i <= n; ++i) {
    dp[i][0] = w[1][i];
    c[i][0] = 1;
    c[i][i] = i-1;
    for(j = i-1; j > 0; j--) {
      dp[i][j] = inf;
      for(k = c[i-1][j]; k <= c[i][j+1]; ++k)
        if(dp[k][j-1]+w[k+1][i] <= dp[i][j]) {
          dp[i][j] = dp[k][j-1] + w[k+1][i];
          c[i][j] = k;
        }
    }
  }
  /* dp done */
  return dp[n][m];
}

int main(void) {
  while(EOF != scanf("%d%d", &n, &m) && n && m) {
    printf("%d\n", wa());
  }
  return 0;
}
```

2.4 Improved by Slope

```
/* type 1: */
/* bzoj 1010 */
#include <bits/stdc++.h>

using namespace std;
typedef long double ll;
#define MAXN    50050
#define eps     (1e-8)

int N;
ll L;
ll S[MAXN];
ll f[MAXN];
ll dp[MAXN];

inline ll k(int j) {
  return (-2.0) * (f[j] + L);
}
inline ll b(int j) {
  return dp[j] + f[j]*f[j] + 2ll*f[j]*L;
}
inline ll g(int j, int i) {
  return k(j) * f[i] + b(j);
}

```

```
/* check if l1 & l3 <= l2 */
inline int check(int l1, int l2, int l3) {
  /*ll left = b(l3)*k(l1)+b(l1)*k(l2)+b(l2)*k(l3);
  ll right = b(l1)*k(l3)+b(l3)*k(l2)+b(l2)*k(l1);*/
  ll left = b(l3)*k(l1)-b(l1)*k(l3);
  ll right = k(l2)*(b(l3)-b(l1))+b(l2)*(k(l1)-k(l3));
  return (left <= right);
}

int Q[MAXN], ql, qr;

int main(void) {
  int i;
  scanf("%d%Lf", &N, &L);
  L += 1.0;
  for(i = 1; i <= N; ++i) {
    scanf("%Lf", S+i);
    S[i] += S[i-1];
    f[i] = S[i] + (double)i;
  }
  Q[qr++] = 0;
  for(i = 1; i <= N; ++i) {
    /* <!-- STARED */
    for(; ql+1 < qr && g(Q[ql],i) >= g(Q[ql+1],i); ql++);
    dp[i] = g(Q[ql], i) + f[i]*f[i] + L*L; //printf("%d: %lld,%lld\n", i, dp[i], dp[i
        ]-f[i]*f[i]);
    for(; ql+1 < qr && check(Q[qr-2], Q[qr-1], i); qr--);
    Q[qr++] = i;
    /* --> */
  }
  printf("%lld\n", (long long int)round(dp[N]));
  return 0;
}
```

3 Geometry

3.1 2D

3.1.1 Point

```
/* 2D Point Class, by Abreto<m@abreto.net> */
#include <cmath>

/**
 * Define ABG2d_USE_LL if you want to use long long int for cordnates.
 */

namespace ab_geometry_2d {

using namespace std;

typedef double ab_float;

const ab_float pi = acos(-1.);

#ifdef ABG2d_USE_LL
typedef long long int T;
#else
typedef ab_float T;
const ab_float eps = 1e-8;
#endif

```

```
23 inline T myabs(T x) {
24   if(x < 0) return (-x);
25   return x;
26 }
27
28 inline int sgn(T x) {
29   /* no difference'' in fact */
30 #ifdef ABG2d_USE_LL
31   if (0 == x) return 0;
32 #else
33   if (myabs(x) < eps) return 0;
34 #endif
35   return (x > 0) ? 1 : -1;
36 }
37
38 inline T sqr(T x) {
39   return (x * x);
40 }
41
42 struct point {
43   T x,y;
44   point(void):x(T()),y(T()) {}
45   point(T xx, T yy):x(xx),y(yy) {}
46   inline T norm2(void) {
47     return sqr(x) + sqr(y);
48   }
49   inline ab_float norm(void) {
50     return sqrt((ab_float)(norm2()));
51   }
52   inline point rotate(const ab_float &cost, const ab_float &sint) {} // TODO:
53   inline point operator-(void) const {
54     return point(-x,-y);
55   }
56   inline point operator+(const point& b) const {
57     return point(x+b.x,y+b.y);
58   }
59   inline point operator-(const point& b) const {
60     return point(x-b.x,y-b.y);
61   }
62   inline point operator->*(const point &b) const {
63     return (b-(*this));
64   }
65   inline T operator*(const point& b) const {
66     return ((x)*(b.x))+((y)*(b.y)); /* inner product */
67   }
68   inline T operator^(const point& b) const {
69     return ((x)*(b.y))-((b.x)*(y)); /* outter product */
70   }
71   inline point& operator+=(const point& b) {
72     point tmp=(*this)+b;
73     (*this)=tmp;
74     return (*this);
75   }
76   inline point& operator-=(const point& b) {
77     point tmp=(*this)-b;
78     (*this)=tmp;
79     return (*this);
80   }
81   inline bool operator==(const point& b) const {
82     return (0==sgn(x-b.x))&&(0==sgn(y-b.y));
83   }
84   inline bool operator!=(const point& b) const {
85     return !((*this)==b);
86   }
```

```
87    inline point operator<<(const ab_float& theta) const {
88      ab_float ct = cos(theta), st = sin(theta);  /* rotate counter-clockwise in radian
          */
89      return point(ct*x - st*y, st*x + ct*y);
90    }
91  };
92
93  typedef point vec;
94
95
96  }   // namespace ab_geometry_2d
```

3.1.2 Convex hull

```
1   /* 2D Convex Hull, by Abreto <m@abreto.net>. */
2   #include "2d_base.hh"
3   #include <cmath>
4   #include <algorithm>
5
6   using namespace std;
7
8   point O;
9
10  bool comp_angle(point_t a, point_t b) {
11    double t = (a-O).X(b-O);
12    if(fe(t,0.0)) return fl((b-O).mag2(),(a-O).mag2());
13    else return fl(0.0,t);
14  }
15
16  void convex_hull_graham(vp& convex, vp src) {
17    int i = 0, top = 0;
18    O = src[0];
19    for(auto pt : src)
20      if( pt.x < O.x || (pt.x == O.x && pt.y < O.y))
21        O = pt;
22    sort(src.begin(), src.end(), comp_angle);
23    convex.push_back(src[0]);
24    convex.push_back(src[1]);
25    top = 1;
26    for(i = 2; i < src.size(); ++i) {
27      while(top>1 && fle((convex[top]-convex[top-1]).X(src[i]-convex[top]),0.0)) {
28        convex.pop_back();
29        --top;
30      }
31      convex.push_back(src[i]);
32      ++top;
33    }
34  }
```

3.1.3 Intersect Area

```
1   #include <cstdio>
2   #include <cmath>
3   #include <algorithm>
4
5   using namespace std;
6
7   //#define inf 1000000000000
8   #define M 8
9   #define LL long long
10  #define eps 1e-12
```

```
#define PI acos(-1.0)
using namespace std;
struct node {
  double x,y;
  node() {}
  node(double xx,double yy) {
    x=xx;
    y=yy;
  }
  node operator -(node s) {
    return node(x-s.x,y-s.y);
  }
  node operator +(node s) {
    return node(x+s.x,y+s.y);
  }
  double operator *(node s) {
    return x*s.x+y*s.y;
  }
  double operator ^(node s) {
    return x*s.y-y*s.x;
  }
};
double max(double a,double b) {
  return a>b?a:b;
}
double min(double a,double b) {
  return a<b?a:b;
}
double len(node a) {
  return sqrt(a*a);
}
double dis(node a,node b) { //两点之间的距离
  return len(b-a);
}
double cross(node a,node b,node c) { //叉乘
  return (b-a)^(c-a);
}
double dot(node a,node b,node c) { //点成
  return (b-a)*(c-a);
}
int judge(node a,node b,node c) { //判断c是否在ab线段上（前提是c在直线ab上）
  if(c.x>=min(a.x,b.x)
      &&c.x<=max(a.x,b.x)
      &&c.y>=min(a.y,b.y)
      &&c.y<=max(a.y,b.y))
    return 1;
  return 0;
}
double area(node b,node c,double r) {
  node a(0.0,0.0);
  if(dis(b,c)<eps)
    return 0.0;
  double h=fabs(cross(a,b,c))/dis(b,c);
  if(dis(a,b)>r-eps&&dis(a,c)>r-eps) { //两个端点都在圆的外面则分为两种情况
    double angle=acos(dot(a,b,c)/dis(a,b)/dis(a,c));
    if(h>r-eps) {
      return 0.5*r*r*angle;
    } else if(dot(b,a,c)>0&&dot(c,a,b)>0) {
      double angle1=2*acos(h/r);
      return 0.5*r*r*fabs(angle-angle1)+0.5*r*r*sin(angle1);
    } else {
      return 0.5*r*r*angle;
    }
  } else if(dis(a,b)<r+eps&&dis(a,c)<r+eps) { //两个端点都在圆内的情况
```

```
75      return 0.5*fabs(cross(a,b,c));
76    } else { //一个端点在圆上一个端点在圆内的情况
77      if(dis(a,b)>dis(a,c)) { //默认b在圆内
78        swap(b,c);
79      }
80      if(fabs(dis(a,b))<eps) { //ab距离为0直接返回0
81        return 0.0;
82      }
83      if(dot(b,a,c)<eps) {
84        double angle1=acos(h/dis(a,b));
85        double angle2=acos(h/r)-angle1;
86        double angle3=acos(h/dis(a,c))-acos(h/r);
87        return 0.5*dis(a,b)*r*sin(angle2)+0.5*r*r*angle3;
88
89      } else {
90        double angle1=acos(h/dis(a,b));
91        double angle2=acos(h/r);
92        double angle3=acos(h/dis(a,c))-angle2;
93        return 0.5*r*dis(a,b)*sin(angle1+angle2)+0.5*r*r*angle3;
94      }
95    }
96  }
97
98  node A, B, C;
99  int R;
100
101 bool compar(node &p1, node &p2) {
102   return (p1^p2)>eps;
103 }
104
105 double f(double x, double y) {
106   node O(x,y);
107   node p[8];
108   p[0] = A-O;
109   p[1] = B-O;
110   p[2] = C-O;
111   sort(p, p+3, compar);
112   p[3] = p[0];
113   O=node(0,0);
114   double sum=0;
115   /* <!-- 求面积交部分 */
116   for(int i=0; i<3; i++) { /* 按顺或逆时针顺序最后取绝对值就好 */
117     int j=i+1;
118     double s=area(p[i],p[j],(double)R);
119     if(cross(O,p[i],p[j])>0)
120       sum+=s;
121     else
122       sum-=s;
123   }
124   if(sum < -eps) sum = -sum;
125   /* --> */
126   return sum;
127 }
128
129 double trifind(double x, double y1, double y2) {
130   double l = y1, r = y2;
131   while(r-l>eps) {
132     double mid = (l+r)/2.0;
133     double mmid = (mid+r)/2.0;
134     if( f(x,mmid) > f(x,mid)+eps )
135       l = mid;
136     else
137       r = mmid;
138   }
```

```
  return f(x,l);
}

double findmin(double x1, double x2, double y1, double y2) {
  double l = x1, r = x2;
  while(r-l>eps) {
    double mid = (l+r)/2.0;
    double mmid = (mid+r)/2.0;
    if( trifind(mmid,y1,y2) > trifind(mid,y1,y2)+eps )
      l = mid;
    else
      r = mmid;
  }
  return trifind(l,y1,y2);
}

double ans(int a, int b, int c, int r) {
  A = node(0,0);
  B = node((double)c,0);
  R = r;
  double da = a, db = b, dc = c;
  double cosa = (db*db+dc*dc-da*da)/(2.0*db*dc);
  double alpha = acos(cosa);
  C = node(db*cosa, db*sin(alpha));
  return findmin(0.0, c, 0.0, db*sin(alpha));
}

int main(void) {
  int a = 0, b = 0, c = 0, r = 0;
  while(EOF != scanf("%d%d%d%d",&a,&b,&c,&r) && (a||b||c||r))
    printf("%.8lf\n", ans(a,b,c,r));
  return 0;
}
```

3.1.4 Universe

```
#include <bits/stdc++.h>
using namespace std;

struct Point {
  double x, y;
  Point(double x = 0, double y = 0) : x(x), y(y) {}
};

typedef Point Vector;

Vector operator + (Vector A, Vector B) {
  return Vector(A.x + B.x, A.y + B.y);
}
Vector operator - (Vector A, Vector B) {
  return Vector(A.x - B.x, A.y - B.y);
}
Vector operator * (Vector A, double p) {
  return Vector(A.x*p, A.x*p);
}
Vector operator / (Vector A, double p) {
  return Vector(A.x/p, A.x/p);
}

bool operator < (const Point& a, const Point b) {
  return a.x < b.x || (a.x == b.x && a.y < b.y);
}

```

```
28 const double EPS = 1e-10;
29
30 int dcmp(double x) {
31   if(fabs(x) < EPS) return 0;
32   else return x < 0 ? -1 : 1;
33 }
34
35 bool operator == (const Point& a, const Point& b) {
36   return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y);
37 }
38
39 //向量a的极角
40 double Angle(const Vector& v) {
41   return atan2(v.y, v.x);//\share\CodeBlocks\templates\wizard\console\cpp
42 }
43
44 //向量点积
45 double Dot(Vector A, Vector B) {
46   return A.x*B.x + A.y*B.y;
47 }
48
49 //向量长度\share\CodeBlocks\templates\wizard\console\cpp
50 double Length(Vector A) {
51   return sqrt(Dot(A, A));
52 }
53
54 //向量夹角
55 double Angle(Vector A, Vector B) {
56   return acos(Dot(A, B) / Length(A) / Length(B));
57 }
58
59 //向量叉积
60 double Cross(Vector A, Vector B) {
61   return A.x*B.y - A.y*B.x;
62 }
63
64 //三角形有向面积的二倍
65 double Area2(Point A, Point B, Point C) {
66   return Cross(B-A, C-A);
67 }
68
69 //向量逆时针旋转rad度(弧度)
70 Vector Rotate(Vector A, double rad) {
71   return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
72 }
73
74 //计算向量A的单位法向量。左转90°，把长度归一。调用前确保A不是零向量。
75 Vector Normal(Vector A) {
76   double L = Length(A);
77   return Vector(-A.y/L, A.x/L);
78 }
79
80 /*********************************************************************
81 使用复数类实现点及向量的简单操作
82
83 #include <complex>
84 typedef complex<double> Point;
85 typedef Point Vector;
86
87 double Dot(Vector A, Vector B) { return real(conj(A)*B)}
88 double Cross(Vector A, Vector B) { return imag(conj(A)*B);}
89 Vector Rotate(Vector A, double rad) { return A*exp(Point(0, rad)); }
90
91 *********************************************************************/
```

```
92
93  /*******************************************************************************
94  * 用直线上的一点p0和方向向量v表示一条指向。直线上的所有点P满足P = P0+t*v;
95  * 如果知道直线上的两个点则方向向量为B-A, 所以参数方程为A+(B-A)*t;
96  * 当t 无限制时, 该参数方程表示直线。
97  * 当t > 0时, 该参数方程表示射线。
98  * 当 0 < t < 1时, 该参数方程表示线段。
99  *******************************************************************************/
100
101 //直线交点,须确保两直线有唯一交点。
102 Point GetLineIntersection(Point P, Vector v, Point Q, Vector w) {
103   Vector u = P - Q;
104   double t = Cross(w, u)/Cross(v, w);
105   return P+v*t;
106 }
107
108 //点到直线距离
109 double DistanceToLine(Point P, Point A, Point B) {
110   Vector v1 = B - A, v2 = P - A;
111   return fabs(Cross(v1, v2) / Length(v1)); //不取绝对值, 得到的是有向距离
112 }
113
114 //点到线段的距离
115 double DistanceToSegmentS(Point P, Point A, Point B) {
116   if(A == B) return Length(P-A);
117   Vector v1 = B-A, v2 = P-A, v3 = P-B;
118   if(dcmp(Dot(v1, v2)) < 0) return Length(v2);
119   else if(dcmp(Dot(v1, v3)) > 0) return Length(v3);
120   else return fabs(Cross(v1, v2)) / Length(v1);
121 }
122
123 //点在直线上的投影
124 Point GetLineProjection(Point P, Point A, Point B) {
125   Vector v = B - A;
126   return A+v*(Dot(v, P-A)/Dot(v, v));
127 }
128
129 //线段相交判定, 交点不在一条线段的端点
130 bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2) {
131   double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);
132   double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);
133   return dcmp(c1)*dcmp(c2) < 0 && dcmp(c3)*dcmp(c4) < 0;
134 }
135
136 //判断点是否在点段上, 不包含端点
137 bool OnSegment(Point P, Point a1, Point a2) {
138   return dcmp(Cross(a1-P, a2-P) == 0 && dcmp((Dot(a1-P, a2-P)) < 0));
139 }
140
141 //计算凸多边形面积
142 double ConvexPolygonArea(Point *p, int n) {
143   double area = 0;
144   for(int i = 1; i < n-1; i++)
145     area += Cross(p[i] - p[0], p[i+1] - p[0]);
146   return area/2;
147 }
148
149 //计算多边形的有向面积
150 double PolygonArea(Point *p, int n) {
151   double area = 0;
152   for(int i = 1; i < n-1; i++)
153     area += Cross(p[i] - p[0], p[i+1] - p[0]);
154   return area/2;
155 }
```

```
156
157 /*************************************************************************
158 * Morley定理：三角形每个内角的三等分线，相交成的三角形是等边三角形。
159 * 欧拉定理：设平面图的定点数，边数和面数分别为V,E,F。则V+F-E = 2;
160 *************************************************************************/
161
162 struct Circle {
163   Point c;
164   double r;
165
166   Circle(Point c, double r) : c(c), r(r) {}
167   //通过圆心角确定圆上坐标
168   Point point(double a) {
169     return Point(c.x + cos(a)*r, c.y + sin(a)*r);
170   }
171 };
172
173 struct Line {
174   Point p;
175   Vector v;
176   double ang;
177   Line() {}
178   Line(Point p, Vector v) : p(p), v(v) {}
179   bool operator < (const Line& L) const {
180     return ang < L.ang;
181   }
182 };
183
184 //直线和圆的交点，返回交点个数，结果存在sol中。
185 //该代码没有清空sol。
186 int getLineCircleIntersecion(Line L, Circle C, double& t1, double& t2, vector<Point>&
        sol) {
187   double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
188   double e = a*a + c*c, f = 2*(a*b + c*d), g = b*b + d*d - C.r*C.r;
189   double delta = f*f - 4*e*g;
190   if(dcmp(delta) < 0) return 0; //相离
191   if(dcmp(delta) == 0) {        //相切
192     t1 = t2 = -f / (2*e);
193     sol.push_back(C.point(t1));
194     return 1;
195   }
196   //相交
197   t1 = (-f - sqrt(delta)) / (2*e);
198   sol.push_back(C.point(t1));
199   t2 = (-f + sqrt(delta)) / (2*e);
200   sol.push_back(C.point(t2));
201   return 2;
202 }
203
204 //两圆相交
205 int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol) {
206   double d = Length(C1.c - C2.c);
207   if(dcmp(d) == 0) {
208     if(dcmp(C1.r - C2.r == 0)) return -1;    //两圆完全重合
209     return 0;                                //同心圆，半径不一样
210   }
211   if(dcmp(C1.r + C2.r - d) < 0) return 0;
212   if(dcmp(fabs(C1.r - C2.r) == 0)) return -1;
213
214   double a = Angle(C2.c - C1.c);               //向量C1C2的极角
215   double da = acos((C1.r*C1.r + d*d - C2.r*C2.r) / (2*C1.r*d));
216   //C1C2到C1P1的角
217   Point p1 = C1.point(a-da), p2 = C1.point(a+da);
218   sol.push_back(p1);
```

```
219   if(p1 == p2) return 1;
220   sol.push_back(p2);
221   return 2;
222 }
223
224 const double PI = acos(-1);
225 //过定点做圆的切线
226 //过点p做圆C的切线, 返回切线个数。v[i]表示第i条切线
227 int getTangents(Point p, Circle C, Vector* v) {
228   Vector u = C.c - p;
229   double dist = Length(u);
230   if(dist < C.r) return 0;
231   else if(dcmp(dist - C.r) == 0) {
232     v[0] = Rotate(u, PI/2);
233     return 1;
234   } else {
235     double ang = asin(C.r / dist);
236     v[0] = Rotate(u, -ang);
237     v[1] = Rotate(u, +ang);
238     return 2;
239   }
240 }
241
242 //两圆的公切线
243 //返回切线的个数, -1表示有无数条公切线。
244 //a[i], b[i] 表示第i条切线在圆A, 圆B上的切点
245 int getTangents(Circle A, Circle B, Point *a, Point *b) {
246   int cnt = 0;
247   if(A.r < B.r) {
248     swap(A, B);
249     swap(a, b);
250   }
251   int d2 = (A.c.x - B.c.x)*(A.c.x - B.c.x) + (A.c.y - B.c.y)*(A.c.y - B.c.y);
252   int rdiff = A.r - B.r;
253   int rsum = A.r + B.r;
254   if(d2 < rdiff*rdiff) return 0;   //内含
255   double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
256   if(d2 == 0 && A.r == B.r) return -1;   //无限多条切线
257   if(d2 == rdiff*rdiff) {         //内切一条切线
258     a[cnt] = A.point(base);
259     b[cnt] = B.point(base);
260     cnt++;
261     return 1;
262   }
263   //有外共切线
264   double ang = acos((A.r-B.r) / sqrt(d2));
265   a[cnt] = A.point(base+ang);
266   b[cnt] = B.point(base+ang);
267   cnt++;
268   a[cnt] = A.point(base-ang);
269   b[cnt] = B.point(base-ang);
270   cnt++;
271   if(d2 == rsum*rsum) {  //一条公切线
272     a[cnt] = A.point(base);
273     b[cnt] = B.point(PI+base);
274     cnt++;
275   } else if(d2 > rsum*rsum) {   //两条公切线
276     double ang = acos((A.r + B.r) / sqrt(d2));
277     a[cnt] = A.point(base+ang);
278     b[cnt] = B.point(PI+base+ang);
279     cnt++;
280     a[cnt] = A.point(base-ang);
281     b[cnt] = B.point(PI+base-ang);
282     cnt++;
```

```
283   }
284   return cnt;
285 }
286
287 typedef vector<Point> Polygon;
288
289 //点在多边形内的判定
290 int isPointInPolygon(Point p, Polygon poly) {
291   int wn = 0;
292   int n = poly.size();
293   for(int i = 0; i < n; i++) {
294     if(OnSegment(p, poly[i], poly[(i+1)%n])) return -1; //在边界上
295     int k = dcmp(Cross(poly[(i+1)%n]-poly[i], p-poly[i]));
296     int d1 = dcmp(poly[i].y - p.y);
297     int d2 = dcmp(poly[(i+1)%n].y - p.y);
298     if(k > 0 && d1 <= 0 && d2 > 0) wn++;
299     if(k < 0 && d2 <= 0 && d1 > 0) wn++;
300   }
301   if(wn != 0) return 1;       //内部
302   return 0;                   //外部
303 }
304
305 //凸包
306 /************************************************************
307 * 输入点数组p, 个数为p, 输出点数组ch。 返回凸包顶点数
308 * 不希望凸包的边上有输入点, 把两个<= 改成 <
309 * 高精度要求时建议用dcmp比较
310 * 输入点不能有重复点。函数执行完以后输入点的顺序被破坏
311 ************************************************************/
312 int ConvexHull(Point *p, int n, Point* ch) {
313   sort(p, p+n);      //先比较x坐标, 再比较y坐标
314   int m = 0;
315   for(int i = 0; i < n; i++) {
316     while(m > 1 && Cross(ch[m-1] - ch[m-2], p[i]-ch[m-2]) <= 0) m--;
317     ch[m++] = p[i];
318   }
319   int k = m;
320   for(int i = n-2; i >= 0; i++) {
321     while(m > k && Cross(ch[m-1] - ch[m-2], p[i]-ch[m-2]) <= 0) m--;
322     ch[m++] = p[i];
323   }
324   if(n > 1) m--;
325   return m;
326 }
327
328 //用有向直线A->B切割多边形poly, 返回“左侧”。 如果退化, 可能会返回一个单点或者线段
329 //复杂度O(n2);
330 Polygon CutPolygon(Polygon poly, Point A, Point B) {
331   Polygon newpoly;
332   int n = poly.size();
333   for(int i = 0; i < n; i++) {
334     Point C = poly[i];
335     Point D = poly[(i+1)%n];
336     if(dcmp(Cross(B-A, C-A)) >= 0) newpoly.push_back(C);
337     if(dcmp(Cross(B-A, C-D)) != 0) {
338       Point ip = GetLineIntersection(A, B-A, C, D-C);
339       if(OnSegment(ip, C, D)) newpoly.push_back(ip);
340     }
341   }
342   return newpoly;
343 }
344
345 //半平面交
346
```

```
347 //点p再有向直线L的左边。（线上不算）
348 bool Onleft(Line L, Point p) {
349   return Cross(L.v, p-L.p) > 0;
350 }
351
352 //两直线交点，假定交点唯一存在
353 Point GetIntersection(Line a, Line b) {
354   Vector u = a.p - b.p;
355   double t = Cross(b.v, u) / Cross(a.v, b.v);
356   return a.p+a.v*t;
357 }
358
359 int HalfplaneIntersection(Line* L, int n, Point* poly) {
360   sort(L, L+n);                //按极角排序
361
362   int first, last;             //双端队列的第一个元素和最后一个元素
363   Point *p = new Point[n];     //p[i]为q[i]和q[i+1]的交点
364   Line *q = new Line[n];       //双端队列
365   q[first = last = 0] = L[0]; //队列初始化为只有一个半平面L[0]
366   for(int i = 0; i < n; i++) {
367     while(first < last && !Onleft(L[i], p[last-1])) last--;
368     while(first < last && !Onleft(L[i], p[first])) first++;
369     q[++last] = L[i];
370     if(fabs(Cross(q[last].v, q[last-1].v)) < EPS) {
371       last--;
372       if(Onleft(q[last], L[i].p)) q[last] = L[i];
373     }
374     if(first < last) p[last-1] = GetIntersection(q[last-1], q[last]);
375   }
376   while(first < last && !Onleft(q[first], p[last-1])) last--;
377   //删除无用平面
378   if(last-first <= 1) return 0;   //空集
379   p[last] = GetIntersection(q[last], q[first]);
380
381   //从deque复制到输出中
382   int m = 0;
383   for(int i = first; i <= last; i++) poly[m++] = p[i];
384   return m;
385 }
```