



# Notebook - Maratona de Programação

Py tá O(N)

## Contents

<b>1</b>	<b>Algoritmos</b>	<b>2</b>	5.4	Mdc . . . . .	11
1.1	Busca Binaria . . . . .	2	5.5	Mmc . . . . .	11
1.2	Busca Binaria Double . . . . .	2	5.6	Pa . . . . .	11
1.3	Busca Binaria Resposta . . . . .	2	5.7	Primos . . . . .	11
1.4	Delta . . . . .	3	<b>6</b>	<b>Template</b>	<b>11</b>
1.5	Fast Exponentiaton . . . . .	3	6.1	Template . . . . .	11
1.6	Psum . . . . .	3			
<b>2</b>	<b>DP</b>	<b>3</b>			
2.1	Dp . . . . .	3			
2.2	Knapsack . . . . .	3			
2.3	Mochila Iterativa . . . . .	4			
2.4	Mochila Recursiva . . . . .	4			
<b>3</b>	<b>ED</b>	<b>4</b>			
3.1	Bit . . . . .	4			
3.2	Merge Sort . . . . .	4			
3.3	Segtrees 1 . . . . .	5			
3.4	Segtrees 2 . . . . .	5			
3.5	Segtrees Lazy Propagation . . . . .	6			
<b>4</b>	<b>Grafos</b>	<b>6</b>			
4.1	Bellman Ford . . . . .	6			
4.2	Bfs . . . . .	7			
4.3	Dfs . . . . .	7			
4.4	Diametro Arvore Bfs . . . . .	7			
4.5	Diametro Arvore Dfs . . . . .	8			
4.6	Dijkstra . . . . .	8			
4.7	Dsu . . . . .	9			
4.8	Floyd Warshall . . . . .	9			
4.9	Kruskall . . . . .	9			
4.10	Lca . . . . .	10			
<b>5</b>	<b>Math</b>	<b>10</b>			
5.1	Combinatoria . . . . .	10			
5.2	Divisores . . . . .	10			
5.3	Fatora . . . . .	11			

# 1 Algoritmos

## 1.1 Busca Binaria

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 bool check(int valor, int x) {
5     return valor <= x;
6 }
7
8 int bb(int a, int b, int x){
9     int l = a;
10    int r = b;
11    while (l < r) {
12        int mid = (l + r) / 2;
13        if (check(mid, x)) r = mid;
14        else l = mid + 1;
15    }
16    return l;
17 }
18
19 bool check(int valor) {
20     return valor <= 10;
21 }
22
23 int bb_menor(int a, int b){
24     int l = a;
25     int r = b;
26     while (l < r) {
27         int mid = (l + r) / 2;
28         if (check(mid)) r = mid;
29         else l = mid + 1;
30     }
31
32     return l;
33 }
34
35
36 int bb_maior(int a, int b){
37     int l = a;
38     int r = b;
39     while (l < r) {
40         int mid = (l + r) / 2;
41         if (!check(mid)) r = mid;
42         else l = mid + 1;
43     }
44 }
45 }
```

## 1.2 Busca Binaria Double

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 typedef long long ll;
5 const int MAX = 1e5+1;
6 const double EPS = 0.0000001;
7
8 vector<int> v(100001);
9 int n;
10 ll check(double x){
11     ll sum = 0;
12     for(int i=0; i<n; i++){
13         sum += (v[i]/x);
14     }
15     return sum;
16 }
17
18 int main(){
19
20     int k;
```

```
21     cin>>n>>k;
22
23     for(int i=0; i<n; i++)cin>>v[i];
24
25     double l=0.0000000, r=10000000.0000000;
26     double mid;
27     while(r-l>EPS){
28         mid = (double)((l + r)/2);
29         if (check(mid)>=k){
30             l=mid;
31         }
32         else{
33             r = mid;
34         }
35     }
36
37     cout<<fixed<<setprecision(7)<<mid<<endl;
38
39     return 0;
40 }
```

## 1.3 Busca Binaria Resposta

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define loop(i,a,n) for(int i=a; i < n; i++)
5
6 ll upperbound(ll maior, ll k, vector<ll> tabuas){
7     ll mid = 0, l = 0, r = maior, count = 300;
8     ll aux;
9     while((l < r) && (count--)){
10         aux = 0;
11         mid = (l+r)/2;
12         loop(i,0,tabuas.size()){
13             if(mid > 0){aux += (tabuas[i]/mid);}
14         }
15         if(aux >= k){l = mid;}
16         else{r = mid;}
17     }
18
19     ll aux_2 = 0;
20     loop(i,0,tabuas.size()){
21         aux_2 += (tabuas[i]/(mid+1));
22     }
23     if(aux_2 >= k){return mid+1;}
24
25     if(aux < k){
26         int aux_2 = 0;
27         loop(i,0,tabuas.size()){
28             if(mid - 1 > 0){aux_2 += (tabuas[i]/(mid
29 -1))};
30         }
31         if(aux_2 >= k){return mid-1;}
32     }
33
34     return mid;
35 }
36
37 int main(){
38     ios::sync_with_stdio(false);
39     cin.tie( NULL);
40     cout.tie(NULL);
41     int n; cin >> n;
42     ll k; cin >> k;
43     vector<ll> tabuas(n);
44     ll maior = 0;
45     loop(i,0,n){
46         cin >> tabuas[i];
47         maior = max(maior,tabuas[i]);
48     }
49     cout << upperbound(maior,k,tabuas);
50 }
```

## 1.4 Delta

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     int n, q;
6     cin >> n >> q;
7     vector<int> v(n,0);
8     vector<int> delta(n+2, 0);
9
10    while(q--){
11        int l, r, x;
12        cin >> l >> r >> x;
13        delta[l] += x;
14        delta[r+1] -= x;
15    }
16
17    int atual = 0;
18    for(int i=0; i < n; i++){
19        atual += delta[i];
20        v[i] = atual;
21    }
22
23    for(int i=0; i < n; i++){
24        cout << v[i] << " ";
25    }
26    cout << endl;
27
28    return 0;
29 }
```

## 1.5 Fast Exponentiation

```
1 int fast_exp(int base, int e){
2     if(e == 0) return 1;
3     if(e % 2) return base * fast_exp(base * base, e/2);
4     ;
5     else return fast_exp(base * base, e/2);
6 }
```

## 1.6 Psum

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define input(x) for (auto &it : x) cin >> it
5 typedef long long ll;
6 vector<ll> psum(1e5);
7
8 int solve(int l, int r){
9     if(l==0) return psum[r];
10    else return psum[r] - psum[l-1];
11 }
12
13 int main(){
14
15     int n, q;
16     cin>>n>>q;
17
18     vector<int> v(n);
19     input(v);
20     for(int i=0; i<n; i++){
21         if(i==0)psum[i] = v[i];
22         else psum[i] = psum[i-1] + v[i];
23     }
24     while(q--){
25         int l, r;
26         cin>>l>>r;
27
28         cout<<(solve(l,r))<<endl;
29     }
```

```
30
31     return 0;
32 }
```

## 2 DP

### 2.1 Dp

```
1 // DP - Dynamic Programming
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 typedef long long ll;
7 const int MAX = 110;
8
9 int n;
10 int tab[MAX];
11 vector<int> v;
12
13 ll dp(int i){
14     if(i>=n) return 0;
15     if(tab[i] != -1) return tab[i];
16
17     int pega = v[i] + dp(i+2);
18     int npega = dp(i+1);
19
20     tab[i] = max(pega, npega);
21     return tab[i];
22 }
23
24 int main(){
25     memset(tab, -1, sizeof(tab));
26     cin>>n;
27
28     v.assign(n, 0);
29
30     cout<<dp(0)<<endl;
31
32     return 0;
33 }
```

### 2.2 Knapsack

```
1 /* A Naive recursive implementation of
2 0-1 Knapsack problem */
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 // Returns the maximum value that
7 // can be put in a knapsack of capacity W
8 int knapSack(int W, int wt[], int val[], int n)
9 {
10
11     // Base Case
12     if (n == 0 || W == 0)
13         return 0;
14
15     // If weight of the nth item is more
16     // than Knapsack capacity W, then
17     // this item cannot be included
18     // in the optimal solution
19     if (wt[n - 1] > W)
20         return knapSack(W, wt, val, n - 1);
21
22     // Return the maximum of two cases:
23     // (1) nth item included
24     // (2) not included
25     else
26         return max(val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1), knapSack(W, wt, val, n - 1));
27 }
```

```

27 }
28
29 // Driver code
30 int main()
31 {
32     int val[] = { 60, 100, 120 };
33     int wt[] = { 10, 20, 30 };
34     int W = 50;
35     int n = sizeof(val) / sizeof(val[0]);
36     cout << knapSack(W, wt, val, n);
37     return 0;
38 }
39
40 // This code is contributed by rathbhupendra

```

## 2.3 Mochila Iterativa

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int maxn = 110, maxp = 1e5+10;
5 const long long inf = 0x3f3f3f3f3f3f3f3f; // ~= 10^18
6
7 int v[maxn], p[maxn];
8 long long dp[maxn][maxp];
9
10 int main() {
11     int n, C; scanf("%d %d", &n, &C);
12     for(int i = 1; i <= n; i++)
13         scanf("%d %d", &p[i], &v[i]);
14
15     long long ans = 0;
16     // inicializando o vetor
17     for(int i = 1; i <= n; i++)
18         for(int P = p[i]; P <= C; P++)
19             dp[i][P] = -inf;
20     // definindo o caso base
21     dp[0][0] = 0;
22
23     for(int i = 1; i <= n; i++) {
24         for(int P = 0; P <= C; P++) {
25             dp[i][P] = dp[i-1][P];
26             if(P >= p[i])
27                 dp[i][P] = max(dp[i][P], dp[i-1][P-p[i]] + v[i]);
28             ans = max(ans, dp[i][P]);
29         }
30     }
31
32     printf("%lld\n", ans);
33 }

```

## 2.4 Mochila Recursiva

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int maxn = 110, maxp = 1e5+10;
5
6 int v[maxn], p[maxn], n;
7 long long dp[maxn][maxp];
8 bool vis[maxn][maxp];
9
10 long long solve(int i, int P) {
11     if(i == n+1) return 0; // caso base, nao ha mais
12     // itens para se considerar
13     if(vis[i][P]) return dp[i][P];
14     vis[i][P] = 1;
15
16     // primeira possibilidade, ão adicionar o
17     // elemento
18     dp[i][P] = solve(i+1, P);
19
20     // segunda possibilidade, adicionar o elemento.
21     // Lembrar de tirar o maximo com o valor ja
22     // calculado da primeira possibilidade
23     if(P >= p[i])
24         dp[i][P] = max(dp[i][P], solve(i+1, P - p[i])
25             + v[i]);
26
27     return dp[i][P];
28 }
29
30 int main() {
31     int C; scanf("%d %d", &n, &C);
32     for(int i = 1; i <= n; i++)
33         scanf("%d %d", &p[i], &v[i]);
34     printf("%lld\n", solve(1, C));
35 }

```

## 3 ED

### 3.1 Bit

```

1 // Bitwise Operations
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 // Verificar se o bit esta ligado
7 bool isSet(int bitPosition, int number) {
8     bool ret = ((number & (1 << bitPosition)) != 0);
9     return ret;
10 }
11
12 // Ligar o bit
13 bool setBit(int bitPosition, int number) {
14     return (number | (1 << bitPosition));
15 }
16
17 // Gerando todos os subconjuntos de um conjunto em
18 // binario
19 void possibleSubsets(char S[], int N) {
20     for(int i = 0; i < (1 << N); ++i) { // i = [0, 2^N - 1]
21         for(int j = 0; j < N; ++j)
22             if(i & (1 << j)) // se o j-esimo bit de
23                 // i esta setado, printamos S[j]
24                 cout << S[j] << " ";
25         cout << endl;
26     }
27 }

```

### 3.2 Merge Sort

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define INF 1000000000
5
6 void merge_sort(vector<int> &v){
7     if(v.size()==1) return;
8
9     vector<int> v1, v2;
10
11     for(int i=0; i<v.size()/2; i++) v1.push_back(v[i]);
12     for(int i=v.size()/2; i<v.size(); i++) v2.push_back(v[i]);
13
14     merge_sort(v1);
15     merge_sort(v2);
16 }

```

```

16
17 v1.push_back(INF);
18 v2.push_back(INF);
19
20 int ini1=0, ini2=0;
21
22 for(int i=0; i<v.size(); i++){
23     if(v1[ini1]<v2[ini2]){
24         v[i] = v1[ini1];
25         ini1++;
26     }else{
27         v[i] = v2[ini2];
28         ini2++;
29     }
30 }
31 return;
32 }

```

### 3.3 Segtree 1

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 class SegTree{
4     vector<int> st;
5     int size;
6
7     int el_neutro = -(1e9 + 7);
8
9     int f(int a, int b){
10         return max(a,b);
11     }
12
13     int query(int sti, int stl, int str, int l, int r
14 ){
15         //0 nodo esta fora do intervalo que estamos
16         interessados, retorne o elemento neutro que nao
17         afeta a consulta
18         if(str < l || r < stl)
19             return el_neutro;
20
21         // 0 nodo esta completamente incluído no
22         intervalos que estamos interessados, retorne a
23         informacao contida naquele nodo.
24         if(stl >= l and str <= r)
25             return st[sti];
26
27         // Se chegarmos aqui, eh porque esse Nodo esta
28         parcialmente contido no intervalo que estamos
29         interessados. Entao, continuamos procurando nos
30         filhos.
31         int mid = (str+stl)/2;
32
33         return f(query(sti*2+1,stl,mid,l,r),query(sti
34 *2+2,mid+1,str,l,r));
35     }
36
37     void update(int sti, int stl, int str, int i, int
38     amm){
39         // Chegamos no indice que queremos, vamos
40         atualizar o valor
41         if(stl == i and str == i){
42             st[sti] = amm;
43             return;
44         }
45         // 0 intervalo que estamos nao contem o
46         indice que queremos atualizar, retorne
47         if(stl > i or str < i)
48             return;
49
50         // 0 intervalo contem o indice, mas temos que
51         chegar no nodo especifico, recurse para os
52         filhos.
53         int mid = (stl + str)/2;

```

```

40         update(sti*2+1,stl,mid,i,amm);
41         update(sti*2+2,mid+1,str,i,amm);
42         // Apos os filhos mais em baixo, precisamos
43         atualizar o valor desse nodo
44         st[sti] = f(st[sti*2+1],st[sti*2+2]);
45     }
46     public:
47         SegTree(int n): st(4*n,0){size = n;}
48         int query(int l, int r){return query(0,0,
49         size-1,l,r);}
50         void update(int i, int amm){update(0,0,size
51         -1,i,amm);}
52 };
53
54 int main(){
55     vector<int> v;
56     SegTree st(v.size());
57     for(int i = 0; i< v.size();i++){
58         st.update(i,v[i]);
59     }

```

### 3.4 Segtree 2

```

1
2 #include <bits/stdc++.h>
3 #define ff first
4 #define ss second
5 #define ll long long
6 #define pb push_back
7 #define sws ios_base::sync_with_stdio(false);cin.tie(
8     NULL);cout.tie(NULL);
9
10 using namespace std;
11
12 const int MAX = 1e5; // tamanho maximo do vetor
13 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
14
15 // End Template //
16
17 ll tree[4*MAX], vet[MAX];
18
19 void build(int l, int r, int no){
20     if(l==r){
21         tree[no] = vet[l];
22         return;
23     }
24     int mid = (l+r)/2;
25     build(l, mid, 2*no);
26     build(mid+1, r, 2*no+1);
27
28     tree[no] = tree[2*no] + tree[2*no+1];
29 }
30
31 void update(int id, int x, int l, int r, int no){
32     if(l==r){
33         tree[no] = x;
34         return;
35     }
36     int mid = (l+r)/2;
37     if(id<=mid)
38         update(id, x, l, mid, 2*no); // esquerda
39     else
40         update(id, x, mid+1, r, 2*no+1);
41
42     tree[no] = tree[2*no] + tree[2*no+1];
43 }
44
45 ll query(int A, int B, int l, int r, int no){
46     // caso 1
47     if(B<l or r<A) return 0;

```

```

48 // caso 2
49 if(A<=l and r<=B) return tree[no];
50 // caso 3
51 int mid = (l+r)/2;
52 ll sumLeft = query(A, B, l, mid, 2*no);
53 ll sumRight = query(A, B, mid+1, r, 2*no+1);
54
55 return sumLeft + sumRight;
56 }
57
58
59 int32_t main()
60 {sws;
61
62     int n, m, opt, id, v, l, r;
63     cin >> n >> m;
64     for(int i=0;i<n;i++){
65         cin >> vet[i];
66
67         build(0, n-1, 1);
68
69         for(int i=0;i<m;i++){
70             cin >> opt;
71             if(opt==1){ // update
72                 cin >> id >> v;
73                 update(id, v, 0, n-1, 1);
74             }else{ // query
75                 cin >> l >> r;
76                 cout << query(l, r-1, 0, n-1, 1) << endl;
77             }
78         }
79
80
81         return 0;
82     }
83 }

```

### 3.5 Segtree Lazy Propagation

```

1 #include <bits/stdc++.h>
2 #define ll long long
3
4 using namespace std;
5
6 const int MAX = 1e5; // tamanho maximo do vetor
7 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
8
9 // End Template //
10
11 vector<ll> lazy(4*MAX, -1);
12 ll tree[4*MAX], vet[MAX];
13 int N;
14
15 ll merge(ll a, ll b){
16     return a + b;
17 }
18
19 void build(int l=0, int r=N-1, int no=1){
20     if(l==r){
21         tree[no] = vet[l];
22         return;
23     }
24     int mid = (l+r)/2;
25     build(l, mid, 2*no);
26     build(mid+1, r, 2*no+1);
27
28     tree[no] = merge(tree[2*no], tree[2*no+1]);
29 }
30
31 void prop(int l, int r, int no){
32     if(lazy[no]!=-1){
33         tree[no] = (r-l+1)*lazy[no];
34         if(l!=r){

```

```

35             lazy[2*no] = lazy[2*no+1] = lazy[no];
36         }
37         lazy[no] = -1;
38     }
39 }
40
41 void update(int A, int B, int x, int l=0, int r=N-1,
42             int no=1){
43     prop(l, r, no);
44     // caso 1
45     if(B<l or r<A) return;
46     // caso 2
47     if(A<=l and r<=B){
48         lazy[no] = x;
49         prop(l, r, no);
50         return;
51     }
52     // caso 3
53     int mid = (l+r)/2;
54
55     update(A, B, x, l, mid, 2*no);
56     update(A, B, x, mid+1, r, 2*no+1);
57
58     tree[no] = merge(tree[2*no], tree[2*no+1]);
59 }
60 ll query(int A, int B, int l=0, int r=N-1, int no=1){
61     prop(l, r, no);
62     // caso 1
63     if(B<l or r<A) return 0;
64     // caso 2
65     if(A<=l and r<=B) return tree[no];
66     // caso 3
67     int mid = (l+r)/2;
68
69     return merge(query(A, B, l, mid, 2*no),
70                 query(A, B, mid+1, r, 2*no+1));
71 }
72
73
74 int32_t main()
75 {
76
77     int Q, opt, a, b, l, r, k;
78     cin >> N >> Q;
79     for(int i=0;i<N;i++){
80         cin >> vet[i];
81
82         build();
83
84         for(int i=0;i<Q;i++){
85             cin >> opt;
86             if(opt==1){ // update
87                 cin >> a >> b >> k;
88                 a--;b--;
89                 update(a, b, k);
90             }else{ // query
91                 cin >> l >> r;
92                 l--;r--; // indice indexado em 0
93                 cout << query(l, r) << endl;
94             }
95         }
96
97         return 0;
98 }

```

## 4 Grafos

### 4.1 Bellman Ford

1 /\*

```

2 Algoritmo de busca de caminho minimo em um digrafo (
   grafo orientado ou dirigido) ponderado, ou seja,
   cujas arestas tem peso, inclusive negativo.
3 */
4
5 #include <bits/stdc++.h>
6 using namespace std;
7
8 // pode usar uma tuple
9 struct Edge {
10     // [de onde vem, pra onde vai, peso]
11     int from, to, custo;
12
13     Edge(int a=0, int b=0, int c=0 ){
14         from = a;
15         to=b;
16         custo = c;
17     }
18 };
19
20 int main(){
21     int n, m;
22     cin>>n>>m;
23     vector<Edge> arestas(m);
24
25     for(int i=0; i<m; i++){
26         int a, b, c;
27         cin>>a>>b>>c;
28         arestas[i] = Edge(a, b, c);
29     }
30
31     vector<int> distancia(n + 1, 100000000);
32     distancia[1]=0;
33     for(int i=0; i<n-1; i++){
34         for(auto aresta : arestas){
35             if (distancia[aresta.from] + aresta.custo
36                 < distancia[aresta.to]){
37                 distancia[aresta.to] = distancia[
38                     aresta.from] + aresta.custo;
39             }
40         }
41     }
42
43     for(int i=1; i<=n; i++){
44         cout<<"Distancia ate o vertice "<<i<<" "<<
45         distancia[i]<<endl;
46     }
47
48     return 0;
49 }

```

## 4.2 Bfs

```

1 #include <bits/stdc++.h>[]
2 using namespace std;
3
4 //-----
5 #define MAXN 50050
6
7 int n, m;
8 bool visited[MAXN];
9 vector<int> lista[MAXN];
10 //-----
11
12 void bfs(int x){
13
14     queue<int> q;
15     q.push(x);
16     while(!q.empty()){
17         int v = q.front();
18         q.pop();

```

```

19         visited[v] = true;
20         for(auto i : lista[v]){
21             if(!visited[i]){
22                 q.push(i);
23             }
24         }
25     }
26 }

```

## 4.3 Dfs

```

1 #include <iostream>
2 #include <vector>
3 #include <stack>
4
5 using namespace std;
6
7 //-----
8 #define MAXN 50050
9
10 int n, m;
11 bool visited[MAXN];
12 vector<int> lista[MAXN];
13 //-----
14
15 void dfs(int x){
16     visited[x] = true;
17     for(auto i : lista[x]){
18         if(!visited[i]){
19             dfs(i);
20         }
21     }
22 }
23
24 void dfsStack(int x){
25     stack<int> s;
26     s.push(x);
27     while(!s.empty()){
28         int v = s.top();
29         s.pop();
30         visited[v] = true;
31         for(auto i : lista[v]){
32             if(!visited[i]){
33                 s.push(i);
34             }
35         }
36     }
37 }

```

## 4.4 Diametro Arvore Bfs

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 typedef long long ll;
5 typedef vector<int> vi;
6 typedef pair<int,int> pii;
7 const int MAX = 1e5+10;
8 const ll MOD = 1e9+5;
9
10 vector<int> adj(MAX);
11
12 pair<int, int> bfs(int s, int N){
13
14     vi dist(N + 1, MAX); dist[s] = 0;
15     queue<int> q; q.push(s);
16     int last = s;
17
18     while(!q.empty()){
19         auto u = q.front(); q.pop();
20         last = u;
21

```

```

22         for(auto v: adj[u]){
23             if(dist[v]==MAX){
24                 dist[v]=dist[u]+1;
25                 q.push(v);
26             }
27         }
28     }
29
30     return {last, dist[last]};
31 }
32
33 int diameter(int N){
34     auto [v, _] = bfs(1, N);
35     auto [w, D] = bfs(v, N);
36
37     return D;
38 }

```

## 4.5 Diametro Arvore Dfs

```

1 // DIAMETRO ARVORE - DFS
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 typedef long long ll;
7 typedef vector<int> vi;
8 typedef pair<int,int> pii;
9 const int MAX = 1e5+10;
10 const ll MOD = 1e9+5;
11 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
12
13 int to_leaf[MAX];
14 int max_length[MAX];
15 int dist[MAX];
16
17 vector<int> adj(MAX);
18 /*
19 void dfs(int u, int p, vector<int> &dist){
20     for(auto [v, w] : adj[u]){
21         if(v!=p){
22             dist[v] = dist[u] + w;
23             dfs(v, u, dist);
24         }
25     }
26 }
27
28 int solve(int n){
29     vector<int> dist(n+1, 0);
30
31     dfs(0, -1, dist);
32
33     auto v = (int)(max_element(dist.begin(), dist.end()
34     ()) - dist.begin());
35
36     dist[v] = 0;
37     dfs(v, -1, dist);
38
39     return *max_element(dist.begin(), dist.end());
40 }*/
41 void dfs(int u, int p){
42     vi ds;
43
44     for(auto v: adj[u]){
45         if(v==p) continue;
46
47         dfs(v, u);
48         ds.pb(to_leaf[v]);
49     }
50     sort(ds.begin(), ds.end());
51
52     to_leaf[u] = ds.empty() ? 0 : ds.back() + 1;

```

```

53
54     auto N = ds.size();
55
56     switch(N){
57         case 0:
58             max_length[u]=0;
59             break;
60         case 1:
61             max_length[u] = ds.back() + 1;
62             break;
63         default:
64             max_length[u] = ds[N-1] + ds[N-2] + 2;
65     }
66 }
67
68 int diameter(int root, int N){
69     dfs(root, 0);
70
71     int d=0;
72
73     for(int u=1; u<=N; u++){
74         d= max(d, max_length[u]);
75     }
76 }

```

## 4.6 Dijkstra

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define ll long long
4
5 const int N = 100005;
6 const ll oo = 1e18;
7
8 ll d[N]; // vetor onde guardamos as distancias
9
10 int n; // numeros de vertices
11
12 // lista de adjacencias guarda
13 // pair <vertice para onde a aresta vai, peso da
14 // aresta>
15 vector<pair<int, ll>> g[N];
16
17 void dijkstra(int start){
18     // inicialmente a distancia do vertice
19     // start para todo os outros eh infinita
20     for(int u = 1; u <= n; u++)
21         d[u] = oo;
22
23     // fila de prioridade de pair<ll, int>, mas que o
24     // menor pair fica no topo da fila
25     // guardamos um pair <distancia ate o vertice,
26     // vertice>
27     // assim o topo da fila sempre eh o vertice com
28     // menor distancia
29     priority_queue<pair<ll, int>, vector<pair<ll, int>>,
30     greater<pair<ll, int>>> > pq;
31
32     d[start] = 0;
33     pq.emplace(d[start], start);
34
35     ll dt, w;
36     int u, v;
37     while(!pq.empty()){
38         tie(dt, u) = pq.top(); pq.pop();
39         if(dt > d[u]) continue;
40         for(auto edge : g[u]){
41             tie(v, w) = edge;
42
43             // se a distancia ate o u somado com o
44             // peso

```



```

42         // da aresta eh menor do que a distancia
ate o v que
43         // tinhamos antes, melhoramos a distancia
ate o v
44         if(d[v] > d[u] + w){
45             d[v] = d[u] + w;
46             pq.emplace(d[v], v);
47         }
48     }
49 }
50 }
51
52 int main(){
53
54     // le o input, qnt de vertices, arestas
55     // e vertice inicial(start)
56     int start = 0; // inicial
57     dijkstra(start);
58
59     for(int u = 1; u <= n; u++){
60         printf("Distancia de %d para %d: %lld\n",
start, u, d[u]);
61     }
62 }
63 }

```

## 4.7 Dsu

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MAX = 1e5+10;
5
6 int parent[MAX];
7
8 void make(int v){
9     parent[v] = v;
10 }
11
12 int find(int v){
13     if (v == parent[v])
14         return v;
15     return parent[v] = find(parent[v]);
16 }
17
18 void _union(int a, int b){
19     a = find(a);
20     b = find(b);
21     if (a != b)
22         parent[b] = a;
23 }
24
25 int main(){
26
27     return 0;
28 }

```

## 4.8 Floyd Warshall

```

1 /*
2 Algoritmo de caminho mais curto com todos os pares, o
que significa que calcula o caminho mais curto
entre todos os pares de nós.
3 */
4
5 #include <bits/stdc++.h>
6 using namespace std;
7
8 const int oo = 100000000; // infinito
9
10 int main(){
11

```

```

12 int n, m;
13 cin>>n>>m;
14
15 vector<vector<int>> dist(n+1, vector<int> (n+1));
16
17 for(int i=0; i<n+1; i++){
18     for(int j=0; j<n+1; j++){
19         dist[i][j] = oo;
20     }
21 }
22
23 for(int i=0; i<n +1; i++){
24     dist[i][i]=0;
25 }
26
27 for(int i=0; i<m; i++){
28     int começa, termina, custo;
29     cin>>começa>>termina>>custo;
30
31     // grafo direcionado
32     dist[começa][termina] = custo;
33 }
34
35 for(int k=1; k<=n; k++){ // intermediario
36     for(int i=1; i<=n; i++){
37         for(int j=1; j<=n; j++){
38             //(i,k,j) = ir de i pra j passando
por k;
39
40             // relaxar distancia de i pra j
41             dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
42         }
43     }
44 }
45     return 0;
46 }

```

## 4.9 Kruskall

```

1 /*
2 Busca uma árvore geradora ínnima para um grafo conexo
com pesos.
3 */
4
5 #include <iostream>
6 #include <algorithm>
7
8 using namespace std;
9
10 struct t_aresta{
11     int dis;
12     int x, y;
13 };
14
15 bool comp(t_aresta a, t_aresta b){ return a.dis < b.
dis; }
16
17 //-----
18 #define MAXN 50500
19 #define MAXM 200200
20
21 int n, m; // únmero de évrtices e arestas
22 t_aresta aresta[MAXM];
23
24 // para o union find
25 int pai[MAXN];
26 int peso[MAXN];
27
28 // a árvore
29 t_aresta mst[MAXM];
30 //-----
31

```

```

32 // funções do union find
33 int find(int x){
34     if(pai[x] == x) return x;
35     return pai[x] = find(pai[x]);
36 }
37
38 void join(int a, int b){
39
40     a = find(a);
41     b = find(b);
42
43     if(peso[a] < peso[b]) pai[a] = b;
44     else if(peso[b] < peso[a]) pai[b] = a;
45     else{
46         pai[a] = b;
47         peso[b]++;
48     }
49 }
50 }
51
52 int main(){
53     // ler a entrada
54     cin >> n >> m;
55
56     for(int i = 1; i <= m; i++){
57         cin >> aresta[i].x >> aresta[i].y >> aresta[i].dis;
58     }
59
60     // inicializar os pais para o union-find
61     for(int i = 1; i <= n; i++) pai[i] = i;
62
63     // ordenar as arestas
64     sort(aresta+1, aresta+m+1, comp);
65
66     int size = 0;
67     for(int i = 1; i <= m; i++){
68         if( find(aresta[i].x) != find(aresta[i].y) ){
69             // se estiverem em componentes distintas
70             join(aresta[i].x, aresta[i].y);
71
72             mst[++size] = aresta[i];
73         }
74     }
75
76     // imprimir a MST
77     for(int i = 1; i <= n; i++) cout << mst[i].x << " "
78     << mst[i].y << " " << mst[i].dis << "\n";
79     return 0;
80 }

```

## 4.10 Lca

```

1 /*
2 Lowest Common ancestor (LCA) - é o nome típico dado
3 para o seguinte problema: dado uma Árvore cuja
4 raiz é um vértice arbitrário e dois vértices u,v
5 que a pertencem, diga qual é o nó mais baixo(
6 relativo a raiz) que é ancestral de u,v.
7 */
8
9 #include <bits/stdc++.h>
10 using namespace std;
11 const int SIZE = 1e5;
12 int depth[SIZE];
13 vector<int> graph[SIZE];
14
15 void pre_process_depth(int u, int d) {
16     depth[u] = d;

```

```

13     for(auto adj : graph[u]) {
14         pre_process_depth(adj, d + 1);
15     }
16 }
17
18 int p2k[SIZE][log2(SIZE)+1];
19 int lca(int u, int v) {
20     if(depth[u] < depth[v]) swap(u,v);
21     for (int i = 20; i >= 0; --i) {
22         if(depth[p2k[u][i]] >= depth[v])
23             u = p2k[u][i];
24     }
25     if(u == v) return u;
26     for (int i = 20; i >= 0; --i) {
27         if(p2k[v][i] != p2k[u][i]) {
28             v = p2k[v][i];
29             u = p2k[u][i];
30         }
31     }
32     return pai[v];
33 }
34
35 int climb(int node, int k){
36     for(int i = 20; i >= 0; i--) {
37         if(k >= (1 << i)) {
38             node = p2k(node,i);
39             k -= (1 << i);
40         }
41     }
42     return node;
43 }
44
45 int dist(int u, int v){
46     return depth[u] + depth[v] - 2*depth[lca(u,v)];
47 }
48
49 int main() {
50     // código
51     // le os pais e monta o grafo
52     pai[raiz] = raiz;
53     pre_process_depth(raiz); // tipicamente qual
54     // vértice é a raiz não importa
55     for(int node = 0; node < SIZE; node++){
56         p2k[node][0] = pai[node];
57     }
58     for(int node = 0; node < SIZE; node++) {
59         for(int k = 1; k <= log2(SIZE); k++) {
60             p2k[node][k] = p2k[p2k[node][k-1]][k-1];
61         }
62     }
63     // resolve problema

```

## 5 Math

### 5.1 Combinatoria

```

1 // quantidade de combinações possíveis sem repetição
2 de 2 números
3 int comb(int k){
4     if(k==1) return 1;
5     else if(k==0) return 0;
6     return (k*(k-1))/2;

```

### 5.2 Divisores

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 vector<long long> get_divisors(long long n){

```

```

5     vector<long long> divs;
6     for(long long i = 1; i*i <=n; i++){
7         if(n%i == 0){
8             divs.push_back(i);
9             long long j = n/i;
10            if(j != i)
11                divs.push_back(j);
12        }
13    }
14    return divs;
15 }

```

## 5.3 Fatora

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 map<int,int> fatora(int n){
5     vector<int> lp;
6     map<int,int> exp;
7     int count=0;
8     while(n>1){
9         exp[lp[n]]++;
10        n/=lp[n];
11    }
12    return exp;
13 }

```

## 5.4 Mdc

```

1 // Greatest common divisor / MDC
2
3 long long gcd(long long a, long long b){
4     return b ? gcd(b, a % b) : a;
5 }
6
7 // or just use __gcd(a,b)

```

## 5.5 Mmc

```

1 // Least Common Multiple - MMC
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 long long lcm(long long a, long long b){
6     return (a/__gcd(a,b)*b);
7 }

```

## 5.6 Pa

```

1 // ôSomatrio de 1 a K
2 int pa(int k){
3     return (k*(k+1))/2;
4 }

```

## 5.7 Primos

```

1 // PRIMALIDADE
2
3 #include <bits/stdc++.h>
4 using namespace std;

```

```

5
6 const int MAX = 1e5+7;
7
8 void crivo(){
9     vector<int> crivo(MAX, 1);
10    for(int i=2; i*i<=MAX; i++){
11        if(crivo[i]==1){
12            for(int j=i+i; j<MAX; j+=i){
13                crivo[j]=0;
14            }
15        }
16    }
17 }
18
19 bool is_prime(int num){
20     for(int i = 2; i*i<= num; i++) {
21         if(num % i == 0) {
22             return false;
23         }
24     }
25     return true;
26 }

```

# 6 Template

## 6.1 Template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 //g++ -std=c++17 -O2 -Wall run.cpp -o run
4
5 #define endl "\n"
6 #define sws std::ios::sync_with_stdio(false); cin.tie
7 #define all(x) x.begin(), x.end()
8 #define input(x) for (auto &it : x) cin >> it
9 #define print(x) for (auto &it : x) cout << it << ' '
10 ; cout<<endl;
11 #define dbg(msg, x) cout << msg << " = " << x << endl
12 #define pb push_back
13 #define mp make_pair
14 #define ff first
15 #define ss second
16 #define TETO(a, b) ((a) + (b-1))/(b)
17 #define loop(i,a,n) for(int i=a; i < n; i++)
18 typedef long long ll;
19 typedef vector<int> vi;
20 typedef pair<int,int> pii;
21 const ll MOD = 1e9+7;
22 const int MAX = 1e4+5;
23 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
24 //-----//
25 //          éF que o AC vem          //
26 //-----//
27
28 int main(){ sws;
29
30     return 0;
31 }

```