GRAFOS

Representação computacional

Implementação computacional

Eficiência

- A eficiência computacional do algoritmo para resolver problemas em grafos depende
  - das suas características intrínsecas,
  - das estruturas de dados utilizadas para representar o grafo

Representação

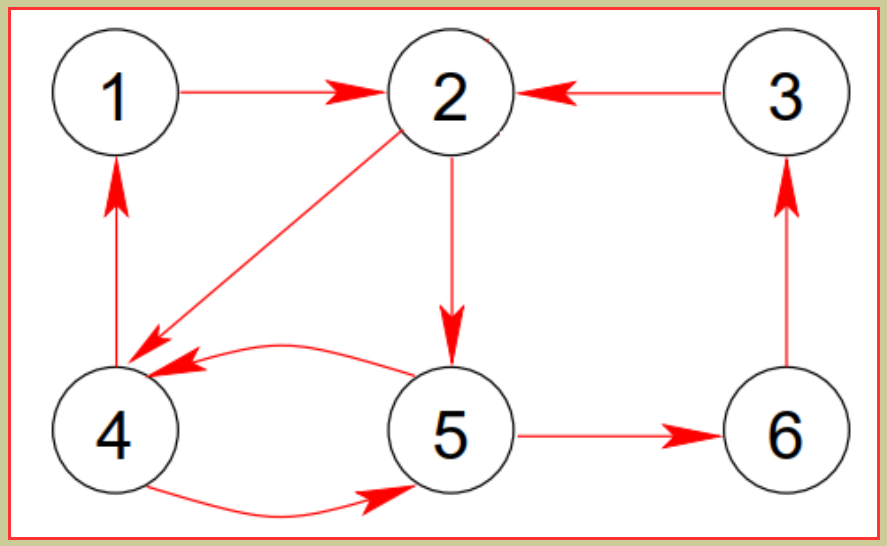
- Informação necessária para representar computacionalmente um grafo é
  - a topologia do grafo (estrutura dos vértices e das arestas),
  - os dados (pesos, custos, ...) associados aos vértices, C_p, e/ou às arestas, C_{ij}
- O esquema utilizado para armazenar a topologia do grafo
  - sugere uma forma de armazenar a informação associada às arestas e aos vértices
- As representações mais comuns de um Grafo são as seguintes
  - matriz de adjacência
  - matriz de incidência
  - listas de adjacências

Matriz de Adjacência

Definição

- Seja G = (V, A) um grafo dirigido com **n** vértices e **m** arestas, a informação do grafo G é armazenada numa matriz M = { $\mathbf{m_{ij}}$ } quadrada de ordem **n**, em que
  - cada linha e cada coluna correspondem a um vértice
  - cada elemento $\mathbf{m_{ij}}$ da matriz M assume um dos seguintes valores:
    - $\mathbf{m_{ij} = 1}$, se $(i, j) \in A$, ou
    - $\mathbf{m_{ij} = 0}$, se $(i, j) \notin A$

Exemplo



	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	0	1	1	0
3	0	1	0	0	0	0
4	1	0	0	0	1	0
5	0	0	0	1	0	1
6	0	0	1	0	0	0

Propriedades

- Nos grafos não dirigidos, pode considerar-se que cada aresta (i, j) corresponde a duas arestas dirigidas: (i, j) e (j, i)
  - neste caso a matriz de adjacência é simétrica
- Representação apropriada para grafos *densos* (com densidade elevada)
- Se o grafo for **pesado** (**rede**) os dados (peso, custo, ...) associados a cada aresta, C_{ij}, pode ser incluído na respectiva posição da matriz, em vez de 1
- O espaço de memória necessário é $O(V^2)$,
  - é independente do número de arestas,
  - torna-se bastante ineficiente quando a densidade do grafo é muito baixa

Propriedades

- A simplicidade desta representação permite a sua utilização para implementar, facilmente, os algoritmos sobre problemas envolvendo grafos, pois pode-se:
  - determinar os parâmetros associado a qualquer aresta (i, j), tomando simplesmente o elemento (i, j) da respectiva matriz, $\mathbf{m_{ij}}$
  - obter facilmente as arestas que saem do vértice i, examinando a linha i:
    - se o j-ésimo elemento dessa linha tem valor 1, então (i, j) é uma aresta do grafo,
  - obter as arestas que chegam ao vértice j, examinando a coluna j:
    - se o i-ésimo elemento dessa coluna tem valor 1, então (i, j) é uma aresta do grafo

Implementação em linguagem C

- Seja G um grafo com **n** vértices e **m** arestas
  - usar um array de 2 dimensões de **inteiros** com **n** linhas e **n** colunas

int M[n][n];

M[i][j] = 0 ⇒ a aresta (i, j) não existe em G

M[i][j] = 1 ⇒ a aresta (i, j) existe em G

- Seja G um grafo pesado (rede) com **n** vértices e **m** arestas
  - usar um array de 2 dimensões de um **tipo** de dados com **n** linhas e **n** colunas

tipo M[n][n]; // **tipo** = tipo de dados associado aos pesos/custos dos arcos

M[i][j] = 0 ⇒ a aresta (i, j) não existe em G

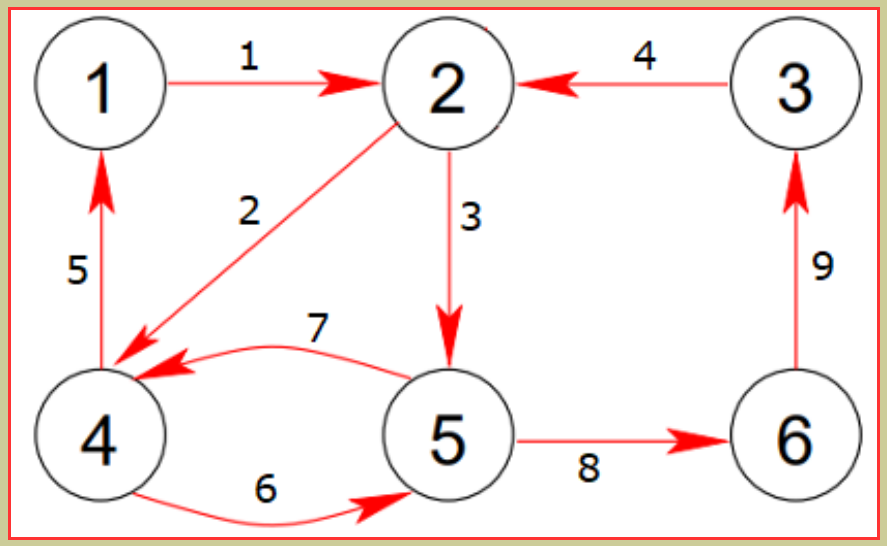
M[i][j] = C_{ij} ⇒ a aresta (i, j) existe em G e tem um peso/custo associado de C_{ij}

Matriz de Incidência

Definição

- Seja G = (V, A) um grafo dirigido com **n** vértices e **m** arestas, a informação do grafo G é armazenada numa matriz M = { $\mathbf{m_{ij}}$ } de ordem **n x m**
  - M é constituída por **n** linhas e **m** colunas, em que
    - cada linha está associada a um vértice de G
    - cada coluna está associada a uma aresta de G
  - cada elemento $\mathbf{m_{ia}}$ assume um dos seguintes valores:
    - $\mathbf{m_{ia} = 1}$, se a aresta **a** tem como origem o vértice **i**
    - $\mathbf{m_{ia} = -1}$, se a aresta **a** tem como destino o vértice **i**
    - $\mathbf{m_{ia} = 0}$, se a aresta **a** não tem como origem nem destino o vértice **i**
- Nos grafos não dirigidos,
  - os valores negativos (−1) são substituídos por positivos (+1), uma vez que todo o vértice é considerado como origem e como destino de uma aresta

Exemplo



	1	2	3	4	5	6	7	8	9
1	1	0	0	0	-1	0	0	0	0
2	-1	1	1	-1	0	0	0	0	0
3	0	0	0	1	0	0	0	0	-1
4	0	-1	0	0	1	1	-1	0	0
5	0	0	-1	0	0	-1	1	1	0
6	0	0	0	0	0	0	0	-1	1

Implementação em linguagem C

- Seja G um grafo com **n** vértices e **m** arestas
  - usar um array de 2 dimensões de **inteiros** com **n** linhas e **m** colunas

int M[n][m];

M[i][k] = 1 ⇒ a aresta k tem como origem o nó i

M[i][k] = -1 ⇒ a aresta k tem como destino o nó i

M[i][k] = 0 ⇒ a aresta k não é adjacente ao nó i (i não é origem nem destino de k)

- Seja G um grafo pesado (rede) com **n** vértices e **m** arestas
  - usar um array de 2 dimensões de um **tipo** de dados com **n** linhas e **m** colunas

tipo M[n][m]; // **tipo** = tipo de dados associado aos pesos/custos dos arcos

M[i][k] = C_{ik} ⇒ a aresta k tem como origem o nó i e tem um peso/custo de C_{ik} (> 0)

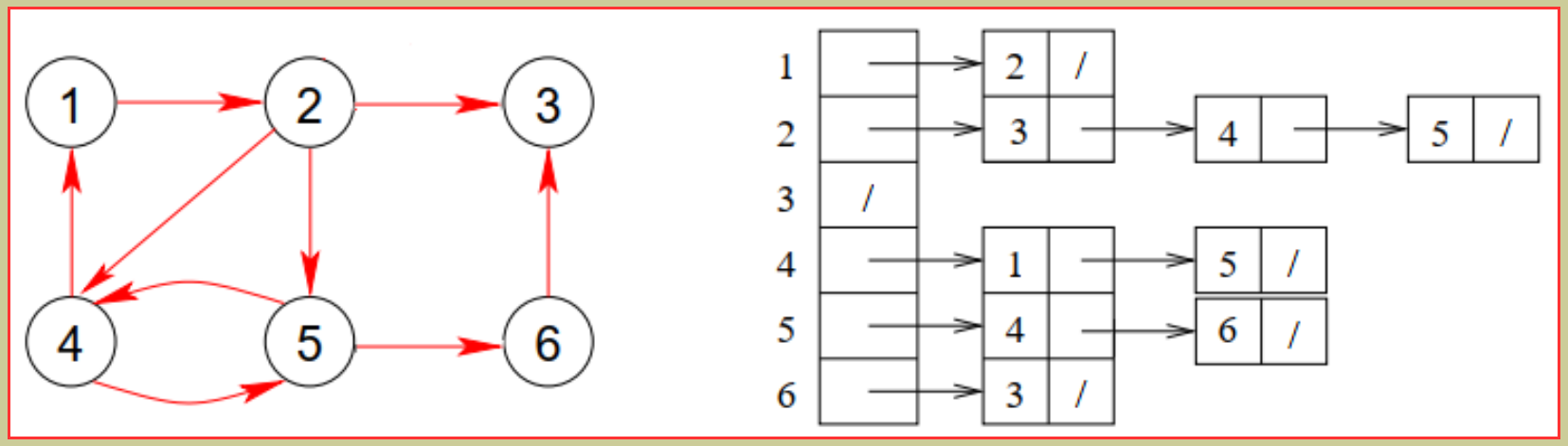
M[i][k] = $-C_{ik}$ ⇒ a aresta k tem como destino o nó i e tem um peso/custo de C_{ik}

M[i][k] = 0 ⇒ a aresta k não é adjacente ao nó i (i não é origem nem destino de k)

Listas de Adjacências de Arestas

Propriedades

- Seja G = (V, A) um grafo dirigido com **n** vértices e **m** arestas, esta representação consiste em armazenar as listas de adjacência de arestas de cada vértice numa lista ligada simples (cada elemento da lista de adjacência é uma lista ligada)
- A lista de adjacência do vértice **i**, **A(i)**, pode ser uma lista ligada com |A(i)| células, em que cada célula corresponde a uma aresta (i, j) do grafo (com todos os vértices adjacentes a i)



Propriedades

- A soma dos comprimentos das listas ligadas é igual ao número de arestas de G
- Se o grafo G for pesado (rede), o peso de cada aresta pode ser incluído no vértice respetivo numa das listas ligadas
- Em grafos não dirigidos, a informação de cada aresta é armazenada em duas células diferentes, pois cada aresta pertence a duas listas de adjacência:
  - como neste caso (i, j) = (j, i), a aresta (i, j) pertence às listas de adjacência
    - do vértice i, $(i, j) \in A(i)$ e
    - do vértice j, $(i, j) \in A(j)$
- Neste caso
  - a representação contém informação redundante e
  - o comprimento total das listas é dobro do número de arestas de G
- Em qualquer dos casos o espaço de memória necessário é O(V+A)

Implementação em linguagem C

- Seja G um grafo com **n** vértices e **m** arestas
  - usar um array de 1 dimensão com **n** elementos, em que o elemento k do array é um array de 1 dimensão com |A[k]| inteiros (|A[k]| = número de arestas com origem em k)

```
int  *A[n];
A[k][j] = p  ⇒  a (j+1)-ésima aresta de G com origem no nó k tem destino no nó p
```

- Seja G um grafo pesado (rede) com **n** vértices e **m** arestas
  - usar um array de 1 dimensão com **n** elementos, em que o elemento k do array é um array de 1 dimensão com |A[k]| elementos de um **tipo** de dados (|A[k]| = número de arestas com origem em k)

```
tipo  *A[n];    // tipo = tipo de dados associado aos pesos/custos dos arcos
A[k][j] = (p, Ckp) ⇒  a (j+1)-ésima aresta de G com origem no nó k tem destino no nó p
e tem associado o valor Ckp
```