TheChosenUAn's
UDLA

Jaider Bautista

11 de octubre de 2024

Índice

1. DataStructure **1**

1.1. indexed set 1
1.2. Mint 1
1.3. Dsu 2
1.4. STable 2
1.5. Fenwick Tree 2
1.6. Segment Tree 2D 3
1.7. Segment Tree Iterative 3
1.8. Segment Tree Lazy 4
1.9. Segment Tree 4
1.10. Mo's 5

2. DP **5**

2.1. Knapsack 5
2.2. Divide and Conquer dp 6
2.3. Edit Distance 6
2.4. groups 7
2.5. Shortest Hamiltonian Path 8
2.6. Money Sums 8
2.7. Digit dp 8
2.8. LCS 9
2.9. LIS 10
2.10. subsequences 11

3. Geometry **12**

3.1. isfigure 12

4. Graph **13**

4.1. bipartite Graph 13
4.2. Floyd Warshall 14
4.3. nx-ny-4 14
4.4. nx-ny-8 14

5. Math **15**

5.1. Propiedades del modulo 15
5.2. TernarySearch 15
5.3. squares in a circle 15

5.4. Pow . . . 16
5.5. dectobin . . . 16
5.6. fraction . . . 16
5.7. nCK . . . 17
5.8. divisores . . . 17

6. Problems 17

6.1. dp+nck . . . 17
6.2. F Less Than G . . . 19
6.3. Nested Circles . . . 19
6.4. Restoring the Expression . . . 20
6.5. matrixexp . . . 21
6.6. Dueling Digits . . . 22

7. String 23

7.1. SThash . . . 23
7.2. HuffmanCoding . . . 23
7.3. palindrome range . . . 24
7.4. Z . . . 25
7.5. ahobit . . . 25
7.6. paltree . . . 26
7.7. hashing . . . 26
7.8. hash table . . . 27
7.9. suffixAutomaton . . . 27
7.10. Manacher . . . 29
7.11. trie . . . 31
7.12. Kmp . . . 31
7.13. suffixAutomaton1 . . . 32
7.14. Min-Max-SuffixCyclic . . . 33
7.15. hashing-mint . . . 34
7.16. Splitear . . . 34
7.17. Suffix Array 1 . . . 34

8. Utilities 35

8.1. util . . . 35
8.2. cmd . . . 36
8.3. template . . . 36
8.4. Pragma . . . 36
8.5. segment whit the maximum sum . . . 37
8.6. random . . . 37

1. DataStructure

1.1. indexed set

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template<class T> using T_set = tree<T, null_type, less<T>,
    rb_tree_tag, tree_order_statistics_node_update>;
template<class L> using T_multiset = tree<L, null_type, less_equal<L>,
    rb_tree_tag, tree_order_statistics_node_update>;
T_set<int> st;
st.insert(1);
st.insert(3);
st.insert(4);
st.insert(10);

The function find_by_order returns an iterator to the element at a
    given position
auto it = *st.find_by_order(1); it = {3}

the function order_of_key returns the position of a given element
```

```
int pos = st.order_of_key(4); pos = 2

If the element does not appear in the set, we get the position that the
element would have in the set
int pos = st.order_of_key(6); pos2 = 3
int pos = st.order_of_key(2); pos2 = 1

st.erase_if([](int x) {return x == 2 || x == 10;});
```

1.2. Mint

```
static constexpr int mod = 998244353;

struct mint {
    static constexpr int  m = 998244353;
    // static inline int  m = 998244353; //to change mod
    int x;
    mint() : x(0) {}
    mint(long long x_) :x(x_% m) { if (x < 0) x += m; }
    int val() { return x; }
    mint& operator+=(mint b) { if ((x += b.x) >= m) x -= m; return
        *this; }
    mint& operator-=(mint b) { if ((x -= b.x) < 0) x += m; return
        *this; }
    mint& operator*=(mint b) { x = (long long)(x)*b.x % m; return
        *this; }
    mint pow(long long e) const {
        mint r = 1, b = *this;
        while (e) {
            if (e & 1) r *= b;
            b *= b;
            e >>= 1;
        }
        return r;
    }
    mint inv() { return pow(m - 2); }
    mint& operator/=(mint b) { return *this *= b.pow(m - 2); }
    friend mint operator+(mint a, mint b) { return a += b; }
    friend mint operator-(mint a, mint b) { return a -= b; }
    friend mint operator/(mint a, mint b) { return a /= b; }
    friend mint operator*(mint a, mint b) { return a *= b; }
    friend bool operator<(mint a, mint b) { return a.x < b.x; }
    friend bool operator==(mint a, mint b) { return a.x == b.x; }
    friend bool operator!=(mint a, mint b) { return a.x != b.x; }
};
```

1.3. Dsu

```
struct dsu {
    vector<int> pad, tam;
    int size;

    dsu(int n) : pad(n), tam(n, 1), size(n) {
        iota(all(pad), 0);
    }

    void make() {
        pad.pb(sz(pad));
        tam.pb(1);
        size++;
    }

    int find(int v) {
        if (v == pad[v]) return v;
        return pad[v] = find(pad[v]);
    }

    void unite(int a, int b) {
        a = find(a);
        b = find(b);
        if (a != b) {
            if (tam[a] < tam[b]) swap(a, b);
            pad[b] = a;
            tam[a] += tam[b];
            size--;
        }
    }

    int same(int a, int b) {
        return find(a) == find(b);
    }

    int count(int v) {
        return tam[find(v)];
    }
};
```

1.4. STable

```
struct STable {
    int n, K;
    vector<vector<int>> st;

    STable(const vector<int>& a) {
        n = sz(a);
        K = int(log2(n)) + 1;
        st.assign(n + 1, vector<int>(K));
        forn(i, n) st[i][0] = a[i];
        forn(j, K - 1)
            for (int i = 0; i + (1 << (j + 1)) <= n; ++i)
                st[i][j + 1] = oper(st[i][j], st[i + (1 << j)][j]);
    }

    int oper(int a, int b) { return __gcd(a, b); }

    int query(int l, int r) {
        int k = 31 - __builtin_clz(r - l + 1);
        return oper(st[l][k], st[r - (1 << k) + 1][k]);
    }
};
```

1.5. Fenwick Tree

```
template<typename T>
struct BIT {
    vector<T> ft;
    BIT(int n) : ft(n + 1) {}
    BIT(const vector<T>& a) : ft(sz(a) + 1) {
        forn(i, sz(a)) { upd(i + 1, a[i]); }
    }

    T qry(int i) {
        T ans = 0;
        for (; i; i -= i & -i) ans += ft[i];
        return ans;
    }

    T qry(int l, int r) { return qry(r) - qry(l - 1); }

    void upd(int i, T v) {
        for (; i < sz(ft); i += i & -i) ft[i] += v;
    }
};
```

1.6. Segment Tree 2D

```
template<typename T>
struct STree {
    int n, m;
    T neutro = T(0);
    vector<vector<T>> st;

    STree(vector<vector<T>>& a) {
        n = sz(a);
        m = sz(a[0]);
        st = vector<vector<T>>(2 * n, vector<T>(2 * m, neutro));
        build(a);
    }

    inline T oper(T a, T b) { return a + b; }

    void build(vector<vector<T>>& a) {
        forn(i, n) forn(j, m) st[i + n][j + m] = a[i][j];
        forn(i, n) {
            for (int j = m - 1; j >= 1; --j) {
                st[i + n][j] = oper(st[i + n][j << 1], st[i + n][j << 1
                    | 1]);
            }
        }
        for (int i = n - 1; i >= 1; --i) {
            forn(j, 2 * m) {
                st[i][j] = oper(st[i << 1][j], st[i << 1 | 1][j]);
            }
        }
    }

    T qry(int x1, int y1, int x2, int y2) { // [x1, y1] [x2, y2]
        T ans = neutro;
        for (int i0 = x1 + n, i1 = x2 + n + 1; i0 < i1; i0 >>= 1, i1
            >>= 1) {
            int t[4], q = 0;
            if (i0 & 1) t[q++] = i0++;
            if (i1 & 1) t[q++] = --i1;
            forn(k, q)
                for (int j0 = y1 + m, j1 = y2 + m + 1; j0 < j1; j0 >>=
                    1, j1 >>= 1) {
                    if (j0 & 1) ans = oper(ans, st[t[k]][j0++]);
                    if (j1 & 1) ans = oper(ans, st[t[k]][--j1]);
                }
        }
        return ans;
    }

    void upd(int l, int r, T val) {
        st[l + n][r + m] = val;
        for (int j = r + m; j > 1; j >>= 1) {
            st[l + n][j >> 1] = oper(st[l + n][j], st[l + n][j ^ 1]);
        }
        for (int i = l + n; i > 1; i >>= 1) {
            for (int j = r + m; j; j >>= 1) {
                st[i >> 1][j] = oper(st[i][j], st[i ^ 1][j]);
            }
        }
    }
};
```

1.7. Segment Tree Iterative

```
template<typename T>
struct STree {
    vector<T> st;
    int n;
    T neutro = T(0);
    T oper(T a, T b) { return a + b; }
    STree(vector<T>& a) {
        n = sz(a);
        st.resize(n * 2);
        forn(i, n) st[n + i] = a[i];
        for (int i = n - 1; i >= 1; i -= 1) st[i] = oper(st[i << 1],
            st[i << 1 | 1]);
    }

    void upd(int p, T val) {
        for (st[p += n] = val; p > 1; p >>= 1) st[p >> 1] =
            oper(st[p], st[p ^ 1]);
    }

    T query(int l, int r) { //[l, r)
        T v = neutro;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
            if (l & 1) v = oper(v, st[l++]);
            if (r & 1) v = oper(v, st[--r]);
        }
        return v;
    }
};
```

1.8. Segment Tree Lazy

```
template<typename T>
struct STree {
    int n; vector<T> st, lazy;
    T neutro = T(0);

    STree(int m) {
        n = m;
        st.resize(n * 4);
        lazy.resize(n * 4);
    }

    STree(vector<T>& a) {
        n = sz(a);
        st.resize(n * 4);
        lazy.resize(n * 4);
        build(1, 0, n - 1, a);
    }

    T oper(T a, T b) { return a + b; }

    void build(int v, int tl, int tr, vector<T>& a) {
        if (tl == tr) {
            st[v] = a[tl];
            return;
        }
        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, a);
        build(v * 2 + 1, tm + 1, tr, a);
        st[v] = oper(st[v * 2], st[v * 2 + 1]);
    }

    void push(int v, int tl, int tr) {
        if (!lazy[v]) return;
        st[v] += (tr - tl + 1) * lazy[v];
        if (tl != tr) {
            lazy[v * 2] += lazy[v];
            lazy[v * 2 + 1] += lazy[v];
        }
        lazy[v] = 0;
    }

    void upd(int v, int tl, int tr, int l, int r, T val) {
        push(v, tl, tr);
        if (tr < l || tl > r) return;
        if (tl >= l && tr <= r) {
            lazy[v] = val;
            push(v, tl, tr);
            return;
        }
        int tm = (tl + tr) / 2;
        upd(v * 2, tl, tm, l, r, val);
        upd(v * 2 + 1, tm + 1, tr, l, r, val);
        st[v] = oper(st[v * 2], st[v * 2 + 1]);
    }

    T query(int v, int tl, int tr, int l, int r) {
        push(v, tl, tr);
        if (tl > r || tr < l) return neutro;
        if (l <= tl && tr <= r) return st[v];
        int tm = (tl + tr) / 2;
        return oper(query(v * 2, tl, tm, l, r), query(v * 2 + 1, tm + 1
            , tr, l, r));
    }

    void upd(int l, int r, T val) { upd(1, 0, n - 1, l, r, val); }
    T query(int l, int r) { return query(1, 0, n - 1, l, r); }
};
```

1.9. Segment Tree

```
template<typename T>
struct STree {
    int n; vector<T> st;
    T neutro = T(0);

    STree(vector<T>& a) {
        n = sz(a);
        st.resize(n * 4);
        build(1, 0, n - 1, a);
    }

    T oper(T a, T b) { return max(a, b); }

    void build(int v, int tl, int tr, vector<T>& a) {
        if (tl == tr) {
            st[v] = a[tl];
            return;
        }
        int tm = (tr + tl) / 2;
        build(v * 2, tl, tm, a);
        build(v * 2 + 1, tm + 1, tr, a);
        st[v] = oper(st[v * 2], st[v * 2 + 1]);
    }

    T query(int v, int tl, int tr, int l, int r) {
        if (tl > r || tr < l) return neutro;
        if (l <= tl && tr <= r) return st[v];
        int tm = (tl + tr) / 2;
        return oper(query(v * 2, tl, tm, l, r), query(v * 2 + 1, tm + 1
            , tr, l, r));
    }

    void upd(int v, int tl, int tr, int pos, T val) {
        if (tl == tr) {
            st[v] = val;
            return;
        }
        int tm = (tr + tl) / 2;
        if (pos <= tm) upd(v * 2, tl, tm, pos, val);
        else upd(v * 2 + 1, tm + 1, tr, pos, val);
        st[v] = oper(st[v * 2], st[v * 2 + 1]);
    }
    // Cantidad de elementos > >= < <= a x en el rango [l,r]
    int countQuery(int v, int tl, int tr, int l, int r, T x) {
        if (tl > r || tr < l) return 0;
        if (l <= tl && tr <= r) {
            if (st[v] <= x) {
                /*
                Para mayores st[v] <= x query max(a,b)
                Para mayores o equ st[v] < x query max(a,b)
                Para menores st[v] >= x query min(a,b)
                Para menores o equ st[v] > x query min(a,b)
                */
                return 0;
            }
            if (tl == tr) return 1;
        }
        int tm = (tl + tr) / 2;
        return countQuery(v * 2, tl, tm, l, r, x) + countQuery(v * 2 +
            1, tm + 1, tr, l, r, x);
    }
    int countQuery(int l, int r, T x) { return countQuery(1, 0, n - 1,
        l, r, x); }
    void upd(int pos, T val) { upd(1, 0, n - 1, pos, val); }
    T query(int l, int r) { return query(1, 0, n - 1, l, r); }
};
```

1.10. Mo's

```
void add(int x) {}
void del(int x) {}
int get_ans() {}

vector<int> mo(const vector<ii> &q) {
  int l = 0, r = -1, blk = 350; // sqrt(n)
  vector<int> inx(sz(q)), ans(sz(q));
  auto K = [&](const ii &x) -> ii {
    return ii(x.ff / blk, x.ss ^ -(x.ff / blk & 1));
  };
  iota(all(inx), 0);
  sort(all(inx), [&](int a, int b) -> bool {
    return K(q[a]) < K(q[b]);
  });
  for (int nxt : inx) {
    ii it = q[nxt];
    while (r < it.ss) add(++r);
    while (l > it.ff) add(--l);
    while (r > it.ss) del(r--);
    while (l < it.ff) del(l++);
    ans[nxt] = get_ans();
  }
  return ans;
}
```

2. DP

2.1. Knapsack

```
    int n, x; cin>>n>>x;
    vector<array<int,2>>arr(n);
    forn(i,n) cin>>arr[i][0];
    forn(i,n) cin>>arr[i][1];

    vector<vector<int>>dp(n+1,vector<int>(x+1,0));
    forne(i,1,n+1){
        forne(j,1,x+1){
            dp[i][j]=dp[i-1][j];
            if(j-arr[i-1][0]>=0){
                int libro=arr[i-1][1];
                int price=arr[i-1][0];
                dp[i][j]=max(dp[i][j], libro+dp[i-1][j-price]);
            }

        }
    }
    cout<<dp[n][x]<<endl;


const ll inf=1e18+7;

ll Knapsack(ll n, ll cty,  vector<ll>& W,vector<ll>& V) {
    ll sum=accumulate(all(V),0LL);
    vector<ll>dp(sum+1,inf);
    dp[0]=0;
    forn(i, n){
        for(int j = sum-V[i]; j >= 0; j--){
            dp[j+V[i]]= min(dp[j+V[i]], dp[j]+W[i]);
        }
    }
    ll ans=0;
    forn(i,sum+1){
        if(dp[i]<= cty) ans=max(ans,ll(i));
    }
    return ans;
}
```

2.2. Divide and Conquer dp

```
/*
Divide and Conquer DP
Particiona o array en k subarrays
minimizando la suma de las queries
*/

ll dp[MAX][2];

void solve(int k, int l, int r, int lk, int rk) {
    if (l > r) return;
    int m = (l + r) / 2, p = -1;
    auto& ans = dp[m][k & 1] = LINF;
    for (int i = max(m, lk); i <= rk; i++) {
        ll at = dp[i + 1][~k & 1] + query(m, i);
        if (at < ans) ans = at, p = i;
    }
    solve(k, l, m - 1, lk, p), solve(k, m + 1, r, p, rk);
}
```

```
ll DC(int n, int k) {
    dp[n][0] = dp[n][1] = 0;
    for (int i = 0; i < n; i++) dp[i][0] = LINF;
    for (int i = 1; i <= k; i++) solve(i, 0, n - i, 0, n - i);
    return dp[0][k & 1];
}
```

2.3. Edit Distance

```
/*
The edit distance between two strings is the minimum number of
    operations required to transform one string into the other.
*/
{
    string a, b; cin >> a >> b;
    int n = sz(a), m = sz(b);
    vector<vector<int>>dp(n + 1, vector<int>(m + 1, inf));
    forne(i, 0, n + 1) dp[i][0] = i;
    forne(j, 0, m + 1) dp[0][j] = j;
    forne(i, 1, n + 1) {
        forne(j, 1, m + 1) {
            dp[i][j] = min({ dp[i][j - 1] + 1,dp[i - 1][j - 1] + (a[i
                - 1] != b[j - 1]),dp[i - 1][j] + 1 });
        }
    }
    cout << dp[n][m] << endl;
}

constexpr int INF = (1e18 - 1);

int edit_distance(const string& s, const string& t) {
    int n = sz(s), m = sz(t);
    vector<int> dp(m + 1);
    iota(all(dp), 0);

    forn(i, n) {
        vector<int> ndp(m + 1, INF);
        ndp[0] = i + 1;
        forn(j, m) {
            ndp[j + 1] = min({ ndp[j] + 1, dp[j + 1] + 1, dp[j] +
                (s[i] != t[j]) });
        }
        dp.swap(ndp);
    }
    return dp[m];
}
vector<string> construct_edit_distance(const string& s, const string&
    t) {

    int n = sz(s), m = sz(t);
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, INF));

    forn(i, n + 1) dp[i][0] = i;
    forn(j, m + 1) dp[0][j] = j;

    forn(i, n) {
        forn(j, m) {
            dp[i + 1][j + 1] = min({ dp[i + 1][j] + 1, dp[i][j + 1] + 1
                , dp[i][j] + (s[i] != t[j]) });
        }
    }

    vector<string> left = { s }, right = { t };

    while (n > 0 || m > 0) {
        if (n > 0 && dp[n][m] == dp[n - 1][m] + 1) {
            n--;
            string str = left.back();
            str.erase(str.begin() + n);
            left.push_back(str);
        }
        else if (m > 0 && dp[n][m] == dp[n][m - 1] + 1) {
            m--;
            string str = right.back();
            str.erase(str.begin() + m);
            right.push_back(str);
        }
        else if (n > 0 && m > 0 && dp[n][m] == dp[n - 1][m - 1] + (s[n
            - 1] != t[m - 1])) {
            n--, m--;
            if (s[n] != t[m]) {
                string str = left.back();
                str[n] = t[m];
                left.push_back(str);
            }
        }
        else {
            assert(false);
        }
    }
    assert(left.back() == right.back());
    right.pop_back();

    while (!right.empty()) {
        left.push_back(right.back());
        right.pop_back();
    }

    return left;
}
```

2.4. groups

```
/*
Dado N pesos y un limite Q, se quiere saber el minimo numero
de grupos en los que se pueden dividir los pesos tal que la
suma de los pesos de cada grupo sea menor o igual a Q
n => sz(nums);
q => maximo peso
nums => vector con los pesos
*/
int calculate(int n, int q, vector<int>& nums) {
    pair<int, int> best[1 << n];
    best[0] = { 1,0 };
    forne(i, 1, 1 << n) {
        best[i] = { n + 1,0 };
        forn(j, n) {
            if (i & (1 << j)) {
                auto cur = best[i ^ (1 << j)];
                if (cur.s + nums[j] <= q) {
                    cur.s += nums[j];
                }
                else {
                    cur.f++;
                    cur.s = nums[j];
                }
                best[i] = min(best[i], cur);
            }
        }
    }
    return best[(1 << n) - 1].f;
}


/*
Dado N pesos y un limite Q, se quiere saber el numero de grupos
consecutivos en los que se pueden dividir los pesos tal que la
suma de los pesos de cada grupo sea menor o igual a Q
n => sz(nums);
q => maximo peso
nums => vector con los pesos
*/

int CalculateLineal(int n, int q, vector<int>& nums) {
    int cnt = 0;
    int currSUM = 0;
    forn(i, n) {
        if (nums[i] + currSUM > q) {
            cnt++;
            currSUM = 0;
        }
        currSUM += nums[i];
    }
    return cnt + (currSUM > 0);
}
```

2.5. Shortest Hamiltonian Path

```
/*
Shortest Hamiltonian Path
Resuelve problemas del tipo de encontrar el camino mas corto
que recorre todos los nodos de un grafo una sola vez.
*/
vector<vector<pair<int, int>>> ady;
int n, m, target;
const int N = 18;
const int MASK = 1 << N;
const int INF = int(1e7);
int dp[N][MASK];

int solve(int v, int mask) {
    if (mask == target) return 0;
    int& ans = dp[v][mask];
    if (ans != -1) return ans;
    ans = INF;
    for (auto& u : ady[v]) {
        if (!(mask & (1 << u.first))) {
            ans = min(ans, solve(u.first, mask | (1 << u.first)) +
                u.second);
        }
    }
    return ans;
}
int main() {
cin >> n >> m;
target = (1 << n) - 1;
ady.assign(n, {});
forn(i, m) {
    int v, u, w; cin >> v >> u >> w;
    v--, u--;
    ady[v].push_back({ u, w });
    ady[u].push_back({ v, w });
}
memset(dp, -1, sizeof dp);
cout << solve(0, 1) << endl;

cout << flush;
return 0;
}
```

2.6. Money Sums

```
//  find all money sums you can create using these coins.
    int n; cin >> n;
    vector<int>nums(n), sums;
    forn(i, n) cin >> nums[i];
    vector<vector<bool>>dp(mxN + 1, vector<bool>(n * mxS + 1));
    dp[0][0] = 1;
    forne(i, 1, n + 1) {
        forn(j, mxS * n + 1) {
            dp[i][j] = dp[i - 1][j];
            if (j - nums[i - 1] >= 0 && dp[i - 1][j - nums[i - 1]])
                dp[i][j] = 1;
        }
    }

    forn(i, mxS * n + 1) {
        if (i && dp[n][i]) sums.pb(i);
    }

    cout << sz(sums) << endl;
    forn(i, sz(sums)) cout << sums[i] << " \n"[i + 1 == sz(sums)];
    cout << endl;
```

2.7. Digit dp

```
// - Descripcion: Cuenta la cantidad de numeros entre [a, b] que no
    tienen digitos iguales seguidos
// - Complejidad: O(NUM_E * NUM_T)

const int MOD = 998244353;
int tam, NUM[55], dp[55][2][2][11];

int solve(int i, bool menor, bool ncero, int last) {
    if (i == tam) return 1;
    int& ans = dp[i][menor][ncero][last];
    if (ans != -1) return ans;
    ans = 0;
    forn(dig, 10) {
        if (dig == last && (ncero || dig)) continue;
        if (menor || dig <= NUM[i]) {
            ans = (ans + solve(i + 1, menor || dig < NUM[i], ncero ||
                dig, dig)) % MOD;
        }
    }
    return ans;
}

bool g(string s) {
    forn(i, sz(s) - 1) {
        if (s[i] == s[i + 1]) return false;
    }
    return true;
}

int build(string s) {
    tam = sz(s);
    forn(i, sz(s)) {
        NUM[i] = s[i] - '0';
    }
    memset(dp, -1, sizeof dp);
    return solve(0, false, false, 10);
}
void solve() {
    string l, r;
    while (cin >> l >> r) {
        cout << ((build(r) - build(l) + MOD) % MOD + g(l)) % MOD <<
            endl;
    }
}
```

2.8. LCS

```
constexpr int mxN = 105;
vector<vector<int>> dp(mxN, vector<int>(mxN, -1));
// n=sz(s), m=sz(p)
int cntsub(const string& s, const string& p, int n, int m) {
    if ((n == 0 && m == 0) || m == 0) return 1;
    if (n == 0) return 0;
    int& ans = dp[n][m];
    if (~ans) return ans;
    if (s[n - 1] == p[m - 1]) {
        return ans = cntsub(s, p, n - 1, m - 1) + cntsub(s, p, n - 1,
            m);
    }
    else {
        return ans = cntsub(s, p, n - 1, m);
    }
}

bool issub(const string& str, const string& sub) {
    int idx = 0;
    for (auto&& i : str) {
        if (idx < sz(sub) && i == sub[idx]) {
            idx++;
        }
    }
    return idx == sz(sub);
}

//quadratic_memory
int lcs(const string& s, const string& t) {
    int n = sz(s);
    int m = sz(t);
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
    forn(i, n) {
        forn(j, m) {
            dp[i + 1][j + 1] = max({ dp[i + 1][j], dp[i][j + 1],
                dp[i][j] + (s[i] == t[j]) });
        }
    }
    return dp[n][m];
}

//best
int lcs(const string& s, const string& t) {
    int n = sz(s);
    int m = sz(t);
    vector<int> dp(m + 1, 0);
    forn(i, n) {
        vector<int> newdp(m + 1, 0);
        forn(j, m) {
            newdp[j + 1] = max({ newdp[j], dp[j + 1], dp[j] + (s[i] ==
                t[j]) });
        }
        dp.swap(newdp);
    }
    return dp[m];
}

//construct lcs
string clcs(const string& s, const string& t) {
    int n = sz(s);
    int m = sz(t);
    vector<int> dp(m + 1, 0);
    vector<vector<bool>> pre(n + 1, vector<bool>(m + 1, false));
    forn(i, n) {
        vector<int> newdp(m + 1, 0);
        forn(j, m) {
            newdp[j + 1] = max({ newdp[j], dp[j + 1], dp[j] + (s[i] ==
                t[j]) });
            pre[i + 1][j + 1] = newdp[j + 1] == newdp[j];
        }
        dp.swap(newdp);
    }
    int a = n, b = m;
    string common;
    while (a > 0 && b > 0) {
        if (s[a - 1] == t[b - 1]) {
            common += s[a - 1];
            a--; b--;
            continue;
        }
        if (pre[a][b]) b--;
        else a--;
    }
    reverse(all(common));
    return common;
}

//best: construct lcs with  Hirschberg Algorithm
string clcsh(const string_view& s, const string_view& t) {
    int n = sz(s);
    int m = sz(t);
    if (n == 0 || m == 0) return "";
    if (n == 1) return t.find(s[0]) == string::npos ? "" : string(1,
        s[0]);

    int mid = n >> 1;
    vector<int> dp_ff(m + 1, 0);
    vector<int> dp_ss(m + 1, 0);
    vector<int> newdp(m + 1, 0);

    forn(i, mid) {
        forn(j, m) {
            newdp[j + 1] = max({ newdp[j], dp_ff[j + 1], dp_ff[j] +
                (s[i] == t[j]) });
        }
        dp_ff.swap(newdp);
    }

    newdp.assign(m + 1, 0);

    for (int i = n - 1; i >= mid; i--) {
        for (int j = m - 1; j >= 0; j--) {
            newdp[j] = max({ newdp[j + 1], dp_ss[j], dp_ss[j + 1] +
                (s[i] == t[j]) });
        }
        dp_ss.swap(newdp);
    }

    int splt = 0;

    forne(j, 1, m + 1) {
        if (dp_ff[j] + dp_ss[j] > dp_ff[splt] + dp_ss[splt]) {
            splt = j;
        }
    }
    dp_ff.clear();
    dp_ss.clear();
    newdp.clear();

    return (clcsh(s.substr(0, mid), t.substr(0, splt)) +
        clcsh(s.substr(mid), t.substr(splt)));
}
```

2.9. LIS

```
int lis(vector<int>& a) {
    vector<int>dp;
    forn(i, sz(a)) {
        auto it = lower_bound(all(dp), a[i]);
        if (it != dp.end()) *it = a[i];
        else dp.pb(a[i]);
    }
    return sz(dp);

}
```

2.10. subsequences

```
struct mint {
    static constexpr int  m = 1e9 + 7;
    //static inline int  m = 998244353; //to change mod
    int x;
    mint() : x(0) {}
    mint(long long x_) :x(x_% m) { if (x < 0) x += m; }
    int val() { return x; }
    mint& operator+=(mint b) { if ((x += b.x) >= m) x -= m; return
        *this; }
    mint& operator-=(mint b) { if ((x -= b.x) < 0) x += m; return
        *this; }
    mint& operator*=(mint b) { x = (long long)(x)*b.x % m; return
        *this; }
    mint pow(long long e) const {
        mint r = 1, b = *this;
        while (e) {
            if (e & 1) r *= b;
            b *= b;
            e >>= 1;
        }
        return r;
    }
    mint inv() { return pow(m - 2); }
    mint& operator/=(mint b) { return *this *= b.pow(m - 2); }
    friend mint operator+(mint a, mint b) { return a += b; }
    friend mint operator-(mint a, mint b) { return a -= b; }
    friend mint operator/(mint a, mint b) { return a /= b; }
    friend mint operator*(mint a, mint b) { return a *= b; }
    friend bool operator<(mint a, mint b) { return a.x < b.x; }
    friend bool operator==(mint a, mint b) { return a.x == b.x; }
    friend bool operator!=(mint a, mint b) { return a.x != b.x; }
};
// Find the number of distinct subsequences of a given string.
// distinct subsequences ending at each of the 26 letters of the
    alphabet.
```

```
template<typename T>int distinctsub(const T& sub) {
    int n = sz(sub);
    vector<mint> dp(n + 1, 0);
    vector<int>last(26, -1);
    // vector<mint>end_count(26, 0);
    dp[0] = 1;
    forn(i, n) {
        dp[i + 1] += 2 * dp[i];
        // end_count[sub[i] - 'a'] += dp[i];
        if (~last[sub[i] - 'a']) {
            dp[i + 1] -= dp[last[sub[i] - 'a']];
            // end_count[sub[i] - 'a'] -= dp[last[sub[i] - 'a']];
        }
        last[sub[i] - 'a'] = i;
    }
    return dp[n].x - 1;
}

// find the number of distinct subsequences of a given string.
// number of distinct subsequences of each length from 1 to n
// number of distinct subsequences of size i -> dp[n][i]
template<typename T>int distinctsub(const T& sub) {
    int n = sz(sub);
    vector<vector<mint>> dp(n + 1, vector<mint>(n + 1, 0));
    dp[0][0] = 1;
    vector<int> last(26, -1);
    // vector<mint> end_count(26, 0);
    forn(i, n) {
        forn(j, i + 1) {
            dp[i + 1][j + 1] = dp[i][j];
            dp[i + 1][j] += dp[i][j];
            // end_count[sub[i] - 'a'] += dp[i][j].x;
        }
        if (~last[sub[i] - 'a']) {
            forn(j, i + 1) {
                dp[i + 1][j + 1] -= dp[last[sub[i] - 'a']][j];
                // end_count[sub[i] - 'a'] -= dp[last[sub[i] -
                    'a']][j].x;
            }
        }
        last[sub[i] - 'a'] = i;
    }

    mint ans = 0;
    forne(i, 1, n + 1) ans += dp[n][i];
    return ans.x;
}
```

3. Geometry

3.1. isfigure

```
#include <bits/stdc++.h>
using namespace std;
#define endl     '\n'
#define f        first
#define s        second
#define ins      insert
#define pb       push_back
#define eb       emplace_back
#define sz(x)    int((x).size())
#define all(x)   begin(x), end(x)
typedef long long ll;
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;


#define LOCAL
void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif



typedef ll T;
struct pt {
    T x, y;
    pt() : x(0), y(0) {}
    pt(T _x, T _y) : x(_x), y(_y) {}
    pt operator+(pt p) { return { x + p.x, y + p.y }; }
    pt operator-(pt p) { return { x - p.x, y - p.y }; }
    pt operator*(T d) { return { x * d, y * d }; }
    pt operator/(T d) { return { x / d, y / d }; }
    bool operator==(pt b) { return x == b.x && y == b.y; }
```

```cpp
    bool operator!=(pt b) { return x != b.x || y != b.y; }
    bool operator<(pt b) { return x == b.x ? y < b.y : x < b.x; }

    void read() {
        cin >> x >> y;
    }
};
const double PI = acos(-1);
double DEG_TO_RAD(double n) { return n * PI / 180.0; }
double RAD_TO_DEG(double n) { return n * 180.0 / PI; }
T sq(pt p) { return p.x * p.x + p.y * p.y; }
T cross(pt v, pt w) { return v.x * w.y - v.y * w.x; }
double abs(pt p) { return sqrt(sq(p)); }
T dot(pt v, pt w) { return v.x * w.x + v.y * w.y; }
T dis(pt a, pt b) { return sq(a - b); }
//Transformaciones
pt translate(pt v, pt p) { return p + v; }
pt scale(pt c, double factor, pt p) { return c + (p - c) * factor; }
pt rot(pt p, double ang) { return { p.x * cos(ang) - p.y * sin(ang),
   p.x * sin(ang) + p.y * cos(ang) }; }
pt perp(pt p) { return { -p.y, p.x }; }
T isParall(pt v, pt w) { return cross(v, w) == 0; }


// A square has four right angles and four sides with equal lengths.
bool isSquare(pt a, pt b, pt c, pt d) {
    T ab = dis(a, b);
    T bc = dis(b, c);
    T cd = dis(c, d);
    T ad = dis(a, d);
    return isParall(a - b, c - d) && isParall(a - d, b - c) && dot(b -
       a, d - a) == 0 && ab == bc && bc == cd && cd == ad;
}

// A rectangle has four right angles.
bool isRectangle(pt a, pt b, pt c, pt d) {
    return isParall(a - b, c - d) && isParall(a - d, b - c) && dot(b -
       a, d - a) == 0;
}

// A rhombus has four sides with equal lengths.
bool isRhombus(pt a, pt b, pt c, pt d) {
    T ab = dis(a, b);
    T bc = dis(b, c);
    T cd = dis(c, d);
    T ad = dis(a, d);
    return ab == bc && bc == cd && cd == ad;
}

// A parallelogram has two pairs of parallel sides.
bool isParallelogram(pt a, pt b, pt c, pt d) {
    return isParall(a - b, c - d) && isParall(a - d, b - c);
}

// A trapezium has one pair of parallel sides.
bool isTrapezium(pt a, pt b, pt c, pt d) {
    return isParall(a - b, c - d) || isParall(a - d, b - c);
}

// A kite has reflection symmetry across a diagonal.
bool isKite(pt a, pt b, pt c, pt d) {
    T ab = dis(a, b);
    T bc = dis(b, c);
    T cd = dis(c, d);
    T ad = dis(a, d);
    return (ab == bc && cd == ad) || (ab == ad && bc == cd);
}
int main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

    pt a, b, c, d;
    a.read(); b.read(); c.read(); d.read();

    if (isSquare(a, b, c, d))              cout << "square" << endl;
    else if (isRectangle(a, b, c, d))      cout << "rectangle" << endl;
    else if (isRhombus(a, b, c, d))        cout << "rhombus" << endl;
    else if (isParallelogram(a, b, c, d)) cout << "parallelogram" <<
       endl;
    else if (isTrapezium(a, b, c, d))      cout << "trapezium" << endl;
    else if (isKite(a, b, c, d))           cout << "kite" << endl;
    else cout << "none" << endl;









    cout << flush;
    return 0;
```

```
}
```

4. Graph

4.1. bipartite Graph

```
template<typename T>
struct Graph {
    int n;
    vector<vector<T>> adj;
    vector<T> side;
    Graph(int size) {
        n = size;
        adj.resize(n);
        side.resize(n, -1);
    }

    void addEdge(int u, int v, int uno) {
        v -= uno; u -= uno;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    bool is_bipartite() {
        bool check = true;
        queue<int> q;
        for (int edge = 0; edge < n; ++edge) {
            if (side[edge] == -1) {
                q.push(edge);
                side[edge] = 0;
                while (q.size()) {
                    int curr = q.front();
                    q.pop();
                    for (auto neig : adj[curr]) {
                        if (side[neig] == -1) {
                            side[neig] = (1 ^ side[curr]);
                            q.push(neig);
                        }
                        else {
                            check &= (side[neig] != side[curr]);
                        }
                    }
                }
            }
        }
        return check;
    }
};
```

4.2. Floyd Warshall

```
template<typename T>
struct FloydW {
    vector<vector<T>> g;
    int n;
    int64_t INF = 1e18 + 7; //mxW
    FloydW(int N) {
        this->n = N;
        g.resize(N, vector<T>(N, INF));
        forn(i, N) g[i][i] = 0;
    }
    void addEdge(int u, int v, T w, int uno) {
        u -= uno; v -= uno;
        g[u][v] = min(g[u][v], w);
        g[v][u] = min(g[v][u], w);
    }

    void init() {
        forn(k, n) {
            forn(i, n) {
                forn(j, n) {
                    g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
                }
            }
        }
    }

    T query(int u, int v) {
        return g[u][v] == INF ? -1 : g[u][v];
    }

};
```

4.3. nx-ny-4

```
vector<vector<char>>board;
vector<vector<bool>>vis;
int n, m;
```

```
// R D L U
int dx[] = { 0, 1, 0, -1 };
int dy[] = { 1, 0, -1, 0 };
void init() {
    board.resize(n + 1, vector<char>(m + 1));
    vis.resize(n + 1, vector<bool>(m + 1, 0));
}

void back(int x, int y) {
    vis[x][y] = 1;
    forn(i, 4) {
        int nx = x + dx[i], ny = y + dy[i];
        if (nx >= 0 && nx < n && ny >= 0 && ny < m && board[nx][ny] !=
            '1' && !vis[nx][ny]) back(nx, ny);
    }
}
```

4.4. nx-ny-8

```
vector<vector<char>>board;
vector<vector<bool>>vis;
int n, m;
// U,UR, R,RD,D,LD,L, UL
int dx[8] = { -1, -1, 0, 1, 1, 1, 0, -1 };
int dy[8] = { 0, 1, 1, 1, 0, -1, -1, -1 };

void init() {
    board.resize(n + 1, vector<char>(m + 1));
    vis.resize(n + 1, vector<bool>(m + 1, 0));
}

void back(int x, int y) {
    vis[x][y] = 1;
    forn(i, 8) {
        int nx = x + dx[i], ny = y + dy[i];
        if (nx >= 0 && nx < n && ny >= 0 && ny < m && board[nx][ny] !=
            '1' && !vis[nx][ny]) back(nx, ny);
    }
}
```

5. Math

5.1. Propiedades del modulo

```
#define suma(a,b,MOD)  ((a%MOD)+(b%MOD))%MOD
#define resta(a,b,MOD) ((a%MOD)-(b%MOD)+MOD)%MOD
#define mult(a,b,MOD)  ((a%MOD)*(b%MOD))%MOD
```

5.2. TernarySearch

```
double f(double x) {
    return x;
}
// ternary_search(0.0, posibleMaximo)
double ternary_search(double l, double r) {
    double eps = 1e-9;
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);
        double f2 = f(m2);
        // if (f1 > c) f1 = f2 minimizar;
        if (f1 < f2) l = m1;
        else r = m2;
    }

    // return l;
    return f(l);
}
```

5.3. squares in a circle

```
#include <bits/stdc++.h>
using namespace std;
#define endl     '\n'
#define f        first
#define s        second
#define ins      insert
#define pb       push_back
#define eb       emplace_back
#define sz(x)    int((x).size())
```

```
#define all(x)   begin(x), end(x)
typedef long long ll;
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;

void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif
ll get(ll mid){
    ll ans=0;
    for(ll i=1; i*i < mid; i++){
        ans+=4*floor(sqrt( mid-i*i));
    }

    return ans;

}

const int inf= 1e9+7;
int main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int test=1;
    #ifdef LOCAL
        freopen("in.txt", "r", stdin);
        freopen("out.txt", "w", stdout);
        test=2;
    #endif

    while(test--){
        ll n; cin>>n;
        ll r=4*n, l=1;
        double pre= 1.0*inf;
        while(l<=r){
            ll mid=(l+r)>>1;
            if(get(mid)>n){
                r=mid-1;
                pre=min(pre,sqrt(mid));
            }else l=mid+1;
        }
        cout<<fixed<< setprecision(30)<<pre<<endl;


    }

    cout << flush;
    return 0;
}
```

5.4. Pow

```
int64_t binpow(int64_t a, int64_t b, int64_t m) {
    a %= m;
    int64_t res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}

int64_t binpow(int64_t a, int64_t b) {
    int64_t res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

5.5. dectobin

```
ll bin(int n) {
    string s;
    while (n) {
        if (n & 1) s.pb('1');
        else s.pb('0');
        n >>= 1;
    }
    reverse(all(s));
    return stoll(s);
}
```

5.6. fraction

```
struct fraction {
    int num, den;

    fraction(int num, int den) :num(num), den(den) {
        check_den();
        simplify();
    }
    void check_den() {
        if (den < 0) {
            num = -num;
            den = -den;
        }
    }
    void simplify() {
        int mcd = __gcd(abs(num), abs(den));
        num /= mcd;
        den /= mcd;
    }
    pair<int, int> x() { return { num,den }; }

    fraction operator + (const fraction& x) const {
        return fraction(num * x.den + den * x.num, den * x.den);
    }
    fraction operator - (const fraction& x) const {
        return fraction(num * x.den - den * x.num, den * x.den);
    }
    fraction operator * (const fraction& x) const {
        return fraction(num * x.num, den * x.den);
    }
    fraction operator / (const fraction& x) const {
        return fraction(num * x.den, den * x.num);
    }
    friend ostream& operator << (ostream& os, const fraction& x) {
        return os << x.num << " / " << x.den;
    }
};
```

5.7. nCK

```
struct mint {
    static constexpr int  m = 1e9 + 7;
    //static inline int  m = 998244353; //to change mod
    int x;
    mint() : x(0) {}
    mint(long long x_) :x(x_% m) { if (x < 0) x += m; }
    int val() { return x; }
    mint& operator+=(mint b) { if ((x += b.x) >= m) x -= m; return
        *this; }
    mint& operator-=(mint b) { if ((x -= b.x) < 0) x += m; return
        *this; }
    mint& operator*=(mint b) { x = (long long)(x)*b.x % m; return
        *this; }
    mint pow(long long e) const {
        mint r = 1, b = *this;
        while (e) {
            if (e & 1) r *= b;
            b *= b;
            e >>= 1;
        }
        return r;
    }
    mint inv() { return pow(m - 2); }
    mint& operator/=(mint b) { return *this *= b.pow(m - 2); }
    friend mint operator+(mint a, mint b) { return a += b; }
    friend mint operator-(mint a, mint b) { return a -= b; }
    friend mint operator/(mint a, mint b) { return a /= b; }
    friend mint operator*(mint a, mint b) { return a *= b; }
    friend bool operator<(mint a, mint b) { return a.x < b.x; }
    friend bool operator==(mint a, mint b) { return a.x == b.x; }
    friend bool operator!=(mint a, mint b) { return a.x != b.x; }
};


const int mxN = 1e6 + 7; // max value of  N,K
mint fact[mxN];
mint inv_fact[mxN];
void init() {
    fact[0] = 1;
    forne(i, 1, mxN) fact[i] = fact[i - 1] * i;
    forn(i, mxN) inv_fact[i] = fact[i].inv();
}
// https://cp-algorithms.com/combinatorics/binomial-coefficients.html

mint nCk(ll n, ll k) {
    return (fact[n] * inv_fact[k]) * inv_fact[n - k];
}
```

5.8. divisores

```
vector<int>div(int n) {
    vector<int> ans;
    for (int i = 1; i * i <= n;i++) {
        if (n % i == 0) {
            ans.pb(i);
            if (i != n / i) {
                ans.pb(n / i);
            }
        }
    }
    return ans;
}
```

6. Problems

6.1. dp+nck

```
#include <bits/stdc++.h>
using namespace std;
#define endl      '\n'
#define f         first
#define s         second
#define ins       insert
#define pb        push_back
#define eb        emplace_back
#define sz(x)     int((x).size())
#define all(x)    begin(x), end(x)
typedef long long ll;
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;


#define LOCAL
void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif


//
    https://github.com/JaiderBR/CompetitiveProgramming/blob/main/Data%20St
static constexpr int mod = 1e9 + 7;
struct mint {
    static constexpr int  m = 1e9 + 7;
    int x;
    mint() : x(0) {}
    mint(long long x_) :x(x_% m) { if (x < 0) x += m; }
    int val() { return x; }
    mint& operator+=(mint b) { if ((x += b.x) >= m) x -= m; return
       *this; }
    mint& operator-=(mint b) { if ((x -= b.x) < 0) x += m; return
       *this; }
    mint& operator*=(mint b) { x = (long long)(x)*b.x % m; return
       *this; }
    mint pow(long long e) const {
        mint r = 1, b = *this;
        while (e) {
            if (e & 1) r *= b;
            b *= b;
            e >>= 1;
        }
        return r;
    }
    mint inv() { return pow(m - 2); }
    mint& operator/=(mint b) { return *this *= b.pow(m - 2); }
    friend mint operator+(mint a, mint b) { return a += b; }
    friend mint operator-(mint a, mint b) { return a -= b; }
    friend mint operator/(mint a, mint b) { return a /= b; }
    friend mint operator*(mint a, mint b) { return a *= b; }
    friend bool operator==(mint a, mint b) { return a.x == b.x; }
    friend bool operator!=(mint a, mint b) { return a.x != b.x; }
};

const int mxN = 105;
mint dp[21][mxN][mxN][mxN];
mint factorial[mxN];
mint inverse_factorial[mxN];

void init() {
    factorial[0] = 1;
    forne(i, 1, mxN) factorial[i] = factorial[i - 1] * i;
    forn(i, mxN) inverse_factorial[i] = factorial[i].inv();
}
// https://cp-algorithms.com/combinatorics/binomial-coefficients.html
mint binomial_coefficient(ll n, ll k) {
    return (factorial[n] * inverse_factorial[k]) * inverse_factorial[n
        - k];
}

mint back(ll n, ll r, ll g, ll b) {
    if (r < 0 || g < 0 || b < 0) return 0;
    if (n == 0) return 1;
    if (dp[n][r][g][b] != -1) return dp[n][r][g][b];
    mint form_1 = 0, form_2 = 0, form_3 = 0;
    form_1 = back(n - 1, r - n, g, b) + back(n - 1, r, g - n, b) +
        back(n - 1, r, g, b - n);
    if (n % 2 == 0) {
        /*
        R G B (R G) - (R B) - (G B)
        */
        form_2 = binomial_coefficient(n, n / 2) * ((back(n - 1, r - n
            / 2, g - n / 2, b) + back(n - 1, r - n / 2, g, b - n / 2)
            + back(n - 1, r, g - n / 2, b - n / 2)));
    }

    if (n % 3 == 0) {
        /*
        n=3 bc_1=3 bc_2=2 = 6
        R G B - R B G - G B R -B G R - G R B - B R G
        */
        mint bc_1 = binomial_coefficient(n, n / 3);
        mint bc_2 = binomial_coefficient(2 * n / 3, n / 3);
        form_3 = (bc_1 * bc_2) * back(n - 1, r - n / 3, g - n / 3, b -
            n / 3);
    }
    return dp[n][r][g][b] = form_1 + form_2 + form_3;
}


int main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    init();
    ll n, r, g, b; cin >> n >> r >> g >> b;
    forn(i, n + 1) forn(j, r + 1) forn(k, g + 1) forn(l, b + 1)
        dp[i][j][k][l] = -1;
    // memset(dp, -1, sizeof dp);
    cout << back(n, r, g, b).val() << endl;

    cout << flush;
    return 0;
}
```

6.2. F Less Than G

```
/*
Given two arrays a and b of n non-negative integers, count the number
    of good pairs
l,r (1<=l<=r<=n), satisfying F(l,r)<G(l,r)
Where F(l,r) is the sum of the square of numbers in the range [l,r]
And G(l,r) is the square of the bitwise OR of the range [l,r]
*/

main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

    // brute();
    int n; cin >> n;
    vector<int> a(n), b(n), prefix(n + 6);
    set<int> pro[25];
    forn(i, n) cin >> a[i];
    forn(i, n) cin >> b[i];
    forn(init, 22) pro[init].insert(n);
    forn(i, n) prefix[i + 1] = prefix[i] + a[i] * a[i];
    prefix.erase(prefix.begin());
    forn(i, n) forn(j, 22) if (b[i] & (1 << j)) pro[j].insert(i);

    int ans = 0;
    forn(i, n) {
        int last = i, Or = b[i];
        while (last < n) {
            int best = n;
            forn(j, 22) {
                if (!(Or & (1 << j))) best = min(best,
                    *pro[j].upper_bound(last));
            }
            int l = last, r = best - 1, x = -1, need = Or * Or;
            while (l <= r) {
                int mid = (l + r) / 2;
                if (prefix[mid] - (i > 0 ? prefix[i - 1] : 0) >= need)
                    r = mid - 1;
                else l = mid + 1, x = mid;
            }
            if (~x) ans += x - last + 1;
            swap(last, best);
            if (last < n) Or |= b[last];
        }
    }
    cout << ans << endl;


    cout << flush;
    return 0;
}
```

6.3. Nested Circles

```
/*
You are given n circles numbered from 1 to n.
Each circle is defined by an integer center (xi,yi) and an integer
    radius ri.

Then we will ask you q questions. In each question,
we will give you an integer point (xi,yi),
and you have to find the number of circles that cover this point.
*/

void solve() {

    map<pair<int, int>, vector<array<int, 3>>> mp;
    map<pair<int, int>, int> mpans;

    int n, q, x, y, r; cin >> n >> q;

    forn(i, n) {
        cin >> x >> y >> r;
        mp[{int(x / 10), int(y / 10)}].pb({ x,y,r });
    }

    while (q--) {
        cin >> x >> y;
        int gX = int(x / 10), gY = int(y / 10);
        int ans = 0;
        if (!mpans.count({ x,y })) {
            forne(dx, -1, 2) {
                forne(dy, -1, 2) {
                    auto it = mp.find({ gX + dx,  gY + dy });
                    if (it != mp.end()) {
                        each(i, it->s) {
                            int xc = i[0], yc = i[1], rc = i[2];
                            if ((x - xc) * (x - xc) + (y - yc) * (y -
                                yc) <= rc * rc) ans++;
                        }
                    }
                }
            }
            cout << ans << endl;
            mpans[{x, y}] = ans;
        }
        else cout << mpans[{x, y}] << endl;

    }
}
```

6.4. Restoring the Expression

```
/*
12345168 = 123+45=168
199100 = 1+99=100
*/

#include <bits/stdc++.h>
using namespace std;
#define endl     '\n'
#define f        first
#define s        second
#define ins      insert
#define pb       push_back
#define eb       emplace_back
#define sz(x)    int((x).size())
#define all(x)   begin(x), end(x)
typedef long long ll;
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;


void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
```

```
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif
#define int int64_t

constexpr int mxN = 1e6 + 7, mod = 998244353;
vector<int>p(mxN);
void pre(int c, int mod) {
    p[0] = 1;
    for (int i = 0; i < mxN - 1; i++) {
        p[i + 1] = (c * p[i]) % mod;
    }
}

struct Hash {
    #warning llamar pre;
    ll c, mod;
    vector<int> h;
    Hash(const string& s, const int c, const int mod) : c(c),
       mod(mod), h(sz(s) + 1) {
        h[0] = 0;
        for (int i = 0; i < sz(s); i++) {
            h[i + 1] = (c * h[i] + s[i] - '0') % mod;
        }
    }
    // Returns hash of interval s[a ... b] (where 0 <= a <= b < sz(s))
    ll get(int a, int b) {
        return (h[b + 1] - ((h[a] * p[b - a + 1]) % mod) + mod) % mod;
    }
};

bool same(Hash& Ha, Hash& Hb, int l, int r) {
    int qa = Ha.get(l, r);
    int qb = Hb.get(sz(Hb.h) - 2 - r, sz(Hb.h) - 2 - l);
    return qa == qb;
}

main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

    pre(10, mod);
    string s; cin >> s;
    Hash ha(s, 10, mod);
    int n = sz(s);

    auto ok = [&](int i, int j) {
        i--, j--;
        if (i - 1 < 0) return;
        int a = ha.get(0, i - 1), b = ha.get(i, j - 1), c = ha.get(j,
           n - 1);
        if (((a + b) % mod) == c && ((s[i] != '0' ? 1 : i == j - 1))
           && ((s[j] != '0' ? 1 : j == n - 1))) {
            cout << string(begin(s), begin(s) + i) << "+" <<
               string(begin(s) + i, begin(s) + j) << "=" <<
               string(begin(s) + j, end(s)) << endl;
            exit(0);
        }
    };

    forne(i, n / 3, ((n / 2) + 1)) {
        ok(i, n - i + 1);
        ok(i + 1, n - i + 1);
        ok(n - i * 2 + 2, n - i + 1);
        ok(n - i * 2 + 1, n - i + 1);
    }









    cout << flush;
    return 0;
}
```

6.5. matrixexp

```
//https://codeforces.com/gym/104758/problem/B

#include <bits/stdc++.h>
using namespace std;
#define endl     '\n'
#define f        first
#define s        second
#define ins      insert
#define pb       push_back
```

```
#define eb       emplace_back
#define sz(x)    int((x).size())
#define all(x)   begin(x), end(x)
typedef long long ll;
#define int ll
typedef unsigned long long ull;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;



void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
   ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
   << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif


const int MOD = 1e9 + 7;
struct matrix {
  int n, m;
  vector<vector<int>>v;

  matrix(int n, int m, bool ones = false) : n(n), m(m), v(n,
     vector<int>(m)) {
    if (ones) forn (i, n) v[i][i] = 1;
  }

  matrix operator * (const matrix &o) {
    matrix ans(n, o.m);
    forn (i, n)
      forn (k, m) if (v[i][k])
        forn (j, o.m)
          ans[i][j] = (v[i][k] * o.v[k][j] + ans[i][j]) % MOD;
    return ans;
  }

  vector<int> & operator [] (int i) {
    return v[i];
  }
};

matrix binpow(matrix b, int e) {
  matrix ans(b.n, b.m, true);
  while (e) {
    if (e & 1) ans = ans * b;
    b = b * b;
    e >>= 1;
  }
  return ans;
}


void solve(){

    matrix a(4, 4), b(4, 4);
    a[0][0] = 4;
    a[0][1] = (-1 + MOD) % MOD;
    a[0][2] = (-1 + MOD) % MOD;
    a[0][3] = (-1 + MOD) % MOD;
    forn(i, 3) a[i + 1][i] = 1;

    b[0][0] = 17;
    b[1][0] = 3;
    b[2][0] = 2;
    b[3][0] = 1;

    int n; cin >> n;
    if(n < 4){
        cout << b[3 - n][0] << endl;
    }else{
        matrix ans = binpow(a, n - 3) * b;
        cout << ans[0][0] << endl;
    }






}
 main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int testcase=1;
    #ifdef LOCAL
        freopen("in.txt", "r", stdin);
        freopen("out.txt", "w", stdout);
        testcase=4;
    #endif

     //cin >> testcase;
     while (testcase--)solve();







    cout << flush;
    return 0;
}
```

6.6. Dueling Digits

```
const int mxN = 800 + 7, mxS = 14400 + 7, mod = 1e9 + 7;
int dp[mxN][mxS << 1];

int back(int pos, int addA) {
    if (pos == 0) return addA == mxS;
    int& ans = dp[pos][addA];
    if (~ans) return ans;
    ans = 0;
    forn(i, 10) {
        forn(j, 10) {
            if (i == j) continue;
            if ((i == 0 || j == 0) && pos == 1) continue;
            ans = (ans + back(pos - 1, addA + i - j)) % mod;
        }
    }
    return ans;
}

void solve() {

    int n; cin >> n;
    cout << back(n, mxS) << endl;

}
```

7. String

7.1. SThash

```
/*
query = or.query(l, r)  ror.query(n - 1 - r, n - 1 - l)
udp   = or.upd(pos, a)  ror.upd(n - 1 - pos, a);

LLAMAR PRECAL!!!!!!!!!!!!!!!!!!!!!!!!!

*/
const int MAX = 2e5 + 6;
const int MOD = 1e9 + 7;
const int BASE = 137;

int BP[MAX];
void precal() {
    BP[0] = 1;
    BP[1] = 1;
    for (int i = 1;i < MAX;i++) {
        BP[i] = (BP[i - 1] * BASE) % MOD;
    }
}
template<typename T>
struct SThash {
    struct node {
        int tam;
        int h;
        node() {}
    };
    int n;
    vector<node>tree;
    SThash(string& s) {
        n = sz(s);
        vector<T> a(n);
        tree.resize(n * 4);
        for (int i = 0;i < n;i++) a[i] = s[i];
        build(1, 0, n - 1, a);
    }

    node Merge(node a, node b) {
        node ret;
        ret.h = ((a.h * BP[b.tam]) + b.h) % MOD;
        ret.tam = a.tam + b.tam;
        return ret;
    }

    void build(int v, int tl, int tr, vector<T>& a) {
        if (tl == tr) {
            tree[v].h = a[tl];
            tree[v].tam = 1;
            return;
        }
        int mid = (tl + tr) >> 1;
        build(v * 2, tl, mid, a);
        build(v * 2 + 1, mid + 1, tr, a);
        tree[v] = Merge(tree[v * 2], tree[v * 2 + 1]);
    }


    void upd(int v, int tl, int tr, int id, int val) {
        if (tl > id or tr < id) return;
        if (tl == tr and tr == id) {
            tree[v].h = val;
            return;
        }
        int mid = (tl + tr) >> 1;
        upd(v * 2, tl, mid, id, val);
        upd(v * 2 + 1, mid + 1, tr, id, val);
        tree[v] = Merge(tree[v * 2], tree[v * 2 + 1]);
    }

    node query(int v, int tl, int tr, int l, int r) {
        if (tl >= l and tr <= r) return tree[v];
        int mid = (tl + tr) / 2;
        if (mid < l) return query(v + v + 1, mid + 1, tr, l, r);
        else if (mid >= r) return query(v + v, tl, mid, l, r);
        else return Merge(query(v + v, tl, mid, l, r), query(v + v + 1
            , mid + 1, tr, l, r));
    }
    void upd(int pos, int val) { upd(1, 0, n - 1, pos, val); }
    int query(int l, int r) { return query(1, 0, n - 1, l, r).h; }
};
```

7.2. HuffmanCoding

```
struct Node {
    char data;
    int freq;
    Node* L, * R;
    Node(char data, int freq) : data(data), freq(freq), L(nullptr),
        R(nullptr) {}
};
struct Huffman {
    unordered_map<char, int> freqMap;
    unordered_map<char, string>hfCodes;
    string str;
    Node* root;
    Huffman(string& str) : str(str) {
        for (auto&& i : str) freqMap[i]++;
        root = build();
        createHF(root, "");
    }

    struct oper {
        bool operator()(const Node* L, const Node* R) const {
            return L->freq > R->freq;
        }
    };

    Node* build() {
        priority_queue<Node*, vector<Node*>, oper> pq;
        each(i, freqMap) {
            pq.push(new Node(i.f, i.s));
        }
        if (sz(pq) == 1) {
            Node* L = pq.top();
            pq.pop();
            Node* parent = new Node('\0', L->freq);
            parent->L = L;
            pq.push(parent);
        }
        while (sz(pq) > 1) {
            Node* L = pq.top();
            pq.pop();
            Node* R = pq.top();
            pq.pop();
            Node* parent = new Node('\0', L->freq + R->freq);
            parent->L = L;
            parent->R = R;
            pq.push(parent);
        }
        return pq.top();
    }

    void createHF(Node* root, string code) {
        if (root == nullptr) return;
        if (!root->L && !root->R) {
            hfCodes[root->data] = code;
        }
        createHF(root->L, code + "0");
        createHF(root->R, code + "1");
    }

    int LengthBinary() {
        int cnt = 0;
        for (auto&& i : str) cnt += sz(hfCodes[i]);
        return cnt;
    }
    void _print() {
        each(i, hfCodes) cout << i.f << ' ' << i.s << endl;
    }

};




int main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);



    string s; cin >> s;
    Huffman hf(s);
    cout << hf.LengthBinary() << endl;

    // hf._print();





    cout << flush;
    return 0;
}
```

7.3. palindrome range

```
    vector<vector<int>>dp(mxN, vector<int>(mxN));
    vector<vector<bool>>pal(mxN, vector<bool>(mxN));
    string s; cin >> s;
    int n = sz(s), q, l, r;

    for (int i = n - 1; i >= 0; i--) {
        dp[i][i] = pal[i][i] = 1;
        for (int j = i + 1; j < n; j++) {
            pal[i][j] = (pal[i + 1][j - 1] || j - i == 1) & (s[i] ==
                s[j]);
            dp[i][j] = dp[i + 1][j] + dp[i][j - 1] - dp[i + 1][j - 1]
                + pal[i][j];
        }
    }
    cin >> q;
    while (q--) {
        cin >> l >> r;
        cout << dp[l - 1][r - 1] << endl;
    }
```

7.4. Z

```
/*
Dado una string s, devuelve un vector Z donde Z[i] representa el
    prefijo
de mayor longitud  de s, que tambien es prefijo del sufijo de s que
    inicia
en i.
01234567
aabzaaba "aab" es un prefijo de s y "aaba" es un sufijo de s, Z[4] = 3.

Otra definicion: Dado un string s retorna un vector z donde z[i] es
    igual al mayor
numero de caracteres desde s[i] que coinciden con los caracteres desde
    s[0]

Complejidad: O(|n|)

*/
vector<int> z_function(string& s) {
    int n = s.size();
    vector<int> z(n);
    for (int i = 1, x = 0, y = 0; i < n; i++) {
        z[i] = max(0ll, min(z[i - x], y - i + 1));
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            x = i, y = i + z[i], z[i]++;
        }
    }
    return z;
}
```

7.5. ahobit

```
/*
ahobit: used to search for a pattern in a string
    - query(l,r): searches for how many times the pattern is repeated
      in the range [l,r]
    - numoc: number of occurrences of the pattern in the string
    - a: vector with the positions of the occurrences of the pattern
    - szp: size of the pattern
    - bs: bitset of the characters in the string
    - oc: bitset of the occurrences of the pattern
    - N: maximum size of the string
*/
struct ahobit {
    static constexpr int N = 1e5 + 9;
    bitset<N>bs[26], oc, _all;
    int szp;
    ahobit(const string& s) {
        for (int i = 0; i < sz(s); i++) bs[s[i] - 'a'][i] = 1, _all[i]
            = 1;
    }
    void add(const string& p) {
        // oc.set();
        oc = _all; szp = sz(p);
        for (int i = 0; i < sz(p); i++) oc &= (bs[p[i] - 'a'] >> i);
    }
    int num_occu() {
        return oc.count();
    }
    vector<int> pos_occu() {
        vector<int> a;
        int pos = oc._Find_first();
        a.clear(); a.pb(pos);
        pos = oc._Find_next(pos);
        while (pos < N) {
            a.pb(pos);
            pos = oc._Find_next(pos);
        }
        return a;
    }

    int query(int l, int r) {
        //1-indexed
        if (szp > r - l + 1) return 0;
        return (oc >> (l - 1)).count() - (oc >> (r - szp + 1)).count();
    }
};
```

7.6. paltree

```
/*
size() number of different palindrome substr
propagate() number of palindrome substr
lps longest palindrome substr {star, len}
*/

struct paltree {
    vector<vector<int>> t;
    int n, last, sz;
    vector<int> s, len, link, qt;
    pair<int, int> lps{ 0,0 };

    paltree(int N) {
        t.assign(N + 2, vector<int>(26, int()));
        s = len = link = qt = vector<int>(N + 2);
        s[0] = -1, link[0] = 1, len[0] = 0, link[1] = 1, len[1] = -1;
        sz = 2, last = 0, n = 1;
    }

    void add(char c) {
        s[n++] = c -= 'a';
        while (s[n - len[last] - 2] != c) last = link[last];
        if (!t[last][c]) {
            int prev = link[last];
            while (s[n - len[prev] - 2] != c) prev = link[prev];
            link[sz] = t[prev][c];
            len[sz] = len[last] + 2;
            t[last][c] = sz++;
            if (len[sz - 1] > lps.s) {
                lps = { n - len[sz - 1] - 1 ,len[sz - 1] };
            }
        }
        qt[last = t[last][c]]++;
    }
    int size() {
        return sz - 2;
    }

    ll propagate() {
        ll cnt = 0;
        for (int i = n; i > 1; i--) {
            qt[link[i]] += qt[i];
            cnt += qt[i];
        }
        return cnt;
    }

};
```

7.7. hashing

```
/*
Usage:
        Good values c = 137, modbest=998244353, mod = 10^9 + 7, mod =
            1e18 + 9.
        If necessary to check too many pairs of hashes, use two
        different hashes.
        If hashing something other than english characters:
            - Don't have elements with value 0.
            - Use c > max element value.
*/

//#define int int64_t
constexpr int mxN = 1e6 + 7;
vector<int>p(mxN);
void pre(int c, int mod) {
    p[0] = 1;
    for (int i = 0; i < mxN - 1; i++) {
        p[i + 1] = (c * p[i]) % mod;
    }
}

struct Hash {
    #warning llamar pre;
    ll c, mod;
    vector<int> h;
    Hash(const string& s, const int c, const int mod) : c(c),
        mod(mod), h(sz(s) + 1) {
        h[0] = 0;
        for (int i = 0; i < sz(s); i++) {
            h[i + 1] = (c * h[i] + s[i]) % mod;
        }
    }
    // Returns hash of interval s[a ... b] (where 0 <= a <= b < sz(s))
    ll get(int a, int b) {
        return (h[b + 1] - ((h[a] * p[b - a + 1]) % mod) + mod) % mod;
    }
};

bool same(Hash& Ha, Hash& Hb, int l, int r) {
    int qa = Ha.get(l, r);
    int qb = Hb.get(sz(Hb.h) - 2 - r, sz(Hb.h) - 2 - l);
    return qa == qb;
}
```

7.8. hash table

```
/*
hash_table
sirve para contar cuantas veces aparece un patron en un string
en un rango [l,r] en O(1) con O(n) de preprocesamiento
ejemplo:
string s;
a b a c a b a d a b a c a b a
string p;
b a
hash_table<int> h(s,p);
0 0 1 1 1 1 2 2 2 2 3 3 3 3 4

hash_table(string s, int m)
sirve para contar cuantas veces aparece un patron de longitud m en un
    string
modificar bulid() segun condicion

*/


template<typename T>
struct hash_table
{
    string s, p;
    int n, m;
    vector<T>prefix;
    hash_table(string s, string p) {
        this->s = s;
        this->p = p;
        this->n = sz(s);
        this->m = sz(p);
        prefix.resize(n + 5, 0);
        build();
    }
    hash_table(string s, int m) {
        this->s = s;
        this->n = sz(s);
        this->m = m;
        prefix.resize(n + 5, 0);
        build();
    }

    void build() {
        forn(i, n - m + 1) {
            int ok = 1;
            forn(j, m) {
                if (s[i + j] != p[j]) {
                    ok = 0;
                    break;
                }
            }
            prefix[i + 1] = prefix[i] + ok;
        }
    }
    int query(int l, int r) {
        if (r - l + 1 < m) return 0;
        return prefix[r - m + 1] - prefix[l - 1];
    }
};
```

7.9. suffixAutomaton

```
// codebreaker suffix automaton
struct suffixAutomaton {
    struct node {
        int len, link; bool end;
        map<char, int> next;
        int cnt; ll in, out, cntSubstrs;
    };

    vector<node> sa;
    //ocurrencias de estados, usar encontrar kth pequena lexico all
        strings
    vector<ll> cntState;
    int last; ll substrs = 0;

    suffixAutomaton() {}
    suffixAutomaton(string& s) {
        sa.reserve(sz(s) * 2);
        // cntState.reserve(sz(s)*2);
        last = add_node();
        sa[0].link = -1;
        sa[0].in = 1;
        for (char& c : s) add_char(c);
        for (int p = last; p; p = sa[p].link) sa[p].end = 1;
    }

    int add_node() { sa.pb({}); return sa.size() - 1; }

    void add_char(char c) {
        int u = add_node(), p = last;
        // cntState[u] = 1;
        sa[u].len = sa[last].len + 1;
        while (p != -1 && !sa[p].next.count(c)) {
            sa[p].next[c] = u;
            sa[u].in += sa[p].in;
            substrs += sa[p].in;
            p = sa[p].link;
        }
        if (p != -1) {
            int q = sa[p].next[c];
            if (sa[p].len + 1 != sa[q].len) {
                int clone = add_node();
                // cntState[clone] = 0;
                sa[clone] = sa[q];
                sa[clone].len = sa[p].len + 1;
                sa[clone].in = 0;
                sa[q].link = sa[u].link = clone;
                while (p != -1 && sa[p].next[c] == q) {
                    sa[p].next[c] = clone;
                    sa[q].in -= sa[p].in;
                    sa[clone].in += sa[p].in;
                    p = sa[p].link;
                }
            }
            else sa[u].link = q;
        }
        last = u;
    }
    //Cuenta la cantidad de ocurrencias de una cadena s
    int match_str(string& s) {
        int u = 0, n = sz(s);
        for (int i = 0; i < n; ++i) {
            if (!sa[u].next.count(s[i])) return 0;
            u = sa[u].next[s[i]];
        }
        return count_occ(u);
    }
```

```cpp
int count_occ(int u) {
    if (sa[u].cnt != 0) return sa[u].cnt;
    sa[u].cnt = sa[u].end;
    for (auto& v : sa[u].next)
        sa[u].cnt += count_occ(v.ss);
    return sa[u].cnt;
}

//Calcular la cantidad de caminos que pertenecen al estado ti,
   desde ti hasta tn
ll count_paths(int u) {
    //Out cuenta la cantidad de caminos (cantidad de cadenas
       distintas)
    if (sa[u].out != 0) return sa[u].out; //sa[u].cntSubstrs != 0
       return sa[u].cntSubstrs
    for (auto& v : sa[u].next)
        sa[u].out += count_paths(v.ss); //sa[u].cntSubstrs +=
           count_paths(v.ss)
    return ++sa[u].out; //sa[u].cntSubstrs += cntState[u];
}

//kth subcadena mas pequena en orden lexicografico
//out para cadenas distintas, cntSubstrs para todas las cadenas
   llamar antes pre
string kth;
void dfs_kth(int u, ll& k) { //Antes llamar a count
    if (k == 0) return; // k < cntState[u] para todas las cadenas
    k--; // k -= cntState[u];
    for (auto& v : sa[u].next) {
        if (k < sa[v.ss].out) { //k < sa[v.ss].cntSubstrs
            kth += v.ff;
            return dfs_kth(v.ss, k);
        }
        k -= sa[v.ss].out; //k -= sa[v.ss],cntSubstrs
    }
}
//calcula la cantidad de ocurrencias de los estados
void pre() {
    vector<ii> v(sz(sa));
    forn(i, sz(sa)) v[i] = { sa[i].len, i };
    sort(all(v), greater<ii>());
    for (auto& it : v) {
        int u = it.ss;
        if (sa[u].link != -1)
            cntState[sa[u].link] += cntState[u];
    }
    cntState[0] = 1;
}
//longest common substring
int lcs(string& t) {
    int n = sz(t);
    int u = 0, l = 0, best = 0, bestPosition = 0;
    forn(i, n) {
        while (u && !sa[u].next.count(t[i])) {
            u = sa[u].link;
            l = sa[u].len;
        }
        if (sa[u].next.count(t[i])) u = sa[u].next[t[i]], l++;
        if (best < l) best = l, bestPosition = i;
    }
    return best;
}
vector<int> LCS, match;
void lcsMatch(string& t) {
    match.assign(sz(sa), 0); //usar pivote si toca resetear mucho
    int u = 0, l = 0;
    for (int i = 0; i < sz(t); ++i) {
        while (u && !sa[u].next.count(t[i])) {
            u = sa[u].link;
            l = sa[u].len;
        }
        if (sa[u].next.count(t[i])) u = sa[u].next[t[i]], l++;
        match[u] = max(match[u], l);
    }
    for (int i = sz(sa) - 1; i > 0; --i)
        match[i] = max(match[i], match[sa[i].link]);
    for (int i = 0; i < sz(sa); ++i)
        LCS[i] = min(LCS[i], match[i]);
}

//longest common substring de n cadenas
int lcs_n(vector<string>& t) {
    const int INF = 1e7;
    LCS.assign(sz(sa), INF);
    forn(i, sz(t)) lcsMatch(t[i]);
    return *max_element(all(LCS));
}

//longitud desde 1 hasta N, return v donde v[i] = num distintas
   substr de i longitud
vector<int> substringDistribution(int lenCadena) {
    vector<int> st(lenCadena + 5);
    forn(i, sz(sa)) {
            int l = sa[sa[i].link].len + 1; // l minlen subcadena que
                pertenece al conjunto sa[i]
            int r = sa[i].len; // r maxlen subcadena que pertenece al
                conjunto s[i]
            if (l > 0) st[l]++, st[r + 1]--;
        }
        forn(i, lenCadena + 1) st[i + 1] += st[i];
        return st;
    }
    //Devuelve V, V[i] = max ocurrencias para una subcadena de S de
        longitud i.
    void maxOcurrenciasLengths(int n) { //Llamar antes count_occ
        vector<int> ans(n + 1);
        forn(i, sz(sa)) ans[sa[i].len] = max(ans[sa[i].len],
            sa[i].cnt);
        forn(i, n) cout << ans[i + 1] << endl;
    }
    node& operator[](int i) { return sa[i]; }
};
```

7.10. Manacher

```
// manacher receives a vector of T and returns the vector with the
   size of the palindromes
// ret[2*i] = size of the largest palindrome centered at i
// ret[2*i+1] = size of the largest palindrome centered at i and i+1
//
// Complexities:
// manacher - O(n)
// palindrome - <O(n), O(1)>
// pal_end - O(n)

template<typename T> vector<int> manacher(const T& s) {
    int l = 0, r = -1, n = s.size();
    vector<int> d1(n), d2(n);
    for (int i = 0; i < n; i++) {
        int k = i > r ? 1 : min(d1[l + r - i], r - i);
        while (i + k < n && i - k >= 0 && s[i + k] == s[i - k]) k++;
        d1[i] = k--;
        if (i + k > r) l = i - k, r = i + k;
    }
    l = 0, r = -1;
    for (int i = 0; i < n; i++) {
        int k = i > r ? 0 : min(d2[l + r - i + 1], r - i + 1); k++;
        while (i + k <= n && i - k >= 0 && s[i + k - 1] == s[i - k])
            k++;
        d2[i] = --k;
        if (i + k - 1 > r) l = i - k, r = i + k - 1;
    }
    vector<int> ret(2 * n - 1);
    for (int i = 0; i < n; i++) ret[2 * i] = 2 * d1[i] - 1;
    for (int i = 0; i < n - 1; i++) ret[2 * i + 1] = 2 * d2[i + 1];
    return ret;
}

// checks if string s[i..j] is palindrome
template<typename T> struct palindrome {
    vector<int> man;

    palindrome(const T& s) : man(manacher(s)) {}
    bool query(int i, int j) {
        return man[i + j] >= j - i + 1;
    }
};

// size of the largest palindrome ending in each position
template<typename T> vector<int> pal_end(const T& s) {
    vector<int> ret(s.size());
    palindrome<T> p(s);
    ret[0] = 1;
    for (int i = 1; i < s.size(); i++) {
        ret[i] = min(ret[i - 1] + 2, i + 1);
        while (!p.query(i - ret[i] + 1, i)) ret[i]--;
    }
    return ret;
}

//expansion
int odd(int d, int i, int n) {
    // d=(manacher[2 * i], i)
    int l = i - (d - 1) / 2;
    int r = i + (d - 1) / 2;

    while (l >= 0 && r < n) {
        //process
        l -= 1; r += 1;
    }
    return ((r - 1) - (l + 1) + 2) / 2;
}
int even(int d, int i, int n) {
    // d=(manacher[2 * i+1], i)
    if (i == n - 1) return 0;
    if (d == 0) d = 2;
```

```cpp
    int l = i - d / 2 + 1;
    int r = i + d / 2;

    while (l >= 0 && r < n) {
        //process
        l -= 1; r += 1;
    }
    return ((r - 1) - (l + 1) + 2) / 2;
}


// largest palindrome
string manacher(const string& s) {
    if (sz(s) == 0) return "";

    string curr = "";
    for (auto&& i : s) {
        curr += i;
        curr += "#";
    }

    curr = "@#" + curr + "&";
    vector<ll> pali(sz(curr), 0);
    ll center = 0;
    ll R = 0;
    for (ll i = 1; i < sz(curr) - 1; i++) {
        if (i < R) pali[i] = min(pali[2 * center - i], R - i);
        while (curr[i + (pali[i] + 1)] == curr[i - (pali[i] + 1)])
            pali[i]++;
        if (i + pali[i] > R) {
            center = i;
            R = i + pali[i];
        }
    }
    ll HC = 0, CI = 0;
    for (ll i = 1; i < sz(curr) - 1; i++) {
        if (pali[i] > HC) {
            HC = pali[i];
            CI = i;
        }
    }
    string ans = "";
    if (HC <= 0) return string(1, s[0]);
    for (ll i = CI - HC + 1; i <= CI + HC - 1; i += 2) ans += curr[i];
    return ans;
}
```

7.11. trie

```cpp
// T.count pref(s) number of strings that have a as a prefix
struct trie {
    vector<vector<int>> to;
    vector<int> end, pref;
    int sigma; char norm;
    int lcpsum = 0;
    trie(int sigma_ = 26, char norm_ = 'a') : sigma(sigma_),
        norm(norm_) {
        to = { vector<int>(sigma) };
        end = { 0 }, pref = { 0 };
    }
    void insert(string s) {
        int x = 0;
        for (auto c : s) {
            int& nxt = to[x][c - norm];
            if (!nxt) {
                nxt = to.size();
                to.pb(vector<int>(sigma));
                end.pb(0), pref.pb(0);
            }
            // else lcpsum += pref[nxt];
            x = nxt, pref[x]++;
        }
        end[x]++, pref[0]++;
    }
    void erase(string s) {
        int x = 0;
        for (char c : s) {
            int& nxt = to[x][c - norm];
            x = nxt, pref[x]--;
            if (!pref[x]) nxt = 0;
        }
        end[x]--, pref[0]--;
    }
    int find(string s) {
        int x = 0;
        for (auto c : s) {
            x = to[x][c - norm];
            if (!x) return -1;
        }
        return x;
    }

    int count_pref(string s) {
        int id = find(s);
        return id >= 0 ? pref[id] : 0;
    }

    string kth_word(int k, int x = 0, string s = "") {
        if (k <= end[x])  return s;
        k -= end[x];
        for (int i = 0; i < sigma; i++) {
            int nxt = to[x][i];
            if (!nxt) continue;
            if (k <= pref[nxt]) return kth_word(k, nxt, s + char(i +
                norm));
            k -= pref[nxt];
        }
        return "-1";
    }

};
```

7.12. Kmp

```
//Cuenta las ocurrencias del string p en el string s.

vector<int> prefix_function(string& s) {
    int n = s.size();
    vector<int> pf(n);
    pf[0] = 0;
    for (int i = 1, j = 0; i < n; i++) {
        while (j && s[i] != s[j]) j = pf[j - 1];
        if (s[i] == s[j]) j++;
        pf[i] = j;
    }
    return pf;
}

int kmp(string& s, string& p) {
    int n = s.size(), m = p.size(), cnt = 0;
    vector<int> pf = prefix_function(p);
    for (int i = 0, j = 0; i < n; i++) {
        while (j && s[i] != p[j]) j = pf[j - 1];
        if (s[i] == p[j]) j++;
        if (j == m) {
            cnt++;
            j = pf[j - 1];
        }
    }
    return cnt;
```

```
}
```

7.13. suffixAutomaton1

```
constexpr int MAX = 1e5 + 7;

namespace sam {
    struct node {
        int len, link, cnt, fpos;
        bool acc;
        map<char, int> next;
    };
    int cur, sz;
    vector<node> sa(MAX * 2);

    void add(char c) {
        int at = cur;
        sa[cur].fpos = sa[sz].len = sa[cur].len + 1;
        sa[cur].fpos -= 1, cur = sz++;
        while (at != -1 && !sa[at].next.count(c)) sa[at].next[c] =
            cur, at = sa[at].link;
        if (at == -1) { sa[cur].link = 0; return; }
        int q = sa[at].next[c];
        if (sa[q].len == sa[at].len + 1) { sa[cur].link = q; return; }
        int qq = sz++;
        sa[qq].len = sa[at].len + 1, sa[qq].next = sa[q].next,
            sa[qq].link = sa[q].link;
        sa[qq].fpos = sa[q].fpos;
        while (at != -1 && sa[at].next[c] == q) sa[at].next[c] = qq,
            at = sa[at].link;
        sa[q].link = sa[cur].link = qq;
    }

    void build(string& s) {
        #warning "clear????";
        sa.assign(MAX * 2, node());
        cur = 0, sz = 0, sa[0].len = 0, sa[0].link = -1, sz++;
        for (auto& i : s) add(i);
        int at = cur;
        while (at) sa[at].acc = 1, at = sa[at].link;
    }
    int64_t distinct_substrings() {
        ll ans = 0;
        for (int i = 1; i < sz; i++) ans += sa[i].len -
            sa[sa[i].link].len;
        return ans;
}
int longest_common_substring(string& S, string& T) {
    build(S);
    int at = 0, l = 0, ans = 0, pos = -1;
    for (int i = 0; i < sz(T); i++) {
        while (at && !sa[at].next.count(T[i])) at = sa[at].link, l
            = sa[at].len;
        if (sa[at].next.count(T[i])) at = sa[at].next[T[i]], l++;
        else at = 0, l = 0;
        if (l > ans) ans = l, pos = i;
    }
    return ans;
    // return T.substr(pos - ans + 1, ans);
}
vector<int> LCS, match;
void lcsMatch(string& t) {
    match.assign(MAX, 0);
    int u = 0, l = 0;
    for (int i = 0; i < sz(t); ++i) {
        while (u && !sa[u].next.count(t[i])) u = sa[u].link, l =
            sa[u].len;
        if (sa[u].next.count(t[i])) u = sa[u].next[t[i]], l++;
        match[u] = max(match[u], l);
    }
    for (int i = MAX - 1; i > 0; --i) match[i] = max(match[i],
        match[sa[i].link]);
    for (int i = 0; i < MAX; ++i) LCS[i] = min(LCS[i], match[i]);
}

int lcs_n(vector<string>& t) {
    const int INF = 1e7;
    LCS.assign(MAX, INF);
    forn(i, sz(t)) lcsMatch(t[i]);
    return *max_element(all(LCS));
}

int isSubstr(string& s) {
    int at = 0;
    for (auto& i : s) {
        if (!sa[at].next.count(i)) return 0;
        at = sa[at].next[i];
    }
    return at;
}

int count_occ(int u) {
    if (sa[u].cnt != 0) return sa[u].cnt;
        sa[u].cnt = sa[u].acc;
        for (auto& v : sa[u].next) sa[u].cnt += count_occ(v.s);
        return sa[u].cnt;
    }

    int pos_occ(string& s) {
        int x = sam::isSubstr(s);
        return x ? (abs(sam::sa[x].fpos - sz(s)) + 1) : -1;
    }

    ll dp[2 * MAX];
    ll paths(int i) {
        auto& x = dp[i];
        if (x) return x;
        x = 1;
        for (char j = 'a';j <= 'z';j++) {
            if (sa[i].next.count(j)) x += paths(sa[i].next[j]);
        }
        return x;
    }

    void kth_substring(int k, int at = 0) { // k=1 : menor substring
        lexicog.
        for (int i = 0; i < 26; i++) if (k && sa[at].next.count(i +
            'a')) {
            if (paths(sa[at].next[i + 'a']) >= k) {
                cout << char(i + 'a');
                kth_substring(k - 1, sa[at].next[i + 'a']);
                return;
            }
            k -= paths(sa[at].next[i + 'a']);
        }
    }
};
```

7.14. Min-Max-SuffixCyclic

```
Dado un string s devuelve el indice donde comienza la rotacion
    lexicograficamente menor de s.

int minimum_expression(string s) { //Factorizacion de lyndon
    s = s+s; // si no se concatena devuelve el indice del sufijo menor
    int len = s.size(), i = 0, j = 1, k = 0;
    while (i+k < len && j+k < len) {
        if (s[i+k] == s[j+k]) k++;
        else if (s[i+k] > s[j+k]) i = i+k+1, k = 0; // cambiar por <
            para maximum
        else j = j+k+1, k = 0;
        if (i == j) j++;
    }
    return min(i, j);
}


/*
max_suffix: retorna el inicio del sufijo lexicograficamente mayor
min_suffix: retorna el inicio del sufijo lexicograficamente menor
max_cyclic_shift: retorna el inicio del shift ciclico
    lexicograficamente mayor
min_cyclic_shift: retorna el inicio del shift ciclico
    lexicograficamente menor
*/
template<typename T> int max_suffix(T s, bool mi = false) {
    s.push_back(*min_element(s.begin(), s.end()) - 1);
    int ans = 0;
    for (int i = 1; i < s.size(); i++) {
        int j = 0;
        while (ans + j < i && s[i + j] == s[ans + j]) j++;
        if (s[i + j] > s[ans + j]) {
            if (!mi or i != s.size() - 2) ans = i;
        }
        else if (j) i += j - 1;
    }
    return ans;
}

template<typename T> int min_suffix(T s) {
    for (auto& i : s) i *= -1;
    s.push_back(*max_element(s.begin(), s.end()) + 1);
    return max_suffix(s, true);
}

template<typename T> int max_cyclic_shift(T s) {
    int n = s.size();
    for (int i = 0; i < n; i++) s.push_back(s[i]);
    return max_suffix(s);
}

template<typename T> int min_cyclic_shift(T s) {
    for (auto& i : s) i *= -1;
    return max_cyclic_shift(s);
}
```

7.15. hashing-mint

```
static constexpr int mod = 998244353;
struct mint {
    static constexpr int  m = 998244353;
    // static inline int  m = 998244353; //to change mod
    int x;
    mint() : x(0) {}
    mint(long long x_) :x(x_% m) { if (x < 0) x += m; }
    int val() { return x; }
    mint& operator+=(mint b) { if ((x += b.x) >= m) x -= m; return
        *this; }
    mint& operator-=(mint b) { if ((x -= b.x) < 0) x += m; return
        *this; }
    mint& operator*=(mint b) { x = (long long)(x)*b.x % m; return
        *this; }
    mint pow(long long e) const {
        mint r = 1, b = *this;
        while (e) {
            if (e & 1) r *= b;
            b *= b;
            e >>= 1;
        }
        return r;
    }

    mint inv() { return pow(m - 2); }
    mint& operator/=(mint b) { return *this *= b.pow(m - 2); }
    friend mint operator+ (mint a, mint b) { return a += b; }
    friend mint operator- (mint a, mint b) { return a -= b; }
    friend mint operator/ (mint a, mint b) { return a /= b; }
    friend mint operator* (mint a, mint b) { return a *= b; }
    friend bool operator==(mint a, mint b) { return a.x == b.x; }
    friend bool operator!=(mint a, mint b) { return a.x != b.x; }
    friend bool operator< (mint a, mint b) { return a.x < b.x; }
};

/*
Usage:
        Good values c = 137, modbest=998244353, mod = 10^9 + 7, mod =
            1e18 + 9.
        If necessary to check too many pairs of hashes, use two
        different hashes.
        If hashing something other than english characters:
            - Don't have elements with value 0.
            - Use c > max element value.
```

```
*/
struct Hash {
    mint c, mod;
    vector<mint> h, p;
    Hash(const string& s, ll c, ll mod) : c(c), mod(mod), h(sz(s) + 1
        ), p(sz(s) + 1) {
        // mint::m = mod;
        p[0] = 1;
        h[0] = 0;
        forn(i, sz(s)) {
            h[i + 1] = (c * h[i] + s[i]);
            p[i + 1] = (c * p[i]);
        }
    }
    // Returns hash of interval s[a ... b] (where 0 <= a <= b < sz(s))
    mint get(int a, int b) {
        return (h[b + 1] - ((h[a] * p[b - a + 1])));
    }
};

bool same(Hash& Ha, Hash& Hb, int l, int r) {
    int qa = Ha.get(l, r).x;
    int qb = Hb.get(sz(Hb.h) - 2 - r, sz(Hb.h) - 2 - l).x;
    return qa == qb;
}
```

7.16. Splitear

```
string split(string &in) {
    string result = "";
    regex pattern("[^a-zA-Z]|paraagregarmas|");
    in = regex_replace(in, pattern, " ");
    transform(all(in), in.begin(), ::toupper);
    istringstream iss(in);
    while (iss >> in) {
        result += in;
    }
    return result;
}
```

7.17. Suffix Array 1

```
vector<int> suffix_array(string s) {
    s += "$";
    int MAX = 260, n = sz(s), N = max(n, MAX);
    vector<int> sa(n), ra(n);
    for (int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];
    for (int k = 0; k < n; k ? k *= 2 : k++) {
        vector<int> nsa(sa), nra(n), cnt(N);
        for (int i = 0; i < n; i++) nsa[i] = (nsa[i] - k + n) % n,
            cnt[ra[i]]++;
        for (int i = 1; i < N; i++) cnt[i] += cnt[i - 1];
        for (int i = n - 1; i + 1; i--) sa[--cnt[ra[nsa[i]]]] = nsa[i];
        for (int i = 1, r = 0; i < n; i++) nra[sa[i]] = r += ra[sa[i]]
            != ra[sa[i - 1]] || ra[(sa[i] + k) % n] != ra[(sa[i - 1] +
            k) % n];
        ra = nra;
        if (ra[sa[n - 1]] == n - 1) break;
    }
    return vector<int>(sa.begin() + 1, sa.end());
}

vector<int> kasai(string s, vector<int> sa) {
    int n = sz(s), k = 0;
    vector<int> ra(n + 1), lcp(n);
    for (int i = 0; i < n; i++) ra[sa[i]] = i;
    for (int i = 0; i < n; i++, k -= !!k) {
        if (ra[i] == n - 1) { k = 0; continue; }
        int j = sa[ra[i] + 1];
        while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
        lcp[ra[i]] = k;
    }
    return lcp;
}
/*
find the number of occurrences of the string t in the string s
*/
int find_str(string& s, string& t, vector<int>& sa) {
    int n = sz(s);
    if (sz(t) > n) return 0;
    int L = 0, R = n - 1;
    int nL, nR;
    for (int i = 0; i < sz(t); i++) {
        int l = L, r = R + 1;
        while (l < r) {
            int m = (l + r) / 2;
            if (i + sa[m] >= n || s[i + sa[m]] < t[i]) l = m + 1;
            else r = m;
        }
        if (l == R + 1 || s[i + sa[l]] > t[i]) return 0;
        nL = l, l = L, r = R + 1;

        while (l < r) {
            int m = (l + r) / 2;
            if (i + sa[m] >= n || s[i + sa[m]] <= t[i]) l = m + 1;
            else r = m;
        }
        l--;
        nR = l, L = nL, R = nR;
    }
    return (nL <= nR ? nR - nL + 1 : 0);
}
/*
find the longest common substring what
appear in the string s at least least twice
*/
string lcs(vector<int>& sa, vector<int>& ka, string& s) {
    int idx = max_element(all(ka)) - begin(ka);
    return (ka[idx] > 0 ? s.substr(sa[idx], ka[idx]) : "-1");
}
/*
Find the longest common substring of two given strings s and t
create a new string s + '#' + t
compute the suffix array of the new string
compute the LCP array of the new string
*/
string find_lcs(string& s, string& t, vector<int>& lcp, vector<int>&
    sa) {
    int best = 0, n = sz(s), pos = INT_MAX;
    for (int i = 0; i < sz(lcp) - 1; i++) {
        bool i_s = (0 <= sa[i] && sa[i] <= n - 1);
        bool j_s = (0 <= sa[i + 1] && sa[i + 1] <= n - 1);
        if (i_s != j_s && best < lcp[i]) {
            best = lcp[i];
            pos = min(sa[i], sa[i + 1]);
        }
    }
    return s.substr(pos, best);
}
```

8. Utilities

8.1. util

```
__builtin_popcount(x);  // Cuenta el numero de bits '1' en la
    representacion binaria de x.
__builtin_parity(x);    // Devuelve 1 si el numero de bits '1' en
    la representacion binaria de x es impar, 0 si es par.
__builtin_clz(x);       // Cuenta el numero de bits en '0' a la
    izquierda, desde el bit mas significativo hasta el primer '1'.
__builtin_ctz(x);       // Cuenta el numero de bits en '0' a la
    derecha, desde el bit menos significativo hasta el primer '1'.
__builtin_ffs(x);       // Encuentra la posicion del primer bit en
    '1' (contando desde 1, desde el bit menos significativo).
__lg(x);                // Devuelve el logaritmo en base 2
n & ~(1 << (x - 1));    // Apaga el m-esimo bit de n (bit 1 si m=1
    es el menos significativo), Si m=1, apaga el bit menos
    significativo.
x &(-x);                // Aisla el bit menos significativo en '1'
    de x (devuelve el bit mas bajo en '1' de x).
~x & (x + 1);           // Aisla el bit menos significativo en '0'
    de x.
x | (x + 1);            // Enciende el bit menos significativo en
    '0' de x.
x &(x - 1);             // Apaga el bit menos significativo en '1'
    de x.
-~n;                    // Suma 1 a n.
~- n;                   // Resta 1 a n.
x && (!(x & (x - 1)));  // Comprueba si x es una potencia de 2.
```

8.2. cmd

```
"C:\w64devkit\bin\gdb.exe" !.exe
"C:\w64devkit\bin\g++.exe" -g !.cpp -o !.exe

g++ -o A A.cpp
./A

A < in.txt
A < in.txt > op.txt
```

8.3. template

```
#include <bits/stdc++.h>
using namespace std;
#define endl     '\n'
#define f        first
#define s        second
```

```
#define ins       insert
#define pb        push_back
#define eb        emplace_back
#define sz(x)     int((x).size())
#define all(x)    begin(x), end(x)
typedef long long ll;
#define forn(i, n) for (int i = 0; i < n; ++i)
#define each(i, x) for (auto &&i : x)
#define forne(i,x,n) for (int i = x; i < n; ++i)
#define show(x) for (auto &&i : x) {cerr << i <<' ';} cerr<< endl;


#define LOCAL
void dbg_out() { cerr << ']' << endl; }
template<typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) { cerr << H;if (sizeof...(T)) cerr <<
    ',' << ' '; dbg_out(T...); }
#ifdef LOCAL
#define dbg(...) cerr << '|' << __LINE__ << '|'<< '{' << #__VA_ARGS__
    << '}'<<':'<<' '<<'[', dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif


int main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);












    cout << flush;
    return 0;
}
```

8.4. Pragma

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
```

8.5. segment whit the maximum sum

```
/*
segment whit the maximum sum
add to segment tree the node struct
T neutro = T();
T oper(T a, T b) {node::get(a, b);}
Check the base ans
*/
constexpr int inf=(1e18);
struct node {
    int lt, rt, sum, ans;
    node() : lt(-inf), rt(-inf), sum(0), ans(-inf) {}
    node(int x) : sum(x) {
        lt = rt = ans = (x);
    }
    static node get(node &a, node&b) {
        node res;
        res.sum = a.sum + b.sum;
        res.lt = max(a.lt, a.sum + b.lt);
        res.rt = max(b.rt, b.sum + a.rt);
        res.ans = max({ a.ans, b.ans, a.rt + b.lt });
        return res;
    }
};
```

8.6. random

```
int rnd(int l, int r) {
    static std::mt19937
       rng(std::chrono::steady_clock::now().time_since_epoch().count());
    return std::uniform_int_distribution<int>(l, r)(rng);
}
```