

**Competitive Programming - Team Notebook**

Instituto Federal de Ciência e Tecnologia do Sul de Minas

Versão 1.1 - 2018

TEMPLATE

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

// Nome de Tipos

**typedef** long ll;

**typedef** unsigned long long ull;

**typedef** long double ld;

// Valores

**#define** INF 0x3F3F3F3F

**#define** LINF 0x3F3F3F3F3F3F3F3FLL

**#define** DINF (double)1e+30

**#define** EPS (double)1e-9

**#define** RAD(x) (double)(x\*PI)/180.0

**#define** PCT(x,y) (double)x\*100.0/y

// Atalhos

**#define** f first

**#define** s second

**#define** pb push\_back

**#define** mp make\_pair

**#define** l length()

**#define** **for**n(i, n) **for** ( int i = 0; i < (n); ++i )

**#define** **for**nx(i, x, n) **for** ( int i = (x); i < (n); ++i )

using namespace std;

int main(){

freopen("input.txt", "r", stdin);

freopen("output.txt", "w", stdout);

**return** 0;

}

#FIM\_TEMPLATE

MATEMÁTICA

Divisão de números inteiros com resto negativo

int a, b, c;

int q, r;

cin >> a >> b;

q = a/b;

r = a%b;

**if**(r < 0){

int c,d;

c = (a < 0) ? a\*-1 : a;

d = (b < 0) ? b\*-1 : b;

q = (c+d)/d;

r = (c - (q \* d))\*-1;

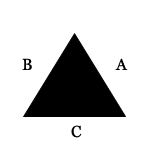
q = (a\*b > 0) ? q : q\*-1;

}

Condição existência de triângulo(As três condições devem ser satisfeitas)

**if**(a-b) < c && (a+b) > c && (a-c) < b && (a+c) > b && (a-b) < c && (a+b) > c)

b-c < a < b+c

a-c < b < a+c

a-b < c < a+b

/\* EXEMPLO DE PROGRAMA PARA VER**IF**ICAR SE TRIANGULO EXISTE E SUA CLASS**IF**ICAÇÃO \*/

int max(int a, int b){

**return** (a>b) ? a : b;

}

int min(int a, int b){

 **return** (a<b) ? a : b;

}

cin >> a >> b >> c;

// x > y > z

int x, y, z;

x = max(a, max(b,c));

z = min(a, min(b,c));

// a+b+c = soma total | -x - z = total-(maior+menor)

y = a + b + c - x - z;

**if**(x < (y + z)){

**if**(x == y && y == z){

cout << "Valido-Equilatero" << endl;

}**else** **if**(x != y && x != z && y != z){

cout << "Valido-Escaleno" << endl;

}**else**{

cout << "Valido-Isoceles" << endl;

}

// pitagoras x²=y²+z²

**if**( (x\*x) == ( (y\*y)+(z\*z) ) ){

cout << "Retangulo: S" << endl;

}**else**{

cout << "Retangulo: N" << endl;

}

}**else**{

cout << "Invalido" << endl;

}

/\* FIM DO EXEMPLO DE PROGRAMA PARA VER**IF**ICAR SE TRIANGULO EXISTE E SUA CLASS**IF**ICAÇÃO \*/

Comparação entre 2 valores tipo Double

bool comparaDouble(double val1, double val2, string cmp){

**if**(cmp == "=="){

**return** fabs(val1 - val2) < EPSILON;

}**else** **if**(cmp == "<="){

**if**(fabs(val1 - val2) < EPSILON){

**return** true;

}**else**{

**return** val1 <= val2;

}

}**else if**(cmp == ">="){

**if**(fabs(val1 - val2) < EPSILON){

**return** true;

}**else**{

**return** val1 >= val2;

}

}

}

Primo rápido – O(√n)

bool e\_primo(int x){

**if**(x == 1) **return** 0;

//note que se o número **for** 2 ele não entra no loop, comportamento desejado

**for**(int i = 2; i\*i <= x; ++i){

**if**(x % i == 0){ //se o resto de x por i **for** 0, então i divide x

**return** 0;

}

}

**return** 1;

}

Crivo de Erastótenes

// dados N e Q, com ambos menores que 10^6, teremos Q inteiros a, menores que N, e devemos responder para cada um deles se ele é primo

bool e\_composto[1000010];

void crivo(int n){

// 1 não é composto, mas o vetor na verdade guarda os números que não são primos

e\_composto[1] = 1;

**for**(int i = 2; i <= n; ++i){

**if**(!e\_composto[i]){

**for**(int j = 2; j\*i <= n; ++j){

e\_composto[i\*j] = 1;

}

}

}

**return**;

}

int main(){

int N, Q, a;

cin >> N >> Q;

crivo(N); // Complexidade O(n\*log(log(n)))

**for**(int i = 0; i < Q; ++i){ // Complexidade O(Q)

cin >> a;

// Se composto é falso, então é primo, caso contrário é composto.

cout << !e\_composto[a] << "\n";

}

**return** 0;

}

Checar se um dado bit está ligado

// Note que uma potência de 2 tem sempre apenas um bit igual a 1, dessa **for**ma se queremos saber se o bit i está igual a 1 , precisamos apenas checar se o and dele e de 2^i é d**if**erente de 0.

bool is\_set(int x, int i){

bool ret = ((x&(1 << i)) != 0);

**return** ret;

}

Extrair o bit menos significante

// O bit menos sign**if**icativo de um número é menor bit de um número igual a 1, chamamos ele de lsb. Exemplo:

// lsb(20) = lsb(10100) = 100 = 4

int lsb(int x){

**return** x&-x;

}

Contar o número de bits iguais a 1

int count\_bits(int x){

int ret = 0;

**while**(x != 0){

++ret;

x -= x&-x;

}

**return** ret;

}

Checar se um número é potência de 2

bool is\_power\_of\_two(int x){

**if**(x == 0) **return** 0;

**return** ((x&(x - 1)) == 0)

}

Ligar um bit em um número

// É bem simples, basta o número receber ele or 2 elevado ao bit que queremos setar

int x, i;

cin >> x >> i;

x |= (1 << i);

Desligar o bit

int x, i;

cin >> x >> i;

x |= (1 << i); // Primeiro eu ligo o bit, caso ele esteja desligado

x ^= (1 << i); // Depois desligo o bit

**OR nos Bits ( | )**

// a = 10010; b = 01110; a|b = 11110

**AND nos bits ( & )**

// a = 10110; b = 10011; a&b = 10010

**XOR nos bits ( ^ )**

// a = 10110; b = 10011; a^b = 00101

**Shift Esquerdo ( << )**

// a = 1; a = a << 8; a = 256, que em binário é 100000000

**Shitf Direito ( >> )**

// b = 260; b >>= 3; b = 32, que em binário é 100000

**Arredondamento para cima**

ceil(numero)

**Número de casas decimais de um número**

ceil(log10(numero+1))

**Zerar conteúdo de um array 2d**

memset(array, 0, sizeof(array[0][0]) \* n \* n)

**Zerar conteúdo de um array 1d**

memset(array, 0, sizeof(array))

**Quando dividir dois números float ou double colocar o .0 mesmo que não tenha casas depois** **da vírgula. Caso contrário a divisão não vai ser efetuada corretamente**

// 1/6=0

// 1.0/6.0=0,1666667

**Conversão inteiro para hexadecimal**

cout << hex << v

// Considerando v um inteiro

**"scientific" adiciona notação científica ao cout**

cout << scient**if**ic => 5e+2;

**"fixed" adiciona casas decimais fixas no número(deve ser usado com setprecision)**

cout << fixed << setprecision(2); => 5.00;

**Volume do cilindro**

pi \* r² \* h

**Área Total**

A = Ab + Al = 2πr(2+h)

Ab = 2·πr2

Al = 2πr·h

**Somatório de Feynman para saber quantos quadrados diferentes existem em um quadriculado de N x N quadrados**

(n\*(n+1)\*((2\*n)+1))/6

**Somatório de um intervalo [a,b] inclusivo**

((a + b) \* (b - a + 1)) / 2

**Distância entre 2 pontos**

sqrt(pow((xf-xi),2) + pow((yf-yi),2))

**Conversão cartesiano para polar**

// r =

// ø = tg-1 b/a

**Conversão polar para cartesiano**

// a = r cos ø

// b = r sem ø

**Número de permutações de um conjunto**

// dados um grupo de 4 pessoas, de quantas **for**mas podemos colocá-los em fila?

// P(n, k) = n!/(n-k)!

// k = número de elementos para permuta; n = número total de elementos

**Número de combinações de um conjunto**

// x = n!/(n-k)!k!

**Tricks do cmath**

// Quando um número **for** muito grande usar powl ao invés de pow. powl terá mais precisão

powl(a, b)

(int)round(p, (1.0/n)) // nth raíz de p

**Máximo entre dois números**

int max(int a, int b) { **return** a>b ? a:b; }

**Mínimo entre dois números**

int min(int a, int b) { **return** a<b ? a:b; }

**Números primos menores que 100**

// there are 25 numbers

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,

41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

**Maior divisor comum – GCD**

int gcd(int a, int b)

{

**if** (b==0) **return** a;

**else** **return** gcd(b, a%b);

}

**Menor divisor comum – LCM**

int lcm(int a, int b)

{

**return** a\*b/gcd(a,b);

}

**Ab mod p**

long powmod(long base, long exp, long modulus) {

base %= modulus;

long result = 1;

**while** (exp > 0) {

**if** (exp & 1) result = (result \* base) % modulus;

base = (base \* base) % modulus;

exp >>= 1;

}

**return** result;

}

**n! mod p**

int factmod (int n, int p) {

long long res = 1;

**while** (n > 1) {

res = (res \* powmod (p-1, n/p, p)) % p;

**for** (int i=2; i<=n%p; ++i)

res=(res\*i) %p;

n /= p;

}

**return** int (res % p);

}

**Fatorização de primos**

vector<int> factors(int n) {

vector<int> f;

**for** (int x = 2; x\*x <= n; x++) {

**while** (n%x == 0) {

f.push\_back(x);

n /= x;

}

}

**if** (n > 1) f.push\_back(n);

**return** f;

}

#FIM\_MATEMATICA

LIMITES C++

|  |  |  |
| --- | --- | --- |
| **name** | **expresses** | **value\*** |
| CHAR\_BIT | Number of bits in a char object (byte) | 8 or greater\* |
| SCHAR\_MIN | Minimum value **for** an object of type signed char | -127 (-27+1) or less\* |
| SCHAR\_MAX | Maximum value **for** an object of type signed char | 127 (27-1) or greater\* |
| UCHAR\_MAX | Maximum value **for** an object of type unsigned char | 255 (28-1) or greater\* |
| CHAR\_MIN | Minimum value **for** an object of type char | either SCHAR\_MIN or 0 |
| CHAR\_MAX | Maximum value **for** an object of type char | either SCHAR\_MAX or UCHAR\_MAX |
| MB\_LEN\_MAX | Maximum number of bytes in a multibyte character, **for** any locale | 1 or greater\* |
| SHRT\_MIN | Minimum value **for** an object of type short int | -32767 (-215+1) or less\* |
| SHRT\_MAX | Maximum value **for** an object of type short int | 32767 (215-1) or greater\* |
| USHRT\_MAX | Maximum value **for** an object of type unsigned short int | 65535 (216-1) or greater\* |
| INT\_MIN | Minimum value **for** an object of type int | -32767 (-215+1) or less\* |
| INT\_MAX | Maximum value **for** an object of type int | 32767 (215-1) or greater\* |
| UINT\_MAX | Maximum value **for** an object of type unsigned int | 65535 (216-1) or greater\* |
| LONG\_MIN | Minimum value **for** an object of type long int | -2147483647 (-231+1) or less\* |
| LONG\_MAX | Maximum value **for** an object of type long int | 2147483647 (231-1) or greater\* |
| ULONG\_MAX | Maximum value **for** an object of type unsigned long int | 4294967295 (232-1) or greater\* |
| LLONG\_MIN | Minimum value **for** an object of type long long int | -9223372036854775807 (-263+1) or less\* |
| LLONG\_MAX | Maximum value **for** an object of type long long int | 9223372036854775807 (263-1) or greater\* |
| ULLONG\_MAX | Maximum value **for** an object of type unsigned long long int | 18446744073709551615 (264-1) or greater\* |

// the actual value depends on the particular system and library implementation, but shall reflect the limits of these types in the target plat**for**m.

//LLONG\_MIN, LLONG\_MAX and ULLONG\_MAX are defined **for** libraries complying with the C standard of 1999 or later (which only includes the C++ standard since 2011: C++11).

#FIM\_LIMITES\_C++

STRINGS

Dividir uma string de acordo com um token

std::string s = "scott>=tiger>=mushroom";

std::string delimiter = ">=";

size\_t pos = 0;

std::string token;

**while** ((pos = s.find(delimiter)) != std::string::npos) {

token = s.substr(0, pos);

std::cout << token << std::endl;

s.erase(0, pos + delimiter.length());

}

std::cout << s << std::endl;

**Uppercase funciona em um cout**

cout << uppercase << "a"

**streamstream utilizado para converter strings**

stringstream ss(line) // Adiciona a string line ao stringstream

ss >> x // converte a string em um float(x = long double)

**Inversão de números pode ser feita utilizado 2 strings e invertendo suas posições(ou pegando %10 e fazendo \*10)**

Conversões

**String para int**

stoi(string, 0, 10)

**String para long long**

stoll(string, 0, 10)

**String para unsigned int**

stoul(string, 0, 10)

**String para unsigned long long**

stoull(string, 0, 10)

**Char para int**

var\_char - 48 ou ((var\_char - '0') % 48)

**Int para String**

int a = 10;

stringstream ss;

ss << a;

string str = ss.str();

**Caracteres minúsculo**

tolower(char)

**Caracteres maiúsculo**

toupper(char)

**Apagar um intervalo de uma string**

n.erase(pos\_inicio, pos\_fim);

// pos\_inicio e pos\_fim são inteiros que representam posições

**Remover um caracter de toda a string**

n.erase(remove(n.begin(), n.end(), caracter\_a\_ser\_removido), n.end());

// caracter\_a\_ser\_removido representa uma variável do tipo char, com o caracter à ser removido da string

**Verificar se uma string está vazia**

n.empty() // Retona true ou false

**Inverter String**

reverse(str1.begin(), str1.end());

**Criar uma nova string a partir de um intervalo de outra string**

string str1 = line.substr(0,meio);

**Verificar se caracter está entre [A-z]**

(line[i] >= 65 && line[i] <= 90) || (line[i] >= 97 && line[i] <= 122)

**Busca**

unsigned int find(**const** string &s2, unsigned int pos1 = 0);

unsigned int rfind(**const** string &s2, unsigned int pos1 = end);

unsigned int find\_first\_of(**const** string &s2, unsigned int pos1 = 0);

unsigned int find\_last\_of(**const** string &s2, unsigned int pos1 = end);

unsigned int find\_first\_not\_of(**const** string &s2, unsigned int pos1 = 0);

unsigned int find\_last\_not\_of(**const** string &s2, unsigned int pos1 = end);

**Insert, Erase, Replace**

string& insert(unsigned int pos1, **const** string &s2);

string& insert(unsigned int pos1, unsigned int repetitions, char c);

string& erase(unsigned int pos = 0, unsigned int len = npos);

string& replace(unsigned int pos1, unsigned int len1, **const** string &s2);

string& replace(unsigned int pos1, unsigned int len1, unsigned int repetitions, char c);

**String streams**

stringstream s1;

int i = 22;

s1 << "Hello world! " << i;

cout << s1.str() << endl;

#FIM\_STRINGS

ESTRUTURAS

**Verificar se elemento existe em um vetor**

find(uniao.begin(), uniao.end(), 1) != uniao.end()

// Esse código ver**if**ica se o número 1 existe no vetor uniao (retorna true se existe, falso se não existe)

**Código para apagar elementos duplicados em um vetor**

sort( uniao.begin(), uniao.end() );

uniao.erase( unique( uniao.begin(), uniao.end() ), uniao.end() );

**Ordenar vector**

sort(notas.begin(), notas.end());

**Ordenar vetor forma decrescente**

sort(p.begin(), p.end(), greater<int>());

**Excluir primeiro elemento de um vector**

notas.erase(notas.begin());

**Excluir ultimo elemento de um vector**

notas.pop\_back();

**Alterar tamanho de um vector**

V.resize(10); //Muda o tamanho do vector V para 10.

**Busca em um vector**

iterator find(iterator first, iterator last, **const** T &value);

iterator find\_**if**(iterator first, iterator last, **const** T &value, TestFunction test);

bool binary\_search(iterator first, iterator last, **const** T &value);

bool binary\_search(iterator first, iterator last, **const** T &value, LessThanOrEqualFunction comp);

**Deque array dinâmico que funciona como vector mas, tem os métodos push\_front() e pop\_front()**

**Definição de um pair**

pair<string, int> P

**Leitura de um pair**

cin>>P.first>>P.second

**Utilizando pair de pair**

pair<string, pair<double, double>> P; //Cria uma variável pair

P.first = "Joao"; //Nome de um aluno

P.second.first = 8.2; //Primeira nota do aluno

P.second.second = 10; //Segunda nota do aluno

**Criando pair com dois valores**

make\_pair(a,b)

**Fila**

queue<int> fila; // Declaração da fila

fila.push(10); // Adicionando um elemento ao final da fila

fila.pop(); // Retira o primeiro elemento

fila.front(); // Retorna o primeiro elemento da fila

fila.empty(); // Ver**if**ica se a fila está vazia

**Pilha**

stack<int> pilha; // Declaração da pilha

pilha.push(10); // Adicionado um elemento ao topo da pilha

pilha.pop(); // Retira o elemento do topo da pilha

pilha.top(); // Retorna o elemento do topo da pilha

pilha.empty(); // Ver**if**ica se a pilha está vazia

**SET**

// busca, inserção e deleção em complexidade O(log n)

// Mantém os elementos ordenados e não permite elementos duplicados

set<int> S; // Declaração do SET

S.insert(10); // Adiciona um elemento

**if**(S.find(3) != S.end()) // Se 3 está no conjunto

S.erase(10); // Apaga o elemento do SET

// clear(): Apaga todos os elementos.

// size(): Retorna a quantidade de elementos.

// begin(): Retorna um ponteiro para o inicio do set

// end(): Retorna um ponteiro para o final do set

**Map**

// Map é uma variação da estrutura set e sua implementação também é feita utilizando Red- Black Trees. A principal d**if**erença entre um set e um map é o segundo armazena os conjuntos chave, valor e o primeiro apenas chave.

map<string, int> M; // Declaração

M.insert(make\_pair("Alana", 10)); //Inserimos uma variável do tipo pair diretamente no map

M["Alana"] = 10; // Relacionando o valor 10 à chave "Alana"

**if**(M.find("Alana") != M.end()) //Se a chave "Alana" foi inserida no map

cout<<M["Alana"]<<"\n"; //Imprime o valor da chave "Alana", no caso, o valor 10.

M.erase("Alana"); //Apaga o elemento que possui a chave "Alana" do map

// clear(): Apaga todos os elementos.

// size(): Retorna a quantidade de elementos.

// begin(): Retorna um ponteiro para o inicio do map

// end(): Retorna um ponteiro para o final do map

**For em map**

**for** (map<string,int>::iterator it=M.begin(); it!=M.end(); ++it){

cout << "(" << it->first << ", " << it->second << ") ";

}

**Fila de prioridades**

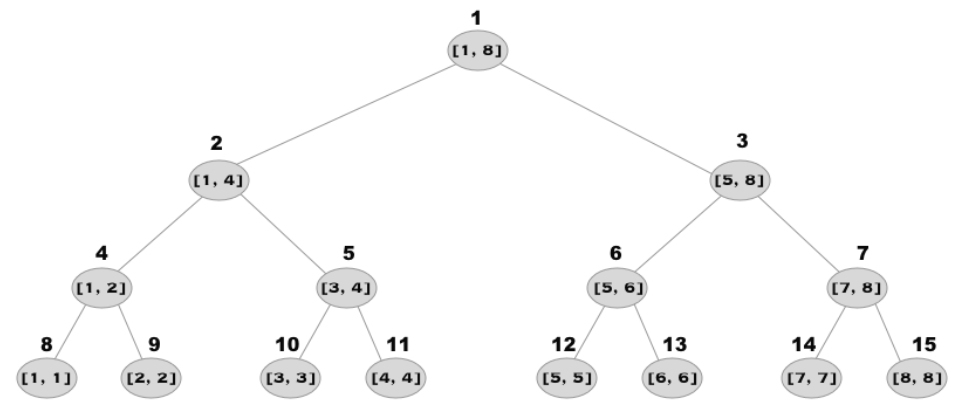
Para utilizar a priority\_queue do C++ é importante apenas saber que o maior elemento sempre estará na primeiro posição. Com exeção disso, todos os outros métodos são semelhantes ao uso de uma queue comum, porém para manter a estrutura organizada, a complexidade da operação de inserção é O(log n).

priority\_queue< pair<int, string> > pokemon;

pokemon.push(make\_pair(poder, nome));

pokemon.top();

pokemon.pop();

Árvore de Segmentos

// Encontra o valor mínimo em um interval em O(log n).

// acao[i] representa o preço da ação de índice i

// arvore[i] representa o valor contido no nó i da árvore.

// Ou seja, arvore[i] contém o índice da ação mais barata

// no intervalo representado pelo nó i

// (no) representa o nó que estamos na função recursiva

// o nó que estamos representa o segmento [i, j]

// A função coloca altera o valor da ação de índice (posicao)

// para (novo\_valor) e altera a árvore de acordo com o necessário

void atualiza(int no, int i, int j, int posicao, int novo\_valor){

// se tivermos i = j, temos i = posicao = j. Logo, estamos no nó mais baixo da ávore

**if**(i == j){

arvore[no] = i;

acao[posicao] = novo\_valor;

}

**else**{

int esquerda = 2\*no; // índice do filho da esquerda

int direita = 2\*no + 1; // índice do filho da direita

int meio = (i + j)/2;

**if**(posicao <= meio) atualiza(esquerda, i, meio, posicao, novo\_valor);

**else** atualiza(direita, meio+1, j, posicao, novo\_valor);

**if**( acao[ arvore[esquerda] ] < acao[ arvore[direita] ] ) arvore[no] = arvore[esquerda];

**else** arvore[no] = arvore[direita];

}

}

int consulta(int no, int i, int j, int A, int B){

**if**(A <= i && j <= B){

**return** arvore[no];

}

**if**(i > B || A > j){

**return** -1;

}

int esquerda = 2\*no;

int direita = 2\*no + 1;

int meio = (i + j)/2;

int resposta\_esquerda = consulta(esquerda, i, meio, A, B);

int resposta\_direita = consulta(direita, meio+1, j, A, B);

**if**(resposta\_esquerda == -1) **return** resposta\_direita;

**if**(resposta\_direita == -1) **return** resposta\_esquerda;

**if**(acao[resposta\_esquerda] < acao[resposta\_direita]) **return** resposta\_esquerda;

**else return** resposta\_direita;

}

Árvore de Indexação Binária (BIT)

// Dado um intervalo 1 a N. Permite adicionar valores aos elementos do intervalo e, efetuar o somatório de um intervalo intermediário entre 1 e N em O(log n).

int soma(int x){

int s = 0;

// vamos reduzindo x até acabarmos (quando chegamos a zero)

**while**(x > 0){

s += arvore[x]; // adicionamos o pedaço de árvore atual à soma

x -= (x & -x); // removemos o bit menos sign**if**icante

}

}

void atualiza(int x, int v){ // adicionar v frutas a caixa x

**while**(x <= N){ // nosso teto, que é quando vamos parar de rodar o algoritmo

arvore[x] += v; // adicionamos v frutas a arvore[x], como devemos

x += (x & -x); // atualizamos o valor de x adicionado ele ao seu LSB

}

}

Lazy Propagation

// Você tem caixas N de frutas, númeradas de 1 a N, e duas possíveis operações.

//Operação 1: adicionar v frutas a cada uma das caixas de índice entre a e b(inclusive).

//Operação 2: responder quantas frutas existem nas caixas de índice entre a e b(inclusive).

// com Lazy Propagation, uma adaptação que se faz na Árvore de Segmentos que permite fazer ambas as operações em O(log n)

// arvore[i] representa o valor contido no nó i da árvore.

// Ou seja, se o nó i representa o intervalo [X, Y],

// arvore[i] representa a soma das caixas de X a Y

// lazy[i] representa a soma de todas as operações

// atrasadas que devemos fazer ao nó i

// (no) representa o nó que estamos na função recursiva

// o nó que estamos representa o segmento [i, j]

// vamos somar (valor) a cada um dos índices no intervalo [a, b]

void atualiza(int no, int i, int j, int a, int b, int valor){

int esquerda = 2\*no; // índice do filho da esquerda

int direita = 2\*no + 1; // índice do filho da direita

int meio = (i + j)/2;

**if**(lazy[no]){

arvore[no] += lazy[no]\*(j - i + 1);

**if**(i != j){

lazy[direita] += lazy[no];

lazy[esquerda] += lazy[no];

}

lazy[no] = 0;

}

**if**(i > j || i > b || a > j) **return**;

**if**(a <= i && j <= b){

arvore[no] += valor\*(j-i+1);

**if**(i != j){

lazy[direita] += valor;

lazy[esquerda] += valor;

}

}

**else**{

// atualizamos o filho da esquerda

atualiza(esquerda, i, meio, a, b, valor);

// atualizamos o filho da direita

atualiza( direita, meio+1, j, a, b, valor);

// atualizamos o nó que estamos

arvore[no] = arvore[esquerda] + arvore[direita];

}

}

// queremos saber a soma de todos os valores de índice no intervalo [A, B]

int consulta(int no, int i, int j, int a, int b){

int esquerda = 2\*no; // índice do filho da esquerda

int direita = 2\*no + 1; // índice do filho da direita

int meio = (i + j)/2;

**if**(lazy[no]){

arvore[no] += lazy[no]\*(j - i + 1);

**if**(i != j){

lazy[direita] += lazy[no];

lazy[esquerda] += lazy[no];

}

lazy[no] = 0;

}

**if**(i > j || i > b || a > j) **return** 0;

**if**(a <= i && j <= b)

**return** arvore[no];

**else**{

int soma\_esquerda = consulta(esquerda, i, meio, a, b);

int soma\_direita = consulta( direita, meio+1, j, a, b);

**return** soma\_esquerda + soma\_direita;

}

}

Sort em structs

Sort em structs

-Criar struct

**typedef** struct {

int moradores;

int gastoss;

int media;

} Imovel;

-Definir comparator para a struct

bool cmp(Imovel **const** & x,Imovel **const** & y){

**if**(x.media < y.media) {

**return** true;

}

**else** {

**return** false;

}

}

// Efetuar o sort no main

Imovel imoveis[10];

sort(&imoveis[0],&imoveis[10], cmp);

#FIM\_ESTRUTURAS

MISC/ADHOC

Tabela ASCII

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Caracter | Dec | Oct | Hex | Caracter | Dec | Oct | Hex | Caracter | Dec | Oct | Hex | Caracter | Dec | Oct | Hex |
| (nul) | 0 | 0 | 0x00 | @ | 64 | 100 | 0x40 | Ç | 128 | 200 | 0x80 | + | 192 | 300 | 0xc0 |
| (soh) | 1 | 1 | 0x01 | A | 65 | 101 | 0x41 | ü | 129 | 201 | 0x81 | - | 193 | 301 | 0xc1 |
| (stx) | 2 | 2 | 0x02 | B | 66 | 102 | 0x42 | é | 130 | 202 | 0x82 | - | 194 | 302 | 0xc2 |
| (etx) | 3 | 3 | 0x03 | C | 67 | 103 | 0x43 | â | 131 | 203 | 0x83 | + | 195 | 303 | 0xc3 |
| (eot) | 4 | 4 | 0x04 | D | 68 | 104 | 0x44 | ä | 132 | 204 | 0x84 | - | 196 | 304 | 0xc4 |
| (enq) | 5 | 5 | 0x05 | E | 69 | 105 | 0x45 | à | 133 | 205 | 0x85 | + | 197 | 305 | 0xc5 |
| (ack) | 6 | 6 | 0x06 | F | 70 | 106 | 0x46 | å | 134 | 206 | 0x86 | ã | 198 | 306 | 0xc6 |
| (bel) | 7 | 7 | 0x07 | G | 71 | 107 | 0x47 | ç | 135 | 207 | 0x87 | Ã | 199 | 307 | 0xc7 |
| (bs) | 8 | 10 | 0x08 | H | 72 | 110 | 0x48 | ê | 136 | 210 | 0x88 | + | 200 | 310 | 0xc8 |
| (ht) | 9 | 11 | 0x09 | I | 73 | 111 | 0x49 | ë | 137 | 211 | 0x89 | + | 201 | 311 | 0xc9 |
| (nl) | 10 | 12 | 0x0a | J | 74 | 112 | 0x4a | è | 138 | 212 | 0x8a | - | 202 | 312 | 0xca |
| (vt) | 11 | 13 | 0x0b | K | 75 | 113 | 0x4b | ï | 139 | 213 | 0x8b | - | 203 | 313 | 0xcb |
| (np) | 12 | 14 | 0x0c | L | 76 | 114 | 0x4c | î | 140 | 214 | 0x8c | ¦ | 204 | 314 | 0xcc |
| (cr) | 13 | 15 | 0x0d | M | 77 | 115 | 0x4d | ì | 141 | 215 | 0x8d | - | 205 | 315 | 0xcd |
| (so) | 14 | 16 | 0x0e | N | 78 | 116 | 0x4e | Ä | 142 | 216 | 0x8e | + | 206 | 316 | 0xce |
| (si) | 15 | 17 | 0x0f | O | 79 | 117 | 0x4f | Å | 143 | 217 | 0x8f | ¤ | 207 | 317 | 0xcf |
| (dle) | 16 | 20 | 0x10 | P | 80 | 120 | 0x50 | É | 144 | 220 | 0x90 | ð | 208 | 320 | 0xd0 |
| (dc1) | 17 | 21 | 0x11 | Q | 81 | 121 | 0x51 | æ | 145 | 221 | 0x91 | Ð | 209 | 321 | 0xd1 |
| (dc2) | 18 | 22 | 0x12 | R | 82 | 122 | 0x52 | Æ | 146 | 222 | 0x92 | Ê | 210 | 322 | 0xd2 |
| (dc3) | 19 | 23 | 0x13 | S | 83 | 123 | 0x53 | ô | 147 | 223 | 0x93 | Ë | 211 | 323 | 0xd3 |
| (dc4) | 20 | 24 | 0x14 | T | 84 | 124 | 0x54 | ö | 148 | 224 | 0x94 | È | 212 | 324 | 0xd4 |
| (nak) | 21 | 25 | 0x15 | U | 85 | 125 | 0x55 | ò | 149 | 225 | 0x95 | i | 213 | 325 | 0xd5 |
| (syn) | 22 | 26 | 0x16 | V | 86 | 126 | 0x56 | û | 150 | 226 | 0x96 | Í | 214 | 326 | 0xd6 |
| (etb) | 23 | 27 | 0x17 | W | 87 | 127 | 0x57 | ù | 151 | 227 | 0x97 | Î | 215 | 327 | 0xd7 |
| (can) | 24 | 30 | 0x18 | X | 88 | 130 | 0x58 | ÿ | 152 | 230 | 0x98 | Ï | 216 | 330 | 0xd8 |
| (em) | 25 | 31 | 0x19 | Y | 89 | 131 | 0x59 | Ö | 153 | 231 | 0x99 | + | 217 | 331 | 0xd9 |
| (sub) | 26 | 32 | 0x1a | Z | 90 | 132 | 0x5a | Ü | 154 | 232 | 0x9a | + | 218 | 332 | 0xda |
| (esc) | 27 | 33 | 0x1b | [ | 91 | 133 | 0x5b | ø | 155 | 233 | 0x9b | \_ | 219 | 333 | 0xdb |
| (fs) | 28 | 34 | 0x1c | \ | 92 | 134 | 0x5c | £ | 156 | 234 | 0x9c | \_ | 220 | 334 | 0xdc |
| (gs) | 29 | 35 | 0x1d | ] | 93 | 135 | 0x5d | Ø | 157 | 235 | 0x9d | ¦ | 221 | 335 | 0xdd |
| (rs) | 30 | 36 | 0x1e | ^ | 94 | 136 | 0x5e | × | 1158 | 236 | 0x9e | Ì | 222 | 336 | 0xde |
| (us) | 31 | 37 | 0x1f | \_ | 95 | 137 | 0x5f | ƒ | 159 | 237 | 0x9f | \_ | 223 | 337 | 0xdf |
| (space) | 32 | 40 | 0x20 | ` | 96 | 140 | 0x60 | á | 160 | 240 | 0xa0 | Ó | 224 | 340 | 0xe0 |
| ! | 33 | 41 | 0x21 | a | 97 | 141 | 0x61 | í | 161 | 241 | 0xa1 | ß | 225 | 341 | 0xe1 |
| " | 34 | 42 | 0x22 | b | 98 | 142 | 0x62 | ó | 162 | 242 | 0xa2 | Ô | 226 | 342 | 0xe2 |
| # | 35 | 43 | 0x23 | c | 99 | 143 | 0x63 | ú | 163 | 243 | 0xa3 | Ò | 227 | 343 | 0xe3 |
| $ | 36 | 44 | 0x24 | d | 100 | 144 | 0x64 | ñ | 164 | 244 | 0xa4 | Õ | 228 | 344 | 0xe4 |
| % | 37 | 45 | 0x25 | e | 101 | 145 | 0x65 | Ñ | 165 | 245 | 0xa5 | Õ | 229 | 345 | 0xe5 |
| & | 38 | 46 | 0x26 | f | 102 | 146 | 0x66 | ª | 166 | 246 | 0xa6 | µ | 230 | 346 | 0xe6 |
| ' | 39 | 47 | 0x27 | g | 103 | 147 | 0x67 | º | 167 | 247 | 0xa7 | Þ | 231 | 347 | 0xe7 |
| ( | 40 | 50 | 0x28 | h | 104 | 150 | 0x68 | ¿ | 168 | 250 | 0xa8 | Þ | 232 | 350 | 0xe8 |
| ) | 41 | 51 | 0x29 | i | 105 | 151 | 0x69 | ® | 169 | 251 | 0xa9 | Ú | 233 | 351 | 0xe9 |
| \* | 42 | 52 | 0x2a | j | 106 | 152 | 0x6a | ¬ | 170 | 252 | 0xaa | Û | 234 | 352 | 0xea |
| + | 43 | 53 | 0x2b | k | 107 | 153 | 0x6b | ½ | 171 | 253 | 0xab | Ù | 235 | 353 | 0xeb |
| , | 44 | 54 | 0x2c | l | 108 | 154 | 0x6c | ¼ | 172 | 254 | 0xac | ý | 236 | 354 | 0xec |
| - | 45 | 55 | 0x2d | m | 109 | 155 | 0x6d | ¡ | 173 | 255 | 0xad | Ý | 237 | 355 | 0xed |
| . | 46 | 56 | 0x2e | n | 110 | 156 | 0x6e | « | 174 | 256 | 0xae | ¯ | 238 | 356 | 0xee |
| / | 47 | 57 | 0x2f | o | 111 | 157 | 0x6f | » | 175 | 257 | 0xaf | ´ | 239 | 357 | 0xef |
| 0 | 48 | 60 | 0x30 | p | 112 | 160 | 0x70 | \_ | 176 | 260 | 0xb0 |  | 240 | 360 | 0xf0 |
| 1 | 49 | 61 | 0x31 | q | 113 | 161 | 0x71 | \_ | 177 | 261 | 0xb1 | ± | 241 | 361 | 0xf1 |
| 2 | 50 | 62 | 0x32 | r | 114 | 162 | 0x72 | \_ | 178 | 262 | 0xb2 | \_ | 242 | 362 | 0xf2 |
| 3 | 51 | 63 | 0x33 | s | 115 | 163 | 0x73 | ¦ | 179 | 263 | 0xb3 | ¾ | 243 | 363 | 0xf3 |
| 4 | 52 | 64 | 0x34 | t | 116 | 164 | 0x74 | ¦ | 180 | 264 | 0xb4 | ¶ | 244 | 364 | 0xf4 |
| 5 | 53 | 65 | 0x35 | u | 117 | 165 | 0x75 | Á | 181 | 265 | 0xb5 | § | 245 | 365 | 0xf5 |
| 6 | 54 | 66 | 0x36 | v | 118 | 166 | 0x76 | Â | 192 | 266 | 0xb6 | ÷ | 246 | 366 | 0xf6 |
| 7 | 55 | 67 | 0x37 | w | 119 | 167 | 0x77 | À | 183 | 267 | 0xb7 | ¸ | 247 | 367 | 0xf7 |
| 8 | 56 | 70 | 0x38 | x | 120 | 170 | 0x78 | © | 184 | 270 | 0xb8 | ° | 248 | 370 | 0xf8 |
| 9 | 57 | 71 | 0x39 | y | 121 | 171 | 0x79 | ¦ | 185 | 271 | 0xb9 | ¨ | 249 | 371 | 0xf9 |
| : | 58 | 72 | 0x3a | z | 122 | 172 | 0x7a | ¦ | 186 | 272 | 0xba | · | 250 | 372 | 0xfa |
| ; | 59 | 73 | 0x3b | { | 123 | 173 | 0x7b | + | 187 | 273 | 0xbb | ¹ | 251 | 373 | 0xfb |
| < | 60 | 74 | 0x3c | | | 124 | 174 | 0x7c | + | 188 | 274 | 0xbc | ³ | 252 | 374 | 0xfc |
| = | 61 | 75 | 0x3d | } | 125 | 175 | 0x7d | ¢ | 189 | 275 | 0xbd | ² | 253 | 375 | 0xfd |
| > | 62 | 76 | 0x3e | ~ | 126 | 176 | 0x7e | ¥ | 190 | 276 | 0xbe | \_ | 254 | 376 | 0xfe |
| ? | 63 | 77 | 0x3f | (del) | 127 | 177 | 0x7f | + | 191 | 277 | 0xbf |  | 255 | 377 | 0xff |

|  |  |  |
| --- | --- | --- |
| Nome ASCII | Descrição | Representação em C |
| nul | null byte | \0 |
| bel | bell character | \a |
| bs | backspace | \b |
| ht | horizontal tab | \t |
| np | **for**mfeed | \f |
| nl | newline | \n |
| cr | carriage **return** | \r |

**Conversão para números romanos**

// fazer teste com 444 IX IV CM CD XC XL

// Lembre-se que I representa 1, V é 5, X é 10, L é 50, C é 100, D é 500 e M 1000

**Antes e depois de Cristo**

// Não existe ano 0, existe 1 A.C. e 1 D.C.

**Ao trabalhar com horas, procurar sempre usar minutos/segundos**

**Problemas com múltiplos casos de teste**

// Focar em resolver 1 caso e depois implementar a solução para n casos

**Problemas com uma linha em branço depois da resposta final**

// Na verdade precisam de 2 \n para serem aceitos (1 da saída, e o outro para judge)

**Ano bissexto**

// Considerar 366 dias

**Ano normal**

// Considerar 365 dias

**Dias de cada mês**

// Janeiro(1) 31

// Fevereiro(2) 28(29 bissexto)

// Março(3) 31

// Abril(4) 30

// Maio(5) 31

// Junho(6) 30

// Julho(7) 31

// Agosto(8) 31

// Setembro(9) 30

// Outubro(10) 31

// Novembro(11) 30

// Dezembro(12) 31

// 30: 4 | 31: 7 | \*28: 1 | \*29: 1

**Formas de se escrever tipos de dados**

// long int = long

// long long int = long long

// unsigned int = unsigned

// unsigned long long int = unsigned long long

**Alfabeto tem 26 Letras, contando K, W e Y**

**Inicializar vetor com valor predefinido**

// **for** 1d array, use STL fill\_n or fill to initialize array

fill(a, a+size\_of\_a, value)

fill\_n(a, size\_of\_a, value)

// **for** 2d array, **if** want to fill in 0 or -1

memset(a, 0, sizeof(a));

// otherwise, use a loop of fill or fill\_n through every a[i]

fill(a[i], a[i]+size\_of\_ai, value) // from 0 to number of row.

**Operações para modificar sequências**

void copy(first, last, result);

void swap(a,b);

void swap(first1, last1, first2); // swap range

void replace(first, last, old\_value, new\_value); // replace in range

void replace\_**if**(first, last, pred, new\_value); // replace in conditions

// pred can be represented in function

// e.x. bool IsOdd (int i) { **return** ((i%2)==1); }

void reverse(first, last); // reverse a range of elements

void reverse\_copy(first, last, result); // copy a reverse of range of elements

void random\_shuffle(first, last); // using built-in random generator to shuffle array

**Permutações**

bool next\_permutation(iterator first, iterator last);

bool next\_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);

bool prev\_permutation(iterator first, iterator last);

bool prev\_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);

**Gerar números aleatórios**

srand(time(NULL));

// generate random numbers between [a,b)

rand() % (b - a) + a;

// generate random numbers between [0,b)

rand() % b;

// generate random permutations

random\_permutation(anArray, anArray + 10);

random\_permutation(aVector, aVector + 10);

Pesquisa Binária

// Necessário vetor estar ordenado

int binarySearch(int arr[], int l, int r, int x)

{

if (r >= l)

{

int mid = l + (r - l)/2;

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

return binarySearch(arr, l, mid-1, x);

return binarySearch(arr, mid+1, r, x);

}

return -1;

}

#FIM\_MISC\_ADHOC

GRAFOS

Struct grafo

**typedef** struct{

// destino, peso

vector<pair <int, int> > grafo[MAXV];

// visitado, tempos estrutura auxiliar DFS e BFS

bool visitado[MAXV];

int tempo\_descoberta[MAXV];

int n\_vertices = 0;

}Grafo;

Flood Fill (BFS E DFS)

void DFS(Grafo &g, int origem){

**forn**(i, g.n\_vertices){

g.visitado[i] = false;

g.tempo\_descoberta[i] = 0;

}

int tempo\_atual = 0;

stack<int> s;

s.push(origem);

g.tempo\_descoberta[origem] = tempo\_atual;

**while**(!s.empty()){

// item do topo da pilha

int u = s.top();

g.tempo\_descoberta[u] = ++tempo\_atual;

s.pop();

**if**(!g.visitado[u]){

g.visitado[u] = true;

**forn**(i, g.grafo[u].size()){

// vértice adjacente de u

int w = g.grafo[u].at(i).first;

**if**(!g.visitado[w]){

s.push(w);

}

}

}

}

}

void **BFS**(Grafo &g, int origem){

**forn**(i, g.n\_vertices){

g.visitado[i] = false;

g.tempo\_descoberta[i] = 0;

}

int tempo\_atual = 0;

g.visitado[origem] = true;

g.tempo\_descoberta[origem] = tempo\_atual;

queue<int> q;

q.push(origem);

**while**(!q.empty()){

int u = q.front();

g.tempo\_descoberta[u] = ++tempo\_atual;

q.pop();

**forn**(i, g.grafo[u].size()){

int w = g.grafo[u].at(i).first;

**if**(!g.visitado[w]){

g.visitado[w] = true;

q.push(w);

}

}

}

}

Caminho Mínimo entre 2 pontos(Dijkstra)

**typedef** pair<int, int> pii;

**#define** MAXN 10100

**#define** INFINITO 999999999

int n, m; // número de vértices e arestas

int cidade\_noic; // cidade onde está o Noic

int cidade\_succa; // cidade onde está o Succa

int distancia[MAXN]; // o array de distâncias à fonte

int processado[MAXN]; // o array que guarda se um vértice foi processado

vector<pii> vizinhos[MAXN]; // nossas listas de adjacência. O primeiro elemento do par representa a distância e o segundo representa o vértice

void Dijkstra(int S){

**for**(int i = 1;i <= n;i++) distancia[i] = INFINITO;

distancia[S] = 0;

priority\_queue< pii, vector<pii>, greater<pii> > fila;

fila.push( pii(distancia[S], S) );

**while**(true){

int davez = -1;

int menor = INFINITO;

**while**(!fila.empty()){

int atual = fila.top().second;

fila.pop();

**if**(!processado[atual]){

davez = atual;

**break**;

}

}

**if**(davez == -1) **break**;

processado[davez] = true;

**for**(int i = 0;i < (int)vizinhos[davez].size();i++){

int dist = vizinhos[davez][i].first;

int atual = vizinhos[davez][i].second;

// A nova possível distância é distancia[davez] + dist.

// Comparamos isso com distancia[atual]

**if**( distancia[atual] > distancia[davez] + dist ){

distancia[atual] = distancia[davez] + dist;

fila.push( pii(distancia[atual], atual) );

}

}

}

}

int main(){

cin >> n >> m;

cin >> cidade\_succa >> cidade\_noic;

**for**(int i = 1;i <= m;i++){

int x, y, tempo;

cin >> x >> y >> tempo;

vizinhos[x].push\_back( pii(tempo, y) );

vizinhos[y].push\_back( pii(tempo, x) );

}

Dijkstra(cidade\_succa);

cout << distancia[cidade\_noic] << endl;

**return** 0;

}

Algoritmo de Kruskal – Árvore geradora mínima(Arestas de peso mínimo que conectam todo o grafo)

struct t\_aresta{

int dis;

int x, y;

};

bool comp(t\_aresta a, t\_aresta b){ **return** a.dis < b.dis; }

//--------------------

**#define** MAXN 50500

**#define** MAXM 200200

int n, m; // número de vértices e arestas

t\_aresta aresta[MAXM];

// para o union find

int pai[MAXN];

int peso[MAXN];

// a árvore

t\_aresta mst[MAXM];

//--------------------

// funções do union find

int find(int x){

**if**(pai[x] == x) **return** x;

**return** pai[x] = find(pai[x]);

}

void join(int a, int b){

a = find(a);

b = find(b);

**if**(peso[a] < peso[b]) pai[a] = b;

**else** **if**(peso[b] < peso[a]) pai[b] = a;

**else**{

pai[a] = b;

peso[b]++;

}

}

int main(){

// ler a entrada

cin >> n >> m;

**for**(int i = 1;i <= m;i++)

cin >> aresta[i].x >> aresta[i].y >> aresta[i].dis);

// inicializar os pais para o union-find

**for**(int i = 1;i <= n;i++) pai[i] = i;

// ordenar as arestas

sort(aresta+1, aresta+m+1, comp);

int size = 0;

**for**(int i = 1;i <= m;i++){

**if**( find(aresta[i].x) != find(aresta[i].y) ){ // se estiverem em componentes distintas

join(aresta[i].x, aresta[i].y);

mst[++size] = aresta[i];

}

}

// imprimir a MST

**for**(int i = 1;i < n;i++) cout << mst[i].x << " " << mst[i].y << " " << mst[i].dis) << "\n";

**return** 0;

}

Algoritmo Prim – Árvore Geradora Mínima

**typedef** pair<int, int> pii;

**#define** MAXN 10100

**#define** INFINITO 999999999

int n, m; // número de vértices e arestas

int distancia[MAXN]; // o array de distâncias à fonte

int processado[MAXN]; // o array que guarda se um vértice foi processado

vector<pii> vizinhos[MAXN]; // nossas listas de adjacência. O primeiro elemento do par representa a distância e o segundo representa o vértice

int Prim(){

**for**(int i = 2;i <= n;i++) distancia[i] = INFINITO;

distancia[1] = 0;

priority\_queue< pii, vector<pii>, greater<pii> > fila;

fila.push( pii(distancia[1], 1) );

**while**(true){

int davez = -1;

**while**(!fila.empty()){

int atual = fila.top().second;

fila.pop();

**if**(!processado[atual]){

davez = atual;

**break**;

}

}

**if**(davez == -1) **break**;

processado[davez] = true;

**for**(int i = 0;i < (int)vizinhos[davez].size();i++){

int dist = vizinhos[davez][i].first;

int atual = vizinhos[davez][i].second;

**if**( distancia[atual] > dist && !processado[atual]){

distancia[atual] = dist;

fila.push( pii(distancia[atual], atual) );

}

}

}

int custo\_arvore = 0;

**for**(int i = 1;i <= n;i++) custo\_arvore += distancia[i];

**return** custo\_arvore;

}

int main(){

cin >> n >> m;

**for**(int i = 1;i <= m;i++){

int x, y, tempo;

cin >> x >> y >> tempo;

vizinhos[x].push\_back( pii(tempo, y) );

vizinhos[y].push\_back( pii(tempo, x) );

}

cout << Prim() << endl;

**return** 0;

}

Caminho Euleniano (um trajeto que passa por todas as arestas do grafo sem repetição)

vector<int> caminho; // guardará nosso Caminho Euleriano (invertido)

vector<int> vizinhos[MAXN]; // nossa lista de adjacência

map< pair<int, int>, bool > deletada; // mapa que checa se a aresta já foi deletada

void acha\_caminho(int v){

**for**(int i = 0;i < (int)vizinhos[v].size();i++){

int viz = vizinhos[v][i];

**if**( deletada[make\_pair(v, viz)] == true ) **continue**;

deletada[make\_pair(v, viz)] = true;

deletada[make\_pair(viz, v)] = true;

acha\_caminho(viz);

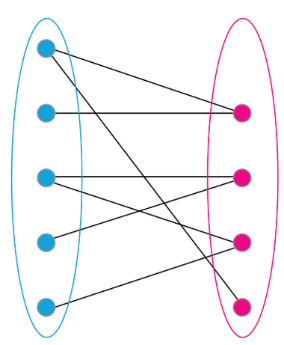
}

caminho.push\_back(v);

}

Grafos Bipartidos

// Um grafo é dito bipartido quando seus vértices podem ser divididos em dois conjuntos disjuntos tais que cada aresta ligue apenas vértices de grupos d**if**erentes.



int n; // número de vértices

vector<int> vizinhos[MAXN]; // a lista de adjacência de cada vértice

int cor[MAXN];

// a cor de cada vértice. Inicialmente, cor[i] = -1 para todos os vértices.

// definimos cor[i] = 0 como sendo azul e cor[i] = 1 como sendo rosa.

void colore(int x){

cor[x] = 0;

vector<int> fila;

fila.push\_back(x);

int pos = 0;

**while**(pos < (int)fila.size()){ // BFS

int atual = fila[pos];

pos++;

**for**(int i = 0;i < (int)vizinhos[atual].size();i++){

int v = vizinhos[atual][i];

**if**(cor[v] == -1){

cor[v] = 1 - cor[atual];

fila.push\_back(v); // adicionamos v a fila da BFS

}

}

}

}

bool checa\_bipartido(){

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

**if**(cor[i] == -1){

colore(i);

}

}

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

**for**(int j = 0;j < (int)vizinhos[i].size();j++){

int v = vizinhos[i][j];

**if**(cor[i] == cor[v]) **return** false;

}

}

**return** true;

}

#FIM\_GRAFOS

TÉCNICAS DE PROGRAMAÇÃO

Contagem de inversões

// Um dos problemas mais clássicos de programação é a contagem de inversões em uma sequência. De maneira simples, seja S = a1, a2,...,an. Uma inversão em S é um par (i,j) com i < j tal que ai > aj. Sabendo disso, faça um programa que calcula o número de inversões em uma sequência S. Complexidade O(N log N)

int merge\_sort(vector<int> &v){

int inv=0;

**if**(v.size()==1) **return** 0;

vector<int> u1, u2;

**for**(int i=0;i<v.size()/2;i++){

u1.push\_back(v[i]);

}

**for**(int i=v.size()/2;i<v.size();i++){

u2.push\_back(v[i]);

}

inv+=merge\_sort(u1);

inv+=merge\_sort(u2);

u1.push\_back(INF);

u2.push\_back(INF);

int ini1=0, ini2=0;

**for**(int i=0;i<v.size();i++){

**if**(u1[ini1]<=u2[ini2]){

v[i]=u1[ini1];

ini1++;

}

**else**{

v[i]=u2[ini2];

ini2++;

inv+=u1.size()-ini1-1;

}

}

**return** inv;

}

Problema da mochila

// Lembre-se de, antes de chamar a função na main, fazer com que todos os valores de tab se tornem -1, com o comando memset(tab,-1,sizeof tab), indicando que nenhum estado foi calculado ainda.

// defino os maiores valores de n e s como 1010

**#define** MAXN 1010

**#define** MAXS 1010

int n, valor[MAXN], peso[MAXN], tab[MAXN][MAXS]

int knapsack(int obj, int aguenta){

**if**(tab[obj][aguenta]>=0) **return** tab[obj][aguenta];

**if**(obj==n or !aguenta) **return** tab[obj][aguenta]=0;

int nao\_coloca=knapsack(obj+1, aguenta);

**if**(peso[obj]<=aguenta){

int coloca=valor[obj]+knapsack(obj+1, aguenta-peso[obj]);

**return** tab[obj][aguenta]=max(coloca, nao\_coloca);

}

**return** tab[obj][aguenta]=nao\_coloca;

}

Maior Subsequência Comum (LCS)

// Dadas duas sequências s1 e s2, uma de tamanho n e outra de tamanho m, qual a maior subsequência comum às duas? Lembre-se que uma subsequência de s1, por exemplo, é simplesmente um subconjunto dos elementos de s1 na mesma ordem em que apareciam antes. Isto sign**if**ica que {1, 3, 5} é uma subsequência de {1, 2, 3, 4, 5}, mesmo 1 não estando do lado do 3 na sequência original.

**#define** MAXN 1010

int s1[MAXN], s2[MAXN], tab[MAXN][MAXN];

int lcs(int a, int b){

**if**(tab[a][b]>=0) **return** tab[a][b];

**if**(a==0 or b==0) **return** tab[a][b]=0;

**if**(s1[a]==s2[b]) **return** 1+lcs(a-1, b-1);

**return** tab[a][b]=max(lcs(a-1, b), lcs(a, b-1));

}

Maior subsequência Crescente - LIS(Tamanho)

// O problema é: dada uma sequência s qualquer, descobrir o tamanho da maior subsequência crescente de s. Por exemplo: s = {3,4,3,5,2,7} a maior subsequência crescente de s é s’ = {3,4,5,7}

**#define** PB push\_back // por simplicidade

int lis(vector<int> &v){

vector<int> pilha;

**for**(int i=0; i<v.size(); i++){

vector<int>::iterator it = lower\_bound(pilha.begin(), pilha.end(), v[i]);

**if**(it==pilha.end()) pilha.PB(v[i]);

**else** \*it = v[i];

}

**return** pilha.size();

}

Maior subsequência Crescente - LIS(Vetor)

**#define** PB push\_back

**#define** MAXN 100100

vector<int> lis(vector<int> &v){

vector<int> pilha, resp;

int pos[MAXN], pai[MAXN];

**for**(int i=0; i<v.size(); i++){

vector<int>::iterator it = lower\_bound(pilha.begin(), pilha.end(), v[i]);

int p = it-pilha.begin();

**if**(it==pilha.end()) pilha.PB(v[i]);

**else** \*it = x;

**if**(p==0) pai[i]=-1;

**else** pai[i]=pos[p-1];

}

int p = pos[pilha.size()-1];

**while**(p>=0){

resp.PB(v[p]);

p=pai[p];

}

**return** resp;

}

Troco

// A ideia é muito simples: sabendo todos os possíveis valores de moedas de um país, é possível **for**mar um determinado valor usando as moedas. Suponha, por exemplo, que um país tem moedas de 2, 5 e 7 centavos. É possível dar um troco de 19 centavos? Sim, basta usarmos 2 moedas de 5, uma de 7 e uma de 2 centavos.

// O código a seguir é de uma função que recebe como parâmetros o valor inteiro x, que desejamos **for**mar, e um vetor de inteiros c, onde c[i] representa o valor da i-ésima moeda. Lembre-se de inicializar todos os valores da DP como -1 (não calculado) e, para cada estado, ela retornará 1, se **for** true, ou 0, caso seja false.

// Dp(x) retornará true se **for** possível conseguir exatamente o valor x com as moedas que tenho disponível, e false caso contrário.

int dp[MAX];

int solve(int x, vector<int> &c){

**if**(x==0) **return** 1;

**if**(x<0) **return** 0;

**if**(dp[x]>=0) **return** dp[x];

**for**(int i=0;i<c.size();i++)

**if**(solve(x-c[i])) **return** dp[x-c[i]]=1;

**return** dp[x]=0;

}

Soma máxima em um intervalo

// Dado uma sequência qualquer S = (s1, s2, s3,..., sn) qual a maior soma que podemos obter escolhendo um subconjunto de termos adjacentes de S? Se a sequência **for**, por exemplo, (1, -3, 5, -2, 1, -1), a soma máxima é 4, com os termos (5, -2, 1)

int max\_sum(vector<int> s){

int resp=0, maior=0;

**for**(int i=0;i<s.size();i++){

maior=max(0,maior+s[i]);

resp=max(resp,maior);

}

**return** resp;

}

Backtracking

// Printa todas as permutações de uma dada string

void swap(char \*x, char \*y)

{

    char temp;

    temp = \*x;

    \*x = \*y;

    \*y = temp;

}

/\* Function to print permutations of string

   This function takes three parameters:

   1. String

   2. Starting index of the string

   3. Ending index of the string. \*/

void permute(char \*a, int l, int r)

{

   int i;

**if** (l == r)

     printf("%s\n", a);

**else**

   {

**for** (i = l; i <= r; i++)

       {

          swap((a+l), (a+i));

          permute(a, l+1, r);

          swap((a+l), (a+i)); //backtrack

       }

   }

}

void search() {

**if** (permutation.size() == n) {

// process permutation

} **else** {

**for** (int i = 0; i < n; i++) {

**if** (chosen[i]) **continue**;

chosen[i] = true;

permutation.push\_back(i);

search();

chosen[i] = false;

permutation.pop\_back();

}

}

}

vector<int> permutation;

**for** (int i = 0; i < n; i++) {

permutation.push\_back(i);

}

do {

// process permutation

} **while** (next\_permutation(permutation.begin(),permutation.end()));

#FIM\_TECNICAS\_PROGRAMAÇÃO