Estrutura de Dados

Avançadas

Engenharia De Sistemas

Informáticos

**Nome:** Henrique da Silva Araújo Neto **Nº** 16626

Escola Superior De Tecnologia

Instituto Plitécnico do Cávado e Do Ave

# Índice

[Índice 1](#_Toc101208457)

[Introdução 2](#_Toc101208458)

[Propósitos e Objetivos 4](#_Toc101208459)

[Estrutura de dados 4](#_Toc101208460)

[Fase 1: 4](#_Toc101208461)

[Testes Realizados 5](#_Toc101208462)

[Fase 1: 5](#_Toc101208463)

[1. Definição de uma estrutura de dados dinâmica para a representação de um *job* com um conjunto finito de n operações: 5](#_Toc101208464)

[2. Armazenamento/leitura de ficheiro de texto com representação de um job; 6](#_Toc101208465)

[3. Inserção de uma nova operação: 8](#_Toc101208466)

[4. Remoção de uma determinada operação: 9](#_Toc101208467)

[5. Alteração de uma determinada operação: 10](#_Toc101208468)

[6. Determinação da quantidade mínima de unidades de tempo necessárias para completar o job e listagem das respetivas operações: 11](#_Toc101208469)

[7.Determinação da quantidade máxima de unidades de tempo necessárias para completar o job e listagem das respetivas operações: 12](#_Toc101208470)

[8. Determinação da quantidade média de unidades de tempo necessárias para completar uma operação, considerando todas as alternativas possíveis: 14](#_Toc101208471)

[Funções de ajuda 15](#_Toc101208472)

[Git 17](#_Toc101208473)

[Execução 17](#_Toc101208474)

[Conclusão 18](#_Toc101208475)

[Bibliografia 19](#_Toc101208476)

# Introdução

Para mostrar os conhecimentos adqueridos relativos a definição e maniplução de estruturas de dados dinâmicas na linguagem de programação C, foi pedido aos alunos de Engenharia de Sistemas Informáticos (LESI) o desenvolvimento de uma solução digital para um problema de escalonamento denominado *Flexible Job Shop Problem (FJSSP).* A solução deverá permitir gerar uma proposta de escalonamento para a produção de um produto envolvendo várias operações e a utilização de várias máquinas, minimizando o tempo as unidades de tempo necessário na sua produção (*makesplan*).

Para a realização da solução, esta foi dividida em duas fases:

**Fase 1:**

1. Definição de uma estrutura de dados dinâmica para a representação de um job com um conjunto finito de n operações;
2. Armazenamento/leitura de ficheiro de texto com representação de um job;
3. Inserção de uma nova operação;
4. Remoção de uma determinada operação;
5. Alteração de uma determinada operação;
6. Determinação da quantidade mínima de unidades de tempo necessárias para completar o job e listagem das respetivas operações;
7. Determinação da quantidade máxima de unidades de tempo necessárias para completar o job e listagem das respetivas operações;
8. Determinação da quantidade média de unidades de tempo necessárias para completar uma operação, considerando todas as alternativas possíveis.

**Fase 2:**

1. Definição de uma estrutura de dados dinâmica para representação de um conjunto finito de m jobs associando a cada job um determinando conjunto finito de operações;
2. Armazenamento/leitura de ficheiro de texto com representação de um process plan;
3. Inserção de um novo job;
4. Remoção de um job;
5. Inserção de uma nova operação num job;
6. Remoção de uma determinada operação de um job;
7. Edição das operações associadas a um job;
8. Cálculo de uma proposta de escalonamento para o problema FJSSP (obrigatoriamente limitado a um tempo máximo de processamento configurável), apresentando a distribuição das operações pelas várias máquinas, minimizando o makespan (unidades de tempo necessárias para a realização de todos os jobs). A proposta de escalonamento deverá ser exportada para um ficheiro de texto possibilitando uma interpretação intuitiva;
9. Representação de diferentes process plan (variando a quantidade de máquinas disponíveis, quantidade de job, e sequência de operações, etc) associando as respetivas propostas de escalonamento.

# Propósitos e Objetivos

O presente trabalho têm como propósito mostrar os conhecimentos adqueridos relativos a definição e maniplução de estruturas de dados dinâmicas na linguagem de programação C, como também mostrar os conhecimentos na utilização de ferramentas como: Git e DoxyGen.

# Estrutura de dados

## Fase 1:

Para a realização da fase 1 do trabalho foi utilizada a estrutura de dados linear e dinâmica, **lista ligada.**

A **lista ligada** é composta por vários “nós” que estão interligados através de apontadores, ou seja, cada “nó” possui um apontador que aponta para o endereço de memória do próximo “nó”.

Imagem preta e branca de relógio ao fundo

Descrição gerada automaticamente com confiança baixa

Figura 1: Representação de uma lista ligada

# Testes Realizados

## Fase 1:

### 1. Definição de uma estrutura de dados dinâmica para a representação de um *job* com um conjunto finito de n operações:

Para representar um job, foi criada uma lista ligada de máquinas:

Texto

Descrição gerada automaticamente

Figura 2: Representação de uma máquina por uma struct

**nOperation:** Operação em que a máquina está inserida;

**nMachine:** Número da máquina;

**vTime:** Unidades de tempo da máquina;

***struct* machine:** apontador para a próxima máquina.

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Figura 3: Lista ligada com os nós de uma maquína

### 2. Armazenamento/leitura de ficheiro de texto com representação de um job;

Para realizar o armazenamento e leitura de um ficheiro de texto foram criadas três funções:

* *saveJob() :* Salva as operações de um job no ficheiro txt;
* *saveJobFromList() : Salva no ficheiro de texto todas as operações do job que está guardada na memória;*
* *readJob(): Realiza a leitura de um ficheiro txt com as operações relacionadas a um job e salva na memória.*

#### **saveJob() :**

Texto

Descrição gerada automaticamente

Figura 4: função para salvar operações de um job

Esta função recebe o número da operação, o número da máquina e o seu tempo, que foram inseridos pelo utilizador.

Caso já exista um ficheiro com a representação do job, a função só salva os novos dados no ficheiro, em caso do ficheiro não existir, este é criado e de seguida é adicionada a nova operação no ficheiro.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Figura 5: Ficheiro txt com a representação de um job

**(a, b, c):**

**a:** Número da operação;

**b:** Número da máquina;

**c:** Unidades de tempo nda máquina;

### 3. Inserção de uma nova operação:

Para realizar a inserção de uma nova operação, foram utilizadas duas funções:

* *insertAtBegin():* Adiciona uma nova operação no começo da lista;
* *newOperationInputs():* Recebe todos os inputs do utilizador; relativos a uma nova operação.

#### **insertAtBegin():**

Texto

Descrição gerada automaticamente

Figura 6: Função para inserir uma nova máquina no começo da lista

Depois de receber o número da operação, número da máquina e unidades de tempo da máquina, a função retorna o novo nó que será inserido no começo da lista de máquinas.

#### **newOperationInputs():**

Depois de verificar e testar todos os dados recebidos pelo utilizador, esta função guarda estes dados dentro das respetivas váriaveis e envia para para a função *insertAtBegin().*



Figura 7: Chamada da função insertAtBeginAndFile

### 4. Remoção de uma determinada operação:

A função *removeOperation()* recebe como parâmetro o número da operação que será removida e com a ajuda de dois apontadores guarda o nó anterior e o nó atual para no fim poder remover e realizar a ligação para o nó seguinte.

Diagrama

Descrição gerada automaticamente

Figura 8: Representação de remoção de um nó

Como vários nós contêm o mesmo número de operação, foi utilizado uma condicional *if* que verifica qual a operação que está contida no nó atual e remove caso seja a que o utilizador inseriu.

Texto

Descrição gerada automaticamente

Figura 9: Condição para se remover uma operação

### 5. Alteração de uma determinada operação:

Através da função *editOperation(),* podemos editar as máquinas da mesma.

#### **editOperation():**

A função recebe como parâmetro o número da operação a ser editada e pergunta ao utilizador qual máquina ele deseja editar.

A edição é feita alterando os dados contidos no nó em que a máquina se encontra.

Texto

Descrição gerada automaticamente

Figura 10: Edição de uma operação

### 6. Determinação da quantidade mínima de unidades de tempo necessárias para completar o job e listagem das respetivas operações:

Com a função *minimumAmountOfTime()* conseguimos percorrer toda a lista de máquinas e criar uma nova somente com as máquinas de menor tempo.

#### **minimumAmountOfTime():**

Com a ajuda de dois apontadores, a função percorre toda a lista de máquinas e verifica quais os nós que devem ser adicionados a nova lista.

Texto

Descrição gerada automaticamente

Figura 11: Verificação de um nó

Diagrama

Descrição gerada automaticamente

Figura 12: Representação da criação de uma nova lista a partir de outra

No final é chamada a função *sumMachineTime()* que recebe uma lista de máquinas e retorna a soma de todos as unidades de tempo.

Texto

Descrição gerada automaticamente

Figura 13: Soma de todas as unidades de tempo

### 7.Determinação da quantidade máxima de unidades de tempo necessárias para completar o job e listagem das respetivas operações:

Com a função *maximumAmountOfTime()* conseguimos percorrer toda a lista de máquinas e criar uma nova somente com as máquinas de maior tempo.

#### **maximumAmountOfTime():**

Com a ajuda de dois apontadores, a função percorre toda a lista de máquinas e verifica quais os nós que devem ser adicionados a nova lista.

Texto

Descrição gerada automaticamente

Figura 14: Verificação de um nó

Interface gráfica do usuário, Texto, chat ou mensagem de texto

Descrição gerada automaticamente

Figura 15: Representação da criação de uma nova lista a partir de outra

No final é chamada a função *sumMachineTime()* que recebe uma lista de máquinas e retorna a soma de todos as unidades de tempo.

Texto

Descrição gerada automaticamente

Figura 16: Soma de todas as unidades de tempo

### 8. Determinação da quantidade média de unidades de tempo necessárias para completar uma operação, considerando todas as alternativas possíveis:

A função *averageOperationTime()* percorre a lista de máquinas e cria uma nova com a operação e sua media de unidade de tempo.

A nova lista criada, têm como base uma nova estrutura representando a média de unidades de tempo das máquinas relativas a operação, *averageOp:*

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Figura 17: Estrutura de uma operação com a sua média de unidades de tempo das máquinas

#### **averageOperationTime():**

Cria uma nova lista de operações a partir da lista de máquinas, guardando o número da operação e a sua respetiva unidade de tempo em um novo nó;

Texto

Descrição gerada automaticamente

Figura 18:Código para verificar nós

Diagrama

Descrição gerada automaticamente

Figura 19: Representação da criação de uma nova lista a partir de outra

### Funções de ajuda

Estas são as funções de ajuda, que servem para “ajudar” as funções vistas anteriormente:

* *verifyInputValues(): Recebe um inteiro e verifica se é maior que 0.*

Figura : Código para verificar se valor é maior que 0

* Texto

  Descrição gerada automaticamente
* *verifyIfOperationExist()*: Recebe o número da operação e verifica se esta já existe.

Texto

Descrição gerada automaticamente

Figura 21: Código para verificar se operação já existe

* Texto

  Descrição gerada automaticamente*VerifyIfMachineExistInOperation():* Recebe o número da operação e da máquina e no final verifica se a máquina já foi inserida na operação.

Figura : Código para verificar se máquina já existe

### Git

Durante o desenvolvimento do programa foi utilizado o github como controle de versões:

<https://github.com/henriquearaujo93/eda-trabalho>

### Execução

Para ajudar a compilar e executar o código foi utilizado o *Makefile* com as diretivas necessárias.

Interface gráfica do usuário, Aplicativo, Teams

Descrição gerada automaticamente

Figura : Arquivo Makefile

**all: build run ->** Serve para xecucutar a diretiva **build** e **run**;

**build:** **->** Serve para somente compilar o código;

**run:** **->** Serve para somente executar o código;

**rm:** **->** Remove o executável gerado.

# Conclusão

Este trabalho permite desenvolver uma solução digital para um problema de escalonamento. Consequentemente, através dele, consegui pôr em prática o que foi apresentado durante as aulas sobre estruturas de dados avançadas em linguagem C. Durante a implementação do código senti que desenvolvi as minhas habilidades em relação a estruturas de dados *listas ligadas.*

# Bibliografia

Repositório Git com os códigos produzidos na aula: <https://github.com/luferIPCA/LESI-EDA-2122.git>;

Documentos de apoio forncecidos pelo professor;

Figura 1: <https://pt.wikipedia.org/wiki/Lista_ligada#/media/Ficheiro:C_language_linked_list.png>