Aula 3 –

Implementação de grafos por matriz de adjacência

Prof. Helton Hideraldo Bíscaro

O Tipo Abstratos de Dados Grafo

- Importante considerar os algoritmos em grafos como **tipos abstratos de dados**.
- Conjunto de operações associado a uma estrutura de dados.
- Independência de implementação para as operações.

Exemplos….

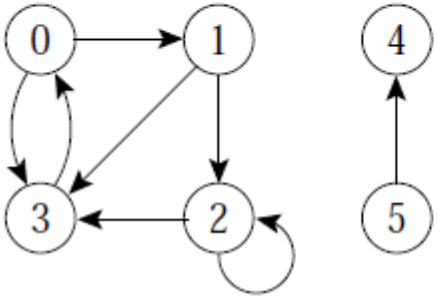
Operadores do TAD Grafo

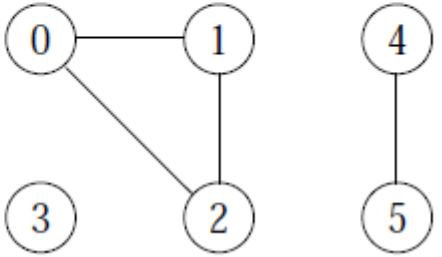
1. *InicializaGrafoVazio(Grafo)*: Inicializa um grafo vazio (sem arestas)
2. *InsereAresta(V1,V2,Peso, Grafo)*: Insere uma aresta no grafo.
3. *ExisteAresta(V1,V2,Grafo)*: Verifica se existe uma determinada aresta.
4. Obtem a lista de vértices adjacentes a um determinado vértice (tratada a seguir).
5. *RetiraAresta(V1,V2,Peso, Grafo)*: Retira uma aresta do grafo.
6. *LiberaGrafo(Grafo)*: Liberar o espaço ocupado por um grafo.
7. *ImprimeGrafo(Grafo)*: Imprime um grafo.
8. *GrafoTransposto(Grafo,GrafoT)*: Obtém o transposto de um grafo direcionado.
9. *RetiraMin(A)*: Obtém a aresta de menor peso de um grafo com peso nas arestas.

Matriz de Adjacência

- A matriz de adjacência de um grafo $G = (V, A)$ contendo n vértices é uma matriz $n \times n$ de *bits*, onde $A[i, j]$ é 1 (ou verdadeiro) se e somente se existe um arco do vértice i para o vértice j.
- Para grafos ponderados $A[i, j]$ contém o rótulo ou peso associado com a aresta e, neste caso, a matriz não é de *bits*.
- Se não existir uma aresta de i para j então é necessário utilizar um valor que não possa ser usado como rótulo ou peso.

Matriz de Adjacência: Exemplo





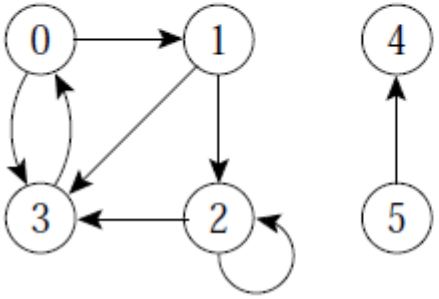
	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5					1	

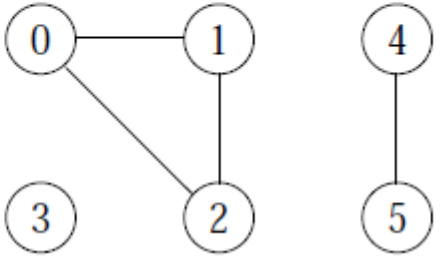
(a)

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

(b)

Matriz de Adjacência: 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 simétrica!
Poderíamos armazenar apenas
a parte triangular inferior ou superior,
mas não compensa

Matriz de Adjacência: Estrutura de Dados

Arquivo 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 */

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

typedef int TipoPeso;
typedef struct  {
  TipoPeso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];
  int numVertices;
  int numArestas;
} TipoGrafo;
typedef int TipoApontador;
```

Matriz de Adjacência: Operações

```c
#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 */

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

typedef int TipoPeso;
typedef struct  {
  TipoPeso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];
  int numVertices;
  int numArestas;
} TipoGrafo;
typedef int TipoApontador;
```

void inicializaGrafo(TipoGrafo* grafo, int nv);

Matriz de Adjacência: Operações

```c
#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 */

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

typedef int TipoPeso;
typedef struct  {
  TipoPeso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];
  int numVertices;
  int numArestas;
} TipoGrafo;
typedef int TipoApontador;
```

```c
/*
  void inicializaGrafo(TipoGrafo* grafo, int nv): Inicializa um grafo com nv vértices
  Preenche as células com AN (representando ausência de aresta)
  Vértices vão de 1 a nv.
*/
void inicializaGrafo(TipoGrafo* grafo, int nv);
```

Pausa para dicas de programação (modularização em C)

- A sua estrutura de dados de grafos poderá ser necessária em vários programas diferentes

→

Pausa para dicas de programação (modularização em C)

- A sua estrutura de dados de grafos poderá ser necessária em vários programas diferentes

  → deveria estar encapsulada em um módulo

Para isso, criaremos um arquivo .h e um .c para implementar apenas a estrutura de dados de grafos por matriz de adjacência (e suas operações) (como fazemos com classes em Java).

Um outro programa .c irá testar esse módulo (implementação de grafo).

Pausa para dicas de programação (mensagens de erro)

- Mensagens de erro devem ser claras:
  - Saber qual é o problema
  - Saber onde está o problema (função)
  - Ser facilmente detectada
- Podemos separar a saída padrão (normal do programa – por default a tela) da saída de erro padrão (por default a tela)
  - stdout (saída padrão), stderr (saída de erro)
  - **printf**(…), e **fprintf**(stdout, ….) imprime na saída padrão
  - **fprintf**(stderr, ….) imprime na saída de erro
  - Podem ser redirecionadas para arquivos, para a entrada de outros de programas ou para um “buraco negro”
  - Separar as duas saídas facilita identificar erros ou outras mensagens, principalmente aquelas que não são emitidas logo antes de um “exit”.

Pausa para dicas de programação (mensagens de erro - Exemplo)

```
/*
  InicializaGrafo(TipoGrafo* 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(TipoGrafo* 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 de ser positivo.\n");
    return false;
  }
  grafo->numVertices = nv;
  for ( i = 0; i <= grafo->numVertices; i++)
    { for ( j = 0; j <= grafo->numVertices; j ++)
        grafo->mat[i][j] = AN;
    }
  return true;
}
```

Compilando com gcc

```
gcc -o hello.exe hello.c
```

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

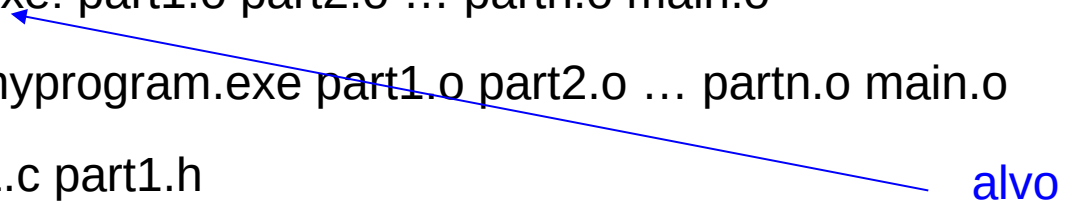
Arquivo contendo a definição de regras para compilação do programa

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

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

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

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



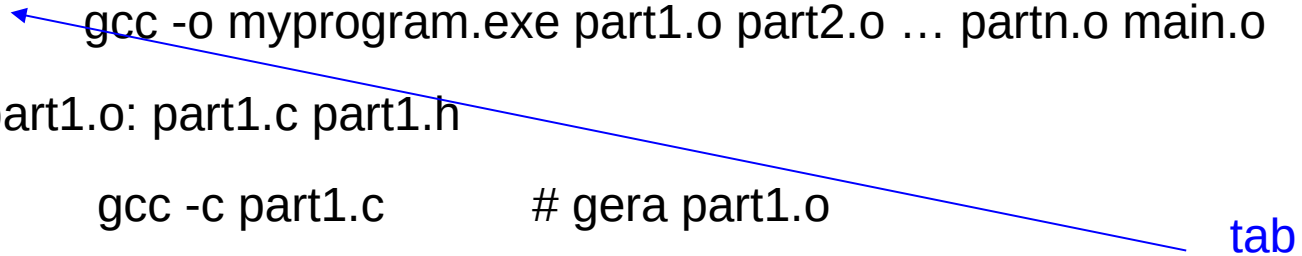
Makefile - Ex

```
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

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

dependências

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

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



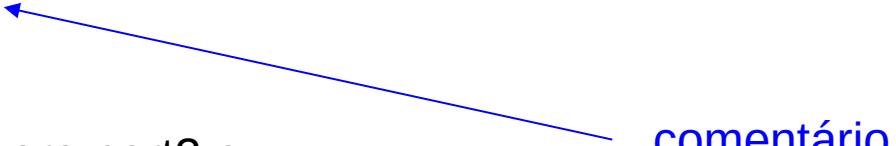
Makefile - Ex

```
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

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

comando

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

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



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

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

Uso na linha de comando:

make // faz o primeiro alvo

ou

make <alvo>

Alguns links sobre Makefiles

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: matriz_adjacencia/grafo_matrizadj.c matriz_adjacencia/grafo_matrizadj.h
	gcc -c matriz_adjacencia/grafo_matrizadj.c

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

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

testa_grafo.o: testa_grafo.c matriz_adjacencia/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

Matriz de Adjacência: Operações

```c
#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 */

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

typedef int TipoPeso;
typedef struct  {
  TipoPeso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];
  int numVertices;
  int numArestas;
} TipoGrafo;
typedef int TipoApontador;
```

```c
/*
  void insereAresta(int v1, int v2, TipoPeso peso, TipoGrafo *grafo):
  Insere a aresta (v1, v2) com peso "peso" no grafo
*/

void insereAresta(int v1, int v2, TipoPeso peso, TipoGrafo *grafo);
```

Matriz de Adjacência: Operações

```c
#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 */

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

typedef int TipoPeso;
typedef struct  {
  TipoPeso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];
  int numVertices;
  int numArestas;
} TipoGrafo;
typedef int TipoApontador;
```

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

bool existeAresta(int v1, int v2, TipoGrafo *grafo);
```

Matriz de Adjacência: Operações

```c
#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 */

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

typedef int TipoPeso;
typedef struct  {
  TipoPeso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];
  int numVertices;
  int numArestas;
} TipoGrafo;
typedef int TipoApontador;
```

```c
/*
  bool removeAresta(int v1, int v2, TipoPeso* peso, TipoGrafo *grafo);
  Remove a aresta (v1, v2) do grafo colocando AN em sua celula (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, TipoPeso* peso, TipoGrafo *grafo);
```

Matriz de Adjacência: Operações

```c
#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 */

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

typedef int TipoPeso;
typedef struct  {
  TipoPeso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];
  int numVertices;
  int numArestas;
} TipoGrafo;
typedef int TipoApontador;
```

```c
/*
   bool listaAdjVazia(int v, TipoGrafo* grafo):
   Retorna true se a lista de adjacencia (de vertices adjacentes) do vertice v é vazia, e false caso contrário.
*/

bool listaAdjVazia(int v, TipoGrafo* grafo);
```

Matriz de Adjacência: Operações

```c
#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 */

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

typedef int TipoPeso;
typedef struct  {
  TipoPeso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];
  int numVertices;
  int numArestas;
} TipoGrafo;
typedef int TipoApontador;
```

```c
/*
   TipoApontador primeiroListaAdj(int v, TipoGrafo* grafo):
   Retorna o primeiro vertice da lista de adjacencia de v.
*/

TipoApontador primeiroListaAdj(int v, TipoGrafo* grafo);
```

Matriz de Adjacência: Operações

```c
#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 */

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

typedef int TipoPeso;
typedef struct  {
  TipoPeso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];
  int numVertices;
  int numArestas;
} TipoGrafo;
typedef int TipoApontador;
```

```c
/*
   TipoApontador proxListaAdj(int v, TipoGrafo* grafo, TipoApontador 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
*/

TipoApontador proxListaAdj(int v, TipoGrafo* grafo, TipoApontador atual);
```

Matriz de Adjacência: Operações

```c
#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 */

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

typedef int TipoPeso;
typedef struct  {
  TipoPeso mat[MAXNUMVERTICES + 1][MAXNUMVERTICES + 1];
  int numVertices;
  int numArestas;
} TipoGrafo;
typedef int TipoApontador;
```

```c
/*
   void imprimeGrafo(TipoGrafo* grafo):
   Imprime a matriz de adjacencia do grafo.
   Assuma que cada vértice e cada peso de aresta são inteiros de até 2 dígitos.
*/

void imprimeGrafo(TipoGrafo* grafo);
```

Para a próxima aula

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

- O que mudaria na implementação de grafos não direcionados?

Referências

Livro do Ziviani (cap 7)