

**Algoritmos e Estruturas de Dados**

**Job Selection**

**Professores:**

Tomás Oliveira e Silva ([tos@ua.pt](mailto:tos@ua.pt))  
Pedro Lavrador ([plavrador@ua.pt](mailto:plavrador@ua.pt))

**Pedro** **Sobral**, 98491 – XX%

**André Freixo**, 98495 – XX%

**Marta Fradique**, 98626 – XX%

XX/01/2021

Índice

[**1 - Introdução** 3](#_Toc59843219)

[**2 - Introdução ao Problema** 4](#_Toc59843220)

[2.1 - Resumo 4](#_Toc59843221)

[2.2 – Compilação e Corrida 4](#_Toc59843222)

[FALAR DO CODIGO DO STOR? 5](#_Toc59843223)

[**Função Recursiva** 6](#_Toc59843224)

[Conclusão 7](#_Toc59843225)

[Bibliografia 8](#_Toc59843226)

# **1 - Introdução**

No âmbito da unidade curricular de AED, foi-nos apresentada a realização deste trabalho prático, sendo este relatório o resultado do código “Job Selection Problem”. Todo o código fonte e informações deste trabalho prático podem ser encontradas neste [repositório do GitHub](https://github.com/TheScorpoi/AED_Trabalho01)1 (mais informações, ler README.md do repositório).

Dado um número de tarefas, e um número de programadores, o programa implementado em C, de forma genérica, através de um algoritmo recursivo feito por nós, retorna a melhor maneira de realização do trabalho, ou seja, a maneira em que se consegue obter mais lucro, sem que ocorra sobreposição de programadores.

A linguagem de programação C, é uma linguagem muito poderosa, pois dá ao programador um controlo íntegro de todo o processo programado, sendo uma linguagem onde o programador tem de lidar com todos os pormenores, torna-se consideravelmente eficiente e otimizada. O algoritmo principal por nós desenvolvido, irá usar a técnica da recursividade, pois dividirmos o problema principal em subproblemas é uma maneira eficiente de chegar a uma resolução viável.

Para a resolução do problema imposto, irá ser elaborado código C job\_selection.c, pois trata-se de uma linguagem com tempos de execução extraordinariamente baixos comparado a outras linguagens, por exemplo, Java e Python. Os resultados obtidos serão guardados num ficheiro .txt dentro de uma pasta com o nMec. Para a análise dos resultados, iremos usar o Matlab, e o nosso conhecimento adquirido ao longo do semestre noutras cadeiras em que o uso desta ferramenta é primordial. Esperamos também criar um script em Shell, para correr o programa job\_selection.c, com o objetivo de conseguir utilizar todos os cores do processador.

Com a realização deste trabalho prático, esperamos veemente alargar os nossos conhecimentos em C, e principalmente em implementar algoritmos eficientes, e otimizados, para que possamos resolver os problemas propostos de maneira otimizada. Esperamos também conseguir concluir com êxito todos os objetivos que são propostos no início (em comentário) do programa job\_selection.c. Para uma abordagem mais eficaz está em base de discussão a implementação da resolução do problema também em Java, no entanto, por motivos de tempo e experiência a programar não sabemos se conseguiremos alcançar este objetivo ambicioso.

1Por motivos de privacidade o repositório encontra-se privado, para visualização, é favor entrar em contacto com os autores do trabalho prático.

# **2 - Introdução ao Problema**

## 2.1 - Resumo

Sendo direto e conciso, o que se pretende obter é maneira mais eficiente, quer computacionalmente, quer a nível de *profit*, para as *Tarefas* e para os *Programadores* dados, ou seja, através do algoritmo recursivo implementado por nós, iremos percorrer todas as possibilidades viáveis, através de uma pesquisa exaustiva das mesmas.

Incompleto….

## 2.2 – Compilação e Corrida

Para compilar o programa é necessário à partida ter um compilador de C instalado na máquina, por exemplo o *gcc*.

Posto isto, para compilar basta:

*cc -Wall -O2 job\_selection.c -o prog*

Para correr a compilação basta:

*./prog NMec T P I*

*,* onde NMec é o número mecanográfico do aluno

, o T é o número de Tarefas a realizar

, o P o número de Programadores

, e o I, um ou zero, o ignore\_profit

# FALAR DO CODIGO DO STOR?

# **Função Recursiva**

Implementamos uma função recursiva, pois achamos ser o método mais eficaz para a realização do problema em questão. A nossa função aceita como argumentos, um *problema*, e o nível da iteração, kinda arvore idk.

À partida, a nossa função tem duas exceções, uma para quando o i é igual a zero, e outra para quando o *i* é igual ao número de tarefas do problema em questão. Quando começa, inicializamos as variáveis *total\_profit* e *best\_total\_profit* com os valores 0 (zero), e através de um ciclo *for*, inicializamos o array *busy* todo a -1 (menos um), onde -1, significa o programador estar livre. Quando acaba, ou seja, o *i* é igual ao número de *Tarefas*, fazemos a comparação do *total\_profit* com o best\_total\_profit, e caso o primeiro seja maior que o segundo, o best\_total\_profit fica atualizado com o valor do total\_profit, ainda incrementamos a variável *terminal\_cases*, para mais tarde sabermos quantas possibilidades viáveis tivemos.

Neste momento, que as exceções já estão tratadas, podemos começar a usar as maravilhas da recursividade. Temos duas opções: avançar sem atribuir a *Tarefa*, ou tentar incluir a *Tarefa*, se conseguir avança.

Para avançar sem atribuir a tarefa, basta chamar a própria função, no nível seguinte, *i+1.* Assim, estamos a não atribuir esta *Tarefa*, sdasdasgfdg

Para tentar incluir a *Tarefa*, temos de executar um ciclo *for*, de zero até ao número de programadores, e ver se há algum livre, isto é, se a data até que o *Programador* está ocupado, *ending\_date*, for menor que a data de começo da *Tarefa, starting\_date.* Caso isso se verifique, entramos no bloco de código que a condição *if* acopla, e aqui são criadas duas variáveis de cariz temporário, o *profit\_tmp* e o *busy\_tmp,* estas variáveis armazenam em si os valores do *total\_profit* e do *busy*, respetivamente. Estas duas variáveis são importantíssimas, porque: xxxxxxxx

É necessário atualizar algumas variáveis, a variável *busy* para o programador *j*, irá ficar com o valor da data final (*ending\_date*) da *Tarefa* *i. À variável* total\_profit é somado o *profit* da *Tarefa* em questão. E a variável *tas*k da *Tarefa i,*  fica atribuída ao programador *j*.

Chamamos a própria função agora, no nível seguinte, xxxxxxxxxxxx

Nesta zona da função voltamos a atribuir às variáveis *total\_profit e busy* os valores que foram guardados anteriormente. xxxxxxxxxxxxx

# Conclusão

# Bibliografia