# 1 Basic

## 1.1 data range

```
int (-2147483648 to 2147483647)
unsigned int(0 to 4294967295)
long(-2147483648 to 2147483647)
unsigned long(0 to 4294967295)
long long(-9223372036854775808 to 9223372036854775807)
unsigned long long (0 to 18446744073709551615)
```

## 1.2 IO_fast

```
ios_base::sync_with_stdio(0);
cin.tie(0);
```

# 2 DP

## 2.1 KMP

```
void ComputePrefix(string s, int next[])
{
    int n = s.length();
    int q, k;
    next[0] = 0;
    for (k = 0, q = 1; q < n; q++)
    {
        while (k > 0 && s[k] != s[q])
            k = next[k];
        if (s[k] == s[q])
            k++;
        next[q] = k;
    }
}
void KMPMatcher(string text, string pattern)
{
    int n = text.length();
    int m = pattern.length();
    int next[pattern.length()];
    ComputePrefix(pattern, next);

    for (int i = 0, q = 0; i < n; i++)
    {
        while (q > 0 && pattern[q] != text[i])
            q = next[q];
        if (pattern[q] == text[i])
            q++;
        if (q == m)
        {
            cout << "Pattern occurs with shift " << i - m + 1 << endl;
            q = 0;
        }
    }
}
// string s = "abcdabcdebcd";
// string p = "bcd";
// KMPMatcher(s, p);
// cout << endl;
```

## 2.2 LCS

```
int LCS(vector<int> Ans, vector<int> num) // Ans 跟 num 都要 index 從1開始放
{
    vector<vector<int>> LCS(N + 1, vector<int>(N + 1, 0));
    for (int i = 1; i <= N; ++i)
    {
        for (int j = 1; j <= N; ++j)
        {
            if (Ans[i] == num[j])
                LCS[i][j] = LCS[i - 1][j - 1] + 1;
            else
                LCS[i][j] = max(LCS[i - 1][j], LCS[i][j - 1]);
        }
    }
    // printf("%d\n", LCS[N][N]);
    return LCS[N][N];
    //列印 LCS
    vector<int> k;
    while (n && m)
    {
        if (LCS[n][m] != max(LCS[n - 1][m], LCS[n][m - 1]))
        {
            k.push_back(arr1[n]);
            n--;
            m--;
        }
        else if (LCS[n][m] == LCS[n - 1][m])
            n--;
        else if (LCS[n][m] == LCS[n][m - 1])
            m--;
    }

    reverse(k.begin(), k.end());
}
```

## 2.3 LIC

```
void getMaxElementAndPos(vector<int> &LISTbl, vector<int> &LISLen, int tNum,
    int tlen, int tStart, int &num, int &pos)
{
    int max = numeric_limits<int>::min();
    int maxPos;
    for (int i = tStart; i >= 0; i--)
    {
        if (LISLen[i] == tlen && LISTbl[i] < tNum)
        {
            if (LISTbl[i] > max)
            {
                max = LISTbl[i];
                maxPos = i;
            }
        }
    }
    num = max;
    pos = maxPos;
}
int LIS(vector<int> &LISTbl)
{
    if (LISTbl.size() == 0)
        return 0;
    vector<int> LISLen(LISTbl.size(), 1);
    for (int i = 1; i < LISTbl.size(); i++)
    {
        for (int j = 0; j < i; j++)
        {
            if (LISTbl[j] < LISTbl[i])
                LISLen[i] = max(LISLen[i], LISLen[j] + 1);
        }
    }

    int maxlen = *max_element(LISLen.begin(), LISLen.end());
    int num, pos;
    vector<int> buf;
    getMaxElementAndPos(LISTbl, LISLen,
                        numeric_limits<int>::max(),
                        maxlen, LISTbl.size() - 1, num, pos);
    buf.push_back(num);
    for (int len = maxlen - 1; len >= 1; len--)
    {
        int tnum = num;
        int tpos = pos;
        getMaxElementAndPos(LISTbl, LISLen,
                            tnum, len, tpos - 1, num, pos);
        buf.push_back(num);
    }
    reverse(buf.begin(), buf.end());
    for (int k = 0; k < buf.size(); k++) //列印
    {
        if (k == buf.size() - 1)
            cout << buf[k] << endl;
        else
            cout << buf[k] << ",";
    }
    return maxlen;
}
```

## 2.4 LPS

```
void LPS(string s)
{
    int maxlen = 0, l, r;
    int n = n;
    for (int i = 0; i < n; i++)
    {
        int x = 0;
        while ((s[i - x] == s[i + x]) && (i - x >= 0) && (i + x < n)) //odd length
            x++;
        x--;
        if (2 * x + 1 > maxlen)
        {
            maxlen = 2 * x + 1;
            l = i - x;
            r = i + x;
        }
        x = 0;
        while ((s[i - x] == s[i + 1 + x]) && (i - x >= 0) && (i + 1 + x < n)) //even length
            x++;
        if (2 * x > maxlen)
        {
            maxlen = 2 * x;
            l = i - x + 1;
            r = i + x;
        }
    }

    cout << maxlen << '\n';  // 最後長度
    cout << l + 1 << ' ' << r + 1 << '\n'; //頭到尾
}
```

## 2.5 Max_subarray

```
/*Kadane's algorithm*/
int maxSubArray(vector<int>& nums) {
    int local_max = nums[0], global_max = nums[0];
    for(int i = 1; i < nums.size(); i++){
        local_max = max(nums[i],nums[i]+local_max);
        global_max = max(local_max, global_max);
    }
    return global_max;
}
```

## 2.6 MFlow

```
typedef long long ll;
struct MF
{
```

```cpp
    static const int N = 5000 + 5;
    static const int M = 60000 + 5;
    static const ll oo = 10000000000000LL;

    int n, m, s, t, tot, tim;
    int first[N], next[M];
    int u[M], v[M], cur[N], vi[N];
    ll cap[M], flow[M], dis[N];
    int que[N + N];

    void Clear()
    {
        tot = 0;
        tim = 0;
        for (int i = 1; i <= n; ++i)
            first[i] = -1;
    }
    void Add(int from, int to, ll cp, ll flw)
    {
        u[tot] = from;
        v[tot] = to;
        cap[tot] = cp;
        flow[tot] = flw;
        next[tot] = first[u[tot]];
        first[u[tot]] = tot;
        ++tot;
    }
    bool bfs()
    {
        ++tim;
        dis[s] = 0;
        vi[s] = tim;

        int head, tail;
        head = tail = 1;
        que[head] = s;
        while (head <= tail)
        {
            for (int i = first[que[head]]; i != -1; i = next[i])
            {
                if (vi[v[i]] != tim && cap[i] > flow[i])
                {
                    vi[v[i]] = tim;
                    dis[v[i]] = dis[que[head]] + 1;
                    que[++tail] = v[i];
                }
            }
            ++head;
        }
        return vi[t] == tim;
    }
    ll dfs(int x, ll a)
    {
        if (x == t || a == 0)
            return a;
        ll flw = 0, f;
        int &i = cur[x];
        for (i = first[x]; i != -1; i = next[i])
        {
            if (dis[x] + 1 == dis[v[i]] && (f = dfs(v[i], min(a, cap[i] - flow[i]))) > 0)
            {
                flow[i] += f;
                flow[i ^ 1] -= f;
                a -= f;
                flw += f;
                if (a == 0)
                    break;
            }
        }
        return flw;
    }
    ll MaxFlow(int s, int t)
    {
        this->s = s;
        this->t = t;
        ll flw = 0;
        while (bfs())
        {
            for (int i = 1; i <= n; ++i)
                cur[i] = 0;
            flw += dfs(s, oo);
        }
        return flw;
    }
};
// MF Net;
// Net.n = n;
// Net.Clear();
// a 到 b (注意從1開始!!!!)
// Net.Add(a, b, w, 0);
// Net.MaxFlow(s, d)
// s 到 d 的 MF
```

# 3 Geometry

## 3.1 Line

```cpp
template <typename T>
struct line
{
    line() {}
    point<T> p1, p2;
    T a, b, c; //ax+by+c=0
    line(const point<T> &x, const point<T> &y) : p1(x), p2(y) {}
    void pton()
    { //轉成一般式
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = -a * p1.x - b * p1.y;
    }
    T ori(const point<T> &p) const
    { //點和有向直線的關係，>0左邊、=0在線上<0右邊
        return (p2 - p1).cross(p - p1);
    }
    T btw(const point<T> &p) const
    { //點投影落在線段上<=0
        return (p1 - p).dot(p2 - p);
    }
    bool point_on_segment(const point<T> &p) const
    { //點是否在線段上
        return ori(p) == 0 && btw(p) <= 0;
    }
    T dis2(const point<T> &p, bool is_segment = 0) const
    { //點跟直線/線段的距離平方
        point<T> v = p2 - p1, v1 = p - p1;
        if (is_segment)
        {
            point<T> v2 = p - p2;
            if (v.dot(v1) <= 0)
                return v1.abs2();
            if (v.dot(v2) >= 0)
                return v2.abs2();
        }
        T tmp = v.cross(v1);
        return tmp * tmp / v.abs2();
    }
    T seg_dis2(const line<T> &l) const
    { //兩線段距離平方
        return min({dis2(l.p1, 1), dis2(l.p2, 1), l.dis2(p1, 1), l.dis2(p2, 1)});
    }
    point<T> projection(const point<T> &p) const
    { //點對直線的投影
        point<T> n = (p2 - p1).normal();
        return p - n * (p - p1).dot(n) / n.abs2();
    }
    point<T> mirror(const point<T> &p) const
    {
        //點對直線的鏡射，要先呼叫pton轉成一般式
        point<T> R;
        T d = a * a + b * b;
        R.x = (b * b * p.x - a * a * p.x - 2 * a * b * p.y - 2 * a * c) / d;
        R.y = (a * a * p.y - b * b * p.y - 2 * a * b * p.x - 2 * b * c) / d;
        return R;
    }
    bool equal(const line &l) const
    { //直線相等
        return ori(l.p1) == 0 && ori(l.p2) == 0;
    }
    bool parallel(const line &l) const
    {
        return (p1 - p2).cross(l.p1 - l.p2) == 0;
    }
    bool cross_seg(const line &l) const
    {
        return (p2 - p1).cross(l.p1 - p1) * (p2 - p1).cross(l.p2 - p1) <= 0; //直線是否交線段
    }
    int line_intersect(const line &l) const
    { //直線相交情況，-1無限多點、1交於一點、0不相交
        return parallel(l) ? (ori(l.p1) == 0 ? -1 : 0) : 1;
    }
    int seg_intersect(const line &l) const
    {
        T c1 = ori(l.p1), c2 = ori(l.p2);
        T c3 = l.ori(p1), c4 = l.ori(p2);
        if (c1 == 0 && c2 == 0)
        { //共線
            bool b1 = btw(l.p1) >= 0, b2 = btw(l.p2) >= 0;
            T a3 = l.btw(p1), a4 = l.btw(p2);
            if (b1 && b2 && a3 == 0 && a4 >= 0)
                return 2;
            if (b1 && b2 && a3 >= 0 && a4 == 0)
                return 3;
            if (b1 && b2 && a3 >= 0 && a4 >= 0)
                return 0;
            return -1; //無限交點
        }
        else if (c1 * c2 <= 0 && c3 * c4 <= 0)
            return 1;
        return 0; //不相交
    }
    point<T> line_intersection(const line &l) const
    { /*直線交點*/
        point<T> a = p2 - p1, b = l.p2 - l.p1, s = l.p1 - p1;
        //if(a.cross(b)==0)return INF;
        return p1 + a * (s.cross(b) / a.cross(b));
    }
    point<T> seg_intersection(const line &l) const
    { //線段交點
        int res = seg_intersect(l);
        if (res <= 0)
            assert(0);
        if (res == 2)
            return p1;
        if (res == 3)
            return p2;
        return line_intersection(l);
    }
};
```

## 3.2 Point

```cpp
template <typename T>
struct point
{
    T x, y;
    point() {}
    point(const T &x, const T &y) : x(x), y(
         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);
    }
    point operator*(const T &b) const
    {
        return point(x * b, y * b);
    }
    point operator/(const T &b) const
    {
        return point(x / b, y / b);
    }
    bool operator==(const point &b) const
    {
        return x == b.x && y == b.y;
    }
    T dot(const point &b) const
    {
        return x * b.x + y * b.y;
    }
    T cross(const point &b) const
    {
        return x * b.y - y * b.x;
    }
    point normal() const
    { //求法向量
        return point(-y, x);
    }
    T abs2() const
    { //向量長度的平方
        return dot(*this);
    }
    T rad(const point &b) const
    { //兩向量的弧度
        return fabs(atan2(fabs(cross(b)),
             dot(b)));
    }
    T getA() const
    {                          //對x軸的弧度
        T A = atan2(y, x); //超過180度會變負
             的
        if (A <= -PI / 2)
            A += PI * 2;
        return A;
    }
};
```

## 3.3 Polygon

```cpp
template <typename T>
struct polygon
{
    polygon() {}
    vector<point<T>> p; //逆時針順序
    T area() const
    { //面積
        T ans = 0;
        for (int i = p.size() - 1, j = 0; j
             < (int)p.size(); i = j++)
            ans += p[i].cross(p[j]);
        return ans / 2;
    }
    point<T> center_of_mass() const
    { //重心
        T cx = 0, cy = 0, w = 0;
        for (int i = p.size() - 1, j = 0; j
             < (int)p.size(); i = j++)
        {
            T a = p[i].cross(p[j]);
            cx += (p[i].x + p[j].x) * a;
            cy += (p[i].y + p[j].y) * a;
            w += a;
        }
        return point<T>(cx / 3 / w, cy / 3 /
             w);
    }
    char ahas(const point<T> &t) const
    { //點是否在簡單多邊形內，是的話回傳1、
         在邊上回傳-1、否則回傳0
        bool c = 0;
        for (int i = 0, j = p.size() - 1; i
             < p.size(); j = i++)
            if (line<T>(p[i], p[j]).
                 point_on_segment(t))
                return -1;
            else if ((p[i].y > t.y) != (p[j
                 ].y > t.y) &&
                     t.x < (p[j].x - p[i].x)
                          * (t.y - p[i].y) /
                          (p[j].y - p[i].y)
                         + p[i].x)
                c = !c;
        return c;
    }
    char point_in_convex(const point<T> &x)
         const
    {
        int l = 1, r = (int)p.size() - 2;
        while (l <= r)
        { //點是否在凸多邊形內，是的話回傳1
             、在邊上回傳-1、否則回傳0
            int mid = (l + r) / 2;
            T a1 = (p[mid] - p[0]).cross(x -
                 p[0]);
            T a2 = (p[mid + 1] - p[0]).cross
                (x - p[0]);
            if (a1 >= 0 && a2 <= 0)
            {
                T res = (p[mid + 1] - p[mid
                     ]).cross(x - p[mid]);
                return res > 0 ? 1 : (res >=
                     0 ? -1 : 0);
            }
            else if (a1 < 0)
                r = mid - 1;
            else
                l = mid + 1;
        }
        return 0;
    }
    vector<T> getA() const
    {                        //凸包邊對x軸的夾角
        vector<T> res; //一定是遞增的
        for (size_t i = 0; i < p.size(); ++i
             )
            res.push_back((p[(i + 1) % p.
                 size()] - p[i]).getA());
        return res;
    }
    bool line_intersect(const vector<T> &A,
         const line<T> &l) const
    { //O(logN)
        int f1 = upper_bound(A.begin(), A.
             end(), (l.p1 - l.p2).getA()) - A
             .begin();
        int f2 = upper_bound(A.begin(), A.
             end(), (l.p2 - l.p1).getA()) - A
             .begin();
        return l.cross_seg(line<T>(p[f1], p[
             f2]));
    }
    polygon cut(const line<T> &l) const
    { //凸包對直線切割，得到直線l左側的凸包
        polygon ans;
        for (int n = p.size(), i = n - 1, j
             = 0; j < n; i = j++)
        {
            if (l.ori(p[i]) >= 0)
            {
                ans.p.push_back(p[i]);
                if (l.ori(p[j]) < 0)
                    ans.p.push_back(l.
                         line_intersection(
                         line<T>(p[i], p[j]))
                         );
            }
            else if (l.ori(p[j]) > 0)
                ans.p.push_back(l.
                     line_intersection(line<T
                     >(p[i], p[j])));
        }
        return ans;
    }
    static bool graham_cmp(const point<T> &a
         , const point<T> &b)
    { //凸包排序函數 // 起始點不同
        // return (a.x < b.x) || (a.x == b.x
             && a.y < b.y);  //最左下角開始
        return (a.y < b.y) || (a.y == b.y &&
             a.x < b.x);  //Y最小開始
    }
    void graham(vector<point<T>> &s)
    { //凸包 Convexhull 2D
        sort(s.begin(), s.end(), graham_cmp)
             ;
        p.resize(s.size() + 1);
        int m = 0;
        // cross >= 0 順時針。cross <= 0 逆
             時針旋轉
        for (size_t i = 0; i < s.size(); ++i
             )
        {
            while (m >= 2 && (p[m - 1] - p[m
                  - 2]).cross(s[i] - p[m -
                 2]) <= 0)
                --m;
            p[m++] = s[i];
        }
        for (int i = s.size() - 2, t = m +
             1; i >= 0; --i)
        {
            while (m >= t && (p[m - 1] - p[m
                  - 2]).cross(s[i] - p[m -
                 2]) <= 0)
                --m;
            p[m++] = s[i];
        }
        if (s.size() > 1) // 重複頭一次需扣
             掉
            --m;
        p.resize(m);
    }
    T diam()
    { //直徑
        int n = p.size(), t = 1;
        T ans = 0;
        p.push_back(p[0]);
        for (int i = 0; i < n; i++)
        {
            point<T> now = p[i + 1] - p[i];
            while (now.cross(p[t + 1] - p[i
                 ]) > now.cross(p[t] - p[i]))
                t = (t + 1) % n;
            ans = max(ans, (p[i] - p[t]).
                 abs2());
        }
        return p.pop_back(), ans;
    }
    T min_cover_rectangle()
    { //最小覆蓋矩形
        int n = p.size(), t = 1, r = 1, l;
        if (n < 3)
            return 0; //也可以做最小周長矩形
        T ans = 1e99;
        p.push_back(p[0]);
        for (int i = 0; i < n; i++)
        {
            point<T> now = p[i + 1] - p[i];
            while (now.cross(p[t + 1] - p[i
                 ]) > now.cross(p[t] - p[i]))
                t = (t + 1) % n;
            while (now.dot(p[r + 1] - p[i])
                 > now.dot(p[r] - p[i]))
                r = (r + 1) % n;
            if (!i)
                l = r;
            while (now.dot(p[l + 1] - p[i])
                 <= now.dot(p[l] - p[i]))
                l = (l + 1) % n;
            T d = now.abs2();
            T tmp = now.cross(p[t] - p[i]) *
                 (now.dot(p[r] - p[i]) - now
                 .dot(p[l] - p[i])) / d;
            ans = min(ans, tmp);
        }
        return p.pop_back(), ans;
    }
    T dis2(polygon &pl)
    { //凸包最近距離平方
        vector<point<T>> &P = p, &Q = pl.p;
        int n = P.size(), m = Q.size(), l = 0, r = 0;
        for (int i = 0; i < n; ++i)
            if (P[i].y < P[l].y)
                l = i;
        for (int i = 0; i < m; ++i)
            if (Q[i].y < Q[r].y)
                r = i;
        P.push_back(P[0]), Q.push_back(Q[0]);
        T ans = 1e99;
        for (int i = 0; i < n; ++i)
        {
            while ((P[l] - P[l + 1]).cross(Q[r + 1] - Q[r]) < 0)
                r = (r + 1) % m;
            ans = min(ans, line<T>(P[l], P[l + 1]).seg_dis2(line<T>(Q[r], Q[r + 1])));
            l = (l + 1) % n;
        }
        return P.pop_back(), Q.pop_back(), ans;
    }
    static char sign(const point<T> &t)
    {
        return (t.y == 0 ? t.x : t.y) < 0;
    }
    static bool angle_cmp(const line<T> &A, const line<T> &B)
    {
        point<T> a = A.p2 - A.p1, b = B.p2 - B.p1;
        return sign(a) < sign(b) || (sign(a) == sign(b) && a.cross(b) > 0);
    }
    int halfplane_intersection(vector<line<T>> &s)
    {

         //半平面交
        sort(s.begin(), s.end(), angle_cmp); //線段左側為該線段半平面
        int L, R, n = s.size();
        vector<point<T>> px(n);
        vector<line<T>> q(n);
        q[L = R = 0] = s[0];
        for (int i = 1; i < n; ++i)
        {
            while (L < R && s[i].ori(px[R - 1]) <= 0)
                --R;
            while (L < R && s[i].ori(px[L]) <= 0)
                ++L;
            q[++R] = s[i];
            if (q[R].parallel(q[R - 1]))
            {
                --R;
                if (q[R].ori(s[i].p1) > 0)
                    q[R] = s[i];
            }
            if (L < R)
                px[R - 1] = q[R - 1].line_intersection(q[R]);
        }
        while (L < R && q[L].ori(px[R - 1]) <= 0)
            --R;
        p.clear();
        if (R - L <= 1)
            return 0;
        px[R] = q[R].line_intersection(q[L]);
        for (int i = L; i <= R; ++i)
            p.push_back(px[i]);
        return R - L + 1;
    }
};
```

## 3.4 Triangle

```cpp
template <typename T>
struct triangle
{
    point<T> a, b, c;
    triangle() {}
    triangle(const point<T> &a, const point<T> &b, const point<T> &c) : a(a), b(b), c(c) {}
    T area() const
    {
        T t = (b - a).cross(c - a) / 2;
        return t > 0 ? t : -t;
    }
    point<T> barycenter() const
    { //重心
        return (a + b + c) / 3;
    }
    point<T> circumcenter() const
    { //外心
        static line<T> u, v;
        u.p1 = (a + b) / 2;
        u.p2 = point<T>(u.p1.x - a.y + b.y, u.p1.y + a.x - b.x);
        v.p1 = (a + c) / 2;
        v.p2 = point<T>(v.p1.x - a.y + c.y, v.p1.y + a.x - c.x);
        return u.line_intersection(v);
    }
    point<T> incenter() const
    { //內心
        T A = sqrt((b - c).abs2()), B = sqrt((a - c).abs2()), C = sqrt((a - b).abs2());
        return point<T>(A * a.x + B * b.x + C * c.x, A * a.y + B * b.y + C * c.y) / (A + B + C);
    }
    point<T> perpencenter() const
    { //垂心
        return barycenter() * 3 - circumcenter() * 2;
    }
};
```

# 4 Graph

## 4.1 Bellman-Ford

```cpp
/*SPA - Bellman-Ford*/
#include<bits/stdc++.h>
#define inf 99999 //define by you maximum edges weight
using namespace std;
vector<vector<int> > edges;
vector<int> dist;
vector<int> ancestor;
void BellmanFord(int start,int node){
    dist[start] = 0;
    for(int it = 0; it < node-1; it++){
        for(int i = 0; i < node; i++){
            for(int j = 0; j < node; j++){
                if(edges[i][j] != -1){
                    if(dist[i] + edges[i][j] < dist[j]){
                        dist[j] = dist[i] + edges[i][j];
                        ancestor[j] = i;
                    }
                }
            }
        }
    }

    for(int i = 0; i < node; i++)  //negative cycle detection
        for(int j = 0; j < node; j++)
            if(dist[i] + edges[i][j] < dist[j])
            {
                cout<<"Negative cycle!"<<endl;
                return;
            }
}
int main(){
    int node;
    cin>>node;
    edges.resize(node,vector<int>(node,inf));
    dist.resize(node,inf);
    ancestor.resize(node,-1);
    int a,b,d;
    while(cin>>a>>b>>d){
        /*input: source destination weight*/
        if(a == -1 && b == -1 && d == -1)
            break;
        edges[a][b] = d;
    }
    int start;
    cin>>start;
    BellmanFord(start,node);
    return 0;
}
```

## 4.2 BFS-queue

```cpp
/*BFS - queue version*/
#include<bits/stdc++.h>
using namespace std;
void BFS(vector<int> &result,vector<pair<int,int> > edges,int node,int start){
    vector<int> pass(node, 0);
    queue<int> q;
    queue<int> p;
    q.push(start);
    int count = 1;
    vector<pair<int, int>> newedges;
    while(!q.empty()){
        pass[q.front()] = 1;
        for (int i = 0; i < edges.size(); i++){
            if(edges[i].first == q.front() && pass[edges[i].second] == 0){
                p.push(edges[i].second);
                result[edges[i].second] = count;
            }
            else if(edges[i].second == q.front() && pass[edges[i].first] == 0){
                p.push(edges[i].first);
                result[edges[i].first] = count;
            }
            else
                newedges.push_back(edges[i]);
        }
        edges = newedges;
        newedges.clear();
        q.pop();
        if(q.empty() == true){
            q = p;
            queue<int> tmp;
            p = tmp;
            count++;
        }
    }
}
int main(){
    int node;
    cin >> node;
    vector<pair<int, int>> edges;
    int a, b;
    while(cin>>a>>b){
        /*a = b = -1 means input edges ended */
        if(a == -1 && b == -1)
            break;
        edges.push_back(pair<int, int>(a, b));
    }
    vector<int> result(node, -1);
    BFS(result, edges, node, 0);

    return 0;
}
```

## 4.3 DFS-rec

```
/*DFS - Recursive version*/
#include<bits/stdc++.h>
using namespace std;
map<pair<int,int>,int> edges;
vector<int> pass;
vector<int> route;
void DFS(int start){
    pass[start] = 1;
    map<pair<int,int>,int>::iterator iter;
    for(iter = edges.begin(); iter != edges.end(); iter++){
        if((*iter).first.first == start && (*iter).second == 0 && pass[(*iter).first.second] == 0){
            route.push_back((*iter).first.second);
            DFS((*iter).first.second);
        }
        else if((*iter).first.second == start && (*iter).second == 0 && pass[(*iter).first.first] == 0){
            route.push_back((*iter).first.first);
            DFS((*iter).first.first);
        }
    }
}
int main(){
    int node;
    cin>>node;
    pass.resize(node,0);
    int a,b;
    while(cin>>a>>b){
        if(a == -1 && b == -1)
            break;
        edges.insert(pair<pair<int,int>,int>(pair<int,int>(a,b),0));
    }
    int start;
    cin>>start;
    route.push_back(start);
    DFS(start);
    return 0;
}
```

## 4.4 Dijkstra

```
/*SPA - Dijkstra*/
#include<bits/stdc++.h>
#define inf INT_MAX
using namespace std;
vector<vector<int> > weight;
vector<int> ancestor;
vector<int> dist;
void dijkstra(int start){
    priority_queue<pair<int,int> ,vector<pair<int,int> > ,greater<pair<int,int> > > pq;
    pq.push(make_pair(0,start));
    while(!pq.empty()){
        int cur = pq.top().second;
        pq.pop();
        for(int i = 0; i < weight[cur].size(); i++){
            if(dist[i] > dist[cur] + weight[cur][i] && weight[cur][i] != -1){
                dist[i] = dist[cur] + weight[cur][i];
                ancestor[i] = cur;
                pq.push(make_pair(dist[i],i));
            }
        }
    }
}
int main(){
    int node;
    cin>>node;
    int a,b,d;
    weight.resize(node,vector<int>(node,-1));
    while(cin>>a>>b>>d){
        /*input: source destination weight*/
        if(a == -1 && b == -1 && d == -1)
            break;
        weight[a][b] = d;
    }
    ancestor.resize(node,-1);
    dist.resize(node,inf);
    int start;
    cin>>start;
    dist[start] = 0;
    dijkstra(start);
    return 0;
}
```

## 4.5 Floyd-warshall

```
/*SPA - Floyd-Warshall*/
#include<bits/stdc++.h>
#define inf 99999
using namespace std;
void floyd_warshall(vector<vector<int>>& distance, vector<vector<int>>& ancestor, int n){
    for (int k = 0; k < n; k++){
        for (int i = 0; i < n; i++){
            for (int j = 0; j < n; j++){
                if(distance[i][k] + distance[k][j] < distance[i][j]){
                    distance[i][j] = distance[i][k] + distance[k][j];
                    ancestor[i][j] = ancestor[k][j];
                }
            }
        }
    }
}
int main(){
    int n;
    cin >> n;
    int a, b, d;
    vector<vector<int>> distance(n, vector<int>(n,99999));
    vector<vector<int>> ancestor(n, vector<int>(n,-1));
    while(cin>>a>>b>>d){
        if(a == -1 && b == -1 && d == -1)
            break;
        distance[a][b] = d;
        ancestor[a][b] = a;
    }
    for (int i = 0; i < n; i++)
        distance[i][i] = 0;
    floyd_warshall(distance, ancestor, n);
    /*Negative cycle detection*/
    for (int i = 0; i < n; i++){
        if(distance[i][i] < 0){
            cout << "Negative cycle!" << endl;
            break;
        }
    }

    return 0;
}
```

## 4.6 union_find

```
int find(int x,vector<int> &union_set){
    if(union_set[x] != x)
        union_set[x] = find(union_set[x], union_set); //compress path
    return union_set[x];
}
void merge(int x,int y,vector<int> &union_set,vector<int> &rank){
    int rx, ry;
    rx = find(x,union_set);
    ry = find(y,union_set);
    if(rx == ry)
        return;
    /*merge by rank -> always merge small tree to big tree*/
    if(rank[rx] > rank[ry])
        union_set[ry] = rx;
    else
    {
        union_set[rx] = ry;
        if(rank[rx] == rank[ry])
            ++rank[ry];
    }
}
int main(){
    int node;
    cin >> node; //Input Node number
    vector<int> union_set(node, 0);
    vector<int> rank(node, 0);
    for (int i = 0; i < node; i++)
        union_set[i] = i;
    int edge;
    cin >> edge; //Input Edge number
    for(int i = 0; i < edge; i++)
    {
        int a, b;
        cin >> a >> b;
        merge(a, b, union_set,rank);
    }
    /*build party*/
    vector<vector<int> > party(node, vector<int>(0));
    for (int i = 0; i < node; i++)
        party[find(i, union_set)].push_back(i);
}
```

# 5 Mathematics

## 5.1 Combination

```
/*input type string or vector*/
for (int i = 0; i < (1 << input.size()); ++i)
{
    string testCase = "";
    for (int j = 0; j < input.size(); ++j)
        if (i & (1 << j))
            testCase += input[j];
}
```

## 5.2 Extended Euclidean

```
// ax + by = gcd(a,b)
pair<long long, long long> extgcd(long long a, long long b)
{
    if (b == 0)
        return {1, 0};
    long long k = a / b;
    pair<long long, long long> p = extgcd(b, a - k * b);
    //cout << p.first << " " << p.second << endl;
    //cout << "商數(k)=  " << k << endl << endl;
    return {p.second, p.first - k * p.second};
}

int main()
{
    int a, b;
    cin >> a >> b;
    pair<long long, long long> xy = extgcd(a, b); //(x0,y0)
    cout << xy.first << " " << xy.second << endl;
    cout << xy.first << " * " << a << " + " << xy.second << " * " << b << endl;
    return 0;
}
// ax + by = gcd(a,b) * r
/*find |x|+|y| -> min*/
int main()
{
    long long r, p, q; /*px+qy = r*/
    int cases;
    cin >> cases;
    while (cases--)
    {
        cin >> r >> p >> q;
        pair<long long, long long> xy =
             extgcd(q, p); //(x0,y0)
        long long ans = 0, tmp = 0;
        double k, k1;
        long long s, s1;
        k = 1 - (double)(r * xy.first) / p;
        s = round(k);
        ans = llabs(r * xy.first + s * p) +
             llabs(r * xy.second - s * q);
        k1 = -(double)(r * xy.first) / p;
        s1 = round(k1);
        /*cout << k << endl << k1 << endl;
            cout << s << endl << s1 << endl;
                 */
        tmp = llabs(r * xy.first + s1 * p) +
              llabs(r * xy.second - s1 * q);
        ans = min(ans, tmp);

        cout << ans << endl;
    }
    return 0;
}
```

## 5.3 Hex to Dec

```
int HextoDec(string num) //16 to 10
{
    int base = 1;
    int temp = 0;
    for (int i = num.length() - 1; i = 0; i
         --)
    {
        if (num[i] = '0' && num[i] = '9')
        {
            temp += (num[i] - 48) base;
            base = base 16;
        }
        else if (num[i] = 'A' && num[i] = 'F
             ')
        {
            temp += (num[i] - 55) base;
            base = base 16;
        }
    }
    return temp;
}
void DecToHex(int p_intValue) //10 to 16
{
    char l_pCharRes = new (char);
    sprintf(l_pCharRes, % X, p_intValue);
    int l_intResult = stoi(l_pCharRes);
    cout l_pCharRes n;
    return l_intResult;
}
```

## 5.4 Mod

```
int pow_mod(int a, int n, int m)
{
    a ^ n mod m;
    a, n, m < 10 ^ 9 if (n == 0) return 1;
    int x = pow_mid(a, n / 2, m);
    long long ans = (long long)x * x % m;
    if (n % 2 == 1)
        ans = ans * a % m;
    return (int)ans;
}

// 加法：(a + b) % p = (a % p + b % p) % p;
// 減法：(a – b) % p = (a % p – b % p + p) %
     p;
// 乘法：(a □ b) % p = (a % p * b % p) % p;
// 次方：(a ^ b) % p = ((a % p) ^ b) % p;
// 加法結合律：((a + b) % p + c) % p = (a +
     (b + c)) % p;
// 乘法結合律：((a * b) % p * c) % p = (a *
     (b * c)) % p;
// 加法交換律：(a + b) % p = (b + a) % p;
// 乘法交換律：(a * b) % p = (b * a) % p;
// 結合律：((a + b) % p * c) = ((a * c) % p
     + (b * c) % p) % p;
```

## 5.5 Permutation

```
// 全排列要先 sort !!!
// num -> vector or string
next_permutation(num.begin(), num.end());
prev_permutation(num.begin(), num.end());
```

## 5.6 PI

```
#define PI acos(-1)
#define PI M_PI
const double PI = atan2(0.0, -1.0);
```

## 5.7 Prime table

```
// 埃拉托斯特尼篩法
const int maxn = 10000000;
bitset<maxn> is_not_prime; // false 是質數
void sieve()
{
    is_not_prime[0] = is_not_prime[1] = 1;
    for (int i = 2; i * i < maxn; ++i)
    {
        if (is_not_prime[i] == 0)
        {
            for (int j = i * i; j < maxn; j
                 += i)
                is_not_prime[j] = 1;
        }
    }
}
```

## 5.8 二分逼近法

```
#define eps 1e-14
void half_interval()
{
    double L = 0, R = /*區間*/, M;
    while (R - L >= eps)
    {
        M = (R + L) / 2;
        if (/*函數*/ > /*方程式目標*/)
            L = M;
        else
            R = M;
    }
    printf("%.3lf\n", R);
}
```

## 5.9 四則運算

```
string s = ""; //開頭是負號要補0
long long int DFS(int le, int ri) // (0,
     string final index)
{
    int c = 0;
    for (int i = ri; i >= le; i--)
    {
        if (s[i] == ')')
            c++;
        if (s[i] == '(')
            c--;
        if (s[i] == '+' && c == 0)
            return DFS(le, i - 1) + DFS(i +
                 1, ri);
        if (s[i] == '-' && c == 0)
            return DFS(le, i - 1) - DFS(i +
                 1, ri);
    }
    for (int i = ri; i >= le; i--)
    {
        if (s[i] == ')')
            c++;
        if (s[i] == '(')
            c--;
        if (s[i] == '*' && c == 0)
            return DFS(le, i - 1) * DFS(i +
                 1, ri);
        if (s[i] == '/' && c == 0)
            return DFS(le, i - 1) / DFS(i +
                 1, ri);
        if (s[i] == '%' && c == 0)
            return DFS(le, i - 1) % DFS(i +
                 1, ri);
    }
    if ((s[le] == '(') && (s[ri] == ')'))
        return DFS(le + 1, ri - 1); //去除刮
             號
    if (s[le] == ' ' && s[ri] == ' ')
        return DFS(le + 1, ri - 1); //去除左
             右兩邊空格
    if (s[le] == ' ')
        return DFS(le + 1, ri); //去除左邊空
             格
    if (s[ri] == ' ')
        return DFS(le, ri - 1); //去除右邊空
             格
    long long int num = 0;
    for (int i = le; i <= ri; i++)
        num = num * 10 + s[i] - '0';
    return num;
}
```

## 5.10 數字乘法組合

```
void toans(vector<vector<int>> &ans, vector<
     int> com)
{
    // sort(com.begin(), com.end());
    ans.push_back(com);
    // for (auto i : com)
    //     cout << i << ' ';
    // cout << endl;
}
void finds(int j, int old, int num, vector<
     int> com, vector<vector<int>> &ans)
{
    for (int i = j; i <= sqrt(num); i++)
    {
        if (old == num)
            com.clear();
        if (num % i == 0)
        {
            vector<int> a;
            a = com;
            a.push_back(i);
            finds(i, old, num / i, a, ans);
            a.push_back(num / i);
            toans(ans, a);
        }
    }
}
int main()
{
    vector<vector<int>> ans;
    vector<int> zero;
    finds(2, num, num, zero, ans);
    // num 為 input 數字
    for (int i = 0; i < ans.size(); i++)
    {
        for (int j = 0; j < ans[i].size() -
             1; j++)
            cout << ans[i][j] << " ";
        cout << ans[i][ans[i].size() - 1] <<
             endl;
    }
}
```

## 5.11 數字加法組合

```
void printCombination(vector<int> const &out
    , int m, vector<vector<int>> &ans)
{
    for (int i : out)
        if (i > m)
            return;
    ans.push_back(out);
}

void recur(int i, int n, int m, vector<int>
    &out, vector<vector<int>> &ans)
{
    if (n == 0)
        printCombination(out, m, ans);
    for (int j = i; j <= n; j++)
    {
        out.push_back(j);
        recur(j, n - j, m, out, ans);
        out.pop_back();
    }
}
int main()
{
    vector<vector<int>> ans;
    vector<int> zero;
    recur(1, num, num, zero, ans);
    // num 為 input 數字
    for (int i = 0; i < ans.size(); i++)
    {
        for (int j = 0; j < ans[i].size() -
             1; j++)
            cout << ans[i][j] << " ";
        cout << ans[i][ans[i].size() - 1] <<
             endl;
    }
}
```

## 5.12 羅馬數字

```
int romanToInt(string s)
{
    unordered_map<char, int> T;
    T['I'] = 1;
    T['V'] = 5;
    T['X'] = 10;
    T['L'] = 50;
    T['C'] = 100;
    T['D'] = 500;
    T['M'] = 1000;

    int sum = T[s.back()];
    for (int i = s.length() - 2; i >= 0; --i
        )
    {
        if (T[s[i]] < T[s[i + 1]])
            sum -= T[s[i]];
        else
            sum += T[s[i]];
    }
    return sum;
}
```

## 5.13 質因數分解

```
void primeFactorization(int n) // 配合質數表
{
    for (int i = 0; i < (int)p.size(); ++i)
    {
        if (p[i] * p[i] > n)
            break;
        if (n % p[i])
            continue;
        cout << p[i] << ' ';
        while (n % p[i] == 0)
        {
            n /= p[i];
        }
    }
    if (n != 1)
    {
        cout << n << ' ';
    }
    cout << '\n';
}
```

# 6 Other

## 6.1 Weighted Job Scheduling

```
struct Job
{
    int start, finish, profit;
};
bool jobComparataor(Job s1, Job s2)
{
    return (s1.finish < s2.finish);
}
int latestNonConflict(Job arr[], int i)
{
    for (int j = i - 1; j >= 0; j--)
    {
        if (arr[j].finish <= arr[i].start)
            return j;
    }
    return -1;
}
int findMaxProfit(Job arr[], int n)
{
    sort(arr, arr + n, jobComparataor);
    int *table = new int[n];
    table[0] = arr[0].profit;
    for (int i = 1; i < n; i++)
    {
        int inclProf = arr[i].profit;
        int l = latestNonConflict(arr, i);
        if (l != -1)
            inclProf += table[l];
        table[i] = max(inclProf, table[i -
            1]);
    }
    int result = table[n - 1];
    delete[] table;

    return result;
}
```

## 6.2 數獨解法

```
int getSquareIndex(int row, int column, int
     n)
{
    return row / n * n + column / n;
}

bool backtracking(vector<vector<int>> &board
    , vector<vector<bool>> &rows, vector<
    vector<bool>> &cols,
                  vector<vector<bool>> &boxs
                      , int index, int n)
{
    int n2 = n * n;
    int rowNum = index / n2, colNum = index
         % n2;
    if (index >= n2 * n2)
        return true;

    if (board[rowNum][colNum] != 0)
        return backtracking(board, rows,
             cols, boxs, index + 1, n);

    for (int i = 1; i <= n2; i++)
    {
        if (!rows[rowNum][i] && !cols[colNum
            ][i] && !boxs[getSquareIndex(
            rowNum, colNum, n)][i])
        {
            rows[rowNum][i] = true;
            cols[colNum][i] = true;
            boxs[getSquareIndex(rowNum,
                 colNum, n)][i] = true;
            board[rowNum][colNum] = i;
            if (backtracking(board, rows,
                 cols, boxs, index + 1, n))
                return true;
            board[rowNum][colNum] = 0;
            rows[rowNum][i] = false;
            cols[colNum][i] = false;
            boxs[getSquareIndex(rowNum,
                 colNum, n)][i] = false;
        }
    }
    return false;
}
/*用法 main*/
int n = sqrt(數獨邊長大小) /*e.g. 9*9 n=3*/
vector<vector<int>> board(n * n + 1, vector<
     int>(n * n + 1, 0));
vector<vector<bool>> isRow(n * n + 1, vector
     <bool>(n * n + 1, false));
vector<vector<bool>> isColumn(n * n + 1,
     vector<bool>(n * n + 1, false));
vector<vector<bool>> isSquare(n * n + 1,
     vector<bool>(n * n + 1, false));

for (int i = 0; i < n * n; ++i)
{
    for (int j = 0; j < n * n; ++j)
    {
        int number;
        cin >> number;
        board[i][j] = number;
        if (number == 0)
            continue;
        isRow[i][number] = true;
        isColumn[j][number] = true;
        isSquare[getSquareIndex(i, j, n)][
             number] = true;
    }
}
if (backtracking(board, isRow, isColumn,
     isSquare, 0, n))
    /*有解答*/
else
    /*解答*/
```

# 7 String

## 7.1 sliding window

```
string minWindow(string s, string t) {
    unordered_map<char, int> letterCnt;
    for (int i = 0; i < t.length(); i++)
        letterCnt[t[i]]++;
    int minLength = INT_MAX, minStart = -1;
    int left = 0, matchCnt = 0;
    for (int i = 0; i < s.length(); i++)
    {
        if (--letterCnt[s[i]] >= 0)
            matchCnt++;
        while (matchCnt == t.length())
        {
            if (i - left + 1 < minLength)
            {
                minLength = i - left + 1;
                minStart = left;
            }
            if (++letterCnt[s[left]] > 0)
                matchCnt--;
            left++;
        }
    }
```

```cpp
    return minLength == INT_MAX ? "" : s.substr(minStart, minLength);
}
```

## 7.2 split

```cpp
vector<string> mysplit(const string& str, const string& delim)
{
    vector<string> res;
    if ("" == str)
        return res;

    char *strs = new char[str.length() + 1];
    strcpy(strs, str.c_str());

    char *d = new char[delim.length() + 1];
    strcpy(d, delim.c_str());

    char *p = strtok(strs, d);
    while (p)
    {
        string s = p;
        res.push_back(s);
        p = strtok(NULL, d);
    }
    return res;
}
```

# 8 data structure

## 8.1 Bigint

```cpp
//台大
struct Bigint{
    static const int LEN = 60;
    static const int BIGMOD = 10000;
    int s;
    int vl, v[LEN];
    //  vector<int> v;
    Bigint() : s(1) { vl = 0; }
    Bigint(long long a) {
        s = 1; vl = 0;
        if (a < 0) { s = -1; a = -a; }
        while (a) {
            push_back(a % BIGMOD);
            a /= BIGMOD;
        }
    }
    Bigint(string str) {
        s = 1; vl = 0;
        int stPos = 0, num = 0;
        if (!str.empty() && str[0] == '-') {
            stPos = 1;
            s = -1;
        }
        for (int i= str.length() - 1, q=1; i>=stPos; i--) {
            num += (str[i] - '0') * q;
            if ((q *= 10) >= BIGMOD) {
                push_back(num);
                num = 0; q = 1;
            }
        }
        if (num) push_back(num);
        n();
    }
    int len() const {
        return vl;//return SZ(v);
    }
    bool empty() const { return len() == 0; }
    void push_back(int x) {
        v[vl++] = x; //v.PB(x);
    }
    void pop_back() {
        vl--; //v.pop_back();
    }
    int back() const {
        return v[vl-1]; //return v.back();
    }
    void n() {
        while (!empty() && !back()) pop_back();
    }
    void resize(int nl) {
        vl = nl; //v.resize(nl);
        fill(v, v+vl, 0); //fill(ALL(v), 0);
    }
    void print() const {
        if (empty()) { putchar('0'); return; }
        if (s == -1) putchar('-');
        printf("%d", back());
        for (int i=len()-2; i>=0; i--)
            printf("%.4d",v[i]);
    }
    friend std::ostream& operator << (std::ostream& out, const Bigint &a) {
        if (a.empty()) { out << "0"; return out; }
        if (a.s == -1) out << "-";
        out << a.back();
        for (int i=a.len()-2; i>=0; i--) {
            char str[10];
            snprintf(str, 5, "%.4d", a.v[i]);
            out << str;
        }
        return out;
    }
    int cp3(const Bigint &b)const {
        if (s != b.s) return s - b.s;
        if (s == -1) return -(-*this).cp3(-b);
        if (len() != b.len()) return len()-b.len();//int
        for (int i=len()-1; i>=0; i--)
            if (v[i]!=b.v[i]) return v[i]-b.v[i];
        return 0;
    }
    bool operator<(const Bigint &b)const
    { return cp3(b)<0; }
    bool operator<=(const Bigint &b)const
    { return cp3(b)<=0; }
    bool operator==(const Bigint &b)const
    { return cp3(b)==0; }
    bool operator!=(const Bigint &b)const
    { return cp3(b)!=0; }
    bool operator>(const Bigint &b)const
    { return cp3(b)>0; }
    bool operator>=(const Bigint &b)const
    { return cp3(b)>=0; }
    Bigint operator - () const {
        Bigint r = (*this);
        r.s = -r.s;
        return r;
    }
    Bigint operator + (const Bigint &b) const {
        if (s == -1) return -(-(*this)+(-b));
        if (b.s == -1) return (*this)-(-b);
        Bigint r;
        int nl = max(len(), b.len());
        r.resize(nl + 1);
        for (int i=0; i<nl; i++) {
            if (i < len()) r.v[i] += v[i];
            if (i < b.len()) r.v[i] += b.v[i];
            if(r.v[i] >= BIGMOD) {
                r.v[i+1] += r.v[i] / BIGMOD;
                r.v[i] %= BIGMOD;
            }
        }
        r.n();
        return r;
    }
    Bigint operator - (const Bigint &b) const {
        if (s == -1) return -(-(*this)-(-b));
        if (b.s == -1) return (*this)+(-b);
        if ((*this) < b) return -(b-(*this));
        Bigint r;
        r.resize(len());
        for (int i=0; i<len(); i++) {
            r.v[i] += v[i];
            if (i < b.len()) r.v[i] -= b.v[i];
            if (r.v[i] < 0) {
                r.v[i] += BIGMOD;
                r.v[i+1]--;
            }
        }
        r.n();
        return r;
    }
    Bigint operator * (const Bigint &b) {
        Bigint r;
        r.resize(len() + b.len() + 1);
        r.s = s * b.s;
        for (int i=0; i<len(); i++) {
            for (int j=0; j<b.len(); j++) {
                r.v[i+j] += v[i] * b.v[j];
                if(r.v[i+j] >= BIGMOD) {
                    r.v[i+j+1] += r.v[i+j] / BIGMOD;
                    r.v[i+j] %= BIGMOD;
                }
            }
        }
        r.n();
        return r;
    }
    Bigint operator / (const Bigint &b) {
        Bigint r;
        r.resize(max(1, len()-b.len()+1));
        int oriS = s;
        Bigint b2 = b; // b2 = abs(b)
        s = b2.s = r.s = 1;
        for (int i=r.len()-1; i>=0; i--) {
            int d=0, u=BIGMOD-1;
            while(d<u) {
                int m = (d+u+1)>>1;
                r.v[i] = m;
                if((r*b2) > (*this)) u = m-1;
                else d = m;
            }
            r.v[i] = d;
        }
        s = oriS;
        r.s = s * b.s;
        r.n();
        return r;
    }
    Bigint operator % (const Bigint &b) {
        return (*this)-(*this)/b*b;
    }
};
```

## 8.2 Trie

```cpp
// biginter字典數
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;
    }
};

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;
}

struct Trie{
    int c[5000005][10];
    int val[5000005];
    int sz;
    int getIndex(char c){
        return c - '0';
    }
    void init(){
        memset(c[0], 0, sizeof(c[0]));
        memset(val, -1, sizeof(val));
        sz = 1;
    }

    void insert(BigInteger x, int v){
        int u = 0;
        int max_len_count = 0;
        int firstNum = x.s.back();
        char firstBuf[20];
        sprintf(firstBuf, "%d", firstNum);
        for(int j = 0; j < strlen(firstBuf); j++){
            int index = getIndex(firstBuf[j]);
            if(!c[u][index]){
                memset(c[sz], 0 , sizeof(c[sz]));
                val[sz] = v;
                c[u][index] = sz++;
            }
            u = c[u][index];
            max_len_count++;
        }

        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) && max_len_count < 50;j++){
                int index = getIndex(buf[j]);
                if(!c[u][index]){
                    memset(c[sz], 0 , sizeof(c[sz]));
                    val[sz] = v;
                    c[u][index] = sz++;
                }
                u = c[u][index];
                max_len_count++;
            }
            if(max_len_count >= 50){
                break;
            }
        }
    }

    int find(const char* s){
        int u = 0;
        int n = strlen(s);
        for(int i = 0 ; i < n;++i)
        {
            int index = getIndex(s[i]);
            if(!c[u][index]){
                return -1;
            }
            u = c[u][index];
        }
        return val[u];
    }
}
```

## 8.3 分數

```cpp
class Rational
{
  friend istream &operator>>(istream &, Rational & );
  friend ostream &operator<<(ostream &, const Rational & );
public:
  Rational()  //constructor one
  {
    m_numeitor = 0;
    m_denominator = 1;
  }
  Rational(int a, int b)  //constructor two
  {
    if (b < 0 || b == 0)  //avoids negative denominators. && prevents a 0 denominator
    {
      cout << "This Rational number can't be used.\n\n";
      m_numeitor = 0;
      m_denominator = 0;
    }
    else
    {
      cout << "This Rational number can be used.\n\n";
      m_numeitor = a;
      m_denominator = b;
    }
  }
  Rational operator+(const Rational& a);  //加
  Rational operator-(const Rational& a);  //減
  Rational operator*(const Rational& a);  //乘
  Rational operator/(const Rational& a);  //除
  bool operator==(const Rational& a);   //相等
  void reduce();  //化簡
private:
  int m_numeitor;
  int m_denominator;
};
istream &operator>>(istream &input, Rational &test )
{
  char temp;

  input >> test.m_numeitor;
  input >> temp;
  input >> test.m_denominator;
  Rational final(test.m_numeitor, test.m_denominator);  //final用來告訴使用者這數字符不符合!
  if (test.m_denominator < 0 || test.m_denominator == 0)   //不符合(再輸入一次)
  {
    while (test.m_denominator < 0 || test.m_denominator == 0)   //有可能輸入的東西還是不符合,所以用迴圈
    {
      cout << "Enter another Rational number (n/d): ";
      input >> test.m_numeitor;
      input >> temp;
      input >> test.m_denominator;
      Rational final(test.m_numeitor, test.m_denominator);  //final用來告訴使用者這數字符不符合!
    }
    return input;
  }
  else
    return input;
}
ostream &operator<<(ostream &output, const Rational &test )
{
  output << test.m_numeitor;
  if(test.m_numeitor == 0)
    return output;
  if (test.m_denominator == 1)
    return output;
  else
  {
    output << "/";
    output << test.m_denominator;
  }
  return output;
}
Rational Rational::operator+(const Rational& a)
{
  Rational c;
  c.m_denominator = this->m_denominator * a.m_denominator;  //通分(同乘)
  c.m_numeitor = (this->m_numeitor * a.m_denominator) + (a.m_numeitor * this->m_denominator);
  c.reduce();
  return c;
}
Rational Rational::operator-(const Rational& a)
{
  Rational c;
  c.m_denominator = this->m_denominator * a.m_denominator;
  c.m_numeitor = (this->m_numeitor * a.m_denominator) - (a.m_numeitor * this->m_denominator);
  c.reduce();
  return c;
}
Rational Rational::operator*(const Rational& a)
{
  Rational c;
  c.m_denominator = this->m_denominator * a.m_denominator;
  c.m_numeitor = this->m_numeitor * a.m_numeitor;
  c.reduce();
  return c;
}
Rational Rational::operator/(const Rational& a)
{
  Rational c;
  c.m_denominator = this->m_denominator * a.m_numeitor;
  c.m_numeitor = this->m_numeitor * a.m_denominator;
  c.reduce();
  return c;
}
bool Rational::operator==(const Rational& a)
{
  if (m_numeitor == a.m_numeitor)
  {
    if (m_denominator == a.m_denominator)
      return true;
    else
      return false;
  }
  else
    return false;
}
void Rational::reduce()
{
  int i;
  int max;
  if(m_numeitor> m_denominator)
    max = m_numeitor;
  else
    max = m_denominator;
  for (i = 2; i <= max; i++)
  {
    if (m_denominator % i == 0 && m_numeitor
          % i == 0)
    {
      m_denominator /= i;
      m_numeitor /= i;
      i = 1;
      max = m_denominator;
      continue;
    }
  }
}
```

# To do writing not thinking

## Contents