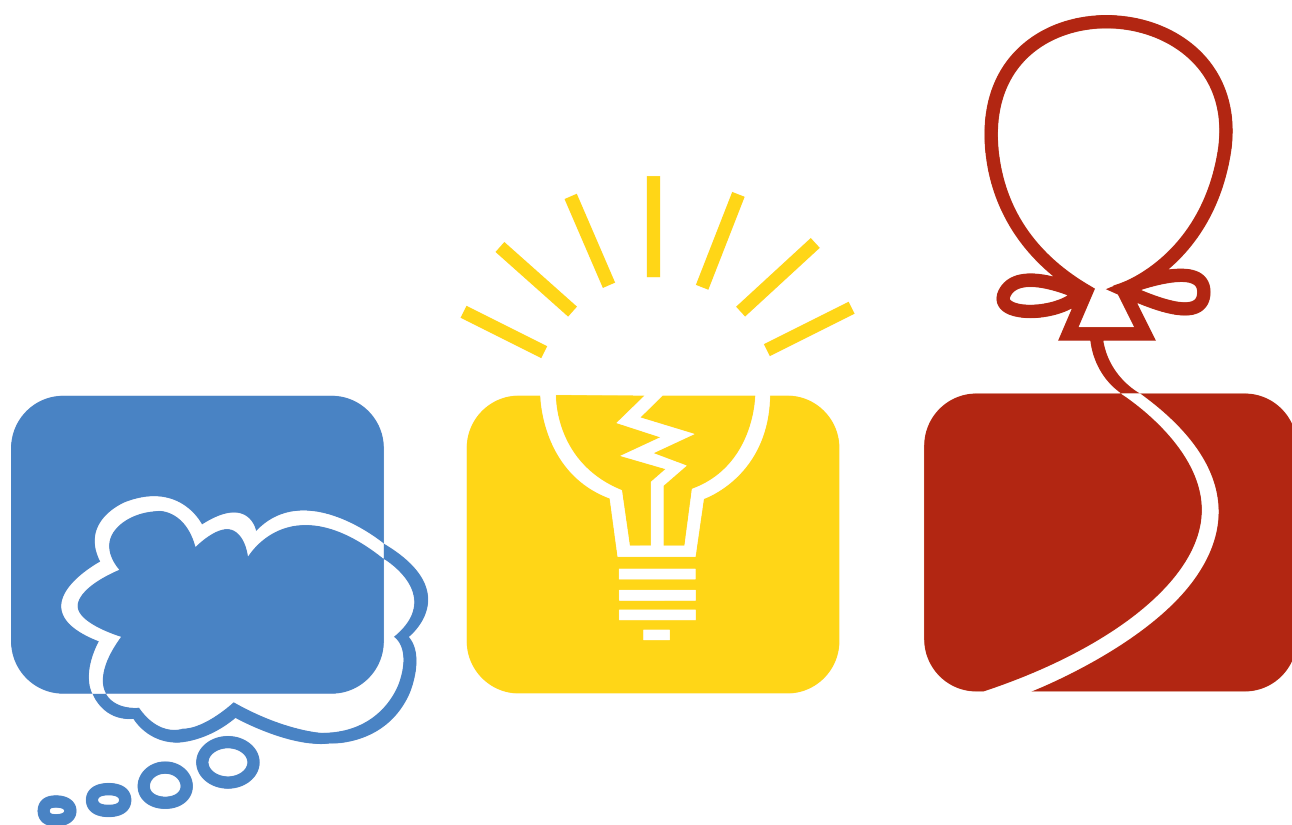


---

# ACM TEMPLATE



**acm** International Collegiate  
Programming Contest

UESTC\_Duiming

Last build at October 26, 2017

# Contents

<b>1</b>	<b>Datastructure</b>	<b>3</b>
1.1	Fenwick . . . . .	3
1.2	BST in pb_ds . . . . .	3
1.3	Segment Tree . . . . .	3
<b>2</b>	<b>Dynamic Programming</b>	<b>5</b>
2.1	LIS $O(n \log n)$ . . . . .	5
2.2	LCS $O(n \log n)$ . . . . .	5
2.3	Sparse-Table . . . . .	6
2.4	Improved by quadrilateral inequality . . . . .	6
2.5	Improved by Slope . . . . .	7
<b>3</b>	<b>Geometry</b>	<b>8</b>
3.1	2D . . . . .	8
3.1.1	Point . . . . .	8
3.1.2	Convex hull . . . . .	9
3.1.3	Intersect Area . . . . .	10
3.1.4	Universe . . . . .	12
<b>4</b>	<b>Graph</b>	<b>19</b>
4.1	Tree . . . . .	19
4.1.1	Universe . . . . .	19
4.1.2	Point Divide and Conquer . . . . .	20
4.2	2-SAT . . . . .	26
4.3	Cut Edge and Point . . . . .	27
4.4	Euler Path . . . . .	28
4.5	Shortest Path . . . . .	29
4.5.1	Dijkstra . . . . .	29
4.5.2	Shortest Path Fast Algorithm . . . . .	30
4.6	Maxflow . . . . .	31
4.7	Strongly Connected Component . . . . .	35
<b>5</b>	<b>Math</b>	<b>38</b>
5.1	Euler Function . . . . .	38
5.2	Chinese Remainder Theorem . . . . .	39
5.3	FFT . . . . .	39
5.4	Number Theory Inverse . . . . .	43
5.5	Linear Programming . . . . .	43
<b>6</b>	<b>String</b>	<b>46</b>
6.1	Hash . . . . .	46
6.2	KMP . . . . .	46
6.3	Suffix Array . . . . .	47
<b>7</b>	<b>Tool</b>	<b>49</b>
7.1	IO plug-in . . . . .	49
7.2	Matrix (including quickpow) . . . . .	50
7.3	Double Class . . . . .	51
7.4	Complex Class . . . . .	52
<b>8</b>	<b>Appendix</b>	<b>53</b>

<b>9</b>	<b>Graph Algorithms</b>	<b>53</b>
9.1	DFS . . . . .	54
9.1.1	DFS tree . . . . .	54
9.1.2	Starting time, finishing time . . . . .	54
9.1.3	Finding cut edges . . . . .	55
9.1.4	Finding cut vertices . . . . .	55
9.1.5	Finding Eulerian tours . . . . .	55
9.2	BFS . . . . .	56
9.2.1	BFS tree . . . . .	56
9.3	SCC . . . . .	56
9.4	Shortest path . . . . .	56
9.4.1	Dijkstra . . . . .	57
9.4.2	Floyd-Warshall . . . . .	58
9.4.3	Bellman-Ford . . . . .	58
9.4.4	SPFA . . . . .	59
9.5	MST . . . . .	59
9.5.1	Kruskal . . . . .	59
9.5.2	Prim . . . . .	60
9.6	Maximum Flow . . . . .	61
9.6.1	Dinic's algorithm . . . . .	63
9.6.2	Maximum Matching in bipartite graphs . . . . .	63
9.7	Trees . . . . .	64
9.7.1	Partial sum on trees . . . . .	64
9.7.2	DSU on trees . . . . .	64
9.7.3	LCA . . . . .	65

# 1 Datastructure

## 1.1 Fenwick

```

1  /* Fenwick Tree (Binary Indexed Tree), by Abreto <m@abreto.net>. */
2
3  #define MAXN 1000001
4  #define LOWBIT(x) ((x)&(-(x)))
5
6  int N;
7  int fen[MAXN];
8
9  void update(int i, int dx) {
10     while(i <= N) {
11         fen[i] += dx;
12         i += LOWBIT(i);
13     }
14 }
15
16 int sum(int i) {
17     int s = 0;
18     while(i > 0) {
19         s += fen[i];
20         i -= LOWBIT(i);
21     }
22     return s;
23 }

```

## 1.2 BST in pb\_ds

```

1  /* Red-Black tree via pb_ds. */
2  #include<bits/stdc++.h>
3  #include<ext/pb_ds/assoc_container.hpp>
4  #include<ext/pb_ds/tree_policy.hpp>
5  using namespace __gnu_pbds;
6  using namespace std;
7  template <typename T>
8  using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
9      tree_order_statistics_node_update>;
10
11 int main() {
12     ordered_set<int> s;
13     s.insert(1);
14     s.insert(3);
15     cout << s.order_of_key(2) << endl; // the number of elements in the s less than 2
16     cout << *s.find_by_order(0) << endl; // print the 0-th smallest number in s(0-based)
17 }

```

## 1.3 Segment Tree

```

1  /* Segment tree (Interval tree, range tree), by Abreto <m@abreto.net>. */
2
3  template <int STMAX = 10000000>
4  struct segment_tree {
5      struct node_t {
6          static inline node_t merge(node_t n1, node_t n2) {
7              node_t ans;
8              ans.l = n1.l;
9              ans.r = n2.r;
10             /* merge n1 and n2 to ans. */

```

```

11     return ans;
12 }
13
14 /* Data field */
15 int l,r;
16 } nodes[(STMAX+1)<<2];
17
18 struct lazy_t {
19     int marked; /* Optional */
20     /* lazy mark. */
21
22     lazy_t(void) {
23         clear();
24     }
25     void clear(void) {
26         marked=0;
27     }
28 } marks[(STMAX+1)<<2];
29
30 inline void maintain_leaf(int o, int idx) {
31     nodes[o].l = nodes[o].r = idx;
32     /* Operations to single elements ... */
33 }
34 inline void maintain(int o) {
35     nodes[o] = node_t::merge(nodes[o<<1], nodes[o<<1|1]);
36 }
37
38 /* Usage: build(1,1,n); */
39 void build(int o, int l, int r) { /* [l,r] */
40     if( r <= l ) {
41         maintain_leaf(o, l);
42     } else {
43         int mid = l+r>>1;
44         build(o<<1, l, mid);
45         build(o<<1|1, mid+1, r);
46         maintain(o);
47     }
48 }
49
50 /* Modify all elements in [l,r] */
51 void mark(lazy_t act, int o) {
52     /* do something .. */
53     marks[o].marked = 1;
54 }
55
56 /* Pass cached updates. */
57 void pushdown(int o) {
58     if( marks[o].marked ) {
59         mark(marks[o], o<<1);
60         mark(marks[o], o<<1|1);
61         marks[o].clear();
62     }
63 }
64
65 /* Do act on all elements in [L,R] */
66 void upd(int L, int R, lazy_t act, int o, int l, int r) {
67     if( L <= l && r <= R ) {
68         mark(act, o);
69     } else if (L <= R) {
70         int mid = (l+r)>>1;
71         pushdown(o);
72         if( L <= mid ) upd(L, R, act, o<<1, l, mid);
73         if( R > mid ) upd(L, R, act, o<<1|1, mid+1, r);
74         maintain(o);

```

```

75     }
76 }
77
78 node_t qry(int L, int R, int o, int l, int r) {
79     if(L <= l && r <= R)
80         return nodes[o];
81     else if (L <= R) {
82         int mid = (l+r)>>1;
83         pushdown(o);
84         if(R <= mid) return qry(L,R,o<<1,l,mid);
85         if(L > mid) return qry(L,R,o<<1|1,mid+1,r);
86         return node_t::merge(qry(L,R,o<<1,l,mid),qry(L,R,o<<1|1,mid+1,r));
87     }
88 }
89
90 int N;
91
92 segment_tree(void):N(STMAX) {}
93 segment_tree(int n):N(n) {}
94 void build(int n) {
95     N = n;
96     build(1,1,N);
97 }
98 void update(int L, int R, lazy_t act) {
99     upd(L,R,act,1,1,N);
100 }
101 node_t query(int L, int R) {
102     return qry(L,R,1,1,N);
103 }
104 };

```

## 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 $O(n \log n)$

```

1
2 /**
3 总的来说，就是把LCS转化成LIS，然后用LIS的NlogN算法来求解。
4 实现如下：（引用）
5 假设有两个序列 s1[ 1~6 ] = { a, b, c, a, d, c }, s2[ 1~7 ] = { c, a, b, e, d, a, b }
6 记录s1中每个元素在s2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：
7 loc(a) = { 6, 2 }, loc( b ) = { 7, 3 }, loc( c ) = { 1 }, loc( d ) = { 5 }。 // 倒着
   扫一遍s2即可把位置扔进vector

```

```

8 将s1中每个元素的位置按s1中元素的顺序排列成一个序列s3 = { 6, 2, 7, 3, 1, 6, 2, 5, 1 }
9 在对s3求LIS得到的值即为求LCS的答案。
10 */

```

## 2.3 Sparse-Table

```

1  /* RMQ with Sparse Table, by Abreto <m@abreto.net>. */
2
3  int min(int a, int b) {
4      return (a<b)?a:b;
5  }
6
7  #define MAXN    100001
8  #define MAXLOG   32
9
10 int N;
11 int A[MAXN];    /* indexed from 0. */
12 int st[MAXN][MAXLOG];
13
14 void st_init() {
15     int i = 0, j = 0, t = 0;
16     for(i = 0; i < N; ++i) st[i][0] = A[i];
17     for(j = 1; (t=(1<<j)) <= N; ++j)
18         for(i = 0; (i+t-1) < N; ++i)
19             st[i][j] = min(st[i][j-1], st[i+(t>>1)][j-1]);
20     /* st(i,j) = min(st(i,j-1), st(i+2^(j-1),j-1)). */
21 }
22
23 int st_query(int l, int r) {
24     int k = 0;
25     while((1<<(k+1)) <= (r-l+1)) k++;
26     return min(st[l][k], st[r-(1<<k)+1][k]);
27 }

```

## 2.4 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)  // i增j减
8  * 或
9  *      s(i,j-1) <= s(i,j) <= s(i+1,j)  // 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

```

```

25 int wa(void) {
26     int i, j, k;
27     for(i = 1; i <= n; ++i) {
28         scanf("%d", v+i);
29         s[i] = v[i] + s[i-1];
30     }
31     for(i = 1; i <= n; ++i) {
32         w[i][i] = 0;
33         for(j = i+1; j <= n; ++j)
34             w[i][j] = w[i][j-1] + v[j] * (s[j-1] - s[i-1]);
35     }
36     /* doing dp */
37     for(i = 1; i <= n; ++i) {
38         dp[i][0] = w[1][i];
39         c[i][0] = 1;
40         c[i][i] = i-1;
41         for(j = i-1; j > 0; j--) {
42             dp[i][j] = inf;
43             for(k = c[i-1][j]; k <= c[i][j+1]; ++k)
44                 if(dp[k][j-1] + w[k+1][i] <= dp[i][j]) {
45                     dp[i][j] = dp[k][j-1] + w[k+1][i];
46                     c[i][j] = k;
47                 }
48         }
49     }
50     /* dp done */
51     return dp[n][m];
52 }
53
54 int main(void) {
55     while(EOF != scanf("%d%d", &n, &m) && n && m) {
56         printf("%d\n", wa());
57     }
58     return 0;
59 }

```

## 2.5 Improved by Slope

```

1  /* type 1: */
2  /* bzoj 1010 */
3  #include <bits/stdc++.h>
4
5  using namespace std;
6  typedef long double ll;
7  #define MAXN    50050
8  #define eps     (1e-8)
9
10 int N;
11 ll L;
12 ll S[MAXN];
13 ll f[MAXN];
14 ll dp[MAXN];
15
16 inline ll k(int j) {
17     return (-2.0) * (f[j] + L);
18 }
19 inline ll b(int j) {
20     return dp[j] + f[j]*f[j] + 2ll*f[j]*L;
21 }
22 inline ll g(int j, int i) {
23     return k(j) * f[i] + b(j);
24 }
25

```



```

26  /* check if l1 & l3 <= l2 */
27  inline int check(int l1, int l2, int l3) {
28      /*ll left = b(l3)*k(l1)+b(l1)*k(l2)+b(l2)*k(l3);
29      ll right = b(l1)*k(l3)+b(l3)*k(l2)+b(l2)*k(l1);*/
30      ll left = b(l3)*k(l1)-b(l1)*k(l3);
31      ll right = k(l2)*(b(l3)-b(l1))+b(l2)*(k(l1)-k(l3));
32      return (left <= right);
33  }
34
35  int Q[MAXN], ql, qr;
36
37  int main(void) {
38      int i;
39      scanf("%d%Lf", &N, &L);
40      L += 1.0;
41      for(i = 1; i <= N; ++i) {
42          scanf("%Lf", S+i);
43          S[i] += S[i-1];
44          f[i] = S[i] + (double)i;
45      }
46      Q[qr++] = 0;
47      for(i = 1; i <= N; ++i) {
48          /* <!-- STARED */
49          for(; ql+1 < qr && g(Q[ql],i) >= g(Q[ql+1],i); ql++);
50          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]);
51          for(; ql+1 < qr && check(Q[qr-2], Q[qr-1], i); qr--);
52          Q[qr++] = i;
53          /* --> */
54      }
55      printf("%lld\n", (long long int)round(dp[N]));
56      return 0;
57  }

```

### 3 Geometry

#### 3.1 2D

##### 3.1.1 Point

```

1  /* 2D Point Class, by Abreto<m@abreto.net> */
2  #include <cmath>
3
4  using namespace std;
5
6  #define EPS (1e-8)
7  bool fe(double a, double b) {
8      return ((a-b>=-EPS)&&(a-b<=EPS));
9  }
10 bool fl(double a, double b) {
11     return (a-b<=-EPS);
12 }
13 bool fle(double a, double b) {
14     return (a-b<=EPS);
15 }
16
17 template <typename T>
18 struct point {
19     T x,y;
20     point(void):x(T()),y(T()) {}
21     point(T xx, T yy):x(xx),y(yy) {}
22     T& operator[](int i) {

```

```

23     if(0==i)return x;
24     else return y;
25 }
26 inline point operator-(void) const {
27     return point(-x,-y);
28 }
29 inline point operator+(const point& b) const {
30     return point(x+b.x,y+b.y);
31 }
32 inline point operator-(const point& b) const {
33     return point(x-b.x,y-b.y);
34 }
35 inline T operator*(const point& b) const {
36     return ((x)*(b.x))+((y)*(b.y)); /* inner product */
37 }
38 inline T operator^(const point& b) const {
39     return ((x)*(b.y))-((b.x)*(y)); /* outter product */
40 }
41 inline point& operator+=(const point& b) {
42     point tmp=(*this)+b;
43     (*this)=tmp;
44     return (*this);
45 }
46 inline point& operator-=(const point& b) {
47     point tmp=(*this)-b;
48     (*this)=tmp;
49     return (*this);
50 }
51 inline bool operator==(const point& b) const {
52     return (x==b.x)&&(y==b.y);
53 }
54 inline bool operator!=(const point& b) const {
55     return !((*this)==b);
56 }
57 };
58
59 #define vec point

```

### 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 100000000000000
8 | #define M 8
9 | #define LL long long
10 | #define eps 1e-12
11 | #define PI acos(-1.0)
12 | using namespace std;
13 | struct node {
14 |     double x,y;
15 |     node() {}
16 |     node(double xx,double yy) {
17 |         x=xx;
18 |         y=yy;
19 |     }
20 |     node operator -(node s) {
21 |         return node(x-s.x,y-s.y);
22 |     }
23 |     node operator +(node s) {
24 |         return node(x+s.x,y+s.y);
25 |     }
26 |     double operator *(node s) {
27 |         return x*s.x+y*s.y;
28 |     }
29 |     double operator ^(node s) {
30 |         return x*s.y-y*s.x;
31 |     }
32 | };
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) { //点乘
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         } else {
89             double angle1=acos(h/dis(a,b));
90             double angle2=acos(h/r);
91             double angle3=acos(h/dis(a,c))-angle2;
92             return 0.5*r*dis(a,b)*sin(angle1+angle2)+0.5*r*r*angle3;
93         }
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(0,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 }

```

### 3.1.4 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.y*p);
19 }
20 Vector operator / (Vector A, double p) {
21     return Vector(A.x/p, A.y/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) == 0;
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  $P = P_0 + t*v$ ;
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: 设平面图形的定点数, 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 getLineCircleInterseccion(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(n^2);
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 }

```

## 4 Graph

### 4.1 Tree

#### 4.1.1 Universe

```

1  |
2  | /* find root(重心) */
3  |
4  | void findroot(int u, int fa) {
5  |     int i;
6  |     size[u] = 1;
7  |     f[u] = 0;
8  |     for (i = last[u]; i; i = e[i][2]) {
9  |         if (!vis[e[i][0]] && e[i][0] != fa) {
10 |             findroot(e[i][0], u);
11 |             size[u] += size[e[i][0]];
12 |             if (f[u] < size[e[i][0]])
13 |                 f[u] = size[e[i][0]];
14 |         }
15 |     }
16 |     if (f[u] < ALL - size[u])
17 |         f[u] = ALL - size[u];
18 |     if (f[u] < f[root]) root = u;
19 | }
20 |
21 | /* —— da —— */
22 |
23 | int dep[MAXN+1];
24 | int ancestor[MAXN+1][MAXLGN];
25 | int minw[MAXN+1][MAXLGN];
26 |
27 | void dfs(int u, int fa) {
28 |     ancestor[u][0] = fa;
29 |     dep[u] = dep[fa] + 1;
30 |     for(int e = u[front]; e; e = E[e].n) {
31 |         int v = E[e].v, w = E[e].w;
32 |         if(v != fa) {
33 |             minw[v][0] = w;
34 |             dfs(v, u);
35 |         }
36 |     }
37 | }
38 |
39 | void init_system(void) {
40 |     int i = 0, w = 0;
41 |     int t = 0;
42 |     dep[0] = -1;
43 |     dfs(1,0);
44 |     for(w = 1; (t=(1<<w)) < N; ++w)
45 |         for(i = 1; i <= N; ++i) if( dep[i] >= t ) {
46 |             ancestor[i][w] = ancestor[ancestor[i][w-1]][w-1];
47 |             minw[i][w] = min(minw[i][w-1], minw[ancestor[i][w-1]][w-1]);
48 |         }
49 | }
50 |
51 | int query(int a, int b) {
52 |     if(dep[a] < dep[b]) return query(b,a);
53 |     else { /* now dep[s] > dep[t] */
54 |         int i = 0;
55 |         int maxbit = MAXLGN-1;
56 |         int ret = INF;
57 |         //while((1<<maxbit) <= dep[a]) maxbit++;
58 |         /* first up a to same dep with b. */

```

```

59     for(i = maxbit; i >= 0; i--)
60         if(dep[a] - (1<<i) >= dep[b]) {
61             ret = min(ret, minw[a][i]);
62             a = ancestor[a][i];
63         }
64     if(a == b) return ret;
65     for(i = maxbit; i >= 0; i--)
66         if(dep[a] - (1<<i) >= 0 && ancestor[a][i] != ancestor[b][i]) {
67             ret = min(ret, min(minw[a][i], minw[b][i]));
68             a = ancestor[a][i];
69             b = ancestor[b][i];
70         }
71     ret = min(ret, min(minw[a][0], minw[b][0]));
72     return ret;
73 }
74 }

```

#### 4.1.2 Point Divide and Conquer

##### Version 1

```

1  /* Tree::Point divide and conquer, by Abreto<m@abreto.net>. */
2  #include <bits/stdc++.h>
3
4  using namespace std;
5  typedef long long int ll;
6
7  #define MAXN      (100001)
8  #define MAXV      (MAXN+1)
9  #define MAXE      (MAXN<<1)
10 struct edge {
11     int v;
12     edge *n;
13     edge(void):v(0),n(NULL) {}
14     edge(int vv,edge *nn):v(vv),n(nn) {}
15 };
16 int nE;
17 edge E[MAXE];
18 edge *front[MAXV];
19 int label[MAXV]; /* 0 for '(', 1 for ')' */
20 void add_edge(int u, int v) {
21     int ne = ++nE;
22     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]]++;
87     }
88     {
89         /* in to root */
90         in[0][u] = in[0][fa];
91         in[1][u] = in[1][fa];
92         if(0 == label[u]) { /* meet '(' */
93             if(in[1][u]) in[1][u]--;
94             else in[0][u]++;
95         } else { /* meet ')' */
96             in[1][u]++;
97         }
98         if(0 == in[1][u])
99             cntin[in[0][u]]++;
100     }
101     /* do something */
102     for(edge *e = u[front]; e; e = e->n) {
103         int v = e->v;
104         if((!del[v]) && (v != fa)) {
105             dfs(v, u);
106         }

```

```

107     }
108 }
109 inline void init_maxdep(void) {
110     maxdep = 0;
111 }
112 inline void update_maxdep(int u) {
113     dep[u] = 1;
114     if(dep[u] > maxdep) maxdep = dep[u];
115     for(edge *e = u[front]; e; e = e->n)
116         if((!del[e->v]))
117             getdeep(e->v, u);
118 }
119 inline void clear(void) {
120     for(int i = 0; i <= maxdep+1; ++i)
121         cntin[i] = cntout[i] = 0;
122 }
123 inline void work(int u) {
124     in[0][u] = in[1][u] = out[0][u] = out[1][u] = 0;
125     in[label[u]][u] = out[label[u]][u] = 1;
126     if(out[0][u] == 0) cntout[out[1][u]]++;
127     if(0 == in[1][u]) cntin[in[0][u]]++;
128     /* update in and out if neccessary */
129     for(edge *e = u[front]; e; e = e->n)
130         if(!del[e->v])
131             dfs(e->v, u);
132 }
133 };
134
135 ll count(int u, int p) {
136     ll ret = 0;
137     workspace::init_maxdep();
138     workspace::update_maxdep(u);
139     workspace::clear();
140     if(-1 == p) {
141         for(edge *e = u[front]; e; e = e->n)
142             if((!(del[e->v])))
143                 workspace::work(e->v);
144         p = label[u];
145         /* single end */
146         if(0 == p) ret = workspace::cntout[1];
147         else ret = workspace::cntin[1];
148     } else {
149         workspace::work(u);
150     }
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         }
80     }
81 }
82 inline void clear(void) {
83     for(int i = 1; i <= all_kind; ++i)
84         cnt[i] = 0;
85 }
86 inline void work(int u) {
87     dp[u] = label[u];
88     cnt[dp[u]] ++;
89     for(edge *e = u[front]; e; e = e->n)
90         if((del[e->v] != ndel))
91             dfs(e->v, u);
92 }
93 inline void show(void) {
94     for(int i = 0; i <= all_kind; ++i)
95         printf("cnt[%d]u=%lld\n", i, cnt[i]);
96     for(int i = 1; i <= n; ++i)
97         printf("dp[%d]u=%d\n", i, dp[i]);
98 }
99 };

```

```

100
101
102 ll count(int u, int p) {
103     ll ret = 0;
104     workspace::clear();
105     //printf("%d,%d  :\n", u, p);
106     if(-1 == p) {
107         for(edge *e = u[front]; e; e = e->n)
108             if(((del[e->v]) != ndel))
109                 workspace::work(e->v);
110         p = label[u];
111         /* single end */
112         for(int i = 1; i <= all_kind; i++)
113             if(all_kind == (i|p))
114                 ret += (workspace::cnt[i]<<1);
115     } else {
116         workspace::work(u);
117     }
118     //workspace::show();
119     for(int i = 1; i <= all_kind; ++i)
120         if( workspace::cnt[i] > 0 )
121             for(int j = 1; j <= all_kind; ++j)
122                 if(all_kind == (i|p|j))
123                     ret += workspace::cnt[i] * workspace::cnt[j];
124     //printf("%lld\n", ret);
125     return ret;
126 }
127
128 void handle(int u) {
129     //printf("proccessing %d\n", u);
130     del[u] = ndel; /* delete current root. */
131     ans += count(u, -1);
132     /* do something */
133     for(edge *e = u[front]; e; e = e->n) {
134         int v = e->v;
135         if(del[v] != ndel) {
136             ans -= count(v, label[u]);
137             /* do something */
138             int r = findroot::find(v, findroot::size[v]);
139             handle(r);
140         }
141     }
142 }
143
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 }

```

## 4.2 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;
36     for(int i = 0; i < n*2; ++i) {
37         i[front] = NULL;
38         i[mark] = 0;

```

```

39     }
40     nE = 0;
41 }
42
43 int dfs(int x) {
44     if(mark[x^1]) return 0;
45     if(mark[x]) return 1;
46     mark[x] = 1;
47     S[c++] = x;
48     for(ep e = x[front]; e; e = e->n)
49         if(!dfs(e->v)) return 0;
50     return 1;
51 }
52
53 int solve(void) {
54     for(int i = 0; i < n*2; i += 2)
55         if(!mark[i] && !mark[i+1]) {
56             c = 0;
57             if(!dfs(i)) {
58                 while(c > 0) mark[S[--c]] = 0;
59                 if(!dfs(i+1)) return 0;
60             }
61         }
62     return 1;
63 }
64 }

```

### 4.3 Cut Edge and Point

```

1  /**
2  Finding cut edges
3  The code below works properly because the lemma above (first lemma):
4  h[root] = 0
5  par[v] = -1
6  dfs (v):
7      d[v] = h[v]
8      color[v] = gray
9      for u in adj[v]:
10         if color[u] == white
11             then par[u] = v and dfs(u) and d[v] = min(d[v], d[u])
12             if d[u] > h[v]
13                 then the edge v-u is a cut edge
14         else if u != par[v])
15             then d[v] = min(d[v], h[u])
16     color[v] = black
17 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).
18
19 Finding cut vertices
20 The code below works properly because the lemma above (first lemma):
21 h[root] = 0
22 par[v] = -1
23 dfs (v):
24     d[v] = h[v]
25     color[v] = gray
26     for u in adj[v]:
27         if color[u] == white
28             then par[u] = v and dfs(u) and d[v] = min(d[v], d[u])
29             if d[u] >= h[v] and (v != root or number_of_children(v) > 1)
30                 then the edge v is a cut vertex
31         else if u != par[v])
32             then d[v] = min(d[v], h[u])

```

```

33     color[v] = black
34 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).
35 ***/

```

#### 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 Dijkstra, 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
26 int d[MAXV];
27 int vis[MAXN];
28 int solid[MAXV];
29
30 int dijkstra(int s, int t) {
31     int v = s[front];
32     spii q;
33     q.insert(make_pair(0, s));
34     while(!q.empty()) {
35         auto it = q.begin();
36         int u = it->second;
37         int v = u[front];
38         q.erase(it);
39         solid[u] = 1;
40         if(u == t) break;

```

```

41     while(v) {
42         int w = edg[v].id;
43         if(!solid[w]) {
44             if( (0==d[w]) || (d[u] + 1 < d[w]) ) {
45                 q.erase(make_pair(d[w],w));
46                 d[w] = d[u] + 1;
47                 q.insert(make_pair(d[w],w));
48             }
49         }
50         v = edg[v].nxt;
51     }
52 }
53 return d[t];
54 }

```

#### 4.5.2 Shortest Path Fast Algorithm

```

1  /* Shortest Path Fast Algorithm, by Abreto<m@abreto.net>. */
2  #include <cstdio>
3  #include <cstring>
4  #include <queue>
5  #include <utility>
6
7  using namespace std;
8
9  #define MAXN    128
10
11 struct edge {
12     int v;
13     int w;
14     int n;
15 };
16 edge edg[MAXN<<1];
17 int nedg;
18 int indegree[MAXN];
19 int front[MAXN];
20 int find_edge(int u, int v) {
21     int e = u[front];
22     while(e) {
23         if(edg[e].v == v) return e;
24         e = edg[e].n;
25     }
26     return 0;
27 }
28 void add_edge(int u, int v, int w) {
29     int e = find_edge(u,v);
30     if(0==e) {
31         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      (1000000)
7  #define MAXE      (10000000)
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 1000000000
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
72 /* Tarjan */
73 #define MAXN    10010
74 #define MAXM    100010
75 struct edge {
76     int v;
77     edge *n;
78     edge(void):v(0),n(NULL) {}
79     edge(int vv, edge *nn):v(vv),n(nn) {}
80 };
81 typedef edge *ep;
82 int nE;
83 edge E[MAXM];
84 edge *front[MAXN];
85 void add_edge(int u, int v) {
86     int ne = ++nE;

```

```

87     E[ne] = edge(v, u[front]);
88     u[front] = &(E[ne]);
89 }
90
91 int mark[MAXN];
92 int dfn[MAXN], low[MAXN];
93 int stk[MAXN];
94 int stk_top;
95
96 void tardfs(int u, int stamp, int &scc) {
97     mark[u] = 1;
98     dfn[u] = low[u] = stamp;
99     stk[stk_top++] = u;
100    for(ep e = u[front]; e; e = e->n) {
101        if(0 == mark[e->v]) tardfs(e->v, ++stamp, scc);
102        if(1 == mark[e->v]) low[u] = min(low[u], low[e->v]);
103    }
104    if(dfn[u] == low[u]) {
105        ++scc;
106        do {
107            low[stk[stk_top-1]] = scc;
108            mark[stk[stk_top-1]] = 2;
109        } while(stk[(stk_top--)-1] != u);
110    }
111 }
112
113 int tarjan(int n) {
114     int scc = 0, lay = 1;
115     for(int i = 1; i <= n; ++i)
116         if(0 == mark[i])
117             tardfs(i, lay, scc);
118     return scc;
119 }
120
121 int N, M;
122
123 void clear(void) {
124     nE = 0;
125     for(int i = 0; i <= N; ++i) {
126         i[front] = NULL;
127         mark[i] = low[i] = 0;
128     }
129     stk_top = 0;
130 }
131
132 /* Garbow */
133 #define MAXN    10010
134 #define MAXM    100010
135
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 }

```

## 5 Math

### 5.1 Euler Function

```

1  /* Euler function phi(x), by Abreto<m@abreto.net>. */
2
3  #define MAXX    3000000
4
5  int phi[MAXX];
6  void get_euler(void) {
7      int i = 0, j = 0;
8      phi[1] = 1;
9      for(i = 2; i < MAXX; ++i)
10         if(!phi[i])
11             for(j = i; j < MAXX; j += i) {
12                 if(!phi[j]) phi[j] = j;

```

```

13     phi[j] = phi[j]/i * (i-1);
14 }
15 }

```

## 5.2 Chinese Remainder Theorem

$$x \equiv a_i \pmod{m_i}$$

```

1  /* Chinese Remainder Theorem, by Abreto<m@abreto.net>. */
2  #include "euler.c"
3
4  #define MAXN    64
5
6  typedef long long int ll;
7
8  ll quickpow(ll a, ll b, ll mod) {
9      ll ret = 1, base = a;
10     while(b > 0) {
11         if(b & 1) ret = (ret * base) % mod;
12         base = (base * base) % mod;
13         b >>= 1;
14     }
15     return ret;
16 }
17
18 ll N;
19 /* x = a[i] (mod m[i]) */
20 ll a[MAXN], m[MAXN]; /* a and m is indexed from 0. */
21 ll x, M;
22 /* x: only solution (mod M) */
23
24 void naive_crt(void) {
25     int i = 0;
26     ll Mi[MAXN], nMi[MAXN];
27     ll t[MAXN];
28
29     M = 1;
30     for(i = 0; i < N; ++i)
31         M *= m[i];
32     for(i = 0; i < N; ++i)
33         Mi[i] = M / m[i];
34     get_euler();
35     for(i = 0; i < N; ++i)
36         nMi[i] = quickpow(Mi[i], phi[a[i]]-1, a[i]);
37     for(i = 0; i < N; ++i)
38         t[i] = ((a[i] * Mi[i]) % M) * nMi[i] % M;
39     for(i = 0; i < N; ++i)
40         x = (x + t[i]) % M;
41 }

```

## 5.3 FFT

```

1  #include <cmath>
2  using namespace std;
3  namespace fft {
4      #define eps (1e-9)
5      template < typename T = double >
6      struct dbl {
7          T x;
8          dbl(void):x(0.0) {}

```



```

9   template <typename U>
10  dbl(U a):x((T)a) {}
11  inline char sgn(void) {
12      return ((x>=eps)&&(x<=eps))?(0):((x>eps)?(1):(-1));
13  }
14  inline T tabs(void) {
15      return ((x>=eps)&&(x<=eps))?(0.0):((x>eps)?(x):(-x));
16  }
17  inline dbl abs(void) {
18      return dbl(tabs());
19  }
20  template <typename U> inline dbl &operator=(const U b) {
21      x=(T)b;
22      return (*this);
23  }
24  inline T *operator&(void) {
25      return &x;
26  }
27  inline dbl operator-(void) const {
28      return dbl(-x);
29  }
30  inline dbl operator+(const dbl &b) const {
31      return dbl(x+b.x);
32  }
33  inline dbl operator-(const dbl &b) const {
34      return dbl(x-b.x);
35  }
36  inline dbl operator*(const dbl &b) const {
37      return dbl(x*b.x);
38  }
39  inline dbl operator/(const dbl &b) const {
40      return dbl(x/b.x);
41  }
42  template <typename U> inline dbl operator^(const U &b) const {
43      T ret=1.0,base=x;
44      while(b) {
45          if(b&1)ret*=base;
46          base*=base;
47          b>>=1;
48      }
49      return dbl(ret);
50  }
51  inline dbl operator+=(const dbl &b) {
52      return dbl(x+=b.x);
53  }
54  inline dbl operator-=(const dbl &b) {
55      return dbl(x-=b.x);
56  }
57  inline dbl operator*=(const dbl &b) {
58      return dbl(x*=b.x);
59  }
60  inline dbl operator/=(const dbl &b) {
61      return dbl(x/=b.x);
62  }
63  template <typename U> inline dbl operator^=(const U &b) {
64      dbl tmp=(*this)^b;
65      *this=tmp;
66      return tmp;
67  }
68  inline bool operator==(const dbl &b) const {
69      return (0 == ((*this)-b).sgn());
70  }
71  inline bool operator!=(const dbl &b) const {
72      return (0 != ((*this)-b).sgn());

```

```

73     }
74     inline bool operator<(const dbl &b) const {
75         return (-1 == ((*this)-b).sgn());
76     }
77     inline bool operator<=(const dbl &b) const {
78         return (((*this)==b) || ((*this)<b));
79     }
80     inline bool operator>(const dbl &b) const {
81         return (b < (*this));
82     }
83     inline bool operator>=(const dbl &b) const {
84         return (((*this)==b) || ((*this)>b));
85     }
86     template <typename U> inline operator U() const {
87         return (U)x;
88     }
89     inline char operator[](unsigned n) {
90         if(n >= 0) {
91             long long int ret=x;
92             while(n--) {
93                 ret/=10;
94             }
95             return (ret%10);
96         } else {
97             T ret=x;
98             n=-n;
99             while(n--)ret*=10.0;
100            return ((long long int)ret)%10;
101        }
102    }
103 };
104 template <typename T>
105 struct Complex {
106     T x,y; /* x + iy */
107     Complex(void):x(T()),y(T()) {}
108     Complex(T xx):x(xx) {}
109     Complex(T xx,T yy):x(xx),y(yy) {}
110     inline Complex operator-(void) const {
111         return Complex(-x,-y);
112     }
113     inline Complex operator+(const Complex& b) const {
114         return Complex(x+b.x,y+b.y);
115     }
116     inline Complex operator-(const Complex& b) const {
117         return Complex(x-b.x,y-b.y);
118     }
119     inline Complex operator*(const Complex& b) const {
120         return Complex(x*b.x-y*b.y,x*b.y+y*b.x);
121     }
122     inline Complex operator/(const Complex& b) const {
123         T bo=b.x*b.x+b.y*b.y;
124         return Complex((x*b.x+y*b.y)/bo,(y*b.x-x*b.y)/bo);
125     }
126     inline Complex& operator+=(const Complex& b) {
127         Complex tmp=(*this)+b;
128         (*this)=tmp;
129         return (*this);
130     }
131     inline Complex& operator-=(const Complex& b) {
132         Complex tmp=(*this)-b;
133         (*this)=tmp;
134         return (*this);
135     }
136     inline Complex& operator*=(const Complex& b) {

```

```

137     Complex tmp=(*this)*b;
138     (*this)=tmp;
139     return (*this);
140 }
141 inline Complex& operator/=(const Complex& b) {
142     Complex tmp=(*this)/b;
143     (*this)=tmp;
144     return (*this);
145 }
146 inline friend Complex operator+(const T& a, const Complex& b) {
147     return Complex(a)+b;
148 }
149 inline friend Complex operator-(const T& a, const Complex& b) {
150     return Complex(a)-b;
151 }
152 inline friend Complex operator*(const T& a, const Complex& b) {
153     return Complex(a)*b;
154 }
155 inline friend Complex operator/(const T& a, const Complex& b) {
156     return Complex(a)/b;
157 }
158 };
159 typedef dbl<> Double;
160 typedef Complex<Double> ComplexD;
161 typedef long long int ll;
162 const int maxn = 2000000; /* !! */
163 const Double pi(acos(-1.0));
164
165 void build(ComplexD _P[], ComplexD P[], int n, int m, int curr, int &cnt) {
166     if(m == n) {
167         _P[curr] = P[cnt++];
168     } else {
169         build(_P, P, n, m*2, curr, cnt);
170         build(_P, P, n, m*2, curr+m, cnt);
171     }
172 }
173
174 void FFT(ComplexD P[], int n, int oper) { /* n should be 2^k. */
175     static ComplexD _P[maxn];
176     int cnt = 0;
177     build(_P, P, n, 1, 0, cnt);
178     copy(_P, _P+n, P);
179     for(int d = 0; (1<<d)<n; ++d) {
180         int m = 1<<d;
181         int m2 = m*2;
182         Double p0 = pi / m * oper;
183         ComplexD unit_p0(cos(p0.x), sin(p0.x));
184         for(int i = 0; i < n; i += m2) {
185             ComplexD unit(1,0);
186             for(int j = 0; j < m; ++j) {
187                 ComplexD &P1 = P[i+j+m], &P2 = P[i+j];
188                 ComplexD t = unit * P1;
189                 P1 = P2 - t;
190                 P2 = P2 + t;
191                 unit *= unit_p0;
192             }
193         }
194     }
195     if(-1 == oper) {
196         for(int i = 0; i < n; ++i)
197             P[i] /= Double(n);
198     }
199 }
200 }

```

## 5.4 Number Theory Inverse

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int n=100000000;      /* */
5  const long long mod=1e9+7; /* prime required. */
6
7  long long fact[n],fiv[n],inv[n];
8
9  int main() {
10     fact[0]=fact[1]=1;
11     fiv[0]=fiv[1]=1;
12     inv[1]=1;
13     for (int i=2; i<n; i++) {
14         fact[i]=fact[i-1]*i%mod;
15         inv[i]=(mod-mod/i)*inv[mod%i]%mod;
16         fiv[i]=inv[i]*fiv[i-1]%mod;
17     }
18     for (int i=1; i<n; i++) {
19         if (fact[i]*fiv[i]%mod!=1) printf("fact wrong: %d\n",i);
20         if (inv[i]*i%mod!=1)      printf("inv wrong: %d\n",i);
21     }
22     cout<<"complete"<<endl;
23     return 0;
24 }

```

## 5.5 Linear Programming

```

1  /* 线性规划 */
2  #include<bits/stdc++.h>
3
4  using namespace std;
5  const int Maxn=110,Maxm=59;
6  class Simplex {
7      /*
8      功能:
9      接受有n个约束, m个基本变量的方程组a[0~n][0~m]
10     a[0][] 存放需要最大化的目标函数, a[][0] 存放常数
11     Base[] 存放基本变量的id, 初始为1~m
12     Rest[] 存放松弛变量的id, 初始为m+1~m+n
13     返回此线性规划的最小值ans
14     要求方案的话, Base[] 中的变量值为0, Rest[] 中的变量值为相应行的[0]
15     如果solve
16     返回1, 说明运行正常ans是它的最大值
17     返回0, 说明无可行解
18     返回-1, 说明解没有最大值
19     测试:
20     m=2,n=3
21     double a[4][3]={
22         {0,1,3},
23         {8,-1,1},
24         {-3,1,1},
25         {2,1,-4}
26     };
27     solve=1,ans=64/3;
28     注意ac不了可能是eps的问题
29     */
30 public:
31     static const double Inf;
32     static const double eps;
33     int n,m;

```

```

34 double a[Maxn][Maxm];
35 int Base[Maxm],Rest[Maxn];
36 double val[Maxm];
37 double ans;
38 void pt() {
39     for(int i=0; i<=n; i++) {
40         for(int j=0; j<=m; j++)printf("%.2f",a[i][j]);
41         puts("");
42     }
43 }
44 void pivot(int x,int y) { //将第x个非基本变量和第y个基本变量调换
45     swap(Rest[x],Base[y]);
46     double tmp=-1./a[x][y];
47     a[x][y]=-1.;
48     for(int j=0; j<=m; j++)a[x][j]*=tmp;
49     for(int i=0; i<=n; i++) {
50         if(i==x||fabs(a[i][y])<eps)continue;
51         tmp=a[i][y];
52         a[i][y]=0;
53         for(int j=0; j<=m; j++)a[i][j]+=tmp*a[x][j];
54     }
55 }
56 bool opt() {
57     while(1) {
58         int csi=0;
59         for(int i=1; i<=m; i++)if(a[0][i]>eps&&(!csi||Base[i]<Base[csi]))csi=i;
60         if(!csi)break;
61         int csj=0;
62         double cur;
63         for(int j=1; j<=n; j++) {
64             if(a[j][csi]>=eps)continue;
65             double tmp=-a[j][0]/a[j][csi];
66             if(!csj||tmp+eps<cur||(fabs(tmp-cur)<eps&&Rest[j]<Rest[csj]))csj=j,cur=tmp;
67         }
68         if(!csj)return 0;
69         pivot(csj,csi);
70     }
71     ans=a[0][0];
72     return 1;
73 }
74 bool init() {
75     ans=0;
76     for(int i=1; i<=m; i++)Base[i]=i;
77     for(int i=1; i<=n; i++)Rest[i]=m+i;
78     int cs=1;
79     for(int i=2; i<=n; i++)if(a[i][0]<a[cs][0])cs=i;
80     if(a[cs][0]>=eps)return 1;
81     static double tmp[Maxm];
82     for(int i=0; i<=m; i++)tmp[i]=a[0][i],a[0][i]=0;
83     for(int i=1; i<=n; i++)a[i][m+1]=1.;
84     a[0][m+1]=-1.;
85     Base[m+1]=m+n+1;
86     pivot(cs,++m);
87     opt();
88     m--;
89     if(a[0][0]<=eps)return 0;
90     cs=-1;
91     for(int i=1; i<=n; i++) {
92         if(Rest[i]>m+n) {
93             cs=i;
94             break;
95         }
96     }
97     if(cs>=1) {

```

```

98     int nxt=-1;
99     m++;
100    for(int i=1; i<=m; i++)if(a[cs][i]>eps||a[cs][i]<=-eps) {
101        nxt=i;
102        break;
103    }
104    pivot(cs,nxt);
105    m--;
106 }
107 for(int i=1; i<=m; i++) {
108     if(Base[i]>m+n) {
109         swap(Base[i],Base[m+1]);
110         for(int j=0; j<=n; j++)a[j][i]=a[j][m+1];
111         break;
112     }
113 }
114 for(int i=1; i<=m; i++)a[0][i]=0;
115 a[0][0]=tmp[0];
116 for(int i=1; i<=m; i++)if(Base[i]<=m)a[0][i]=tmp[Base[i]];
117 for(int i=1; i<=n; i++) {
118     if(Rest[i]<=m) {
119         for(int j=0; j<=m; j++)a[0][j]+=tmp[Rest[i]]*a[i][j];
120     }
121 }
122 return 1;
123 }
124 void getval() {
125     for(int i=1; i<=m; i++)val[i]=0;
126     for(int i=1; i<=n; i++)if(Rest[i]<=m)val[Rest[i]]=a[i][0];
127     //for(int i=1;i<=m;i++)printf("%.2f ",val[i]);puts("");
128 }
129 int solve() {
130     if(!init())return 0;
131     if(!opt())return -1;
132     getval();
133     return 1;
134 }
135 } solver;
136 const double Simplex:: Inf=1e80;
137 const double Simplex:: eps=1e-8;
138 int main() {
139     int m,n,type;
140     scanf("%d%d%d",&m,&n,&type);
141     solver.a[0][0]=0;
142     for(int i=1; i<=m; i++)scanf("%lf",&solver.a[0][i]);
143     for(int i=1; i<=n; i++) {
144         for(int j=1; j<=m+1; j++) {
145             if(j==m+1)scanf("%lf",&solver.a[i][0]);
146             else {
147                 scanf("%lf",&solver.a[i][j]);
148                 solver.a[i][j]=-solver.a[i][j];
149             }
150         }
151     }
152     solver.m=m,solver.n=n;
153     int rep=solver.solve();
154     if(rep==0)puts("Infeasible");
155     else if(rep==-1)puts("Unbounded");
156     else {
157         printf("%.12f\n",solver.ans);
158         if(type==1) {
159             for(int i=1; i<=m; i++)printf("%.12f%c",solver.val[i],i==m?'\\n':' ');
160         }
161     }

```

162 |}

## 6 String

### 6.1 Hash

```

1  /* Common hash for any substrings. */
2
3  typedef unsigned long long int llu;
4  #define MAXN 1000000
5  int n;
6  char s[MAXN];
7  llu H[MAXN], xP[MAXN], P = 9999111;
8  void init(void) {
9      int i = 0;
10     xP[0] = 111;
11     for(i = 1; i < MAXN; ++i) xP[i] = xP[i-1] * P;
12     H[n] = 0;
13     for(i = n-1; i >= 0; --i) H[i] = H[i+1]*P + s[i];
14 }
15 #define HASH(i,l)    (H[i] - H[i+l]*xP[l])

```

### 6.2 KMP

```

1  /* KMP, by Abreto<m@abreto.net>. */
2  #include <string.h>
3
4  /* !!NEED IMPROVING!! */
5
6  #define MAXW    10002
7  #define MAXT    1000002
8  char W[MAXW];   /* pattern */
9  char T[MAXT];
10 int pi[MAXW];
11
12 void compute(void) {
13     int i = 0, k = 0;
14     int m = strlen(W+1);
15     pi[1] = 0;
16     for(i = 2; i <= m; i++) {
17         while(k && W[k+1] != W[i])
18             k = pi[k];
19         if(W[k+1] == W[i])
20             k++;
21         pi[i] = k;
22     }
23 }
24 int kmp(void) {
25     int i = 0, q = 0;
26     int ret = 0;
27     int n = strlen(T+1), m = strlen(W+1);
28     compute();
29     for(i = 1; i <= n; ++i) {
30         while(q && W[q+1] != T[i])
31             q = pi[q];
32         if(W[q+1] == T[i])
33             q++;
34         if(q == m) {
35             ret++;
36             q = pi[q];
37         }

```

```

38 | }
39 | return ret;
40 | }

```

### 6.3 Suffix Array

```

1 | /* Suffix Array, copied. */
2 |
3 | #define MAXN    (200010)
4 | namespace mzry_sa {
5 | int wx[MAXN],wy[MAXN],*x,*y,wss[MAXN],wv[MAXN];
6 |
7 | bool dacmp(int *r,int n,int a,int b,int l) {
8 |     return a+l<n && b+l<n && r[a]==r[b]&&r[a+l]==r[b+l];
9 | }
10 | void da(int str[],int sa[],int rank[],int height[],int n,int m) {
11 |     int *s = str;
12 |     int *x=wx,*y=wy,*t,p;
13 |     int i,j;
14 |     for(i=0; i<m; i++)wss[i]=0;
15 |     for(i=0; i<n; i++)wss[x[i]=s[i]]++;
16 |     for(i=1; i<m; i++)wss[i]+=wss[i-1];
17 |     for(i=n-1; i>=0; i--)sa[--wss[x[i]]]=i;
18 |     for(j=1,p=1; p<n && j<n; j*=2,m=p) {
19 |         for(i=n-j,p=0; i<n; i++)y[p++]=i;
20 |         for(i=0; i<n; i++)if(sa[i]-j>=0)y[p++]=sa[i]-j;
21 |         for(i=0; i<n; i++)wv[i]=x[y[i]];
22 |         for(i=0; i<m; i++)wss[i]=0;
23 |         for(i=0; i<n; i++)wss[wv[i]]++;
24 |         for(i=1; i<m; i++)wss[i]+=wss[i-1];
25 |         for(i=n-1; i>=0; i--)sa[--wss[wv[i]]]=y[i];
26 |         for(t=x,x=y,y=t,p=1,i=1,x[sa[0]]=0; i<n; i++)
27 |             x[sa[i]]=dacmp(y,n,sa[i-1],sa[i],j)?p-1:p++;
28 |     }
29 |     for(int i=0; i<n; i++) rank[sa[i]]=i;
30 |     for(int i=0,j=0,k=0; i<n; height[rank[i++]]=k)
31 |         if(rank[i]>0)
32 |             for(k?k--:0,j=sa[rank[i]-1];
33 |                 i+k < n && j+k < n && str[i+k]==str[j+k];
34 |                 k++);
35 | }
36 | }
37 |
38 | /*
39 | Suffix array O(n lg^2 n)
40 | LCP table O(n)
41 | */
42 | #include <cstdio>
43 | #include <algorithm>
44 | #include <cstring>
45 |
46 | using namespace std;
47 |
48 | #define REP(i, n) for (int i = 0; i < (int)(n); ++i)
49 |
50 | namespace SuffixArray {
51 | const int MAXN = 1 << 21;
52 | char * S;
53 | int N, gap;
54 | int sa[MAXN], pos[MAXN], tmp[MAXN], lcp[MAXN];
55 |
56 | bool sufCmp(int i, int j) {
57 |     if (pos[i] != pos[j])

```



```

58     return pos[i] < pos[j];
59     i += gap;
60     j += gap;
61     return (i < N && j < N) ? pos[i] < pos[j] : i > j;
62 }
63
64 void buildSA() {
65     N = strlen(S);
66     REP(i, N) sa[i] = i, pos[i] = S[i];
67     for (gap = 1; gap <= N; gap <<= 1) {
68         sort(sa, sa + N, sufCmp);
69         REP(i, N - 1) tmp[i + 1] = tmp[i] + sufCmp(sa[i], sa[i + 1]);
70         REP(i, N) pos[sa[i]] = tmp[i];
71         if (tmp[N - 1] == N - 1) break;
72     }
73 }
74
75 void buildLCP() {
76     for (int i = 0, k = 0; i < N; ++i) if (pos[i] != N - 1) {
77         for (int j = sa[pos[i] + 1]; S[i + k] == S[j + k];)
78             ++k;
79         lcp[pos[i]] = k;
80         if (k) --k;
81     }
82 }
83 } // end namespace SuffixArray
84
85 namespace HashSuffixArray {
86     const int
87     MAXN = 1 << 21;
88
89     typedef unsigned long long hash;
90
91     const hash BASE = 137;
92
93     int N;
94     char * S;
95     int sa[MAXN];
96     hash h[MAXN], hPow[MAXN];
97
98     #define getHash(lo, size) (h[lo] - h[(lo) + (size)] * hPow[size])
99
100    inline bool sufCmp(int i, int j) {
101        int lo = 1, hi = min(N - i, N - j);
102        while (lo <= hi) {
103            int mid = (lo + hi) >> 1;
104            if (getHash(i, mid) == getHash(j, mid))
105                lo = mid + 1;
106            else
107                hi = mid - 1;
108        }
109        return S[i + hi] < S[j + hi];
110    }
111
112    void buildSA() {
113        N = strlen(S);
114        hPow[0] = 1;
115        for (int i = 1; i <= N; ++i)
116            hPow[i] = hPow[i - 1] * BASE;
117        h[N] = 0;
118        for (int i = N - 1; i >= 0; --i)
119            h[i] = h[i + 1] * BASE + S[i], sa[i] = i;
120
121        stable_sort(sa, sa + N, sufCmp);

```

```

122 }
123
124 } // end namespace HashSuffixArray
125
126 namespace lrj_sa {
127 const int MAXN = 1000;
128 char s[MAXN]; /* 原始字符数组（最后一个字符应必须是0，而前面的字符必须非0） */
129 int sa[MAXN], t[MAXN], t2[MAXN], c[MAXN], n; /* n seems to be the length of s. */
130 /* every character is in [0,m-1] */
131 void build_sa(int m) {
132     int i, *x = t, *y = t2;
133     for(i = 0; i < m; ++i) c[i] = 0;
134     for(i = 0; i < n; ++i) c[x[i]=s[i]]++;
135     for(i = 1; i < m; ++i) c[i] += c[i-1];
136     for(i = n-1; i >= 0; --i) sa[--c[x[i]]] = i;
137     for(int k = 1; k <= n; k <<= 1) {
138         int p = 0;
139         for(i = n-k; i < n; ++i) y[p++] = i;
140         for(i = 0; i < n; ++i) if(sa[i] >= k) y[p++] = sa[i]-k;
141         for(i = 0; i < m; ++i) c[i] = 0;
142         for(i = 0; i < n; ++i) c[x[y[i]]]++;
143         for(i = 0; i < m; ++i) c[i] += c[i-1];
144         for(i = n-1; i >= 0; --i) sa[--c[x[y[i]]]] = y[i];
145         swap(x,y);
146         p = 1;
147         x[sa[0]] = 0;
148         for(i = 1; i < n; ++i)
149             x[sa[i]] = y[sa[i-1]]==y[sa[i]] && y[sa[i-1]+k]==y[sa[i]+k] ? p-1:p++;
150         if(p >= n) break;
151         m = p;
152     }
153 }
154 int rank[MAXN], height[MAXN];
155 void get_height(void) {
156     int i,j,k = 0;
157     for(i = 0; i < n; ++i) rank[sa[i]] = i;
158     for(i = 0; i < n; ++i) {
159         if(k) k--;
160         j = sa[rank[i]-1];
161         while(s[i+k]==s[j+k]) k++;
162         height[rank[i]] = k;
163     }
164 }
165 } // end namespace lrj_sa

```

## 7 Tool

### 7.1 IO plug-in

```

1 /* I/O Plug-in, by Abreto <m@abreto.net>. */
2 #include <stdio.h>
3
4 #if ( _WIN32 || __WIN32__ || _WIN64 || __WIN64__ )
5 #define INT64 "%I64d"
6 #else
7 #define INT64 "%lld"
8 #endif
9
10 #if ( _WIN32 || __WIN32__ || _WIN64 || __WIN64__ )
11 #define UNS64 "%I64u"
12 #else
13 #define UNS64 "%llu"

```

```

14 #endif
15
16 #define ISDIGIT(x) ((x>='0')&&(x<='9'))
17 int readn(int *n) {
18     int c=0;
19     *n=0;
20     for(; !ISDIGIT(c); c=getchar());
21     for(; ISDIGIT(c); c=getchar()) *n=( *n)*10+c-'0';
22     return (*n);
23 }
24 void putn(int n) {
25     int ns[16]= {0,n%10},nd=1;
26     while(n/=10) ns[++nd]=n%10;
27     while(nd) putchar(ns[nd--]+'0');
28 }

```

## 7.2 Matrix (including quickpow)

```

1
2 typedef long long int ll;
3 template <class T, int maxn, ll mod>
4 struct mat {
5     int N;
6     T a[maxn][maxn];    /* 1-based. */
7     mat(void):N(0) {}
8     mat(int n, int v = 0) {
9         N = n;
10        for(int i = 1; i <= N; ++i)
11            for(int j = 1; j <= N; ++j)
12                a[i][j] = 0;
13        for(int i = 1; i <= N; ++i)
14            a[i][i] = v;
15    }
16    mat operator-(void) const {
17        mat ret(N);
18        for(int i = 1; i <= N; ++i)
19            for(int j = 1; j <= N; ++j)
20                ret.a[i][j] = (mod-a[i][j])%mod;
21        return ret;
22    }
23    mat operator+(const mat &b) const {
24        mat ret(N);
25        for(int i = 1; i <= N; ++i)
26            for(int j = 1; j <= N; ++j)
27                ret.a[i][j] = (a[i][j] + b.a[i][j])%mod;
28        return ret;
29    }
30    mat operator-(const mat &b) const {
31        mat ret(N);
32        for(int i = 1; i <= N; ++i)
33            for(int j = 1; j <= N; ++j)
34                ret.a[i][j] = (a[i][j] - b.a[i][j] + mod) % mod;
35        return ret;
36    }
37    mat operator*(const mat &b) const {
38        mat ret(N);
39        for(int i = 1; i <= N; ++i)
40            for(int j = 1; j <= N; ++j)
41                for(int k = 1; k <= N; ++k) {
42                    /* T t = (a[i][k] * b.a[k][j]) % mod; // delete %mod if get TLE. */
43                    ret.a[i][j] = (ret.a[i][j] + a[i][k] * b.a[k][j]) % mod; /* seprate this
44                        line if get WA. */
45                }
46    }

```

```

45     return ret;
46 }
47 mat operator^(long long int p) const {
48     mat ret(N,1);
49     mat base = (*this);
50     while(p) {
51         if(p & 1) ret = ret * base;
52         base = base * base;
53         p >>= 1;
54     }
55     return ret;
56 }
57 };

```

### 7.3 Double Class

```

1  /* Double Class, by Abreto<m@abreto.net>. */
2
3  #define eps (1e-9)
4  template < typename T = double >
5  struct dbl {
6      T x;
7      dbl(void):x(0.0) {}
8      template < typename U>
9      dbl(U a):x((T)a) {}
10     inline char sgn(void) {
11         return ((x>=-eps)&&(x<=eps))?(0):((x>eps)?(1):(-1));
12     }
13     inline T tabs(void) {
14         return ((x>=-eps)&&(x<=eps))?(0.0):((x>eps)?(x):(-x));
15     }
16     inline dbl abs(void) {
17         return dbl(tabs());
18     }
19     template < typename U> inline dbl &operator=(const U b) {
20         x=(T)b;
21         return (*this);
22     }
23     inline T *operator&(void) {
24         return &x;
25     }
26     inline dbl operator-(void) const {
27         return dbl(-x);
28     }
29     inline dbl operator+(const dbl &b) const {
30         return dbl(x+b.x);
31     }
32     inline dbl operator-(const dbl &b) const {
33         return dbl(x-b.x);
34     }
35     inline dbl operator*(const dbl &b) const {
36         return dbl(x*b.x);
37     }
38     inline dbl operator/(const dbl &b) const {
39         return dbl(x/b.x);
40     }
41     template < typename U> inline dbl operator^(const U &b) const {
42         T ret=1.0,base=x;
43         while(b) {
44             if(b&1)ret*=base;
45             base*=base;
46             b>>=1;
47         }

```

```

48     return dbl(ret);
49 }
50 inline dbl operator+=(const dbl &b) {
51     return dbl(x+=b.x);
52 }
53 inline dbl operator-=(const dbl &b) {
54     return dbl(x-=b.x);
55 }
56 inline dbl operator*=(const dbl &b) {
57     return dbl(x*=b.x);
58 }
59 inline dbl operator/=(const dbl &b) {
60     return dbl(x/=b.x);
61 }
62 template <typename U> inline dbl operator^=(const U &b) {
63     dbl tmp=(*this)^b;
64     *this=tmp;
65     return tmp;
66 }
67 inline bool operator==(const dbl &b) const {
68     return (0 == ((*this)-b).sgn());
69 }
70 inline bool operator!=(const dbl &b) const {
71     return (0 != ((*this)-b).sgn());
72 }
73 inline bool operator<(const dbl &b) const {
74     return (-1 == ((*this)-b).sgn());
75 }
76 inline bool operator<=(const dbl &b) const {
77     return (((*this)==b) || ((*this)<b));
78 }
79 inline bool operator>(const dbl &b) const {
80     return (b < (*this));
81 }
82 inline bool operator>=(const dbl &b) const {
83     return (((*this)==b) || ((*this)>b));
84 }
85 template <typename U> inline operator U() const {
86     return (U)x;
87 }
88 inline char operator[](unsigned n) {
89     if(n >= 0) {
90         long long int ret=x;
91         while(n--) {
92             ret/=10;
93         }
94         return (ret%10);
95     } else {
96         T ret=x;
97         n=-n;
98         while(n--)ret*=10.0;
99         return ((long long int)ret)%10;
100     }
101 }
102 };
103
104 typedef dbl<> Double;

```

## 7.4 Complex Class

```

1  /* Complex Class, by Abreto<m@abreto.net>. */
2
3  template <typename T>

```

```

4 struct Complex {
5     T x,y; /* x + iy */
6     Complex(void):x(T()),y(T()) {}
7     Complex(T xx):x(xx) {}
8     Complex(T xx,T yy):x(xx),y(yy) {}
9     inline Complex operator-(void) const {
10         return Complex(-x,-y);
11     }
12     inline Complex operator+(const Complex& b) const {
13         return Complex(x+b.x,y+b.y);
14     }
15     inline Complex operator-(const Complex& b) const {
16         return Complex(x-b.x,y-b.y);
17     }
18     inline Complex operator*(const Complex& b) const {
19         return Complex(x*b.x-y*b.y,x*b.y+y*b.x);
20     }
21     inline Complex operator/(const Complex& b) const {
22         T bo=b.x*b.x+b.y*b.y;
23         return Complex((x*b.x+y*b.y)/bo,(y*b.x-x*b.y)/bo);
24     }
25     inline Complex& operator+=(const Complex& b) {
26         Complex tmp=(*this)+b;
27         (*this)=tmp;
28         return (*this);
29     }
30     inline Complex& operator-=(const Complex& b) {
31         Complex tmp=(*this)-b;
32         (*this)=tmp;
33         return (*this);
34     }
35     inline Complex& operator*=(const Complex& b) {
36         Complex tmp=(*this)*b;
37         (*this)=tmp;
38         return (*this);
39     }
40     inline Complex& operator/=(const Complex& b) {
41         Complex tmp=(*this)/b;
42         (*this)=tmp;
43         return (*this);
44     }
45     inline friend Complex operator+(const T& a, const Complex& b) {
46         return Complex(a)+b;
47     }
48     inline friend Complex operator-(const T& a, const Complex& b) {
49         return Complex(a)-b;
50     }
51     inline friend Complex operator*(const T& a, const Complex& b) {
52         return Complex(a)*b;
53     }
54     inline friend Complex operator/(const T& a, const Complex& b) {
55         return Complex(a)/b;
56     }
57 };

```

## 8 Appendix

## 9 Graph Algorithms

Welcome to the new episode of PrinceOfPersia presents: Fun with algorithms ;)

You can find all the definitions here in the book "Introduction to graph theory", Douglas.B West.

Important graph algorithms :

## 9.1 DFS

The most useful graph algorithms are search algorithms. DFS (Depth First Search) is one of them. While running DFS, we assign colors to the vertices (initially white). Algorithm itself is really simple :

```
dfs (v):
    color[v] = gray
    for u in adj[v]:
        if color[u] == white
            then dfs(u)
    color[v] = black
```

Black color here is not used, but you can use it sometimes.

Time complexity :  $O(n + m)$ .

### 9.1.1 DFS tree

DFS tree is a rooted tree that is built like this :

```
let T be a new tree
dfs (v):
    color[v] = gray
    for u in adj[v]:
        if color[u] == white
            then dfs(u) and par[u] = v (in T)

    color[v] = black
```

Lemma: There is no cross edges, it means if there is an edge between  $v$  and  $u$ , then  $v = \text{par}[u]$  or  $u = \text{par}[v]$ .

### 9.1.2 Starting time, finishing time

Starting time of a vertex is the time we enter it (the order we enter it) and its finishing time is the time we leave it. Calculating these are easy :

```
TIME = 0
dfs (v):
    st[v] = TIME ++
    color[v] = gray
    for u in adj[v]:
        if color[u] == white
            then dfs(u)
    color[v] = black
    ft[v] = TIME // or we can use TIME ++
```

It is useable in specially data structure problems (convert the tree into an array).

Lemma: If we run  $\text{dfs}(\text{root})$  in a rooted tree, then  $v$  is an ancestor of  $u$  if and only if  $\text{stv} \leq \text{stu} < \text{ftu} < \text{ftv}$ .

So, given arrays  $\text{st}$  and  $\text{ft}$  we can rebuild the tree.

### 9.1.3 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  $w$ ).

### 9.1.4 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
            then par[u] = v and dfs(u) and d[v] = min(d[v], d[u])
            if d[u] >= h[v] and (v != root or number_of_children(v) > 1)
                then the edge v is a cut vertex
        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  $w$ ).

### 9.1.5 Finding Eulerian tours

It is quite like DFS, with a little change :

```

vector E
dfs (v):
    color[v] = gray
    for u in adj[v]:
        erase the edge v-u and dfs(u)
    color[v] = black
    push v at the end of e

```

$e$  is the answer.



## 9.2 BFS

BFS is another search algorithm (Breadth First Search). It is usually used to calculate the distances from a vertex  $v$  to all other vertices in unweighted graphs.

Code :

```

BFS(v):
    for each vertex i
        do  $d[i] = \text{inf}$ 
     $d[v] = 0$ 
    queue q
    q.push(v)
    while q is not empty
         $u = \text{q.front}()$ 
        q.pop()
        for each w in adj[u]
            if  $d[w] == \text{inf}$ 
                then  $d[w] = d[u] + 1$ , q.push(w)

```

Distance of vertex  $u$  from  $v$  is  $d[u]$ .

Time complexity :  $O(n + m)$ .

### 9.2.1 BFS tree

BFS tree is a rooted tree that is built like this :

let  $T$  be a new tree

```

BFS(v):
    for each vertex i
        do  $d[i] = \text{inf}$ 
     $d[v] = 0$ 
    queue q
    q.push(v)
    while q is not empty
         $u = \text{q.front}()$ 
        q.pop()
        for each w in adj[u]
            if  $d[w] == \text{inf}$ 
                then  $d[w] = d[u] + 1$ , q.push(w) and  $\text{par}[w] = u$  (in  $T$ )

```

## 9.3 SCC

The most useful and fast-coding algorithm for finding SCCs is Kosaraju.

In this algorithm, first of all we run DFS on the graph and sort the vertices in decreasing of their finishing time (we can use a stack).

Then, we start from the vertex with the greatest finishing time, and for each vertex  $v$  that is not yet in any SCC, do : for each  $u$  that  $v$  is reachable by  $u$  and  $u$  is not yet in any SCC, put it in the SCC of vertex  $v$ . The code is quite simple.

## 9.4 Shortest path

Shortest path algorithms are algorithms to find some shortest paths in directed or undirected graphs.

### 9.4.1 Dijkstra

This algorithm is a single source shortest path (from one source to any other vertices). Pay attention that you can't have edges with negative weight.

Pseudo code :

```
dijkstra(v) :
    d[i] = inf for each vertex i
    d[v] = 0
    s = new empty set
    while s.size() < n
        x = inf
        u = -1
        for each i in V-s // V is the set of vertices
            if x >= d[i]
                then x = d[i], u = i
        insert u into s
        // The process from now is called Relaxing
        for each i in adj[u]
            d[i] = min(d[i], d[u] + w(u,i))
```

There are two different implementations for this. Both are useful (C++11).

One)  $O(n^2)$

```
int mark[MAXN];
void dijkstra(int v){
    fill(d,d + n, inf);
    fill(mark, mark + n, false);
    d[v] = 0;
    int u;
    while(true){
        int x = inf;
        u = -1;
        for(int i = 0;i < n;i ++){
            if(!mark[i] and x >= d[i])
                x = d[i], u = i;
        }
        if(u == -1) break;
        mark[u] = true;
        for(auto p : adj[u]) // adj[v][i] = pair(vertex, weight)
            if(d[p.first] > d[u] + p.second)
                d[p.first] = d[u] + p.second;
    }
}
```

Two)

1) Using std :: set :

```
void dijkstra(int v){
    fill(d,d + n, inf);
    d[v] = 0;
    int u;
    set<pair<int,int> > s;
    s.insert({d[v], v});
    while(!s.empty()){
        u = s.begin() -> second;
```

```

s.erase(s.begin());
for(auto p : adj[u]) //adj[v][i] = pair(vertex, weight)
if(d[p.first] > d[u] + p.second){
s.erase({d[p.first], p.first});
d[p.first] = d[u] + p.second;
s.insert({d[p.first], p.first});
}
}
}

```

2) Using std :: priority\_queue (better):

```

bool mark[MAXN];
void dijkstra(int v){
fill(d,d + n, inf);
fill(mark, mark + n, false);
d[v] = 0;
int u;
priority_queue<pair<int,int>,vector<pair<int,int> >, less<pair<int,int> > > pq;
pq.push({d[v], v});
while(!pq.empty()){
u = pq.top().second;
pq.pop();
if(mark[u])
continue;
mark[u] = true;
for(auto p : adj[u]) //adj[v][i] = pair(vertex, weight)
if(d[p.first] > d[u] + p.second){
d[p.first] = d[u] + p.second;
pq.push({d[p.first], p.first});
}
}
}
}

```

### 9.4.2 Floyd-Warshall

Floyd-Warshall algorithm is an all-pairs shortest path algorithm using dynamic programming. It is too simple and undrestandable :

```

Floyd-Warshall()
d[v][u] = inf for each pair (v,u)
d[v][v] = 0 for each vertex v
for k = 1 to n
for i = 1 to n
for j = 1 to n
    d[i][j] = min(d[i][j], d[i][k] + d[k][j])

```

Time complexity :  $O(n^3)$ .

### 9.4.3 Bellman-Ford

Bellman-Ford is an algorithm for single source shortest path where edges can be negative (but if there is a cycle with negative weight, then this problem will be NP).

The main idea is to relax all the edges exactly  $n - 1$  times (read relaxation above in dijkstra). You can prove this algorithm using induction.

If in the  $n$ -th step, we relax an edge, then we have a negative cycle (this is if and only if).

Code :

```

Bellman-Ford(int v)
d[i] = inf for each vertex i
d[v] = 0
for step = 1 to n
  for all edges like e
    i = e.first // first end
    j = e.second // second end
    w = e.weight
    if d[j] > d[i] + w
      if step == n
        then return "Negative cycle found"
          d[j] = d[i] + w

```

Time complexity :  $O(nm)$ .

#### 9.4.4 SPFA

SPFA (Shortest Path Faster Algorithm) is a fast and simple algorithm (single source) that its complexity is not calculated yet. But if  $m = O(n^2)$  it's better to use the first implementation of Dijkstra.

The origin of this algorithm is unknown. It's said that at first Chinese coders used it in programming contests.

Its code looks like the combination of Dijkstra and BFS :

```

SPFA(v):
d[i] = inf for each vertex i
d[v] = 0
queue q
q.push(v)
while q is not empty
  u = q.front()
  q.pop()
  for each i in adj[u]
    if d[i] > d[u] + w(u,i)
      then d[i] = d[u] + w(u,i)
      if i is not in q
        then q.push(i)

```

Time complexity : Unknown!.

### 9.5 MST

MST = Minimum Spanning Tree :) (if you don't know what it is, google it).

Best MST algorithms :

#### 9.5.1 Kruskal

In this algorithm, first we sort the edges in ascending order of their weight in an array of edges. Then in order of the sorted array, we add each edge if and only if after adding it there won't be any cycle (check it using DSU).

Code :

```

Kruskal()
solve all edges in ascending order of their weight in an array e
ans = 0
for i = 1 to m
v = e.first
u = e.second
w = e.weight
if merge(v,u) // there will be no cycle
    then ans += w

```

Time complexity :  $O(m \log m)$ .

### 9.5.2 Prim

In this approach, we act like Dijkstra. We have a set of vertices  $S$ , in each step we add the nearest vertex to  $S$ , in  $S$  (distance of  $v$  from  $S = \min_{u \in S}(\text{weight}(u, v))$  where  $\text{weight}(i, j)$  is the weight of the edge from  $i$  to  $j$ ) .

So, pseudo code will be like this:

```

Prim()
S = new empty set
for i = 1 to n
d[i] = inf
while S.size() < n
x = inf
v = -1
for each i in V - S // V is the set of vertices
if x >= d[v]
then x = d[v], v = i
d[v] = 0
S.insert(v)
for each u in adj[v]
do d[u] = min(d[u], w(v,u))

```

C++ code: One)  $O(n^2)$

```

bool mark[MAXN];
void prim(){
fill(d, d + n, inf);
fill(mark, mark + n, false);
int x,v;
while(true){
x = inf;
v = -1;
for(int i = 0;i < n;i ++){
if(!mark[i] and x >= d[i])
x = d[i], v = i;
if(v == -1)
break;
d[v] = 0;
mark[v] = true;
for(auto p : adj[v]){ //adj[v][i] = pair(vertex, weight)
int u = p.first, w = p.second;
d[u] = min(d[u], w);
}
}
}

```

```

}
}

```

Two)  $O(m \log n)$

```

void prim(){
fill(d, d + n, inf);
set<pair<int,int> > s;
for(int i = 0;i < n;i ++){
s.insert({d[i],i});
int v;
while(!s.empty()){
v = s.begin() -> second;
s.erase(s.begin());
for(auto p : adj[v]){
int u = p.first, w = p.second;
if(d[u] > w){
s.erase({d[u], u});
d[u] = w;
s.insert({d[u], u});
}
}
}
}
}

```

As Dijkstra you can use `std :: priority_queue` instead of `std :: set`.

## 9.6 Maximum Flow

I only wanna put the source code here (EdmondsKarp):

```

algorithm EdmondsKarp
input:
    C[1..n, 1..n] (Capacity matrix)
    E[1..n, 1..?] (Neighbour lists)
    s              (Source)
    t              (Sink)
output:
    f              (Value of maximum flow)
    F              (A matrix giving a legal flow with the maximum value)
f := 0 (Initial flow is zero)
F := array(1..n, 1..n) (Residual capacity from u to v is C[u,v] - F[u,v])
forever
    m, P := BreadthFirstSearch(C, E, s, t, F)
    if m = 0
        break
    f := f + m
    (Backtrack search, and write flow)
    v := t
    while v != s
        u := P[v]
        F[u,v] := F[u,v] + m
        F[v,u] := F[v,u] - m
        v := u
return (f, F)

```

algorithm BreadthFirstSearch

input:

C, E, s, t, F

output:

M[t] (Capacity of path found)

P (Parent table)

P := array(1..n)

for u in 1..n

P[u] := -1

P[s] := -2 (make sure source is not rediscovered)

M := array(1..n) (Capacity of found path to node)

M[s] :=  $\infty$

Q := queue()

Q.offer(s)

while Q.size() > 0

u := Q.poll()

for v in E[u]

(If there is available capacity, and v is not seen before in search)

if  $C[u,v] - F[u,v] > 0$  and  $P[v] = -1$

P[v] := u

M[v] := min(M[u],  $C[u,v] - F[u,v]$ )

if v = t

Q.offer(v)

else

return M[t], P

return 0, P

EdmondsKarp pseudo code using Adjacency nodes:

algorithm EdmondsKarp

input:

graph (Graph with list of Adjacency nodes with capacities,flow,reverse and destinations)

s (Source)

t (Sink)

output:

flow (Value of maximum flow)

flow := 0 (Initial flow to zero)

q := array(1..n) (Initialize q to graph length)

while true

qt := 0 (Variable to iterate over all the corresponding edges for a source)

q[qt++] := s (initialize source array)

pred := array(q.length) (Initialize predecessor List with the graph length)

for qh=0;qh < qt && pred[t] == null

cur := q[qh]

for (graph[cur]) (Iterate over list of Edges)

Edge[] e := graph[cur] (Each edge should be associated with Capacity)

if pred[e.t] == null && e.cap > e.f

pred[e.t] := e

q[qt++] := e.t

if pred[t] == null

break

```

int df := MAX VALUE (Initialize to max integer value)
for u = t; u != s; u = pred[u].s
    df := min(df, pred[u].cap - pred[u].f)
for u = t; u != s; u = pred[u].s
    pred[u].f := pred[u].f + df
    pEdge := array(PredEdge)
    pEdge := graph[pred[u].t]
    pEdge[pred[u].rev].f := pEdge[pred[u].rev].f - df;
flow := flow + df
return flow

```

### 9.6.1 Dinic's algorithm

Here is Dinic's algorithm as you wanted.

Input: A network  $G = ((V, E), c, s, t)$ .

Output: A max  $s - t$  flow.

1.set  $f(e) = 0$  for each  $e$  in  $E$

2.Construct  $G\_L$  from  $G\_f$  of  $G$ . if  $\text{dist}(t) == \text{inf}$ , then stop and output  $f$

3.Find a blocking flow  $fp$  in  $G\_L$

4.Augment flow  $f$  by  $fp$  and go back to step 2.

Time complexity :  $O(mm \log n)$ .

Theorem: Maximum flow = minimum cut.

### 9.6.2 Maximum Matching in bipartite graphs

Maximum matching in bipartite graphs is solvable also by maximum flow like below :

Add two vertices  $S, T$  to the graph, every edge from  $X$  to  $Y$  (graph parts) has capacity 1, add an edge from  $S$  with capacity 1 to every vertex in  $X$ , add an edge from every vertex in  $Y$  with capacity 1 to  $T$ .

Finally, answer = maximum matching from  $S$  to  $T$  .

But it can be done really easier using DFS.

As, you know, a bipartite matching is the maximum matching if and only if there is no augmenting path (read Introduction to graph theory).

The code below finds a augmenting path:

```

bool dfs(int v){ // v is in X, it reaturns true if and only if there is an augmenting path starting from v
if(mark[v])
return false;
mark[v] = true;
for(auto &u : adj[v])
if(match[u] == -1 or dfs(match[u])) // match[i] = the vertex i is matched with in the current matching, initialia
return match[v] = u, match[u] = v, true;
return false;
}

```

An easy way to solve the problem is:

```

for(int i = 0; i < n; i++){
if(match[i] == -1){
memset(mark, false, sizeof mark);
dfs(i);
}
}

```

But there is a faster way:



```

while(true){
memset(mark, false, sizeof mark);
bool fnd = false;
for(int i = 0; i < n; i++) if(match[i] == -1 && !mark[i])
fnd |= dfs(i);
if(!fnd)
break;
}

```

In both cases, time complexity =  $O(nm)$ .

## 9.7 Trees

Trees are the most important graphs.

In the last lectures we talked about segment trees on trees and heavy-light decomposition.

### 9.7.1 Partial sum on trees

We can also use partial sum on trees.

Example: Having a rooted tree, each vertex has a value (initially 0), each query gives you numbers  $v$  and  $u$  ( $v$  is an ancestor of  $u$ ) and asks you to increase the value of all vertices in the path from  $u$  to  $v$  by 1.

So, we have an array  $p$ , and for each query, we increase  $p[u]$  by 1 and decrease  $p[\text{par}[v]]$  by 1. The we run this (like a normal partial sum):

```

void dfs(int v){
for(auto u : adj[v])
if(u - par[v])
dfs(u), p[v] += p[u];
}

```

### 9.7.2 DSU on trees

We can use DSU on a rooted tree (not tree DSUs, DSUs like vectors).

For example, in each node, we have a vector, all nodes in its subtree (this can be used only for offline queries, because we may have to delete it for memory usage).

Here again we use DSU technique, we will have a vector  $V$  for every node. When we want to have  $V[v]$  we should merge the vectors of its children. I mean if its children are  $u_1, u_2, \dots, u_k$  where  $V[u_1].size() \leq V[u_2].size() \leq \dots \leq V[u_k].size()$ , we will put all elements from  $V[u_i]$  for every  $1 \leq i < k$ , in  $V[k]$  and then,  $V[v] = V[u_k]$ .

Using this trick, time complexity will be .

C++ example (it's a little complicated) :

```

typedef vector<int> vi;
vi *V[MAXN];
void dfs(int v, int par = -1){
int mx = 0, chl = -1;
for(auto u : adj[v])if(par - u){
dfs(u,v);
if(mx < V[u]->size()){
mx = V[u]->size();
chl = u;
}
}
for(auto u : adj[v])if(par - u and chl - u){

```

```

for(auto a : *V[u])
V[chl]->push_back(a);
delete V[u];
}
if(chl + 1)
V[v] = V[chl];
else{
V[v] = new vi;
V[v]->push_back(v);
}
}
}

```

### 9.7.3 LCA

LCA of two vertices in a rooted tree, is their lowest common ancestor.

There are so many algorithms for this, I will discuss the important ones.

Each algorithm has complexities  $< O(f(n)), O(g(n)) >$ , it means that this algorithm's preprocess is  $O(f(n))$  and answering a query is  $O(g(n))$ .

In all algorithms,  $h[v]$  = height of vertex  $v$ . One) Brute force  $< O(n), O(n) >$

The simplest approach. We go up enough to achieve the goal.

Preprocess :

```

void dfs(int v,int p = -1){
if(par + 1)
h[v] = h[p] + 1;
par[v] = p;
for(auto u : adj[v]) if(p - u)
dfs(u,v);
}

```

Query :

```

int LCA(int v,int u){
if(v == u)
return v;
if(h[v] < h[u])
swap(v,u);
return LCA(par[v], u);
}

```

Two) SQRT decomposition

I talked about SQRT decomposition in the first lecture.

Here, we will cut the tree into  $\sqrt{H}$  ( $H$  = height of the tree), starting from 0,  $k$ -th of them contains all vertices with  $h$  in interval  $[k\sqrt{H}, (k+1)\sqrt{H}]$ .

Also, for each vertex  $v$  in  $k$ -th piece, we store  $r[v]$  that is, its lowest ancestor in the piece number  $k-1$ .

Preprocess:

```

void dfs(int v,int p = -1){
if(par + 1)
h[v] = h[p] + 1;
par[v] = p;
if(h[v] % SQRT == 0)
r[v] = p;
else

```

```

r[v] = r[p];
for(auto u : adj[v]) if(p - u)
dfs(u,v);
}

```

Query:

```

int LCA(int v,int u){
if(v == u)
return v;
if(h[v] < h[u])
swap(v,u);
if(h[v] == h[u])
return (r[v] == r[u] ? LCA(par[v], par[u]) : LCA(r[v], r[u]));
if(h[v] - h[u] < SQRT)
return LCA(par[v], u);
return LCA(r[v], u);
}

```

Three) Sparse table  $\langle O(n \log n), O(1) \rangle$

Let's introduce you an order of tree vertices, haas and I named it Euler order. It is like DFS order, but every time we enter a vertex, we write it's number down (even when we come from a child to this node in DFS).

Code for calculate this :

```

vector<int> euler;
void dfs(int v,int p = -1){
euler.push_back(v);
for(auto u : adj[v]) if(p - u)
dfs(u,v), euler.push_back(v);
}

```

If we have a vector<pair<int,int> > instead of this and push  $h[v]$ ,  $v$  in the vector, and the first time  $h[v]$ ,  $v$  is appeared is  $s[v]$  and  $s[v] < s[u]$  then  $LCA(v, u) = (\min_{i=s[v]}^{s[u]} euler[i]).second$ .

For this propose we can use RMQ problem, and the best algorithm for that, is to use Sparse table.

Four) Something like Sparse table :)  $\langle O(n \log n), O(\log n) \rangle$

This is the most useful and simple (among fast algorithms) algorithm.

For each vector  $v$  and number  $i$ , we store its  $2^i$ -th ancestor. This can be done in  $O(n \log n)$ . Then, for each query, we find the lowest ancestors of them which are in the same height, but different (read the source code for understanding).

Preprocess:

```

int par[MAXN][MAXLOG]; // initially all -1
void dfs(int v,int p = -1){
par[v][0] = p;
if(p + 1)
h[v] = h[p] + 1;
for(int i = 1; i < MAXLOG; i++)
if(par[v][i-1] + 1)
par[v][i] = par[par[v][i-1]][i-1];
for(auto u : adj[v]) if(p - u)
dfs(u,v);
}

```

Query:

```

int LCA(int v,int u){
if(h[v] < h[u])
swap(v,u);
for(int i = MAXLOG - 1;i >= 0;i --)
if(par[v][i] + 1 and h[par[v][i]] >= h[u])
v = par[v][i];
// now h[v] = h[u]
if(v == u)
return v;
for(int i = MAXLOG - 1;i >= 0;i --)
if(par[v][i] - par[u][i])
v = par[v][i], u = par[u][i];
return par[v][0];
}

```

Five) Advance RMQ  $< O(n), O(1) >$

In the third approach, we said that LCA can be solved by RMQ.

When you look at the vector euler you see that for each  $i$  that  $1 \leq i < \text{euler.size}()$ ,  $|\text{euler}[i].\text{first} - \text{euler}[i + 1].\text{first}| = 1$ .

So, we can convert the euler from its size (we consider its size is  $n + 1$ ) into a binary sequence of length  $n$  (if  $\text{euler}[i].\text{first} - \text{euler}[i + 1].\text{first} = 1$  we put 1 otherwise 0).

So, we have to solve the problem on a binary sequence  $A$ .

To solve this restricted version of the problem we need to partition  $A$  into blocks of size  $l$ . Let  $A'[i]$  be the minimum value for the  $i$ -th block in  $A$  and  $B[i]$  be the position of this minimum value in  $A$ . Both  $A$  and  $B$  are long. Now, we preprocess  $A'$  using the Sparse Table algorithm described in lecture 1. This will take time and space. After this preprocessing we can make queries that span over several blocks in  $O(1)$ . It remains now to show how the in-block queries can be made. Note that the length of a block is  $l$ , which is quite small. Also, note that  $A$  is a binary array. The total number of binary arrays of size  $l$  is  $2^l$ . So, for each binary block of size  $l$  we need to lock up in a table  $P$  the value for RMQ between every pair of indices. This can be trivially computed in time and space. To index table  $P$ , preprocess the type of each block in  $A$  and store it in array  $T$ . The block type is a binary number obtained by replacing  $-1$  with 0 and  $+1$  with 1 (as described above).

Now, to answer  $\text{RMQA}(i, j)$  we have two cases:

$i$  and  $j$  are in the same block, so we use the value computed in  $P$  and  $T$

$i$  and  $j$  are in different blocks, so we compute three values: the minimum from  $i$  to the end of  $i$ 's block using  $P$  and  $T$ , the minimum of all blocks between  $i$ 's and  $j$ 's block using precomputed queries on  $A'$  and the minimum from the beginning of  $j$ 's block to  $j$ , again using  $T$  and  $P$ ; finally return the position where the overall minimum is using the three values you just computed.

Six) Tarjan's algorithm  $O(na(n))$  ( $a(n)$  is the inverse ackermann function)

Tarjan's algorithm is offline; that is, unlike other lowest common ancestor algorithms, it requires that all pairs of nodes for which the lowest common ancestor is desired must be specified in advance. The simplest version of the algorithm uses the union-find data structure, which unlike other lowest common ancestor data structures can take more than constant time per operation when the number of pairs of nodes is similar in magnitude to the number of nodes. A later refinement by Gabow & Tarjan (1983) speeds the algorithm up to linear time.

The pseudocode below determines the lowest common ancestor of each pair in  $P$ , given the root  $r$  of a tree in which the children of node  $n$  are in the set  $n.\text{children}$ . For this offline algorithm, the set  $P$  must be specified in advance. It uses the `MakeSet`, `Find`, and `Union` functions of a disjoint-set forest. `MakeSet(u)` removes  $u$  to a singleton set, `Find(u)` returns the standard representative of the set containing  $u$ , and `Union(u, v)` merges the set containing  $u$  with the set containing  $v$ . `TarjanOLCA(r)` is first called on the root  $r$ .

```

function TarjanOLCA(u)
    MakeSet(u);
    u.ancestor := u;

```

---

```

for each v in u.children do
    TarjanOLCA(v);
    Union(u,v);
    Find(u).ancestor := u;
u.colour := black;
for each v such that {u,v} in P do
    if v.colour == black
        print "Tarjan's Lowest Common Ancestor of " + u +
            " and " + v + " is " + Find(v).ancestor + ";

```

Each node is initially white, and is colored black after it and all its children have been visited. The lowest common ancestor of the pair  $u, v$  is available as  $\text{Find}(v).\text{ancestor}$  immediately (and only immediately) after  $u$  is colored black, provided  $v$  is already black. Otherwise, it will be available later as  $\text{Find}(u).\text{ancestor}$ , immediately after  $v$  is colored black.

```

function MakeSet(x)

```

```

    x.parent := x
    x.rank  := 0

```

```

function Union(x, y)

```

```

    xRoot := Find(x)
    yRoot := Find(y)
    if xRoot.rank > yRoot.rank
        yRoot.parent := xRoot
    else if xRoot.rank < yRoot.rank
        xRoot.parent := yRoot
    else if xRoot != yRoot
        yRoot.parent := xRoot
        xRoot.rank := xRoot.rank + 1

```

```

function Find(x)

```

```

    if x.parent == x
        return x
    else
        x.parent := Find(x.parent)
        return x.parent

```