Licenciatura em Engenharia Informática e de Computadores

Fundamentos de Programação

Projecto - Terceira Parte

2 de Dezembro de 2011

Caminhos mais curtos em grafos

Os grafos são estruturas de informação importantes em Informática. Uma das tarefas que é frequente ter de realizar sobre grafos é a determinação do caminho mais curto entre dois nós, tarefa para a qual existem vários algoritmos.

O objectivo deste projecto é o desenvolvimento de um programa para a determinação do caminho mais curto entre dois nós de um grafo, usando o algoritmo de Dijkstra.

1 O algoritmo de Dijkstra

O algoritmo de Dijkstra determina o caminho mais curto entre dois nós de um grafo rotulado. Considere-se o grafo da Figura 1. Neste grafo, existem dois caminhos entre os nós A e C: $[A, C]$, de comprimento 14, e $[A, B, C]$, de comprimento 10. Assim, o caminho mais curto entre A e C é o caminho $[A, B, C]$.

Durante o seu funcionamento, o algoritmo associa determinada informação aos nós do grafo. A *informação associada a um nó* consiste em:

- uma *distância*, um inteiro,
- um *nó anterior*, um nó.

Dado um nó n, representamos por $dist(n)$ e $ant(n)$, a distância e o nó anterior associados ao nó n, respectivamente. As distâncias associadas a todos os nós são inicializadas a ∞, excepto a do nó origem, que é inicializada a zero. Os nós anteriores associados a todos os nós começam por ser indefinidos. Considerando o grafo da Figura 2, e assumindo que o

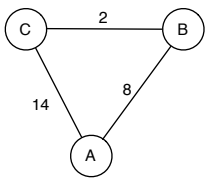

Figura 1: Exemplo de grafo.

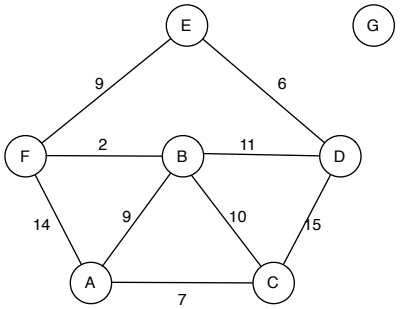

Figura 2: Exemplo de grafo.

n	A	B	C	D	E	F	G
$dist(n)$	0	∞	∞	∞	∞	∞	∞
$ant(n)$	?	?	?	?	?	?	?

Tabela 1: Valores iniciais.

nó origem é o nó A, os valores de $dist(n)$ e $ant(n)$ seriam os apresentados na Tabela 1 (o valor "indefinido"é representado pelo símbolo "?").

Apresenta-se abaixo o pseudo-código do algoritmo. O símbolo "<-" representa a operação de atribuição.

```
caminho-mais-curto(origem,destino,grafo)

nos-G <- nos(grafo)
Inicializar a informação associada a cada um dos nós de nos-G
ENQUANTO pertence?(destino,nos-G)
    n <-  nó de nos-G com a menor distância associada
    nos-G <-  retira-conj(n,nos-G)
    SE n não for o nó destino
     PARA CADA v do conjunto interseccao(vizinhos(n,grafo),nos-G)
          nova-dist <- distancia-entre(n,v,grafo) + dist(n)
          SE nova-dist < dist(v)
                   dist(v) <- nova-dist
                   ant(v) <- n
          FIM SE
     FIM PARA CADA
    FIM SE
FIM ENQUANTO
caminho <- nova-lista()
n <- destino
ENQUANTO n não for indefinido
    caminho <- insere(n,caminho)
    n <- ant(n)
FIM ENQUANTO
DEVOLVE caminho
```

n	A	B	C	D	E	F	G
$dist(n)$	0	9	7	∞	∞	14	∞
$ant(n)$	?	A	A	?	?	A	?

Tabela 2: Valores após o processamento do nó `A`.

n	A	B	C	D	E	F	G
$dist(n)$	0	9	7	22	∞	14	∞
$ant(n)$	?	A	A	C	?	A	?

Tabela 3: Valores após o processamento do nó `C`.

Nas Tabelas 2, 3 e 4 apresentam-se, para cada iteração do ciclo `ENQUANTO`, os valores da distância e do nó anterior associados a cada nó do grafo, assumindo que o grafo é o apresentado na Figura 2, e que os nós origem e destino são os nós `A` e `F`, respectivamente. Na última iteração é escolhido o nó `F` que, por ser o nó destino, já não é processado, provocando o fim do ciclo. Assim, os valores finais são os apresentados na Figura 4. A partir destes valores é construído o caminho mais curto entre os nós `A` e `F`: `(A B F)`.

2 Programa a desenvolver

Deverá desenvolver um procedimento, `(caminho-mais-curto origem destino grafo)`, correspondente a uma implementação do algoritmo de Dijkstra, que, dados um nó origem, um nó destino, e um grafo, determina o caminho mais curto entre a origem e o destino, considerando o grafo dado. O seu procedimento deve validar os argumentos que recebe, isto é, verificar se o terceiro argumento é um grafo, e se os dois primeiros argumentos correspondem a nós desse grafo.

Para além de calcular o caminho mais curto entre dois nós de um grafo, o seu procedimento deverá também apresentar a distância entre nós adjacentes do caminho, bem como a distância total. O procedimento `caminho-mais-curto` deve **devolver** um par em que:

- O primeiro elemento do par é uma lista que contém os nós do caminho, bem como, entre cada dois nós, a distância entre eles.
- O segundo elemento do par é o comprimento total do caminho.

Usando o seu procedimento, e considerando que `grafo` é o grafo da Figura 2, deverá ser possível obter a seguinte interacção:

n	A	B	C	D	E	F	G
$dist(n)$	0	9	7	20	∞	11	∞
$ant(n)$	?	A	A	B	?	B	?

Tabela 4: Valores após o processamento do nó `B`.

```
> (caminho-mais-curto 'A 'F grafo)
((A 9 B 2 F) . 11)
> (caminho-mais-curto 'A 'E grafo)
((A 9 B 2 F 9 E) . 20)
> (caminho-mais-curto 'E 'E grafo)
(() . 0)
> (caminho-mais-curto 'A 'G grafo)
(() . +inf.0)
```

O resultado da última interacção indica que não existe nenhum caminho entre os nós `A` e `G`. A constante primitiva `+inf.0` representa o valor $+\infty$.

Na secção seguinte são definidos os tipos de informação a desenvolver antes de implementar o algoritmo de Dijkstra.

3 Tipos de informação

Para além dos tipos da segunda parte do projecto, deverá implementar os tipos abaixo especificados.

Para não sobrecarregar desnecessariamente o código, apenas os construtores de cada tipo devem verificar a validade dos argumentos que recebem.

3.1 O tipo info-nó

O tipo info-nó é usado para representar a informação associada a cada nó pelo algoritmo de Dijkstra. Um elemento do tipo info-nó é constituído por um nó n, pela distância associada a n, e pelo nó anterior associado a n. Note que cada coluna das tabelas apresentadas na Secção 1 corresponde a um elemento do tipo info-nó. O tipo info-nó deverá disponibilizar as seguintes operações básicas:

- Construtor:
  - $novo\text{-}info\text{-}no : nó \mapsto info\text{-}nó$
    $novo\text{-}info\text{-}no(n)$ devolve o elemento do tipo info-nó cujo nó é n, cuja distância é ∞, e cujo nó anterior é "indefinido".[1]
- Selectores:
  - $no\text{-}info\text{-}no : info\text{-}nó \mapsto nó$
    $no\text{-}info\text{-}no(i)$ devolve o nó de i.
  - $dist\text{-}info\text{-}no : info\text{-}nó \mapsto inteiro$
    $dist\text{-}info\text{-}no(i)$ devolve a distância de i.
  - $ant\text{-}info\text{-}no : info\text{-}nó \mapsto nó$
    $ant\text{-}info\text{-}no(i)$ devolve o nó anterior de i.

[1] Na implementação, represente o valor "indefinido" por `null`.

- Modificadores:
  - $modifica\text{-}dist! : info\text{-}nó \times inteiro \mapsto info\text{-}nó$
    $modifica\text{-}dist!(i, d)$ altera a distância de i para d. Devolve i com a distância alterada.
  - $modifica\text{-}ant! : info\text{-}nó \times nó \mapsto info\text{-}nó$
    $modifica\text{-}ant!(i, n)$ altera o nó anterior de i para n. Devolve i com o nó anterior alterado.

3.2 O tipo info-nós

O tipo info-nós é usado para representar a informação associada aos nós de um grafo pelo algoritmo de Dijkstra. Um elemento do tipo info-nós é constituído por uma colecção de elementos do tipo info-nó. Note que cada uma das tabelas apresentadas na Secção 1 corresponde a um elemento do tipo info-nós. O tipo info-nós deverá disponibilizar as seguintes operações básicas:

- Construtor:
  - $cria\text{-}info\text{-}inicial : conjunto\ de\ nós \mapsto info\text{-}nós$
    $cria\text{-}info\text{-}inicial(c)$ devolve a colecção de elementos do tipo info-nó obtidos por aplicação de $novo\text{-}info\text{-}no$ a cada nó do conjunto c.
- Selectores:
  - $obtem\text{-}info\text{-}no : info\text{-}nós \times nó \mapsto info\text{-}nó$
    $obtem\text{-}info\text{-}no(i, n)$ devolve o elemento de i cujo nó é n.
  - $no{<}dist : info\text{-}nós \times conjunto\ de\ nós \mapsto nó$
    $no{<}dist(i, c)$ devolve o nó do conjunto c com a menor distância associada em i.
- Modificador:
  - $actualiza\text{-}dist\text{-}ant\text{-}no! : info\text{-}nós \times nó \times inteiro \times nó \mapsto info\text{-}nós$
    $actualiza\text{-}dist\text{-}ant\text{-}no!(i, n, d, a)$ altera a distância e o nó anterior associados ao nó n em i, para d e a, respectivamente. Devolve i com a informação associada ao nó n alterada.

Para uma melhor compreensão do funcionamento destas operações básicas, apresentam-se de seguida alguns exemplos.

A operação $cria\text{-}info\text{-}inicial(\{A, B, C, D, E, F, G\})$ devolve um elemento do tipo info-nós com o seguinte conteúdo:

A	B	C	D	E	F	G
∞	∞	∞	∞	∞	∞	∞
?	?	?	?	?	?	?

Designando por i o valor devolvido pela operação anterior, a execução da sequência

$actualiza\text{-}dist\text{-}ant\text{-}no!(i, A, 0, ?)$
$actualiza\text{-}dist\text{-}ant\text{-}no!(i, C, 7, A)$
$actualiza\text{-}dist\text{-}ant\text{-}no!(i, B, 9, A)$
$actualiza\text{-}dist\text{-}ant\text{-}no!(i, F, 14, A)$

altera o valor de i para

A	B	C	D	E	F	G
0	9	7	∞	∞	14	∞
?	A	A	?	?	A	?

e devolve i.

A operação $obtem\text{-}info\text{-}no(i, F)$ devolve o elemento do tipo info-nó com o seguinte conteúdo:

F
14
A

Finalmente, a operação $no{<}dist(i, \{B, C, D, E, F, G\})$ devolve o nó C.

4 Passos a seguir

Apresentam-se a seguir os passos que deve seguir no desenvolvimento do seu programa:

1. É **obrigatória** a utilização da solução da segunda parte disponível na página. Para tal, coloque a forma `(require "solucao-parte2.rkt")` a seguir à forma `#lang scheme`.
2. Implemente e teste os tipos de informação descritos na Secção 3.
3. Antes de pensar em desenvolver o seu programa, é **essencial** que compreenda perfeitamente o algoritmo de Dijkstra. Para tal deverá ser capaz de executar o algoritmo "à mão". Use, entre outros, os exemplos de interacção apresentados.
4. Modifique o algoritmo apresentado de forma a que, para além do caminho mais curto entre dois nós de um grafo, sejam também apresentadas as distâncias entre nós adjacentes do caminho, e o comprimento total do caminho. A forma destes resultados deve seguir escrupulosamente as indicações dadas na Secção 2.
5. Implemente o algoritmo modificado.

5 Determinação incremental da informação associada aos nós

Se se pretender determinar o caminho mais curto entre vários pares de nós de um grafo, o algoritmo apresentado não é eficiente, pois não tira partido do trabalho efectuado anteriormente na determinação da distância e do nó anterior associados aos nós do grafo. Por exemplo, em relação ao grafo da Figura 2, a informação associada aos nós durante a determinação do caminho mais curto entre os nós `A` e `F` é calculada de novo a partir do zero, desnecessariamente, para a determinação do caminho mais curto entre os nós `A` e `E`. Com efeito, bastaria processar o nó `F` para obter esta informação.

Para evitar esta situação, podemos pensar num procedimento com estado interno, que mantém um registo da informação calculada anteriormente e usa esta informação quando tem que determinar o caminho mais curto entre dois nós de um grafo, não a calculando de novo.

Assim, pretende-se agora que escreva um procedimento com estado interno que determine o caminho mais curto entre dois nós de um grafo, tal como o procedimento `caminho-mais-curto`, mas que não faça cálculos repetidos da informação associada aos nós.

Apresenta-se de seguida um exemplo de interacção com o seu programa. Suponha que `grafo1` e `grafo2` correspondem aos grafos das Figuras 1 e 2, respectivamente.[2]

```
> (define caminho-mais-curto1
          (cria-proc-caminhos-mais-curtos grafo1))
> (define caminho-mais-curto2
          (cria-proc-caminhos-mais-curtos grafo2))
> (caminho-mais-curto1 'A 'C)
Nós processados: (A B)
((A 8 B 2 C) . 10)
> (caminho-mais-curto1 'A 'B)
Nós processados: ()
((A 8 B) . 8)
> (caminho-mais-curto2 'A 'F)
Nós processados: (A C B)
((A 9 B 2 F) . 11)
> (caminho-mais-curto2 'A 'E)
Nós processados: (F)
((A 9 B 2 F 9 E) . 20)
> (caminho-mais-curto2 'A 'C)
Nós processados: ()
((A 7 C) . 7)
> (caminho-mais-curto2 'D 'F)
Nós processados: (D E B)
((D 11 B 2 F) . 13)
```

Para obter a interacção acima, deve escrever o procedimento

[2] A interacção apresentada foi obtida colocando instruções de escrita temporárias no procedimento com estado interno, de forma a obter a lista dos nós processados, isto é, os sucessivos valores da variável `n` do algoritmo da pág. 2, à excepção do nó `destino`, uma vez que este já não é processado.

```
(cria-proc-caminhos-mais-curtos grafo)
```

que recebe um grafo, e devolve um procedimento com estado interno que determina o caminho mais curto entre dois nós de `grafo`, sem fazer cálculos repetidos da informação associada aos nós.

Sugerem-se os seguintes passos:

1. Comece por identificar a informação que é necessário registar, isto é, quais as variáveis de estado que o procedimento com estado interno deve ter.
2. Numa primeira fase, e para facilitar a depuração do seu programa, as variáveis de estado podem corresponder a variáveis globais.
3. Altere o algoritmo de Dijskstra de forma a que seja verificado se é necessário processar mais nós, ou se, pelo contrário, toda ou parte da informação necessária já foi calculada em execuções anteriores e, consequentemente, já se encontra nas variáveis de estado. O algoritmo modificado deverá registar nas variáveis de estado toda a nova informação que calcula. **ATENÇÃO: não deve alterar o procedimento `caminhos-mais-curtos`.**
4. Escreva agora o procedimento `cria-proc-caminhos-mais-curtos` de forma a que as variáveis de estado que identificou passem a ser internas.

6 Classificação

A nota da terceira parte do projecto será baseada nos seguintes aspectos:

1. Execução correcta (40%).
   (a) Tipos de informação (Secção 3) - 2 valores.
   (b) Algoritmo de Dijkstra - 3.5 valores.
   (c) Procedimento com estado interno - 2.5 valores.

   Esta parte da avaliação é feita recorrendo a um programa de avaliação automática.
2. Facilidade de leitura, nomeadamente abstracção procedimental, abstracção de dados, nomes bem escolhidos, paragrafação correcta, qualidade (e não quantidade) dos comentários[3] e tamanho dos procedimentos (25%).
3. Estilo de programação (5%).
4. Relatório (30%).

Os pesos das notas das três partes na nota final do projecto são os seguintes:

1. Primeira parte - 10%.
2. Segunda parte - 30%.
3. Terceira parte - 60%.

[3]Tal como na primeira parte do projecto, todos os comentários do seu programa devem ser feitos utilizando a opção “Comment Out with Semicolons". Programas que utilizem a opção “Comment Out with a Box” ou caracteres acentuados serão penalizados com três valores.

7 Condições de realização e prazos

O ficheiro contendo o código deverá conter na primeira linha `#lang scheme`, na segunda linha `(provide (all-defined-out))`, na terceira linha `(require "solucao-parte2.rkt")` e, a partir da quarta linha, em comentário, os números e os nomes dos alunos do grupo, bem como o número do grupo.

O projecto deve ser realizado pelos mesmos grupos da primeira e segunda partes.

A terceira parte do projecto deve ser entregue até às 15:00 horas do dia **3 de Janeiro de 2012**, na Reprografia do DEI ou na portaria do Tagus (consoante o campus que corresponda ao grupo do projecto), e deverá constar de um relatório, incluindo uma listagem do código (numa fonte "monospace" como, por exemplo, a fonte "courier"). Um modelo de relatório, que podem/devem adaptar de acordo com as vossas necessidades, será disponibilizado no sistema Fénix. Projectos em atraso serão aceites até dia 4 de Janeiro, sendo penalizados com 1 valor. A partir das 15:00 horas do dia 4 de Janeiro de 2012 não se aceitam quaisquer projectos, seja qual for o pretexto. O relatório deve ser entregue dentro de uma capa Roma (à venda na Reprografia do DEI), apresentando visivelmente o número do grupo e número e nome dos seus autores, na capa. Projectos que não sejam entregues nestas condições serão penalizados com três valores.

Para além disto, a submissão do código por via electrónica, *através do sistema Fénix*, é *obrigatória* e deverá ser feita nos mesmos prazos que a entrega do relatório. Se o código e o relatório forem entregues em horas diferentes, o desconto será o correspondente ao da última entrega. O código do projecto deve estar contido num único ficheiro. Se durante o desenvolvimento usaram vários ficheiros devem, no final, antes de submeter o código, colocar todo o código num único ficheiro, chamado `FP1112-parte3-grupo`n`.rkt`, em que n é o número do grupo. Por exemplo, o ficheiro do grupo nº 5 deverá chamar-se `FP1112-parte3-grupo5.rkt`.

Durante o desenvolvimento do programa é importante não se esquecer da *Lei de Murphy*:

1. Todos os problemas são mais difíceis do que parecem;
2. Tudo demora mais tempo do que nós pensamos;
3. Se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis.

Pode ou não haver uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

Projectos iguais, ou muito semelhantes, serão considerados cópias. O corpo docente da cadeira será o único juiz do que se considera ou não copiar no projecto.