Matemáticos

Módulo
Exponenciação Modular Rápida
Máximo Divisor Comum
Máximo Divisor Comum Estendido
Teste de Primalidade
Crivo de Erastótenes
Resolvedor de Equação Modula**r**
Troca (Swap) sem memória auxiliar

Grafos
Matriz de Adjacência
Lista de Adjacência

Pilha
APLICAÇÃO BUSCA EM PROFUNDIDADE EM C
IMPLEMENTAÇÃO BUSCA EM PROFUNDIDADE C++

Fila
IMPLEMENTAÇÂO USANDO VETOR
IMPLEMENTAÇÃO em C++
IMPLEMENTAÇÃO EM JAVA

Fila de Prioridade
Down
Construção
Extrair Mínimo
Atualiza valor
Algoritmo de Prim
Algoritmo do Caminho Mínimo
CAMINHO MÍNIMO COM PRIORITY QUEUE C++

Gerando Permutação
Permutação em C
Permutação em C++

Gerando Subconjuntos

Problema das Oitos Rainhas

Programação Dinâmica

Módulo

Entrada:
- Inteiros `a` (podendo ser negativo!) e `n` > 0.

Saída:
- Valor de `a` módulo `n`.

Complexidade: `O( 1 )`
Global:
```
int mod(int a, int n) {
return (a%n + n)%n;
}
```

Exponenciação Modular Rápida

Entrada:
- Inteiros `a`, `b` e `n`.

Saída:
- Valor de `a^b mod n`.

Complexidade: `O( log(b) )`
Global:
```
int expmod(int a, int b, int n) {
if(b == 0)
return 1;
else {
long long res = expmod(a, b/2, n);
res = (res*res) % n;
if(b%2 == 1)
res = (res*a) % n;
return (int) res;
}
}
```

Máximo Divisor Comum

Utiliza o algoritmo de Euclides.
Entrada:
- Inteiros `a` e `b`.

Saída:
- Maior inteiro que divide `a` e `b`.

Complexidade: `O( log(a) + log(b) )`
Global:
```
int mdc(int a, int b) {
if(a<0) a = -a;
if(b<0) b = -b;
```

```
if(b == 0)
return a;
else
return mdc(b, a%b);
}
```

Máximo Divisor Comum Estendido

Utiliza o algoritmo de Euclides estendido.
Entrada:

- Inteiros positivos `a` e `b`.

Saída:

- Maior inteiro que divide `a` e `b` como retorno.
- Variáveis inteiras `x` e `y` tais que `a.x + b.y = mdc(a,b)`.

Complexidade: `O( log(a) + log(b) )`
Global:

```
int mdc(int a, int b, int *x, int *y) {
int xx, yy, d;
if(b==0) {
*x=1; *y=0;
return a;
}

d = mdc(b, a%b, &xx, &yy);
*x = yy;
*y = xx - a/b*yy;
return d;
}
```

Teste de Primalidade

Entrada:

- Inteiro `n`.

Saída:

- Valor booleano que indica se `n` é primo ou não.

Complexidade: `O( sqrt(n) )`
Global:

```
int ehPrimo(int n) {
if(n==2) return 1;
if(n<=1 || n%2 == 0) return 0;

for(int i=3; i*i<=n; i+=2)
if(n%i == 0)
return 0;

return 1;
}

Implementação alternativa em C++:
```

bool is_prime(int n) {
if (n < 0) return is_prime(-n);

```
if (n < 5 || n % 2 == 0 || n % 3 == 0) return (n == 2 || n == 3);
int maxP = sqrt(n) + 2;
for (int p = 5; p < maxP; p += 6)
if (n % p == 0 || n % (p+2) == 0) return false;
return true;
}
```

Crivo de Eratóstenes

Entrada:

- Inteiro `n` <= `MAXN`.

Saída:

- Vetor `ehprimo[]` que indica se os números menores ou iguais a `n` são primos ou não.

Complexidade: `O( n.log(n) )`

Global:

```
int ehprimo[MAXN+1];

void achaPrimos(int n) {
ehprimo[0] = ehprimo[1] = 0;
ehprimo[2] = 1;

for(int i=3; i<=n; i++)
ehprimo[i] = i%2;

for(int i=3; i*i<=n; i+=2)
if(ehprimo[i])
for(int j=i*i; j<=n; j+=i)
ehprimo[j] = 0;
}
```

Resolvedor de Equação Modular Linear

Entrada:

- Inteiro `a` e `b` quaiquer e `n` positivo.

Saída:

- Menor valor positivo `x` tal que `a.x == b (mod n)` ou `-1` se não houver solução.

Complexidade: `O( n )`

Dependências:

- Módulo: `mod(int a, int n)`.
- MDC Estendido: `mdc(int a,int b,int *x, int*y)`.

Global:

```
int solve(int a, int b, int n) {
int d, x, y;

a = mod(a, n);
b = mod(b, n);
d = mdc(a, n, &x, &y);
if(b%d==0)
return mod( ((long long)x%(n/d)) * ((b/d)%(n/d)) , n/d );
```

```
else
return -1;
}
```

Troca (Swap) sem memória auxiliar

Ex:

```
a=4;
b=3;

//a = 100
//b = 011

a = a^b; // (100)^(011) = 111
b = a^b; // (111)^(011) = 100
a = a^b; // (111)^(100) = 011

//a=3 e b=4.
```

Matriz de Adjacência
Inicialização

```
for(i=1;i<=n;i++){
             for(j=1;j<=n;j++){
               g[i][j] = 0;
             }
}
```

Arestas

```
for(i=1;i<=m;i++){
             scanf("%d %d",&a,&b);
             g[a][b] = g[b][a] = 1;
}
```

Lista de Adjacência
Inicialização

```
for(i=1;i<=n;i++){
   d[i]=0;
}
```

Arestas

```
for(i=1;i<=m;i++){
       scanf("%d %d",&a,&b);
       g[a][d[a]++]=b;
       g[b][d[b]++]=a;
}
```

APLICAÇÃO BUSCA EM PROFUNDIDADE EM C

```
#include <stdio.h>
#define MAX 101
int pilha[MAX], d[MAX], g[MAX][MAX], marc[MAX],topo;
int inicializa(){
	topo = 0;
}
void push(int x){
	pilha[topo++] = x;
}
int pop(){
	return pilha[--topo];
}
int vazia(){
	return topo==0;
}
int main(){
	int n,m,u,i,v,a,b,cont,teste=1;
	while(1){
		scanf("%d %d",&n,&m);
		if(n==0 && m==0) break;
		for(i=1;i<=n;i++){
		 d[i] = 0;
		 marc[i]=0;
		}
		for(i=1;i<=m;i++){
			scanf("%d %d",&a,&b);
			g[a][d[a]++]=b;
			g[b][d[b]++]=a;
		}
		inicializa();
		marc[1]=1;
		push(1);
		cont = 0;
		while(!vazia()){
			u = pop();
			cont++;

			for(i=0;i<d[u];i++){
				v = g[u][i];
				if(marc[v]==0){
					marc[v]=1;
					push(v);
				}
			}
		}
		printf("Teste %d\n",teste++);
		if(cont==n) printf("normal\n\n");
		else printf("falha\n\n");

	}

	return 0;
```

```
}
```

IMPLEMENTAÇÃO BUSCA EM PROFUNDIDADE C++

```
#include <iostream>
#include <vector>
#include <stack>
#define MAX 101
using namespace std;
vector <int> lista[MAX];
vector <int>::iterator it;
stack <int> pilha;
bool marc[MAX];
int main(){
	int n,m,u,v,i,a,b,cont,teste=1;
	while(true){
		cin >> n >> m;
		if(n==0 && m==0) break;
		for(i=1;i<=n;i++){
			 marc[i]=false;
			 lista[i].clear();
		}
		for(i=1;i<=m;i++){
			cin >> a >> b;
			lista[a].push_back(b);
			lista[b].push_back(a);
		}
		marc[1] = true;
		pilha.push(1);
		cont = 0;
		while(!pilha.empty()){
			u = pilha.top();
			pilha.pop();
			cont++;
			for(it = lista[u].begin(); it!=lista[u].end(); it++){
				if(!marc[*it]){
					marc[*it]=true;
					pilha.push(*it);
				}
			}
		}
		cout << "Teste " << teste++ << endl;
		if(cont==n)
		 cout << "normal" << endl;
		else
		 cout << "falha" << endl;
		cout << endl;
	}
}
```

Fila

IMPLEMENTAÇÂO USANDO VETOR

```
#define MAXN 1000
typedef struct{
	int v[MAXN];
	int fi,ff;
}TFila;
void nova(TFila * fila){
	fila->fi = 0;
	fila->ff = 0;
}
void enfila(TFila *fila,int x){
	fila->v[fila->ff++] = x;
}
int desenfila(TFila *fila){
	return fila->v[fila->fi++];
}
int esta_vazia(TFila fila){
	return fila.fi == fila.ff;
}
void mostra(TFila fila){
	printf("fi %d ff %d\n",fila.fi,fila.ff);
}
int main(){
  TFila fila;
	nova(&fila);
	enfila(&fila,2);
	enfila(&fila,3);
	while(!esta_vazia(fila)){
		printf("%d\n",desenfila(&fila));
		mostra(fila);
	}
	return 0;
}
```

IMPLEMENTAÇÃO em C++

```
#include <iostream>
#include <queue>
#include <stdlib.h>
using namespace std;
queue <int> fila;
int main(){
	fila = queue <int> (); //queue não tem o método clear (limpar)
	fila.push(2);//método push coloca um elmento na fila
	fila.push(3);
	fila.push(4);
	cout << fila.size() << endl; //método size retorna o tamanho da fila
	cout << fila.back() << endl; // método back retorna a referencia
para o ultimo da fila
	while( !fila.empty() ){//método empty testa se a fila está vazia
		cout << fila.front() << endl;//método front retorna o primeiro
elemento da fila
	  fila.pop(); //método pop retira o primeiro elemento da fila
```

```
        }
        system("PAUSE");
}
IMPLEMENTAÇÃO EM JAVA
import java.util.Queue;
import java.util.LinkedList;
import java.util.Scanner;

public class FilaDemo {
        public static void main(String args[] ){
              int n;
       Queue<Integer> fila = new LinkedList<Integer>();
              Scanner scan = new Scanner( System.in );
             n = scan.nextInt();
       fila.add(2);
              fila.add(3);
              fila.add(4);
              while( !fila.isEmpty() ){
                     System.out.println( fila.remove() );

              }

        }

}
```

Fila de Prioridade

```
DOWN(i)
       E  = ESQ(i)
       D  =  DIR(i)
       SE (  E <= TAMANHO_HEAP e A[E] < A[i])
       ENTÃO menor = E
       SENÃO  menor = i
       SE (  D<= TAMANHO_HEAP e A[D] < A[menor])
       ENTÃO menor = D
       SE ( menor != i )
       ENTÃO trocar(A[i],A[menor])
              DOWN(menor)
```

CONSTRUÇÃO FILA DE PRIORIDADE

```
       PARA I:= TAMANHO_HEAP/2 até 1 FAÇA
            DOWN(I)
```

EXTRAIR O MÍNIMO

```
       SE TAMANHO_HEAP < 1 ENTÃO RETURN -1
       MIN = A[1]
       A[1] = A[TAMANHO_HEAP]
```

```
      TAMANHO_HEAP--;
      DOWN(1)
      RETURN MIN
```

DECREASE_KEY(i,chave)
A[i] = chave
ENQUANTO i > 1 e A[PAI(i)] > A[i] **FAÇA**
trocar(A[i],A[PAI(i)])
i = PAI(i)

ALGORITMO PRIM

```
#include <values.h>
const int INF = MAXINT/2;

int fixo[MAXN];
int custo[MAXN]; int total = 0;
for(int i=0; i<n; i++) {
  fixo[i] = 0;
  custo[i] = INF;
}
custo[0] = 0;

for(int faltam = n; faltam>0; faltam--) {
  int no = -1;
  for(int i=0; i<n; i++)
    if(!fixo[i] && (no==-1 || custo[i] < custo[no]))
      no = i;
  fixo[no] = 1;

  if(custo[no] == INF) {
    total = INF;
    break;
  }
  total += custo[no];

  for(int i=0; i<n; i++)
    if(custo[i] > G[no][i])
      custo[i] = G[no][i];
}
```

CAMINHO MÍNIMO

```
#include <values.h>
const int INF = MAXINT/2;

int fixo[MAXN];
int dist[MAXN]; for(int i=0; i<n; i++) {
  fixo[i] = 0;
  dist[i] = INF;
}
dist[0] = 0;

for(int faltam = n; faltam>0; faltam--) {
  int no = -1;
```

```
  for(int i=0; i<n; i++)
    if(!fixo[i] && (no==-1 || dist[i] < dist[no]))
      no = i;
  fixo[no] = 1;

  if(dist[no] == INF)
    break;

  for(int i=0; i<n; i++)
    if(dist[i] > dist[no]+G[no][i])
      dist[i] = dist[no]+G[no][i];
}
```

CAMINHO MÍNIMO COM PRIORITY QUEUE C++

```
#include <stdio.h>
#include <queue>

#define MAXN 1001
#define MAXINT MAXN*MAXN
#define range(i,n,m) for(i=n;i<=m;i++)

using namespace std;

int distances[MAXN];
int father[MAXN];
int visit[MAXN];
int g[MAXN][MAXN];
int n,m;

int dijkstra(int start,int end)
{
  priority_queue<pair<int,int> > queue;
  pair <int,int> nodotmp;
  int i, j;

  range(i,1,n){
    distances[i] = MAXINT;
    father[i] = -1;
    visit[i] = false;
  }

  distances[start] = 0;
  queue.push(pair <int,int> (distances[start], start));

  while(!queue.empty()) {
    nodotmp = queue.top();
    queue.pop();
    i = nodotmp.second;
    if (!visit[i]) {
      visit[i] = true;
```

```
      range(j,1,n)
        if (!visit[j] && g[i][j] > 0 && distances[i] + g[i][j] <
distances[j]) {
          distances[j] = distances[i] + g[i][j];
          father[j] = i;
          queue.push(pair <int,int>(-distances[j], j));
        }
    }
  }
  return distances[end];

}
```

INICIALIZA

```
range(i,1,n) range(j,1,n) g[i][j]=0;
```

Gerando Permutação
Permutação em C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int marc[4];//vetor marc[x] = 1 se o elemento esta na permutação
int v[4]; //vetor original
int p[4]; //vetor da permutação
int i;
int n=3;
int permute(int i){
int j;
if(i<=n){ //escolhendo o i-ésimo elemento
        for(j=1;j<=n;j++){
               if(marc[j]==0){ //se um element não foi escolhido
                      marc[j]=1; // este elemento é marcado
                       p[i]=j;     //  o  i-ésimo  elemento  da  permutação
corresponde
                                   // ao j-ésimo elemento do vetor
                      permute(i+1); //escolhe-se o próximo elemento
                      marc[j]=0;
               }
        }
}else{ //Imprime a permutação escolhida
        printf("%d",v[p[1]]);
        for(j=2;j<=n;j++){
               printf(" %d",v[p[j]]);
        }
        printf("\n");
}
}

int main(){
int i;
```

```
memset(marc,0,sizeof(marc));
for(i=1;i<=n;i++) v[i]=i;
permute(1);
  system("PAUSE");
}
```

Permutações em C++

```
#include <vector>
#include <algorithm>
#include <iostream>
#include <stdlib.h>
using namespace std;

int main(){

int v[] = {1,2,3};
//sort(inicio,fim)
sort (v,v+3); //ordena o vetor

  do {
    cout << v[0] << " " << v[1] << " " << v[2] << endl;
  } while ( next_permutation (v,v+3) ); // gera a próxima permutação

  system("PAUSE");

}
```

Gerando Subconjuntos

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 3
int marc[MAX+1];
int v[MAX+1];
int n=MAX;
void subset(int i){
int j;
if(i<=n){
        marc[i]=0;
        subset(i+1);
        marc[i]=1;
        subset(i+1);
}else{
        printf("{ ");
        for(j=1;j<=n;j++){
               if(marc[j]==1)
                printf("%d ", v[j]);
        }
        printf(" }\n");
}
}
int main(){
int i;
for(i=1;i<=n;i++) v[i]=i;
```

```
subset(1);
system("PAUSE");
}
```

Problema das Oitos Rainhas

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int linha[9];
int d1[15];
int d2[15];
int cont;

void queen(int i){
int j;
if(i<=8){
        //escolhendo onde colocar a i-ésima rainha
        for(j=1;j<=8;j++){
               //nao tem nenhuma outra rainha na mesma linha ou diagonal
               if(linha[j]==0 && d1[i+j-2]==0 && d2[i-j+7]==0){
                      linha[j]=i;
                      d1[i+j-2]=1;
                      d2[i-j+7]=1;
                      queen(i+1);
                      linha[j]=0;
                      d1[i+j-2]=0;
                      d2[i-j+7]=0;
               }
        }
}else{
        cont++;
        printf("%d. ",cont);
        printf("%d",linha[1]);
        for(j=2;j<=8;j++){
               printf(" %d",linha[j]);
        }
        printf("\n");
}


}

int main(){
memset(linha,0,sizeof(linha));
memset(d1,0,sizeof(d1));
memset(d2,0,sizeof(d2));
cont = 0;
queen(1);
  system("PAUSE");
```

```
}
```