ACMICPC Standard Code Library

Beijing University of Posts and Telecommunications

April 29, 2016

Contents

1 Math

1.1 fft

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

const double PI = acos(-1);
struct Complex {
  double x, y;
  Complex() {
    x = 0;
    y = 0;
  }
  Complex(double _x, double _y) {
    x = _x;
    y = _y;
  }
  Complex operator-(const Complex &b) const {
    return Complex(x - b.x, y - b.y);
  }
  Complex operator+(const Complex &b) const {
    return Complex(x + b.x, y + b.y);
  }
  Complex operator*(const Complex &b) const {
    return Complex(x * b.x - y * b.y, x * b.y + y * b.x);
  }
};
void change(Complex y[], int len) {
  for (int i = 1, j = len / 2; i < len - 1; i++) {
    if (i < j) {
      swap(y[i], y[j]);
    }

    int k = len / 2;

    while (j >= k) {
      j -= k;
      k /= 2;
    }

    if (j < k) {
      j += k;
    }
  }
}
void fft(Complex y[], int len, int on) {
  change(y, len);

  for (int h = 2; h <= len; h <<= 1) {
    Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));

    for (int j = 0; j < len; j += h) {
      Complex w(1, 0);

      for (int k = j; k < j + h / 2; k++) {
        Complex u = y[k];
        Complex t = w * y[k + h / 2];
        y[k] = u + t;
        y[k + h / 2] = u - t;
        w = w * wn;
      }
    }
  }

  if (on == -1) {
    for (int i = 0; i < len; i++) {
      y[i].x /= len;
    }
```

```
66   }
67 }
```

2 String

2.1 sa

```
1  int r[maxn], wa[maxn], wb[maxn], wv[maxn], ws[maxn], sa[maxn];
2  int rank[maxn], height[maxn];
3
4  /*
5   *       nn  -10
6   *         sasa[1]~sa[n]
7   *       rankrank[0]~rank[n  -1],
8   *    heightii       -12~  nsa
9   */
10
11 inline bool cmp(int *r, int a, int b, int l) {
12   return r[a] == r[b] && r[a + l] == r[b + l];
13 }
14
15 void da(int n, int m) {
16   int i, j, p, *x = wa, *y = wb;
17
18   for (i = 0; i < m; i++) {
19     ws[i] = 0;
20   }
21
22   for (i = 0; i < n; i++) {
23     ws[x[i] = r[i]]++;
24   }
25
26   for (i = 1; i < m; i++) {
27     ws[i] += ws[i - 1];
28   }
29
30   for (i = n - 1; i >= 0; i--) {
31     sa[--ws[x[i]]] = i;
32   }
33
34   for (j = 1, p = 1; p < n; j <<= 1, m = p) {
35     for (p = 0, i = n - j; i < n; i++) {
36       y[p++] = i;
37     }
38
39     for (i = 0; i < n; i++)
40       if (sa[i] >= j) {
41         y[p++] = sa[i] - j;
42       }
43
44     for (i = 0; i < n; i++) {
45       wv[i] = x[y[i]];
46     }
47
48     for (i = 0; i < m; i++) {
49       ws[i] = 0;
50     }
51
52     for (i = 0; i < n; i++) {
53       ws[wv[i]]++;
54     }
55
56     for (i = 1; i < m; i++) {
57       ws[i] += ws[i - 1];
58     }
59
60     for (i = n - 1; i >= 0; i--) {
61       sa[--ws[wv[i]]] = y[i];
62     }
```

```
63
64      swap(x, y);
65
66      for (p = 1, x[sa[0]] = 0, i = 1; i < n; i++) {
67        x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
68      }
69    }
70
71    return;
72  }
73
74  void calheitght(int n) {
75    int i, j, k = 0;
76
77    for (i = 1; i < n; i++) {
78      rank[sa[i]] = i;
79    }
80
81    // print(rank, n);
82    for (i = 0; i < n; height[rank[i++]] = k)
83      for (k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++)
84        {}
85
86    return;
87  }
```

2.2 sam

```
1   void copy(int x, int y) {
2     pre[x] = pre[y];
3     len[x] = len[y];
4     memcpy(son[x], son[y], sizeof son[0]);
5   }
6
7   void insert(int c, int l) {
8     int p = tail, np = ++tot;
9     len[np] = l;
10    tail = np;
11
12    while (p && son[p][c] == 0) {
13      son[p][c] = np, p = pre[p];
14    }
15
16    if (p == 0) {
17      pre[np] = root;
18    } else {
19      int q = son[p][c];
20
21      if (len[p] + 1 == len[q]) {
22        pre[np] = q;
23      } else {
24        int nq = ++tot;
25        copy(nq, q);
26        len[nq] = len[p] + 1;
27        pre[np] = pre[q] = nq;
28
29        while (p && son[p][c] == q) {
30          son[p][c] = nq, p = pre[p];
31        }
32      }
33    }
34  }
35
36  void build(int n) {
37    for (int i = 1; i <= tot; i++) {
38      cnt[len[i]]++;
39    }
40
41    for (int i = 1; i <= n; i++) {
```

```
42      cnt[i] += cnt[i - 1];
43    }
44
45    for (int i = 1; i <= tot; i++) {
46      b[--cnt[len[i]]] = i;
47    }
48
49    for (int i = tot - 1; i >= 0; i--) {
50      int p = b[i], k = 0;
51      g[p] = 1;
52
53      for (int j = 0; j < 26; j++)
54        if (son[p][j]) {
55          int v = son[p][j];
56          g[p] += g[v];
57          son[p][k] = v;
58          gao[v] = j + 'a';
59          k++;
60        }
61
62      son[p][k] = 0;
63    }
64  }
```

2.3 kmp

```
1   void getFail() {
2     m = strlen(s);
3     f[0] = f[1] = 0;
4
5     for (int i = 1; i < m; i++) {
6       int j = f[i];
7
8       while (j && s[i] != s[j]) {
9         j = f[j];
10      }
11
12      f[i + 1] = s[i] == s[j] ? j + 1 : 0;
13    }
14  }
```

2.4 ac

```
1   void init() {
2     sz = 1;
3     ch[0].init();
4   }
5   void build() {
6     queue<int> q;
7     ch[0].fail = 0;
8
9     for (int c = 0; c < 4; c++) {
10      int u = ch[0].next[c];
11
12      if (u) {
13        ch[u].fail = 0;
14        q.push(u);
15      }
16    }
17
18    while (!q.empty()) {
19      int r = q.front();
20      q.pop();
21
22      for (int c = 0; c < 4; c++) {
23        int u = ch[r].next[c];
24
25        if (!u) {
26          ch[r].next[c] = ch[ch[r].fail].next[c];
```

```
27          continue;
28        }
29
30        q.push(u);
31        int v = ch[r].fail;
32
33        while (v && !ch[v].next[c]) {
34          v = ch[v].fail;
35        }
36
37        ch[u].fail = ch[v].next[c];
38        ch[u].isend |= ch[ch[u].fail].isend;
39      }
40    }
41  }
```

3 Geometry

3.1 c2c

```
1  Point rotate(const Point &p, double cost, double sint) {
2    double x = p.x, y = p.y;
3    return Point(x * cost - y * sint, x * sint + y * cost);
4  }
5
6  void circle_cross_circle(Circle a, Circle b, Point cro[]) {
7    double d = (a.o - b.o).len();
8    double cost = (a.r * a.r + d * d - b.r * b.r) / (2 * a.r * d);
9    double sint = sqrt(max(1.0 - cost * cost, 0.0));
10   Point v = (b.o - a.o) * (a.r / d);
11   cro[0] = a.o + rotate(v, cost, -sint);
12   cro[1] = a.o + rotate(v, cost, sint);
13 }
```

3.2 c2l

```
1  Point crosspt(const Point &a, const Point &b, const Point &p, const Point &q) {
2    double a1 = (b - a) * (p - a);
3    double a2 = (b - a) * (q - a);
4    return (p * a2 - q * a1) / (a2 - a1);
5  }
6  double sector_area(const Point &a, const Point &b) {
7    double theta = atan2(a.y, a.x) - atan2(b.y, b.x);
8
9    while (theta <= 0) {
10     theta += 2 * PI;
11   }
12
13   while (theta > 2 * PI) {
14     theta -= 2 * PI;
15   }
16
17   theta = min(theta, 2 * PI - theta);
18   return r * r * theta / 2;
19 }
20 double sqr(double x) { return x * x; }
21 void circle_cross_line(Point a, Point b, Point o, double r, Point ret[],
22                        const int &num) {
23   double x0 = o.x, y0 = o.y;
24   double x1 = a.x, y1 = a.y;
25   double x2 = b.x, y2 = b.y;
26   double dx = x2 - x1, dy = y2 - y1;
27   double A = dx * dx + dy * dy;
28   double B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
29   double C = sqr(x1 - x0) + sqr(y1 - y0) - sqr(r);
30   double delta = B * B - 4 * A * C;
31   num = 0;
32
33   if (dlcmp(delta) >= 0) {
```

```
34      double t1 = (-B - sqrt(max(delta, 0.0))) / (2 * A);
35      double t2 = (-B + sqrt(max(delta, 0.0))) / (2 * A);
36
37      if (dlcmp(t1 - 1) <= 0 && dlcmp(t1) >= 0) {
38        ret[num++] = Point(x1 + t1 * dx, y1 + t1 * dy);
39      }
40
41      if (dlcmp(t2 - 1) <= 0 && dlcmp(t2) >= 0) {
42        ret[num++] = Point(x1 + t2 * dx, y1 + t2 * dy);
43      }
44    }
45  }
46  double calc(const Point &a, const Point &b) {
47    Point p[2];
48    int num = 0;
49    int ina = dlcmp(a.len() - r) < 0;
50    int inb = dlcmp(b.len() - r) < 0;
51
52    if (ina) {
53      if (inb) {
54        return fabs(a * b) / 2;
55      } else {
56        circle_cross_line(a, b, Point(0, 0), r, p, num);
57        return sector_area(b, p[0]) + fabs(a * p[0]) / 2;
58      }
59    } else {
60      if (inb) {
61        circle_cross_line(a, b, Point(0, 0), r, p, num);
62        return sector_area(p[0], a) + fabs(p[0] * b) / 2;
63      } else {
64        circle_cross_line(a, b, Point(0, 0), r, p, num);
65
66        if (false) {
67          return sector_area(a, p[0]) + sector_area(p[1], b) +
68                 fabs(p[0] * p[1]) / 2;
69        } else {
70          return sector_area(a, b);
71        }
72      }
73    }
74  }
75  double area() {
76    double ret = 0;
77
78    for (int i = 0; i < n; i++) {
79      int sgn = dlcmp(res[i] * res[i + 1]);
80
81      if (sgn != 0) {
82        ret += sgn * calc(res[i], res[i + 1]);
83      }
84    }
85
86    return ret;
87  }
```

3.3 halfplaneintersection

```
1  struct Halfplane {
2    Point a, b;
3    Halfplane() {}
4    Halfplane(Point a, Point b) : a(a), b(b) {}
5
6    bool satisfy(const Point &rhs) const { return sgn((rhs - a) * (b - a)) <= 0; }
7    bool operator<(const Halfplane &rhs) const {
8      int res = sgn((b - a).arg() - (rhs.b - rhs.a).arg());
9      return res == 0 ? rhs.satisfy(a) : res < 0;
10   }
11 };
```

```
12
13  Point crosspoint(const Halfplane &a, const Halfplane &b) {
14    double k = (b.a - b.b) * (a.a - b.b);
15    k = k / (k - ((b.a - b.b) * (a.b - b.b)));
16    return a.a + (a.b - a.a) * k;
17  }
18
19  vector<Point> halfplaneIntersection(vector<Halfplane> v) {
20    sort(v.begin(), v.end());
21    deque<Halfplane> q;
22    deque<Point> ans;
23    q.push_back(v[0]);
24
25    for (int i = 1; i < v.size(); i++) {
26      if (sgn((v[i - 1].b - v[i - 1].a) * (v[i].b - v[i].a)) == 0) {
27        continue;
28      }
29
30      while (ans.size() > 0 && !v[i].satisfy(ans.back())) {
31        ans.pop_back();
32        q.pop_back();
33      }
34
35      while (ans.size() > 0 && !v[i].satisfy(ans.front())) {
36        ans.pop_front();
37        q.pop_front();
38      }
39
40      ans.push_back(crosspoint(q.back(), v[i]));
41      q.push_back(v[i]);
42    }
43
44    while (ans.size() > 0 && !q.front().satisfy(ans.back())) {
45      ans.pop_back();
46      q.pop_back();
47    }
48
49    while (ans.size() > 0 && !q.back().satisfy(ans.front())) {
50      ans.pop_front();
51      q.pop_front();
52    }
53
54    ans.push_back(crosspoint(q.back(), q.front()));
55    return vector<Point>(ans.begin(), ans.end());
56  }
57
58  double area(const vector<Point> &p, int ansi) {
59    double res = 0;
60
61    for (int i = ansi; i + 1 < p.size(); i++) {
62      res += p[i] * p[i + 1];
63    }
64
65    res += p.back() * p[ansi];
66    return fabs(res) / 2;
67  }
68
69  double ptol(Point a, Point b, Point c) {
70    double are = fabs((b - a) * (c - a));
71    return are / (b - c).len();
72  }
73
74  int main() {
75    int T_T, n, nc = 0;
76    cin >> T_T;
77    Point __0(0, 0), __1(1, 0), __2(1, 1), __3(0, 1);
78
79    while (T_T--) {
80      printf("Case #%d:\n", ++nc);
```

```
81      scanf("%d", &n);
82
83      for (int i = 0; i < n; i++) {
84        p[i].input();
85      }
86
87      for (int i = 0; i < n; i++) {
88        vector<Halfplane> v;
89        v.push_back(Halfplane(__0, __1));
90        v.push_back(Halfplane(__1, __2));
91        v.push_back(Halfplane(__2, __3));
92        v.push_back(Halfplane(__3, __0));
93
94        for (int j = 0; j < n; j++)
95          if (i != j) {
96            Point a = (p[i] + p[j]) / 2;
97            Point b = a + (p[i] - p[j]).rev();
98
99            if (!Halfplane(a, b).satisfy(p[i])) {
100             swap(a, b);
101           }
102
103           v.push_back(Halfplane(a, b));
104         }
105
106       vector<Point> ans = halfplaneIntersection(v);
107
108       double ret = 0, low = 1e100;
109       int ansi = 0;
110
111       for (int j = 0; j < ans.size(); j++)
112         if (ans[j].z() < low) {
113           low = ans[j].z(), ansi = j;
114         }
115
116       for (int j = 0; j < ansi; j++) {
117         ans.push_back(ans[j]);
118       }
119
120       ret = area(ans, ansi) * low;
121
122       for (int j = ansi + 1; j + 1 < ans.size(); j++) {
123         double ll = (ans[j] - ans[j + 1]).len();
124
125         if (ll < eps) {
126           continue;
127         }
128
129         double s = (ans[j].z() + ans[j + 1].z() - low * 2) * ll / 2;
130         double h = ptol(ans[ansi], ans[j], ans[j + 1]);
131         ret += s * h / 3;
132       }
133
134       printf("%.6f\n", ret);
135     }
136   }
137
138   return 0;
139 }
```

4 DataStruct

4.1 lct

```
1  int ch[MAXN][2], pre[MAXN], key[MAXN];
2  int add[MAXN], Max[MAXN], rev[MAXN], n;
3  bool rt[MAXN];
4  void update_add(int r, int d) {
5    if (!r) {
6      return;
```

```
7     }
8
9     key[r] += d;
10    add[r] += d;
11    Max[r] += d;
12  }
13  void update_rev(int r) {
14    if (!r) {
15      return;
16    }
17
18    swap(ch[r][0], ch[r][1]);
19    rev[r] ^= 1;
20  }
21  void push_down(int r) {
22    if (add[r]) {
23      update_add(ch[r][0], add[r]);
24      update_add(ch[r][1], add[r]);
25      add[r] = 0;
26    }
27
28    if (rev[r]) {
29      update_rev(ch[r][0]);
30      update_rev(ch[r][1]);
31      rev[r] = 0;
32    }
33  }
34  void display() {
35    for (int i = 1; i <= n; i++) {
36      printf("%d %d %d %d <%d>  ", i, ch[i][0], ch[i][1], pre[i], rt[i]);
37      printf("%d %d %d\n", add[i], key[i], Max[i]);
38    }
39  }
40  void push_up(int r) { Max[r] = max(max(Max[ch[r][0]], Max[ch[r][1]]), key[r]); }
41  void rotate(int x) {
42    int y = pre[x], kind = ch[y][1] == x;
43    ch[y][kind] = ch[x][!kind];
44    pre[ch[y][kind]] = y;
45    pre[x] = pre[y];
46    pre[y] = x;
47    ch[x][!kind] = y;
48
49    if (rt[y]) {
50      rt[y] = 0, rt[x] = 1;
51    } else {
52      ch[pre[x]][ch[pre[x]][1] == y] = x;
53    }
54
55    push_up(y);
56  }
57  void P(int r) {
58    if (!rt[r]) {
59      P(pre[r]);
60    }
61
62    push_down(r);
63  }
64  void splay(int r) {
65    P(r);
66
67    while (!rt[r]) {
68      int f = pre[r], ff = pre[f];
69
70      if (rt[f]) {
71        rotate(r);
72      } else if ((ch[ff][1] == f) == (ch[f][1] == r)) {
73        rotate(f), rotate(r);
```

```
74      } else {
75        rotate(r), rotate(r);
76      }
77    }
78
79    push_up(r);
80  }
81  int access(int x) {
82    int y = 0;
83
84    for (; x; x = pre[y = x]) {
85      splay(x);
86      rt[ch[x][1]] = 1, rt[ch[x][1] = y] = 0;
87      push_up(x);
88    }
89
90    return y;
91  }
92  bool judge(int u, int v) {
93    while (pre[u]) {
94      u = pre[u];
95    }
96
97    while (pre[v]) {
98      v = pre[v];
99    }
100
101   return u == v;
102 }
103 void mroot(int r) {
104   access(r);
105   splay(r);
106   update_rev(r);
107 }
108 void lca(const int &u, const int &v) {
109   access(v), v = 0;
110
111   // puts("--------------");display();
112   while (u) {
113     splay(u);
114
115     if (!pre[u]) {
116       return;
117     }
118
119     rt[ch[u][1]] = 1;
120     rt[ch[u][1] = v] = 0;
121     push_up(u);
122     u = pre[v = u];
123   }
124 }
125 void link(int u, int v) {
126   if (judge(u, v)) {
127     puts("-1");
128     return;
129   }
130
131   mroot(u);
132   pre[u] = v;
133 }
134 void cut(int u, int v) {
135   if (u == v || !judge(u, v)) {
136     puts("-1");
137     return;
138   }
139
140   mroot(u);
141   splay(v);
```

```
142     pre[ch[v][0]] = pre[v];
143     pre[v] = 0;
144     rt[ch[v][0]] = 1;
145     ch[v][0] = 0;
146     push_up(v);
147   }
148   void ADD(int u, int v, int w) {
149     if (!judge(u, v)) {
150       puts("-1");
151       return;
152     }
153
154     lca(u, v);
155     update_add(ch[u][1], w);
156     update_add(v, w);
157     key[u] += w;
158     push_up(u);
159   }
160   void query(int u, int v) {
161     if (!judge(u, v)) {
162       puts("-1");
163       return;
164     }
165
166     lca(u, v);
167     printf("%d\n", max(max(Max[v], Max[ch[u][1]]), key[u]));
168   }
169   vector<int> G[MAXN];
170   int que[MAXN];
171   void bfs() {
172     int front = 0, rear = 0;
173     que[rear++] = 1;
174     pre[1] = 0;
175
176     while (front < rear) {
177       int u = que[front++];
178
179       for (int i = 0; i < G[u].size(); i++) {
180         int v = G[u][i];
181
182         if (v == pre[u]) {
183           continue;
184         }
185
186         pre[v] = u;
187         que[rear++] = v;
188       }
189     }
190   }
191   int main() {
192     int q, u, v;
193
194     while (~scanf("%d", &n)) {
195       memset(add, 0, sizeof add);
196       memset(pre, 0, sizeof pre);
197       memset(rev, 0, sizeof rev);
198       memset(ch, 0, sizeof ch);
199
200       for (int i = 0; i <= n; i++) {
201         G[i].clear();
202         rt[i] = 1;
203       }
204
205       Max[0] = -INF;
206
207       for (int i = 1; i < n; i++) {
208         scanf("%d%d", &u, &v);
209         G[u].push_back(v);
```

```
210       G[v].push_back(u);
211     }
212
213     for (int i = 1; i <= n; i++) {
214       scanf("%d", &key[i]);
215       Max[i] = key[i];
216     }
217
218     scanf("%d", &q);
219     bfs();
220
221     int op, x, y, w;
222
223     while (q--) {
224       scanf("%d", &op);
225
226       if (op == 1) {
227         scanf("%d%d", &x, &y);
228         link(x, y);
229       } else if (op == 2) {
230         scanf("%d%d", &x, &y);
231         cut(x, y);
232       } else if (op == 3) {
233         scanf("%d%d%d", &w, &x, &y);
234         ADD(x, y, w);
235       } else {
236         scanf("%d%d", &x, &y);
237         query(x, y);
238       }
239
240       // display();
241     }
242
243     puts("");
244   }
245
246   return 0;
247 }
```

4.2 kdt

```
1  bool cmpx(const Node &a, const Node &b) { return a.x < b.x; }
2  bool cmpy(const Node &a, const Node &b) { return a.y < b.y; }
3
4  LL dis(const Node &a, const Node &b) { return sqr(a.x - b.x) + sqr(a.y - b.y); }
5
6  void build(int l, int r) {
7    if (l > r) {
8      return;
9    }
10
11   LL minx = min_element(p + l, p + r + 1, cmpx)->x;
12   LL maxx = max_element(p + l, p + r + 1, cmpx)->x;
13   LL miny = min_element(p + l, p + r + 1, cmpy)->y;
14   LL maxy = max_element(p + l, p + r + 1, cmpy)->y;
15   int mid = l + (r - l) / 2;
16   d[mid] = maxx - minx > maxy - miny;
17   nth_element(p + l, p + mid, p + r + 1, d[mid] ? cmpx : cmpy);
18
19   build(l, mid - 1);
20   build(mid + 1, r);
21 }
22
23 void query(int l, int r, const Node &a) {
24   if (l > r) {
25     return;
26   }
27
28   int mid = l + (r - l) / 2;
```

```
29    LL dist = dis(a, p[mid]);
30    LL d1 = d[mid] ? a.x - p[mid].x : a.y - p[mid].y;
31
32    if (dist > 0) {
33      res = min(res, dist);
34    }
35
36    int l1 = l, r1 = mid - 1;
37    int l2 = mid + 1, r2 = r;
38
39    if (d1 > 0) {
40      swap(l1, l2);
41      swap(r1, r2);
42    }
43
44    query(l1, r1, a);
45
46    if (d1 * d1 < res) {
47      query(l2, r2, a);
48    }
49  }
```

5 Graph

5.1 targan point connecting

```
1   void Tarjan(int u, int pre) {
2     Low[u] = DFN[u] = ++Index;
3     Stack[top++] = u;
4     Instack[u] = true;
5
6     for (int i = head[u]; i != -1; i = edge[i].next) {
7       int v = edge[i].to;
8
9       if (v == pre) {
10        continue;    //
11      }
12
13      if (!DFN[v]) {
14        Tarjan(v, u);
15
16        if (Low[u] > Low[v]) {
17          Low[u] = Low[v];
18        }
19        /*
20        if (Low[v] >= DFN[u]) {
21          block++;
22          int vn;
23          cc = 0;
24          memset(ok, false, sizeof(ok));
25
26          do {
27            vn = Stack[--top];
28            Belong[vn] = block;
29            Instack[vn] = false;
30            ok[vn] = true;
31            tmp[cc++] = vn;
32          } while (vn != v);
33
34          ok[u] = 1;
35          memset(color, -1, sizeof(color));
36
37          if (!dfs(u, 0)) {
38            can[u] = true;
39
40            while (cc--) {
41              can[tmp[cc]] = true;
42            }
43          }
```

```
44        }
45        */
46      } else if (Instack[v] && Low[u] > DFN[v]) {
47        Low[u] = DFN[v];
48      }
49    }
50
51    /*  targan
52    if (Low[u] == DFN[u]) {
53      scc++;
54
55      do {
56        v = Stack[--top];
57        Instack[v] = false;
58        Belong[v] = scc;
59        num[scc]++;
60      } while (v != u);
61    }
62    */
63  }
```

5.2 cut point bridge

```
1   const int MAXN = 10010;
2   const int MAXM = 100010;
3   struct Edge {
4     int to, next;
5     bool cut;  //
6   } edge[MAXM];
7   int head[MAXN], tot;
8   int Low[MAXN], DFN[MAXN], Stack[MAXN];
9   int Index, top;
10  bool Instack[MAXN];
11  bool cut[MAXN];
12  int add_block[MAXN];  //
13  int bridge;
14  void addedge(int u, int v) {
15    edge[tot].to = v;
16    edge[tot].next = head[u];
17    edge[tot].cut = false;
18    head[u] = tot++;
19  }
20  void Tarjan(int u, int pre) {
21    Low[u] = DFN[u] = ++Index;
22    Stack[top++] = u;
23    Instack[u] = true;
24    int son = 0;
25
26    for (int i = head[u]; i != -1; i = edge[i].next) {
27      int v = edge[i].to;
28
29      if (v == pre) {
30        continue;
31      }
32
33      if (!DFN[v]) {
34        son++;
35        Tarjan(v, u);
36
37        if (Low[u] > Low[v]) {
38          Low[u] = Low[v];
39        }
40
41        if (Low[v] > DFN[u]) {  //
42          bridge++;
43          edge[i].cut = true;
44          edge[i ^ 1].cut = true;
45        }
46
```

```
47          if (u != pre && Low[v] >= DFN[u]) {   //
48            cut[u] = true;
49            add_block[u]++;
50          }
51        } else if (Low[u] > DFN[v]) {
52          Low[u] = DFN[v];
53        }
54
55        /*
56         * else if( Instack[v] && Low[u] > DFN[v] )
57         *     Low[u] = DFN[v];
58         * }
59         * if(Low[u] == DFN[u]){
60         *     block++;
61         *     do
62         *     {
63         *         v = Stack[--top];
64         *         Instack[v] = false;
65         *         Belong[v] = block;
66         *     }while( v!=u );
67         * }
68         */
69    }
70
71    if (u == pre && son > 1) {
72      cut[u] = true;  //     ,1
73    }
74
75    if (u == pre) {
76      add_block[u] = son - 1;
77    }
78
79    Instack[u] = false;
80    top--;
81  }
82  void solve(int N) {
83    memset(DFN, 0, sizeof(DFN));
84    memset(Instack, false, sizeof(Instack));
85    memset(add_block, 0, sizeof(add_block));
86    memset(cut, false, sizeof(cut));
87    Index = top = 0;
88    bridge = 0;
89
90    for (int i = 1; i <= N; i++)
91      if (!DFN[i]) {
92        Tarjan(i, i);
93      }
94
95    printf("%d critical links\n", bridge);
96  }
97  void init() {
98      tot = 0;
99      memset(head, -1, sizeof(head));
100 }
```

5.3 hungary

```
1   bool dfs(int u) {
2     for (int i = head[u]; i != -1; i = edge[i].next) {
3       int v = edge[i].to;
4
5       if (!used[v]) {
6         used[v] = true;
7
8         if (linker[v] == -1 || dfs(linker[v])) {
9           linker[v] = u;
10          return true;
11        }
12      }
```

```
13    }
14
15    return false;
16  }
17  int hungary() {
18    memset(linker, -1, sizeof(linker));
19
20    for (int u = 0; u < uN; u++) {  //   0~uN-1
21      memset(used, false, sizeof(used));
22
23      if (dfs(u)) {
24        res++;
25      }
26    }
27
28    return res;
29  }
```

5.4 maxflow

```
1  #include <bits/stdc++.h>
2  using std;
3
4  const int MAXN = 100010;  //
5  const int MAXM = 400010;  //
6  const int oo = 0x3f3f3f3f;
7  struct Edge {
8    int to, next, cap, flow;
9  } edge[MAXM];  //  MAXM
10 int tol;
11 int head[MAXN];
12 int gap[MAXN], dep[MAXN], cur[MAXN];
13 void init() {
14   tol = 0;
15   memset(head, -1, sizeof(head));
16 }
17 void addedge(int u, int v, int w, int rw = 0) {
18   edge[tol].to = v;
19   edge[tol].cap = w;
20   edge[tol].flow = 0;
21   edge[tol].next = head[u];
22   head[u] = tol++;
23   edge[tol].to = u;
24   edge[tol].cap = rw;
25   edge[tol].flow = 0;
26   edge[tol].next = head[v];
27   head[v] = tol++;
28 }
29 int Q[MAXN];
30 void BFS(int ss, int tt) {
31   memset(dep, -1, sizeof(dep));
32   memset(gap, 0, sizeof(gap));
33   gap[0] = 1;
34   int front = 0, rear = 0;
35   dep[tt] = 0;
36   Q[rear++] = tt;
37
38   while (front != rear) {
39     int u = Q[front++];
40
41     for (int i = head[u]; i != -1; i = edge[i].next) {
42       int v = edge[i].to;
43
44       if (dep[v] != -1) {
45         continue;
46       }
47
48       Q[rear++] = v;
49       dep[v] = dep[u] + 1;
```

```
50        gap[dep[v]]++;
51      }
52    }
53  }
54  int S[MAXN];
55  int sap(int ss, int tt, int N) {
56    BFS(ss, tt);
57    memcpy(cur, head, sizeof(head));
58    int top = 0;
59    int u = ss;
60    int ans = 0;
61
62    while (dep[ss] < N) {
63      if (u == tt) {
64        int mi = oo;
65        int inser;
66
67        for (int i = 0; i < top; i++)
68          if (mi > edge[S[i]].cap - edge[S[i]].flow) {
69            mi = edge[S[i]].cap - edge[S[i]].flow;
70            inser = i;
71          }
72
73        for (int i = 0; i < top; i++) {
74          edge[S[i]].flow += mi;
75          edge[S[i] ^ 1].flow -= mi;
76        }
77
78        ans += mi;
79        top = inser;
80        u = edge[S[top] ^ 1].to;
81        continue;
82      }
83
84      bool flag = false;
85      int v;
86
87      for (int i = cur[u]; i != -1; i = edge[i].next) {
88        v = edge[i].to;
89
90        if (edge[i].cap - edge[i].flow && dep[v] + 1 == dep[u]) {
91          flag = true;
92          cur[u] = i;
93          break;
94        }
95      }
96
97      if (flag) {
98        S[top++] = cur[u];
99        u = v;
100       continue;
101     }
102
103     int mi = N;
104
105     for (int i = head[u]; i != -1; i = edge[i].next)
106       if (edge[i].cap - edge[i].flow && dep[edge[i].to] < mi) {
107         mi = dep[edge[i].to];
108         cur[u] = i;
109       }
110
111     gap[dep[u]]--;
112
113     if (!gap[dep[u]]) {
114       return ans;
115     }
116
117     dep[u] = mi + 1;
118     gap[dep[u]]++;
119
```

```
120     if (u != ss) {
121       u = edge[S[--top] ^ 1].to;
122     }
123   }
124
125   return ans;
126 }
```

5.5 costflow

```
1  const int MAXN = 10000;
2  const int MAXM = 100000;
3  const int INF = 0x3f3f3f3f;
4  struct Edge {
5    int to, next, cap, flow, cost;
6  } edge[MAXM];
7  int head[MAXN], tol;
8  int pre[MAXN], dis[MAXN];
9  bool vis[MAXN];
10 int N;  //      ,0~N-1
11 void init(int n) {
12   N = n;
13   tol = 0;
14   memset(head, -1, sizeof(head));
15 }
16 void addedge(int u, int v, int cap, int cost) {
17   edge[tol].to = v;
18   edge[tol].cap = cap;
19   edge[tol].cost = cost;
20   edge[tol].flow = 0;
21   edge[tol].next = head[u];
22   head[u] = tol++;
23   edge[tol].to = u;
24   edge[tol].cap = 0;
25   edge[tol].cost = -cost;
26   edge[tol].flow = 0;
27   edge[tol].next = head[v];
28   head[v] = tol++;
29 }
30 bool spfa(int s, int t) {
31   queue<int> q;
32
33   for (int i = 0; i < N; i++) {
34     dis[i] = INF;
35     vis[i] = false;
36     pre[i] = -1;
37   }
38
39   dis[s] = 0;
40   vis[s] = true;
41   q.push(s);
42
43   while (!q.empty()) {
44     int u = q.front();
45     q.pop();
46     vis[u] = false;
47
48     for (int i = head[u]; i != -1; i = edge[i].next) {
49       int v = edge[i].to;
50
51       if (edge[i].cap > edge[i].flow && dis[v] > dis[u] + edge[i].cost) {
52         dis[v] = dis[u] + edge[i].cost;
53         pre[v] = i;
54
55         if (!vis[v]) {
56           vis[v] = true;
57           q.push(v);
58         }
59       }
```

```
60      }
61    }
62
63    if (pre[t] == -1) {
64      return false;
65    } else {
66      return true;
67    }
68  }
69  //     ,    cost
70  int minCostMaxflow(int s, int t, const int &cost) {
71    int flow = 0;
72    cost = 0;
73
74    while (spfa(s, t)) {
75      int Min = INF;
76
77      for (int i = pre[t]; i != -1; i = pre[edge[i ^ 1].to]) {
78        if (Min > edge[i].cap - edge[i].flow) {
79          Min = edge[i].cap - edge[i].flow;
80        }
81      }
82
83      for (int i = pre[t]; i != -1; i = pre[edge[i ^ 1].to]) {
84        edge[i].flow += Min;
85        edge[i ^ 1].flow -= Min;
86        cost += edge[i].cost * Min;
87      }
88
89      flow += Min;
90    }
91
92    return flow;
93  }
```

5.6 min tree graph

```
1   const int INF = 0x3f3f3f3f;
2   const int MAXN = 1010;
3   const int MAXM = 40010;
4   struct Edge {
5     int u, v, cost;
6   };
7   Edge edge[MAXM];
8   int pre[MAXN], id[MAXN], visit[MAXN], in[MAXN];
9   int zhuliu(int root, int n, int m, Edge edge[]) {
10    int res = 0, u, v;
11
12    while (1) {
13      for (int i = 0; i < n; i++) {
14        in[i] = INF;
15      }
16
17      for (int i = 0; i < m; i++)
18        if (edge[i].u != edge[i].v && edge[i].cost < in[edge[i].v]) {
19          pre[edge[i].v] = edge[i].u;
20          in[edge[i].v] = edge[i].cost;
21        }
22
23      for (int i = 0; i < n; i++)
24        if (i != root && in[i] == INF) {
25          return -1;      //
26        }
27
28      int tn = 0;
29      memset(id, -1, sizeof(id));
30      memset(visit, -1, sizeof(visit));
31      in[root] = 0;
32
33      for (int i = 0; i < n; i++) {
```

```
34        res += in[i];
35        v = i;
36
37        while (visit[v] != i && id[v] == -1 && v != root) {
38          visit[v] = i;
39          v = pre[v];
40        }
41
42        if (v != root && id[v] == -1) {
43          for (int u = pre[v]; u != v; u = pre[u]) {
44            id[u] = tn;
45          }
46
47          id[v] = tn++;
48        }
49      }
50
51      if (tn == 0) {
52        break;      //
53      }
54
55      for (int i = 0; i < n; i++)
56        if (id[i] == -1) {
57          id[i] = tn++;
58        }
59
60      for (int i = 0; i < m;) {
61        v = edge[i].v;
62        edge[i].u = id[edge[i].u];
63        edge[i].v = id[edge[i].v];
64
65        if (edge[i].u != edge[i].v) {
66          edge[i++].cost -= in[v];
67        } else {
68          swap(edge[i], edge[--m]);
69        }
70      }
71
72      n = tn;
73      root = id[root];
74    }
75
76    return res;
77  }
```

5.7 flowertree

```
1   const int MAXN = 250;
2   int N;  //       ,1N
3   bool Graph[MAXN][MAXN];
4   int Match[MAXN];
5   bool InQueue[MAXN], InPath[MAXN], InBlossom[MAXN];
6   int Head, Tail;
7   int Queue[MAXN];
8   int Start, Finish;
9   int NewBase;
10  int Father[MAXN], Base[MAXN];
11  int Count;  //      ,Count/2
12  void CreateGraph() {
13    int u, v;
14    memset(Graph, false, sizeof(Graph));
15    scanf("%d", &N);
16
17    while (scanf("%d%d", &u, &v) == 2) {
18      Graph[u][v] = Graph[v][u] = true;
19    }
20  }
21  void Push(int u) {
22    Queue[Tail] = u;
```

```
23    Tail++;
24    InQueue[u] = true;
25  }
26  int Pop() {
27    int res = Queue[Head];
28    Head++;
29    return res;
30  }
31  int FindCommonAncestor(int u, int v) {
32    memset(InPath, false, sizeof(InPath));
33
34    while (true) {
35      u = Base[u];
36      InPath[u] = true;
37
38      if (u == Start) {
39        break;
40      }
41
42      u = Father[Match[u]];
43    }
44
45    while (true) {
46      v = Base[v];
47
48      if (InPath[v]) {
49        break;
50      }
51
52      v = Father[Match[v]];
53    }
54
55    return v;
56  }
57  void ResetTrace(int u) {
58    while (Base[u] != NewBase) {
59      int v = Match[u];
60      InBlossom[Base[u]] = InBlossom[Base[v]] = true;
61      u = Father[v];
62
63      if (Base[u] != NewBase) {
64        Father[u] = v;
65      }
66    }
67  }
68  void BloosomContract(int u, int v) {
69    NewBase = FindCommonAncestor(u, v);
70    memset(InBlossom, false, sizeof(InBlossom));
71    ResetTrace(u);
72    ResetTrace(v);
73
74    if (Base[u] != NewBase) {
75      Father[u] = v;
76    }
77
78    if (Base[v] != NewBase) {
79      Father[v] = u;
80    }
81
82    for (int tu = 1; tu <= N; tu++)
83      if (InBlossom[Base[tu]]) {
84        Base[tu] = NewBase;
85
86        if (!InQueue[tu]) {
87          Push(tu);
88        }
89      }
90  }
```

```
91
92  void FindAugmentingPath() {
93    memset(InQueue, false, sizeof(InQueue));
94    memset(Father, 0, sizeof(Father));
95
96    for (int i = 1; i <= N; i++) {
97      Base[i] = i;
98    }
99
100   Head = Tail = 1;
101   Push(Start);
102   Finish = 0;
103
104   while (Head < Tail) {
105     int u = Pop();
106
107     for (int v = 1; v <= N; v++)
108       if (Graph[u][v] && (Base[u] != Base[v]) && (Match[u] != v)) {
109         if ((v == Start) || ((Match[v] > 0) && Father[Match[v]] > 0)) {
110           BloosomContract(u, v);
111         } else if (Father[v] == 0) {
112           Father[v] = u;
113
114           if (Match[v] > 0) {
115             Push(Match[v]);
116           } else {
117             Finish = v;
118             return;
119           }
120         }
121       }
122   }
123 }
124 void AugmentPath() {
125   int u = Finish;
126
127   while (u > 0) {
128     int v = Father[u];
129     int w = Match[v];
130     Match[v] = u;
131     Match[u] = v;
132     u = w;
133   }
134 }
135 void Edmonds() {
136   memset(Match, 0, sizeof(Match));
137
138   for (int u = 1; u <= N; u++)
139     if (Match[u] == 0) {
140       Start = u;
141       FindAugmentingPath();
142
143       if (Finish > 0) {
144         AugmentPath();
145       }
146     }
147 }
148 void PrintMatch() {
149   Count = 0;
150
151   for (int u = 1; u <= N; u++)
152     if (Match[u] > 0) {
153       Count++;
154     }
155
156   printf("%d\n", Count);
157
158   for (int u = 1; u <= N; u++)
159     if (u < Match[u]) {
```

```
160         printf("%d %d\n", u, Match[u]);
161       }
162   }
163   int main() {
164     CreateGraph();   //
165     Edmonds();       //
166     PrintMatch();    //
167     return 0;
168   }
```

5.8 2-sat

```
1   const int MAXN = 20020;
2   const int MAXM = 100010;
3   struct Edge {
4     int to, next;
5   } edge[MAXM];
6   int head[MAXN], tot;
7   void init() {
8     tot = 0;
9     memset(head, -1, sizeof(head));
10  }
11  void addedge(int u, int v) {
12    edge[tot].to = v;
13    edge[tot].next = head[u];
14    head[u] = tot++;
15  }
16  bool vis[MAXN];     //    , true
17  int S[MAXN], top;   //
18  bool dfs(int u) {
19    if (vis[u ^ 1]) {
20      return false;
21    }
22
23    if (vis[u]) {
24      return true;
25    }
26
27    vis[u] = true;
28    S[top++] = u;
29
30    for (int i = head[u]; i != -1; i = edge[i].next)
31      if (!dfs(edge[i].to)) {
32        return false;
33      }
34
35    return true;
36  }
37  bool Twosat(int n) {
38    memset(vis, false, sizeof(vis));
39
40    for (int i = 0; i < n; i += 2) {
41      if (vis[i] || vis[i ^ 1]) {
42        continue;
43      }
44
45      top = 0;
46
47      if (!dfs(i)) {
48        while (top) {
49          vis[S[--top]] = false;
50        }
51
52        if (!dfs(i ^ 1)) {
53          return false;
54        }
55      }
56    }
57
58    return true;
```

```
}
int main() {
  int n, m;
  int u, v;

  while (scanf("%d%d", &n, &m) == 2) {
    init();

    while (m--) {
      scanf("%d%d", &u, &v);
      u--;
      v--;
      addedge(u, v ^ 1);
      addedge(v, u ^ 1);
    }

    if (Twosat(2 * n)) {
      for (int i = 0; i < 2 * n; i++)
        if (vis[i]) {
          printf("%d\n", i + 1);
        }
    } else {
      printf("NIE\n");
    }
  }

  return 0;
}
```

5.9 km

```
bool DFS(int x) {
  visx[x] = true;

  for (int y = 0; y < ny; y++) {
    if (visy[y]) {
      continue;
    }

    int tmp = lx[x] + ly[y] - g[x][y];

    if (tmp == 0) {
      visy[y] = true;

      if (linker[y] == -1 || DFS(linker[y])) {
        linker[y] = x;
        return true;
      }
    } else if (slack[y] > tmp) {
      slack[y] = tmp;
    }
  }

  return false;
}
int KM() {
  memset(linker, -1, sizeof(linker));
  memset(ly, 0, sizeof(ly));

  for (int i = 0; i < nx; i++) {
    lx[i] = -INF;

    for (int j = 0; j < ny; j++)
      if (g[i][j] > lx[i]) {
        lx[i] = g[i][j];
      }
  }

  for (int x = 0; x < nx; x++) {
    for (int i = 0; i < ny; i++) {
```

```
40        slack[i] = INF;
41      }
42
43      while (true) {
44        memset(visx, false, sizeof(visx));
45        memset(visy, false, sizeof(visy));
46
47        if (DFS(x)) {
48          break;
49        }
50
51        int d = INF;
52
53        for (int i = 0; i < ny; i++)
54          if (!visy[i] && d > slack[i]) {
55            d = slack[i];
56          }
57
58        for (int i = 0; i < nx; i++)
59          if (visx[i]) {
60            lx[i] -= d;
61          }
62
63        for (int i = 0; i < ny; i++) {
64          if (visy[i]) {
65            ly[i] += d;
66          } else {
67            slack[i] -= d;
68          }
69        }
70      }
71    }
72
73    int res = 0;
74
75    for (int i = 0; i < ny; i++)
76      if (linker[i] != -1) {
77        res += g[linker[i]][i];
78      }
79
80    return res;
81  }
```