Algoritmos e Estruturas de Dados II

Grafos – tipo abstrato de dados

Thiago A. S. Pardo
Profa. M. Cristina
Material de aula da Profa.
Josiane M. Bueno

Grafos
Estruturas de Dados

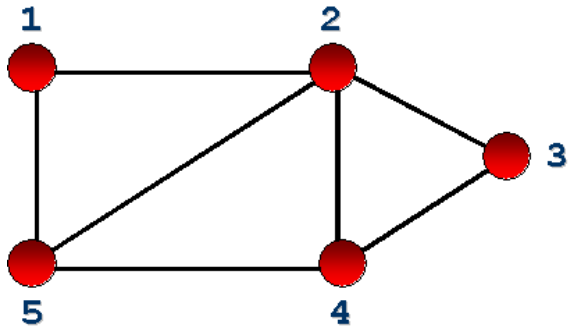
- Como vimos na última aula...

- **Matriz de Adjacências**
  - Custo em espaço: $|V|^2$
  - Matriz simétrica para grafos não orientados ➔ redundância e eventualmente esparsidade
  - Acesso direto a cada aresta [i,j]
- Alternativa: só representar as arestas presentes no grafo ➔ **Listas de Adjacência**

Grafos

Matriz de Adjacências

- **Exemplo**



$M =$

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

vértices

Grafos

Listas de Adjacências

- Dado um grafo $G = (V, E)$, as **listas de adjacências** A é um conjunto de |V| listas $A(v)$, uma para cada vértice v pertencente a V

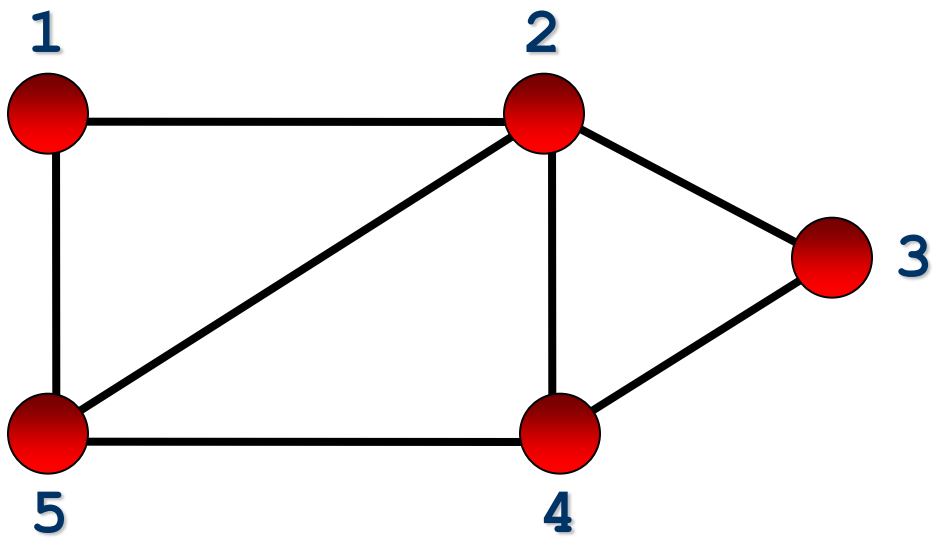
- Cada lista $A(v)$ é denominada **lista de adjacências** do vértice v e contém os vértices w adjacentes a v em G

- Ou seja, as **listas de adjacências** consistem um vetor de |V| elementos que são capazes de apontar, cada um, para uma lista linear
  - O i-ésimo elemento do vetor aponta para a lista linear das arestas que são adjacentes ao vértice i

Grafos

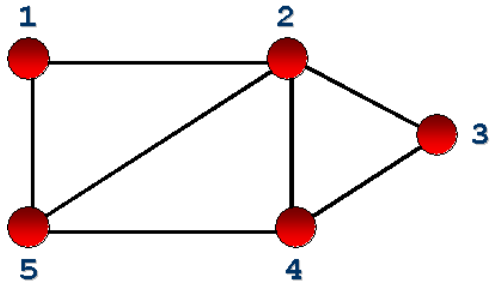
Listas de Adjacências

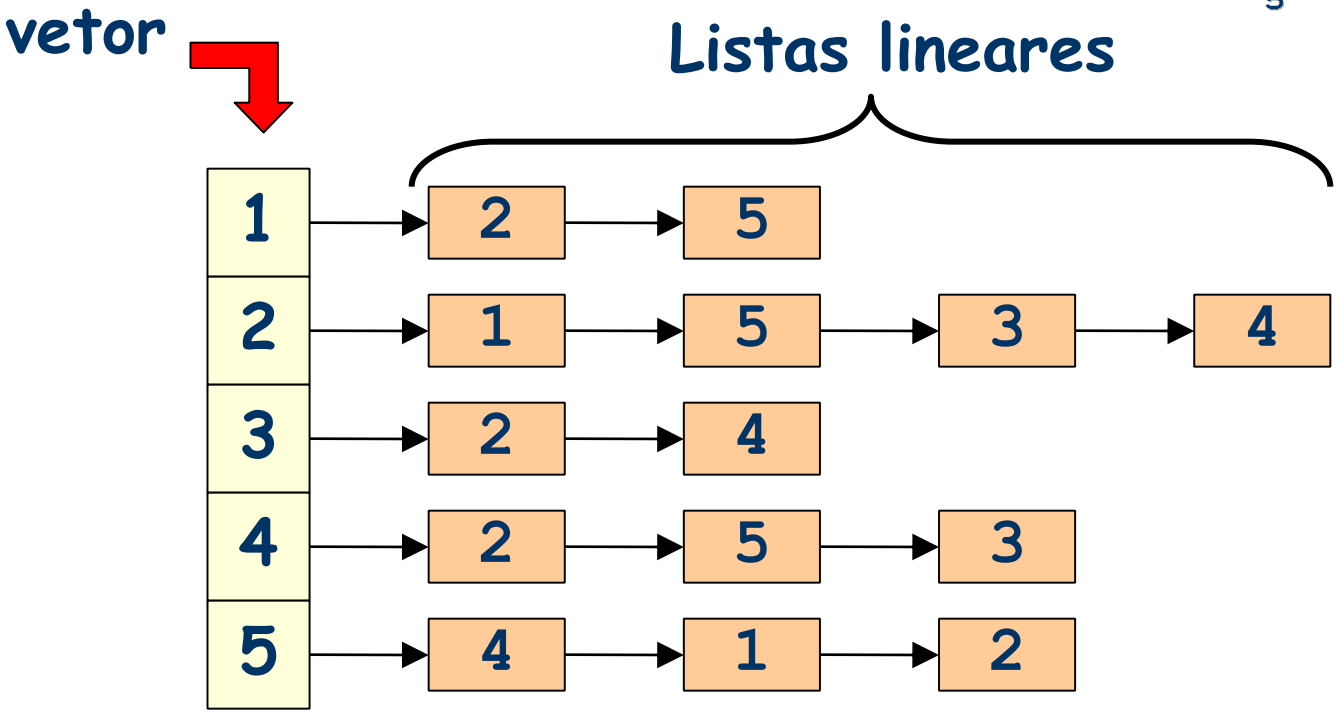
- Como são as listas de adjacências do grafo a seguir?



Grafos
Listas de Adjacências

- Possível resposta:





Grafos

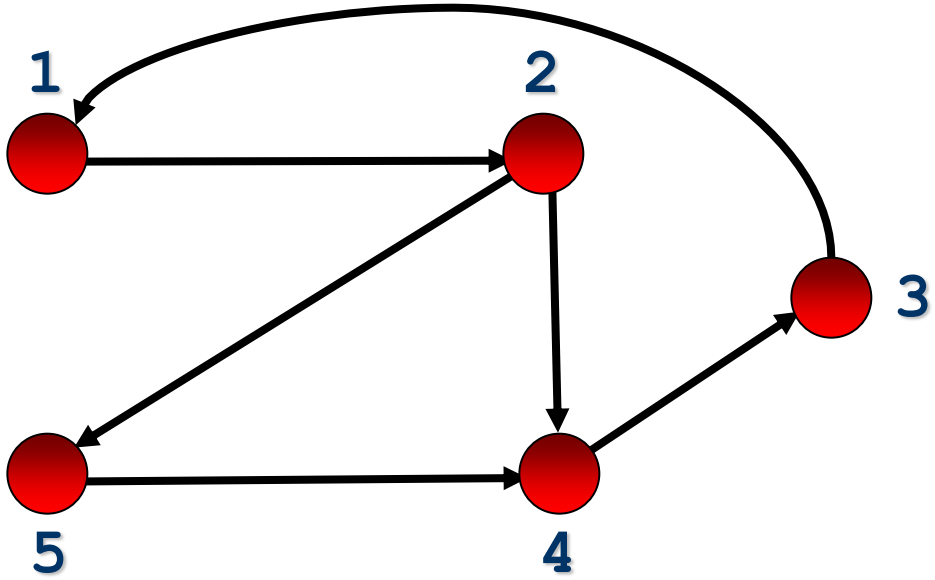
Listas de Adjacências

- **Em grafos não direcionados, cada aresta é representada 2 vezes**

Grafos

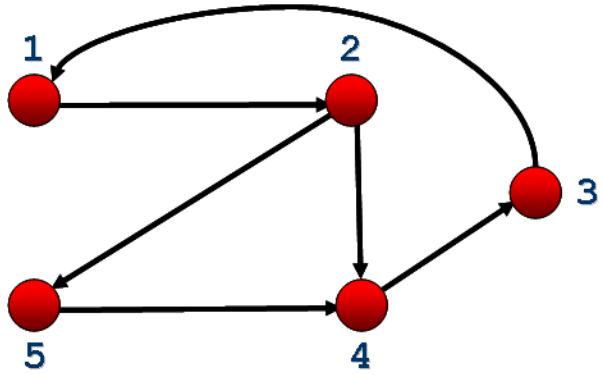
Listas de Adjacências

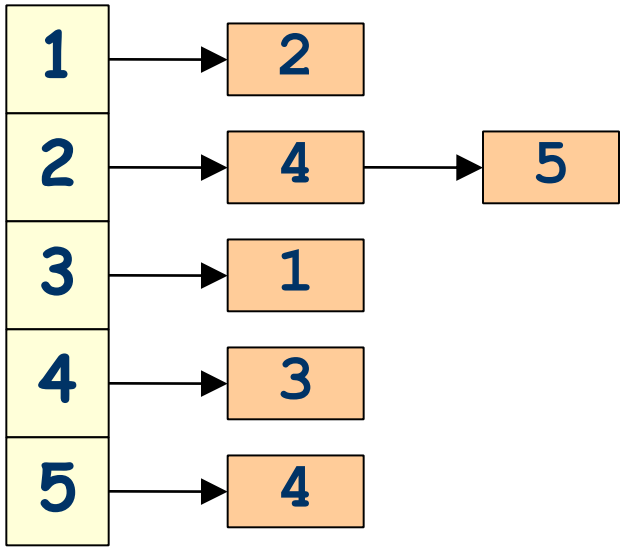
- Como representar o dígrafo abaixo?



Grafos
Listas de Adjacências

- **Possível resposta:**



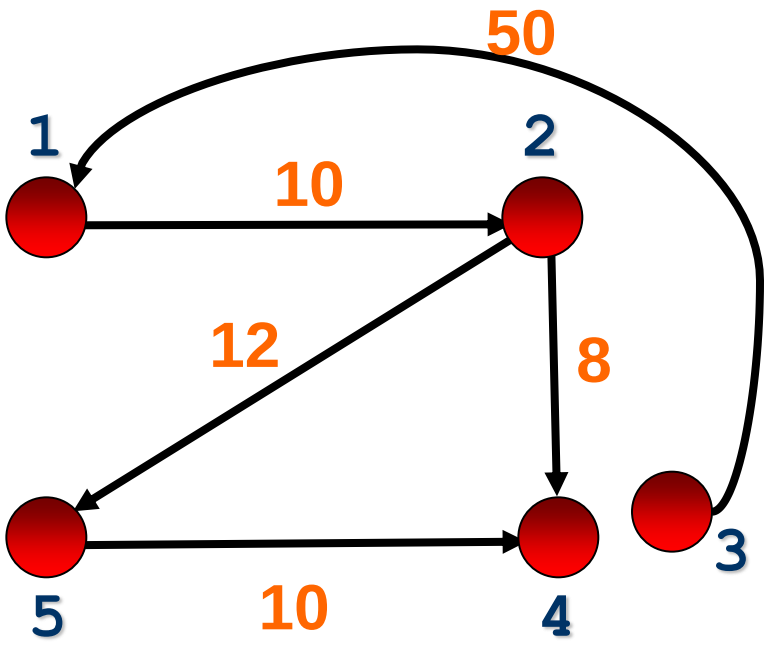


Vértice i aponta para vértices adjacentes

Grafos

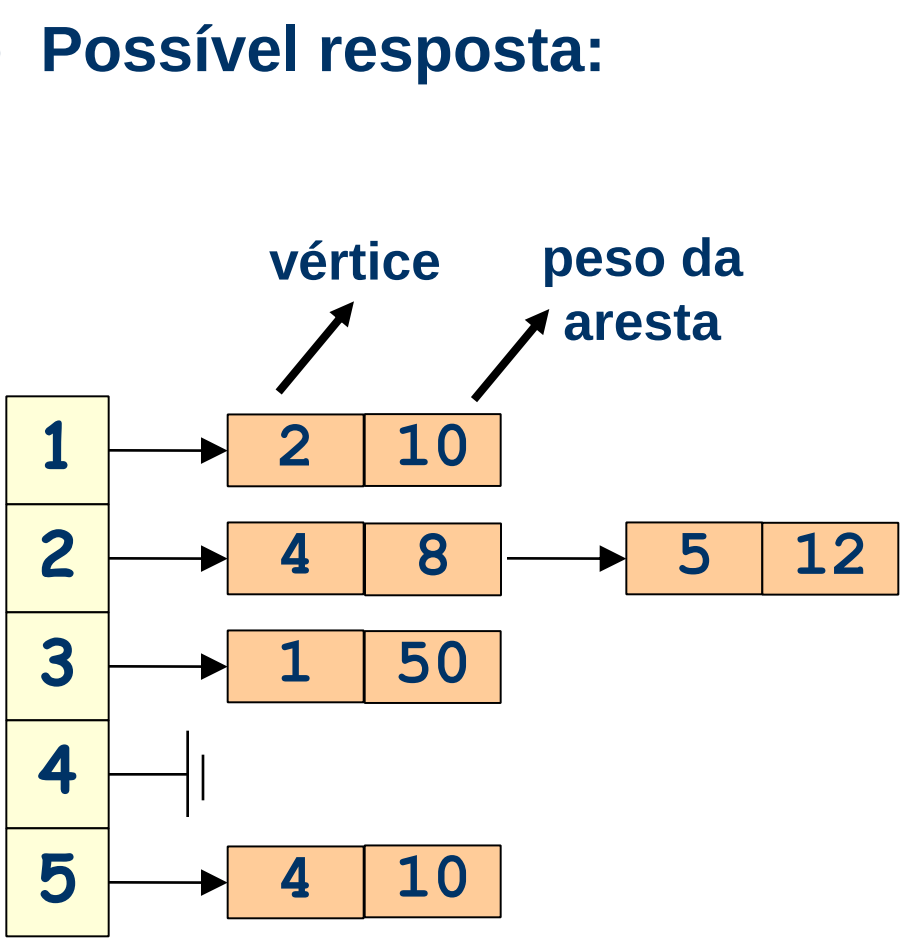
Listas de Adjacências

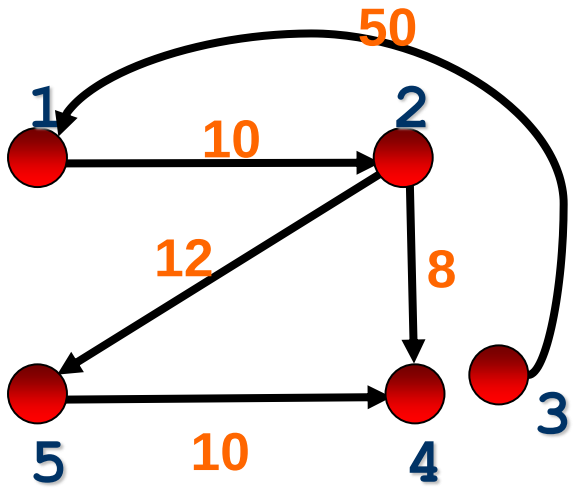
- Como representar o grafo direcionado e valorado abaixo?



Grafos
Listas de Adjacências

- **Possível resposta:**





Grafos

Listas de Adjacências

- Maior complexidade na representação
- Propriedades
  - Armazenamento: ?
  - Teste se aresta (i,j) está no grafo: ?

Grafos

Listas de Adjacências

- Maior complexidade na representação

- Propriedades
  - Armazenamento: O(|V| + |E|)
  - Teste se aresta (i,j) está no grafo:

  não é mais direto➔O(d_i), com d_i sendo o grau do vértice i

Grafos

Listas de Adjacências

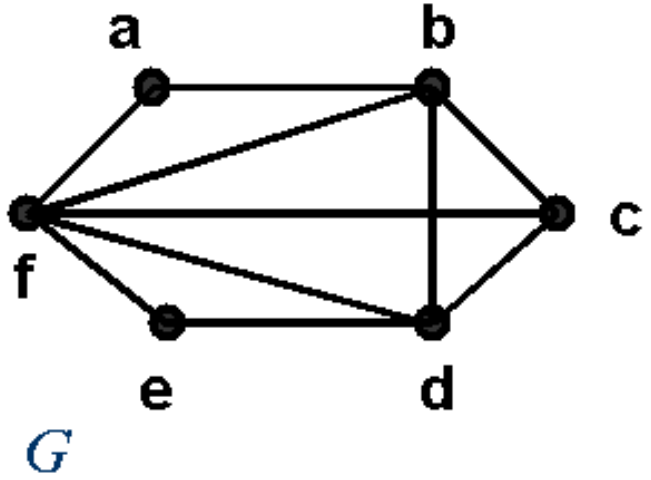
- Boa representação para grafos esparsos, em que |E| é muito menor do que $|V|^2$
- Representação compacta e usualmente utilizada na maioria das aplicações
- Desvantagem: tempo O(|V|) para determinar se existe uma aresta entre i e j, pois podem haver |V| elementos na lista de adjacências de i *(contra O(1) na matriz de adjacência)*

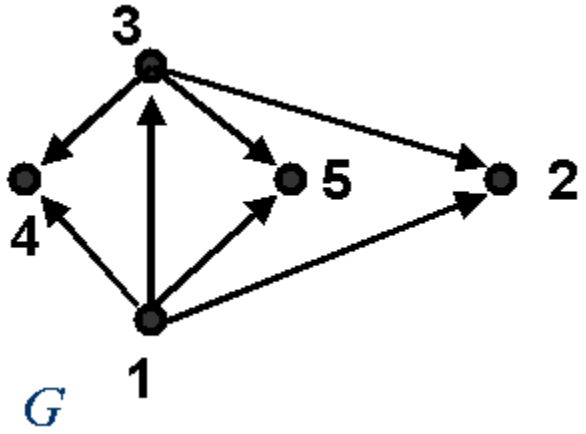
Grafos
Listas de Adjacências

- Atenção
  - Os vértices adjacentes a um vértice i podem ser armazenados na lista de adjacências de i em ordem arbitrária ou não
  - Como em qualquer lista, há liberdade para haver variações na representação (nós cabeça, sentinelas, etc.)
  - Haveria vantagem em representar cada lista de adjacência dos vértices num array?

Grafos

Exercício de Fixação





- **Represente os grafos acima utilizando listas de adjacências**

Grafos
Listas de Adjacências

- Implementação de algumas das operações mais comuns
  - Criar grafo vazio
  - Inserir aresta
  - Retirar aresta
  - Existe aresta?
  - Obter lista de vértices adjacentes a um determinado vértice
    - Lista está vazia?
    - Retornar primeiro vértice da lista
    - Retornar próximo vértice adjacente da lista
  - Liberar memória utilizada pelo grafo
  - Imprimir grafo

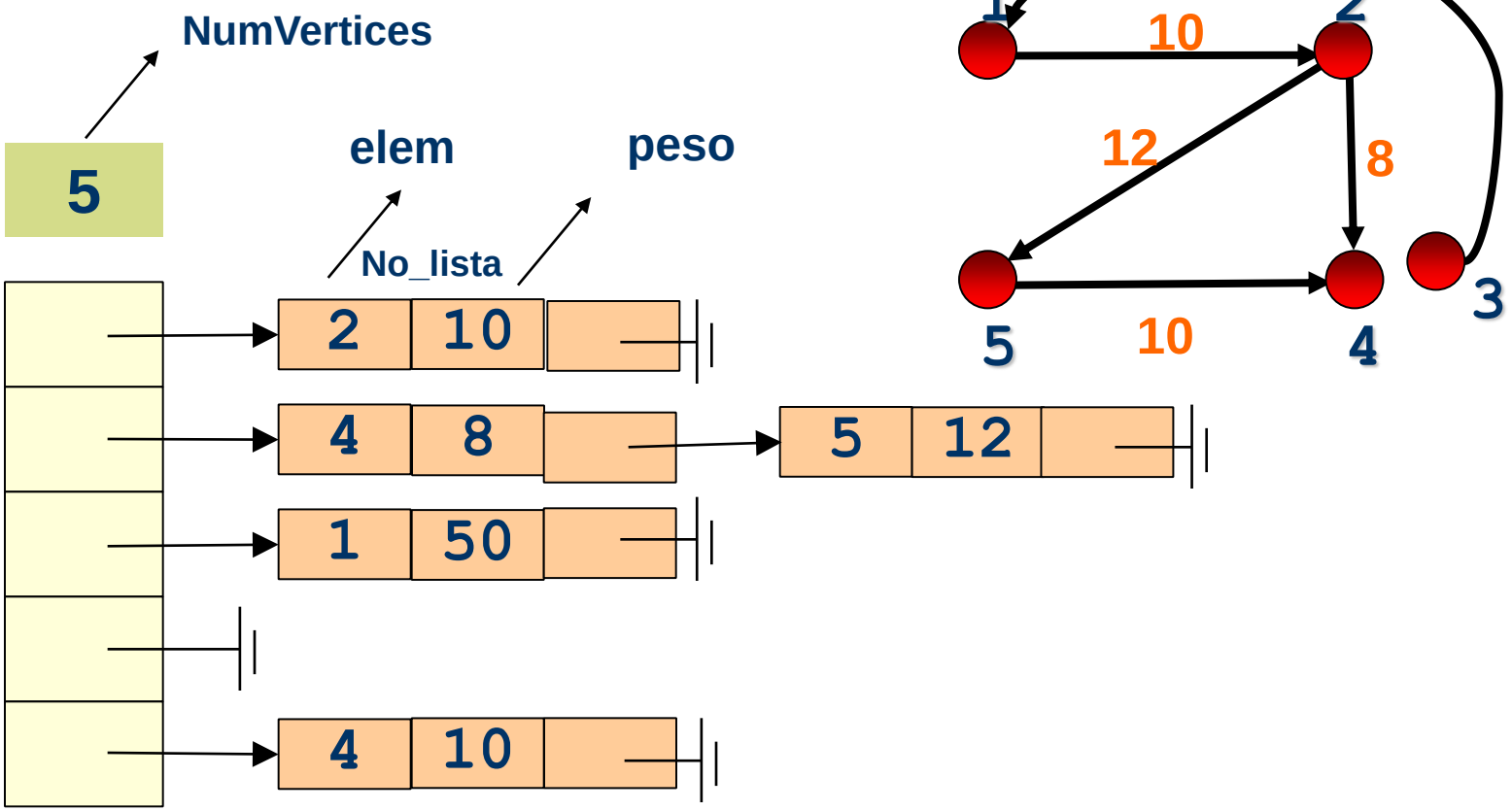
GrafoNaoDirecionado.h (Listas Adj)

```
typedef int elem;
typedef struct no_lista {
        int v;
        elem peso;
        struct no_lista *prox;
} no_lista;
typedef struct {
        no_lista *inicio;
} no_vertice;
typedef struct {
no_vertice LAdj[MaxNumVertices];
   int NumVertices;
} Grafo;
```

```
void criar(Grafo*, int, int*);

void inserir_aresta(Grafo*, int, int,
elem, int*);

int existe_aresta(Grafo*, int, int, int*);

void retirar_aresta(Grafo*, int, int,
elem*, int*);

int grau_vertice(Grafo*, int, int*);

void imprimir(Grafo*);
```

Grafos
Listas de Adjacências

- **No TAD Grafo:**



GrafoNaoDirecionado.c (Listas Adj)

```
#include “GrafoNaoDirecionado.h"

/*função que inicializa um grafo com um determinado número de vértices dado pelo usuário */
void criar(Grafo *G, int NumVertices, int *erro) {
    if (NumVertices>MaxNumVertices)
        *erro=1;
      else {
          *erro=0;
          G->NumVertices=NumVertices;
          int i;
          for (i=1; i<=G->NumVertices; i++) {
              G->LAdj[i].inicio= NULL;
          }
          }
}
```

GrafoNaoDirecionado.c

```
/*função que insere uma aresta de peso P entre V1 e V2 nas listas de V1 e V2 – vistas como Pilhas*
void inserir_aresta(Grafo *G, int V1, int V2, elem Peso, int *erro) {
    if ((V1>G->NumVertices) || (V2>G->NumVertices))     *erro=1;
    else {
        *erro=0;   no_lista *p;
        p=(no_lista*) malloc(sizeof(no_lista));
        p->prox= G->LAdj[V1].inicio;
        p->v=V2;
        p->peso=Peso;
        G->LAdj[V1].inicio=p;
        p=(no_lista*) malloc(sizeof(no_lista));
        p->prox= G->LAdj[V2].inicio;
        p->v=V1;
        p->peso=Peso;
        G->LAdj[V2].inicio=p;
    }
}
```

GrafoNaoDirecionado.c

```
/*função que verifica se uma aresta existe entre 2 vértices*/
int existe_aresta(Grafo *G, int V1, int V2, int *erro) {
if ((V1>G->NumVertices) || (V2>G->NumVertices)) {
        *erro=1;
        return 0; }
    else {
         *erro=0;
         int encontrou=0;
         no_lista *aux=G->LAdj[V1].inicio;
         while ((aux!=NULL) && (!encontrou))
               if (aux->v==V2)
                  encontrou=1;
               else aux=aux->prox;
         return(encontrou);
    }
}
```

GrafoNaoDirecionado.c

/*função que retira uma aresta do grafo, retornando seu peso*/

void retirar_aresta(Grafo *G, int V1, int V2, elem *P, int *erro)

Fazer em casa

Ideia: divida em duas partes (1) retirar (V1,V2); (2) retirar (V2, V1)

GrafoNaoDirecionado.c

```
/*função que calcula o grau de um vértice*/

int grau_vertice(Grafo *G, int V1, int *erro) {
    int grau;
    if (V1>G->NumVertices) *erro=1;
    else {
          *erro=0
          grau=0;
          no_lista *aux = G->LAdj[V1].inicio;
          while (aux!=NULL) {
                    grau++;
                   aux = aux->prox;
             }
         }
      return grau;
 }
```

Grafos
Listas de Adjacências

- **Exercício**
  - **Implementar sub-rotina que encontra o vértice adjacente a x com aresta de menor peso em um grafo valorado e direcionado**

Grafos
Comparação

Comparação	Vencedor
Rapidez para saber se(x,y) está no grafo	Matriz de adjacências
Rapidez para determinar o grau de um vértice	Equivalentes
Menor memória em grafos pequenos	Listas: $(\|V\| + \|E\|)$ Matriz: $\|V\|^2$

Grafos
Comparação

Comparação	Vencedor
Menos memória em grafos grandes	Matriz de adjacências
Inserção/remoção de arestas	Matriz: O(1) Listas: O(1)
Melhor na maioria dos problemas	Listas de adjacências
Rapidez para percorrer o grafo	Listas: $O(\|V\| + \|E\|)$ Matriz: $O(\|V\|^2)$