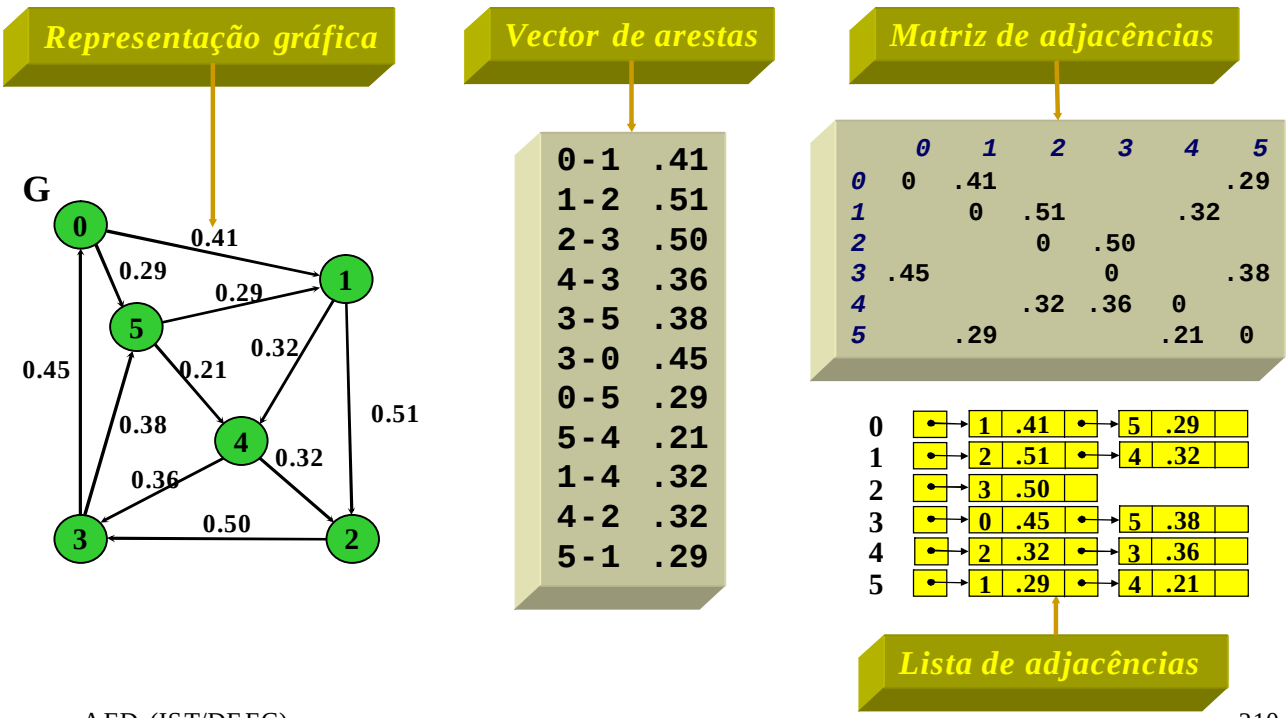
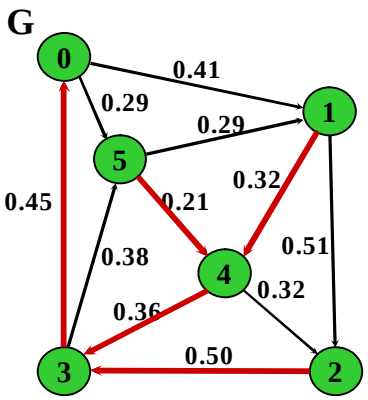
Grafos – Caminhos mais curtos

- Cada caminho num digrafo ponderado possui um peso - a soma dos pesos das arestas que o constituem.
- Esta característica origina directamente problemas como: determinar o caminho de menor peso entre dois vértices.
- Este e uma série de outros semelhantes são conhecidos como problemas de caminhos mais curtos - “shortest path problems”.
- Ir-se-á considerar grafos ponderados e direccionados, os quais serão designados como redes.

Grafos – Uma rede e suas representações

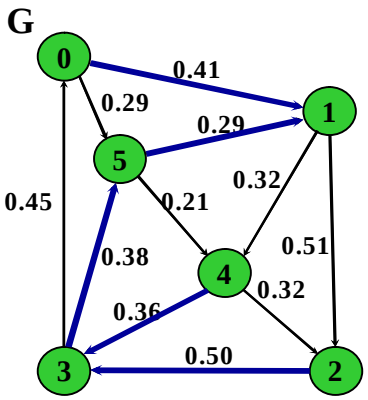


Grafos – Caminhos mais curtos (0)



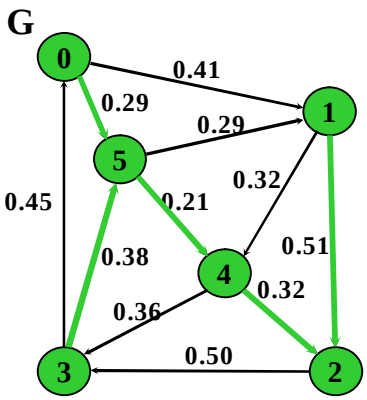

```
      0
1.13  1-4-3-0
0.95  2-3-0
0.45  3-0
0.81  4-3-0
1.02  5-4-3-0
```


Grafos – Caminhos mais curtos (1)



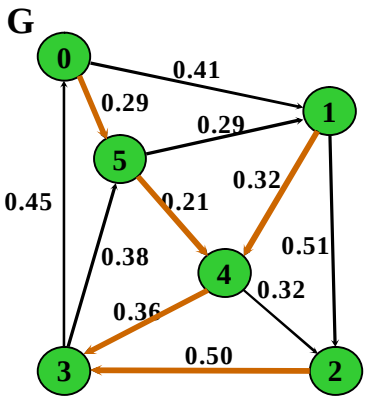

```
0.41  0-1
      1
1.17  2-3-5-1
0.67  3-5-1
1.03  4-3-5-1
0.29  5-1
```

Grafos – Caminhos mais curtos (2)



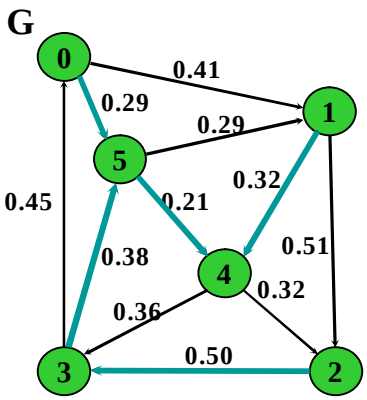

```
0.82  0-5-4-2
0.51  1-2
      2
0.91  3-5-4-2
0.32  4-2
0.53  5-4-2
```


Grafos – Caminhos mais curtos (3)



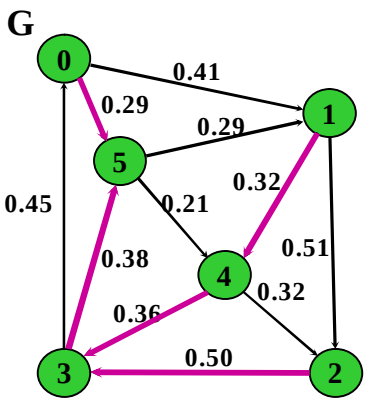

```
0.86  0-5-4-3
0.68  1-4-3
0.50  2-3
      3
0.36  4-3
0.57  5-4-3
```

Grafos – Caminhos mais curtos (4)



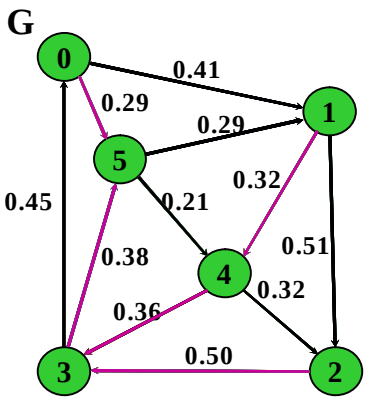

```
0.50  0-5-4
0.32  1-4
1.09  2-3-5-4
0.59  3-5-4
      4
0.21  5-4
```


Grafos – Caminhos mais curtos (5)




```
0.29  0-5
1.06  1-4-3-5
0.88  2-3-5
0.38  3-5
0.74  4-3-5
      5
```

Grafos – Caminhos mais curtos




```
      0
1.13  1-4-3-0
0.95  2-3-0
0.45  3-0
0.81  4-3-0
1.02  5-4-3-0
```

```
0.41  0-1
      1
1.17  2-3-5-1
0.67  3-5-1
1.03  4-3-5-1
0.29  5-1
```

```
0.82  0-5-4-2
0.51  1-2
      2
0.91  3-5-4-2
0.32  4-2
0.53  5-4-2
```

```
0.86  0-5-4-3
0.68  1-4-3
0.50  2-3
      3
0.36  4-3
0.57  5-4-3
```

```
0.50  0-5-4
0.32  1-4
1.09  2-3-5-4
0.59  3-5-4
      4
0.21  5-4
```

```
0.29  0-5
1.06  1-4-3-5
0.88  2-3-5
0.38  3-5
0.74  4-3-5
      5
```


Grafos – Definição e problemas

- Def. Um *caminho mais curto* entre dois vértices *s* e *t* numa rede é um caminho direccionado de *s* para *t* com a propriedade de não existir outro caminho direccionado com menor peso.
- Problema 1 - Caminho mais curto fonte-destino
  - Dado um vértice inicial *s* e um vértice destino *t*, qual o caminho mais curto no grafo de *s* para *t*?
- Problema 2 - Caminhos mais curtos de fonte única
  - Dado um vértice inicial *s*, quais os caminhos mais curtos que ligam *s* a todos os outros vértices?
- Problema 3 - Caminhos mais curtos entre todos
  - Quais os caminhos mais curtos ligando todos os vértices de um grafo?

Grafos – Motivação (1)

- Mapas de estrada
  - Serão os grafos de estradas entre cidades necessariamente não direccionados?
    - É igual ir até à torre da Serra da Estrela e vir de lá?
    - Igual em distância e/ou custo?
  - Sabendo a posição presente e o destino, pretende-se saber qual a próxima cidade a tomar.
- Rotas aéreas
  - Minimização do tempo necessário para viajar entre duas cidades?
  - Minimização do custo para viajar entre duas cidades?
    - Minimização de custo é mais complexo que minimização de distâncias.
    - Pode ser mais económico fazer uma escala do que o vôo directo.

Grafos – Motivação (2)

- Os algoritmos que estudaremos têm aplicação para lá destes exemplos e de outros similares.
- Existem problemas, aparentemente nada ligados com estes, para os quais os problemas de caminhos mais curtos podem ser relevantes.
- Enquanto que os problemas de caminhos mais curtos possuem uma natural intuição gráfica, outros não a possuem.
- Através de redução podem-se transformar alguns problemas em versões generalizadas dos caminhos mais curtos.

Grafos – Princípios de funcionamento (1)

- Os algoritmos de caminhos mais curtos a estudar assentam numa operação básica que dá pelo nome de ***relaxação***.

- Esta surge de várias formas nas diferentes implementações.

- Em essência consiste no seguinte:
  - Inicia-se um algoritmo conhecendo apenas as arestas e seus pesos.
  - À medida que se avança, recolhe-se informação sobre os caminhos mais curtos que unem vários pares de vértices.
  - A informação é actualizada incrementalmente, realizando-se inferências sobre os melhores caminhos a partir da informação recolhida.

Grafos – Princípios de funcionamento (2)

- Em cada passo testa-se se existe um caminho mais curto do que algum dos conhecidos.

- O termo “relaxação” é usado para descrever este último passo, porque tem como efeito relaxar algumas das restrições impostas ao caminho mais curto até ao momento.

- Pode-se imaginar um elástico esticado sobre o caminho mais curto entre dois vértices.

- Uma relaxação bem sucedida tem como efeito relaxar a tensão desse elástico sobre o caminho mais curto.

Grafos – Princípios de funcionamento (3)

- Os algoritmos são baseados na aplicação repetida de um dos seguintes tipos de relaxações
  - Testar se uma dada aresta define um novo caminho mais curto para o seu vértice destino.
  - Testar se a passagem por um dado vértice define um novo caminho mais curto ligando dois outros vértices dados.

- Ao primeiro tipo dá-se o nome de *relaxação de aresta* e ao segundo o nome de *relaxação de caminho*.

Grafos – Relaxação de aresta (1)

- Todos os *algoritmos de fonte única* considerados são baseados no seguinte passo.

  - Dada uma aresta, será que conduz a um caminho mais curto para o seu vértice destino?

- Para responder a esta questão torna-se necessário utilizar uma estrutura de dados adequada.

Grafos – Relaxação de aresta (2)

- Primeiro, é preciso calcular o comprimento de caminho mais curto do vértice fonte a todos os outros vértices.
- Armazenar esses valores num vector indexado pelos vértices, **`wt`**.
- Depois, para testar se uma dada aresta *v*-*w* com peso **`e.wt`** define um novo caminho mais curto do vértice destino até *w*, basta testar se **`wt[v] + e.wt`** é menor que **`wt[w]`**.
- Para ser possível calcular os caminhos, usa-se um outro vector indexado pelos vértices, **`st`**, que armazena o vértice antecessor no caminho mais curto a partir do vértice fonte para cada outro vértice.

Grafos – Relaxação de aresta (3)

- Def. Dada uma rede e um vértice fonte, *s*, *a árvore de caminhos mais curtos* (SPT*) para *s* define-se como sendo a união dos caminhos mais curtos de *s* para cada um dos restantes vértices da rede.

Código usado na relaxação de aresta:

```
if (wt[w] > wt[v] + e.wt)
  {
        wt[w] = wt[v] + e.wt;
        st[w] = v;
  }
```

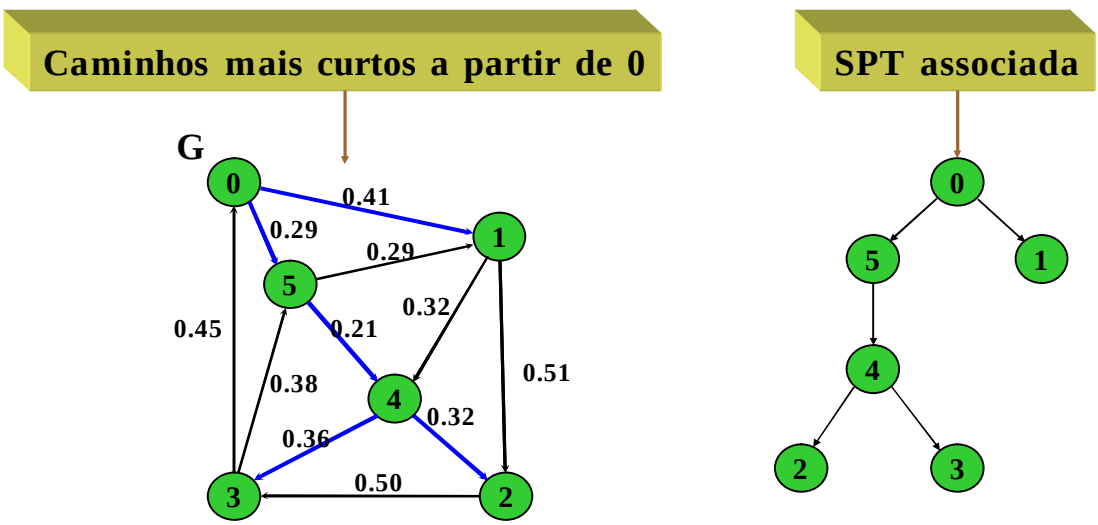
Como gerar o caminho a partir de **`st`**?

**** Shortest Path Tree***

Grafos – SPT's

- A definição da SPT descreve um conjunto de arestas.
- É uma árvore por ser um grafo ligado (todos os vértices estão ligados a *s* por um caminho) e não possui ciclos.
- Todas as arestas direccionadas apontam no sentido descendente da árvore.
- É uma árvore de suporte da sub-rede gerada pelos vértices atingíveis a partir do vértice fonte.

Grafos – Exemplo de SPT



Grafos – Relaxação de caminho (1)

- Todos os algoritmos de *caminhos mais curtos entre todos* considerados fazem uso da relaxação de caminho.

- Ou seja, fazem a seguinte pergunta:
  - A inclusão de e passagem por um dado vértice permite que o caminho entre outros dois vértices seja encurtado?

- Dados três vértices *s*, *x* e *t*, pretende-se saber se é mais económico ir de *s* a *t* passando por *x* ou não passando por *x*.

Grafos – Relaxação de caminho (2)

- Mantém-se uma matriz **`d`**, em que **`d[s][t]`** é o comprimento do caminho mais curto entre *s* e *t*.

- Mais uma matriz **`p`**, em que **`p[s][t]`** é o próximo vértice no caminho mais curto entre *s* e *t*.

- À primeira dá-se o nome de *matriz de distâncias* e à segunda de *matriz de caminhos*, ou sucessores.

Grafos – Relaxação de caminho (3)

Código usado na relaxação de caminho:

```
if (d[s][t] > d[s][x] + d[x][t])
 {
  d[s][t] = d[s][x] + d[x][t];
  p[s][t] = p[s][x];
 }
```

Como gerar o caminho a partir de **p**?

• Como ambas as relaxações se suportam nos operadores + e <, o resultado seguinte estabelece a correcção dos algoritmos nelas baseados.

Grafos – Princípio de optimalidade

- Prop. Se um vértice *x* pertence ao caminho mais curto entre *s* e *t*, então esse caminho consiste do caminho mais curto entre *s* e *x*, seguido do caminho mais curto entre *x* e *t*.

- Demonstração
  - Por redução ao absurdo.
  - Imagine-se que um dos dois caminhos não é o mais curto.
  - Então poder-se-ia usar o caminho mais curto entre *s* e *x* ou entre *x* e *t* para construir um caminho mais curto entre *s* e *t*.
  - O que contradiz a hipótese de ser óptimo o caminho original.

Grafos – Síntese da Aula 10

- Representações de redes
- Caminhos mais curtos
  - Caminho mais curto fonte-destino
  - Caminhos mais curtos de fonte única
  - Caminhos mais curtos entre todos
- Relaxação de aresta
  - SPT – *Shortest Path Tree*
- Relaxação de caminho
- Princípio de optimalidade

Grafos – Algoritmo de Dijkstra (1)

- No algoritmo de Prim, para construção da MST, adiciona-se uma aresta de cada vez, sendo esta a mais curta a ligar um vértice que já está na MST a um vértice que ainda não esteja.
  - Algoritmo "greedy" – voraz, avarento, ...
- O esquema para construção da SPT pode usar princípios semelhantes.
- Coloca-se o vértice fonte na raiz da árvore e constrói-se a árvore uma aresta de cada vez.
- Toma-se sempre a aresta que estabelece o caminho mais curto do vértice fonte a algum vértice que ainda não esteja na SPT.

Grafos – Algoritmo de Dijkstra (2)

- Por outras palavras, adicionam-se vértices à SPT por ordem crescente da sua distância (através da SPT) ao vértice de partida.
  - Também um algoritmo "greedy".
- Este algoritmo pode ser visto como um método de procura generalizada, diferente da DFS, da BFS e do algoritmo de Prim apenas pela regra de entrada de arestas na árvore.
- Este procedimento é conhecido como o *Algoritmo de Dijkstra*.

Grafos – Implementação

(listas de adjacências)

```
#define P (wt[v] + t->wt)
void GRAPHpfs(Graph G, int s, int st[], double
wt[])
{ int v, w; link t;
  PQinit();
  for (v = 0; v < G->V; v++)
   { st[v] = -1; wt[v] = maxWT; PQinsert(v);}
  wt[s] = 0.0; PQdec(s);
  while (!PQempty())
   if (wt[v = PQdelmin()] != maxWT)
    for (t = G->adj[v]; t != NULL; t = t->next)
     if (wt[w = t->v] > P)
      { wt[w] = P; PQdec(w); st[w] = v;}
}
```

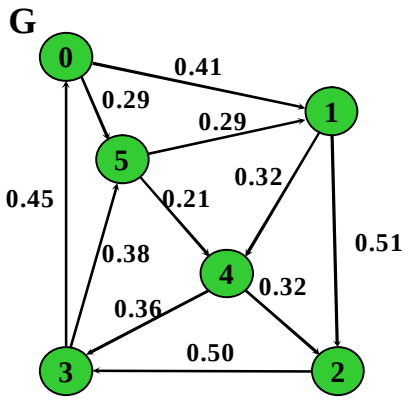
`PQinit` – inicializa uma fila baseada em prioridades (ver sec. 9.6).

`PQinsert` – insere um novo elemento de acordo com a sua prioridade.

`PQdelmin` – retira o elemento de mais baixa prioridade.

`PQdec` – altera a posição do elemento, por ter baixado a sua prioridade.

Grafos – Exemplo de execução (1)

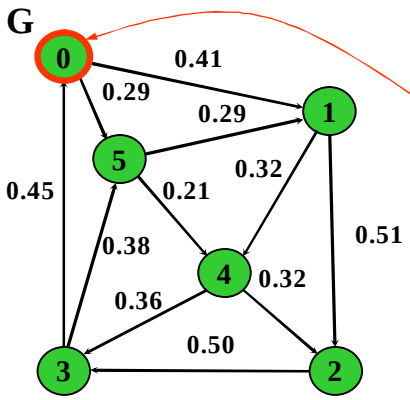


Lista de adjacências

- 0 : 1 - 5
- 1 : 4 - 2
- 2 : 3
- 3 : 0 - 5
- 4 : 3 - 2
- 5 : 1 - 4

- Chamada à função para determinar os caminhos mais curtos tomando o vértice 0 como fonte.
  - GRAPHpfs(G, 0, st, wt)

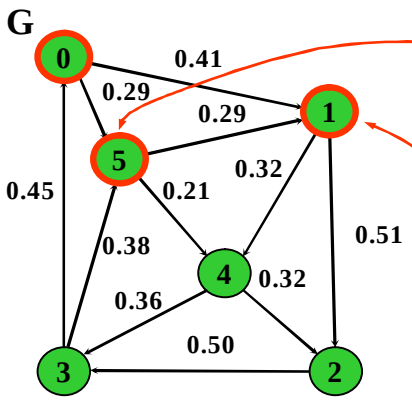
Grafos – Exemplo de execução (2)



Após a inicialização da fila tem-se
`Fila ¬ (0, 5, 4, 3, 2, 1)`
`st ¬ [-1 -1 -1 -1 -1 -1]`
`wt ¬ [0.0 100.0 100.0 100.0 100.0 100.0]`
Ciclo while
`v ¬ 0`
`wt[0] != 100.0 ü Fila ¬ (5, 4, 3, 2, 1)`
Ciclo for
`t ¬ 1`
`wt[1] > wt[0] + t->wt ü`
`wt[1] ¬ 0.41`
`Fila ¬ (1, 5, 4, 3, 2), st[1] ¬ 0`
`t ¬ 5`
`wt[5] > wt[0] + t->wt ü`
`wt[5] ¬ 0.29`
`Fila ¬ (5, 1, 4, 3, 2), st[5] ¬ 0`
Não há mais adjacentes de 0

`st` *vale* `[-1 0 -1 -1 -1 0]`

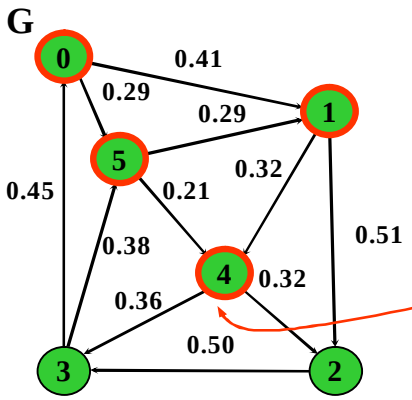
Grafos – Exemplo de execução (3)




```
st vale [-1 0 -1 -1 5 0]
```

```
Ciclo while (continuação)
v ¬ 5
  wt[5] != 100.0 ü Fila ¬ (1, 4, 3, 2)
  Ciclo for
    t ¬ 1
      wt[1] > wt[5] + t->wt X
    t ¬ 4
      wt[4] > wt[5] + t->wt ü
        wt[4] ¬ 0.50
      Fila ¬ (1, 4, 3, 2), st[4] ¬ 5
  Não há mais adjacentes de 5
v ¬ 1
  wt[1] != 100.0 ü Fila ¬ (4, 3, 2)
  Ciclo for
    t ¬ 4
      wt[4] > wt[1] + t->wt X
```

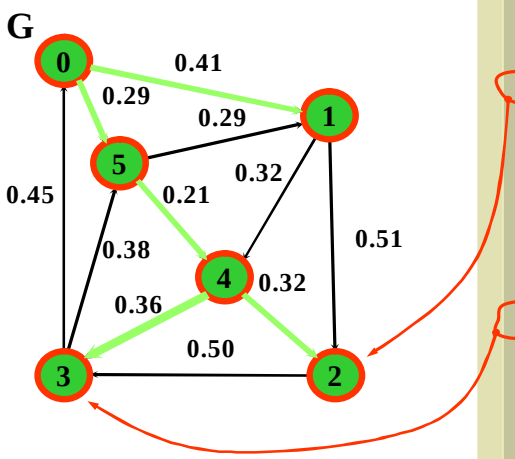

Grafos – Exemplo de execução (4)




```
st vale [-1 0 4 4 5 0]
```

```
Ciclo for (continuação)
  t ¬ 2
    wt[2] > wt[1] + t->wt ü
      wt[2] ¬ 0.91
    Fila ¬ (4, 2, 3), st[2] ¬ 1
  Não há mais adjacentes de 1
v ¬ 4
  wt[4] != 100.0 ü Fila ¬ (2, 3)
  Ciclo for
    t ¬ 3
      wt[3] > wt[4] + t->wt ü
        wt[3] ¬ 0.86
      Fila ¬ (3, 2), st[3] ¬ 4
    t ¬ 2
      wt[2] > wt[4] + t->wt ü
        wt[2] ¬ 0.82
      Fila ¬ (2, 3), st[2] ¬ 4
```

Grafos – Exemplo de execução (5)




```
st vale [-1 0 4 4 5 0]
```

```
Ciclo for (continuação)
  Não há mais adjacentes de 4
v ¬ 2
  wt[2] != 100.0 ü Fila ¬ (3)
  Ciclo for
    t ¬ 3
      wt[3] > wt[2] + t->wt x
  Não há mais adjacentes de 2
v ¬ 3
  wt[3] != 100 ü Fila ¬ ()
  Ciclo for
    t ¬ 0
      wt[0] > wt[3] + t->wt x
    t ¬ 5
      wt[5] > wt[3] + t->wt x
  Não há mais adjacentes de 3
Ciclo while termina!!!
```


Grafos – Propriedade do Alg. Dijkstra

- Prop. *O algoritmo de Dijkstra calcula SPT's correctamente.*
- Demonstração ***(assumindo todos os pesos não negativos):***
  - Dado um vértice fonte, *s*, temos que estabelecer que o caminho gerado pelo algoritmo desde *s* até um vértice *x* é o mais curto no grafo para esses dois vértices.
  - Prova-se por indução.
  - Assumindo que a árvore parcial calculada possui essa propriedade, necessitamos verificar se adicionando um novo vértice *x* se adiciona o caminho mais curto para esse vértice.
  - Mas todos os caminhos para *x* necessitam ter início num caminho da árvore seguido por uma aresta para um vértice ainda não na árvore.
  - Por construção, todos esses caminhos são mais compridos do que caminho em consideração.

Grafos – Observações e complexidade

- Observações
  - O algoritmo não funciona adequadamente para arestas com pesos negativos – *(Porquê?)*.
  - Alterando a definição de **P** para

    ```
    #define P (wt[v] + G->adj[v][w])
    ```

    obtém-se uma implementação adequada a grafos densos.
- Prop. *O Algoritmo de Dijkstra calcula qualquer SPT num grafo denso em tempo linear.*
- Demonstração
  - Resultado imediato da análise do código, dado que a entrada de um novo vértice impõe a re-avaliação das distâncias de todos os vértices fora da árvore e considerando que não existe procedimento de ordenação parcial ou total da fila.

Grafos – Dijkstra para grafos esparsos

- Em grafos esparsos a complexidade depende da forma como a lista de prioridades é gerida.
- As implementações mais eficientes requerem a utilização de "heaps"
- Para as variantes possíveis com "heaps" a complexidade varia entre $E \lg V$ e $E \lg_d V$ (d é um parâmetro associado com uma implementação de heap parcial).
- Neste último caso, a complexidade pode ser considerada linear, a menos que o grafo seja excessivamente esparso.

Grafos – SPT's de todos para todos

- Existem duas formas de resolver o problema de determinar os caminhos mais curtos entre qualquer par de vértices.
- A primeira solução deverá ser óbvia nesta altura...
  - Aplicar o algoritmo de Dijkstra *V* vezes tomando cada um dos vértices como fonte.
- A complexidade desta solução também deverá ser óbvia...
  - É *V**(o tempo de aplicar o algoritmo de Dijkstra uma vez).
  - Entre parêntesis está uma complexidade que depende da implementação específica e do carácter esparso ou denso do grafo em questão.
- A segunda, resolve o problema directamente num tempo proporcional a V^3 – *Algoritmo de Floyd*.

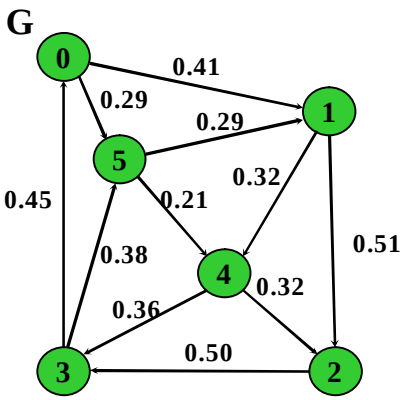
Grafos – Algoritmo de Floyd

(Implementação mat. adjacên

```
void GRAPHspALL(Graph G)
{ int i, s, t;
  double **d = MATRIXdouble(G->V, G->V,
maxWT);
  int **p = MATRIXint(G->V, G->V, G->V);
  for (s = 0; s < G->V; s++)
   for (t = 0; t < G->V; t++)
    if ((d[s][t] = G->adj[s][t]) < maxWT)
     p[s][t] = t;
  for (i = 0; i < G->V; i++)
   for (s = 0; s < G->V; s++)
    for (t = 0; t < G->V; t++)
     if (d[s][t] > d[s][i] + d[i][t])
      { p[s][t] = p[s][i];
        d[s][t] = d[s][i] + d[i][t];}
  G->dist = d; G->path = p;
}
```

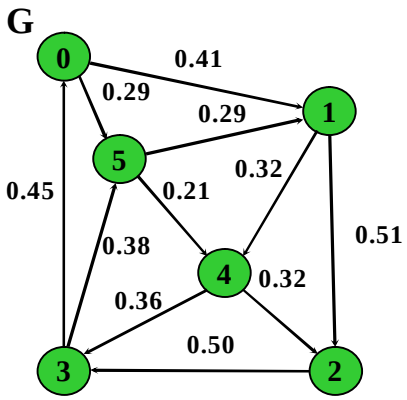
1. Inicialização da matriz de distâncias.
2. Inicialização da matriz de caminhos.
3. Melhores caminhos iniciais são os directos.
4. Relaxação de caminho.
5. Registo dos melhores caminhos e respectivas distâncias como campos da estrutura do grafo.

Grafos – Exemplo de execução (1)



- Chamada à função para determinar todos os caminhos mais curtos.
  - GRAPHspALL(G)

Grafos – Exemplo de execução (2)



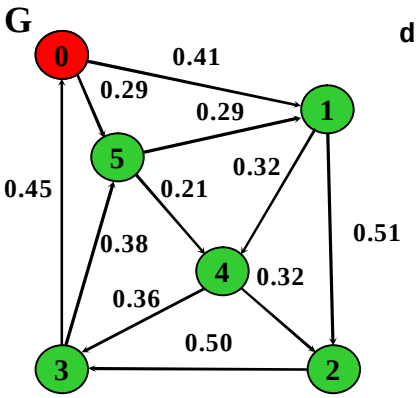
Inicialização #1

$$d = \begin{pmatrix} ¥ & ¥ & ¥ & ¥ & ¥ & ¥ \\ ¥ & ¥ & ¥ & ¥ & ¥ & ¥ \\ ¥ & ¥ & ¥ & ¥ & ¥ & ¥ \\ ¥ & ¥ & ¥ & ¥ & ¥ & ¥ \\ ¥ & ¥ & ¥ & ¥ & ¥ & ¥ \\ ¥ & ¥ & ¥ & ¥ & ¥ & ¥ \end{pmatrix} \quad p = \begin{pmatrix} 6 & 6 & 6 & 6 & 6 & 6 \\ 6 & 6 & 6 & 6 & 6 & 6 \\ 6 & 6 & 6 & 6 & 6 & 6 \\ 6 & 6 & 6 & 6 & 6 & 6 \\ 6 & 6 & 6 & 6 & 6 & 6 \\ 6 & 6 & 6 & 6 & 6 & 6 \end{pmatrix}$$

Inicialização #2

$$d = \begin{pmatrix} 0 & 0.41 & ¥ & ¥ & ¥ & 0.29 \\ ¥ & 0 & 0.51 & ¥ & 0.32 & ¥ \\ ¥ & ¥ & 0 & 0.50 & ¥ & ¥ \\ 0.45 & ¥ & ¥ & 0 & ¥ & 0.38 \\ ¥ & ¥ & 0.32 & 0.36 & 0 & ¥ \\ ¥ & 0.29 & ¥ & ¥ & 0.21 & 0 \end{pmatrix} \quad p = \begin{pmatrix} 0 & 1 & 6 & 6 & 6 & 5 \\ 6 & 1 & 2 & 6 & 4 & 6 \\ 6 & 6 & 2 & 3 & 6 & 6 \\ 0 & 6 & 6 & 3 & 6 & 5 \\ 6 & 6 & 2 & 3 & 4 & 6 \\ 6 & 1 & 6 & 6 & 4 & 5 \end{pmatrix}$$

Grafos – Exemplo de execução (3)



Ciclo for exterior com i = 0

d = (d[i][t] →)

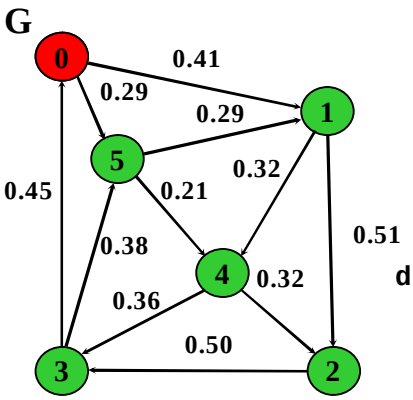
0	0.41	¥	¥	¥	0.29
¥	0	0.51	¥	0.32	¥
¥	¥	0	0.50	¥	¥
0.45	¥	¥	0	¥	0.38
¥	¥	0.32	0.36	0	¥
¥	0.29	¥	¥	0.21	0

d[s][i]

Únicas entradas finitas

```
s = 0
 Ciclo for interior não faz nada
s = 3
 Ciclo for interior
  t ¬ 0 nada
  t ¬ 1
   d[3][1] > d[3][0] + d[0][1] ü
  t ¬ 2, 3, 4, 5 nada
```


Grafos – Exemplo de execução (4)



Alterações resultantes para i = 0

```
d[3][1] ¬ 0.86
p[3][1] ¬ p[3][0] = 0
```

d =

0	0.41	¥	¥	¥	0.29
¥	0	0.51	¥	0.32	¥
¥	¥	0	0.50	¥	¥
0.45	0.86	¥	0	¥	0.38
¥	¥	0.32	0.36	0	¥
¥	0.29	¥	¥	0.21	0

p =

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

Quando i = 0 está-se a considerar se é mais económica a ligação directa ou via vértice 0. Neste caso, a ligação directa não existia e passou a ser possível atingir o vértice 1 a partir do vértice 3.

Grafos – Exemplo de execução (5)

Ciclo for exterior com i = 1

$$d = \begin{pmatrix} 0 & 0.41 & 0.92 & ¥ & 0.73 & 0.29 \\ ¥ & 0 & 0.51 & ¥ & 0.32 & ¥ \\ ¥ & ¥ & 0 & 0.50 & ¥ & ¥ \\ 0.45 & 0.86 & 1.37 & 0 & 1.18 & 0.38 \\ ¥ & ¥ & 0.32 & 0.36 & 0 & ¥ \\ ¥ & 0.29 & 0.80 & ¥ & 0.21 & 0 \end{pmatrix} \quad p = \begin{pmatrix} 0 & 1 & 1 & 6 & 1 & 5 \\ 6 & 1 & 2 & 6 & 4 & 6 \\ 6 & 6 & 2 & 3 & 6 & 6 \\ 0 & 0 & 0 & 3 & 0 & 5 \\ 6 & 6 & 2 & 3 & 4 & 6 \\ 6 & 1 & 1 & 6 & 4 & 5 \end{pmatrix}$$

Ciclo for exterior com i = 2

$$d = \begin{pmatrix} 0 & 0.41 & 0.92 & 1.42 & 0.73 & 0.29 \\ ¥ & 0 & 0.51 & 1.01 & 0.32 & ¥ \\ ¥ & ¥ & 0 & 0.50 & ¥ & ¥ \\ 0.45 & 0.86 & 1.37 & 0 & 1.18 & 0.38 \\ ¥ & ¥ & 0.32 & 0.36 & 0 & ¥ \\ ¥ & 0.29 & 0.80 & 1.30 & 0.21 & 0 \end{pmatrix} \quad p = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 5 \\ 6 & 1 & 2 & 2 & 4 & 6 \\ 6 & 6 & 2 & 3 & 6 & 6 \\ 0 & 0 & 0 & 3 & 0 & 5 \\ 6 & 6 & 2 & 3 & 4 & 6 \\ 6 & 1 & 1 & 1 & 4 & 5 \end{pmatrix}$$

Grafos – Exemplo de execução (6)

Ciclo for exterior com i = 3

$$d = \begin{pmatrix} 0 & 0.41 & 0.92 & 1.42 & 0.73 & 0.29 \\ 1.46 & 0 & 0.51 & 1.01 & 0.32 & 1.39 \\ 0.95 & 1.36 & 0 & 0.50 & 1.68 & 0.88 \\ 0.45 & 0.86 & 1.37 & 0 & 1.18 & 0.38 \\ 0.81 & 1.22 & 0.32 & 0.36 & 0 & 0.74 \\ 1.75 & 0.29 & 0.80 & 1.30 & 0.21 & 0 \end{pmatrix} \quad p = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 5 \\ 2 & 1 & 2 & 2 & 4 & 2 \\ 3 & 3 & 2 & 3 & 3 & 3 \\ 0 & 0 & 0 & 3 & 0 & 5 \\ 3 & 3 & 2 & 3 & 4 & 3 \\ 1 & 1 & 1 & 1 & 4 & 5 \end{pmatrix}$$

Ciclo for exterior com i = 4

$$d = \begin{pmatrix} 0 & 0.41 & 0.92 & 1.09 & 0.73 & 0.29 \\ 1.13 & 0 & 0.51 & 0.68 & 0.32 & 1.06 \\ 0.95 & 1.36 & 0 & 0.50 & 1.68 & 0.88 \\ 0.45 & 0.86 & 1.37 & 0 & 1.18 & 0.38 \\ 0.81 & 1.22 & 0.32 & 0.36 & 0 & 0.74 \\ 1.02 & 0.29 & 0.53 & 0.57 & 0.21 & 0 \end{pmatrix} \quad p = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 5 \\ 4 & 1 & 2 & 4 & 4 & 4 \\ 3 & 3 & 2 & 3 & 3 & 3 \\ 0 & 0 & 0 & 3 & 0 & 5 \\ 3 & 3 & 2 & 3 & 4 & 3 \\ 4 & 1 & 4 & 4 & 4 & 5 \end{pmatrix}$$

Grafos – Exemplo de execução (7)

Ciclo for exterior com i = 5

```
d =  0    0.41 0.82 0.86 0.50 0.29
    1.13   0    0.51 0.68 0.32 1.06
    0.95 1.17   0    0.50 1.09 0.88
    0.45 0.67 0.91   0    0.59 0.38
    0.81 1.03 0.32 0.36   0    0.74
    1.02 0.29 0.53 0.57 0.21   0
```

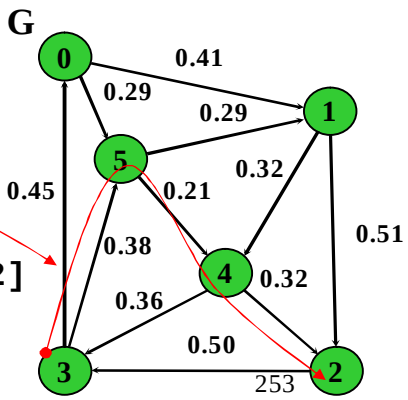
```
p =  0 1 5 5 5 5
     4 1 2 4 4 4
     3 3 2 3 3 3
     0 5 5 3 5 5
     3 3 2 3 4 3
     4 1 4 4 4 5
```

Qual o caminho mais curto entre o vértice 3 e o vértice 2?

```
p[3][2] = 5
p[5][2] = 4
p[4][2] = 2
```

Qual o seu comprimento?

```
d[3][2] = 0.91
d[3][2] = adj[3][5] + adj[5][4] + adj[4][2]
        = 0.38 + 0.21 + 0.32
```



Grafos – Correcção e eficiência (1)

- Prop. Com o algoritmo de Floyd, pode-se determinar todos os caminhos mais curtos de todos para todos em tempo proporcional a V^3.
- Demonstração
  - O tempo de execução é trivialmente determinado por inspecção do código.
  - Quanto à correcção do algoritmo, prova-se por indução.
  - A *i*-ésima iteração do ciclo principal calcula o caminho mais curto de *s* para *t* não incluindo vértices de índice superior a *i*, excepto possivelmente apenas *s* e *t*.
  - Assumindo esta afirmação como verdadeira vamos ver o que se passa na iteração *i+1*.

Grafos – Correcção e eficiência (2)

- Demonstração (continuação)
  - O caminho mais curto entre *s* e *t* que não inclua vértices de índice superior a *i+1* é:
  - Um caminho entre *s* e *t* que não inclui nenhum vértice de índice superior a *i* e cujo comprimento é **`d[s][t].`**
    - Terá sido encontrado numa iteração anterior à *i*-ésima, logo aplica-se a hipótese da indução.
  - Um caminho constituído por um primeiro caminho entre *s* e *i* e um segundo caminho entre *i* e *t*.
    - Nenhum destes dois inclui vértices de índice superior a i.
  - Em qualquer dos dois casos o ciclo interno actualiza **`d[s][t]`**.

Grafos – Síntese da Aula 11