Algoritmos e Estruturas de Dados

grafos

2010-2011

Carlos Lisboa Bento

Grafos

conceitos

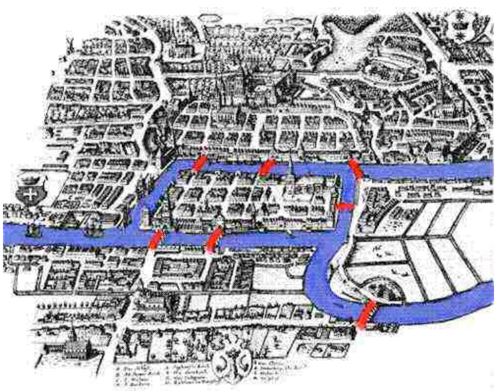
As pontes de Koenigsberg

Bibliografia

Algorithms in C, ROBERT SEDGEWICK

Data Structures and Problem Solving Using JAVA, MARK ALLEN WEISS

Data Structures and Algorithms in JAVA, Adam Drozdek



- Problema:
- Será possível, partindo de um ponto qualquer, atravessar todas as pontes uma única vez e regressar ao mesmo sítio?

Grafos
conceitos

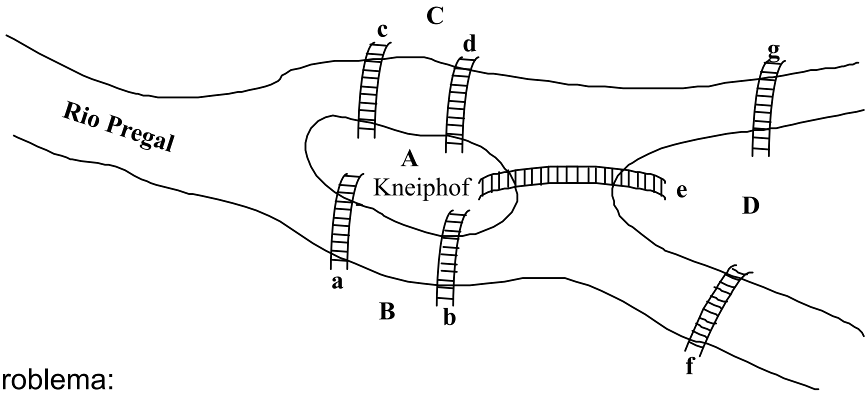
As pontes de Koenigsberg

Bibliografia

Algorithms in C, ROBERT SEDGEWICK

Data Structures and Problem Solving Using JAVA, MARK ALLEN WEISS

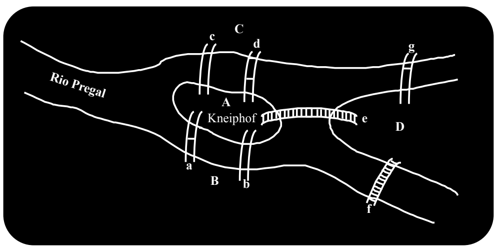
Data Structures and Algorithms in JAVA, Adam Drozdek



- Problema:
- Será possível, partindo de um ponto qualquer, atravessar todas as pontes uma única vez e regressar ao mesmo sítio?

Grafos
conceitos

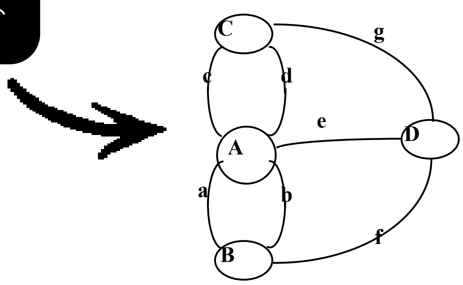
Euler (1707-1783) resolve o problema das pontes de Koenigsberg



Grafo

Terra - nós

Pontes - arcos



<u>Definição</u>:

Grau de um nó é o número de arcos incidentes nesse nó

Euler provou que:

Há um caminho a começar em qualquer nó que percorre todos os arcos uma única vez e que termina no nó inicial se e só se o grau de cada nó é par (Euleriano)

Grafos
conceitos

Para que servem os grafos?

Alguns exemplos:

- Teoria da computação (exp: Maquinas de Turing);
- Optimização de percursos (exp. Caminhos num mapa);
- Análise e planeamento de projectos (exp: diagrama de Gantt);
- Identificação de componentes químicos (exp: interacções entre moléculas);
- Genética (exp: co-relações genéticas);
- Linguística (exp: representação de gramáticas; redes semânticas);
- Ciências sociais (exp: interacções entre agentes);
- Robótica (exp: máquina de estados finitos)

Grafos
conceitos

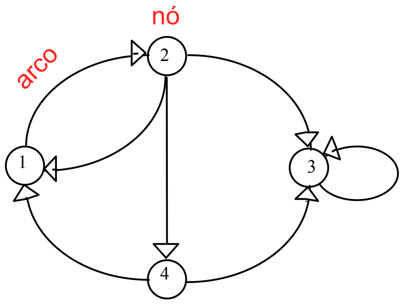
Grafos: terminologia e representações

Um grafo G é constituído por dois conjuntos, N e A:

- N é um conjunto finito, não vazio, de elementos denominados **nós** (ou vértices);
- A é um conjunto de pares de nós (n_i, n_j) denominados **arcos**. Há uma relação binária entre cada elemento do conjunto A (pares de nós).

G = (N, A)

Representação gráfica de G



Exemplo

G = (N, A) em que:

N = {1, 2, 3, 4}

A = {(1,2), (2,1), (2,3), (2,4), (3,3), (4,1), (4,3)}

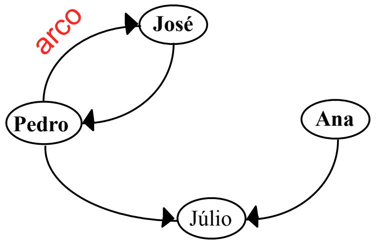
Grafos
conceitos

Grafos dirigidos e grafos não dirigidos

EXEMPLOS

G1 - grafo dirigido

Relação: *A ajuda B*

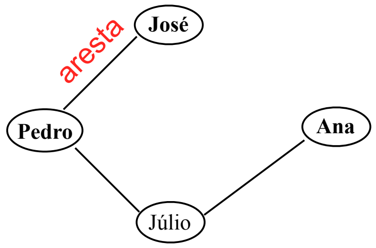


N1 = {José, Ana, Júlio, Pedro}

A1 = {(Pedro,José), (José,Pedro),
(Pedro,Júlio), (Ana,Júlio)}

G2 - grafo não dirigido

Relação: *A é parente de B*

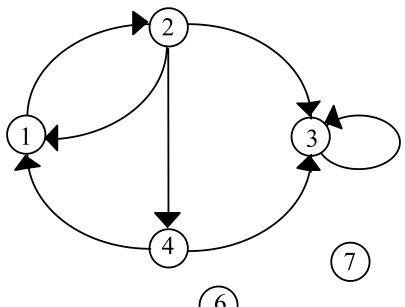


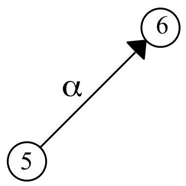
N2 = {José, Ana, Júlio, Pedro}

A2 = {(Pedro,José), (Pedro,Júlio), (Ana,Júlio)}

Grafos
conceitos

Grafos: terminologia e representação (cont.)



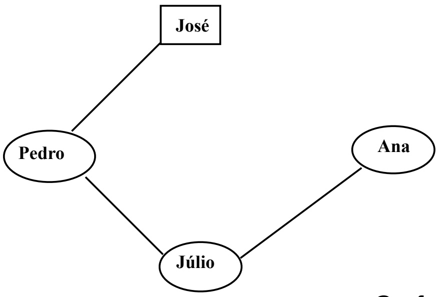


- Um nó pode não ter nenhum arco associado (caso do nó 7).
- Nós ligados por arcos são ditos <u>adjacentes</u>: o nó 2 é adjacente de 1, 3 e 4.
- Um arco que liga nós adjacentes diz-se <u>incidente</u> a esses nós.

Grafos dirigidos:

- Um arco α é <u>incidente ao</u> nó n se chega a esse nó. O arco α é incidente ao nó 6.
- Um arco a é <u>incidente do</u> nó n se parte desse nó. O arco α é incidente do nó 5.
- O arco α tem a cauda no nó 5 e a cabeça no nó 6. Notar que α representa-se por (5,6).

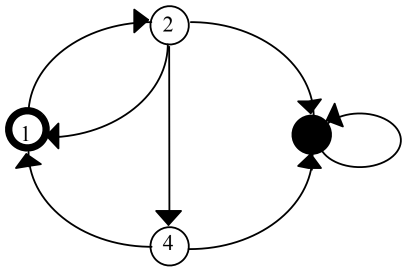
Grafos
conceitos



• Grau de um nó: número de arestas incidentes nesse nó.

Ex: o grau do nó "Júlio" é 2

Grafos dirigidos

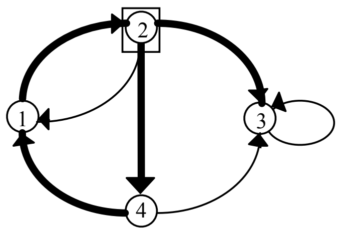


- Grau interno de um nó: número de arcos que têm esse nó como cabeça.
- Grau externo de um nó: número de arcos que têm esse nó como cauda.

Ex: Nó 1
- Grau interno = 2
- Grau externo = 1

Grafos
conceitos

Grafos: terminologia e representação (cont.)



Caminho:
sequência de um ou mais arcos em que o segundo nó de cada arco coincide com o primeiro do arco seguinte:

{(a,n1), (n1,n2), ... (ni, b)} – caminho de a para b

se a = b temos um ciclo

• É possível a partir do nó 1 atingir o nó 3 percorrendo os arcos (1,2) e (2,3). Estes arcos formam um caminho de 1 para 3.

• O comprimento do caminho é igual ao número de arcos existentes no caminho

• Se o nó de partida coincide com o de chegada temos um ciclo. Existe um ciclo unindo os nós 1, 2 e 4.

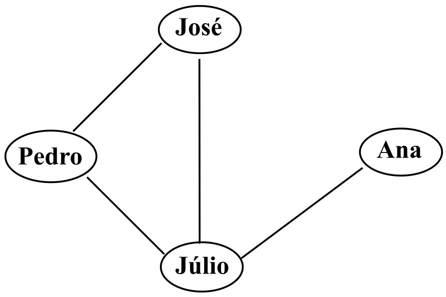
• Um ciclo de um único arco chama-se um laço. Caso do nó 3.

• Grafos cíclicos: os que contêm um ciclo. Caso contrário são acíclicos

Grafos

conceitos

Grafos: terminologia e representação (cont.)



• No caso de grafos não dirigidos usa-se o termo circuito em vez de ciclo

• Existe um circuito unindo os nós Pedro, José e Júlio

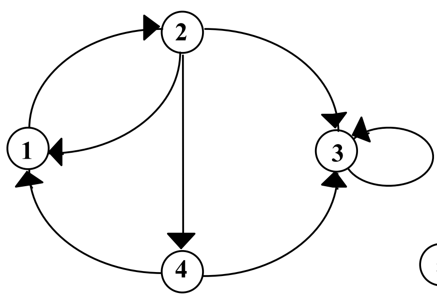
Grafos

conceitos

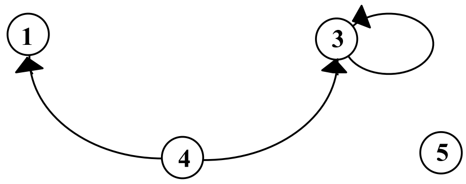
Grafos: terminologia e representação (cont.)

Subgrafo: Subconjunto de nós de um dado grafo juntamente com todos os arcos cujas duas extermidades são nós desse subconjunto

G



G' é um subgrafo de G



N' = {1, 4, 3, 5}

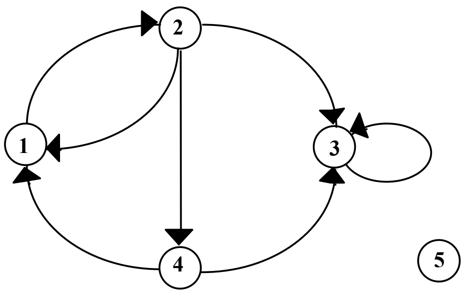
A' = {(4,1), (4,3), (3,3)}

Grafos
conceitos

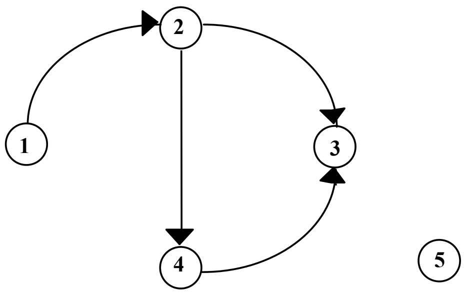
Grafos: terminologia e representação (cont.)

Grafo parcial: grafo constituído pelos mesmos nós do grafo original mas em que se considera apenas um subconjunto dos arcos

G



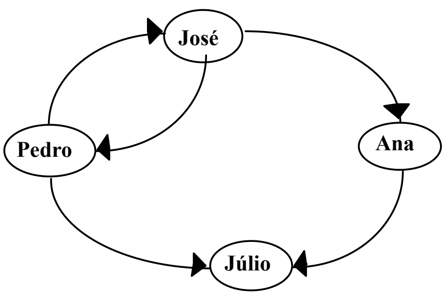
G' é um grafo parcial de G



Grafos
conceitos

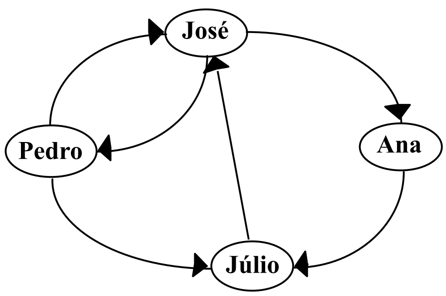
Grafos: terminologia e representação (cont.)

Grafo conexo: se tem pelo menos um nó a partir do qual existem caminhos para todos os restantes



Ex: a partir de Pedro é possível atingir qualquer dos outros nós.

Grafo fortemente conexo: se de todos os nós é possível atingir todos os demais



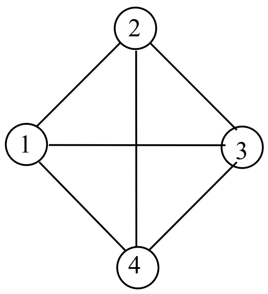
Grafos
conceitos

Grafo completo: se tem o número máximo de arcos

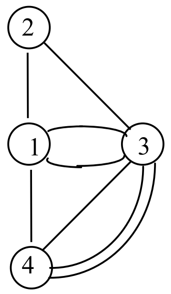
Multigrafo: se tem múltiplas ocorrências de um mesmo arco

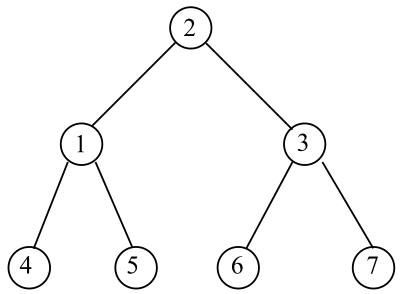
Uma árvore é um caso particular de um grafo, mas nem todos os grafos são árvores

Grafo completo



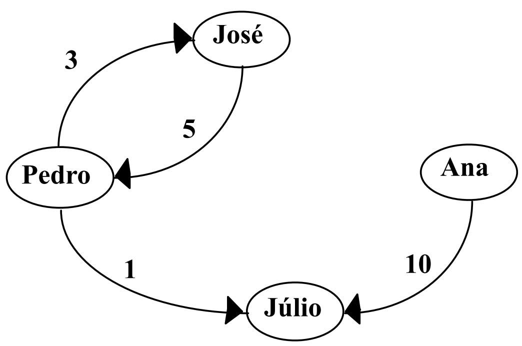
Multigrafo





Grafos
conceitos

Grafos ponderados



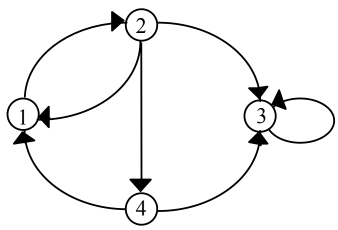
Um grafo pode ter números associados a cada arco. Nesse caso o grafo chama-se ponderado e o número junto a cada arco designa-se por peso do arco

Grafos
representação

Grafos
representação

Representação de grafos

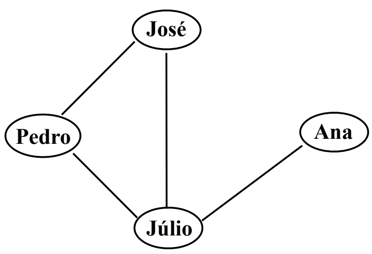
Matriz de adjacência: dado um grafo G = (N, A) de n nós a sua matriz de adjacência MA é uma matriz n x n com a propriedade de M(i, j) = 1 se existe um arco do nó n_i para o nó n_j. Caso contrário M(i, j) = 0.



j

i

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



	José	Pedro	Júlio	Ana
José	0	1	1	0
Pedro	1	0	1	0
Júlio	1	1	0	1
Ana	0	0	1	0

Matriz

simétrica

Grafos
representação

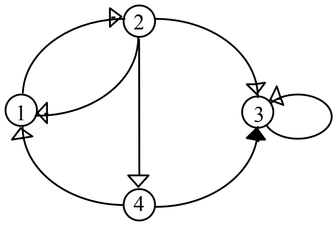
Vantagens da matriz de adjacência

Torna fácil verificar se dois nós são adjacentes (estão ligados por um arco)

Torna fácil juntar ou retirar arcos (ou arestas) do grafo

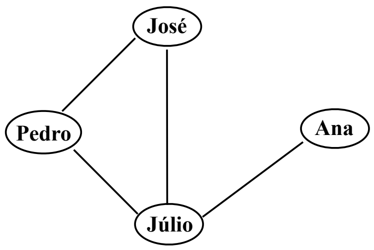
Torna fácil determinar o grau de um grafo

Grafo



Matriz de adjacência

i \ j	1	2	3	4	Grau externo
1	0	1	0	0	*1*
2	1	0	1	1	*3*
3	0	0	1	0	*1*
4	1	0	1	0	*2*
Grau interno	*2*	*1*	*3*	*1*	



	José	Pedro	Júlio	Ana	Grau
José	0	1	1	0	*2*
Pedro	1	0	1	0	*2*
Júlio	1	1	0	1	*3*
Ana	0	0	1	0	*1*

No caso de não haver informação associada aos nós nem aos arcos o grafo pode ser completamente descrito pela sua matriz de adjacência

Grafos
representação

Caminhos e transitividade

Num grafo descrito pela sua matriz de adjacência adj

adj[i, k] == 1 && adj[k, j] == 1 **Verdadeira se e só se existe um arco de i para k e um arco de k para j**

⇕

Existe um caminho de comprimento 2 do nó i para o nó j passando pelo nó k

Consideremos a expressão:

$$\underbrace{\textbf{(adj[i,0] == 1 \&\& adj[0, j] == 1) || ... || (adj[i, max-1] == 1 \&\& adj[max-1, j] == 1)}}$$

Verdadeira se e só se há pelo menos um caminho de comprimento 2 entre os nós i e j

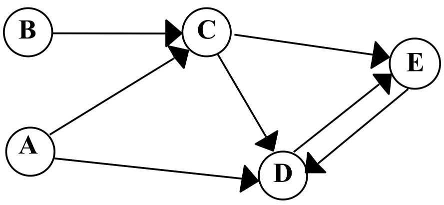
Matriz de caminhos de comprimento 2

adj $[i, j]_2$ — Matriz em que cada elemento tem o valor 1 se e só se existe um caminho de comprimento 2 entre i e j

Grafos

representação

Matriz de caminhos de comprimento 2 — adj_2



	A	B	C	D	E
A	0	0	1	1	0
B	0	0	1	0	0
C	0	0	0	1	1
D	0	0	0	0	1
E	0	0	0	1	0

adj

MATRIZ DE ADJACÊNCIA

adj_2 obtem-se através do produto lógico de adj por ela própria

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \wedge \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

adj_2

Matriz de caminhos de comprimento 2

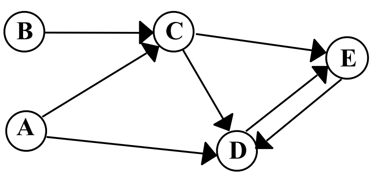
Nota: produto lógico de duas matrizes obtem-se multiplicando as duas matrizes mas substituindo a multiplicação por *AND* e a soma por *OR*

Grafos

representação

adj_3 — Matriz de caminhos de comprimento 3: cada elemento $adj_3[i, j]$ é verdadeiro se e só se houver pelo menos um caminho de comprimento 3 entre i e j

$adj_3 = adj \wedge adj_2$



adj $\wedge$ **adj_3** = **adj_4**

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 0 & 1 & 1 & 0 \\ B & 0 & 0 & 1 & 0 & 0 \\ C & 0 & 0 & 0 & 1 & 1 \\ D & 0 & 0 & 0 & 0 & 1 \\ E & 0 & 0 & 0 & 1 & 0 \end{array} \wedge \begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 0 & 0 & 1 & 1 \\ B & 0 & 0 & 0 & 1 & 1 \\ C & 0 & 0 & 0 & 1 & 1 \\ D & 0 & 0 & 0 & 0 & 1 \\ E & 0 & 0 & 0 & 1 & 0 \end{array} = \begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 0 & 0 & 1 & 1 \\ B & 0 & 0 & 0 & 1 & 1 \\ C & 0 & 0 & 0 & 1 & 1 \\ D & 0 & 0 & 0 & 1 & 0 \\ E & 0 & 0 & 0 & 0 & 1 \end{array}$$

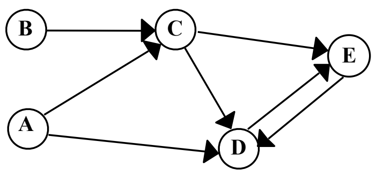
Genericamente: $adj_n = adj \wedge adj_{n-1}$

Grafos
representação

Matriz dos caminho de comprimento <= n

- **Existe um caminho de comprimento menor ou igual a 3 se a seguinte expressão for verdadeira:**

$$\mathbf{adj \vee adj_2 \vee adj_3}$$



	A	B	C	D	E
A	0	0	1	1	1
B	0	0	1	1	1
C	0	0	0	1	1
D	0	0	0	1	1
E	0	0	0	1	1

Matriz de caminhos de comprimento <= 3

Grafos
representação

QUESTÃO:
Dado um grafo, pretende-se saber se entre dois quaisquer nós existe um caminho, qualquer que seja o comprimento desse caminho

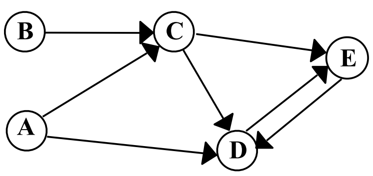
SOLUÇÃO:

Para um grafo de n nós:

$$\mathbf{Caminho[i, j] = adj\ [i, j] \vee adj_2\ [i, j] \vee \ldots \vee adj_n\ [i, j]}$$

Esta matriz chama-se fecho transitivo

Caminho = adj || adj$_2$ || adj$_3$ || adj$_4$ || adj$_5$



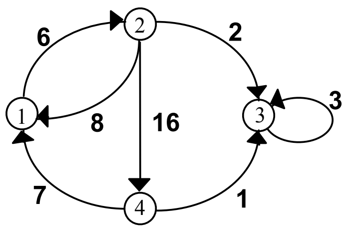
	A	B	C	D	E
A	0	0	1	1	1
B	0	0	1	1	1
C	0	0	0	1	1
D	0	0	0	1	1
E	0	0	0	1	1

Se Caminho[i, j] = 1 – existe um caminho entre o no i e o nó j

Grafos
representação

Representação de grafos ponderados

Matriz de adjacência: dado um grafo G = (N, A) de n nós a sua matriz de adjacência MA é uma matriz n x n com a propriedade de M(i, j) = x se existe um arco do nó n_i para o nó n_j sendo x o peso associado ao arco do nó n_i para n_j. Caso contrário M(i, j) = ∞ .

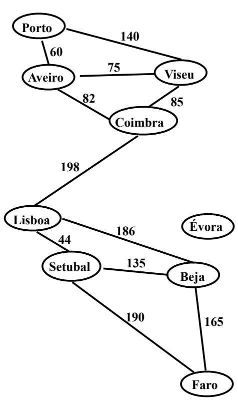


	1	2	3	4
1	∞	6	∞	∞
2	8	∞	2	16
3	∞	∞	3	∞
4	7	∞	1	∞

Grafos
representação

Um problema da representação de grafos através da matriz de adjacência

Grafo



Matriz de adjacência

	Porto	Aveiro	Viseu	Coimbra	Lisboa	Évora	Setúbal	Beja	Faro
Porto	0	60	140						
Aveiro	60	0	75	82					
Viseu	140	75	0	85					
Coimbra		82	85	0	198				
Lisboa				198	0		44	186	
Évora									
Setúbal					44		0	135	190
Beja					186		135	0	165
Faro							190	165	0

Responder a questões do tipo:

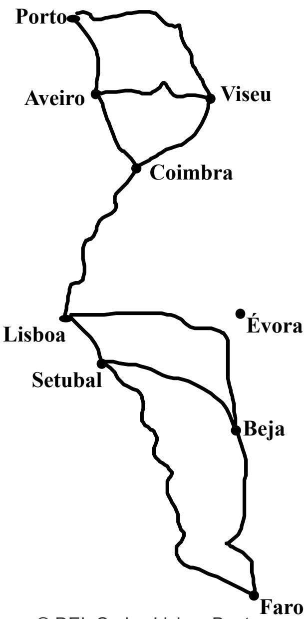
- Quantas arestas existem neste grafo?
- O grafo é conexo?

algorítmos de Ordem n^2

i.e., é necessário examinar todos os elementos da matriz, incluindo os nulos

Grafos
representação

Outro problema da representação de grafos através da matriz de adjacência

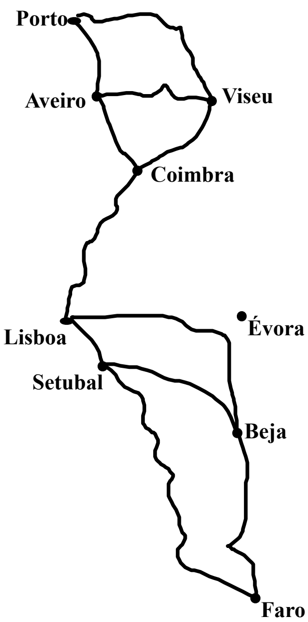


- É necessário conhecer previamente o número de nós existentes no grafo.
- O que fazer no caso de ser necessário aumentar o número de nós?

Novos nós:
Braga
Tomar
Sines
Lagos
Tavira

Grafos
representação

Ainda outro probl. da represent. de grafos através da matriz de adjacência



	Porto	Aveiro	Viseu	Coimbra	Lisboa	Évora	Setúbal	Beja	Faro	Braga	Tomar	Sines	Lagos	Tavira
Porto	0	60	140											
Aveiro	60	0	75	82										
Viseu	140	75	0	85										
Coimbra		82	85	0	198									
Lisboa				198	0		44	186						
Évora														
Setúbal					44		0	135	190					
Beja					186		135	0	165					
Faro							190	165	0					
Braga														
Tomar														
Sines														
Lagos														
Tavira														

Há tendência para a matriz de adjacência ficar esparsa

Grafos
representação

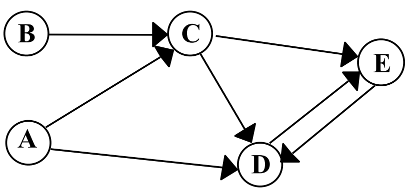
Resumindo… nas matrizes de adjacências temos:

- Ineficiente na alteração dinâmica do número de nós - na prática o número de nós tem de ser conhecido previamente (ou, pelo menos, majorado);
- Os algorítmos de manipulação do grafo podem ter complexidade inadequada (Ordem N^2);
- A matriz de adjacência pode ser esparsa implicando elevados custos de armazenamento.

Grafos
representação

Listas de adjacência (alternativa às matrizes de adjacência)

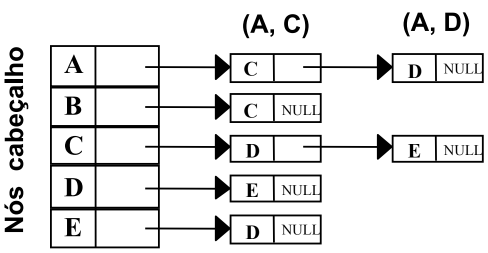
Ex: Grafo



Matriz de adjacência

	A	B	C	D	E
A	0	0	1	1	0
B	0	0	1	0	0
C	0	0	0	1	1
D	0	0	0	0	1
E	0	0	0	1	0

Listas de adjacência

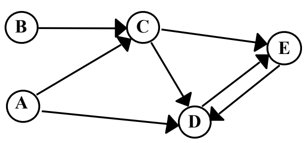


- Para cada nó do grafo constroi-se uma lista ligada (lista de adjacência);
- A lista de adjacência correspondente a um dado nó contém referência a todos os nós adjacentes a esse nó ;
- Cada elemento da lista de adjacência representa, pois, um arco entre o nó cabeçalho e o nó correspondente a esse elemento.

Grafos
representação

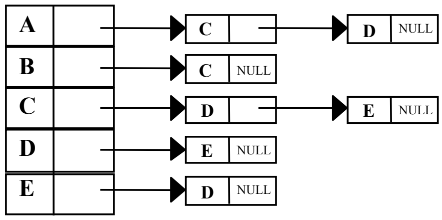
Matrizes e listas de adjacência

Grafo



Listas de adjacência

Estrutura Dinâmica



É visível a equivalência entra as duas estruturas

Matriz de adjacência

Estrutura Estática

	A	B	C	D	E
A	0	0	1	1	0
B	0	0	1	0	0
C	0	0	0	1	1
D	0	0	0	0	1
E	0	0	0	1	0

Omitindo os "0"

	A	B	C	D	E
A			1	1	
B			1		
C				1	1
D					1
E				1	

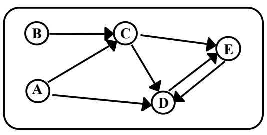
Dado outra forma

A	C	D	
B	C		
C		D	E
D			E
E		D	

Grafos
representação

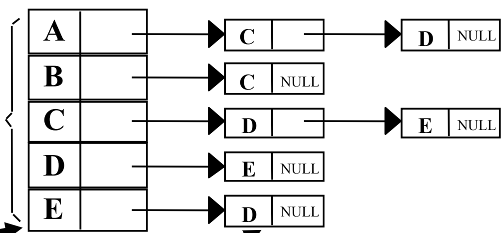
Listas de adjacência em detalhe

Grafo



Os nós cabeçalho estão num *array*, o que permite acesso rápido à lista de adjacência de qualquer nó

Listas de adjacência



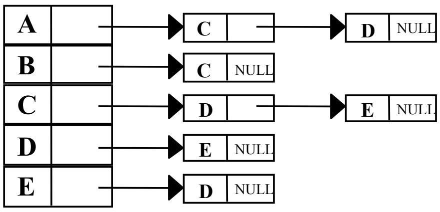
Cada lista tem um nó cabeçalho

Cada elemento da lista tem, pelo menos, dois campos: *identificador de arco* (através do nó destino do arco) e *ponteiro para seguinte*. Sendo necessário, pode haver campos para informação associada ao arco correspondente (grafos ponderados)

Grafos
representação

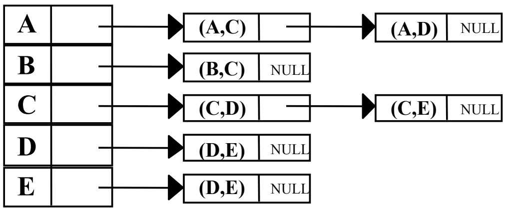
Duas convenções para os elementos da lista de adjacência

Cada elemento de uma lista de adjacência representa um arco



Representa-se apenas o nó adjacente ao nó cabeçalho => arco entre o nó cabeçalho e esse nó

Estas representações diferem apenas na simbologia. Ambas representam o mesmo grafo.

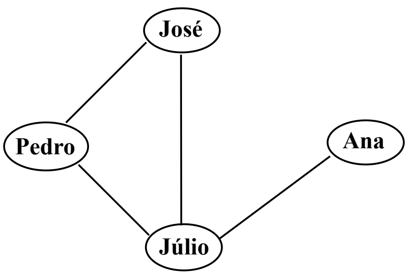


O arco é representado explicitamente

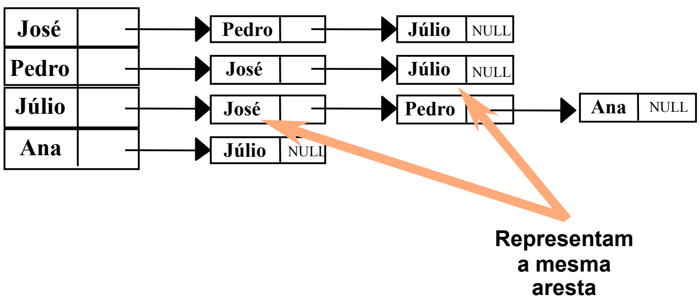
Grafos
representação

Listas de adjacência de grafos não dirigidos

Grafo



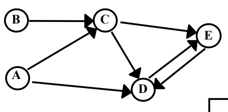
Listas de adjacência



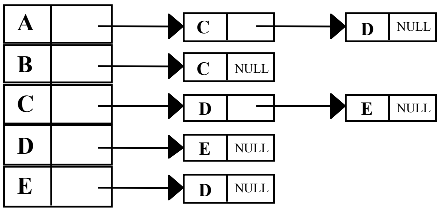
Cada aresta (v_i, v_j) é representada por duas entradas, uma na lista correspondente ao nó v_i e outra na lista do nó v_j

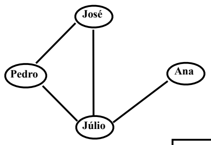
Grafos
representação

Cálculo do grau de um nó

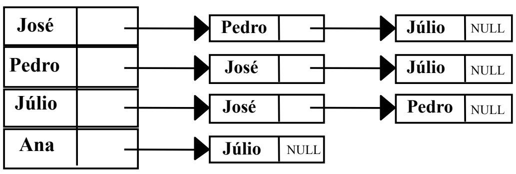


Grafo dirigido





Grafo não dirigido



- Grau externo de um nó - nº de elementos na lista de adjacência desse nó.
- Grau interno - não pode ser calculado directamente: é necessário percorrer todas as listas de adjacência.

- Grau de um nó - nº de elementos na lista de adjacência desse nó.

Grafos
representação

Vantagens das listas de adjacência

- Reduzem o tempo necessário para efectuar certas operações sobre os grafos.

Ex: Determinar quantos arcos tem um grafo de <u>n</u> nós e <u>a</u> arcos é uma operação de Ordem (n^2) através da matriz de adjacência e de Ordem (<u>n</u> + <u>a</u>) através das listas de adjacência

- Permitem economizar espaço para o caso de grafos cujas matrizes de adjacência são esparsas.

Grafos
representação

Desvantagens das listas de adjacência

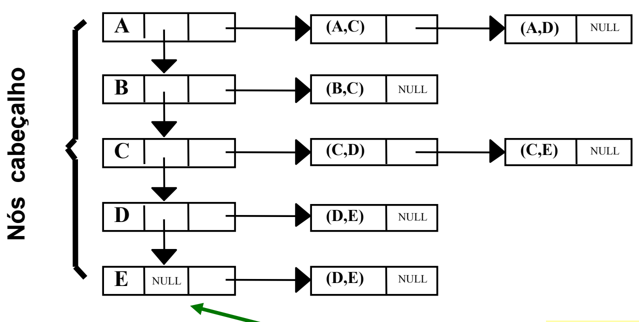
• Certas operações básicas são mais complexas do que no caso da matriz de adjacência.

> Ex: ligar dois nós através de um arco ou remover o arco entre dois nós são operações em listas, enquanto que na matriz de adjacência consistem em alterar apenas um elemento da matriz

Grafos
representação

Nós cabeçalho numa estrutura dinâmica

Listas de adjacência totalmente baseadas em estruturas dinâminas

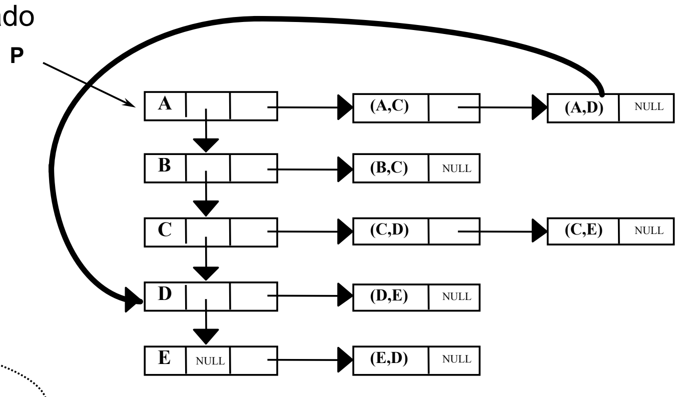


Cada nó cabeçalho contém:
- Campo de informação;
- Ponteiro para nó cabeçalho seguinte;

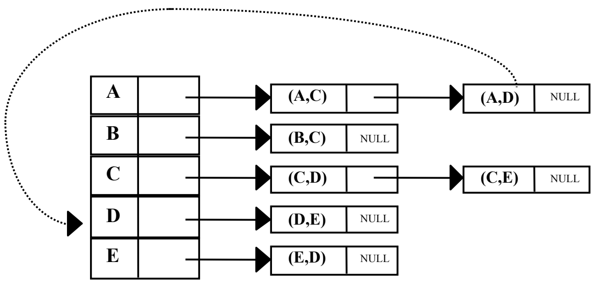
Grafos
representação

Um problema...

percorrer o grafo pode ser demorado



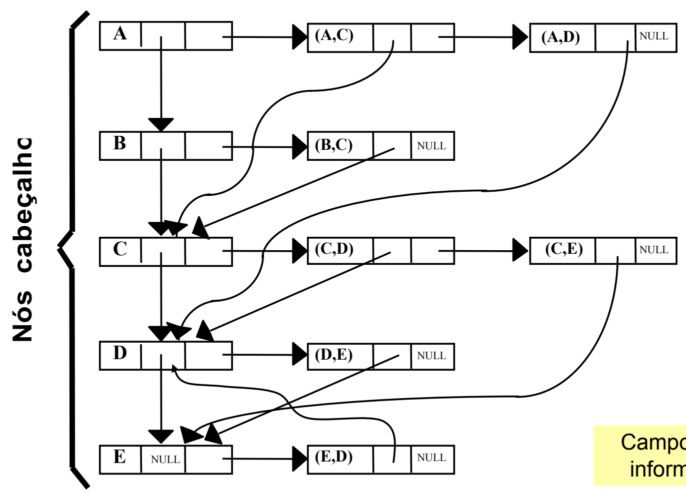
O acesso ao nó destino do arco (A,D) é através de uma lista ligada



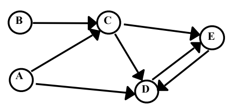
Acesso directo ao nó destino do arco (A,D)

Grafos
representação

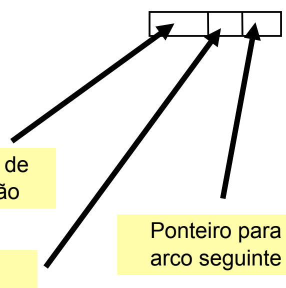
Uma solução... representação ligada do grafo



Grafo



Cada nó da lista de adjacência contém



Campo(s) de informação

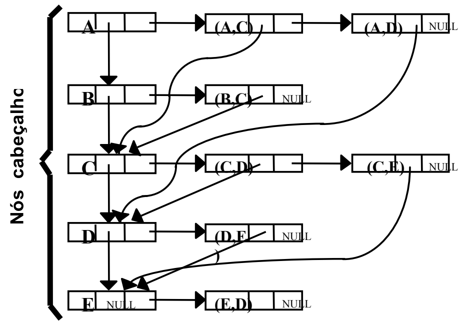
Ponteiro para arco seguinte

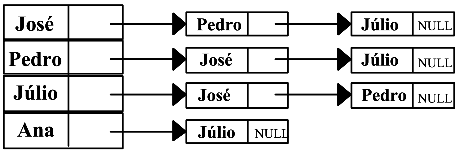
Ponteiro para nó cabeçalho correspondente ao outro extremo do arco

Grafos

representação

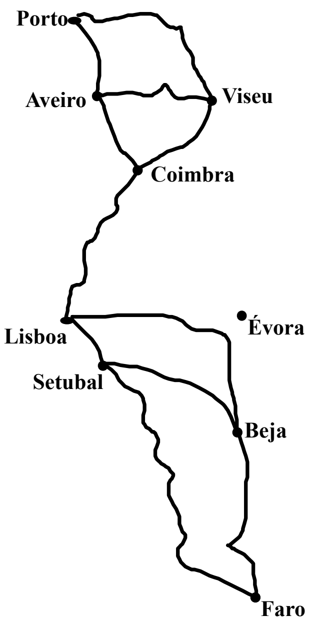
	Porto	Aveiro	Viseu	Coimbra	Lisboa	Évora	Setúbal	Beja	Faro	Braga	Tomar	Sines	Lagos	Tavira
Porto	0	60	140											
Aveiro	60	0	75	82										
Viseu	140	75	0	85										
Coimbra		82	85	0	198									
Lisboa				198	0		44	186						
Évora														
Setúbal					44		0	135	190					
Beja					186		135	0	165					
Faro							190	165	0					
Braga														
Tomar														
Sines														
Lagos														
Tavira														

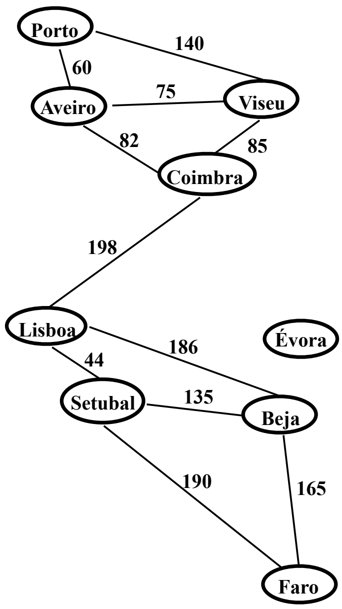




Grafos

representação





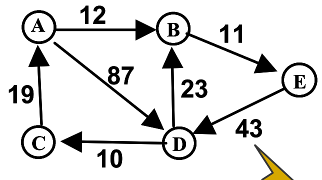
Algumas perguntas

- Há algum caminho entre um dado nó A e outro nó B?
- Qual o comprimento de um dado caminho entre dois nós A e B?
- Se há vários caminhos entre os dois nós A e B qual é o mais curto?

Grafos

implementação

Grafo (representação visual)



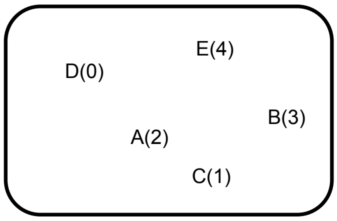


Tabela de Dispersão

D	C	10
A	B	12
D	B	23
A	D	87
E	D	43
B	E	11
C	A	19

Dados de Entrada

	dist	*antec*	*nome*	*listaAdj*
0	66	4	D	
1	76	0	C	
2	0	-1	A	
3	12	2	B	
4	23	3	E	

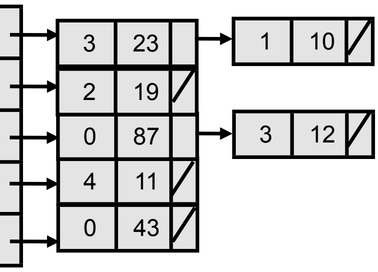
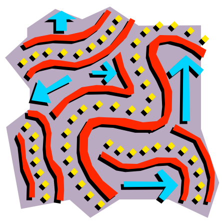


Tabela do Grafo

Grafos

procura do caminho mais curto – grafos não ponderados

Caminho mais curto num grafo não ponderado

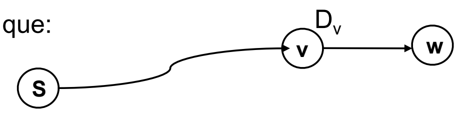


Problema

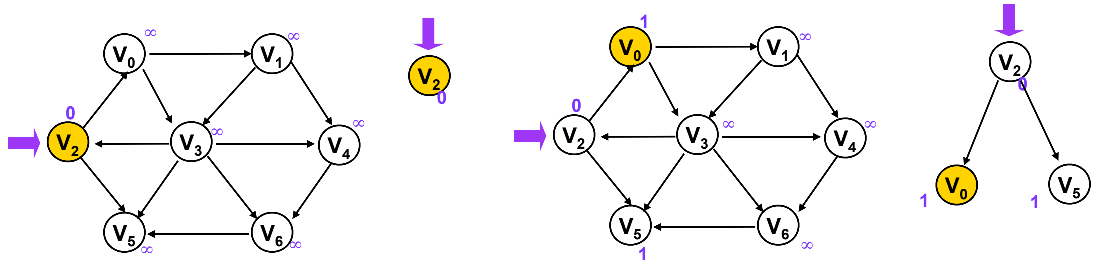
Determinar o caminho mais curto (definido pelo número de arcos) do nó S para todos os nós.

Procura primeiro em largura (por níveis)

Explora os nós por camadas em que:

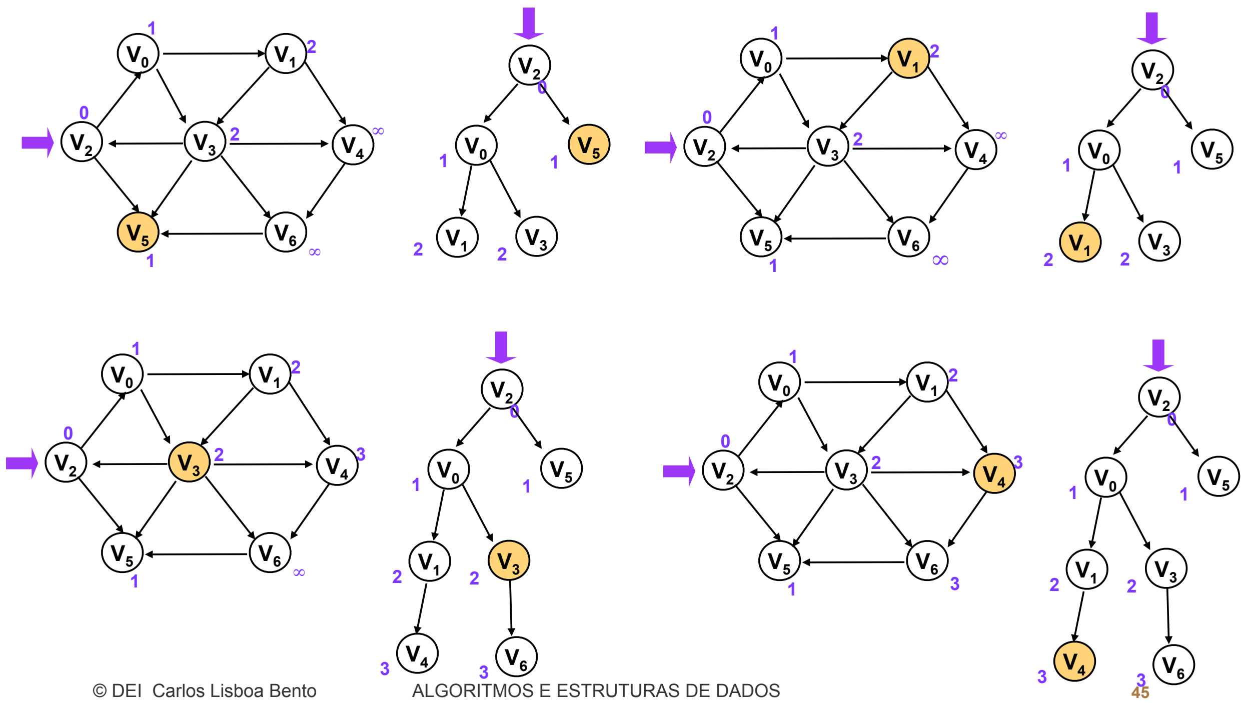


$D_w = D_v + 1$ se $D_w = \infty$



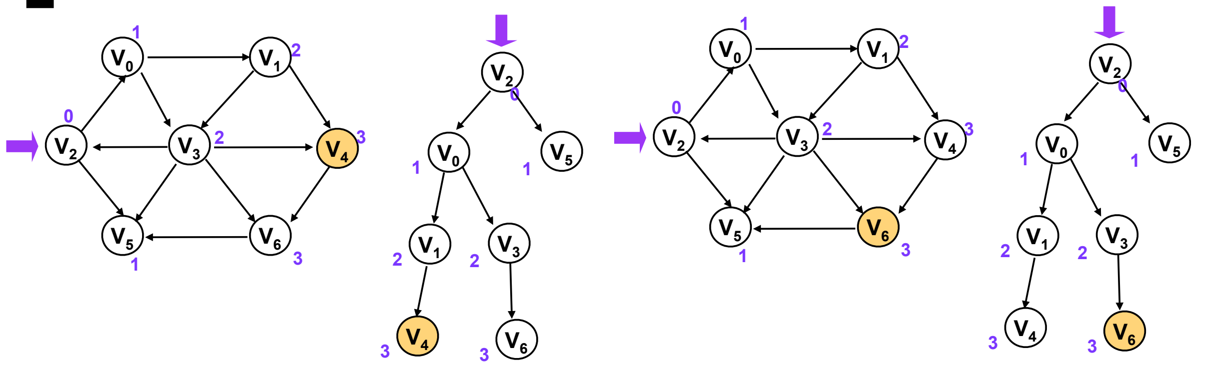
Grafos

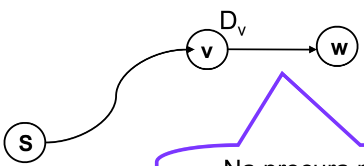
procura do caminho mais curto – grafos não ponderados



Grafos

procura do caminho mais curto – grafos não ponderados





$D_w = D_v + 1$ se $D_w = \infty$

Na procura primeiro em largura
também fica claro que v é o nó que
precede w no percurso mais curto

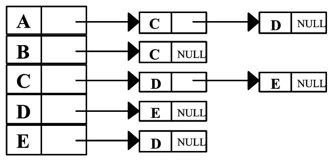
Grafos

procura do caminho mais curto – grafos não ponderados

Caminho mais curto num grafo não ponderado (estruturas de dados)

Considerando que o grafo é representado por listas de adjacências as principais operações que vamos ter vão ser

1. Procura do nó que vai ser explorado



2. Acesso a todos os nós adjacentes ao nó que vai ser explorado

Grafos

procura do caminho mais curto – grafos não ponderados

Caminho mais curto num grafo não ponderado (estruturas de dados)

Colocar cada nó cujo valor D foi actualizado numa fila de espera.

O próximo nó a ser explorado é o nó na cabeça da fila de espera.

No início a fila está vazia e o primeiro nó que vai receber é o nó S.

Como **cada nó vai entrar e sair uma única vez na fila** de espera temos que este processo tem complexidade **O(N)**.

O custo de procurar o caminho mais curto usando procura primeiro em largura vai ser dominado pela procura na lista de adjacências ou seja **O(A)**, ou seja linear relativamente ao tamanho do grafo.

Grafos

procura do caminho mais curto – grafos ponderados

Dependendo do número de vezes que as etiquetas com o valor de distância são actualizadas temos:

- Métodos de fixação de etiquetas
  cada vértice uma vez etiquetado não volta a ser processado (só aplicável a grafos com pesos positivos)

- Métodos de revisão de etiquetas
  cada vértice pode ser re-etiquetado quando for necessário (aplicável a grafos com pesos negativos e com ciclos só com pesos positivos)

Grafos

procura do caminho mais curto – grafos ponderados

Tanto os ***métodos fixação*** como de ***correcção de etiquetas*** podem ser genericamente descritos da seguinte forma (Gallo and Pallotino 86):

```
genericShortestPathAlgorithm( weighted simple digraph, vertex first)
   for all vertices v
      currDist(v) = ∞;
   currDist(first) = 0;
   initialize toBeChecked;

   while toBeChecked is not empty
      v = a vertex in toBeChecked;
      remove v from toBeChecked;

      for all nodes u adjacent to v
         if currDist(u) > currDist(v)+weigth(edge(v u))
            currDist(u) = currDist(v)+weigth(edge(v u));
            predecessor(u) = v;
            add u to toBeChecked if it is not there;
```

QUESTÕES EM ABERTO:: estrutura de toBeChecked

QUESTÕES EM ABERTO:: ordem de atribuição a v de valores de toBeChecked

Grafos

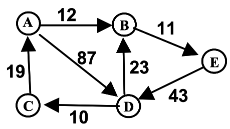
procura do caminho mais curto – grafos ponderados

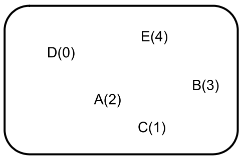
Cada nó etiquetado pela distância corrente e pela indicação do seu antecessor:

label(v) = (*currDist*(v) , *predecessor*(v))

Recordar… …

Grafo (representação visual)





Dicionário

D	C	10
A	B	12
D	B	23
A	D	87
E	D	43
B	E	11
C	A	19

Dados de Entrada

	dist	*antec*	*nome*	*listaAdj*
0	66	4	D	
1	76	0	C	
2	0	-1	A	
3	12	2	B	
4	23	3	E	

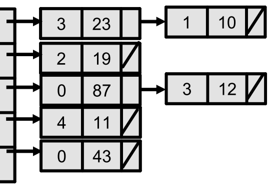


Tabela do Grafo

Grafos

algoritmo de dijkstra

- São explorados caminhos $p_1,\ldots,p_n$ com origem em v;
- a cada momento o caminho mais curto p_i entre $p_1,\ldots,p_n$ é escolhido;
- é acrescentada uma aresta *a* a p_i;
- se p_i+a deixou de ser o caminho mais curto então é abandonado e o novo caminho mais curto é retomado;
- cada vértice só é explorado uma vez;
- após todos os vértices terem sido explorados o processo termina.

```
genericShortestPathAlgorithm( weighted simple digraph, vertex first)
    for all vertices v
        currDist(v) = ∞;
    currDist(first) = 0;
    initialize toBeChecked;

    while toBeChecked is not empty
        v = a vertex in toBeChecked;
        remove v from toBeChecked;

        for all nodes u adjacent to v
            if currDist(u) > currDist(v)+weigth(edge(v u))
                currDist(u) = currDist(v)+weigth(edge(v u));
                predecessor(u) = v;
                add u to toBeChecked if it is not there;
```

Grafos

algoritmo de dijkstra

genericShortestPathAlgorithm(*weighted simple* digraph, *vertex* first)
  for *all vertices* v
    currDist(v) = ∞;
  currDist(first) = 0;
  initialize toBeChecked;

  while toBeChecked *is not empty*
    v = a vertex in toBeChecked;
    remove v *from* toBeChecked;

    for *all nodes* u *adjacent to* v
      if *currDist*(u) > *currDist*(v)+*weigth*(*edge*(v u))
        currDist(u) = *currDist*(v)+*weigth*(edge(v u));
        predecessor(u) = v;
        add u *to* toBeChecked *if it is not there*;

DijkstraAlgorithm(*weighted simple* digraph, *vertex* first)
  for *all vertices* v
    currDist(v) = ∞;
  currDist(first) = 0;
  toBeChecked = *all vertices*;

  while toBeChecked *is not empty*
    v = *a vertex in* toBeChecked ***with minimal currDist*(v);**
    remove v *from* toBeChecked;

    for *all nodes* u *adjacent to* v ***and in* toBeChecked**
      if *currDist*(u) > *currDist*(v)+*weigth*(*edge*(v u))
        currDist(u) = *currDist*(v)+*weigth*(edge(v u));
        predecessor(u) = v;
        ~~***add u to toBeChecked if it is not there;***~~

Grafos

procura do caminho mais curto – grafos ponderados

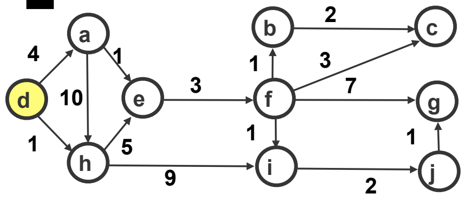
Dependendo do número de vezes que as etiquetas com o valor de distância são actualizadas temos:

- Métodos de fixação de etiquetas
  cada vértice uma vez etiquetado não volta a ser processado
  (só aplicável a grafos com pesos positivos)

- Métodos de revisão de etiquetas
  cada vértice pode ser re-etiquetado quando for necessário
  (aplicável a grafos com pesos negativos e com ciclos só com pesos positivos)

Grafos

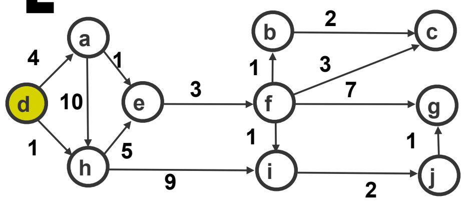
algoritmo de dijkstra



Iteração:	init	1	2	3	4	5	6	7	8	9	10
Nó activo		d	h	a	e	f	b	i	c	j	g
a	∞	4	4								
b	∞	∞	∞	∞	∞	9					
c	∞	∞	∞	∞	∞	11	11	11			
d	0										
e	∞	∞	6	5							
f	∞	∞	∞	∞	8						
g	∞	∞	∞	∞	∞	15	15	15	15	12	
h	∞	1									
i	∞	∞	10	10	10	9	9				
j	∞	∞	∞	∞	∞	∞	∞	11	11		

Grafos

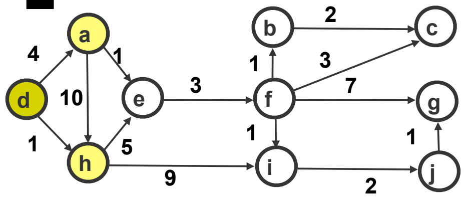
algoritmo de dijkstra



Iteração:	init	1	2	3	4	5	6	7	8	9	10
Nó activo		d	h	a	e	f	b	i	c	j	g
a	∞	4	4								
b	∞	∞	∞	∞	∞	9					
c	∞	∞	∞	∞	∞	11	11	11			
d	0										
e	∞	∞	6	5							
f	∞	∞	∞	∞	8						
g	∞	∞	∞	∞	∞	15	15	15	15	12	
h	∞	1									
i	∞	∞	10	10	10	9	9				
j	∞	∞	∞	∞	∞	∞	∞	11	11		

Grafos

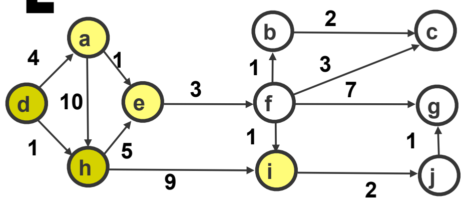
algoritmo de dijkstra



Iteração: Nó activo	init	1 d	2 h	3 a	4 e	5 f	6 b	7 i	8 c	9 j	10 g
a	∞	4	4								
b	∞	∞	∞	∞	∞	9					
c	∞	∞	∞	∞	∞	11	11	11			
d	0										
e	∞	∞	6	5							
f	∞	∞	∞	∞	8						
g	∞	∞	∞	∞	∞	15	15	15	15	12	
h	∞	1									
i	∞	∞	10	10	10	9	9				
j	∞	∞	∞	∞	∞	∞	∞	11	11		

Grafos

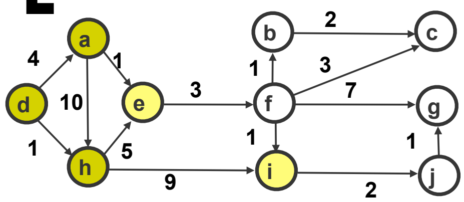
algoritmo de dijkstra



Iteração: Nó activo	init	1 d	2 h	3 a	4 e	5 f	6 b	7 i	8 c	9 j	10 g
a	∞	4	4								
b	∞	∞	∞	∞	∞	9					
c	∞	∞	∞	∞	∞	11	11	11			
d	0										
e	∞	∞	6	5							
f	∞	∞	∞	∞	8						
g	∞	∞	∞	∞	∞	15	15	15	15	12	
h	∞	1									
i	∞	∞	10	10	10	9	9				
j	∞	∞	∞	∞	∞	∞	∞	11	11		

Grafos

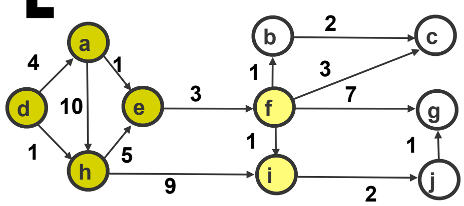
algoritmo de dijkstra



Iteração: Nó activo	init	1 d	2 h	3 a	4 e	5 f	6 b	7 i	8 c	9 j	10 g
a	∞	4	4								
b	∞	∞	∞	∞	∞	9					
c	∞	∞	∞	∞	∞	11	11	11			
d	0										
e	∞	∞	6	5							
f	∞	∞	∞	∞	8						
g	∞	∞	∞	∞	∞	15	15	15	15	12	
h	∞	1									
i	∞	∞	10	10	10	9	9				
j	∞	∞	∞	∞	∞	∞	∞	11	11		

Grafos

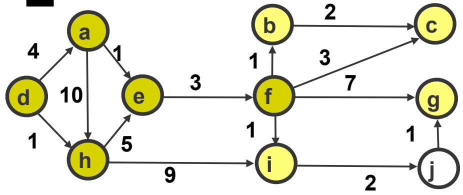
algoritmo de dijkstra



Iteração: Nó activo	init	1 d	2 h	3 a	4 e	5 f	6 b	7 i	8 c	9 j	10 g
a	∞	4	4								
b	∞	∞	∞	∞	∞	9					
c	∞	∞	∞	∞	∞	11	11	11			
d	0										
e	∞	∞	6	5							
f	∞	∞	∞	∞	8						
g	∞	∞	∞	∞	∞	15	15	15	15	12	
h	∞	1									
i	∞	∞	10	10	10	9	9				
j	∞	∞	∞	∞	∞	∞	∞	11	11		

Grafos

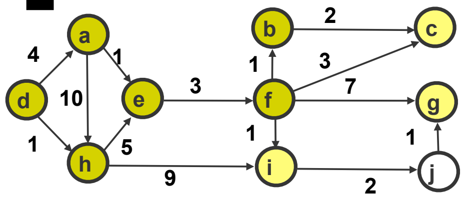
algoritmo de dijkstra



Iteração: Nó activo	init	1 d	2 h	3 a	4 e	5 f	6 b	7 i	8 c	9 j	10 g
a	∞	**4**	**4**								
b	∞	∞	∞	∞	∞	**9**					
c	∞	∞	∞	∞	∞	**11**	11	11			
d	**0**										
e	∞	∞	**6**	**5**							
f	∞	∞	∞	∞	**8**						
g	∞	∞	∞	∞	∞	**15**	15	15	15	12	
h	∞	**1**									
i	∞	∞	**10**	**10**	**10**	**9**	9				
j	∞	∞	∞	∞	∞	∞	∞	11	11		

Grafos

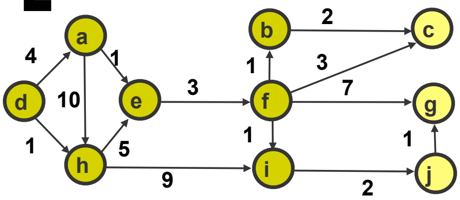
algoritmo de dijkstra



Iteração: Nó activo	init	1 d	2 h	3 a	4 e	5 f	6 b	7 i	8 c	9 j	10 g
a	∞	**4**	**4**								
b	∞	∞	∞	∞	∞	**9**					
c	∞	∞	∞	∞	∞	**11**	**11**	11			
d	**0**										
e	∞	∞	**6**	**5**							
f	∞	∞	∞	∞	**8**						
g	∞	∞	∞	∞	∞	**15**	**15**	15	15	12	
h	∞	**1**									
i	∞	∞	**10**	**10**	**10**	**9**	**9**				
j	∞	∞	∞	∞	∞	∞	∞	11	11		

Grafos

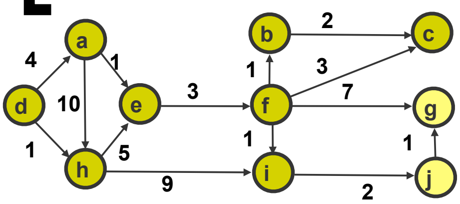
algoritmo de dijkstra



Iteração: Nó activo	init	1 d	2 h	3 a	4 e	5 f	6 b	7 i	8 c	9 j	10 g
a	∞	**4**	**4**								
b	∞	∞	∞	∞	∞	**9**					
c	∞	∞	∞	∞	∞	**11**	**11**	**11**			
d	**0**										
e	∞	∞	**6**	**5**							
f	∞	∞	∞	∞	**8**						
g	∞	∞	∞	∞	∞	**15**	**15**	**15**	**15**	**12**	
h	∞	**1**									
i	∞	∞	**10**	**10**	**10**	**9**	**9**				
j	∞	∞	∞	∞	∞	∞	∞	**11**	**11**		

Grafos

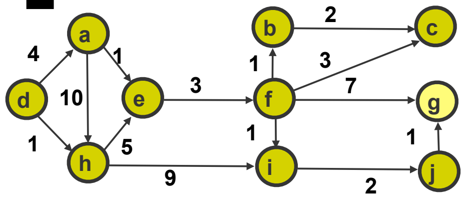
algoritmo de dijkstra



Iteração: Nó activo	init	1 d	2 h	3 a	4 e	5 f	6 b	7 i	8 c	9 j	10 g
a	∞	**4**	**4**								
b	∞	∞	∞	∞	∞	**9**					
c	∞	∞	∞	∞	∞	**11**	**11**	**11**			
d	**0**										
e	∞	∞	**6**	**5**							
f	∞	∞	∞	∞	**8**						
g	∞	∞	∞	∞	∞	**15**	**15**	**15**	**15**	**12**	
h	∞	**1**									
i	∞	∞	**10**	**10**	**10**	**9**	**9**				
j	∞	∞	∞	∞	∞	∞	∞	**11**	**11**		

Grafos

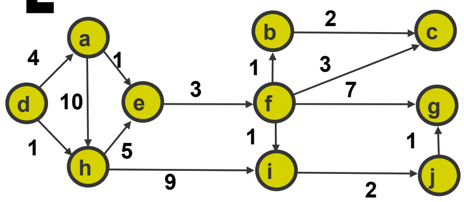
algoritmo de dijkstra



Iteração: Nó activo	init	1 d	2 h	3 a	4 e	5 f	6 b	7 i	8 c	9 j	10 g
a	∞	**4**	**4**								
b	∞	∞	∞	∞	∞	**9**					
c	∞	∞	∞	∞	∞	**11**	**11**	**11**			
d	**0**										
e	∞	∞	**6**	**5**							
f	∞	∞	∞	∞	**8**						
g	∞	∞	∞	∞	∞	**15**	**15**	**15**	**15**	**12**	
h	∞	**1**									
i	∞	∞	**10**	**10**	**10**	**9**	**9**				
j	∞	∞	∞	∞	∞	∞	∞	**11**	**11**		

Grafos

algoritmo de dijkstra



Iteração: Nó activo	init	1 d	2 h	3 a	4 e	5 f	6 b	7 i	8 c	9 j	10 g
a	∞	**4**	**4**								
b	∞	∞	∞	∞	∞	**9**					
c	∞	∞	∞	∞	∞	**11**	**11**	**11**			
d	**0**										
e	∞	∞	**6**	**5**							
f	∞	∞	∞	∞	**8**						
g	∞	∞	∞	∞	∞	**15**	**15**	**15**	**15**	**12**	
h	∞	**1**									
i	∞	∞	**10**	**10**	**10**	**9**	**9**				
j	∞	∞	∞	∞	∞	∞	∞	**11**	**11**		

Grafos
algoritmo de dijkstra

QUESTÕES

1. Como é calculado D_w ?

2. Como é determinado o próximo nó a ser explorado ?

2. Criar uma fila de prioridades.

Sempre que um nó tem o seu valor D_w reduzido coloca nó/D_W na fila de prioridades.

Para seleccionar um novo nó para explorar remove o item com menor valor D_w (distância à origem) na lista de prioridades. Analisa os vizinhos directos até que um nó ainda não explorado seja encontrado. Esse é o novo nó a explorar.

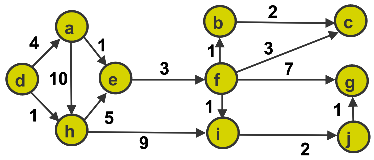
Grafos
algoritmo de dijkstra – Demos na Web

http://carbon.cudenver.edu/~hgreenbe/sessions/dijkstra/DijkstraApplet.html
(não disponível em 9 de Abril de 2007)

http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/DijkstraApp.shtml?demo3

Grafos

algoritmo de dijkstra



Iteração: Nó activo	init	1 d	2 h	3 a	4 e	5 f	6 b	7 i	8 c	9 j	10 g
a	∞	4	4								
b	∞	∞	∞	∞	∞	9					
c	∞	∞	∞	∞	∞	11	11	11			
d	0										
e	∞	∞	6	5							
f	∞	∞	∞	∞	8						
g	∞	∞	∞	∞	∞	15	15	15	15	12	
h	∞	1									
i	∞	∞	10	10	10	9	9				
j	∞	∞	∞	∞	∞	∞	∞	11	11		

DESV.: o algoritmo de dijkstra falha quando temos pesos negativos !!

Grafos

procura do caminho mais curto – grafos ponderados c/ p. neg.

<u>Problema</u>

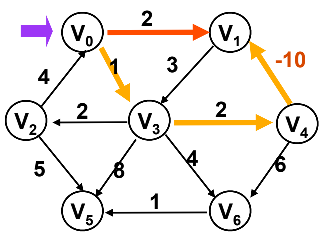
Determinar o caminho mais curto (definido pelo custo total associado aos arcos que compõem o caminho) do nó S para todos os nós.

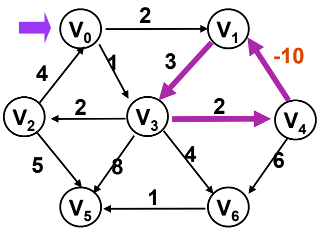
Considerar que os arcos podem ter custos negativos.

Consideremos o seguinte grafo:

Duas situações criticas:

1. Quando um vértice v é processado pode haver um outro vértice u (não processado) a partir do qual voltando a v tenhamos um caminho de custo mais baixo.





2. Existência de ciclos de custo negativo

Grafos

procura do caminho mais curto – grafos ponderados c/ p. neg.

```
genericShortestPathAlgorithm( weighted simple digraph, vertex first)
    for all vertices v
        currDist(v) = ∞;
    currDist(first) = 0;
    initialize toBeChecked;

    while toBeChecked is not empty
        v = a vertex in toBeChecked;
        remove v from toBeChecked;

        for all nodes u adjacent to v
            if currDist(u) > currDist(v)+weigth(edge(v u))
                currDist(u) = currDist(v)+weigth(edge(v u));
                predecessor(u) = v;
                add u to toBeChecked if it is not there;
```

```
labelCorrectingAlgorithm( weighted simple digraph,
                                        vertex first)
    for all vertices v
        currDist(v) = ∞;
    currDist(first) = 0;
    toBeChecked = { first };

    while toBeChecked is not empty
        v = a vertex in toBeChecked;
        remove v from toBeChecked;

        for all nodes u adjacent to v
            if currDist(u) > currDist(v)+weigth(edge(v u))
                currDist(u) = currDist(v)+weigth(edge(v u));
                predecessor(u) = v;
                add u to toBeChecked if it is not there;
```

aplicável a grafos com pesos negativos e com ciclos com pesos positivos

Grafos

procura do caminho mais curto – grafos ponderados c/ p. neg.

Observações:

1. **Quando D_w tem o seu valor alterado este nó deve ser revisitado no futuro, logo recolocá-lo na fila de nós a explorar com $D_w = D_v + c_{v,w}$**
2. **Quando um nó v é explorado pela i-ésima vez, significa que existem pelo menos i caminhos diferentes entre o nó origem e o nó v. Assim não havendo ciclos negativos, o número máximo possível de visitas será igual ao número de arestas, pois caso fosse superior, teria que existir um ciclo.**

Detecção de ciclos negativos!!!

Grafos

procura do caminho mais curto – grafos ponderados c/ p. neg.

```
/**
 * Run shortest path algorithm;
 * Negative edge weights are allowed.
 * Return false if negative cycle is detected
 */
private boolean negative( int startNode )
{
    int v, w;
    Queue q = new QueueAr( );
    int cvw;

    clearData( );
    table[ startNode ].dist = 0;
    q.enqueue( new Integer( startNode ) );
    table[ startNode ].scratch++;

    try
    {
        while( !q.isEmpty( ) )
        {
            v = ( (Integer) q.dequeue( ) ).intValue( );

            if( table[ v ].scratch++ > 2 * numVertices )
                return false;

            ListItr p = new LinkedListItr( table[ v ].adj );
```

Quando um nó entra da fila de espera scratch é incrementado

Quando um nó sai da fila de espera scratch é incrementado

Scratch/2 indica o número de vezes que o nó entrou e saiu da fila de espera

Grafos

procura do caminho mais curto – grafos ponderados c/ p. neg.

```
for( ; p.isInList( ); p.advance( ) )
        {
            w = ( (Edge) p.retrieve( ) ).dest;
            cvw = ( (Edge) p.retrieve( ) ).cost;
            if( table[ w ].dist > table[ v ].dist + cvw )
            {
                table[ w ].dist = table[ v ].dist + cvw;
                table[ w ].prev = v;

                // Enqueue only if not already on queue
                if( table[ w ].scratch++ % 2 == 0 )
                    q.enqueue( new Integer( w ) );
                else
                    table[ w ].scratch++;  // In effect, adds 2
            }
        }
    }
}
catch( Underflow e ) { }   // Cannot happen

return true;
}
```

Quando um nó entra para a fila de espera scratch é incrementado

Quando um nó que deve entrar para a fila de espera já lá está, o scratch é incrementado duas vezes continuando assim com valor par (nó na fila de espera). Desta forma temos uma saida e entrada do nó para a fila só em termos lógicos

Grafos

procura do caminho mais curto – grafos ponderados c/ p. neg.

Observações:

1. **Quando D_w tem o seu valor alterado este nó deve ser revisitado no futuro, logo recolocá-lo na fila de nós a explorar com $D_w = D_v + c_{v,w}$**
2. **Quando um nó v é explorado pela i-ésima vez, significa que existem pelo menos i caminhos diferentes entre o nó origem e o nó v. Assim não havendo ciclos negativos, o número máximo possível de visitas será igual ao número de arestas, pois caso fosse superior, teria que existir um ciclo.**

•Complexidade: cada nó vai ser retirado da fila de nós pelo menos uma vez. Existem N nós. Cada nós pode ser retirado no máximo A vezes, sendo A o número de arestas.

•O algoritmo vai assim ter complexidade O(A*N) com A igual ao número de arestas e N igual ao número de nós.

• Se um vértice é retirado da fila de vértices a explorar mais do que N vezes, significa que temos um ciclo de custo negativo.

Grafos

procura do caminho mais curto – grafos acíclicos

<u>Problema</u>

Determinar o caminho mais curto (definido pelo custo total associados aos arcos que compõem o caminho) do nó S para todos os nós.

Considerar que o grafo não tem ciclos.

Ordenamento Topológico

Num ordenamento topológico os nós de um grafo acíclico estão ordenados de tal forma que se existir um caminho de u para v, u precede v no grafo.

Ex.: grafo usado para representar as precedências das disciplinas de um curso. Um arco (v, w) representa que a disciplina v deve estar concluida antes de frequentar a disciplina w.

Um grafo pode ter vários ordenamentos topológicos

Grafos

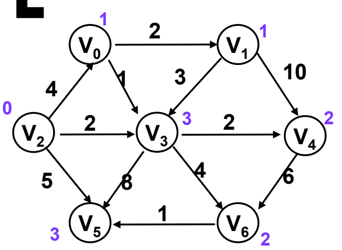
procura do caminho mais curto – grafos acíclicos

Ordenamento topológico

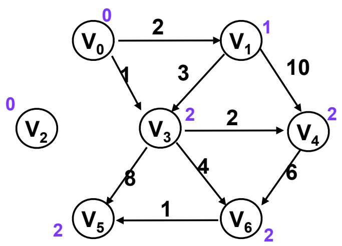
1. Calcular o grau interno de todos os nós
2. Enquanto existirem nós no grafo com grau interno 0 Faz
    1. Seleccionar um nó v cujo grau interno seja 0
    2. Processar o nó v
    3. Remover o nó v do grafo juntamente com os arcos incidentes desse nó
    4. Actualizar o grau interno dos nós adjacentes a v
5. Fim Enquanto

Grafos

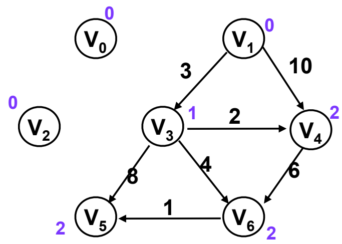
procura do caminho mais curto – grafos acíclicos



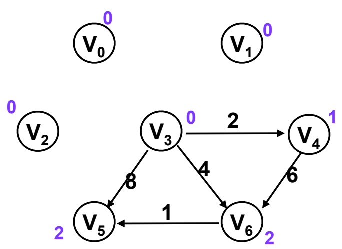
Ordem topolog.= [ ]



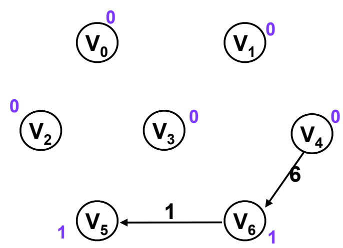
Ordem topolog.= [V_2]



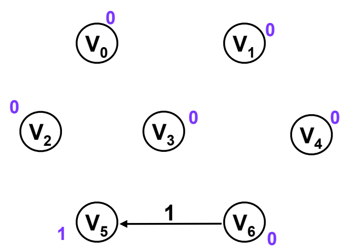
Ordem topolog.= [V_2 V_0]



Ordem topolog.= [V_2 V_0 V_1]



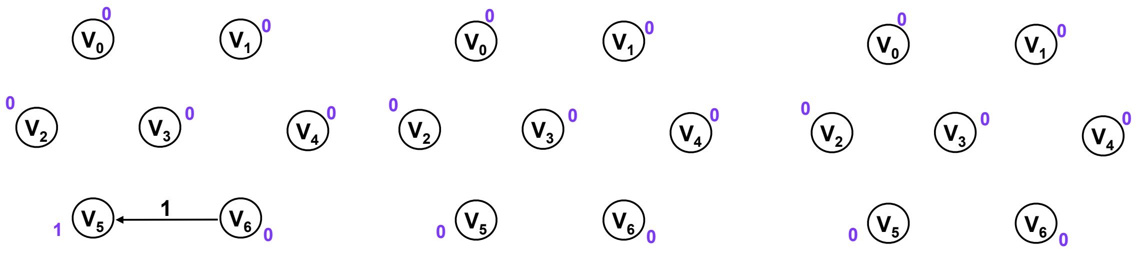
Ordem topolog.= [V_2 V_0 V_1 V_3]



Ordem topolog.= [V_2 V_0 V_1 V_3 V_4]

Grafos

procura do caminho mais curto – grafos acíclicos



Ordem topolog.= [V_2 V_0 V_1 V_3 V_4]   **Ordem topolog.= [V_2 V_0 V_1 V_3 V_4 V_6]**   **Ordem topolog.= [V_2 V_0 V_1 V_3 V_4 V_6 V_5]**

IDEIA: Para encontrar o caminho mais curto num grafo acíclico explorar os nós do grafo em ordenamento topológico.

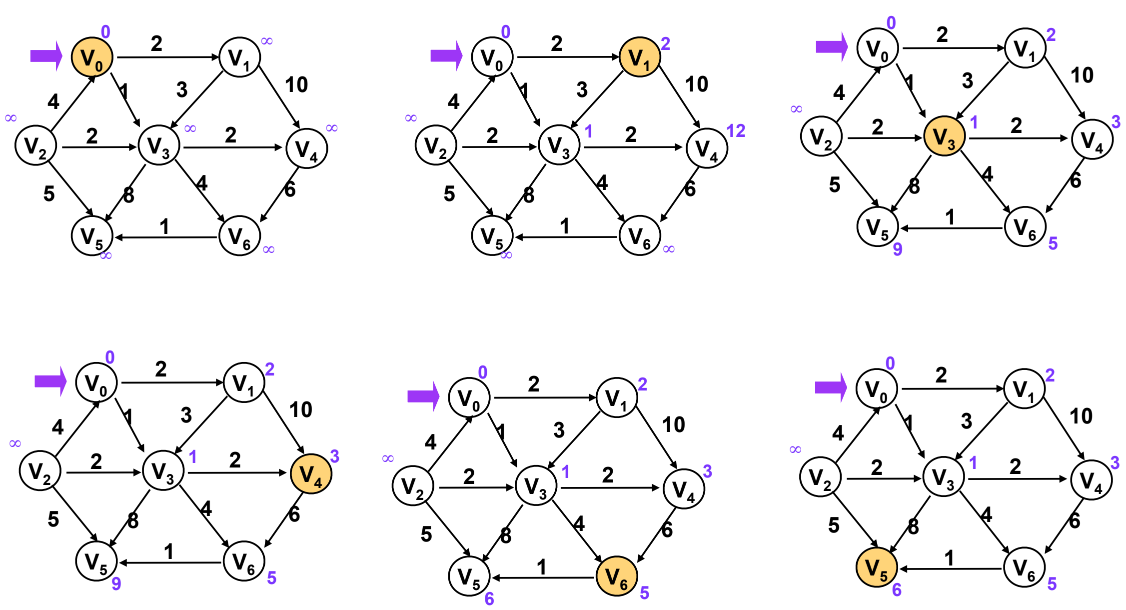
++ Deixamos de necessitar de uma <u>fila de prioridades</u> e passamos ter uma <u>fila de espera</u>.

++ O algoritmo tem <u>complexidade linear</u> e admite arcos com custo negativo.

Grafos

procura do caminho mais curto – grafos acíclicos

Ordem topológica= [V_2 V_0 V_1 V_3 V_4 V_6 V_5]



Grafos

- Os **grafos** são estruturas de dados **versáteis** adequadas para: **análise de circuitos electricos**; **métodos de detecção de erros em computadores**; **optimização de percursos**; **análise e planeamento de projectos**; **identificação de componentes químicos**; **genética**; **linguística**; **ciências sociais**;**robótica**.
- Um **grafo** pode ser representado por **G=(N,A)**, com N conj. de nós e A conj. de ligações (designados por arcos ou arestas).
- Os **grafos** podem ser **dirigidos**, **não dirigidos**, **ponderados**, **não ponderados**.
- Alguns conceitos associados aos **grafos** são: **grau**, **grau interno**, **grau externo**, **caminho**, **ciclo**, **laço**, **circuito**, **subgrafo**, **grafo parcial**, **grafo conexo**, **fortemente conexo**, **completo**, **multigrafo**.
- Um **grafo** pode ser **representado** por uma **matriz de adjacências** ou por **listas de adjacências**.
- Na representação por **listas de adjacências** os **cabeçalhos** podem estar organizados num ***array***, numa **lista ligada** ou numa ***hash table***.

Grafos

- Os **grafos** são estruturas aplicadas a um grande número de problemas nomeadamente em circunstâncias em que se pretende encontrar o **caminho mais curto entre dois nós**.
- Para **grafos não ponderados,** o caminho mais curto pode ser encontrado fazendo presquisa **primeiro em largura** com **complexidade linear**.
- Para **grafos ponderados com valores positivos,** no **algoritmo de Dijkstra** com recurso a uma **fila de prioridades,** com tempo de computação **ligeiramente superior** ao algoritmo para grafos não ponderados.
- Para **grafos ponderados com valores negativos** recorremos a algoritmos que comportam **correcção de etiquetas** (vs algoritmos de fixação de etiquetas, de que o Dijkstra é um exemplo) com tempo de computação significativamente superior – **O (a*n)**.
- Para **grafos acíclicos com pesos positivos e negativos** a complexidade é linear se recorrermos a **ordenamento topográfico**.

Grafos

… end ;-)