

# Aula 4

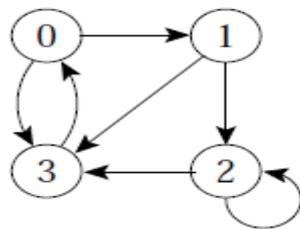
## Implementação de grafos por matriz de adjacência (parte 2)

Profa. Ariane Machado Lima

# Aula passada...

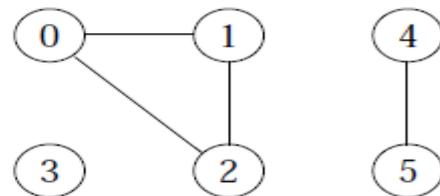
- Implementação por matriz de adjacência  $A_{n \times n}$ ,  $n = \text{nr de vértices}$

## Matriz de Adjacência: Exemplo



	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

(a)



	0	1	2	3	4	5
0		1	1			
1	1		1			
2	1	1				
3						
4						1
5					1	

(b)

# Matriz de Adjacência: Estrutura de Dados

## Arquivo grafo\_matrizadj.h

```
#include <stdbool.h> /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100
#define AN -1 /* aresta nula, ou seja, valor que representa ausencia de aresta */

typedef int Peso;
typedef struct {
    Peso mat[MAXNUMVERTICES][MAXNUMVERTICES];
    int numVertices;
    int numArestas;
} Grafo;
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

Alocação estática de memória

- Velocidade de alocação (em tempo de compilação e não de execução)
- Menos problemas para o programador (*segmentation faults...*)

Você pode querer reutilizar esse código para diferentes tamanhos de grafos... → número de vértice é um parâmetro 5

# Matriz de Adjacência: Estrutura de Dados

## Arquivo grafo\_matrizadj.h

```
#include <stdbool.h> /* variaveis bool assumem valores "true" ou "false" */

#define MAXNUMVERTICES 100
#define AN -1 /* aresta nula, ou seja, valor que representa ausencia de aresta */

typedef int Peso;
typedef struct {
    Peso mat[MAXNUMVERTICES][MAXNUMVERTICES];
    int numVertices;
    int numArestas;
} Grafo;
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5						1

### Flexibilidade no tipo de aresta

/\*  
bool inicializaGrafo(Grafo\* grafo, int nv): Inicializa um grafo com nv vértices  
Preenche as células com **AN** (representando ausência de aresta)  
Vértices vão de 1 a nv.  
Retorna true se inicializou com sucesso e false c.c.  
\*/  
bool inicializaGrafo(Grafo\* grafo, int nv);

```
#include <stdio.h>
#include "grafo_matrizadj.h"
```

## Arquivo grafo\_matrizadj.c

```
/*
 InicializaGrafo(Grafo* grafo, int nv): Inicializa um grafo com nv vertices
 Vertices vao de 1 a nv.
 Preenche as celulas com AN (representando ausencia de aresta)
 Retorna true se inicializou com sucesso e false caso contrario
*/
bool inicializaGrafo(Grafo* grafo, int nv)
{ int i , j ;
  if (nv > MAXNUMVERTICES) {
    fprintf(stderr, "ERRO na chamada de inicializaGrafo: Numero de vertices maior \
que o maximo permitido de %d.\n", MAXNUMVERTICES);

    return false;
  }
  if (nv <= 0) {
    fprintf(stderr, "ERRO na chamada de inicializaGrafo: Numero de vertices deve ser positivo.\n");
    return false;
  }
  grafo->numVertices = nv;
  grafo->numArestas = 0;
  for ( i = 0; i < grafo->numVertices; i++)
    { for ( j = 0; j < grafo->numVertices; j ++ )
      grafo->mat[i][j] = AN;
    }
  return true;
}
```

Chamada da função:

```
Grafo g;
inicializaGrafo(&g, 10);
```

# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
→ inicializaGrafo	$O(v^2)$	
→ imprimeGrafo	$O(v^2)$	
insereAresta		
existeAresta		
removeAresta		
listaAdjVazia		
proxListaAdj		
liberaGrafo		

# Arquivo testa\_grafo.c

```
#include "grafo_matrizadj.h"
#include <stdio.h>

int main()
{
    Grafo g1;
    int numVertices;

    //inicializaGrafo(&g1, 10);

    do {
        printf("Digite o número de vértices do grafo\n");
        scanf("%d", &numVertices);
    } while (!inicializaGrafo(&g1, numVertices));

    //imprimeGrafo(&g1);

    return 0;
}
```

# Para compilar tudo (por enquanto...)

```
$ gcc -c grafo_matrizadj.c
```

```
$ gcc -c testa_grafo.c
```

```
$ gcc -o testa_grafo.exe grafo_matrizadj.o testa_grafo.o
```

# Dúvidas?



# Aula de hoje

- Demais operações – assumindo que o grafo é **direcionado**
- Makefile

# Inserção de aresta

Quais parâmetros e retorno?

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

```
typedef int Peso;  
typedef struct {  
    Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];  
    int numVertices;  
    int numArestas;  
} Grafo;
```

# Inserção de aresta

```
/*  
void insereAresta(int v1, int v2, Peso peso, Grafo *grafo):  
  Insere a aresta (v1, v2) com peso "peso" no grafo.  
  Não verifica se a aresta já existia (isso deve ser feito pelo usuário antes, se necessário).  
*/  
void insereAresta(int v1, int v2, Peso peso, Grafo *grafo){
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5						1

```
typedef int Peso;  
typedef struct {  
  Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];  
  int numVertices;  
  int numArestas;  
} Grafo;
```

# Inserção de aresta

```
/*  
 void insereAresta(int v1, int v2, Peso peso, Grafo *grafo):  
 Inse a aresta (v1, v2) com peso "peso" no grafo.  
 Nao verifica se a aresta ja existia (isso deve ser feito pelo usuario antes, se necessario).  
*/  
void insereAresta(int v1, int v2, Peso peso, Grafo *grafo){  
  if (! (verificaValidadeVertice(v1, grafo) && verificaValidadeVertice(v2, grafo)))  
    return;  
  grafo->mat[v1][v2] = peso;  
  grafo->numArestas++;  
}
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5						1

```
typedef int Peso;  
typedef struct {  
  Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];  
  int numVertices;  
  int numArestas;  
} Grafo;
```

/\*

bool verificaValidadeVertice(int v, Grafo \*grafo): verifica se o nr do vertice eh valido no grafo, ou seja, se ele é maior ou igual a zero e menor que o nr total de vertices do grafo.

\*/

```
bool verificaValidadeVertice(int v, Grafo *grafo){
    if (v > grafo->numVertices) {
        fprintf(stderr, "ERRO: Numero do vertice (%d) maior ou igual que o numero total de vertices\
(%d).\n", v, grafo->numVertices);
        return false;
    }
    if (v < 0) {
        fprintf(stderr, "ERRO: Numero do vertice (%d) deve ser não negativo.\n", v);
        return false;
    }
    return true;
}
```



# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
<b>inicializaGrafo</b>	$O(v^2)$	
<b>imprimeGrafo</b>	$O(v^2)$	
→ <b>insereAresta</b>		
<b>existeAresta</b>		
<b>removeAresta</b>		
<b>listaAdjVazia</b>		
<b>proxListaAdj</b>		
<b>liberaGrafo</b>		

# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
<b>inicializaGrafo</b>	$O(v^2)$	
<b>imprimeGrafo</b>	$O(v^2)$	
→ <b>insereAresta</b>	$O(1)$	
<b>existeAresta</b>		
<b>removeAresta</b>		
<b>listaAdjVazia</b>		
<b>proxListaAdj</b>		
<b>liberaGrafo</b>		

# Existência de aresta

Quais parâmetros e retorno?

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

```
typedef int Peso;  
typedef struct {  
    Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];  
    int numVertices;  
    int numArestas;  
} Grafo;
```

# Existência de aresta

```
/*  
 bool existeAresta(int v1, int v2, Grafo *grafo):  
 Retorna true se existe a aresta (v1, v2) no grafo e false caso contrário  
*/  
bool existeAresta(int v1, int v2, Grafo *grafo){
```

Essa é exercício, não vou mostrar (mas o que você acha que precisa fazer?)

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5						1

```
typedef int Peso;  
typedef struct {  
    Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];  
    int numVertices;  
    int numArestas;  
} Grafo;
```

# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
<b>inicializaGrafo</b>	$O(v^2)$	
<b>imprimeGrafo</b>	$O(v^2)$	
<b>insereAresta</b>	$O(1)$	
→ <b>existeAresta</b>		
<b>removeAresta</b>		
<b>listaAdjVazia</b>		
<b>proxListaAdj</b>		
<b>liberaGrafo</b>		

# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
<b>inicializaGrafo</b>	$O(v^2)$	
<b>imprimeGrafo</b>	$O(v^2)$	
<b>insereAresta</b>	$O(1)$	
→ <b>existeAresta</b>	$O(1)$	
<b>removeAresta</b>		
<b>listaAdjVazia</b>		
<b>proxListaAdj</b>		
<b>liberaGrafo</b>		

# Obtenção do peso da aresta (similar ao teste de existência)

```
/*  
  Peso obtemPesoAresta(int v1, int v2, Grafo *grafo):  
  Retorna o peso da aresta (v1, v2) no grafo se ela existir e AN caso contrário  
*/  
Peso obtemPesoAresta(int v1, int v2, Grafo *grafo)  
{
```

$O(1)$

# Remoção de aresta

Quais parâmetros e retorno?

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

```
typedef int Peso;  
typedef struct {  
    Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];  
    int numVertices;  
    int numArestas;  
} Grafo;
```

# Remoção de aresta

```
/*  
bool removeAresta(int v1, int v2, Peso* peso, Grafo *grafo);  
Remove a aresta (v1, v2) do grafo colocando AN em sua célula (representando ausencia de aresta).  
Se a aresta existia, coloca o peso dessa aresta em "peso" e retorna true,  
caso contrario retorna false (e "peso" é inalterado).  
*/  
bool removeAresta(int v1, int v2, Peso* peso, Grafo *grafo){  
    if (! (verificaValidadeVertice(v1, grafo) && verificaValidadeVertice(v2, grafo)))  
        return false;  
  
    /* se aresta existe */  
    if (grafo->mat[v1][v2] != AN) {  
        *peso = grafo->mat[v1][v2];  
        grafo->mat[v1][v2] = AN;  
        grafo->numArestas--;  
        return true;  
    }  
    /* aresta nao existe */  
    return false;  
}
```

# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
<b>inicializaGrafo</b>	$O(v^2)$	
<b>imprimeGrafo</b>	$O(v^2)$	
<b>insereAresta</b>	$O(1)$	
<b>existeAresta</b>	$O(1)$	
→ <b>removeAresta</b>		
<b>listaAdjVazia</b>		
<b>proxListaAdj</b>		
<b>liberaGrafo</b>		

# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
<b>inicializaGrafo</b>	$O(v^2)$	
<b>imprimeGrafo</b>	$O(v^2)$	
<b>insereAresta</b>	$O(1)$	
<b>existeAresta</b>	$O(1)$	
→ <b>removeAresta</b>	$O(1)$	
<b>listaAdjVazia</b>		
<b>proxListaAdj</b>		
<b>liberaGrafo</b>		

# Verificação se a lista de adjacência de um vértice é vazia

listaAdjVazia: **true** se a lista de adjacentes de um dado vértice é vazia, **false** c.c.

Quais parâmetros e retorno? O que faz?

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

```
typedef int Peso;  
typedef struct {  
    Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];  
    int numVertices;  
    int numArestas;  
} Grafo;
```

# Verificação se a lista de adjacência de um vértice é vazia

```
/*  
    bool listaAdjVazia(int v, Grafo* grafo):  
    Retorna true se a lista de adjacencia (de vertices adjacentes)  
    do vertice v é vazia, e false caso contrário.  
*/  
bool listaAdjVazia(int v, Grafo* grafo){  
    if (!verificaValidadeVertice(v, grafo))  
        return true;  
  
    int i;  
    for (i = 0; i < grafo->numVertices; i++)  
        if (grafo->mat[v][i] != AN) return false;  
    return true;  
}
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5						1

```
typedef int Peso;  
typedef struct {  
    Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];  
    int numVertices;  
    int numArestas;  
} Grafo;
```

# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
<b>inicializaGrafo</b>	$O(v^2)$	
<b>imprimeGrafo</b>	$O(v^2)$	
<b>insereAresta</b>	$O(1)$	
<b>existeAresta</b>	$O(1)$	
<b>removeAresta</b>	$O(1)$	
<b>listaAdjVazia</b>		
<b>proxListaAdj</b>		
<b>liberaGrafo</b>		



# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
<b>inicializaGrafo</b>	$O(v^2)$	
<b>imprimeGrafo</b>	$O(v^2)$	
<b>insereAresta</b>	$O(1)$	
<b>existeAresta</b>	$O(1)$	
<b>removeAresta</b>	$O(1)$	
→ <b>listaAdjVazia</b>	$O(v)$	
<b>proxListaAdj</b>		
<b>liberaGrafo</b>		

# Próximo da lista de adjacência

proxListaAdj: retorna o próximo vértice adjacente de um dado vértice (próximo em relação a um adjacente “atual” passado como parâmetro); na primeira chamada retorna o primeiro; pense no que fazer se não houver próximo...

Ex: quem é o próximo adjacente de 0, sendo que o (vértice adjacente a 0) atual é 1?  
Ou seja, quem vem depois do vért. 1 na lista de adjacentes do vért. 0?

Quais parâmetros e retorno?

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

```
typedef int Peso;  
typedef struct {  
    Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];  
    int numVertices;  
    int numArestas;  
} Grafo;
```

# Próximo da lista de adjacência

proxListaAdj: retorna o próximo vértice adjacente de um dado vértice (próximo em relação a um adjacente “atual” passado como parâmetro); na primeira chamada retorna o primeiro; pense no que fazer se não houver próximo...

Ex: quem é o próximo adjacente de 0, sendo que o (vértice adjacente a 0) atual é 1?  
Ou seja, quem vem depois do vért. 1 na lista de adjacentes do vért. 0?

```
/*  
int proxListaAdj(int v, Grafo* grafo, int atual):  
Trata-se de um iterador sobre a lista de adjacência do vertice v.  
Retorna o proximo vertice adjacente a v, partindo do vertice "atual" adjacente a v  
ou VERTICE_INVALIDO se a lista de adjacencia tiver terminado sem um novo proximo.  
*/  
int proxListaAdj(int v, Grafo* grafo, int atual){
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

```
typedef int Peso;  
typedef struct {  
    Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];  
    int numVertices;  
    int numArestas;  
} Grafo;
```

# Próximo da lista de adjacência

```
/*
int proxListaAdj(int v, Grafo* grafo, int atual):
Trata-se de um iterador sobre a lista de adjacência do vertice v.
Retorna o proximo vertice adjacente a v, partindo do vertice "atual" adjacente a v
ou VERTICE_INVALIDO se a lista de adjacencia tiver terminado sem um novo proximo.
*/
int proxListaAdj(int v, Grafo* grafo, int atual){
    if (!verificaValidadeVertice(v, grafo))
        return VERTICE_INVALIDO;
    atual++;
    while ((atual < grafo->numVertices) && (grafo->mat[v][atual] == AN))
        atual++;
    if (atual >= grafo->numVertices) {
        return VERTICE_INVALIDO;
    }
    return atual;
}
```

Ex: o próximo adjacente de 0, sendo que o atual é 1, é o 3

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

```
typedef int Peso;
typedef struct {
    Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];
    int numVertices;
    int numArestas;
} Grafo;
```

# grafo\_matrizadj.h

```
#define MAXNUMVERTICES 100
#define AN -1 /* aresta nula, ou seja, valor que representa ausencia de aresta */
#define VERTICE_INVALIDO -1 /* numero de vertice invalido ou ausente */

typedef int Peso;
typedef struct {
    Peso mat[MAXNUMVERTICES][MAXNUMVERTICES];
    int numVertices;
    int numArestas;
} Grafo;
```

# Primeiro da lista de adjacência

`proxListaAdj(v, grafo, ?);`

# Primeiro da lista de adjacência

```
proxListaAdj(v, grafo, -1);
```

# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
<b>inicializaGrafo</b>	$O(v^2)$	
<b>imprimeGrafo</b>	$O(v^2)$	
<b>insereAresta</b>	$O(1)$	
<b>existeAresta</b>	$O(1)$	
<b>removeAresta</b>	$O(1)$	
<b>listaAdjVazia</b>	$O(v)$	
<b>proxListaAdj</b>		
<b>liberaGrafo</b>		



# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
<b>inicializaGrafo</b>	$O(v^2)$	
<b>imprimeGrafo</b>	$O(v^2)$	
<b>insereAresta</b>	$O(1)$	
<b>existeAresta</b>	$O(1)$	
<b>removeAresta</b>	$O(1)$	
<b>listaAdjVazia</b>	$O(v)$	
<b>proxListaAdj</b>	$O(v)$	
<b>liberaGrafo</b>		



# Liberação do espaço em memória

Quais parâmetros e retorno? O que precisa fazer?

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

```
typedef int Peso;  
typedef struct {  
    Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];  
    int numVertices;  
    int numArestas;  
} Grafo;
```

# Liberação do espaço em memória

```
/* Não precisa fazer nada para matrizes de adjacência */  
void liberaGrafo(Grafo* grafo){}
```

	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

```
typedef int Peso;  
typedef struct {  
    Peso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];  
    int numVertices;  
    int numArestas;  
} Grafo;
```

# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
<b>inicializaGrafo</b>	$O(v^2)$	
<b>imprimeGrafo</b>	$O(v^2)$	
<b>insereAresta</b>	$O(1)$	
<b>existeAresta</b>	$O(1)$	
<b>removeAresta</b>	$O(1)$	
<b>listaAdjVazia</b>	$O(v)$	
<b>proxListaAdj</b>	$O(v)$	
<b>liberaGrafo</b>		



# Complexidades

(considerando um grafo de  $v$  vértices e  $a$  arestas)

	Matriz de adj.	?
<b>inicializaGrafo</b>	$O(v^2)$	
<b>imprimeGrafo</b>	$O(v^2)$	
<b>insereAresta</b>	$O(1)$	
<b>existeAresta</b>	$O(1)$	
<b>removeAresta</b>	$O(1)$	
<b>listaAdjVazia</b>	$O(v)$	
<b>proxListaAdj</b>	$O(v)$	
<b>liberaGrafo</b>	$O(1)$	



# Adiantando o EP 1:

- Implemente (em C) a estrutura de dados e as operações aqui definidas, utilizando matriz de adjacência, para grafos direcionados.
- Faça o mesmo para grafos não-direcionados

# Compilando com gcc

```
gcc -o programa.exe programa.c
```

# Compilando com gcc

Para vários módulos:

```
gcc -c grafo_matrizadj.c // gera grafo_matrizadj.o
```

```
gcc -c testa_grafo.c // gera testa_grafo.o
```

```
gcc -o testa_grafoMat.exe grafo_matrizadj.o testa_grafo.o
```

# Compilando com gcc

Para vários módulos:

```
gcc -c part1.c      // gera part1.o
```

```
gcc -c part2.c      // gera part2.o
```

...

```
gcc -c partn.c      // gera partn.o
```

```
gcc -c main.c       // gera main.o
```

```
gcc -o myprogram.exe part1.o part2.o ... partn.o main.o
```

# Compilando com gcc

Para vários módulos:

```
gcc -c part1.c      // gera part1.o
```

```
gcc -c part2.c      // gera part2.o
```

...

```
gcc -c partn.c      // gera partn.o
```

```
gcc -c main.c       // gera main.o
```

```
gcc -o myprogram.exe part1.o part2.o ... partn.o main.o
```

Chato....

Geralmente nos perguntamos: “Fiz alguma modificação no arquivo parti.c?”

Tem que recompilar tudo de novo!

# Makefile

Arquivo contendo a definição de regras para compilação do programa

# Makefile - Ex

```
all: myprogram.exe

myprogram.exe: part1.o part2.o ... partn.o main.o
    gcc -o myprogram.exe part1.o part2.o ... partn.o main.o

part1.o: part1.c part1.h
    gcc -c part1.c      # gera part1.o

part2.o: part2.c part2.h
    gcc -c part2.c      # gera part2.o

...

partn.o: partn.c partn.h
    gcc -c partn.c      # gera partn.o

main.o: main.c
    gcc -c main.c      # gera main.o
```



# Makefile - Ex

```
all: myprogram.exe
```

```
myprogram.exe: part1.o part2.o ... partn.o main.o
```

```
gcc -o myprogram.exe part1.o part2.o ... partn.o main.o
```

```
part1.o: part1.c part1.h
```

```
gcc -c part1.c      # gera part1.o
```

```
part2.o: part2.c part2.h
```

```
gcc -c part2.c      # gera part2.o
```

```
...
```

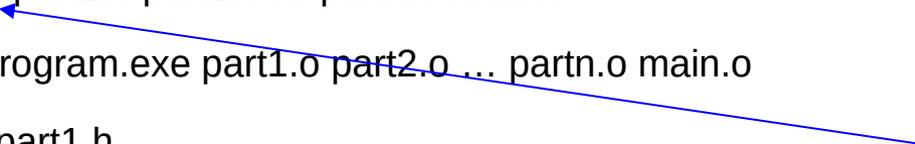
```
partn.o: partn.c partn.h
```

```
gcc -c partn.c      # gera partn.o
```

```
main.o: main.c
```

```
gcc -c main.c       # gera main.o
```

alvo



# Makefile - Ex

```
all: myprogram.exe
```

```
myprogram.exe: part1.o part2.o ... partn.o main.o
```

dependências

```
gcc -o myprogram.exe part1.o part2.o ... partn.o main.o
```

```
part1.o: part1.c part1.h
```

```
gcc -c part1.c      # gera part1.o
```

```
part2.o: part2.c part2.h
```

```
gcc -c part2.c      # gera part2.o
```

```
...
```

```
partn.o: partn.c partn.h
```

```
gcc -c partn.c      # gera partn.o
```

```
main.o: main.c
```

```
gcc -c main.c       # gera main.o
```



# Makefile - Ex

```
all: myprogram.exe
```

```
myprogram.exe: part1.o part2.o ... partn.o main.o
```

```
gcc -o myprogram.exe part1.o part2.o ... partn.o main.o
```

```
part1.o: part1.c part1.h
```

```
gcc -c part1.c      # gera part1.o
```

```
part2.o: part2.c part2.h
```

```
gcc -c part2.c      # gera part2.o
```

```
...
```

```
partn.o: partn.c partn.h
```

```
gcc -c partn.c      # gera partn.o
```

```
main.o: main.c
```

```
gcc -c main.c       # gera main.o
```

tab



# Makefile - Ex

```
all: myprogram.exe
```

```
myprogram.exe: part1.o part2.o ... partn.o main.o
```

```
gcc -o myprogram.exe part1.o part2.o ... partn.o main.o
```

comando

```
part1.o: part1.c part1.h
```

```
gcc -c part1.c      # gera part1.o
```

```
part2.o: part2.c part2.h
```

```
gcc -c part2.c      # gera part2.o
```

```
...
```

```
partn.o: partn.c partn.h
```

```
gcc -c partn.c      # gera partn.o
```

```
main.o: main.c
```

```
gcc -c main.c       # gera main.o
```



# Makefile - Ex

```
all: myprogram.exe
```

```
myprogram.exe: part1.o part2.o ... partn.o main.o
```

```
    gcc -o myprogram.exe part1.o part2.o ... partn.o main.o
```

```
part1.o: part1.c part1.h
```

```
    gcc -c part1.c    # gera part1.o
```

```
part2.o: part2.c part2.h
```

```
    gcc -c part2.c    # gera part2.o
```

```
...
```

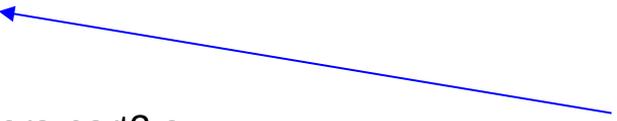
```
partn.o: partn.c partn.h
```

```
    gcc -c partn.c    # gera partn.o
```

```
main.o: main.c
```

```
    gcc -c main.c     # gera main.o
```

comentário



# Makefile - Ex

```
all: myprogram.exe

myprogram.exe: part1.o part2.o ... partn.o main.o
    gcc -o myprogram.exe part1.o part2.o ... partn.o main.o

part1.o: part1.c part1.h
    gcc -c part1.c      # gera part1.o

part2.o: part2.c part2.h
    gcc -c part2.c      # gera part2.o

...

partn.o: partn.c partn.h
    gcc -c partn.c      # gera partn.o

main.o: main.c
    gcc -c main.c       # gera main.o
```

Uso na linha de comando:

```
make          // faz o primeiro alvo
```

ou

```
make <alvo>
```



EACH  
USP

# Alguns links sobre Makefiles

[http://www.ntu.edu.sg/home/ehchua/programming/cpp/gcc\\_make.html](http://www.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html)

<http://www.cs.usask.ca/staff/oster/makefiles.html>

<http://www.gnu.org/software/make/manual/make.html>

# Nosso Makefile

```
testa_matriz: grafo_matrizadj.o testa_grafo.o
    gcc -o testa_grafo_matriz.exe grafo_matrizadj.o testa_grafo.o

grafo_matrizadj.o: grafo_matrizadj.c grafo_matrizadj.h
    gcc -c grafo_matrizadj.c

testa_grafo.o: testa_grafo.c grafo_matrizadj.h
    gcc -c testa_grafo.c

clean:
    rm -f *.o *.exe
```

# Nosso Makefile

```
testa_matriz: grafo_matrizadj.o testa_grafo.o
    gcc -o testa_grafo_matriz.exe grafo_matrizadj.o testa_grafo.o

grafo_matrizadj.o: grafo_matrizadj.c grafo_matrizadj.h
    gcc -c grafo_matrizadj.c

testa_grafo.o: testa_grafo.c grafo_matrizadj.h
    gcc -c testa_grafo.c

clean:
    rm -f *.o *.exe
```

Executando (no linux, via linha de comando):

```
$ ./testa_grafo_matriz.exe
```

Ou redirecionando a entrada de um arquivo (com símbolo "<"):

```
$ ./testa_grafo_matriz.exe < entrada_teste.txt
```

Ou redirecionando também a saída para um arquivo (com símbolo ">"):

```
$ ./testa_grafo_matriz.exe < entrada_teste.txt > saida.txt
```

Ou redirecionando também a saída de erro para um outro arquivo (com símbolo "2>"):

```
$ ./testa_grafo_matriz.exe < entrada_teste.txt > saida.txt 2> erro.txt
```

# Arquivo testa\_grafo.c

```
#include "grafo_matrizadj.h"
#include <stdio.h>

int main()
{
    Grafo g1;
    int numVertices;

    //inicializaGrafo(&g1, 10);

    do {
        printf("Digite o número de vértices do grafo\n");
        scanf("%d", &numVertices);
    } while (!inicializaGrafo(&g1, numVertices));

    //imprimeGrafo(&g1);

    return 0;
}
```

# Matriz de adjacência - Reflexões

Essa representação por matriz adjacência é sempre eficiente?

	<b>Matriz de adj.</b>
<b>inicializaGrafo</b>	$O(v^2)$
<b>imprimeGrafo</b>	$O(v^2)$
<b>insereAresta</b>	$O(1)$
<b>existeAresta</b>	$O(1)$
<b>removeAresta</b>	$O(1)$
<b>listaAdjVazia</b>	$O(v)$
<b>proxListaAdj</b>	$O(v)$
<b>liberaGrafo</b>	$O(1)$