

Métodos Formais em Eng.^a de Software

ALGUNS EXERCÍCIOS DE EXAMES ANTERIORES

- Qual o valor das expressões, em VDM, que se seguem:
 - $\text{dom } \{mk_{(1,2)}.#1 \mapsto 3, mk_{(2,3)}.#2 \mapsto 4\}$
 - $[[5,6],[3,1,1],[5]] ++ \{2 \mapsto [5,5], 3 \mapsto [8]\}$
 - $\{mk_{(x,y)} \mid x \text{ in set elems } ([1,2,2,1] \wedge [2]), y \text{ in set inds } [0,1] \ \& \ x \leq y\}$
 - $\{x \mapsto y \mid x \text{ in set dom } (\{1 \mapsto 2, 2 \mapsto 3\} \rightarrow \{3\}), y \text{ in set rng } \{1 \mapsto 4\} \ \& \ y = x * 2\}$
 - $\text{conc } ([[1,2],[2],[3,2]] ++ \{1 \mapsto [3]\})$
 - $\{1 \mapsto 2, 2 \mapsto 1, 4 \mapsto 4\} \text{ union } (\{1 \mapsto 1, 2 \mapsto 2\} ++ \{1 \mapsto 2, 2 \mapsto 1, 3 \mapsto 1\})$
- Polígonos são figuras geométricas planas limitas por linhas poligonais fechadas, por exemplo, um hexágono é um polígono de seis lados. Um polígono é denominado regular se todos os seus lados e todos os seus ângulos forem congruentes.

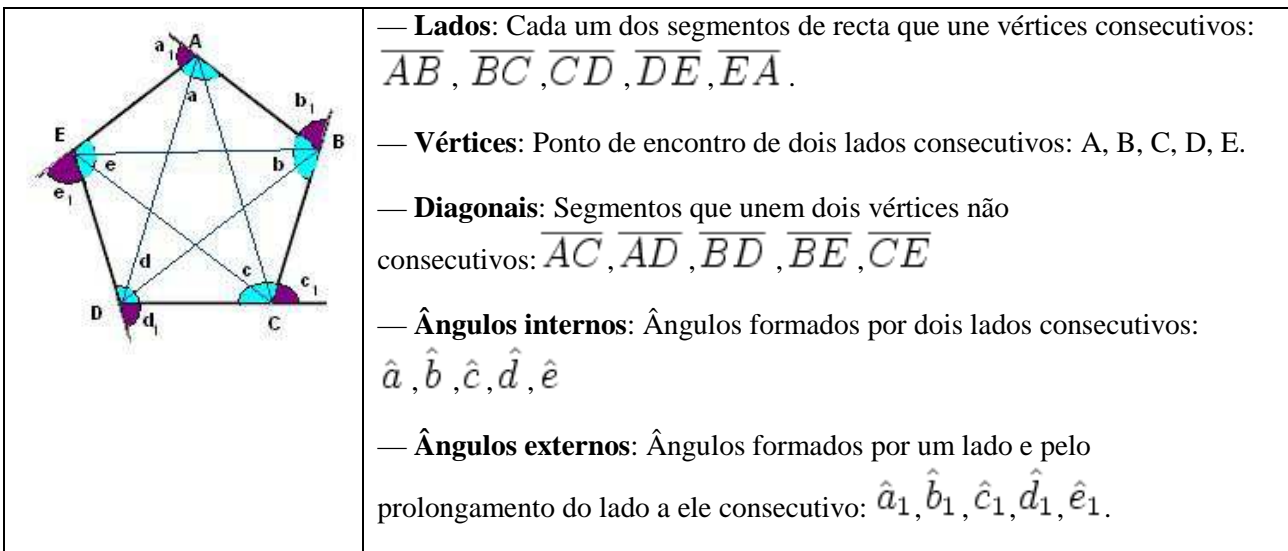


Fig. 1: Exemplos de um polígono regular (pentágono)

Considere a seguinte representação de um polígono em VDM++ (Figura 2)

```
class Poligono
types
  public ponto:: x: nat1
                y: nat1;
instance variables
  public lados : seq of ponto;
  public angulos: map ponto to nat1;
end Poligono
```

Fig. 2: Representação de um grafo em VDM++

- Escreva uma função, em VDM++, que determine se um dado polígono tem os ângulos internos todos iguais.
- Escreva o corpo e pré-condição de uma função (*Remove*), em VDM++, que remova um vértice de um polígono mantendo-o fechado (fig 3). (Nota: Não actualize a informação relativa aos ângulos).

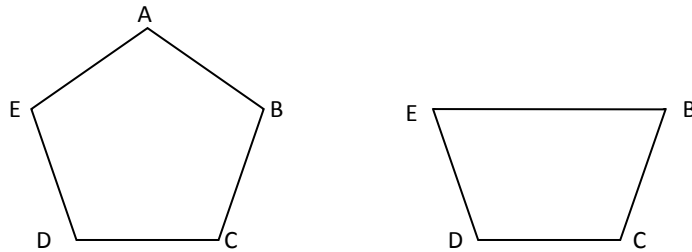


Fig. 3: Exemplo de remoção de um vértice (A) de um pentágono

- c. Escreva uma função (*Diagonais*), em VDM++, que retorne o conjunto das diagonais (Fig. 1) de um polígono. Para isso, considere a definição de um tipo adicional (*segrecta*) e complete a especificação em baixo (Fig. 4).

```
class Poligono
  types
    public segrecta :: ponto
                    ponto;
  operations
    public Diagonais: () ==> set of segrecta
    // escreva "aqui" o corpo da função
end Poligono
```

Fig. 4: Exemplo de remoção de um vértice (A) de um pentágono

3. Uma árvore é um conjunto de segmentos de recta ligados pelos seus extremos e sem ciclos fechados (figs 5 e 6).

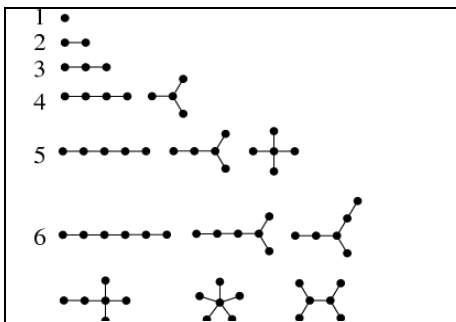


Figura 5: Exemplos de árvores

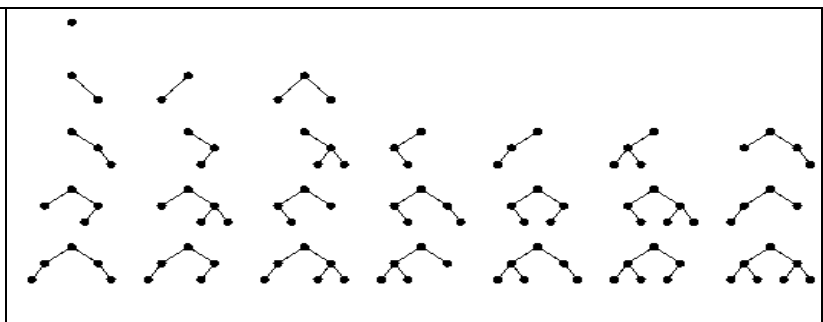


Figura 6: Exemplos de árvores binárias

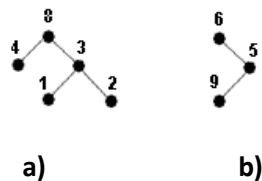


Figura 7: a árvore b) é sub-árvore da árvore a)

```
class Tree
  instance variables
    public root : nat1;
    public edge : map nat1 to nat1;
end Tree
```

Figura 8: Representação de uma árvore (*Tree*) em VDM++

- a. Especifique uma função, em VDM++, que retorne o conjunto das folhas de uma árvore (representada em VDM++ pela figura 8).
- b. Especifique uma função *Insertedge*, em VDM++, que insira uma aresta numa árvore.

- c. Se a árvore de input da função *Insertedge* (da alínea anterior) for binária (Figura 6), após executar essa mesma função, a árvore continua binária? Justifique. Em caso negativo, escreva a pré-condição necessária e suficiente para garantir que a árvore resultante da inserção continua binária.
- d. Assuma que existe uma função *Distance(f: nat1, a:Tree)* que retorna a distância da folha *f* à raiz da árvore *a*. Especifique uma função, em VDM++, que determine a altura de uma árvore (representada em VDM++ pela fig 8). A altura de uma árvore é a maior distância existente entre uma folha e a raiz (*root*) da árvore.

4. Árvores com raiz balanceadas são grafos conexos sem ciclos (acíclicos) onde as folhas estão a uma distância *d* ou *d+1* da raiz. Uma árvore pode ser representada em VDM++ por:

```

class Tree
types
public edge:: no1: nat1
           no2: nat1;
instance variables
  public Root : nat1:=0;
  public Edges : set of edge := {};

--inv1 Uma árvore com n nós tem n-1 arestas (grafo acíclico).
--inv2 A raiz da árvore (Root) tem que ser um nó da árvore.
--inv3 A árvore tem que ser conexa.
--inv4 A árvore é balanceada (folhas a uma distância d ou d+1 da raiz)
--inv5 entre dois nós arbitrários da árvore existe um único caminho.

end Tree

```

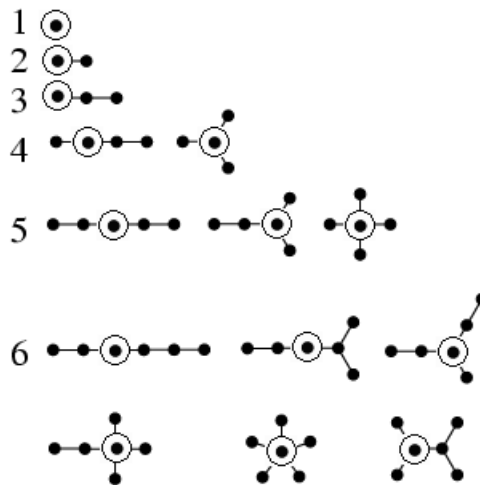


Fig. 9: Árvores balanceadas com 1, 2, ..., 6 nós

- a. Formalize os invariantes *inv2* e *inv4*. Considere que existe uma função *dist(nat1)* que calcula a distância de um ponto à raiz da árvore
- b. Escreva uma função (*Insert*) que acrescente uma aresta a uma árvore. (nota: considere a situação em que a raiz da árvore poderá ter que ser alterada para que a árvore permaneça balanceada)
- c. Determine se uma dada árvore (representada pela classe *Tree* em VDM++) é uma árvore binária. Numa árvore binária, cada nó tem no máximo dois filhos.

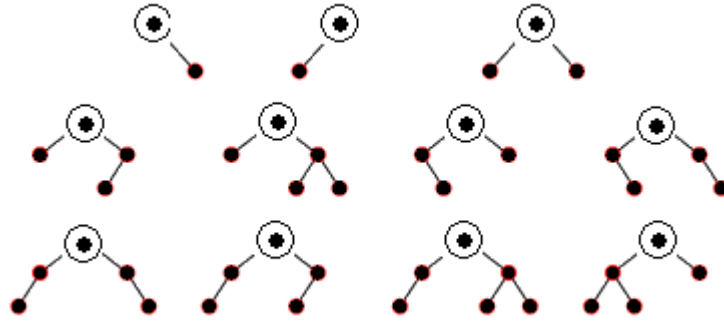


Fig. 10: Exemplos de árvores binárias

- d. Escreva uma função (*Mutation*) que transforme uma árvore (representada pela classe *Tree*) no conjunto de todos os caminhos da raiz às folhas (representada pela classe VDM++, *TreePaths*, descrita de seguida).

```
class TreePaths
  type path: seq of nat1;

  instance variables
    public paths: set path;
end TreePaths
```

5. Um grafo dirigido conexo G (Fig. 21a) pode ser representado pelo par (V, A) , em que

V – é um conjunto de vértices ou nós e

A – é um conjunto de arestas com um vértice de origem e um conjunto de vértices destino.

Em VDM++, um grafo dirigido conexo pode ser formalizado por:

```
class Graph
  instance variables
    public V : set of nat1:={};
    public A : map nat1 to set of nat1:= {|->};

  --inv1 Uma aresta só pode ligar vértices do grafo a que pertence.
  --inv2 Um grafo tem que ser conexo, i.e., não podem existir dois vértices sem
  --caminho de ligação entre eles (independentemente do sentido das arestas).
end Graph
```

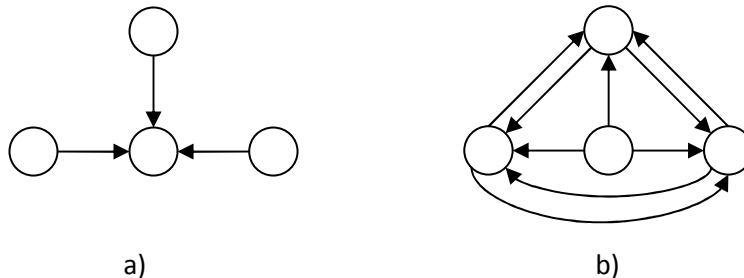


Fig. 21: a) Grafo dirigido conexo. b) Grafo complementar

- Formalize os invariantes *inv1* e *inv2*.
- Formalize a função *CompGraph* que determina o grafo H (Fig. 21b) complementar de um grafo G . Um grafo H complementar de G tem o mesmo conjunto de vértices de G e o conjunto de arestas não presentes em G . Formalize também a pós-condição desta função da forma mais abstracta possível.

- c. Usando a função *CompGraph* definida em cima formalize uma função *SubGraph* que determina se um grafo H é subgrafo de G.
- d. Formalize uma função *TransitiveClosure* que determina o conjunto dos vértices que é possível atingir (com *n* passos) a partir de um dado vértice inicial (considere o sentido das arestas).
- e. Considere a especificação de uma função *InsertEdge* que adiciona uma aresta a um grafo dirigido:

```
public InsertEdge: nat1 * nat1 ==> Graph
InsertEdge(n1, n2) ==
(
  dcl V_old: set of nat1 := V;
  V := V union {n1, n2};
  A := A ++ if n1 in set V_old then {n1 |-> A(n1) union {n2}}
           else {n1 |-> {n2}};
  return self;
)
```

- i. "A especificação apresentada não é consistente". Comente a afirmação.
- ii. Formalize uma pré-condição para a função *InsertEdge* que garanta a consistência do modelo.

6. Um "cut-vertex" de um grafo conexo é um nó do grafo cuja remoção torna o grafo desconexo. Um grafo sem "cut-vertices" é um grafo bi-conexo (Figura 12).

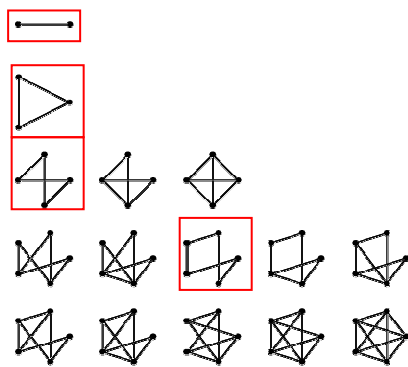


Figura 12: Exemplos de grafos bi-conexos

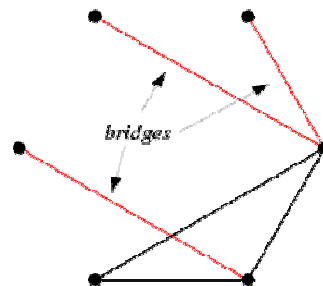


Figura 13: Exemplo de grafo com pontes (bridges)

```
class Graph
  instance variables
    public edge : map nat1 to nat1;
end Graph
```

Figura 14: Representação de um grafo em VDM++

- a. Especifique uma função, em VDM++, que retorne o nº de vértices de um grafo (especificado pela classe *Graph* na fig. 14) é bi-conexo.
- b. Especifique o corpo da função *removearesta*, em VDM++, que remova uma aresta de um grafo conexo.
- c. Uma **ponte** (*bridge*) de um grafo conexo é uma aresta (*edge*) do grafo cuja remoção torna o grafo não conexo (Figura 13). Especifique uma função *pontes*, em VDM++, que devolva todas as pontes de um grafo. Escreva também, em VDM++, a pós-condição desta função.