1 data_structure

1.1 Sparse_Table

```cpp
#include <bits/stdc++.h>
using namespace std;
int n;
int v[1000009];
int sparse[22][1000009];
// O(nlogn) preprocess O(1)Query
// sp[x][y] is the answer from (v[x], v[x+2^y-1])
inline void init()
{
    for (int i = 0; i < n; ++i)
        sparse[0][i] = v[i];
    for (int j = 1; (1 << j) <= n; ++j)
        for (int i = 0; i + (1 << j) <= n; ++i)
        sparse[j][i] = min(
        sparse[j - 1][i],
        sparse[j - 1][i + (1 << (j - 1))]
        );
}

inline int query(int l, int r)
{
    int k = __lg(r - l + 1);
    return min(sparse[k][l], sparse[k][r - (1 << k) + 1]);
}
```

1.2 segment_Tree

```cpp
#define LL long long
#define IL(X) ((X << 1) + 1)
#define IR(X) ((X << 1) + 2)
#define MAXN 500005
// add tag
// tag += tag
// val += tag*size

struct segID{
    struct Node{
        LL val;
        LL lazy_tag;
        int size;
    };
    LL dataseq[MAXN];
    Node seq[MAXN * 4 + 5];
    void pull(int index){
        seq[index].val = seq[IL(index)].val + seq[IR(index)].
            val;
    }
    void push(int index){
        seq[IL(index)].lazy_tag += seq[index].lazy_tag;
        seq[IL(index)].val += seq[index].lazy_tag * seq[IL(
            index)].size;
        seq[IR(index)].lazy_tag += seq[index].lazy_tag;
        seq[IR(index)].val += seq[index].lazy_tag * seq[IR(
            index)].size;
        seq[index].lazy_tag = 0;
    }

    void build(int L, int R, int index){
        if(L == R){
            seq[index].val = dataseq[L];
            seq[index].size = 1;
            seq[index].lazy_tag = 0;
            return;
        }
        int M = (L + R) / 2;
        build(L, M, IL(index));
        build(M + 1, R, IR(index));
        seq[index].size = seq[IL(index)].size + seq[IL(index)
            ].size;
        pull(index);
    }

    void modify(int l, int r, int L, int R, int index, long
         long Add){
        if(l == L && r == R){
            seq[index].lazy_tag += Add;
            seq[index].val += Add * seq[index].size;
            return;
        }
        push(index);
        int M = (L + R) / 2;

        if(r <= M){
            modify(l, r, L, M, IL(index), Add);
        }else if(l > M){
            modify(l, r, M + 1, R, IR(index), Add);
        }else{
            modify(l, M, L, M, IL(index), Add);
            modify(M + 1, r, M + 1, R, IR(index), Add);
        }
        pull(index);
    }

    long long Query(int l, int r, int L, int R, int index){
        if(l == L && r == R){
            return seq[index].val;
        }
        int M = (L + R) / 2;
        push(index);
        if(r <= M){
            return Query(l, r, L, M, IL(index));
        }else if(l > M){
            return Query(l, r, M + 1, R, IR(index));
        }else{
            return Query(l, M, L, M, IL(index)) +
            Query(M + 1, r, M + 1, R, IR(index));
        }

    }
};
```

1.3 disjointset

```cpp
#include <algorithm>
using namespace std;
#define MAX_N 200005
struct  disjointset
{
    int rank[MAX_N];
    int f[MAX_N];
    void init(int N){
        for (int i = 0; i < N;i++){
            f[i] = i;
            rank[i] = 1;
        }

    }
    int find(int v){
        if( f[v] == v)
            return v;
        return f[v] = find(f[v]);
    }
    bool same(int a, int b){
        return find(a) == find(b);
    }
    void Union(int a, int b){
        // f[find(a)] = find(b);
        if(!same(a,b)){
            if(rank[a] < rank[b])
                swap(a, b);
            f[f[b]] = f[a];
            rank[a]++;
        }
    }
};
```

2 geometry

2.1 closest_point

```cpp
template <typename T>
T ClosestPairSquareDistance(typename vector<Point<T>>::
    iterator l,
                            typename vector<Point<T>>::
                                iterator r)
{
    auto delta = numeric_limits<T>::max();
    if (r - l > 1)
    {
        auto m = l + (r - l >> 1);
        nth_element(l, m, r); // Lexicographical order in
            default
        auto x = m->x;
        delta = min(ClosestPairSquareDistance<T>(l, m),
                    ClosestPairSquareDistance<T>(m, r));
        auto square = [&](T y) { return y * y; };
        auto sgn = [=](T a, T b) {
            return square(a - b) <= delta ? 0 : a < b ? -1 :
                1;
        };
        vector<Point<T>> x_near[2];
        copy_if(l, m, back_inserter(x_near[0]), [=](Point<T>
            a) {
            return sgn(a.x, x) == 0;
        });
        copy_if(m, r, back_inserter(x_near[1]), [=](Point<T>
            a) {
            return sgn(a.x, x) == 0;
        });
        for (int i = 0, j = 0; i < x_near[0].size(); ++i)
```

```
        {
            while (j < x_near[1].size() and
                   sgn(x_near[1][j].y, x_near[0][i].y) == -1)
                ++j;
            for (int k = j; k < x_near[1].size() and
                            sgn(x_near[1][k].y, x_near[0][i].
                                y) == 0;
                 ++k)
            {
                delta = min(delta, (x_near[0][i] - x_near[1][
                    k]).norm());
            }
        }
        inplace_merge(l, m, r, [](Point<T> a, Point<T> b) {
            return a.y < b.y;
        });
    }
    return delta;
}
```

2.2 points

```
template <typename T>
struct Point
{
    T x, y;
    Point() : x(0), y(0) {}
    Point(const T x, const T y) : x(x), y(y) {}
    template <class F>
    explicit operator Point<F>() const
    {
        return Point<F>((F)x, (F)y);
    }

    Point operator+(const Point b) const
    {
        return Point(x + b.x, y + b.y);
    }
    Point operator-(const Point b) const
    {
        return Point(x - b.x, y - b.y);
    }
    template <class F>
    Point<F> operator*(const F fac)
    {
        return Point<F>(x * fac, y * fac);
    }
    template <class F>
    Point<F> operator/(const F fac)
    {
        return Point<F>(x / fac, y / fac);
    }

    T operator&(const Point b) const { return x * b.x + y * b
        .y; }
    // 內 積 運 算 子
    T operator^(const Point b) const { return x * b.y - y * b
        .x; }
    // 外 積 運 算 子
    bool operator==(const Point b) const
    {
        return x == b.x and y == b.y;
    }
    bool operator<(const Point b) const
    {
        return x == b.x ? y < b.y : x < b.x;
    } // 字 典 序

    Point operator-() const { return Point(-x, -y); }
    T norm() const { return *this & *this; }    // 歐 式 長
        度 平 方
    Point prep() const { return Point(-y, x); } // 左 旋 直
        角 法 向 量

    template <class F>
    istream &operator>>(istream &is, Point<F> &pt)
    {
        return is >> pt.x >> pt.y;
    }
    template <class F>
    ostream &operator<<(ostream &os, const Point<F> &pt)
    {
        return os << pt.x << " " << pt.y;
    }
};
```

2.3 lines

```
template <typename T, typename Real = double>
struct Line
{
    Point<T> st, ed;
    Point<T> vec() const { return ed - st; }
    T ori(const Point<T> p) const { return (ed - st) ^ (p -
        st); }
    Line(const Point<T> x, const Point<T> y) : st(x), ed(y)
        {}
    template <class F>
    operator Line<F>() const
    {
        return Line<F>((Point<F>)st, (Point<F>)ed);
    }

    // sort by arg, the left is smaller for parallel lines
    bool operator<(Line B) const
    {
        Point<T> a = vec(), b = B.vec();
        auto sgn = [](const Point<T> t) { return (t.y == 0 ?
            t.x : t.y) < 0; };
        if (sgn(a) != sgn(b))
            return sgn(a) < sgn(b);
        if (abs(a ^ b) == 0)
            return B.ori(st) > 0;
        return (a ^ b) > 0;
    }

    // Regard a line as a function
    template <typename F>
    Point<F> operator()(const F x) const
    {
        return Point<F>(st) + vec() * x;
    }

    bool isSegProperIntersection(const Line l) const
    {
        return l.ori(st) * l.ori(ed) < 0 and ori(l.st) * ori(
            l.ed) < 0;
    }

    bool isPointOnSegProperly(const Point<T> p) const
    {
        return ori(p) == 0 and ((st - p) & (ed - p)) < 0;
    }

    Point<Real> getIntersection(const Line<Real> l)
    {
        Line<Real> h = *this;
        return l((l.st ^ h.vec()) / (h.vec() ^ l.vec()));
    }

    Point<Real> projection(const Point<T> p) const
    {
        return operator()(((p - st) & vec()) / (Real)(vec().
            norm()));
    }
};
```

2.4 geometry_template

```
// import from https://codeforces.com/blog/entry/48122
#include <iostream>
using namespace std;
template <class F>
struct Point {
  F x, y;
  Point() : x(0), y(0) {}
  Point(const F& x, const F& y) : x(x), y(y) {}

  void swap(Point& other) { using std::swap; swap(x, other.x)
      ; swap(y, other.y); }
  template <class F1> explicit operator Point<F1> () const {
    return Point<F1>(static_cast<F1>(x), static_cast<F1>(y));
      }
  template <class F1> Point& operator = (const Point<F1>&
      other) {
    x = other.x; y = other.y; return *this; }
  template <class F1> Point& operator += (const Point<F1>&
      other) {
    x += other.x; y += other.y; return *this; }
  template <class F1> Point& operator -= (const Point<F1>&
      other) {
    x -= other.x; y -= other.y; return *this; }
  template <class F1> Point& operator *= (const F1& factor) {
    x *= factor; y *= factor; return *this; }
  template <class F1> Point& operator /= (const F1& factor) {
    x /= factor; y /= factor; return *this; }
};

template <class F> int read(Point<F>& point) { return read(
    point.x, point.y) / 2; }
template <class F> int write(const Point<F>& point) { return
    write(point.x, point.y); }

template <class F> istream& operator >> (istream& is, Point<F
    >& point) {
  return is >> point.x >> point.y; }
```

```
template <class F> ostream& operator << (ostream& os, const Point<F>& point) {
  return os << point.x << ' ' << point.y; }

template <class F> inline Point<F> makePoint(const F& x, const F& y) { return Point<F>(x, y); }
template <class F> void swap(Point<F>& lhs, Point<F>& rhs) { lhs.swap(rhs); }

#define FUNC1(name, arg, expr) \
template <class F> inline auto name(const arg) -> decltype(expr) { return expr; }
#define FUNC2(name, arg1, arg2, expr) \
template <class F1, class F2> \
inline auto name(const arg1, const arg2) -> decltype(expr) { return expr; }
#define FUNC3(name, arg1, arg2, arg3, expr) \
template <class F1, class F2, class F3> \
inline auto name(const arg1, const arg2, const arg3) -> decltype(expr) { return expr; }

FUNC1(operator -, Point<F>& point, makePoint(-point.x, -point.y))
FUNC2(operator +, Point<F1>& lhs, Point<F2>& rhs, makePoint(lhs.x + rhs.x, lhs.y + rhs.y))
FUNC2(operator -, Point<F1>& lhs, Point<F2>& rhs, makePoint(lhs.x - rhs.x, lhs.y - rhs.y))
FUNC2(operator *, F1& factor, Point<F2>& rhs, makePoint(factor * rhs.x, factor * rhs.y))
FUNC2(operator *, Point<F1>& lhs, F2& factor, makePoint(lhs.x * factor, lhs.y * factor))
FUNC2(operator /, Point<F1>& lhs, F2& factor, makePoint(lhs.x / factor, lhs.y / factor))

FUNC2(operator *, Point<F1>& lhs, Point<F2>& rhs, lhs.x * rhs.x + lhs.y * rhs.y)
FUNC2(operator ^, Point<F1>& lhs, Point<F2>& rhs, lhs.x * rhs.y - lhs.y * rhs.x)

// < 0 if rhs <- lhs counter-clockwise, 0 if collinear, > 0 if clockwise.
FUNC2(ccw, Point<F1>& lhs, Point<F2>& rhs, rhs ^ lhs)
FUNC3(ccw, Point<F1>& lhs, Point<F2>& rhs, Point<F3>& origin, ccw(lhs - origin, rhs - origin))

FUNC2(operator ==, Point<F1>& lhs, Point<F2>& rhs, lhs.x == rhs.x && lhs.y == rhs.y)
FUNC2(operator !=, Point<F1>& lhs, Point<F2>& rhs, !(lhs == rhs))

FUNC2(operator <, Point<F1>& lhs, Point<F2>& rhs,
    lhs.y < rhs.y || (lhs.y == rhs.y && lhs.x < rhs.x))
FUNC2(operator >, Point<F1>& lhs, Point<F2>& rhs, rhs < lhs)
FUNC2(operator <=, Point<F1>& lhs, Point<F2>& rhs, !(lhs > rhs))
FUNC2(operator >=, Point<F1>& lhs, Point<F2>& rhs, !(lhs < rhs))

// Angles and rotations (counter-clockwise).
FUNC1(angle, Point<F>& point, atan2(point.y, point.x))
FUNC2(angle, Point<F1>& lhs, Point<F2>& rhs, atan2(lhs ^ rhs, lhs * rhs))
FUNC3(angle, Point<F1>& lhs, Point<F2>& rhs, Point<F3>& origin,
    angle(lhs - origin, rhs - origin))
FUNC3(rotate, Point<F1>& point, F2& angleSin, F3& angleCos,
    makePoint(angleCos * point.x - angleSin * point.y,
              angleSin * point.x + angleCos * point.y))
FUNC2(rotate, Point<F1>& point, F2& angle, rotate(point, sin(angle), cos(angle)))
FUNC3(rotate, Point<F1>& point, F2& angle, Point<F3>& origin,
    origin + rotate(point - origin, angle))
FUNC1(perp, Point<F>& point, makePoint(-point.y, point.x))

// Distances.
FUNC1(abs, Point<F>& point, point * point)
FUNC1(norm, Point<F>& point, sqrt(abs(point)))
FUNC2(dist, Point<F1>& lhs, Point<F2>& rhs, norm(lhs - rhs))
FUNC2(dist2, Point<F1>& lhs, Point<F2>& rhs, abs(lhs - rhs))
FUNC2(bisector, Point<F1>& lhs, Point<F2>& rhs, lhs * norm(rhs) + rhs * norm(lhs))

#undef FUNC1
#undef FUNC2
#undef FUNC3
```

2.5 cp_geometry.

```
#include <bits/stdc++.h>
using namespace std;
const double eps = 1e-6;
inline int fcmp(const double &a, const double &b){
    if(fabs(a - b) < eps)
        return 0;
    return ((a - b > 0.0) * 2) - 1;
}

template <typename T>
struct Point
{
    T x, y;
    Point() : x(0), y(0) {}
    Point(const T x, const T y) : x(x), y(y) {}
    template <class F>
    explicit operator Point<F>() const
    {
        return Point<F>((F)x, (F)y);
    }

    Point operator+(const Point b) const
    {
        return Point(x + b.x, y + b.y);
    }
    Point operator-(const Point b) const
    {
        return Point(x - b.x, y - b.y);
    }
    template <class F>
    Point<F> operator*(const F fac)
    {
        return Point<F>(x * fac, y * fac);
    }
    template <class F>
    Point<F> operator/(const F fac)
    {
        return Point<F>(x / fac, y / fac);
    }
    T operator&(const Point b) const { return x * b.x + y * b.y; }
    // dot operator
    T operator^(const Point b) const { return x * b.y - y * b.x; }
    // cross operator
    bool operator==(const Point b) const
    {
        return x == b.x and y == b.y;
    }
    bool operator<(const Point b) const
    {
        return x == b.x ? y < b.y : x < b.x;
    } // 字 典 序

    Point operator-() const { return Point(-x, -y); }
    T norm() const { return *this & *this; }    // 歐 式 長 度 平 方
    Point prep() const { return Point(-y, x); } // 左 旋 直 角 法 向 量
    template <class F>
    friend istream &operator>>(istream &is, Point<F> &pt);
    template<class F>
    friend ostream &operator<<(ostream &os, const Point<F> &pt);
};
template <class F>
ostream &operator<<(ostream &os, const Point<F> &pt)
{
    return os << "(" << pt.x << " " << pt.y << ")";
}
template <class F>
istream &operator>>(istream &is, Point<F> &pt)
{
    return is >> pt.x >> pt.y;
}
template <class F>
bool collinearity(const Point<F>& p1, const Point<F>& p2, const Point<F>& p3){
//    return (p1 - p3) ^ (p2 - p3) == 0;
    return fcmp((p1 - p3) ^ (p2 - p3), 0.0) == 0;
}
// check co-line first. properly
template<class F>
inline bool btw(const Point<F>& p1, const Point<F>& p2, const Point<F>& p3){
    return fcmp((p1 - p3) & (p2 - p3), 0.0) <= 0;
}

// is p3 on (p1, p2)?
template<class F>
inline bool pointOnSegment(const Point<F>& p1, const Point<F>& p2, const Point<F>& p3){
    return collinearity(p1, p2, p3) && btw(p1, p2, p3);
}

template <typename T, typename Real = double>
struct Line
{
    Point<T> st, ed;
    Point<T> vec() const { return ed - st; }
    T ori(const Point<T> p) const { return (ed - st) ^ (p - st); }
    int orint(const Point<T> p) const{
        T a = this->ori(p);
```

```
        return (fcmp(a, 0.0)); // 1 on posi-side // -1 nega-side
        // a little bit useless?
    }
    Line(const Point<T> x, const Point<T> y) : st(x), ed(y) {}
    template <class F>
    operator Line<F>() const
    {
        return Line<F>((Point<F>)st, (Point<F>)ed);
    }

    // sort by arg, the left is smaller for parallel lines
    bool operator<(Line B) const
    {
        Point<T> a = vec(), b = B.vec();
        auto sgn = [](const Point<T> t) { return (t.y == 0 ? t.x : t.y) < 0; };
        if (sgn(a) != sgn(b))
            return sgn(a) < sgn(b);
        if (abs(a ^ b) == 0)
            return B.ori(st) > 0;
        return (a ^ b) > 0;
    }

    // Regard a line as a function
    template <typename F>
    Point<F> operator()(const F x) const // A + AB * x = the point position.
    {
        return Point<F>(st) + vec() * x;
    }

    bool isSegProperIntersection(const Line l) const
    {
        return l.ori(st) * l.ori(ed) < 0 and ori(l.st) * ori(l.ed) < 0;
    }
    bool isSegIntersection(const Line l)const{
        Line<Real> h = *this;
        // hst = 1, hed = 2, lst = 3, led = 4
        double hlst = h.ori(l.st);
        double hled = h.ori(l.ed);
        double lhst = l.ori(h.st);
        double lhed = l.ori(h.ed);
        if(fcmp(hlst, 0.0) == 0 && fcmp(hled,0.0) == 0)
            return h.isPointOnSeg(l.st) || h.isPointOnSeg(l.ed) || l.isPointOnSeg(h.st) || l.isPointOnSeg(h.ed);

        return fcmp(hlst * hled, 0.0) <= 0 && fcmp(lhst * lhed, 0.0) <= 0;
    }

    bool isPointOnSegProperly(const Point<T> p) const
    {
        return fcmp(ori(p), 0.0) == 0 and fcmp(((st - p) & (ed - p)), 0.0) < 0;
    }
    bool isPointOnSeg(const Point<T>p) const{
        return fcmp(ori(p), 0.0) == 0 and fcmp((st - p) & (ed - p), 0.0) <= 0;
    }
    Real disP2Line(const Point<T> p) const
    {
        return Line<double>(projection(p), Point<double>(p)).vec().norm();
    }

    // notice if you should check Segment intersect or not;
    // be careful divided by 0
    Point<Real> getIntersection(Line<Real> l)
    {
        Line<Real> h = *this;
//        return l(((l.st - h.st)^ h.vec()) / (h.vec() ^ l.vec())); // use operator()
        Real hlst = -h.ori(l.st);
        Real hled = h.ori(l.ed);
        return ((l.st * hled) + (l.ed * hlst)) / (hlst + hled);

        // 需要確認+-號的合理性
        // Area of triangle(l.st, h.st, h.ed) divided by Area of Quadrilateral(h.st, l.st, h.ed, l.ed)
    }

    Point<Real> projection(const Point<T> p) const
    {
        return operator()(((p - st) & vec()) / (Real)(vec().norm()));
    }
    template <class F>
    friend ostream &operator<<(ostream &os, const Line<F> &l);
};
template <class F>
ostream &operator<<(ostream &os, const Line<F> &l)
{
    return os << "(" << l.st.x << ", " << l.st.y << ") to (" << l.ed.x << ", " << l.ed.y << ")";
}
template <class F>
using Polygon = vector<Point<F>>;
template <class F>
Polygon<F> getConvexHull(Polygon <F> points) {
    sort(points.begin(), points.end());
    Polygon<F> CH;
    CH.reserve(points.size() + 1); // for what ??
    for (int round = 0; round < 2; round++){
        int start = CH.size();
        for (Point<int> &pt: points) {
            while (CH.size() - start >= 2 && Line<F>(CH[CH.size() - 2], CH[CH.size() - 1]).ori(pt) <= 0) // ? Line is different than senpai's .
            {
                CH.pop_back();
            }
            CH.emplace_back(pt);
        }
        CH.pop_back();

        reverse(points.begin(), points.end());
    }
    if (CH.size() == 2 && CH[0] == CH[1])
        CH.pop_back();
    return CH;
}
```

3 geometry/Convex_Hull

3.1 Andrew's_Monotone_Chain

```
// Andrew's Monotone Chain
template <class F>
using Polygon = vector<Point<F>>;

template <class F>
Polygon<F> getConvexHull(Polygon<F> points)
{
    sort(begin(points), end(points));
    Polygon<F> hull;
    hull.reserve(points.size() + 1);
    for (int phase = 0; phase < 2; ++phase)
    {
        auto start = hull.size();
        for (auto &point : points)
        {
            while (hull.size() >= start + 2 &&
                   Line<F>(hull.back(), hull[hull.size() - 2]).ori(point) <= 0)
                hull.pop_back();

            // whenever point is at the RIGHT(NEGATIVE) part of the line(hull[size - 1], hull[size-2])
            // pop the last point because it causes concave hull
            hull.push_back(point);
        }
        hull.pop_back();
        reverse(begin(points), end(points));
    }
    if (hull.size() == 2 and hull[0] == hull[1])
        hull.pop_back();
    return hull;
}
```

4 graph

4.1 Kosaraju_for_SCC

```
#include <vector>
#include <stack>
using namespace std;
#define MAX_N 200005
class Kosaraju_for_SCC{
    int NodeNum;
    vector<vector<int>> G;
    vector<vector<int>> GT;
    stack<int> st;
    vector<bool> visited;
    vector<int> scc;
    int sccID;

public:
    void init(int N){
        NodeNum = N;
```

```
        G.clear();
        G.resize(N + 5);
        GT.clear();
        GT.resize(N + 5);
        while(!st.empty())
            st.pop();
        visited.clear();
        visited.resize(N + 5, false);
        scc.clear();
        scc.resize(N + 5);
        sccID = 1;
    }
    void addEdge(int w, int v){
        G[w].emplace_back(v);
        GT[v].emplace_back(w);
    }
    void DFS(bool isG, int v, int k = -1){
        visited[v] = true;
        scc[v] = k;
        vector<vector<int>> &dG = (isG ? G : GT);
        for(int w: dG[v])
        {
            if(!visited[w]){
                DFS(isG, w, k);
            }
        }
        if(isG){
            st.push(v);
        }
    }
    void Kosaraju(int N){
        visited.clear();
        visited.resize(N + 5, false);
        for (int i = 1; i <= N; i++){
            if(!visited[i])
                DFS(true, i);
        }
        visited.clear();
        visited.resize(N + 5, false);
        while(!st.empty()){
            if(!visited[st.top()])
                DFS(false, st.top(), sccID++);
            st.pop();
        }
    }
    vector<vector<int>> generateReG(){
        vector<vector<int>> reG;
        reG.resize(sccID);
        for (int i = 1; i <= NodeNum; i++){
            for(int w: G[i]){
               if(scc[i] == scc[w])
                    continue;
                reG[scc[i]].emplace_back(scc[w]);
            }
        }
        return reG;
    }
};
```

4.2 Tarjan_for_BridgeCC

```
// BCC for bridge connected component
// by sylveon a.k.a LFsWang
#include <vector>
#include <stack>
#include <algorithm>
using namespace std;
#define MAX_N 200005
int timestamp = 1;
int bccid = 1;
int D[MAX_N];
int L[MAX_N];
int bcc[MAX_N];
stack<int> st;
vector<int> adj[MAX_N];
bool inSt[MAX_N];
void DFS(int v, int fa) { //call DFS(v,v) at first
    D[v] = L[v] = timestamp++; //timestamp > 0
    st.emplace(v);

    for (int w:adj[v]) {
        if( w==fa ) continue;
        if ( !D[w] ) { // D[w] = 0 if not visited
            DFS(w,v);
            L[v] = min(L[v], L[w]);
        }
        L[v] = min(L[v], D[w]);
    }
    if (L[v]==D[v]) {
        bccid++;
        int x;
        do {
            x = st.top(); st.pop();
            bcc[x] = bccid;
        } while (x!=v);
    }
    return ;
}
```

4.3 Tarjan_for_AP_Bridge

```
#include <vector>
#include <utility>
using namespace std;
#define MAX_N 200005;
#define enp pair<int, int> // edge-weight, node-index
#define con pair<int, int> // connection

class tarjan{
    vector<vector<int>> G; // adjacency List
    vector<int> D;  // visit or visited and D-value
    vector<int> L;  // for L-value
    vector<con> edgeBridge;
    vector<int> APnode;
    int timestamp;
    tarjan(int size = 1){
        timestamp = 0;
        G.resize(size);
        D.resize(size, 0);
        L.resize(size, 0);
        edgeBridge.clear();
        APnode.clear();
    }
    void init(int size = 1){
        tarjan(size);
    }
    void addedge(int u, int v)
    {   // undirected graph
        G[u].push_back(v);
        G[v].push_back(u);
    }
    void DFS(int v, int pa){ // init: call DFS(v,v)
        D[v] = L[v] = timestamp++;
        int Childcount = 0;
        bool isAP = false;
        for(int w: G[v]){
            if(w == pa)
                continue;
            if(!D[w]){ // 用 D[w] == 0 if not visited
                DFS(w, v);
                Childcount++;
                if(D[v] <= L[w])
                    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)
                if(D[v] < L[w])
                    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)
            isAP = false;
        if(isAP)
            APnode.emplace_back(v);
    }
};
```

4.4 Tarjan_for_SCC

```
// by atsushi
#include <vector>
#include <stack>
using namespace std;

class tarjan_for_SCC{
private:
    vector<vector<int>> G; // adjacency list
    vector<int> D;
    vector<int> L;
    vector<int> sccID;
    stack<int> st; // for SccID
    vector<bool> inSt;
    vector<vector<int>> reG;
    int timeStamp, sccIDstamp;
public:
    void init(int size = 1){
        G.clear();
        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

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

// V times DFS O(EV)
vector<int> V[205];
// V[i]記錄了左半邊可以配到右邊的那些點
int match[205]; // A<=B
// match[i] 記錄了右半邊配對到左半邊的哪個點
bool used[205];
int n;
bool dfs(int v)
{
    for(int e:V[v])
    {
        if( used[e] ) continue;
        used[e] = true;
        if( match[e] == -1 || dfs( match[e] ) )
        {
            match[e] = v;
            return true;
        }
    }
    return false;
}
int konig()
{
    memset(match,-1,sizeof(match));

    int ans=0;

    for(int i=1;i<=n;++i)
    {
        memset(used, 0, sizeof(used));
        if( dfs(i) )
            ans++;
    }

    return ans;
}
```

5.2 Kuhn-Munkres

```
// Max weight perfect bipartite matching
// O(V^3)
// by jinkela
#define MAXN 405
#define INF 0x3f3f3f3f3f3f3f3f
int n;// 1-base，0表示沒有匹配
LL g[MAXN][MAXN]; //input graph
int My[MAXN],Mx[MAXN]; //output match
LL lx[MAXN],ly[MAXN],pa[MAXN],Sy[MAXN];
bool vx[MAXN],vy[MAXN];
void augment(int y){
  for(int x, z; y; y = z){
    x=pa[y],z=Mx[x];
    My[y]=x,Mx[x]=y;
  }
}
void bfs(int st){
  for(int i=1; i<=n; ++i)
    Sy[i] = INF, vx[i]=vy[i]=0;
  queue<int> q; q.push(st);
  for(;;){
    while(q.size()){
      int x=q.front(); q.pop();
      vx[x]=1;
      for(int y=1; y<=n; ++y) if(!vy[y]){
        LL t = lx[x]+ly[y]-g[x][y];
        if(t==0){
          pa[y]=x;
          if(!My[y]){augment(y);return;}
          vy[y]=1,q.push(My[y]);
        }else if(Sy[y]>t) pa[y]=x,Sy[y]=t;
      }
    }
    LL cut = INF;
    for(int y=1; y<=n; ++y)
      if(!vy[y]&&cut>Sy[y]) cut=Sy[y];
    for(int j=1; j<=n; ++j){
      if(vx[j]) lx[j] -= cut;
      if(vy[j]) ly[j] += cut;
      else Sy[j] -= cut;
    }
    for(int y=1; y<=n; ++y){
      if(!vy[y]&&Sy[y]==0){
        if(!My[y]){augment(y);return;}
        vy[y]=1, q.push(My[y]);
      }
    }
  }
}
LL KM(){
  memset(My,0,sizeof(int)*(n+1));
  memset(Mx,0,sizeof(int)*(n+1));
  memset(ly,0,sizeof(LL)*(n+1));
  for(int x=1; x<=n; ++x){
    lx[x] = -INF;
    for(int y=1; y<=n; ++y)
      lx[x] = max(lx[x],g[x][y]);
  }
  for(int x=1; x<=n; ++x) bfs(x);
  LL ans = 0;
  for(int y=1; y<=n; ++y) ans+=g[My[y]][y];
  return ans;
}
```

6 graph/Flow

6.1 Ford_Fulkerson

```
#include <vector>
#include <tuple>
#include <cstring>
using namespace std;

// O((V+E)F)
```

```cpp
#define maxn 101
// remember to change used into the maxNode size -- kattis elementary math
bool used[MAXN];
int End;
vector<int> V[MAXN];
vector<tuple<int, int>> E;

// x=>y 可以流 C
// if undirected or 2-direc edge, bakcward Capacity become C;
// Graph build by edge array
// 反向邊的編號只要把自己的編號 xor 1 就能取得
void add_edge(int x, int y,int c)
{
    V[x].emplace_back( E.size() );
    E.emplace_back(y,c);
    V[y].emplace_back( E.size() );
    E.emplace_back(x,0);
}
int dfs(int v, int f)
{
    if( v==End ) return f;
    used[v] = true;
    int e,w;
    for( int eid : V[v] )
    {
        tie(e,w) = E[eid];
        if( used[e] || w==0 ) continue;

        w = dfs(e, min(w,f));
        if( w>0 )
        {
            // 更新流量
            get<1>(E[eid  ]) -= w;
            get<1>(E[eid^1]) += w;
            return w;
        }
    }
    return 0;// Fail!
}
int ffa(int s,int e)
{
    int ans = 0, f;
    End = e;
    while(true)
    {
        memset(used, 0, sizeof(used));
        f = dfs(s, INT_MAX);
        if( f<=0 ) break;
        ans += f;
    }
    return ans;
}
```

6.2 Edmonds-Karp-adjmax

```cpp
// O((V+E)VE) ，簡單寫成 O(VE²)
#include <cstring>
#include <queue>
using namespace std;
#define maxn 100
typedef int Graph[MAXN][MAXN];  // adjacency matrix
Graph C, F, R;  // 容量上限、流量、剩餘容量
bool visit[MAXN]; // BFS經過的點
int path[MAXN];   // BFS tree
int flow[MAXN];   // 源點到各點的流量瓶頸

int BFS(int s, int t)   // 源點與匯點
{
    memset(visit, false, sizeof(visit));

    queue<int> Q;   // BFS queue
    visit[s] = true;
    path[s] = s;
    flow[s] = 1e9;
    Q.push(s);

    while (!Q.empty())
    {
        int i = Q.front(); Q.pop();
        for (int j=0; j<100; ++j)
            // 剩餘網路找擴充路徑
            if (!visit[j] && R[i][j] > 0)
            {
                visit[j] = true;
                path[j] = i;
                // 一邊找最短路徑，一邊計算流量瓶頸。
                flow[j] = min(flow[i], R[i][j]);
                Q.push(j);

                if (j == t) return flow[t];
            }
    }
    return 0;   // 找不到擴充路徑了，流量為零。
}

int Edmonds_Karp(int s, int t)
{
    memset(F, 0, sizeof(F));
    memcpy(R, C, sizeof(C));

    int f, df;  // 最大流的流量、擴充路徑的流量
    for (f=0; df=BFS(s, t); f+=df)
        // 更新擴充路徑上每一條邊的流量
        for (int i=path[t], j=t; i!=j; i=path[j=i])
        {
            F[i][j] = F[i][j] + df;
            F[j][i] = -F[i][j];
            R[i][j] = C[i][j] - F[i][j];
            R[j][i] = C[j][i] - F[j][i];
        }
    return f;
}
```

6.3 Dinic_algorithm

```cpp
#include <bits/stdc++.h>
using namespace std;
// O(V^2E) O(VE) finding argument path
// if unit capacity network then O(min(V^3/2, E^1/2) E)
// solving bipartite matching O(E V^1/2) better than konig and flow(EV)
#define MAXN 101
#define INT_MAX 10000000
int End, dist[MAXN];
vector<tuple<int, int, int>> V[MAXN];
// vertex-index, cap, the index of the reverse edge
void addEdge(int u, int v, int c){
    V[u].emplace_back(v, c, V[v].size());
    V[v].emplace_back(u, 0, V[u].size() - 1);
}
bool bfs(int s) {
    memset(dist, -1, sizeof(dist));
    queue<int> qu;
    qu.emplace(s);
    dist[s]=0;

    while( !qu.empty() ) {
        int S = qu.front(); qu.pop();
        for(auto &p : V[S]) {
            int E, C;
            tie(E, C, ignore) = p;
            if( dist[E]==-1 && C!=0 ) {
                dist[E]=dist[S]+1;
                qu.emplace(E);
            }
        }
    }
    return dist[End] != -1;
}
int dfs(int v, int f) {
    int e,w,rev;
    if( v==End || f==0 ) return f;
    for( auto &t : V[v] )
    {
        tie(e,w,rev) = t;
        if( dist[e]!=dist[v]+1 || w==0 )
            continue;

        w = dfs(e, min(w,f));
        if( w>0 )
        {
            get<1>(t) -= w;
            get<1>(V[e][rev]) += w;
            return w;
        }
    }
    dist[v] = -1; //優化，這個點沒用了
    return 0;// Fail!
}
int dinic(int s,int e)
{
    int ans = 0, f;
    End = e;
    while(bfs(s))
    {
        while( f = dfs(s, INT_MAX) )
            ans += f;
    }
    return ans;
}
```

6.4 Edmonds_Karp_2

```cpp
#include <bits/stdc++.h>
struct Edge{
```

```
    int from, to, cap, flow;
    Edge(int u, int v, int c, int f):from(u), to(v), cap(c),
        flow(f){}
};
const maxn = 200005;
struct EdmondsKarp{
    int n, m;
    vector<Edge> edges;
    vector<int> G[maxn];
    int a[maxn];
    int p[maxn];
    void init(int n){
        for (int i = 0; i < n; i++)
            G[i].clear();
        edges.clear();
    }
    void AddEdge(int from, int to, int cap){
        edges.push_back(Edge(from, to, cap, 0));
        edges.push_back(Edge(to, from, 0, 0)) // 反向弧
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }
    int Maxflow(int s, int t){
        int flow = 0;
        for (;;){
            memset(a, 0, sizeof(a));
            queue<int> Q;
            Q.push(s);
            a[s] = INF;
            while(!Q.empty()){
                int x = Q.front();
                Q.pop();
                for (int i = 0; i < G[x].size(); i++){
                    Edge &e = edges[G[x][i]];
                    if(!a[e.to] && e.cap > e.flow){
                        p[e.to] = G[x][i];
                        a.[e.to] = min(a[x], e.cap - e.flow);
                        Q.push(e.to);
                    }
                }
                if(a[t])
                    break;
            }
        }
        if(!a[t])
            break;
        for (int u = t; u != s; u = edges[p[u]].from){
            edges[p[u]].flow += a[t];
            edges[p[u] ^ 1].flow -= a[t];
        }
        flow += a[t];
    }
    return flow;
}
```

6.5 MinCostMaxFlow

```
// by jinkela
template<typename TP>
struct MCMF{
  static const int MAXN=440;
  static const TP INF=999999999;
  struct edge{
    int v,pre;
    TP r,cost;
    edge(int v,int pre,TP r,TP cost):v(v),pre(pre),r(r),cost(
        cost){}
  };
  int n,S,T;
  TP dis[MAXN],PIS,ans;
  bool vis[MAXN];
  vector<edge> e;
  int g[MAXN];
  void init(int _n){
    memset(g,-1,sizeof(int)*((n=_n)+1));
    e.clear();
  }
  void add_edge(int u,int v,TP r,TP cost,bool directed=false)
      {
    e.push_back(edge(v,g[u],r,cost));
    g[u]=e.size()-1;
    e.push_back(
    edge(u,g[v],directed?0:r,-cost));
    g[v]=e.size()-1;
  }
  TP augment(int u,TP CF){
    if(u==T||!CF)return ans+=PIS*CF,CF;
    vis[u]=1;
    TP r=CF,d;
    for(int i=g[u];~i;i=e[i].pre){
      if(e[i].r&&!e[i].cost&&!vis[e[i].v]){
        d=augment(e[i].v,min(r,e[i].r));
        e[i].r-=d;
        e[i^1].r+=d;
        if(!(r-=d))break;
      }
    }
    return CF-r;
  }
  bool modlabel(){
    for(int u=0;u<=n;++u)dis[u]=INF;
    static deque<int>q;
    dis[T]=0,q.push_back(T);
    while(q.size()){
      int u=q.front();q.pop_front();
      TP dt;
      for(int i=g[u];~i;i=e[i].pre){
        if(e[i^1].r&&(dt=dis[u]-e[i].cost)<dis[e[i].v]){
          if((dis[e[i].v]=dt)<=dis[q.size()?q.front():S]){
            q.push_front(e[i].v);
          }else q.push_back(e[i].v);
        }
      }
    }
    for(int u=0;u<=n;++u)
      for(int i=g[u];~i;i=e[i].pre)
        e[i].cost+=dis[e[i].v]-dis[u];
    return PIS+=dis[S], dis[S]<INF;
  }
  TP mincost(int s,int t){
    S=s,T=t;
    PIS=ans=0;
    while(modlabel()){
      do memset(vis,0,sizeof(bool)*(n+1));
      while(augment(S,INF));
    }return ans;
  }
};
```

7 graph/Matching

7.1 blossom_matching

```
// by jinkela
// 最大圖匹配
//  O(V²(V+E))
#define MAXN 505
int n; //1-base
vector<int> g[MAXN];
int MH[MAXN]; //output MH
int pa[MAXN],st[MAXN],S[MAXN],v[MAXN],t;
int lca(int x,int y){
  for(++t;;swap(x,y)){
    if(!x) continue;
    if(v[x]==t) return x;
    v[x] = t;
    x = st[pa[MH[x]]];
  }
}
#define qpush(x) q.push(x),S[x]=0
void flower(int x,int y,int l,queue<int>&q){
  while(st[x]!=l){
    pa[x]=y;
    if(S[y=MH[x]]==1)qpush(y);
    st[x]=st[y]=l, x=pa[y];
  }
}
bool bfs(int x){
  iota(st+1, st+n+1, 1);
  memset(S+1,-1,sizeof(int)*n);
  queue<int>q; qpush(x);
  while(q.size()){
    x=q.front(),q.pop();
    for(int y:g[x]){
      if(S[y]==-1){
        pa[y]=x,S[y]=1;
        if(!MH[y]){
          for(int lst;x;y=lst,x=pa[y])
            lst=MH[x],MH[x]=y,MH[y]=x;
          return 1;
        }
        qpush(MH[y]);
      }else if(!S[y]&&st[y]!=st[x]){
        int l=lca(y,x);
        flower(y,x,l,q),flower(x,y,l,q);
      }
    }
  }
  return 0;
}
int blossom(){
  memset(MH+1,0,sizeof(int)*n);
  int ans=0;
  for(int i=1; i<=n; ++i)
    if(!MH[i]&&bfs(i)) ++ans;
  return ans;
}
```

8 graph/Minimum_Spanning_Tree

8.1 prim

```cpp
#include <vector>
#include <queue>
#include <utility>
using namespace std;
#define enp pair<int, int> // pair<edge_val, node>
int prim_pq(vector<vector<enp>> E){
    vector<bool> vis;
    vis.resize(E.size(), false);
    vis[0] = true;
    priority_queue<enp> pq;
    for(auto e: E[0]){
        pq.emplace(-e.first, e.second);
    }
    int ans = 0; // min value for MST
    while(pq.size()){
        int w, v; // edge-weight, vertex index
        tie(w, v) = pq.top();
        pq.pop();
        if(vis[v])
            continue;
        w = -w;
        vis[v] = true;
        ans += w;
        for(auto e: E[v]){
            pq.emplace(-e.first, e.second);
        }
    }
    return ans;
}
```

8.2 Kruskal

```cpp
#include <tuple>
#include <vector>
#include <algorithm>
#include <numeric> // for iota(first, last, val) setting iterator value
using namespace std;

struct DSU // disjoint set no rank-comp-merge
{
    vector<int> fa;
    DSU(int n) : fa(n) { iota(fa.begin(), fa.end(), 0); } // auto fill fa from 0 to n-1
    int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }
    void merge(int x, int y) { fa[find(x)] = find(y); }
};
int kruskal(int V, vector<tuple<int, int, int>> E) // save all edges into E, instead of saving graph via adjacency list
{
    sort(E.begin(), E.end());
    DSU dsu(V);
    int mcnt = 0;
    int ans = 0;
    for (auto e : E)
    {
        int w, u, v; // w for start, u for des, v for val
        tie(w, u, v) = e;
        if (dsu.find(u) == dsu.find(v))
            continue;
        dsu.merge(u, v);
        ans += w;
        if (++mcnt == V - 1)
            break;
    }
    return ans;
}
```

9 graph/Shortest_Path

9.1 dijkstra

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <utility>
using namespace std;

#define con pair<int, int> // first for distance, second for index
vector<vector<con>> Graph; //
vector<int> dis; // distance;
int main(){
    priority_queue<con, vector<con>, greater<con>> pq;
    dis[0] = 0;
    pq.emplace(con(0, 0));
    while(pq.size()){
        con cur = pq.top();
        pq.pop();
        if(cur.first != dis[cur.second])
            continue;
        for(auto it: Graph[cur.second]){
            if(cur.first + it.first < dis[it.second]){
                dis[it.second] = cur.first + it.first;
                pq.emplace(dis[it.second], it.second);
            }
        }
    }
    return 0;
}
```

9.2 dijkstra-alrightchiu-version

```cpp
// C++ code
#include <iostream>
#include <vector>
#include <list>
#include <utility>          // for std::pair<>
#include <iomanip>          // for std::setw()
#include <cmath>            // for std::floor
// #include "Priority_Queue_BinaryHeap.h"

const int Max_Distance = 100;
class Graph_SP{             // SP serves as Shortest Path
private:
    int num_vertex;
    std::vector<std::list<std::pair<int,int>>> AdjList;
    std::vector<int> predecessor, distance;
    std::vector<bool> visited;
public:
    Graph_SP():num_vertex(0){};
    Graph_SP(int n):num_vertex(n){
        AdjList.resize(num_vertex);
    }
    void AddEdge(int from, int to, int weight);
    void PrintDataArray(std::vector<int> array);
    void PrintIntArray(int *array);

    void InitializeSingleSource(int Start);     // 以Start作為起點
    void Relax(int X, int Y, int weight);       // edge方向：from X to Y

    void Dijkstra(int Start = 0);        // 需要Min-Priority Queue
    friend class BinaryHeap;             // 以Binary Heap實現Min-Priority Queue
};
void Graph_SP::Dijkstra(int Start){

    InitializeSingleSource(Start);

    BinaryHeap minQueue(num_vertex);   // object of min queue
    minQueue.BuildMinHeap(distance);

    visited.resize(num_vertex, false);   // initializa visited[] as {0,0,0,...,0}

    while (!minQueue.IsHeapEmpty()) {
        int u = minQueue.ExtractMin();
        for (std::list<std::pair<int, int>>::iterator itr = AdjList[u].begin();
             itr != AdjList[u].end(); itr++) {

            Relax(u, (*itr).first, (*itr).second);
            minQueue.DecreaseKey((*itr).first, distance[(*itr).first]); // definition at alrightchiu's priority queue self-version
        }
    }
    std::cout << "\nprint predecessor:\n";
    PrintDataArray(predecessor);
    std::cout << "\nprint distance:\n";
    PrintDataArray(distance);
}
void Graph_SP::InitializeSingleSource(int Start){

    distance.resize(num_vertex);
    predecessor.resize(num_vertex);

    for (int i = 0; i < num_vertex; i++) {
        distance[i] = Max_Distance;
        predecessor[i] = -1;
    }
    distance[Start] = 0;
}
void Graph_SP::Relax(int from, int to, int weight){
```

```
67
68      if (distance[to] > distance[from] + weight) {
69          distance[to] = distance[from] + weight;
70          predecessor[to] = from;
71      }
72  }
73  void Graph_SP::AddEdge(int from, int to, int weight){
74
75      AdjList[from].push_back(std::make_pair(to,weight));
76  }
77  void Graph_SP::PrintDataArray(std::vector<int> array){
78      for (int i = 0; i < num_vertex; i++)
79          std::cout << std::setw(4) << i;
80      std::cout << std::endl;
81      for (int i = 0; i < num_vertex; i++)
82          std::cout << std::setw(4) << array[i];
83      std::cout << std::endl;
84  }
85  int main(){
86
87      Graph_SP g9(6);
88      g9.AddEdge(0, 1, 8);g9.AddEdge(0, 5, 1);
89      g9.AddEdge(1, 0, 3);g9.AddEdge(1, 2, 1);
90      g9.AddEdge(2, 0, 5);g9.AddEdge(2, 3, 2);g9.AddEdge(2, 4,
            2);
91      g9.AddEdge(3, 1, 4);g9.AddEdge(3, 2, 6);g9.AddEdge(3, 4,
            7);g9.AddEdge(3, 5, 3);
92      g9.AddEdge(5, 3, 2);g9.AddEdge(5, 4, 8);
93
94      g9.Dijkstra(0);
95
96      return 0;
97  }
```

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
            為起點
22      void Relax(int X, int Y, int weight);       // 對edge(X,Y
            )進行Relax
23      bool BellmanFord(int Start = 0);            // 以Start作
            為起點
24                                                  // if there
                                                       is
                                                       negative
                                                        cycle,
                                                       return
                                                       false
25  };
26
27  bool Graph_SP::BellmanFord(int Start){
28
29      InitializeSingleSource(Start);
30
31      for (int i = 0; i < num_vertex-1; i++) {
            // |V-1|次的iteration
32          // for each edge belonging to E(G)
33          for (int j = 0 ; j < num_vertex; j++) {
                // 把AdjList最外層的vector走一遍
34              for (std::list<std::pair<int,int> >::iterator itr
                     = AdjList[j].begin();
35                   itr != AdjList[j].end(); itr++) {
                        // 各個vector中, 所有edge走一遍
36                  Relax(j, (*itr).first, (*itr).second);
37              }
38          }
39      }
40
41      // check if there is negative cycle
42      for (int i = 0; i < num_vertex; i++) {
43          for (std::list<std::pair<int,int> >::iterator itr =
                AdjList[i].begin();
44               itr != AdjList[i].end(); itr++) {
45              if (distance[(*itr).first] > distance[i]+(*itr).
                    second) {   // i是from, *itr是to
46                  return false;
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
69      distance.resize(num_vertex);
70      predecessor.resize(num_vertex);
71
72      for (int i = 0; i < num_vertex; i++) {
73          distance[i] = Max_Distance;
74          predecessor[i] = -1;
75      }
76      distance[Start] = 0;
77  }
78  void Graph_SP::Relax(int from, int to, int weight){
79
80      if (distance[to] > distance[from] + weight) {
81          distance[to] = distance[from] + weight;
82          predecessor[to] = from;
83      }
84  }
85  void Graph_SP::AddEdge(int from, int to, int weight){
86      AdjList[from].push_back(std::make_pair(to,weight));
87  }
88
89  int main(){
90
91      Graph_SP g7(6);
92      g7.AddEdge(0, 1, 5);
93      g7.AddEdge(1, 4, -4);g7.AddEdge(1, 2, 6);
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              if (i == j){
29                  AdjMatrix[i][j] = 0;
30              }
```

```
        }
    }
}
void Graph_SP_AllPairs::InitializeData(){
    Distance.resize(num_vertex);
    Predecessor.resize(num_vertex);

    for (int i = 0; i < num_vertex; i++) {
        Distance[i].resize(num_vertex);
        Predecessor[i].resize(num_vertex, -1);
        for (int j = 0; j < num_vertex; j++) {
            Distance[i][j] = AdjMatrix[i][j];
            if (Distance[i][j] != 0 && Distance[i][j] !=
                 MaxDistance) {
                Predecessor[i][j] = i;
            }
        }
    }
}
void Graph_SP_AllPairs::FloydWarshall(){

    InitializeData();

    std::cout << "initial Distance[]:\n";
    PrintData(Distance);
    std::cout << "\ninitial Predecessor[]:\n";
    PrintData(Predecessor);

    for (int k = 0; k < num_vertex; k++) {
        std::cout << "\nincluding vertex(" << k << "):\n";
        for (int i = 0; i < num_vertex; i++) {
            for (int j = 0; j < num_vertex; j++) {
                if ((Distance[i][j] > Distance[i][k]+Distance
                     [k][j])
                     && (Distance[i][k] != MaxDistance)) {
                    Distance[i][j] = Distance[i][k]+Distance[
                         k][j];
                    Predecessor[i][j] = Predecessor[k][j];
                }
            }
        }
        // print data after including new vertex and updating
             the shortest paths
        std::cout << "Distance[]:\n";
        PrintData(Distance);
        std::cout << "\nPredecessor[]:\n";
        PrintData(Predecessor);
    }
}
void Graph_SP_AllPairs::PrintData(std::vector< std::vector<
     int> > array){

    for (int i = 0; i < num_vertex; i++){
        for (int j = 0; j < num_vertex; j++) {
            std::cout << std::setw(5) << array[i][j];
        }
        std::cout << std::endl;
    }
}
void Graph_SP_AllPairs::AddEdge(int from, int to, int weight)
     {
    AdjMatrix[from][to] = weight;
}

int main(){

    Graph_SP_AllPairs g10(4);
    g10.AddEdge(0, 1, 2);g10.AddEdge(0, 2, 6);g10.AddEdge(0,
         3, 8);
    g10.AddEdge(1, 2, -2);g10.AddEdge(1, 3, 3);
    g10.AddEdge(2, 0, 4);g10.AddEdge(2, 3, 1);

    g10.FloydWarshall();

    return 0;
}
```

9.5 shortest-path__on__DAG

```
// C++ code
// O(V+E)
#include <iostream>
#include <vector>
#include <list>
#include <utility>          // for std::pair<>
#include <iomanip>          // for std::setw()

const int Max_Distance = 100;
class Graph_SP{             // SP serves as Shortest Path
private:
    int num_vertex;
    std::vector<std::list<std::pair<int,int>>> AdjList;
    std::vector<int> predecessor, distance;
public:
    Graph_SP():num_vertex(0){};
    Graph_SP(int n):num_vertex(n){
        AdjList.resize(num_vertex);
    }
    void AddEdge(int from, int to, int weight);
    void PrintDataArray(std::vector<int> array);
    void PrintIntArray(int *array);

    void InitializeSingleSource(int Start);     // 以Start作
         為起點
    void Relax(int X, int Y, int weight);       // 對edge(X,Y
         )進行Relax

    void DAG_SP(int Start = 0);                      // 需要
         DFS, 加一個額外的Linked list
    void GetTopologicalSort(int *array, int Start);
    void DFSVisit_TS(int *array, int *color, int *discover,
                     int *finish, int vertex, int &time, int
                          &count);

};
void Graph_SP::GetTopologicalSort(int *array, int Start){

    int color[num_vertex], discover[num_vertex], finish[
         num_vertex];

    for (int i = 0; i < num_vertex; i++) {
        color[i] = 0;
        discover[i] = 0;
        finish[i] = 0;
        predecessor[i] = -1;
    }

    int time = 0,
        count = num_vertex-1,        // count 為
             topologicalsort[] 的 index
        i = Start;

    for (int j = 0; j < num_vertex; j++) {
        if (color[i] == 0) {
            DFSVisit_TS(array, color, discover, finish, i,
                 time, count);
        }
        i = j;
    }
    std::cout << "\nprint discover time:\n";
    PrintIntArray(discover);
    std::cout << "\nprint finish time:\n";
    PrintIntArray(finish);
}
void Graph_SP::DFSVisit_TS(int *array, int *color, int *
     discover,
                           int *finish, int vertex, int &time
                                , int &count){

    color[vertex] = 1;  // set gray
    discover[vertex] = ++time;
    for (std::list<std::pair<int,int>>::iterator itr =
         AdjList[vertex].begin();
         itr != AdjList[vertex].end(); itr++) {
        if (color[(*itr).first] == 0) {
            predecessor[(*itr).first] = vertex;
            DFSVisit_TS(array, color, discover, finish, (*itr
                 ).first, time, count);
        }
    }
    color[vertex] = 2;  // set black
    finish[vertex] = ++time;
    array[count--] = vertex;            // 產生Topological
         Sort
}
void Graph_SP::DAG_SP(int Start){

    InitializeSingleSource(Start);      // distance[],
         predecessor[]的initialization

    int topologicalsort[num_vertex];
    GetTopologicalSort(topologicalsort, Start);

    for (int i = 0; i < num_vertex; i++) {
        int v = topologicalsort[i];
        for (std::list<std::pair<int, int>>::iterator itr =
             AdjList[v].begin();
             itr != AdjList[v].end(); itr++) {
            Relax(v, (*itr).first, (*itr).second);
        }
    }
    std::cout << "\nprint predecessor:\n";
    PrintDataArray(predecessor);
    std::cout << "\nprint distance:\n";
    PrintDataArray(distance);
}
void Graph_SP::PrintDataArray(std::vector<int> array){
    for (int i = 0; i < num_vertex; i++)
        std::cout << std::setw(4) << i;
    std::cout << std::endl;
    for (int i = 0; i < num_vertex; i++)
        std::cout << std::setw(4) << array[i];
    std::cout << std::endl;
```

```
}
void Graph_SP::PrintIntArray(int *array){
    for (int i = 0; i < num_vertex; i++)
        std::cout << std::setw(4) << i;
    std::cout << std::endl;
    for (int i = 0; i < num_vertex; i++)
        std::cout << std::setw(4) << array[i];
    std::cout << std::endl;
}
void Graph_SP::InitializeSingleSource(int Start){

    distance.resize(num_vertex);
    predecessor.resize(num_vertex);

    for (int i = 0; i < num_vertex; i++) {
        distance[i] = Max_Distance;
        predecessor[i] = -1;
    }
    distance[Start] = 0;
}
void Graph_SP::Relax(int from, int to, int weight){

    if (distance[to] > distance[from] + weight) {
        distance[to] = distance[from] + weight;
        predecessor[to] = from;
    }
}
void Graph_SP::AddEdge(int from, int to, int weight){
    AdjList[from].push_back(std::make_pair(to,weight));
}

int main(){

    Graph_SP g8(7);
    g8.AddEdge(0, 1, 3);g8.AddEdge(0, 2, -2);
    g8.AddEdge(1, 3, -4);g8.AddEdge(1, 4, 4);
    g8.AddEdge(2, 4, 5);g8.AddEdge(2, 5, 6);
    g8.AddEdge(3, 5, 8);g8.AddEdge(3, 6, 2);
    g8.AddEdge(4, 3, -3);g8.AddEdge(4, 6, -2);
    g8.AddEdge(5, 6, 2);

    g8.DAG_SP(0);        // 以vertex(0)作為起點

    return 0;
}
```

10 graph/Tree

10.1 Lowest_Common_Ancestor

```
#define MAXN 200005
#define MAXLOG 200
int D[MAXN];
int P[MAXLOG][MAXLOG];
#include <cmath>
#include <algorithm>
using namespace std;
#define MAXN 200005
#define MAXLOG 200
int N = MAXN;
int lgN = log(N) / log(2);
int D[MAXN];
int P[MAXLOG][MAXLOG];
int LCA(int u, int v)
{
    if (D[u] > D[v])
        swap(u, v);
    int s = D[v] - D[u]; // adjust D until D[v] = D[u]

    for (int i = 0; i <= lgN; ++i) // 調整他們到二進位數一樣
        if (s & (1 << i))
            v = P[v][i];
    if (u == v)
        return v;
    // because they are at same depth
    // jump up if they are different
    // think about that if P[u][i] == P[v][i]
    // then that point must be the ancestor of LCA or LCA
        itself
    // by this, we will stop at LCA's child
    for (int i = lgN; i >= 0; --i)
        //
        if (P[u][i] != P[v][i])
        {
            u = P[u][i];
            v = P[v][i];
        }
    return P[u][0];
}

void ComputeP()
{
    int n = N;
    for (int i = 0; i < lgN; ++i) // to lgN enough
    {
        for (int x = 0; x < n; ++x)
        {
            if (P[x][i] == -1)
                P[x][i + 1] = -1;
            else
                P[x][i + 1] = P[P[x][i]][i]; // equal to move
                    on the parent direction
                // And P[x][i] move 2 ^ n steps to a parent
                    we call it y
                // P[y][i] means continue move 2 ^ n step
                    from y to a parent we call z
                // so the total equal to move 2 ^ n * 2 ^ n
                    steps from x to z
                // which is move 2 ^ (n + 1) steps to z
        }
    }
}
```

10.2 Tree_Centroid

```
#include <utility>
#include <vector>
#include <algorithm>
using namespace std;


int subTsize[200005];
vector<int> adj[200005];
int n; // n for node num ??
pair<int, int> Tree_Centroid(int v, int pa)
{
    // return (最 大 子 樹 節 點 數 , 節 點ID)
    subTsize[v] = 1;
    pair<int, int> res(INT_MAX, -1); // ans: tree cnetroid
    int max_subT = 0; // 最大子樹節點數
    for (size_t i = 0; i < adj[v].size(); ++i)
    {
        int x = adj[v][i];
        if (x == pa)
            continue;
        res = min(res, Tree_Centroid(x, v));
        subTsize[v] += subTsize[x];
        max_subT = max(max_subT, subTsize[x]);
    }
    res = min(res, make_pair(max(max_subT, n - subTsize[v]),
         v)); // (n - subTsize[v]) for maybe parent tree is
         the biggest
    // min because all res will be greater than n/2;
    // the min one is the tree centroid
    return res;
}

// Tree_Centroid2
vector<int> V[10005];
int N;
int center, csize;
int dfs(int v, int fa)
{
    int sz = 1;
    int maxsub = 0;

    for(int u:V[v])
    {
        if (u==fa)continue;
        int sub = dfs(u, v);
        maxsub = max(maxsub, sub);
        sz += sub;
    }
    maxsub = max(maxsub, N-sz);

    if (maxsub<csize)
    {
        center = v;
        csize = maxsub;
    }
    return sz;
}
```

11 hashing

11.1 hashingVec

```
#include<bits/stdc++.h>
struct VectorHash {
    size_t operator()(const std::vector<int>& v) const {
        std::hash<int> hasher;
        size_t seed = 0;
        for (const int& i : v) {
            seed ^= hasher(i) + 0x9e3779b9 + (seed<<6) + (
                seed>>2);
```

```
        }
        return seed;
    }
};
std::unordered_set<std::vector<int>, VectorHash> H;
```

12 number_theory

12.1 Fib

```
#include <bits/stdc++.h>
using namespace std;
// Cassini's identity : F_{n-1}F_{n+1} - F_n^2= (-1)^n
// The "addition" rule : F_{n+k} = F_kF_{n+1}+F_{k-1}F_n
// k = n, F_{2n} = F_n*(F_{n+1} + F_{n-1})
// F_{2k} = F_k*(2F_{K+1}-F_k)
// F_{2k+1} = F_{K+1}^2+F_k^2
pair<int, int> fib (int n) {
    if (n == 0)
        return {0, 1};
    auto p = fib(n >> 1);
    int c = p.first * (2 * p.second - p.first);
    int d = p.first * p.first + p.second * p.second;
    if (n & 1)
        return {d, c + d};
    else
        return {c, d};
}
```

12.2 BigInterger

```
#include <vector>
#include <string>
#include <iostream>
#include <cmath>
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
struct BigInteger{
    static const int BASE = 100000000;
    static const int WIDTH = 8;
    vector<int> s;

    BigInteger(long long num = 0) { *this = num; }
    BigInteger operator = (long long num)   {
        s.clear();
        do{
            s.push_back(num % BASE);
            num /= BASE;
        } while (num > 0);
        return *this;
    }

    BigInteger operator = (const string& str){
        s.clear();
        int x, len = (str.length() - 1) / WIDTH + 1;
        for (int i = 0; i < len;i++){
            int end = str.length() - i * WIDTH;
            int start = max(0, end - WIDTH);
            sscanf(str.substr(start, end - start).c_str(), "%d", &x);
            s.push_back(x);
        }
        return *this;
    }

    BigInteger operator+ (const BigInteger b) const{
        BigInteger c;
        c.s.clear();
        for(int i=0,g=0;;i++){
            if(g== 0 && i >=s.size() && i >=b.s.size())
                break;
            int x = g;
            if(i<s.size()) x+=s[i];
            if(i<b.s.size()) x+=b.s[i];
            c.s.push_back(x % BASE);
            g = x/ BASE;
        }
        return c;
    }
    BigInteger operator+=(const BigInteger& b){
        *this = *this + b;
        return *this;
    }
    BigInteger operator* (const BigInteger b)const{
        BigInteger c;
        c.s.clear();
        long long mul;
        for (int i = 0;i < s.size(); i++)
        {
            long long carry = 0;
            for (int g = 0; g < b.s.size();g++){
                mul = (long long)(s[i]) * (long long)(b.s[g])
                    ;
                mul += carry;
                if(i + g < c.s.size()){
                    c.s[i+g] += mul % BASE;
                }else{
                    c.s.push_back(mul % BASE);
                }
                carry = mul / BASE;
            }
        }
        for (int i = 0; i < c.s.size(); i++){
            if(c.s[i] >= BASE){
                if(i + 1 < c.s.size()){
                    c.s.push_back(c.s[i] / BASE);
                }else{
                    c.s[i + 1] += c.s[i] / BASE;
                }
                c.s[i] %= BASE;
            }
        }
        return c;
    }
    bool operator< (const BigInteger& b) const{
        if(s.size() != b.s.size()) return s.size() < b.s.size
            ();
        for(int i=s.size() -1 ; i>=0;i--)
            if(s[i] != b.s[i]) return s[i] < b.s[i];
        return false; // Equal
    }
    bool operator> (const BigInteger& b) const{return b < *
        this;}
    bool operator<= (const BigInteger& b) const {return !(b<*
        this);}
    bool operator>=(const BigInteger& b) const {return !(*
        this < b);}
    bool operator!=(const BigInteger& b) const {return b< *
        this || *this < b;}
    bool operator==(const BigInteger& b)const {return !(b<*
        this) && !(*this<b);}
};
ostream& operator<< (ostream &out, const BigInteger& x){
    out << x.s.back();
    for (int i = x.s.size() - 2;i >= 0;i--){
        char buf[20];
        sprintf(buf,"%08d",x.s[i]);
        for(int j = 0;j<strlen(buf);j++) out << buf[j];
    }
    return out;
}
istream& operator>> (istream &in, BigInteger & x){
    string s;
    if(!(in >> s)) return in;
    x= s;
    return in;
}
```

12.3 gcds

```
#include <bits/stdc++.h>
using namespace std;
// O(log(min(a, b)))
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

bool find_any_solution(int a, int b, int c, int &x0, int &y0,
     int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

// finding all solution
void shift_solution(int & x, int & y, int a, int b, int cnt)
    {
    x += cnt * b;
```

```
    y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
// smallest possible val
// x' + y' = x + y + k(b-a)g, minimize b-a
```

12.4 nCr

```
using i64 = long long;
#define maxn 300005
i64 fact[MAXN], tcaf[MAXN];

#define P 998244353
#define REP1(i, n) for (int i = 1; i <= (int)(n); ++i)
#define REP(i, n) for (int i = (int)(n) - 1; i >= 0; --i)
void init(int n){
    fact[0] = 1;
    for (int i = 1; i <= n; i++)
        fact[i] = i * fact[i - 1] % P;
    for (int i = n; i >= 0; --i)
        tcaf[i] = deg(fact[i], -1);

}

i64 deg(i64 x, i64 d) {
    if (d < 0) d += P - 1;
    i64 y = 1;
    while (d) {
        if (d & 1) (y *= x) %= P;
        d /= 2;
        (x *= x) %= P;
    }
    return y;
}

i64 cnk(int n, int k) {
    if (k < 0 || k > n) return 0;
    return fact[n] * tcaf[k] % P * tcaf[n - k] % P;
}
```

13 string

13.1 KMP

```
#include <iostream>
#include <string>
#include <regex>
#include <vector>
using namespace std;
// T for Text, P for Patterm
vector <int> build_kmp(const string &P) {
    vector <int> f(P.size());
    int fp = f[0] = -1;
    for (int i = 1; i < P.size(); ++i) {
        while (~fp && P[fp + 1] != P[i])
            fp = f[fp];
        if (P[fp + 1] == P[i])
            ++fp;
        f[i] = fp;
    }

    return f;
}
vector <int> kmp_match(vector <int> fail, const string &P, const string &T)
{
    vector <int> res; // start from these points
    const int n = P.size();
    for (int j = 0, i = -1; j < T.size(); ++j) {
        while (~i && T[j] != P[i + 1])
            i = fail[i];
        if (P[i + 1] == T[j])
            ++i;
        if (i == n - 1)
            res.push_back(j - n + 1), i = fail[i];
    }
    return res;
}
int main(){
    char control;
    string test_patterm = "a";
    string test_text = "abcdabcdabceabcd";
    // for testing
    cout << "Do you want to Enter by ys?(y/n)";
    cin >> control;
    if(control == 'y' || control == 'Y'){
        cout << "Enter text:";
        cin >> test_text;
        cout << "Enter patterm:";
        cin >> test_patterm;
    }

    vector<int> V = build_kmp(test_patterm);
    vector<int> Ans = kmp_match(V, test_patterm, test_text);
    cout << '\n';
    for(auto it = V.begin(); it != V.end(); ++it)
        cout << *it <<' ';
    cout << '\n';
    for(auto it = Ans.begin(); it != Ans.end(); ++it)
        cout << *it <<' ';
    return 0;
}
```

ACM ICPC Team Reference - Duracell!

Contents