Sparse Table

```cpp
#include <bits/stdc++.h>
using namespace std;

// Código de https://www.youtube.com/watch?v=0jWeUdxrGm4

const int MAX_N = 100'005;
const int LOG = 17;
int a[MAX_N];
int m[MAX_N][LOG];
int log_precom[MAX_N];

int query(int L, int R) {
    int length = R - L + 1;
    int k = log_precom[length];

    return min(m[L][k], m[R-(1<<k)+1][k]);
}

int main() {
    int n; cin >> n;

    // Log precomputation to n
    log_precom[1] = 0;
    for (int i = 2; i <= n; i++) {
        log_precom[i] = log_precom[i/2]+1;
    }

    for (int i = 0; i < n; i++) {
        cin >> a[i];
        m[i][0] = a[i];
    }

    // Preprocessing
    for (int k = 1; k < LOG; k++) {
        for (int i = 0; i + (1<<k) -1 < n; i++) {
            m[i][k] = min(m[i][k-1], m[i+(1<<(k-1))][k-1]);
        }
    }

    // Queries
    int q;
    cin >> q;
    for (int i = 0; i < q; i++) {
        int L, R;
        cin >> L >> R;
        cout << query(L, R) << "\n";
    }
}
```

Suffix array ITMO

```
void binary_search(vi & p, vi&c, string  s, string  si, int L, int R){
      if(L>R){
      indeX = -1;
      return;
      }

      int mid = (L+R)/2;

      if(  s[p[mid]]  == si[0]){
      indeX = mid;
      return;
      }

      if(si[0] < s[p[mid]]){
      binary_search(p,c,s,si, L,mid-1);
      }
      else{
      binary_search(p,c,s,si, mid+1, R);
      }
      return;
}


// Sort the array p based on the keys in the array c
void count_sort(vi & p, vi & c){
      int n = p.size();
      vi cont(n);

      for(int x: c){
      cont[x]++;
      }

      vi p_new(n), position(n);
      position[0] = 0;
      // Count the frecuencies of every ordered number
      For(i,1,n){
      position[i] = position[i-1] + cont[i-1];
      }

      // Pun the elements in the corresponding buckets
      for(int x: p){
      int i = c[x];
      p_new[position[i]] = x;
      // If in the future exist some repeated number, just add 1 to add it
in the next cell
```

```
        position[i]++;
        }
        p = p_new;

}


int main(){
        // string t
        string s; cin >> s;

        int n = s.size();
        vi p(n), c(n);

        {
        // Phase 0
        // Array of pairs
        vector<pair<char,int>> ArrayLetterAndIndex(n);
        For(i,0,n){
              // pair = letter and indeX
              ArrayLetterAndIndex[i] = {s[i], i};
        }
        // Sorted based on the letter and then the indeX
        sort(ArrayLetterAndIndex.begin(), ArrayLetterAndIndex.end());
        // p is an array of ordered indexs of starting positions of the
suffixs
        For(i,0,n){
              p[i] = ArrayLetterAndIndex[i].second;
        }
        For(i,1,n){
              c[p[i]] = c[p[i-1]];
              // If the letters are different
              if(ArrayLetterAndIndex[i].first !=
ArrayLetterAndIndex[i-1].first){
                    c[p[i]]++;
              }
        }
        }

        // Next phase
        int k = 1;
        while(k < n){
        For(i,0,n){
              p[i] = (p[i] - k + n) % n;
        }
        // We only need to sort one time
        count_sort(p,c);
```

```
	vi c_new(n);
	c_new[p[0]] = 0;
	For(i,1,n){
		pii prev = {c[p[i-1]], c[ (p[i-1] + k ) % n ]};
		pii now = {c[p[i]], c[ (p[i] + k ) % n ]};
		c_new[p[i]] = c_new[p[i-1]];
		if(now != prev){
			c_new[p[i]]++;
		}
	}
	c = c_new;
	k<<=1;
	}

	//Suffix Array
	// For(i,0,n){
	//	cout << p[i] << " " ;
	// }
	// pendl;
	cin >> n;
	string si;

	while(n--){
	cin >> si;

	binary_search(p, c, s, si, 0, s.size()-1);

	while(indeX-1 >= 0 and si[0] == s[p[indeX-1]]){
		indeX--;
	}
	function<bool()>findPattern = [&]()->bool{
		if(indeX == -1){
			return false;
		}

		while (indeX < s.size() and s[p[indeX]] == si[0]){
			// Si la longitud del sufijo es menor que si pues para
que checarlo si no dará el ancho
			// Solo si lo anterior no se cumple evaluar
			//cout << "First: " << s.size() - p[indeX] << endl;
			//cout << "Second: " << si.size() << endl;
			if( !( (s.size() - p[indeX]) < si.size() ) ){
			// Empezando en el indeX del sufijo, buscar si desde
aquí está el patrón
			for(int i = p[indeX], origen = 0; origen < si.size();
i++, origen++){
				if(s[i] != si[origen]){
```

```
                              break;
                        }
                        if(origen == si.size()-1){
                              return true;
                        }
                  }
                  }
                  indeX++;
            }

            return false;
      };

      bool flag = findPattern();

      cout << (flag ? "Yes":"No") << endl;

      }

      return 0;
}
```

Knapsack

Minimizar - Alien Crop Triangles

```
int main(){tempest
      int B, N;
      cin >> B >> N;
      vpii arr(B);
      cin >> arr;
      int menorPrecioDeBolsa = INT_MAX;
      int alMenosUnaNoCero = 0;

      for(pii & x: arr){
      alMenosUnaNoCero = max(alMenosUnaNoCero, x.first);
      menorPrecioDeBolsa = min(menorPrecioDeBolsa, x.second);
      }

      if(N == 0){
      cout << "0" << endl;
      return 0;
      }

      if(alMenosUnaNoCero == 0){
      cout << "-1" << endl;
      return 0;
      }

      if(menorPrecioDeBolsa == 0){
      cout << "0" << endl;
      return 0;
      }

      ld area = 0;
      For ( i, 0 ,N) {
      int A, B, C;
      cin >> A >> B >> C;
      ll s = A + B + C;
      A <<= 1;
      B <<= 1;
      C <<= 1;
      area += sqrt((ld)s * (s - A) * (ld)(s - B) * (s - C));
      }
      area /= (ld)(120.0);

      ll kgNecesarios = area;
      if (abs((ld)kgNecesarios - area) > eps)
      kgNecesarios++;
      vll DP(kgNecesarios + 30, INT_MAX >> 1);

      DP[0] = 0;
      for (pii &it : arr) {
      for (int i = it.first ; i < kgNecesarios + 30 ; i++) {
            DP[i] = min(DP[i - it.first] + it.second, DP[i]);
      }
      }

      ll ans = DP[kgNecesarios];
      For(i,kgNecesarios,kgNecesarios+30){
      if(DP[i] < ans){
            ans = DP[i];
      }
      }
      cout << ans << endl;
      return 0;
}
```

Maximizar - O-N Knapsack

```
int N_objetos, M_Capacidad;

MatrixI dp;
vi cost;
vi profit;

// Return the max profit considering objetcs the number of objects and the
max capacity of the bag
int KnapSack(int object, int capital){
      if(object <= 0 || capital <= 0){
      return 0;
      }
      // Si ya está calculado
      if(dp[object][capital] != -1){
      return dp[object][capital];
      }
      // If weight of the nth item is more than
      // Knapsack capacity W, then this item cannot
      // be included in the optimal solution
      if(cost[object] > capital){
      dp[object][capital] = KnapSack(object-1, capital);
      return dp[object][capital];
      }
      dp[object][capital] = max(KnapSack(object-1, capital),
KnapSack(object,capital-cost[object]) + profit[object]);
      return dp[object][capital];
}
```

```
int main(){
	cin >>  N_objetos >> M_Capacidad;
	cost.resize(N_objetos+1);
	profit.resize(N_objetos+1);
	// Cada objeto tiene un beneficio y un coste
	ForI(i,1,N_objetos){
	int benefit, weight; cin >> weight >> benefit;
	cost[i] = weight;
	profit[i] = benefit;
	}
	dp.resize(N_objetos+1, vi(M_Capacidad+1,-1));

	int ans = KnapSack(N_objetos, M_Capacidad);

	cout << ans << endl;

	return 0;
}
```

0-1 Knapsack

```
// Return the max profit considering objetcs the number of objects and the
max capacity of the bag
int KnapSack(int object, int capital){
	if(object <= 0 || capital <= 0){
	return 0;
	}
	// Si ya está calculado
	if(dp[object][capital] != -1){
	return dp[object][capital];
	}
	// If weight of the nth item is more than
	// Knapsack capacity W, then this item cannot
	// be included in the optimal solution
	if(cost[object] > capital){
	dp[object][capital] = KnapSack(object-1, capital);
	return dp[object][capital];
	}
	dp[object][capital] = max(KnapSack(object-1,capital-cost[object]) +
profit[object], KnapSack(object-1, capital));
	return dp[object][capital];
}
```

Plantilla

```
// Basic
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
#define endl "\n"
// Loops
#define For(i,l,r) for(int i = l; i < (r); ++i)
#define ForI(i,l,r) for(int i = l; i <= (r); ++i)
#define ForR(i,l,r) for(int i = r-1; i >= 0; i--)
#define ForRI(i,l,r) for(int i = r; i >= (l); i--)
// Shortcuts
#define ___ ios::sync_with_stdio(false);cin.tie(0);
#define pb push_back
#define popb pop_back
#define empb emplace_back
#define sz(a) ((int)((a).size()))
#define mp make_pair
#define pb push_back
#define f first
#define s second
#define ll long long int
// Basic DS
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;
typedef tuple<int, int, int> Tupleiii;
// Vectors
typedef vector<ll> vll;
typedef vector<string> vs;
typedef vector<char> vc;
typedef vector<int> vi;
typedef vector<bool> vb;
typedef vector<pii> vpii;
typedef vector<double> vd;
typedef vector<ld> vld;
// Matrixs
typedef vector<vi> MatrixI;
typedef vector<vc> MatrixC;
typedef vector<vb> MatrixB;
typedef vector<vld> Matrixld;
typedef vector<vpii> MatrixPii;

//    BITS
#define LSOne(S) ((S) & -(S))
```

```
#define isOn(S, j) (S & (1<<j))
#define setBit(S, j) (S |= (1<<j))
#define clearBit(S, j) (S &= ~(1<<j))
#define toggleBit(S, j) (S ^= (1<<j))
#define lowBit(S) (S & (-S))
#define setAll(S, n) (S = (1<<n)-1)

#define IPOT(S) (!(S & (S-1)))      // isPowerOfTwo
#define modB(S, N) ((S) & (N-1))    // returns S % N, where N is a power
of 2
#define NPOT(S) (1<<lround(log2(S))) //nearestPowerOfTwo
#define offLB(S) ((S) & (S-1))      //turnOffLastBit
#define onLZ(S) ((S) | (S+1))       //turnOnLastZero
#define offLCB(S) ((S) & (S+1))     //turnOffLastConsecutiveBits
#define onLCZ(S) ((S) | (S-1))      //turnOnLastConsecutiveZeroes
// count the number of active bits
// 32 bits - __builtin_popcount(S)
// 64 bits - __builtin_ctz(S)


// Constants
const ld eps = (ld)1e-9;
#define INF 99999999999
const ll mod = 1e9 + 7;
const int MaxN = 1000010;
const int M = 1007;


// Sobrecarga
// Input Operatirons Pair, Vector
template<class T, class V>istream& operator>>(istream &in, pair<T, V> &a)
{in >> a.first >> a.second; return in;}
template<class T>istream& operator>>(istream &in, vector<T> &a) {for (auto
&i : a) {in >> i;} return in;}

// Output Operations Pair, Vector
template<class T, class V>ostream& operator<<(ostream &os, pair<T, V> &a)
{os << a.first << " " << a.second; return os;}
template<class T>ostream& operator<<(ostream &os, vector<T> &a) {for (int
i = 0 ; i < sz(a) ; i++) {if (i != 0) {os << ' ';} os << a[i];} return
os;}

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;

// x-y
// Movements in a Matrix
vpii movements = {{0,1},{1,0},{0,-1},{-1,0}};
//                      Up   Up-Right  Right  Right-Down   Down   Down-Left
Left   Left-Up
// vpii movements = {{0,1}, {1,1}   ,{1,0},     {1,-1}   ,{0,-1},   {-1,-1}
,{-1,0}, {-1,1}} ;

// min heap =  priority_queue <int, vector<int>, greater<int> > pq;

int n,m;
bool possible(int x, int y){
      return x >= 0 and x < n and y >= 0 and y < m;
}


int main(){___

}
```