

Algoritmos e Estruturas de Dados II

Grafos II: Estruturas de Dados

Ricardo J. G. B. Campello

Parte deste material é baseado em adaptações e extensões de slides disponíveis em <http://ww3.datastructures.net> (Goodrich & Tamassia).

1

Organização

- ◆ TAD Grafo
- ◆ Estruturas de Dados para Grafos
 - Lista de Arestas
 - Lista de Adjacências
 - Matriz de Adjacências
- ◆ Análise Assintótica Comparativa
 - Desempenhos das Principais Operações do TAD
- ◆ Exemplo de Implementação Simples
 - Código C de uma ED Alternativa

2

TAD Grafo

◆ Dados: vértices e arestas

- relações de incidência
- elementos armazenados

◆ Operações (Gerais):

- **endVertices**(G, e): retorna os dois vértices finais da aresta e (e.g. arranjo $c/2$ elementos).
- **opposite**(G, v, e): retorna o vértice oposto a v na aresta e .
- **areAdjacent**(G, v, w): verdadeiro se e somente se os vértices v e w forem adjacentes.
- **replaceVertex**(G, v, x): substitui o elemento do vértice v por x .
- **replaceEdge**(G, e, x): substitui o elemento da aresta e por x .

◆ Operações (de Atualização):

- **insertVertex**(G, o): insere um novo vértice isolado, armazenando nele o elemento o , e retorna uma referência.
- **insertEdge**(G, v, w, o): insere uma aresta (v, w) , armazenando nela o elemento o , e retorna uma referência.
- **removeVertex**(G, v): remove o vértice v (e suas arestas incidentes) e retorna o elemento armazenado nele.
- **removeEdge**(G, e): remove aresta e , retornando o elemento armazenado.

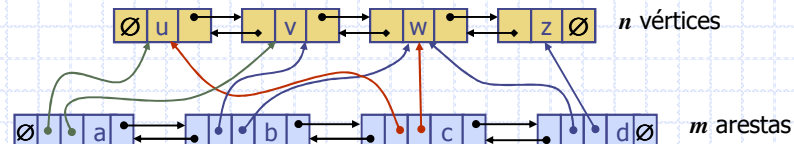
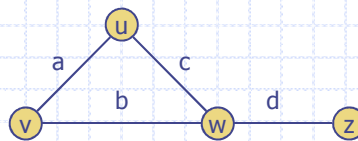
◆ Operações (para Iteradores):

- **incidentEdges**(G, v): retorna iterador das arestas incidentes em v .
- **vertices**(G): retorna iterador dos vértices do grafo G .
- **edges**(G): retorna iterador das arestas do grafo G .

Nota: Métodos Específicos podem ser Adicionados para Grafos Direcionados

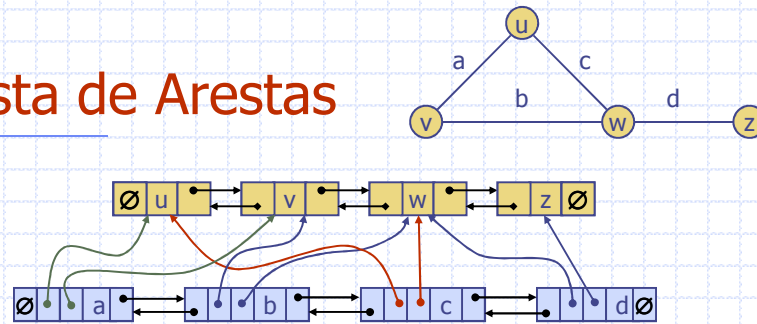
3

Estrutura de Dados – Lista de Arestas



4

Lista de Arestas



◆ Principais Desvantagens:

- Verificar as arestas incidentes em um dado vértice v demanda percorrer toda a lista de arestas
 - ◆ Operação **incidentEdges**(G, v) do TAD é $O(m)$
- O mesmo vale para checar se dois vértices v e w são adjacentes, bem como para remover um dado vértice v
 - ◆ Operação **areAdjacent**(G, v, w) do TAD é $O(m)$
 - ◆ Operação **removeVertex**(G, v) do TAD é $O(m)$

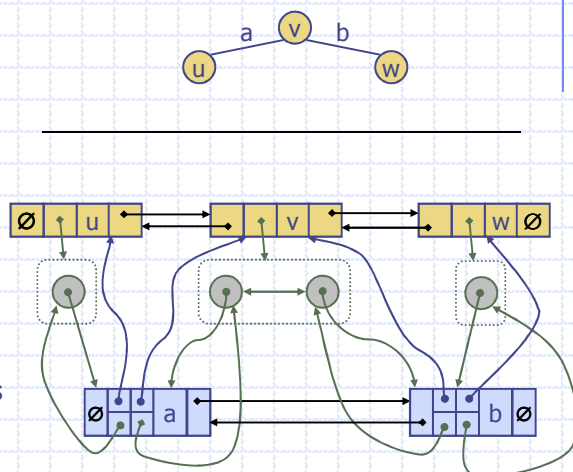
5

Estrutura de Dados - Lista de Adjacências

- ◆ A cada elemento da lista de vértices é adicionado um ponteiro para uma lista de incidências:

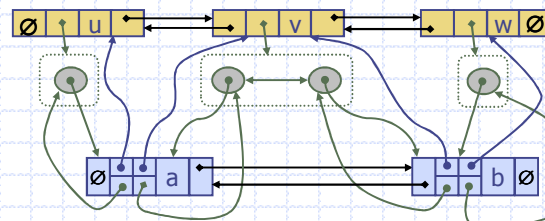
- lista de ponteiros para suas arestas incidentes.

- ◆ A cada elemento da lista de arestas são adicionados ponteiros para as posições correspondentes nas listas de incidência dos seus vértices.



6

Lista de Adjacências



- A partir de um vértice v , podemos acessar suas arestas incidentes em tempo proporcional ao seu número:

 - $\text{incidentEdges}(G, v)$ do TAD é $O(\deg(v))$
 - $\text{removeVertex}(G, v)$ do TAD é $O(\deg(v))$
- Podemos verificar se dois vértices v e w são adjacentes percorrendo a menor dentre as respectivas listas auxiliares de incidência:

 - $\text{areAdjacent}(G, v, w)$ do TAD é $O(\min(\deg(v), \deg(w)))$

7

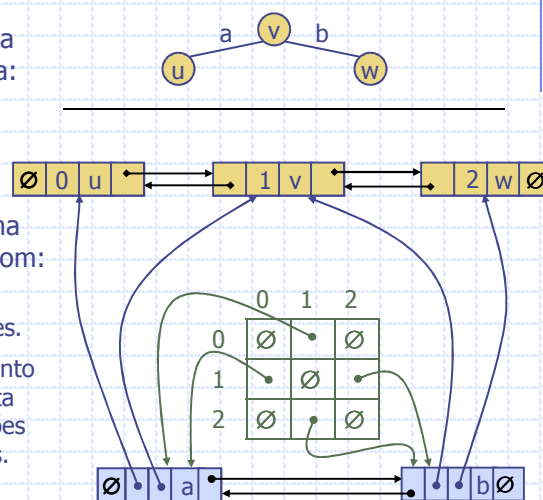
Estrutura de Dados – Matriz de Adjacências

- A cada elemento da lista de vértices é adicionada:

- Uma chave (índice) associada ao vértice.

- Acrescenta-se ainda uma matriz de adjacências com:

- Nulo nas posições de vértices não adjacentes.
 - Ponteiro para o elemento correspondente da lista de arestas, nas posições de vértices adjacentes.



8

Desempenho Assintótico das Principais Operações do TAD

◆ Grafo simples com n vértices e m arestas ◆ Notação <i>Big O</i>	Lista de Arestas	Lista de Adjacências	Matriz de Adjacências
Espaço (memória)	$n + m^*$	$n + m^*$	$n^2 + m$
incidentEdges (G, v)	m	$\deg(v)$	n
areAdjacent (G, v, w)	m	$\min(\deg(v), \deg(w))$	1
insertVertex (G, o)	1	1	n^2^*
insertEdge (G, v, w, o)	1	1	1
removeVertex (G, v)	m	$\deg(v)$	n^2^*
removeEdge (G, e)	1	1	1

9

Implementação Simples: Estrutura Alternativa (Skiena & Revilla, 2003)

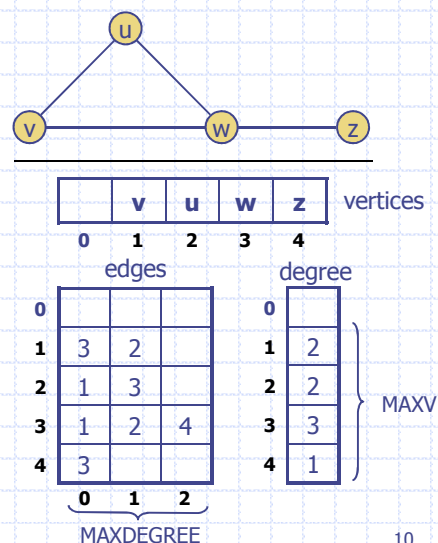
◆ Vértices rotulados:

- Chaves (índices) são associadas aos vértices

◆ Arestas sem elementos.

◆ Grafos estáticos, ou:

- Inserções e remoções sem redimensionamento das matrizes e vetores
- Matrizes e vetores superdimensionados para:
 - ◆ No. de vértices (MAXV)
 - ◆ Grau (MAXDEGREE)



10

Estrutura Alternativa (Skiena & Revilla, 2003)

```

/* graph.h */

#define MAXV      100      /* maximum number of vertices */
#define MAXDEGREE 50      /* maximum outdegree of a vertex */

typedef struct {
    int edges[MAXV+1][MAXDEGREE]; /* adjacency info */
    int degree[MAXV+1];
    int nvertices;
    int nedges;
} graph;

#include "graph.h"
initialize_graph(graph *g) {
    int i,j; /* counters */

    g->nvertices = 0;
    g->nedges = 0;

    for (i=1; i<=MAXV; i++) {
        g->degree[i] = 0;
        for (j=1; j<=MAXDEGREE; j++)
            g->edges[i][j] = 0;
    }
}

```

Nota: Como as arestas não são representadas explicitamente, essa estrutura não permite a implementação das operações **endVertices**, **opposite** e **replaceEdge** do TAD Grafo.

Estrutura Alternativa (Skiena & Revilla, 2003)

```

#include "bool.h"
#include "graph.h"

read_graph(graph *g, bool directed) {
    int i; /* counter */
    int m; /* number of edges */
    int x, y; /* vertices in edge (x,y) */

    initialize_graph(g);

    scanf("%d %d", &(g->nvertices), &m);

    for (i=1; i<=m; i++) {
        scanf("%d %d", &x, &y);
        insert_edge(g, x, y, directed);
    }
}

```

```

/* bool.h */

#define TRUE 1
#define FALSE 0

typedef int bool;

```

Estrutura Alternativa (Skiena & Revilla, 2003)

```
insert_edge(graph *g, int x, int y, bool directed) {
    if (g->degree[x] > MAXDEGREE)
        printf("Warning: insertion(%d,%d) exceeds max degree\n", x, y);

    g->edges[x][g->degree[x]] = y;
    g->degree[x]++;

    if (directed == FALSE) insert_edge(g, y, x, TRUE);
    else g->nedges++;
}
```

Duas arestas
direcionadas,
(x,y) e (y,x),
representando
uma aresta
não-direcionada

```
delete_edge(graph *g, int x, int y, bool directed) {
    int i; /* counter */
    for (i=0; i<g->degree[x]; i++)
        if (g->edges[x][i] == y) {
            g->degree[x]--;
            g->edges[x][i] = g->edges[x][g->degree[x]];
            if (directed == FALSE)
                delete_edge(g, y, x, TRUE);
            return;
        }
    printf("Warning: deletion(%d,%d) not found in g.\n", x, y);
}
```

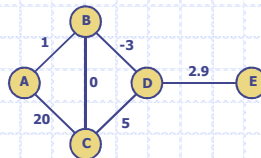
Exercícios

1. Na implementação simples de grafo em C discutida em aula, não se incluiu na estrutura principal (struct graph) um vetor vertices para armazenar os elementos dos vértices. Modifique essa estrutura de tal forma que esse vetor seja contemplado. Nota: Considere que os elementos nos vértices são strings com no máx. 30 caracteres e que vértices inexistentes armazenam nulo (\0).
2. Na implementação simples de grafo em C discutida em aula, foram apresentadas as rotinas correspondentes às operações **insertEdge** e **removeEdge** do TAD Grafo (rotinas `insert_edge` e `delete_edge`). Já considerando a modificação no código realizada no Exercício 1, implemente as rotinas correspondentes às seguintes operações:
 - **areAdjacent**, **replaceVertex**, **removeVertex**, **insertVertex**
 - Nota: Após remover um vértice, é preciso remover todas as suas arestas incidentes (em ambos os sentidos, para arestas não direcionadas), o que implica reestruturar a matriz edges de forma apropriada. Pode-se aproveitar essa reestruturação para preencher o espaço deixado pelo vértice removido e permitir que inserções sejam feitas sempre ao final.

Exercícios

3. Represente graficamente o grafo abaixo em cada uma das três diferentes estruturas de dados clássicas discutidas em aula:

- lista de arestas, lista de adjacências e matriz de adjacências



4. Ignore os pesos das arestas e repita o exemplo anterior para a estrutura alternativa também discutida. Nesse caso, represente o conteúdo da matriz `edges` e dos vetores `degree` e `vertices`.
5. Elabore alguns diferentes grafos e repita os Exercícios 3 e 4.
6. Explique a razão de cada uma das complexidades computacionais da tabela do slide 9, além daquelas que foram discutidas.

15

Referências

- ◆ M. T. Goodrich and R. Tamassia, *Data Structures and Algorithms in C++/Java*, John Wiley & Sons, 2002/2005.
- ◆ N. Ziviani, *Projeto de Algoritmos*, Thomson, 2a. Edição, 2004.
- ◆ T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 2nd Edition, 2001.
- ◆ S. Skiena e M. Revilla, *Programming Challenges: The Programming Contest Training Manual*, Springer-Verlag, 2003.

16