



Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

1 Misc	2	7 Grafos	7
1.1 Split	2	7.1 Floyd Warshall	7
1.2 Int128	2	7.2 Tree Diameter	8
2 Template	2	7.3 Cycle Path Recovery	8
2.1 Template	2	7.4 Bfs	8
2.2 Template Clean	2	7.5 Bipartite	8
3 Strings	2	7.6 Dfs	9
3.1 Kmp	2	7.7 Find Cycle	9
3.2 Generate All Permutations	2	7.8 Dinic	9
3.3 Generate All Sequences Length K	2	7.9 Bellman Ford	10
3.4 Z-function	3	7.10 Dijkstra	10
4 ED	3	7.11 Kruskall	10
4.1 Seg Lazy	3	7.12 Lca	11
4.2 Seg Tree	4	8 Math	11
4.3 Dsu	4	8.1 Log	11
4.4 Seg Pqru	5	8.2 Multiplicative Inverse	11
5 DP	5	8.3 Divisors	11
5.1 Knapsack	5	8.4 Prime Factors	12
5.2 Minimum Coin Change	5	8.5 Binary To Decimal	12
5.3 Digits	5	8.6 Mmc	12
5.4 Coins	6	8.7 Combinatory	12
5.5 Substr Palindromo	6	8.8 Sieve Of Eratosthenes	12
5.6 Kadane	6	8.9 Check If Bit Is On	12
6 Algoritmos	6	8.10 Crt	12
6.1 Lis	6	8.11 Matrix Exponentiation	12
6.2 Binary Exponentiation	6	8.12 Formulas	13
6.3 Ternary Search	7	8.13 Mdc	13
6.4 Binary Search First True	7	8.14 Fast Exponentiation	14
6.5 Edit Distance	7		
6.6 Ceil	7		
6.7 Delta-encoding	7		
6.8 Binary Search Last True	7		

1 Misc

1.1 Split

```
1 vector<string> split(string txt, char key = ' '){
2     vector<string> ans;
3
4     string palTemp = "";
5     for(int i = 0; i < txt.size(); i++){
6
7         if(txt[i] == key){
8             if(palTemp.size() > 0){
9                 ans.push_back(palTemp);
10                palTemp = "";
11            }
12        } else{
13            palTemp += txt[i];
14        }
15    }
16
17    if(palTemp.size() > 0)
18        ans.push_back(palTemp);
19
20    return ans;
21 }
22 }
```

1.2 Int128

```
1 __int128 read() {
2     __int128 x = 0, f = 1;
3     char ch = getchar();
4     while (ch < '0' || ch > '9') {
5         if (ch == '-') f = -1;
6         ch = getchar();
7     }
8     while (ch >= '0' && ch <= '9') {
9         x = x * 10 + ch - '0';
10        ch = getchar();
11    }
12    return x * f;
13 }
14 void print(__int128 x) {
15     if (x < 0) {
16         putchar('-');
17         x = -x;
18     }
19     if (x > 9) print(x / 10);
20     putchar(x % 10 + '0');
21 }
```

2 Template

2.1 Template

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define int long long
5 #define optimize std::ios::sync_with_stdio(false);
6     cin.tie(NULL);
7 #define vi vector<int>
8 #define ll long long
9 #define pb push_back
10 #define mp make_pair
11 #define ff first
12 #define ss second
13 #define pii pair<int, int>
14 #define MOD 1000000007
15 #define sqr(x) ((x) * (x))
```

```
15 #define all(x) (x).begin(), (x).end()
16 #define FOR(i, j, n) for (int i = j; i < n; i++)
17 #define qle(i, n) (i == n ? "\n" : " ")
18 #define endl "\n"
19 const int oo = 1e9;
20 const int MAX = 1e6;
21
22 int32_t main(){ optimize;
23
24     return 0;
25 }
```

2.2 Template Clean

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 // g++ teste.cpp -o teste -std=c++17
5 // ./teste < teste.txt
6
7 // cout << fixed << setprecision(12) << value << endl
8 ;
9 // freopen("input.txt", "r", stdin);
10 // freopen("output.txt", "w", stdout);
11
12 int main() {
13     ios::sync_with_stdio(false);
14     cin.tie(NULL);
15
16
17
18     return 0;
19 }
```

3 Strings

3.1 Kmp

```
1 vector<int> prefix_function(string s) {
2     int n = (int)s.length();
3     vector<int> pi(n);
4     for (int i = 1; i < n; i++) {
5         int j = pi[i-1];
6         while (j > 0 && s[i] != s[j])
7             j = pi[j-1];
8         if (s[i] == s[j])
9             j++;
10        pi[i] = j;
11    }
12    return pi;
13 }
```

3.2 Generate All Permutations

```
1 vector<string> generate_permutations(string s) {
2     int n = s.size();
3     vector<string> ans;
4
5     sort(s.begin(), s.end());
6
7     do {
8         ans.push_back(s);
9     } while (next_permutation(s.begin(), s.end()));
10
11    return ans;
12 }
```

3.3 Generate All Sequences Length K

```

1 // gera todas as possíveis sequências usando as letras
  em set (de comprimento n) e que tenham tamanho k
2 // sequence = ""
3 vector<string> generate_sequences(char set[], string
  sequence, int n, int k) {
4     if (k == 0) {
5         return { sequence };
6     }
7
8     vector<string> ans;
9     for (int i = 0; i < n; i++) {
10        auto aux = generate_sequences(set, sequence +
11        set[i], n, k - 1);
12        ans.insert(ans.end(), aux.begin(), aux.end())
13        ;
14        // for (auto e : aux) ans.push_back(e);
15    }
16    return ans;

```

3.4 Z-function

```

1 vector<int> z_function(string s) {
2     int n = (int) s.length();
3     vector<int> z(n);
4     for (int i = 1, l = 0, r = 0; i < n; ++i) {
5         if (i <= r)
6             z[i] = min(r - i + 1, z[i - l]);
7         while (i + z[i] < n && s[z[i]] == s[i + z[i]
8         ])
9             ++z[i];
10        if (i + z[i] - 1 > r)
11            l = i, r = i + z[i] - 1;
12    }
13    return z;

```

4 ED

4.1 Seg Lazy

```

1 using ll = long long;
2
3 struct segTree {
4     int size;
5     vector<ll> tree, lazy;
6
7     ll modify_op(ll a, ll b, ll len) {
8         if (b == -1) return a;
9         return b * len;
10    }
11
12    void apply_mod_op(ll &a, ll b, ll len) {
13        a = modify_op(a, b, len);
14    }
15
16    ll merge(ll a, ll b) {
17        return a + b;
18    }
19
20    void init(int n) {
21        size = 1;
22        while (size < n) size *= 2;
23        tree.assign(2 * size, 0LL);
24        lazy.assign(2 * size, -1);
25    }
26
27    void propagate(int x, int lx, int rx) {
28        if (rx - lx == 1) return;
29

```

```

        int m = (lx + rx) / 2;
        apply_mod_op(lazy[2 * x + 1], lazy[x], 1);
        apply_mod_op(tree[2 * x + 1], lazy[x], m - lx
    );

        apply_mod_op(lazy[2 * x + 2], lazy[x], 1);
        apply_mod_op(tree[2 * x + 2], lazy[x], rx - m
    );

        lazy[x] = -1;
    }

    void build(vector<int> &arr, int x, int lx, int
    rx) {
        if (rx - lx == 1) {
            if (lx < (int)arr.size())
                tree[x] = arr[lx];

            return;
        }

        int m = (lx + rx) / 2;
        build(arr, 2 * x + 1, lx, m);
        build(arr, 2 * x + 2, m, rx);

        tree[x] = merge(tree[2 * x + 1], tree[2 * x +
        2]);
    }

    void build(vector<int> &arr) {
        build(arr, 0, 0, size);
    }

    void update(int l, int r, int v, int x, int lx,
    int rx) {
        propagate(x, lx, rx);

        if (lx >= r || l >= rx) return;
        if (lx >= l && rx <= r) {
            apply_mod_op(lazy[x], v, 1);
            apply_mod_op(tree[x], v, rx - lx);
            return;
        }

        int m = (lx + rx) / 2;
        update(l, r, v, 2 * x + 1, lx, m);
        update(l, r, v, 2 * x + 2, m, rx);

        tree[x] = merge(tree[2 * x + 1], tree[2 * x +
        2]);
    }

    void update(int l, int r, int v) {
        update(l, r, v, 0, 0, size);
    }

    ll query(int l, int r, int x, int lx, int rx) {
        propagate(x, lx, rx);

        if (lx >= r || l >= rx) return 0;
        if (lx >= l && rx <= r) return tree[x];

        int m = (lx + rx) / 2;
        ll s1 = query(l, r, 2 * x + 1, lx, m);
        ll s2 = query(l, r, 2 * x + 2, m, rx);

        return merge(s1, s2);
    }

    ll query(int l, int r) {
        return query(l, r, 0, 0, size);
    }

```

```

97 void debug() {
98     for (auto e : tree)
99         cout << e << ' ';
100     cout << endl;
101
102     for (auto e : lazy)
103         cout << e << ' ';
104     cout << endl;
105 }
106 };

```

4.2 Seg Tree

```

1 class SegTree{
2     vector<int> seg;
3     vector<int> v;
4     int size;
5     int el_neutro = INT_MAX;
6
7     int f(int a, int b){
8         return min(a,b);
9     }
10
11 void update(int pos, int ini, int fim, int i, int
12 val){
13     if(i < ini or i > fim) return;
14     if(ini == fim){
15         seg[pos] = val; return;
16     }
17
18     int m = (ini+fim)/2;
19     int e = 2*pos, d = 2*pos+1;
20     update(e, ini, m, i, val);
21     update(d, m+1, fim, i, val);
22
23     seg[pos] = f(seg[e], seg[d]);
24 }
25
26 int query(int pos, int ini, int fim, int p, int q
27 ){
28     if(q < ini or p > fim) return el_neutro;
29     if(p <= ini and fim <= q) return seg[pos];
30
31     int m = (ini + fim)/2;
32     int e = 2*pos, d = 2*pos+1;
33     return f(query(e,ini,m,p,q), query(d,m+1,fim,
34 p,q));
35 }
36
37 void build(int pos, int ini, int fim){
38     if(ini == fim){
39         seg[pos] = v[ini]; return;
40     }
41
42     int m = (ini+fim)/2;
43     int e = 2*pos, d=2*pos+1;
44
45     build(e,ini,m);
46     build(d,m+1,fim);
47
48     seg[pos] = f(seg[e], seg[d]);
49 }
50
51 public:
52     SegTree(int n, vector<int> source): seg(4*
53 size), v(size){
54     size = n;
55     for(int i=0; i<size; i++) v[i] = source[i]
56 ];
57 }
58
59 void update(int i, int val){ return update
60 (1,1,size,i,val); }

```

```

55
56     int query(int p, int q){ return query(1,1,
57 size,p,q); }
58
59     void build(){ return build(1,1,size); }
60 };

```

4.3 Dsu

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAX = 1e6+17;
6
7 struct DSU {
8     int n;
9     vector<int> link, sizes;
10
11     DSU(int n) {
12         this->n = n;
13         link.assign(n+1, 0);
14         sizes.assign(n+1, 1);
15
16         for (int i = 0; i <= n; i++)
17             link[i] = i;
18     }
19
20     int find(int x) {
21         while (x != link[x])
22             x = link[x];
23
24         return x;
25     }
26
27     bool same(int a, int b) {
28         return find(a) == find(b);
29     }
30
31     void unite(int a, int b) {
32         a = find(a);
33         b = find(b);
34
35         if (a == b) return;
36
37         if (sizes[a] < sizes[b])
38             swap(a, b);
39
40         sizes[a] += sizes[b];
41         link[b] = a;
42     }
43
44     int size(int x) {
45         return sizes[x];
46     }
47 };
48
49 int main() {
50     ios::sync_with_stdio(false);
51     cin.tie(NULL);
52
53     int cities, roads; cin >> cities >> roads;
54     vector<int> final_roads;
55     int ans = 0;
56     DSU dsu = DSU(cities);
57     for (int i = 0, a, b; i < roads; i++) {
58         cin >> a >> b;
59         dsu.unite(a, b);
60     }
61
62     for (int i = 2; i <= cities; i++) {
63         if (!dsu.same(1, i)) {
64             ans++;

```

```

65         final_roads.push_back(i);
66         dsu.unite(1,i);
67     }
68 }
69
70 cout << ans << '\n';
71 for (auto e : final_roads) {
72     cout << "1 " << e << '\n';
73 }
74
75 }

```

4.4 Seg Pqru

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  class SegTree{
5      vector<int> seg;
6      vector<int> v;
7      int size;
8      int el_neutro = INT_MAX;
9
10     int f(int a, int b){
11         return min(a,b);
12     }
13
14     void update_range(int pos, int ini, int fim, int
15     l, int r, int val){
16         if(r < ini or l > fim) return;
17         if(l <= ini and fim <= r){
18             seg[pos] += val;
19         }
20
21         int mid = (ini+fim)/2;
22
23         update_range(2*pos, ini, mid, l, r, val);
24         update_range(2*pos+1, mid+1, fim, l, r, val);
25     }
26
27     int query_point(int pos, int ini, int fim, int i)
28     {
29         if(ini == fim) return seg[pos];
30
31         int mid = (ini + fim)/2;
32         if(i <= mid)
33             return query_point(2*pos, ini, mid, i);
34         else
35             return query_point(2*pos+1, mid+1, fim, i
36         );
37     }
38
39     void build(int pos, int ini, int fim){
40         if(ini == fim){
41             seg[pos] = v[ini]; return;
42         }
43
44         int m = (ini+fim)/2;
45         int e = 2*pos, d=2*pos+1;
46
47         build(e,ini,m);
48         build(d,m+1,fim);
49
50         seg[pos] = f(seg[e], seg[d]);
51     }
52
53     public:
54     SegTree(int n, vector<int> source): seg(4*size),
55     v(size){
56         size = n;
57         for(int i=0; i<size; i++) v[i] = source[i];
58     }
59 }

```

```

56     void update(int l, int r, int val){ return
57     update_range(1,1,size,l, r,val); }
58
59     int query(int i){ return query_point(1,1,size,i);
60     }
61
62     void build(){ return build(1,1,size); }
63 };

```

5 DP

5.1 Knapsack

```

1  int val[MAXN], peso[MAXN], dp[MAXN][MAXS];
2
3  int knapsack(int n, int m){ // n Objetos | Peso max
4      for(int i=0; i<=n; i++){
5          for(int j=0; j<=m; j++){
6              if(i==0 or j==0)
7                  dp[i][j] = 0;
8              else if(peso[i-1]<=j)
9                  dp[i][j] = max(val[i-1]+dp[i-1][j-
10             peso[i-1]], dp[i-1][j]);
11              else
12                  dp[i][j] = dp[i-1][j];
13          }
14      }
15      return dp[n][m];
16 }

```

5.2 Minimum Coin Change

```

1  int n;
2  vector<int> valores;
3
4  int tabela[1005];
5
6  int dp(int k){
7      if(k == 0){
8          return 0;
9      }
10     if(tabela[k] != -1)
11         return tabela[k];
12     int melhor = 1e9;
13     for(int i = 0; i < n; i++){
14         if(valores[i] <= k)
15             melhor = min(melhor, 1 + dp(k - valores[i
16         ]));
17     }
18     return tabela[k] = melhor;
19 }

```

5.3 Digits

```

1  // achar a quantidade de numeros menores que R que
2  // possuem no maximo 3 digitos nao nulos
3  // a ideia eh utilizar da ordem lexicografica para
4  // verificar isso pois se temos por exemplo
5  // o numero 8500, a gente sabe que se pegarmos o
6  // numero 7... qualquer digito depois do 7
7  // sera necessariamente menor q 8500
8
9  string r;
10 int tab[20][2][5];
11
12 // i - digito de R
13 // menor - ja pegou um numero menor que um digito de
14 // R
15 // qt - quantidade de digitos nao nulos
16 int dp(int i, bool menor, int qt){
17     if(qt > 3) return 0;
18     if(i >= r.size()) return 1;
19 }

```

```

15     if(tab[i][menor][qt] != -1) return tab[i][menor][qt];
16
17     int dr = r[i]-'0';
18     int res = 0;
19
20     for(int d = 0; d <= 9; d++) {
21         int dnn = qt + (d > 0);
22         if(menor == true) {
23             res += dp(i+1, true, dnn);
24         }
25         else if(d < dr) {
26             res += dp(i+1, true, dnn);
27         }
28         else if(d == dr) {
29             res += dp(i+1, false, dnn);
30         }
31     }
32
33     return tab[i][menor][qt] = res;
34 }

```

5.4 Coins

```

1 int tb[1005];
2 int n;
3 vector<int> moedas;
4
5 int dp(int i){
6     if(i >= n)
7         return 0;
8     if(tb[i] != -1)
9         return tb[i];
10
11     tb[i] = max(dp(i+1), dp(i+2) + moedas[i]);
12     return tb[i];
13 }
14
15 int main(){
16     memset(tb, -1, sizeof(tb));
17 }

```

5.5 Substr Palindromo

```

1 // êvoc deve informar se a substring de S formada
   pelos elementos entre os indices i e j
2 // é um palindromo ou ãno.
3
4 char s[MAX];
5 int calculado[MAX][MAX]; // inciado com false, ou 0
6 int tabela[MAX][MAX];
7
8 int is_palin(int i, int j){
9     if(calculado[i][j]){
10         return tabela[i][j];
11     }
12     if(i == j) return true;
13     if(i + 1 == j) return s[i] == s[j];
14
15     int ans = false;
16     if(s[i] == s[j]){
17         if(is_palin(i+1, j-1)){
18             ans = true;
19         }
20     }
21     calculado[i][j] = true;
22     tabela[i][j] = ans;
23     return ans;
24 }

```

5.6 Kadane

```

1 // achar uma subsequencia continua no array que a
   soma seja a maior possivel
2 // nesse caso vc precisa multiplicar exatamente 1
   elemento da subsequencia
3 // e achar a maior soma com isso
4
5 int n, x, arr[MAX], tab[MAX][2]; // tab[maior
   resposta no intervalo][foi multiplicado ou ãno]
6
7 int dp(int i, bool mult) {
8     if (i == n-1) {
9         if (!mult) return arr[n-1]*x;
10        return arr[n-1];
11    }
12    if (tab[i][mult] != -1) return tab[i][mult];
13
14    int res;
15
16    if (mult) {
17        res = max(arr[i], arr[i] + dp(i+1, 1));
18    }
19    else {
20        res = max({
21            arr[i]*x,
22            arr[i]*x + dp(i+1, 1),
23            arr[i] + dp(i+1, 0)
24        });
25    }
26
27    return tab[i][mult] = res;
28 }
29
30 int main() {
31
32     memset(tab, -1, sizeof(tab));
33
34     int ans = -oo;
35     for (int i = 0; i < n; i++) {
36         ans = max(ans, dp(i, 0));
37     }
38
39     return 0;
40 }

```

6 Algoritmos

6.1 Lis

```

1 int lis(vector<int> const& a) {
2     int n = a.size();
3     vector<int> d(n, 1);
4     for (int i = 0; i < n; i++) {
5         for (int j = 0; j < i; j++) {
6             if (a[j] < a[i])
7                 d[i] = max(d[i], d[j] + 1);
8         }
9     }
10
11     int ans = d[0];
12     for (int i = 1; i < n; i++) {
13         ans = max(ans, d[i]);
14     }
15     return ans;
16 }

```

6.2 Binary Exponentiation

```

1 long long power(long long a, long long b) {
2     long long res = 1;
3     while (b > 0) {
4         if (b & 1)

```

```

5         res = res * a;
6         a = a * a;
7         b >>= 1;
8     }
9     return res;
10 }

```

6.3 Ternary Search

```

1 double ternary_search(double l, double r) {
2     double eps = 1e-9;           //set the error
    limit here
3     while (r - l > eps) {
4         double m1 = l + (r - l) / 3;
5         double m2 = r - (r - l) / 3;
6         double f1 = f(m1);       //evaluates the
    function at m1
7         double f2 = f(m2);       //evaluates the
    function at m2
8         if (f1 < f2)
9             l = m1;
10        else
11            r = m2;
12    }
13    return f(l);                 //return the
    maximum of f(x) in [l, r]
14 }

```

6.4 Binary Search First True

```

1 int first_true(int lo, int hi, function<bool(int)> f)
    {
2     hi++;
3     while (lo < hi) {
4         int mid = lo + (hi - lo) / 2;
5         if (f(mid)) {
6             hi = mid;
7         } else {
8             lo = mid + 1;
9         }
10    }
11    return lo;
12 }

```

6.5 Edit Distance

```

1 int editDist(string str1, string str2, int m, int n)
    {
2     if (m == 0) return n;
3     if (n == 0) return m;
4
5     if (str1[m - 1] == str2[n - 1]) return editDist(
        str1, str2, m - 1, n - 1);
6     return 1 + min({editDist(str1, str2, m, n - 1),
        editDist(str1, str2, m - 1, n), editDist(str1,
        str2, m - 1, n - 1)});
7 }

```

6.6 Ceil

```

1 long long division_ceil(long long a, long long b) {
2     return 1 + ((a - 1) / b); // if a != 0
3 }

```

6.7 Delta-encoding

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     int n, q;

```

```

6     cin >> n >> q;
7     int [n];
8     int delta[n+2];
9
10    while(q--){
11        int l, r, x;
12        cin >> l >> r >> x;
13        delta[l] += x;
14        delta[r+1] -= x;
15    }
16
17    int curr = 0;
18    for(int i=0; i < n; i++){
19        curr += delta[i];
20        v[i] = curr;
21    }
22
23    for(int i=0; i < n; i++){
24        cout << v[i] << ' ';
25    }
26    cout << '\n';
27
28    return 0;
29 }

```

6.8 Binary Search Last True

```

1 int last_true(int lo, int hi, function<bool(int)> f)
    {
2     lo--;
3     while (lo < hi) {
4         int mid = lo + (hi - lo + 1) / 2;
5         if (f(mid)) {
6             lo = mid;
7         } else {
8             hi = mid - 1;
9         }
10    }
11    return lo;
12 }

```

6.9 Kadane

```

1 int ans = a[0], ans_l = 0, ans_r = 0;
2 int sum = 0, minus_pos = -1;
3
4 for (int r = 0; r < n; ++r) {
5     sum += a[r];
6     if (sum > ans) {
7         ans = sum;
8         ans_l = minus_pos + 1;
9         ans_r = r;
10    }
11    if (sum < 0) {
12        sum = 0;
13        minus_pos = r;
14    }
15 }

```

7 Grafos

7.1 Floyd Warshall

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 using ll = long long;
5
6 const int MAX = 507;
7 const long long INF = 0x3f3f3f3f3f3f3f3f;
8

```

```

9 ll dist[MAX][MAX];
10 int n;
11
12 void floyd_warshall() {
13     for (int i = 0; i < n; i++) {
14         for (int j = 0; j < n; j++) {
15             if (i == j) dist[i][j] = 0;
16             else if (!dist[i][j]) dist[i][j] = INF;
17         }
18     }
19
20     for (int k = 0; k < n; k++) {
21         for (int i = 0; i < n; i++) {
22             for (int j = 0; j < n; j++) {
23                 // trata o caso no qual o grafo tem
24                 arestas com peso negativo
25                 if (dist[i][k] < INF && dist[k][j] <
26                     INF){
27                     dist[i][j] = min(dist[i][j], dist
28                     [i][k] + dist[k][j]);
29                 }
30             }
31         }
32     }
33 }

```

7.2 Tree Diameter

```

1 #include<bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAX = 3e5+17;
6
7 vector<int> adj[MAX];
8 bool visited[MAX];
9
10 int max_depth = 0, max_node = 1;
11
12 void dfs (int v, int depth) {
13     visited[v] = true;
14
15     if (depth > max_depth) {
16         max_depth = depth;
17         max_node = v;
18     }
19
20     for (auto u : adj[v]) {
21         if (!visited[u]) dfs(u, depth + 1);
22     }
23 }
24
25 int tree_diameter() {
26     dfs(1, 0);
27     max_depth = 0;
28     for (int i = 0; i < MAX; i++) visited[i] = false;
29     dfs(max_node, 0);
30     return max_depth;
31 }

```

7.3 Cycle Path Recovery

```

1 int n;
2 vector<vector<int>> adj;
3 vector<char> color;
4 vector<int> parent;
5 int cycle_start, cycle_end;
6
7 bool dfs(int v) {
8     color[v] = 1;
9     for (int u : adj[v]) {
10         if (color[u] == 0) {

```

```

11             parent[u] = v;
12             if (dfs(u))
13                 return true;
14             } else if (color[u] == 1) {
15                 cycle_end = v;
16                 cycle_start = u;
17                 return true;
18             }
19         }
20         color[v] = 2;
21         return false;
22     }
23
24 void find_cycle() {
25     color.assign(n, 0);
26     parent.assign(n, -1);
27     cycle_start = -1;
28
29     for (int v = 0; v < n; v++) {
30         if (color[v] == 0 && dfs(v))
31             break;
32     }
33
34     if (cycle_start == -1) {
35         cout << "Acyclic" << endl;
36     } else {
37         vector<int> cycle;
38         cycle.push_back(cycle_start);
39         for (int v = cycle_end; v != cycle_start; v =
40             parent[v])
41             cycle.push_back(v);
42         cycle.push_back(cycle_start);
43         reverse(cycle.begin(), cycle.end());
44
45         cout << "Cycle found: ";
46         for (int v : cycle)
47             cout << v << " ";
48         cout << endl;
49     }
50 }

```

7.4 Bfs

```

1 void bfs(int start){
2
3     queue<int> q;
4     q.push(start);
5
6     vector<bool> visited(Graph_Max_Size, false);
7     visited[start] = true;
8     while(q.size()){
9         int u = q.front();
10        q.pop();
11        for(int w: graph[u]){
12            if(not visited[w]){
13                q.push(w);
14                visited[w] = true;
15            }
16        }
17    }
18 }
19 }

```

7.5 Bipartite

```

1 const int NONE = 0, BLUE = 1, RED = 2;
2 vector<vector<int>> graph(100005);
3 vector<bool> visited(100005);
4 int color[100005];
5
6 bool bfs(int s = 1){
7

```



```

8     queue<int> q;
9     q.push(s);
10    color[s] = BLUE;
11
12    while (not q.empty()){
13        auto u = q.front(); q.pop();
14
15        for (auto v : graph[u]){
16            if (color[v] == NONE){
17                color[v] = 3 - color[u];
18                q.push(v);
19            }
20            else if (color[v] == color[u]){
21                return false;
22            }
23        }
24    }
25
26    return true;
27 }
28
29 bool is_bipartite(int n){
30
31     for (int i = 1; i<=n; i++)
32         if (color[i] == NONE and not bfs(i))
33             return false;
34
35     return true;
36 }

```

7.6 Dfs

```

1 vector<vector<int>>> graph;
2 vector<bool> visited;
3
4 void dfs(int vertex){
5     visited[vertex] = true;
6
7     for(int w: graph[vertex]){
8         if(!visited[w]){
9             dfs(w);
10        }
11    }
12 }

```

7.7 Find Cycle

```

1 bitset<MAX> visited;
2 vector<int> path;
3 vector<int> adj[MAX];
4
5 bool dfs(int u, int p){
6
7     if (visited[u]) return false;
8
9     path.pb(u);
10    visited[u] = true;
11
12    for (auto v : adj[u]){
13        if (visited[v] and u != v and p != v){
14            path.pb(v); return true;
15        }
16
17        if (dfs(v, u)) return true;
18    }
19
20    path.pop_back();
21    return false;
22 }
23
24 bool has_cycle(int N){
25

```

```

26     visited.reset();
27
28     for (int u = 1; u <= N; ++u){
29         path.clear();
30         if (not visited[u] and dfs(u,-1))
31             return true;
32     }
33
34     return false;
35 }
36 }

```

7.8 Dinic

```

1 const int N = 300;
2
3 struct Dinic {
4     struct Edge{
5         int from, to; ll flow, cap;
6     };
7     vector<Edge> edge;
8
9     vector<int> g[N];
10    int ne = 0;
11    int lvl[N], vis[N], pass;
12    int qu[N], px[N], qt;
13
14    ll run(int s, int sink, ll minE) {
15        if(s == sink) return minE;
16
17        ll ans = 0;
18
19        for(; px[s] < (int)g[s].size(); px[s]++) {
20            int e = g[s][ px[s] ];
21            auto &v = edge[e], &rev = edge[e^1];
22            if(lvl[v.to] != lvl[s]+1 || v.flow >= v.
23                cap) continue; // v.cap - v.flow
24            < lim
25            ll tmp = run(v.to, sink,min(minE, v.cap-v
26                .flow));
27            v.flow += tmp, rev.flow -= tmp;
28            ans += tmp, minE -= tmp;
29            if(minE == 0) break;
30        }
31        return ans;
32    }
33
34    bool bfs(int source, int sink) {
35        qt = 0;
36        qu[qt++] = source;
37        lvl[source] = 1;
38        vis[source] = ++pass;
39        for(int i = 0; i < qt; i++) {
40            int u = qu[i];
41            px[u] = 0;
42            if(u == sink) return true;
43            for(auto& ed : g[u]) {
44                auto v = edge[ed];
45                if(v.flow >= v.cap || vis[v.to] ==
46                    pass) continue; // v.cap - v.flow < lim
47                vis[v.to] = pass;
48                lvl[v.to] = lvl[u]+1;
49                qu[qt++] = v.to;
50            }
51        }
52        return false;
53    }
54
55    ll flow(int source, int sink) {
56        reset_flow();
57        ll ans = 0;
58        //for(lim = (1LL << 62); lim >= 1; lim /= 2)
59        while(bfs(source, sink))

```

```

56         ans += run(source, sink, LLINF);
57         return ans;
58     }
59     void addEdge(int u, int v, ll c, ll rc) {
60         Edge e = {u, v, 0, c};
61         edge.pb(e);
62         g[u].push_back(ne++);
63
64         e = {v, u, 0, rc};
65         edge.pb(e);
66         g[v].push_back(ne++);
67     }
68     void reset_flow() {
69         for(int i = 0; i < ne; i++)
70             edge[i].flow = 0;
71         memset(lvl, 0, sizeof(lvl));
72         memset(vis, 0, sizeof(vis));
73         memset(qu, 0, sizeof(qu));
74         memset(px, 0, sizeof(px));
75         qt = 0; pass = 0;
76     }
77 };

```

7.9 Bellman Ford

```

1 struct edge
2 {
3     int a, b, cost;
4 };
5
6 int n, m, v;
7 vector<edge> e;
8 const int INF = 1000000000;
9
10 void solve()
11 {
12     vector<int> d(n, INF);
13     d[v] = 0;
14     for (int i=0; i<n-1; ++i)
15         for (int j=0; j<m; ++j)
16             if (d[e[j].a] < INF)
17                 d[e[j].b] = min (d[e[j].b], d[e[j].a]
18 + e[j].cost);
19 }

```

7.10 Dijkstra

```

1 const int MAX = 2e5+7;
2 const int INF = 1000000000;
3 vector<vector<pair<int, int>>> adj(MAX);
4
5 void dijkstra(int s, vector<int> & d, vector<int> & p
6 ) {
7     int n = adj.size();
8     d.assign(n, INF);
9     p.assign(n, -1);
10
11     d[s] = 0;
12     set<pair<int, int>> q;
13     q.insert({0, s});
14     while (!q.empty()) {
15         int v = q.begin()->second;
16         q.erase(q.begin());
17
18         for (auto edge : adj[v]) {
19             int to = edge.first;
20             int len = edge.second;
21
22             if (d[v] + len < d[to]) {
23                 q.erase({d[to], to});
24                 d[to] = d[v] + len;
25                 p[to] = v;

```

```

26                 q.insert({d[to], to});
27             }
28         }
29     }
30
31     vector<int> restore_path(int s, int t) {
32         vector<int> path;
33
34         for (int v = t; v != s; v = p[v])
35             path.push_back(v);
36         path.push_back(s);
37
38         reverse(path.begin(), path.end());
39         return path;
40     }
41
42     int adj[MAX][MAX];
43     int dist[MAX];
44     int minDistance(int dist[], bool sptSet[], int V) {
45         int min = INT_MAX, min_index;
46
47         for (int v = 0; v < V; v++)
48             if (sptSet[v] == false && dist[v] <= min)
49                 min = dist[v], min_index = v;
50
51         return min_index;
52     }
53
54     void dijkstra(int src, int V) {
55         bool sptSet[V];
56         for (int i = 0; i < V; i++)
57             dist[i] = INT_MAX, sptSet[i] = false;
58
59         dist[src] = 0;
60
61         for (int count = 0; count < V - 1; count++) {
62             int u = minDistance(dist, sptSet, V);
63
64             sptSet[u] = true;
65
66             for (int v = 0; v < V; v++)
67                 if (!sptSet[v] && adj[u][v]
68                     && dist[u] != INT_MAX
69                     && dist[u] + adj[u][v] < dist[v])
70                     dist[v] = dist[u] + adj[u][v];
71         }
72     }
73 }
74 }

```

7.11 Kruskall

```

1 vector<int> parent, rank;
2
3 void make_set(int v) {
4     parent[v] = v;
5     rank[v] = 0;
6 }
7
8 int find_set(int v) {
9     if (v == parent[v])
10         return v;
11     return parent[v] = find_set(parent[v]);
12 }
13
14 void union_sets(int a, int b) {
15     a = find_set(a);
16     b = find_set(b);
17     if (a != b) {
18         if (rank[a] < rank[b])
19             swap(a, b);
20         parent[b] = a;

```

```

21         if (rank[a] == rank[b])
22             rank[a]++;
23     }
24 }
25
26 struct Edge {
27     int u, v, weight;
28     bool operator<(Edge const& other) {
29         return weight < other.weight;
30     }
31 };
32
33 int n;
34 vector<Edge> edges;
35
36 int cost = 0;
37 vector<Edge> result;
38 parent.resize(n);
39 rank.resize(n);
40 for (int i = 0; i < n; i++)
41     make_set(i);
42
43 sort(edges.begin(), edges.end());
44
45 for (Edge e : edges) {
46     if (find_set(e.u) != find_set(e.v)) {
47         cost += e.weight;
48         result.push_back(e);
49         union_sets(e.u, e.v);
50     }
51 }

```

7.12 Lca

```

1  const int MAX = 2e5+17;
2
3  int n, l;
4  vector<vector<int>> adj;
5  // vector<pair<int, int>> adj[MAX];
6  // int dist[MAX];
7
8  int timer;
9  vector<int> tin, tout;
10 vector<vector<int>> up;
11
12 void dfs(int v, int p)
13 {
14     tin[v] = ++timer;
15     up[v][0] = p;
16     for (int i = 1; i <= l; ++i)
17         up[v][i] = up[up[v][i-1]][i-1];
18
19     for (int u : adj[v]) {
20         if (u != p)
21             dfs(u, v);
22     }
23
24     /*for (auto [u, peso] : adj[v]) {
25         if (u != p) {
26             dist[u] = dist[v] + peso;
27             dfs(u, v);
28         }
29     }*/
30
31     tout[v] = ++timer;
32 }
33
34 bool is_ancestor(int u, int v)
35 {
36     return tin[u] <= tin[v] && tout[u] >= tout[v];
37 }
38
39 int lca(int u, int v)

```

```

40 {
41     if (is_ancestor(u, v))
42         return u;
43     if (is_ancestor(v, u))
44         return v;
45     for (int i = l; i >= 0; --i) {
46         if (!is_ancestor(up[u][i], v))
47             u = up[u][i];
48     }
49     return up[u][0];
50 }
51
52 void preprocess(int root) {
53     tin.resize(MAX);
54     tout.resize(MAX);
55     timer = 0;
56     up.assign(MAX, vector<int>(32));
57     dfs(root, root);
58 }
59
60 //distance between a and b
61 // dist[a] + dist[b] - 2*dist[lca(a, b)]

```

8 Math

8.1 Log

```

1 int intlog(double base, double x) {
2     return (int)(log(x) / log(base));
3 }

```

8.2 Multiplicative Inverse

```

1 ll extend_euclid(ll a, ll b, ll &x, ll &y) {
2     if (a == 0)
3     {
4         x = 0; y = 1;
5         return b;
6     }
7     ll x1, y1;
8     ll d = extend_euclid(b%a, a, x1, y1);
9     x = y1 - (b / a) * x1;
10    y = x1;
11    return d;
12 }
13
14 // gcd(a, m) = 1 para existir solucao
15 // ax + my = 1, ou a*x = 1 (mod m)
16 ll inv_gcd(ll a, ll m) { // com gcd
17     ll x, y;
18     extend_euclid(a, m, x, y);
19     return ((x % m) + m) % m;
20 }
21
22 ll inv(ll a, ll phim) { // com phi(m), se m for primo
23     entao phi(m) = p-1
24     ll e = phim-1;
25     return fexp(a, e, MOD);
26 }

```

8.3 Divisors

```

1 vector<long long> all_divisors(long long n) {
2     vector<long long> ans;
3     for(long long a = 1; a*a <= n; a++){
4         if(n % a == 0) {
5             long long b = n / a;
6             ans.push_back(a);
7             if(a != b) ans.push_back(b);
8         }
9     }

```

```

10 sort(ans.begin(), ans.end());
11 return ans;
12 }

```

8.4 Prime Factors

```

1 vector<pair<long long, int>> fatora(long long n) {
2     vector<pair<long long, int>> ans;
3     for(long long p = 2; p*p <= n; p++) {
4         if(n % p == 0) {
5             int expoente = 0;
6             while(n % p == 0) {
7                 n /= p;
8                 expoente++;
9             }
10            ans.emplace_back(p, expoente);
11        }
12    }
13    if(n > 1) ans.emplace_back(n, 1);
14    return ans;
15 }

```

8.5 Binary To Decimal

```

1 int binary_to_decimal(long long n) {
2     int dec = 0, i = 0, rem;
3
4     while (n!=0) {
5         rem = n % 10;
6         n /= 10;
7         dec += rem * pow(2, i);
8         ++i;
9     }
10
11    return dec;
12 }
13
14 long long decimal_to_binary(int n) {
15     long long bin = 0;
16     int rem, i = 1;
17
18     while (n!=0) {
19         rem = n % 2;
20         n /= 2;
21         bin += rem * i;
22         i *= 10;
23     }
24
25    return bin;
26 }

```

8.6 Mmc

```

1 long long lcm(long long a, long long b){
2     return (a/__gcd(a,b)*b);
3 }

```

8.7 Combinatory

```

1 int comb(int k){
2     if(k==1 or k==0) return 0;
3     return (k*(k-1))/2;
4 }

```

8.8 Sieve Of Eratosthenes

```

1 int n;
2 vector<bool> is_prime(n+1, true);
3 is_prime[0] = is_prime[1] = false;
4 for (int i = 2; i <= n; i++) {
5     if (is_prime[i] && (long long)i * i <= n) {

```

```

6         for (int j = i * i; j <= n; j += i)
7             is_prime[j] = false;
8     }
9 }

```

8.9 Check If Bit Is On

```

1 // msb de 0 é undefined
2 #define msb(n) (32 - __builtin_clz(n))
3 // #define msb(n) (64 - __builtin_clzll(n))
4
5 bool bit_on(int n, int bit) {
6     if(1 & (n >> bit)) return true;
7     else return false;
8 }

```

8.10 Crt

```

1 ll crt(const vector<pair<ll, ll>> &vet){
2     ll ans = 0, lcm = 1;
3     ll a, b, g, x, y;
4     for(const auto &p : vet) {
5         tie(a, b) = p;
6         tie(g, x, y) = gcd(lcm, b);
7         if((a - ans) % g != 0) return -1; // no
8         solution
9         ans = ans + x * ((a - ans) / g) % (b / g) *
10        lcm;
11        lcm = lcm * (b / g);
12        ans = (ans % lcm + lcm) % lcm;
13    }
14    return ans;
15 }

```

8.11 Matrix Exponentiation

```

1 #include <bits/stdc++.h>
2 #define debug(x) cout << "[" << #x << " = " << x << "
3 ] "
4 #define ff first
5 #define ss second
6
7 using namespace std;
8 using ll = long long;
9 using ld = long double;
10 using pii = pair<int,int>;
11 using vi = vector<int>;
12 using tii = tuple<int,int,int>;
13
14 const int oo = (int)1e9;
15 const ll OO = 0x3f3f3f3f3f3f3fLL;
16
17 const int MOD = 1e9+7;
18
19 struct Mat{
20     vector<vector<ll>> matriz;
21     int l, c;
22
23     Mat(vector<vector<ll>>& mat){
24         matriz = mat;
25         l = mat.size();
26         c = mat[0].size();
27     }
28
29     Mat(int r, int col, bool identidade=false){
30         l = r; c = col;
31         matriz.assign(l, vector<ll>(col, 0));
32         if(identidade){
33             for(int i = 0; i < min(l,col); i++){
34                 matriz[i][i] = 1;
35             }

```

```

36     }
37
38     Mat operator * (const Mat& a) const{
39         assert(c == a.l);
40         vector<vector<ll>> resp(1, vector<ll>(a.c, 0)
41     );
42
43         for(int i = 0; i < l; i++){
44             for(int j = 0; j < a.c; j++){
45                 for(int k = 0; k < a.l; k++){
46                     resp[i][j] = (resp[i][j] + (
47                         matriz[i][k]*a.matriz[k][j]) % MOD) % MOD;
48                 }
49             }
50         }
51         return Mat(resp);
52     }
53
54     Mat operator + (const Mat& a) const{
55         assert(l == a.l && c == a.c);
56         vector<vector<ll>> resp(1, vector<ll>(c,0));
57         for(int i = 0; i < l; i++){
58             for(int j = 0; j < c; j++){
59                 resp[i][j] = (resp[i][j] + matriz[i][
60                     j] + a.matriz[i][j]) % MOD;
61             }
62         }
63         return Mat(resp);
64     }
65
66     Mat fexp(Mat& base, ll expoente, ll sz){
67         Mat result = Mat(sz, sz, 1);
68         while(expoente > 0){
69             if(expoente & 1) result = result * base;
70             base = base * base;
71             expoente /= 2;
72         }
73         return result;
74     }
75
76     int main() {
77         ios::sync_with_stdio(false);
78         cin.tie(NULL);
79
80         ll n, a, b;
81         cin >> a >> b >> n;
82
83         Mat X(2,2);
84
85         X.matriz[0][1] = 1;
86         X.matriz[1][0] = -1;
87         X.matriz[1][1] = 1;
88
89         Mat y = fexp(X,n-1,2);
90
91         ll ans = y.matriz[0][0] * a + y.matriz[0][1] * b;
92
93         while(ans < 0)
94             ans += MOD;
95
96         cout << ans % MOD << endl;
97     }
98 }

```

8.12 Formulas

```

1 #define lcm(a,b) (a*b)/gcd(a,b)
2
3 int gcd(int a, int b) {
4     if (b == 0) return a;
5     return gcd(b, a % b);
6 }
7

```

```

8 // number of elements
9 long long sum_of_n_first_squares(int n) {
10     return (n * (n - 1) * (2 * n - 1)) / 6;
11 }
12
13 // first element, last element, number of elements
14 long long sum_pa(int a1, int an, int n) {
15     return ((a1 + an) * n) / 2;
16 }
17
18 // first element, number of elements, ratio
19 long long general_term_pa(int a1, int n, int r) {
20     return a1 + (n - 1) * r;
21 }
22
23 // first term, numbers of elements, ratio
24 long long sum_pg(int a1, int n, int q) {
25     return (a1 * (fexp(q, n) - 1)) / (q - 1);
26 }
27
28 // -1 < q < 1
29 // first term, ratio
30 long long sum_infinite_pg(int a1, double q) {
31     return a1 * (1 - q);
32 }
33
34 // first term, number of elements, ratio
35 long long general_term_pg(int a1, int n, int q) {
36     return a1 * fexp(q, n - 1);
37 }
38
39 // first element of original pa, first element of
40 // derived pa, number of elements of original pa,
41 // ratio of derived pa
42 long long sum_second_order_pa(int a1, int b1, int n,
43     int r) {
44     return a1 * n + (b1 * n * (n - 1)) / 2 + (r * n * (n
45         - 1) * (n - 2)) / 6
46 }
47
48 // GEOMETRIA
49 // seno
50 a / sen(a) = b / sen(b) = c / sen(c)
51
52 //cosseno
53 a^2 = b^2 + c^2 - 2*b*c*cos(a)
54
55 // area losango
56 A = (1/2) * diagonal_maior * diagonal_menor
57
58 // volume prisma
59 V = B * H
60
61 //volume esfera
62 V = (4/3) * PI * R^3
63
64 //volume piramide
65 V = (1/3) * B * H
66
67 //volume cone
68 V = (1/3) * PI * R^2 * H
69
70 //condicao de existencia
71 a - b | < c < a + b

```

8.13 Mdc

```

1 long long gcd(long long a, long long b){
2     return b ? gcd(b, a % b) : a;
3 }
4
5 // or just use __gcd(a,b)

```

8.14 Fast Exponentiation

```
1 ll fexp(ll b, ll e, ll mod) {  
2     ll res = 1;  
3     b %= mod;  
4     while(e){  
5         if(e & 1LL)
```

```
6         res = (res * b) % mod;  
7         e = e >> 1LL;  
8         b = (b * b) % mod;  
9     }  
10    return res;  
11 }
```