

1 data_structure

1.1 Sparse_Table

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int n;
4 int v[1000009];
5 int sparse[22][1000009];
6 // O(nlogn) preprocess O(1)Query
7 // sp[x][y] is the answer from (v[x], v[x+2^y-1])
8 inline void init()
9 {
10     for (int i = 0; i < n; ++i)
11         sparse[0][i] = v[i];
12     for (int j = 1; (1 << j) <= n; ++j)
13         for (int i = 0; i + (1 << j) <= n; ++i)
14             sparse[j][i] = min(
15                 sparse[j - 1][i],
16                 sparse[j - 1][i + (1 << (j - 1))])
17             );
18 }
19
20 inline int query(int l, int r)
21 {
22     int k = __lg(r - l + 1);
23     return min(sparse[k][l], sparse[k][r - (1 << k) + 1]);
24 }

```

1.2 segment_Tree

```

1 #define LL long long
2 #define IL(X) ((X << 1) + 1)
3 #define IR(X) ((X << 1) + 2)
4 #define MAXN 500005
5 // add tag
6 // tag += tag
7 // val += tag*size
8
9 struct segID{
10     struct Node{
11         LL val;
12         LL lazy_tag;
13         int size;
14     };
15     LL dataseq[MAXN];
16     Node seq[MAXN * 4 + 5];
17     void pull(int index){
18         seq[index].val = seq[IL(index)].val + seq[IR(index)].val;
19     }
20     void push(int index){
21         seq[IL(index)].lazy_tag += seq[index].lazy_tag;
22         seq[IL(index)].val += seq[index].lazy_tag * seq[IL(index)].size;
23         seq[IR(index)].lazy_tag += seq[index].lazy_tag;
24         seq[IR(index)].val += seq[index].lazy_tag * seq[IR(index)].size;
25         seq[index].lazy_tag = 0;
26 }

```

```

27
28 void build(int L, int R, int index){
29     if(L == R){
30         seq[index].val = dataseq[L];
31         seq[index].size = 1;
32         seq[index].lazy_tag = 0;
33         return;
34     }
35     int M = (L + R) / 2;
36     build(L, M, IL(index));
37     build(M + 1, R, IR(index));
38     seq[index].size = seq[IL(index)].size + seq[IR(index)].size;
39     pull(index);
40 }
41
42 void modify(int l, int r, int L, int R, int index, long
43     long Add){
44     if(l == L && r == R){
45         seq[index].lazy_tag += Add;
46         seq[index].val += Add * seq[index].size;
47         return;
48     }
49     push(index);
50     int M = (L + R) / 2;
51
52     if(r <= M){
53         modify(l, r, L, M, IL(index), Add);
54     }else if(l > M){
55         modify(l, r, M + 1, R, IR(index), Add);
56     }else{
57         modify(l, M, L, M, IL(index), Add);
58         modify(M + 1, r, M + 1, R, IR(index), Add);
59     }
60     pull(index);
61 }
62
63 long long Query(int l, int r, int L, int R, int index){
64     if(l == L && r == R){
65         return seq[index].val;
66     }
67     int M = (L + R) / 2;
68     push(index);
69     if(r <= M){
70         return Query(l, r, L, M, IL(index));
71     }else if(l > M){
72         return Query(l, r, M + 1, R, IR(index));
73     }else{
74         return Query(l, M, L, M, IL(index)) +
75             Query(M + 1, r, M + 1, R, IR(index));
76     }
77 }
78 };

```

1.3 disjointset

```

1 #include <algorithm>
2 using namespace std;
3 #define MAX_N 200005
4 struct disjointset
5 {
6     int rank[MAX_N];

```

```

7     int f[MAX_N];
8     void init(int N){
9         for (int i = 0; i < N; ++i){
10             f[i] = i;
11             rank[i] = 1;
12         }
13     }
14     int find(int v){
15         if( f[v] == v)
16             return v;
17         return f[v] = find(f[v]);
18     }
19     bool same(int a, int b){
20         return find(a) == find(b);
21     }
22     void Union(int a, int b){
23         // f[find(a)] = find(b);
24         if(!same(a,b)){
25             if(rank[a] < rank[b])
26                 swap(a, b);
27             f[f[b]] = f[a];
28             rank[a]++;
29         }
30     }
31 }
32 };

```

2 geometry

2.1 closest_point

```

1 template <typename T>
2 T ClosestPairSquareDistance(typename vector<Point<T>>::
3     iterator l,
4     typename vector<Point<T>>::
5     iterator r)
6 {
7     auto delta = numeric_limits<T>::max();
8     if (r - l > 1)
9     {
10         auto m = l + (r - l >> 1);
11         nth_element(l, m, r); // Lexicographical order in
12             default
13         auto x = m->x;
14         delta = min(ClosestPairSquareDistance<T>(l, m),
15             ClosestPairSquareDistance<T>(m, r));
16         auto square = [=](T y) { return y * y; };
17         auto sgn = [=](T a, T b) {
18             return square(a - b) <= delta ? 0 : a < b ? -1 : 1;
19         };
20         vector<Point<T>> x_near[2];
21         copy_if(l, m, back_inserter(x_near[0]), [=](Point<T>
22             a) {
23                 return sgn(a.x, x) == 0;
24             });
25         copy_if(m, r, back_inserter(x_near[1]), [=](Point<T>
26             a) {
27                 return sgn(a.x, x) == 0;
28             });
29         for (int i = 0, j = 0; i < x_near[0].size(); ++i)

```

```

25 {
26     while (j < x_near[1].size() and
27           sgn(x_near[1][j].y, x_near[0][i].y) == -1)
28         ++j;
29     for (int k = j; k < x_near[1].size() and
30         sgn(x_near[1][k].y, x_near[0][i].y) == 0;
31         ++k)
32     {
33         delta = min(delta, (x_near[0][i] - x_near[1][k]).norm());
34     }
35     inplace_merge(l, m, r, [](Point<T> a, Point<T> b) {
36         return a.y < b.y;
37     });
38 }
39 return delta;
40 }
41 }

```

2.2 points

```

1 template <typename T>
2 struct Point
3 {
4     T x, y;
5     Point() : x(0), y(0) {}
6     Point(const T x, const T y) : x(x), y(y) {}
7     template <class F>
8     explicit operator Point<F>() const
9     {
10         return Point<F>((F)x, (F)y);
11     }
12
13     Point operator+(const Point b) const
14     {
15         return Point(x + b.x, y + b.y);
16     }
17     Point operator-(const Point b) const
18     {
19         return Point(x - b.x, y - b.y);
20     }
21     template <class F>
22     Point<F> operator*(const F fac)
23     {
24         return Point<F>(x * fac, y * fac);
25     }
26     template <class F>
27     Point<F> operator/(const F fac)
28     {
29         return Point<F>(x / fac, y / fac);
30     }
31
32     T operator&(const Point b) const { return x * b.x + y * b.y; }
33     // 內積運算子
34     T operator^(const Point b) const { return x * b.y - y * b.x; }
35     // 外積運算子
36     bool operator==(const Point b) const
37     {
38         return x == b.x and y == b.y;

```

```

39 }
40 bool operator<(const Point b) const
41 {
42     return x == b.x ? y < b.y : x < b.x;
43 } // 字典序
44
45 Point operator-() const { return Point(-x, -y); }
46 T norm() const { return *this & *this; } // 歐式長度平方
47 Point prep() const { return Point(-y, x); } // 左旋直角法向量
48
49 template <class F>
50 istream &operator>>(istream &is, Point<F> &pt)
51 {
52     return is >> pt.x >> pt.y;
53 }
54
55 template <class F>
56 ostream &operator<<(ostream &os, const Point<F> &pt)
57 {
58     return os << pt.x << " " << pt.y;
59 }

```

2.3 lines

```

1 template <typename T, typename Real = double>
2 struct Line
3 {
4     Point<T> st, ed;
5     Point<T> vec() const { return ed - st; }
6     T ori(const Point<T> p) const { return (ed - st) ^ (p - st); }
7     Line(const Point<T> x, const Point<T> y) : st(x), ed(y) {}
8     template <class F>
9     operator Line<F>() const
10    {
11        return Line<F>((Point<F>)st, (Point<F>)ed);
12    }
13
14    // sort by arg, the left is smaller for parallel lines
15    bool operator<(Line B) const
16    {
17        Point<T> a = vec(), b = B.vec();
18        auto sgn = [](const Point<T> t) { return (t.y == 0 ? t.x : t.y) < 0; };
19        if (sgn(a) != sgn(b))
20            return sgn(a) < sgn(b);
21        if (abs(a ^ b) == 0)
22            return B.ori(st) > 0;
23        return (a ^ b) > 0;
24    }
25
26    // Regard a line as a function
27    template <typename F>
28    Point<F> operator()(const F x) const
29    {
30        return Point<F>(st) + vec() * x;
31    }
32
33    bool isSegProperIntersection(const Line l) const

```

```

34 {
35     return l.ori(st) * l.ori(ed) < 0 and ori(l.st) * ori(l.ed) < 0;
36 }
37
38 bool isPointOnSegProperly(const Point<T> p) const
39 {
40     return ori(p) == 0 and ((st - p) & (ed - p)) < 0;
41 }
42
43 Point<Real> getIntersection(const Line<Real> l)
44 {
45     Line<Real> h = *this;
46     return l((l.st ^ h.vec()) / (h.vec() ^ l.vec()));
47 }
48
49 Point<Real> projection(const Point<T> p) const
50 {
51     return operator()(((p - st) & vec()) / (Real)(vec().norm()));
52 }
53 };

```

2.4 geometry_template

```

1 // import from https://codeforces.com/blog/entry/48122
2 #include <iostream>
3 using namespace std;
4 template <class F>
5 struct Point {
6     F x, y;
7     Point() : x(0), y(0) {}
8     Point(const F& x, const F& y) : x(x), y(y) {}
9
10    void swap(Point& other) { using std::swap; swap(x, other.x); swap(y, other.y); }
11    template <class F1> explicit operator Point<F1>() const {
12        return Point<F1>(static_cast<F1>(x), static_cast<F1>(y));
13    }
14    template <class F1> Point& operator = (const Point<F1>& other) {
15        x = other.x; y = other.y; return *this; }
16    template <class F1> Point& operator += (const Point<F1>& other) {
17        x += other.x; y += other.y; return *this; }
18    template <class F1> Point& operator -= (const Point<F1>& other) {
19        x -= other.x; y -= other.y; return *this; }
20    template <class F1> Point& operator *= (const F1& factor) {
21        x *= factor; y *= factor; return *this; }
22    template <class F1> Point& operator /= (const F1& factor) {
23        x /= factor; y /= factor; return *this; }
24    };
25
26    template <class F> int read(Point<F>& point) { return read(point.x, point.y) / 2; }
27    template <class F> int write(const Point<F>& point) { return write(point.x, point.y); }
28
29    template <class F> istream& operator >> (istream& is, Point<F>& point) {
30        return is >> point.x >> point.y; }

```

```

30 template <class F> ostream& operator << (ostream& os, const
    Point<F>& point) {
31     return os << point.x << ' ' << point.y; }
32
33 template <class F> inline Point<F> makePoint(const F& x,
    const F& y) { return Point<F>(x, y); }
34 template <class F> void swap(Point<F>& lhs, Point<F>& rhs) {
    lhs.swap(rhs); }
35
36 #define FUNC1(name, arg, expr) \
37 template <class F> inline auto name(const arg) -> decltype(
    expr) { return expr; }
38 #define FUNC2(name, arg1, arg2, expr) \
39 template <class F1, class F2> \
40 inline auto name(const arg1, const arg2) -> decltype(expr) {
    return expr; }
41 #define FUNC3(name, arg1, arg2, arg3, expr) \
42 template <class F1, class F2, class F3> \
43 inline auto name(const arg1, const arg2, const arg3) ->
    decltype(expr) { return expr; }
44
45 FUNC1(operator -, Point<F>& point, makePoint(-point.x, -point
    .y))
46 FUNC2(operator +, Point<F1>& lhs, Point<F2>& rhs, makePoint(
    lhs.x + rhs.x, lhs.y + rhs.y))
47 FUNC2(operator -, Point<F1>& lhs, Point<F2>& rhs, makePoint(
    lhs.x - rhs.x, lhs.y - rhs.y))
48 FUNC2(operator *, F1& factor, Point<F2>& rhs, makePoint(
    factor * rhs.x, factor * rhs.y))
49 FUNC2(operator *, Point<F1>& lhs, F2& factor, makePoint(lhs.x
    * factor, lhs.y * factor))
50 FUNC2(operator /, Point<F1>& lhs, F2& factor, makePoint(lhs.x
    / factor, lhs.y / factor))
51
52 FUNC2(operator *, Point<F1>& lhs, Point<F2>& rhs, lhs.x * rhs
    .x + lhs.y * rhs.y)
53 FUNC2(operator ^, Point<F1>& lhs, Point<F2>& rhs, lhs.x * rhs
    .y - lhs.y * rhs.x)
54
55 // < 0 if rhs <- lhs counter-clockwise, 0 if collinear, > 0
    if clockwise.
56 FUNC2(ccw, Point<F1>& lhs, Point<F2>& rhs, rhs ^ lhs)
57 FUNC3(ccw, Point<F1>& lhs, Point<F2>& rhs, Point<F3>& origin,
    ccw(lhs - origin, rhs - origin))
58
59 FUNC2(operator ==, Point<F1>& lhs, Point<F2>& rhs, lhs.x ==
    rhs.x && lhs.y == rhs.y)
60 FUNC2(operator !=, Point<F1>& lhs, Point<F2>& rhs, !(lhs ==
    rhs))
61
62 FUNC2(operator <, Point<F1>& lhs, Point<F2>& rhs,
    lhs.y < rhs.y || (lhs.y == rhs.y && lhs.x < rhs.x))
63
64 FUNC2(operator >, Point<F1>& lhs, Point<F2>& rhs, rhs < lhs)
65 FUNC2(operator <=, Point<F1>& lhs, Point<F2>& rhs, !(lhs >
    rhs))
66 FUNC2(operator >=, Point<F1>& lhs, Point<F2>& rhs, !(lhs <
    rhs))
67
68 // Angles and rotations (counter-clockwise).
69 FUNC1(angle, Point<F>& point, atan2(point.y, point.x))
70 FUNC2(angle, Point<F1>& lhs, Point<F2>& rhs, atan2(lhs ^ rhs,
    lhs * rhs))
71 FUNC3(angle, Point<F1>& lhs, Point<F2>& rhs, Point<F3>&
    origin,
72     angle(lhs - origin, rhs - origin))
73 FUNC3(rotate, Point<F1>& point, F2& angleSin, F3& angleCos,

```

```

74     makePoint(angleCos * point.x - angleSin * point.y,
75         angleSin * point.x + angleCos * point.y))
76 FUNC2(rotate, Point<F1>& point, F2& angle, rotate(point, sin(
    angle), cos(angle)))
77 FUNC3(rotate, Point<F1>& point, F2& angle, Point<F3>& origin,
    origin + rotate(point - origin, angle))
78
79 FUNC1(perp, Point<F>& point, makePoint(-point.y, point.x))
80
81 // Distances.
82 FUNC1(abs, Point<F>& point, point * point)
83 FUNC1(norm, Point<F>& point, sqrt(abs(point)))
84 FUNC2(dist, Point<F1>& lhs, Point<F2>& rhs, norm(lhs - rhs))
85 FUNC2(dist2, Point<F1>& lhs, Point<F2>& rhs, abs(lhs - rhs))
86 FUNC2(bisector, Point<F1>& lhs, Point<F2>& rhs, lhs * norm(
    rhs) + rhs * norm(lhs))
87
88 #undef FUNC1
89 #undef FUNC2
90 #undef FUNC3

```

2.5 cp_geometry.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const double eps = 1e-6;
4 inline int fcmp(const double &a, const double &b){
5     if(fabs(a - b) < eps)
6         return 0;
7     return ((a - b > 0.0) * 2) - 1;
8 }
9
10 template <typename T>
11 struct Point
12 {
13     T x, y;
14     Point() : x(0), y(0) {}
15     Point(const T x, const T y) : x(x), y(y) {}
16     template <class F>
17     explicit operator Point<F>() const
18     {
19         return Point<F>((F)x, (F)y);
20     }
21
22     Point operator+(const Point b) const
23     {
24         return Point(x + b.x, y + b.y);
25     }
26     Point operator-(const Point b) const
27     {
28         return Point(x - b.x, y - b.y);
29     }
30     template <class F>
31     Point<F> operator*(const F fac)
32     {
33         return Point<F>(x * fac, y * fac);
34     }
35     template <class F>
36     Point<F> operator/(const F fac)
37     {
38         return Point<F>(x / fac, y / fac);
39     }
40

```

```

41 T operator&(const Point b) const { return x * b.x + y * b
    .y; }
42 // dot operator
43 T operator^(const Point b) const { return x * b.y - y * b
    .x; }
44 // cross operator
45 bool operator==(const Point b) const
46 {
47     return x == b.x and y == b.y;
48 }
49 bool operator<(const Point b) const
50 {
51     return x == b.x ? y < b.y : x < b.x;
52 } // 字典序
53
54 Point operator-() const { return Point(-x, -y); }
55 T norm() const { return *this * *this; } // 歐式長
    度平方
56 Point prep() const { return Point(-y, x); } // 左旋直
    角法向量
57 template <class F>
58 friend istream &operator>>(istream &is, Point<F> &pt);
59 template <class F>
60 friend ostream &operator<<(ostream &os, const Point<F> &
    pt);
61
62 template <class F>
63 ostream &operator<<(ostream &os, const Point<F> &pt)
64 {
65     return os << "(" << pt.x << " " << pt.y << ")";
66 }
67 template <class F>
68 istream &operator>>(istream &is, Point<F> &pt)
69 {
70     return is >> pt.x >> pt.y;
71 }
72 template <class F>
73 bool collinearity(const Point<F>& p1, const Point<F>& p2,
    const Point<F>& p3){
74     // return (p1 - p3) ^ (p2 - p3) == 0;
75     return fcmp((p1 - p3) ^ (p2 - p3), 0.0) == 0;
76 }
77 // check co-line first. properly
78 template <class F>
79 inline bool btw(const Point<F>& p1, const Point<F>& p2, const
    Point<F>& p3){
80     return fcmp((p1 - p3) & (p2 - p3), 0.0) <= 0;
81 }
82
83 // is p3 on (p1, p2)?
84 template <class F>
85 inline bool pointOnSegment(const Point<F>& p1, const Point<F>
    & p2, const Point<F>& p3){
86     return collinearity(p1, p2, p3) && btw(p1, p2, p3);
87 }
88
89 template <typename T, typename Real = double>
90 struct Line
91 {
92     Point<T> st, ed;
93     Point<T> vec() const { return ed - st; }
94     T ori(const Point<T> p) const { return (ed - st) ^ (p -
    st); }
95     int orint(const Point<T> p) const{
96         T a = this->ori(p);

```

```

97     return (fcmp(a, 0.0)); // 1 on posi-side // -1 nega-
98         side
99     // a little bit useless?
100 }
101 Line(const Point<T> x, const Point<T> y) : st(x), ed(y)
102 {}
103 template <class F>
104 operator Line<F>() const
105 {
106     return Line<F>((Point<F>)st, (Point<F>)ed);
107 }
108 // sort by arg, the left is smaller for parallel lines
109 bool operator<(Line B) const
110 {
111     Point<T> a = vec(), b = B.vec();
112     auto sgn = [](const Point<T> t) { return (t.y == 0 ?
113         t.x : t.y) < 0; };
114     if (sgn(a) != sgn(b))
115         return sgn(a) < sgn(b);
116     if (abs(a ^ b) == 0)
117         return B.ori(st) > 0;
118     return (a ^ b) > 0;
119 }
120 // Regard a line as a function
121 template <typename F>
122 Point<F> operator()(const F x) const // A + AB * x = the
123     point position.
124 {
125     return Point<F>(st) + vec() * x;
126 }
127 bool isSegProperIntersection(const Line l) const
128 {
129     return l.ori(st) * l.ori(ed) < 0 and ori(l.st) * ori(
130         l.ed) < 0;
131 }
132 bool isSegIntersection(const Line l) const {
133     Line<Real> h = *this;
134     // hst = 1, hed = 2, lst = 3, led = 4
135     double hlst = h.ori(l.st);
136     double hled = h.ori(l.ed);
137     double lhst = l.ori(h.st);
138     double lhed = l.ori(h.ed);
139     if (fcmp(hlst, 0.0) == 0 && fcmp(hled, 0.0) == 0)
140         return h.isPointOnSeg(l.st) || h.isPointOnSeg(l.
141             ed) || l.isPointOnSeg(h.st) || l.
142             isPointOnSeg(h.ed);
143     return fcmp(hlst * hled, 0.0) <= 0 && fcmp(lhst *
144         lhed, 0.0) <= 0;
145 }
146 bool isPointOnSegProperly(const Point<T> p) const
147 {
148     return fcmp(ori(p), 0.0) == 0 and fcmp(((st - p) & (
149         ed - p)), 0.0) < 0;
150 }
151 bool isPointOnSeg(const Point<T> p) const {
152     return fcmp(ori(p), 0.0) == 0 and fcmp((st - p) & (ed
153         - p), 0.0) <= 0;
154 }
155 Real disP2Line(const Point<T> p) const
156 {

```

```

157     return Line<double>(projection(p), Point<double>(p)).
158         vec().norm();
159 }
160 // notice if you should check Segment intersect or not;
161 // be careful divided by 0
162 Point<Real> getIntersection(Line<Real> l)
163 {
164     Line<Real> h = *this;
165     return l(((l.st - h.st) ^ h.vec()) / (h.vec() ^ l.
166         vec())); // use operator()
167     Real hlst = -h.ori(l.st);
168     Real hled = h.ori(l.ed);
169     return ((l.st * hled) + (l.ed * hlst)) / (hlst + hled
170         );
171 }
172 // 需要確認+-號的合理性
173 // Area of triangle(l.st, h.st, h.ed) divided by Area
174 // of Quadrilateral(h.st, l.st, h.ed, l.ed)
175 }
176 Point<Real> projection(const Point<T> p) const
177 {
178     return operator()(((p - st) & vec()) / (Real)(vec().
179         norm()));
180 }
181 template <class F>
182 friend ostream &operator<<(ostream &os, const Line<F> &l)
183 ;
184 };
185 template <class F>
186 ostream &operator<<(ostream &os, const Line<F> &l)
187 {
188     return os << "(" << l.st.x << ", " << l.st.y << ") to ("
189         << l.ed.x << ", " << l.ed.y << ")";
190 }
191 template <class F>
192 using Polygon = vector<Point<F>>;
193 template <class F>
194 Polygon<F> getConvexHull(Polygon<F> points) {
195     sort(points.begin(), points.end());
196     Polygon<F> CH;
197     CH.reserve(points.size() + 1); // for what ??
198     for (int round = 0; round < 2; round++){
199         int start = CH.size();
200         for (Point<int> &pt: points) {
201             while (CH.size() - start >= 2 && Line<F>(CH[CH.
202                 size() - 2], CH[CH.size() - 1]).ori(pt) <=
203                 0) // ? Line is different than senpai's .
204             {
205                 CH.pop_back();
206             }
207             CH.emplace_back(pt);
208         }
209         CH.pop_back();
210         reverse(points.begin(), points.end());
211     }
212     if (CH.size() == 2 && CH[0] == CH[1])
213         CH.pop_back();
214     return CH;
215 }

```

3 geometry/Convex_Hull

3.1 Andrew's_Monotone_Chain

```

1 // Andrew's Monotone Chain
2 template <class F>
3 using Polygon = vector<Point<F>>;
4
5 template <class F>
6 Polygon<F> getConvexHull(Polygon<F> points)
7 {
8     sort(begin(points), end(points));
9     Polygon<F> hull;
10    hull.reserve(points.size() + 1);
11    for (int phase = 0; phase < 2; ++phase)
12    {
13        auto start = hull.size();
14        for (auto &point : points)
15        {
16            while (hull.size() >= start + 2 &&
17                Line<F>(hull.back(), hull[hull.size() -
18                    2]).ori(point) <= 0)
19                hull.pop_back();
20            // whenever point is at the RIGHT(NEGATIVE) part
21            // of the line(hull[size - 1], hull[size-2])
22            // pop the last point because it causes concave
23            hull
24            hull.push_back(point);
25        }
26        hull.pop_back();
27        reverse(begin(points), end(points));
28    }
29    if (hull.size() == 2 and hull[0] == hull[1])
30        hull.pop_back();
31    return hull;
32 }

```

4 graph

4.1 Kosaraju_for_SCC

```

1 #include <vector>
2 #include <stack>
3 using namespace std;
4 #define MAX_N 200005
5 class Kosaraju_for_SCC{
6     int NodeNum;
7     vector<vector<int>>> G;
8     vector<vector<int>>> GT;
9     stack<int> st;
10    vector<bool> visited;
11    vector<int> scc;
12    int sccID;
13
14    public:
15    void init(int N){
16        NodeNum = N;

```

```

17 G.clear();
18 G.resize(N + 5);
19 GT.clear();
20 GT.resize(N + 5);
21 while(!st.empty())
22     st.pop();
23 visited.clear();
24 visited.resize(N + 5, false);
25 scc.clear();
26 scc.resize(N + 5);
27 sccID = 1;
28 }
29 void addEdge(int w, int v){
30     G[w].emplace_back(v);
31     GT[v].emplace_back(w);
32 }
33 void DFS(bool isG, int v, int k = -1){
34     visited[v] = true;
35     scc[v] = k;
36     vector<vector<int>> &dG = (isG ? G : GT);
37     for(int w: dG[v])
38     {
39         if(!visited[w]){
40             DFS(isG, w, k);
41         }
42     }
43     if(isG){
44         st.push(v);
45     }
46 }
47 void Kosaraju(int N){
48     visited.clear();
49     visited.resize(N + 5, false);
50     for (int i = 1; i <= N; i++){
51         if(!visited[i])
52             DFS(true, i);
53     }
54     visited.clear();
55     visited.resize(N + 5, false);
56     while(!st.empty()){
57         if(!visited[st.top()])
58             DFS(false, st.top(), sccID++);
59         st.pop();
60     }
61 }
62 vector<vector<int>> generateReG(){
63     vector<vector<int>> reG;
64     reG.resize(sccID);
65     for (int i = 1; i <= NodeNum; i++){
66         for(int w: G[i]){
67             if(scc[i] == scc[w])
68                 continue;
69             reG[scc[i]].emplace_back(scc[w]);
70         }
71     }
72     return reG;
73 }
74 };

```

4.2 Tarjan_for_BridgeCC

```

1 // BCC for bridge connected component
2 // by sylveon a.k.a LFSWang

```

```

3 #include <vector>
4 #include <stack>
5 #include <algorithm>
6 using namespace std;
7 #define MAX_N 200005
8 int timestamp = 1;
9 int bccid = 1;
10 int D[MAX_N];
11 int L[MAX_N];
12 int bcc[MAX_N];
13 stack<int> st;
14 vector<int> adj[MAX_N];
15 bool inSt[MAX_N];
16 void DFS(int v, int fa) { //call DFS(v,v) at first
17     D[v] = L[v] = timestamp++; //timestamp > 0
18     st.emplace(v);
19
20     for (int w:adj[v]) {
21         if( w==fa ) continue;
22         if ( !D[w] ) { // D[w] = 0 if not visited
23             DFS(w,v);
24             L[v] = min(L[v], L[w]);
25         }
26         L[v] = min(L[v], D[w]);
27     }
28     if (L[v]==D[v]) {
29         bccid++;
30         int x;
31         do {
32             x = st.top(); st.pop();
33             bcc[x] = bccid;
34         } while (x!=v);
35     }
36     return ;
37 }

```

4.3 Tarjan_for_AP_Bridge

```

1 #include <vector>
2 #include <utility>
3 using namespace std;
4 #define MAX_N 200005;
5 #define enp pair<int, int> // edge-weight, node-index
6 #define con pair<int, int> // connection
7
8 class tarjan{
9     vector<vector<int>> G; // adjacency List
10     vector<int> D; // visit or visited and D-value
11     vector<int> L; // for L-value
12     vector<con> edgeBridge;
13     vector<int> APnode;
14     int timestamp;
15     tarjan(int size = 1){
16         timestamp = 0;
17         G.resize(size);
18         D.resize(size, 0);
19         L.resize(size, 0);
20         edgeBridge.clear();
21         APnode.clear();
22     }
23     void init(int size = 1){
24         tarjan(size);
25     }

```

```

26 void addedge(int u, int v)
27 { // undirected graph
28     G[u].push_back(v);
29     G[v].push_back(u);
30 }
31 void DFS(int v, int pa){ // init: call DFS(v,v)
32     D[v] = L[v] = timestamp++;
33     int Childcount = 0;
34     bool isAP = false;
35     for(int w: G[v]){
36         if(w == pa)
37             continue;
38         if(!D[w]){ // 用 D[w] == 0 if not visited
39             DFS(w, v);
40             Childcount++;
41             if(D[v] <= L[w])
42                 isAP = true; // 結論 2 對於除了 root 點
43                             // 以外的所有點 v · v 點在 G 上為 AP 的
44                             // 充要條件為其在 T 中至少有一個子節點
45                             // w 滿足 D(v) ≤ L(w)
46             if(D[v] < L[w])
47                 edgeBridge.emplace_back(v,w); // 結論 3
48                             // 對於包含 r 在內的所有點 v 和 v 在 T
49                             // 中的子節點 w · 邊 e(v, w) 在圖 G 中為
50                             // bridge 的充要條件為 D(v) < L(w) ·
51             L[v] = min(L[v], L[w]);
52         }
53         L[v] = min(L[v], D[w]);
54     }
55     if(v == pa && Childcount < 2)
56         isAP = false;
57     if(isAP)
58         APnode.emplace_back(v);
59 }
60 };

```

4.4 Tarjan_for_SCC

```

1 // by atsushi
2 #include <vector>
3 #include <stack>
4 using namespace std;
5
6 class tarjan_for_SCC{
7 private:
8     vector<vector<int>> G; // adjacency list
9     vector<int> D;
10    vector<int> L;
11    vector<int> sccID;
12    stack<int> st; // for SccID
13    vector<bool> inSt;
14    vector<vector<int>> reG;
15    int timeStamp, sccIDstamp;
16 public:
17     void init(int size = 1){
18         G.clear();
19         G.resize(size + 3);
20         D.clear();
21         D.resize(size + 3, 0);
22         L.clear();
23         L.resize(size + 3, 0);

```

```

24 sccID.clear();
25 sccID.resize(size + 3, 0);
26 while(!st.empty())
27     st.pop();
28 inSt.clear();
29 inSt.resize(size + 3, false);
30 reG.clear();
31 sccIDstamp = timeStamp = 1;
32 }
33 void addEdge(int from, int to){
34     G[from].emplace_back(to);
35 }
36 void DFS(int v, int pa){ //call DFS(v,v) at first
37     D[v] = L[v] = timeStamp++; //timestamp > 0
38     st.push(v);
39     inSt[v] = true;
40
41     for(int w: G[v]){ // directed graph don't need w ==
42         pa
43         if(!D[w]){ // D[w] = 0 if not visited
44             DFS(w, v);
45             L[v] = min(L[v], L[w]);
46         } else if(inSt[w])
47             /* w has been visited.
48             if we don't add this, the L[v] will think
49             that v can back to node whose index less
50             to v.
51             inSt[w] is true that v -> w is a cross edge
52             opposite it's a forward edge
53             */
54             L[v] = min(L[v], D[w]); // why D[w] instead
55             of L[w]?
56         }
57     }
58     if(D[v] == L[v]){
59         int w;
60         do{
61             w = st.top();
62             st.pop();
63             sccID[w] = sccIDstamp; // scc ID for this
64             point at which SCC
65             inSt[w] = false;
66         } while (w != v);
67         sccIDstamp++;
68     }
69 }
70 void generateReG(int N = 1){
71     reG.clear();
72     reG.resize(sccIDstamp);
73     for(int i = 1; i <= N; i++){
74         for(int w: G[i]){
75             if(sccID[i] == sccID[w])
76                 continue;
77             reG[sccID[i]].emplace_back(sccID[w]);
78         }
79     }
80 }
81 bool visited(int v){
82     return D[v];
83 }
84 };

```

5 graph/Bipartite

5.1 konig_algorithm

```

1 #include <vector>
2 #include <cstring>
3 using namespace std;
4
5 // V times DFS O(EV)
6 vector<int> V[205];
7 // V[i]記錄了左半邊可以配到右邊的那些點
8 int match[205]; // A<=B
9 // match[i] 記錄了右半邊配對到左半邊的哪個點
10 bool used[205];
11 int n;
12 bool dfs(int v)
13 {
14     for(int e:V[v])
15     {
16         if( used[e] ) continue;
17         used[e] = true;
18         if( match[e] == -1 || dfs( match[e] ) )
19         {
20             match[e] = v;
21             return true;
22         }
23     }
24     return false;
25 }
26 int konig()
27 {
28     memset(match, -1, sizeof(match));
29
30     int ans=0;
31
32     for(int i=1; i<=n; ++i)
33     {
34         memset(used, 0, sizeof(used));
35         if( dfs(i) )
36             ans++;
37     }
38
39     return ans;
40 }

```

5.2 Kuhn-Munkres

```

1 // Max weight perfect bipartite matching
2 // O(V^3)
3 // by jinkela
4 #define MAXN 405
5 #define INF 0x3f3f3f3f3f3f3f3f
6 int n; // 1-base · 0表示沒有匹配
7 LL g[MAXN][MAXN]; //input graph
8 int My[MAXN], Mx[MAXN]; //output match
9 LL lx[MAXN], ly[MAXN], pa[MAXN], Sy[MAXN];
10 bool vx[MAXN], vy[MAXN];
11 void augment(int y){
12     for(int x, z; y; y = z){
13         x=pa[y], z=Mx[x];

```

```

14     My[y]=x, Mx[x]=y;
15 }
16 }
17 void bfs(int st){
18     for(int i=1; i<=n; ++i)
19         Sy[i] = INF, vx[i]=vy[i]=0;
20     queue<int> q; q.push(st);
21     for(;;){
22         while(q.size()){
23             int x=q.front(); q.pop();
24             vx[x]=1;
25             for(int y=1; y<=n; ++y) if(!vy[y]){
26                 LL t = lx[x]+ly[y]-g[x][y];
27                 if(t==0){
28                     pa[y]=x;
29                     if(!My[y]){augment(y);return;}
30                     vy[y]=1, q.push(My[y]);
31                 } else if(Sy[y]>t) pa[y]=x, Sy[y]=t;
32             }
33         }
34         LL cut = INF;
35         for(int y=1; y<=n; ++y)
36             if(!vy[y]&&cut>Sy[y]) cut=Sy[y];
37         for(int j=1; j<=n; ++j){
38             if(vx[j]) lx[j] -= cut;
39             if(vy[j]) ly[j] += cut;
40             else Sy[j] -= cut;
41         }
42         for(int y=1; y<=n; ++y){
43             if(!vy[y]&&Sy[y]==0){
44                 if(!My[y]){augment(y);return;}
45                 vy[y]=1, q.push(My[y]);
46             }
47         }
48     }
49 }
50 LL KM(){
51     memset(My, 0, sizeof(int)*(n+1));
52     memset(Mx, 0, sizeof(int)*(n+1));
53     memset(ly, 0, sizeof(LL)*(n+1));
54     for(int x=1; x<=n; ++x){
55         lx[x] = -INF;
56         for(int y=1; y<=n; ++y)
57             lx[x] = max(lx[x], g[x][y]);
58     }
59     for(int x=1; x<=n; ++x) bfs(x);
60     LL ans = 0;
61     for(int y=1; y<=n; ++y) ans+=g[My[y]][y];
62     return ans;
63 }

```

6 graph/Flow

6.1 Ford_Fulkerson

```

1 #include <vector>
2 #include <tuple>
3 #include <cstring>
4 using namespace std;
5
6 // O((V+E)F)

```



```

7 #define maxn 101
8 // remember to change used into the maxNode size -- kattis
  elementary math
9 bool used[MAXN];
10 int End;
11 vector<int> V[MAXN];
12 vector<tuple<int, int>> E;
13
14 // x==y 可以流 C
15 // if undirected or 2-direc edge, bakcward Capacity become C;
16 // Graph build by edge array
17 // 反向邊的編號只要把自己的編號 xor 1 就能取得
18 void add_edge(int x, int y, int c)
19 {
20     V[x].emplace_back( E.size() );
21     E.emplace_back(y, c);
22     V[y].emplace_back( E.size() );
23     E.emplace_back(x, 0);
24 }
25 int dfs(int v, int f)
26 {
27     if( v==End ) return f;
28     used[v] = true;
29     int e, w;
30     for( int eid : V[v] )
31     {
32         tie(e, w) = E[eid];
33         if( used[e] || w==0 ) continue;
34
35         w = dfs(e, min(w, f));
36         if( w>0 )
37         {
38             // 更新流量
39             get<1>(E[eid]) -= w;
40             get<1>(E[eid^1]) += w;
41             return w;
42         }
43     }
44     return 0; // Fail!
45 }
46 int ffa(int s, int e)
47 {
48     int ans = 0, f;
49     End = e;
50     while(true)
51     {
52         memset(used, 0, sizeof(used));
53         f = dfs(s, INT_MAX);
54         if( f<=0 ) break;
55         ans += f;
56     }
57     return ans;
58 }

```

6.2 Edmonds-Karp-adjmax

```

1 // O((V+E)VE) · 簡單寫成 O(VE^2)
2 #include <cstring>
3 #include <queue>
4 using namespace std;
5 #define maxn 100
6 typedef int Graph[ MAXN ][ MAXN ]; // adjacency matrix

```

```

7 Graph C, F, R; // 容量上限、流量、剩餘容量
8 bool visit[ MAXN ]; // BFS經過的點
9 int path[ MAXN ]; // BFS tree
10 int flow[ MAXN ]; // 源點到各點的流量瓶頸
11
12 int BFS(int s, int t) // 源點與匯點
13 {
14     memset(visit, false, sizeof(visit));
15
16     queue<int> Q; // BFS queue
17     visit[s] = true;
18     path[s] = s;
19     flow[s] = 1e9;
20     Q.push(s);
21
22     while (!Q.empty())
23     {
24         int i = Q.front(); Q.pop();
25         for (int j=0; j<100; ++j)
26             // 剩餘網路找擴充路徑
27             if (!visit[j] && R[i][j] > 0)
28             {
29                 visit[j] = true;
30                 path[j] = i;
31                 // 一邊找最短路徑 · 一邊計算流量瓶頸。
32                 flow[j] = min(flow[i], R[i][j]);
33                 Q.push(j);
34
35                 if (j == t) return flow[t];
36             }
37     }
38     return 0; // 找不到擴充路徑了 · 流量為零。
39 }
40
41 int Edmonds_Karp(int s, int t)
42 {
43     memset(F, 0, sizeof(F));
44     memcpy(R, C, sizeof(C));
45
46     int f, df; // 最大流的流量、擴充路徑的流量
47     for (f=0; df=BFS(s, t); f+=df)
48         // 更新擴充路徑上每一條邊的流量
49         for (int i=path[t], j=t; i!=s; i=path[j-1], j=j-1)
50         {
51             F[i][j] = F[i][j] + df;
52             F[j][i] = -F[i][j];
53             R[i][j] = C[i][j] - F[i][j];
54             R[j][i] = C[j][i] - F[j][i];
55         }
56     return f;
57 }

```

6.3 Dinic_algorithm

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 // O(V^2E) O(VE) finding argument path
4 // if unit capacity network then O(min(V^3/2, E^1/2) E)
5 // solving bipartite matching O(E V^1/2) better than konig
  and flow(EV)
6 #define MAXN 101

```

```

7 #define INT_MAX 10000000
8 int End, dist[ MAXN ];
9 vector<tuple<int, int, int>> V[ MAXN ];
10 // vertex-index, cap, the index of the reverse edge
11 void addEdge(int u, int v, int c){
12     V[u].emplace_back(v, c, V[v].size());
13     V[v].emplace_back(u, 0, V[u].size() - 1);
14 }
15 bool bfs(int s) {
16     memset(dist, -1, sizeof(dist));
17     queue<int> qu;
18     qu.emplace(s);
19     dist[s]=0;
20
21     while( !qu.empty() ) {
22         int S = qu.front(); qu.pop();
23         for(auto &p : V[S]) {
24             int E, C;
25             tie(E, C, ignore) = p;
26             if( dist[E]==-1 && C!=0 ) {
27                 dist[E]=dist[S]+1;
28                 qu.emplace(E);
29             }
30         }
31     }
32     return dist[End] != -1;
33 }
34 int dfs(int v, int f) {
35     int e, w, rev;
36     if( v==End || f==0 ) return f;
37     for( auto &t : V[v] )
38     {
39         tie(e, w, rev) = t;
40         if( dist[e]!=dist[v]+1 || w==0 )
41             continue;
42
43         w = dfs(e, min(w, f));
44         if( w>0 )
45         {
46             get<1>(t) -= w;
47             get<1>(V[e][rev]) += w;
48             return w;
49         }
50     }
51     dist[v] = -1; //優化 · 這個點沒用了
52     return 0; // Fail!
53 }
54 int dinic(int s, int e)
55 {
56     int ans = 0, f;
57     End = e;
58     while(bfs(s))
59     {
60         while( f = dfs(s, INT_MAX) )
61             ans += f;
62     }
63     return ans;
64 }

```

6.4 Edmonds_Karp_2

```

1 #include <bits/stdc++.h>
2 struct Edge{

```

```

3  int from, to, cap, flow;
4  Edge(int u, int v, int c, int f):from(u), to(v), cap(c),
   flow(f){}
5  };
6  const maxn = 200005;
7  struct EdmondsKarp{
8      int n, m;
9      vector<Edge> edges;
10     vector<int> G[maxn];
11     int a[maxn];
12     int p[maxn];
13     void init(int n){
14         for (int i = 0; i < n; i++)
15             G[i].clear();
16         edges.clear();
17     }
18     void AddEdge(int from, int to, int cap){
19         edges.push_back(Edge(from, to, cap, 0));
20         edges.push_back(Edge(to, from, 0, 0)) // 反向弧
21         m = edges.size();
22         G[from].push_back(m - 2);
23         G[to].push_back(m - 1);
24     }
25     int Maxflow(int s, int t){
26         int flow = 0;
27         for (;;){
28             memset(a, 0, sizeof(a));
29             queue<int> Q;
30             Q.push(s);
31             a[s] = INF;
32             while(!Q.empty()){
33                 int x = Q.front();
34                 Q.pop();
35                 for (int i = 0; i < G[x].size(); i++){
36                     Edge &e = edges[G[x][i]];
37                     if(!a[e.to] && e.cap > e.flow){
38                         p[e.to] = G[x][i];
39                         a[e.to] = min(a[x], e.cap - e.flow);
40                         Q.push(e.to);
41                     }
42                 }
43                 if(a[t])
44                     break;
45             }
46             if(!a[t])
47                 break;
48             for (int u = t; u != s; u = edges[p[u]].from){
49                 edges[p[u]].flow += a[t];
50                 edges[p[u] ^ 1].flow -= a[t];
51             }
52             flow += a[t];
53         }
54         return flow;
55     }
56 }

```

6.5 MinCostMaxFlow

```

1 // by jinkela
2 template<typename TP>
3 struct MCMF{
4     static const int MAXN=440;
5     static const TP INF=999999999;

```

```

6     struct edge{
7         int v,pre;
8         TP r,cost;
9         edge(int v,int pre,TP r,TP cost):v(v),pre(pre),r(r),cost(
10            cost){}
11     };
12     int n,S,T;
13     TP dis[MAXN],PIS,ans;
14     bool vis[MAXN];
15     vector<edge> e;
16     int g[MAXN];
17     void init(int _n){
18         memset(g,-1,sizeof(int)*((n=_n)+1));
19         e.clear();
20     }
21     void add_edge(int u,int v,TP r,TP cost,bool directed=false)
22     {
23         e.push_back(edge(v,g[u],r,cost));
24         g[u]=e.size()-1;
25         e.push_back(
26             edge(u,g[v],directed?0:r,-cost));
27         g[v]=e.size()-1;
28     }
29     TP augment(int u,TP CF){
30         if(u==T||!CF)return ans+=PIS*CF,CF;
31         vis[u]=1;
32         TP r=CF,d;
33         for(int i=g[u];~i;i=e[i].pre){
34             if(e[i].r&&e[i].cost&&!vis[e[i].v]){
35                 d=augment(e[i].v,min(r,e[i].r));
36                 e[i].r-=d;
37                 e[i^1].r+=d;
38                 if(!r==d)break;
39             }
40         }
41         return CF-r;
42     }
43     bool modlabel(){
44         for(int u=0;u<=n;++u)dis[u]=INF;
45         static deque<int>q;
46         dis[T]=0,q.push_back(T);
47         while(q.size()){
48             int u=q.front();q.pop_front();
49             TP dt;
50             for(int i=g[u];~i;i=e[i].pre){
51                 if(e[i^1].r&&(dt=dis[u]-e[i].cost)<dis[e[i].v]){
52                     if((dis[e[i].v]=dt)<=dis[q.size()?q.front():S]){
53                         q.push_front(e[i].v);
54                     }else q.push_back(e[i].v);
55                 }
56             }
57         }
58         for(int u=0;u<=n;++u)
59             for(int i=g[u];~i;i=e[i].pre)
60                 e[i].cost+=dis[e[i].v]-dis[u];
61         return PIS+=dis[S], dis[S]<INF;
62     }
63     TP mincost(int s,int t){
64         S=s,T=t;
65         PIS=ans=0;
66         while(modlabel()){
67             do memset(vis,0,sizeof(bool)*(n+1));
68             while(augment(S,INF));
69         }return ans;
70     }
71 }

```

7 graph/Matching

7.1 blossom_matching

```

1 // by jinkela
2 // 最大圖匹配
3 // O(V^2(V+E))
4 #define MAXN 505
5 int n; //1-base
6 vector<int> g[MAXN];
7 int MH[MAXN]; //output MH
8 int pa[MAXN],st[MAXN],S[MAXN],v[MAXN],t;
9 int lca(int x,int y){
10     for(++t;swap(x,y)){
11         if(!x) continue;
12         if(v[x]==t) return x;
13         v[x] = t;
14         x = st[pa[MH[x]]];
15     }
16 }
17 #define qpush(x) q.push(x),S[x]=0
18 void flower(int x,int y,int l,queue<int>&q){
19     while(st[x]!=1){
20         pa[x]=y;
21         if(S[y==MH[x]]==1)qpush(y);
22         st[x]=st[y]=1, x=pa[y];
23     }
24 }
25 bool bfs(int x){
26     iota(st+1, st+n+1, 1);
27     memset(S+1,-1,sizeof(int)*n);
28     queue<int>q; qpush(x);
29     while(q.size()){
30         x=q.front(),q.pop();
31         for(int y:g[x]){
32             if(S[y]==-1){
33                 pa[y]=x,S[y]=1;
34                 if(!MH[y]){
35                     for(int lst;x=y,lst,x=pa[y])
36                         lst=MH[x],MH[x]=y,MH[y]=x;
37                     return 1;
38                 }
39                 qpush(MH[y]);
40             }else if(!S[y]&&st[y]!=st[x]){
41                 int l=lca(y,x);
42                 flower(y,x,l,q),flower(x,y,l,q);
43             }
44         }
45     }
46     return 0;
47 }
48 int blossom(){
49     memset(MH+1,0,sizeof(int)*n);
50     int ans=0;
51     for(int i=1; i<=n; ++i)
52         if(!MH[i]&&bfs(i)) ++ans;
53     return ans;
54 }

```


8 graph/Minimum_Spanning_Tree

8.1 prim

```

1 #include <vector>
2 #include <queue>
3 #include <utility>
4 using namespace std;
5 #define enp pair<int, int> // pair<edge_val, node>
6 int prim_pq(vector<vector<enp>> E){
7     vector<bool> vis;
8     vis.resize(E.size(), false);
9     vis[0] = true;
10    priority_queue<enp> pq;
11    for(auto e: E[0]){
12        pq.emplace(-e.first, e.second);
13    }
14    int ans = 0; // min value for MST
15    while(pq.size()){
16        int w, v; // edge-weight, vertex index
17        tie(w, v) = pq.top();
18        pq.pop();
19        if(vis[v])
20            continue;
21        w = -w;
22        vis[v] = true;
23        ans += w;
24        for(auto e: E[v]){
25            pq.emplace(-e.first, e.second);
26        }
27    }
28    return ans;
29 }

```

8.2 Kruskal

```

1 #include <tuple>
2 #include <vector>
3 #include <algorithm>
4 #include <numeric> // for iota(first, last, val) setting
5 // iterator value
6 using namespace std;
7
8 struct DSU // disjoint set no rank-comp-merge
9 {
10     vector<int> fa;
11     DSU(int n) : fa(n) { iota(fa.begin(), fa.end(), 0); } //
12     // auto fill fa from 0 to n-1
13     int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }
14     void merge(int x, int y) { fa[find(x)] = find(y); }
15 };
16
17 int kruskal(int V, vector<tuple<int, int, int>> E) // save
18 // all edges into E, instead of saving graph via adjacency
19 // list
20 {
21     sort(E.begin(), E.end());
22     DSU dsu(V);
23     int mcnt = 0;
24     int ans = 0;

```

```

20 for (auto e : E)
21 {
22     int w, u, v; // w for start, u for des, v for val
23     tie(w, u, v) = e;
24     if (dsu.find(u) == dsu.find(v))
25         continue;
26     dsu.merge(u, v);
27     ans += w;
28     if (++mcnt == V - 1)
29         break;
30 }
31 return ans;
32 }

```

9 graph/Shortest_Path

9.1 dijkstra

```

1 #include <iostream>
2 #include <vector>
3 #include <queue>
4 #include <utility>
5 using namespace std;
6
7 #define con pair<int, int> // first for distance, second for
8 // index
9 vector<vector<con>> Graph; //
10 vector<int> dis; // distance;
11 int main(){
12     priority_queue<con, vector<con>, greater<con>> pq;
13     dis[0] = 0;
14     pq.emplace(con(0, 0));
15     while(pq.size()){
16         con cur = pq.top();
17         pq.pop();
18         if(cur.first != dis[cur.second])
19             continue;
20         for(auto it: Graph[cur.second]){
21             if(cur.first + it.first < dis[it.second]){
22                 dis[it.second] = cur.first + it.first;
23                 pq.emplace(dis[it.second], it.second);
24             }
25         }
26     }
27     return 0;

```

9.2 dijkstra-alrightchiu-version

```

1 // C++ code
2 #include <iostream>
3 #include <vector>
4 #include <list>
5 #include <utility> // for std::pair<>
6 #include <iomanip> // for std::setw()
7 #include <cmath> // for std::floor
8 // #include "Priority_Queue_BinaryHeap.h"

```

```

10 const int Max_Distance = 100;
11 class Graph_SP{ // SP serves as Shortest Path
12 private:
13     int num_vertex;
14     std::vector<std::list<std::pair<int, int>>> AdjList;
15     std::vector<int> predecessor, distance;
16     std::vector<bool> visited;
17 public:
18     Graph_SP():num_vertex(0){};
19     Graph_SP(int n):num_vertex(n){
20         AdjList.resize(num_vertex);
21     }
22     void AddEdge(int from, int to, int weight);
23     void PrintDataArray(std::vector<int> array);
24     void PrintIntArray(int *array);
25
26     void InitializeSingleSource(int Start); // 以Start作
27 // 為起點
28     void Relax(int X, int Y, int weight); // edge方向:
29 // from X to Y
30
31     void Dijkstra(int Start = 0); // 需要Min-Priority
32 // Queue
33     friend class BinaryHeap; // 以Binary Heap實現
34 // Min-Priority Queue
35 };
36 void Graph_SP::Dijkstra(int Start){
37
38     InitializeSingleSource(Start);
39
40     BinaryHeap minQueue(num_vertex); // object of min queue
41     minQueue.BuildMinHeap(distance);
42
43     visited.resize(num_vertex, false); // initializa
44 // visited[] as {0,0,0,...,0}
45
46     while (!minQueue.IsHeapEmpty()) {
47         int u = minQueue.ExtractMin();
48         for (std::list<std::pair<int, int>>::iterator itr =
49             AdjList[u].begin();
50             itr != AdjList[u].end(); itr++) {
51             Relax(u, (*itr).first, (*itr).second);
52             minQueue.DecreaseKey((*itr).first, distance[(*itr)
53                 ].first]); // definition at alrightchiu's
54 // priority queue self-version
55         }
56     }
57     std::cout << "\nprint predecessor:\n";
58     PrintDataArray(predecessor);
59     std::cout << "\nprint distance:\n";
60     PrintDataArray(distance);
61 }
62 void Graph_SP::InitializeSingleSource(int Start){
63
64     distance.resize(num_vertex);
65     predecessor.resize(num_vertex);
66
67     for (int i = 0; i < num_vertex; i++) {
68         distance[i] = Max_Distance;
69         predecessor[i] = -1;
70     }
71     distance[Start] = 0;
72 }
73 void Graph_SP::Relax(int from, int to, int weight){

```

```

67     if (distance[to] > distance[from] + weight) {
68         distance[to] = distance[from] + weight;
69         predecessor[to] = from;
70     }
71 }
72 void Graph_SP::AddEdge(int from, int to, int weight){
73     AdjList[from].push_back(std::make_pair(to,weight));
74 }
75 void Graph_SP::PrintDataArray(std::vector<int> array){
76     for (int i = 0; i < num_vertex; i++)
77         std::cout << setw(4) << i;
78     std::cout << std::endl;
79     for (int i = 0; i < num_vertex; i++)
80         std::cout << setw(4) << array[i];
81     std::cout << std::endl;
82 }
83 int main(){
84     Graph_SP g9(6);
85     g9.AddEdge(0, 1, 8);g9.AddEdge(0, 5, 1);
86     g9.AddEdge(1, 0, 3);g9.AddEdge(1, 2, 1);
87     g9.AddEdge(2, 0, 5);g9.AddEdge(2, 3, 2);g9.AddEdge(2, 4,
88         2);
89     g9.AddEdge(3, 1, 4);g9.AddEdge(3, 2, 6);g9.AddEdge(3, 4,
90         7);g9.AddEdge(3, 5, 3);
91     g9.AddEdge(5, 3, 2);g9.AddEdge(5, 4, 8);
92     g9.Dijkstra(0);
93     return 0;
94 }

```

9.3 bellman-Ford

```

1 // C++ code
2 #include <iostream>
3 #include <vector>
4 #include <list>
5 #include <utility> // for std::pair<>
6 #include <iomanip> // for std::setw()
7
8 const int Max_Distance = 100;
9 class Graph_SP{ // SP serves as Shortest Path
10 private:
11     int num_vertex;
12     std::vector<std::list<std::pair<int,int>>> AdjList;
13     std::vector<int> predecessor, distance;
14 public:
15     Graph_SP():num_vertex(0){};
16     Graph_SP(int n):num_vertex(n){
17         AdjList.resize(num_vertex);
18     }
19     void AddEdge(int from, int to, int weight);
20     void PrintDataArray(std::vector<int> array);
21     void InitializeSingleSource(int Start); // 以Start作
22     // 為起點
23     void Relax(int X, int Y, int weight); // 對edge(X,Y
24     // 進行Relax
25     bool BellmanFord(int Start = 0); // 以Start作
26     // 為起點

```

```

25 };
26
27 bool Graph_SP::BellmanFord(int Start){
28     InitializeSingleSource(Start);
29     for (int i = 0; i < num_vertex-1; i++) {
30         // |V-1|次的iteration
31         // for each edge belonging to E(G)
32         for (int j = 0; j < num_vertex; j++) {
33             // 把AdjList最外層的vector走一遍
34             for (std::list<std::pair<int,int> >::iterator itr
35                 = AdjList[j].begin();
36                 itr != AdjList[j].end(); itr++) {
37                 // 各個vector中, 所有edge走一遍
38                 Relax(j, (*itr).first, (*itr).second);
39             }
40         }
41     }
42     // check if there is negative cycle
43     for (int i = 0; i < num_vertex; i++) {
44         for (std::list<std::pair<int,int> >::iterator itr =
45             AdjList[i].begin();
46             itr != AdjList[i].end(); itr++) {
47             if (distance[(*itr).first] > distance[i]+(*itr).
48                 second) { // i是from, *itr是to
49                 return false;
50             }
51         }
52     }
53     // print predecessor[] & distance[]
54     std::cout << "predecessor[]:\n";
55     PrintDataArray(predecessor);
56     std::cout << "distance[]:\n";
57     PrintDataArray(distance);
58     return true;
59 }
60 void Graph_SP::PrintDataArray(std::vector<int> array){
61     for (int i = 0; i < num_vertex; i++)
62         std::cout << setw(4) << i;
63     std::cout << std::endl;
64     for (int i = 0; i < num_vertex; i++)
65         std::cout << setw(4) << array[i];
66     std::cout << std::endl << std::endl;
67 }
68 void Graph_SP::InitializeSingleSource(int Start){
69     distance.resize(num_vertex);
70     predecessor.resize(num_vertex);
71     for (int i = 0; i < num_vertex; i++) {
72         distance[i] = Max_Distance;
73         predecessor[i] = -1;
74     }
75     distance[Start] = 0;
76 }

```

```

// if there
is
negative
cycle,
return
false

```

```

78 void Graph_SP::Relax(int from, int to, int weight){
79     if (distance[to] > distance[from] + weight) {
80         distance[to] = distance[from] + weight;
81         predecessor[to] = from;
82     }
83 }
84 void Graph_SP::AddEdge(int from, int to, int weight){
85     AdjList[from].push_back(std::make_pair(to,weight));
86 }
87 int main(){
88     Graph_SP g7(6);
89     g7.AddEdge(0, 1, 5);
90     g7.AddEdge(1, 4, -4);g7.AddEdge(1, 2, 6);
91     g7.AddEdge(2, 4, -3);g7.AddEdge(2, 5, -2);
92     g7.AddEdge(3, 2, 4);
93     g7.AddEdge(4, 3, 1);g7.AddEdge(4, 5, 6);
94     g7.AddEdge(5, 0, 3);g7.AddEdge(5, 1, 7);
95     if (g7.BellmanFord(0))
96         std::cout << "There is no negative cycle.\n";
97     else
98         std::cout << "There is negative cycle.\n";
99     return 0;
100 }

```

9.4 Floyd-Warshall

```

1 // C++ code
2 // by alrightchiu
3 // all pairs shortest path
4 #include <iostream>
5 #include <vector>
6 #include <iomanip> // for setw()
7
8 const int MaxDistance = 1000;
9 class Graph_SP_AllPairs{
10 private:
11     int num_vertex;
12     std::vector< std::vector<int> > AdjMatrix, Distance,
13     Predecessor;
14 public:
15     Graph_SP_AllPairs():num_vertex(0){};
16     Graph_SP_AllPairs(int n);
17     void AddEdge(int from, int to, int weight);
18     void PrintData(std::vector< std::vector<int> > array);
19     void InitializeData();
20     void FloydWarshall();
21 };
22 Graph_SP_AllPairs::Graph_SP_AllPairs(int n):num_vertex(n){
23     // Constructor, initialize AdjMatrix with 0 or
24     // MaxDistance
25     AdjMatrix.resize(num_vertex);
26     for (int i = 0; i < num_vertex; i++) {
27         AdjMatrix[i].resize(num_vertex, MaxDistance);
28         for (int j = 0; j < num_vertex; j++) {
29             if (i == j){
30                 AdjMatrix[i][j] = 0;

```

```

31     }
32 }
33 }
34 void Graph_SP_AllPairs::InitializeData(){
35     Distance.resize(num_vertex);
36     Predecessor.resize(num_vertex);
37
38     for (int i = 0; i < num_vertex; i++) {
39         Distance[i].resize(num_vertex);
40         Predecessor[i].resize(num_vertex, -1);
41         for (int j = 0; j < num_vertex; j++) {
42             Distance[i][j] = AdjMatrix[i][j];
43             if (Distance[i][j] != 0 && Distance[i][j] !=
44                 MaxDistance) {
45                 Predecessor[i][j] = i;
46             }
47         }
48     }
49 void Graph_SP_AllPairs::FloydWarshall(){
50
51     InitializeData();
52
53     std::cout << "initial Distance[]:\n";
54     PrintData(Distance);
55     std::cout << "\ninitial Predecessor[]:\n";
56     PrintData(Predecessor);
57
58     for (int k = 0; k < num_vertex; k++) {
59         std::cout << "\nincluding vertex(" << k << "):\n";
60         for (int i = 0; i < num_vertex; i++) {
61             for (int j = 0; j < num_vertex; j++) {
62                 if ((Distance[i][j] > Distance[i][k]+Distance
63                     [k][j])
64                     && (Distance[i][k] != MaxDistance)) {
65                     Distance[i][j] = Distance[i][k]+Distance
66                         [k][j];
67                     Predecessor[i][j] = Predecessor[k][j];
68                 }
69             }
70             // print data after including new vertex and updating
71             // the shortest paths
72             std::cout << "Distance[]:\n";
73             PrintData(Distance);
74             std::cout << "\nPredecessor[]:\n";
75             PrintData(Predecessor);
76         }
77     }
78 void Graph_SP_AllPairs::PrintData(std::vector< std::vector<
79     int> > array){
80
81     for (int i = 0; i < num_vertex; i++){
82         for (int j = 0; j < num_vertex; j++) {
83             std::cout << std::setw(5) << array[i][j];
84         }
85         std::cout << std::endl;
86     }
87 }
88 void Graph_SP_AllPairs::AddEdge(int from, int to, int weight){
89     AdjMatrix[from][to] = weight;
90 }
91 int main(){

```

```

91     Graph_SP_AllPairs g10(4);
92     g10.AddEdge(0, 1, 2);g10.AddEdge(0, 2, 6);g10.AddEdge(0,
93         3, 8);
94     g10.AddEdge(1, 2, -2);g10.AddEdge(1, 3, 3);
95     g10.AddEdge(2, 0, 4);g10.AddEdge(2, 3, 1);
96
97     g10.FloydWarshall();
98     return 0;
99 }

```

9.5 shortest-path_on_DAG

```

1 // C++ code
2 // O(V+E)
3 #include <iostream>
4 #include <vector>
5 #include <list>
6 #include <utility> // for std::pair<>
7 #include <iomanip> // for std::setw()
8
9 const int Max_Distance = 100;
10 class Graph_SP{ // SP serves as Shortest Path
11 private:
12     int num_vertex;
13     std::vector<std::list<std::pair<int,int>>> AdjList;
14     std::vector<int> predecessor, distance;
15 public:
16     Graph_SP():num_vertex(0){};
17     Graph_SP(int n):num_vertex(n){
18         AdjList.resize(num_vertex);
19     }
20     void AddEdge(int from, int to, int weight);
21     void PrintDataArray(std::vector<int> array);
22     void PrintIntArray(int *array);
23
24     void InitializeSingleSource(int Start); // 以Start作
25     // 為起點
26     void Relax(int X, int Y, int weight); // 對edge(X,Y
27     // 進行Relax
28
29     void DAG_SP(int Start = 0); // 需要
30     // DFS, 加一個額外的Linked list
31     void GetTopologicalSort(int *array, int Start);
32     void DFSVisit_TS(int *array, int *color, int *discover,
33         int *finish, int vertex, int &time, int
34         &count);
35 };
36 void Graph_SP::GetTopologicalSort(int *array, int Start){
37
38     int color[num_vertex], discover[num_vertex], finish[
39         num_vertex];
40
41     for (int i = 0; i < num_vertex; i++) {
42         color[i] = 0;
43         discover[i] = 0;
44         finish[i] = 0;
45         predecessor[i] = -1;
46     }
47
48     int time = 0,

```

```

45     count = num_vertex-1, // count 為
46         // topologicalsort[] 的 index
47     i = Start;
48
49     for (int j = 0; j < num_vertex; j++) {
50         if (color[i] == 0) {
51             DFSVisit_TS(array, color, discover, finish, i,
52                 time, count);
53         }
54         i = j;
55     }
56     std::cout << "\nprint discover time:\n";
57     PrintIntArray(discover);
58     std::cout << "\nprint finish time:\n";
59     PrintIntArray(finish);
60 }
61 void Graph_SP::DFSVisit_TS(int *array, int *color, int *
62     discover,
63         int *finish, int vertex, int &time
64         , int &count){
65
66     color[vertex] = 1; // set gray
67     discover[vertex] = ++time;
68     for (std::list<std::pair<int,int>>::iterator itr =
69         AdjList[vertex].begin();
70         itr != AdjList[vertex].end(); itr++) {
71         if (color[(*itr).first] == 0) {
72             predecessor[(*itr).first] = vertex;
73             DFSVisit_TS(array, color, discover, finish, (*itr
74                 ).first, time, count);
75         }
76     }
77     color[vertex] = 2; // set black
78     finish[vertex] = ++time;
79     array[count++] = vertex; // 產生Topological
80     Sort
81 }
82 void Graph_SP::DAG_SP(int Start){
83
84     InitializeSingleSource(Start); // distance[],
85     // predecessor[]的initialization
86
87     int topologicalsort[num_vertex];
88     GetTopologicalSort(topologicalsort, Start);
89
90     for (int i = 0; i < num_vertex; i++) {
91         int v = topologicalsort[i];
92         for (std::list<std::pair<int, int>>::iterator itr =
93             AdjList[v].begin();
94             itr != AdjList[v].end(); itr++) {
95             Relax(v, (*itr).first, (*itr).second);
96         }
97     }
98     std::cout << "\nprint predecessor:\n";
99     PrintDataArray(predecessor);
100     std::cout << "\nprint distance:\n";
101     PrintDataArray(distance);
102 }
103 void Graph_SP::PrintDataArray(std::vector<int> array){
104     for (int i = 0; i < num_vertex; i++)
105         std::cout << std::setw(4) << i;
106     std::cout << std::endl;
107     for (int i = 0; i < num_vertex; i++)
108         std::cout << std::setw(4) << array[i];
109     std::cout << std::endl;

```

```

101 }
102 void Graph_SP::PrintIntArray(int *array){
103     for (int i = 0; i < num_vertex; i++)
104         std::cout << std::setw(4) << i;
105     std::cout << std::endl;
106     for (int i = 0; i < num_vertex; i++)
107         std::cout << std::setw(4) << array[i];
108     std::cout << std::endl;
109 }
110 void Graph_SP::InitializeSingleSource(int Start){
111     distance.resize(num_vertex);
112     predecessor.resize(num_vertex);
113     for (int i = 0; i < num_vertex; i++) {
114         distance[i] = Max_Distance;
115         predecessor[i] = -1;
116     }
117     distance[Start] = 0;
118 }
119 void Graph_SP::Relax(int from, int to, int weight){
120     if (distance[to] > distance[from] + weight) {
121         distance[to] = distance[from] + weight;
122         predecessor[to] = from;
123     }
124 }
125 void Graph_SP::AddEdge(int from, int to, int weight){
126     AdjList[from].push_back(std::make_pair(to, weight));
127 }
128 int main(){
129     Graph_SP g8(7);
130     g8.AddEdge(0, 1, 3); g8.AddEdge(0, 2, -2);
131     g8.AddEdge(1, 3, -4); g8.AddEdge(1, 4, 4);
132     g8.AddEdge(2, 4, 5); g8.AddEdge(2, 5, 6);
133     g8.AddEdge(3, 5, 8); g8.AddEdge(3, 6, 2);
134     g8.AddEdge(4, 3, -3); g8.AddEdge(4, 6, -2);
135     g8.AddEdge(5, 6, 2);
136     g8.DAG_SP(0); // 以vertex(0)作為起點
137     return 0;
138 }

```

10 graph/Tree

10.1 Lowest_Common_Ancestor

```

1 #define MAXN 200005
2 #define MAXLOG 200
3 int D[MAXN];
4 int P[MAXLOG][MAXLOG];
5 #include <cmath>
6 #include <algorithm>
7 using namespace std;
8 #define MAXN 200005
9 #define MAXLOG 200
10 int N = MAXN;
11 int lgN = log(N) / log(2);

```

```

12 int D[MAXN];
13 int P[MAXLOG][MAXLOG];
14 int LCA(int u, int v)
15 {
16     if (D[u] > D[v])
17         swap(u, v);
18     int s = D[v] - D[u]; // adjust D until D[v] = D[u]
19     for (int i = 0; i <= lgN; ++i) // 調整他們到二進位數一樣
20         if (s & (1 << i))
21             v = P[v][i];
22     if (u == v)
23         return v;
24     // because they are at same depth
25     // jump up if they are different
26     // think about that if P[u][i] == P[v][i]
27     // then that point must be the ancestor of LCA or LCA
28     itself
29     // by this, we will stop at LCA's child
30     for (int i = lgN; i >= 0; --i)
31         //
32         if (P[u][i] != P[v][i])
33             {
34                 u = P[u][i];
35                 v = P[v][i];
36             }
37     return P[u][0];
38 }
39 void ComputeP()
40 {
41     int n = N;
42     for (int i = 0; i < lgN; ++i) // to lgN enough
43     {
44         for (int x = 0; x < n; ++x)
45         {
46             if (P[x][i] == -1)
47                 P[x][i + 1] = -1;
48             else
49                 P[x][i + 1] = P[P[x][i]][i]; // equal to move
50                 on the parent direction
51                 // And P[x][i] move 2 ^ n steps to a parent
52                 // we call it y
53                 // P[y][i] means continue move 2 ^ n step
54                 // from y to a parent we call z
55                 // so the total equal to move 2 ^ n * 2 ^ n
56                 // steps from x to z
57                 // which is move 2 ^ (n + 1) steps to z
58         }
59     }
60 }

```

10.2 Tree_Centroid

```

1 #include <utility>
2 #include <vector>
3 #include <algorithm>
4 using namespace std;
5
6 int subTsize[200005];
7 vector<int> adj[200005];
8 int n; // n for node num ??

```

```

10 pair<int, int> Tree_Centroid(int v, int pa)
11 {
12     // return (最大子樹節點數, 節點ID)
13     subTsize[v] = 1;
14     pair<int, int> res(INT_MAX, -1); // ans: tree centroid
15     int max_subT = 0; // 最大子樹節點數
16     for (size_t i = 0; i < adj[v].size(); ++i)
17     {
18         int x = adj[v][i];
19         if (x == pa)
20             continue;
21         res = min(res, Tree_Centroid(x, v));
22         subTsize[v] += subTsize[x];
23         max_subT = max(max_subT, subTsize[x]);
24     }
25     res = min(res, make_pair(max(max_subT, n - subTsize[v]),
26                             v)); // (n - subTsize[v]) for maybe parent tree is
27     // the biggest
28     // min because all res will be greater than n/2;
29     // the min one is the tree centroid
30     return res;
31 }
32 // Tree_Centroid2
33 vector<int> V[10005];
34 int N;
35 int center, csize;
36 int dfs(int v, int fa)
37 {
38     int sz = 1;
39     int maxsub = 0;
40     for (int u:V[v])
41     {
42         if (u==fa)continue;
43         int sub = dfs(u, v);
44         maxsub = max(maxsub, sub);
45         sz += sub;
46     }
47     maxsub = max(maxsub, N-sz);
48     if (maxsub<csize)
49     {
50         center = v;
51         csize = maxsub;
52     }
53     return sz;
54 }
55 }

```

11 hashing

11.1 hashingVec

```

1 #include<bits/stdc++.h>
2 struct VectorHash {
3     size_t operator()(const std::vector<int>& v) const {
4         std::hash<int> hasher;
5         size_t seed = 0;
6         for (const int& i : v) {
7             seed ^= hasher(i) + 0x9e3779b9 + (seed<<6) + (
8             seed>>2);
9         }
10     }
11 };

```

```

8     }
9     return seed;
10 }
11 };
12 std::unordered_set<std::vector<int>, VectorHash> H;

```

12 number_theory

12.1 Fib

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 // Cassini's identity : F_{n-1}F_{n+1} - F_n^2 = (-1)^n
4 // The "addition" rule : F_{n+k} = F_kF_{n+1} + F_{k-1}F_n
5 // k = n, F_{2n} = F_n(F_{n+1} + F_{n-1})
6 // F_{2k} = F_k(2F_{k+1} - F_k)
7 // F_{2k+1} = F_{k+1}^2 + F_k^2
8 pair<int, int> fib (int n) {
9     if (n == 0)
10         return {0, 1};
11     auto p = fib(n >> 1);
12     int c = p.first * (2 * p.second - p.first);
13     int d = p.first * p.first + p.second * p.second;
14     if (n & 1)
15         return {d, c + d};
16     else
17         return {c, d};
18 }

```

12.2 BigInteger

```

1 #include <vector>
2 #include <string>
3 #include <iostream>
4 #include <cmath>
5 #include <algorithm>
6 #include <cstdio>
7 #include <cstring>
8 using namespace std;
9 struct BigInteger{
10     static const int BASE = 100000000;
11     static const int WIDTH = 8;
12     vector<int> s;
13
14     BigInteger(long long num = 0) { *this = num; }
15     BigInteger operator = (long long num) {
16         s.clear();
17         do{
18             s.push_back(num % BASE);
19             num /= BASE;
20         } while (num > 0);
21         return *this;
22     }
23
24     BigInteger operator = (const string& str){
25         s.clear();
26         int x, len = (str.length() - 1) / WIDTH + 1;
27         for (int i = 0; i < len; i++){

```

```

28         int end = str.length() - i * WIDTH;
29         int start = max(0, end - WIDTH);
30         sscanf(str.substr(start, end - start).c_str(), "%
31             d", &x);
32         s.push_back(x);
33     }
34     return *this;
35 }
36
37 BigInteger operator+ (const BigInteger b) const{
38     BigInteger c;
39     c.s.clear();
40     for(int i=0, g=0; i<b.s.size(); i++){
41         if(g==0 && i >= s.size() && i >= b.s.size())
42             break;
43         int x = g;
44         if(i<s.size()) x+=s[i];
45         if(i<b.s.size()) x+=b.s[i];
46         c.s.push_back(x % BASE);
47         g = x / BASE;
48     }
49     return c;
50 }
51
52 BigInteger operator+=(const BigInteger& b){
53     *this = *this + b;
54     return *this;
55 }
56
57 BigInteger operator* (const BigInteger b) const{
58     BigInteger c;
59     c.s.clear();
60     long long mul;
61     for (int i = 0; i < s.size(); i++)
62     {
63         long long carry = 0;
64         for (int g = 0; g < b.s.size(); g++){
65             mul = (long long)(s[i]) * (long long)(b.s[g])
66                 + carry;
67             if(i + g < c.s.size()){
68                 c.s[i+g] += mul % BASE;
69             }else{
70                 c.s.push_back(mul % BASE);
71             }
72             carry = mul / BASE;
73         }
74         for (int i = 0; i < c.s.size(); i++){
75             if(c.s[i] >= BASE){
76                 if(i + 1 < c.s.size()){
77                     c.s.push_back(c.s[i] / BASE);
78                 }else{
79                     c.s[i + 1] += c.s[i] / BASE;
80                 }
81                 c.s[i] %= BASE;
82             }
83         }
84         return c;
85     }
86 }
87
88 bool operator< (const BigInteger& b) const{
89     if(s.size() != b.s.size()) return s.size() < b.s.size();
90     for(int i=s.size()-1; i>=0; i--){
91         if(s[i] != b.s[i]) return s[i] < b.s[i];
92     }
93     return false; // Equal
94 }

```

```

95 bool operator> (const BigInteger& b) const{return b < *
96     this;}
97 bool operator<= (const BigInteger& b) const {return !(b<*
98     this);}
99 bool operator>= (const BigInteger& b) const {return !(*
100     this < b);}
101 bool operator!=(const BigInteger& b) const {return b<*
102     this || *this < b;}
103 bool operator==(const BigInteger& b) const {return !(b<*
104     this) && !(*this<b);}
105 };
106 ostream& operator<< (ostream &out, const BigInteger& x){
107     out << x.s.back();
108     for (int i = x.s.size() - 2; i >= 0; i--){
109         char buf[20];
110         sprintf(buf, "%08d", x.s[i]);
111         for(int j = 0; j<strlen(buf); j++) out << buf[j];
112     }
113     return out;
114 }
115
116 istream& operator>> (istream &in, BigInteger &x){
117     string s;
118     if(!(in >> s)) return in;
119     x = s;
120     return in;
121 }

```

12.3 gcds

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 // O(log(min(a, b)))
4 int gcd(int a, int b, int& x, int& y) {
5     if (b == 0) {
6         x = 1;
7         y = 0;
8         return a;
9     }
10     int x1, y1;
11     int d = gcd(b, a % b, x1, y1);
12     x = y1;
13     y = x1 - y1 * (a / b);
14     return d;
15 }
16
17 bool find_any_solution(int a, int b, int c, int &x0, int &y0,
18     int &g) {
19     g = gcd(abs(a), abs(b), x0, y0);
20     if (c % g) {
21         return false;
22     }
23
24     x0 *= c / g;
25     y0 *= c / g;
26     if (a < 0) x0 = -x0;
27     if (b < 0) y0 = -y0;
28     return true;
29 }
30
31 // finding all solution
32 void shift_solution(int &x, int &y, int a, int b, int cnt)
33 {
34     x += cnt * b;

```

```

33     y -= cnt * a;
34 }
35
36 int find_all_solutions(int a, int b, int c, int minx, int
    maxx, int miny, int maxy) {
37     int x, y, g;
38     if (!find_any_solution(a, b, c, x, y, g))
39         return 0;
40     a /= g;
41     b /= g;
42
43     int sign_a = a > 0 ? +1 : -1;
44     int sign_b = b > 0 ? +1 : -1;
45
46     shift_solution(x, y, a, b, (minx - x) / b);
47     if (x < minx)
48         shift_solution(x, y, a, b, sign_b);
49     if (x > maxx)
50         return 0;
51     int lx1 = x;
52
53     shift_solution(x, y, a, b, (maxx - x) / b);
54     if (x > maxx)
55         shift_solution(x, y, a, b, -sign_b);
56     int rx1 = x;
57
58     shift_solution(x, y, a, b, -(miny - y) / a);
59     if (y < miny)
60         shift_solution(x, y, a, b, -sign_a);
61     if (y > maxy)
62         return 0;
63     int lx2 = x;
64
65     shift_solution(x, y, a, b, -(maxy - y) / a);
66     if (y > maxy)
67         shift_solution(x, y, a, b, sign_a);
68     int rx2 = x;
69
70     if (lx2 > rx2)
71         swap(lx2, rx2);
72     int lx = max(lx1, lx2);
73     int rx = min(rx1, rx2);
74
75     if (lx > rx)
76         return 0;
77     return (rx - lx) / abs(b) + 1;
78 }
79 // smallest possible val
80 // x' + y' = x + y + k(b-a)g, minimize b-a

```

12.4 nCr

```

1 using i64 = long long;
2 #define maxn 300005
3 i64 fact[MAXN], tcaf[MAXN];
4
5 #define P 998244353
6 #define REP1(i, n) for (int i = 1; i <= (int)(n); ++i)
7 #define REP(i, n) for (int i = (int)(n) - 1; i >= 0; --i)
8 void init(int n){
9     fact[0] = 1;
10    for (int i = 1; i <= n; i++)
11        fact[i] = i * fact[i - 1] % P;

```

```

12    for (int i = n; i >= 0; --i)
13        tcaf[i] = deg(fact[i], -1);
14
15 }
16
17 i64 deg(i64 x, i64 d) {
18     if (d < 0) d += P - 1;
19     i64 y = 1;
20     while (d) {
21         if (d & 1) (y *= x) %= P;
22         d /= 2;
23         (x *= x) %= P;
24     }
25     return y;
26 }
27
28 i64 cnk(int n, int k) {
29     if (k < 0 || k > n) return 0;
30     return fact[n] * tcaf[k] % P * tcaf[n - k] % P;
31 }

```

13 string

13.1 KMP

```

1 #include <iostream>
2 #include <string>
3 #include <regex>
4 #include <vector>
5 using namespace std;
6 // T for Text, P for Pattern
7 vector<int> build_kmp(const string &P) {
8     vector<int> f(P.size());
9     int fp = f[0] = -1;
10    for (int i = 1; i < P.size(); ++i) {
11        while (~fp && P[fp + 1] != P[i])
12            fp = f[fp];
13        if (P[fp + 1] == P[i])
14            ++fp;
15        f[i] = fp;
16    }
17
18    return f;
19 }
20 vector<int> kmp_match(vector<int> fail, const string &P,
    const string &T)
21 {
22     vector<int> res; // start from these points
23     const int n = P.size();
24     for (int j = 0, i = -1; j < T.size(); ++j) {
25         while (~i && T[j] != P[i + 1])
26             i = fail[i];
27         if (P[i + 1] == T[j])
28             ++i;
29         if (i == n - 1)
30             res.push_back(j - n + 1, i = fail[i]);
31     }
32     return res;
33 }
34 int main(){
35     char control;

```

```

36     string test_pattern = "a";
37     string test_text = "abcdabcbabceabcd";
38     // for testing
39     cout << "Do you want to Enter by ys?(y/n)";
40     cin >> control;
41     if(control == 'y' || control == 'Y'){
42         cout << "Enter text:";
43         cin >> test_text;
44         cout << "Enter pattern:";
45         cin >> test_pattern;
46     }
47
48     vector<int> V = build_kmp(test_pattern);
49     vector<int> Ans = kmp_match(V, test_pattern, test_text);
50     cout << '\n';
51     for(auto it = V.begin(); it != V.end(); ++it)
52         cout << *it << ' ';
53     cout << '\n';
54     for(auto it = Ans.begin(); it != Ans.end(); ++it)
55         cout << *it << ' ';
56     return 0;
57 }

```


ACM ICPC TEAM REFERENCE - DURACELL!

Contents

1 data_structure	1	2 geometry	1	6 graph/Flow	6	9.5 shortest-path_on_DAG . . .	11
1.1 Sparse_Table	1	2.1 closest_point	1	6.1 Ford_Fulkerson	6	10 graph/Tree	12
1.2 segment_Tree	1	2.2 points	2	6.2 Edmonds-Karp-adjmax	7	10.1 Lowest_Common_Ancestor .	12
1.3 disjointset	1	2.3 lines	2	6.3 Dinic_algorithm	7	10.2 Tree_Centroid	12
		2.4 geometry_template	2	6.4 Edmonds_Karp_2	7		
		2.5 cp_geometry.	3	6.5 MinCostMaxFlow	8	11 hashing	12
		3 geometry/Convex_Hull	4	7 graph/Matching	8	11.1 hashingVec	12
		3.1 Andrew's_Monotone_Chain	4	7.1 blossom_matching	8		
		4 graph	4	8 graph/Minimum_Spanning_Tree	9	12 number_theory	13
		4.1 Kosaraju_for_SCC	4	8.1 prim	9	12.1 Fib	13
		4.2 Tarjan_for_BridgeCC	5	8.2 Kruskal	9	12.2 BigInterger	13
		4.3 Tarjan_for_AP_Bridge	5	9 graph/Shortest_Path	9	12.3 gcds	13
		4.4 Tarjan_for_SCC	5	9.1 dijkstra	9	12.4 nCr	14
		5 graph/Bipartite	6	9.2 dijkstra-alrightchiu-version . .	9	13 string	14
		5.1 konig_algorithm	6	9.3 bellman-Ford	10	13.1 KMP	14
		5.2 Kuhn-Munkres	6	9.4 Floyd-Warshall	10		