

DóndeEstásCR7

DóndeEstásCR7

09/08/2025

Índice

Bit Manipulation	2
1.1 Bits	2
Graph	2
2.1 Scc	2
Number Theorry	3
3.1 Number Theory	3
3.2 Phi Euler	4
3.3 Potenciación Binaria	5
3.4 Sum Of Divisors	5
Segment Tree	5
4.1 Find Two Numbers	5
4.2 Segment Tree Recursivo	5
4.3 Segment Tree V2	6
4.4 Segment Tree V3	7

Bit Manipulation

Técnicas para manipular bits individuales y operaciones a nivel de bit. Incluye macros útiles para competencias de programación.

1.1 Bits

Macros esenciales para manipulación de bits: verificar potencias de 2, establecer/limpiar bits, contar bits, y operaciones con LSB/MSB.

```
1 using ull = unsigned long long;
2 const ull UNSIGNED_LL_MAX = 18'446'744'073'709'551'615;
3 #define isPowerOfTwo(S) ((S) && !((S) & ((S) - 1))) //
4     Verifica si S es potencia de dos (y distinto de cero)
5 #define nearestPowerOfTwo(S) (1LL << lround(log2(S))) //
6     Retorna la potencia de dos mas cercana a S
7 #define modulo(S, N) ((S) & ((N) - 1)) // Calcula S % N
8     cuando N es potencia de dos
9
10 #define isOn(S, i) ((S) & (1LL<<(i))) // Verifica si el bit
11     esta encendido (bit en 1)
12 #define setBit(S, i) ((S) |= (1LL<<(i))) // Enciende el bit
13     (Lo pone en 1)
14 #define clearBit(S, i) ((S) &= ~(1LL<<(i))) // Apaga el bit
15     (Lo pone en 0)
16 #define toggleBit(S, i) ((S) ^= (1LL<<(i))) // Invierte el
17     estado del bit (0 <-> 1)
18 #define setAll(S, n) ((S) = ((n)>=64 ? ~0LL : (1LL <<
19     (n))-1)) // Enciende los primeros 'n' bits (idx-0)
20
21 #define lsb(S) ((S) & -(S)) // Extrae el bit menos
22     significativo 0100 (Least Significant Bit)
23 #define idxLastBit(x) __builtin_ctzll(x) // Numero de ceros
24     a la derecha (Posicion del LSB, idx-0)
25 #define msb(S) (1LL << (63 - __builtin_clzll(S))) // Extrae
26     el bit mas significativo 0100 (Most Significant Bit)
27 #define idxFirstBit(x) (63 - __builtin_clzll(x)) //
28     Posicion del MSB (63 - ceros a la izquierda, idx-0)
29 #define countAllOnes(x) __builtin_popcountll(x)
30 #define turnOffLastBit(S) ((S) & ((S) - 1)) // Apaga el
31     ultimo bit encendido (el menos significativo)
32 #define turnOnLastZero(S) ((S) | ((S) + 1)) // Enciende el
33     ultimo cero menos significativo
34 #define turnOffLastConsecutiveBits(S) ((S) & ((S) + 1)) //
35     Apaga todos los bits encendidos mas a la derecha
36     consecutivos
```

```
21 #define turnOnLastConsecutiveZeroes(S) ((S) | ((S) - 1)) //
22     Enciende los ceros consecutivos mas a la derecha
23 // Mascara de bits (mask -> subconjunto) 0(2^N)
24 for (int mask = 0; mask < (1 << N); mask++)
25
26 // Recorrer subconjuntos de un superconjunto (menos el
27     vacio)
28 int b = 0b1011; // Representacion binaria de un decimal en
29     int
30 for (int i = b; i; i = (i - 1) & b) {
31     cout << bitset<4>(i) << "\n";
32 }
33
34 void printBin(ll x) {
35     // 63 -> unsigned ll, 62 -> ll, 31 -> unsigned int, 30 ->
36     int
37     for (ll i = 63; i >= 0; i--)
38         cout << ((x >> i) & 1);
39     cout << '\n';
40 }
```

Graph

Algoritmos de grafos: DFS, BFS, componentes fuertemente conexas, y otras estructuras de datos para problemas de grafos.

2.1 SCC

Algoritmo de Tarjan para encontrar componentes fuertemente conexas (SCC) en un grafo dirigido.

```
1 // "These works to find a componente fuertemente conexas
2     that it's in directed graph"
3 struct SCC{
4     int N = 0, id;
5     vector<vector<int>> adj;
6     vector<int> ind, low;
7     stack<int> s;
8     vector<bool> in_stack;
9     vector<vector<int>> components;
```

```

9   vector<int> component_id;
10
11  //1-indexed
12  SCC(int n = 0){ N = n + 1, adj.assign(N, {}); }
13  SCC(const vector<vector<int>> & _adj){ adj = _adj, N =
14      adj.size(); }
15
16  void add_edge(int from, int to){
17      adj[from].push_back(to);
18  }
19
20  void dfs(int u){
21      low[u] = ind[u] = id++;
22      s.push(u);
23      in_stack[u] = true;
24      for(int v : adj[u]){
25          if(ind[v] == -1){
26              dfs(v);
27              low[u] = min(low[u], low[v]);
28          } else if(in_stack[v]){
29              low[u] = min(low[u], ind[v]);
30          }
31      }
32      if(low[u] == ind[u]){
33          components.emplace_back();
34          vector<int> & comp = components.back();
35          while(true){
36              assert(!s.empty());
37              int x = s.top(); s.pop();
38              in_stack[x] = false;
39              component_id[x] = components.size() - 1;
40              comp.push_back(x);
41              if(x == u) break;
42          }
43      }
44  }
45
46  vector<vector<int>> get(){
47      ind.assign(N, -1); low.assign(N, -1);
48      component_id.assign(N, -1);
49      s = stack<int>();
50      in_stack.assign(N, false);
51      id = 0;
52      components = {};
53      for(int i = 1; i < N; i++){
54          if(ind[i] == -1) dfs(i);
55      }
56      // reverse(components.begin(), components.end());
57      return components; // SCC in topological order
58  }
59  return components; // SCC in reverse topological order
60  }
61  };

```

Number Theory

3.1 Number Theory

```

1  class EulerTotiente {
2  public:
3      /* metodo en O(sqrt(n))
4      template <typename T>
5      T euler_classic(T n) {
6          T result = n;
7          for(T i = 2; i * i <= n; i++) {
8              if(n % i == 0) {
9                  while(n % i == 0) n /= i;
10                 result -= result / i;
11             }
12         }
13         if(n > 1) {
14             result -= result / n;
15         }
16         return result;
17     }
18
19     /* metodo en O(nlog(log(n))
20     void euler_faster(int n) {
21         vector<int> phi(n + 1);
22         for(int i = 0; i <= n; i++) {
23             phi[i] = i;
24         }
25         for(int i = 2; i <= n; i++) {
26             if(phi[i] == i) {
27                 for(int j = i; j <= n; j += i) {
28                     phi[j] -= phi[j] / i;
29                 }
30             }
31         }
32         for(int i = 1; i <= n; i++) {
33             cout << i << ' ' << phi[i] << '\n';
34         }
35     }
36 };
37
38 // Criba de Eratostenes: Hasta N = 10^6
39 // Con bitset<N> Hasta N = 10^8 en 1s
40 void sieve(vector<bool>& is_prime) {
41     int N = (int) is_prime.size();

```

```

42 if (!is_prime[0]) is_prime.assign(N+1, true);
43 is_prime[0] = is_prime[1] = false;
44 for (int p = 2; p * p <= N; p++) {
45     if (is_prime[p]) {
46         for (int i = p * p; i <= N; i += p) {
47             is_prime[i] = false;
48         }
49     }
50 }
51 }
52
53 // Divisores de N: Hasta N = 10^6
54 vector<int> divisores(int N) {
55     vector<int> divs;
56     for (int d = 1; d * d <= N; d++) {
57         if (N % d == 0) {
58             divs.push_back(d);
59             if (N / d != d) divs.push_back(N / d);
60         }
61     }
62     return divs;
63 }
64
65 // Factorizacion de N: Hasta N = 10^6
66 vector<pair<int, int>> factorizar(int N) {
67     vector<pair<int, int>> facts;
68     for (int p = 2; p * p <= N; p++) {
69         if (N % p == 0) {
70             int exp = 0;
71             while (N % p == 0) {
72                 exp++;
73                 N /= p;
74             }
75             facts.push_back({ p, exp });
76         }
77     }
78     if (N > 1) facts.push_back({ N, 1 });
79     return facts;
80 }
81
82 // Primalidad: Hasta N = 10^6 - O(sqrt(N))
83 bool isPrime(int N) {
84     if (N < 2) return false;
85     for (int d = 2; d * d <= N; d++) {
86         if (N % d == 0) return false;
87     }
88     return true;
89 }
90
91 // Maximo comun divisor (GCD): Algoritmo de Euclides
92 int gcd(int a, int b) {
93     if (a > b) swap(a, b);
94     if (a == 0) return b;

```

```

95     return gcd(b % a, a);
96 }
97
98 // Minimo comun multiplo (LCM): Calculado con GCD
99 int lcm(int a, int b) {
100     return (a * b) / gcd(a, b);
101 }

```

3.2 Phi Euler

```

1  /* Phi Euler
2  /* Phi(n) = contar la cantidad de numero coprimos entre 1
   a n
3  int phi(int n) {
4      int ans = n;
5      for (int i = 2; i * i <= n; i++) {
6          if (n % i == 0) {
7              while (n % i == 0) {
8                  n /= i;
9              }
10             ans -= ans / i;
11         }
12     }
13     if (n > 1) {
14         ans -= ans / n;
15     }
16     return ans;
17 }
18
19
20 /* phi(n) -> complex: O(log(log(n)))
21 void phi_1_to_n(int n) {
22     vector<int> phi(n + 1);
23     for (int i = 0; i <= n; i++)
24         phi[i] = i;
25
26     for (int i = 2; i <= n; i++) {
27         if (phi[i] == i) {
28             for (int j = i; j <= n; j += i)
29                 phi[j] -= phi[j] / i;
30         }
31     }
32 }

```

3.3 Potenciación Binaria

```
1  /* Binpow
2  long long binpow(long long a, long long b, long long m) {
3      a %= m;
4      long long res = 1;
5      while (b > 0) {
6          if (b & 1)
7              res = res * a % m;
8          a = a * a % m;
9          b >>= 1;
10     }
11     return res;
12 }
```

3.4 Sum Of Divisors

```
1  /* Sum of divs
2  long long SumOfDivisors(long long num) {
3      long long total = 1;
4
5      for (int i = 2; (long long)i * i <= num; i++) {
6          if (num % i == 0) {
7              int e = 0;
8              do {
9                  e++;
10                 num /= i;
11             } while (num % i == 0);
12
13             long long sum = 0, pow = 1;
14             do {
15                 sum += pow;
16                 pow *= i;
17             } while (e-- > 0);
18             total *= sum;
19         }
20     }
21     if (num > 1) {
22         total *= (1 + num);
23     }
24     return total;
25 }
```

Segment Tree

4.1 Find Two Numbers

```
1  // "find two number where the sum is x, and gcd(a, b) > 1" b
2  auto find = [&](ll x){
3      for(int d = 2; d <= x / 2; d++){
4          if(x % d == 0){
5              ll m = 1, n = (x / d) - 1;
6              ll a = d * m, b = d * n;
7              if(_gcd(a, b) > 1){
8                  cout << a << ' ' << b;
9                  ps();
10                 return;
11             }
12         }
13     }
14     };
```

4.2 Segment Tree Recursivo

```
1  template<typename T>
2  struct segment_tree{
3      int N;
4      T Z = 0;
5      vector<T> tree;
6      segment_tree(int N) : N(N) {
7          tree.resize(2 * N);
8      }
9
10     segment_tree(vector<T>& A){
11         N = (int) A.size();
12         tree.resize(2 * N);
13         build(A, 1, 0, N - 1);
14     }
15
16     auto& operator[](size_t i) { return tree[i]; } // this
17     // funcion works for get element int this position
18 private:
19
20     T op(T& a, T& b){ return a + b; }
21     // 0 (n)
22     void build(vector<T>& values, int node, int l, int r){
23         // if l and r are equal both are leaf node
24         // left node = [l, m]
```

```

24 // m = (l + r) / 2
25 // left and right are nodes
26 // left interval = [l, m], right interval = [m + 1, r]
27 // after complete fill nodes of left and right, we need
   // to fill the [l, r] node
28 if(l == r){
29     tree[node] = values[l];
30     return;
31 }
32 int m = (l + r) >> 1;
33 int left = node + 1;
34 int right = node + 2 * (m - l + 1);
35
36 build(values, left, l, m);
37 build(values, right, m + 1, r);
38
39 tree[node] = op(tree[left], tree[right]);
40 }
41
42 // O(log N)
43 void modify(int pos, T value, int node, int l, int r){
44     // if l and r are equal, we found our node and update it
45     if(l == r){
46         tree[node] = value;
47         return;
48     }
49     int m = (l + r) >> 1; // we get the mid
50     int left = node + 1;
51     int right = node + 2 * (m - l + 1);
52
53     if(pos <= m) modify(pos, value, left, l, m);
54     else modify(pos, value, right, m + 1, r);
55
56     tree[node] = op(tree[left], tree[right]);
57 }
58
59 void update(int pos, T value, int node, int l, int r){
60     // if l and r are equal, we found our node and update it
61     if(l == r){
62         tree[node] = op(tree[node], value);
63         return;
64     }
65     int m = (l + r) >> 1; // we get the mid
66     int left = node + 1;
67     int right = node + 2 * (m - l + 1);
68
69     if(pos <= m) update(pos, value, left, l, m);
70     else update(pos, value, right, m + 1, r);
71
72     tree[node] = op(tree[left], tree[right]);
73 }
74
75 // O(log N)

```

```

76 T query(int ql, int qr, int node, int l, int r){
77     if(r < ql || l > qr) return Z; // CHECK
78     if(ql <= l && r <= qr) return tree[node];
79     int m = (l + r) >> 1;
80     int left = node + 1;
81     int right = node + 2 * (m - l + 1);
82     T ansL = query(ql, qr, left, l, m);
83     T ansR = query(ql, qr, right, m + 1, r);
84     return op(ansL, ansR);
85 }
86 public:
87     void build(vector<T>& values){ build(values, 1, 0, N -
   1); }
88
89     void modify(int pos, T value){ modify(pos, value, 1, 0, N
   - 1); }
90
91     void update(int pos, T value){ update(pos, value, 1, 0, N
   - 1); }
92
93     T query(int ql, int qr){ return query(ql, qr, 1, 0, N -
   1); }
94 };

```

4.3 Segment Tree V2

```

1 // "This segment_tree I understand better how it works"
2 template<typename T>
3 struct seg_tree {
4     int N;
5     T Z = 0;
6     vector<T> tree;
7
8     seg_tree(int N) : N(N) {
9         tree.resize(4 * N);
10    }
11
12    seg_tree(vector<T>& A) {
13        N = (int)A.size();
14        tree.resize(4 * N);
15        build(A, 1, 0, N-1);
16    }
17 private:
18     T op(T a, T b) {
19         return a + b;
20     }
21 }
22

```

```

23 void build(vector<T>& a, int node, int left, int right)
24 {
25     if(left == right) {
26         tree[node] = a[left];
27         return;
28     }
29     int mid = (left + right) >> 1;
30     build(a, 2 * node, left, mid);
31     build(a, 2 * node + 1, mid + 1, right);
32     tree[node] = op(tree[2 * node], tree[2 * node + 1]);
33 }
34 void modify(int pos, T value, int node, int left, int
35             right) {
36     if(left == right) {
37         tree[node] = value;
38         return;
39     }
40     int mid = (left + right) >> 1;
41     if(pos <= mid)
42         modify(pos, value, 2 * node, left, mid);
43     else
44         modify(pos, value, 2 * node + 1, mid + 1,
45             right);
46     tree[node] = op(tree[2 * node], tree[2 * node + 1]);
47 }
48 T query(int l, int r, int node, int left, int right) {
49     if(r < left || l > right) return Z;
50     if(l <= left && right <= r) return tree[node];
51     int mid = (left + right) >> 1;
52     T leftSum = query(l, r, 2 * node, left, mid);
53     T rightSum = query(l, r, 2 * node + 1, mid + 1,
54         right);
55     return op(leftSum, rightSum);
56 }
57 public:
58 void build(vector<T>& a) { build(a, 1, 0, N-1); }
59 void modify(int pos, T value) { modify(pos, value, 1,
60     0, N-1); }
61 T query(int l, int r) { return query(l, r, 1, 0, N-1); }
62 };

```

4.4 Segment Tree V3

```

2 template<class T>
3 struct segment_tree{
4     int n;
5     vector<T> tree;
6
7     segment_tree(int n){
8         this->n = n;
9         tree.resize(2 * n);
10    }
11
12    segment_tree(vector<T>& values){
13        this->n = values.size();
14        tree.resize(2 * n);
15        for(int i = 0; i < n; i++) upd(i, values[i]);
16    }
17
18    //CHANGE
19    T compare(T a, T b){
20        return a + b;
21    }
22
23    void modify(int index, T value){
24        index += n;
25        tree[index] = value;
26        for(index >= 1; index >= 1; index >= 1)
27            tree[index] = compare(tree[2 * index], tree[2 *
28                index + 1]);
29    }
30
31    void upd(int index, T value){
32        index += n;
33        tree[index] = compare(tree[index], value);
34        for(index >= 1; index >= 1; index >= 1)
35            tree[index] = compare(tree[2 * index], tree[2 *
36                index + 1]);
37    }
38
39    //BOTTOM - TOP
40    T query(int first, int last){
41        first += n, last += n;
42        T ans = 0;
43        while(first <= last){
44            if(first % 2 == 1) ans = compare(ans,
45                tree[first++]);
46            if(last % 2 == 0) ans = compare(ans,
47                tree[last--]);
48            first >= 1, last >= 1;
49        }
50        return ans;
51    }
52 };

```

```
1 // snippet seg_tree_2 "Description" b
```