

ICPC - Notebook

Contents

1 data structures	2	7.8 Fast Exp	16
1.1 Ordered Set Gnu Pbds	2	7.9 Lcm-using-factorization	16
1.2 Segtree Rmq Lazy Max Update	2	7.10 Euler-phi	16
1.3 Segtree Rmq Lazy Range	2	7.11 Polynomial	16
1.4 Segtree Point Rmq	3	7.12 Integer Mod	16
1.5 Sparse-segment-tree	3	7.13 Count Divisors Memo	17
1.6 Segtree Rsq Lazy Range Sum	4	7.14 Lcm	17
1.7 Segtree Rxq Lazy Range Xor	5	7.15 Factorial-factorization	17
1.8 Dsu	5	7.16 Factorization-with-primes	17
1.9 Dsu	5	7.17 Modular-inverse-using-phi	17
1.10 Sparse Table Rmq	6	7.18 Factorization	18
2 graphs	6	7.19 Gcd	18
2.1 Scc-nodes-(kosajaru)	6	7.20 Combinatorics With Repetitions	18
2.2 2-sat-(struct)	6	8 strings	18
2.3 Floyd Warshall	7	8.1 Rabin-karp	18
2.4 Topological-sorting	7	8.2 Trie-naive	18
2.5 Lowest Common Ancestor Sparse Table	7	8.3 String-psum	19
2.6 Count-scc-(kosajaru)	8		
2.7 Kruskal	8		
2.8 Scc-(struct)	8		
2.9 Check-bipartite	9		
2.10 Dijkstra	9		
3 extras	9		
3.1 Binary To Gray	9		
3.2 Bigint	9		
3.3 Get-permutation-cicles	12		
4 dynamic programming	12		
4.1 Edit Distance	12		
4.2 Money Sum Bottom Up	12		
4.3 Knapsack Dp Values 01	13		
4.4 Tsp	13		
5 trees	13		
5.1 Maximum-distances	13		
5.2 Heavy-light-decomposition	14		
5.3 Tree Diameter	14		
6 searching	15		
6.1 Ternary Search Recursive	15		
7 math	15		
7.1 Power-sum	15		
7.2 Sieve-list-primes	15		
7.3 Factorial	15		
7.4 Permutation-count	15		
7.5 N-choose-k-count	15		
7.6 Gcd-using-factorization	15		
7.7 Is-prime	15		

1 data structures

1.1 Ordered Set Gnu Pbds

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 template<typename T>
5 // using ordered_set = tree<T, null_type, less<T>,
   rb_tree_tag, tree_order_statistics_node_update>;
6
7 // if you want to find the elements less or equal :p
8 using ordered_set = tree<T, null_type, less_equal<T>,
   rb_tree_tag, tree_order_statistics_node_update>;
```

1.2 Segtree Rmq Lazy Max Update

```
1 struct SegmentTree
2 {
3     int N;
4     vll ns, lazy;
5     SegmentTree(const vll &xs)
6         : N(xs.size()), ns(4 * N, 0), lazy(4
   * N, 0)
7     {
8         for (size_t i = 0; i < xs.size(); ++
   i)
9             {
10                 update(i, i, xs[i]);
11             }
12     }
13     void
14     update(int a, int b, ll value)
15     {
16         update(1, 0, N - 1, a, b, value);
17     }
18     void
19     update(int node, int L, int R, int a, int b,
   ll value)
20     {
21         if (lazy[node])
22         {
23             ns[node] = max (ns[
   node], lazy[node]);
24             if (L < R)
25             {
26                 lazy
   [2 * node]
27                 =
28                 max (lazy[2 * node],
29                     lazy[node]);
30                 lazy[2 * node + 1]
31                 =
32                 max (lazy[2 * node + 1],
33                     lazy[node]);
34             }
35             lazy[node] = 0;
36             if (a > R or b < L)
37                 return;
38             if (a <= L and R <= b)
39             {
40                 ns[node] = max (ns[
   node], value);
41                 if (L < R)
42                 {
43                     lazy
   [2 * node] = max (
44                     value, lazy[2 * node]);
45                     lazy[2 * node + 1] = max (
   value, lazy[2 * node + 1]);
```

```
46             }
47             return;
48         }
49         update (2 * node, L, (L + R) / 2, a,
   b, value);
50         update (2 * node + 1, (L + R) / 2 +
   1, R, a, b, value);
51         ns[node] = max (ns[node * 2], ns[node
   * 2 + 1]);
52     }
53
54     ll
55     RMQ (int a, int b)
56     {
57         return RMQ (1, 0, N - 1, a, b);
58     }
59     ll
60     RMQ (int node, int L, int R, int a, int b)
61     {
62         if (lazy[node])
63         {
64             ns[node] = max (ns[
   node], lazy[node]);
65             if (L < R)
66             {
67                 lazy[
   node * 2]
68                 =
69                 max (lazy[node * 2],
70                     lazy[node]);
71                 lazy[
   node * 2 + 1]
72                 =
73                 max (lazy[node * 2 + 1],
74                     lazy[node]);
75             }
76             lazy[node] = 0;
77             if (a > R or b < L)
78                 return 0;
79             if (a <= L and R <= b)
80                 return ns[node];
81             ll x = RMQ (2 * node, L, (L + R) / 2,
   a, b);
82             ll y = RMQ (2 * node + 1, (L + R) / 2
   + 1, R, a, b);
83             return max (x, y);
84         }
85     };
```

1.3 Segtree Rmq Lazy Range

```
1 struct SegmentTree {
2     int N;
3     vll ns, lazy;
4     SegmentTree(const vll &xs)
5         : N(xs.size()), ns(4 * N, INT_MAX),
   lazy(4 * N, 0) {
6         for (size_t i = 0; i < xs.size(); ++i
   ) update(i, i, xs[i]);
7     }
8     void update(int a, int b, ll value) {
9         update(1, 0, N - 1, a, b, value);
10    }
11    void update(int node, int L, int R, int a,
   int b, ll value) {
12        if (lazy[node]) {
13            ns[node] = ns[node] ==
   INT_MAX ? lazy[node]
14            : ns[node] + lazy[node];
15            if (L < R) {
16                lazy[2 * node] +=
```

```

17         lazy[2 * node + 1] += 17
        lazy[node];
18     }
19     lazy[node] = 0;
20
21     }
22     if (a > R or b < L) return;
23     if (a <= L and R <= b) {
24         ns[node] =
25         ns[node] == INT_MAX ?
26         value : ns[node] + value;
27         if (L < R) {
28             lazy[2 * node] +=
29             lazy[2 * node + 1] +=
30             value;
31             value;
32         }
33         return;
34     }
35     update(2 * node, L, (L + R) / 2, a, b
36     , value);
37     update(2 * node + 1, (L + R) / 2 + 1,
38     R, a, b, value);
39     ns[node] = min(ns[2 * node], ns[2 *
40     node + 1]);
41     }
42     ll RMQ(int a, int b) { return RMQ(1, 0, N -
43     1, a, b); }
44     ll RMQ(int node, int L, int R, int a, int b)
45     {
46         if (lazy[node]) {
47             ns[node] = ns[node] ==
48             INT_MAX ? lazy[node]
49             : ns[node] + lazy[node];
50             if (L < R) {
51                 lazy[2 * node] +=
52                 lazy[2 * node + 1] +=
53                 lazy[node];
54             }
55             lazy[node] = 0;
56         }
57         if (a > R or b < L) return INT_MAX;
58         if (a <= L and R <= b) return ns[node]
59     ];
60     ll x = RMQ(2 * node, L, (L + R) / 2,
61     a, b);
62     ll y = RMQ(2 * node + 1, (L + R) / 2
63     + 1, R, a, b);
64     return min(x, y);
65     }
66 }

```

1.4 Segtree Point Rmq

```

1 class SegTree {
2 public:
3     int n;
4     vector<ll> st;
5     SegTree(const vector<ll> &v) : n((int)v.size()),
6     st(n * 4 + 1, LLONG_MAX) {
7         for (int i = 0; i < n; ++i) update(i, v[i]);
8     }
9     void update(int p, ll v) { update(1, 0, n - 1, p,
10     v); }
11     ll RMQ(int l, int r) { return RMQ(1, 0, n - 1, l,
12     r); }
13 private:
14     void update(int node, int l, int r, int p, ll v)
15     {
16         if (p < l or p > r) return; // fora do
17         intervalo.
18         if (l == r) {
19             st[node] = v;

```

```

        return;
    }
    int mid = l + (r - l) / 2;
    update(node * 2, l, mid, p, v);
    update(node * 2 + 1, mid + 1, r, p, v);
    st[node] = min(st[node * 2], st[node * 2 +
1]);
}
ll RMQ(int node, int L, int R, int l, int r) {
    if (l <= L and r >= R) return st[node];
    if (L > r or R < l) return LLONG_MAX;
    if (L == R) return st[node];
    int mid = L + (R - L) / 2;
    return min(RMQ(node * 2, L, mid, l, r),
        RMQ(node * 2 + 1, mid + 1, R, l, r
    ));
}
};

```

1.5 Sparse-segment-tree

```

1 #include <bits/stdc++.h>
2 #pragma GCC optimize("O3")
3 #define FOR(i, x, y) for (int i = x; i < y; i++)
4 #define MOD 1000000007
5 typedef long long ll;
6 using namespace std;
7
8 struct Node {
9     int sum, lazy, tl, tr, l, r;
10     Node() : sum(0), lazy(0), l(-1), r(-1) {}
11 };
12
13 const int MAXN = 123456;
14 Node segtree[64 * MAXN];
15 int cnt = 2;
16 const ll v = 10;
17
18 void push_lazy(int node) {
19     if (segtree[node].lazy) {
20         segtree[node].sum = (segtree[node].tr - segtree[
21         node].tl + 1) * v;
22         int mid = (segtree[node].tl + segtree[node].tr) /
23         2;
24         if (segtree[node].l == -1) {
25             segtree[node].l = cnt++;
26             segtree[segtree[node].l].tl = segtree[node].tl;
27             segtree[segtree[node].l].tr = mid;
28         }
29         if (segtree[node].r == -1) {
30             segtree[node].r = cnt++;
31             segtree[segtree[node].r].tl = mid + 1;
32             segtree[segtree[node].r].tr = segtree[node].tr;
33         }
34         segtree[segtree[node].l].lazy = segtree[segtree[
35         node].r].lazy = v;
36         segtree[node].lazy = 0;
37     }
38
39     void update(int node, int l, int r) {
40         push_lazy(node);
41         if (l == segtree[node].tl && r == segtree[node].tr)
42         {
43             segtree[node].lazy = v;
44             push_lazy(node);
45         } else {
46             int mid = (segtree[node].tl + segtree[node].tr) /
47             2;
48             if (segtree[node].l == -1) {
49                 segtree[node].l = cnt++;

```

```

46     segtree[segtree[node].l].tl = segtree[node].tl;
47     segtree[segtree[node].l].tr = mid;
48 }
49 if (segtree[node].r == -1) {
50     segtree[node].r = cnt++;
51     segtree[segtree[node].r].tl = mid + 1;
52     segtree[segtree[node].r].tr = segtree[node].tr;
53 }
54
55 if (l > mid)
56     update(segtree[node].r, l, r);
57 else if (r <= mid)
58     update(segtree[node].l, l, r);
59 else {
60     update(segtree[node].l, l, mid);
61     update(segtree[node].r, mid + 1, r);
62 }
63
64 push_lazy(segtree[node].l);
65 push_lazy(segtree[node].r);
66 segtree[node].sum =
67     segtree[segtree[node].l].sum + segtree[segtree[
68     node].r].sum;
69 }
70
71 int query(int node, int l, int r) {
72     push_lazy(node);
73     if (l == segtree[node].tl && r == segtree[node].tr)
74         return segtree[node].sum;
75     else {
76         int mid = (segtree[node].tl + segtree[node].tr) /
77             2;
78         if (segtree[node].l == -1) {
79             segtree[node].l = cnt++;
80             segtree[segtree[node].l].tl = segtree[node].tl;
81             segtree[segtree[node].l].tr = mid;
82         }
83         if (segtree[node].r == -1) {
84             segtree[node].r = cnt++;
85             segtree[segtree[node].r].tl = mid + 1;
86             segtree[segtree[node].r].tr = segtree[node].tr;
87         }
88         if (l > mid)
89             return query(segtree[node].r, l, r);
90         else if (r <= mid)
91             return query(segtree[node].l, l, r);
92         else
93             return query(segtree[node].l, l, mid) +
94                 query(segtree[node].r, mid + 1, r);
95     }
96 }
97 int main() {
98     ios::sync_with_stdio(false);
99     cin.tie(0);
100     int m;
101     cout << "enter m: ";
102     cin >> m;
103
104     segtree[1].sum = 0;
105     segtree[1].lazy = 0;
106     segtree[1].tl = 1;
107     segtree[1].tr = 1e9;
108
109     int c = 0;
110     FOR(_, 0, m) {
111         int d, x, y;
112         cout << "enter d, x, y: ";
113         cin >> d >> x >> y;
114         if (d == 1) {
115             c = query(1, x, y);
116             cout << "c: ";
117             cout << c << '\n';
118         } else {
119             update(1, x, y);
120         }

```

1.6 Segtree Rsq Lazy Range Sum

```

1 struct SegTree {
2     int N;
3     vector<ll> ns, lazy;
4
5     SegTree(const vector<ll> &xs)
6         : N(xs.size()), ns(4 * N, 0), lazy(4
7         * N, 0) {
8         for (size_t i = 0; i < xs.size(); ++i
9         ) update(i, i, xs[i]);
10    }
11
12    void update(int a, int b, ll value) {
13        update(1, 0, N - 1, a, b, value);
14    }
15
16    void update(int node, int L, int R, int a,
17    int b, ll value) {
18        // Lazy propagation
19        if (lazy[node]) {
20            ns[node] += (R - L + 1) *
21            lazy[node];
22
23            if (L < R) // Se o nó não é
24            uma folha, propaga
25            {
26                lazy[2 * node] +=
27                lazy[node];
28                lazy[2 * node + 1] +=
29                lazy[node];
30            }
31            lazy[node] = 0;
32        }
33        if (a > R or b < L) return;
34        if (a <= L and R <= b) {
35            ns[node] += (R - L + 1) *
36            value;
37
38            if (L < R) {
39                lazy[2 * node] +=
40                value;
41                lazy[2 * node + 1] +=
42                value;
43            }
44            return;
45        }
46        update(2 * node, L, (L + R) / 2, a, b
47        , value);
48        update(2 * node + 1, (L + R) / 2 + 1,
49        R, a, b, value);
50
51        ns[node] = ns[2 * node] + ns[2 * node
52        + 1];
53    }
54
55    ll RSQ(int a, int b) { return RSQ(1, 0, N -
56    1, a, b); }
57
58    ll RSQ(int node, int L, int R, int a, int b)
59    {
60        if (lazy[node]) {
61            ns[node] += (R - L + 1) *
62            lazy[node];
63
64            if (L < R) {
65                lazy[2 * node] +=

```

```

55         lazy[2 * node + 1] += + 1];
        lazy[node];           }
56     }                       46
57                             47
58         lazy[node] = 0;     48
59     }                       49
60                             50
61         if (a > R or b < L) return 0; 51
62         if (a <= L and R <= b) return ns[node] 52
63 ];                           53
64                             54
65         ll x = RSQ(2 * node, L, (L + R) / 2, 55
a, b);                         ll y = RSQ(2 * node + 1, (L + R) / 2
+ 1, R, a, b);               56
67                             57
68         return x + y;       58
69     }                       59
70 };                           60

```

1.7 Segtree Rxq Lazy Range Xor

```

1 struct SegTree {
2     int N;
3     vector<ll> ns, lazy;
4
5     SegTree(const vector<ll> &xs)
6         : N(xs.size()), ns(4 * N, 0), lazy(4
* N, 0) {
7         for (size_t i = 0; i < xs.size(); ++i
) update(i, i, xs[i]);
8     }
9
10    void update(int a, int b, ll value) {
11        update(1, 0, N - 1, a, b, value);
12    }
13
14    void update(int node, int L, int R, int a,
int b, ll value) {
15        // Lazy propagation
16        if (lazy[node]) {
17            ns[node] ^= lazy[node];
18
19            if (L < R) // Se o nó não é
uma folha, propaga
20            {
21                lazy[2 * node] ^=
lazy[node];
22                lazy[2 * node + 1] ^=
lazy[node];
23            }
24            lazy[node] = 0;
25        }
26        if (a > R or b < L) return;
27        if (a <= L and R <= b) {
28            ns[node] ^= value;
29
30            if (L < R) {
31                lazy[2 * node] ^=
value;
32                lazy[2 * node + 1] ^=
value;
33            }
34            return;
35        }
36        update(2 * node, L, (L + R) / 2, a, b
, value);
37        update(2 * node + 1, (L + R) / 2 + 1,
R, a, b, value);
38        ns[node] = ns[2 * node] ^ ns[2 * node

```

```

11 rxq(int a, int b) { return RSQ(1, 0, N -
1, a, b); }
12
13 rxq(int node, int L, int R, int a, int b)
{
14     if (lazy[node]) {
15         ns[node] ^= lazy[node];
16
17         if (L < R) {
18             lazy[2 * node] ^=
lazy[node];
19             lazy[2 * node + 1] ^=
lazy[node];
20         }
21         lazy[node] = 0;
22     }
23     if (a > R or b < L) return 0;
24     if (a <= L and R <= b) return ns[node
];
25
26     ll x = rxq(2 * node, L, (L + R) / 2,
a, b);
27     ll y = rxq(2 * node + 1, (L + R) / 2
+ 1, R, a, b);
28     return x ^ y;
29 }
30 };

```

1.8 Dsu

```

1 class DSU:
2     def __init__(self, n):
3         self.n = n
4         self.p = [x for x in range(0, n + 1)]
5         self.size = [0 for i in range(0, n + 1)]
6
7     def find_set(self, x): # log n
8         if self.p[x] == x:
9             return x
10        else:
11            self.p[x] = self.find_set(self.p[x])
12            return self.p[x]
13
14    def same_set(self, x, y): # log n
15        return bool(self.find_set(x) == self.find_set
(y))
16
17    def union_set(self, x, y): # log n
18        px = self.find_set(x)
19        py = self.find_set(y)
20
21        if px == py:
22            return
23
24        size_x = self.size[px]
25        size_y = self.size[py]
26
27        if size_x > size_y:
28            self.p[py] = self.p[px]
29            self.size[px] += self.size[py]
30        else:
31            self.p[px] = self.p[py]
32            self.size[py] += self.size[px]

```

1.9 Dsu

```

1 struct DSU {
2     vector<int> ps;
3     vector<int> size;

```

```

4 DSU(int N) : ps(N + 1), size(N + 1, 1) { iota(ps. 10
    begin(), ps.end(), 0); } 11
5 int find_set(int x) { return ps[x] == x ? x : ps[x] 12
    = find_set(ps[x]); }
6 bool same_set(int x, int y) { return find_set(x) == 13
    find_set(y); }
7 void union_set(int x, int y) {
8     if (same_set(x, y))
9         return;
10
11     int px = find_set(x);
12     int py = find_set(y);
13
14     if (size[px] < size[py])
15         swap(px, py);
16
17     ps[py] = px;
18     size[px] += size[py];
19 }
20 };

```

1.10 Sparse Table Rmq

```

1 /*
2     Sparse table implementation for rmq.
3     build: O(NlogN)
4     query: O(1)
5 */
6 int fastlog2(ll x) {
7     ull i = x;
8     return i ? __builtin_clzll(1) -
9         __builtin_clzll(i) : -1;
10 }
11 template <typename T>
12 class SparseTable {
13 public:
14     int N;
15     int K;
16     vector<vector<T>> st;
17     SparseTable(vector<T> vs)
18         : N((int)vs.size()),
19           K(fastlog2(N) + 1),
20           st(K + 1, vector<T>(N + 1)) {
21         copy(vs.begin(), vs.end(), st[0].
22             begin());
23
24         for (int i = 1; i <= K; ++i)
25             for (int j = 0; j + (1 << i)
26                 <= N; ++j)
27                 st[i][j] = min(st[i -
28                     1][j],
29                     st[i -
30                     1][j + (1 << (i - 1))]);
31         T RMQ(int l, int r) { // [l, r], 0 indexed
32             int i = fastlog2(r - l + 1);
33             return min(st[i][l], st[i][r - (1 <<
34                 i) + 1]);
35         }
36     };
37 };

```

2 graphs

2.1 Scc-nodes-(kosajaru)

```

1 /*
2 * O(n+m)
3 * Returns a pair <a, b>
4 * a: number of SCCs
5 * b: vector of size n, where b[i] is the SCC id
6 * of node i
7 */
8 void dfs(ll u, vchar &visited, const vll2d &g, vll &
9     scc, bool buildScc, ll id,
10     vll &sccid) {
11     visited[u] = true;

```

```

12     sccid[u] = id;
13     for (auto &v : g[u])
14         if (!visited[v]) dfs(v, visited, g,
15             scc, buildScc, id, sccid);
16
17     // if it's the first pass, add the node to
18     the scc
19     if (buildScc) scc.eb(u);
20 }
21
22 pair<ll, vll> kosajaru(vll2d &g) {
23     ll n = len(g);
24     vll scc;
25     vchar vis(n);
26     vll sccid(n);
27     for (ll i = 0; i < n; i++)
28         if (!vis[i]) dfs(i, vis, g, scc, true
29             , 0, sccid);
30
31     // build the transposed graph
32     vll2d gt(n);
33     for (int i = 0; i < n; ++i)
34         for (auto &v : g[i]) gt[v].eb(i);
35
36     // run the dfs on the previous scc order
37     ll id = 1;
38     vis.assign(n, false);
39     for (ll i = len(scc) - 1; i >= 0; i--)
40         if (!vis[scc[i]]) {
41             dfs(scc[i], vis, gt, scc,
42                 false, id++, sccid);
43         }
44     return {id - 1, sccid};
45 }

```

2.2 2-sat-(struct)

```

1 struct SAT2 {
2     ll n;
3     vll2d adj, adj_t;
4     vc used;
5     vll order, comp;
6     vc assignment;
7     bool solvable;
8     SAT2(ll _n)
9         : n(2 * _n),
10           adj(n),
11           adj_t(n),
12           used(n),
13           order(n),
14           comp(n, -1),
15           assignment(n / 2) {}
16     void dfs1(int v) {
17         used[v] = true;
18         for (int u : adj[v]) {
19             if (!used[u]) dfs1(u);
20         }
21         order.push_back(v);
22     }
23
24     void dfs2(int v, int cl) {
25         comp[v] = cl;
26         for (int u : adj_t[v]) {
27             if (comp[u] == -1) dfs2(u, cl);
28         }
29     }
30
31     bool solve_2SAT() {
32         // find and label each SCC
33         for (int i = 0; i < n; ++i) {
34             if (!used[i]) dfs1(i);
35         }
36         reverse(all(order));
37         ll j = 0;
38         for (auto &v : order) {
39             if (comp[v] == -1) dfs2(v, j++);
40         }

```

```

41     assignment.assign(n / 2, false);
42     for (int i = 0; i < n; i += 2) {
43         // x and !x belong to the same SCC
44         if (comp[i] == comp[i + 1]) {
45             solvable = false;
46             return false;
47         }
48     }
49     assignment[i / 2] = comp[i] > comp[i +
50 1];
51 }
52 solvable = true;
53 return true;
54 }
55
56 void add_disjunction(int a, bool na, int b, bool
57 nb) {
58     a = (2 * a) ^ na;
59     b = (2 * b) ^ nb;
60     int neg_a = a ^ 1;
61     int neg_b = b ^ 1;
62     adj[neg_a].push_back(b);
63     adj[neg_b].push_back(a);
64     adj_t[b].push_back(neg_a);
65     adj_t[a].push_back(neg_b);
66 };

```

2.3 Floyd Warshall

```

1 vector<vll> floyd_warshall(const vector<vll> &adj, ll
2 n) {
3     auto dist = adj;
4     for (int i = 0; i < n; ++i) {
5         for (int j = 0; j < n; ++j) {
6             for (int k = 0; k < n; ++k) {
7                 dist[j][k] = min(dist
8 [j][k],
9 dist
10 [j][i] + dist[i][k]);
11 }
12 }
13 return dist;
14 }

```

2.4 Topological-sorting

```

1 /*
2 * O(V)
3 * assumes:
4 *     * vertices have index [0, n-1]
5 * if is a DAG:
6 *     * returns a topological sorting
7 * else:
8 *     * returns an empty vector
9 * */
10 enum class state { not_visited, processing, done };
11 bool dfs(const vector<vll> &adj, ll s, vector<state>
12 &states, vll &order) {
13     states[s] = state::processing;
14     for (auto &v : adj[s]) {
15         if (states[v] == state::not_visited)
16             if (not dfs(adj, v, states,
17 order)) return false;
18     } else if (states[v] == state::
19 processing)
20         return false;
21     }
22     states[s] = state::done;
23     order.pb(s);
24     return true;
25 }
26 vll topologicalSorting(const vector<vll> &adj) {

```

```

24     ll n = len(adj);
25     vll order;
26     vector<state> states(n, state::not_visited);
27     for (int i = 0; i < n; ++i) {
28         if (states[i] == state::not_visited)
29             if (not dfs(adj, i, states,
30 order)) return {};
31     }
32     reverse(all(order));
33     return order;
34 }

```

2.5 Lowest Common Ancestor Sparse Table

```

1 int fastlog2(ll x) {
2     ull i = x;
3     return i ? __builtin_clzll(1) -
4         __builtin_clzll(i) : -1;
5 }
6 template <typename T>
7 class SparseTable {
8 public:
9     int N;
10    int K;
11    vector<vector<T>> st;
12    SparseTable(vector<T> vs)
13        : N((int)vs.size()),
14          K(fastlog2(N) + 1),
15          st(K + 1, vector<T>(N + 1)) {
16        copy(vs.begin(), vs.end(), st[0].
17        begin());
18
19        for (int i = 1; i <= K; ++i)
20            for (int j = 0; j + (1 << i)
21                <= N; ++j)
22                st[i][j] = min(st[i -
23 1][j],
24 st[i -
25 1][j + (1 << (i - 1))]);
26    }
27    SparseTable() {}
28    T RMQ(int l, int r) {
29        int i = fastlog2(r - l + 1);
30        return min(st[i][l], st[i][r - (1 <<
31 i) + 1]);
32    }
33 };
34 class LCA {
35 public:
36     int p;
37     int n;
38     vi first;
39     vector<char> visited;
40     vi vertices;
41     vi height;
42     SparseTable<int> st;
43
44     LCA(const vector<vi> &g)
45         : p(0),
46           n((int)g.size()),
47           first(n + 1),
48           visited(n + 1, 0),
49           height(n + 1) {
50         build_dfs(g, 1, 1);
51         st = SparseTable<int>(vertices);
52     }
53
54     void build_dfs(const vector<vi> &g, int u,
55 int hi) {
56         visited[u] = true;
57         height[u] = hi;
58         first[u] = vertices.size();
59         vertices.push_back(u);
60         for (auto uv : g[u]) {
61             if (!visited[uv]) {

```

```

55         build_dfs(g, uv, hi + 24
56         1);
57     };
58     }
59 }
60
61 int lca(int a, int b) {
62     int l = min(first[a], first[b]);
63     int r = max(first[a], first[b]);
64     return st.RMQ(l, r);
65 }
66 };

```

2.6 Count-scc-(kosajaru)

```

1 void dfs(ll u, vchar &visited, const vll2d &g, vll &
  scc, bool buildScc) {
2     visited[u] = true;
3     for (auto &v : g[u])
4         if (!visited[v]) dfs(v, visited, g,
  scc, buildScc);
5
6     // if it's the first pass, add the node to
  the scc
7     if (buildScc) scc.eb(u);
8 }
9
10 ll kosajaru(vll2d &g) {
11     ll n = len(g);
12     vll scc;
13     vchar vis(n);
14     for (ll i = 0; i < n; i++)
15         if (!vis[i]) dfs(i, vis, g, scc, true
  );
16
17     // build the transposed graph
18     vll2d gt(n);
19     for (int i = 0; i < n; ++i)
20         for (auto &v : g[i]) gt[v].eb(i);
21
22     // run the dfs on the previous scc order
23     ll scccnt = 0;
24     vis.assign(n, false);
25     for (ll i = len(scc) - 1; i >= 0; i--)
26         if (!vis[scc[i]]) dfs(scc[i], vis, gt
  , scc, false), scccnt++;
27     return scccnt;
28 }

```

2.7 Kruskal

```

1 class DSU:
2     def __init__(self, n):
3         self.n = n
4         self.p = [x for x in range(0, n + 1)]
5         self.size = [0 for i in range(0, n + 1)]
6
7     def find_set(self, x):
8         if self.p[x] == x:
9             return x
10        else:
11            self.p[x] = self.find_set(self.p[x])
12            return self.p[x]
13
14    def same_set(self, x, y):
15        return bool(self.find_set(x) == self.find_set
  (y))
16
17    def union_set(self, x, y):
18        px = self.find_set(x)
19        py = self.find_set(y)
20
21        if px == py:
22            return
23

```

```

24        size_x = self.size[px]
25        size_y = self.size[py]
26
27        if size_x > size_y:
28            self.p[py] = self.p[px]
29            self.size[px] += self.size[py]
30        else:
31            self.p[px] = self.p[py]
32            self.size[py] += self.size[px]
33
34    def kruskal(gv, n):
35        """
36        Receives te list of edges as a list of tuple in
  the form:
37            d, u, v
38            d: distance between u and v
39            And also n as the total of verties.
40        """
41        dsu = DSU(n)
42
43        c = 0
44        for e in gv:
45            d, u, v = e
46            if not dsu.same_set(u, v):
47                c += d
48                dsu.union_set(u, v)
49
50        return c
51

```

2.8 Scc-(struct)

```

1 struct SCC {
2     ll N;
3     vll2d adj, tadj;
4     vll todo, comps, comp;
5     vector<set<ll>> sccadj;
6     vchar vis;
7     SCC(ll _N)
8         : N(_N), adj(_N), tadj(_N), comp(_N,
  -1), sccadj(_N), vis(_N) {}
9
10    void add_edge(ll x, ll y) { adj[x].eb(y),
  tadj[y].eb(x); }
11
12    void dfs(ll x) {
13        vis[x] = 1;
14        for (auto &y : adj[x])
15            if (!vis[y]) dfs(y);
16        todo.pb(x);
17    }
18    void dfs2(ll x, ll v) {
19        comp[x] = v;
20        for (auto &y : tadj[x])
21            if (comp[y] == -1) dfs2(y, v)
  ;
22    }
23    void gen() {
24        for (ll i = 0; i < N; ++i)
25            if (!vis[i]) dfs(i);
26        reverse(all(todo));
27        for (auto &x : todo)
28            if (comp[x] == -1) {
29                dfs2(x, x);
30                comps.pb(x);
31            }
32    }
33
34    void genSCCGraph() {
35        for (ll i = 0; i < N; ++i) {
36            for (auto &j : adj[i]) {
37                if (comp[i] != comp[j]
  ) {
38                    sccadj[comp[i]
  ].insert(comp[j]);
39                }
40            }
41        }
42    }

```



```

41         }
42     }
43 };

```

2.9 Check-bipartite

```

1 // O(V)
2 bool checkBipartite(const ll n, const vector<vll> &
    adj) {
3     ll s = 0;
4     queue<ll> q;
5     q.push(s);
6     vll color(n, INF);
7     color[s] = 0;
8     bool isBipartite = true;
9     while (!q.empty() && isBipartite) {
10         ll u = q.front();
11         q.pop();
12         for (auto &v : adj[u]) {
13             if (color[v] == INF) {
14                 color[v] = 1 - color[u];
15                 q.push(v);
16             } else if (color[v] == color[u]) {
17                 return false;
18             }
19         }
20     }
21     return true;
22 }

```

2.10 Dijkstra

```

1 ll __inf = LLONG_MAX >> 5;
2 vll dijkstra(const vector<vector<pll>> &g, ll n) {
3     priority_queue<pll, vector<pll>, greater<pll>
    >> pq;
4     vll dist(n, __inf);
5     vector<char> vis(n);
6     pq.emplace(0, 0);
7     dist[0] = 0;
8     while (!pq.empty()) {
9         auto [d1, v] = pq.top();
10        pq.pop();
11        if (vis[v]) continue;
12        vis[v] = true;
13
14        for (auto [d2, u] : g[v]) {
15            if (dist[u] > d1 + d2) {
16                dist[u] = d1 + d2;
17                pq.emplace(dist[u], u);
18            }
19        }
20    }
21    return dist;
22 }

```

3 extras

3.1 Binary To Gray

```

1 string binToGray(string bin){
2     string gray(bin.size(), '0');
3     int n = bin.size()-1;
4     gray[0] = bin[0];
5     for(int i = 1; i <= n; i++){
6         gray[i] = '0'+(bin[i-1]=='1')^(bin[i]=='1');
7     }
8     return gray;
9 }

```

3.2 Bigint

```

1 const int maxn = 1e2 + 14, lg = 15;
2 const int base = 1000000000;
3 const int base_digits = 9;
4 struct bigint {
5     vector<int> a;
6     int sign;
7
8     int size() {
9         if (a.empty()) return 0;
10        int ans = (a.size() - 1) *
    base_digits;
11        int ca = a.back();
12        while (ca) ans++, ca /= 10;
13        return ans;
14    }
15    bigint operator^(const bigint &v) {
16        bigint ans = 1, a = *this, b = v;
17        while (!b.isZero()) {
18            if (b % 2) ans *= a;
19            a *= a, b /= 2;
20        }
21        return ans;
22    }
23    string to_string() {
24        stringstream ss;
25        ss << *this;
26        string s;
27        ss >> s;
28        return s;
29    }
30    int sumof() {
31        string s = to_string();
32        int ans = 0;
33        for (auto c : s) ans += c - '0';
34        return ans;
35    }
36    /*</arpa>*/
37    bigint() : sign(1) {}
38
39    bigint(long long v) { *this = v; }
40
41    bigint(const string &s) { read(s); }
42
43    void operator=(const bigint &v) {
44        sign = v.sign;
45        a = v.a;
46    }
47
48    void operator=(long long v) {
49        sign = 1;
50        a.clear();
51        if (v < 0) sign = -1, v = -v;
52        for (; v > 0; v = v / base) a.
    push_back(v % base);
53    }
54
55    bigint operator+(const bigint &v) const {
56        if (sign == v.sign) {
57            bigint res = v;
58
59            for (int i = 0, carry = 0;
60                 i < (int)max(a.size(), v
    .a.size()) || carry; ++i) {
61                if (i == (int)res.a.
    size()) res.a.push_back(0);
62                res.a[i] +=
63                    carry + (i <
    (int)a.size() ? a[i] : 0);
64                carry = res.a[i] >=
    base;
65                if (carry) res.a[i]
    -= base;
66            }
67            return res;
68        }
69        return *this - (-v);
70    }

```

```

71         bigint operator-(const bigint &v) const {
72             if (sign == v.sign) {
73                 if (abs() >= v.abs()) {
74                     bigint res = *this;
75                     for (int i = 0, carry = 0; i < (int)v.a.size(); ++i) {
76                         res.a[i] -= v.a[i];
77                         if (res.a[i] < 0) {
78                             res.a[i] += base;
79                             carry = 1;
80                         }
81                     }
82                     return res;
83                 }
84                 return -(v - *this);
85             }
86             return *this + (-v);
87         }
88
89         void operator+=(int v) {
90             if (v < 0) sign = -sign, v = -v;
91             for (int i = 0, carry = 0; i < (int)a.size(); ++i) {
92                 if (i == (int)a.size()) a.push_back(0);
93                 long long cur = a[i] + (long long)v + carry;
94                 carry = (int)(cur / base);
95                 a[i] = (int)(cur % base);
96                 // asm("divl %%ecx" : "=a" : "A"(cur), "c"(base));
97             }
98             trim();
99         }
100
101         bigint operator*(int v) const {
102             bigint res = *this;
103             res *= v;
104             return res;
105         }
106
107         void operator*=(long long v) {
108             if (v < 0) sign = -sign, v = -v;
109             if (v > base) {
110                 *this = *this * (v / base) *
111                 base + *this * (v % base);
112                 return;
113             }
114             for (int i = 0, carry = 0; i < (int)a.size(); ++i) {
115                 if (i == (int)a.size()) a.push_back(0);
116                 long long cur = a[i] * (long long)v + carry;
117                 carry = (int)(cur / base);
118                 a[i] = (int)(cur % base);
119                 // asm("divl %%ecx" : "=a" : "A"(cur), "c"(base));
120             }
121             trim();
122         }
123
124         bigint operator*(long long v) const {
125             bigint res = *this;
126             res *= v;
127             return res;
128         }
129
130         friend pair<bigint, bigint> divmod(const
131         bigint &a1, const bigint &b1) {
132             int norm = base / (b1.a.back() + 1);
133             bigint a = a1.abs() * norm;
134             bigint b = b1.abs() * norm;
135             bigint q, r;
136             q.a.resize(a.a.size());
137
138             for (int i = a.a.size() - 1; i >= 0; i--) {
139                 r *= base;
140                 r += a.a[i];
141                 int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
142                 int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
143                 int d = ((long long)base * s1 + s2) / b.a.back();
144                 r -= b * d;
145                 while (r < 0) r += b, --d;
146                 q.a[i] = d;
147             }
148
149             q.sign = a1.sign * b1.sign;
150             r.sign = a1.sign;
151             q.trim();
152             r.trim();
153             return make_pair(q, r / norm);
154         }
155
156         bigint operator/(const bigint &v) const {
157             return divmod(*this, v).first;
158         }
159
160         bigint operator%(const bigint &v) const {
161             return divmod(*this, v).second;
162         }
163
164         void operator/=(int v) {
165             if (v < 0) sign = -sign, v = -v;
166             for (int i = (int)a.size() - 1, rem = 0; i >= 0; --i) {
167                 long long cur = a[i] + rem * (long long)base;
168                 a[i] = (int)(cur / v);
169                 rem = (int)(cur % v);
170             }
171             trim();
172         }
173
174         bigint operator/(int v) const {
175             bigint res = *this;
176             res /= v;
177             return res;
178         }
179
180         int operator%(int v) const {
181             if (v < 0) v = -v;
182             int m = 0;
183             for (int i = a.size() - 1; i >= 0; --i) {
184                 m = (a[i] + m * (long long)base) % v;
185                 return m * sign;
186             }
187         }
188
189         void operator+=(const bigint &v) { *this = *this + v; }
190         void operator-=(const bigint &v) { *this = *this - v; }
191         void operator*=(const bigint &v) { *this = *this * v; }

```

```

196     void operator/=(const bigint &v) { *this = * 257
this / v; } 258
197
198     bool operator<(const bigint &v) const { 259
199         if (sign != v.sign) return sign < v. 260
sign;
200         if (a.size() != v.a.size()) 261
201             return a.size() * sign < v.a.
size() * v.sign; 262
202         for (int i = a.size() - 1; i >= 0; i 263
-- ) 264
203             if (a[i] != v.a[i]) return a[265
i] * sign < v.a[i] * sign; 266
204             return false; 267
205     }
206
207     bool operator>(const bigint &v) const { 268
return v < *this; } 269
208     bool operator<=(const bigint &v) const { 270
return !(v < *this); } 271
209     bool operator>=(const bigint &v) const { 272
return !(*this < v); } 273
210     bool operator==(const bigint &v) const { 274
211         return !(*this < v) && !(v < *this); 275
212     } 276
213     bool operator!=(const bigint &v) const { 277
214         return *this < v || v < *this; 278
215     } 279
216     void trim() { 280
217         while (!a.empty() && !a.back()) a. 281
pop_back(); 282
218         if (a.empty()) sign = 1; 283
219     } 284
220
221     bool isZero() const { return a.empty() || (a.285
size() == 1 && !a[0]); } 286
222
223     bigint operator-() const { 287
224         bigint res = *this; 288
225         res.sign = -sign; 289
226         return res; 290
227     } 291
228
229     bigint abs() const { 292
230         bigint res = *this; 293
231         res.sign *= res.sign; 294
232         return res; 295
233     } 296
234
235     long long longValue() const { 297
236         long long res = 0; 298
237         for (int i = a.size() - 1; i >= 0; i 299
-- ) res = res * base + a[i]; 300
238         return res * sign; 301
239     } 302
240
241     friend bigint gcd(const bigint &a, const 303
bigint &b) { 304
242         return b.isZero() ? a : gcd(b, a % b)305
; 306
243     } 307
244     friend bigint lcm(const bigint &a, const 308
bigint &b) { 309
245         return a / gcd(a, b) * b; 310
246     } 311
247
248     void read(const string &s) { 312
249         sign = 1; 313
250         a.clear(); 314
251         int pos = 0; 315
252         while (pos < (int)s.size() && 316
(s[pos] == '-' || s[pos] == '+') 317
{ 318
253             if (s[pos] == '-') sign = - 319
sign; 320
254             ++pos; 321
255         } 322
256     } 323

```

```

    }
    for (int i = s.size() - 1; i >= pos;
i -= base_digits) {
        int x = 0;
        for (int j = max(pos, i -
base_digits + 1); j <= i; j++)
            x = x * 10 + s[j] - '
0';
        a.push_back(x);
    }
    trim();
}

friend istream &operator>>(istream &stream,
bigint &v) {
    string s;
    stream >> s;
    v.read(s);
    return stream;
}

friend ostream &operator<<(ostream &stream,
const bigint &v) {
    if (v.sign == -1) stream << '-';
    stream << (v.a.empty() ? 0 : v.a.back
());
    for (int i = (int)v.a.size() - 2; i
>= 0; --i)
        stream << setw(base_digits)
<< setfill('0') << v.a[i];
    return stream;
}

static vector<int> convert_base(const vector<
int> &a, int old_digits,
                                int
new_digits) {
    vector<long long> p(max(old_digits,
new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < (int)p.size(); i
++) p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int)a.size(); i
++) {
        cur += a[i] * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >=
new_digits) {
            res.push_back(int(cur
% p[new_digits]));
            cur /= p[new_digits];
            cur_digits -=
new_digits;
        }
        res.push_back((int)cur);
        while (!res.empty() && !res.back())
res.pop_back();
        return res;
    }

    typedef vector<long long> vll;

    static vll karatsubaMultiply(const vll &a,
const vll &b) {
        int n = a.size();
        vll res(n + n);
        if (n <= 32) {
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n
; j++)
                    res[i + j] +=
a[i] * b[j];
            return res;

```

```

314     }
315
316     int k = n >> 1;
317     vll a1(a.begin(), a.begin() + k);
318     vll a2(a.begin() + k, a.end());
319     vll b1(b.begin(), b.begin() + k);
320     vll b2(b.begin() + k, b.end());
321
322     vll a1b1 = karatsubaMultiply(a1, b1);
323     vll a2b2 = karatsubaMultiply(a2, b2);
324
325     for (int i = 0; i < k; i++) a2[i] +=
a1[i];
326     for (int i = 0; i < k; i++) b2[i] +=
b1[i];
327
328     vll r = karatsubaMultiply(a2, b2);
329     for (int i = 0; i < (int)a1b1.size();
i++) r[i] -= a1b1[i];
330     for (int i = 0; i < (int)a2b2.size();
i++) r[i] -= a2b2[i];
331
332     for (int i = 0; i < (int)r.size(); i
++) res[i + k] += r[i];
333     for (int i = 0; i < (int)a1b1.size();
i++) res[i] += a1b1[i];
334     for (int i = 0; i < (int)a2b2.size();
i++)
335         res[i + n] += a2b2[i];
336     return res;
337 }
338
339 bigint operator*(const bigint &v) const {
340     vector<int> a6 = convert_base(this->a
, base_digits, 6);
341     vector<int> b6 = convert_base(v.a,
base_digits, 6);
342     vll a(a6.begin(), a6.end());
343     vll b(b6.begin(), b6.end());
344     while (a.size() < b.size()) a.
push_back(0);
345     while (b.size() < a.size()) b.
push_back(0);
346     while (a.size() & (a.size() - 1))
347         a.push_back(0), b.push_back
(0);
348     vll c = karatsubaMultiply(a, b);
349     bigint res;
350     res.sign = sign * v.sign;
351     for (int i = 0, carry = 0; i < (int)c
.size(); i++) {
352         long long cur = c[i] + carry;
353         res.a.push_back((int)(cur %
1000000));
354         carry = (int)(cur / 1000000);
355     }
356     res.a = convert_base(res.a, 6,
base_digits);
357     res.trim();
358     return res;
359 }
360 };

```

3.3 Get-permutation-cicles

```

1  /*
2  * receives a permutation [0, n-1]
3  * returns a vector of cycles
4  * for example: [ 1, 0, 3, 4, 2] -> [[0, 1], [2, 3,
4]]
5  * */
6  vector<vll> getPermutationCicles(const vll &ps) {
7      ll n = len(ps);
8      vector<char> visited(n);
9      vector<vll> cicles;
10     for (int i = 0; i < n; ++i) {
11         if (visited[i]) continue;

```

```

12
13         vll cicle;
14         ll pos = i;
15         while (!visited[pos]) {
16             cicle.pb(pos);
17             visited[pos] = true;
18             pos = ps[pos];
19         }
20
21         cicles.push_back(vll(all(cicle)));
22     }
23     return cicles;
24 }

```

4 dynamic programming

4.1 Edit Distance

```

1  int edit_distance(const string &a, const string &b) {
2      int n = a.size();
3      int m = b.size();
4      vector<vi> dp(n + 1, vi(m + 1, 0));
5
6      int ADD = 1, DEL = 1, CHG = 1;
7      for (int i = 0; i <= n; ++i) {
8          dp[i][0] = i * DEL;
9      }
10     for (int i = 1; i <= m; ++i) {
11         dp[0][i] = ADD * i;
12     }
13
14     for (int i = 1; i <= n; ++i) {
15         for (int j = 1; j <= m; ++j) {
16             int add = dp[i][j - 1] + ADD;
17             int del = dp[i - 1][j] + DEL;
18             int chg = dp[i - 1][j - 1] +
(a[i - 1] == b[j -
1] ? 0 : 1) * CHG;
19             dp[i][j] = min({add, del, chg
});
20         }
21     }
22     return dp[n][m];
23 }
24 }
25 }

```

4.2 Money Sum Bottom Up

```

1  /*
2  find every possible sum using
3  the given values only once.
4  */
5  set<int> money_sum(const vi &xs) {
6      using vc = vector<char>;
7      using vvc = vector<vc>;
8      int _m = accumulate(all(xs), 0);
9      int _n = xs.size();
10     vvc _dp(_n + 1, vc(_m + 1, 0));
11     set<int> _ans;
12     _dp[0][xs[0]] = 1;
13     for (int i = 1; i < _n; ++i) {
14         for (int j = 0; j <= _m; ++j) {
15             if (j == 0 or _dp[i - 1][j])
16                 _dp[i][j + xs[i]] =
1;
17             _dp[i][j] = 1;
18         }
19     }
20     return _ans;
21 }
22
23 for (int i = 0; i < _n; ++i)
24     for (int j = 0; j <= _m; ++j)
25         if (_dp[i][j]) _ans.insert(j);
26
27 return _ans;

```

```
26 }
```

4.3 Knapsack Dp Values 01

```
1 const int MAX_N = 1001;
2 const int MAX_S = 100001;
3 array<array<int, MAX_S>, MAX_N> dp;
4 bool check[MAX_N][MAX_S];
5 pair<int, vi> knapsack(int S, const vector<pii> &xs)
6 {
7     int N = (int)xs.size();
8     for (int i = 0; i <= N; ++i) dp[i][0] = 0;
9     for (int m = 0; m <= S; ++m) dp[0][m] = 0;
10    for (int i = 1; i <= N; ++i) {
11        for (int m = 1; m <= S; ++m) {
12            dp[i][m] = dp[i - 1][m];
13            check[i][m] = false;
14
15            auto [w, v] = xs[i - 1];
16
17            if (w <= m and (dp[i - 1][m -
18                w] + v) >= dp[i][m]) {
19                dp[i][m] = dp[i - 1][
20                    m - w] + v;
21                check[i][m] = true;
22            }
23        }
24    }
25    int m = S;
26    vi es;
27
28    for (int i = N; i >= 1; --i) {
29        if (check[i][m]) {
30            es.push_back(i);
31            m -= xs[i - 1].first;
32        }
33    }
34    reverse(es.begin(), es.end());
35
36    return {dp[N][S], es};
37 }
38
39 }
```

4.4 Tsp

```
1 using vi = vector<int>;
2 vector<vi> dist;
3 vector<vi> memo;
4 /* 0 ( N^2 * 2^N )*/
5 int tsp(int i, int mask, int N) {
6     if (mask == (1 << N) - 1) return dist[i][0];
7     if (memo[i][mask] != -1) return memo[i][mask];
8
9     int ans = INT_MAX << 1;
10    for (int j = 0; j < N; ++j) {
11        if (mask & (1 << j)) continue;
12        auto t = tsp(j, mask | (1 << j), N) +
13            dist[i][j];
14        ans = min(ans, t);
15    }
16    return memo[i][mask] = ans;
17 }
18
19 }
```

5 trees

5.1 Maximum-distances

```
1 /*
2  * Returns the maximum distance from every node to
3  * any other node in the tree.
4  */
```

```
4 pll mostDistantFrom(const vector<vll> &adj, ll n, ll
5     root) {
6     // 0 indexed
7     ll mostDistantNode = root;
8     ll nodeDistance = 0;
9     queue<pll> q;
10    vector<char> vis(n);
11    q.emplace(root, 0);
12    vis[root] = true;
13    while (!q.empty()) {
14        auto [node, dist] = q.front();
15        q.pop();
16        if (dist > nodeDistance) {
17            nodeDistance = dist;
18            mostDistantNode = node;
19        }
20        for (auto u : adj[node]) {
21            if (!vis[u]) {
22                vis[u] = true;
23                q.emplace(u, dist +
24                    1);
25            }
26        }
27    }
28    return {mostDistantNode, nodeDistance};
29 }
30
31 ll twoNodesDist(const vector<vll> &adj, ll n, ll a,
32     ll b) {
33     queue<pll> q;
34     vector<char> vis(n);
35     q.emplace(a, 0);
36     while (!q.empty()) {
37         auto [node, dist] = q.front();
38         q.pop();
39         if (node == b) return dist;
40         for (auto u : adj[node]) {
41             if (!vis[u]) {
42                 vis[u] = true;
43                 q.emplace(u, dist +
44                     1);
45             }
46         }
47     }
48     return -1;
49 }
50
51 tuple<ll, ll, ll> tree_diameter(const vector<vll> &
52     adj, ll n) {
53     // returns two points of the diameter and the
54     // diameter itself
55     auto [node1, dist1] = mostDistantFrom(adj, n,
56         0);
57     auto [node2, dist2] = mostDistantFrom(adj, n,
58         node1);
59     auto diameter = twoNodesDist(adj, n, node1,
60         node2);
61     return make_tuple(node1, node2, diameter);
62 }
63
64 vll everyDistanceFromNode(const vector<vll> &adj, ll
65     n, ll root) {
66     // Single Source Shortest Path, from a given
67     // root
68     queue<pair<ll, ll>> q;
69     vll ans(n, -1);
70     ans[root] = 0;
71     q.emplace(root, 0);
72     while (!q.empty()) {
73         auto [u, d] = q.front();
74         q.pop();
75
76         for (auto w : adj[u]) {
77             if (ans[w] != -1) continue;
78             ans[w] = d + 1;
79             q.emplace(w, d + 1);
80         }
81     }
82 }
```

```

70     }
71     return ans;
72 }
73
74 vll maxDistances(const vector<vll> &adj, ll n) {
75     auto [node1, node2, diameter] = tree_diameter
76     (adj, n);
77     auto distances1 = everyDistanceFromNode(adj,
78     n, node1);
79     auto distances2 = everyDistanceFromNode(adj,
80     n, node2);
81     vll ans(n);
82     for (int i = 0; i < n; ++i) ans[i] = max(
83     distances1[i], distances2[i]);
84     return ans;
85 }

```

5.2 Heavy-light-decomposition

```

1 // iagorrr ;)
2 #include <bits/stdc++.h>
3 using namespace std;
4 #ifdef DEBUG
5 #include "debug.cpp"
6 #else
7 #define dbg(...) 666
8 #endif
9 #define endl '\n'
10 #define fastio \
11     ios_base::sync_with_stdio(false); \
12     cin.tie(0); \
13     cout.tie(0);
14 #define rep(i, l, r) for (int i = (l); i < (r); i++)
15 #define len(__x) (ll) __x.size()
16 using ll = long long;
17 using vll = vector<ll>;
18 using pll = pair<ll, ll>;
19 using vll2d = vector<vll>;
20 using vi = vector<int>;
21 using vi2d = vector<vi>;
22 using pii = pair<int, int>;
23 using vii = vector<pii>;
24 using vc = vector<char>;
25 #define all(a) a.begin(), a.end()
26 #define INV(xxxx) \
27     for (auto &xxx : xxxx) cin >> xxx;
28 #define PRINTV(___x) \
29     for_each(all(___x), [](ll &___x) { cout << ___x
30     << ' '; }, cout << '\n');
31 #define snd second
32 #define fst first
33 #define pb(___x) push_back(___x)
34 #define mp(___a, ___b) make_pair(___a, ___b)
35 #define eb(___x) emplace_back(___x)
36 #define rsz(___x, ___n) resize(___x, ___n)
37 const ll INF = 1e18;
38
39 struct HLD {
40     int n;
41     vi sizes;
42     vi2d g;
43     vi groups;
44     vi heavy;
45     HLD(int n) : n(n), sizes(n + 1), g(n + 1), groups
46     (n + 1), heavy(n + 1) {}
47     void get_sizes(int u, int p) {
48         int sz = 1;
49         int bigc = -1;
50         for (auto &v : g[u])
51             if (v != p) {
52                 get_sizes(v, u);
53                 if (bigc == -1 or sizes[bigc] < sizes
54                     [v])
55                     bigc = v, heavy[u] = v;
56                 sz += sizes[v];
57             }
58     }
59 }

```

```

56     sizes[u] = sz;
57 }
58 void decompose(int u, int p) {
59     groups[u] = p;
60     for (auto &v : g[u])
61         if (v != p) {
62             if (v == heavy[u])
63                 decompose(v, p);
64             else
65                 decompose(v, v);
66         }
67 }
68 };
69 void run() {}
70 int32_t main(void) {
71     fastio;
72     int t;
73     t = 1;
74     // cin >> t;
75     while (t--) run();
76 }

```

5.3 Tree Diameter

```

1 pll mostDistantFrom(const vector<vll> &adj, ll n, ll
2 root) {
3     // 0 indexed
4     ll mostDistantNode = root;
5     ll nodeDistance = 0;
6     queue<pll> q;
7     vector<char> vis(n);
8     q.emplace(root, 0);
9     vis[root] = true;
10    while (!q.empty()) {
11        auto [node, dist] = q.front();
12        q.pop();
13        if (dist > nodeDistance) {
14            nodeDistance = dist;
15            mostDistantNode = node;
16        }
17        for (auto u : adj[node]) {
18            if (!vis[u]) {
19                vis[u] = true;
20                q.emplace(u, dist +
21                    1);
22            }
23        }
24    }
25    return {mostDistantNode, nodeDistance};
26 }
27
28 pll twoNodesDist(const vector<vll> &adj, ll n, ll a,
29 ll b) {
30     // 0 indexed
31     queue<pll> q;
32     vector<char> vis(n);
33     q.emplace(a, 0);
34     while (!q.empty()) {
35         auto [node, dist] = q.front();
36         q.pop();
37         if (node == b) {
38             return dist;
39         }
40         for (auto u : adj[node]) {
41             if (!vis[u]) {
42                 vis[u] = true;
43                 q.emplace(u, dist +
44                     1);
45             }
46         }
47     }
48     return -1;
49 }
50
51 pll tree_diameter(const vector<vll> &adj, ll n) {
52     // 0 indexed !!!
53     auto [node1, dist1] = mostDistantFrom(adj, n,
54     0);
55     auto [node2, dist2] = mostDistantFrom(adj, n,

```

```

    node1);
49     auto diameter = twoNodesDist(adj, n, node1,
node2);
50     return diameter;
51 }

```

6 searching

6.1 Ternary Search Recursive

```

1 const double eps = 1e-6;
2
3 // IT MUST BE AN UNIMODAL FUNCTION
4 double f(int x) { return x * x + 2 * x + 4; }
5
6 double ternary_search(double l, double r) {
7     if (fabs(f(l) - f(r)) < eps) return f((l + (r
- l) / 2.0));
8
9     auto third = (r - l) / 3.0;
10    auto m1 = l + third;
11    auto m2 = r - third;
12
13    // change the signal to find the maximum
point.
14    return m1 < m2 ? ternary_search(m1, r) :
ternary_search(l, m2);
15 }

```

7 math

7.1 Power-sum

```

1 // calculates  $K^0 + K^1 \dots + K^n$ 
2 ll fastpow(ll a, int n) {
3     if (n == 1) return a;
4     ll x = fastpow(a, n / 2);
5     return x * x * (n & 1 ? a : 1);
6 }
7 ll powersum(ll n, ll k) { return (fastpow(n, k + 1) -
1) / (n - 1); }

```

7.2 Sieve-list-primes

```

1 // lsit every prime until MAXN
2 const ll MAXN = 1e5;
3 vll list_primes(ll n) { // Nlog * log N
4     vll ps;
5     bitset<MAXN> sieve;
6     sieve.set();
7     sieve.reset(1);
8     for (ll i = 2; i <= n; ++i) {
9         if (sieve[i]) ps.push_back(i);
10        for (ll j = i * 2; j <= n; j += i) {
11            sieve.reset(j);
12        }
13    }
14    return ps;
15 }

```

7.3 Factorial

```

1 const ll MAX = 18;
2 vll fv(MAX, -1);
3 ll factorial(ll n) {
4     if (fv[n] != -1) return fv[n];
5     if (n == 0) return 1;
6     return n * factorial(n - 1);
7 }

```

7.4 Permutation-count

```

1 const ll MAX = 18;
2 vll fv(MAX, -1);
3 ll factorial(ll n) {
4     if (fv[n] != -1) return fv[n];
5     if (n == 0) return 1;
6     return n * factorial(n - 1);
7 }
8
9 template <typename T>
10 ll permutation_count(vector<T> xs) {
11     map<T, ll> h;
12     for (auto xi : xs) h[xi]++;
13     ll ans = factorial((ll)xs.size());
14     dbg(ans);
15     for (auto [v, cnt] : h) {
16         dbg(cnt);
17         ans /= cnt;
18     }
19
20     return ans;
21 }

```

7.5 N-choose-k-count

```

1 /*
2  * O(nm) time, O(m) space
3  * equal to n choose k
4  */
5 ll binom(ll n, ll k) {
6     if (k > n) return 0;
7     vll dp(k + 1, 0);
8     dp[0] = 1;
9     for (ll i = 1; i <= n; i++)
10         for (ll j = k; j > 0; j--) dp[j] = dp
[j] + dp[j - 1];
11     return dp[k];
12 }

```

7.6 Gcd-using-factorization

```

1 // O(sqrt(n))
2 map<ll, ll> factorization(ll n) {
3     map<ll, ll> ans;
4     for (ll i = 2; i * i <= n; i++) {
5         ll count = 0;
6         for (; n % i == 0; count++, n /= i)
7             ;
8         if (count) ans[i] = count;
9     }
10    if (n > 1) ans[n]++;
11    return ans;
12 }
13
14 ll gcd_with_factorization(ll a, ll b) {
15     map<ll, ll> fa = factorization(a);
16     map<ll, ll> fb = factorization(b);
17     ll ans = 1;
18     for (auto fai : fa) {
19         ll k = min(fai.second, fb[fai.first])
;
20         while (k--> 0) ans *= fai.first;
21     }
22     return ans;
23 }

```

7.7 Is-prime

```

1 bool isprime(ll n) { // O(sqrt(n))
2     if (n < 2) return false;
3     if (n == 2) return true;
4     if (n % 2 == 0) return false;
5     for (ll i = 3; i * i < n; i += 2)
6         if (n % i == 0) return false;
7     return true;
8 }

```


7.8 Fast Exp

```
1 /*
2  Fast exponentiation algorithm,
3  compute a^n in O(log(n))
4 */
5 ll fexp(ll a, int n){
6     if(n == 0) return 1;
7     if (n==1) return a;
8     ll x = fexp(a, n/2);
9     return x*x*(n&1?a:1);
10 }
```

7.9 Lcm-using-factorization

```
1 map<ll, ll> factorization(ll n) {
2     map<ll, ll> ans;
3     for (ll i = 2; i * i <= n; i++) {
4         ll count = 0;
5         for (; n % i == 0; count++, n /= i)
6             ;
7         if (count) ans[i] = count;
8     }
9     if (n > 1) ans[n]++;
10    return ans;
11 }
12
13 ll lcm_with_factorization(ll a, ll b) {
14     map<ll, ll> fa = factorization(a);
15     map<ll, ll> fb = factorization(b);
16     ll ans = 1;
17     for (auto fai : fa) {
18         ll k = max(fai.second, fb[fai.first]);
19         ;
20         while (k--) ans *= fai.first;
21     }
22     return ans;
23 }
```

7.10 Euler-phi

```
1 const ll MAXN = 1e5;
2 vll list_primes(ll n) { // Nlog * log N
3     vll ps;
4     bitset<MAXN> sieve;
5     sieve.set();
6     sieve.reset(1);
7     for (ll i = 2; i <= n; ++i) {
8         if (sieve[i]) ps.push_back(i);
9         for (ll j = i * 2; j <= n; j += i) {
10             sieve.reset(j);
11         }
12     }
13     return ps;
14 }
15
16 vector<pll> factorization(ll n, const vll &primes) {
17     vector<pll> ans;
18     for (auto &p : primes) {
19         if (n == 1) break;
20         ll cnt = 0;
21         while (n % p == 0) {
22             cnt++;
23             n /= p;
24         }
25         if (cnt) ans.emplace_back(p, cnt);
26     }
27     return ans;
28 }
29
30 ll phi(ll n, vector<pll> factors) {
31     if (n == 1) return 1;
32     ll ans = n;
33
34     for (auto [p, k] : factors) {
35         ans /= p;
```

```
36         ans *= (p - 1);
37     }
38
39     return ans;
40 }
```

7.11 Polynomial

```
1 using polynomial = vector<ll>;
2 int degree(const polynomial &xs) { return xs.size() - 1; }
3 ll horner_evaluate(const polynomial &xs, ll x) {
4     ll ans = 0;
5     ll n = degree(xs);
6     for (int i = n; i >= 0; --i) {
7         ans *= x;
8         ans += xs[i];
9     }
10    return ans;
11 }
12 polynomial operator+(const polynomial &a, const
13     polynomial &b) {
14     int n = degree(a);
15     int m = degree(b);
16     polynomial r(max(n, m) + 1, 0);
17
18     for (int i = 0; i <= n; ++i) r[i] += a[i];
19     for (int j = 0; j <= m; ++j) r[j] += b[j];
20     while (!r.empty() and r.back() == 0) r.
21         pop_back();
22     if (r.empty()) r.push_back(0);
23     return r;
24 }
25 polynomial operator*(const polynomial &p, const
26     polynomial &q) {
27     int n = degree(p);
28     int m = degree(q);
29     polynomial r(n + m + 1, 0);
30     for (int i = 0; i <= n; ++i)
31         for (int j = 0; j <= m; ++j) r[i + j]
32             += (p[i] * q[j]);
33     return r;
34 }
```

7.12 Integer Mod

```
1 const ll INF = 1e18;
2 const ll mod = 998244353;
3 template <ll MOD = mod>
4 struct Modular {
5     ll value;
6     static const ll MOD_value = MOD;
7
8     Modular(ll v = 0) {
9         value = v % MOD;
10        if (value < 0) value += MOD;
11    }
12
13    Modular(ll a, ll b) : value(0) {
14        *this += a;
15        *this /= b;
16    }
17
18    Modular& operator+=(Modular const& b) {
19        value += b.value;
20        if (value >= MOD) value -= MOD;
21        return *this;
22    }
23
24    Modular& operator-=(Modular const& b) {
25        value -= b.value;
26        if (value < 0) value += MOD;
27        return *this;
28    }
29
30    Modular& operator*=(Modular const& b) {
31        value = (ll)value * b.value % MOD;
32        return *this;
33    }
34 }
```



```

32     friend Modular mexp(Modular a, ll e) {
33         Modular res = 1;
34         while (e) {
35             if (e & 1) res *= a;
36             a *= a;
37             e >>= 1;
38         }
39         return res;
40     }
41     friend Modular inverse(Modular a) { return
mexp(a, MOD - 2); }
42
43     Modular& operator/=(Modular const& b) {
44     return *this *= inverse(b); }
45     friend Modular operator+(Modular a, Modular
const b) { return a += b; }
46     Modular operator++(int) {
47         return this->value = (this->value +
1) % MOD;
48     }
49     Modular operator++() { return this->value = (
this->value + 1) % MOD; }
50     friend Modular operator-(Modular a, Modular
const b) { return a -= b; }
51     friend Modular operator-(Modular const a) {
return 0 - a; }
52     Modular operator--(int) {
53         return this->value = (this->value - 1
+ MOD) % MOD;
54     }
55     Modular operator--() {
56         return this->value = (this->value - 1
+ MOD) % MOD;
57     }
58     friend Modular operator*(Modular a, Modular
const b) { return a *= b; }
59     friend Modular operator/(Modular a, Modular
const b) { return a /= b; }
60     friend std::ostream& operator<<(std::ostream&
os, Modular const& a) {
61         return os << a.value;
62     }
63     friend bool operator==(Modular const& a,
Modular const& b) {
64         return a.value == b.value;
65     }
66     friend bool operator!=(Modular const& a,
Modular const& b) {
67         return a.value != b.value;
68     }
69 };

```

7.13 Count Divisors Memo

```

1  const ll mod = 1073741824;
2  const ll maxd = 100 * 100 * 100 + 1;
3  vector<ll> memo(maxd, -1);
4  ll countdivisors(ll x) {
5      ll ox = x;
6      ll ans = 1;
7      for (ll i = 2; i <= x; ++i) {
8          if (memo[x] != -1) {
9              ans *= memo[x];
10             break;
11         }
12         ll count = 0;
13         while (x and x % i == 0) {
14             x /= i;
15             count++;
16         }
17         ans *= (count + 1);
18     }
19     memo[ox] = ans;
20     return ans;
21 }

```

7.14 Lcm

```

1  ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }
2  ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }

```

7.15 Factorial-factorization

```

1  // O(logN) greater k that p^k | n
2  ll E(ll n, ll p) {
3      ll k = 0, b = p;
4      while (b <= n) {
5          k += n / b;
6          b *= p;
7      }
8      return k;
9  }
10
11 // lsit every prime until MAXN O(Nlog * log N)
12 const ll MAXN = 1e5;
13 vll list_primes(ll n) {
14     vll ps;
15     bitset<MAXN> sieve;
16     sieve.set();
17     sieve.reset(1);
18     for (ll i = 2; i <= n; ++i) {
19         if (sieve[i]) ps.push_back(i);
20         for (ll j = i * 2; j <= n; j += i)
21             sieve.reset(j);
22     }
23     return ps;
24 }
25 // O(pi(N)*logN)
26 map<ll, ll> factorial_factorization(ll n, const vll &
primes) {
27     map<ll, ll> fs;
28     for (const auto &p : primes) {
29         if (p > n) break;
30         fs[p] = E(n, p);
31     }
32     return fs;
33 }

```

7.16 Factorization-with-primes

```

1  // Nlog * log N
2  const ll MAXN = 1e5;
3  vll list_primes(ll n) {
4      vll ps;
5      bitset<MAXN> sieve;
6      sieve.set();
7      sieve.reset(1);
8      for (ll i = 2; i <= n; ++i) {
9          if (sieve[i]) ps.push_back(i);
10         for (ll j = i * 2; j <= n; j += i)
11             sieve.reset(j);
12     }
13     return ps;
14 }
15 // O(pi(sqrt(n)))
16 map<ll, ll> factorization(ll n, const vll &primes) {
17     map<ll, ll> ans;
18     for (auto p : primes) {
19         if (p * p > n) break;
20         ll count = 0;
21         for (; n % p == 0; count++, n /= p)
22             ;
23         if (count) ans[p] = count;
24     }
25     return ans;
26 }

```

7.17 Modular-inverse-using-phi

```

1 map<ll, ll> factorization(ll n) {
2     map<ll, ll> ans;
3     for (ll i = 2; i * i <= n; i++) {
4         ll count = 0;
5         for (; n % i == 0; count++, n /= i)
6             ;
7         if (count) ans[i] = count;
8     }
9     if (n > 1) ans[n]++;
10    return ans;
11 }
12
13 ll phi(ll n) {
14     if (n == 1) return 1;
15
16     auto fs = factorization(n);
17     auto res = n;
18
19     for (auto [p, k] : fs) {
20         res /= p;
21         res *= (p - 1);
22     }
23
24     return res;
25 }
26
27 ll fexp(ll a, ll n, ll mod) {
28     if (n == 0) return 1;
29     if (n == 1) return a;
30     ll x = fexp(a, n / 2, mod);
31     return x * x * (n & 1 ? a : 1) % mod;
32 }
33
34 ll inv(ll a, ll mod) { return fexp(a, phi(mod) - 1, mod); }

```

7.18 Factorization

```

1 // O(sqrt(n))
2 map<ll, ll> factorization(ll n) {
3     map<ll, ll> ans;
4     for (ll i = 2; i * i <= n; i++) {
5         ll count = 0;
6         for (; n % i == 0; count++, n /= i)
7             ;
8         if (count) ans[i] = count;
9     }
10    if (n > 1) ans[n]++;
11    return ans;
12 }

```

7.19 Gcd

```

1 ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }

```

7.20 Combinatorics With Repetitions

```

1 void combinations_with_repetition(int n, int k,
2     function<void(const
3     vector<int> &> process) {
4     vector<int> v(k, 1);
5     int pos = k - 1;
6
7     while (true) {
8         process(v);
9
10        v[pos]++;
11
12        while (pos > 0 and v[pos] > n) {
13            --pos;
14            v[pos]++;
15        }
16
17        if (pos == 0 and v[pos] > n) break;
18
19        for (int i = pos + 1; i < k; ++i) v[i]
20        ] = v[pos];

```

```

19
20        pos = k - 1;
21    }
22 }

```

8 strings

8.1 Rabin-karp

```

1 vi rabin_karp(string const &s, string const &t) {
2     ll p = 31;
3     ll m = 1e9 + 9;
4     int S = s.size(), T = t.size();
5
6     vll p_pow(max(S, T));
7     p_pow[0] = 1;
8     for (int i = 1; i < (int)p_pow.size(); i++)
9         p_pow[i] = (p_pow[i - 1] * p) % m;
10
11     vll h(T + 1, 0);
12     for (int i = 0; i < T; i++)
13         h[i + 1] = (h[i] + (t[i] - 'a' + 1) *
14         p_pow[i]) % m;
15     ll h_s = 0;
16     for (int i = 0; i < S; i++)
17         h_s = (h_s + (s[i] - 'a' + 1) * p_pow
18         [i]) % m;
19
20     vi occurrences;
21     for (int i = 0; i + S - 1 < T; i++) {
22         ll cur_h = (h[i + S] + m - h[i]) % m;
23         // IT DON'T CONSIDERE CONLISIONS !
24         if (cur_h == h_s * p_pow[i] % m)
25             occurrences.push_back(i);
26     }
27     return occurrences;
28 }

```

8.2 Trie-naive

```

1 // time: O(n^2) memory: O(n^2)
2 using Node = map<char, int>;
3 using vi = vector<int>;
4 using Trie = vector<Node>;
5
6 Trie build(const string &s) {
7     int n = (int)s.size();
8     Trie trie(1);
9     string suffix;
10
11     for (int i = n - 1; i >= 0; --i) {
12         suffix = s.substr(i) + '#';
13
14         int v = 0; // root
15         for (auto c : suffix) {
16             if (c == '#') { // makrs the
17                 poistion of an occurrence
18                 trie[v][c] = i;
19                 break;
20             }
21             if (trie[v][c])
22                 v = trie[v][c];
23             else {
24                 trie.push_back({});
25                 trie[v][c] = trie.size() - 1;
26                 v = trie.size() - 1;
27             }
28         }
29         return trie;
30     }
31
32     vi search(Trie &trie, string s) {
33         int p = 0;
34         vi occ;

```

```

35     for (auto &c : s) {
36         p = trie[p][c];
37         if (!p) return occ;
38     }
39
40     queue<int> q;
41     q.push(0);
42     while (!q.empty()) {
43         auto cur = q.front();
44         q.pop();
45         for (auto [c, v] : trie[cur]) {
46             if (c == '#')
47                 occ.push_back(v);
48             else
49                 q.push(v);
50         }
51     }
52     return occ;
53 }
54
55 ll distinct_substr(const Trie &trie) {
56     ll cnt = 0;
57     queue<int> q;
58     q.push(0);
59     while (!q.empty()) {
60         auto u = q.front();
61         q.pop();
62
63         for (auto [c, v] : trie[u]) {
64             if (c != '#') {

```

```

65         cnt++;
66         q.push(v);
67     }
68 }
69 }
70 return cnt;
71 }

```

8.3 String-psum

```

1 struct strPsum {
2     ll n;
3     ll k;
4     vector<vll> psum;
5     strPsum(const string &s) : n(s.size()), k
6         (100), psum(k, vll(n + 1)) {
7         for (ll i = 1; i <= n; ++i) {
8             for (ll j = 0; j < k; ++j) {
9                 psum[j][i] = psum[j][
10                    i - 1];
11             }
12             psum[s[i - 1]][i]++;
13         }
14     }
15     ll qtd(ll l, ll r, char c) { // [0,n-1]
16         return psum[c][r + 1] - psum[c][l];
17     }

```