Hanyang University AlChoHol Teamnote

Strongly Connected Component & Bi-connected Component 0020

```
cc::graph[x].push_back(y); // 정점 x와 y가 연결됨

result = cc::scc(size); // Strongly Connected Component의 개수
f = (connected[i] == connected[j]); // 정점 i와 j가 같은 SCC에 속하는가?

cc::bcc(size); n = cc::cut_vertex_num; // 절점의 개수
b = cc::cut_vertex[i]; // 정점 i가 절점인가?
n = cc::cut_edge_num; // 절선의 개수
p = cc::cut_edge[i][0], q = cc::cut_edge[i][1]; // i번째 절선 p-q
```

```
#include <cstdlib>
#include <vector>
using namespace std;

namespace cc {
   const int SIZE = 10000;

   vector<int> graph[SIZE];
   int connected[SIZE];
   int cut_vertex_num;
   bool cut_vertex[SIZE];
   int cut_edge_num, cut_edge[SIZE][2];

   int order[SIZE];
   int visit_time[SIZE], finish[SIZE], back[SIZE];
   int stack[SIZE], seen[SIZE];

   #define MIN(a,b) (a) = ((a)<(b))?(a):(b)
   int dfs(int size) {
      int top, cnt, cnt2, cnt3;
      int i;
      cnt = cnt2 = cnt3 = 0;
      stack[0] = 0;
      for (i = 0 ; i < size ; i++) visit_time[i] = -1;
      for (i = 0 ; i < size ; i++) cut_vertex[i] = false; // CUT VERTEX
      cut_edge_num = 0; // CUT_EDGE
      for (i = 0 ; i < size ; i++) {
         if (visit_time[order[i]] == -1) {
            top = 1;
            stack[top] = order[i];
            seen[top] = 0;
            visit_time[order[i]] = cnt++;
            connected[order[i]] = cnt3++;
            int root_child = 0; // CUT VERTEX
            while (top > 0) {
               int j, now = stack[top];
               if (seen[top] == 0) back[now] = visit_time[now]; // NOT FOR SCC
               for (j = seen[top] ; j < graph[now].size() ; j++) {
                  int next = graph[now][j];
                  if (visit_time[next] == -1) {
                  if (top == 1) root_child++; // CUT VERTEX
                     seen[top] = j + 1;
                     stack[++top] = next;
                     seen[top] = 0;
                     visit_time[next] = cnt++;
                     connected[next] =
                     connected[now];
                     break;
                  }
                  else if (top == 1 || next != stack[top - 1]) // NOT FOR SCC
                     MIN(back[now], visit_time[next]); // NOT FOR SCC
               }
               if (j == graph[now].size()) {
                  finish[cnt2++] = now; // NOT FOR BCC
                  top--;
                  if (top > 1) {
                     MIN(back[stack[top]], back[now]); // NOT FOR SCC
                     if (back[now] >= visit_time[stack[top]]) { // CUT VERTEX
                        cut_vertex[stack[top]] = true;
                        cut_vertex_num++;
                     }
                  }                    // CUT EDGE
                  if (top > 0 && visit_time[stack[top]] < back[now]) {
                     cut_edge[cut_edge_num][0] = stack[top];
                     cut_edge[cut_edge_num][1] = now;
                     cut_edge_num++;
                  }
               }
            }
            if (root_child > 1) { // CUT VERTEX
```

```
                cut_vertex[order[i]] = true;
                cut_vertex_num++;
            }
         }
     }
     return cnt3; // number of connected component
}
#undef MIN

vector<int> graph_rev[SIZE];
void graph_reverse(int size) {
   for (int i = 0 ; i < size ; i++) graph_rev[i].clear();
   for (int i = 0 ; i < size ; i++)
      for (int j = 0 ; j < graph[i].size() ; j++)
         graph_rev[graph[i][j]].push_back(i);
   for (int i = 0 ; i < size ; i++) graph[i] = graph_rev[i];
}

int scc(int size) {
   int n;
   for (int i = 0 ; i < size ; i++) order[i] = i;
   dfs(size);
   graph_reverse(size);
   for (int i = 0 ; i < size ; i++) order[i] = finish[size - i - 1];
   n = dfs(size);
   graph_reverse(size);
   return n;
}

void bcc(int size) {
   for (int i = 0 ; i < size ; i++) order [ i ] = i;
   dfs(size);
   cut_vertex_num = 0;
   for (int i = 0 ; i < size ; i++)
      if (cut_vertex[i])
         cut_vertex_num++;
}
} // namespace cc
```

Hungarian Method

```
hungarian::n = XX; // 정점 개수
hungarian::cost[i][j] = XX; // 비용 테이블
result = hungarian::hungarian(); // 최대 매칭
y = hungarian::xy[x]; // 정점 x와 연결된 정점 번호
x = hungarian::yx[y]; // 정점 y와 연결된 정점 번호
```

```
#include <cstring>
#include <queue>
#include <algorithm>
#include <limits>
using namespace std;
namespace hungarian {
    typedef double val_t;
    const int SIZE = 100;
    const val_t INF = numeric_limits<double>::infinity();
    //  두 값이 같은지 비교
    inline bool eq(val_t a, val_t b) {
        static const double eps = 1e
            - 9;
        return (a
            - eps < b && b < a + eps);
    }
    int n;
    val_t cost[SIZE][SIZE];
    int xy[SIZE], yx[SIZE];
    int match_num;
    val_t lx[SIZE], ly[SIZE];
    bool s[SIZE], t[SIZE];
    int prev[SIZE];
    val_t hungarian() {
        memset(xy,
            -1, sizeof(xy));
        memset(yx,
            -1, sizeof(yx));
        memset(ly, 0, sizeof(ly));
        match_num = 0;
        int x, y;
        for (x = 0; x < n; x++) {
            lx[x] = cost[x][0];
```

```
    for (y = 1; y < n; y++)
        lx[x] = max(lx[x], cost[x][y]);

}
for (x = 0; x < n; x++)
    for (y = 0; y < n; y++)
        if (eq(cost[x][y], lx[x] + ly[y]) && yx[y] == -1) {
            xy[x] = y;
            yx[y] = x;
            match_num++;
            break;

        }
while (match_num < n) {
    memset(s, false, sizeof(s));
    memset(t, false, sizeof(t));
    memset(prev,
        -1, sizeof(prev));
    queue<int> q;
    for (x = 0; x < n; x++) {
        if (xy[x] ==
            -1) {
            q.push(x);
            s[x] = true;
            break;

        }
    }
    bool flg = false;
    while (!q.empty() && !flg) {
        x = q.front();
        q.pop();
        for (y = 0; y < n; y++) {
            if (eq(cost[x][y], lx[x] + ly[y])) {
                t[y] = true;
                if (yx[y] == -1) {
                    flg = true;
                    break;
                }
                if (!s[yx[y]]) {
                    s[yx[y]] = true;
                    q.push(yx[y]);
                    prev[yx[y]] = x;
                }
            }
        }
    }
    if (flg) {
        int t1, t2;
        while (x != -1) {
            t1 = prev[x];
            t2 = xy[x];
            xy[x] = y;
            yx[y] = x;
            x = t1;
            y = t2;
        }
        match_num++;
    }
    else {
        val_t alpha = INF;
        for (x = 0; x < n; x++) if (s[x])
            for (y = 0; y < n; y++) if (!t[y])
                alpha = min(alpha, lx[x] + ly[y] - cost[x][y]);
        for (x = 0; x < n; x++) if (s[x]) lx[x] -= alpha;
        for (y = 0; y < n; y++) if (t[y]) ly[y] += alpha;
    }
}
val_t ret = 0;
for (x = 0; x < n; x++)
    ret += cost[x][xy[x]];
return ret;
}
} // namespace hungarian
```

Extended GCD

ac + bd = gcd(a, b)가 되는 (c, d)를 찾는다.

Dependencies:

```
pair<long long, long long> extended_gcd(long long a, long long b) {
    if (b == 0) return make_pair(1, 0);
    pair<long long, long long> t = extended_gcd(b, a % b);
    return make_pair(t.second, t.first - t.second * (a / b));
```

```
}
```

Modular Inverse

ax = gcd(a, m) (mod m)가 되는 x를 찾는다.

Dependencies: extended_gcd(a, b)

```
long long modinverse(long long a, long long m) {
    return (extended_gcd(a, m).first % m + m) % m;
}
```

Chinese Remainder Theorem

x = a (mod n)가 되는 x를 찾는다.

Dependencies: gcd(a, b), modinverse(a, m)

```
long long chinese_remainder(long long *a, long long *n, int size) {
    if (size == 1) return *a;
    long long tmp = modinverse(n[0], n[1]);
    long long tmp2 = (tmp * (a[1] - a[0]) % n[1] + n[1]) % n[1];
    long long ora = a[1];
    long long tgcd = gcd(n[0], n[1]);
    a[1] = a[0] + n[0] / tgcd * tmp2;
    n[1] *= n[0] / tgcd;
    long long ret = chinese_remainder(a + 1, n + 1, size - 1);
    n[1] /= n[0] / tgcd;
    a[1] = ora;
    return ret;
}
```

Catalan Number

Dependencies: binomial(n, m)

```
long long catalan_number(int n) {
    return binomial(n * 2, n) / (n + 1);
}
```

Euler's Totient Function

phi(n), n 이하의 양수 중 n과 서로 소인 것의 개수를 구한다.

Dependencies: -

```
// phi(n) = (p_1 - 1) * p_1 ^ (k_1 - 1) * (p_2 - 1) * p_2 ^ (k_2-1)
long long euler_totient2(long long n, long long ps) {
    for (long long i = ps; i * i <= n; i++) {
        if (n % i == 0) {
            long long p = 1;
            while (n % i == 0) {
                n /= i;
                p *= i;
            }
            return (p - p / i) * euler_totient2(n, i + 1);
        }
        if (i > 2) i++;
    }
    return n - 1;
}
long long euler_totient(long long n) {
    return euler_totient2(n, 2);
}
```

Matrix Inverse

Dependencies: -

```
inline bool eq(double a, double b) {
    static const double eps = 1e-9;
    return fabs(a - b) < eps;
}
// returns empty vector if fails
vector<vector<double> > mat_inverse(vector<vector<double> > matrix, int n) {
    int i, j, k;
    vector<vector<double> > ret;
    ret.resize(n);
    for (i = 0; i < n; i++) {
        ret[i].resize(n);
        for (j = 0; j < n; j++)
            ret[i][j] = 0;
        ret[i][i] = 1;
    }
    for (i = 0; i < n; i++) {
        if (eq(matrix[i][i], 0)) {
            for (j = i + 1; j < n; j++) {
                if (!eq(matrix[j][i], 0)) {
                    for (k = 0; k < n; k++) {
                        matrix[i][k] += matrix[j][k];
                        ret[i][k] += ret[j][k];
```

```
                }
                break;
            }
        }
        if (j == n) {
            ret.clear();
            return ret;
        }
    }
    double tmp = matrix[i][i];
    for (k = 0; k < n; k++) {
        matrix[i][k] /= tmp;
        ret[i][k] /= tmp;
    }
    for (j = 0; j < n; j++) {
        if (j == i) continue;
        tmp = matrix[j][i];
        for (k = 0; k < n; k++) {
            matrix[j][k] -= matrix[i][k] * tmp;
            ret[j][k] -= ret[i][k] * tmp;
        }
    }
}
return ret;
}
```

Matrix Determinants

Dependencies: -

```
double mat_det(vector<vector<double> > matrix, int n) {
    int i, j, k;
    double ret = 1;
    for (i = 0; i < n; i++) {
        if (eq(matrix[i][i], 0)) {
            for (j = i + 1; j < n; j++) {
                if (!eq(matrix[j][i], 0)) {
                    for (k = 0; k < n; k++)
                        matrix[i][k] += matrix[j][k];
                    break;
                }
            }
            if (j == n)
                return 0;
        }
        double tmp = matrix[i][i];
        for (k = 0; k < n; k++)
            matrix[i][k] /= tmp;
            ret *= tmp;
        for (j = 0; j < n; j++) {
            if (j == i) continue;
            tmp = matrix[j][i];
            for (k = 0; k < n; k++)
                matrix[j][k] -= matrix[i][k] * tmp;
        }
    }
    return ret;
}
```

Convex Hull (Subset of Geometry Library)

hull = convex_hull(points); // convex hull의 꼭지점 좌표 vector

정수 좌표를 사용하고 싶다면 모든 double을 int나 long long으로 치환하라.

```
#include <cmath>
#include <vector>
#include <algorithm>
using namespace std;
const double eps = 1e-9;
inline int diff(double lhs, double rhs) {
    if (lhs - eps < rhs && rhs < lhs + eps) return 0;
    return (lhs < rhs) ? -1 : 1;
}
struct Point {
    double x, y;
    Point() {}
    Point(double x_, double y_) : x(x_), y(y_) {}
};
inline int ccw(const Point& a, const Point& b, const Point& c) {
    return diff(a.x * b.y + b.x * c.y + c.x * a.y
        - a.y * b.x - b.y * c.x - c.y * a.x, 0);
}
inline double dist2(const Point &a, const Point &b) {
    double dx = a.x - b.x;
```

```
    double dy = a.y - b.y;
    return dx * dx + dy * dy;
}
struct PointSorter {
    Point origin;
    PointSorter(const vector<Point>& points) {
        origin = points[0];
        for (int i = 1; i < points.size(); i++) {
            int det = diff(origin.x, points[i].x);
            if (det > 0)
                origin = points[i];
            else if (det == 0 && diff(origin.y, points[i].y) > 0)
                origin = points[i];
        }
    }
    bool operator()(const Point &a, const Point &b) {
        if (diff(b.x, origin.x) == 0 && diff(b.y, origin.y) == 0) return false;
        if (diff(a.x, origin.x) == 0 && diff(a.y, origin.y) == 0) return true;
        int det = ccw(origin, a, b);
        if (det == 0) return dist2(a, origin) < dist2(b, origin);
        return det < 0;
    }
};
vector<Point> convex_hull(vector<Point> points) {
    if (points.size() <= 3)
        return points;
    PointSorter cmp(points);
    sort(points.begin(), points.end(), cmp);
    vector<Point> ans;
    ans.push_back(points[0]);
    ans.push_back(points[1]);
    for (int i = 2; i < points.size(); i++) {
        while (ans.size() > 1 &&
            ccw(ans[ans.size() - 2], ans[ans.size() - 1], points[i]) >= 0)
            ans.pop_back();
        ans.push_back(points[i]);
    }
    return ans;
}
```

General Geometry Library

```
#include <cmath>
#include <vector>
using namespace std;
const double eps = 1e-9;
inline int diff(double lhs, double rhs) {
    if (lhs - eps < rhs && rhs < lhs + eps) return 0;
    return (lhs < rhs) ? -1 : 1;
}
inline bool is_between(double check, double a, double b) {
    if (a < b)
        return (a - eps < check && check < b + eps);
    else
        return (b - eps < check && check < a + eps);
}
struct Point {
    double x, y;
    Point() {}
    Point(double x_, double y_) : x(x_), y(y_) {}
    bool operator==(const Point& rhs) const {
        return diff(x, rhs.x) == 0 && diff(y, rhs.y) == 0;
    }
    const Point operator+(const Point& rhs) const {
        return Point(x + rhs.x, y + rhs.y);
    }
    const Point operator-(const Point& rhs) const {
        return Point(x - rhs.x, y - rhs.y);
    }
    const Point operator*(double t) const {
        return Point(x * t, y * t);
    }
};
struct Circle {
    Point center;
    double r;
    Circle() {}
    Circle(const Point& center_, double r_) : center(center_), r(r_) {}
};
struct Line {
```

```
    Point pos, dir;
    Line() {}
    Line(const Point& pos_, const Point& dir_) : pos(pos_), dir(dir_) {}
};
inline double inner(const Point& a, const Point& b) {
    return a.x * b.x + a.y * b.y;
}
inline double outer(const Point& a, const Point& b) {
    return a.x * b.y - a.y * b.x;
}
inline int ccw_line(const Line& line, const Point& point) {
    return diff(outer(line.dir, point - line.pos), 0);
}
inline int ccw(const Point& a, const Point& b, const Point& c) {
    return diff(outer(b - a, c - a), 0);
}
inline double dist(const Point& a, const Point& b) {
    return sqrt(inner(a - b, a - b));
}
inline double dist2(const Point &a, const Point &b) {
    return inner(a - b, a - b);
}
inline double dist(const Line& line, const Point& point, bool segment = false) {
    double c1 = inner(point - line.pos, line.dir);
    if (segment && diff(c1, 0) <= 0) return dist(line.pos, point);
    double c2 = inner(line.dir, line.dir);
    if (segment && diff(c2, c1) <= 0) return dist(line.pos + line.dir, point);
    return dist(line.pos + line.dir * (c1 / c2), point);
}
bool get_cross(const Line& a, const Line& b, Point& ret) {
    double mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    double t2 = outer(a.dir, b.pos - a.pos) / mdet;
    ret = b.pos + b.dir * t2;
    return true;
}
bool get_segment_cross(const Line& a, const Line& b, Point& ret) {
    double mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    double t1 = -outer(b.pos - a.pos, b.dir) / mdet;
    double t2 = outer(a.dir, b.pos - a.pos) / mdet;
    if (!is_between(t1, 0, 1) || !is_between(t2, 0, 1)) return false;
    ret = b.pos + b.dir * t2;
    return true;
}
const Point inner_center(const Point &a, const Point &b, const Point &c) {
    double wa = dist(b, c), wb = dist(c, a), wc = dist(a, b);
    double w = wa + wb + wc;
    return Point(
        (wa * a.x + wb * b.x + wc * c.x) / w,
        (wa * a.y + wb * b.y + wc * c.y) / w);
}
const Point outer_center(const Point &a, const Point &b, const Point &c) {
    Point d1 = b - a, d2 = c - a;
    double area = outer(d1, d2);
    double dx = d1.x * d1.x * d2.y - d2.x * d2.x * d1.y
        + d1.y * d2.y * (d1.y - d2.y);
    double dy = d1.y * d1.y * d2.x - d2.y * d2.y * d1.x
        + d1.x * d2.x * (d1.x - d2.y);
    return Point(a.x + dx / area / 2.0, a.y - dy / area / 2.0);
}
vector<Point> circle_line(const Circle& circle, const Line& line) {
    vector<Point> result;
    double a = 2 * inner(line.dir, line.dir);
    double b = 2 * (line.dir.x * (line.pos.x - circle.center.x)
        + line.dir.y * (line.pos.y - circle.center.y));
    double c = inner(line.pos - circle.center, line.pos - circle.center)
        - circle.r * circle.r;
    double det = b * b - 2 * a * c;
    int pred = diff(det, 0);
    if (pred == 0)
        result.push_back(line.pos + line.dir * (-b / a));
    else if (pred > 0) {
        det = sqrt(det);
        result.push_back(line.pos + line.dir * ((-b + det) / a));
        result.push_back(line.pos + line.dir * ((-b - det) / a));
    }
    return result;
}
vector<Point> circle_circle(const Circle& a, const Circle& b) {
    vector<Point> result;
    int pred = diff(dist(a.center, b.center), a.r + b.r);
```

```
    if (pred > 0) return result;
    if (pred == 0) {
        result.push_back((a.center * b.r + b.center * a.r) * (1 / (a.r + b.r)));
        return result;
    }
    double aa = a.center.x * a.center.x + a.center.y * a.center.y - a.r * a.r;
    double bb = b.center.x * b.center.x + b.center.y * b.center.y - b.r * b.r;
    double tmp = (bb - aa) / 2.0;
    Point cdiff = b.center - a.center;
    if (diff(cdiff.x, 0) == 0) {
        if (diff(cdiff.y, 0) == 0)
            return result; // if (diff(a.r, b.r) == 0): same circle
        return circle_line(a, Line(Point(0, tmp / cdiff.y), Point(1, 0)));
    }
    return circle_line(a,
        Line(Point(tmp / cdiff.x, 0), Point(-cdiff.y, cdiff.x)));
}
const Circle circle_from_3pts(const Point& a, const Point& b, const Point& c) {
    Point ba = b - a, cb = c - b;
    Line p((a + b) * 0.5, Point(ba.y, -ba.x));
    Line q((b + c) * 0.5, Point(cb.y, -cb.x));
    Circle circle;
    if (!get_cross(p, q, circle.center))
        circle.r = -1;
    else
        circle.r = dist(circle.center, a);
    return circle;
}
const Circle circle_from_2pts_rad(const Point& a, const Point& b, double r) {
    double det = r * r / dist2(a, b) - 0.25;
    Circle circle;
    if (det < 0)
        circle.r = -1;
    else {
        double h = sqrt(det);
        // center is to the left of a->b
        circle.center = (a + b) * 0.5 + Point(a.y - b.y, b.x - a.x) * h;
        circle.r = r;
    }
    return circle;
}
```

KMP Algorithm

```
result = kmp::match(text, pattern); // 모든 matched point의 vector
```

```
#include <vector>
using namespace std;
namespace kmp
{
    typedef vector<int> seq_t;
    void calculate_pi(vector<int>& pi, const seq_t& str) {
        pi[0] = -1;
        int j = -1;
        for (int i = 1; i < str.size(); i++) {
            while (j >= 0 && str[i] != str[j + 1]) j = pi[j];
            if (str[i] == str[j + 1])
                pi[i] = ++j;
            else
                pi[i] = -1;
        }
    }
    /* returns all positions matched */
    vector<int> match(seq_t text, seq_t pattern) {
        vector<int> pi(pattern.size());
        vector<int> ans;
        if (pattern.size() == 0) return ans;
        calculate_pi(pi, pattern);
        int j = -1;
        for (int i = 0; i < text.size(); i++) {
            while (j >= 0 && text[i] != pattern[j + 1]) j = pi[j];
            if (text[i] == pattern[j + 1]) {
                j++;
                if (j + 1 == pattern.size()) {
                    ans.push_back(i - j);
                    j = pi[j];
                }
            }
        }
        return ans;
    }
} // namespace kmp
```

Suffix Array O(n lg n)

```
#include <cstdio>
#include <algorithm>
using namespace std;
int n, K;
int dat[20003];
int ians[20003]; // ans -> index : 답의 반대
int ans[20003]; // index -> ans : 구하고자 하는 suffix array
int tmpans[20003]; // ans의 중간과정 저장
int bucket[20003]; // bucket -> index : starting points
int bucketcnt[20003]; // bucket -> count
int cntbucket; // number of buckets
int bucketmark[20003]; // ans -> bucket : 어느 bucket에 속하는가?
int bucketupdate[20003]; // ans -> bucketnumber. -1이면 새 거.
inline int sf(const int& a, const int& b) {
    return dat[a] < dat[b];
}
int main() {
    int i, H;
    scanf("%d%d", &n, &K);
    for (i = 0; i < n; i++) {
        scanf("%d", &dat[i]);
        dat[i]++;
        ans[i] = i;
        ians[i] = i;
    }
    // constructing suffix array by doubling method
    // phase 1: init
    sort(ans, ans + n, sf);
    for (i = 0; i < n; i++) {
        if (i == 0 || dat[ans[i]] != dat[ans[i - 1]]) {
            bucket[cntbucket] = i;
            bucketcnt[cntbucket] = 0;
            cntbucket++;
        }
        bucketmark[ans[i]] = cntbucket - 1;
    }
    // phase 2: doubling
    for (H = 1; ; H *= 2) {
        // phase 2-1: rearrangement
        // 현재 위치의 H만큼 뒤를 보면서 위치를 바꿈, 결과를 tmpans에 저장
        for (i = 0; i < n; i++) {
            if (ans[i] >= n - H) {
                // 이 뒤는 null 문자이므로 앞으로 가야 한다.
                int tbuck = bucketmark[ans[i]];
                bucketupdate[ans[i]] = -1;
                tmpans[bucket[tbuck] + bucketcnt[tbuck]] = ans[i];
                bucketcnt[tbuck]++;
            }
        }
        for (i = 0; i < n; i++) {
            if (ans[i] >= H) {
                // 위에서 처리하지 않은 나머지 것들
                int tbuck = bucketmark[ans[i] - H];
                bucketupdate[ans[i] - H] = bucketmark[ans[i]];
                tmpans[bucket[tbuck] + bucketcnt[tbuck]] = ans[i] - H;
                bucketcnt[tbuck]++;
            }
        }
        /* 만약 정확히 길이가 K인 문자열 중 중복되는 것의 개수를 세려고 한다면,
        * 여기서 처리하라. 그래야 bucketmark가 H인 상태로 남아 있고
        * (bucketmark가 같으면 그 자리에서 H글자만큼의 문자열은 같다는 뜻)
        * 정렬은 2H 길이를 기준으로 되어 있으니까, tmpans를 이용하기.
        * 부분 문자열의 길이 K는 H 이상 2 * H 이하여야 함. */
        // phase 2-2: identify new buckets
        int lastbucket = bucketmark[tmpans[0]];
        for (i = 1; i < n; i++) {
            if (bucket[bucketmark[tmpans[i]]] != i) {
                if (bucketupdate[tmpans[i]] != bucketupdate[tmpans[i - 1]]) {
                    // found new bucket
                    bucket[cntbucket] = i;
                    lastbucket = cntbucket;
                    cntbucket++;
                }
            }
            else {
                lastbucket = bucketmark[tmpans[i]];
            }
            bucketmark[tmpans[i]] = lastbucket;
        }
```

```
        // phase 2-3: copy ans and calculate ians
        int flg = 0;
        bucketmark[n] = -1;
        for (i = 0; i < n; i++) {
            if (bucketmark[tmpans[i]] == bucketmark[tmpans[i + 1]]) flg = 1;
            ans[i] = tmpans[i];
            ians[ans[i]] = i;
            bucketcnt[bucketmark[ans[i]]] = 0;
        }
        if (flg == 0) break;
    }
    return 0;
}
```

Suffix Array O(N lg^2 n) with LCP

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
// L: doubling method 정렬을 위한 정보
// P[stp][i]: 길이가 1 << stp인 원래 문자열의 위치 i부터 시작하는 버켓 번호
int N, i, stp, cnt;
int A[65536];
struct entry {
    int nr[2], p;
} L[65536];
int P[17][65536];
int suffix_array[65536];
int lcp[65536]; // lcp(i, i + 1)
int cmp(struct entry a, struct entry b) {
    return (a.nr[0] == b.nr[0]) ? (a.nr[1] < b.nr[1]) : (a.nr[0] < b.nr[0]);
}

// calclcp(x, y) = min(lcp[x], lcp[x + 1], ..., lcp[y - 1])
// binary indexed tree needed for speedup
int calclcp(int x, int y) { // x, y: start position in original string
    int k, ret = 0;
    if (x == y) return N - x;
    for (k = stp - 1; k >= 0 && x < N && y < N; k--)
        if (P[k][x] == P[k][y])
            x += 1 << k, y += 1 << k, ret += 1 << k;
    return ret;
}
int main(void) {
    int i;
    scanf("%d", &N);
    for (i = 0; i < N; i++) {
        scanf("%d", &A[i]);
        P[0][i] = A[i];
    }
    for (stp = 1, cnt = 1; (cnt >> 1) < N; stp++, cnt <<= 1) {
        for (i = 0; i < N; i++) {
            L[i].nr[0] = P[stp - 1][i];
            L[i].nr[1] = (i + cnt < N) ? P[stp - 1][i + cnt] : -1;
            L[i].p = i;
        }
        sort(L, L + N, cmp);
        for (i = 0; i < N; i++) {
            P[stp][L[i].p] = (i > 0 && L[i].nr[0] == L[i - 1].nr[0]
                && L[i].nr[1] == L[i - 1].nr[1]) ? P[stp][L[i - 1].p] : i;
        }
    }
    for (i = 0; i < N; i++)
        suffix_array[P[stp - 1][i]] = i;
    for (i = 0; i + 1 < N; i++)
        lcp[i] = calclcp(i, i + 1);
    return 0;
}
```

LIS Tracking

```
typedef pair<int, int> ii;

struct mycomp {
    bool operator() (const ii &l, const ii &r) const {
        return l.second < r.second;
    }
};
vector<ii> LIS(vector<ii> v) {
    map < ii, int, mycomp> m;
    map < ii, int, mycomp>::iterator k, l;
```

```
    vector<ii> res;

    const int N = v.size();
    vector<int> pre(N, -1);

    for (int i = 0; i < N; i++) {
        if (m.insert({ v[i], i }).second) {
            k = m.find(v[i]);
            l = k; k++;
            if (l == m.begin()) {
                pre[i] = -1;
            }
            else {
                l--;
                pre[i] = l->second;
            }
            if (k != m.end()) {
                m.erase(k);
            }
        }
    }
    k = m.end(); k--;
    int j = k->second;
    while (j != -1) {
        res.push_back(v[j]);
        j = pre[j];
    }
    reverse(res.begin(), res.end());
    return res;
}

int main(void) {
    int N; scanf("%d", &N);

    vector<ii> v;
    for (int i = 0; i < N; i++) {
        int a, b; scanf("%d %d", &a, &b);
        v.push_back({ a, b });
    }
    sort(v.begin(), v.end());
    auto r = LIS(v);
    auto it = r.begin();
    printf("%d\n", v.size() - r.size());
    for (auto e : v) {
        if (e != (*it)) {
            printf("%d\n", e.first);
        }
        else {
            it++;
        }
    }
}
```

MCST (Kruskal)

```
#include<stdio.h>
#include<queue>
#include<algorithm>
#pragma warning(disable:4996)
using namespace std;

int n, m;
priority_queue<pair<int, pair<int, int>>> Q;
int parent[10005], sum;

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) parent[i] = i;
    for (int i = 0; i < m; i++) {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        Q.push({ -c,{ a,b } });
    }

    while (!Q.empty()) {
        int x, y, z;
        x = Q.top().second.first;
        y = Q.top().second.second;
        z = Q.top().first * (-1);
        Q.pop();
        vector<int> path;
```

```
        while (parent[x] != x) {
            path.push_back(x);
            x = parent[x];
        }   path.push_back(x);
        while (parent[y] != y) {
            path.push_back(y);
            y = parent[y];
        }   path.push_back(y);

        if (x != y) {
            for (int i = 0; i < path.size(); i++) parent[path[i]] = x;
            sum += z;
        }
    }
    printf("%d", sum);

    return 0;
}
```

Farthest Points

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

#define x first
#define y second

using namespace std;

typedef pair<long long, long long> point;

vector<point> list;
vector<point> convex;
long long n, tx, ty;
long long res;
int mi;
int l, r;

double la, ra;
int nl, nr, rr, ll;
point o, lp, rp;

long long cp(point a, point b, point o) { // 벡터의 외적
   return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x);
}

long long dist(point a, point b) { // 두 점 사이의 거리의 제곱
   return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}

bool cmp_angle(point a, point b) { // 각도로 정렬, 각도가 같으면 거리가 가까운 것을 앞으로
   long long res = cp(a, b, list[0]);
   if (res == 0) {
      return dist(a, list[0]) < dist(b, list[0]);
   } else {
      return res > 0;
   }
}

long long dist_square(point a, point b) {
   return ((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

double ang(point a, point b) {
   return (double)(a.x * b.x + a.y * b.y) / (sqrt(a.x * a.x + a.y * a.y) * sqrt(b.x * b.x +
b.y * b.y));
}

void convexHull() {
   for (int i = 1; i < n; i++) { // y좌표가 제일 작은 점 찾기
      if (list[i].y < list[mi].y || list[i].y == list[mi].y && list[i].x < list[mi].x) {
         mi = i;
      }
   }

   swap(list[0], list[mi]);
   sort(list.begin() + 1, list.end(), cmp_angle); // 각도로 정렬

   convex.push_back(list[0]); // 첫 두 점을 convex hull에 넣음
```

```
   convex.push_back(list[1]);

   for (int i = 2; i < n; i++) {
      while (convex.size() >= 2 && cp(convex.back(), list[i], convex[convex.size() - 2]) <= 0)
{ // 현재 점이 마지막 직선보다 오른쪽에 있으면
         convex.pop_back(); // 마지막 점을 뺀다
      }
      convex.push_back(list[i]);
   }
}

void farthest() {
   for (int i = 0; i < convex.size(); i++) {
      if (convex[i].x > convex[rr].x) {
         rr = i;
      }
      if (convex[i].x < convex[ll].x) {
         ll = i;
      }
   }

   o = { 0, 1 };

   l = ll;
   r = rr;

   while (l != rr || r != ll) {
      nl = (l - 1 + convex.size()) % convex.size();
      nr = (r - 1 + convex.size()) % convex.size();
      lp = { convex[nl].x - convex[l].x, convex[nl].y - convex[l].y };
      rp = { convex[nr].x - convex[r].x, convex[nr].y - convex[r].y };
      la = ang(o, lp);
      ra = ang({ -o.x, -o.y }, rp);

      if (l == rr) {
         o = { -rp.x, -rp.y };
         r = nr;
      } else if (r == ll) {
         o = lp;
         l = nl;
      } else if (la >= ra) {
         o = lp;
         l = nl;
      } else {
         o = { -rp.x, -rp.y };
         r = nr;
      }

      res = max(res, dist_square(convex[l], convex[r]));
   }
}

int main() {
   scanf("%lld", &n);

   for (int i = 0; i < n; i++) {
      scanf("%lld %lld", &tx, &ty);
      list.push_back({ tx, ty });
   }

   convexHull();
   farthest();

   printf("%.6f\n", sqrt(res));

   return 0;
}
```

Closest Points

```
#include <cstdio>
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
#include <cmath>

using namespace std;

const int MAX = 100000;
```

```
struct point {
   int x, y;
   point() {}
   point(int x, int y) : x(x), y(y) {}
   bool operator < (point p) const { // compare by y
      if (y == p.y) {
         return x < p.x;
      } else {
         return y < p.y;
      }
   }
};

bool cmp_by_x(point a, point b) {
   if (a.x == b.x) {
      return a.y < b.y;
   } else {
      return a.x < b.x;
   }
}

int dist(point a, point b) {
   return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}

int main() {
   int n, tx, ty, res, x, d;
   int start;
   vector<point> list;
   set<point> candidates;
   scanf("%d", &n);

   for (int i = 0; i < n; i++) {
      scanf("%d %d", &tx, &ty);
      list.push_back(point(tx, ty));
   }

   sort(list.begin(), list.end(), cmp_by_x); // 전체 점 배열은 x좌표로 정렬

   candidates.insert(list[0]);
   candidates.insert(list[1]);
   res = dist(list[0], list[1]);

   start = 0; // candidates에는 list[start]부터 list[n - 1]까지의 점이 들어감

   for (int i = 2; i < n; i++) {
      point now = list[i];

      while (start < i) {
         point p = list[start];
         x = now.x - p.x;
         if (x * x > res) { // candidates 안의 점이 수직선과 res 이상 떨어지게 되면
candidates에서 제거
            candidates.erase(p);
            start++;
         } else {
            break;
         }
      }

      d = (int)sqrt(res) + 1; // res에는 거리의 제곱이 저장되기 때문

      point lower_point = point(-MAX, now.y - d); // candidates의 위아래를 자름
      point upper_point = point(MAX, now.y + d);
      auto lower = candidates.lower_bound(lower_point);
      auto upper = candidates.upper_bound(upper_point);

      for (auto it = lower; it != upper; it++) {
         d = dist(now, *it);
         res = min(d, res); // res 갱신
      }

      candidates.insert(now);
   }

   printf("%d\n", res);

   return 0;
}
```

Segment Tree with Lazy Propagation

```
#include<stdio.h>
#pragma warning(disable: 4996)

long long n, tree[4000005], lazy[4000005];

void update(int node, int s, int e, int index, int val) {
    if (index < s || e < index) return;
    if (s == e) {
        tree[node] = val;
        return;
    }

    update(node * 2, s, (s + e) / 2, index, val);
    update(node * 2 + 1, (s + e) / 2 + 1, e, index, val);
    tree[node] = tree[node * 2] + tree[node * 2 + 1];
}

void update_lazy(int node, int s, int e) {
    if (lazy[node] != 0) {
        tree[node] += (e - s + 1)*lazy[node];
        if (s != e) {
            lazy[node * 2] += lazy[node];     lazy[node * 2 + 1] += lazy[node];
        }
        lazy[node] = 0;
    }
    return;
}

void update_range(int node, int s, int e, int l, int r, long long val) {
    update_lazy(node, s, e);

    if (l > e || r < s) return;
    if (l <= s & e <= r) {
        tree[node] += (e - s + 1)*val;
        if (s != e) {
            lazy[node * 2] += val;     lazy[node * 2 + 1] += val;
        }
        return;
    }

    update_range(node * 2, s, (s + e) / 2, l, r, val);
    update_range(node * 2 + 1, (s + e) / 2 + 1, e, l, r, val);
    tree[node] = tree[node * 2] + tree[node * 2 + 1];
}

long long sum(int node, int s, int e, int l, int r) {
    update_lazy(node, s, e);
    if (l > e || r < s) return 0;
    if (l <= s && e <= r) return tree[node];

    return sum(node * 2, s, (s + e) / 2, l, r) + sum(node * 2 + 1, (s + e) / 2 + 1, e, l, r);
}

int main() {
    int m, k;
    scanf("%lld%d%d", &n, &m, &k);
    for (int i = 0; i < n; i++) {
        int a;  scanf("%d", &a);
        update(1, 1, n, i + 1, a);
    }

    for (int i = 0; i < m + k; i++) {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        if (a == 1) {
            long long d;
            scanf("%lld", &d);
            update_range(1, 1, n, b, c, d);
        }
        else if (a == 2) {
            printf("%lld\n", sum(1, 1, n, b, c));
        }
    }

    return 0;
}
```

Plane Sweeping with Segment Tree

```
#include<stdio.h>
#include<vector>
#include<algorithm>
#pragma warning(disable:4996)

using namespace std;

struct line_info {
    int x, y1, y2, flag;

    bool operator() (line_info p, line_info q) {
        return p.x < q.x;
    }
};

struct Tree {
    int sum, cnt;
};

int n, s;
vector<line_info> v;
Tree t[500000];
vector<int> y_list;

void update(int node, int start, int end, int left, int right, int flag) {
    if (start > right || end < left) return;
    if (left <= start && end <= right) {
        t[node].cnt += flag;
    }
    else {
        update(node * 2, start, (start + end) / 2, left, right, flag);
        update(node * 2 + 1, (start + end) / 2 + 1, end, left, right, flag);
    }

    t[node].sum = 0;
    if (t[node].cnt > 0) t[node].sum = y_list[end] - y_list[start - 1];
    else if(start<end) t[node].sum = t[node * 2].sum + t[node * 2 + 1].sum;
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        int x1, x2, y1, y2;
        scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        line_info tmp;
        tmp.x = x1; tmp.y1 = y1;   tmp.y2 = y2;   tmp.flag = 1;
        v.push_back(tmp);
        tmp.x = x2; tmp.flag = -1;
        v.push_back(tmp);
        y_list.push_back(y1);
        y_list.push_back(y2);
    }
    sort(v.begin(), v.end(), line_info());
    sort(y_list.begin(), y_list.end());
    y_list.erase(unique(y_list.begin(), y_list.end()), y_list.end());

    int px = v[0].x;
    for (int i = 0; i < v.size(); i++) {
        s += t[1].sum*(v[i].x - px);

        int p1 = lower_bound(y_list.begin(), y_list.end(), v[i].y1) - y_list.begin() + 1;
        int p2 = lower_bound(y_list.begin(), y_list.end(), v[i].y2) - y_list.begin();

        update(1, 1, y_list.size(), p1, p2, v[i].flag);

        px = v[i].x;
    }

    printf("%d", s);

    return 0;
}
```

MaximumFlow Struct

```
struct MaximnumFlow {
    struct Edge {
        int from, to, cap, flow, rev;
        Edge(int from, int to, int cap) : from(from), to(to), cap(cap), flow(0) {}
    };
    vector<vector<Edge>> adj;
    vector<pair<int, int>> p;
    vector<bool> visit;
    int N, source, sink;
    MaximnumFlow(int N, int source, int sink) :N(N), source(source), sink(sink) {
        adj.resize(N);
        p.resize(N);
        visit.resize(N);
    }
    void add_edge(int from, int to, int cap) {
        adj[from].emplace_back(from, to, cap);
        adj[to].emplace_back(to, from, 0);
        adj[from].back().rev = adj[to].size() - 1;
        adj[to].back().rev = adj[from].size() - 1;
    }
    void add_edge_from_source(int to, int cap) {
        add_edge(source, to, cap);
    }
    void add_edge_to_sink(int from, int cap) {
        add_edge(from, sink, cap);
    }
    int Flow() {
        int ret = 0;
        queue<int> q;
        while (1) {
            int flow = INF;
            fill(p.begin(), p.end(), make_pair(-1, -1));
            fill(visit.begin(), visit.end(), false);
            q.push(source);
            visit[source] = true;
            while (!q.empty()) {
                int now = q.front();
                q.pop();
                for (int i = 0; i < adj[now].size(); i++) {
                    auto &e = adj[now][i];
                    if (e.cap - e.flow > 0 && !visit[e.to]) {
                        visit[e.to] = true;
                        p[e.to] = { e.from,i };
                        q.push(e.to);
                    }
                }
            }

            if (p[sink].first == -1) break;
            for (int now = sink; p[now].first != -1; now = p[now].first) {
                auto &e = adj[p[now].first][p[now].second];
                flow = min(flow, e.cap - e.flow);
            }
            for (int now = sink; p[now].first != -1; now = p[now].first) {
                auto &e = adj[p[now].first][p[now].second];
                e.flow += flow;
                adj[e.to][e.rev].flow -= flow;
            }
            ret += flow;
        }

        return ret;
    }
};

int n, k;

int main() {
    scanf("%d%d", &n, &k);
    MaximnumFlow mf(2 * n + 2, 2 * n, 2 * n + 1);
    for (int i = 0; i < n; i++) {
        mf.add_edge_from_source(i, 1);
        mf.add_edge_to_sink(n + i, 1);
    }
    for (int i = 0; i < k; i++) {
        int a, b;  scanf("%d%d", &a, &b);
        mf.add_edge(a - 1, n + b - 1, 1);
    }
```

```
    printf("%d", mf.Flow());

    return 0;
}
```

MCMF

```
#include<bits/stdc++.h>
using namespace std;
#define INF 1000000000

vector<vector<pair<int, int>>> adj;
vector<pair<int, int>> p;
vector<int> dist, w, b;
bool inq[1005];
int n, m = 2, c[1005][1005], f[1005][1005];

int main() {
    int x, y;
    while (~scanf("%d%d", &x, &y)) {
        w.push_back(x), b.push_back(y);
    }
    n = w.size();
    adj.resize(n + m + 2);
    for (int i = 1; i <= n; i++) {
        adj[0].push_back({ i,0 }), adj[i].push_back({ 0,0 });
        c[0][i] = 1;
        adj[i].push_back({ n + 1, -w[i - 1] }), adj[i].push_back({ n + 2, -b[i - 1] });
        adj[n + 1].push_back({ i, w[i - 1] }), adj[n + 2].push_back({ i,b[i - 1] });
        c[i][n + 1] = 1, c[i][n + 2] = 1;
    }
    adj[n + 1].push_back({ n + m + 1,0 }), adj[n + 2].push_back({ n + m + 1,0 });
    c[n + 1][n + m + 1] = c[n + 2][n + m + 1] = 15;

    int ans = 0;
    queue<int> Q;
    dist.resize(n + m + 2);
    p.resize(n + m + 2);
    while (1) {
        int flow = INF;
        fill(dist.begin(), dist.end(), INF);
        fill(p.begin(), p.end(), make_pair(-1, -1));
        Q.push(0);
        dist[0] = 0;

        while (!Q.empty()) {
            int now = Q.front();
            Q.pop();
            inq[now] = false;
            for (int i = 0; i < adj[now].size(); i++) {
                auto togo = adj[now][i];
                if (c[now][togo.first] - f[now][togo.first] > 0 && dist[togo.first] > dist[now]
+ togo.second) {
                    p[togo.first] = { now,i };
                    dist[togo.first] = dist[now] + togo.second;
                    if (!inq[togo.first]) {
                        inq[togo.first] = true;
                        Q.push(togo.first);
                    }
                }
            }
        }

        if (p[n + m + 1].first == -1) break;

        for (int now = n + m + 1; p[now].first != -1; now = p[now].first) {
            flow = min(flow, c[p[now].first][now] - f[p[now].first][now]);
        }
        for (int now = n + m + 1; p[now].first != -1; now = p[now].first) {
            auto i = adj[p[now].first][p[now].second];
            f[p[now].first][now] += flow;
            f[now][p[now].first] -= flow;
            ans += (-1)*i.second*flow;
        }
    }
    printf("%d\n", ans);

    return 0;
}
```

Bipartite Matching

```
#include<stdio.h>
#include<vector>
#pragma warning(disable:4996)
using namespace std;

vector<vector<int>> v;
vector<bool> visit;
vector<int> match;

bool BM(int x) {
    if (visit[x]) return false;
    visit[x] = true;
    for (int i = 0; i < v[x].size(); i++) {
        if (match[v[x][i] - 1] == -1 || BM(match[v[x][i] - 1])) {
            match[v[x][i] - 1] = x;
            return true;
        }
    }
    return false;
}

int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    v.resize(n);
    match = vector<int>(m, -1);
    for (int i = 0; i < n; i++) {
        int s;
        scanf("%d", &s);
        for (int j = 0; j < s; j++) {
            int c;
            scanf("%d", &c);
            v[i].push_back(c);
        }
    }

    int ans = 0;
    for (int i = 0; i < n; i++) {
        visit = vector<bool>(n, false);
        if (BM(i)) ans++;
    }

    printf("%d", ans);

    return 0;
}
```