DóndeEstásCR7

DóndeEstásCR7

09/08/2025

Índice

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.

```cpp
using ull = unsigned long long;
const ull UNSIGNED_LL_MAX = 18'446'744'073'709'551'615;
#define isPowerOfTwo(S) ((S) && !((S) & ((S) - 1))) // Verifica si S es potencia de dos (y distinto de cero)
#define nearestPowerOfTwo(S) (1LL << lround(log2(S))) // Retorna la potencia de dos mas cercana a S
#define modulo(S, N) ((S) & ((N) - 1)) // Calcula S % N cuando N es potencia de dos

#define isOn(S, i) ((S) & (1LL<<(i))) // Verifica si el bit esta encendido (bit en 1)
#define setBit(S, i) ((S) |= (1LL<<(i))) // Enciende el bit (Lo pone en 1)
#define clearBit(S, i) ((S) &= ~(1LL<<(i))) // Apaga el bit (Lo pone en 0)
#define toggleBit(S, i) ((S) ^= (1LL<<(i))) // Invierte el estado del bit (0 <-> 1)
#define setAll(S, n) ((S) = ((n)>=64 ? ~0LL : (1LL << (n))-1)) // Enciende los primeros 'n' bits (idx-0)

#define lsb(S) ((S) & -(S)) // Extrae el bit menos significativo 0100 (Least Significant Bit)
#define idxLastBit(x) __builtin_ctzll(x) // Numero de ceros a la derecha (Posicion del LSB, idx-0)
#define msb(S) (1LL << (63 - __builtin_clzll(S))) // Extrae el bit mas significativo 0100 (Most Significant Bit)
#define idxFirstBit(x) (63 - __builtin_clzll(x)) // Posicion del MSB (63 - ceros a la izquierda, idx-0)
#define countAllOnes(x) __builtin_popcountll(x)
#define turnOffLastBit(S) ((S) & ((S) - 1)) // Apaga el ultimo bit encendido (el menos significativo)
#define turnOnLastZero(S) ((S) | ((S) + 1)) // Enciende el ultimo cero menos significativo
#define turnOffLastConsecutiveBits(S) ((S) & ((S) + 1)) // Apaga todos los bits encendidos mas a la derecha consecutivos
#define turnOnLastConsecutiveZeroes(S) ((S) | ((S) - 1)) // Enciende los ceros consecutivos mas a la derecha

// Mascara de bits (mask -> subconjunto) O(2^N)
for (int mask = 0; mask < (1 << N); mask++)

// Recorrer subconjuntos de un superconjunto (menos el vacio)
int b = 0b1011; // Representacion binaria de un decimal en int
for (int i = b; i; i = (i - 1) & b) {
  cout << bitset<4>(i) << "\n";
}

void printBin(ll x) {
  // 63 -> unsigned ll, 62 -> ll, 31 -> unsigned int, 30 -> int
  for (ll i = 63; i >= 0; i--)
    cout << ((x >> i) & 1);
  cout << '\n';
}
```

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.

```cpp
// "These works to find a componente fuertemente conexa that it's in directed graph"
struct SCC{
  int N = 0, id;
  vector<vector<int>> adj;
  vector<int> ind, low;
  stack<int> s;
  vector<bool> in_stack;
  vector<vector<int>> components;
```

```
  vector<int> component_id;

  //1-indexed
  SCC(int n = 0){ N = n + 1, adj.assign(N, {}); }
  SCC(const vector<vector<int>> & _adj){ adj = _adj, N =
      adj.size(); }

  void add_edge(int from, int to){
    adj[from].push_back(to);
  }

  void dfs(int u){
    low[u] = ind[u] = id++;
    s.push(u);
    in_stack[u] = true;
    for(int v : adj[u]){
      if(ind[v] == -1){
        dfs(v);
        low[u] = min(low[u], low[v]);
      }else if(in_stack[v]){
        low[u] = min(low[u], ind[v]);
      }
    }
    if(low[u] == ind[u]){
      components.emplace_back();
      vector<int> & comp = components.back();
      while(true){
        assert(!s.empty());
        int x = s.top(); s.pop();
        in_stack[x] = false;
        component_id[x] = components.size() - 1;
        comp.push_back(x);
        if(x == u) break;
      }
    }
  }

  vector<vector<int>> get(){
    ind.assign(N, - 1); low.assign(N, -1);
        component_id.assign(N, -1);
    s = stack<int>();
    in_stack.assign(N, false);
    id = 0;
    components = {};
    for(int i = 1; i < N; i++)
      if(ind[i] == -1) dfs(i);

    // reverse(components.begin(), components.end());
        return components; // SCC in topological order
    return components; // SCC in reverse topological order
  }
};
```

Number Theorry

3.1 Number Theory

```
class EulerTotiente {
  public:
  //* metodo en O(sqrt(n))
  template <typename T>
  T euler_classic(T n) {
    T result = n;
    for(T i = 2; i * i <= n; i++) {
      if(n % i == 0) {
        while(n % i == 0) n /= i;
        result -= result / i;
      }
    }
    if(n > 1) {
      result -= result / n;
    }
    return result;
  }

  //* metodo en O(nlog(log(n))
  void euler_faster(int n) {
    vector<int> phi(n + 1);
    for(int i = 0; i <= n; i++) {
      phi[i] = i;
    }
    for(int i = 2; i <= n; i++) {
      if(phi[i] == i) {
        for(int j = i; j <= n; j += i) {
          phi[j] -= phi[j] / i;
        }
      }
    }
    for(int i = 1; i <= n; i++) {
      cout << i << ' ' << phi[i] << '\n';
    }
  }
};

// Criba de Eratostenes: Hasta N = 10^6
// Con bitset<N> Hasta N = 10^8 en 1s
void sieve(vector<bool>& is_prime) {
  int N = (int) is_prime.size();
```

```cpp
  if (!is_prime[0]) is_prime.assign(N+1, true);
  is_prime[0] = is_prime[1] = false;
  for (int p = 2; p * p <= N; p++) {
    if (is_prime[p]) {
      for (int i = p * p; i <= N; i += p) {
        is_prime[i] = false;
      }
    }
  }
}

// Divisores de N: Hasta N = 10^6
vector<int> divisores(int N) {
  vector<int> divs;
  for (int d = 1; d * d <= N; d++) {
    if (N % d == 0) {
      divs.push_back(d);
      if (N / d != d) divs.push_back(N / d);
    }
  }
  return divs;
}

// Factorizacion de N: Hasta N = 10^6
vector<pair<int, int>> factorizar(int N) {
  vector<pair<int, int>> facts;
  for (int p = 2; p * p <= N; p++) {
    if (N % p == 0) {
      int exp = 0;
      while (N % p == 0) {
        exp++;
        N /= p;
      }
      facts.push_back({ p, exp });
    }
  }
  if (N > 1) facts.push_back({ N, 1 });
  return facts;
}

// Primalidad: Hasta N = 10^6 - O(sqrt(N))
bool isPrime(int N) {
  if (N < 2) return false;
  for (int d = 2; d * d <= N; d++) {
    if (N % d == 0) return false;
  }
  return true;
}

// Maximo comun divisor (GCD): Algoritmo de Euclides
int gcd(int a, int b) {
  if (a > b) swap(a, b);
  if (a == 0) return b;
  return gcd(b % a, a);
}

// Minimo comun multiplo (LCM): Calculado con GCD
int lcm(int a, int b) {
  return (a * b) / gcd(a, b);
}
```

3.2 Phi Euler

```cpp
//* Phi Euler
//* Phi(n) = contar la cantidad de numero coprimos entre 1 a n
int phi(int n) {
    int ans = n;
    for(int i = 2; i * i <= n; i++) {
        if(n % i == 0) {
            while (n % i == 0) {
                n /= i;
            }
            ans -= ans / i;
        }
    }
    if(n > 1) {
        ans -= ans / n;
    }
    return ans;
}


//* phi(n) -> complex: O(log(log(n)))
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

3.3 Potenciación Binaria

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

3.4 Sum Of Divisors

```cpp
//* Sum of divs
long long SumOfDivisors(long long num) {
    long long total = 1;

    for (int i = 2; (long long)i * i <= num; i++) {
        if (num % i == 0) {
            int e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);

            long long sum = 0, pow = 1;
            do {
                sum += pow;
                pow *= i;
            } while (e-- > 0);
            total *= sum;
        }
    }
    if (num > 1) {
        total *= (1 + num);
    }
    return total;
}
```

Segment Tree

4.1 Find Two Numbers

```cpp
// "find two number where the sum is x, and gcd(a, b) > 1" b
auto find = [&](ll x){
  for(int d = 2; d <= x / 2; d++){
    if(x % d == 0){
      ll m = 1, n = (x / d) - 1;
      ll a = d * m, b = d * n;
      if(__gcd(a, b) > 1){
        cout<< a << ' ' << b;
        ps();
        return;
      }
    }
  }
};
```

4.2 Segment Tree Recursivo

```cpp
template<typename T>
struct segment_tree{
  int N;
  T Z = 0;
  vector<T> tree;
  segment_tree(int N) : N(N) {
    tree.resize(2 * N);
  }

  segment_tree(vector<T>& A){
    N = (int) A.size();
    tree.resize(2 * N);
    build(A, 1, 0, N - 1);
  }

  auto& operator[](size_t i) { return tree[i]; } // this
      funciton works for get element int this position
private:

  T op(T& a, T& b){ return a + b; }
  // O (n)
  void build(vector<T>& values, int node, int l, int r){
    // if l and r are equal both are leaf node
    // left node = [l, m]
```

```
    // m = (l + r) / 2
    // left and right are nodes
    // left interval = [l, m], right intervla = [m + 1, r]
    // after complete fill nodes of left and right, we need to fill the [l, r] node
    if(l == r){
      tree[node] = values[l];
      return;
    }
    int m = (l + r) >> 1;
    int left = node + 1;
    int right = node + 2 * (m - l + 1);

    build(values, left, l, m);
    build(values, right, m + 1, r);

    tree[node] = op(tree[left], tree[right]);
  }

  // O (log N)
  void modify(int pos, T value, int node,  int l,  int r){
    // if l and r are equal, we found our node and update it
    if(l == r){
      tree[node] = value;
      return;
    }
    int m = (l + r) >> 1; // we get the mid
    int left = node + 1;
    int right = node + 2 * (m - l + 1);

    if(pos <= m) modify(pos, value, left, l, m);
    else modify(pos, value, right, m + 1, r);

    tree[node] = op(tree[left], tree[right]);
  }

  void update(int pos, T value, int node,  int l,  int r){
    // if l and r are equal, we found our node and update it
    if(l == r){
      tree[node] = op(tree[node], value);
      return;
    }
    int m = (l + r) >> 1; // we get the mid
    int left = node + 1;
    int right = node + 2 * (m - l + 1);

    if(pos <= m) update(pos, value, left, l, m);
    else update(pos, value, right, m + 1, r);

    tree[node] = op(tree[left], tree[right]);
  }

  // O(log N)
  T query(int ql, int qr, int node, int l, int r){
    if(r < ql || l > qr) return Z; // CHECK
    if(ql <= l && r <= qr) return tree[node];
    int m = (l + r) >> 1;
    int left = node + 1;
    int right = node + 2 * (m - l + 1);
    T ansL = query(ql, qr, left, l, m);
    T ansR = query(ql, qr, right, m + 1, r);
    return op(ansL, ansR);
  }
public:
  void build(vector<T>& values){ build(values, 1, 0, N - 1); }

  void modify(int pos, T value){ modify(pos, value, 1, 0, N - 1); }

  void update(int pos, T value){ update(pos, value, 1, 0, N - 1); }

  T query(int ql, int qr){ return query(ql, qr, 1, 0, N - 1); }
};
```

4.3 Segment Tree V2

```
// "This segment_tree I understand better how it works"
template<typename T>
struct seg_tree {
    int N;
    T Z = 0;
    vector<T> tree;

    seg_tree(int N) : N(N) {
        tree.resize(4 * N);
    }

    seg_tree(vector<T>& A) {
        N = (int)A.size();
        tree.resize(4 * N);
        build(A, 1, 0, N-1);
    }

private:
    T op(T a, T b) {
        return a + b;
    }

```

```
    void build(vector<T>& a, int node, int left, int right)
        {
        if(left == right) {
            tree[node] = a[left];
            return;
        }
        int mid = (left + right) >> 1;
        build(a, 2 * node, left, mid);
        build(a, 2 * node + 1, mid + 1, right);
        tree[node] = op(tree[2 * node], tree[2 * node + 1]);
    }

    void modify(int pos, T value, int node, int left, int
        right) {
        if(left == right) {
            tree[node] = value;
            return;
        }
        int mid = (left + right) >> 1;
        if(pos <= mid)
            modify(pos, value, 2 * node, left, mid);
        else
            modify(pos, value, 2 * node + 1, mid + 1,
                right);
        tree[node] = op(tree[2 * node], tree[2 * node + 1]);
    }

    T query(int l, int r, int node, int left, int right) {
        if(r < left || l > right) return Z;
        if(l <= left && right <= r) return tree[node];
        int mid = (left + right) >> 1;
        T leftSum = query(l, r, 2 * node, left, mid);
        T rightSum = query(l, r, 2 * node + 1, mid + 1,
            right);
        return op(leftSum, rightSum);
    }

public:
    void build(vector<T>& a) { build(a, 1, 0, N-1); }
    void modify(int pos, T value) { modify(pos, value, 1,
        0, N-1); }
    T query(int l, int r) { return query(l, r, 1, 0, N-1); }
};
```

4.4 Segment Tree V3

```
// snippet seg_tree_2 "Description" b
template<class T>
struct segment_tree{
    int n;
    vector<T> tree;

    segment_tree(int n){
        this -> n = n;
        tree.resize(2 * n);
    }

    segment_tree(vector<T>& values){
        this -> n = values.size();
        tree.resize(2 * n);
        for(int i = 0; i < n; i++) upd(i, values[i]);
    }

    //CHANGE
    T compare(T a, T b){
        return a + b;
    }

    void modify(int index, T value){
        index += n;
        tree[index] = value;
        for(index >>= 1; index >= 1; index >>= 1)
            tree[index]= compare(tree[2 * index], tree[2 *
            index + 1]);
    }

    void upd(int index, T value){
        index += n;
        tree[index] = compare(tree[index], value);
        for(index >>= 1; index >= 1; index >>= 1)
            tree[index]= compare(tree[2 * index], tree[2 *
            index + 1]);
    }

    //BOTTOM - TOP
    T query(int first, int last){
        first += n, last += n;
        T ans = 0;
        while(first <= last){
            if(first % 2 == 1) ans = compare(ans,
                tree[first++]);
            if(last % 2 == 0) ans = compare(ans,
                tree[last--]);
            first >>= 1, last >>= 1;
        }
        return ans;
  }
};
```