Team Notebook

[FEI] (sei ? trivial : bizarro)

August 29, 2024

Contents

1 dp

1.1 distinctSubsequences

```cpp
#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'

#define f first
#define s second

#define vi vector<int>
#define grafo vector<vector<int>>

#define dbg(x) cout << #x << " = " << x << endl;

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

// Funcao para encontrar a qtd de substring iguais a B em A
// Time Complexity: O(NxM) | Space: O(M);
int dp(string& s1, string& s2){
 int n = s1.size();
 int m = s2.size();

 vector<int> prev(m+1, 0);

 prev[0] = 1;

 for(int i = 1; i <= n; i++){
  for(int j = m; j >= 1; j--){
   if(s1[i-1]==s2[j-1])
    prev[j] = prev[j-1] + prev[j];
  }
 }

 return prev[m];
}

int main(){ _
 string s1 = "brootgroot";
 string s2 = "brt";

 cout << dp(s1, s2) << endl;
 return 0;
}
```

1.2 editDistance

```cpp
#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'

#define f first
#define s second

#define vi vector<int>
#define grafo vector<vector<int>>

#define dbg(x) cout << #x << " = " << x << endl;

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
const int MAX = 5*1e3+10;

int memo[MAX][MAX];

// Caixa preta de codigo para contar o numero
// minimo de movimentos para transformar uma
// string em outra. Complexity: O(M x N) , Space: O(M x N).

int dp(string s1, string s2){
 int n = s1.size(), m = s2.size();

 for(int i = 0; i <= n; i++) memo[i][0] = i;
 for(int j = 0; j <= m; j++) memo[0][j] = j;

 for(int i = 1; i <= n; i++)
  for(int j = 1; j <= m; j++){
   if(s1[i - 1] == s2[j - 1])// Verifica se os carac sao
       iguais
    memo[i][j] = memo[i - 1][j - 1];
   else
    memo[i][j] = 1 + min(memo[i - 1][j - 1], min(memo[i - 1][
        j], memo[i][j - 1]));
  }

 return memo[n][m];
}

int main(){ _
 string s1 , s2;cin>>s1>>s2;

 memset(memo, -1, sizeof memo);

 cout << dp(s1, s2) << endl;
 return 0;
}

// CSES Edit Distance
```

1.3 longestCommonSubsequence

```cpp
#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'

#define f first
#define s second

#define vi vector<int>
#define grafo vector<vector<int>>

#define dbg(x) cout << #x << " = " << x << endl;

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

//Implementacao de LCS de maneira recursiva
//retorna apenas o tamanho da maior subsequencia comum
//Complexity: O(NxM) | Space Complexity: O(NxM)
const int MAX = 1e3;
int memo[MAX][MAX];
int lcs(string& s1, string& s2, int i, int j){
 if(i<0 or j<0) return 0;

 int& pdm = memo[i][j];

 if(pdm != -1) return pdm;

 if(s1[i]==s2[j]){
  return pdm = lcs(s1, s2, i-1, j-1)+1;
 }
```

```
 return pdm = max(lcs(s1, s2, i-1, j), lcs(s1, s2, i, j-1));
}

//Implementando de maneira iterativa
// retorna apenas o tamanho da maior subsequencua comum
// caso deseja a maior sequencia palindroca
// inserir como parametros s1 e rev(s1)
// caso deseja o minimo de insercoes para tornar
// a string um palindromo basta
// size(s1) - lcs(s1,rev(s1)
// Complexity: O(NxM) | Space Complexity: O(N+M)
int lcs(string& s1, string& s2){
 int n = s1.size();
 int m = s2.size();

 vector<int> prev(m+1, 0), cur(m+1, 0);

 for(int i = 1; i<=n; i++){
  for(int j = 1; j<=m; j++){
   if(s1[i-1]==s2[j-1])
    cur[j]=prev[j-1]+1;
   else
    cur[j]=max(prev[j], cur[j-1]);
  }
  prev=cur;
 }
 return prev[m];
}

// Funcao para imprimir a maior subsequencia comum
// Complexity: O((NxM)+(N+M)) | Space Complexity: O(NxM)
string imprimirLCS(string& s1, string& s2){
 int n = s1.size();
 int m = s2.size();

 vector<vector<int>> dp(n+1, vector<int>(m+1, 0));

 for(int i = 1; i <= n; i++){
  for(int j = 1; j <= m; j++){
   if(s1[i-1]==s2[j-1])
    dp[i][j]=dp[i-1][j-1]+1;
   else
    dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
  }
 }
 int len = dp[n][m];
 string ans = "";

 for(int i = 0; i < len; i++)ans+='$';

 int idx = len-1;
 int i = n, j = m;

 while(i>0 and j>0){
  if(s1[i-1]==s2[j-1]){
   ans[idx]=s1[i-1];
   idx--;i--;j--;
  }
  else if (dp[i-1][j]>dp[i][j-1]){
   i--;
  }
  else{
   j--;
  }
 }
 return ans;
}
int main(){ _

 string s1 = "abcde";
 string s2 = "cde";

 int n = s1.size()-1;
 int m = s2.size()-1;

 memset(memo, -1, sizeof memo);

 cout << lcs(s1, s2) << endl;
 cout << imprimirLCS(s1, s2) << endl;
 return 0;
}
```

1.4 longestCommonSubstring

```
#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'

#define f first
#define s second

#define vi vector<int>
#define grafo vector<vector<int>>

#define dbg(x) cout << #x << " = " << x << endl;

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

int lcs(string& s1, string& s2){
 int n = s1.size();
 int m = s2.size();

 vector<int> prev(m+1, 0), cur(m+1, 0);
 int ans = -INF;

 for(int i = 1; i<=n; i++){
  for(int j = 1; j<=m; j++){
   if(s1[i-1]==s2[j-1]){
    cur[j]=prev[j-1]+1;
    ans = max(ans, cur[j]);
   }
   else cur[j]=0;
  }
  prev=cur;
 }

 return ans;
}

int main(){ _
 string s1 = "abcde";
 string s2 = "abde";

 cout << lcs(s1, s2) << endl;
 return 0;
}
```

1.5 longestIncreasingOrder

```
#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'

#define f first
#define s second

#define vi vector<int>
#define grafo vector<vector<int>>
```

```
#define dbg(x) cout << #x << " = " << x << endl;

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

// LIS sendo calculador por meio de busca binaria,
// podendo retornar os elementos da maior
// subsequencia ou somente o tamanho dela
// Complexity: O(n * logN) | Space: O(N)
int lis(vector<int>& arr , int n){
 vector<int> d(n+1 , INF) , prev(n , -1) , pos(n+1 , -1);

 d[0] = -INF;

 for(int i = 0 ; i < n ; i++){
  int l = upper_bound(d.begin() , d.end() , arr[i]) - d.
       begin();

  if(d[l-1] < arr[i] and arr[i] < d[l]){
   d[l] = arr[i];
   pos[l] = i;
   prev[i] = pos[l-1];
  }
 }

 int ans = 0;

 for(int l = 0 ; l <= n ; l++){
  if(d[l] < INF)
   ans = l;
 }

 vector<int> subseq;
 int k = pos[ans];

 while(k != -1){
  subseq.push_back(arr[k]);
  k = prev[k];
 }

 reverse(subseq.begin() , subseq.end());

 for(int& x : subseq) cout << x << ' ';cout<<endl;

 return ans;
}

int main(){ _
 vector<int> v = {7 , 3 , 5 , 3 , 6 , 2 , 9 , 8};

 cout << lis(v , 8) << endl;

 return 0;
}
```

1.6 longestPalindromicSubstring

```
#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'

#define f first
#define s second

#define vi vector<int>
#define grafo vector<vector<int>>

#define dbg(x) cout << #x << " = " << x << endl;

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

string longestPalinSubstring(string &s) {
    int n = s.size();
    if (n == 0) return "";

    int start = 0, maxLength = 1;

    // Function to expand around the center and find the
         longest palindrome
    auto expandAroundCenter = [&](int left, int right) {
        while (left >= 0 && right < n && s[left] == s[right])
             {
            left--;
            right++;
        }
        int length = right - left - 1;
        if (length > maxLength) {
            start = left + 1;
            maxLength = length;
        }
    };

    for (int i = 0; i < n; i++) {
        // Odd length palindrome (single center)
        expandAroundCenter(i, i);
        // Even length palindrome (two centers)
        expandAroundCenter(i, i + 1);
    }

    return s.substr(start, maxLength);
}

int main(){ _
 string s = "abbbba";
 string s2 = "abccbapp";

 cout << longestPalinSubstring(s) << endl;
 cout << longestPalinSubstring(s2) << endl;
 return 0;
}
```

1.7 wildCardMatching

```
#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'

#define f first
#define s second

#define vi vector<int>
#define grafo vector<vector<int>>

#define dbg(x) cout << #x << " = " << x << endl;

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;


bool ans(string& pattern, string& text){

 int n = pattern.size(), m = text.size();

 vector<bool> prev(m+1, false), cur(m+1, false);
```

```
 prev[0]=true;

 for(int j = 1; j<=m; j++)
 prev[j] = false;

 for(int i = 1; i<=n; i++){

  bool flag = false;
  for(int ii = 1; ii<=i; ii++){
   if(pattern[ii-1]!='*'){
    flag=true;
    break;
   }
  }
  prev[0]=flag;

  for(int j = 1; j<=m; j++){
   if(pattern[i-1]==text[j-1] or pattern[i-1]=='?')
    cur[j]=prev[j-1];
   else if(pattern[i-1]=='*')
    cur[j]=prev[j] or cur[j-1];
   else cur[j]=false;
  }
  prev=cur;
 }
 return prev[m];
}

int main(){ _
 string pattern = "ab*cd";
 string text = "abcdefcd";


 cout << (ans(pattern,text) ? "foi" : "n foi :(") << endl;
 return 0;
}
```

2 ds

2.1 bit

```
/*
binary indexed tree (fenwick)

descricao:
 serve pra queries associativas em range [0, r];

complexidades:
 memoria: o(n);
 build: o(n);
 query: o(logn), soma de [l, r]
 update: o(logn), soma x em i
*/

namespace bit {
 vector<int> fen, v;
 int n;

 void build() {
  fen.assign(n, 0);
  for (int i = 0; i < n; i++) {
   fen[i] += v[i];
   int r = i | (i + 1);
   if (r < n) fen[r] += fen[i];
  }
 }

 int pre(int r) {
  int res = 0;
  for (; r >= 0; r = (r & (r + 1)) - 1)
   res += fen[r];
  return res;
 }

 int query(int l, int r) {
  return pre(r) - pre(l - 1);
 }

 void update(int i, int x) {
  for (; i < n; i = i | (i + 1))
   fen[i] += x;
 }
}

// para range update e point query:
{
 void update(int i, int x) {
  for (++i; i < n; i += i & -i)
   fen[i] += x;
 }

 void range_update(int l, int r, int x) {
  update(l, x);
  update(r + 1, -x);
 }

 int point_query(int i) {
  int res = 0;
  for (++i; i < n; i += i & -i)
   res += fen[i];
  return res
 }
}
```

2.2 bit2d

```
/*
binary indexed tree 2d (fenwick)

descricao:
 serve pra queries associativas agora com uma dimensao a
      mais

complexidades:
 memoria: o(n^2);
 build: o(n^2log^2n);
 query: o(log^2n), soma retangulo p(x2, y2), p(x1, y1);
 update: o(log^2n), soma x em i;
*/

namespace bit2d {
 vector<vector<int>> fen, v;
 int n;

 void build() {
  fen.assign(n, vector<int>(n, 0));
  for (int i = 0; i < n; i++) {
   for (int j = 0; j < n; j++) {
    fen[i][j] = v[i][j];
   }
  }

  for (int i = 0; i < n; i++) {
   for (int j = 0; j < n; j++) {
    int x = i | (i + 1), y = j | (j + 1);
    if (x < n) fen[x][j] += fen[i][j];
    if (y < n) fen[i][y] += fen[i][j];
    if (x < n && y < n) fen[x][y] -= fen[i][j];
   }
  }
 }

 int pre(int x, int y) {
     int res = 0;
     for (int i = x; i >= 0; i = (i & (i + 1)) - 1) {
         for (int j = y; j >= 0; j = (j & (j + 1)) - 1) {
             res += fen[i][j];
```

```
        }
    }
    return res;
}

int query(int x1, int y1, int x2, int y2) {
    return pre(x2, y2) - pre(x1 - 1, y2) - pre(x2, y1 - 1) +
          pre(x1 - 1, y1 - 1);
}

void update(int x, int y, int val) {
    for (int i = x; i < n; i = i | (i + 1)) {
        for (int j = y; j < n; j = j | (j + 1)) {
            fen[i][j] += val;
        }
    }
}
}
```

2.3 dsu

```
/*
union find

descricao:
 representa e mantem um conjunto. merge tem small
 to large e find tem compressao de caminho. nem
 sempre eh necessario, mas com essas duas a
 complexidade eh a melhor possivel. tirar id[i]=
 do find nao precisar de compressao e tirar o que
 for relacionado com sz no merge se nao precisar
 de small to large.

complexidades:
 memoria: o(n);
 find: o(1/ackermann(n));
 merge: o(1/ackermann(n)) ~ 1;
*/

struct DSU {
 vector<int> id, sz;
 DSU(int n) : id(n), sz(n, 1) {
  iota(id.begin(), id.end(), 0);
 }

 int find(int i) {
  return (i == id[i] ? i : id[i] = find(id[i]));
 }
 void merge(int i, int j) {
  if ((i = find(i)) == (j = find(j))) return;
  if (sz[i] < sz[j]) swap(i, j);
  sz[i] += sz[j], id[j] = i;
 }
};
```

2.4 hashtable

```
/*
hash-table dopada

descricao:
 basicamente a mesma coisa do unordered_set padrao
 mas com uma funcao hash melhor pra minimizar colisoes
 e quase garantir os o(1) que a hash table promete.
*/
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM = chrono::steady_clock
         ::now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};

gp_hash_table<int, int, custom_hash> mp;
```

2.5 hld

```
/*
heavy-light decomposition

descricao:
 um jeito de querys e atualizacoes em arvores. da pra
 colocar qualquer estrutura no lugar da seg, mas essa
 implementacao usa a seg lazy que tem aqui tambem.
 logo, precisa codar a seg tambem.

complexidades:
 memoria: o(n);
 build: o(n);
 query_path: o(log^2(n));
 update_path: o(log^2(n));
 query_subtree: o(log(n));
 update_subtree: o(log(n));
*/

// namespace seg { ... }

namespace hld {
 vector<pair<int, int> > g[MAX];
 int pos[MAX], sz[MAX];
 int sobe[MAX], pai[MAX];
 int h[MAX], v[MAX], t;

 void build_hld(int k, int p = -1, int f = 1) {
  v[pos[k] = t++] = sobe[k]; sz[k] = 1;
  for (auto& i : g[k]) if (i.first != p) {
   auto [u, w] = i;
   sobe[u] = w; pai[u] = k;
   h[u] = (i == g[k][0] ? h[k] : u);
   build_hld(u, k, f); sz[k] += sz[u];

   if (sz[u] > sz[g[k][0].first] or g[k][0].first == p)
    swap(i, g[k][0]);
  }
  if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
 }
 void build(int root = 0) {
  t = 0;
  build_hld(root);
  seg::build(t, v);
 }
 ll query_path(int a, int b) {
  if (a == b) return 0;
  if (pos[a] < pos[b]) swap(a, b);

  if (h[a] == h[b]) return seg::query(pos[b]+1, pos[a]);
  return seg::query(pos[h[a]], pos[a]) + query_path(pai[h[a
       ]], b);
 }
 void update_path(int a, int b, int x) {
  if (a == b) return;
  if (pos[a] < pos[b]) swap(a, b);

  if (h[a] == h[b]) return (void)seg::update(pos[b]+1, pos[a
       ], x);
```

```
  seg::update(pos[h[a]], pos[a], x); update_path(pai[h[a]],
      b, x);
 }
 ll query_subtree(int a) {
  if (sz[a] == 1) return 0;
  return seg::query(pos[a]+1, pos[a]+sz[a]-1);
 }
 void update_subtree(int a, int x) {
  if (sz[a] == 1) return;
  seg::update(pos[a]+1, pos[a]+sz[a]-1, x);
 }
 int lca(int a, int b) {
  if (pos[a] < pos[b]) swap(a, b);
  return h[a] == h[b] ? b : lca(pai[h[a]], b);
 }
}
```

2.6 hldV

```
/*
heavy-light decomposition (vertices)

descricao:
 mesma coisa da outra, mas com os valore nos vertices.

complexidades:
 memoria: o(n);
 build: o(n);
 query_path: o(log^2(n));
 update_path: o(log^2(n));
 query_subtree: o(log(n));
 update_subtree: o(log(n));
*/

// namespace seg { ... }

namespace hld {
 vector<int> g[MAX];
 int pos[MAX], sz[MAX];
 int peso[MAX], pai[MAX];
 int h[MAX], v[MAX], t;

 void build_hld(int k, int p = -1, int f = 1) {
  v[pos[k] = t++] = peso[k]; sz[k] = 1;
  for (auto& i : g[k]) if (i != p) {
   pai[i] = k;
   h[i] = (i == g[k][0] ? h[k] : i);
   build_hld(i, k, f); sz[k] += sz[i];
   if (sz[i] > sz[g[k][0]] or g[k][0] == p) swap(i, g[k][0])
       ;
  }
  if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
 }
 void build(int root = 0) {
  t = 0;
  build_hld(root);
  seg::build(t, v);
 }
 ll query_path(int a, int b) {
  if (pos[a] < pos[b]) swap(a, b);

  if (h[a] == h[b]) return seg::query(pos[b], pos[a]);
  return seg::query(pos[h[a]], pos[a]) + query_path(pai[h[a
      ]], b);
 }
 void update_path(int a, int b, int x) {
  if (pos[a] < pos[b]) swap(a, b);

  if (h[a] == h[b]) return (void)seg::update(pos[b], pos[a],
       x);
  seg::update(pos[h[a]], pos[a], x); update_path(pai[h[a]],
      b, x);
 }
 ll query_subtree(int a) {
  return seg::query(pos[a], pos[a]+sz[a]-1);
 }
 void update_subtree(int a, int x) {
  seg::update(pos[a], pos[a]+sz[a]-1, x);
 }
 int lca(int a, int b) {
  if (pos[a] < pos[b]) swap(a, b);
  return h[a] == h[b] ? b : lca(pai[h[a]], b);
 }
}
```

2.7 orderedset

```
/*
ordered set

descricao:
 set normal, mas deixa fazer perguntas sobre posicao.
 o multiset permite a insercao de varios valores iguais.
 so colocar um inteiro nadave no .s.
 insert, erase, begin, end, clear, size;
 find_by_order(i): iterador pro elemento na posicao i;
 order_of_key(i): quantidade de valores menores que i;
complexidades:
 tudo log(n);
*/
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using ordered_set = tree<int, null_type, less<int>,
    rb_tree_tag, tree_order_statistics_node_update>;

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <class T>
using ordered_multiset = tree<pair<T, int>, null_type, less<
    pair<T, int>>, rb_tree_tag,
    tree_order_statistics_node_update>;
```

2.8 segtree

```
/*
segtree com point update

descricao:
 resume resultados de segmentos em nodos de uma
 arvore binaria.

complexidades:
 memoria: o(n);
 query: o(logn), soma de [a, b];
 update: o(logn), soma x na pos i;
*/

namespace seg {
 ll seg[4*MAX], v[MAX];
 int n;

 ll build(int p=1, int l=0, int r=n-1) {
  if (l == r) return seg[p] = v[l];
  int m = (l+r)/2;
  return seg[p] = build(2*p, l, m) + build(2*p+1, m+1, r);
 }
 ll query(int a, int b, int p=1, int l=0, int r=n-1) {
  if (b < l or r < a) return 0;
  if (a <= l and r <= b) return seg[p];
  int m = (l+r)/2;
  return query(a, b, 2*p, l, m) + query(a, b, 2*p+1, m+1, r)
      ;
 }
```

```
 ll update(int i, int x, int p=1, int l=0, int r=n-1) {
  if (i < l or r < i) return seg[p];
  if (l == r) return seg[p] = x;
  int m = (l+r)/2;
  return seg[p] = update(i, x, 2*p, l, m) + update(i, x, 2*p
      +1, m+1, r);
 }
};
```

2.9 segtreeLazy

```
/*
segtree com lazy propagation

descricao:
 mesma coisa da com point upd, mas essa permite
 update em range. 'lazy' eh porque ela so da update
 quando precisa, stackando os updates anteriores.
 portante, o que tem que ta no lazy tem que ser
 associativo tambem;

complexidades:
 memoria: o(n);
 build: o(n);
 prop: o(1), atualiza valores do nodo;
 query: o(logn), soma de [a, b];
 update: o(logn), soma x em [a, b];
*/


namespace seg {
 ll seg[4*MAX], lazy[4*MAX];
 int n, *v;

 ll build(int p=1, int l=0, int r=n-1) {
  lazy[p] = 0;
  if (l == r) return seg[p] = v[l];
  int m = (l+r)/2;
  return seg[p] = build(2*p, l, m) + build(2*p+1, m+1, r);
 }
 void build(int n2, int* v2) {
  n = n2, v = v2;
  build();
 }
 void prop(int p, int l, int r) {
  seg[p] += lazy[p]*(r-l+1);
  if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] += lazy[p];
  lazy[p] = 0;
 }
 ll query(int a, int b, int p=1, int l=0, int r=n-1) {
  prop(p, l, r);
  if (a <= l and r <= b) return seg[p];
  if (b < l or r < a) return 0;
  int m = (l+r)/2;
  return query(a, b, 2*p, l, m) + query(a, b, 2*p+1, m+1, r)
      ;
 }
 ll update(int a, int b, int x, int p=1, int l=0, int r=n-1)
      {
  prop(p, l, r);
  if (a <= l and r <= b) {
   lazy[p] += x;
   prop(p, l, r);
   return seg[p];
  }
  if (b < l or r < a) return seg[p];
  int m = (l+r)/2;
  return seg[p] = update(a, b, x, 2*p, l, m) +
   update(a, b, x, 2*p+1, m+1, r);
 }
};

// em uma seg de max, acha primeiro i que v[i] >= val;
int get_left(int a, int b, int val, int p=1, int l=0, int r=
    n-1) {
 prop(p, l, r);
 if (b < l or r < a or seg[p] < val) return -1;
 if (r == l) return l;
 int m = (l+r)/2;
 int x = get_left(a, b, val, 2*p, l, m);
 // para ultimo i, int x = get_right(a, b, val, 2*p+1, m+1,
     r);
 if (x != -1) return x;
 return get_left(a, b, val, 2*p+1, m+1, r);
}
```

2.10 sparseTable

```
/*
sparse-table

descricao:
 resume intervalos com uma manha de bits
 bem parecida com a Fenwick.

complexidades:
 memoria: o(n);
 build: o(nlogn);
 query: o(1), min de [a, b];
*/

namespace sparse {
    int t[MAX][18], v[MAX];
    int n;

    void build() {
        for(int i = 0; i < n; ++i) t[i][0] = v[i];
        for(int k = 1; (1 << k) <= n; ++k) {
            for(int i = 0; i + (1 << k) <= n; ++i) {
                t[i][k] = min(t[i][k - 1], t[i + (1 << (k - 1)
                    )][k - 1]);
            }
        }
    }

    int query(int l, int r) {
        int k = 31 - __builtin_clz(r - l + 1);
        return min(t[l][k], t[r - (1 << k) + 1][k]);
    }
};
```

2.11 staticMergeSort

```
/*
merge sort tree estatica

descricao:
 cada nodo guarda um vetor ordenado de todos os
 valores do range que ele resume. desse jeito
 retorna todos os valores menores ou iguais a x
 em [l, r];

complexidades:
 memoria: o(nlog(n)) acho;
 build: o(nlog(n));
 query: o(nlog^2(n)), por causa da bin search;
*/

namespace seg {
    vector<vector<int>> seg;
 vector<int> v;
    int n;

    void build(int p=1, int l=0, int r=n-1) {
        if (l == r)
  {
            seg[p].push_back(v[l]);
```

```
            return;
        }
        int m = (l+r)/2;
        build(2*p, l, m);
        build(2*p+1, m+1, r);
        merge(seg[2*p].begin(), seg[2*p].end(), seg[2*p+1].
             begin(), seg[2*p+1].end(), back_inserter(seg[p])
             );
    }
    void build(int n2, vector<int>& v2) {
        n = n2, v = v2;
        seg.resize(4*n);
        build();
    }
    int query(int a, int b, int x, int p=1, int l=0, int r=n
         -1) {
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return seg[p].end() -
             upper_bound(seg[p].begin(), seg[p].end(), x);
        int m = (l+r)/2;
        return query(a, b, x, 2*p, l, m) + query(a, b, x, 2*p
             +1, m+1, r);
    }
};
```

2.12 trie

```
/*
trie

descricao:
 arvore que mantem um dicionario.
 TRIE t() constroi uma trie proa alfabeto das
 letras minustulas. t(tamanho do alfabeto, menor c)
 tambem pode ser usado.

complexidades:
 memoria: o(n|s|);
 insert: o(|s|*sigma)
 erase: o(|s|)
 find: o(|s|) retorna a posicao, -1 se nao achar
 count_pref: o(|s|) numero de strings s eh prefixo
*/

struct TRIE {
 vector<vector<int>> to;
 vector<int> end, pref;
 int sigma; char norm;
 TRIE(int sigma_=26, char norm_='a') : sigma(sigma_), norm(
      norm_) {
  to = {vector<int>(sigma)};
  end = {0}, pref = {0};
 }
 void insert(string s) {
  int x = 0;
  for (auto c : s) {
   int &nxt = to[x][c-norm];
   if (!nxt) {
    nxt = to.size();
    to.push_back(vector<int>(sigma));
    end.push_back(0), pref.push_back(0);
   }
   x = nxt, pref[x]++;
  }
  end[x]++, pref[0]++;
 }
 void erase(string s) {
  int x = 0;
  for (char c : s) {
   int &nxt = to[x][c-norm];
   x = nxt, pref[x]--;
   if (!pref[x]) nxt = 0;
  }
  end[x]--, pref[0]--;
 }
 int find(string s) {
  int x = 0;
  for (auto c : s) {
   x = to[x][c-norm];
   if (!x) return -1;
  }
  return x;
 }
 int count_pref(string s) {
  int id = find(s);
  return id >= 0 ? pref[id] : 0;
 }
};
```

2.13 wavelet

```
/*
wavelet tree estatica

descricao:
 uma segtree mas encima de valores e nao de indices.

complexidades (d = maxn - minn):
 memoria: o(nlog(d));
 build: o(nlog(d));
 conta: o(log(d)),
  conta quantos em [x, y] que ocorrem em [i, j);

 soma: o(log(d)),
  soma os em [x, y] que ocorrem em [i, j);

 somak: o(log(d)),
  soma os primeiros que ocorrem em [i, j) ordenado;

 kth:  o(log(d)),
  k-esimo menor valor que ocorre em [i, j) ordenado;

 swp:  o(log(d)),
  inverte o valor n indice i com o i+1;
*/
namespace wave {
    int n, v[MAX], delta = maxn-minn;
    vector<int> esq[4*delta], pre[4*delta];

    void build(int b=0, int e=n, int p=1, int l=minn, int r=
         maxn) {
        if (l == r) return;
        int m=(l+r)/2; esq[p].push_back(0), pre[p].push_back
             (0);
        for (int i = b; i < e; i++) {
            esq[p].push_back(esq[p].back() + (v[i] <= m));
            pre[p].push_back(pre[p].back() + v[i]);
        }
        int mid = stable_partition(v + b, v + e, [m](int x) {
             return x <= m; })-v;
        build(b, mid, 2*p, l, m), build(mid, e, 2*p+1, m+1, r
             );
    }

    int conta(int i, int j, int x, int y, int p=1, int l=minn
         , int r=maxn) {
        if (y < l or x > r) return 0;
        if (x <= l and r <= y) return j-i;
        int m=(l+r)/2, ei = esq[p][i], ej = esq[p][j];
        return conta(ei, ej, x, y, 2*p, l, m) + conta(i-ei, j
             -ej, x, y, 2*p+1, m+1, r);
    }

    int soma(int i, int j, int x, int y, int p=1, int l=minn,
          int r=maxn) {
        if (y < l or x > r) return 0;
        if (x <= l and r <= y) return pre[p][j]-pre[p][i];
```

```
        int m=(l+r)/2, ei = esq[p][i], ej = esq[p][j];
        return soma(ei, ej, x, y, 2*p, l, m) + soma(i-ei, j-
             ej, x, y, 2*p+1, m+1, r);
    }

    int somak(int i, int j, int k, int p=1, int l=minn, int r
         =maxn) {
        if (l == r) return k*l;
        int m=(l+r)/2, ei = esq[p][i], ej = esq[p][j];
        if (k <= ej-ei) return somak(ei, ej, k, 2*p, l, m);
        return pre[2*p][ej] - pre[2*p][ei] + somak(i-ei, j-ej
             , k-(ej-ei), 2*p+1, m+1, r);
    }

    int kth(int i, int j, int k, int p=1, int l=minn, int r=
         maxn) {
        if (l == r) return l;
        int m=(l+r)/2, ei = esq[p][i], ej = esq[p][j];
        if (k <= ej-ei) return kth(ei, ej, k, 2*p, l, m);
        return kth(i-ei, j-ej, k-(ej-ei), 2*p+1, m+1, r);
    }

 void swp(int i, int p=1, int l=minn, int r=maxn) {
  if (l == r) return;
  int m=(l+r)/2;
  bool b1 = v[i]<=m, b2 = v[i+1]<=m;
  if (b1 != b2) {
   esq[p][i+1] = esq[p][i];
   esq[p][i] += (b1 ? -1 : 1);
   esq[p][i+1] += (b1 ? 1 : -1);
  }
  pre[p][i+1] = pre[p][i];
  swap(v[i], v[i+1]);
  if (b1) swp(i, 2*p, l, m);
  else swp(i, 2*p+1, m+1, r);

 }
}
```

3 extra

3.1 origem

```
#include <bits/stdc++.h>
#define f first
#define s second
#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define dbg(x) cout << #x << " = " << x << endl
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
using namespace std;

int main()
{
    _;

    return(0);
}
```

3.2 vimrc

```
set ts=4 sw=4 mouse=a number ai si undofile
syntax on
```

4 grafo

4.1 DFS-EXEMPLO

```
#include <bits/stdc++.h>
#define f first
#define s second
#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define dbg(x) cout << #x << " = " << x << endl
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
using namespace std;

int* visitados;
void dfs(int rotulo);
vector<vector<int>> grafos;

int main() {
       int vert;
       int arestas;
       int v1, v2;
       int nodo;

       cin >> nodo;
       cin >> vert >> arestas;
       grafos.resize(vert,);
       visitados = new int[vert];
       fill_n(visitados, vert, 0);

       for (int j = 0; j < arestas; j++) {
           cin >> v1 >> v2;
           grafos[v1].push_back(v2);
           grafos[v2].push_back(v1);
       }

       dfs(nodo);

       delete[] visitados;
       grafos.clear();

       return 0;
}

void dfs(int rotulo) {
    visitados[rotulo] = 1;
    cout << rotulo << " "; // Imprime o vrtice visitado

    for (int i = 0; i < grafos[rotulo].size(); i++) {
        if (visitados[grafos[rotulo][i]] == 0) {
            dfs(grafos[rotulo][i]);
        }
    }
}
```

4.2 DFS-Tree-Inorder

```
#include <bits/stdc++.h>
#define f first
#define s second
#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define dbg(x) cout << #x << " = " << x << endl
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
using namespace std;

struct No {
    int valor;
    struct No *esquerda, *direita;
    No(int valor) {
        this->valor = valor;
        esquerda = direita = NULL;
    }
```

```
};

void imprimirEmOrdem(struct No* no) {
    if (no == NULL)
        return;

    imprimirEmOrdem(no->esquerda);
    cout << no->valor << " ";
    imprimirEmOrdem(no->direita);
}

No* inserir(No* raiz, int valor) {
    if (raiz == NULL) {
        return new No(valor);
    }
    queue<No*> q;
    q.push(raiz);
    while (!q.empty()) {
        No* temp = q.front();
        q.pop();

        if (temp->esquerda == NULL) {
            temp->esquerda = new No(valor);
            break;
        } else {
            q.push(temp->esquerda);
        }

        if (temp->direita == NULL) {
            temp->direita = new No(valor);
            break;
        } else {
            q.push(temp->direita);
        }
    }
    return raiz;
}

int main() {
    _;
    int n, m;
    cin >> n;
    cin >> m;

    No* raiz = NULL;
    for (int i = 0; i < n; ++i) {
        int valor;
        cin >> valor;
        raiz = inserir(raiz, valor);
    }

    cout << "\nA travessia em ordem da rvore binria  \n";
    imprimirEmOrdem(raiz);
    cout << endl;

    return 0;
}
```

4.3 DFS-Tree-Postorder

```
#include <bits/stdc++.h>
#define f first
#define s second
#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define dbg(x) cout << #x << " = " << x << endl
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
using namespace std;

struct No {
    int valor;
    No *esq, *dir;
    No(int valor) {
        this->valor = valor;
        esq = dir = nullptr;
    }
};

void inserir(No*& raiz, int valor) {
    if (!raiz) {
        raiz = new No(valor);
    } else {
        queue<No*> fila;
        fila.push(raiz);

        while (!fila.empty()) {
            No* atual = fila.front();
            fila.pop();

            if (!atual->esq) {
                atual->esq = new No(valor);
                break;
            } else {
                fila.push(atual->esq);
            }

            if (!atual->dir) {
                atual->dir = new No(valor);
                break;
            } else {
                fila.push(atual->dir);
            }
        }
    }
}

void imprimirPosOrdem(No* no) {
    if (!no) return;
    imprimirPosOrdem(no->esq);
    imprimirPosOrdem(no->dir);
    cout << no->valor << " ";
}

int main() { _
    No* raiz = nullptr;
    int n, valor;

    cin >> n;

    for (int i = 0; i < n; i++) {
        cin >> valor;
        inserir(raiz, valor);
    }

    cout << "\nA travessia ps-ordem da rvore  binria   :\n";
    imprimirPosOrdem(raiz);

    return 0;
}
```

4.4 Dinic

```
#include <bits/stdc++.h>
#define f first
#define s second
#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define dbg(x) cout << #x << " = " << x << endl
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
using namespace std;

vector<int> level, work;
```

```
bool bfs(int source, int sink, vector<vector<int>>& adj,
     vector<vector<ll>>& capacity) {
    fill(level.begin(), level.end(), -1);
    queue<int> q;
    q.push(source);
    level[source] = 0;
    while (!q.empty()) {
        int node = q.front(); q.pop();
        for (int neighbor : adj[node]) {
            if (level[neighbor] == -1 && capacity[node][
                 neighbor] > 0) {
                level[neighbor] = level[node] + 1;
                q.push(neighbor);
            }
        }
    }
    return level[sink] != -1;
}

ll dfs(int node, ll flow, int sink, vector<vector<int>>& adj
     , vector<vector<ll>>& capacity, vector<tuple<int, int,
     ll>>& edges_used) {
    if (node == sink) return flow;
    for (int &i = work[node]; i < adj[node].size(); i++) {
        int neighbor = adj[node][i];
        if (level[neighbor] == level[node] + 1 && capacity[
             node][neighbor] > 0) {
            ll min_flow = min(flow, capacity[node][neighbor])
                 ;
            ll pushed_flow = dfs(neighbor, min_flow, sink,
                 adj, capacity, edges_used);
            if (pushed_flow > 0) {
                capacity[node][neighbor] -= pushed_flow;
                capacity[neighbor][node] += pushed_flow;
                edges_used.push_back({node, neighbor,
                     pushed_flow});
                return pushed_flow;
            }
        }
    }
    return 0;
}

ll dinic(int source, int sink, vector<vector<int>>& adj,
     vector<vector<ll>>& capacity, vector<tuple<int, int, ll
     >>& edges_used) {
    ll max_flow = 0;
    while (bfs(source, sink, adj, capacity)) {
        work.assign(adj.size(), 0);
        while (ll flow = dfs(source, LINF, sink, adj,
             capacity, edges_used)) {
            max_flow += flow;
        }
    }
    return max_flow;
}

int main() {
    _;

    int n, m;
    cin >> n >> m;

    vector<vector<ll>> capacity(n, vector<ll>(n));
    vector<vector<int>> adj(n);
    level.resize(n);
    work.resize(n);

    for (int i = 0; i < m; i++) {
        int u, v;
        ll c;
        cin >> u >> v >> c;
        --u, --v;
        adj[u].push_back(v);
        adj[v].push_back(u);
        capacity[u][v] += c;
        capacity[v][u] += c;
    }

    int source = 0;
    int sink = n - 1;

    vector<tuple<int, int, ll>> edges_used;
    ll max_flow = dinic(source, sink, adj, capacity,
         edges_used);

    cout << "Max Flow: " << max_flow << endl;
    cout << "Edges used in the flow:" << endl;
    for (auto& edge : edges_used) {
        int u, v;
        ll flow;
        tie(u, v, flow) = edge;
        cout << u + 1 << " -> " << v + 1 << " with flow " <<
             flow << endl;
    }

    return 0;
}
```

4.5 EdmondsKarp

```
// Este cdigo    um exemplo de  implementao  do algoritmo de
     Edmonds-Karp,
// que    uma  verso  especfica  do algoritmo de Ford-
     Fulkerson usada para calcular o fluxo mximo em uma rede
      de fluxo.
// O algoritmo de Edmonds-Karp utiliza buscas em largura (
     BFS) para encontrar o caminho aumentante de menor
     comprimento (em termos de arestas)
// do  n  de origem (source) at o  n  de destino (sink).


#include <bits/stdc++.h>
#define f first
#define s second
#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define dbg(x) cout << #x << " = " << x << endl
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
using namespace std;

ll max_flow(vector<vector<int>> adj, vector<vector<ll>>
     capacity, int source, int sink) {
    int n = adj.size();
    vector<int> parent(n, -1);

    auto reachable = [&]() -> bool {
        queue<int> q;
        q.push(source);
        fill(parent.begin(), parent.end(), -1);
        parent[source] = -2;
        while (!q.empty()) {
            int node = q.front();
            q.pop();
            for (int son : adj[node]) {
                ll w = capacity[node][son];
                if (w <= 0 || parent[son] != -1) continue;
                parent[son] = node;
                q.push(son);
            }
        }
        return parent[sink] != -1;
    };

    ll flow = 0;
    while (reachable()) {
        int node = sink;
```

```
        ll curr_flow = LINF;

        cout << "Caminho encontrado: ";
        while (node != source) {
            curr_flow = min(curr_flow, capacity[parent[node
                ]][node]);
            cout << node + 1 << " <- ";
            node = parent[node];
        }
        cout << source + 1 << endl;
        cout << "Fluxo atual do caminho: " << curr_flow <<
             endl;

        node = sink;
        while (node != source) {
            cout << "Reduzindo capacidade: " << parent[node]
                 + 1 << " -> " << node + 1
                 << " de " << capacity[parent[node]][node];
            capacity[parent[node]][node] -= curr_flow;
            cout << " para " << capacity[parent[node]][node]
                 << endl;

            cout << "Aumentando capacidade residual: " <<
                 node + 1 << " -> " << parent[node] + 1
                 << " de " << capacity[node][parent[node]];
            capacity[node][parent[node]] += curr_flow;
            cout << " para " << capacity[node][parent[node]]
                 << endl;

            node = parent[node];
        }

        flow += curr_flow;
        cout << "Fluxo total at agora: " << flow << endl <<
             endl;
    }

    return flow;
}

int main() {
    _;

    int n, m;
    cin >> n >> m;

    vector<vector<ll>> capacity(n, vector<ll>(n));
    vector<vector<int>> adj(n);
    for (int i = 0; i < m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        adj[--a].push_back(--b);
        adj[b].push_back(a);
        capacity[a][b] += c;
    }

    cout << "Fluxo mximo: " << max_flow(adj, capacity, 0, n -
         1) << endl;

    return 0;
}
```

4.6 Ex-Topological-Sorting

```
#include <bits/stdc++.h>
#define f first
#define s second
#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
using namespace std;

vector<vector<int>> grafos;
vector<int> indeg, ordem;
priority_queue<int, vector<int>, greater<int>> pq;

//  Funo  para realizar a  ordenao  topolgica  em um grafo
     direcionado
// A  funo  retorna true se for possvel obter uma  ordenao
     topolgica  vlida
// Caso contrrio, retorna false, indicando que o grafo contm
     um ciclo
// Complexidade: O(V + E), onde V  o nmero de vrtices e E
     o  nmero  de arestas
bool ordenacao_topologica(int vert) {
    int cnt = 0;

    // Adiciona todos os vrtices com grau de entrada 0 na
         fila de prioridade
    for (int i = 1; i <= vert; i++) {
        if (indeg[i] == 0) {
            pq.push(i);
        }
    }

    // Enquanto houver vrtices na fila de prioridade
    while (!pq.empty()) {
        int u = pq.top();
        pq.pop();
        ordem.push_back(u);
        cnt++;

        // Para cada vizinho do vrtice u, decrementa o grau
             de entrada
        // Se o grau de entrada de algum vizinho se tornar 0,
              adiciona-o na fila
        for (int v : grafos[u]) {
            indeg[v]--;
            if (indeg[v] == 0) {
                pq.push(v);
            }
        }
    }

    // Se a contagem de vrtices processados for igual ao
         nmero de vrtices
    // Significa que foi possvel realizar uma ordenao
         topolgica  vlida
    return (cnt == vert);
}

int main() {_
    int vert, arestas, v1, v2;

    cin >> vert >> arestas;

    grafos.resize(vert + 1);
    indeg.assign(vert + 1, 0);

    // Constri  o grafo e calcula os graus de entrada de cada
          vrtice
    for (int j = 0; j < arestas; j++) {
        cin >> v1 >> v2;
        grafos[v1].push_back(v2);
        indeg[v2]++;
    }

    // Chama a  funo  de  ordenao  topolgica
    if (ordenacao_topologica(vert)) {
        // Se a  ordenao  for bem-sucedida, imprime a ordem
             topolgica
        for (int u : ordem) {
            cout << u << " ";
        }
        cout << endl;
    } else {
```

```
        // Se a  ordenao  no  for possvel , imprime uma
            mensagem indicando o problema
        cout << "No    possvel  realizar uma  ordenao
            topolgica  vlida ." << endl;
    }

    return 0;
}
```

4.7 Hopcroft–Karp

```
#include <bits/stdc++.h>
#define f first
#define s second
#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define dbg(x) cout << #x << " = " << x << endl
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
using namespace std;

// Variveis  globais
int esquerda, direita, ligacoes; // Nmero de vrtices em cada
     lado do grafo bipartido e nmero de conexes
vector<int> match, dist; // Vetores para armazenar
    emparelhamentos e distncias
vector<vector<int>> g; // Lista de adjacncia para o grafo

//  Funo  para realizar a busca em largura (BFS) e
    encontrar caminhos alternantes
bool bfs() {
    queue<int> q;
    // Inicializa a distncia de cada vrtice da esquerda e
        coloca os vrtices no emparelhados na fila
    for (int node = 1; node <= esquerda; node++) {
        if (!match[node]) {
            q.push(node);
            dist[node] = 0;
        } else {
            dist[node] = INF;
        }
    }

    dist[0] = INF; // N  fictcio  que representa um
        emparelhamento vazio

    // Executa a BFS para encontrar caminhos aumentantes
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        if (dist[node] >= dist[0]) { continue; }
        for (int son : g[node]) {
            if (dist[match[son]] == INF) {
                dist[match[son]] = dist[node] + 1;
                q.push(match[son]);
            }
        }
    }


    return dist[0] != INF; // Retorna true se houver um
        caminho aumentante
}

//  Funo  para realizar a busca em profundidade (DFS) e
    encontrar um caminho aumentante
bool dfs(int node) {
    if (node == 0) { return true; } // Chegou ao n fictcio,
        caminho aumentante encontrado
    for (int son : g[node]) {
        if (dist[match[son]] == dist[node] + 1 && dfs(match[
            son])) {
            match[node] = son;
            match[son] = node;
            return true;
        }
    }
    dist[node] = INF; // Marca o n como no  acessvel  para
        evitar ciclos
    return false;
}

//  Funo  para implementar o algoritmo de Hopcroft-Karp e
    encontrar o emparelhamento mximo
int hopcroft_karp() {
    int cnt = 0; // Contador para o nmero mximo de pares
    while (bfs()) {
        for (int node = 1; node <= esquerda; node++) {
            if (!match[node] && dfs(node)) {
                cnt++; // Conta cada novo emparelhamento
                    encontrado
            }
        }
    }
    return cnt; // Retorna o nmero mximo de pares encontrados
}

int main() {
    _;

    // Leitura dos valores
    cout << "Digite o nmero de vrtices no lado esquerdo: ";
    cin >> esquerda;
    cout << "Digite o nmero de vrtices no lado direito: ";
    cin >> direita;
    cout << "Digite o nmero de conexes: ";
    cin >> ligacoes;

    g.assign(esquerda + 1, vector<int>());
    match.assign(esquerda + direita + 1, 0);
    dist.assign(esquerda + 1, 0);

    cout << "Digite as conexes (vrtice da esquerda e vrtice
        da direita):" << endl;
    for (int i = 0; i < ligacoes; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v + esquerda); // Adiciona a conexo ao
            grafo (ajustando o ndice para o lado direito)
    }

    int resultado = hopcroft_karp(); // Calcula o nmero mximo
        de pares possveis
    cout << "Nmero  mximo  de pares possveis: " << resultado
        << endl;

    // Imprime as conexes feitas
    cout << "Pares emparelhados:" << endl;
    for (int i = 1; i <= esquerda; i++) {
        if (match[i] != 0) {
            cout << "Vrtice da esquerda " << i << " est
                emparelhado com o vrtice da direita " <<
                match[i] - esquerda << endl;
        }
    }

    return 0;
}
```

4.8 bipartido

```
/*
bipartido

descricao:
 checa se o grafo eh bipartivel.
```

```
complexidade:
 o(n);
*/
function<bool(int, int)> dfs = [&](int u, int c) {
 cor[u] = c;
 for (int v : g[u]) {
  if (cor[v] == -1) {
   if (!dfs(v, !c)) {
    return 0;
   }
  }
  else if (cor[v] == c) return 0;
 }
 return 1;
};

for (int i = 0; i < n; i++) {
 if (cor[i] == -1) {
  if (!dfs(i, 1)) return cout << "IMPOSSIBLE" << endl, 0;
 }
}
```

5 matematica

5.1 crivoEratostenes

```
#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'

#define f first
#define s second

#define vi vector<int>
#define grafo vector<vector<int>>

#define dbg(x) cout << #x << " = " << x << endl;

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

const int MAX = 1e6;

bool eh_primo[MAX];

//Crivo Basico para achar todos os primos
// Complexidade O(nlnln*sqrt(n) + o(n))
void crivo(){
       memset(eh_primo , true , sizeof eh_primo);

       eh_primo[0] = eh_primo[1] = false;

       for(int i = 2 ; i * i <= MAX ; i++){
              if(eh_primo[i]){
                     for(int j = i * i ; j <= MAX ; j += i)
                         eh_primo[j] = false;
                }
           }
   }

//Versao de crivo para pegar todos os divisores primos
//Complexidade O(nloglogn)

int primo[MAX];

void crivo2(){
      memset(primo , -1 , sizeof primo);
      primo[0] = 0;

      for(int i = 2 ; i <= MAX ; i++){
             if(primo[i] == -1){//Caso o num seja primo,
                  marca
                           //tds os nums divisiveis por
                               ele
                for(int j = i ; j <= MAX ; j+=i)
                    if(primo[j] == -1)
                        primo[j] = i;
                }
          }
   }

void fatorizacao(int num){
       //Guarda a menor fatorizacao para qualquer numero
       vector<int>fatores;
       while(num != 1){
              fatores.push_back(primo[num]);
              num = num / primo[num];
          }
      for(auto& nums : fatores) cout << nums << " ";
      cout << endl;
   }

int main(){ _
//    crivo();
  //  if(eh_primo[11]) cout << 11 << " eh primo\n";
    //if(!eh_primo[4]) cout << 4 << " nao eh primo\n";

    crivo2();
    fatorizacao(12);
 return 0;
}
```

5.2 divisores

```
#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'

#define f first
#define s second

#define vi vector<int>
#define grafo vector<vector<int>>

#define dbg(x) cout << #x << " = " << x << endl;

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

ll numeroDeDivisores(ll num){
    ll total = 1;
    for(int i = 2 ; (ll)i * i <= num ; i++){
           if(num % i == 0){
                  int e = 0;
                  do{
                     e++;
                     num /= i;
                      }while(num % i == 0);
                      total *= e + 1;
              }
       }
       if(num > 1)
          total *= 2;
       return total;
}

//Complexidade(N^1/3)
```

```
//Nota: precisa de crivo para a execucao
int numeroDevisores(int num){
    int ans = 1;

    for(auto& p : primos){
           if(p*p*p > num) break;
           int cont = 1;
           while(num % p == 0){
                   num /= p;
                   cont++;
               }
           ans *= cont;
    }
    if(primo[num])
        ans *= 2;
    else if((int)sqrt(num) == sqrt(num) and primo[(int)sqrt(
         num)]){
        ans *= 3;
    }
    else if(num != 1)
        ans *= 4;
    return ans;
}

ll somaDosDivisores(ll num){
    ll total = 1;

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

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

       return total;
}

int main(){ _
    cout << numeroDeDivisores(4) << endl;
    cout << somaDosDivisores(6) << endl;
 return 0;
}
```

5.3 millerRabin

```
/*
miller-rabin

descricao:
 checa se o numero eh primo em o(log(n))
 normalmente depende de probabilidade, mas
 com esses primos hardcodes funciona ate
 n <= 3*10^18
*/

ll mul(ll a, ll b, ll m) {
 ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
 return ret < 0 ? ret+m : ret;
}

ll pow(ll x, ll y, ll m) {
 if (!y) return 1;
 ll ans = pow(mul(x, x, m), y/2, m);
 return y%2 ? mul(x, ans, m) : ans;
}

bool prime(ll n) {
 if (n < 2) return 0;
 if (n <= 3) return 1;
 if (n % 2 == 0) return 0;
 ll r = __builtin_ctzll(n - 1), d = n >> r;

  // com esses primos, o teste funciona garantido para n <=
      2^64
 // funciona para n <= 3*10^24 com os primos ate 41
 for (int a : {2, 325, 9375, 28178, 450775, 9780504,
      1795265022}) {
  ll x = pow(a, d, n);
  if (x == 1 or x == n - 1 or a % n == 0) continue;

  for (int j = 0; j < r - 1; j++) {
   x = mul(x, x, n);
   if (x == n - 1) break;
  }
  if (x != n - 1) return 0;
 }
 return 1;
}
```

5.4 mint

```
/*
aritmetica modular

descricao:
 da pra usar mint como se fosse int normal

complexidades:
 tudo o(1) menos '/' e pow que eh alguma coisa
 esquisita
*/

template<int mod>
struct modInt {
   unsigned x;
   modInt() : x(0) { }
 modInt(signed sig) : x((sig%mod + mod)%mod) { }
 modInt(signed long long sig) : x((sig%mod + mod)%mod) { }
 int get() const { return (int)x; }
 modInt pow(ll p) {
  modInt res = 1, a = *this;
  while (p) {
   if (p & 1) res *= a;
   a *= a; p >>= 1;
  }
  return res;
 }

  modInt &operator+=(modInt that) { if ((x += that.x) >= mod
       ) x -= mod; return *this; }
 modInt &operator-=(modInt that) { if ((x += mod - that.x)
      >= mod) x -= mod; return *this; }
 modInt &operator*=(modInt that) { x = (__int128)x * that.x
      % mod; return *this; }
 modInt &operator/=(modInt that) { return (*this) *= that.
      pow(mod - 2); }

 modInt operator+(modInt that) const { return modInt(*this)
      += that; }
 modInt operator-(modInt that) const { return modInt(*this)
      -= that; }
 modInt operator*(modInt that) const { return modInt(*this)
      *= that; }
 modInt operator/(modInt that) const { return modInt(*this)
      /= that; }
 bool operator<(modInt that) const { return x < that.x; }
}
```

```
 bool operator>(modInt that) const { return x > that.x; }
 friend ostream& operator<<(ostream &os, modInt a) { os << a
     .x; return os; }
 friend istream& operator>>(istream &is, modInt &a) { is >>
     a.x; return is; }
};
typedef modInt<1000000007> mint;
```

6 misc

6.1 intersection

```
/**
 * Este programa resolve o problema de encontrar intersees
     entre segmentos horizontais e verticais
 * em um plano 2D. Utiliza uma combinao de compresso de
     coordenadas e uma rvore binria indexada (BIT)
 * para alcanar uma soluo eficiente.
 *
 * A abordagem geral a seguinte:
 * 1. L os segmentos de entrada na main.
 * 2. Para cada segmento lido, so gerados eventos:
 *    - Incio e fim de segmentos horizontais.
 *    - Segmentos verticais.
 * 3. Comprima as coordenadas y para reduzir o espao de
     coordenadas e trabalhar eficientemente.
 * 4. Ordena os eventos por coordenada x.
 * 5. Usa um BIT para manter e consultar a contagem de
     segmentos horizontais ativos durante a varredura dos
     eventos.
 * 6. Calcula e imprime o nmero de intersees entre os
     segmentos horizontais e verticais.
 */

#include <bits/stdc++.h>
#define f first
#define s second
#define _ ios_base::sync_with_stdio(0);cin.tie(0); // Macros
     para otimizao de I/O
#define endl '\n' // Definio de endl para melhorar
    performance de sada
#define dbg(x) cout << #x << " = " << x << endl // Macro
    para debug

typedef long long ll; // Definio de tipo para nmeros
    inteiros longos
using namespace std;

typedef pair<int,int> point; // Tipo de dado para
    representar um ponto (coordenadas x e y)
struct event {
    point p1, p2; // Pontos que definem o evento
    int type; // Tipo de evento (0: incio de segmento
        horizontal, 1: fim de segmento horizontal, 2:
        segmento vertical)
    event() {}; // Construtor padro
    event(point p1, point p2, int type) : p1(p1), p2(p2),
        type(type) {}; // Construtor com parmetros
};

const int MAX = 1e6+10; // Tamanho mximo pr-definido para
    arrays
int n, e; // Variveis para o nmero de segmentos e eventos
event events[MAX]; // Array para armazenar os eventos
vector<int> coords; // Vetor para armazenar as coordenadas y
     nicas
map<int, int> compress; // Mapa para mapear coordenadas
    comprimidas para ndices do BIT
int bit[MAX]; // BIT (Binary Indexed Tree) para manter
    contagem de segmentos ativos

// Funo de comparao para ordenao dos eventos
bool compare(event a, event b) {
    if (a.p1.f == b.p1.f)
        return a.type < b.type; // Se coordenadas x so iguais
            , ordena por tipo (0 antes de 1)
    return a.p1.f < b.p1.f; // Ordena por coordenada x
}

// Funo para atualizar o BIT
void update(int idx, int val) {
    while (idx < MAX) {
        bit[idx] += val; // Adiciona val ao BIT na posio idx
        idx += idx & -idx; // Atualiza idx para o prximo
            ndice no BIT
    }
}

// Funo para consultar a soma acumulada no BIT at o
    ndice idx
int query(int idx) {
    int sum = 0;
    while (idx > 0) {
        sum += bit[idx]; // Soma o valor do BIT na posio idx
        idx -= idx & -idx; // Atualiza idx para o ndice
            anterior no BIT
    }
    return sum;
}

// Funo principal para calcular o nmero de intersees
ll hv_intersection() {
    ll cont = 0; // Varivel para contar intersees
    for (int i = 0; i < e; ++i) { // Itera sobre todos os
        eventos
        event c = events[i]; // Obtm o evento atual
        if (c.type == 0) { // Se o incio de um segmento
            horizontal
            update(compress[c.p1.s], 1); // Adiciona um ao
                BIT na posio correspondente coordenada y
        } else if (c.type == 1) { // Se o fim de um segmento
            horizontal
            update(compress[c.p2.s], -1); // Remove um do BIT
                na posio correspondente coordenada y
        } else { // Se um segmento vertical
            // Calcula o nmero de segmentos horizontais
                ativos entre as coordenadas y do segmento
                vertical
            cont += query(compress[c.p2.s]) - query(compress[
                c.p1.s] - 1);
        }
    }
    return cont; // Retorna o nmero total de intersees
        encontradas
}

int main() {
    _
    cin >> n; // L o nmero de segmentos

    e = 0;
    for (int i = 0; i < n; ++i) {
        int p1x, p1y, p2x, p2y;
        cin >> p1x >> p1y >> p2x >> p2y; // L as coordenadas
            do segmento

        // Gera os eventos para o segmento atual
        if (p1x == p2x) { // Se o segmento vertical
            events[e++] = event(make_pair(p1x, min(p1y, p2y))
                , make_pair(p2x, max(p1y, p2y)), 2);
            coords.push_back(min(p1y, p2y)); // Adiciona
                coordenada y mnima
            coords.push_back(max(p1y, p2y)); // Adiciona
                coordenada y mxima
        } else { // Se o segmento horizontal
            events[e++] = event(make_pair(min(p1x, p2x), p1y)
                , make_pair(max(p1x, p2x), p2y), 0);
```

```
            events[e++] = event(make_pair(max(p1x, p2x), p1y)
                 , make_pair(min(p1x, p2x), p2y), 1);
            coords.push_back(p1y); // Adiciona coordenada y
                 do primeiro ponto
            coords.push_back(p2y); // Adiciona coordenada y
                 do segundo ponto
        }
    }

    sort(coords.begin(), coords.end()); // Ordena as
         coordenadas
    coords.erase(unique(coords.begin(), coords.end()), coords
         .end()); // Remove duplicatas

    for (int i = 0; i < coords.size(); ++i) {
        compress[coords[i]] = i + 1; // Mapeia cada
             coordenada para um ndice no BIT
    }

    sort(events, events + e, compare); // Ordena os eventos
         por coordenada x

    cout << hv_intersection() << endl; // Calcula e imprime o
          nmero de intersees entre os segmentos

    return 0; // Retorna 0 para indicar sucesso
}
```

7 problemas

7.1 berntown

```
/*
berntown roads

explicacao:
 transforme um grafo nao direcionado em um
 direcionado mas o mantenha conexo (da pra
 ir de qualquer um pra qualquer outro. o
 mais importante aqui sao os bridges. uma
 bridge eh um vertice que, se tirado,
 aumenta o numero de componentes conexos
*/
int main()
{
    _;
 // todas as span-edges vao pra baixo e as back-edges pra
      cima.
 // mas se tiver alguma bridge eh impossivel.

 int n, m; cin >> n >> m;
 map<pair<int, int>, int> ed;
 vector<vector<int>> g(n);
 for (int i = 0, u, v; i < m; i++) {
  cin >> u >> v; u--, v--;
  g[u].push_back(v);
  g[v].push_back(u);
  ed[ii(u, v)] = 0;
 }

 int cont = 0;
 vector<int> dp(n, 0), d(n, -1); d[0] = 0;
 function<void(int)> bridge = [&](int u) {
  for (int v : g[u]) {
   if (d[v] == -1) {
    d[v] = d[u] + 1;
    bridge(v);
    dp[u] += dp[v];
    if (ed.count(ii(u, v))) ed[ii(u, v)] = 1;
    else ed[ii(v, u)] = 1;
   } else if (d[v] < d[u]) {
    dp[u]++;
   } else if (d[v] > d[u]) {
    dp[u]--;
   }
  }

  dp[u]--;
  if (d[u] and !dp[u]) cont++;
 };
 bridge(0);

 if (cont) return cout << 0 << endl, 0;

 for (auto& [p, dir] : ed) {
  int baixo = (d[p.f] < d[p.s] ? p.f : p.s), alto = baixo ^
       p.f ^ p.s;
  int a = (dir ? baixo : alto), b = a ^ p.f ^ p.s;
  a++, b++;

  cout << a << ' ' << b << endl;
 }

    return(0);
}
```

7.2 forest

```
/*
forest queries

explicacao:
 prefix sum em 2d. quer saber quantas arvores
 no retangulo definido dentro da array;
*/

int main()
{
    _;

 int n, q; cin >> n >> q;
 vector<vector<int>> mp(n, vector<int> (n)), pre(n + 1,
      vector<int>(n + 1));

 for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++)
  {
   char c; cin >> c;
   mp[i][j] = (c == '*');
  }

 for (int i = 1; i <= n; i++)
  for (int j = 1; j <= n; j++)
   pre[i][j] = mp[i - 1][j - 1] + pre[i - 1][j] + pre[i][j -
        1] - pre[i - 1][j - 1];

 while (q--)
 {
  int y1, x1, y2, x2;
  cin >> y1 >> x1 >> y2 >> x2;
  cout << pre[y2][x2] - pre[y2][x1 - 1] - pre[y1 - 1][x2] +
       pre[y1 - 1][x1 - 1] << endl;
 }

    return(0);
}
```

7.3 gorilas

```
/*
gorillas

explicacao:
 atualiza todos os quadrados de tamanho k
 em uma matriz maior. legal porque da pra
```

```
 atualizar um retangulo todo em o(1).
*/
void solve() {
    int n, m, k; cin >> n >> m >> k;
 int w; cin >> w;
    vector<int> v(w);
    for (int& i : v) cin >> i;
    sort(v.rbegin(), v.rend());

    vector<vector<ll>> cont(n + 1, vector<ll>(m + 1, 0));
    for (int i = 0; i + k <= n; i++) {
        for (int j = 0; j + k <= m; j++) {
            cont[i][j]++;
            cont[i + k][j]--;
            cont[i][j + k]--;
            cont[i + k][j + k]++;
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (i > 0) cont[i][j] += cont[i - 1][j];
            if (j > 0) cont[i][j] += cont[i][j - 1];
            if (i > 0 && j > 0) cont[i][j] -= cont[i - 1][j -
                 1];
        }
    }

    priority_queue<ll> pq;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            pq.push(cont[i][j]);
   //cout << cont[i][j] << ' ';
        }
  //cout << endl;
    }

    ll res = 0;
 for (int i : v) {
  res += (ll)i * (ll)pq.top();
  pq.pop();
 }

    cout << res << endl;
}

int main() {
    _;

    int t; cin >> t;
    while (t--) {
        solve();
    }

    return 0;
}
```

7.4 misere

```
/*
misere nim

explicacao:
 basicamente nim normal mas quem ganha eh
 que pega o ultimo e nao o penultimo. a sol
 eh xor normal mas se tudo for 1 o primeiro
 ganha tbm;
*/
int main()
{
    _;

 int t; cin >> t;
 while (t--) {
  int n, res = 0, uns = 0; cin >> n;
  for (int i = 0, si; i < n; i++) {
   cin >> si; res ^= si;
   if (si == 1) uns++;
  }
  cout << (res or uns == n ? "First" : "Second") << endl;
 }

    return(0);
}
```

7.5 moneysums

```
/*
money sums

descricao:
 quais valores eu consigo fazer usando qualquer
 quantidade distinta de notas. em o(nk).
*/
int main()
{
    _;

    int n, k = 0;
    cin >> n;
    vector<int> v(n);
 for (int i = 0; i < n; i++) {
  cin >> v[i], k += v[i];
 }

 vector<vector<int>> dp(n + 1, vector<int> (k + 1));
 // dp[i][j]: 1 se da pra fazer valor j usando os i
      primeiros valores.

 dp[0][0] = 1;
 for (int i = 1; i <= n; i++) {
  for (int j = 0; j <= k; j++) {
   dp[i][j] = dp[i - 1][j];
   int ult = j - v[i - 1];
   if (ult >= 0 and dp[i - 1][ult]) {
    dp[i][j] = 1;
   }
  }
 }

 vector<int> res;
 for (int i = 1; i <= k; i++) {
  if (dp[n][i]) {
   res.push_back(i);
  }
 }

 cout << res.size() << endl;
 for (int i : res)
  cout << i << ' ';
 cout << endl;

    return 0;
}
```

7.6 periodicstrings

```
/*
periodic strings

explicacao:
 quais tamanhos de prefixo que podem 'resumir'
 a string (se repetir o prefixo, reconstruo a
 string igual toda). legal por causa do hash na
 string e ainda o hash em [l, r] mt brabo
*/
```

```cpp
#include <bits/stdc++.h>
#define f first
#define s second
#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
using namespace std;

const ll P = 69420;
const ll M = 1e9+9;

int main() {
    _;

    string s; cin >> s;
    int n = s.size();

    vector<ll> base(n), pre(n + 1);
    base[0] = 1;
    for (int i = 1; i < n; i++) {
        base[i] = base[i - 1] * P % M;
    }

    for (int i = 1; i <= n; i++) {
        pre[i] = (pre[i - 1] * P + s[i - 1]) % M;
    }

    auto hash = [&] (int l, int r) -> ll {
        ll h = (pre[r + 1] - base[r - l + 1] * pre[l] % M + M
             ) % M;
        return h;
    };

    for (int i = 0; i < n; i++) {
        int j = 0, ok = 1;
        while (j < n) {
            int tam = min(i + 1, n - j);
            if (hash(0, tam - 1) != hash(j, j + tam - 1)) {
                ok = 0;
                break;
            }
            j += tam;
        }
        if (ok) {
  cout << i + 1 << ' ';
  break;
 }
  }
    cout << endl;

    return 0;
}
```

7.7 snim

```cpp
/*
s-nim

explicacao:
 quase um nim normal, mas pra cada posicao eu so posso
 algum valor em s[]. o exemplo mais simples pra uso de
 mex. lembra que uma posicao eh ganha em nim se o ^ de
 tudo nao eh 0.
*/

int main()
{
    _;

 int k; cin >> k;
 vector<int> s(k);
 for (int& i : s) cin >> i;

 // calculo dos grundy para cada heap possivel
 // nao eh um nim simples, entao o grundy number permite que
       eu reduza o jogo pra um nim que eu conheco.
 vector<int> grundy(MAX + 1, 0);
 for (int i = 1; i <= MAX; i++) {
  set<int> chega;
  for (int si : s) {
   if (i >= si) {
    chega.insert(grundy[i - si]);
   }
  }
  int mex = 0;
  while (chega.count(mex)) {
   mex++;
  }
  grundy[i] = mex;
 }

 int m; cin >> m;
 while (m--) {
  int l, res = 0; cin >> l;
  for (int i = 0, x; i < l; i++) {
   cin >> x; res ^= grundy[x];
  }
  cout << (res ? "W" : "L");
 }
 cout << endl;

    return(0);
}
```

7.8 teletransporte

```cpp
/*
teletransporte

explicacao:
 todos os caminhos de tamanho l entre
 nodo a e b. exponencia a matriz de adj
 por l.
*/
matriz matrixMultiply (const matriz& a, const matriz& b, int
      n) {
 matriz c(n, vector<int>(n, 0));
 for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
   for (int k = 0; k < n; k++) {
    c[i][j] = (c[i][j] + a[i][k] * b[k][j]) % mod;
   }
  }
 }
 return c;
}

matriz matrixExpo (matriz& a, ll k, int n) {
 matriz res(n, vector<int>(n, 0));
 for (int i = 0; i < n; i++) res[i][i] = 1;

 while (k > 0) {
  if (k & 1) res = matrixMultiply(res, a, n);
  a = matrixMultiply(a, a, n);
  k >>= 1;
 }
 return res;
}

int main()
{
    _;

 int n, s, t;
 ll l;
 while (cin >> n >> l >> s >> t) {
```

```
  s--, t--;
  matriz adj(n, vector<int>(n));
  for (int u = 0; u < n; u++) {
   for (int j = 0; j < 4; j++) {
    int v; cin >> v; v--;
    adj[u][v]++;
   }
  }

  matriz res = matrixExpo(adj, l, n);
  cout << res[s][t] << endl;
 }

    return(0);
}
```

7.9 treedistances

```
/*
tree distances I

explicacao:
 distancia pro vertice mais distante pra
 cada vertice do grafo. tres dfs pra achar
 diametro (a, b) do grafo.
*/
int main()
{
    _;

 int n; cin >> n;
 vector<vector<int>> g(n);
 for (int i = 0, u, v; i < n-1; i++) {
  cin >> u >> v; u--, v--;
  g[u].push_back(v);
  g[v].push_back(u);
 }

 auto dfs = [&](int src, vector<int>& d) {
  stack<int> s; s.push(src);
  d[src] = 0;

  int longe = 0, onde = 0;
  while (!s.empty()) {
   int u = s.top(); s.pop();

   for (int v : g[u]) {
    if (d[v] != -1) continue;
    s.push(v);
    d[v] = d[u] + 1;
    if (d[v] > longe) {
     longe = d[v];
     onde = v;
    }
   }
  }
  return make_pair(onde, longe);
 };

 vector<int> dx(n, -1), da(n, -1), db(n, -1);
 dfs(dfs(dfs(0, dx).f, da).f, db);

 for (int i = 0; i < n; i++) {
  cout << max(da[i], db[i]) << ' ';
 }
 cout << endl;

    return(0);
}
```

8 string

8.1 GeralSufixArray

```
#include <bits/stdc++.h>
#define f first
#define s second
#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define dbg(x) cout << #x << " = " << x << endl
typedef long long ll;
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
using namespace std;

// Estrutura para armazenar  informaes  de um sufixo
struct suffix
{
    int index; // Para armazenar o ndice original
    int rank[2]; // Para armazenar os ranks e o par do prximo
         rank
};

//    Funo    de    comparao    usada pelo sort() para comparar
     dois sufixos
// Compara dois pares, retorna 1 se o primeiro par for menor
int cmp(struct suffix a, struct suffix b)
{
    return (a.rank[0] == b.rank[0]) ? (a.rank[1] < b.rank[1]
        ? 1 : 0) :
                                      (a.rank[0] < b.rank[0] ? 1
                                           : 0);
}

//    Funo    principal que recebe uma string 'txt' de
     tamanho n como argumento,
//  constri  e retorna o array de sufixos para a string dada
vector<int> buildSuffixArray(string txt, int n)
{
    // Estrutura para armazenar sufixos e seus ndices
    struct suffix suffixes[n];

    // Armazena sufixos e seus ndices em um array de
        estruturas.
    // A estrutura    necessria  para ordenar os sufixos
        alfabeticamente
    // e manter seus ndices antigos durante a  ordenao
    for (int i = 0; i < n; i++)
    {
        suffixes[i].index = i;
        suffixes[i].rank[0] = txt[i] - 'a';
        suffixes[i].rank[1] = ((i + 1) < n) ? (txt[i + 1] - '
            a') : -1;
    }

    // Ordena os sufixos usando a  funo  de   comparao
        definida acima
    sort(suffixes, suffixes + n, cmp);

    // Neste ponto, todos os sufixos esto ordenados de acordo
         com os primeiros
    // 2 caracteres. Agora vamos ordenar os sufixos de acordo
         com os primeiros
    // 4 caracteres, depois 8 e assim por diante
    int ind[n]; // Este array   necessrio  para obter o
        ndice  em suffixes[]
    // a partir do ndice original. Esse mapeamento
         necessrio  para obter
    // o  prximo  sufixo.
    for (int k = 4; k < 2 * n; k = k * 2)
    {
        // Atribuindo valores de rank e ndice ao primeiro
            sufixo
        int rank = 0;
        int prev_rank = suffixes[0].rank[0];
        suffixes[0].rank[0] = rank;
```

```
    ind[suffixes[0].index] = 0;

    // Atribuindo rank aos sufixos
    for (int i = 1; i < n; i++)
    {
        // Se o primeiro rank e o prximo rank forem
            iguais aos do sufixo
        // anterior no array, atribua o mesmo novo rank a
            este sufixo
        if (suffixes[i].rank[0] == prev_rank &&
            suffixes[i].rank[1] == suffixes[i - 1].rank
                [1])
        {
            prev_rank = suffixes[i].rank[0];
            suffixes[i].rank[0] = rank;
        }
        else // Caso contrrio, incremente o rank e
            atribua
        {
            prev_rank = suffixes[i].rank[0];
            suffixes[i].rank[0] = ++rank;
        }
        ind[suffixes[i].index] = i;
    }

    // Atribuir o prximo rank a cada sufixo
    for (int i = 0; i < n; i++)
    {
        int nextindex = suffixes[i].index + k / 2;
        suffixes[i].rank[1] = (nextindex < n) ?
                              suffixes[ind[nextindex]].rank
                                  [0] : -1;
    }

    // Ordenar os sufixos de acordo com os primeiros k
        caracteres
    sort(suffixes, suffixes + n, cmp);
}

// Armazena os ndices de todos os sufixos ordenados no
    array de sufixos
vector<int> suffixArr;
for (int i = 0; i < n; i++)
    suffixArr.push_back(suffixes[i].index);

// Retorna o array de sufixos
return suffixArr;
}
```

```
/* Para construir e retornar o LCP (Longest Common Prefix)
    */
vector<int> kasai(string txt, vector<int> suffixArr)
{
    int n = suffixArr.size();

    // Para armazenar o array LCP
    vector<int> lcp(n, 0);

    // Um array auxiliar para armazenar o inverso do array de
        sufixos
    // Por exemplo, se suffixArr[0] for 5, invSuff[5]
        armazenar 0.
    // Isso    usado para obter a prxima string de sufixo do
        array de sufixos.
    vector<int> invSuff(n, 0);

    // Preencher os valores em invSuff[]
    for (int i = 0; i < n; i++)
        invSuff[suffixArr[i]] = i;

    // Inicializa o comprimento do LCP anterior
    int k = 0;

    // Processa todos os sufixos um por um comeando do
        primeiro sufixo em txt[]
    for (int i = 0; i < n; i++)
    {
        /* Se o sufixo atual est em n-1, ento   no  temos o
            prximo
        substring para considerar. Portanto, o LCP no
            definido
        para este substring, colocamos zero. */
        if (invSuff[i] == n - 1)
        {
            k = 0;
            continue;
        }

        /* j  contm  o  ndice  do  prximo  substring a ser
            considerado
        para   comparao   com o substring atual, ou seja, a
            prxima
        string no array de sufixos */
        int j = suffixArr[invSuff[i] + 1];

        //  Comea  a   comparao   diretamente a partir do
            ndice k,  j  que
        // pelo menos k-1 caracteres vo coincidir
        while (i + k < n && j + k < n && txt[i + k] == txt[j
            + k])
            k++;

        lcp[invSuff[i]] = k; // LCP para o sufixo atual.

        // Deletar o caractere inicial da string.
        if (k > 0)
            k--;
    }

    // Retorna o array LCP construdo
    return lcp;
}

void printArr(vector<int> arr, string txt)
{
    int n = arr.size();
    for (int i = 0; i < n; i++)
        cout << arr[i] << " " << txt.substr(arr[i]) << endl;
}

int main()
{
    _;

    string str;
    cin >> str;

    cout << str.size() << " " << endl;

    vector<int> suffixArr = buildSuffixArray(str, str.length
        ());
    int n = suffixArr.size();

    printArr(suffixArr, str);

    return 0;
}
```

8.2 kmp

```
/*
 kmp

 descricao:
  retorna todas as posicoes em que s ocorre em t;
  da pra fazer com memoria o(n) se ler t cada char por vez
```

```
 complexidade:
  memoria: o(|n| + |m|)
  acha: o(|n| + |m|
*/

// [...] pi;
vector<int> pi(string& s) {
    int n = (int) s.size();
    vector<int> p(n, 0);
    for (int i = 1, j = 0; i < n; i++) {
        while (j and s[i] != s[j])
            j = p[j - 1];
  j += s[i] == s[j];
        p[i] = j;
    }
    return p;
}
```

```
vector<int> kmp(string& s, string& t) {
    int n = (int)s.size(), m = (int)t.size();
    string q = s + "&" + t;
    vector<int> res, c = pi(q);
    for (int i = n + 1; i < n + m + 1; i++) {
        if (c[i] == n) res.push_back(i - 2 * n);
    }
    return res;
}
```

8.3 prefixFunction

```
/*
 prefix function

 descricao:
  retorna pi(s), sendo pi[i] o maior prefixo de s[0...i] que
  eh tambem sufixo de s[0..1]

 complexidade:
  o(|s|)
*/
vector<int> pi(string& s) {
 int n = (int) s.size();
 vector<int> p(n, 0);
 for (int i = 1, j = 0; i < n; i++) {
  while (j and s[i] != s[j])
   j = p[j - 1];

  j += s[i] == s[j];
  p[i] = j;
 }
 return p;
}
```