

# Competitive Programming Library

Bernardo Flores Salmeron

<b>1</b>	<b>Template</b>	<b>3</b>			
<b>2</b>	<b>Funcoes Interessantes</b>	<b>3</b>			
2.1	GCD . . . . .	3			
2.2	HIPOTENUSA . . . . .	3			
2.3	SCANF DE UMA STRING . . . . .	3			
2.4	PRINTF PARA UMA STRING . . . . .	3			
2.5	CLIMITS . . . . .	3			
2.6	ROTATE (LEFT) . . . . .	3			
2.7	ROTATE (RIGHT) . . . . .	3			
2.8	WIDTH . . . . .	3			
2.9	INT TO STRING (C++11) . . . . .	3			
2.10	PERMUTAÇÃO . . . . .	3			
2.11	MAIOR E MENOR ELEMENTO NUM VETOR . . . . .	3			
2.12	CHECAGEM E TRANSFORMAÇÃO DE CARACTERE . . . . .	3			
2.13	SUBSTRING . . . . .	3			
2.14	REMOVE REPETIÇÕES CONTÍNUAS NUM VETOR . . . . .	4			
2.15	CHECAGEM DE BITS . . . . .	4			
2.16	INT TO BINARY STRING . . . . .	4			
2.17	BINARY STRING TO INT . . . . .	4			
2.18	STRING TO LONG LONG . . . . .	4			
2.19	SPLIT FUNCTION . . . . .	4			
2.20	LEITURA NUM ARQUIVO . . . . .	4			
2.21	ESCRITA EM ARQUIVO . . . . .	4			
2.22	CHECAGEM BRUTE FORCE COM SOLUCAO . . . . .	4			
<b>3</b>	<b>DP</b>	<b>5</b>			
3.1	PROBLEMA DO TROCO . . . . .	5			
3.2	PROBLEMA DA MOCHILA . . . . .	5			
3.3	CATALAN (1, 1, 2, 5, 14, 42, 132, 429) - $O(n)$ . . . . .	5			
3.4	LONGEST COMMON SUBSEQUENCE - $O(n^2)$ . . . . .	5			
3.5	LONGEST COMMON SUBSTRING - $O(n^2)$ . . . . .	6			
3.6	LONGEST INCREASING SUBSEQUENCE - $O(n \log(n))$ . . . . .	6			
3.7	LONGEST INCREASING SUBSEQUENCE 2D (NOT SORTED) - $O(n \log(n))$ . . . . .	6			
3.8	LONGEST INCREASING SUBSEQUENCE 2D (SORTED) - $O(n \log(n))$ . . . . .	7			
			<b>3.9</b>	ACHAR MAIOR PALÍNDROMO . . . . .	7
			3.10	SUBSET SUM COM BITSET - $O((maxSum)*n/32)$ . . . . .	7
			3.11	DIGIT DP . . . . .	7
		<b>4</b>	<b>Math</b>		<b>8</b>
		4.1	INCLUSÃO-EXCLUSÃO . . . . .	8	
		4.2	TODOS OS DIVISORES DE N - $O(\sqrt{n})$ . . . . .	8	
		4.3	ALGORITMO DE EUCLIDES ESTENDIDO - $O(\log(\min(a, b)))$ . . . . .	8	
		4.4	TEOREMA CHINES DO RESTO . . . . .	9	
		4.5	BELL NUMBERS (NUMBER OF WAYS TO PARTITION A SET) - $O(n^2)$ . . . . .	9	
		4.6	FATORAÇÃO SIMPLES - $O(\sqrt{n})$ . . . . .	9	
		4.7	FATORAÇÃO PRA MULTIPLAS QUERIES - $O(n(\log(\log(n))))$ . . . . .	9	
		4.8	CRIVO COMUM E SEGMENTADO COM BITSET . . . . .	10	
		4.9	CHECAR SE UM NUMERO É PRIMO - $O(\sqrt{n})$ . . . . .	10	
		4.10	EXPONENCIAÇÃO BINÁRIA - $O(\log(n))$ . . . . .	10	
		4.11	PRECOMPUTAR COMBINAÇÃO nCr - $O(n^2)$ . . . . .	10	
		4.12	COMBINAÇÃO nCr - $O(n)$ . . . . .	11	
		4.13	POLLARD RHO (FIND A DIVISOR FOR N) - $O(n^{1/4})$ . . . . .	11	
		4.14	EQUAÇÃO DIOFANTINA (ACHAR UMA SOLUÇÃO) . . . . .	11	
		4.15	EULER'S TOTIENT FUNCTION . . . . .	11	
		<b>5</b>	<b>Geometry</b>		<b>12</b>
		5.1	PRODUTO VETORIAL . . . . .	12	
		5.2	DISTANCIA PONTO RETA . . . . .	12	
		5.3	ÁREA DE POLÍGONOS . . . . .	13	
		5.4	PONTOS DENTRO DE UM POLIGONO . . . . .	13	
		5.5	PONTOS DENTRO E NA BORDA DE UM POLIGONO . . . . .	13	
		5.6	INTERSECÇÃO DE RETAS . . . . .	14	
		5.7	INTERSECÇÃO DE SEGMENTOS . . . . .	14	
		5.8	INTERSECÇÃO DE CIRCULOS (2PTOS) . . . . .	14	
		5.9	PONTO DENTRO DE UM POLIGONO . . . . .	15	
		5.10	CLOSEST PAIR OF POINTS . . . . .	15	
		5.11	CENTRO DE MASSA DE UM POLÍGONO . . . . .	16	
		5.12	POINT AND LINE STRUCT . . . . .	16	
		5.13	CONVEX HULL - $O(n \log(n))$ . . . . .	17	

5.14	UPPER AND LOWER HULL - $O(n \log(n))$	17	9.4	INFIX TO PREFIX	32
5.15	CONDIÇÃO DE EXISTÊNCIA DE UM TRIÂNGULO	17	9.5	KADANE (MAIOR SOMA NUM VETOR) – $O(n)$	32
5.16	ÁREA POLÍGONO 3D – $O(n)$	18	9.6	KADANE 2D – $O(n^3)$	32
<b>6</b>	<b>Strings</b>	<b>18</b>	9.7	KADANE (SEGMENT TREE) - (QUERY IN RANGE $O(\log n)$ )	33
6.1	KMP ALGORITMO – $O(n+m)$	18	9.8	COMPRESSAO DE PONTOS	33
6.2	TRIE	19	9.9	TORRE DE HANOI – $O(2^{n-1})$	34
6.3	TRIE (CONTANDO TRANSICOES)	19	9.10	FIBONACCI MATRIX EXPONENTIATION - $O(\log n)$	34
6.4	TRIE (MAXIMUM XOR BETWEEN TWO ELEMENTS)	20	9.11	2-SAT PROBLEM - $O(V+E)$	34
6.5	TRIE (MAXIMUM XOR SUM)	20			
6.6	Z-FUNCTION – $O(n)$	20			
<b>7</b>	<b>Data Structures</b>	<b>21</b>			
7.1	RMQ MIN-MAX + LAZY PROPAGATION	21			
7.2	ARVORE BINARIA	21			
7.3	SQRT DECOMPOSITION	22			
7.4	MO'S ALGORITHM (MOST FREQUENT VALUE IN INTERVALS) - $O((m+n)*\sqrt{n})$	22			
7.5	MERGE SORT TREE (K-ESIMO MAIOR ELEMENTO NUM INTERVALO, VALORES MAIORES QUE K NUM INTERVALO, ...)	23			
7.6	ORDENAÇÃO DE ESTRUTURAS(PQ,ETC)	23			
7.7	POLICY BASED DATA STRUCTURES - ORDERED SET	23			
7.8	CONTANDO INVERSÕES	24			
<b>8</b>	<b>GRAPHS</b>	<b>24</b>			
8.1	CICLO GRAFO - $O(V+E)$	24			
8.2	CHECA GRAFO BIPARTIDO - $O(V+E)$	24			
8.3	BELLMAN FORD (MENOR CAMINHO ARESTAS NEGATIVAS) - $O(V*E)$	25			
8.4	DIAMETRO EM ARVORE (MAIOR CAMINHO ENTRE DOIS VERTICES)	25			
8.5	PONTES NUM GRAFO - $O(V+E)$	25			
8.6	PONTOS DE ARTICULAÇÃO NUM GRAFO (se retirar esses vértices o grafo fica desconexo) - $O(V+E)$	26			
8.7	LCA (shortest path)	26			
8.8	FORD FULKERSSON (MAXIMUM FLOW) – $O(V*(E^2))$	27			
8.9	DINIC (MAXIMUM FLOW) - $O(E+(V^2))$ or $O(n * m)$ for Matching	27			
8.10	CAMINHO EULERIANO (Caminho para visitar todas as arestas) – $O(V+E)$	29			
8.11	DIJKSTRA COM PRIORITY QUEUE - $O(E*(\log V))$	29			
8.12	ORDENAÇÃO TOPOLOGICA (FILA) - $O(V+E)$	30			
8.13	KRUSKAL (UNION FIND – $O(\log(n))$ ) – $O(E*\log(V))$	30			
8.14	FLOYD WARSHALL – $O(V^3)$	30			
8.15	SCC (Kosaraju) – $O(V+E)$	30			
<b>9</b>	<b>Diverse</b>	<b>31</b>			
9.1	INTERVAL SCHEDULING	31			
9.2	PROBLEMA DAS OITO RAINHAS	31			
9.3	3SUM PROBLEM ( $a[x]+a[y]+a[z] = \text{valor}$ ) - $O(n^2)$	32			

## 1. Template

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define INF 1ll << 62
6 #define pb push_back
7 #define ii pair<int,int>
8 #define OK cerr <<"OK"<< endl
9 #define debug(x) cerr << #x " = " << (x) << endl
10 #define ff first
11 #define ss second
12 #define int long long
13
14 signed main () {
15
16     ios_base::sync_with_stdio(false);
17     cin.tie(NULL);
18
19 }
```

## 2. Funcoes Interessantes

### 2.1. GCD

```

1 int _gcd(int a, int b){
2     if(a == 0 || b == 0) return 0;
3     else return abs(__gcd(a,b));
4 }
```

### 2.2. HIPOTENUSA

```

1 cout << hypot(3,4); // output: 5
```

### 2.3. SCANF DE UMA STRING

```

1 char sentence[]="Rudolph is 12 years old";
2 char str [20]; int i;
3 sscanf (sentence,"%s %s %d",str,&i);
4 printf ("%s -> %d\n",str,i);
5 // Output: Rudolph -> 12
```

### 2.4. PRINTF PARA UMA STRING

```

1 char buffer [50];
2 int n, a=5, b=3;
3 n=sprintf (buffer, "%d plus %d is %d", a, b, a+b);
4 printf ("%s] is a string %d chars long\n",buffer,n);
5 // Output:
6 // [5 plus 3 is 8] is a string 13 chars long
```

### 2.5. CLIMITS

```

1 LONG_MIN -> (-2^31+1) :: LONG_MAX -> (2^31-1)
2 ULONG_MAX -> (2^32-1) -> UNSIGNED
3 LLONG_MIN, LLONG_MAX, ULLONG_MAX
```

### 2.6. ROTATE (LEFT)

```

1 Passado o inicio o meio e o fim ele rotaciona de forma que o meio seja o
  novo inicio.
2 vector<int> arr(n); // 1 2 3 4 5 6 7 8 9
3 rotate(arr.begin(),arr.begin()+3,arr.end()); //4 5 6 7 8 9 1 2 3
```

### 2.7. ROTATE (RIGHT)

```

1 vector<int> arr(n); // 1 2 3 4 5 6 7 8 9
2 rotate(arr.begin(),arr.rbegin()+3,arr.rend()); //7 8 9 1 2 3 4 5 6
```

### 2.8. WIDTH

```

1 cout << width(13);
2 cout << 100 << endl; // "      100      "
3 cout.fill('x');
4 cout.width(13);
5 cout << 100 << endl; // "xxxxx100xxxxx"
6 cout << right << 100 << endl; "xxxxxxx100"
```

### 2.9. INT TO STRING (C++11)

```

1 int a; string b;
2 b = to_string(a);
```

### 2.10. PERMUTAÇÃO

```

1 int v[] = {1,2,3};
2 sort(v, v+3);
3 do {
4     cout << v[0] << ' ' << v[1] << ' ' << v[2];
5 } while(next_permutation(v, v+3));
```

### 2.11. MAIOR E MENOR ELEMENTO NUM VETOR

```

1 int maior = *max_element(arr.begin(), arr.end());
2 int menor = *min_element(arr.begin(), arr.end());
3 // OBS: Retorna iterador
```

### 2.12. CHECAGEM E TRANSFORMAÇÃO DE CARACTERE

```

1 #include <cctype>
2 isdigit(str[i]); //checa se str[i] é número
3 isalpha(str[i]); //checa se é uma letra
4 islower(str[i]); //checa minúsculo
5 isupper(str[i]); //checa maiúsculo
6 isalnum(str[i]); //checa letra ou número
7 tolower(str[i]); //converte para minusculo
8 toupper(str[i]); //converte para maiusculo
```

### 2.13. SUBSTRING

```

1 string s = "abcdef";
2 s.substr(posição inicial, qtd de char(opcional));
3 string s2 = s.substr(3,2); // s2 = "de"
4 string s3 = s.substr(2); // s3 = "cdef"
```

## 2.14. REMOVE REPETIÇÕES CONTÍNUAS NUM VETOR

```

1 // arr = {10,20,20,20,30,20,20,10}
2 it = unique(arr.begin(), arr.end());
3 // arr = {10,20,30,20,10, iterator aponta pra aqui, ...}
4 arr.resize(distance(arr.begin(), it));
5 // arr = {10,20,30,20,10}

```

## 2.15. CHECAGEM DE BITS

```

1 // OBS: SO FUNCIONA PARA INT (NAO FUNCIONA COM LONG LONG)
2 __builtin_popcount(int) -> Número de bits ativos;
3 __builtin_ctz(int) -> Número de zeros à direita
4 __builtin_clz(int) -> Número de zeros à esquerda
5 __builtin_parity(int) -> Retorna se a quantidade de uns é ímpar(1) ou par(0)

```

## 2.16. INT TO BINARY STRING

```

1 string s = bitset<qtdDeBits>(intVar).to_string();
2 Ex: x = 10, qtdDeBits = 32;
3 s = bitset<32>(x).to_string(); // s = 00...0001010

```

## 2.17. BINARY STRING TO INT

```

1 int y = bitset<qtdDeBits>(stringVar).to_ulong();
2 Ex: x = 1010, qtdDeBits = 32;
3 y = bitset<32>(x).to_ulong(); // y = 10

```

## 2.18. STRING TO LONG LONG

```

1 string s = "0xFFFF"; int base = 16;
2 string::size_type sz = 0;
3 int ll = stoll(s,&sz,base); // ll = 65535, sz = 6;
4 OBS: Não precisa colocar o sz, pode colocar 0; // stoll(s,0,base);

```

## 2.19. SPLIT FUNCTION

```

1 // SEPARA STRING POR UM DELIMITADOR
2 // EX: str=A-B-C split -> x = {A,B,C}
3 vector<string> split(const string &s, char delim) {
4     stringstream ss(s);
5     string item;
6     vector<string> tokens;
7     while (getline(ss, item, delim)) {
8         tokens.push_back(item);
9     }
10    return tokens;
11 }
12 int main () {
13     vector<string> x = split("cap-one-best-opinion-language", '-');
14     // x = {cap,one,best,opinion,language};
15 }

```

## 2.20. LEITURA NUM ARQUIVO

```

1 ifstream cin("input.txt");

```

## 2.21. ESCRITA EM ARQUIVO

```

1 ofstream cout("output.txt");

```

## 2.22. CHECAGEM BRUTE FORCE COM SOLUCAO

```

1 $ g++ -std=c++11 gen.cpp && ./a.out > gen.out && g++ -std=c++11 brute.cpp &&
   (. /a.out < gen.in) > brute.out && g++ -std=c++11 sol.cpp && (. /a.out <
   gen.in) > sol.out && diff brute.out sol.out

```

### 3. DP

#### 3.1. PROBLEMA DO TROCO

```

1 // função que recebe o valor de troco N, o número de moedas disponíveis M,
2 // e um vetor com as moedas disponíveis arr
3 // essa função deve retornar o número mínimo de moedas,
4 // de acordo com a solução com Programação Dinâmica.
5 int num_moedas(int N, int M, int arr[]) {
6     int dp[N+1];
7     // caso base
8     dp[0] = 0;
9     // sub-problemas
10    for(int i=1; i<=N; i++) {
11        // é comum atribuir um valor alto, que concerteza
12        // é maior que qualquer uma das próximas possibilidades,
13        // sendo assim substituído
14        dp[i] = 1000000;
15        for(int j=0; j<M; j++) {
16            if(i-arr[j] >= 0) {
17                dp[i] = min(dp[i], dp[i-arr[j]]+1);
18            }
19        }
20    }
21    // solução
22    return dp[N];
23 }

```

#### 3.2. PROBLEMA DA MOCHILA

```

1 int dp[2001][2001];
2 int moc(int q,int p,vector<ii> vec) {
3     for(int i = 1; i <= q; i++)
4     {
5         for(int j = 1; j <= p; j++) {
6             if(j >= vec[i-1].ff)
7                 dp[i][j] = max(dp[i-1][j],vec[i-1].ss + dp[i-1][j-vec[i-1].ff]);
8             else
9                 dp[i][j] = dp[i-1][j];
10        }
11    }
12    return dp[q][p];
13 }
14 int main(int argc, char *argv[])
15 {
16     int p,q;
17     vector<ii> vec;
18     cin >> p >> q;
19     int x,y;
20     for(int i = 0; i < q; i++) {
21         cin >> x >> y;
22         vec.push_back(make_pair(x,y));
23     }
24     for(int i = 0; i <= p; i++)
25         dp[0][i] = 0;
26     for(int i = 1; i <= q; i++)
27         dp[i][0] = 0;
28     sort(vec.begin(),vec.end());
29     cout << moc(q,p,vec) << endl;
30 }

```

#### 3.3. CATALAN (1, 1, 2, 5, 14, 42, 132, 429) - $O(n)$

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^n \frac{n+k}{k} \quad \text{para } n \geq 0.$$

```

1 // The first few Catalan numbers for n = 0, 1, 2, 3, ...
2 // are 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, ...
3 // Formula Recursiva:
4 // cat(0) = 0
5 // cat(n+1) = somatorio(i from 0 to n) (cat(i)*cat(n-i))
6 //
7 // Using Binomial Coefficient
8 // We can also use the below formula to find nth catalan number in O(n) time.
9 // Formula acima
10
11 // Returns value of Binomial Coefficient C(n, k)
12
13 int binomialCoeff(int n, int k) {
14     int res = 1;
15
16     // Since C(n, k) = C(n, n-k)
17     if (k > n - k)
18         k = n - k;
19
20     // Calculate value of [n*(n-1)*---*(n-k+1)] / [k*(k-1)*---*1]
21     for (int i = 0; i < k; ++i) {
22         res *= (n - i);
23         res /= (i + 1);
24     }
25
26     return res;
27 }
28
29 // A Binomial coefficient based function to find nth catalan
30 // number in O(n) time
31 int catalan(int n) {
32     // Calculate value of 2nCn
33     int c = binomialCoeff(2*n, n);
34
35     // return 2nCn/(n+1)
36     return c/(n+1);
37 }

```

#### 3.4. LONGEST COMMON SUBSEQUENCE - $O(n^2)$

```

1 void lcs(char *X, char *Y, int m, int n){
2     int L[m+1][n+1];
3     /* Following steps build L[m+1][n+1] in bottom up fashion. Note that
4        L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] */
5     for (int i=0; i<=m; i++){
6         for (int j=0; j<=n; j++){
7             if (i == 0 || j == 0)
8                 L[i][j] = 0;
9             else if (X[i-1] == Y[j-1])
10                L[i][j] = L[i-1][j-1] + 1;
11            else
12                L[i][j]=max(L[i-1][j],L[i][j-1]);
13        }
14    }
15    // Following code is used to print LCS
16    int index = L[m][n]; // L[m][n] contem o numero de caracteres

```

```

17 //Create char array to save the lcs string
18 char lcs[index+1];
19 lcs[index] = '\0'; // Set the terminating character
20 // Start from the right-most-bottom-most corner and
21 // one by one store characters in lcs[]
22 int i = m, j = n;
23 while (i > 0 && j > 0) {
24     // If current character in X[] and Y are same, then
25     // current character is part of LCS
26     if (X[i-1] == Y[j-1]){
27         lcs[index-1] = X[i-1]; // Put current character in result
28         i--; j--; index--; // reduce values of i, j and index
29     }
30 // If not same then find the larger of two and go in the direction of larger
    value
    else if (L[i-1][j] > L[i][j-1])
        i--;
    else
        j--;
35 }
36 // Print the lcs
37 cout<<"LCSof"<<X<<"and"<<Y<<" is " <<lcs;
38 }
39 int main()
40 {
41     char X[] = "AGGTAB";
42     char Y[] = "GXTXAYB";
43
44     int m = strlen(X);
45     int n = strlen(Y);
46     lcs(X, Y, m, n);
47     return 0;
48 }
49

```

### 3.5. LONGEST COMMON SUBSTRING - $O(n^2)$

```

1 int LCSuff(char *X, char *Y, int m, int n) {
2     // Create a table to store lengths of longest common suffixes of
3     // substrings. Notethat LCSuff[i][j] contains length of longest
4     // common suffix of X[0..i-1] and Y[0..j-1]. The first row and
5     // first column entries have no logical meaning, they are used only
6     // for simplicity of program
7     int LCSuff[m+1][n+1];
8     int result = 0; // To store length of the longest common substring
9
10    /* Following steps build LCSuff[m+1][n+1] in bottom up fashion. */
11    for (int i=0; i<=m; i++) {
12        for (int j=0; j<=n; j++) {
13            if (i == 0 || j == 0)
14                LCSuff[i][j] = 0;
15
16            else if (X[i-1] == Y[j-1]) {
17                LCSuff[i][j] = LCSuff[i-1][j-1] + 1;
18                result = max(result, LCSuff[i][j]);
19            }
20            else LCSuff[i][j] = 0;
21        }
22    }
23    return result;
24 }

```

### 3.6. LONGEST INCREASING SUBSEQUENCE - $O(n \log(n))$

```

1 int lis(vector<int> &arr) {
2     int n = arr.size();
3     vector<int> lis;
4     for(int i = 0; i < n; i++){
5         int l = 0, r = (int)lis.size() - 1;
6         int ansj = -1;
7         while(l <= r){
8             int mid = (l+r)/2;
9             // OBS: PARA >= TROCAR SINAL EMBAIXO POR <=
10            if(arr[i] < lis[mid]){
11                r = mid - 1;
12                ansj = mid;
13            }
14            else l = mid + 1;
15        }
16        if(ansj == -1){
17            // se arr[i] e maior que todos
18            lis.push_back(arr[i]);
19        }
20        else {
21            lis[ansj] = arr[i];
22        }
23    }
24
25    return lis.size();
26 }

```

### 3.7. LONGEST INCREASING SUBSEQUENCE 2D (NOT SORTED) - $O(n \log(n))$

```

1 set<ii> s[(int)2e6];
2 bool check(ii par, int ind) {
3
4     auto it = s[ind].lower_bound(ii(par.ff, -INF));
5     if(it == s[ind].begin())
6         return false;
7
8     it--;
9
10    if(it->ss < par.ss)
11        return true;
12    return false;
13 }
14
15 int lis2d(vector<ii> &arr) {
16
17     int n = arr.size();
18     s[1].insert(arr[0]);
19
20     int maior = 1;
21     for(int i = 1; i < n; i++) {
22
23         ii x = arr[i];
24
25         int l = 1, r = maior;
26         int ansbb = 0;
27         while(l <= r) {
28             int mid = (l+r)/2;
29             if(check(x, mid)) {
30                 l = mid + 1;
31                 ansbb = mid;
32             } else {
33                 r = mid - 1;

```

```

34     }
35 }
36
37 // inserting in list
38 auto it = s[ansbb+1].lower_bound(ii(x.ff, -INF));
39 while(it != s[ansbb+1].end() && it->ss >= x.ss)
40     it = s[ansbb+1].erase(it);
41
42 it = s[ansbb+1].lower_bound(ii(x.ff, -INF));
43 if(s[ansbb+1].size() > 0 && it != s[ansbb+1].end() && it->ff == x.ff &&
44    it->ss <= x.ss)
45     continue;
46 s[ansbb+1].insert(arr[i]);
47
48 maior = max(maior, ansbb + 1);
49 }
50 return maior;
51
52 }

```

### 3.8. LONGEST INCREASING SUBSEQUENCE 2D (SORTED) - $O(n \log(n))$

```

1 set<ii> s[(int)2e6];
2 bool check(ii par, int ind) {
3
4     auto it = s[ind].lower_bound(ii(par.ff, -INF));
5     if(it == s[ind].begin())
6         return false;
7
8     it--;
9
10    if(it->ss < par.ss)
11        return true;
12    return false;
13 }
14
15 int lis2d(vector<ii> &arr) {
16
17     int n = arr.size();
18     s[1].insert(arr[0]);
19
20     int maior = 1;
21     for(int i = 1; i < n; i++) {
22
23         ii x = arr[i];
24
25         int l = 1, r = maior;
26         int ansbb = 0;
27         while(l <= r) {
28             int mid = (l+r)/2;
29             if(check(x, mid)) {
30                 l = mid + 1;
31                 ansbb = mid;
32             } else {
33                 r = mid - 1;
34             }
35         }
36
37         // inserting in list
38         auto it = s[ansbb+1].lower_bound(ii(x.ff, -INF));
39         while(it != s[ansbb+1].end() && it->ss >= x.ss)
40             it = s[ansbb+1].erase(it);
41

```

```

42     it = s[ansbb+1].lower_bound(ii(x.ff, -INF));
43     if(s[ansbb+1].size() > 0 && it != s[ansbb+1].end() && it->ff == x.ff &&
44        it->ss <= x.ss)
45         continue;
46     s[ansbb+1].insert(arr[i]);
47
48     maior = max(maior, ansbb + 1);
49 }
50 return maior;
51
52 }

```

### 3.9. ACHAR MAIOR PALÍNDROMO

1 Fazer LCS da string com o reverso

### 3.10. SUBSET SUM COM BITSET - $O((maxSum)*n/32)$

```

1 bitset<312345> bit;
2 int arr[112345];
3 void subsetSum(int n) {
4     bit.reset();
5     bit.set(0);
6     for(int i = 0; i < n; i++) {
7         bit |= (bit << arr[i]);
8     }
9 }

```

### 3.11. DIGIT DP

```

1 // How many numbers x are there in the range a to b, where the digit d
2 // occurs exactly k times in x?
3 vector<int> num;
4 int a, b, d, k;
5 int DP[12][12][2];
6 // DP[p][c][f] = Number of valid numbers <= b from this state
7 // p = current position from left side (zero based)
8 // c = number of times we have placed the digit d so far
9 // f = the number we are building has already become smaller than b? [0 =
10 //    no, 1 = yes]
11
12 int call(int pos, int cnt, int f){
13     if(cnt > k) return 0;
14
15     if(pos == num.size()){
16         if(cnt == k) return 1;
17         return 0;
18     }
19
20     if(DP[pos][cnt][f] != -1) return DP[pos][cnt][f];
21     int res = 0;
22     int lim = (f ? 9 : num[pos]);
23
24     // Try to place all the valid digits such that the number doesn't exceed b
25     for(int dgt = 0; dgt <= LMT; dgt++){
26         int nf = f;
27         int ncnt = cnt;
28         if(f == 0 && dgt < LMT) nf = 1; // The number is getting smaller at
29         this position
30         if(dgt == d) ncnt++;
31         if(ncnt <= k) res += call(pos+1, ncnt, nf);
32     }
33 }

```

```

29     }
30
31     return DP[pos][cnt][f] = res;
32 }
33
34 int solve(int b){
35     num.clear();
36     while(b>0){
37         num.push_back(b%10);
38         b/=10;
39     }
40     reverse(num.begin(), num.end());
41     /// Stored all the digits of b in num for simplicity
42
43     memset(DP, -1, sizeof(DP));
44     int res = call(0, 0, 0);
45     return res;
46 }
47
48 int main () {
49
50     cin >> a >> b >> d >> k;
51     int res = solve(b) - solve(a-1);
52     cout << res << endl;
53
54     return 0;
55 }

```

## 4. Math

### 4.1. INCLUSÃO-EXCLUSÃO

$$\left| \bigcup_{i=1}^n A_i \right| = \sum_{k=1}^n (-1)^{k+1} \left( \sum_{1 \leq i_1 < \dots < i_k \leq n} |A_{i_1} \cap \dots \cap A_{i_k}| \right)$$

```

1 // |A ∪ B ∪ C| = |A| + |B| + |C| - |A ∩ B| - |A ∩ C| - |B ∩ C| + |A ∩ B ∩ C|
2
3 // EXEMPLO: Quantos números de 1 a 10^9 são múltiplos de 42, 54, 137 ou 201?
4
5 int f(vector<int> arr, int LIMIT) {
6
7     int n = arr.size();
8     int c = 0;
9
10    for(int mask = 1; mask < (1<<n); mask++) {
11        int lcm = 1;
12        for(int i = 0; i < n; i++)
13            if(mask & (1<<i))
14                lcm = lcm * arr[i] / __gcd(lcm, arr[i]);
15        // se o numero de conjuntos a unir for impar entao soma
16        if(__builtin_popcount(mask) % 2 == 1)
17            c += LIMIT / lcm;
18        else // se nao subtrai
19            c -= LIMIT / lcm;
20    }
21
22    return LIMIT - c;
23
24 }

```

### 4.2. TODOS OS DIVISORES DE N - $O(\sqrt{n})$

```

1 // OBS: ATÉ  $\sqrt[3]{N}$  DIVISORES DE N
2 vector<int> divisors(int n) {
3     int sq = sqrt(n);
4     vector<int> ans;
5     for (int i=1; i<=sq+1; i++) {
6         if (n%i==0) {
7             // If divisors are equal, print only one
8             if (n/i == i)
9                 ans.pb(i);
10
11            else // Otherwise print both
12                ans.pb(i), ans.pb(n/i);
13        }
14    }
15    return ans;
16 }

```

### 4.3. ALGORITMO DE EUCLIDES ESTENDIDO - $O(\log(\min(a,b)))$

```

1 int gcd,x,y;
2
3 //Ax + By = gcd(A,B)
4

```



```

5 void extendedEuclidian(int a,int b){
6
7     if(b==0){
8         gcd=a;
9         x=1;
10        y=0;
11    } else{
12        extendedEuclidian(b, a%b);
13
14        int temp = x;
15        x=y;
16        y = temp - (a/b)*y;
17    }
18 }

```

#### 4.4. TEOREMA CHINES DO RESTO

```

1  /*
2   Ex:  $x \equiv 2 \pmod{3}$ 
3        $x \equiv 3 \pmod{5}$ 
4        $x \equiv 2 \pmod{7}$ 
5  */
6
7  int inv(int a, int m) {
8      int m0 = m, t, q;
9      int x0 = 0, x1 = 1;
10
11     if (m == 1)
12         return 0;
13
14     // Apply extended Euclid Algorithm
15     while (a > 1) {
16         // q is quotient
17         q = a / m;
18         t = m;
19         // m is remainder now, process same as euclid's algo
20         m = a % m, a = t;
21         t = x0;
22         x0 = x1 - q * x0;
23         x1 = t;
24     }
25
26     // Make x1 positive
27     if (x1 < 0)
28         x1 += m0;
29
30     return x1;
31 }
32
33 // k is size of num[] and rem[]. Returns the smallest
34 // number x such that:
35 // x % num[0] = rem[0],
36 // x % num[1] = rem[1],
37 // .....
38 // x % num[k-2] = rem[k-1]
39 // Assumption: Numbers in num[] are pairwise coprime
40 // (gcd for every pair is 1)
41 int findMinX(int num[], int rem[], int k){
42     // Compute product of all numbers
43     int prod = 1;
44     for (int i = 0; i < k; i++)
45         prod *= num[i];
46
47     // Initialize result

```

```

48     int result = 0;
49
50     // Apply above formula
51     for (int i = 0; i < k; i++){
52         int pp = prod / num[i];
53         result+=rem[i]*inv(pp,num[i])*pp;
54     }
55
56     return result % prod;
57 }

```

#### 4.5. BELL NUMBERS (NUMBER OF WAYS TO PARTITION A SET) - $O(n^2)$

```

1  int bellNumber(int n) {
2      int bell[n+1][n+1];
3      bell[0][0] = 1;
4      for (int i=1; i<=n; i++) {
5          // Explicitly fill for j = 0
6          bell[i][0] = bell[i-1][i-1];
7
8          // Fill for remaining values of j
9          for (int j=1; j<=i; j++)
10             bell[i][j] = bell[i-1][j-1] + bell[i][j-1];
11     }
12     return bell[n][0];
13 }

```

#### 4.6. FATORAÇÃO SIMPLES - $O(\sqrt{n})$

```

1  set<int> primeFactors(int n) {
2      set<int> ret;
3      while (n%2 == 0) {
4          ret.insert(2);
5          n = n/2;
6      }
7
8      int sq = sqrt(n);
9      for (int i = 3; i <= sq+2; i = i+2) {
10         while (n%i == 0) {
11             ret.insert(i);
12             n = n/i;
13         }
14         /* OBS1
15         IF (N < 1E7)
16             FATORE COM SPF
17         */
18     }
19
20     if (n > 2)
21         ret.insert(n);
22
23     return ret;
24 }

```

#### 4.7. FATORAÇÃO PRA MULTIPLAS QUERIES - $O(n(\log(\log(n))))$

```

1  //stor smallest prime factor for every num
2  int spf[MAXN];
3  // Calculating SPF (Smallest Prime Factor) for every number till MAXN.
4  // Time Complexity : O(nloglogn)
5  void sieve() {
6      spf[1] = 1;

```

```

7   for (int i=2; i<MAXN; i++)
8       // marking smallest prime factor for every number to be itself.
9       spf[i] = i;
10
11  // separatelyMarking spf for every even
12  // number as 2
13  for (int i=4; i<MAXN; i+=2)
14      spf[i] = 2;
15
16  for (int i=3; i<MAXN; i++) {
17      // checking if i is prime
18      if (spf[i] == i) {
19          // marking SPF for all numbers divisible by i
20          for (int j=i*i; j<MAXN; j+=i)
21              // marking spf[j] if it is not previously marked
22              if (spf[j]==j)
23                  spf[j] = i;
24      }
25  }
26
27  // A O(log n) function returning primefactorization
28  // by dividing by smallest prime factor at every step
29  vector<int> getFactorization(int x) {
30      vector<int> ret;
31      while (x != 1) {
32          ret.push_back(spf[x]);
33          x = x / spf[x];
34      }
35      return ret;
36  }

```

#### 4.8. CRIVO COMUM E SEGMENTADO COM BITSET

```

1  bitset<(int)1e5+3> primesInSeg;
2  bitset<(int)1e6+3> isPrime;
3  int spf[(int)1e6+3];
4  vector<int> primes;
5  vector<int> segPrimes;
6
7  void sieve(int n = (int)1e6) {
8
9      isPrime.set();
10     for (int i = 2; i*i <= n; i++) {
11         if (!isPrime[i])
12             continue;
13
14         for (int j = i*i; j <= n; j+=i) {
15             isPrime[j] = false;
16             spf[j] = min(i, spf[j]);
17         }
18         primes.pb(i);
19     }
20 }
21
22 vector<int> getFactorization(int x) {
23     vector<int> ret;
24     while (x != 1) {
25         ret.push_back(spf[x]);
26         x = x / spf[x];
27     }
28     return ret;
29 }
30
31 void segSieve(int l, int r) {

```

```

32 // primos de l..r
33 // trasladados de 0..(l-r)
34 segPrimes.clear();
35 primesInSeg.set();
36 int sq = sqrt(r);
37
38 for (int p: primes) {
39     if (p > sq)
40         break;
41
42     for (int i = l - l%p; i <= r; i += p) {
43         if (i - l < 0)
44             continue;
45
46         // se i for menor q 1e6 checa na array do crivo
47         if (i >= (int)1e6 || !isPrime[i])
48             primesInSeg[i-l] = false;
49     }
50 }
51
52 for (int i = 0; i < r-l+1; i++) {
53     if (primesInSeg[i])
54         segPrimes.pb(i+l);
55 }
56 }

```

#### 4.9. CHECAR SE UM NUMERO É PRIMO - $O(\sqrt{n})$

```

1 bool isPrime(int n) {
2     if (n <= 1) return false;
3     if (n <= 3) return true;
4     // This is checked so that we can skip
5     // middle five numbers in below loop
6     if (n%2 == 0 || n%3 == 0)
7         return false;
8     for (int i=5; i*i<=n; i += 6)
9         if (n%i == 0 || n%(i+2) == 0)
10             return false;
11     return true;
12 }

```

#### 4.10. EXPONENCIAÇÃO BINÁRIA - $O(\log(n))$

```

1 int power(int x, int p, int MOD) {
2     if (p == 0)
3         return 1%MOD;
4     if (p == 1)
5         return x%MOD;
6     int res = power(x, p/2, MOD);
7     res = (long long)res*res%MOD;
8     if (p&1)
9         res = (long long)res*x%MOD;
10    return res;
11 }

```

#### 4.11. PRECOMPUTAR COMBINAÇÃO $nCr$ - $O(n^2)$

```

1 int C[1123][1123];
2
3 int mod(int n) {return n%((int)1e9+7);}
4
5 int nCr(int n, int k) {

```

```

6   for(int i = 0; i <= n; i++) {
7       for(int j = 0; j <= min(i,k); j++) {
8           if(j == 0 || j == i) {
9               C[i][j] = 1;
10          } else {
11              C[i][j]=mod(C[i-1][j-1]+C[i-1][j]);
12          }
13      }
14  }
15 }

```

#### 4.12. COMBINAÇÃO $nCr$ - $O(n)$

```

1  // Returns value of Binomial Coefficient C(n, k)
2  int binomialCoeff(int n, int k) {
3      int res = 1;
4      // Since C(n, k) = C(n, n-k)
5      if (k > n - k)
6          k = n - k;
7      // Calculate value of [n*(n-1)*---*(n-k+1)] / [k*(k-1)*---*1]
8      for (int i = 0; i < k; ++i) {
9          res *= (n - i);
10         res /= (i + 1);
11     }
12     return res;
13 }

```

#### 4.13. POLLARD RHO (FIND A DIVISOR FOR N) - $O(n^{1/4})$

```

1  /* Function to calculate (base^exponent)%modulus */
2  int modular_pow(int base, int exponent,
3                  int modulus) {
4      /* initialize result */
5      int result = 1;
6      while (exponent > 0) {
7          /* if y is odd, multiply base with result */
8          if (exponent & 1)
9              result = (result * base) % modulus;
10         /* exponent = exponent/2 */
11         exponent = exponent >> 1;
12         /* base = base * base */
13         base = (base * base) % modulus;
14     }
15     return result;
16 }
17
18 /* method to return prime divisor for n */
19 int PollardRho(int n) {
20     /* initialize random seed */
21     srand (time(NULL));
22
23     /* no prime divisor for 1 */
24     if (n==1) return n;
25
26     /* even number means one of the divisors is 2 */
27     if (n % 2 == 0) return 2;
28
29     /* we will pick from the range [2, N) */
30     int x = (rand()%(n-2))+2;
31     int y = x;
32
33     /* the constant in f(x).
34     * Algorithm can be re-run with a different c

```

```

35     * if it throws failure for a composite. */
36     int c = (rand()%(n-1))+1;
37
38     /* Initialize candidate divisor (or result) */
39     int d = 1;
40
41     /* until the prime factor isn't obtained.
42     If n is prime, return n */
43     while (d==1) {
44         /* Tortoise Move: x(i+1) = f(x(i)) */
45         x = (modular_pow(x, 2, n) + c + n)%n;
46
47         /* Hare Move: y(i+1) = f(f(y(i))) */
48         y = (modular_pow(y, 2, n) + c + n)%n;
49         y = (modular_pow(y, 2, n) + c + n)%n;
50
51         /* check gcd of |x-y| and n */
52         d = __gcd(abs(x-y), n);
53
54         /* retry if the algorithm fails to find prime factor
55         * with chosen x and c */
56         if (d==n) return PollardRho(n);
57     }
58     return d;
59 }
60
61 /* driver function */
62 signed main() {
63     int n = 12;
64     printf("One of the divisors for %lld is %lld.",
65           n, PollardRho(n));
66     return 0;
67 }

```

#### 4.14. EQUAÇÃO DIOFANTINA (ACHAR UMA SOLUÇÃO)

```

1  int gcd(int a, int b, int &x, int &y) {
2      if (a == 0) {
3          x = 0; y = 1;
4          return b;
5      }
6      int x1, y1;
7      int d = gcd(b%a, a, x1, y1);
8      x = y1 - (b / a) * x1;
9      y = x1;
10     return d;
11 }
12
13 bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
14     g = gcd(abs(a), abs(b), x0, y0);
15     if (c % g)
16         return false;
17
18     x0 *= c / g;
19     y0 *= c / g;
20     if (a < 0) x0 = -x0;
21     if (b < 0) y0 = -y0;
22     return true;
23 }

```

#### 4.15. EULER'S TOTIENT FUNCTION

```

1 int phi(int n) {
2     int result = n;
3     for (int i = 2; i * i <= n; i++) {
4         if (n % i == 0) {
5             while (n % i == 0)
6                 n /= i;
7             result -= result / i;
8         }
9     }
10
11     if (n > 1)
12         result -= result / n;
13     return result;
14 }

```

## 5. Geometry

### 5.1. PRODUTO VETORIAL

```

1 // Outra forma de produto vetorial
2 // reta ab,ac se for zero e colinear
3 // se for < 0 entao antiHorario, > 0 horario
4 bool ehcol(pto a,pto b,pto c) {
5     return ((b.y-a.y)*(c.x-a.x) - (b.x-a.x)*(c.y-a.y));
6 }
7
8 //-----
9 //Produto vetorial AB x AC, se for zero e colinear
10 int cross(pto A, pto B, pto C){
11     pto AB, AC;
12     AB.x = B.x-A.x;
13     AB.y = B.y-A.y;
14     AC.x = C.x-A.x;
15     AC.y = C.y-A.y;
16     int cross = AB.x*AC.y-AB.y * AC.x;
17     return cross;
18 }
19 // OBS: DEFINE ÁREA DE QUADRILÁTERO FORMADO PELAS RETAS, A ÁREA DO TRIÂNGULO
    É A METADE

```

### 5.2. DISTANCIA PONTO RETA

```

1 double ptoReta(double x1, double y1, double x2,double y2,double pointX,
2     double pointY, double *ptox,double *ptoy){
3     double diffX = x2 - x1;
4     float diffY = y2 - y1;
5     if ((diffX == 0) && (diffY == 0)) {
6         diffX = pointX - x1;
7         diffY = pointY - y1;
8         //se os dois sao pontos
9         return hypot(pointX - x1,pointY - y1);
10    }
11    float t = ((pointX - x1) * diffX + (pointY - y1) * diffY) /
12            (diffX * diffX + diffY * diffY);
13    if (t < 0) {
14        //point is nearest to the first point i.e x1 and y1
15        // Ex:
16        // cord do pto na reta = pto inicial(x1,y1);
17        *ptox = x1, *ptoy = y1;
18        diffX = pointX - x1;
19        diffY = pointY - y1;
20    } else if (t > 1) {
21        //point is nearest to the end point i.e x2 and y2
22        // Ex :
23        // cord do pto na reta = pto final(x2,y2);
24        *ptox = x2, *ptoy = y2;
25        diffX = pointX - x2;
26        diffY = pointY - y2;
27    } else {
28        //if perpendicular line intersect the line segment.
29        // pto nao esta mais proximo de uma das bordas do segmento
30        // Ex:
31        // |
32        // | (Ângulo Reto)
33        // |
34        // cord x do pto na reta = (x1 + t * diffX)
35        // cord y do pto na reta = (y1 + t * diffY)
36        *ptox = (x1 + t * diffX), *ptoy = (y1 + t * diffY);
37        diffX = pointX - (x1 + t * diffX);

```

```

37     diffY = pointY - (y1 + t * diffY);
38 }
39 //returning shortest distance
40 return sqrt(diffX * diffX + diffY * diffY);
41 }

```

### 5.3. ÁREA DE POLÍGONOS

```

1
2 double polygonArea(vector<pto> &arr, int n) {
3     int area = 0;
4     // N = quantidade de pontos no polígono e armazenados em p;
5     // OBS: VALE PARA CONVEXO E NÃO CONVEXO
6     for(int i = 0; i<n; i++){
7         area += cross(arr[i], arr[(i+1)%n]);
8     }
9     return (double)abs(area/2.0);
10 }

```

### 5.4. PONTOS DENTRO DE UM POLIGONO

```

1
2 /* Traça-se uma reta do ponto até um outro ponto qualquer fora do triangulo
   e checka o número de interseção com a borda do polígono se este for impar
   então está dentro se não está fora */
3
4 // Define Infinite (Using INT_MAX caused overflow problems)
5 #define INF 10000
6
7 struct pto {
8     int x, y;
9     pto() {}
10    pto(int x, int y) : x(x), y(y) {}
11 };
12
13 // Given three colinear ptos p, q, r, the function checks if
14 // pto q lies on line segment 'pr'
15 bool onSegment(pto p, pto q, pto r) {
16     if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
17         q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
18         return true;
19     return false;
20 }
21
22 // To find orientation of ordered triplet (p, q, r).
23 // The function returns following values
24 // 0 --> p, q and r are colinear
25 // 1 --> Clockwise
26 // 2 --> Counterclockwise
27 int orientation(pto p, pto q, pto r) {
28     int val = (q.y - p.y) * (r.x - q.x) -
29             (q.x - p.x) * (r.y - q.y);
30
31     if (val == 0) return 0; // colinear
32     return (val > 0)? 1: 2; // clock or counterclock wise
33 }
34
35 // The function that returns true if line segment 'p1q1'
36 // and 'p2q2' intersect.
37 bool doIntersect(pto p1, pto q1, pto p2, pto q2) {
38     // Find the four orientations needed for general and
39     // special cases
40     int o1 = orientation(p1, q1, p2);

```

```

41     int o2 = orientation(p1, q1, q2);
42     int o3 = orientation(p2, q2, p1);
43     int o4 = orientation(p2, q2, q1);
44
45     // General case
46     if (o1 != o2 && o3 != o4)
47         return true;
48
49     // Special Cases
50     // p1, q1 and p2 are colinear and p2 lies on segment p1q1
51     if (o1 == 0 && onSegment(p1, p2, q1)) return true;
52
53     // p1, q1 and p2 are colinear and q2 lies on segment p1q1
54     if (o2 == 0 && onSegment(p1, q2, q1)) return true;
55
56     // p2, q2 and p1 are colinear and p1 lies on segment p2q2
57     if (o3 == 0 && onSegment(p2, p1, q2)) return true;
58
59     // p2, q2 and q1 are colinear and q1 lies on segment p2q2
60     if (o4 == 0 && onSegment(p2, q1, q2)) return true;
61
62     return false; // Doesn't fall in any of the above cases
63 }
64
65 // Returns true if the pto p lies inside the polygon[] with n vertices
66 bool isInside(pto polygon[], int n, pto p) {
67     // There must be at least 3 vertices in polygon[]
68     if (n < 3) return false;
69
70     // Create a pto for line segment from p to infinite
71     pto extreme = pto(INF, p.y);
72
73     // Count intersections of the above line with sides of polygon
74     int count = 0, i = 0;
75     do {
76         int next = (i+1)%n;
77
78         // Check if the line segment from 'p' to 'extreme' intersects
79         // with the line segment from 'polygon[i]' to 'polygon[next]'
80         if (doIntersect(polygon[i], polygon[next], p, extreme)) {
81             // If the pto 'p' is colinear with line segment 'i-next',
82             // then check if it lies on segment. If it lies, return true,
83             // otherwise false
84             if (orientation(polygon[i], p, polygon[next]) == 0)
85                 return onSegment(polygon[i], p, polygon[next]);
86
87             count++;
88         }
89         i = next;
90     } while (i != 0);
91
92     // Return true if count is odd, false otherwise
93     return count%2 == 1; // Same as (count%2 == 1)
94 }

```

### 5.5. PONTOS DENTRO E NA BORDA DE UM POLIGONO

```

1 int cross(pto a, pto b) {
2     return a.x * b.y - b.x * a.y;
3 }
4
5 int boundaryCount(pto a, pto b) {
6     if(a.x == b.x)
7         return abs(a.y-b.y)-1;

```

```

8   if(a.y == b.y)
9       return abs(a.x-b.x)-1;
10  return _gcd(abs(a.x-b.x), abs(a.y-b.y))-1;
11 }
12
13 int totalBoundaryPolygon(vector<pto> &arr, int n) {
14
15     int boundPoint = n;
16     for(int i = 0; i < n; i++) {
17         boundPoint += boundaryCount(arr[i], arr[(i+1)%n]);
18     }
19     return boundPoint;
20 }
21
22 int polygonArea2(vector<pto> &arr, int n) {
23     int area = 0;
24     // N = quantidade de pontos no polígono e armazenados em p;
25     // OBS: VALE PARA CONVEXO E NÃO CONVEXO
26     for(int i = 0; i < n; i++){
27         area += cross(arr[i], arr[(i+1)%n]);
28     }
29     return abs(area);
30 }
31
32 int internalCount(vector<pto> &arr, int n) {
33
34     int area_2 = polygonArea2(arr, n);
35     int boundPoints = totalBoundaryPolygon(arr,n);
36     return (area_2 - boundPoints + 2)/2;
37 }

```

## 5.6. INTERSECÇÃO DE RETAS

```

1  // Intersecção de retas Ax + By = C    dados pontos (x1,y1) e (x2,y2)
2  A = y2-y1
3  B = x1-x2
4  C = A*x1+B*y1
5  //Retas definidas pelas equações:
6  Alx + Bly = C1
7  A2x + B2y = C2
8  //Encontrar x e y resolvendo o sistema
9  double det = A1*B2 - A2*B1;
10 if(det == 0){
11     //Lines are parallel
12 }else{
13     double x = (B2*C1 - B1*C2)/det;
14     double y = (A1*C2 - A2*C1)/det;
15 }

```

## 5.7. INTERSECÇÃO DE SEGMENTOS

```

1  // Given three colinear points p, q, r, the function checks if
2  // point q lies on line segment 'pr'
3  int onSegment(Point p, Point q, Point r) {
4      if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) && q.y <= max(p.y, r.y)
5          && q.y >= min(p.y, r.y))
6          return true;
7      return false;
8  }
9  /* PODE SER RETIRADO
10 int onSegmentNotBorda(Point p, Point q, Point r) {
11     if (q.x < max(p.x, r.x) && q.x > min(p.x, r.x) && q.y <= max(p.y, r.y)
12         && q.y >= min(p.y, r.y))

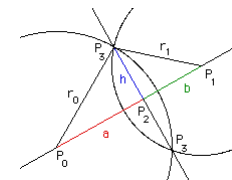
```

```

11     return true;
12     if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) && q.y < max(p.y, r.y)
13         && q.y > min(p.y, r.y))
14         return true;
15     return false;
16 }
17 // To find orientation of ordered triplet (p, q, r).
18 // The function returns following values
19 // 0 --> p, q and r are colinear
20 // 1 --> Clockwise
21 // 2 --> Counterclockwise
22 int orientation(Point p, Point q, Point r) {
23     int val = (q.y - p.y) * (r.x - q.x) -
24             (q.x - p.x) * (r.y - q.y);
25     if (val == 0) return 0; // colinear
26     return (val > 0)? 1: 2; // clock or counterclock wise
27 }
28 // The main function that returns true if line segment 'p1p2'
29 // and 'q1q2' intersect.
30 int doIntersect(Point p1, Point p2, Point q1, Point q2) {
31     // Find the four orientations needed for general and
32     // special cases
33     int o1 = orientation(p1, p2, q1);
34     int o2 = orientation(p1, p2, q2);
35     int o3 = orientation(q1, q2, p1);
36     int o4 = orientation(q1, q2, p2);
37
38     // General case
39     if (o1 != o2 && o3 != o4) return 2;
40
41     /* PODE SER RETIRADO
42     if(o1 == o2 && o2 == o3 && o3 == o4 && o4 == 0) {
43         //INTERCEPTAM EM RETA
44         if(onSegmentNotBorda(p1,q1,p2) || onSegmentNotBorda(p1,q2,p2)) return 1;
45         if(onSegmentNotBorda(q1,p1,q2) || onSegmentNotBorda(q1,p2,q2)) return 1;
46     }
47     */
48     // Special Cases (INTERCEPTAM EM PONTO)
49     // p1, p2 and q1 are colinear and q1 lies on segment p1p2
50     if (o1 == 0 && onSegment(p1, q1, p2)) return 2;
51     // p1, p2 and q1 are colinear and q2 lies on segment p1p2
52     if (o2 == 0 && onSegment(p1, q2, p2)) return 2;
53     // q1, q2 and p1 are colinear and p1 lies on segment q1q2
54     if (o3 == 0 && onSegment(q1, p1, q2)) return 2;
55     // q1, q2 and p2 are colinear and p2 lies on segment q1q2
56     if (o4 == 0 && onSegment(q1, p2, q2)) return 2;
57     return false; // Doesn't fall in any of the above cases
58 }
59 // OBS: SE (C2/A2 == C1/A1) SÃO COLINEARES

```

## 5.8. INTERSECÇÃO DE CIRCULOS (2PTOS)



```

1 void circle_circle_intersection(/*...*/) {
2     double a, d, h, rx, ry, x2, y2;
3     d = hypot(x1 - x0, y1 - y0);
4     /* Determine the distance from point 0 to point 2. */
5     a = ((r0*r0) - (r1*r1) + (d*d)) / (2.0 * d);
6     /* Determine the coordinates of point 2 */
7     x2 = x0 + (dx * a/d); y2 = y0 + (dy * a/d);
8     /* Determine the distance from point 2 to either of the intersection
9        points. */
10    h = sqrt((r0*r0) - (a*a));
11    /* Now determine the offsets of the intersection points from
12       point 2. */
13    rx = -dy * (h/d); ry = dx * (h/d);
14    /* Determine the absolute intersection points. */
15    *xi = x2 + rx;
16    *xi_prime = x2 - rx;
17    *yi = y2 + ry;
18    *yi_prime = y2 - ry;
19 }

```

### 5.9. PONTO DENTRO DE UM POLIGONO

```

1 /* Traça-se uma reta do ponto até um outro ponto qualquer fora do triângulo
2    e checa o número de interseção com a borda do polígono se este for ímpar
3    então está dentro se não está fora */
4 // Define Infinite (Using INT_MAX caused overflow problems)
5 #define INF 10000
6
7 struct pto {
8     int x, y;
9     pto() {}
10    pto(int x, int y) : x(x), y(y) {}
11 };
12
13 // Given three colinear ptos p, q, r, the function checks if
14 // pto q lies on line segment 'pr'
15 bool onSegment(pto p, pto q, pto r) {
16     if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
17         q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
18         return true;
19     return false;
20 }
21
22 // To find orientation of ordered triplet (p, q, r).
23 // The function returns following values
24 // 0 --> p, q and r are colinear
25 // 1 --> Clockwise
26 // 2 --> Counterclockwise
27 int orientation(pto p, pto q, pto r) {
28     int val = (q.y - p.y) * (r.x - q.x) -
29              (q.x - p.x) * (r.y - q.y);
30
31     if (val == 0) return 0; // colinear
32     return (val > 0) ? 1 : 2; // clock or counterclock wise
33 }
34
35 // The function that returns true if line segment 'p1q1'
36 // and 'p2q2' intersect.
37 bool doIntersect(pto p1, pto q1, pto p2, pto q2) {
38     // Find the four orientations needed for general and
39     // special cases

```

```

40     int o1 = orientation(p1, q1, p2);
41     int o2 = orientation(p1, q1, q2);
42     int o3 = orientation(p2, q2, p1);
43     int o4 = orientation(p2, q2, q1);
44
45     // General case
46     if (o1 != o2 && o3 != o4)
47         return true;
48
49     // Special Cases
50     // p1, q1 and p2 are colinear and p2 lies on segment p1q1
51     if (o1 == 0 && onSegment(p1, p2, q1)) return true;
52
53     // p1, q1 and p2 are colinear and q2 lies on segment p1q1
54     if (o2 == 0 && onSegment(p1, q2, q1)) return true;
55
56     // p2, q2 and p1 are colinear and p1 lies on segment p2q2
57     if (o3 == 0 && onSegment(p2, p1, q2)) return true;
58
59     // p2, q2 and q1 are colinear and q1 lies on segment p2q2
60     if (o4 == 0 && onSegment(p2, q1, q2)) return true;
61
62     return false; // Doesn't fall in any of the above cases
63 }
64
65 // Returns true if the pto p lies inside the polygon[] with n vertices
66 bool isInside(pto polygon[], int n, pto p) {
67     // There must be at least 3 vertices in polygon[]
68     if (n < 3) return false;
69
70     // Create a pto for line segment from p to infinite
71     pto extreme = pto(INF, p.y);
72
73     // Count intersections of the above line with sides of polygon
74     int count = 0, i = 0;
75     do {
76         int next = (i+1)%n;
77
78         // Check if the line segment from 'p' to 'extreme' intersects
79         // with the line segment from 'polygon[i]' to 'polygon[next]'
80         if (doIntersect(polygon[i], polygon[next], p, extreme)) {
81             // If the pto 'p' is colinear with line segment 'i-next',
82             // then check if it lies on segment. If it lies, return true,
83             // otherwise false
84             if (orientation(polygon[i], p, polygon[next]) == 0)
85                 return onSegment(polygon[i], p, polygon[next]);
86
87             count++;
88         }
89         i = next;
90     } while (i != 0);
91
92     // Return true if count is odd, false otherwise
93     return count%2 == 1; // Same as (count%2 == 1)
94 }

```

### 5.10. CLOSEST PAIR OF POINTS

```

1 struct Point {
2     int x, y;
3 };
4 int compareX(const void *a, const void *b) {
5     Point *p1 = (Point *)a, *p2 = (Point *)b;
6     return (p1->x - p2->x);

```

```

7 }
8 int compareY(const void *a, const void *b) {
9     Point *p1 = (Point *)a, *p2 = (Point *)b;
10    return (p1->y - p2->y);
11 }
12 float dist(Point p1, Point p2) {
13    return sqrt((p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y));
14 }
15 float bruteForce(Point P[], int n) {
16    float min = FLT_MAX;
17    for (int i = 0; i < n; ++i)
18        for (int j = i+1; j < n; ++j)
19            if (dist(P[i], P[j]) < min)
20                min = dist(P[i], P[j]);
21    return min;
22 }
23 float min(float x, float y) {
24    return (x < y) ? x : y;
25 }
26 float stripClosest(Point strip[], int size, float d) {
27    float min = d;
28    for (int i = 0; i < size; ++i)
29        for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
30            if (dist(strip[i], strip[j]) < min)
31                min = dist(strip[i], strip[j]);
32    return min;
33 }
34 float closestUtil(Point Px[], Point Py[], int n) {
35    if (n <= 3)
36        return bruteForce(Px, n);
37    int mid = n/2;
38    Point midPoint = Px[mid];
39    Point Pyl[mid+1];
40    Point Pyr[n-mid-1];
41    int li = 0, ri = 0;
42    for (int i = 0; i < n; ++i)
43        if (Py[i].x <= midPoint.x)
44            Pyl[li++] = Py[i];
45        else
46            Pyr[ri++] = Py[i];
47
48    float dl = closestUtil(Px, Pyl, mid);
49    float dr = closestUtil(Px + mid, Pyr, n-mid);
50    float d = min(dl, dr);
51    Point strip[n];
52    int j = 0;
53    for (int i = 0; i < n; ++i)
54        if (abs(Py[i].x - midPoint.x) < d)
55            strip[j] = Py[i], j++;
56    return min(d, stripClosest(strip, j, d));
57 }
58
59 float closest(Point P[], int n) {
60    Point Px[n];
61    Point Py[n];
62    for (int i = 0; i < n; ++i) {
63        Px[i] = P[i];
64        Py[i] = P[i];
65    }
66    qsort(Px, n, sizeof(Point), compareX);
67    qsort(Py, n, sizeof(Point), compareY);
68    return closestUtil(Px, Py, n);
69 }

```

## 5.11. CENTRO DE MASSA DE UM POLÍGONO

```

1 double area = 0;
2 pto c;
3
4 c.x = c.y = 0;
5 for(int i = 0; i < n; i++) {
6     double aux = (arr[i].x * arr[i+1].y) - (arr[i].y * arr[i+1].x); // shoelace
7     area += aux;
8     c.x += aux*(arr[i].x + arr[i+1].x);
9     c.y += aux*(arr[i].y + arr[i+1].y);
10 }
11
12 c.x /= (3.0*area);
13 c.y /= (3.0*area);
14
15 cout << c.x << ' ' << c.y << endl;

```

## 5.12. POINT AND LINE STRUCT

```

1 int sgn(double x) {
2     if(abs(x) < 1e-8) return 0;
3     return x > 0 ? 1 : -1;
4 }
5 inline double sqr(double x) { return x * x; }
6
7 struct Point {
8     double x, y, z;
9     Point() {}
10    Point(double a, double b): x(a), y(b) {}
11    Point(double x, double y, double z): x(x), y(y), z(z) {}
12
13    void input() { scanf("%lf %lf", &x, &y); }
14    friend Point operator+(const Point &a, const Point &b) {
15        return Point(a.x + b.x, a.y + b.y);
16    }
17    friend Point operator-(const Point &a, const Point &b) {
18        return Point(a.x - b.x, a.y - b.y);
19    }
20
21    bool operator !=(const Point &a) const {
22        return (x != a.x || y != a.y);
23    }
24
25    bool operator <(const Point &a) const {
26        if(x == a.x)
27            return y < a.y;
28        return x < a.x;
29    }
30
31    double norm() {
32        return sqrt(sqr(x) + sqr(y));
33    }
34 };
35 double det(const Point &a, const Point &b) {
36    return a.x * b.y - a.y * b.x;
37 }
38 double dot(const Point &a, const Point &b) {
39    return a.x * b.x + a.y * b.y;
40 }
41 double dist(const Point &a, const Point &b) {
42    return (a-b).norm();
43 }
44

```



```

45 struct Line {
46     Point a, b;
47     Line() {}
48     Line(Point x, Point y): a(x), b(y) {};
49 };
50
51 double dis_point_segment(const Point p, const Point s, const Point t) {
52     if(sgn(dot(p-s, t-s)) < 0)
53         return (p-s).norm();
54     if(sgn(dot(p-t, s-t)) < 0)
55         return (p-t).norm();
56     return abs(det(s-p, t-p) / dist(s, t));
57 }
58

```

### 5.13. CONVEX HULL - $O(n \log(n))$

```

1 // Asymptotic complexity:  $O(n \log n)$ .
2 struct pto {
3     double x, y;
4     bool operator <(const pto &p) const {
5         return x < p.x || (x == p.x && y < p.y);
6         /* a impressao será em prioridade por mais a esquerda, mais
7            abaixo, e anti-horário pelo cross abaixo */
8     }
9 };
10
11 double cross(const pto &O, const pto &A, const pto &B) {
12     return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
13 }
14
15 vector<pto> convex_hull(vector<pto> P) {
16     int n = P.size(), k = 0;
17     vector<pto> H(2 * n);
18     // Sort points lexicographically
19     sort(P.begin(), P.end());
20     // Build lower hull
21     for (int i = 0; i < n; ++i) {
22         // esse <= 0 representa sentido anti-horario, caso deseje mudar
23         // trocar por >= 0
24         while (k >= 2 && cross(H[k - 2], H[k - 1], P[i]) <= 0)
25             k--;
26         H[k++] = P[i];
27     }
28     // Build upper hull
29     for (int i = n - 2, t = k + 1; i >= 0; i--) {
30         // esse <= 0 representa sentido anti-horario, caso deseje mudar
31         // trocar por >= 0
32         while (k >= t && cross(H[k - 2], H[k - 1], P[i]) <= 0)
33             k--;
34         H[k++] = P[i];
35     }
36     H.resize(k);
37     /* o último ponto do vetor é igual ao primeiro, atente para isso
38        as vezes é necessário mudar */
39     return H;
40 }

```

### 5.14. UPPER AND LOWER HULL - $O(n \log(n))$

```

1 struct pto {
2     double x, y;
3     bool operator <(const pto &p) const {

```

```

4         return x < p.x || (x == p.x && y < p.y);
5         /* a impressao será em prioridade por mais a esquerda, mais
6            abaixo, e anti-horário pelo cross abaixo */
7     }
8 };
9 double cross(const pto &O, const pto &A, const pto &B) {
10     return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
11 }
12
13 vector<pto> lower, upper;
14
15 vector<pto> convex_hull(vector<pto> &P) {
16     int n = P.size(), k = 0;
17     vector<pto> H(2 * n);
18     // Sort points lexicographically
19     sort(P.begin(), P.end());
20     // Build lower hull
21     for (int i = 0; i < n; ++i) {
22         // esse <= 0 representa sentido anti-horario, caso deseje mudar
23         // trocar por >= 0
24         while (k >= 2 && cross(H[k - 2], H[k - 1], P[i]) <= 0)
25             k--;
26         H[k++] = P[i];
27     }
28     // Build upper hull
29     for (int i = n - 2, t = k + 1; i >= 0; i--) {
30         // esse <= 0 representa sentido anti-horario, caso deseje mudar
31         // trocar por >= 0
32         while (k >= t && cross(H[k - 2], H[k - 1], P[i]) <= 0)
33             k--;
34         H[k++] = P[i];
35     }
36     H.resize(k);
37     /* o último ponto do vetor é igual ao primeiro, atente para isso
38        as vezes é necessário mudar */
39
40     int j = 1;
41     lower.pb(H.front());
42     while(H[j].x >= H[j-1].x) {
43         lower.pb(H[j++]);
44     }
45
46     int l = H.size()-1;
47     while(l >= j) {
48         upper.pb(H[l--]);
49     }
50     upper.pb(H[l--]);
51
52     return H;
53 }

```

### 5.15. CONDIÇÃO DE EXISTÊNCIA DE UM TRIÂNGULO

```

1
2 | b - c | < a < b + c
3 | a - c | < b < a + c
4 | a - b | < c < a + b

```

Para  $a < b < c$ , basta checar  
 $a + b > c$

OBS: Para um conjunto  $n \geq 100$  sempre existe um triângulo válido, pois a sequência de triângulos não válidos seguem a sequência de Fibonacci e  $\text{Fib}(100) > 2^{64}$

5.16. ÁREA POLÍGONO 3D -  $O(n)$ 

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 struct point{
6     double x,y,z;
7     void operator=(const point & b){
8         x = b.x;
9         y = b.y;
10        z = b.z;
11    }
12 };
13
14 point cross(point a, point b){
15     point ret;
16     ret.x = a.y*b.z - b.y*a.z;
17     ret.y = a.z*b.x - a.x*b.z;
18     ret.z = a.x*b.y - a.y*b.x;
19     return ret;
20 }
21
22 int main(){
23     int num;
24     cin >> num;
25     point v[num];
26     for(int i=0; i<num; i++) cin >> v[i].x >> v[i].y >> v[i].z;
27
28     point cur;
29     cur.x = 0, cur.y = 0, cur.z = 0;
30
31     for(int i=0; i<num; i++){
32         point res = cross(v[i], v[(i+1)%num]);
33         cur.x += res.x;
34         cur.y += res.y;
35         cur.z += res.z;
36     }
37
38     double ans = sqrt(cur.x*cur.x + cur.y*cur.y + cur.z*cur.z);
39
40     double area = abs(ans);
41
42     cout << fixed << setprecision(9) << area/2. << endl;
43 }

```

## 6. Strings

6.1. KMP ALGORITMO -  $O(n+m)$ 

```

1 // (achar substring numa string)
2 // Complexidade: O(n)
3
4 // TXT = "ABABABABA", PAT = "AB"
5 // Fills lps[] for given pattern pat[0..M-1]
6 void compute(string &pat, int m, int lps[]) {
7     // length of the previous longest prefix suffix
8     int len = 0;
9     lps[0] = 0; // lps[0] is always 0
10
11     // the loop calculates lps[i] for i = 1 to m-1
12     int i = 1;
13     while (i < m) {
14         if (pat[i] == pat[len]) {
15             len++, i++;
16             lps[i] = len;
17         } else { // (pat[i] != pat[len])
18             if (len != 0)
19                 len = lps[len-1];
20             else
21                 lps[i++] = 0;
22         }
23     }
24 }
25
26 // Driver program to test above function
27 // Prints occurrences of txt[] in pat[]
28 int KMPSearch(string pat, string txt) {
29     int n = txt.size(), m = pat.size();
30     // create lps[] that will hold the longest prefix suffix values for pattern
31     int lps[m], c = 0;
32
33     // Preprocess the pattern (calculate lps[] array)
34     compute(pat, m, lps);
35
36     int i = 0; // index for txt[]
37     int j = 0; // index for pat[]
38     while (i < n) {
39         if (pat[j] == txt[i])
40             j++, i++;
41
42         if (j == m) {
43             printf("pattern at %d \n", i-j);
44             j = lps[j-1];
45             c++;
46             // Múltiplos matches
47         }
48
49         // mismatch after j matches
50         else if (i < n && pat[j] != txt[i]){
51             // Do not match lps[0..lps[j-1]] characters, they will match anyway
52             if (j != 0)
53                 j = lps[j-1];
54             else
55                 i = i+1;
56         }
57     }
58     // return the number of occurrences
59     return c;
60 }
61 int main() {

```

```

62 string txt = "ABABDABACDABABCABAB";
63 string pat = "AB";
64 KMPSearch(pat, txt);
65 }
66 Output:
67 pattern at 0,2,5,10,12,15,17

```

## 6.2. TRIE

```

1 struct No {
2     map<char, No*> trans;
3     bool word = false;
4 };
5
6 No* head;
7
8 No* makeNo() {
9     return new No();
10 }
11
12 void insert(string x) {
13
14     No* aux = head;
15
16     for(char c: x) {
17         if(aux->trans[c]) {
18             aux = aux->trans[c];
19         } else {
20             aux = aux->trans[c] = makeNo();
21         }
22     }
23     aux->word = true;
24 }
25
26 bool search(string s){
27     no* aux = head;
28     for(char c : s){
29         if(aux->trans[c])
30             aux = aux->trans[c];
31         else
32             return false;
33     }
34     return aux->word;
35 }
36
37 int main(){
38     head = makeNo();
39 }

```

## 6.3. TRIE (CONTANDO TRANSICOES)

```

1 // Contém também delegação de números e conversão
2 // NUMERO TA EM LONG LONG
3 struct No {
4     map<char, pair<No*, int> > trans;
5     bool word = false;
6 };
7
8 No *head;
9
10 No *makeNo() {
11     return new No();
12 }

```

```

13 char comp(char x) {
14     return (x == '0' ? '1' : '0');
15 }
16
17 string toBin(int x) {
18     string r;
19     for(int i = 32; i >= 0; i--)
20         r += ((bool)(x & (1ll<<i))) + '0';
21     return r;
22 }
23
24 string toBinQuery(int x) {
25     string r = "";
26     for(int i = 32; i >= 0; i--) {
27         if((bool)(x & (1ll<<i)))
28             r += '0';
29         else
30             r += '1';
31     }
32     return r;
33 }
34
35 int toInt(string x) {
36     int c = 0;
37     for(int i = 0; i < 33; i++) {
38         if(x[i] == '1')
39             c += (1ll<<(32-i));
40     }
41     return c;
42 }
43
44 void insert(string x) {
45     No *aux = head;
46     for(char c: x) {
47
48         if(aux->trans[c].ff) {
49             aux->trans[c].ss += 1;
50         } else {
51             aux->trans[c].ff = makeNo();
52             aux->trans[c].ss = 1;
53         }
54         aux = aux->trans[c].ff;
55     }
56     aux->word = true;
57 }
58
59 void del(string x) {
60     No *aux = head;
61     for(char c: x) {
62         if(--(aux->trans[c].ss) == 0) {
63             aux->trans[c].ff = NULL;
64             return;
65         }
66         aux = aux->trans[c].ff;
67     }
68 }
69
70 string query(string x) {
71     No *aux = head;
72     string r = "";
73     for(char c: x) {
74         if(aux->trans[c].ff) {
75             r += c;
76         }
77     }

```

```

78     aux = aux->trans[c].ff;
79 } else {
80     r += comp(c);
81     aux = aux->trans[comp(c)].ff;
82 }
83 }
84 return r;
85 }
86
87 int main() {
88     No *head = makeNo();
89 }

```

```

24     if (i <= r)
25         z[i] = min (r-i+1, z[i - 1]);
26     while (i + z[i] < n && s[z[i]] == s[i + z[i]])
27         ++z[i];
28     if (i + z[i] - 1 > r)
29         l = i, r = i + z[i] - 1;
30 }
31 return z;
32 }

```

#### 6.4. TRIE (MAXIMUM XOR BETWEEN TWO ELEMENTS)

```

1 1. Dada uma trie de números binários e um numero X, tente achar o número
   máximo que resultante da operação XOR
2
3 Ex: Para o número 10(=(1010)2), o número que resulta no xor máximo é (0101)2
   , tente acha-lo na trie.

```

#### 6.5. TRIE (MAXIMUM XOR SUM)

```

1 // XOR(L,R) = XOR(1,L-1) ^ XOR(1,R)
2 ans= pre = 0
3 Trie.insert(0)
4 for i=1 to N:
5     pre = pre XOR a[i]
6     Trie.insert(pre)
7     ans=max(ans, Trie.query(pre))
8 print ans
9
10 // a funcao query é a mesma da maximum xor between two elements

```

#### 6.6. Z-FUNCTION - $O(n)$

```

1 // What is Z Array?
2 // For a string str[0..n-1], Z array is of same length as string. An element
   Z[i] of Z array stores length of the longest substring starting from
   str[i] which is also a prefix of str[0..n-1]. The first entry of Z array
   is meaning less as complete string is always prefix of itself.
3 // Example:
4 // Index
5 // 0   1   2   3   4   5   6   7   8   9  10  11
6 // Text
7 // a   a   b   c   a   a   b   x   a   a   a   z
8 // Z values
9 // X   1   0   0   3   1   0   0   2   2   1   0
10 // More Examples:
11 // str = "aaaaaa"
12 // Z[] = {x, 5, 4, 3, 2, 1}
13
14 // str = "aabaacd"
15 // Z[] = {x, 1, 0, 2, 1, 0, 0}
16
17 // str = "abababab"
18 // Z[] = {x, 0, 6, 0, 4, 0, 2, 0}
19
20 vector<int> z_function(string s) {
21     int n = (int) s.length();
22     vector<int> z(n);
23     for (int i = 1, l = 0, r = 0; i < n; ++i) {

```

## 7. Data Structures

### 7.1. RMQ MIN-MAX + LAZY PROPAGATION

```

1 struct node {
2
3     int menor, maior, lazy;
4     node() {}
5     node(int menor, int maior, int lazy) : menor(menor), maior(maior),
        lazy(lazy) {}
6
7     node operator^(const node &x) const {
8         return node(min(menor, x.menor), max(maior, x.maior), 0);
9     }
10
11 };
12
13 int arr[1000003];
14 node tree[4123456];
15
16 void propagate(int l, int r, int pos) {
17
18     if(tree[pos].lazy != 0) {
19         tree[pos].menor += tree[pos].lazy;
20         tree[pos].maior += tree[pos].lazy;
21         if(l != r) {
22             tree[2*pos+1].lazy += tree[pos].lazy;
23             tree[2*pos+2].lazy += tree[pos].lazy;
24         }
25         tree[pos].lazy = 0;
26     }
27
28 }
29
30 node build(int l, int r, int pos) {
31     if(l == r)
32         return tree[pos] = node(arr[l], arr[l], 0);
33
34     int mid = (l+r) >> 1;
35     return tree[pos] = build(l, mid, 2*pos+1)^build(mid + 1, r, 2*pos+2);
36 }
37
38 node query(int l, int r, int i, int j, int pos) {
39
40     propagate(l, r, pos);
41
42     if(l > r || l > j || r < i) {
43         return node(INF, -INF, 0);
44     }
45
46     if(i <= l && r <= j) {
47         return tree[pos];
48     }
49
50     int mid = (l+r)>>1;
51     return query(l, mid, i, j, 2*pos+1)^query(mid+1, r, i, j, 2*pos+2);
52 }
53
54 // range sum update
55 node update(int l, int r, int i, int j, int v, int pos) {
56
57     propagate(l, r, pos);
58
59     if(l > r || l > j || r < i) {
60         return tree[pos];

```

```

61     }
62
63     if(i <= l && r <= j) {
64         tree[pos].lazy += v;
65         propagate(l, r, pos);
66         return tree[pos] = node(tree[pos].menor, tree[pos].maior, 0);
67     }
68
69     int mid = (l+r)>>1;
70     return tree[pos] = update(l, mid, i, j, v, 2*pos+1)^update(mid+1, r, i, j,
        v, 2*pos+2);
71
72 }

```

### 7.2. ARVORE BINARIA

```

1 // C program to demonstrate delete operation in binary search tree
2 #include<stdio.h>
3 #include<stdlib.h>
4
5 struct node {
6     int key;
7     struct node *left, *right;
8 };
9
10 // A utility function to create a new BST node
11 struct node *newNode(int item) {
12     struct node *temp = (struct node *)malloc(sizeof(struct node));
13     temp->key = item;
14     temp->left = temp->right = NULL;
15     return temp;
16 }
17
18 // A utility function to do inorder traversal of BST
19 void inorder(struct node *root) {
20     if (root != NULL) {
21         inorder(root->left);
22         printf("%d ", root->key);
23         inorder(root->right);
24     }
25 }
26
27 /* A utility function to insert a new node with given key in BST */
28 struct node* insert(struct node* node, int key) {
29     /* If the tree is empty, return a new node */
30     if (node == NULL) return newNode(key);
31
32     /* Otherwise, recur down the tree */
33     if (key < node->key)
34         node->left = insert(node->left, key);
35     else
36         node->right = insert(node->right, key);
37
38     /* return the (unchanged) node pointer */
39     return node;
40 }
41
42 /* Given a non-empty binary search tree, return the node with minimum
43    key value found in that tree. Note that the entire tree does not
44    need to be searched. */
45 struct node * minValueNode(struct node* node) {
46     struct node* current = node;
47
48     /* loop down to find the leftmost leaf */

```

```

49 while (current->left != NULL)
50     current = current->left;
51
52 return current;
53 }
54
55 /* Given a binary search tree and a key, this function deletes the key
56 and returns the new root */
57 struct node* deleteNode(struct node* root, int key) {
58     // base case
59     if (root == NULL) return root;
60
61     // If the key to be deleted is smaller than the root's key,
62     // then it lies in left subtree
63     if (key < root->key)
64         root->left = deleteNode(root->left, key);
65
66     // If the key to be deleted is greater than the root's key,
67     // then it lies in right subtree
68     else if (key > root->key)
69         root->right = deleteNode(root->right, key);
70
71     // if key is same as root's key, then This is the node
72     // to be deleted
73     else {
74         // node with only one child or no child
75         if (root->left == NULL) {
76             struct node *temp = root->right;
77             free(root);
78             return temp;
79         } else if (root->right == NULL) {
80             struct node *temp = root->left;
81             free(root);
82             return temp;
83         }
84
85         // node with two children: Get the inorder successor (smallest
86         // in the right subtree)
87         struct node* temp = minValueNode(root->right);
88
89         // Copy the inorder successor's content to this node
90         root->key = temp->key;
91
92         // Delete the inorder successor
93         root->right = deleteNode(root->right, temp->key);
94     }
95     return root;
96 }
97
98 // Driver Program to test above functions
99 int main() {
100     /* Let us create following BST
101
102         50
103        /  \
104       30   70
105      / \  / \
106     20 40 60 80 */
107     struct node *root = NULL;
108     root = insert(root, 50);
109     root = insert(root, 30);
110     root = insert(root, 20);
111     root = insert(root, 40);
112     root = insert(root, 70);
113     root = insert(root, 60);
114     root = insert(root, 80);

```

```

114
115 printf("Inorder traversal of the given tree \n");
116 inorder(root);
117
118 printf("\nDelete 20\n");
119 root = deleteNode(root, 20);
120 printf("Inorder traversal of the modified tree \n");
121 inorder(root);
122
123 printf("\nDelete 30\n");
124 root = deleteNode(root, 30);
125 printf("Inorder traversal of the modified tree \n");
126 inorder(root);
127
128 printf("\nDelete 50\n");
129 root = deleteNode(root, 50);
130 printf("Inorder traversal of the modified tree \n");
131 inorder(root);
132
133 return 0;
134 }

```

### 7.3. SQRT DECOMPOSITION

```

1 // Problem: Sum from l to r
2 // Ver MO'S ALGORITHM
3 // -----
4 int getId(int indx,int blockSZ) {
5     return indx/blockSZ;
6 }
7 void init(int sz) {
8     for(int i=0; i<=sz; i++)
9         BLOCK[i]=inf;
10 }
11 int query(int left, int right) {
12     int startBlockIndex=left/sqrt;
13     int endIBlockIndex = right / sqrt;
14     int sum = 0;
15     for (int i = startBlockIndex + 1; i < endIBlockIndex; i++) {
16         sum += blockSums[i];
17     }
18     for(i=left...(startBlockIndex*BLOCK_SIZE-1))
19         sum += a[i];
20     for(j = endIBlockIndex*BLOCK_SIZE ... right)
21         sum += a[i];
22 }

```

### 7.4. MO'S ALGORITHM (MOST FREQUENT VALUE IN INTERVALS) - $O((m+n)*\sqrt{n})$

```

1 struct Tree {
2     int l, r, ind;
3 };
4 Tree query[311111];
5 int arr[311111];
6 int freq[111111];
7 int ans[311111];
8 int block = sqrt(n), cont = 0;
9
10 bool cmp(Tree a, Tree b) {
11     if(a.l/block == b.l/block)
12         return a.r < b.r;
13     return a.l/block < b.l/block;

```

```

14 }
15
16 void add(int pos) {
17     freq[arr[pos]]++;
18     if(freq[arr[pos]] == 1) {
19         cont++;
20     }
21 }
22 void del(int pos) {
23     freq[arr[pos]]--;
24     if(freq[arr[pos]] == 0)
25         cont--;
26 }
27 int main () {
28     int n; cin >> n;
29     block = sqrt(n);
30
31     for(int i = 0; i < n; i++) {
32         cin >> arr[i];
33         freq[arr[i]] = 0;
34     }
35
36     int m; cin >> m;
37
38     for(int i = 0; i < m; i++) {
39         cin >> query[i].l >> query[i].r;
40         query[i].l--, query[i].r--;
41         query[i].ind = i;
42     }
43     sort(query, query + m, cmp);
44
45     int s,e;
46     s = e = query[0].l;
47     add(s);
48     for(int i = 0; i < m; i++) {
49         while(s > query[i].l)
50             add(--s);
51         while(s < query[i].l)
52             del(s++);
53         while(e < query[i].r)
54             add(++e);
55         while(e > query[i].r)
56             del(e--);
57         ans[query[i].ind] = cont;
58     }
59
60     for(int i = 0; i < m; i++)
61         cout << ans[i] << endl;
62 }

```

#### 7.5. MERGE SORT TREE (K-ESIMO MAIOR ELEMENTO NUM INTERVALO, VALORES MAIORES QUE K NUM INTERVALO, ...)

```

1 // retornar a qtd de números maiores q um numero k numa array de i...j
2 struct Tree {
3     vector<int> vet;
4 };
5 Tree tree[4*(int)3e4];
6 int arr[(int)5e4];
7
8 int query(int l,int r, int i, int j, int k, int pos) {
9     if(l > j || r < i)
10         return 0;
11

```

```

12     if(i <= l && r <= j) {
13         auto it = upper_bound(tree[pos].vet.begin(), tree[pos].vet.end(), k);
14         return tree[pos].vet.end()-it;
15     }
16
17     int mid = (l+r)>>1;
18     return query(l, mid, i, j, k, 2*pos+1) + query(mid+1,r,i,j,k,2*pos+2);
19 }
20
21 void build(int l, int r, int pos) {
22
23     if(l == r) {
24         tree[pos].vet.pb(arr[l]);
25         return;
26     }
27
28     int mid = (l+r)>>1;
29     build(l, mid, 2*pos+1);
30     build(mid + 1, r, 2*pos+2);
31
32     merge(tree[2*pos+1].vet.begin(), tree[2*pos+1].vet.end(),
33           tree[2*pos+2].vet.begin(), tree[2*pos+2].vet.end(),
34           back_inserter(tree[pos].vet));
35 }

```

#### 7.6. ORDENAÇÃO DE ESTRUTURAS (PQ,ETC)

```

1 struct cmp {
2     bool operator(ii a, ii b) {
3         //ordena primeiro pelo first(decrecente), dps pelo second(crescente)
4         if(a.first == b.first)
5             return a.second < b.second;
6         return a.first > b.first;
7     }
8 }
9 Ex: pq<ii,vector<ii>,cmp> fila;

```

#### 7.7. POLICY BASED DATA STRUCTURES - ORDERED SET

```

1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/assoc_container.hpp>
3 #include <ext/pb_ds/trie_policy.hpp>
4
5 using namespace std;
6 using namespace __gnu_pbds;
7
8 typedef tree<
9     int,
10     null_type,
11     less<int>,
12     rb_tree_tag,
13     tree_order_statistics_node_update>
14     ordered_set;
15
16 ordered_set X;
17 X.insert(1); X.insert(2);
18 X.insert(4); X.insert(8);
19 X.insert(16);
20 // 1, 2, 4, 8, 16
21 // retorna o k-ésimo maior elemento a partir de 0
22 cout<<*X.find_by_order(1)<<endl; // 2
23 cout<<*X.find_by_order(2)<<endl; // 4
24 cout<<*X.find_by_order(4)<<endl; // 16

```

```

25 cout<<(end(X)==X.find_by_order(6))<<endl; // true
26
27 // retorna o número de itens estritamente menores que o número
28 cout<<X.order_of_key(-5)<<endl; // 0
29 cout<<X.order_of_key(1)<<endl; // 0
30 cout<<X.order_of_key(3)<<endl; // 2
31 cout<<X.order_of_key(4)<<endl; // 2
32 cout<<X.order_of_key(400)<<endl; // 5

```

### 7.8. CONTANDO INVERSÕES

```

1 int count_inversion_merge(int array[], int first, int last) {
2     int mid = (first+last)/2;
3     int ai = first, bi = mid+1;
4     int final[last-first+1], finali=0;
5     int inversion = 0, i;
6
7     while (ai <= mid && bi <= last) {
8         if (array[ai] <= array[bi]) {
9             final[finali++] = array[ai++];
10        } else {
11            final[finali++] = array[bi++];
12            inversion += mid - ai + 1;
13        }
14    }
15
16    while (ai <= mid)
17        final[finali++] = array[ai++];
18
19    while (bi <= last)
20        final[finali++] = array[bi++];
21
22    for (i=0 ; i<last-first+1 ; i++)
23        array[i+first] = final[i];
24
25    return inversion;
26 }
27
28 int count_inversion(int array[], int a, int b) {
29     // a começa com 0 e b com n-1
30     int x, y, z, mid;
31     if (a >= b) return 0;
32
33     mid = (a+b)/2;
34
35     x = count_inversion(array, a, mid);
36     y = count_inversion(array, mid+1, b);
37     z = count_inversion_merge(array, a, b);
38
39     return x+y+z;
40 }

```

## 8. GRAPHS

### 8.1. CICLO GRAFO - $O(V+E)$

```

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 (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 = parent[v])
40             cycle.push_back(v);
41         cycle.push_back(cycle_start);
42         reverse(cycle.begin(), cycle.end());
43
44         cout << "Cycle found: ";
45         for (int v : cycle)
46             cout << v << " ";
47         cout << endl;
48     }
49 }

```

### 8.2. CHECA GRAFO BIPARTIDO - $O(V+E)$

```

1 bool isBipartite(int src) {
2
3     int colorArr[V];
4     for (int i = 0; i < V; ++i)
5         colorArr[i] = -1;
6     colorArr[src] = 1;
7
8     queue<int> q; q.push(src);

```



```

9
10 while (!q.empty()) {
11     int u = q.front(); q.pop();
12
13     // Find all non-colored adjacent vertices
14     for (auto it = adj[u].begin(); it != adj[u].end(); it++) {
15         //Return false if there is a self-loop
16         if (u == *it)
17             return false;
18         // An edge from u to v exists and destination v is not colored
19
20         if (colorArr[*it] == -1) {
21             // Assign alternate color to this adjacent v of u
22             colorArr[*it] = 1 - colorArr[u];
23             q.push(*it);
24         }
25         // An edge from u to v exists and destination v is colored with same
26         // color as u
27         else if (colorArr[*it] == colorArr[u])
28             return false;
29     }
30     // If we reach here, then all adjacent vertices can be colored with
31     // alternate color
32     return true;
33 }

```

### 8.3. BELLMAN FORD (MENOR CAMINHO ARESTAS NEGATIVAS) - $O(V \cdot E)$

```

1 void BellmanFord(vector<edge> edges, int src, int V, int E) {
2     // V = qtd of vertices, E = qtd de arestas
3     int dist[V];
4
5     // Step 1: Initialize distances from src to all other vertices
6     // as INFINITE
7     for (int i = 0; i < V; i++)
8         dist[i] = INF;
9     dist[src] = 0;
10    // Step 2: Relax all edges |V| - 1 times. A simple shortest path from src
11    // to any other vertex can have at-most |V| - 1 edges
12    for (int i = 1; i <= V-1; i++) {
13        for (int j = 0; j < E; j++) {
14            int u = edges[j].src;
15            int v = edges[j].dest;
16            int weight = edges[j].weight;
17            if (dist[u] != INF && dist[u] + weight < dist[v])
18                dist[v] = dist[u] + weight;
19        }
20    }
21    // Step 3: check for NEGATIVE-WEIGHT CYCLES. The above step guarantees
22    // shortest distances if graph doesn't contain negative weight cycle. If
23    // we get a shorter path, then there is a cycle.
24    for (int i = 0; i < E; i++) {
25        int u = edges[i].src;
26        int v = edges[i].dest;
27        int weight = edges[i].weight;
28        if (dist[u] != INF && dist[u] + weight < dist[v])
29            printf("Graph contains negative weight cycle");
30    }
31    printArr(dist, V);
32 }

```

### 8.4. DIAMETRO EM ARVORE (MAIOR CAMINHO ENTRE DOIS VERTICES)

1 Calcula qual o vértice a mais distante de um qualquer vértice X e do vértice A calcula-se o vértice B mais distante dele.

### 8.5. PONTES NUM GRAFO - $O(V+E)$

```

1 //SE TIRA-LAS O GRAFO FICA DESCONEXO
2 // OBS: PRESTAR ATENCAO EM SELF-LOOPS, é MELHOR NÃO ADICIONA-LOS
3 // SO FUNCIONA EM GRAFO NÃO DIRECIONADO
4 int t=1;
5 vector<int> T((int)2e6,0); //Tempo necessário para chegar naquele vértice na
6 // dfs
7 vector<int> adj[(int)2e6];
8 vector<int> Low((int)2e6); // Tempo "mínimo" para chegar naquele vértice na
9 // dfs
10 vector<int> ciclo((int)2e6, false);
11 vector<ii> bridges;
12 void dfs(int u, int p){
13     Low[u] = T[u] = t;
14     t++;
15     for(auto v : adj[u]){
16         if(v==p){
17             //checa arestas paralelas
18             p=-1;
19             continue;
20         }
21         //se ele ainda não foi visited
22         else if(T[v]==0){
23             dfs(v,u);
24             Low[u]=min(Low[u], Low[v]);
25             if(Low[v]>T[u]) {
26                 bridges.pb(ii(min(u,v), (max(u,v))));
27                 // ponte de u para v
28             }
29         }
30         else
31             Low[u]=min(Low[u], T[v]);
32         ciclo[u] |= (T[u]>=Low[v]);
33         //checa se o vértice u faz parte de um ciclo
34     }
35 }
36 void clear() {
37     for(int i = 0; i <= n; i++) {
38         T[i] = 0, Low[i] = 0, adj[i].clear(), ciclo[i] = false;
39     }
40     bridges.clear();
41 }
42
43 signed main () {
44     for(int i = 0; i < n; i++) {
45         if(T[i] == 0)
46             dfs(i, -1);
47     }
48     sort(bridges.begin(), bridges.end());
49
50     cout << (int)bridges.size() << endl;
51     for(int i = 0; i < bridges.size(); i++) {
52         cout << bridges[i].ff << " - " << bridges[i].ss << endl;
53     }
54     cout << endl;
55 }

```

```

57
58 clear();
59
60 }

```

8.6. PONTOS DE ARTICULAÇÃO NUM GRAFO (se retirar esses vértices o grafo fica desconexo) -  $O(V+E)$

```

1 // SE TIRAR TAIS VERTICES O GRAFO FICA DESCONEXO
2

```

```

3 vector<bool> ap(100000,false);
4 vector<int> low(100000,0), T(100000,0);
5 int tempo = 1;
6 list<int> adj[100000];
7
8 void artPoint(int u, int p) {
9
10     low[u] = T[u] = tempo++;
11     int children = 0;
12
13     for(int v: adj[u]) {
14
15         // cuidado com arestas paralelas
16         // se tiver nao podemos fazer assim
17
18         if(T[v] == 0) {
19
20             children++;
21             artPoint(v,u);
22             low[u] = min(low[v],low[u]);
23
24             if(p == -1 && children > 1) {
25                 ap[u] = true;
26             }
27
28             if(p != -1 && low[v] > T[u])
29                 ap[u] = true;
30             } else if(v != p)
31                 low[u] = min(low[u], T[v]);
32
33     }
34 }
35
36 int main() {
37
38     for(int i = 0; i < n; i++)
39         if(T[i] == 0)
40             artPoint(i,-1);
41 }

```

8.7. LCA (shortest path)

```

1 // pra achar so o lca é so tirar o q ta comentado com (// short path)
2 int anc[105000][(int)log2(150000)+2];
3
4 vector<ii> adj[105000];
5 vector<int> level(105000), dist(105000), vis(105000,false);
6
7 void dfs(int u, int p, int d = 1) {
8
9     level[u] = d;
10     vis[u] = true;
11

```

```

12     for(ii x: adj[u]) {
13         if(vis[x.ff])
14             continue;
15         dist[x.ff] = dist[u] + x.ss; // short path
16         dfs(x.ff,u,d+1);
17     }
18 }
19
20 void build(int n) {
21
22     for(int i = 0; i <= n; i++) {
23         for(int j = 0; (1<<j) < n; j++) {
24             anc[i][j] = -1;
25         }
26     }
27
28     // ler a aresta i -> x com peso c
29     for(int i = 1; i < n; i++) {
30         int x,c;
31         cin >> x >> c;
32         adj[x].pb(ii(i,c));
33         adj[i].pb(ii(x,c));
34         anc[i][0] = x;
35     }
36
37     dist[0] = 0; // short path
38     dfs(0,-1);
39
40     for(int j = 1; (1<<j) < n; j++) {
41         for(int i = 0; i < n; i++) {
42             if(anc[i][j-1] != -1)
43                 anc[i][j] = anc[anc[i][j-1]][j-1];
44         }
45     }
46 }
47
48 int lca(int a, int b) {
49
50     int ac = a, bc = b;
51     // a esta mais embaixo na arvore
52     if(level[b] > level[a])
53         swap(a,b);
54
55     int lg = log2(level[a]);
56
57     // colocando a e b no msm nivel
58     for(int i = lg; i >= 0; i--) {
59         if(level[a]-(1<<i) >= level[b]) {
60             a = anc[a][i];
61         }
62     }
63
64     // return a; // to find lca
65     if(a == b)
66         return dist[ac] + dist[bc] - 2*dist[a]; // short path
67
68     // achando o filho do lca
69     for(int i = lg; i >= 0; i--) {
70         if(anc[a][i] != -1 && anc[a][i] != anc[b][i]) {
71             a = anc[a][i];
72             b = anc[b][i];
73         }
74     }
75     // anc[a][0] é o LCA
76     // return anc[a][0]; retorna o lca

```

```

77 //cout << "LCA = " << anc[a][0] << endl;
78 return dist[ac] + dist[bc] - 2*dist[anc[a][0]];
79
80 }
81 /*
82 Codigo para voltar k posicoes a partir de x
83 for(int i = 0; i < 32; i++) {
84     if(k & (1<<i))
85         x = anc[x][i];
86 }
87 return x;
88 */

```

### 8.8. FORD FULKERSSON (MAXIMUM FLOW) - $O(V*(E^2))$

```

1 int rGraph[2000][2000];
2 int graph[2000][2000];
3
4 int V;
5 bool bfs(int s, int t, int parent[]) {
6     bool visited[V];
7     memset(visited, 0, sizeof(visited));
8
9     // Create a queue, enqueue source vertex and mark source vertex
10    // as visited
11    queue<int> q;
12    q.push(s);
13    visited[s] = true;
14    parent[s] = -1;
15
16    // Standard BFS Loop
17    while (!q.empty()) {
18        int u = q.front();
19        q.pop();
20
21        for (int v=0; v<V; v++) {
22            if (visited[v]==false && rGraph[u][v] > 0) {
23                q.push(v);
24                parent[v] = u;
25                visited[v] = true;
26            }
27        }
28    }
29    // If we reached sink in BFS starting from source, then return true, else
30    // false
31    return (visited[t] == true);
32 }
33
34 // Returns the maximum flow from s to t in the given graph
35 int fordFulkerson(int s, int t) {
36     int u, v;
37     // Create a residual graph and fill the residual graph with given
38     // capacities in the original graph as residual capacities in residual
39     // graph residual capacity of edge from i to j (if there is an edge. If
40     // rGraph[i][j] is 0, then there is not)
41     for (u = 0; u < V; u++)
42         for (v = 0; v < V; v++)
43             rGraph[u][v] = graph[u][v];
44
45     int parent[V]; // This array is filled by BFS and to store path
46
47     int max_flow = 0; // There is no flow initially
48
49     // Augment the flow while there is path from source to sink

```

```

46 while (bfs(s, t, parent)) {
47     // Find minimum residual capacity of the edges along the path filled by
48     // BFS. Or we can say find the maximum flow through the path found.
49     int path_flow = INT_MAX;
50     for (v=t; v!=s; v=parent[v]) {
51         u = parent[v];
52         path_flow = min(path_flow, rGraph[u][v]);
53     }
54
55     // update residual capacities of the edges and reverse edges
56     // along the path
57     for (v=t; v != s; v=parent[v]) {
58         u = parent[v];
59         rGraph[u][v] -= path_flow;
60         rGraph[v][u] += path_flow;
61     }
62
63     // Add path flow to overall flow
64     max_flow += path_flow;
65 }
66
67 // Return the overall flow
68 return max_flow;
69
70 // PRINT THE FLOW AFTER RUNNING THE ALGORITHM
71 void print(int n) {
72     for(int i = 1; i <= m; i++) {
73         for(int j = m+1; j <= m*2; j++) {
74             cout << "flow from i(left) to j(right) is " << graph[i][j] -
75                 rGraph[i][j] << endl;
76         }
77     }
78 }
79
80 void addEdge(int l, int r, int n, int x) {
81     graph[l][r+n] = x;
82 }
83
84 void addEdgeSource(int l, int x) {
85     graph[0][l] = x;
86 }
87
88 void addEdgeSink(int r, int n, int x) {
89     graph[r+n][V-1] = x;
90 }

```

### 8.9. DINIC (MAXIMUM FLOW) - $O(E+(V^2))$ or $O(n * m)$ for Matching

```

1 struct Dinic {
2
3     struct FlowEdge{
4         int v, rev, c, cap;
5         FlowEdge() {}
6         FlowEdge(int v, int c, int cap, int rev) : v(v), c(c), cap(cap),
7             rev(rev) {}
8     };
9
10    vector<vector<FlowEdge>> adj;
11    vector<int> level, used;
12    int src, snk, V;
13    int sz;
14    int max_flow;
15    Dinic(){}

```

```

15 Dinic(int n){
16     src = 0;
17     snk = n+1;
18     adj.resize(n+2, vector< FlowEdge >());
19     level.resize(n+2);
20     used.resize(n+2);
21     sz = n+2;
22     V = n+2;
23     max_flow = 0;
24 }
25
26 void add_edge(int u, int v, int c){
27     int id1 = adj[u].size();
28     int id2 = adj[v].size();
29     adj[u].pb(FlowEdge(v, c, c, id2));
30     adj[v].pb(FlowEdge(u, 0, 0, id1));
31 }
32
33 void add_to_src(int v, int c){
34     adj[src].pb(FlowEdge(v, c, c, -1));
35 }
36
37 void add_to_snk(int u, int c){
38     adj[u].pb(FlowEdge(snk, c, c, -1));
39 }
40
41 bool bfs(){
42     for(int i=0; i<sz; i++){
43         level[i] = -1;
44     }
45
46     level[src] = 0;
47     queue<int> q; q.push(src);
48
49     while(!q.empty()){
50         int cur = q.front();
51         q.pop();
52         for(FlowEdge e : adj[cur]){
53             if(level[e.v] == -1 && e.c > 0){
54                 level[e.v] = level[cur]+1;
55                 q.push(e.v);
56             }
57         }
58     }
59
60     return (level[snk] == -1 ? false : true);
61 }
62
63 int send_flow(int u, int flow){
64     if(u == snk) return flow;
65
66     for(int &i = used[u]; i<adj[u].size(); i++){
67         FlowEdge &e = adj[u][i];
68
69         if(level[u]+1 != level[e.v] || e.c <= 0) continue;
70
71         int new_flow = min(flow, e.c);
72         int adjusted_flow = send_flow(e.v, new_flow);
73
74         if(adjusted_flow > 0){
75             e.c -= adjusted_flow;
76             if(e.rev != -1) adj[e.v][e.rev].c += adjusted_flow;
77             return adjusted_flow;
78         }
79     }

```

```

80
81     return 0;
82 }
83
84 void calculate(){
85     if(src == snk){max_flow = -1; return;} //not sure if needed
86
87     max_flow = 0;
88
89     while(bfs()){
90         for(int i=0; i<sz; i++) used[i] = 0;
91         while(int inc = send_flow(src, INF)) max_flow += inc;
92     }
93
94 }
95
96 vector< ii > mincut(){
97     bool vis[sz];
98     for(int i=0; i<sz; i++) vis[i] = false;
99     queue<int> q;
100     q.push(src);
101     vis[src] = true;
102     while(!q.empty()){
103         int cur = q.front();
104         q.pop();
105         for(FlowEdge e : adj[cur]){
106             if(e.c > 0 && !vis[e.v]){
107                 q.push(e.v);
108                 vis[e.v] = true;
109             }
110         }
111     }
112     vector< ii > cut;
113     for(int i=1; i<=sz-2; i++){
114         if(!vis[i]) continue;
115         for(FlowEdge e : adj[i]){
116             if(1 <= e.v && e.v <= sz-2 && !vis[e.v] && e.cap > 0 && e.c == 0)
117                 cut.pb(ii(i, e.v));
118         }
119     }
120     return cut;
121 }
122
123 vector< ii > min_edge_cover(){
124     bool covered[sz];
125     for(int i=0; i<sz; i++) covered[i] = false;
126     vector< ii > edge_cover;
127     for(int i=1; i<sz-1; i++){
128         for(FlowEdge e : adj[i]){
129             if(e.cap == 0 || e.v > sz-2) continue;
130             if(e.c == 0){
131                 edge_cover.pb(ii(i, e.v));
132                 covered[i] = true;
133                 covered[e.v] = true;
134                 break;
135             }
136         }
137     }
138     for(int i=1; i<sz-1; i++){
139         for(FlowEdge e : adj[i]){
140             if(e.cap == 0 || e.v > sz-2) continue;
141             if(e.c == 0) continue;
142             if(!covered[i] || !covered[e.v]){
143                 edge_cover.pb(ii(i, e.v));
144                 covered[i] = true;

```

```

144         covered[e.v] = true;
145     }
146 }
147 }
148 return edge_cover;
149 }
150
151 vector<vector<int>> allFlow() {
152     vector<int> row(V, 0);
153     vector<vector<int>> ret(V, row);
154
155     for(int i = 0; i < V; i++) {
156         for(FlowEdge x: adj[i]) {
157             // flow from vertex i to x.v
158             ret[i][x.v] = x.cap - x.c;
159         }
160     }
161
162     // for(int i = 0; i < V; i++) {
163     //     for(int j = 0; j < V; j++) {
164     //         cout << ret[i][j] << ' ';
165     //     }
166     //     cout << endl;
167     // }
168
169     return ret;
170 }
171
172 };

```

8.10. CAMINHO EULERIANO (Caminho para visitar todas as arestas) -  $O(V+E)$

```

1 void DFSUtil(int v, bool vis[]) {
2     // Mark the current node as vis and print it
3     vis[v] = true;
4
5     // Recur for all the vertices adjacent to this vertex
6     vector<int>::iterator i;
7     for (i = adj[v].begin(); i != adj[v].end(); ++i)
8         if (!vis[*i])
9             DFSUtil(*i, vis);
10 }
11
12 // Method to check if all non-zero degree vertices are connected.
13 // It mainly does DFS traversal starting from
14 bool isConnected() {
15     // Mark all the vertices as not vis
16     bool vis[V];
17     int i;
18     for (i = 0; i < V; i++)
19         vis[i] = false;
20
21     // Find a vertex with non-zero degree
22     for (i = 0; i < V; i++)
23         if (adj[i].size() != 0)
24             break;
25
26     // If there are no edges in the graph, return true
27     if (i == V)
28         return true;
29
30     // Start DFS traversal from a vertex with non-zero degree
31     DFSUtil(i, vis);

```

```

32
33     // Check if all non-zero degree vertices are vis
34     for (i = 0; i < V; i++)
35         if (vis[i] == false && adj[i].size() > 0)
36             return false;
37
38     return true;
39 }
40
41 /* The function returns one of the following values
42 0 --> If graph is not Eulerian
43 1 --> If graph has an Euler path (Semi-Eulerian)
44 2 --> If graph has an Euler Circuit (Eulerian) */
45 int isEulerian() {
46     // Check if all non-zero degree vertices are connected
47     if (isConnected() == false)
48         return 0;
49
50     // Count vertices with odd degree
51     int odd = 0;
52     for (int i = 0; i < V; i++)
53         if (adj[i].size() & 1)
54             odd++;
55
56     // If count is more than 2, then graph is not Eulerian
57     if (odd > 2)
58         return 0;
59
60     // If odd count is 2, then semi-eulerian. If odd count is 0, then eulerian
61     // Note that odd count can never be 1 for undirected graph
62     return (odd) ? 1 : 2;
63 }

```

8.11. DIJKSTRA COM PRIORITY QUEUE -  $O(E*(\log V))$

```

1 int Dijkstra(int src, int dest, int n) {
2
3     priority_queue<ii, vector<ii>, greater<ii>> pq;
4     vector<int> dist(n+1, INF);
5     // vector<vector<int>> > parent(n+1);
6     // for(int i = 0; i <= n; i++)
7     //     parent[i].pb(i);
8     pq.push(make_pair(0, src));
9     dist[src] = 0;
10
11     while (!pq.empty()) {
12         int u = pq.top().ss;
13         pq.pop();
14
15         for (ii x: adj[u]) {
16             int v = x.ff;
17             int w = x.ss;
18
19             if (dist[u] + w < dist[v]) {
20                 // parent[v].clear();
21                 // parent[v].pb(u);
22                 dist[v] = dist[u] + w;
23                 pq.push(ii(dist[v], v));
24             }
25             // else if(dist[u] + w == dist[v]) {
26             //     parent[v].pb(u);
27             // }
28         }
29     }

```

```

30     return dist[dest];
31 }
32

```

### 8.12. ORDENAÇÃO TOPOLOGICA (FILAS) - $O(V+E)$

```

1 void topologicalSort(int n) {
2     vector<int> in_degree(n, 0);
3
4     for (int u=0; u<n; u++){
5         vector<int>::iterator itr;
6         for (itr = adj[u].begin(); itr != adj[u].end(); itr++)
7             in_degree[*itr]++;
8     }
9
10    queue<int> q;
11    for (int i = 0; i < n; i++)
12        if (in_degree[i] == 0)
13            q.push(i);
14
15    int cnt = 0;
16    vector<int> top_order;
17    while (!q.empty()) {
18        int u = q.front(); q.pop();
19        top_order.push_back(u);
20        vector<int>::iterator itr;
21        for (itr = adj[u].begin(); itr != adj[u].end(); itr++) {
22            if (--in_degree[*itr] == 0)
23                q.push(*itr);
24            cnt++;
25        }
26
27        if (cnt != n) {
28            cout << "There exists a cycle in the graph\n";
29            return;
30        }
31
32        for (int i=0; i<top_order.size(); i++)
33            cout << top_order[i] << " ";
34        cout << endl;
35    }

```

### 8.13. KRUSKAL (UNION FIND - $O(\log(n))$ ) - $O(E \cdot \log(V))$

```

1 struct edge {
2     int u, v, w;
3     edge() {}
4     edge(int u, int v, int w) : u(u), v(v), w(w) {}
5 }
6
7 vector<edge> edges((int)2e6);
8 vector<int> root((int)2e6), sz((int)2e6);
9
10 void init(int n) {
11
12     iota(root.begin(), root.begin() + n + 1, 0);
13     fill(sz.begin(), sz.begin() + n + 1, 1);
14 }
15
16 int Find(int x) {
17     if (root[x] == x)
18         return x;
19

```

```

20     return root[x] = Find(root[x]);
21 }
22
23 bool Union(int p, int q) {
24     p = Find(p), q = Find(q);
25     if (p == q)
26         return false;
27
28     if (sz[p] > sz[q]) {
29         root[q] = p;
30         sz[p] += sz[q];
31     } else {
32         root[p] = q;
33         sz[q] += sz[p];
34     }
35     return true;
36 }
37
38 int kruskal(int n, int m) {
39
40     init(n);
41
42     int c = 0;
43     for (int i = 0; i < (int)edges.size(); i++) {
44         if (Union(edges[i].u, edges[i].v)) {
45             c += edges[i].w;
46         }
47     }
48
49     // returns weight of mst
50     return c;
51 }
52

```

### 8.14. FLOYD WARSHALL - $O(V^3)$

```

1 // OBS: ZERAR adj[i][i] sempre
2 for (int i = 0; i < n; i++)
3     adj[i][i] = 0;
4
5 for (int k = 0; k < n; k++) {
6     for (int i = 0; i < n; i++) {
7         for (int j = 0; j < n; j++) {
8             adj[i][j] = min(adj[i][j], adj[i][k] + adj[k][j]);
9         }
10    }
11 }

```

### 8.15. SCC (Kosaraju) - $O(V+E)$

```

1 int comp[312345];
2 vector<int> adj[312345];
3 // grafo reverso de adj
4 vector<int> trans[312345];
5 vector<int> scc[312345];
6 int V;
7
8 void dfsTrans(int u, int id, int comp[]) {
9     comp[u] = id;
10    scc[id].push_back(u);
11
12    for (int v: trans[u])
13        if (!comp[v])

```

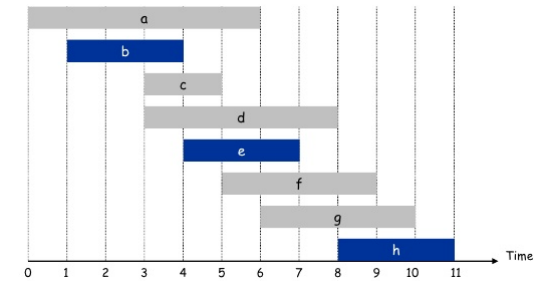
```

14     dfsTrans(v, id, comp);
15 }
16
17 void getTranspose() {
18     for (int u = 0; u < V; u++)
19         for (int v: adj[u])
20             trans[v].push_back(u);
21 }
22
23 void dfsFillOrder(int u, int comp[], stack<int> &Stack) {
24     comp[u] = true;
25
26     for (int v: adj[u])
27         if (!comp[v])
28             dfsFillOrder(v, comp, Stack);
29
30     Stack.push(u);
31 }
32
33 // The main function that finds and prints all strongly connected
34 // components
35 void computeSCC() {
36
37     stack<int> Stack;
38     // Fill vertices in stack according to their finishing times
39     for (int i = 0; i < V; i++)
40         if (comp[i] == false)
41             dfsFillOrder(i, comp, Stack);
42
43     // Create a reversed graph
44     getTranspose();
45
46     memset(comp, 0, sizeof(comp));
47
48     // Now process all vertices in order defined by Stack
49     int id = 1;
50     while (Stack.empty() == false) {
51         int v = Stack.top();
52         Stack.pop();
53
54         if (comp[v] == false)
55             dfsTrans(v, id++, comp);
56     }
57 }

```

## 9. Diverse

### 9.1. INTERVAL SCHEDULING



- 1 1 -> Ordena pelo final **do** evento, depois pelo inicio.
- 2 2 -> Vai iterando pelos eventos, se eles não tiverem horário em comum então adiciona o evento à lista.

### 9.2. PROBLEMA DAS OITO RAINHAS

```

1 #define N 4
2 bool isSafe(int mat[N][N], int row, int col) {
3     for (int i = row - 1; i >= 0; i--)
4         if (mat[i][col])
5             return false;
6     for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)
7         if (mat[i][j])
8             return false;
9     for (int i = row - 1, j = col + 1; i >= 0 && j < N; i--, j++)
10        if (mat[i][j])
11            return false;
12    return true;
13 }
14 // inicialmente a matriz esta zerada
15 int queen(int mat[N][N], int row = 0) {
16     if (row >= N) {
17         for (int i = 0; i < N; i++) {
18             for (int j = 0; j < N; j++) {
19                 cout << mat[i][j] << ' ';
20             }
21             cout << endl;
22         }
23         cout << endl << endl;
24         return false;
25     }
26     for (int i = 0; i < N; i++) {
27         if (isSafe(mat, row, i)) {
28             mat[row][i] = 1;
29             if (queen(mat, row+1))
30                 return true;
31             mat[row][i] = 0;
32         }
33     }
34     return false;
35 }

```

9.3. 3SUM PROBLEM ( $a[x]+a[y]+a[z] = \text{valor}$ ) -  $O(n^2)$ 

```

1 // vetor arr e valor x, a soma de três valores desse vetor deve ser igual a x
2
3 bool sum3(int arr[], int x, int n) {
4     sort(arr, arr + n);
5     for(int i = 0; i < n-2; i++) {
6         int l = i+1, r = n-1;
7         /* 2SUM problem -> ponteiro que aponta para o primeiro e ultimo da
            sequencia e caso a soma for menor do que x adianta em uma casa o
            ponteiro da esquerda caso seja maior diminui em uma casa o ponteiro da
            direita */
8         while(l < r) {
9             if(arr[i] + arr[l] + arr[r] == x) {
10                 return true;
11             } else if(arr[i] + arr[l] + arr[r] < x)
12                 l++;
13             else
14                 r--;
15         }
16     }
17     return false;
18 }

```

## 9.4. INFIX TO PREFIX

```

1 int main() {
2     map<char,int> prec;
3     stack<char> op;
4
5     string postfix;
6     string infix;
7     cin >> infix;
8
9     prec['+'] = prec['-'] = 1;
10    prec['*'] = prec['/'] = 2;
11    prec['^'] = 3;
12    for(int i = 0; i < infix.length(); i++) {
13        char x = infix[i];
14        if('0' <= x && x <= '9') {
15            for(i; i < infix.length() && ('0' <= infix[i] && infix[i] <= '9'); i++)
16                postfix += infix[i];
17            i--;
18        } else if(('a' <= x && x <= 'z') || ('A' <= x && x <= 'Z')) {
19            postfix += x;
20        } else if (x == '(')
21            op.push('(');
22        else if(x == ')') {
23            while(!op.empty() && op.top() != '(') {
24                postfix += op.top();
25                op.pop();
26            }
27            op.pop();
28        } else {
29            while(!op.empty() && prec[op.top()] >= prec[x]) {
30                postfix += op.top();
31                op.pop();
32            }
33            op.push(x);
34        }
35    }
36    while(!op.empty()) {
37        postfix += op.top();
38        op.pop();
39    }

```

```

39 }
40 cout << postfix << endl;
41 }

```

9.5. KADANE (MAIOR SOMA NUM VETOR) -  $O(n)$ 

```

1 int kadane(int arr[], int l) {
2
3     int soma, total;
4     soma = total = arr[0];
5
6     for(int i = 1; i < l; i++) {
7         soma = max(arr[i], arr[i] + soma);
8         if(soma > total)
9             total = soma;
10    }
11    return total;
12
13 }

```

9.6. KADANE 2D -  $O(n^3)$ 

```

1 // Program to find maximum sum subarray in a given 2D array
2 #include <stdio.h>
3 #include <string.h>
4 #include <limits.h>
5 int mat[1001][1001]
6 int ROW = 1000, COL = 1000;
7
8 // Implementation of Kadane's algorithm for 1D array. The function
9 // returns the maximum sum and stores starting and ending indexes of the
10 // maximum sum subarray at addresses pointed by start and finish pointers
11 // respectively.
12 int kadane(int* arr, int* start, int* finish, int n) {
13     // initialize sum, maxSum and
14     int sum = 0, maxSum = INT_MIN, i;
15
16     // Just some initial value to check for all negative values case
17     *finish = -1;
18
19     // local variable
20     int local_start = 0;
21
22     for (i = 0; i < n; ++i) {
23         sum += arr[i];
24         if (sum < 0) {
25             sum = 0;
26             local_start = i+1;
27         }
28         else if (sum > maxSum){
29             maxSum = sum;
30             *start = local_start;
31             *finish = i;
32         }
33     }
34
35     // There is at-least one non-negative number
36     if (*finish != -1)
37         return maxSum;
38
39     // Special Case: When all numbers in arr[] are negative
40     maxSum = arr[0];
41 }

```



```

42     *start = *finish = 0;
43
44     // Find the maximum element in array
45     for (i = 1; i < n; i++) {
46         if (arr[i] > maxSum) {
47             maxSum = arr[i];
48             *start = *finish = i;
49         }
50     }
51     return maxSum;
52 }
53
54 // The main function that finds maximum sum rectangle in mat[][]
55 int findMaxSum() {
56     // Variables to store the final output
57     int maxSum = INT_MIN, finalLeft, finalRight, finalTop, finalBottom;
58
59     int left, right, i;
60     int temp[ROW], sum, start, finish;
61
62     // Set the left column
63     for (left = 0; left < COL; ++left) {
64         // Initialize all elements of temp as 0
65         for (int i = 0; i < ROW; i++)
66             temp[i] = 0;
67
68         // Set the right column for the left column set by outer loop
69         for (right = left; right < COL; ++right) {
70             // Calculate sum between current left and right for every row 'i'
71             for (i = 0; i < ROW; ++i)
72                 temp[i] += mat[i][right];
73
74             // Find the maximum sum subarray in temp[]. The kadane()
75             // function also sets values of start and finish. So 'sum' is
76             // sum of rectangle between (start, left) and (finish, right)
77             // which is the maximum sum with boundary columns strictly as
78             // left and right.
79             sum = kadane(temp, &start, &finish, ROW);
80
81             // Compare sum with maximum sum so far. If sum is more, then
82             // update maxSum and other output values
83             if (sum > maxSum) {
84                 maxSum = sum;
85                 finalLeft = left;
86                 finalRight = right;
87                 finalTop = start;
88                 finalBottom = finish;
89             }
90         }
91     }
92
93     return maxSum;
94     // Print final values
95     printf("Top, Left) (%d, %d)\n", finalTop, finalLeft);
96     printf("Bottom, Right) (%d, %d)\n", finalBottom, finalRight);
97     printf("Max sum is: %d\n", maxSum);
98 }

```

### 9.7. KADANE (SEGMENT TREE) - (QUERY IN RANGE $O(\log n)$ )

```

1 struct node {
2     int pref, suf, tot, best;
3     node () {}

```

```

4     node(int pref, int suf, int tot, int best) : pref(pref), suf(suf),
5         tot(tot), best(best) {}
6
7     node tree[500000];
8     int arr[100000];
9
10    node query(int l, int r, int i, int j, int pos) {
11
12        node x;
13
14        if (l > r || l > j || r < i) {
15            return node(-INF, -INF, -INF, -INF);
16        }
17
18        if (i <= l && r <= j) {
19            return node(tree[pos].pref, tree[pos].suf, tree[pos].tot, tree[pos].best);
20        }
21
22        int mid = (l + r) / 2;
23        node left = query(l, mid, i, j, 2*pos+1), right = query(mid+1, r, i, j, 2*pos+2);
24        x.pref = max({left.pref, left.tot, left.tot + right.pref});
25        x.suf = max({right.suf, right.tot, right.tot + left.suf});
26        x.tot = left.tot + right.tot;
27        x.best = max({left.best, right.best, left.suf + right.pref});
28        // imprimir.best
29        return x;
30    }
31
32    void build(int l, int r, int pos) {
33
34        if (l == r) {
35            tree[pos] = node(arr[l], arr[l], arr[l], arr[l]);
36            return;
37        }
38
39        int mid = (l + r) / 2;
40        build(l, mid, 2*pos+1); build(mid+1, r, 2*pos+2);
41        l = 2*pos+1, r = 2*pos+2;
42        tree[pos].pref = max({tree[l].pref, tree[l].tot, tree[l].tot +
43            tree[r].pref});
44        tree[pos].suf = max({tree[r].suf, tree[r].tot, tree[r].tot + tree[l].suf});
45        tree[pos].tot = tree[l].tot + tree[r].tot;
46        tree[pos].best = max({tree[l].best, tree[r].best, tree[l].suf +
47            tree[r].pref});
48    }

```

### 9.8. COMPRESSAO DE PONTOS

```

1 map<int, int> rev;
2
3 vector<int> compress(vector<int> &arr) {
4
5     for (int x : arr) {
6         s1.insert(x);
7     }
8
9     vector<int> aux;
10    for (int x : s1) aux.pb(x);
11    for (int i=0; i<n; i++){
12        int id = lower_bound(aux.begin(), aux.end(), arr[i]) - aux.begin();
13        // rev[id] = arr[i];
14        arr[i] = id;
15    }

```

15 }  
16 }

### 9.9. TORRE DE HANOI - $O(2^{n-1})$

```
1 #include <stdio.h>
2
3 // C recursive function to solve tower of hanoi puzzle
4 void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
5     if (n == 1) {
6         printf("\n Move disk 1 from rod %c to rod %c", from_rod, to_rod);
7         return;
8     }
9     towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
10    printf("\n Move disk %d from rod %c to rod %c", n, from_rod, to_rod);
11    towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
12 }
13
14 int main() {
15     int n = 4; // Number of disks
16     towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
17     return 0;
18 }
```

### 9.10. FIBONACCI MATRIX EXPONENTIATION - $O(\log n)$

```
1 int fib (int n) {
2     long long fib[2][2] = {{1,1},{1,0}};
3     int ret[2][2] = {{1,0},{0,1}};
4     int tmp[2][2] = {{0,0},{0,0}};
5     int i,j,k;
6     while(n) {
7         if(n&1) {
8             memset(tmp,0,sizeof tmp);
9             for(i=0; i<2; i++)
10                for(j=0; j<2; j++)
11                    for(k=0; k<2; k++)
12                        tmp[i][j] = (tmp[i][j] + ret[i][k]*fib[k][j]);
13             for(i=0; i<2; i++)
14                 for(j=0; j<2; j++)
15                     ret[i][j] = tmp[i][j];
16         }
17         memset(tmp,0,sizeof tmp);
18         for(i=0; i<2; i++)
19             for(j=0; j<2; j++)
20                 for(k=0; k<2; k++)
21                     tmp[i][j] = (tmp[i][j] + fib[i][k]*fib[k][j]);
22         for(i=0; i<2; i++)
23             for(j=0; j<2; j++)
24                 fib[i][j] = tmp[i][j];
25         n/=2;
26     }
27     return (ret[0][1]);
28 }
```

### 9.11. 2-SAT PROBLEM - $O(V+E)$

```
1 int comp[312345];
2 vector<int> adj[312345];
3 // grafo reverso de adj
4 vector<int> trans[312345];
5 vector<int> scc[312345];
```

```
6 int V;
7 // Nao esquecer de preencher o V com o tamanho dos vertices
8
9 void dfsTrans(int u, int id, int comp[]) {
10     comp[u] = id;
11     scc[id].push_back(u);
12
13     for (int v: trans[u])
14         if (!comp[v])
15             dfsTrans(v, id, comp);
16 }
17
18 void getTranspose() {
19     for (int u = 0ll; u < V; u++)
20         for(int v: adj[u])
21             trans[v].push_back(u);
22 }
23
24 void dfsFillOrder(int u, int comp[], stack<int> &Stack) {
25     comp[u] = true;
26
27     for(int v: adj[u])
28         if(!comp[v])
29             dfsFillOrder(v, comp, Stack);
30
31     Stack.push(u);
32 }
33
34 // The main function that finds and prints all strongly connected
35 // components
36 void computeSCC() {
37
38     stack<int> Stack;
39     // Fill vertices in stack according to their finishing times
40     for(int i = 0ll; i < V; i++)
41         if(comp[i] == false)
42             dfsFillOrder(i, comp, Stack);
43
44     // Create a reversed graph
45     getTranspose();
46
47     memset(comp, 0ll, sizeof(comp));
48
49     // Now process all vertices in order defined by Stack
50     int id = 1ll;
51     while(Stack.empty() == false) {
52         int v = Stack.top();
53         Stack.pop();
54
55         if(comp[v] == false)
56             dfsTrans(v, id++, comp);
57     }
58 }
59
60 // (X v Y) = (X -> ~Y) and (~X -> Y)
61 void orEdge(int u, int v, int idxu, int idxv, int n) {
62     // idx represents the index of the atoms
63     // Example there are atoms X, ~X, Y, ~Y
64     // Then for Clause (X v ~Y) idxu = 0ll and idxv = 3ll
65     idxv ^= 1ll;
66     adj[u + idxu*n].pb(v + idxv*n);
67     idxu ^= 1ll, idxv ^= 1ll;
68     adj[v + idxv*n].pb(u + idxu*n);
69 }
70 }
```

```
71 // (X xor Y) = (X v Y) and (~X v ~Y)
72 // for this function the result is always 0 1 or 1 0
73 void xorEdge(int u, int v, int idxu, int idxv, int n) {
74     orEdge(u, v, idxu, idxv, n);
75     orEdge(u, v, idxu^111, idxv^111, n);
76 }
77
78 bool check(int n) {
79
80     for(int i = 011; i < V; i += 211*n) {
81         for(int j = i, k = 011; k < n; k++, j++) {
82             if(comp[j] == comp[j+n] && comp[j] != 0) {
83                 return false;
84             }
85         }
86     }
87     return true;
88 }
89
90
91 signed main () {
92
93     computeSCC();
94
95     if(check(n)) {
96         cout << "YES" << endl;
97     } else
98         cout << "NO" << endl;
99 }
```