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;
    }
  }
}
```

2 String

2.1 sa

```
int r[maxn], wa[maxn], wb[maxn], wv[maxn], ws[maxn], sa[maxn];
int rank[maxn], height[maxn];

/*
 *         nn   -10
 *           sasa[1]~sa[n]
 *        rankrank[0]~rank[n    -1],
 *    heightii        -12~   nsa
 */

inline bool cmp(int *r, int a, int b, int l) {
  return r[a] == r[b] && r[a + l] == r[b + l];
}

void da(int n, int m) {
  int i, j, p, *x = wa, *y = wb;

  for (i = 0; i < m; i++) {
    ws[i] = 0;
  }

  for (i = 0; i < n; i++) {
    ws[x[i] = r[i]]++;
  }

  for (i = 1; i < m; i++) {
    ws[i] += ws[i - 1];
  }

  for (i = n - 1; i >= 0; i--) {
    sa[--ws[x[i]]] = i;
  }

  for (j = 1, p = 1; p < n; j <<= 1, m = p) {
    for (p = 0, i = n - j; i < n; i++) {
      y[p++] = i;
    }

    for (i = 0; i < n; i++)
      if (sa[i] >= j) {
        y[p++] = sa[i] - j;
      }

    for (i = 0; i < n; i++) {
      wv[i] = x[y[i]];
    }

    for (i = 0; i < m; i++) {
      ws[i] = 0;
    }

    for (i = 0; i < n; i++) {
      ws[wv[i]]++;
    }

    for (i = 1; i < m; i++) {
      ws[i] += ws[i - 1];
    }

    for (i = n - 1; i >= 0; i--) {
      sa[--ws[wv[i]]] = y[i];
    }

    swap(x, y);

    for (p = 1, x[sa[0]] = 0, i = 1; i < n; i++) {
      x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
    }
  }

  return;
}

void calheitght(int n) {
  int i, j, k = 0;

  for (i = 1; i < n; i++) {
    rank[sa[i]] = i;
  }

  // print(rank, n);
  for (i = 0; i < n; height[rank[i++]] = k)
    for (k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++)
      {}

  return;
}
```

2.2 sam

```
void copy(int x, int y) {
  pre[x] = pre[y];
  len[x] = len[y];
  memcpy(son[x], son[y], sizeof son[0]);
}

void insert(int c, int l) {
  int p = tail, np = ++tot;
  len[np] = l;
  tail = np;

  while (p && son[p][c] == 0) {
    son[p][c] = np, p = pre[p];
  }

  if (p == 0) {
    pre[np] = root;
  } else {
    int q = son[p][c];

    if (len[p] + 1 == len[q]) {
      pre[np] = q;
    } else {
      int nq = ++tot;
      copy(nq, q);
      len[nq] = len[p] + 1;
      pre[np] = pre[q] = nq;

      while (p && son[p][c] == q) {
        son[p][c] = nq, p = pre[p];
      }
    }
  }
}

void build(int n) {
  for (int i = 1; i <= tot; i++) {
    cnt[len[i]]++;
  }

  for (int i = 1; i <= n; i++) {
    cnt[i] += cnt[i - 1];
  }

  for (int i = 1; i <= tot; i++) {
    b[--cnt[len[i]]] = i;
  }

  for (int i = tot - 1; i >= 0; i--) {
    int p = b[i], k = 0;
    g[p] = 1;

    for (int j = 0; j < 26; j++)
      if (son[p][j]) {
        int v = son[p][j];
        g[p] += g[v];
        son[p][k] = v;
        gao[v] = j + 'a';
        k++;
      }

    son[p][k] = 0;
  }
}
```

2.3 kmp

```
void getFail() {
 m = strlen(s);
  f[0] = f[1] = 0;

  for (int i = 1; i < m; i++) {
    int j = f[i];

    while (j && s[i] != s[j]) {
      j = f[j];
    }

    f[i + 1] = s[i] == s[j] ? j + 1 : 0;
  }
}
```

2.4 ac

```
void init() {
  sz = 1;
  ch[0].init();
}
void build() {
  queue<int> q;
  ch[0].fail = 0;

  for (int c = 0; c < 4; c++) {
    int u = ch[0].next[c];

    if (u) {
      ch[u].fail = 0;
      q.push(u);
    }
  }

  while (!q.empty()) {
    int r = q.front();
    q.pop();

    for (int c = 0; c < 4; c++) {
      int u = ch[r].next[c];

      if (!u) {
        ch[r].next[c] = ch[ch[r].fail].next[c];
        continue;
      }

      q.push(u);
      int v = ch[r].fail;

      while (v && !ch[v].next[c]) {
        v = ch[v].fail;
      }

      ch[u].fail = ch[v].next[c];
      ch[u].isend |= ch[ch[u].fail].isend;
    }
  }
}
```

3 Geometry

3.1 c2c

```
Point rotate(const Point &p, double cost, double sint) {
  double x = p.x, y = p.y;
  return Point(x * cost - y * sint, x * sint + y * cost);
}

void circle_cross_circle(Circle a, Circle b, Point cro[]) {
  double d = (a.o - b.o).len();
  double cost = (a.r * a.r + d * d - b.r * b.r) / (2 * a.r * d);
  double sint = sqrt(max(1.0 - cost * cost, 0.0));
  Point v = (b.o - a.o) * (a.r / d);
  cro[0] = a.o + rotate(v, cost, -sint);
  cro[1] = a.o + rotate(v, cost, sint);
}
```

3.2 c2l

```
Point crosspt(const Point &a, const Point &b, const Point &p, const Point &q) {
  double a1 = (b - a) * (p - a);
  double a2 = (b - a) * (q - a);
  return (p * a2 - q * a1) / (a2 - a1);
}
double sector_area(const Point &a, const Point &b) {
  double theta = atan2(a.y, a.x) - atan2(b.y, b.x);

  while (theta <= 0) {
    theta += 2 * PI;
  }

  while (theta > 2 * PI) {
    theta -= 2 * PI;
  }

  theta = min(theta, 2 * PI - theta);
  return r * r * theta / 2;
}
double sqr(double x) { return x * x; }
void circle_cross_line(Point a, Point b, Point o, double r, Point ret[],
                       const int &num) {
  double x0 = o.x, y0 = o.y;
  double x1 = a.x, y1 = a.y;
  double x2 = b.x, y2 = b.y;
  double dx = x2 - x1, dy = y2 - y1;
  double A = dx * dx + dy * dy;
  double B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
  double C = sqr(x1 - x0) + sqr(y1 - y0) - sqr(r);
  double delta = B * B - 4 * A * C;
  num = 0;

  if (dlcmp(delta) >= 0) {
    double t1 = (-B - sqrt(max(delta, 0.0))) / (2 * A);
    double t2 = (-B + sqrt(max(delta, 0.0))) / (2 * A);

    if (dlcmp(t1 - 1) <= 0 && dlcmp(t1) >= 0) {
      ret[num++] = Point(x1 + t1 * dx, y1 + t1 * dy);
    }

    if (dlcmp(t2 - 1) <= 0 && dlcmp(t2) >= 0) {
      ret[num++] = Point(x1 + t2 * dx, y1 + t2 * dy);
    }
  }
}
double calc(const Point &a, const Point &b) {
  Point p[2];
  int num = 0;
  int ina = dlcmp(a.len() - r) < 0;
  int inb = dlcmp(b.len() - r) < 0;

  if (ina) {
    if (inb) {
      return fabs(a * b) / 2;
    } else {
      circle_cross_line(a, b, Point(0, 0), r, p, num);
      return sector_area(b, p[0]) + fabs(a * p[0]) / 2;
    }
  } else {
    if (inb) {
      circle_cross_line(a, b, Point(0, 0), r, p, num);
      return sector_area(p[0], a) + fabs(p[0] * b) / 2;
    } else {
      circle_cross_line(a, b, Point(0, 0), r, p, num);

      if (false) {
        return sector_area(a, p[0]) + sector_area(p[1], b) +
               fabs(p[0] * p[1]) / 2;
      } else {
        return sector_area(a, b);
      }
    }
  }
}
double area() {
  double ret = 0;

  for (int i = 0; i < n; i++) {
    int sgn = dlcmp(res[i] * res[i + 1]);

    if (sgn != 0) {
      ret += sgn * calc(res[i], res[i + 1]);
    }
  }

  return ret;
}
```

3.3 halfplaneintersection

```
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;
  }
};
```

```cpp

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];
  return fabs(res) / 2;
}

double ptol(Point a, Point b, Point c) {
  double are = fabs((b - a) * (c - a));
  return are / (b - c).len();
}

int main() {
  int T_T, n, nc = 0;
  cin >> T_T;
  Point __0(0, 0), __1(1, 0), __2(1, 1), __3(0, 1);

  while (T_T--) {
    printf("Case #%d:\n", ++nc);
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
      p[i].input();
    }

    for (int i = 0; i < n; i++) {
      vector<Halfplane> v;
      v.push_back(Halfplane(__0, __1));
      v.push_back(Halfplane(__1, __2));
      v.push_back(Halfplane(__2, __3));
      v.push_back(Halfplane(__3, __0));

      for (int j = 0; j < n; j++)
        if (i != j) {
          Point a = (p[i] + p[j]) / 2;
          Point b = a + (p[i] - p[j]).rev();

          if (!Halfplane(a, b).satisfy(p[i])) {
            swap(a, b);
          }

          v.push_back(Halfplane(a, b));
        }

      vector<Point> ans = halfplaneIntersection(v);

      double ret = 0, low = 1e100;
      int ansi = 0;

      for (int j = 0; j < ans.size(); j++)
        if (ans[j].z() < low) {
          low = ans[j].z(), ansi = j;
        }

      for (int j = 0; j < ansi; j++) {
        ans.push_back(ans[j]);
      }

      ret = area(ans, ansi) * low;

      for (int j = ansi + 1; j + 1 < ans.size(); j++) {
        double ll = (ans[j] - ans[j + 1]).len();

        if (ll < eps) {
          continue;
        }

        double s = (ans[j].z() + ans[j + 1].z() - low * 2) * ll / 2;
        double h = ptol(ans[ansi], ans[j], ans[j + 1]);
        ret += s * h / 3;
      }

      printf("%.6f\n", ret);
    }
  }

  return 0;
}
```

4 DataStruct

4.1 lct

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

  key[r] += d;
  add[r] += d;
  Max[r] += d;
}
void update_rev(int r) {
  if (!r) {
    return;
  }

  swap(ch[r][0], ch[r][1]);
  rev[r] ^= 1;
}
void push_down(int r) {
  if (add[r]) {
    update_add(ch[r][0], add[r]);
    update_add(ch[r][1], add[r]);
    add[r] = 0;
  }

  if (rev[r]) {
    update_rev(ch[r][0]);
    update_rev(ch[r][1]);
    rev[r] = 0;
  }
}
void display() {
  for (int i = 1; i <= n; i++) {
    printf("%d %d %d %d <%d>  ", i, ch[i][0], ch[i][1], pre[i], rt[i]);
    printf("%d %d %d\n", add[i], key[i], Max[i]);
  }
}
void push_up(int r) { Max[r] = max(max(Max[ch[r][0]], Max[ch[r][1]]), key[r]); }
void rotate(int x) {
  int y = pre[x], kind = ch[y][1] == x;
  ch[y][kind] = ch[x][!kind];
  pre[ch[y][kind]] = y;
  pre[x] = pre[y];
  pre[y] = x;
  ch[x][!kind] = y;

  if (rt[y]) {
    rt[y] = 0, rt[x] = 1;
  } else {
    ch[pre[x]][ch[pre[x]][1] == y] = x;
  }

  push_up(y);
}
void P(int r) {
  if (!rt[r]) {
    P(pre[r]);
  }

  push_down(r);
}
void splay(int r) {
  P(r);

  while (!rt[r]) {
    int f = pre[r], ff = pre[f];

    if (rt[f]) {
      rotate(r);
    } else if ((ch[ff][1] == f) == (ch[f][1] == r)) {
      rotate(f), rotate(r);
    } else {
      rotate(r), rotate(r);
    }
  }

  push_up(r);
}
int access(int x) {
  int y = 0;

  for (; x; x = pre[y = x]) {
    splay(x);
    rt[ch[x][1]] = 1, rt[ch[x][1] = y] = 0;
    push_up(x);
  }

  return y;
}
bool judge(int u, int v) {
  while (pre[u]) {
    u = pre[u];
  }

  while (pre[v]) {
    v = pre[v];
  }

  return u == v;
}
void mroot(int r) {
  access(r);
  splay(r);
  update_rev(r);
}
void lca(const int &u, const int &v) {
  access(v), v = 0;

  // puts("-------------");display();
  while (u) {
    splay(u);

    if (!pre[u]) {
      return;
    }

    rt[ch[u][1]] = 1;
    rt[ch[u][1] = v] = 0;
    push_up(u);
    u = pre[v = u];
  }
}
void link(int u, int v) {
  if (judge(u, v)) {
    puts("-1");
    return;
  }

  mroot(u);
  pre[u] = v;
}
void cut(int u, int v) {
  if (u == v || !judge(u, v)) {
    puts("-1");
    return;
  }

  mroot(u);
  splay(v);
  pre[ch[v][0]] = pre[v];
  pre[v] = 0;
  rt[ch[v][0]] = 1;
  ch[v][0] = 0;
  push_up(v);
}
void ADD(int u, int v, int w) {
  if (!judge(u, v)) {
    puts("-1");
    return;
  }

  lca(u, v);
  update_add(ch[u][1], w);
  update_add(v, w);
  key[u] += w;
  push_up(u);
}
void query(int u, int v) {
  if (!judge(u, v)) {
    puts("-1");
    return;
  }

  lca(u, v);
  printf("%d\n", max(max(Max[v], Max[ch[u][1]]), key[u]));
}
vector<int> G[MAXN];
int que[MAXN];
void bfs() {
  int front = 0, rear = 0;
  que[rear++] = 1;
  pre[1] = 0;

  while (front < rear) {
    int u = que[front++];

    for (int i = 0; i < G[u].size(); i++) {
      int v = G[u][i];

      if (v == pre[u]) {
        continue;
      }

      pre[v] = u;
      que[rear++] = v;
    }
  }
}
int main() {
  int q, u, v;

  while (~scanf("%d", &n)) {
    memset(add, 0, sizeof add);
    memset(pre, 0, sizeof pre);
    memset(rev, 0, sizeof rev);
    memset(ch, 0, sizeof ch);

    for (int i = 0; i <= n; i++) {
      G[i].clear();
      rt[i] = 1;
    }

    Max[0] = -INF;

    for (int i = 1; i < n; i++) {
      scanf("%d%d", &u, &v);
      G[u].push_back(v);
      G[v].push_back(u);
    }

    for (int i = 1; i <= n; i++) {
      scanf(”%d”, &key[i]);
      Max[i] = key[i];
    }

    scanf(”%d”, &q);
    bfs();

    int op, x, y, w;

    while (q--) {
      scanf(”%d”, &op);

      if (op == 1) {
        scanf(”%d%d”, &x, &y);
        link(x, y);
      } else if (op == 2) {
        scanf(”%d%d”, &x, &y);
        cut(x, y);
      } else if (op == 3) {
        scanf(”%d%d%d”, &w, &x, &y);
        ADD(x, y, w);
      } else {
        scanf(”%d%d”, &x, &y);
        query(x, y);
      }

      // display();
    }

    puts(””);
  }

  return 0;
}
```

4.2 kdt

```
bool cmpx(const Node &a, const Node &b) { return a.x < b.x; }
bool cmpy(const Node &a, const Node &b) { return a.y < b.y; }

LL dis(const Node &a, const Node &b) { return sqr(a.x - b.x) + sqr(a.y - b.y); }

void build(int l, int r) {
  if (l > r) {
    return;
  }

  LL minx = min_element(p + l, p + r + 1, cmpx)->x;
  LL maxx = max_element(p + l, p + r + 1, cmpx)->x;
  LL miny = min_element(p + l, p + r + 1, cmpy)->y;
  LL maxy = max_element(p + l, p + r + 1, cmpy)->y;
  int mid = l + (r - l) / 2;
  d[mid] = maxx - minx > maxy - miny;
  nth_element(p + l, p + mid, p + r + 1, d[mid] ? cmpx : cmpy);

  build(l, mid - 1);
  build(mid + 1, r);
}

void query(int l, int r, const Node &a) {
  if (l > r) {
    return;
  }

  int mid = l + (r - l) / 2;
  LL dist = dis(a, p[mid]);
  LL d1 = d[mid] ? a.x - p[mid].x : a.y - p[mid].y;

  if (dist > 0) {
    res = min(res, dist);
  }

  int l1 = l, r1 = mid - 1;
  int l2 = mid + 1, r2 = r;

  if (d1 > 0) {
    swap(l1, l2);
    swap(r1, r2);
  }

  query(l1, r1, a);

  if (d1 * d1 < res) {
    query(l2, r2, a);
  }
}
```

5 Graph

5.1 targan point connecting

```
void Tarjan(int u, int pre) {
  Low[u] = DFN[u] = ++Index;
  Stack[top++] = u;
  Instack[u] = true;

  for (int i = head[u]; i != -1; i = edge[i].next) {
    int v = edge[i].to;

    if (v == pre) {
      continue;    //
    }

    if (!DFN[v]) {
      Tarjan(v, u);

      if (Low[u] > Low[v]) {
        Low[u] = Low[v];
      }
      /*
      if (Low[v] >= DFN[u]) {
        block++;
        int vn;
        cc = 0;
        memset(ok, false, sizeof(ok));

        do {
          vn = Stack[--top];
          Belong[vn] = block;
          Instack[vn] = false;
          ok[vn] = true;
          tmp[cc++] = vn;
        } while (vn != v);

        ok[u] = 1;
        memset(color, -1, sizeof(color));

        if (!dfs(u, 0)) {
          can[u] = true;

          while (cc--) {
            can[tmp[cc]] = true;
          }
        }
      }
      */
    } else if (Instack[v] && Low[u] > DFN[v]) {
      Low[u] = DFN[v];
    }
  }

  /*  targan
  if (Low[u] == DFN[u]) {
    scc++;

    do {
      v = Stack[--top];
      Instack[v] = false;
      Belong[v] = scc;
      num[scc]++;
    } while (v != u);
  }
  */
}
```

5.2 cut point bridge

```
const int MAXN = 10010;
const int MAXM = 100010;
struct Edge {
  int to, next;
  bool cut;  //
} edge[MAXM];
int head[MAXN], tot;
int Low[MAXN], DFN[MAXN], Stack[MAXN];
int Index, top;
bool Instack[MAXN];
bool cut[MAXN];
int add_block[MAXN];  //
int bridge;
void addedge(int u, int v) {
  edge[tot].to = v;
  edge[tot].next = head[u];
  edge[tot].cut = false;
  head[u] = tot++;
}
void Tarjan(int u, int pre) {
  Low[u] = DFN[u] = ++Index;
  Stack[top++] = u;
  Instack[u] = true;
  int son = 0;

  for (int i = head[u]; i != -1; i = edge[i].next) {
    int v = edge[i].to;

    if (v == pre) {
      continue;
    }

    if (!DFN[v]) {
      son++;
      Tarjan(v, u);

      if (Low[u] > Low[v]) {
        Low[u] = Low[v];
      }

      if (Low[v] > DFN[u]) {  //
        bridge++;
        edge[i].cut = true;
        edge[i ^ 1].cut = true;
      }

      if (u != pre && Low[v] >= DFN[u]) {  //
        cut[u] = true;
        add_block[u]++;
      }
    } else if (Low[u] > DFN[v]) {
      Low[u] = DFN[v];
    }

    /*
     * else if( Instack[v] && Low[u] > DFN[v] )
     *      Low[u] = DFN[v];
     * }
     * if(Low[u] == DFN[u]){
     *      block++;
     *      do
     *      {
     *            v = Stack[--top];
     *            Instack[v] = false;
     *            Belong[v] = block;
     *      }while( v!=u );
     * }
     */
  }

  if (u == pre && son > 1) {
    cut[u] = true;  //      ,1
  }

  if (u == pre) {
    add_block[u] = son - 1;
  }

  Instack[u] = false;
  top--;
}
void solve(int N) {
  memset(DFN, 0, sizeof(DFN));
  memset(Instack, false, sizeof(Instack));
  memset(add_block, 0, sizeof(add_block));
  memset(cut, false, sizeof(cut));
  Index = top = 0;
  bridge = 0;

  for (int i = 1; i <= N; i++)
    if (!DFN[i]) {
      Tarjan(i, i);
    }

  printf("%d critical links\n", bridge);
}
void init() {
    tot = 0;
    memset(head, -1, sizeof(head));
}
```

5.3 hungary

```
bool dfs(int u) {
  for (int i = head[u]; i != -1; i = edge[i].next) {
    int v = edge[i].to;

    if (!used[v]) {
      used[v] = true;

      if (linker[v] == -1 || dfs(linker[v])) {
        linker[v] = u;
        return true;
      }
    }
  }

  return false;
}
int hungary() {
  memset(linker, -1, sizeof(linker));

  for (int u = 0; u < uN; u++) {  //   0~uN-1
    memset(used, false, sizeof(used));

    if (dfs(u)) {
      res++;
    }
  }

  return res;
}
```

5.4 maxflow

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

const int MAXN = 100010;  //
const int MAXM = 400010;  //
const int oo = 0x3f3f3f3f;
struct Edge {
  int to, next, cap, flow;
} edge[MAXM];  //  MAXM
int tol;
int head[MAXN];
int gap[MAXN], dep[MAXN], cur[MAXN];
void init() {
  tol = 0;
  memset(head, -1, sizeof(head));
}
void addedge(int u, int v, int w, int rw = 0) {
  edge[tol].to = v;
  edge[tol].cap = w;
  edge[tol].flow = 0;
  edge[tol].next = head[u];
  head[u] = tol++;
  edge[tol].to = u;
  edge[tol].cap = rw;
  edge[tol].flow = 0;
  edge[tol].next = head[v];
  head[v] = tol++;
}
int Q[MAXN];
void BFS(int ss, int tt) {
  memset(dep, -1, sizeof(dep));
  memset(gap, 0, sizeof(gap));
  gap[0] = 1;
  int front = 0, rear = 0;
  dep[tt] = 0;
  Q[rear++] = tt;

  while (front != rear) {
    int u = Q[front++];

    for (int i = head[u]; i != -1; i = edge[i].next) {
      int v = edge[i].to;

      if (dep[v] != -1) {
        continue;
      }

      Q[rear++] = v;
      dep[v] = dep[u] + 1;
      gap[dep[v]]++;
    }
  }
}
int S[MAXN];
int sap(int ss, int tt, int N) {
  BFS(ss, tt);
  memcpy(cur, head, sizeof(head));
  int top = 0;
  int u = ss;
  int ans = 0;

  while (dep[ss] < N) {
    if (u == tt) {
      int mi = oo;
      int inser;

      for (int i = 0; i < top; i++)
        if (mi > edge[S[i]].cap - edge[S[i]].flow) {
          mi = edge[S[i]].cap - edge[S[i]].flow;
          inser = i;
        }

      for (int i = 0; i < top; i++) {
        edge[S[i]].flow += mi;
        edge[S[i] ^ 1].flow -= mi;
      }

      ans += mi;
      top = inser;
      u = edge[S[top] ^ 1].to;
      continue;
    }

    bool flag = false;
    int v;

    for (int i = cur[u]; i != -1; i = edge[i].next) {
      v = edge[i].to;

      if (edge[i].cap - edge[i].flow && dep[v] + 1 == dep[u]) {
        flag = true;
        cur[u] = i;
        break;
      }
    }

    if (flag) {
      S[top++] = cur[u];
      u = v;
      continue;
    }

    int mi = N;

    for (int i = head[u]; i != -1; i = edge[i].next)
      if (edge[i].cap - edge[i].flow && dep[edge[i].to] < mi) {
        mi = dep[edge[i].to];
        cur[u] = i;
      }

    gap[dep[u]]--;

    if (!gap[dep[u]]) {
      return ans;
    }

    dep[u] = mi + 1;
    gap[dep[u]]++;

    if (u != ss) {
      u = edge[S[--top] ^ 1].to;
    }
  }

  return ans;
}
```

5.5 costflow

```
const int MAXN = 10000;
const int MAXM = 100000;
const int INF = 0x3f3f3f3f;
struct Edge {
  int to, next, cap, flow, cost;
} edge[MAXM];
int head[MAXN], tol;
int pre[MAXN], dis[MAXN];
bool vis[MAXN];
int N;   //       ,0~N-1
void init(int n) {
  N = n;
  tol = 0;
  memset(head, -1, sizeof(head));
}
void addedge(int u, int v, int cap, int cost) {
  edge[tol].to = v;
  edge[tol].cap = cap;
  edge[tol].cost = cost;
  edge[tol].flow = 0;
  edge[tol].next = head[u];
  head[u] = tol++;
  edge[tol].to = u;
  edge[tol].cap = 0;
  edge[tol].cost = -cost;
  edge[tol].flow = 0;
  edge[tol].next = head[v];
  head[v] = tol++;
}
bool spfa(int s, int t) {
  queue<int> q;

  for (int i = 0; i < N; i++) {
    dis[i] = INF;
    vis[i] = false;
    pre[i] = -1;
  }

  dis[s] = 0;
  vis[s] = true;
  q.push(s);

  while (!q.empty()) {
    int u = q.front();
    q.pop();
    vis[u] = false;

    for (int i = head[u]; i != -1; i = edge[i].next) {
      int v = edge[i].to;

      if (edge[i].cap > edge[i].flow && dis[v] > dis[u] + edge[i].cost) {
        dis[v] = dis[u] + edge[i].cost;
        pre[v] = i;

        if (!vis[v]) {
          vis[v] = true;
          q.push(v);
        }
      }
    }
  }

  if (pre[t] == -1) {
    return false;
  } else {
    return true;
  }
}
//     ,    cost
int minCostMaxflow(int s, int t, const int &cost) {
  int flow = 0;
  cost = 0;

  while (spfa(s, t)) {
    int Min = INF;

    for (int i = pre[t]; i != -1; i = pre[edge[i ^ 1].to]) {
      if (Min > edge[i].cap - edge[i].flow) {
        Min = edge[i].cap - edge[i].flow;
      }
    }

    for (int i = pre[t]; i != -1; i = pre[edge[i ^ 1].to]) {
      edge[i].flow += Min;
      edge[i ^ 1].flow -= Min;
      cost += edge[i].cost * Min;
    }

    flow += Min;
  }

  return flow;
}
```

5.6 min tree graph

```
const int INF = 0x3f3f3f3f;
const int MAXN = 1010;
const int MAXM = 40010;
struct Edge {
  int u, v, cost;
};
Edge edge[MAXM];
int pre[MAXN], id[MAXN], visit[MAXN], in[MAXN];
int zhuliu(int root, int n, int m, Edge edge[]) {
  int res = 0, u, v;

  while (1) {
    for (int i = 0; i < n; i++) {
      in[i] = INF;
    }

    for (int i = 0; i < m; i++)
      if (edge[i].u != edge[i].v && edge[i].cost < in[edge[i].v]) {
        pre[edge[i].v] = edge[i].u;
        in[edge[i].v] = edge[i].cost;
      }

    for (int i = 0; i < n; i++)
      if (i != root && in[i] == INF) {
        return -1;      //
      }

    int tn = 0;
    memset(id, -1, sizeof(id));
    memset(visit, -1, sizeof(visit));
    in[root] = 0;

    for (int i = 0; i < n; i++) {
      res += in[i];
      v = i;

      while (visit[v] != i && id[v] == -1 && v != root) {
        visit[v] = i;
        v = pre[v];
      }

      if (v != root && id[v] == -1) {
        for (int u = pre[v]; u != v; u = pre[u]) {
          id[u] = tn;
        }

        id[v] = tn++;
      }
    }

    if (tn == 0) {
      break;     //
    }

    for (int i = 0; i < n; i++)
      if (id[i] == -1) {
        id[i] = tn++;
      }

    for (int i = 0; i < m;) {
      v = edge[i].v;
      edge[i].u = id[edge[i].u];
      edge[i].v = id[edge[i].v];

      if (edge[i].u != edge[i].v) {
        edge[i++].cost -= in[v];
      } else {
        swap(edge[i], edge[--m]);
      }
    }

    n = tn;
    root = id[root];
  }

  return res;
}
```

5.7 flowertree

```
const int MAXN = 250;
int N;  //       ,1N
bool Graph[MAXN][MAXN];
int Match[MAXN];
bool InQueue[MAXN], InPath[MAXN], InBlossom[MAXN];
int Head, Tail;
int Queue[MAXN];
int Start, Finish;
int NewBase;
int Father[MAXN], Base[MAXN];
int Count;  //       ,Count/2
void CreateGraph() {
  int u, v;
  memset(Graph, false, sizeof(Graph));
  scanf(”%d”, &N);

  while (scanf(”%d%d”, &u, &v) == 2) {
    Graph[u][v] = Graph[v][u] = true;
  }
}
void Push(int u) {
  Queue[Tail] = u;
  Tail++;
  InQueue[u] = true;
}
int Pop() {
  int res = Queue[Head];
  Head++;
  return res;
}
int FindCommonAncestor(int u, int v) {
  memset(InPath, false, sizeof(InPath));

  while (true) {
    u = Base[u];
    InPath[u] = true;

    if (u == Start) {
      break;
    }

    u = Father[Match[u]];
  }

  while (true) {
    v = Base[v];

    if (InPath[v]) {
      break;
    }

    v = Father[Match[v]];
  }

  return v;
}
void ResetTrace(int u) {
  while (Base[u] != NewBase) {
    int v = Match[u];
    InBlossom[Base[u]] = InBlossom[Base[v]] = true;
    u = Father[v];

    if (Base[u] != NewBase) {
      Father[u] = v;
    }
  }
}
void BloosomContract(int u, int v) {
  NewBase = FindCommonAncestor(u, v);
  memset(InBlossom, false, sizeof(InBlossom));
  ResetTrace(u);
  ResetTrace(v);

  if (Base[u] != NewBase) {
    Father[u] = v;
  }

  if (Base[v] != NewBase) {
    Father[v] = u;
  }

  for (int tu = 1; tu <= N; tu++)
    if (InBlossom[Base[tu]]) {
      Base[tu] = NewBase;

      if (!InQueue[tu]) {
        Push(tu);
      }
    }
}
```

```

void FindAugmentingPath() {
  memset(InQueue, false, sizeof(InQueue));
  memset(Father, 0, sizeof(Father));

  for (int i = 1; i <= N; i++) {
    Base[i] = i;
  }

  Head = Tail = 1;
  Push(Start);
  Finish = 0;

  while (Head < Tail) {
    int u = Pop();

    for (int v = 1; v <= N; v++)
      if (Graph[u][v] && (Base[u] != Base[v]) && (Match[u] != v)) {
        if ((v == Start) || ((Match[v] > 0) && Father[Match[v]] > 0)) {
          BloosomContract(u, v);
        } else if (Father[v] == 0) {
          Father[v] = u;

          if (Match[v] > 0) {
            Push(Match[v]);
          } else {
            Finish = v;
            return;
          }
        }
      }
  }
}
void AugmentPath() {
  int u = Finish;

  while (u > 0) {
    int v = Father[u];
    int w = Match[v];
    Match[v] = u;
    Match[u] = v;
    u = w;
  }
}
void Edmonds() {
  memset(Match, 0, sizeof(Match));

  for (int u = 1; u <= N; u++)
    if (Match[u] == 0) {
      Start = u;
      FindAugmentingPath();

      if (Finish > 0) {
        AugmentPath();
      }
    }
}
void PrintMatch() {
  Count = 0;

  for (int u = 1; u <= N; u++)
    if (Match[u] > 0) {
      Count++;
    }

  printf(”%d\n”, Count);

  for (int u = 1; u <= N; u++)
    if (u < Match[u]) {
      printf("%d %d\n", u, Match[u]);
    }
}
int main() {
  CreateGraph();   //
  Edmonds();       //
  PrintMatch();    //
  return 0;
}
```

5.8 2-sat

```
const int MAXN = 20020;
const int MAXM = 100010;
struct Edge {
  int to, next;
} edge[MAXM];
int head[MAXN], tot;
void init() {
  tot = 0;
  memset(head, -1, sizeof(head));
}
void addedge(int u, int v) {
  edge[tot].to = v;
  edge[tot].next = head[u];
  head[u] = tot++;
}
bool vis[MAXN];     //     ,  true
int S[MAXN], top;   //
bool dfs(int u) {
  if (vis[u ^ 1]) {
    return false;
  }

  if (vis[u]) {
    return true;
  }

  vis[u] = true;
  S[top++] = u;

  for (int i = head[u]; i != -1; i = edge[i].next)
    if (!dfs(edge[i].to)) {
      return false;
    }

  return true;
}
bool Twosat(int n) {
  memset(vis, false, sizeof(vis));

  for (int i = 0; i < n; i += 2) {
    if (vis[i] || vis[i ^ 1]) {
      continue;
    }

    top = 0;

    if (!dfs(i)) {
      while (top) {
        vis[S[--top]] = false;
      }

      if (!dfs(i ^ 1)) {
        return false;
      }
    }
  }

  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++) {
      slack[i] = INF;
    }

    while (true) {
      memset(visx, false, sizeof(visx));
      memset(visy, false, sizeof(visy));

      if (DFS(x)) {
        break;
      }

      int d = INF;

      for (int i = 0; i < ny; i++)
        if (!visy[i] && d > slack[i]) {
          d = slack[i];
        }

      for (int i = 0; i < nx; i++)
        if (visx[i]) {
          lx[i] -= d;
        }

      for (int i = 0; i < ny; i++) {
        if (visy[i]) {
          ly[i] += d;
        } else {
          slack[i] -= d;
        }
      }
    }
  }

  int res = 0;

  for (int i = 0; i < ny; i++)
    if (linker[i] != -1) {
      res += g[linker[i]][i];
    }

  return res;
}
```