

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;
14             ++i)
15             sparse[j][i] = min(
16                 sparse[j - 1][i],
17                 sparse[j - 1][i + (1 << (j - 1))]);
18 }
19 inline int query(int l, int r)
20 {
21     int k = __lg(r - l + 1);
22     return min(sparse[k][l], sparse[k][r -
23         (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
19             + seq[IR(index)].val;
20     }
21     void push(int index){
22         seq[IL(index)].lazy_tag += seq[index
23             ].lazy_tag;
24         seq[IR(index)].val += seq[index].
25             lazy_tag * seq[IL(index)].size;
```

```

26         seq[IR(index)].lazy_tag += seq[index
27             ].lazy_tag;
28         seq[IR(index)].val += seq[index].
29             lazy_tag * seq[IR(index)].size;
30         seq[index].lazy_tag = 0;
31     }
32
33     void build(int L, int R, int index){
34         if(L == R){
35             seq[index].val = dataseq[L];
36             seq[index].size = 1;
37             seq[index].lazy_tag = 0;
38             return;
39         }
40         int M = (L + R) / 2;
41         build(L, M, IL(index));
42         build(M + 1, R, IR(index));
43         seq[index].size = seq[IL(index)].size
44             + seq[IR(index)].size;
45         pull(index);
46     }
47
48     void modify(int l, int r, int L, int R,
49         int index, long long Add){
50         if(l == L && r == R){
51             seq[index].lazy_tag += Add;
52             seq[index].val += Add * seq[
53                 index].size;
54             return;
55         }
56         push(index);
57         int M = (L + R) / 2;
58
59         if(r <= M){
60             modify(l, r, L, M, IL(index),
61                 Add);
62         }
63         else if(l > M){
64             modify(l, r, M + 1, R, IR(index),
65                 Add);
66         }
67         else{
68             modify(l, M, L, M, IL(index),
69                 Add);
70             modify(M + 1, r, M + 1, R, IR(
71                 index), Add);
72         }
73         pull(index);
74     }
75
76     long long Query(int l, int r, int L, int
77         R, int index){
78         if(l == L && r == R){
79             return seq[index].val;
80         }
81         int M = (L + R) / 2;
82         push(index);
83         if(r <= M){
84             return Query(l, r, L, M, IL(
85                 index));
86         }
87         else if(l > M){
88             return Query(l, r, M + 1, R, IR(
89                 index));
90         }
91         else{
92             return Query(l, M, L, M, IL(
93                 index)) +
94                 Query(M + 1, r, M + 1, R, IR(
95                     index));
96         }
97     }
98 }
```

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<
3     Point<T>>::iterator l,
4     typename vector<
5     Point<T>>::
6     iterator r)
7 {
8     auto delta = numeric_limits<T>::max();
9     if (r - l > 1)
10     {
11         auto m = l + (r - l >> 1);
12         nth_element(l, m, r); //
13         Lexicographical order in default
14         auto x = m->x;
```

```

11         delta = min(
12             ClosestPairSquareDistance<T>(l,
13                 m),
14             ClosestPairSquareDistance
15                 <T>(m, r));
16         auto square = [&](T y) { return y *
17             y; };
18         auto sgn = [=](T a, T b) {
19             return square(a - b) <= delta ?
20                 0 : a < b ? -1 : 1;
21         };
22         vector<Point<T>> x_near[2];
23         copy_if(l, m, back_inserter(x_near
24             [0]), [=](Point<T> a) {
25                 return sgn(a.x, x) == 0;
26             });
27         copy_if(m, r, back_inserter(x_near
28             [1]), [=](Point<T> a) {
29                 return sgn(a.x, x) == 0;
30             });
31         for (int i = 0, j = 0; i < x_near
32             [0].size(); ++i)
33         {
34             while (j < x_near[1].size() and
35                 sgn(x_near[1][j].y,
36                     x_near[0][i].y) ==
37                     -1)
38                 ++j;
39             for (int k = j; k < x_near[1].
40                 size() and
41                     sgn(x_near[1][k]
42                         .y, x_near
43                             [0][i].y) ==
44                             0;
45                 ++k)
46             {
47                 delta = min(delta, (x_near
48                     [0][i] - x_near[1][k]).
49                     norm());
50             }
51         }
52         inplace_merge(l, m, r, [](Point<T> a
53             , Point<T> b) {
54             return a.y < b.y;
55         });
56     }
57     return delta;
58 }
```

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     {}
8     template <class F>
9     explicit operator Point<F>() const
10     {
11         return Point<F>((F)x, (F)y);
12     }
13 }
```

```

12 Point operator+(const Point b) const
13 {
14     return Point(x + b.x, y + b.y);
15 }
16 Point operator-(const Point b) const
17 {
18     return Point(x - b.x, y - b.y);
19 }
20 template <class F>
21 Point<F> operator*(const F fac)
22 {
23     return Point<F>(x * fac, y * fac);
24 }
25 template <class F>
26 Point<F> operator/(const F fac)
27 {
28     return Point<F>(x / fac, y / fac);
29 }
30 T operator&(const Point b) const {
31     return x * b.x + y * b.y; }
32 // 內積運算子
33 T operator^(const Point b) const {
34     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(-
46     x, -y); }
47 T norm() const { return *this & *this; }
48 // 歐式長度平方
49 Point prep() const { return Point(-y, x)
50     ; } // 左旋直角法向量
51
52 template <class F>
53 istream &operator>>(istream &is, Point<F
54     > &pt)
55 {
56     return is >> pt.x >> pt.y;
57 }
58 template <class F>
59 ostream &operator<<(ostream &os, const
60     Point<F> &pt)
61 {
62     return os << pt.x << " " << pt.y;
63 }
64 };

```

2.3 lines

```

1 template <typename T, typename Real = double
2     >
3 struct Line

```

```

4 Point<T> st, ed;
5 Point<T> vec() const { return ed - st; }
6 T ori(const Point<T> p) const { return (
7     ed - st) ^ (p - st); }
8 Line(const Point<T> x, const Point<T> y)
9 : st(x), ed(y) {}
10 template <class F>
11 operator Line<F>() const
12 {
13     return Line<F>((Point<F>)st, (Point<
14     F>)ed);
15 }
16 // sort by arg, the left is smaller for
17 // parallel lines
18 bool operator<(Line B) const
19 {
20     Point<T> a = vec(), b = B.vec();
21     auto sgn = [](const Point<T> t) {
22         return (t.y == 0 ? t.x : t.y) <
23         0; };
24     if (sgn(a) != sgn(b))
25         return sgn(a) < sgn(b);
26     if (abs(a ^ b) == 0)
27         return B.ori(st) > 0;
28     return (a ^ b) > 0;
29 }
30 // Regard a line as a function
31 template <typename F>
32 Point<F> operator()(const F x) const
33 {
34     return Point<F>(st) + vec() * x;
35 }
36 bool isSegProperIntersection(const Line
37     l) const
38 {
39     return l.ori(st) * l.ori(ed) < 0 and
40     ori(l.st) * ori(l.ed) < 0;
41 }
42 bool isPointOnSegProperly(const Point<T>
43     p) const
44 {
45     return ori(p) == 0 and ((st - p) & (
46     ed - p)) < 0;
47 }
48 Point<Real> getIntersection(const Line<
49     Real> l)
50 {
51     Line<Real> h = *this;
52     return l((l.st ^ h.vec()) / (h.vec()
53     ^ l.vec()));
54 }
55 Point<Real> projection(const Point<T> p)
56     const
57 {
58     return operator()(((p - st) & vec())
59     / (Real)(vec().norm()));
60 }
61 };

```

2.4 geometry_template

```

1 // import from https://codeforces.com/blog/
2 // entry/48122
3 #include <iostream>
4 using namespace std;
5 template <class F>
6 struct Point {
7     F x, y;
8     Point() : x(0), y(0) {}
9     Point(const F& x, const F& y) : x(x), y(y)
10     {}
11 void swap(Point& other) { using std::swap;
12     swap(x, other.x); swap(y, other.y); }
13 template <class F1> explicit operator
14     Point<F1> () const {
15         return Point<F1>(static_cast<F1>(x),
16         static_cast<F1>(y)); }
17 template <class F1> Point& operator = (
18     const Point<F1>& other) {
19     x = other.x; y = other.y; return *this;
20 }
21 template <class F1> Point& operator += (
22     const Point<F1>& other) {
23     x += other.x; y += other.y; return *this
24     ; }
25 template <class F1> Point& operator -= (
26     const Point<F1>& other) {
27     x -= other.x; y -= other.y; return *this
28     ; }
29 template <class F1> Point& operator *= (
30     const F1& factor) {
31     x *= factor; y *= factor; return *this;
32 }
33 template <class F1> Point& operator /= (
34     const F1& factor) {
35     x /= factor; y /= factor; return *this;
36 }
37 };
38 template <class F> int read(Point<F>& point)
39 { return read(point.x, point.y) / 2; }
40 template <class F> int write(const Point<F>&
41     point) { return write(point.x, point.y)
42     ; }
43 template <class F> istream& operator >> (
44     istream& is, Point<F>& point) {
45     return is >> point.x >> point.y; }
46 template <class F> ostream& operator << (
47     ostream& os, const Point<F>& point) {
48     return os << point.x << " " << point.y; }
49 template <class F> inline Point<F> makePoint
50     (const F& x, const F& y) { return Point<
51     F>(x, y); }
52 template <class F> void swap(Point<F>& lhs,
53     Point<F>& rhs) { lhs.swap(rhs); }
54 #define FUNC1(name, arg, expr) \
55 template <class F> inline auto name(const
56     arg) -> decltype(expr) { return expr; }
57 #define FUNC2(name, arg1, arg2, expr) \
58 template <class F1, class F2> \

```

```

40 inline auto name(const arg1, const arg2) ->
41     decltype(expr) { return expr; }
42 #define FUNC3(name, arg1, arg2, arg3, expr) \
43     \
44 template <class F1, class F2, class F3> \
45 inline auto name(const arg1, const arg2,
46     const arg3) -> decltype(expr) { return
47     expr; }
48 FUNC1(operator -, Point<F>& point, makePoint
49     (-point.x, -point.y))
50 FUNC2(operator +, Point<F1>& lhs, Point<F2>&
51     rhs, makePoint(lhs.x + rhs.x, lhs.y +
52     rhs.y))
53 FUNC2(operator -, Point<F1>& lhs, Point<F2>&
54     rhs, makePoint(lhs.x - rhs.x, lhs.y -
55     rhs.y))
56 FUNC2(operator *, F1& factor, Point<F2>& rhs
57     , makePoint(factor * rhs.x, factor * rhs
58     .y))
59 FUNC2(operator *, Point<F1>& lhs, F2& factor
60     , makePoint(lhs.x * factor, lhs.y *
61     factor))
62 FUNC2(operator /, Point<F1>& lhs, F2& factor
63     , makePoint(lhs.x / factor, lhs.y /
64     factor))
65 FUNC2(operator *, Point<F1>& lhs, Point<F2>&
66     rhs, lhs.x * rhs.x + lhs.y * rhs.y)
67 FUNC2(operator ^, Point<F1>& lhs, Point<F2>&
68     rhs, lhs.x * rhs.y - lhs.y * rhs.x)
69 // < 0 if rhs <- lhs counter-clockwise, 0 if
70 // collinear, > 0 if clockwise.
71 FUNC2(ccw, Point<F1>& lhs, Point<F2>& rhs,
72     rhs ^ lhs)
73 FUNC3(ccw, Point<F1>& lhs, Point<F2>& rhs,
74     Point<F3>& origin, ccw(lhs - origin, rhs
75     - origin))
76 FUNC2(operator ==, Point<F1>& lhs, Point<F2
77     >& rhs, lhs.x == rhs.x && lhs.y == rhs.y)
78 FUNC2(operator !=, Point<F1>& lhs, Point<F2
79     >& rhs, !(lhs == rhs))
80 FUNC2(operator <, Point<F1>& lhs, Point<F2>&
81     rhs,
82     lhs.y < rhs.y || (lhs.y == rhs.y && lhs.
83     x < rhs.x))
84 FUNC2(operator >, Point<F1>& lhs, Point<F2>&
85     rhs, rhs < lhs)
86 FUNC2(operator <=, Point<F1>& lhs, Point<F2
87     >& rhs, !(lhs > rhs))
88 FUNC2(operator >=, Point<F1>& lhs, Point<F2
89     >& rhs, !(lhs < rhs))
90 // Angles and rotations (counter-clockwise).
91 FUNC1(angle, Point<F>& point, atan2(point.y,
92     point.x))
93 FUNC2(angle, Point<F1>& lhs, Point<F2>& rhs,
94     atan2(lhs ^ rhs, lhs * rhs))
95 FUNC3(angle, Point<F1>& lhs, Point<F2>& rhs,
96     Point<F3>& origin,
97     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,
78     origin + rotate(point - origin, angle)
    )
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
17     template <class F>
18     explicit operator Point<F>() const
19     {
20         return Point<F>((F)x, (F)y);
21     }
22
23     Point operator+(const Point b) const
24     {
25         return Point(x + b.x, y + b.y);
26     }
27     Point operator-(const Point b) const
28     {
29         return Point(x - b.x, y - b.y);
30     }

```

```

31     template <class F>
32     Point<F> operator*(const F fac)
33     {
34         return Point<F>(x * fac, y * fac);
35     }
36     template <class F>
37     Point<F> operator/(const F fac)
38     {
39         return Point<F>(x / fac, y / fac);
40     }
41
42     T operator&(const Point b) const {
43         return x * b.x + y * b.y; }
44     // dot operator
45     T operator^(const Point b) const {
46         return x * b.y - y * b.x; }
47     // cross operator
48     bool operator==(const Point b) const
49     {
50         return x == b.x and y == b.y;
51     }
52     bool operator<(const Point b) const
53     {
54         return x == b.x ? y < b.y : x < b.x;
55     } // 字典序
56
57     Point operator-() const { return Point(-
58         x, -y); }
59     T norm() const { return *this & *this; }
60     // 歐式長度平方
61     Point prep() const { return Point(-y, x)
62         ; } // 左旋直角法向量
63
64     template <class F>
65     friend istream &operator>>(istream &is,
66         Point<F> &pt);
67     template <class F>
68     friend ostream &operator<<(ostream &os,
69         const Point<F> &pt);
70
71     template <class F>
72     bool collinearity(const Point<F>& p1, const
73         Point<F>& p2, const Point<F>& p3){
74         return (p1 - p3) ^ (p2 - p3) == 0;
75     }
76     // return fcmp((p1 - p3) ^ (p2 - p3), 0.0)
77     == 0;
78
79     // check co-line first. properly
80     template <class F>
81     inline bool btw(const Point<F>& p1, const
82         Point<F>& p2, const Point<F>& p3){
83         return fcmp((p1 - p3) & (p2 - p3), 0.0)
84             <= 0;
85     }

```

```

86     // is p3 on (p1, p2)?
87     template <class F>
88     inline bool pointOnSegment(const Point<F>&
89         p1, const Point<F>& p2, const Point<F>&
90         p3){
91         return collinearity(p1, p2, p3) && btw(
92             p1, p2, p3);
93     }
94
95     template <typename T, typename Real = double
96         >
97     struct Line
98     {
99         Point<T> st, ed;
100         Point<T> vec() const { return ed - st; }
101         T ori(const Point<T> p) const { return (
102             ed - st) ^ (p - st); }
103         int orint(const Point<T> p) const{
104             T a = this->ori(p);
105             return (fcmp(a, 0.0)); // 1 on posi-
106                 side // -1 nega-side
107             // a little bit useless?
108         }
109         Line(const Point<T> x, const Point<T> y)
110             : st(x), ed(y) {}
111         template <class F>
112         operator Line<F>() const
113         {
114             return Line<F>((Point<F>)st, (Point<
115                 F>)ed);
116         }
117
118         // sort by arg, the left is smaller for
119         parallel lines
120         bool operator<(Line B) const
121         {
122             Point<T> a = vec(), b = B.vec();
123             auto sgn = [](const Point<T> t) {
124                 return (t.y == 0 ? t.x : t.y) <
125                     0; };
126             if (sgn(a) != sgn(b))
127                 return sgn(a) < sgn(b);
128             if (abs(a ^ b) == 0)
129                 return B.ori(st) > 0;
130             return (a ^ b) > 0;
131         }
132
133         // Regard a line as a function
134         template <typename F>
135         Point<F> operator()(const F x) const //
136             A + AB * x = the point position.
137         {
138             return Point<F>(st) + vec() * x;
139         }
140
141         bool isSegProperIntersection(const Line
142             l) const
143         {
144             return l.ori(st) * l.ori(ed) < 0 and
145                 ori(l.st) * ori(l.ed) < 0;
146         }
147         bool isSegIntersection(const Line l)
148             const{
149             Line<Real> h = *this;

```

```

150         // hst = 1, hed = 2, lst = 3, led =
151             4
152         double hlst = h.ori(l.st);
153         double hled = h.ori(l.ed);
154         double lhst = l.ori(h.st);
155         double lhed = l.ori(h.ed);
156         if(fcmp(hlst, 0.0) == 0 && fcmp(hled
157             , 0.0) == 0)
158             return h.isPointOnSeg(l.st) || h
159                 .isPointOnSeg(l.ed) || l.
160                 isPointOnSeg(h.st) || l.
161                 isPointOnSeg(h.ed);
162         return fcmp(hlst * hled, 0.0) <= 0
163             && fcmp(lhst * lhed, 0.0) <= 0;
164     }
165
166     bool isPointOnSegProperly(const Point<T>
167         p) const
168     {
169         return fcmp(ori(p), 0.0) == 0 and
170             fcmp(((st - p) & (ed - p)), 0.0)
171                 < 0;
172     }
173     bool isPointOnSeg(const Point<T>p) const
174     {
175         return fcmp(ori(p), 0.0) == 0 and
176             fcmp((st - p) & (ed - p), 0.0)
177                 <= 0;
178     }
179     Real disP2Line(const Point<T> p) const
180     {
181         return Line<double>(projection(p),
182             Point<double>(p)).vec().norm();
183     }
184
185     // notice if you should check Segment
186     intersect or not;
187     // be careful divided by 0
188     Point<Real> getIntersection(Line<Real> l
189         )
190     {
191         Line<Real> h = *this;
192         return l(((l.st - h.st) ^ h.vec())
193             / (h.vec() ^ l.vec())); // use operator
194             ()
195         Real hlst = -h.ori(l.st);
196         Real hled = h.ori(l.ed);
197         return ((l.st * hled) + (l.ed * hlst
198             )) / (hlst + hled);
199
200         // 需要確認+-號的合理性
201         // Area of triangle(l.st, h.st, h.ed
202             ) divided by Area of
203             Quadrilateral(h.st, l.st, h.ed,
204                 l.ed)
205     }
206
207     Point<Real> projection(const Point<T> p)
208         const
209     {
210         return operator()(((p - st) & vec())
211             / (Real)(vec().norm()));
212     }
213     template <class F>

```

```

174 friend ostream &operator<<(ostream &os,
175     const Line<F> &l);
176 };
177 template <class F>
178 ostream &operator<<(ostream &os, const Line<
179     F> &l)
180 {
181     return os << "(" << l.st.x << ", " << l.
182         st.y << ")" to "(" << l.ed.x << ", "
183         << l.ed.y << ")";
184 }
185 template <class F>
186 using Polygon = vector<Point<F>>;
187 template <class F>
188 Polygon<F> getConvexHull(Polygon<F> points)
189 {
190     sort(points.begin(), points.end());
191     Polygon<F> CH;
192     CH.reserve(points.size() + 1); // for
193     what ??
194     for (int round = 0; round < 2; round++){
195         int start = CH.size();
196         for (Point<int> &pt: points) {
197             while (CH.size() - start >= 2 &&
198                 Line<F>(CH[CH.size() - 2],
199                     CH[CH.size() - 1]).ori(pt)
200                     <= 0) // ? Line is different
201                     than senpai's .
202             {
203                 CH.pop_back();
204             }
205             CH.emplace_back(pt);
206         }
207         CH.pop_back();
208         reverse(points.begin(), points.end()
209             );
210     }
211     if (CH.size() == 2 && CH[0] == CH[1])
212         CH.pop_back();
213     return CH;
214 }

```

3 geometry/Con- vex_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             &&
18             Line<F>(hull.back(), hull
19                 [hull.size() - 2]).
20                 ori(point) <= 0)
21             hull.pop_back();
22         // whenever point is at the
23         RIGHT(NEGATIVE) part of the
24         line(hull[size - 1], hull[
25             size-2])
26         // pop the last point because it
27         causes concave hull
28         hull.push_back(point);
29     }
30    hull.pop_back();
31    reverse(begin(points), end(points));
32    if (hull.size() == 2 and hull[0] == hull
33        [1])
34        hull.pop_back();
35    return hull;
36 }

```

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 :
37         GT);
38     for(int w: dG[v])
39     {
40         if(!visited[w]){
41             DFS(isG, w, k);
42         }
43     }
44     if(isG){
45         st.push(v);
46     }
47 }
48 void Kosaraju(int N){
49     visited.clear();
50     visited.resize(N + 5, false);
51     for (int i = 1; i <= N; i++){
52         if(!visited[i])
53             DFS(true, i);
54     }
55     visited.clear();
56     visited.resize(N + 5, false);
57     while(!st.empty()){
58         if(!visited[st.top()])
59             DFS(false, st.top(), sccID
60                 ++);
61         st.pop();
62     }
63 }
64 vector<vector<int>> generateReG(){
65     vector<vector<int>> reG;
66     reG.resize(sccID);
67     for (int i = 1; i <= NodeNum; i++){
68         for(int w: G[i]){
69             if(scc[i] == scc[w])
70                 continue;
71             reG[scc[i]].emplace_back(scc[
72                 w]);
73         }
74     }
75     return reG;
76 }
77 };

```

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
17     first
18     D[v] = L[v] = timestamp++; //timestamp >
19     0
20     st.emplace(v);
21     for (int w:adj[v]) {
22         if( w==fa ) continue;
23         if ( !D[w] ) { // D[w] = 0 if not
24             visited
25             DFS(w,v);
26             L[v] = min(L[v], L[w]);
27         }
28         L[v] = min(L[v], D[w]);
29     }
30     if (L[v]==D[v]) {
31         bccid++;
32         int x;
33         do {
34             x = st.top(); st.pop();
35             bcc[x] = bccid;
36         } while (x!=v);
37     }
38     return ;
39 }

```

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,
6     node-index
7 #define con pair<int, int> // connection
8
9 class tarjan{
10     vector<vector<int>> G; // adjacency List
11     vector<int> D; // visit or visited and
12     D-value
13     vector<con> edgeBridge;
14     vector<int> APnode;
15     int timestamp;
16     tarjan(int size = 1){
17         timestamp = 0;
18         G.resize(size);
19         D.resize(size, 0);
20         L.resize(size, 0);
21         edgeBridge.clear();
22         APnode.clear();
23     }
24     void init(int size = 1){
25         tarjan(size);
26     }
27     void addedge(int u, int v)
28     { // undirected graph
29         G[u].push_back(v);
30         G[v].push_back(u);
31     }
32 }

```

```

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 點以外
                的所有點 v · v 點在 G
                上為 AP 的充要條件
                為其在 T 中至少有一
                個子節點 w 滿足 D(v)
                ≤ L(w)
43         if(D[v] < L[w])
44             edgeBridge.emplace_back(
                v,w); // 結論 3 對於
                包含 r 在內的所有點
                v 和 v 在 T 中的子節
                點 w · 邊 e(v, w) 在
                圖 G 中為bridge 的充
                要條件為 D(v) < L(w)
                。
                L[v] = min(L[v], L[w]);
            }
            L[v] = min(L[v], D[w]);
        }
        if(v == pa && Childcount < 2)
45             isAP = false;
46         if(isAP)
47             APnode.emplace_back(v);
48     }
49 }
50 }
51 }
52 }
53 }
54 }

```

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);

```

```

        D.clear();
        D.resize(size + 3, 0);
        L.clear();
        L.resize(size + 3, 0);
        sccID.clear();
        sccID.resize(size + 3, 0);
        while(!st.empty())
            st.pop();
        inSt.clear();
        inSt.resize(size + 3, false);
        reG.clear();
        sccIDstamp = timeStamp = 1;
    }
    void addEdge(int from, int to){
        G[from].emplace_back(to);
    }
    void DFS(int v, int pa){ //call DFS(v,v)
        at first
        D[v] = L[v] = timeStamp++; //
            timestamp > 0
        st.push(v);
        inSt[v] = true;
        for(int w: G[v]){ // directed graph
            don't need w == pa
            if(!D[w]){ // D[w] = 0 if not
                visited
            DFS(w, v);
            L[v] = min(L[v], L[w]);
            }else if(inSt[w])
            { /* w has been visited.
                if we don't add this, the L[
                v] will think that v can
                back to node whose
                index less to v.
            inSt[w] is true that v -> w
                is a cross edge
                opposite it's a forward edge
            */
            L[v] = min(L[v], D[w]); //
                why D[w] instead of L[w]
                ]??
            }
        }
        if(D[v] == L[v]){
            int w;
            do{
                w = st.top();
                st.pop();
                sccID[w] = sccIDstamp; //
                    scc ID for this pooint
                    at which SCC
                inSt[w] = false;
            } while (w != v);
            sccIDstamp++;
        }
    }
    void generateReG(int N = 1){
        reG.clear();
        reG.resize(sccIDstamp);
        for (int i = 1; i <= N; i++){
            for(int w: G[i]){
                if(sccID[i] == sccID[w])
                    continue;
                reG[sccID[i]].emplace_back(
                    sccID[w]);

```

```

        }
    }
    bool visited(int v){
        return D[v];
    }
};

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         {
21             match[e] = v;
22             return true;
23         }
24     }
25     return false;
26 }
27 int konig()
28 {
29     memset(match, -1, sizeof(match));
30
31     int ans=0;
32     for(int i=1; i<=n; ++i)
33     {
34         memset(used, 0, sizeof(used));
35         if( dfs(i) )
36             ans++;
37     }
38     return ans;
39 }
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, bakward
   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(VE²)
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]=i)
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(V²E) 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 }
16 bool bfs(int s) {
17     memset(dist, -1, sizeof(dist));
18     queue<int> qu;
19     qu.emplace(s);
20     dist[s]=0;
21
22     while( !qu.empty() ) {
23         int S = qu.front(); qu.pop();
24         for(auto &p : V[S]) {
25             int E, C;
26             tie(E, C, ignore) = p;
27             if( dist[E]==-1 && C!=0 ) {
28                 dist[E]=dist[S]+1;
29                 qu.emplace(E);
30             }
31         }
32     }
33     return dist[End] != -1;
34 }
35 int dfs(int v, int f) {
36     int e, w, rev;
37     if( v==End || f==0 ) return f;
38     for( auto &t : V[v] )
39     {
40         tie(e, w, rev) = t;
41         if( dist[e]!=dist[v]+1 || w==0 )
42             continue;
43
44         w = dfs(e, min(w, f));
45         if( w>0 )
46         {
47             get<1>(t) -= w;
48             get<1>(V[e][rev]) += w;
49             return w;
50         }
51     }
52 }

```

```

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,
20            0));
21        edges.push_back(Edge(to, from, 0, 0)
22            ) // 反向弧
23        m = edges.size();
24        G[from].push_back(m - 2);
25        G[to].push_back(m - 1);
26    }
27    int Maxflow(int s, int t){
28        int flow = 0;
29        for (;;){
30            memset(a, 0, sizeof(a));
31            queue<int> Q;
32            Q.push(s);
33            a[s] = INF;
34            while(!Q.empty()){
35                int x = Q.front();
36                Q.pop();
37                for (int i = 0; i < G[x].
38                    size(); i++){
39                    Edge &e = edges[G[x][i]
40                        ];
41                    if(!a[e.to] && e.cap > e
42                        .flow){
43                        p[e.to] = G[x][i];
44                    }
45                }
46            }
47        }
48    }

```

```

39         a[e.to] = min(a[x], 38
40             e.cap - e.flow); 39
41         ;
42         Q.push(e.to);
43     }
44     if(a[t])
45         break;
46 }
47 if(!a[t])
48     break;
49 for (int u = t; u != s; u = edges[p[
50     u]].from){
51     edges[p[u]].flow += a[t];
52     edges[p[u] ^ 1].flow -= a[t];
53 }
54 flow += a[t];
55 return flow;
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),
10             pre(pre),r(r),cost(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,
22     bool directed=false){
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)break;
39 }

```

```

39     }
40     return CF-r;
41 }
42 bool modlabel(){
43     for(int u=0;u<n;++u)dis[u]=INF;
44     static deque<int>q;
45     dis[T]=0,q.push_back(T);
46     while(q.size()){
47         int u=q.front();q.pop_front();
48         TP dt;
49         for(int i=g[u];~i;i=e[i].pre){
50             if(e[i^1].r&&(dt=dis[u]-e[i].cost)<
51                 dis[e[i].v]){
52                 if((dis[e[i].v]=dt)<=dis[q.size()]){
53                     q.front():S;
54                     q.push_front(e[i].v);
55                 }else q.push_back(e[i].v);
56             }
57         }
58     }
59     for(int u=0;u<n;++u)
60         for(int i=g[u];~i;i=e[i].pre)
61             e[i].cost+=dis[e[i].v]-dis[u];
62     return PIS+=dis[S], dis[S]<INF;
63 }
64 TP mincost(int s,int t){
65     S=s,T=t;
66     PIS=ans=0;
67     while(modlabel()){
68         do memset(vis,0,sizeof(bool)*(n+1));
69         while(augment(S,INF));
70     }return ans;
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]==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 }

```

```

23     ans += w;
24     for(auto e: E[v]){
25         pq.emplace(-e.first, e.second);
26     }
27     return ans;
28 }

```

8.2 Kruskal

```

1 #include <tuple>
2 #include <vector>
3 #include <algorithm>
4 #include <numeric> // for iota(first, last,
5     val) setting iterator value
6 using namespace std;
7 struct DSU // disjoint set no rank-comp-
8     merge
9 {
10     vector<int> fa;
11     DSU(int n) : fa(n) { iota(fa.begin(), fa
12         .end(), 0); } // auto fill fa from 0
13     to n-1
14     int find(int x) { return fa[x] == x ? x
15         : fa[x] = find(fa[x]); }
16     void merge(int x, int y) { fa[find(x)] =
17         find(y); }
18 };
19 int kruskal(int V, vector<tuple<int, int,
20     int>> E) // save all edges into E,
21     instead of saving graph via adjacency
22     list
23 {
24     sort(E.begin(), E.end());
25     DSU dsu(V);
26     int mcnt = 0;
27     int ans = 0;
28     for (auto e : E)
29     {
30         int w, u, v; // w for start, u for
31         des, v for val
32         tie(w, u, v) = e;
33         if (dsu.find(u) == dsu.find(v))
34             continue;
35         dsu.merge(u, v);
36         ans += w;
37         if (++mcnt == V - 1)
38             break;
39     }
40     return ans;
41 }

```

8 graph/Minimum_Spanning

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,
6     node>
7 int prim_pq(vector<vector<enp>> E){
8     vector<bool> vis;
9     vis.resize(E.size(), false);
10     vis[0] = true;
11     priority_queue<enp> pq;
12     for(auto e: E[0]){
13         pq.emplace(-e.first, e.second);
14     }
15     int ans = 0; // min value for MST
16     while(pq.size()){
17         int w, v; // edge-weight, vertex
18         index
19         tie(w, v) = pq.top();
20         pq.pop();
21         if(vis[v])
22             continue;
23         w = -w;
24         vis[v] = true;

```

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
8     distance, second for index
9 vector<vector<con>> Graph; //
10 vector<int> dis; // distance;
11 int main(){
12     priority_queue<con, vector<con>, greater
13         <con>> pq;
14     dis[0] = 0;
15     pq.emplace(con(0, 0));
16     while(pq.size()){
17         con cur = pq.top();
18         pq.pop();
19         if(cur.first != dis[cur.second])
20             continue;
21         for(auto it: Graph[cur.second]){
22             if(cur.first + it.first < dis[it
23                 .second]){
24                 dis[it.second] = cur.first +
25                     it.first;
26                 pq.emplace(dis[it.second],
27                     it.second);
28             }
29         }
30     }
31     return 0;
32 }

```

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::
8     floor
9 // #include "Priority_Queue_BinaryHeap.h"
10 const int Max_Distance = 100;
11 class Graph_SP{ // SP serves as
12     Shortest Path
13 private:
14     int num_vertex;
15     std::vector<std::list<std::pair<int,int>
16         >>> AdjList;
17     std::vector<int> predecessor, distance;
18     std::vector<bool> visited;
19 public:
20     Graph_SP():num_vertex(0){};
21     Graph_SP(int n):num_vertex(n){
22         AdjList.resize(num_vertex);
23     }
24     void AddEdge(int from, int to, int
25         weight);

```

```

23 void PrintDataArray(std::vector<int>
24     array);
25 void PrintIntArray(int *array);
26
27 void InitializeSingleSource(int Start);
28     // 以Start作為起點
29 void Relax(int X, int Y, int weight);
30     // edge方向: from X to Y
31
32 void Dijkstra(int Start = 0); //
33     需要Min-Priority Queue
34 friend class BinaryHeap; //
35     以Binary Heap實現Min-Priority Queue
36 };
37 void Graph_SP::Dijkstra(int Start){
38
39     InitializeSingleSource(Start);
40
41     BinaryHeap minQueue(num_vertex); //
42     object of min queue
43     minQueue.BuildMinHeap(distance);
44
45     visited.resize(num_vertex, false); //
46     initialize visited[] as
47     {0,0,0,...,0}
48
49     while (!minQueue.IsHeapEmpty()) {
50         int u = minQueue.ExtractMin();
51         for (std::list<std::pair<int, int>
52             >>>:iterator itr = AdjList[u].
53             begin();
54             itr != AdjList[u].end(); itr++)
55             {
56                 Relax(u, (*itr).first, (*itr).
57                     second);
58                 minQueue.DecreaseKey((*itr).
59                     first, distance[(*itr).first
60                     ]); // definition at
61                     alrightchiu's priority queue
62                     self-version
63             }
64     }
65     std::cout << "\nprint predecessor:\n";
66     PrintDataArray(predecessor);
67     std::cout << "\nprint distance:\n";
68     PrintDataArray(distance);
69 }
70 void Graph_SP::InitializeSingleSource(int
71     Start){
72
73     distance.resize(num_vertex);
74     predecessor.resize(num_vertex);
75
76     for (int i = 0; i < num_vertex; i++) {
77         distance[i] = Max_Distance;
78         predecessor[i] = -1;
79     }
80     distance[Start] = 0;
81 }
82 void Graph_SP::Relax(int from, int to, int
83     weight){
84
85     if (distance[to] > distance[from] +
86         weight) {

```

```

69     distance[to] = distance[from] +
70         weight;
71     predecessor[to] = from;
72 }
73 void Graph_SP::AddEdge(int from, int to, int
74     weight){
75     AdjList[from].push_back(std::make_pair(
76         to,weight));
77 }
78 void Graph_SP::PrintDataArray(std::vector<
79     int> array){
80     for (int i = 0; i < num_vertex; i++)
81         std::cout << std::setw(4) << i;
82     std::cout << std::endl;
83     for (int i = 0; i < num_vertex; i++)
84         std::cout << std::setw(4) << array[i
85             ];
86     std::cout << std::endl;
87 }
88 int main(){
89
90     Graph_SP g9(6);
91     g9.AddEdge(0, 1, 8);g9.AddEdge(0, 5, 1);
92     g9.AddEdge(1, 0, 3);g9.AddEdge(1, 2, 1);
93     g9.AddEdge(2, 0, 5);g9.AddEdge(2, 3, 2);
94     g9.AddEdge(2, 4, 2);
95     g9.AddEdge(3, 1, 4);g9.AddEdge(3, 2, 6);
96     g9.AddEdge(3, 4, 7);g9.AddEdge(3, 5,
97         3);
98     g9.AddEdge(5, 3, 2);g9.AddEdge(5, 4, 8);
99
100     g9.Dijkstra(0);
101
102     return 0;
103 }

```

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
10     Shortest Path
11 private:
12     int num_vertex;
13     std::vector<std::list<std::pair<int,int>
14         >>> AdjList;
15     std::vector<int> predecessor, distance;
16 public:
17     Graph_SP():num_vertex(0){};
18     Graph_SP(int n):num_vertex(n){
19         AdjList.resize(num_vertex);
20     }

```

```

19 void AddEdge(int from, int to, int
20     weight);
21 void PrintDataArray(std::vector<int>
22     array);
23 void InitializeSingleSource(int Start);
24     // 以Start作為起點
25 void Relax(int X, int Y, int weight);
26     // 對edge(X,Y)進行Relax
27 bool BellmanFord(int Start = 0);
28     // 以Start作為起點
29
30 //
31
32 };
33 bool Graph_SP::BellmanFord(int Start){
34
35     InitializeSingleSource(Start);
36
37     for (int i = 0; i < num_vertex-1; i++) {
38         // |V-1|次的
39         iteration
40         // for each edge belonging to E(G)
41         for (int j = 0; j < num_vertex; j
42             ++){
43             // 把AdjList最
44             外層的vector走一遍
45             for (std::list<std::pair<int,int>
46                 >>>:iterator itr = AdjList[
47                 j].begin();
48                 itr != AdjList[j].end();
49                 itr++){
50                 // 各
51                 個vector中, 所有edge走
52                 一遍
53                 Relax(j, (*itr).first, (*itr
54                     ).second);
55             }
56         }
57     }
58
59     // check if there is negative cycle
60     for (int i = 0; i < num_vertex; i++) {
61         for (std::list<std::pair<int,int>
62             >>>:iterator itr = AdjList[i].
63             begin();
64             itr != AdjList[i].end(); itr++)
65             {
66                 if (distance[(*itr).first] >
67                     distance[i]+(*itr).second) {
68                     // i是from, *itr是to
69                     return false;
70                 }
71             }
72     }
73 }

```



```

47     }
48 }
49 }
50
51 // print predecessor[] & distance[]
52 std::cout << "predecessor[]:\n";
53 PrintDataArray(predecessor);
54 std::cout << "distance[]:\n";
55 PrintDataArray(distance);
56
57 return true;
58 }
59 void Graph_SP::PrintDataArray(std::vector<
    int> array){
60     for (int i = 0; i < num_vertex; i++)
61         std::cout << std::setw(4) << i;
62     std::cout << std::endl;
63     for (int i = 0; i < num_vertex; i++)
64         std::cout << std::setw(4) << array[i];
65     std::cout << std::endl << std::endl;
66 }
67 void Graph_SP::InitializeSingleSource(int
    Start){
68     distance.resize(num_vertex);
69     predecessor.resize(num_vertex);
70
71     for (int i = 0; i < num_vertex; i++) {
72         distance[i] = Max_Distance;
73         predecessor[i] = -1;
74     }
75     distance[Start] = 0;
76 }
77 void Graph_SP::Relax(int from, int to, int
    weight){
78     if (distance[to] > distance[from] +
        weight) {
79         distance[to] = distance[from] +
            weight;
80         predecessor[to] = from;
81     }
82 }
83 void Graph_SP::AddEdge(int from, int to, int
    weight){
84     AdjList[from].push_back(std::make_pair(
        to, weight));
85 }
86
87 }
88
89 int main(){
90     Graph_SP g7(6);
91     g7.AddEdge(0, 1, 5);
92     g7.AddEdge(1, 4, -4); g7.AddEdge(1, 2, 6)
93     ;
94     g7.AddEdge(2, 4, -3); g7.AddEdge(2, 5,
        -2);
95     g7.AddEdge(3, 2, 4);
96     g7.AddEdge(4, 3, 1); g7.AddEdge(4, 5, 6);
97     g7.AddEdge(5, 0, 3); g7.AddEdge(5, 1, 7);
98
99     if (g7.BellmanFord(0))
100         std::cout << "There is no negative
            cycle.\n";
101     else

```

```

102     std::cout << "There is negative
        cycle.\n";
103
104     return 0;
105 }

```

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, Predecessor;
13 public:
14     Graph_SP_AllPairs():num_vertex(0){};
15     Graph_SP_AllPairs(int n);
16     void AddEdge(int from, int to, int
        weight);
17     void PrintData(std::vector< std::vector<
        int> > array);
18     void InitializeData();
19     void FloydWarshall();
20 };
21
22 Graph_SP_AllPairs::Graph_SP_AllPairs(int n):
    num_vertex(n){
23     // Constructor, initialize AdjMatrix
        with 0 or MaxDistance
24     AdjMatrix.resize(num_vertex);
25     for (int i = 0; i < num_vertex; i++) {
26         AdjMatrix[i].resize(num_vertex,
            MaxDistance);
27         for (int j = 0; j < num_vertex; j++)
28             {
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,
                -1);
40             for (int j = 0; j < num_vertex; j++)
41                 {
42                     Distance[i][j] = AdjMatrix[i][j];
43                     if (Distance[i][j] != 0 &&
                        Distance[i][j] !=
                            MaxDistance) {

```

```

44         Predecessor[i][j] = i;
45     }
46 }
47
48 }
49 void Graph_SP_AllPairs::FloydWarshall(){
50     InitializeData();
51     PrintData(Distance);
52     std::cout << "\ninitial Predecessor[]:\n";
53     PrintData(Predecessor);
54
55     for (int k = 0; k < num_vertex; k++) {
56         std::cout << "\nincluding vertex("
            << k << "):\n";
57         for (int i = 0; i < num_vertex; i++)
58             {
59                 for (int j = 0; j < num_vertex;
                    j++) {
60                     if ((Distance[i][j] >
                        Distance[i][k]+Distance[
                            k][j])
                        && (Distance[i][k] !=
                            MaxDistance)) {
61                         Distance[i][j] =
                            Distance[i][k]+
                                Distance[k][j];
62                         Predecessor[i][j] =
                            Predecessor[k][j];
63                     }
64                 }
65             }
66         // print data after including new
            vertex and updating the shortest
            paths
67         std::cout << "Distance[]:\n";
68         PrintData(Distance);
69         std::cout << "\nPredecessor[]:\n";
70         PrintData(Predecessor);
71     }
72 }
73 void Graph_SP_AllPairs::PrintData(std::
    vector< std::vector<int> > array){
74     for (int i = 0; i < num_vertex; i++){
75         for (int j = 0; j < num_vertex; j++)
76             {
77                 std::cout << std::setw(5) <<
                    array[i][j];
78             }
79         std::cout << std::endl;
80     }
81 }
82 void Graph_SP_AllPairs::AddEdge(int from,
    int to, int weight){
83     AdjMatrix[from][to] = weight;
84 }
85
86 int main(){
87     Graph_SP_AllPairs g10(4);
88     g10.AddEdge(0, 1, 2); g10.AddEdge(0, 2,
        6); g10.AddEdge(0, 3, 8);

```

```

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

```

9.5 shortest-path_on_DAG

```

1 // C++ code
2 // 0(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);
25     // 以Start作為起點
26     void Relax(int X, int Y, int weight);
27     // 對edge(X,Y)進行Relax
28
29     void DAG_SP(int Start = 0);
30     // 需要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 &
                count);
34 };
35 void Graph_SP::GetTopologicalSort(int *array
    , int Start){

```

```

35 int color[num_vertex], discover[
    num_vertex], finish[num_vertex];
36
37 for (int i = 0; i < num_vertex; i++) {
38     color[i] = 0;
39     discover[i] = 0;
40     finish[i] = 0;
41     predecessor[i] = -1;
42 }
43
44 int time = 0,
45     count = num_vertex-1, //
46     count 為 topologicalsort[] 的
47     index
48     i = Start;
49
50 for (int j = 0; j < num_vertex; j++) {
51     if (color[i] == 0) {
52         DFSVisit_TS(array, color,
53             discover, finish, i, time,
54             count);
55     }
56     i = j;
57 }
58 std::cout << "\nprint discover time:\n";
59 PrintIntArray(discover);
60 std::cout << "\nprint finish time:\n";
61 PrintIntArray(finish);
62
63 void Graph_SP::DFSVisit_TS(int *array, int *
64     color, int *discover,
65     int *finish, int
66     vertex, int &
67     time, int &
68     count){
69
70     color[vertex] = 1; // set gray
71     discover[vertex] = ++time;
72     for (std::list<std::pair<int,int>>::
73         iterator itr = AdjList[vertex].begin
74         ();
75         itr != AdjList[vertex].end(); itr
76         ++){
77         if (color[(itr).first] == 0) {
78             predecessor[(itr).first] =
79                 vertex;
80             DFSVisit_TS(array, color,
81                 discover, finish, (*itr).
82                 first, time, count);
83         }
84     }
85     color[vertex] = 2; // set black
86     finish[vertex] = ++time;
87     array[count--] = vertex; //
88     產生Topological Sort
89 }
90
91 void Graph_SP::DAG_SP(int Start){
92
93     InitializeSingleSource(Start); //
94     distance[],predecessor[]的
95     initialization
96
97     int topologicalsort[num_vertex];
98     GetTopologicalSort(topologicalsort,
99     Start);

```

```

81 for (int i = 0; i < num_vertex; i++) {
82     int v = topologicalsort[i];
83     for (std::list<std::pair<int, int
84         >>::iterator itr = AdjList[v].
85         begin();
86         itr != AdjList[v].end(); itr++)
87         {
88             Relax(v, (*itr).first, (*itr).
89                 second);
90         }
91     }
92     std::cout << "\nprint predecessor:\n";
93     PrintDataArray(predecessor);
94     std::cout << "\nprint distance:\n";
95     PrintDataArray(distance);
96 }
97
98 void Graph_SP::PrintDataArray(std::vector<
99     int> array){
100     for (int i = 0; i < num_vertex; i++)
101         std::cout << std::setw(4) << i;
102     std::cout << std::endl;
103     for (int i = 0; i < num_vertex; i++)
104         std::cout << std::setw(4) << array[i]
105         ];
106     std::cout << std::endl;
107 }
108
109 void Graph_SP::PrintIntArray(int *array){
110     for (int i = 0; i < num_vertex; i++)
111         std::cout << std::setw(4) << i;
112     std::cout << std::endl;
113     for (int i = 0; i < num_vertex; i++)
114         std::cout << std::setw(4) << array[i]
115         ];
116     std::cout << std::endl;
117 }
118
119 void Graph_SP::InitializeSingleSource(int
120     Start){
121     distance.resize(num_vertex);
122     predecessor.resize(num_vertex);
123
124     for (int i = 0; i < num_vertex; i++) {
125         distance[i] = Max_Distance;
126         predecessor[i] = -1;
127     }
128     distance[Start] = 0;
129 }
130
131 void Graph_SP::Relax(int from, int to, int
132     weight){
133     if (distance[to] > distance[from] +
134         weight) {
135         distance[to] = distance[from] +
136             weight;
137         predecessor[to] = from;
138     }
139 }
140
141 void Graph_SP::AddEdge(int from, int to, int
142     weight){
143     AdjList[from].push_back(std::make_pair(
144         to,weight));
145 }
146
147 int main(){

```

```

134 Graph_SP g8(7);
135 g8.AddEdge(0, 1, 3);g8.AddEdge(0, 2, -2)
136 ;
137 g8.AddEdge(1, 3, -4);g8.AddEdge(1, 4, 4)
138 ;
139 g8.AddEdge(2, 4, 5);g8.AddEdge(2, 5, 6);
140 g8.AddEdge(3, 5, 8);g8.AddEdge(3, 6, 2);
141 g8.AddEdge(4, 3, -3);g8.AddEdge(4, 6,
142     -2);
143 g8.AddEdge(5, 6, 2);
144
145 g8.DAG_SP(0); // 以vertex(0)作為
146     起點
147
148 return 0;
149 }

```

10 graph/Tree

10.1 Lowest_Common_Ancesor

```

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
19         [v] = D[u]
20
21     for (int i = 0; i <= lgN; ++i) // 調整他
22         們到二進位數一樣
23         if (s & (1 << i))
24             v = P[v][i];
25     if (u == v)
26         return v;
27     // because they are at same depth
28     // jump up if they are different
29     // think about that if P[u][i] == P[v][i]
30     ]
31     // then that point must be the ancestor
32     of LCA or LCA itself
33     // by this, we will stop at LCA's child
34     for (int i = lgN; i >= 0; --i)
35         //
36         if (P[u][i] != P[v][i])
37         {
38             u = P[u][i];
39             v = P[v][i];

```

```

36     }
37     return P[u][0];
38 }
39
40 void ComputeP()
41 {
42     int n = N;
43     for (int i = 0; i < lgN; ++i) // to lgN
44         enough
45     {
46         for (int x = 0; x < n; ++x)
47         {
48             if (P[x][i] == -1)
49                 P[x][i + 1] = -1;
50             else
51                 P[x][i + 1] = P[P[x][i]][i];
52             // equal to move on the
53             parent direction
54             // And P[x][i] move 2 ^ n
55             steps to a parent we
56             call it y
57             // P[y][i] means continue
58             move 2 ^ n step from y
59             to a parent we call z
60             // so the total equal to
61             move 2 ^ n * 2 ^ n steps
62             from x to z
63             // which is move 2 ^ (n + 1)
64             steps to z
65         }
66     }
67 }

```

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 ??
9 pair<int, int> Tree_Centroid(int v, int pa)
10 {
11     // return (最大子樹節點數, 節點
12     ID)
13     subTsize[v] = 1;
14     pair<int, int> res(INT_MAX, -1); // ans:
15     tree cnetroid
16     int max_subT = 0; // 最大子樹節點數
17     for (size_t i = 0; i < adj[v].size(); ++
18         i)
19     {
20         int x = adj[v][i];
21         if (x == pa)
22             continue;
23         res = min(res, Tree_Centroid(x, v));
24         subTsize[v] += subTsize[x];
25         max_subT = max(max_subT, subTsize[x]
26             ]);

```

```

24 }
25 res = min(res, make_pair(max(max_subT, n
    - subTsize[v]), v)); // (n -
    subTsize[v]) for maybe parent tree
    is the biggest
26 // min because all res will be greater
    than n/2;
27 // the min one is the tree centroid
    return res;
28 }
29 // Tree_Centroid2
30 vector<int> V[10005];
31 int N;
32 int center, csize;
33 int dfs(int v, int fa)
34 {
35     int sz = 1;
36     int maxsub = 0;
37     for(int u:V[v])
38     {
39         if (u==fa)continue;
40         int sub = dfs(u, v);
41         maxsub = max(maxsub, sub);
42         sz += sub;
43     }
44     maxsub = max(maxsub, N-sz);
45
46     if (maxsub<csize)
47     {
48         center = v;
49         csize = maxsub;
50     }
51     return sz;
52 }
53 }

```

11 hashing

11.1 hashingVec

```

1 #include<bits/stdc++.h>
2 struct VectorHash {
3     size_t operator()(const std::vector<int>
4     &v) const {
5         std::hash<int> hasher;
6         size_t seed = 0;
7         for (const int& i : v) {
8             seed ^= hasher(i) + 0x9e3779b9 +
9             (seed<<6) + (seed>>2);
10        }
11        return seed;
12    }
13 };
14 std::unordered_set<std::vector<int>,
15     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
4 // ^2= (-1)^n
5 // The "addition" rule : F_{n+k} = F_kF_{n-1}
6 // +1)+F_{k-1}F_n
7 // k = n, F_{2n} = F_n*(F_{n+1} + F_{n-1})
8 // F_{2k} = F_k*(2F_{k+1}-F_k)
9 // F_{2k+1} = F_{k+1}^2+F_k^2
10 pair<int, int> fib (int n) {
11     if (n == 0)
12         return {0, 1};
13     auto p = fib(n >> 1);
14     int c = p.first * (2 * p.second - p.
15     first);
16     int d = p.first * p.first + p.second * p
17     .second;
18     if (n & 1)
19         return {d, c + d};
20     else
21         return {c, d};
22 }

```

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 = 1000000000;
11     static const int WIDTH = 8;
12     vector<int> s;
13
14     BigInteger(long long num = 0) { *this =
15     num; }
16     BigInteger operator = (long long num)
17     {
18         s.clear();
19         do{
20             s.push_back(num % BASE);
21             num /= BASE;
22         } while (num > 0);
23         return *this;
24     }
25     BigInteger operator = (const string& str
26     ){
27         s.clear();
28         int x, len = (str.length() - 1) /
29         WIDTH + 1;
30         for (int i = 0; i < len;i++){

```

```

31         int end = str.length() - i *
32         WIDTH;
33         int start = max(0, end - WIDTH);
34         sscanf(str.substr(start, end -
35         start).c_str(), "%d", &x);
36         s.push_back(x);
37     }
38     return *this;
39 }
40
41 BigInteger operator+ (const BigInteger b
42 ) const{
43     BigInteger c;
44     c.s.clear();
45     for(int i=0,g=0;i++){
46         if(g== 0 && i >=s.size() && i >=
47         b.s.size())
48             break;
49         int x = g;
50         if(i<s.size()) x+=s[i];
51         if(i<b.s.size()) x+=b.s[i];
52         c.s.push_back(x % BASE);
53         g = x / BASE;
54     }
55     return c;
56 }
57 BigInteger operator+=(const BigInteger&
58 b){
59     *this = *this + b;
60     return *this;
61 }
62
63 BigInteger operator* (const BigInteger b
64 )const{
65     BigInteger c;
66     c.s.clear();
67     long long mul;
68     for (int i = 0;i < s.size(); i++)
69     {
70         long long carry = 0;
71         for (int g = 0; g < b.s.size();g
72         ++){
73             mul = (long long)(s[i]) * (
74             long long)(b.s[g]);
75             mul += carry;
76             if(i + g < c.s.size()){
77                 c.s[i+g] += mul % BASE;
78             }else{
79                 c.s.push_back(mul % BASE
80                 );
81             }
82             carry = mul / BASE;
83         }
84     }
85     c.s[i] += carry;
86     return c;
87 }
88
89 for (int i = 0; i < c.s.size(); i++)
90 {
91     if(c.s[i] >= BASE){
92         if(i + 1 < c.s.size()){
93             c.s.push_back(c.s[i] /
94             BASE);
95         }else{
96             c.s[i + 1] += c.s[i] /
97             BASE;
98         }
99         c.s[i] %= BASE;
100     }
101 }

```

```

82     return c;
83 }
84 bool operator< (const BigInteger& b)
85 const{
86     if(s.size() != b.s.size()) return s.
87     size() < b.s.size();
88     for(int i=s.size() - 1; i>=0;i--){
89         if(s[i] != b.s[i]) return s[i] <
90         b.s[i];
91     }
92     return false; // Equal
93 }
94 bool operator> (const BigInteger& b)
95 const{return b < *this;}
96 bool operator<= (const BigInteger& b)
97 const {return !(b<*this);}
98 bool operator>=(const BigInteger& b)
99 const {return !(*this < b);}
100 bool operator!=(const BigInteger& b)
101 const {return b< *this || *this < b
102 ;}
103 bool operator==(const BigInteger& b)
104 const {return !(b<*this) && !(*this<
105 b);}
106 };
107 ostream& operator<< (ostream &out, const
108 BigInteger& x){
109     out << x.s.back();
110     for (int i = x.s.size() - 2;i >= 0;i--){
111         char buf[20];
112         sprintf(buf,"%08d",x.s[i]);
113         for(int j = 0;j<strlen(buf);j++) out
114         << buf[j];
115     }
116     return out;
117 }
118 istream& operator>> (istream &in, BigInteger
119 & x){
120     string s;
121     if(!(in >> s)) return in;
122     x = s;
123     return in;
124 }

```

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,
18     int &x0, int &y0, int &g) {

```

```

18 g = gcd(abs(a), abs(b), x0, y0);
19 if (c % g) {
20     return false;
21 }
22
23 x0 *= c / g;
24 y0 *= c / g;
25 if (a < 0) x0 = -x0;
26 if (b < 0) y0 = -y0;
27 return true;
28 }
29
30 // finding all solution
31 void shift_solution(int & x, int & y, int a,
32     int b, int cnt) {
33     x += cnt * b;
34     y -= cnt * a;
35 }
36
37 int find_all_solutions(int a, int b, int c,
38     int minx, int maxx, int miny, int maxy)
39 {
40     int x, y, g;
41     if (!find_any_solution(a, b, c, x, y, g))
42         return 0;
43     a /= g;
44     b /= g;
45
46     int sign_a = a > 0 ? +1 : -1;
47     int sign_b = b > 0 ? +1 : -1;
48
49     shift_solution(x, y, a, b, (minx - x) /
50         b);
51     if (x < minx)
52         shift_solution(x, y, a, b, sign_b);
53     if (x > maxx)
54         return 0;
55     int lx1 = x;
56
57     shift_solution(x, y, a, b, (maxx - x) /
58         b);
59     if (x > maxx)
60         shift_solution(x, y, a, b, -sign_b);
61     int rx1 = x;
62
63     shift_solution(x, y, a, b, -(miny - y) /
64         a);
65     if (y < miny)
66         shift_solution(x, y, a, b, -sign_a);
67     if (y > maxy)
68         return 0;
69     int lx2 = x;
70
71     shift_solution(x, y, a, b, -(maxy - y) /
72         a);
73     if (y > maxy)
74         shift_solution(x, y, a, b, sign_a);
75     int rx2 = x;
76
77     if (lx2 > rx2)
78         swap(lx2, rx2);
79     int lx = max(lx1, lx2);
80     int rx = min(rx1, rx2);
81
82     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], tcac[MAXN];
4
5 #define P 998244353
6 #define REP1(i, n) for (int i = 1; i <= (int)
7     )(n); ++i)
8 #define REP(i, n) for (int i = (int)(n) - 1;
9     i >= 0; --i)
10 void init(int n){
11     fact[0] = 1;
12     for (int i = 1; i <= n; i++)
13         fact[i] = i * fact[i - 1] % P;
14     for (int i = n; i >= 0; --i)
15         tcac[i] = deg(fact[i], -1);
16 }
17
18 i64 deg(i64 x, i64 d) {
19     if (d < 0) d += P - 1;
20     i64 y = 1;
21     while (d) {
22         if (d & 1) (y *= x) %= P;
23         d /= 2;
24         (x *= x) %= P;
25     }
26     return y;
27 }
28
29 i64 cnk(int n, int k) {
30     if (k < 0 || k > n) return 0;
31     return fact[n] * tcac[k] % P * tcac[n -
32         k] % P;
33 }

```

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,
21     const string &P, const string &T)
22 {
23     vector<int> res; // start from these
24     points
25     const int n = P.size();
26     for (int j = 0, i = -1; j < T.size(); ++
27         j) {
28         while (~i && T[j] != P[i + 1])
29             i = fail[i];
30         if (P[i + 1] == T[j])
31             ++i;
32         if (i == n - 1)
33             res.push_back(j - n + 1), i =
34                 fail[i];
35     }
36     return res;
37 }
38
39 int main(){
40     char control;
41     string test_pattern = "a";
42     string test_text = "abcdabcbcdabceabcd";
43     // for testing
44     cout << "Do you want to Enter by ys?(y/n
45         )";
46     cin >> control;
47     if(control == 'y' || control == 'Y'){
48         cout << "Enter text:";
49         cin >> test_text;
50         cout << "Enter pattern:";
51         cin >> test_pattern;
52     }
53
54     vector<int> V = build_kmp(test_pattern);
55     vector<int> Ans = kmp_match(V,
56         test_pattern, test_text);
57     cout << '\n';
58     for(auto it = V.begin(); it != V.end();
59         ++it)
60         cout << *it << ' ';
61     cout << '\n';
62     for(auto it = Ans.begin(); it != Ans.end
63         (); ++it)
64         cout << *it << ' ';
65     return 0;
66 }

```

ACM ICPC TEAM REFERENCE - DURACELL!

Contents

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