**Ordenação de Tarefas**

**Relatório**



**Mestrado Integrado em Engenharia Informática e Computação**

**Concepção e Análise de Algoritmos**

José Carlos Rocha Lima – ei10012

Nuno Filipe Dinis Cruz – ei10082

Vasco Manuel Pérola Filipe – ei10031

**28 de Abril de 2014**

Índice

Introdução 3

Implementação 4

Estrutura do programa 6

Diagrama de classes 7

Lista de casos de utilização 7

Conclusão 8

Bibliografia 9

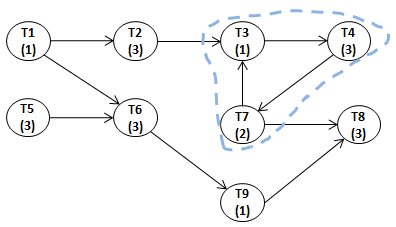
# Introdução

No ambito da disciplina de Concepção e Análise de Algoritmos, temos como primeiro trabalho prático a elaboração de um projeto de programação em C++ de algoritmia em grafos e técnicas de concepção de algoritmos.

Pretende-se desenvolver um programa capaz de resolver eficientemente a ordenação de tarefas, em que temos uma lista de tarefas a realizar, com dependências e prioridades numéricas, podendo existir várias tarefas com a mesma prioridade.

# Implementação

Inicialmente, usámos o algoritmo de Tarjan para a agregação de tarefas que fazem parte do mesmo componente fortemente conexo numa única macro-tarefa de prioridade igual à média das prioridades.



O algoritmo consiste numa pesquisa do grafo em profundidade, sendo que sempre que um vértice é visitado, é adicionado a uma pilha de vértices, e no fim da visita, é retirado da mesma. Assim, se durante a pesquisa for encontrado um vértice já na pilha, pode-se concluir que foi encontrado um ciclo. Nesse caso, o programa vai entrar num loop, retirando vértices da pilha, um a um, até encontrar a posição em que se encontra. Todos os elementos da pilha que são retirados constituem o ciclo, pelo que são adicionados a um vector com esse mesmo nome.

O algoritmo de Tarjan’s foi implementado para combater o problema de existirem vários ciclos no mesmo, pois quando eram retiradas posições da pilha, era comprometida a continuação do algoritmo. Seguindo essa linha de pensamento, criamos uma pilha auxiliar, que armazena os vértices pertencentes ao ciclo, e depois de o processar, adiciona-os de novo à pilha principal, para a função poder continuar o seu percurso sem erros.

A partir desse grafo acíclico, procurámos todas as tarefas sem dependências pendentes, e agrupámo-las numa lista. É então executada a tarefa de maior prioridade (retirando-a da lista), sendo agora procuradas as tarefas que entretanto ficaram disponíveis, e adicionadas à lista. O algoritmo acaba quando a lista ficar vazia, estando todas as tarefas ordenadas. Foi criado um booleano *visited* para cada tarefa de modo a evitar que cada uma seja pesquisada mais que uma vez, tornando o algoritmo mais eficiente - **O (n log n).**

Dados de Entrada:

Grafo<Tarefa,Dependências>, Tarefa<Nome, Prioridade>

Dados de Saída:

Lista<Tarefa>. Esta lista é ordenada em função da execução das tarefas, sendo o primeiro elemento da lista a primeira tarefa a ser executada.

Os dados de entrada são lidos a partir do ficheiro de texto “*graph.txt”*. A estrutura do ficheiro de texto é a seguinte:

- *{Prioridade} {Nome da Tarefa}* para criar os vértices, feito a seguir a *“Vertex:”*;

- *{Número do Vértice de origem} -> {Número do Vértice de destino} para criar as arestas, feito a seguir a “Edge:”* . Os números correspondem à ordem de criação dos vértices no passo anterior.

# Estrutura do programa

Foi usada a estrutura de grafos das aulas práticas como base do projecto. Os métodos implementados sobre a estrutura foram os seguintes:

**Vertex:**

- *void appendAdj(vector<Edge<T> > vec)*, função que recebe uma lista de arestas e as adiciona ao vector de adjacências do vértice. É usada na função *unifyCycles()*;

- *vector<Edge<T> > getAdj()*, método *get* do vector de adjacências;

**Graph:**

- *void findCycles()*, função que percorre o grafo e encontra os ciclos (tarefas que fazem parte do mesmo componente fortemente conexo);

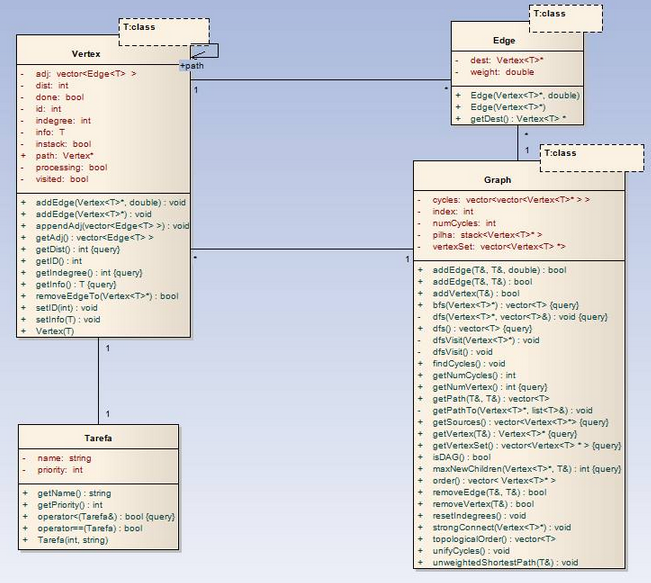
- *void strongConnect(Vertex<T> \*v)*, função auxiliar de *findCycles();*

- *void unifyCycles()*, função que trata de unificar os ciclos, ou seja, tornar as diferentes tarefas de um ciclo numa única macro-tarefa com prioridade igual à média das prioridades das tarefas componentes e mantendo as dependências para fora do grupo;

*- vector< Vertex<T>\* > order()*, função que ordena as tarefas por dependências e prioridade.

Foi ainda criada uma estrutura de dados adicional, a classe *Tarefa* cujo construtor tem como parâmetros uma prioridade e um nome, estrutura esta a ser usada como *Info* de um vértice. Nesta estrutura foi ainda feito o *overload* do operador == para comparação de tarefas, usado na estrutura de grafos, e do operador < que é usado na função *sort* de vectores.

## Diagrama de classes



Um grafo é composto por várias arestas e vértices, sendo estes uma Tarefa.

## 

## Lista de casos de utilização

**-** Leitura de grafo a partir de um ficheiro de texto;

- Desenho do grafo;

- Identificação de ciclos de tarefas;

- Substituição de ciclos de tarefas por super-tarefas;

- Ordenação de tarefas.

# Conclusão

Uma boa gestão do tempo permitiu-nos desenvolver o trabalho por completo, sendo que não foram encontrados erros na execução do mesmo. Decidimos melhor trabalhar em conjunto, pelo que todos trabalharam um pouco em cada tarefa, facilitando a resolução dos problemas com que nos deparámos.

O trabalho decorreu sem grandes entraves, com a excepção da função responsável por encontrar ciclos, principalmente na manipulação de pilhas.

# Bibliografia

[*http://en.wikipedia.org/wiki/Tarjan's\_strongly\_connected\_components\_algorithm*](http://en.wikipedia.org/wiki/Tarjan's_strongly_connected_components_algorithm)

[*http://www.cplusplus.com/*](http://www.cplusplus.com/)