ACMICPC Standard Code Library

Beijing University of Posts and Telecommunications

April 29, 2016

Contents

1 Math **3**
1.1 fft . . . 3

2 String **4**
2.1 sa . . . 4
2.2 sam . . . 5
2.3 kmp . . . 6
2.4 ac . . . 6

3 Geometry **7**
3.1 c2c . . . 7
3.2 c2l . . . 7
3.3 halfplaneintersection . . . 9

4 DataStruct **11**
4.1 lct . . . 11
4.2 kdt . . . 14

5 Graph **15**
5.1 targan point connecting . . . 15
5.2 cut point bridge . . . 16
5.3 hungary . . . 18
5.4 maxflow . . . 18
5.5 costflow . . . 20
5.6 min tree graph . . . 21
5.7 flowertree . . . 23
5.8 2-sat . . . 25
5.9 km . . . 26

1 Math

1.1 fft

```
1  // Copyright [2017] <dmnsn7>
2
3  #include <bits/stdc++.h>
4  using std;
5
6  const double PI = acos(-1);
7  struct Complex {
8    double x, y;
9    Complex() {
10     x = 0;
11     y = 0;
12   }
13   Complex(double _x, double _y) {
14     x = _x;
15     y = _y;
16   }
17   Complex operator-(const Complex &b) const {
18     return Complex(x - b.x, y - b.y);
19   }
20   Complex operator+(const Complex &b) const {
21     return Complex(x + b.x, y + b.y);
22   }
23   Complex operator*(const Complex &b) const {
24     return Complex(x * b.x - y * b.y, x * b.y + y * b.x);
25   }
26 };
27 void change(Complex y[], int len) {
28   for (int i = 1, j = len / 2; i < len - 1; i++) {
29     if (i < j) {
30       swap(y[i], y[j]);
31     }
32
33     int k = len / 2;
34
35     while (j >= k) {
36       j -= k;
37       k /= 2;
38     }
39
40     if (j < k) {
41       j += k;
42     }
43   }
44 }
45 void fft(Complex y[], int len, int on) {
46   change(y, len);
47
48   for (int h = 2; h <= len; h <<= 1) {
49     Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
50
51     for (int j = 0; j < len; j += h) {
52       Complex w(1, 0);
53
54       for (int k = j; k < j + h / 2; k++) {
55         Complex u = y[k];
56         Complex t = w * y[k + h / 2];
57         y[k] = u + t;
58         y[k + h / 2] = u - t;
59         w = w * wn;
60       }
61     }
62   }
63
64   if (on == -1) {
65     for (int i = 0; i < len; i++) {
```

```
66        y[i].x /= len;
67      }
68    }
69  }
```

2 String

2.1 sa

```
1   // Copyright [2017] <dmnsn7>
2
3   int r[maxn], wa[maxn], wb[maxn], wv[maxn], ws[maxn], sa[maxn];
4   int rank[maxn], height[maxn];
5
6   /*
7    *       nn   -10
8    *         sasa[1]~sa[n]
9    *       rankrank[0]~rank[n   -1],
10   *    heightii        -12~  nsa
11   */
12
13  inline bool cmp(int *r, int a, int b, int l) {
14    return r[a] == r[b] && r[a + l] == r[b + l];
15  }
16
17  void da(int n, int m) {
18    int i, j, p, *x = wa, *y = wb;
19
20    for (i = 0; i < m; i++) {
21      ws[i] = 0;
22    }
23
24    for (i = 0; i < n; i++) {
25      ws[x[i] = r[i]]++;
26    }
27
28    for (i = 1; i < m; i++) {
29      ws[i] += ws[i - 1];
30    }
31
32    for (i = n - 1; i >= 0; i--) {
33      sa[--ws[x[i]]] = i;
34    }
35
36    for (j = 1, p = 1; p < n; j <<= 1, m = p) {
37      for (p = 0, i = n - j; i < n; i++) {
38        y[p++] = i;
39      }
40
41      for (i = 0; i < n; i++)
42        if (sa[i] >= j) {
43          y[p++] = sa[i] - j;
44        }
45
46      for (i = 0; i < n; i++) {
47        wv[i] = x[y[i]];
48      }
49
50      for (i = 0; i < m; i++) {
51        ws[i] = 0;
52      }
53
54      for (i = 0; i < n; i++) {
55        ws[wv[i]]++;
56      }
57
58      for (i = 1; i < m; i++) {
59        ws[i] += ws[i - 1];
60      }
```

```
61
62      for (i = n - 1; i >= 0; i--) {
63        sa[--ws[wv[i]]] = y[i];
64      }
65
66      swap(x, y);
67
68      for (p = 1, x[sa[0]] = 0, i = 1; i < n; i++) {
69        x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
70      }
71    }
72
73    return;
74  }
75
76  void calheitght(int n) {
77    int i, j, k = 0;
78
79    for (i = 1; i < n; i++) {
80      rank[sa[i]] = i;
81    }
82
83    // print(rank, n);
84    for (i = 0; i < n; height[rank[i++]] = k)
85      for (k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++)
86        {}
87
88    return;
89  }
```

2.2 sam

```
1   // Copyright [2017] <dmnsn7>
2
3   void copy(int x, int y) {
4     pre[x] = pre[y];
5     len[x] = len[y];
6     memcpy(son[x], son[y], sizeof son[0]);
7   }
8
9   void insert(int c, int l) {
10    int p = tail, np = ++tot;
11    len[np] = l;
12    tail = np;
13
14    while (p && son[p][c] == 0) {
15      son[p][c] = np, p = pre[p];
16    }
17
18    if (p == 0) {
19      pre[np] = root;
20    } else {
21      int q = son[p][c];
22
23      if (len[p] + 1 == len[q]) {
24        pre[np] = q;
25      } else {
26        int nq = ++tot;
27        copy(nq, q);
28        len[nq] = len[p] + 1;
29        pre[np] = pre[q] = nq;
30
31        while (p && son[p][c] == q) {
32          son[p][c] = nq, p = pre[p];
33        }
34      }
35    }
36  }
37
38  void build(int n) {
```

```
39    for (int i = 1; i <= tot; i++) {
40      cnt[len[i]]++;
41    }
42
43    for (int i = 1; i <= n; i++) {
44      cnt[i] += cnt[i - 1];
45    }
46
47    for (int i = 1; i <= tot; i++) {
48      b[--cnt[len[i]]] = i;
49    }
50
51    for (int i = tot - 1; i >= 0; i--) {
52      int p = b[i], k = 0;
53      g[p] = 1;
54
55      for (int j = 0; j < 26; j++)
56        if (son[p][j]) {
57          int v = son[p][j];
58          g[p] += g[v];
59          son[p][k] = v;
60          gao[v] = j + 'a';
61          k++;
62        }
63
64      son[p][k] = 0;
65    }
66  }
```

2.3 kmp

```
1   // Copyright [2017] <dmnsn7>
2
3   void getFail() {
4     m = strlen(s);
5     f[0] = f[1] = 0;
6
7     for (int i = 1; i < m; i++) {
8       int j = f[i];
9
10      while (j && s[i] != s[j]) {
11        j = f[j];
12      }
13
14      f[i + 1] = s[i] == s[j] ? j + 1 : 0;
15    }
16  }
```

2.4 ac

```
1   // Copyright [2017] <dmnsn7>
2
3   void init() {
4     sz = 1;
5     ch[0].init();
6   }
7   void build() {
8     queue<int> q;
9     ch[0].fail = 0;
10
11    for (int c = 0; c < 4; c++) {
12      int u = ch[0].next[c];
13
14      if (u) {
15        ch[u].fail = 0;
16        q.push(u);
17      }
18    }
19
```

```
20    while (!q.empty()) {
21      int r = q.front();
22      q.pop();
23
24      for (int c = 0; c < 4; c++) {
25        int u = ch[r].next[c];
26
27        if (!u) {
28          ch[r].next[c] = ch[ch[r].fail].next[c];
29          continue;
30        }
31
32        q.push(u);
33        int v = ch[r].fail;
34
35        while (v && !ch[v].next[c]) {
36          v = ch[v].fail;
37        }
38
39        ch[u].fail = ch[v].next[c];
40        ch[u].isend |= ch[ch[u].fail].isend;
41      }
42    }
43  }
```

3 Geometry

3.1 c2c

```
1  // Copyright [2017] <dmnsn7>
2
3  Point rotate(const Point &p, double cost, double sint) {
4    double x = p.x, y = p.y;
5    return Point(x * cost - y * sint, x * sint + y * cost);
6  }
7
8  void circle_cross_circle(Circle a, Circle b, Point cro[]) {
9    double d = (a.o - b.o).len();
10   double cost = (a.r * a.r + d * d - b.r * b.r) / (2 * a.r * d);
11   double sint = sqrt(max(1.0 - cost * cost, 0.0));
12   Point v = (b.o - a.o) * (a.r / d);
13   cro[0] = a.o + rotate(v, cost, -sint);
14   cro[1] = a.o + rotate(v, cost, sint);
15 }
```

3.2 c2l

```
1  // Copyright [2017] <dmnsn7>
2
3  Point crosspt(const Point &a, const Point &b, const Point &p, const Point &q) {
4    double a1 = (b - a) * (p - a);
5    double a2 = (b - a) * (q - a);
6    return (p * a2 - q * a1) / (a2 - a1);
7  }
8  double sector_area(const Point &a, const Point &b) {
9    double theta = atan2(a.y, a.x) - atan2(b.y, b.x);
10
11   while (theta <= 0) {
12     theta += 2 * PI;
13   }
14
15   while (theta > 2 * PI) {
16     theta -= 2 * PI;
17   }
18
19   theta = min(theta, 2 * PI - theta);
20   return r * r * theta / 2;
21 }
22 double sqr(double x) { return x * x; }
```

```
23 void circle_cross_line(Point a, Point b, Point o, double r, Point ret[],
24                       const int &num) {
25   double x0 = o.x, y0 = o.y;
26   double x1 = a.x, y1 = a.y;
27   double x2 = b.x, y2 = b.y;
28   double dx = x2 - x1, dy = y2 - y1;
29   double A = dx * dx + dy * dy;
30   double B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
31   double C = sqr(x1 - x0) + sqr(y1 - y0) - sqr(r);
32   double delta = B * B - 4 * A * C;
33   num = 0;
34
35   if (dlcmp(delta) >= 0) {
36     double t1 = (-B - sqrt(max(delta, 0.0))) / (2 * A);
37     double t2 = (-B + sqrt(max(delta, 0.0))) / (2 * A);
38
39     if (dlcmp(t1 - 1) <= 0 && dlcmp(t1) >= 0) {
40       ret[num++] = Point(x1 + t1 * dx, y1 + t1 * dy);
41     }
42
43     if (dlcmp(t2 - 1) <= 0 && dlcmp(t2) >= 0) {
44       ret[num++] = Point(x1 + t2 * dx, y1 + t2 * dy);
45     }
46   }
47 }
48 double calc(const Point &a, const Point &b) {
49   Point p[2];
50   int num = 0;
51   int ina = dlcmp(a.len() - r) < 0;
52   int inb = dlcmp(b.len() - r) < 0;
53
54   if (ina) {
55     if (inb) {
56       return fabs(a * b) / 2;
57     } else {
58       circle_cross_line(a, b, Point(0, 0), r, p, num);
59       return sector_area(b, p[0]) + fabs(a * p[0]) / 2;
60     }
61   } else {
62     if (inb) {
63       circle_cross_line(a, b, Point(0, 0), r, p, num);
64       return sector_area(p[0], a) + fabs(p[0] * b) / 2;
65     } else {
66       circle_cross_line(a, b, Point(0, 0), r, p, num);
67
68       if (false) {
69         return sector_area(a, p[0]) + sector_area(p[1], b) +
70                fabs(p[0] * p[1]) / 2;
71       } else {
72         return sector_area(a, b);
73       }
74     }
75   }
76 }
77 double area() {
78   double ret = 0;
79
80   for (int i = 0; i < n; i++) {
81     int sgn = dlcmp(res[i] * res[i + 1]);
82
83     if (sgn != 0) {
84       ret += sgn * calc(res[i], res[i + 1]);
85     }
86   }
87
88   return ret;
89 }
```

3.3 halfplaneintersection

```
// Copyright [2017] <dmnsn7>

struct Halfplane {
  Point a, b;
  Halfplane() {}
  Halfplane(Point a, Point b) : a(a), b(b) {}

  bool satisfy(const Point &rhs) const { return sgn((rhs - a) * (b - a)) <= 0; }
  bool operator<(const Halfplane &rhs) const {
    int res = sgn((b - a).arg() - (rhs.b - rhs.a).arg());
    return res == 0 ? rhs.satisfy(a) : res < 0;
  }
};

Point crosspoint(const Halfplane &a, const Halfplane &b) {
  double k = (b.a - b.b) * (a.a - b.b);
  k = k / (k - ((b.a - b.b) * (a.b - b.b)));
  return a.a + (a.b - a.a) * k;
}

vector<Point> halfplaneIntersection(vector<Halfplane> v) {
  sort(v.begin(), v.end());
  deque<Halfplane> q;
  deque<Point> ans;
  q.push_back(v[0]);

  for (int i = 1; i < v.size(); i++) {
    if (sgn((v[i - 1].b - v[i - 1].a) * (v[i].b - v[i].a)) == 0) {
      continue;
    }

    while (ans.size() > 0 && !v[i].satisfy(ans.back())) {
      ans.pop_back();
      q.pop_back();
    }

    while (ans.size() > 0 && !v[i].satisfy(ans.front())) {
      ans.pop_front();
      q.pop_front();
    }

    ans.push_back(crosspoint(q.back(), v[i]));
    q.push_back(v[i]);
  }

  while (ans.size() > 0 && !q.front().satisfy(ans.back())) {
    ans.pop_back();
    q.pop_back();
  }

  while (ans.size() > 0 && !q.back().satisfy(ans.front())) {
    ans.pop_front();
    q.pop_front();
  }

  ans.push_back(crosspoint(q.back(), q.front()));
  return vector<Point>(ans.begin(), ans.end());
}

double area(const vector<Point> &p, int ansi) {
  double res = 0;

  for (int i = ansi; i + 1 < p.size(); i++) {
    res += p[i] * p[i + 1];
  }

  res += p.back() * p[ansi];
```

```
68    return fabs(res) / 2;
69  }
70
71  double ptol(Point a, Point b, Point c) {
72    double are = fabs((b - a) * (c - a));
73    return are / (b - c).len();
74  }
75
76  int main() {
77    int T_T, n, nc = 0;
78    cin >> T_T;
79    Point __0(0, 0), __1(1, 0), __2(1, 1), __3(0, 1);
80
81    while (T_T--) {
82      printf("Case #%d:\n", ++nc);
83      scanf("%d", &n);
84
85      for (int i = 0; i < n; i++) {
86        p[i].input();
87      }
88
89      for (int i = 0; i < n; i++) {
90        vector<Halfplane> v;
91        v.push_back(Halfplane(__0, __1));
92        v.push_back(Halfplane(__1, __2));
93        v.push_back(Halfplane(__2, __3));
94        v.push_back(Halfplane(__3, __0));
95
96        for (int j = 0; j < n; j++)
97          if (i != j) {
98            Point a = (p[i] + p[j]) / 2;
99            Point b = a + (p[i] - p[j]).rev();
100
101           if (!Halfplane(a, b).satisfy(p[i])) {
102             swap(a, b);
103           }
104
105           v.push_back(Halfplane(a, b));
106         }
107
108       vector<Point> ans = halfplaneIntersection(v);
109
110       double ret = 0, low = 1e100;
111       int ansi = 0;
112
113       for (int j = 0; j < ans.size(); j++)
114         if (ans[j].z() < low) {
115           low = ans[j].z(), ansi = j;
116         }
117
118       for (int j = 0; j < ansi; j++) {
119         ans.push_back(ans[j]);
120       }
121
122       ret = area(ans, ansi) * low;
123
124       for (int j = ansi + 1; j + 1 < ans.size(); j++) {
125         double ll = (ans[j] - ans[j + 1]).len();
126
127         if (ll < eps) {
128           continue;
129         }
130
131         double s = (ans[j].z() + ans[j + 1].z() - low * 2) * ll / 2;
132         double h = ptol(ans[ansi], ans[j], ans[j + 1]);
133         ret += s * h / 3;
134       }
135
136       printf("%.6f\n", ret);
```

```
137     }
138   }
139
140   return 0;
141 }
```

4 DataStruct

4.1 lct

```
1  // Copyright [2017] <dmnsn7>
2
3  int ch[MAXN][2], pre[MAXN], key[MAXN];
4  int add[MAXN], Max[MAXN], rev[MAXN], n;
5  bool rt[MAXN];
6  void update_add(int r, int d) {
7    if (!r) {
8      return;
9    }
10
11   key[r] += d;
12   add[r] += d;
13   Max[r] += d;
14 }
15 void update_rev(int r) {
16   if (!r) {
17     return;
18   }
19
20   swap(ch[r][0], ch[r][1]);
21   rev[r] ^= 1;
22 }
23 void push_down(int r) {
24   if (add[r]) {
25     update_add(ch[r][0], add[r]);
26     update_add(ch[r][1], add[r]);
27     add[r] = 0;
28   }
29
30   if (rev[r]) {
31     update_rev(ch[r][0]);
32     update_rev(ch[r][1]);
33     rev[r] = 0;
34   }
35 }
36 void display() {
37   for (int i = 1; i <= n; i++) {
38     printf("%d %d %d %d <%d>  ", i, ch[i][0], ch[i][1], pre[i], rt[i]);
39     printf("%d %d %d\n", add[i], key[i], Max[i]);
40   }
41 }
42 void push_up(int r) { Max[r] = max(max(Max[ch[r][0]], Max[ch[r][1]]), key[r]); }
43 void rotate(int x) {
44   int y = pre[x], kind = ch[y][1] == x;
45   ch[y][kind] = ch[x][!kind];
46   pre[ch[y][kind]] = y;
47   pre[x] = pre[y];
48   pre[y] = x;
49   ch[x][!kind] = y;
50
51   if (rt[y]) {
52     rt[y] = 0, rt[x] = 1;
53   } else {
54     ch[pre[x]][ch[pre[x]][1] == y] = x;
55   }
56
57   push_up(y);
58 }
```

```
59  void P(int r) {
60    if (!rt[r]) {
61      P(pre[r]);
62    }
63
64    push_down(r);
65  }
66  void splay(int r) {
67    P(r);
68
69    while (!rt[r]) {
70      int f = pre[r], ff = pre[f];
71
72      if (rt[f]) {
73        rotate(r);
74      } else if ((ch[ff][1] == f) == (ch[f][1] == r)) {
75        rotate(f), rotate(r);
76      } else {
77        rotate(r), rotate(r);
78      }
79    }
80
81    push_up(r);
82  }
83  int access(int x) {
84    int y = 0;
85
86    for (; x; x = pre[y = x]) {
87      splay(x);
88      rt[ch[x][1]] = 1, rt[ch[x][1] = y] = 0;
89      push_up(x);
90    }
91
92    return y;
93  }
94  bool judge(int u, int v) {
95    while (pre[u]) {
96      u = pre[u];
97    }
98
99    while (pre[v]) {
100     v = pre[v];
101   }
102
103   return u == v;
104 }
105 void mroot(int r) {
106   access(r);
107   splay(r);
108   update_rev(r);
109 }
110 void lca(const int &u, const int &v) {
111   access(v), v = 0;
112
113   // puts("--------------");display();
114   while (u) {
115     splay(u);
116
117     if (!pre[u]) {
118       return;
119     }
120
121     rt[ch[u][1]] = 1;
122     rt[ch[u][1] = v] = 0;
123     push_up(u);
124     u = pre[v = u];
125   }
126 }
```

```
127 void link(int u, int v) {
128   if (judge(u, v)) {
129     puts("-1");
130     return;
131   }
132
133   mroot(u);
134   pre[u] = v;
135 }
136 void cut(int u, int v) {
137   if (u == v || !judge(u, v)) {
138     puts("-1");
139     return;
140   }
141
142   mroot(u);
143   splay(v);
144   pre[ch[v][0]] = pre[v];
145   pre[v] = 0;
146   rt[ch[v][0]] = 1;
147   ch[v][0] = 0;
148   push_up(v);
149 }
150 void ADD(int u, int v, int w) {
151   if (!judge(u, v)) {
152     puts("-1");
153     return;
154   }
155
156   lca(u, v);
157   update_add(ch[u][1], w);
158   update_add(v, w);
159   key[u] += w;
160   push_up(u);
161 }
162 void query(int u, int v) {
163   if (!judge(u, v)) {
164     puts("-1");
165     return;
166   }
167
168   lca(u, v);
169   printf("%d\n", max(max(Max[v], Max[ch[u][1]]), key[u]));
170 }
171 vector<int> G[MAXN];
172 int que[MAXN];
173 void bfs() {
174   int front = 0, rear = 0;
175   que[rear++] = 1;
176   pre[1] = 0;
177
178   while (front < rear) {
179     int u = que[front++];
180
181     for (int i = 0; i < G[u].size(); i++) {
182       int v = G[u][i];
183
184       if (v == pre[u]) {
185         continue;
186       }
187
188       pre[v] = u;
189       que[rear++] = v;
190     }
191   }
192 }
193 int main() {
194   int q, u, v;
```

```
195
196   while (~scanf("%d", &n)) {
197     memset(add, 0, sizeof add);
198     memset(pre, 0, sizeof pre);
199     memset(rev, 0, sizeof rev);
200     memset(ch, 0, sizeof ch);
201
202     for (int i = 0; i <= n; i++) {
203       G[i].clear();
204       rt[i] = 1;
205     }
206
207     Max[0] = -INF;
208
209     for (int i = 1; i < n; i++) {
210       scanf("%d%d", &u, &v);
211       G[u].push_back(v);
212       G[v].push_back(u);
213     }
214
215     for (int i = 1; i <= n; i++) {
216       scanf("%d", &key[i]);
217       Max[i] = key[i];
218     }
219
220     scanf("%d", &q);
221     bfs();
222
223     int op, x, y, w;
224
225     while (q--) {
226       scanf("%d", &op);
227
228       if (op == 1) {
229         scanf("%d%d", &x, &y);
230         link(x, y);
231       } else if (op == 2) {
232         scanf("%d%d", &x, &y);
233         cut(x, y);
234       } else if (op == 3) {
235         scanf("%d%d%d", &w, &x, &y);
236         ADD(x, y, w);
237       } else {
238         scanf("%d%d", &x, &y);
239         query(x, y);
240       }
241
242       // display();
243     }
244
245     puts("");
246   }
247
248   return 0;
249 }
```

4.2 kdt

```
1  // Copyright [2017] <dmnsn7>
2
3  bool cmpx(const Node &a, const Node &b) { return a.x < b.x; }
4  bool cmpy(const Node &a, const Node &b) { return a.y < b.y; }
5
6  LL dis(const Node &a, const Node &b) { return sqr(a.x - b.x) + sqr(a.y - b.y); }
7
8  void build(int l, int r) {
9    if (l > r) {
10     return;
11   }
```

```
12
13    LL minx = min_element(p + l, p + r + 1, cmpx)->x;
14    LL maxx = max_element(p + l, p + r + 1, cmpx)->x;
15    LL miny = min_element(p + l, p + r + 1, cmpy)->y;
16    LL maxy = max_element(p + l, p + r + 1, cmpy)->y;
17    int mid = l + (r - l) / 2;
18    d[mid] = maxx - minx > maxy - miny;
19    nth_element(p + l, p + mid, p + r + 1, d[mid] ? cmpx : cmpy);
20
21    build(l, mid - 1);
22    build(mid + 1, r);
23  }
24
25  void query(int l, int r, const Node &a) {
26    if (l > r) {
27      return;
28    }
29
30    int mid = l + (r - l) / 2;
31    LL dist = dis(a, p[mid]);
32    LL d1 = d[mid] ? a.x - p[mid].x : a.y - p[mid].y;
33
34    if (dist > 0) {
35      res = min(res, dist);
36    }
37
38    int l1 = l, r1 = mid - 1;
39    int l2 = mid + 1, r2 = r;
40
41    if (d1 > 0) {
42      swap(l1, l2);
43      swap(r1, r2);
44    }
45
46    query(l1, r1, a);
47
48    if (d1 * d1 < res) {
49      query(l2, r2, a);
50    }
51  }
```

5 Graph

5.1 targan point connecting

```
1  // Copyright [2017] <dmnsn7>
2
3  void Tarjan(int u, int pre) {
4    Low[u] = DFN[u] = ++Index;
5    Stack[top++] = u;
6    Instack[u] = true;
7
8    for (int i = head[u]; i != -1; i = edge[i].next) {
9      int v = edge[i].to;
10
11     if (v == pre) {
12       continue;    //
13     }
14
15     if (!DFN[v]) {
16       Tarjan(v, u);
17
18       if (Low[u] > Low[v]) {
19         Low[u] = Low[v];
20       }
21       /*
22       if (Low[v] >= DFN[u]) {
23         block++;
24         int vn;
```

```
25          cc = 0;
26          memset(ok, false, sizeof(ok));
27
28          do {
29            vn = Stack[--top];
30            Belong[vn] = block;
31            Instack[vn] = false;
32            ok[vn] = true;
33            tmp[cc++] = vn;
34          } while (vn != v);
35
36          ok[u] = 1;
37          memset(color, -1, sizeof(color));
38
39          if (!dfs(u, 0)) {
40            can[u] = true;
41
42            while (cc--) {
43              can[tmp[cc]] = true;
44            }
45          }
46        }
47        */
48      } else if (Instack[v] && Low[u] > DFN[v]) {
49        Low[u] = DFN[v];
50      }
51    }
52
53    /*  targan
54    if (Low[u] == DFN[u]) {
55      scc++;
56
57      do {
58        v = Stack[--top];
59        Instack[v] = false;
60        Belong[v] = scc;
61        num[scc]++;
62      } while (v != u);
63    }
64    */
65  }
```

5.2 cut point bridge

```
1  // Copyright [2017] <dmnsn7>
2
3  const int MAXN = 10010;
4  const int MAXM = 100010;
5  struct Edge {
6    int to, next;
7    bool cut;  //
8  } edge[MAXM];
9  int head[MAXN], tot;
10 int Low[MAXN], DFN[MAXN], Stack[MAXN];
11 int Index, top;
12 bool Instack[MAXN];
13 bool cut[MAXN];
14 int add_block[MAXN];  //
15 int bridge;
16 void addedge(int u, int v) {
17   edge[tot].to = v;
18   edge[tot].next = head[u];
19   edge[tot].cut = false;
20   head[u] = tot++;
21 }
22 void Tarjan(int u, int pre) {
23   Low[u] = DFN[u] = ++Index;
24   Stack[top++] = u;
```

```
25    Instack[u] = true;
26    int son = 0;
27
28    for (int i = head[u]; i != -1; i = edge[i].next) {
29      int v = edge[i].to;
30
31      if (v == pre) {
32        continue;
33      }
34
35      if (!DFN[v]) {
36        son++;
37        Tarjan(v, u);
38
39        if (Low[u] > Low[v]) {
40          Low[u] = Low[v];
41        }
42
43        if (Low[v] > DFN[u]) {  //
44          bridge++;
45          edge[i].cut = true;
46          edge[i ^ 1].cut = true;
47        }
48
49        if (u != pre && Low[v] >= DFN[u]) {  //
50          cut[u] = true;
51          add_block[u]++;
52        }
53      } else if (Low[u] > DFN[v]) {
54        Low[u] = DFN[v];
55      }
56
57      /*
58       * else if( Instack[v] && Low[u] > DFN[v] )
59       *     Low[u] = DFN[v];
60       * }
61       * if(Low[u] == DFN[u]){
62       *     block++;
63       *     do
64       *     {
65       *         v = Stack[--top];
66       *         Instack[v] = false;
67       *         Belong[v] = block;
68       *     }while( v!=u );
69       * }
70       */
71    }
72
73    if (u == pre && son > 1) {
74      cut[u] = true;  //     ,1
75    }
76
77    if (u == pre) {
78      add_block[u] = son - 1;
79    }
80
81    Instack[u] = false;
82    top--;
83  }
84  void solve(int N) {
85    memset(DFN, 0, sizeof(DFN));
86    memset(Instack, false, sizeof(Instack));
87    memset(add_block, 0, sizeof(add_block));
88    memset(cut, false, sizeof(cut));
89    Index = top = 0;
90    bridge = 0;
91
92    for (int i = 1; i <= N; i++)
93      if (!DFN[i]) {
```

```
94        Tarjan(i, i);
95      }
96
97    printf("%d critical links\n", bridge);
98  }
99  void init() {
100     tot = 0;
101     memset(head, -1, sizeof(head));
102 }
```

5.3 hungary

```
1   // Copyright [2017] <dmnsn7>
2
3   bool dfs(int u) {
4     for (int i = head[u]; i != -1; i = edge[i].next) {
5       int v = edge[i].to;
6
7       if (!used[v]) {
8         used[v] = true;
9
10        if (linker[v] == -1 || dfs(linker[v])) {
11          linker[v] = u;
12          return true;
13        }
14      }
15    }
16
17    return false;
18  }
19  int hungary() {
20    memset(linker, -1, sizeof(linker));
21
22    for (int u = 0; u < uN; u++) {  //   0~uN-1
23      memset(used, false, sizeof(used));
24
25      if (dfs(u)) {
26        res++;
27      }
28    }
29
30    return res;
31  }
```

5.4 maxflow

```
1   // Copyright [2017] <dmnsn7>
2
3   #include <bits/stdc++.h>
4   using std;
5
6   const int MAXN = 100010;  //
7   const int MAXM = 400010;  //
8   const int oo = 0x3f3f3f3f;
9   struct Edge {
10    int to, next, cap, flow;
11  } edge[MAXM];  //  MAXM
12  int tol;
13  int head[MAXN];
14  int gap[MAXN], dep[MAXN], cur[MAXN];
15  void init() {
16    tol = 0;
17    memset(head, -1, sizeof(head));
18  }
19  void addedge(int u, int v, int w, int rw = 0) {
20    edge[tol].to = v;
21    edge[tol].cap = w;
22    edge[tol].flow = 0;
23    edge[tol].next = head[u];
24    head[u] = tol++;
```

```
25    edge[tol].to = u;
26    edge[tol].cap = rw;
27    edge[tol].flow = 0;
28    edge[tol].next = head[v];
29    head[v] = tol++;
30  }
31  int Q[MAXN];
32  void BFS(int ss, int tt) {
33    memset(dep, -1, sizeof(dep));
34    memset(gap, 0, sizeof(gap));
35    gap[0] = 1;
36    int front = 0, rear = 0;
37    dep[tt] = 0;
38    Q[rear++] = tt;
39
40    while (front != rear) {
41      int u = Q[front++];
42
43      for (int i = head[u]; i != -1; i = edge[i].next) {
44        int v = edge[i].to;
45
46        if (dep[v] != -1) {
47          continue;
48        }
49
50        Q[rear++] = v;
51        dep[v] = dep[u] + 1;
52        gap[dep[v]]++;
53      }
54    }
55  }
56  int S[MAXN];
57  int sap(int ss, int tt, int N) {
58    BFS(ss, tt);
59    memcpy(cur, head, sizeof(head));
60    int top = 0;
61    int u = ss;
62    int ans = 0;
63
64    while (dep[ss] < N) {
65      if (u == tt) {
66        int mi = oo;
67        int inser;
68
69        for (int i = 0; i < top; i++)
70          if (mi > edge[S[i]].cap - edge[S[i]].flow) {
71            mi = edge[S[i]].cap - edge[S[i]].flow;
72            inser = i;
73          }
74
75        for (int i = 0; i < top; i++) {
76          edge[S[i]].flow += mi;
77          edge[S[i] ^ 1].flow -= mi;
78        }
79
80        ans += mi;
81        top = inser;
82        u = edge[S[top] ^ 1].to;
83        continue;
84      }
85
86      bool flag = false;
87      int v;
88
89      for (int i = cur[u]; i != -1; i = edge[i].next) {
90        v = edge[i].to;
91
92        if (edge[i].cap - edge[i].flow && dep[v] + 1 == dep[u]) {
93          flag = true;
```

```
94            cur[u] = i;
95            break;
96          }
97        }
98
99        if (flag) {
100         S[top++] = cur[u];
101         u = v;
102         continue;
103       }
104
105       int mi = N;
106
107       for (int i = head[u]; i != -1; i = edge[i].next)
108         if (edge[i].cap - edge[i].flow && dep[edge[i].to] < mi) {
109           mi = dep[edge[i].to];
110           cur[u] = i;
111         }
112
113       gap[dep[u]]--;
114
115       if (!gap[dep[u]]) {
116         return ans;
117       }
118
119       dep[u] = mi + 1;
120       gap[dep[u]]++;
121
122       if (u != ss) {
123         u = edge[S[--top] ^ 1].to;
124       }
125     }
126
127     return ans;
128 }
```

5.5 costflow

```
1  // Copyright [2017] <dmnsn7>
2
3  const int MAXN = 10000;
4  const int MAXM = 100000;
5  const int INF = 0x3f3f3f3f;
6  struct Edge {
7    int to, next, cap, flow, cost;
8  } edge[MAXM];
9  int head[MAXN], tol;
10 int pre[MAXN], dis[MAXN];
11 bool vis[MAXN];
12 int N;  //      ,0~N-1
13 void init(int n) {
14   N = n;
15   tol = 0;
16   memset(head, -1, sizeof(head));
17 }
18 void addedge(int u, int v, int cap, int cost) {
19   edge[tol].to = v;
20   edge[tol].cap = cap;
21   edge[tol].cost = cost;
22   edge[tol].flow = 0;
23   edge[tol].next = head[u];
24   head[u] = tol++;
25   edge[tol].to = u;
26   edge[tol].cap = 0;
27   edge[tol].cost = -cost;
28   edge[tol].flow = 0;
29   edge[tol].next = head[v];
30   head[v] = tol++;
31 }
```

```
32 bool spfa(int s, int t) {
33   queue<int> q;
34
35   for (int i = 0; i < N; i++) {
36     dis[i] = INF;
37     vis[i] = false;
38     pre[i] = -1;
39   }
40
41   dis[s] = 0;
42   vis[s] = true;
43   q.push(s);
44
45   while (!q.empty()) {
46     int u = q.front();
47     q.pop();
48     vis[u] = false;
49
50     for (int i = head[u]; i != -1; i = edge[i].next) {
51       int v = edge[i].to;
52
53       if (edge[i].cap > edge[i].flow && dis[v] > dis[u] + edge[i].cost) {
54         dis[v] = dis[u] + edge[i].cost;
55         pre[v] = i;
56
57         if (!vis[v]) {
58           vis[v] = true;
59           q.push(v);
60         }
61       }
62     }
63   }
64
65   if (pre[t] == -1) {
66     return false;
67   } else {
68     return true;
69   }
70 }
71 //     ,    cost
72 int minCostMaxflow(int s, int t, const int &cost) {
73   int flow = 0;
74   cost = 0;
75
76   while (spfa(s, t)) {
77     int Min = INF;
78
79     for (int i = pre[t]; i != -1; i = pre[edge[i ^ 1].to]) {
80       if (Min > edge[i].cap - edge[i].flow) {
81         Min = edge[i].cap - edge[i].flow;
82       }
83     }
84
85     for (int i = pre[t]; i != -1; i = pre[edge[i ^ 1].to]) {
86       edge[i].flow += Min;
87       edge[i ^ 1].flow -= Min;
88       cost += edge[i].cost * Min;
89     }
90
91     flow += Min;
92   }
93
94   return flow;
95 }
```

5.6 min tree graph

```
1 // Copyright [2017] <dmnsn7>
2
```

```
3  const int INF = 0x3f3f3f3f;
4  const int MAXN = 1010;
5  const int MAXM = 40010;
6  struct Edge {
7    int u, v, cost;
8  };
9  Edge edge[MAXM];
10 int pre[MAXN], id[MAXN], visit[MAXN], in[MAXN];
11 int zhuliu(int root, int n, int m, Edge edge[]) {
12   int res = 0, u, v;
13
14   while (1) {
15     for (int i = 0; i < n; i++) {
16       in[i] = INF;
17     }
18
19     for (int i = 0; i < m; i++)
20       if (edge[i].u != edge[i].v && edge[i].cost < in[edge[i].v]) {
21         pre[edge[i].v] = edge[i].u;
22         in[edge[i].v] = edge[i].cost;
23       }
24
25     for (int i = 0; i < n; i++)
26       if (i != root && in[i] == INF) {
27         return -1;      //
28       }
29
30     int tn = 0;
31     memset(id, -1, sizeof(id));
32     memset(visit, -1, sizeof(visit));
33     in[root] = 0;
34
35     for (int i = 0; i < n; i++) {
36       res += in[i];
37       v = i;
38
39       while (visit[v] != i && id[v] == -1 && v != root) {
40         visit[v] = i;
41         v = pre[v];
42       }
43
44       if (v != root && id[v] == -1) {
45         for (int u = pre[v]; u != v; u = pre[u]) {
46           id[u] = tn;
47         }
48
49         id[v] = tn++;
50       }
51     }
52
53     if (tn == 0) {
54       break;     //
55     }
56
57     for (int i = 0; i < n; i++)
58       if (id[i] == -1) {
59         id[i] = tn++;
60       }
61
62     for (int i = 0; i < m;) {
63       v = edge[i].v;
64       edge[i].u = id[edge[i].u];
65       edge[i].v = id[edge[i].v];
66
67       if (edge[i].u != edge[i].v) {
68         edge[i++].cost -= in[v];
69       } else {
70         swap(edge[i], edge[--m]);
```

```
71        }
72      }
73
74      n = tn;
75      root = id[root];
76    }
77
78    return res;
79  }
```

5.7 flowertree

```
1   // Copyright [2017] <dmnsn7>
2
3   const int MAXN = 250;
4   int N;  //       ,1N
5   bool Graph[MAXN][MAXN];
6   int Match[MAXN];
7   bool InQueue[MAXN], InPath[MAXN], InBlossom[MAXN];
8   int Head, Tail;
9   int Queue[MAXN];
10  int Start, Finish;
11  int NewBase;
12  int Father[MAXN], Base[MAXN];
13  int Count;  //      ,Count/2
14  void CreateGraph() {
15    int u, v;
16    memset(Graph, false, sizeof(Graph));
17    scanf("%d", &N);
18
19    while (scanf("%d%d", &u, &v) == 2) {
20      Graph[u][v] = Graph[v][u] = true;
21    }
22  }
23  void Push(int u) {
24    Queue[Tail] = u;
25    Tail++;
26    InQueue[u] = true;
27  }
28  int Pop() {
29    int res = Queue[Head];
30    Head++;
31    return res;
32  }
33  int FindCommonAncestor(int u, int v) {
34    memset(InPath, false, sizeof(InPath));
35
36    while (true) {
37      u = Base[u];
38      InPath[u] = true;
39
40      if (u == Start) {
41        break;
42      }
43
44      u = Father[Match[u]];
45    }
46
47    while (true) {
48      v = Base[v];
49
50      if (InPath[v]) {
51        break;
52      }
53
54      v = Father[Match[v]];
55    }
56
57    return v;
58  }
```

```
59  void ResetTrace(int u) {
60    while (Base[u] != NewBase) {
61      int v = Match[u];
62      InBlossom[Base[u]] = InBlossom[Base[v]] = true;
63      u = Father[v];
64
65      if (Base[u] != NewBase) {
66        Father[u] = v;
67      }
68    }
69  }
70  void BloosomContract(int u, int v) {
71    NewBase = FindCommonAncestor(u, v);
72    memset(InBlossom, false, sizeof(InBlossom));
73    ResetTrace(u);
74    ResetTrace(v);
75
76    if (Base[u] != NewBase) {
77      Father[u] = v;
78    }
79
80    if (Base[v] != NewBase) {
81      Father[v] = u;
82    }
83
84    for (int tu = 1; tu <= N; tu++)
85      if (InBlossom[Base[tu]]) {
86        Base[tu] = NewBase;
87
88        if (!InQueue[tu]) {
89          Push(tu);
90        }
91      }
92  }
93
94  void FindAugmentingPath() {
95    memset(InQueue, false, sizeof(InQueue));
96    memset(Father, 0, sizeof(Father));
97
98    for (int i = 1; i <= N; i++) {
99      Base[i] = i;
100   }
101
102   Head = Tail = 1;
103   Push(Start);
104   Finish = 0;
105
106   while (Head < Tail) {
107     int u = Pop();
108
109     for (int v = 1; v <= N; v++)
110       if (Graph[u][v] && (Base[u] != Base[v]) && (Match[u] != v)) {
111         if ((v == Start) || ((Match[v] > 0) && Father[Match[v]] > 0)) {
112           BloosomContract(u, v);
113         } else if (Father[v] == 0) {
114           Father[v] = u;
115
116           if (Match[v] > 0) {
117             Push(Match[v]);
118           } else {
119             Finish = v;
120             return;
121           }
122         }
123       }
124   }
125 }
126 void AugmentPath() {
```

```
127   int u = Finish;
128
129   while (u > 0) {
130     int v = Father[u];
131     int w = Match[v];
132     Match[v] = u;
133     Match[u] = v;
134     u = w;
135   }
136 }
137 void Edmonds() {
138   memset(Match, 0, sizeof(Match));
139
140   for (int u = 1; u <= N; u++)
141     if (Match[u] == 0) {
142       Start = u;
143       FindAugmentingPath();
144
145       if (Finish > 0) {
146         AugmentPath();
147       }
148     }
149 }
150 void PrintMatch() {
151   Count = 0;
152
153   for (int u = 1; u <= N; u++)
154     if (Match[u] > 0) {
155       Count++;
156     }
157
158   printf("%d\n", Count);
159
160   for (int u = 1; u <= N; u++)
161     if (u < Match[u]) {
162       printf("%d %d\n", u, Match[u]);
163     }
164 }
165 int main() {
166   CreateGraph();  //
167   Edmonds();      //
168   PrintMatch();   //
169   return 0;
170 }
```

5.8 2-sat

```
1  // Copyright [2017] <dmnsn7>
2
3  const int MAXN = 20020;
4  const int MAXM = 100010;
5  struct Edge {
6    int to, next;
7  } edge[MAXM];
8  int head[MAXN], tot;
9  void init() {
10   tot = 0;
11   memset(head, -1, sizeof(head));
12 }
13 void addedge(int u, int v) {
14   edge[tot].to = v;
15   edge[tot].next = head[u];
16   head[u] = tot++;
17 }
18 bool vis[MAXN];     //     ,  true
19 int S[MAXN], top;   //
20 bool dfs(int u) {
21   if (vis[u ^ 1]) {
22     return false;
```

```
23    }
24
25    if (vis[u]) {
26      return true;
27    }
28
29    vis[u] = true;
30    S[top++] = u;
31
32    for (int i = head[u]; i != -1; i = edge[i].next)
33      if (!dfs(edge[i].to)) {
34        return false;
35      }
36
37    return true;
38  }
39  bool Twosat(int n) {
40    memset(vis, false, sizeof(vis));
41
42    for (int i = 0; i < n; i += 2) {
43      if (vis[i] || vis[i ^ 1]) {
44        continue;
45      }
46
47      top = 0;
48
49      if (!dfs(i)) {
50        while (top) {
51          vis[S[--top]] = false;
52        }
53
54        if (!dfs(i ^ 1)) {
55          return false;
56        }
57      }
58    }
59
60    return true;
61  }
62  int main() {
63    int n, m;
64    int u, v;
65
66    while (scanf("%d%d", &n, &m) == 2) {
67      init();
68
69      while (m--) {
70        scanf("%d%d", &u, &v);
71        u--;
72        v--;
73        addedge(u, v ^ 1);
74        addedge(v, u ^ 1);
75      }
76
77      if (Twosat(2 * n)) {
78        for (int i = 0; i < 2 * n; i++)
79          if (vis[i]) {
80            printf("%d\n", i + 1);
81          }
82      } else {
83        printf("NIE\n");
84      }
85    }
86
87    return 0;
88  }
```

5.9 km

```
1  // Copyright [2017] <dmnsn7>
```

```
2
3  bool DFS(int x) {
4    visx[x] = true;
5
6    for (int y = 0; y < ny; y++) {
7      if (visy[y]) {
8        continue;
9      }
10
11     int tmp = lx[x] + ly[y] - g[x][y];
12
13     if (tmp == 0) {
14       visy[y] = true;
15
16       if (linker[y] == -1 || DFS(linker[y])) {
17         linker[y] = x;
18         return true;
19       }
20     } else if (slack[y] > tmp) {
21       slack[y] = tmp;
22     }
23   }
24
25   return false;
26 }
27 int KM() {
28   memset(linker, -1, sizeof(linker));
29   memset(ly, 0, sizeof(ly));
30
31   for (int i = 0; i < nx; i++) {
32     lx[i] = -INF;
33
34     for (int j = 0; j < ny; j++)
35       if (g[i][j] > lx[i]) {
36         lx[i] = g[i][j];
37       }
38   }
39
40   for (int x = 0; x < nx; x++) {
41     for (int i = 0; i < ny; i++) {
42       slack[i] = INF;
43     }
44
45     while (true) {
46       memset(visx, false, sizeof(visx));
47       memset(visy, false, sizeof(visy));
48
49       if (DFS(x)) {
50         break;
51       }
52
53       int d = INF;
54
55       for (int i = 0; i < ny; i++)
56         if (!visy[i] && d > slack[i]) {
57           d = slack[i];
58         }
59
60       for (int i = 0; i < nx; i++)
61         if (visx[i]) {
62           lx[i] -= d;
63         }
64
65       for (int i = 0; i < ny; i++) {
66         if (visy[i]) {
67           ly[i] += d;
68         } else {
69           slack[i] -= d;
70         }
```

```
71          }
72        }
73      }
74
75      int res = 0;
76
77      for (int i = 0; i < ny; i++)
78        if (linker[i] != -1) {
79          res += g[linker[i]][i];
80        }
81
82      return res;
83  }
```