U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Suporte para geração de testes a partir de máquinas de estados

Conceção e Análise de Algoritmos

28 de Abril de 2014

Grupo E:

Henrique Ferrolho – 120509079 – henriqueferrolho@gmail.com
Maria Marques – 120509104 – mariajoao.rmarques@gmail.com

Índice

1. Introdução

No âmbito da unidade curricular Conceção e Análise de Algoritmos do curso Mestrado Integrado em Engenharia Informática e Computação foi-nos proposto a realização de um problema onde procederíamos à análise de uma máquina de estados. Para isso recorremos à criação de ficheiros texto que representam máquinas de estados, que por sua vez, vão ser representadas por grafos.

O primeiro objetivo foi verificarmos se as máquinas de estados são válidas; o segundo foi determinar o caminho mais curto para atingir um determinado estado; o terceiro foi encontrar os caminhos que passassem por todos os estados com custo mínimo; o quarto (opcional) foi determinar se a máquina de estados era equivalente a uma outra fornecida.

Neste relatório iremos explicar o contexto do problema, contemplando a sua formalização e respetivos esquemas de forma a facilitar a compreensão do mesmo.

2. Perguntas e respostas

Q: Como vão representar as máquinas de estados?

A: Recorrendo a grafos dirigidos.

Q: Quais irão ser os elementos dos grafos?

A: Os grafos vão ter:

- Vértices (*vertexes*) que representam estados;
- Arestas (*edges*) que representam transições entre estados, causadas por eventos;
- Um único estado inicial;
- Pelo menos um estado final;
- Todos os vértices devem ser atingíveis;
- Não pode haver transições com o mesmo evento, mesmo estado de origem e diferentes estados de destino.

Q: O que é um caminho ponta-a-ponta e o que é que representa?

A: É um caminho que começa no estado inicial e termina num estado final. Representa uma execução possível da máquina de estados.

Q: Como vão ser representados os caminhos ponta-a-ponta?

A: Vão ser representados por uma sequência alternada de nomes de vértices (estados) e arestas (eventos).

Q: Como é dado o comprimento de um caminho?

A: É igual ao número de arestas.

3. Especificações da aplicação

3.1. Dados

Construção de um grafo dirigido, $G = (T, E)$, em que:

- Vértices (T): corresponde aos estados da máquina de estados;
- Arestas (E): corresponde às transições entre estados causadas por eventos.

3.2. Input

Máquina de estados: $G = <T, E>$, onde:

- T - conjunto de estados;
- Ei,j - evento da tarefa Ti para a tarefa Tj;

3.3. Outputs

3.3.1. Tarefa 1

Uma notificação sobre a validade da máquina de estados. Caso não seja válida é também mostrada uma lista das razões pela qual a máquina é inválida.

3.3.2. Tarefa 2

$S = (Ti, <Ti, Eij>, <Tf, \_>)$

Ti - é o estado inicial

Tf - é o estado final

Função objetivo: $min|S|$

É mostrada uma lista de estados e de eventos, representando o caminho mais curto até ao estado especificado.

3.3.3. Tarefa 4

Uma notificação sobre a equivalência de máquinas de estados. Caso as máquinas não sejam equivalentes é também mostrada uma lista das razões pelas quais não o são.

3.4. Restrições

3.3.1. Tarefa 1

- Um só estado inicial;
- Pelo menos um estado final;
- Todos os estados têm de ser atingíveis;
- Não pode haver transições com o mesmo evento, mesmo estado de origem e diferentes estados de destino.

3.3.2. Tarefa 2

- O estado de destino tem de existir e tem de ser atingível.

3.3.3. Tarefa 4

- As máquinas têm de ter os mesmos estados iniciais, finais, o mesmo número de estados, o mesmo número de transições em cada estado e as mesmas transições para os mesmo estados.

4. Principais algoritmos implementados

4.1. Tarefa 1

Para testarmos a validade das máquinas de estados, temos que verificar que existe um e um só estado inicial, pelo menos um estado final, que todos os estados são atingíveis e que não pode haver transições com o mesmo evento, mesmo estado de origem e diferentes estados de destino.

Para verificarmos todos estes requisitos fizemos o seguinte:

- No construtor do grafo ao carregar os vértices, sempre que um vértice é um estado inicial ou final é incrementada uma variável associada ao grafo. No final a variável correspondente ao número de estados iniciais tem de ser igual a 1 e a de estados finais tem de ser maior ou igual a 1;
- Para verificar se todos os estados são atingíveis usámos o método *getSources()*, que retorna um vetor com os estados que não têm arestas incidentes. Para o grafo ser válido, o vetor retornado tem que ser vazio ou conter apenas o estado inicial. O método *getSources()* percorre todo o *vertex set* do grafo, portanto apresenta complexidade temporal de:

$$O(|T|)$$

- Para fazer a última verificação é mais difícil: percorremos cada estado do grafo e asseguramos que não há transições com o mesmo evento para estados diferentes, ou seja, asseguramos que o grafo não é um NFA. Como esta fase requer verificar novamente todos os vértices e, no pior dos casos, para cada um deles, verificar todos os outros vértices, a complexidade temporal será:

$$O(|T|.|T|)$$

No final, a validação de uma máquina de estados, segundo este algoritmo, terá no pior dos casos, uma complexidade temporal de:

$$O(|T|.|T|.|T|)$$

Pseudocódigo para validar um grafo:

```
bool Graph::isValid() {
	bool initStateFlag = numInitStates != 1;
	bool finalStateFlag = numFinalStates < 1;
	bool unreachableStateFlag = |getSources()| > 1  ||
		(|getSources()| == 1 && sources[0] != initialState);
	bool nfaFlag = getNfaFlag();

	bool invalidStateMachine = initStateFlag ||
		FinalStateFlag || unreachableStateFlag || nfaFlag;
}
```

4.2. Tarefa 2

Para encontrar o caminho mais curto desde o estado inicial até um determinado estado utilizamos o método *getPath()*, que por sua vez usa o método *unweightedShortestPath()* que usámos nas aulas práticas. O algoritmo por trás deste método calcula o caminho mais curto desde um vértice de origem para todos os outros vértices. Depois basta verificar o caminho do vértice de interesse.

O método *unweightedShortestPath()* apresenta:

- Complexidade temporal:

$$O(|T|+|E|)$$

- Complexidade espacial:

$$O(|T|)$$

Assim sendo, a complexidade temporal de *getPath()*, no pior dos casos, será de:

$$O(2.|T|+|E|)$$

4.3. Tarefa 4

Para determinar a equivalência entre duas máquinas de estados, é feita uma *depth-first search* (DFS) a cada um dos dois grafos a serem comparados. Posteriormente, são analisados os seguintes parâmetros:

- Se um estado é inicial e o outro não;
- Se um estado é final e o outro não;
- Se o número de transições do estado a analisar é diferente do estado do outro grafo;
- Se algum evento de transição do estado a analisar não corresponde a nenhum evento de transição do outro estado.

Se algum destes parâmetros se verificar podemos concluir que os grafos não são equivalentes.

Pseudocódigo para determinar equivalência entre grafos:

```
inspectGraphsEquality(Graph* g1, Graph* g2) {
	vector<State> dfs1 = g1.dfs();
	vector<State> dfs2 = g2.dfs();

	for (i = 0; i < dfs1.size; i++) {
		if (dfs1[i].isInit() != dfs2[i].isInit())
			difInitialStateFlag = true;
		if (dfs1[i].isFinal() != dfs2[i].isFinal())
			difFinalStateFlag = true;

		dfs1StateNumTrans = dfs1[i].getTransitions.size;
		dfs2StateNumTrans = dfs2[i].getTransitions.size;
		if (dfs1StateNumTrans != dfs2StateNumTrans)
			difTransitionsFlag = true;
		else {
			for (j = 0; j < dfs1StateNumTransitions; j++) {
				Transition trans = dfs1[i].getTransitions[j];

				bool foundEquivalentTransition = false;
				for (k = 0; k < dfs2StateNumTransitions; k++) {
					if (trans == dfs2[i].getTransitions[k]) {
						foundEquivalentTransition = true;
						break;
					}
				}

				if (!foundEquivalentTransition) {
					difTransitionsFlag = true;
					break;
				}
			}
		}

		if (difInitialStateFlag &&
			difFinalStateFlag && difTransitionsFlag)
			break;
		}
	}
}
```

Sejam os dois grafos a serem comparados os grafos *G1* e *G2*:

- *G1 = <T1, E1>*
- *G2 = <T2, E2>*

Para determinar a equivalência entre *G1* e *G2*, é necessário fazer uma *DFS* a cada um, o que corresponde a uma complexidade temporal de:

$$O(|T1|+|T2|)$$

De seguida, para cada estado *E* pertencente a *E1*, verifica-se se é equivalente ao estado *E'* pertencente a *E2*. O teste verifica se:

- *E* é estado final enquanto que *E*' não é, e vice-versa;
- Idem para se verificar o estado inicial;
- O número de transições associado ao estado *E* é diferente do número de transições associado a *E*'.

Considere-se que um estado *E* pertencente a uma máquina de estados tem, associadas a si, um conjunto de *Te* transições. Assim, para cada estado, a verificação das transições associada a este apresenta complexidade temporal de:

$$O(|Te1|.|Te2|)$$

A complexidade temporal total para determinar a equivalência entre máquinas de estados será então, no pior dos casos:

$$O(|T1|+|T2|+|T1|.|Te1|.|Te2|)$$

5. Lista de casos de utilização

É esperado que o utilizador use a aplicação tanto como uma ferramenta de testes, como um visualizador de máquinas de estados.

Em baixo encontra-se um diagrama das funcionalidades disponibilizadas pela aplicação desenvolvida.

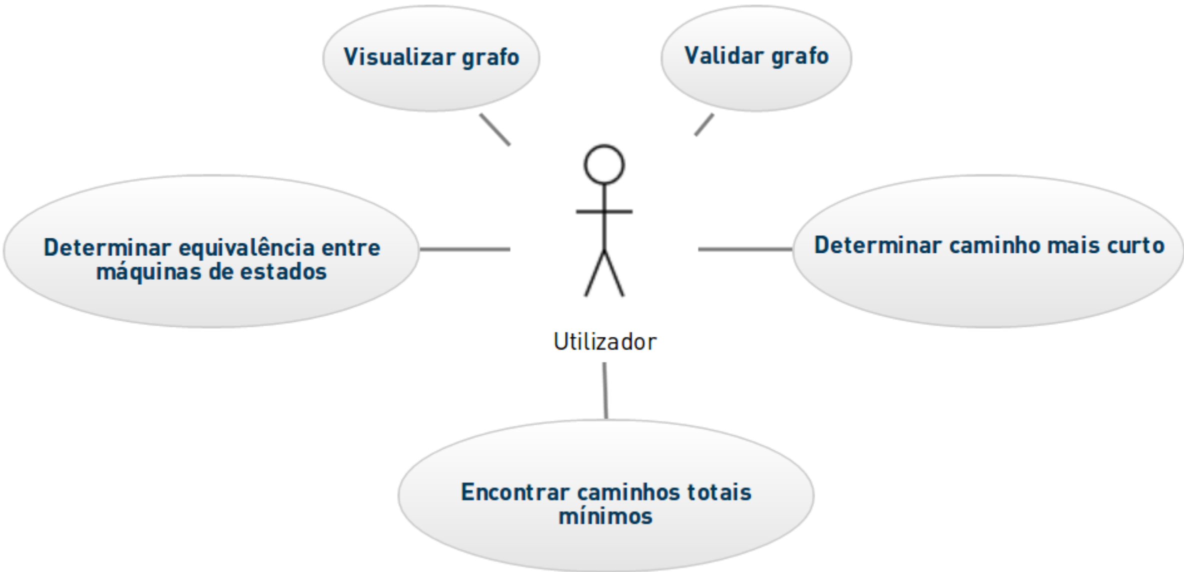

Fig. 1. *Diagrama de casos de utilização.*

A interface da aplicação foi desenvolvida para a linha de comandos. Para facilitar a navegação pela aplicação, todos os menus, sub-menus e respetivas solicitações de input para navegar entre estes estão implementados e encapsulados numa classe que gere toda a interface da aplicação em "MenusInterface.cpp".

De seguida é apresentado um diagrama relativamente aos diferentes menus e sub-menus da aplicação:

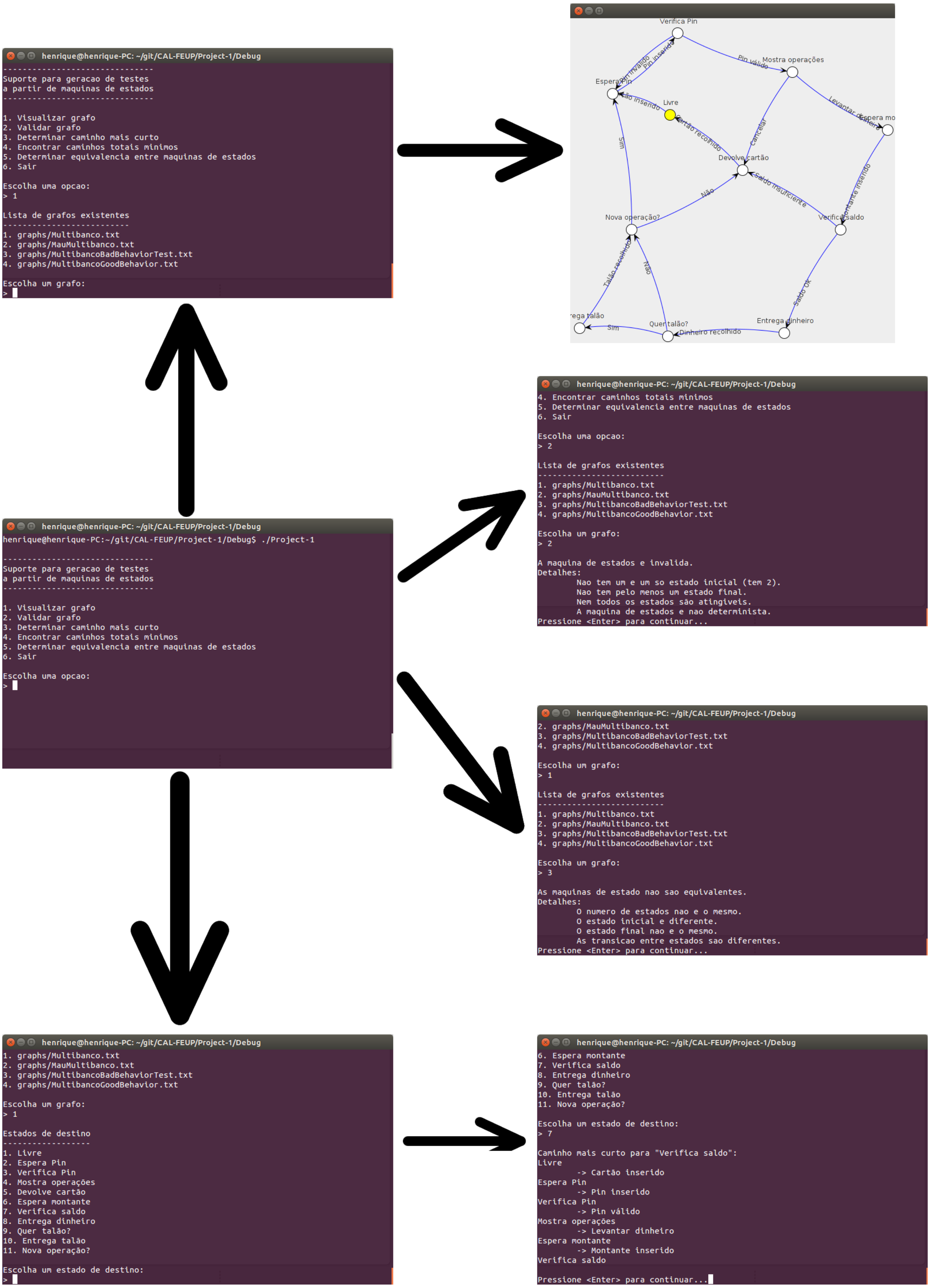

Fig. 2. Diagrama da interface da aplicação.

6. UML

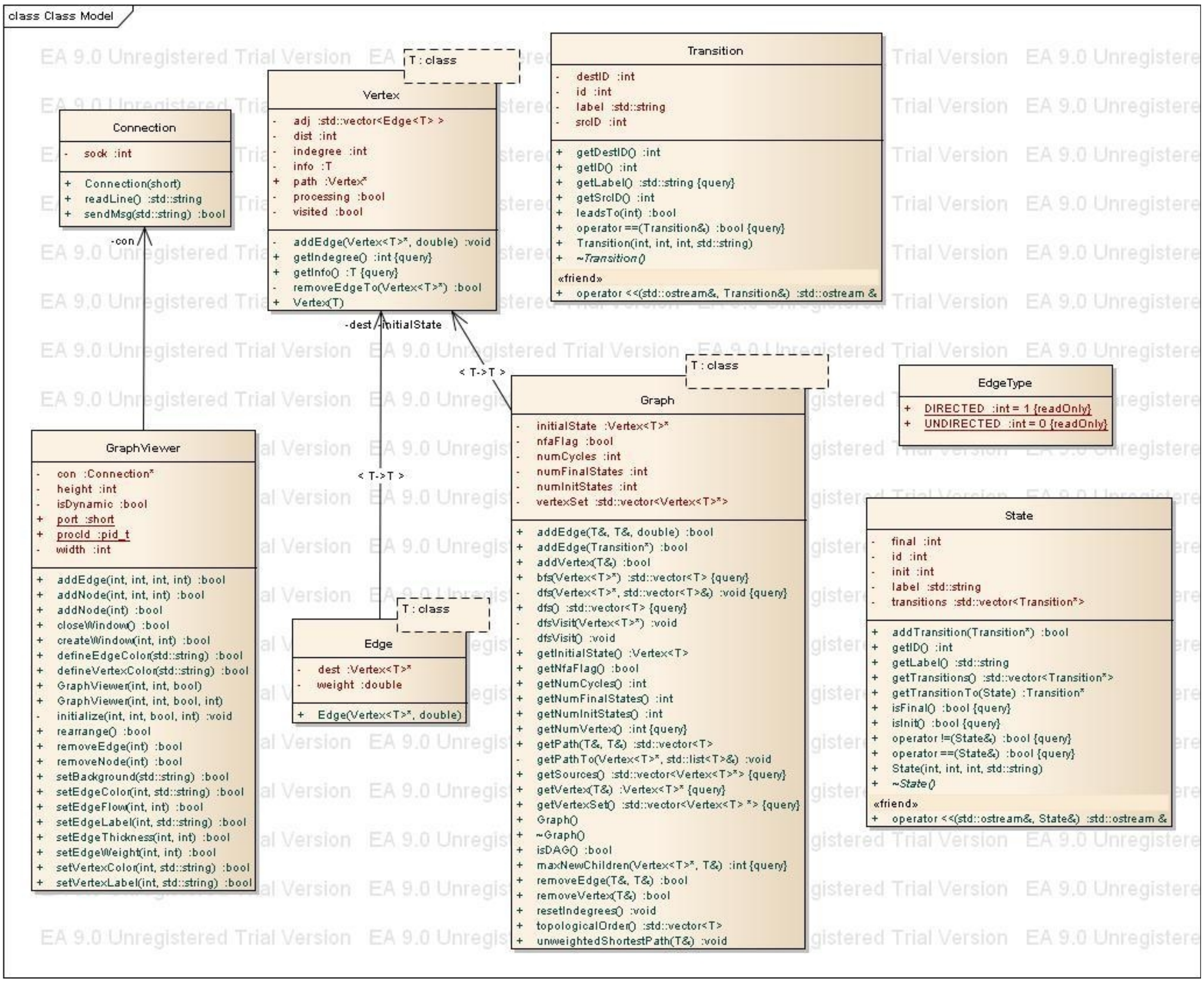

Fig. 3. Diagrama de classes UML.

7. Conclusão

7.1. Dificuldades

A principal dificuldade sentida foi na terceira tarefa: é-nos pedido encontrar um conjunto de caminhos ponta-a-ponta de comprimento total mínimo, cobrindo todas as transições. O enunciado não parece fazer sentido e após falar com a professora Ana Paula Rocha decidimos que não seria esperado submeter esta tarefa e que iríamos falar sobre ela com mais detalhe brevemente.

7.2. Contribuição no projeto

Henrique Ferrolho – 80%

Maria João Marques – 20%