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 Circle

Base

```
1   /* 2D Circle Base Class, by Abreto<m@abreto.net>. */
2
3   /* requirement: point.cc */
4   #include "point.cc"
5
6   #include <utility>
7
8   namespace ab_geometry_2d {
9
10  using namespace std;
11
12  struct circle {
13    point o;
14    T r;
15    circle(void) : r(T()) {}
16    circle(point center, T radius) : o(center), r(radius) {}
17
18    inline ab_float arclen(ab_float theta) {
19      return theta * r;
20    }
21    inline ab_float circumference(void) {
22      return 2. * pi * r;
23    }
24    inline ab_float area(void) {
25      return pi * r * r;
26    }
27
28    /* bool contain(const circle &C, const bool including_touch = false) const
29    {
30        T dis2 = (o->*(C.o)).norm2();
31        T raw_diff = r - C.r;
32        if ( -1 == sgn(raw_diff) ) return false;
33        T dr2 = sqr(raw_diff);
34        return (dis2 < dr2) || (including_touch && (dis2 == dr2));
35    }
36    inline bool in(const circle &C, const bool including_touch = false) const
37    {
38      return C.contain(*this, including_touch);
39    } */
40    enum relation_t {
41      same = 0x00000,
42      contain = 0x00001,
43      intouch = 0x00010,
44      intersect = 0x00100,
45      outtouch = 0x01000,
46      separate = 0x10000,
47      unknow_relation = 0xfffff
```

```
48    };
49    relation_t with(const circle &C) const {
50      T dis2 = (o->*(C.o)).norm2();
51      T dr2 = sqr(r - C.r), rs2 = sqr(r + C.r);
52      if ( 0 == sgn(dis2) && 0 == sgn(dr2) ) return same;
53      if ( -1 == sgn(dis2 - dr2) ) return contain;
54      if ( 0 == sgn(dis2 - dr2) ) return intouch;
55      if ( -1 == sgn(dr2 - dis2) && -1 == sgn(dis2 - rs2) ) return intersect;
56      if ( 0 == sgn(dis2 - rs2) ) return outtouch;
57      if ( -1 == sgn(rs2 - dis2) ) return separate;
58      return unknow_relation;
59    }
60
61    enum point_relation_t {
62      in = 0x001,
63      on = 0x010,
64      out = 0x100,
65      unknow_point_relation = 0xfff
66    };
67    point_relation_t with(const point &P) const {
68      T dis2 = (o->*P).norm2();
69      T r2 = sqr(r);
70      int type = sgn(dis2 - r2);
71      if (-1 == type) return in;
72      if ( 0 == type) return on;
73      if (+1 == type) return out;
74      return unknow_point_relation;
75    }
76
77    ab_float central_angle(const point &A, const point &B, const bool reflex = false)
          const {
78      T dot = (A * B);
79      if (0 == sgn(dot)) return 1. * (A != B) * pi;
80      ab_float angle = ((ab_float)(dot)) / r / r;
81      if ( reflex ) angle = 2. * pi - angle;
82      return angle;
83    }
84
85    /* be sure (*this) intersect with C */
86    pair<point,point> crosspoint(const circle &C) const {
87      ab_float d = (o ->* (C.o)).norm();
88      // TODO:
89    }
90  };
91
92  }
```

k 次圆交

```
1   //china no.1
2   #pragma comment(linker, "/STACK:1024000000,1024000000")
3   #include <vector>
4   #include <iostream>
5   #include <string>
6   #include <map>
7   #include <stack>
8   #include <cstring>
9   #include <queue>
10  #include <list>
11  #include <stdio.h>
12  #include <set>
13  #include <algorithm>
14  #include <cstdlib>
15  #include <cmath>
16  #include <iomanip>
```

```
17 #include <cctype>
18 #include <sstream>
19 #include <functional>
20 #include <stdlib.h>
21 #include <time.h>
22 #include <bitset>
23 using namespace std;
24
25 #define pi acos(-1)
26 #define PI acos(-1)
27 #define endl '\n'
28 #define srand() srand(time(0));
29 #define me(x,y) memset(x,y,sizeof(x));
30 #define foreach(it,a) for(__typeof((a).begin()) it=(a).begin();it!=(a).end();it++)
31 #define close() ios::sync_with_stdio(0); cin.tie(0);
32 #define FOR(x,n,i) for(int i=x;i<=n;i++)
33 #define FOr(x,n,i) for(int i=x;i<n;i++)
34 #define W while
35 #define sgn(x) ((x) < 0 ? -1 : (x) > 0)
36 #define bug printf("**********\n");
37 #define db double
38 typedef long long LL;
39 const int INF=0x3f3f3f3f;
40 const LL LINF=0x3f3f3f3f3f3f3f3fLL;
41 const int dx[]= {-1,0,1,0,1,-1,-1,1};
42 const int dy[]= {0,1,0,-1,-1,1,-1,1};
43 const int maxn=1e3+10;
44 const int maxx=1e6+100;
45 const double EPS=1e-8;
46 const double eps=1e-8;
47 const int mod=10000007;
48 template<class T>inline T min(T a,T b,T c) {
49   return min(min(a,b),c);
50 }
51 template<class T>inline T max(T a,T b,T c) {
52   return max(max(a,b),c);
53 }
54 template<class T>inline T min(T a,T b,T c,T d) {
55   return min(min(a,b),min(c,d));
56 }
57 template<class T>inline T max(T a,T b,T c,T d) {
58   return max(max(a,b),max(c,d));
59 }
60 template <class T>
61 inline bool scan_d(T &ret) {
62   char c;
63   int sgn;
64   if (c = getchar(), c == EOF) {
65     return 0;
66   }
67   while (c != '-' && (c < '0' || c > '9')) {
68     c = getchar();
69   }
70   sgn = (c == '-') ? -1 : 1;
71   ret = (c == '-') ? 0 : (c - '0');
72   while (c = getchar(), c >= '0' && c <= '9') {
73     ret = ret * 10 + (c - '0');
74   }
75   ret *= sgn;
76   return 1;
77 }
78
79 inline bool scan_lf(double &num) {
80   char in;
```

```
  double Dec=0.1;
  bool IsN=false,IsD=false;
  in=getchar();
  if(in==EOF) return false;
  while(in!='-'&&in!='.'&&(in<'0'||in>'9'))in=getchar();
  if(in=='-') {
    IsN=true;
    num=0;
  } else if(in=='.') {
    IsD=true;
    num=0;
  } else num=in-'0';
  if(!IsD) {
    while(in=getchar(),in>='0'&&in<='9') {
      num*=10;
      num+=in-'0';
    }
  }
  if(in!='.') {
    if(IsN) num=-num;
    return true;
  } else {
    while(in=getchar(),in>='0'&&in<='9') {
      num+=Dec*(in-'0');
      Dec*=0.1;
    }
  }
  if(IsN) num=-num;
  return true;
}

void Out(LL a) {
  if(a < 0) {
    putchar('-');
    a = -a;
  }
  if(a >= 10) Out(a / 10);
  putchar(a % 10 + '0');
}
void print(LL a) {
  Out(a),puts("");
}
//freopen( "in.txt" , "r" , stdin );
//freopen( "data.txt" , "w" , stdout );
//cerr << "run time is " << clock() << endl;
/*struct Point
{
    double x, y;
    Point(const Point& rhs): x(rhs.x), y(rhs.y) { } //拷贝构造函数
    Point(double x = 0, double y = 0) : x(x), y(y) { }
    inline void input()
    {
        scanf("%lf%lf",&x,&y);
    }
    inline void print()
    {
        printf("%.6lf %.6lf\n",x,y);
    }
};*/
db sqr(db x) {
  return x*x;
}
int dcmp(double x) {
  if(fabs(x) < EPS) return 0;
```

```
145   else return x < 0 ? -1 : 1;
146 }
147 struct Circle {
148   double x, y, r, angle;
149   int d;
150   Circle() {}
151   Circle(double xx, double yy, double ang = 0, int t = 0) {
152     x = xx;
153     y = yy;
154     angle = ang;
155     d = t;
156   }
157   void get() {
158     scanf("%lf%lf%lf", &x, &y, &r);
159     d = 1;
160   }
161 };
162 Circle cir[maxn],tp[maxn*2];
163 double area[maxn];
164 double dis(Circle a,Circle b) {
165   return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
166 }
167 double cross(Circle p0,Circle p1,Circle p2) {
168   return (p1.x - p0.x) * (p2.y - p0.y) - (p1.y - p0.y) * (p2.x - p0.x);
169 }
170 //圆相交
171 int CirCrossCir(Circle p1, double r1,Circle p2, double r2,Circle &cp1,Circle &cp2) {
172   double mx = p2.x - p1.x, sx = p2.x + p1.x, mx2 = mx * mx;
173   double my = p2.y - p1.y, sy = p2.y + p1.y, my2 = my * my;
174   double sq = mx2 + my2, d = -(sq - sqr(r1 - r2)) * (sq - sqr(r1 + r2));
175   if (d + eps < 0) return 0;
176   if (d < eps) d = 0;
177   else d = sqrt(d);
178   double x = mx * ((r1 + r2) * (r1 - r2) + mx * sx) + sx * my2;
179   double y = my * ((r1 + r2) * (r1 - r2) + my * sy) + sy * mx2;
180   double dx = mx * d, dy = my * d;
181   sq *= 2;
182   cp1.x = (x - dy) / sq;
183   cp1.y = (y + dx) / sq;
184   cp2.x = (x + dy) / sq;
185   cp2.y = (y - dx) / sq;
186   if (d > eps) return 2;
187   else return 1;
188 }
189 bool circmp(const Circle& u, const Circle& v) {
190   return dcmp(u.r - v.r) < 0;
191 }
192 bool cmp(const Circle& u, const Circle& v) {
193   if (dcmp(u.angle - v.angle)) return u.angle < v.angle;
194   return u.d > v.d;
195 }
196 //0.5*r*r*(K-sin(K))
197 double calc(Circle cir,Circle cp1,Circle cp2) {
198   double ans = (cp2.angle - cp1.angle) * sqr(cir.r)
199                - cross(cir, cp1, cp2) + cross(Circle(0, 0), cp1, cp2);
200   return ans / 2;
201 }
202
203 void CirUnion(Circle cir[], int n) {
204   Circle cp1, cp2;
205   sort(cir, cir + n, circmp);
206   for (int i = 0; i < n; ++i)
207     for (int j = i + 1; j < n; ++j)
208       if (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir[j].r) <= 0)
```

```
209         cir[i].d++;
210   for (int i = 0; i < n; ++i) {
211     int tn = 0, cnt = 0;
212     for (int j = 0; j < n; ++j) {
213       if (i == j) continue;
214       if (CirCrossCir(cir[i], cir[i].r, cir[j], cir[j].r,
215                       cp2, cp1) < 2) continue;
216       cp1.angle = atan2(cp1.y - cir[i].y, cp1.x - cir[i].x);
217       cp2.angle = atan2(cp2.y - cir[i].y, cp2.x - cir[i].x);
218       cp1.d = 1;
219       tp[tn++] = cp1;
220       cp2.d = -1;
221       tp[tn++] = cp2;
222       if (dcmp(cp1.angle - cp2.angle) > 0) cnt++;
223     }
224     tp[tn++] = Circle(cir[i].x - cir[i].r, cir[i].y, pi, -cnt);
225     tp[tn++] = Circle(cir[i].x - cir[i].r, cir[i].y, -pi, cnt);
226     sort(tp, tp + tn, cmp);
227     int p, s = cir[i].d + tp[0].d;
228     for (int j = 1; j < tn; ++j) {
229       p = s;
230       s += tp[j].d;
231       area[p] += calc(cir[i], tp[j - 1], tp[j]);
232     }
233   }
234 }
235 int n;
236 void solve() {
237   for(int i=0; i<n; i++)
238     cir[i].get();
239   me(area,0);
240   CirUnion(cir,n);
241   for(int i=1; i<=n; i++) {
242     area[i]-=area[i+1];
243     printf("[%d] = %.3f\n", i, area[i]);
244   }
245 }
246 int main() {
247   while(scanf("%d",&n)!=EOF)
248     solve();
249 }
```

universe

```
1
2  Point CircumCenter(Point a,Point b,Point c) { //三角形的外心
3    Point cp;
4    double a1 = b.x-a.x,b1 = b.y-a.y,c1 = (a1*a1 + b1*b1)/2;
5    double a2 = c.x-a.x,b2 = c.y-a.y,c2 = (a2*a2 + b2*b2)/2;
6    double d = a1*b2 - a2*b1;
7    cp.x = a.x + (c1*b2-c2*b1)/d;
8    cp.y = a.y + (a1*c2-a2*c1)/d;
9    return cp;
10 }
```

3.1.3 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;
```

```

point O;

bool comp_angle(point_t a, point_t b) {
  double t = (a-O).X(b-O);
  if(fe(t,0.0)) return fl((b-O).mag2(),(a-O).mag2());
  else return fl(0.0,t);
}

void convex_hull_graham(vp& convex, vp src) {
  int i = 0, top = 0;
  O = src[0];
  for(auto pt : src)
    if( pt.x < O.x || (pt.x == O.x && pt.y < O.y))
      O = pt;
  sort(src.begin(), src.end(), comp_angle);
  convex.push_back(src[0]);
  convex.push_back(src[1]);
  top = 1;
  for(i = 2; i < src.size(); ++i) {
    while(top>1 && fle((convex[top]-convex[top-1]).X(src[i]-convex[top]),0.0)) {
      convex.pop_back();
      --top;
    }
    convex.push_back(src[i]);
    ++top;
  }
}
```

3.1.4 Intersect Area

```
#include <cstdio>
#include <cmath>
#include <algorithm>

using namespace std;

//#define inf 1000000000000
#define M 8
#define LL long long
#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;
  }
};
```

```
33 double max(double a,double b) {
34   return a>b?a:b;
35 }
36 double min(double a,double b) {
37   return a<b?a:b;
38 }
39 double len(node a) {
40   return sqrt(a*a);
41 }
42 double dis(node a,node b) { //两点之间的距离
43   return len(b-a);
44 }
45 double cross(node a,node b,node c) { //叉乘
46   return (b-a)^(c-a);
47 }
48 double dot(node a,node b,node c) { //点成
49   return (b-a)*(c-a);
50 }
51 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）
52   if(c.x>=min(a.x,b.x)
53       &&c.x<=max(a.x,b.x)
54       &&c.y>=min(a.y,b.y)
55       &&c.y<=max(a.y,b.y))
56     return 1;
57   return 0;
58 }
59 double area(node b,node c,double r) {
60   node a(0.0,0.0);
61   if(dis(b,c)<eps)
62     return 0.0;
63   double h=fabs(cross(a,b,c))/dis(b,c);
64   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
65     double angle=acos(dot(a,b,c)/dis(a,b)/dis(a,c));
66     if(h>r-eps) {
67       return 0.5*r*r*angle;
68     } else if(dot(b,a,c)>0&&dot(c,a,b)>0) {
69       double angle1=2*acos(h/r);
70       return 0.5*r*r*fabs(angle-angle1)+0.5*r*r*sin(angle1);
71     } else {
72       return 0.5*r*r*angle;
73     }
74   } 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   }
139   return f(x,l);
140 }
141
142 double findmin(double x1, double x2, double y1, double y2) {
143   double l = x1, r = x2;
144   while(r-l>eps) {
145     double mid = (l+r)/2.0;
146     double mmid = (mid+r)/2.0;
147     if( trifind(mmid,y1,y2) > trifind(mid,y1,y2)+eps )
148       l = mid;
149     else
150       r = mmid;
151   }
152   return trifind(l,y1,y2);
153 }
154
155 double ans(int a, int b, int c, int r) {
156   A = node(0,0);
157   B = node((double)c,0);
158   R = r;
159   double da = a, db = b, dc = c;
160   double cosa = (db*db+dc*dc-da*da)/(2.0*db*dc);
```

```
161   double alpha = acos(cosa);
162   C = node(db*cosa, db*sin(alpha));
163   return findmin(0.0, c, 0.0, db*sin(alpha));
164 }
165
166 int main(void) {
167   int a = 0, b = 0, c = 0, r = 0;
168   while(EOF != scanf("%d%d%d%d",&a,&b,&c,&r) && (a||b||c||r))
169     printf("%.8lf\n", ans(a,b,c,r));
170   return 0;
171 }
```

3.1.5 Universe

```
  1 #include <bits/stdc++.h>
  2 using namespace std;
  3
  4 struct Point {
  5   double x, y;
  6   Point(double x = 0, double y = 0) : x(x), y(y) {}
  7 };
  8
  9 typedef Point Vector;
 10
 11 Vector operator + (Vector A, Vector B) {
 12   return Vector(A.x + B.x, A.y + B.y);
 13 }
 14 Vector operator - (Vector A, Vector B) {
 15   return Vector(A.x - B.x, A.y - B.y);
 16 }
 17 Vector operator * (Vector A, double p) {
 18   return Vector(A.x*p, A.x*p);
 19 }
 20 Vector operator / (Vector A, double p) {
 21   return Vector(A.x/p, A.x/p);
 22 }
 23
 24 bool operator < (const Point& a, const Point b) {
 25   return a.x < b.x || (a.x == b.x && a.y < b.y);
 26 }
 27
 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
```

```
//凸包
/*****************************************************************
* 输入点数组p， 个数为p， 输出点数组ch。 返回凸包顶点数
* 不希望凸包的边上有输入点，把两个<= 改成 <
* 高精度要求时建议用dcmp比较
* 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
*****************************************************************/
int ConvexHull(Point *p, int n, Point* ch) {
  sort(p, p+n);      //先比较x坐标，再比较y坐标
  int m = 0;
  for(int i = 0; i < n; i++) {
    while(m > 1 && Cross(ch[m-1] - ch[m-2], p[i]-ch[m-2]) <= 0) m--;
    ch[m++] = p[i];
  }
  int k = m;
  for(int i = n-2; i >= 0; i++) {
    while(m > k && Cross(ch[m-1] - ch[m-2], p[i]-ch[m-2]) <= 0) m--;
    ch[m++] = p[i];
  }
  if(n > 1) m--;
  return m;
}

//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
//复杂度O(n2);
Polygon CutPolygon(Polygon poly, Point A, Point B) {
  Polygon newpoly;
  int n = poly.size();
  for(int i = 0; i < n; i++) {
    Point C = poly[i];
    Point D = poly[(i+1)%n];
    if(dcmp(Cross(B-A, C-A)) >= 0) newpoly.push_back(C);
    if(dcmp(Cross(B-A, C-D)) != 0) {
      Point ip = GetLineIntersection(A, B-A, C, D-C);
      if(OnSegment(ip, C, D)) newpoly.push_back(ip);
    }
  }
  return newpoly;
}

//半平面交

//点p再有向直线L的左边。（线上不算）
bool Onleft(Line L, Point p) {
  return Cross(L.v, p-L.p) > 0;
}

//两直线交点，假定交点唯一存在
Point GetIntersection(Line a, Line b) {
  Vector u = a.p - b.p;
  double t = Cross(b.v, u) / Cross(a.v, b.v);
  return a.p+a.v*t;
}

int HalfplaneIntersection(Line* L, int n, Point* poly) {
  sort(L, L+n);              //按极角排序

  int first, last;           //双端队列的第一个元素和最后一个元素
  Point *p = new Point[n];   //p[i]为q[i]和q[i+1]的交点
  Line *q = new Line[n];     //双端队列
  q[first = last = 0] = L[0]; //队列初始化为只有一个半平面L[0]
  for(int i = 0; i < n; i++) {
    while(first < last && !Onleft(L[i], p[last-1])) last--;
    while(first < last && !Onleft(L[i], p[first])) first++;
```

```
    q[++last] = L[i];
    if(fabs(Cross(q[last].v, q[last-1].v)) < EPS) {
      last--;
      if(Onleft(q[last], L[i].p)) q[last] = L[i];
    }
    if(first < last) p[last-1] = GetIntersection(q[last-1], q[last]);
  }
  while(first < last && !Onleft(q[first], p[last-1])) last--;
  //删除无用平面
  if(last-first <= 1) return 0;   //空集
  p[last] = GetIntersection(q[last], q[first]);

  //从deque复制到输出中
  int m = 0;
  for(int i = first; i <= last; i++) poly[m++] = p[i];
  return m;
}
```

4 Graph

4.1 Tree

4.1.1 Universe

```

/* find root(重心) */

void findroot(int u, int fa) {
  int i;
  size[u] = 1;
  f[u] = 0;
  for (i = last[u]; i; i = e[i][2]) {
    if (!vis[e[i][0]] && e[i][0] != fa) {
      findroot(e[i][0], u);
      size[u] += size[e[i][0]];
      if (f[u] < size[e[i][0]])
        f[u] = size[e[i][0]];
    }
  }
  if (f[u] < ALL - size[u])
    f[u] = ALL - size[u];
  if (f[u] < f[root]) root = u;
}

/* ---- da ---- */

int dep[MAXN+1];
int ancestor[MAXN+1][MAXLGN];
int minw[MAXN+1][MAXLGN];

void dfs(int u, int fa) {
  ancestor[u][0] = fa;
  dep[u] = dep[fa] + 1;
  for(int e = u[front]; e; e = E[e].n) {
    int v = E[e].v, w = E[e].w;
    if(v != fa) {
      minw[v][0] = w;
      dfs(v, u);
    }
  }
}

```

```
void init_system(void) {
  int i = 0, w = 0;
  int t = 0;
  dep[0] = -1;
  dfs(1,0);
  for(w = 1; (t=(1<<w)) < N; ++w)
    for(i = 1; i <= N; ++i) if( dep[i] >= t ) {
        ancestor[i][w] = ancestor[ancestor[i][w-1]][w-1];
        minw[i][w] = min(minw[i][w-1], minw[ancestor[i][w-1]][w-1]);
      }
}

int query(int a, int b) {
  if(dep[a] < dep[b]) return query(b,a);
  else {  /* now dep[s] > dep[t] */
    int i = 0;
    int maxbit = MAXLGN-1;
    int ret = INF;
    //while((1<<maxbit) <= dep[a]) maxbit++;
    /* first up a to same dep with b. */
    for(i = maxbit; i >= 0; i--)
      if(dep[a] - (1<<i) >= dep[b]) {
        ret = min(ret, minw[a][i]);
        a = ancestor[a][i];
      }
    if(a == b) return ret;
    for(i = maxbit; i >= 0; i--)
      if(dep[a] - (1<<i) >= 0 && ancestor[a][i] != ancestor[b][i]) {
        ret = min(ret, min(minw[a][i], minw[b][i]));
        a = ancestor[a][i];
        b = ancestor[b][i];
      }
    ret = min(ret, min(minw[a][0], minw[b][0]));
    return ret;
  }
}
```

4.1.2 Point Divide and Conquer

Version 1

```
/* Tree::Point divide and conquer, by Abreto<m@abreto.net>. */
#include <bits/stdc++.h>

using namespace std;
typedef long long int ll;

#define MAXN    (100001)
#define MAXV    (MAXN+1)
#define MAXE    (MAXN<<1)
struct edge {
  int v;
  edge *n;
  edge(void):v(0),n(NULL) {}
  edge(int vv,edge *nn):v(vv),n(nn) {}
};
int nE;
edge E[MAXE];
edge *front[MAXV];
int label[MAXV];    /* 0 for '(', 1 for ')' */
void add_edge(int u, int v) {
  int ne = ++nE;
  E[ne] = edge(v, u[front]);
```

```
23    u[front] = &(E[ne]);
24  }
25
26  int n;
27  ll ans;
28
29  char del[MAXV];
30  namespace findroot {
31  int ALL;
32  int nfind;
33  int vis[MAXV];
34  int size[MAXV];
35  int f[MAXV];
36  int root;
37  void __find(int u, int fa) {
38    vis[u] = nfind;
39    size[u] = 1;
40    f[u] = 0;
41    for(edge *e=u[front]; e; e = e->n) {
42      int v = e->v;
43      if((!del[v]) && (vis[v] != nfind) && (v != fa)) {
44        __find(v, u);
45        size[u] += size[v];
46        if(f[u] < size[v])  f[u] = size[v];
47      }
48    }
49    if(f[u] < ALL - size[u])  f[u] = ALL - size[u];
50    if(f[u] < f[root])  root = u;
51  }
52  int find(int u, int all) {
53    ++nfind;
54    ALL = all;
55    f[root = 0] = MAXV;
56    __find(u,0);
57    return root;
58  }
59  }
60
61  namespace workspaces {
62  int maxdep;
63  int dep[MAXV];
64  ll cntin[MAXV], cntout[MAXV];
65  int in[2][MAXV];    /* 0 for '(', 1 for ')' */
66  int out[2][MAXV];
67  void getdeep(int u, int fa) {
68    dep[u] = dep[fa] + 1;
69    if(dep[u] > maxdep) maxdep = dep[u];
70    for(edge *e = u[front]; e; e = e->n)
71      if((!del[e->v]) && (fa != e->v))
72        getdeep(e->v, u);
73  }
74  void dfs(int u, int fa) {
75    {
76      /* out from root */
77      out[0][u] = out[0][fa];
78      out[1][u] = out[1][fa];
79      if(0 == label[u]) { /* meet '(' */
80        out[0][u]++;
81      } else {    /* meet ')' */
82        if(out[0][u]) out[0][u]--;
83        else out[1][u]++;
84      }
85      if(out[0][u] == 0)
86        cntout[out[1][u]]++;
```

```
  }
  {
    /* in to root */
    in[0][u] = in[0][fa];
    in[1][u] = in[1][fa];
    if(0 == label[u]) { /* meet '(' */
      if(in[1][u]) in[1][u]--;
      else in[0][u]++;
    } else {    /* meet ')' */
      in[1][u]++;
    }
    if(0 == in[1][u])
      cntin[in[0][u]]++;
  }
  /* do something */
  for(edge *e = u[front]; e; e = e->n) {
    int v = e->v;
    if((!del[v]) && (v != fa)) {
      dfs(v, u);
    }
  }
}
inline void init_maxdep(void) {
  maxdep = 0;
}
inline void update_maxdep(int u) {
  dep[u] = 1;
  if(dep[u] > maxdep) maxdep = dep[u];
  for(edge *e = u[front]; e; e = e->n)
    if((!del[e->v]))
      getdeep(e->v, u);
}
inline void clear(void) {
  for(int i = 0; i <= maxdep+1; ++i)
    cntin[i] = cntout[i] = 0;
}
inline void work(int u) {
  in[0][u] = in[1][u] = out[0][u] = out[1][u] = 0;
  in[label[u]][u] = out[label[u]][u] = 1;
  if(out[0][u] == 0) cntout[out[1][u]]++;
  if(0 == in[1][u]) cntin[in[0][u]]++;
  /* update in and out if neccessary */
  for(edge *e = u[front]; e; e = e->n)
    if(!(del[e->v]))
      dfs(e->v, u);
}
};

ll count(int u, int p) {
  ll ret = 0;
  workspace::init_maxdep();
  workspace::update_maxdep(u);
  workspace::clear();
  if(-1 == p) {
    for(edge *e = u[front]; e; e = e->n)
      if((!(del[e->v])))
        workspace::work(e->v);
    p = label[u];
    /* single end */
    if(0 == p) ret = workspace::cntout[1];
    else ret = workspace::cntin[1];
  } else {
    workspace::work(u);
  }
```

```
151   if(0 == p) { /* p is '(' */
152     for(int i = 0; i < workspace::maxdep; ++i)  /* concatenation */
153       ret += workspace::cntin[i] * workspace::cntout[i+1];
154   } else {    /* p is ')' */
155     for(int i = 0; i < workspace::maxdep; ++i)  /* concatenation */
156       ret += workspace::cntin[i+1] * workspace::cntout[i];
157   }
158   return ret;
159 }
160
161 void handle(int u) {
162   del[u] = 1; /* delete current root. */
163   ans += count(u, -1);
164   /* do something */
165   for(edge *e = u[front]; e; e = e->n) {
166     int v = e->v;
167     if(!del[v]) {
168       ans -= count(v, label[u]);
169       /* do something */
170       int r = findroot::find(v, findroot::size[v]);
171       handle(r);
172     }
173   }
174 }
175
176 void proc(void) {
177   int r = findroot::find(1,n);
178   handle(r);
179 }
180
181 char ls[MAXV+1];
182 int main(void) {
183   int i = 0;
184   scanf("%d", &n);
185   scanf("%s", ls);
186   for(i = 0; i < n; ++i)
187     label[i+1] = ls[i] - '(';
188   for(i = 1; i < n; ++i) {
189     int ai, bi;
190     scanf("%d␣%d", &ai, &bi);
191     add_edge(ai, bi);
192     add_edge(bi, ai);
193   }
194   proc();
195   printf("%lld\n", ans);
196   return 0;
197 }
```

Version 2

```
1  /* 2016 ACM/ICPC Asia Regional Dalian. Problem , by Abreto<m@abreto.net>. */
2  #include <bits/stdc++.h>
3
4  using namespace std;
5  typedef long long int ll;
6
7  /* offset in [1,k] */
8  #define GET(i,offset)   (((i)>>((offset)-1))&1)
9  #define SET(i,offset)   ((i)|(1<<((offset)-1)))
10 #define REV(i,offset)   ((i)^(1<<((offset)-1)))
11
12 #define MAXN    (50005)
13 #define MAXV    (MAXN+1)
14 #define MAXE    (MAXN<<1)
15 struct edge {
```

```
16   int v;
17   edge *n;
18   edge(void):v(0),n(NULL) {}
19   edge(int vv,edge *nn):v(vv),n(nn) {}
20 };
21 int nE;
22 edge E[MAXE];
23 edge *front[MAXV];
24 int label[MAXV];    /* each kind */
25 void add_edge(int u, int v) {
26   int ne = ++nE;
27   E[ne] = edge(v, u[front]);
28   u[front] = &(E[ne]);
29 }
30
31 int n, k;
32 ll ans;
33 int all_kind;
34
35 int ndel;
36 int del[MAXV];
37 namespace findroot {
38 int ALL;
39 ll nfind;
40 ll vis[MAXV];
41 int size[MAXV];
42 int f[MAXV];
43 int root;
44 void __find(int u, int fa) {
45   vis[u] = nfind;
46   size[u] = 1;
47   f[u] = 0;
48   for(edge *e=u[front]; e; e = e->n) {
49     int v = e->v;
50     if((del[v] != ndel) && (vis[v] != nfind) && (v != fa)) {
51       __find(v, u);
52       size[u] += size[v];
53       if(f[u] < size[v])  f[u] = size[v];
54     }
55   }
56   if(f[u] < ALL - size[u])  f[u] = ALL - size[u];
57   if(f[u] < f[root])  root = u;
58 }
59 int find(int u, int all) {
60   ++nfind;
61   ALL = all;
62   f[root = 0] = MAXV;
63   __find(u,0);
64   return root;
65 }
66 }
67
68 namespace workspace {
69 ll cnt[1024];
70 int dp[MAXV];
71 void dfs(int u, int fa) {
72   dp[u] = dp[fa] | label[u];
73   cnt[dp[u]] ++;
74   /* dig into children */
75   for(edge *e = u[front]; e; e = e->n) {
76     int v = e->v;
77     if((del[v] != ndel) && (v != fa)) {
78       dfs(v, u);
79     }
```

```
  }
}
inline void clear(void) {
  for(int i = 1; i <= all_kind; ++i)
    cnt[i] = 0;
}
inline void work(int u) {
  dp[u] = label[u];
  cnt[dp[u]] ++;
  for(edge *e = u[front]; e; e = e->n)
    if((del[e->v] != ndel))
      dfs(e->v, u);
}
inline void show(void) {
  for(int i = 0; i <= all_kind; ++i)
    printf("cnt[%d] = %lld\n", i, cnt[i]);
  for(int i = 1; i <= n; ++i)
    printf("dp[%d] = %d\n", i, dp[i]);
}
};


ll count(int u, int p) {
  ll ret = 0;
  workspace::clear();
  //printf("%d,%d  :\n", u, p);
  if(-1 == p) {
    for(edge *e = u[front]; e; e = e->n)
      if(((del[e->v]) != ndel))
        workspace::work(e->v);
    p = label[u];
    /* single end */
    for(int i = 1; i <= all_kind; i++)
      if(all_kind == (i|p))
        ret += (workspace::cnt[i]<<1);
  } else {
    workspace::work(u);
  }
  //workspace::show();
  for(int i = 1; i <= all_kind; ++i)
    if( workspace::cnt[i] > 0 )
      for(int j = 1; j <= all_kind; ++j)
        if(all_kind == (i|p|j))
          ret += workspace::cnt[i] * workspace::cnt[j];
  //printf("%lld\n", ret);
  return ret;
}

void handle(int u) {
  //printf("proccessing %d\n", u);
  del[u] = ndel; /* delete current root. */
  ans += count(u, -1);
  /* do something */
  for(edge *e = u[front]; e; e = e->n) {
    int v = e->v;
    if(del[v] != ndel) {
      ans -= count(v, label[u]);
      /* do something */
      int r = findroot::find(v, findroot::size[v]);
      handle(r);
    }
  }
}

```

```
144 void proc(void) {
145   int r = findroot::find(1,n);
146   handle(r);
147 }
148
149 void clear(void) {
150   int i;
151   ans = 0;
152   nE = 0;
153   for(i = 0; i <= n; ++i) {
154     front[i] = NULL;
155   }
156   //findroot::nfind = 0;
157   ndel++;
158 }
159
160 void mozhu(void) {
161   int i = 0;
162   int li;
163   for(i = 1; i <= n; ++i) {
164     scanf("%d", &li);
165     label[i] = 1<<(li-1);
166   }
167   for(i = 1; i < n; ++i) {
168     int ai, bi;
169     scanf("%d %d", &ai, &bi);
170     add_edge(ai, bi);
171     add_edge(bi, ai);
172   }
173   all_kind = (1<<k)-1;
174   proc();
175   if(1 == k) ans += n;
176   printf("%lld\n", ans);
177 }
178
179 int main(void) {
180   while( EOF != scanf("%d%d", &n, &k) ) {
181     clear();
182     mozhu();
183   }
184   return 0;
185 }
```

4.1.3 Hevay chain decompostion

```
1  /* bzoj 1036 */
2  /* 树链剖分 */
3  #include <bits/stdc++.h>
4
5  using namespace std;
6
7  #define MAXN    30030
8  #define MAXM    (MAXN<<1)
9  struct edge {
10   int v;
11   edge *n;
12   edge(void) {}
13   edge(int vv, edge *nn):v(vv),n(nn) {}
14 };
15 typedef edge *ep;
16 int nE;
17 edge E[MAXM];
18 ep front[MAXN];
```

```
19 void add_edge(int u, int v) {
20   int ne = ++nE;
21   E[ne] = edge(v, u[front]);
22   u[front] = &(E[ne]);
23 }
24
25 int n;
26 int fa[MAXN], son[MAXN], sz[MAXN], dep[MAXN];
27 int top[MAXN];
28 int id[MAXN];
29 int tot;
30
31 void calc(int u, int uf) {
32   dep[u] = dep[uf] + 1;
33   fa[u] = uf;
34   sz[u] = 1;
35   son[u] = -1;
36   for(ep e = u[front]; e; e = e->n) {
37     if(e->v != uf) {
38       calc(e->v, u);
39       sz[u] += sz[e->v];
40       if( -1 == son[u] || sz[son[u]] < sz[e->v] )
41         son[u] = e->v;
42     }
43   }
44 }
45 void link(int u, int f) {
46   id[u] = (++tot);
47   top[u] = f;
48   if(son[u] > 0) {
49     link(son[u], f);
50   }
51   for(ep e = u[front]; e; e = e->n) {
52     if(e->v != fa[u] && e->v != son[u]) {
53       link(e->v, e->v);
54     }
55   }
56 }
57
58 /* 其实是树链剖分 */
59 void make_link_cut_tree(void) {
60   calc(1, 0);
61   link(1, 1);
62 }
63
64 int w[MAXN];
65 int sum[MAXN<<2], mx[MAXN<<2];
66
67 void maintain(int o, int l, int r) {
68   sum[o] = sum[o<<1] + sum[o<<1|1];
69   mx[o] = max(mx[o<<1], mx[o<<1|1]);
70 }
71 void build(int o = 1, int l = 1, int r = n) {
72   if(r == l) {
73     sum[o] = w[l];
74     mx[o] = w[l];
75   } else {
76     int mid = l+r>>1;
77     build(o<<1, l, mid);
78     build(o<<1|1, mid+1, r);
79     maintain(o, l, r);
80   }
81 }
82 void update(int p, int x, int o = 1, int l = 1, int r = n) {
```

```
 83   if(p <= l && r <= p) {
 84     sum[o] = x;
 85     mx[o] = x;
 86   } else {
 87     int mid = l+r>>1;
 88     if(p <= mid) update(p,x,o<<1,l,mid);
 89     else update(p,x,o<<1|1,mid+1,r);
 90     maintain(o,l,r);
 91   }
 92 }
 93 int qs(int L, int R, int o = 1, int l = 1, int r = n) {
 94   if(R < l || r < L) return 0;
 95   else if (L <= l && r <= R) {
 96     return sum[o];
 97   } else {
 98     int mid = l+r>>1;
 99     return qs(L,R,o<<1,l,mid)+qs(L,R,o<<1|1,mid+1,r);
100   }
101 }
102 int qm(int L, int R, int o = 1, int l = 1, int r = n) {
103   if(L <= l && r <= R) {
104     return mx[o];
105   } else {
106     int mid = l+r>>1;
107     if(R <= mid) return qm(L, R, o<<1, l, mid);
108     else if ( L > mid ) return qm(L, R, o<<1|1, mid+1, r);
109     else return max(qm(L, R, o<<1, l, mid),qm(L, R, o<<1|1, mid+1, r));
110   }
111 }
112
113 void change(int u, int t) {
114   update(id[u], t);
115 }
116 int qmax(int u, int v) {
117   int ret = -1000000000;
118   while(top[u] != top[v]) {
119     if( dep[top[u]] > dep[top[v]] ) {
120       /* jump u */
121       ret = max(ret, qm(id[top[u]], id[u]));
122       u = fa[top[u]];
123     } else {
124       ret = max(ret, qm(id[top[v]], id[v]));
125       v = fa[top[v]];
126     }
127   }
128   ret = max(ret, qm(min(id[u],id[v]),max(id[u],id[v])));
129   return ret;
130 }
131 int qsum(int u, int v) {
132   int ret = 0;
133   while(top[u] != top[v]) {
134     if( dep[top[u]] > dep[top[v]] ) {
135       /* jump u */
136       ret += qs(id[top[u]], id[u]);
137       u = fa[top[u]];
138     } else {
139       ret += qs(id[top[v]], id[v]);
140       v = fa[top[v]];
141     }
142   }
143   ret += qs(min(id[u],id[v]),max(id[u],id[v]));
144   return ret;
145 }
146
```

```
147 int main(void) {
148   int i;
149   scanf("%d", &n);
150   for(i = 1; i < n; ++i) {
151     int a, b;
152     scanf("%d%d", &a, &b);
153     add_edge(a, b);
154     add_edge(b, a);
155   }
156   make_link_cut_tree();
157   for(i = 1; i <= n; ++i) {
158     scanf("%d", &(w[id[i]]));
159   }
160   build();
161   scanf("%d", &i);
162   while(i--) {
163     char command[8];
164     int a, b;
165     scanf("%s␣%d␣%d", command, &a, &b);
166     if('C' == command[0]) change(a, b);
167     else if ('M' == command[1]) printf("%d\n", qmax(a, b));
168     else if ('S' == command[1]) printf("%d\n", qsum(a, b));
169   }
170   return 0;
171 }
```

4.2 2-SAT

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  namespace two_sat {
6  const int maxn = 100000;
7  const int maxm = 1000000;
8  struct edge {
9    int v;
10   edge *n;
11   edge(void):v(0),n(NULL) {}
12   edge(int vv, edge *nn):v(vv),n(nn) {}
13 };
14 typedef edge *ep;
15 int n;
16 int nE;
17 edge E[maxm];
18 ep front[maxn];
19 void add_edge(int u, int v) {
20   int ne = ++nE;
21   E[ne] = edge(v, u[front]);
22   u[front] = &(E[ne]);
23 }
24 /* (x = xval or y = yval), indexed from 0 */
25 void add_clause(int x, int xv, int y, int yv) {
26   x = x*2 + xv;
27   y = y*2 + yv;
28   add_edge(x^1, y);
29   add_edge(y^1, x);
30 }
31
32 char mark[maxn<<1];
33 int S[maxn<<1], c;
34 void init(int N) {
35   n = N;
```

```
  for(int i = 0; i < n*2; ++i) {
    i[front] = NULL;
    i[mark] = 0;
  }
  nE = 0;
}

int dfs(int x) {
  if(mark[x^1]) return 0;
  if(mark[x]) return 1;
  mark[x] = 1;
  S[c++] = x;
  for(ep e = x[front]; e; e = e->n)
    if(!dfs(e->v)) return 0;
  return 1;
}

int solve(void) {
  for(int i = 0; i < n*2; i += 2)
    if(!mark[i] && !mark[i+1]) {
      c = 0;
      if(!dfs(i)) {
        while(c > 0) mark[S[--c]] = 0;
        if(!dfs(i+1)) return 0;
      }
    }
  return 1;
}
}
```

4.3 Cut Edge and Point

```
Finding cut edges
The code below works properly because the lemma above (first lemma):
  h[root] = 0
            par[v] = -1
                     dfs (v):
                     d[v] = h[v]
                            color[v] = gray
                                     for u in adj[v]:
                                         if color[u] == white
                                           then par[u] = v and dfs(u) and d[v] = min(
                                               d[v], d[u])
                                            if d[u] > h[v]
                                              then the edge v-u is a cut edge
                                              else if u != par[v])
          then d[v] = min(d[v], h[u])
                      color[v] = black
                                In this code, h[v] = height of vertex v in the DFS
                                   tree and d[v] = min(h[w] where there is at least
                                   vertex u in subtree of v in the DFS tree where
                                   there is an edge between u and w).

                                Finding cut vertices
                                The code below works properly because the lemma
                                   above (first lemma):
                                h[root] = 0
                                          par[v] = -1
                                              dfs (v):
                                              d[v] = h[v]
                                                  color[v] = gray
                                            for u in adj[v]:
                                                      if color[u] == white
```

```
27                                                                  then par[u] = v and dfs(
                                                                       u) and d[v] = min(d[v
                                                                       ], d[u])
28                                                                     if d[u] >= h[v]
                                                                          and (v != root
                                                                          or
                                                                          number_of_children
                                                                          (v) > 1)
29                                                                    then the edge v
                                                                         is a cut
                                                                         vertex
30                                                                    else if u != par
                                                                         [v])
31                  then d[v] = min(d[v], h[u])
32                            color[v] = black
33                                       In this code, h[v] = height of vertex v in
                                            the DFS tree and d[v] = min(h[w] where
                                            there is at least vertex u in subtree of v
                                            in the DFS tree where there is an edge
                                            between u and w).
```

4.4 Euler Path

```
1  /* Euler path, by Abreto<m@abreto.net>. */
2  #define MAXV    (1024)
3  #define MAXE    (MAXV*MAXV)
4
5  typedef struct {
6    int id;
7    int nxt;
8    int del;
9  } egde_t;
10 int front[MAXV];
11 egde_t edg[MAXE];
12 int d[MAXV];
13 int ind[MAXV], outd[MAXV];
14 int nedges;
15 void add_edge(int u, int v) {
16   int newedge = ++nedges;
17   edg[newedge].id = v;
18   edg[newedge].nxt = u[front];
19   edg[newedge].del = 0;
20   u[front] = newedge;
21   outd[u]++;
22   ind[v]++;
23   d[u]++;
24   d[v]++;
25 }
26 void del_edge(int u, int v) {
27   int e = 0;
28   for(e=u[front]; e; e=edg[e].nxt)
29     if(edg[e].id==v) {
30       edg[e].del = 1;
31       outd[u]--;
32       ind[v]--;
33       d[u]--;
34       d[v]--;
35       return;
36     }
37 }
38
39 int path[MAXV];
40 int l;
```

```
41
42 void add2path(int u) {
43   path[l++] = u;
44 }
45
46 /* Directed graph */
47 void euler(int x) {
48   if(outd[x]) {
49     int e = 0;
50     for(e=x[front]; e; e=edg[e].nxt)
51       if(!edg[e].del) {
52         int v = edg[e].id;
53         del_edge(x,v);
54         euler(v);
55       }
56   }
57   add2path(x);
58 }
59
60 /* Undirected graph */
61 void euler(int x) {
62   if(d[x]) {
63     int e = 0;
64     for(e=x[front]; e; e=edg[e].nxt)
65       if(!edg[e].del) {
66         int v = edg[e].id;
67         del_edge(x,v);
68         del_edge(v,x);
69         euler(v);
70       }
71   }
72   add2path(x);
73 }
```

4.5 Shortest Path

4.5.1 Dijkstra

```
1 /* Shortest Path Dijstra, by Abreto<m@abreto.net>. */
2 #include <cstdio>
3 #include <set>
4 #include <utility>
5
6 using namespace std;
7 typedef set< pair<int,int> > spii;
8
9 #define MAXN    512
10 #define MAXV    (MAXN*MAXN)
11
12 struct egde_t {
13   int id;
14   int nxt;
15 };
16 int front[MAXV];
17 egde_t edg[MAXV<<3];
18 int nedges;
19 void add_edge(int u, int v) {
20   int newedge = ++nedges;
21   edg[newedge].id = v;
22   edg[newedge].nxt = u[front];
23   u[front] = newedge;
24 }
25
```

```
int d[MAXV];
int vis[MAXN];
int solid[MAXV];

int dijstra(int s, int t) {
  int v = s[front];
  spii q;
  q.insert(make_pair(0, s));
  while(!q.empty()) {
    auto it = q.begin();
    int u = it->second;
    int v = u[front];
    q.erase(it);
    solid[u] = 1;
    if(u == t) break;
    while(v) {
      int w = edg[v].id;
      if(!solid[w]) {
        if( (0==d[w]) || (d[u] + 1 < d[w]) ) {
          q.erase(make_pair(d[w],w));
          d[w] = d[u] + 1;
          q.insert(make_pair(d[w],w));
        }
      }
      v = edg[v].nxt;
    }
  }
  return d[t];
}
```

4.5.2 Shortest Path Fast Algorithm

```
/* Shortest Path Fast Algorithm, by Abreto<m@abreto.net>. */
#include <cstdio>
#include <cstring>
#include <queue>
#include <utility>

using namespace std;

#define MAXN    128

struct edge {
  int v;
  int w;
  int n;
};
edge edg[MAXN<<1];
int nedg;
int indegree[MAXN];
int front[MAXN];
int find_edge(int u, int v) {
  int e = u[front];
  while(e) {
    if(edg[e].v == v) return e;
    e = edg[e].n;
  }
  return 0;
}
void add_edge(int u, int v, int w) {
  int e = find_edge(u,v);
  if(0==e) {
    int newnode = ++nedg;
```

```
32      edg[newnode].v = v;
33      edg[newnode].w = w;
34      edg[newnode].n = u[front];
35      u[front] = newnode;
36      indegree[v]++;
37    } else {
38      edg[e].w = (w < edg[e].w)?w:(edg[e].w);
39    }
40  }
41
42  int n;
43
44  char inq[MAXN];
45  int vis[MAXN];
46  int d[MAXN];
47  int spfa(int s) { /* return 1 if fuhuan exists. */
48    queue<int> q;
49    memset(inq, 0, sizeof(inq));
50    memset(d, -1, sizeof(d));
51    memset(vis, 0, sizeof(vis));
52    d[s] = 0;
53    inq[s] = 1;
54    q.push(s);
55    while(!q.empty()) {
56      int u = q.front();
57      q.pop();
58      printf("proc %d..\n", u);
59      inq[u] = 0;
60      if(vis[u]++ > n)
61        return 1;
62      for(int e = front[u]; e; e = edg[e].n) {
63        int v = edg[e].v, w = edg[e].w;
64        if( -1==d[v] || d[u] + w < d[v] ) {
65          d[v] = d[u] + w;
66          if(!inq[v]) {
67            inq[v] = 1;
68            q.push(v);
69          }
70        }
71      }
72    }
73    return 0;
74  }
```

4.6 Maxflow

```
1   /* Max Flow Problem, by Abreto<m@abreto.net> */
2
3   #include <bits/stdc++.h>
4   using namespace std;
5
6   #define MAXV    (100000)
7   #define MAXE    (1000000)
8   struct edge {
9     static int N;
10    int v, w;
11    edge *n;
12    edge(void):v(0),w(0),n(NULL) {}
13    edge(int vv, int ww, edge *nn):v(vv),w(ww),n(nn) {}
14  };
15  int nE;
16  edge E[MAXE];
17  edge *front[MAXV];
```

```
18 void add_edge(int u, int v, int w) {
19   int ne = ++nE;
20   E[ne] = edge(v, w, u[front]);
21   u[front] = &(E[ne]);
22 }
23 edge *find_edge(int u, int v) {
24   for(edge *e = u[front]; e != NULL; e = e->n)
25     if(e->v == v)
26       return e;
27   return NULL;
28 }
29 void grant_e(int u, int v, int w) {
30   edge *e = find_edge(u, v);
31   if(NULL == e) add_edge(u,v,w);
32   else e->w += w;
33 }
34
35 int vis[MAXV];
36 int path[MAXV];
37 int dfs(int u, int t) {
38   vis[u] = 1;
39   if(u == t) return 1;
40   for(edge *e = u[front]; e != NULL; e = e->n) {
41     int v = e->v;
42     if(!vis[v] && e->w && dfs(v,t)) {
43       path[u] = v;
44       return 1;
45     }
46   }
47   return 0;
48 }
49 int find_path(int s, int t) {
50   memset(vis, 0, sizeof(vis));
51   return dfs(s,t);
52 }
53 int max_flow(int s, int t) {
54   int flow = 0;
55   while(find_path(s,t)) {
56     int i = 0;
57     int minf = find_edge(s,path[s])->w;
58     for(i = path[s]; i != t; i = path[i])
59       minf = min(minf, find_edge(i,path[i])->w);
60     for(i = s; i != t; i = path[i]) {
61       grant_e(i, path[i], -minf);
62       grant_e(path[i], i, minf);
63     }
64     flow += minf;
65   }
66   return flow;
67 }
68
69 /* Dinic */
70 #define N 1000
71 #define INF 100000000
72
73 struct Edge {
74   int from,to,cap,flow;
75   Edge(int u,int v,int c,int f):from(u),to(v),cap(c),flow(f) {}
76 };
77
78 struct Dinic {
79   int n,m,s,t;//结点数，边数（包括反向弧），源点编号，汇点编号
80   vector<Edge>edges;//边表，dges[e]和dges[e^1]互为反向弧
81   vector<int>G[N];//邻接表，G[i][j]表示结点i的第j条边在e数组中的编号
```

```
 82    bool vis[N]; //BFS的使用
 83    int d[N]; //从起点到i的距离
 84    int cur[N]; //当前弧下标
 85
 86    void addedge(int from,int to,int cap) {
 87      edges.push_back(Edge(from,to,cap,0));
 88      edges.push_back(Edge(to,from,0,0));
 89      int  m=edges.size();
 90      G[from].push_back(m-2);
 91      G[to].push_back(m-1);
 92    }
 93
 94    bool bfs() {
 95      memset(vis,0,sizeof(vis));
 96      queue<int>Q;
 97      Q.push(s);
 98      d[s]=0;
 99      vis[s]=1;
100      while(!Q.empty()) {
101        int x=Q.front();
102        Q.pop();
103        for(int i=0; i<G[x].size(); i++) {
104          Edge&e=edges[G[x][i]];
105          if(!vis[e.to]&&e.cap>e.flow) { //只考虑残量网络中的弧
106            vis[e.to]=1;
107            d[e.to]=d[x]+1;
108            Q.push(e.to);
109          }
110        }
111
112      }
113      return vis[t];
114    }
115
116    int dfs(int x,int a) { //x表示当前结点，a表示目前为止的最小残量
117      if(x==t||a==0)return a;//a等于0时及时退出，此时相当于断路了
118      int flow=0,f;
119      for(int&i=cur[x]; i<G[x].size(); i++) { //从上次考虑的弧开始，注意要使用引用，同
           时修改cur[x]
120        Edge&e=edges[G[x][i]];//e是一条边
121        if(d[x]+1==d[e.to]&&(f=dfs(e.to,min(a,e.cap-e.flow)))>0) {
122          e.flow+=f;
123          edges[G[x][i]^1].flow-=f;
124          flow+=f;
125          a-=f;
126          if(!a)break;//a等于0及时退出，当a!=0,说明当前节点还存在另一个曾广路分支。
127
128        }
129      }
130      return flow;
131    }
132
133    int Maxflow(int s,int t) { //主过程
134      this->s=s,this->t=t;
135      int flow=0;
136      while(bfs()) { //不停地用bfs构造分层网络，然后用dfs沿着阻塞流增广
137        memset(cur,0,sizeof(cur));
138        flow+=dfs(s,INF);
139      }
140      return flow;
141    }
142  };
143
144  /* ISAP */
```

```
145 struct Edge {
146   int from,to,cap,flow;
147 };
148 const int maxn=650;
149 const int INF=0x3f3f3f3f;
150 struct ISAP {
151   int n,m,s,t;//结点数，边数（包括反向弧），源点编号，汇点编号
152   vector<Edge>edges;
153   vector<int>G[maxn];
154   bool vis[maxn];
155   int d[maxn];
156   int cur[maxn];
157   int p[maxn];
158   int num[maxn];
159   void AddEdge(int from,int to,int cap) {
160     edges.push_back((Edge) {
161       from,to,cap,0
162     });
163     edges.push_back((Edge) {
164       to,from,0,0
165     });
166     m=edges.size();
167     G[from].push_back(m-2);
168     G[to].push_back(m-1);
169   }
170   bool RevBFS() {
171     memset(vis,0,sizeof(vis));
172     queue<int>Q;
173     Q.push(t);
174     d[t]=0;
175     vis[t]=1;
176     while(!Q.empty()) {
177       int x=Q.front();
178       Q.pop();
179       for(int i=0; i<G[x].size(); i++) {
180         Edge &e =edges[G[x][i]^1];
181         if(!vis[e.from]&&e.cap>e.flow) {
182           vis[e.from]=1;
183           d[e.from]=d[x]+1;
184           Q.push(e.from);
185         }
186       }
187     }
188     return vis[s];
189   }
190   int Augment() {
191     int x=t, a=INF;
192     while(x!=s) {
193       Edge &e = edges[p[x]];
194       a= min(a,e.cap-e.flow);
195       x=edges[p[x]].from;
196     }
197     x=t;
198     while(x!=s) {
199       edges[p[x]].flow+=a;
200       edges[p[x]^1].flow-=a;
201       x=edges[p[x]].from;
202     }
203     return a;
204   }
205   int Maxflow(int s,int t,int n) {
206     this->s=s,this->t=t,this->n=n;
207     int flow=0;
208     RevBFS();
```

```
209     memset(num,0,sizeof(num));
210     for(int i=0; i<n; i++) {
211       num[d[i]]++;
212     }
213     int x=s;
214     memset(cur,0,sizeof(cur));
215     while(d[s]<n) {
216       if(x==t) {
217         flow+=Augment();
218         x=s;
219       }
220       int ok=0;
221       for(int i=cur[x]; i<G[x].size(); i++) {
222         Edge &e =edges[G[x][i]];
223         if(e.cap>e.flow&&d[x]==d[e.to]+1) {
224           ok=1;
225           p[e.to]=G[x][i];
226           cur[x]=i;
227           x=e.to;
228           break;
229         }
230       }
231       if(!ok) {
232         int m=n-1;
233         for(int i=0; i<G[x].size(); i++) {
234           Edge &e =edges[G[x][i]];
235           if(e.cap>e.flow)
236             m=min(m,d[e.to]);
237         }
238         if(--num[d[x]]==0)
239           break;
240         num[d[x]=m+1]++;
241         cur[x]=0;
242         if(x!=s)
243           x=edges[p[x]].from;
244       }
245     }
246     return flow;
247   }
248 };
249 int main() {
250   int n,m,a,b,c,res;
251   while(scanf("%d%d",&m,&n)!=EOF) {
252     ISAP tmp;
253     for(int i=0; i<m; i++) {
254       scanf("%d%d%d",&a,&b,&c);
255       tmp.AddEdge(a,b,c);
256     }
257     res=tmp.Maxflow(1,n,n);
258     printf("%d\n",res);
259   }
260   return 0;
261 }
```

4.7 Strongly Connected Component

```
1 /* Kosaraju */
2 #define MAXN    10010
3 #define MAXM    100010
4 struct edge {
5   int v;
6   edge *n;
7   edge(void):v(0),n(NULL) {}
```

```
 8   edge(int vv, edge *nn):v(vv),n(nn) {}
 9 };
10 int nE;
11 edge E[MAXM<<1];
12 edge *ori[MAXN];
13 edge *inv[MAXN];
14 void add_edge(edge *front[], int u, int v) {
15   int ne = ++nE;
16   E[ne] = edge(v, u[front]);
17   u[front] = &(E[ne]);
18 }
19 void connect(int u, int v) {
20   add_edge(ori, u, v);
21   add_edge(inv, v, u);
22 }
23
24 int vis[MAXN];
25 int vst[MAXN];
26 void first_dfs(int u, int &sig) {
27   vis[u] = 1;
28   for(edge *e = u[ori]; e; e = e->n)
29     if(!vis[e->v])
30       first_dfs(e->v, sig);
31   vst[++sig] = u;
32 }
33 int mark[MAXN];
34 void second_dfs(int u, int sig) {
35   vis[u] = 1;
36   mark[u] = sig;
37   for(edge *e = u[inv]; e; e = e->n)
38     if(!vis[e->v])
39       second_dfs(e->v, sig);
40 }
41
42 int N, M;
43
44 int kosaraju(void) {
45   int i;
46   int sig = 0;
47   for(i = 0; i <= N; ++i) vis[i] = 0;
48   for(i = 1; i <= N; ++i) {
49     if(!vis[i])
50       first_dfs(i, sig);
51   }
52   sig = 1;
53   for(i = 0; i <= N; ++i) vis[i] = 0;
54   for(i = N; i > 0; --i) {
55     if(!vis[vst[i]])
56       second_dfs(vst[i], sig++);
57   }
58   for(i = 1; i <= N; ++i)
59     if(mark[i] != 1)
60       return 0;
61   return 1;
62 }
63
64
65 void clear(void) {
66   nE = 0;
67   for(int i = 0; i <= N; ++i) {
68     ori[i] = inv[i] = NULL;
69   }
70 }
71
```

```
/* Tarjan */
#define MAXN    10010
#define MAXM    100010
struct edge {
  int v;
  edge *n;
  edge(void):v(0),n(NULL) {}
  edge(int vv, edge *nn):v(vv),n(nn) {}
};
typedef edge *ep;
int nE;
edge E[MAXM];
edge *front[MAXN];
void add_edge(int u, int v) {
  int ne = ++nE;
  E[ne] = edge(v, u[front]);
  u[front] = &(E[ne]);
}

int mark[MAXN];
int dfn[MAXN], low[MAXN];
int stk[MAXN];
int stk_top;

void tardfs(int u, int stamp, int &scc) {
  mark[u] = 1;
  dfn[u] = low[u] = stamp;
  stk[stk_top++] = u;
  for(ep e = u[front]; e; e = e->n) {
    if(0 == mark[e->v]) tardfs(e->v, ++stamp, scc);
    if(1 == mark[e->v]) low[u] = min(low[u], low[e->v]);
  }
  if(dfn[u] == low[u]) {
    ++scc;
    do {
      low[stk[stk_top-1]] = scc;
      mark[stk[stk_top-1]] = 2;
    } while(stk[(stk_top--)-1] != u);
  }
}

int tarjan(int n) {
  int scc = 0, lay = 1;
  for(int i = 1; i <= n; ++i)
    if(0 == mark[i])
      tardfs(i, lay, scc);
  return scc;
}

int N, M;

void clear(void) {
  nE = 0;
  for(int i = 0; i <= N; ++i) {
    i[front] = NULL;
    mark[i] = low[i] = 0;
  }
  stk_top = 0;
}

/* Garbow */
#define MAXN    10010
#define MAXM    100010

```

```
136 struct edge {
137   int v;
138   edge *n;
139   edge(void):v(0),n(NULL) {}
140   edge(int vv, edge *nn):v(vv),n(nn) {}
141 };
142 typedef edge *ep;
143
144 int nE;
145 edge E[MAXM];
146 edge *front[MAXN];
147 void add_edge(int u, int v) {
148   int ne = ++nE;
149   E[ne] = edge(v, u[front]);
150   u[front] = &(E[ne]);
151 }
152
153 int stk1[MAXN], stk1t;
154 int stk2[MAXN], stk2t;
155 int low[MAXN], belg[MAXN];
156
157 void garbowdfs(int u, int lay, int &scc) {
158   stk1[++stk1t] = u;
159   stk2[++stk2t] = u;
160   low[u] = ++lay;
161   for(ep e=u[front]; e; e = e->n) {
162     if(!low[e->v]) garbowdfs(e->v, lay, scc);
163     else if (0 == belg[e->v])
164       while(low[stk2[stk2t]] > low[e->v])
165         --stk2t;
166   }
167   if(stk2[stk2t] == u) {
168     stk2t--;
169     scc++;
170     do {
171       belg[stk1[stk1t]] = scc;
172     } while(stk1[stk1t--] != u);
173   }
174 }
175
176 int grabow(int n) {
177   int i;
178   int scc = 0, lay = 0;
179   for(i = 0; i <= n; ++i) {
180     belg[i] = low[i] = 0;
181   }
182   for(i = 1; i <= n; ++i)
183     if(0 == low[i])
184       garbowdfs(i, lay, scc);
185   return scc;
186 }
187
188 int N, M;
189
190 void clear(void) {
191   nE = 0;
192   for(int i = 0; i <= N; ++i) {
193     front[i] = NULL;
194   }
195 }
```