Instituto Politécnico do Cávado e do Ave
Escola Superior de Tecnologia

Licenciatura
em
Engenharia Informática Médica

Relatório do TP-EDA Fase1

Rui Manuel Barbosa Cruz - 17078

Março 2022

Instituto Politécnico do Cávado e do Ave
Escola Superior de Tecnologia

Licenciatura
em
Engenharia Informática Médica

Relatório do TP-EDA Fase1

Estruturas de Dados Avançadas (EDA)

Flexible Job Shop Problem (FJSP)

Rui Manuel Barbosa Cruz - 17078

Março 2022

Índice

Índice de Figuras

Índice de Tabelas

1 Introdução

O aumento da recolha de dados e o seu armazenamento em conjunto com o aumento exponencial do poder de processamento de dados, estão a gerar grandes mudanças na industria mundial. Tal como a automação nos anos 70 foi revolucionária no seu tempo, agora é o aumento da industria digital tambem conhecida como a industria 4.0. Sensores, maquinas, peças e bases de dados que quando conectados formam *data centers*. Estes *data centers* serão capazes de analisar dados, prever falhas adaptarem-se a mudanças e melhorar a qualidade. Como consequencia o processamento será muito mais rapido, flexivel e eficiente, produzindo produtos costumizados de alta qualidade em pouco tempo e a um custo mais baixo. A industria 4.0 vai possibilitar que as empresas possam cortar custos, tempo no serviço e na produção através de um melhor planeamento e um melhor interface Homem-Maquina, niveis de stock e consumos de energia.

É na mira da Industria 4.0 que o problema de agendar e reagendar surge. Difinir o melhor tempo de execução de um processo é uma tarefa difícil ou um problema muito complexo dependendo do ambiente e condições de processamento. Particularizando o problema de linhas de produção, pode-se detetar dois fluxos principais:

- O job shop problem, JSP
- O flexible job shop problem, FJSP

O primeiro representa um dos problemas mais complexos e comuns na indústria, no qual um numero de operacoes, requer uma unica maquina, que trabalha ininterruptamente em uma operação de cada vez, e que se encontra continuamente disponivel. O segundo extende o primeiro, permitindo que a operação seja executada por uma maquina de uma lista de maquinas possiveis. O principal objetivo do JSP é sequenciar as operações para maximizar a performance medida por um indicador á escolha. O FJSP apresenta uma nova dimensão de decisões, uma vez que não so necessita de uma sequencia de operações, mas tambem da alocação de maquinas.

2 Propósitos e Objetivos

A essencia deste trabalho reside no desenvolvimento de uma solução digital para o problema de escalonamento denominado *Flexible Job Shop Scheduling Problem* (FJSSP). Um FJSSP pode ser formulado da seguinte forma:

- Um conjunto independente de jobs J = {J1, J2, …, Jn};
- Um conjunto de maquinas M = {M1 , M2 , ..., Mn};
- Cada job é formado por uma sequencia de operações (Oi, 1 , Oi, 2 , ...,Oi, ni);
- Cada operação pode ser executada num certo conjunto de maquinas;
- Uma maquina pode apenas executar uma operação de cada vez;

Dada a formulação, um job do problema pode ser sumarizado da forma como se encontra apresentado na tabela 1, onde as linhas correspondem as operações e as colunas às aos tempos das maquinas onde a operação pode ser executada.

	M1	*M2*	*M3*	*M4*	*M5*	*M6*	*M7*	*M8*
OP4	-	-	-	5	5	4	5	9

Tabela 1 - Exemplo da operação 4 do job 1

3 Estruturas de Dados

3.1 Funcionalidades

Em termos de funcionalidades o programa, permite ao utilizador:

- Fazer gestão da lista de *Maquinas*, que inclui operações de remoção, inserção e alteração de nome;
- Fazer gestão da lista de *Jobs*, que inclui operações de remoção, insersão e alteração de nome;
- Fazer gestão da lista de *Operações*, que se encontram associadas a um determinado *Job*, que inclui operações de remoção, inserção e alteração de nome;
- Fazer gestão do conjunto de *Maquinas* disponivel por operação, bem como o tempo de execução de cada *Maquina*.
- Identificar a quantidade minima de de unidades de tempo necessárias para completar um *Job*, não atendendo á disponibilidade das maquinas.
- Identificar a quantidade maxima de unidades de tempo necessárias para completar um *Job*, não atendento á disponibilidade das maquinas.
- Identificar a quantidade média de unidades de tempo necessárias para completar as operações de um *Job*, não atendendo á disponibilidade das maquinas.

3.2 Estrutura do Programa

Inicialmente o programa foi pensado e desenhado para ter como base de armazenamento de dados os vetores, ideia essa que acabou por ser repensada. Atualmente o programa utilizada listas de estruturas simples ligadas, visto que apenas é guardado o endereço do elemento seguinte.

O desenvolvimento do programa teve como base uma divisão estrutural de onde resultaram os seguinte capitulos:

- Maquinas – Contem a estrutura de todos os recursos capazes de executar operações;
- Operações – Estrutura das tarefas individuas que podem ser executadas por um conjunto de *n* maquinas e estão associadas a um *Job especifico*;
- *Job* – Estrutura dos processos de produção de uma instancia de um produto especifico, composto por *n* operações;
- Interface – Construido em cima dos outros modulos. Facilita a interação do utilizador com o programa;
- Armazenamento/Leitura de Ficheiros;

3.2.1 Maquinas

Este capitulo é iniciado pela parametrização de uma estrutura de dados. Esta estrutura é o pilar de todos os desenvolvimentos que serão apresentados neste capitulo.

Esta estrutura denominada de "*Maquina*", permite não só guardar a informação relativa a cada maquina em blocos de memoria autonomos, mas tambem criar uma ligação para um proximo bloco autonomo, criando uma ligação para o elemento adjacente. A ligaçao entre estes blocos autonomos faz com que esta estrutura se torne na verdade numa lista ligada simples.

```
typedef struct Maquina
{
    int idMaquina;
    char descMaquina[100];
    float tempo;
    struct Maquina *seguinte;
} Maquinas;
```

Figura 1 - Estrutura para armazenar lista de máquinas

Cada elemento da lista ligada, ou cada maquina, é composto por:

- IdMaquina – Este id, é incrementado automaticamente, como se de uma chave primaria de uma base de dados se trata-se. Sempre que o utilizador parametriza uma nova máquina, o programa acede a um ficheiro, que apenas serve para controlar o id das maquinas, e lê qual é o proximo id a ser atribuido, e incrementando o mesmo, e escrevendo por cima.
- desMaquina – Campo de texto definido pelo utilizador.
- Tempo – Campo tambem definido pelo utilizador. Este campo apenas recebe um valor, quando a respetiva maquina for associada a uma operação.

Quando o programa é iniciado, esta lista é tambem inicia com um valor de *NULL*, que logo de seguida é populada através da leitura de um ficheiro *.txt*, ficheiro este que é atualizado sempre que a lista sofre alguma alteração.

Essencialmente esta lista pode ser manipulada através da utilização de 3 funções diferentes.

3.2.1.1 Adicionar Máquina

Exitem 2 tipos de parametrização de maquinas. Podemos parametrizar uma maquina, em que a mesma fica apenas inserida na lista de máquinas disponiveis, ou podemos parametrizar uma maquina dentro de uma operação, em que esta fica associada à operação com um tempo definido, e fica tambem inserida na lista de maquinas disponiveis.

```
Maquinas *AdicionarMaquina(Maquinas *maquinas, int tipo)
{
    int idMaquina;//VARIAVEL PARA ARMAZENAR O ID DA MAQUINA
    char descMaquina[100];//VARIAVEL PARA ARMAZENAR A DESCRIÇÃO DA MAQUINA
    float tempo;//VARIAVEL PARA ARMAZENAR O TEMPO DA MAQUINA

    system("cls");//LIMPA A TELA
    printf("||=======================================================||\n");
    printf("||                    INSERIR MAQUINA                    ||\n");
    printf("||=======================================================||\n");

    fflush(stdin);//LIMPA O BUFFER DO TECLADO
    printf("|| DIGITE O NOME DA MAQUINA: ");// INSERIR NOME DA MAQUINA ESTRUTURA
    scanf("%[^\n]s", descMaquina);//LÊ A DESCRIÇÃO DA MAQUINA
    fflush(stdin);//LIMPA O BUFFER DO TECLADO
    idMaquina = proximoIdMaquina();

    if (tipo == 1)
    {
        //SE O TIPO FOR 1
        printf("INSIRA O TEMPO DE DURAÇÃO DA OPERAÇÃO: ");
        scanf("%f", &tempo);//LÊ O TEMPO DA MAQUINA
    }
    else
    {
        //SE O TIPO FOR 2
        tempo = 0;//TEMPO DA MAQUINA COM 0
    }

    system("cls");//LIMPA A TELA
    maquinas = InserirInicioListaMaquinas(maquinas, idMaquina, descMaquina,
tempo);//CHAMA A FUNÇÃO PARA INSERIR A MAQUINA NA LISTA
    return maquinas;//RETORNA A LISTA DE MAQUINAS
}
```

Figura 2 - Inserir nova maquina

Acima vemos apresenda uma função, denominada por “*Adicionar Maquina*”, que recebe duas variaveis, quando chamada. Sendo a primeira a lista de maquinas atual, na qual a maquina que está a ser parametrizada será incluida, e a segunda, uma variavel do tipo *int*, que apenas pode tomar 2 valores, sendo eles 0, e 1. Se esta variavél “*tipo*” estiver igual a 0, a maquina será apenas adicionada á lista de maquinas disponiveis para associar a mesma a uma operação, enquanto que se a variavel “*tipo*” estiver igual a 1, será pedido ao utilizador que parametrize um tempo de duraçao na operação, e esta é igualmente adicionada á lista de maquinas disponiveis, e será chamada uma nova função para associar a maquina a uma operação.

Esta função apenas permite recolher informação sobre a maquina que pretendemos adicionar, para a adicionar à lista, é chamada uma outra função.

```
Maquinas *InserirInicioListaMaquinas(Maquinas *maquinas, int idMaquina, char
descMaquina[100], float tempo)
{
    Maquinas *novaMaquina = (Maquinas *)malloc(sizeof(Maquinas));
    novaMaquina->idMaquina = idMaquina;
    strcpy(novaMaquina->descMaquina, descMaquina);
    novaMaquina->tempo = tempo;
    novaMaquina->seguinte = maquinas;
    return novaMaquina;
}
```

Figura 3 - Inserir maquina na lista

A função "*InserirInicioListaMaquinas*", recebe as variaveis parametrizadas na função anterior, reserva um bloco na memoria para a estrutura *Maquinas*, preenche o seu conteudo, atribui ao campo seguinte, a antiga lista de maquinas, e retorna a nova lista de maquinas, já com a nova maquina inserida.

3.2.1.2 Remover Máquina

Esta função "*DeleteMaquina*" recebe 2 variaveis, sendo a primeira, a lista de maquinas, e a segunda, o id da máquina que pretendemos eliminar. Foi esta a função que criou a necessidade de implementar um controlo do id. Numa fase inicial do programa, era pedido ao utilizador que defini-se qual o id da maquina. Isto permitia que existissem maquinas com o mesmo id. O que acontecia era que o programa percorria todas as maquinas até encontrar uma com o id correto, em que se existisse mais que uma maquina com o mesmo id, apenas a primeira a ser encontrada era eliminada.

Nesta função, são criadas duas listas novas, em que a "*nodoAtual*" é igualada á atual lista de maquinas, e a lista "*nodaAnterior*" fica inicialmente sem valor. É tambem criada uma int denomindade de "*found*", esta é inicializada com o valor de 0, e no decorrer da função, pode tomar o valor de 1. Esta serve para no final da função, fornecer feedback ao utilizador. Se no final da função a variavel "*found*" tiver o valor de 0, é porque nenhuma maquina foi eliminada, logo o utilizador sera notificado do mesmo. Em caso contrario, se a varivel "*found*" tiver o valor de 1, é porque a maquina que se prentendia eliminar foi encontrada, e a mesma foi eliminada.

O codigo para a remoção de uma maquina da lista é apresentado de seguida. De notar que é necessario tratar de forma diferente os casos de remoção da primeira maquina, da remoção das restantes. No caso de a maquina se encontrar na primeira posição da lista, o conceito que é aplicado, é o seguinte, a nova lista de maquinas, será igual à lista de maquinas atual na posição seguinte. Sendo assim a maquina que pretendemos eliminar já não consta na lista, contudo o espaço de memoria da mesma continua ocupado, por isso é necessário dar *free* do mesmo. Por outro lado, para remover uma maquina que não se encontre na primeira posição da lista, temos primeiramente que percorrer a lista até encontrar a maquina pretendida, ou até a lista ser igual a *NULL*. Caso a maquina seja encontrada, são realizados os mesmos passos que para uma maquina na primeira posição da lista, como se pode verificar no codigo abaixo.

```c
Maquinas *DeleteMaquina(Maquinas *maquinas, int idMaquina)
{
    system("cls");//LIMPA A TELA
    Maquinas *nodoAtual = maquinas;
    Maquinas *nodoAnterior;//VARIAVEL PARA ARMAZENAR A LISTA DE MAQUINAS ANTERIOR
    int found = 0;//VARIAVEL PARA VERIFICAR SE A MAQUINA FOI ENCONTRADA
    if (nodoAtual->idMaquina == idMaquina)
    {
        // SE A MAQUINA FOR ENCONTRADA NA PRIMEIRA POSIÇÃO
        found = 1;//MAQUINA ENCONTRADA
        maquinas = nodoAtual->seguinte;//ATUALIZA A LISTA DE MAQUINAS
        free(nodoAtual);//LIBERTA A MEMORIA
    }
    else
    {
        nodoAnterior = maquinas;//ATUALIZA A LISTA DE MAQUINAS
        nodoAtual = nodoAnterior->seguinte;//ATUALIZA A LISTA DE MAQUINAS ATUAL
        while ((nodoAtual != NULL) && (nodoAtual->idMaquina != idMaquina))
        {
            // ATÉ A MAQUINA FOR ENCONTRADA OU ATÉ A LISTA DE MAQUINAS ACABAR
            nodoAnterior = nodoAtual;
            nodoAtual = nodoAtual->seguinte;
        }
        if (nodoAtual != NULL)
        {
            nodoAnterior->seguinte = nodoAtual->seguinte;
            free(nodoAtual);//LIBERTA A MEMORIA
            found = 1;//MAQUINA ENCONTRADA
        }
    }
    if (found == 0)
    {
        system("cls");//LIMPA A TELA
        printf("||=========================================================||\n");
        printf("||               ID DA MAQUINA NÃO ENCONTRADO              ||\n");
        printf("||=========================================================||\n");
        system("pause");
    }else{
        system("cls");//LIMPA A TELA
        printf("||=========================================================||\n");
        printf("||               MAQUINA REMOVIDA COM SUCESSO              ||\n");
        printf("||=========================================================||\n");
        system("pause");
    }
    return maquinas;//RETORNA A LISTA DE MAQUINAS ATUALIZADA
}
```

Figura 4 - Remover maquina

3.2.1.3 Editar Máquina

A função "*EditarMaquina*" recebe 2 duas variaveis, sendo elas a lista de maquinas, e o id da maquina que se pretende alterar. Existe novamente uma variavel do tipo *int*, denominada de *found* inicializada com o valor 0. Esta é utilizada para no final da função poder dar feedback ao utilizadar. É criada uma nova lista de maquinas, igualada à lista de maquinas que entrou na função, e a mesma é percorrida até a lista estar vazia, caso pelo caminho seja encontrada uma maquina com id igual ao id que está a ser procurado, será disponibilizada a possibilidade de alterar o nome da mesma. No final a função retorna a nova lista de maquinas atualizada.

```
Maquinas *EditarMaquina(Maquinas *maquinas, int idMaquina)
{
    int found = 0;//VARIAVEL PARA VERIFICAR SE A MAQUINA FOI ENCONTRADA
    system("cls");//LIMPA A TELA
    Maquinas *listaDeMaquinas = maquinas;
    while (listaDeMaquinas != NULL)
    {
        if (listaDeMaquinas->idMaquina == idMaquina)
        {
            // SE A MAQUINA FOR ENCONTRADA
            found = 1;//MAQUINA ENCONTRADA
            printf("||=========================================================||\n");
            printf("||                      EDITAR MAQUINA                     ||\n");
            printf("||=========================================================||\n");
            printf("\tID Maquina: %d\n", listaDeMaquinas->idMaquina);
            printf("\tNome da Maquina: %s\n", listaDeMaquinas->descMaquina);
            printf("||=========================================================||\n");
            printf("|| NOVO NOME DA MAQUINA:");
            scanf("%s", listaDeMaquinas->descMaquina);//LÊ O NOVO NOME DA MAQUINA
        }

        listaDeMaquinas = listaDeMaquinas->seguinte;//PASSAR PARA A PROXIMA MAQUINA
    }
    if (found == 0)
    {
        // SE A MAQUINA NÃO FOR ENCONTRADA
        printf("||=========================================================||\n");
        printf("||              ID DA MAQUINA NÃO ENCONTRADO               ||\n");
        printf("||=========================================================||\n");
        system("pause");
    }
    return maquinas;//RETORNA A LISTA DE MAQUINAS ATUALIZADA
}
```

Figura 5 - Editar máquina

3.2.2 Operações

Este capitulo, tal como o das maquinas, é iniciado pela parametrização de uma estrutura de dados. Esta estrutura é o pilar de todos os desenvolvimentos que serão apresentados neste capitulo.

```
typedef struct Operacao
{
    int idOperacao;
    char descOperacao[100];
    Maquinas *maquinas;
    struct Operacao *seguinte;
} Operacoes;
```

Figura 6 - Estrutura para armazenar lista de operações

Cada elemento da lista ligada, ou cada operação, é composto por:

- IdOperacao – Este id é incrementado automaticamente.
- desOperacao – Campo de texto definido pelo utilizador.
- Maquinas – Lista de máquinas, onde uma especifica operação pode ser realizada.

Visto que o programa foi desenvolvido a pensar na fase 2, para tentar prevenir ao maximo que á medida que o programa vai evoluindo exista a necessidade de alterar os desenvolvimentos da fase 1, todas as funções que apresendadas daqui para a frente, utilzam sempre como variavel de entrada, a lista de *Jobs*. Quando o programa é iniciado, esta lista é tambem inicia com um valor de *NULL*, que logo de seguida é populada através da leitura de um ficheiro *.txt*, ficheiro este que é atualizado sempre que a lista sofre alguma alteração.

Tal como no capitulo das maquinas, o capitulo das operações contem funções para adicionar, editar e remover operações, funções estas que devido ás semelhanças que tem com as funções descritas nas maquinas não serão descritas neste capitulo. Assim sendo o capitulo das operações tem 3 funções novas:

3.2.2.1 Adicionar máquina a operação

Esta é a função que permite associar maquinas a uma operação especifica, recebe como variaveis de entrada, a lista de *Jobs*, o id do *job* no qual a operação está incuida, e o id da operação. A função começa por ler o ficheiro que contem todas as maquinas do programa, guardando-as numa lista ligada, e listando a mesma no erca do utilizador. Apos a listagem, é pedido ao utilizador que escolha qual das máquinas apresentadas pretende associar à operação. Assim que o utilizador inserir o id da máquina, o programa irá percorrer a lista de *jobs* até encontrar o id do *job*, que apos encontrado, serão percorridas as operações do respetivo *job*, até econtrar o id de operação escolhido, que quando encontrado, será percorrida a lista de máquinas associadas à respetiva operação para fazer um primeira verificação, e identificar se a máquina escolhida já se encontra associada à operação. Se sim, os ciclos serão quebrados, não serão realizadas alterações e o utilizador será

redirecionado para o anteriror menu. Se não, será pedido ao utilizador que insira o tempo que a maquina escolhida demorará a concluir a operação. Apos estes ciclos a nova lista de *jobs* será atualizada fazendo com que a respetiva operação possa agora ser executada em uma outra máquina.

3.2.2.2 Editar máquina operação

Esta função, permite editar o tempo de processamento de uma máquina, numa especifica operação. Recebe como variaveis de entrada a lista de *Jobs*, o id do *job* em que se encontra a operação, o id da operação em que se encontra a máquina e o id da máquina que se pretende editar.

Dentro da função, é parametrizada uma nova variável, denominada de *found*, do tipo inteiro, inicialmento com o valor de 0. Esta é a variavel que permite dar feedback ao utilizador no final da função. De seguida é incializada uma nova lista de jobs, que toma o valor da lista de *jobs* que deu entrada na função, esta é percorrida até ser encontrado o id do *job* pretendido. Dentro do *job* pretendido, e criada uma nova lista de operações, que toma o valor da lista de operações do *job*, esta mesma lista é percorrida até ser encontrada a operação cujo id de operação deu entrada na função. Apos encontrada, é parametrizada uma nova lista de maquinas, que inicialmente toma o valor da lista de máquinas associada á operação anteriormente referida. Esta lista é percorrida até ser encontrado o id da máquina que deu entrada na função. Apos encontrado, são listadas as informações atuais da máquina, e é dada a possiblidade ao utilizador de alterar o tempo de execução da operação na consequente máquina.

Caso durante todo o processo anteriormente referido, algo corra mal, a variavel *found*, vai continuar com o valor de 0, e o utilizador será informado, de que o id de máquina pretendido, não foi encontrado.

3.2.2.3 Remover maquina operação

Esta função "*DeleteMaquinaOperacao*" recebe 4 variáveis, sendo a primeira a lista de *jobs*, a segunda o id da operação, a terceira o id do *job*, e a quarta o id da máquina que pretendemos eliminar.

É incializada uma nova lista de jobs, que toma o valor da lista de *jobs* que deu entrada na função, esta é percorrida até ser encontrado o id do *job* pretendido. Dentro do *job* pretendido, e criada uma nova lista de operações que toma o valor da lista de operações do *job*, esta mesma lista é percorrida até ser encontrada a operação cujo id de operação deu entrada na função. É neste ponto que são criadas duas listas novas, em que a "*nodoAtual*" é igualada á atual lista de máquinas da operação, e a lista "*nodaAnterior*" fica inicialmente sem valor.

De notar que é necessario tratar de forma diferente os casos de remoção da primeira máquina, da remoção das restantes. No caso de a máquina se encontrar na primeira posição da lista, o conceito que é aplicado, é o seguinte, a nova lista de máquinas, será igual à lista de máquinas atual na posição seguinte. Sendo assim a máquina que pretendemos eliminar já não consta na lista, contudo o espaço de memoria da mesma

continua ocupado, por isso é necessário dar *free* do mesmo. Por outro lado, para remover uma máquina que não se encontre na primeira posição da lista, temos primeiramente que percorrer a lista até encontrar a máquina pretendida, ou até a lista ser igual a *NULL*. Caso a máquina seja encontrada, são realizados os mesmos passos que para uma máquina na primeira posição da lista.

3.2.3 *Jobs*

Este capitulo, tal como os dois anteriores é iniciado pela parametrização de uma estrutura de dados.

```
typedef struct Trabalho
{
    int idTrabalho;
    char descTrabalho[100];
    Operacoes *operacoes;
    struct Trabalho *seguinte;
} Trabalhos;
```

Figura 7- Estrutura para armazenar lista de jobs

Cada elemento da lista ligada, ou cada job é composto por:

- IdTrabalho – Este id, é incrementado automaticamente.
- DesTrabalho – Campo de texto definido pelo utilizador.
- Operações – Lista de operações.

Devido á grande semelhança entre as funções dos *jobs*, e as funções anteriormente descritas, as mesmas não serão descritas novamente.

3.2.4 Interface

Devido à elevada quantidade de funcionaliades, surgiu a necessiadade de desenvolver varios menus interligados entre si.

Uma vez que programa é corrido em ambiente de consola, a única possibilidade de ter um interface agradavél, é jogar com uma combinação de caracteres e simbolos. Para tornar o programa simples e de facil utilização, foi desenvolvido um interface que é utilizado em todos os menus. Em todos os menus está presente a tecla 0, que permite fechar o programa, e em grande parte dos menus, está tambem presente a tecla 9, que permite andar um menu para trás.

```c
void menuPrincipal(Trabalhos *trabalhos)
{
    system("cls");
    int opcao;
    Maquinas *maquinas = NULL;
    printf("||=========================================================||\n");
    printf("||                        MENU PRINCIPAL                   ||\n");
    printf("||=========================================================||\n");
    printf("||                                                         ||\n");
    printf("|| 1 - VER JOBS                                            ||\n");
    printf("|| 2 - VER MAQUINAS                                        ||\n");
    printf("|| 3 - VER ESTATISTICAS DO TRABALHO 1                      ||\n");
    printf("|| 0 - SAIR                                                ||\n");
    printf("||                                                         ||\n");
    printf("||=========================================================||\n");
    printf("|| OPÇÃO: ");
    scanf("%d", &opcao);
    switch (opcao)
    {
    case 1:
        menuTrabalhos(trabalhos);
        break;
    case 2:
        maquinas = lerFicheiroMaquinas(maquinas);
        listarMaquinas(maquinas, trabalhos);
        break;
    case 3:
        EstatisticasJob1(trabalhos);
        break;
    case 0:
        exit(0);
        break;
    default:
        printf("||=========================================================||\n");
        printf("||                  OPÇÃO INVÁLIDA                         ||\n");
        printf("||=========================================================||\n");
        system("pause");
        menuPrincipal(trabalhos);
        break;
    }
}
```

Figura 8- Menu principal

3.2.5 Armazenamento/Leitura de Ficheiros

Este capitulo descreve o processo de leitura e escrita em ficheiros.

Numa fase inicial do projeto, devido a uma falha de perceção do que era pedido para a fase 1, a leitura e escrita, foi desenvolvida para o programa ler e escrever em ficheiros binários. Uma vez que o que era pedido, era que o programa lê-se e escreve-se em ficheiros de texto, esta area teve que ser desenvolvida de novo.

Quando o novo desenvolvimento foi iniciado, já existiam colegas com esta area concluida, o que permitiu uma perceção do que iria ser o projeto final. Existiu a possibilidade de estudar o que os colegas tinham, e a conclusão foi que os ficheiros eram muito confusos, quase impossivél de perceber o que lá estava, e foi assim que surgiu a oportunidade de levar o projeto a um nivel superior.

Assim sendo, foi desevolvida de raiz, uma livraria que permite ler e escrever em ficheiros de texto em formato de ficheiro JSON, o que não só permite que o programa consiga ler e escrever no ficheiro, mas tambem que o utilizador consiga fazer o mesmo com muita facilidade, como se pode ver no exemplo abaixo apresentado, onde está representado um *job*.

```
[
        {
                "NomeTrabalho": "Process1"
                "IDTrabalho": 1
                "Operações": [
                        {
                                "NomeOperação": "Operacao4"
                                "IDOperação": 4
                                "Maquinas": [
                                        {
                                                "NomeMaquina": "Maquina8"
                                                "IDMaquina": 8
                                                "TempoMaquina": 9.00
                                        }
                                ]
                        }
                ]
        }
]
```

Figura 9 - Exempo de Job

3.3 Testes Realizados

A atividade de teste é o processo de executar um programa com a intenção de descobrir um erro, assim sendo, um bom caso de testes é aquele que tem uma elevada probabilidade de revelar erros que ainda não haviam sido descobertos. Esta não tem a capacidade de mostrar a ausência de *bugs*, mas sim demostrar se estão presentes defeitos no software.

Tem-se como principal objetivo, projetar testes que descubram sistematicamente erros diferentes. Se a atividade for realizada com sucesso, erros serão descobertos.

Nos dois tipos de teste aplicados no programa, no primeiro, foi testado o funcionamento geral do programa. Existiram testes de leitura e escrita em ficheiros através da inserção, edição e remoção de máquinas, edição e remoção de operações e edição e remoção de jobs. Este teste levou ao descobrimento de algumas falhas, sendo elas que na remoção de uma operação, as máquinas nunca chegam a ser eliminadas da memoria, o mesmo acontece quando se tenta adicionar uma maquina a uma operação. O programa lê de um ficheiro quais as máquinas disponiveis, guardando-as numa lista ligada, mas apos concluir os processos, nunca chega a dar *free* da memoria. O mesmo problema foi detetado na remoção de um *job*, o job é removido, mas as operações e máquinas que estavam associadas ao mesmo, nunca são apagadas da memoria. Estes erros serão corrigidos antes de iniciar a fase 2.

Em segundo, foi testado se as funções de gestão de máquinas numa operação estavam a funcionar corretamente. Destas funções foram obtidos 3 resultados diferentes, estando eles apresentados no gráfico abaixo.

A vermelho, temos apresentados os piores tempos de execução para o *job* 1, que é composto por 4 operações. Para a operação 1, a máquina escolhida foi a 3 com uma duração de 5 unidades de tempo. Para a operação 2, a máquina escolhida foi a 4 com uma duração de 5 unidade de tempo. Para a operação 3, a máquina escolhida foi a 5 com uma duração de 6 unidade de tempo. Para a operação 4, a máquina escolhida foi a 8 com uma duração de 9 unidade de tempo.

A verde, temos apresentados os melhores tempos de execução para o *job* 1, que é composto por 4 operações. Para a operação 1, a máquina escolhida foi a 1 com uma duração de 4 unidades de tempo. Para a operação 2, a máquina escolhida foi a 2 com uma duração de 4 unidade de tempo. Para a operação 3, a máquina escolhida foi a 3 com uma duração de 5 unidade de tempo. Para a operação 4, a máquina escolhida foi a 6 com uma duração de 4 unidade de tempo.

A azul, temos apresentados a média entre os melhores e os piores tempos de execução para o *job* 1, que é composto por 4 operações. Para a operação 1, a média de tempo de execução é de 4,5 unidades de tempo. Para a operação 2, a média de tempo de execução é de 4,5 unidades de tempo. Para a operação 3, a média de tempo de execução é de 5,5 unidades de tempo. Para a operação 4, a média de tempo de execução é de 5,6 unidades de tempo.

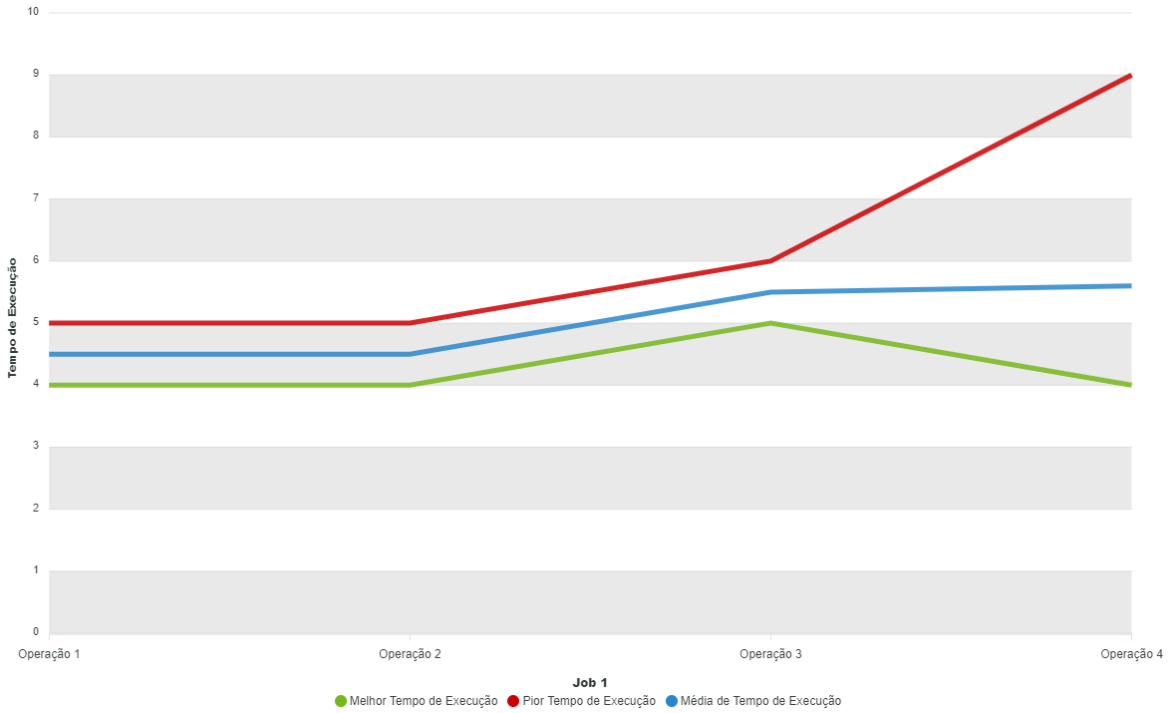

Figura 10 - Gráfico com Tempos de Execução

4 Conclusão

Este relatório apresenta um conjunto de ferramentas capazes de armazenar informação relativa a várias linhas de produção. Para atingir o objetivo, foram desenvolvidas listas ligadas de estruturas lineares. Numa fase de planeamento foram aplicadas com sucesso duas estruturas, sendo elas uma estrutura de máquinas e uma estrutura de operações, sendo que esta última estrutura continha um apontador para a primeira. Os desenvolvimentos iniciais desta fase mostraram os seus problemas, aquando do desenvolvimento da função de remoção de máquinas, em que era possível que existissem duas máquinas com o mesmo id. Isto levou ao planeamento de uma segunda fase de desenvolvimentos, em que foram aplicadas funções de controlo dos ids, e desenvolvida uma nova lista de estruturas lineares. Esta lista denominada de *Jobs*, continha dentro de si um apontador para a estrutura de operações, completando assim as linhas de produção.

Para concluir, todos os desenvolvimentos foram aplicados com sucesso mas quase todos eles poderiam ser melhorados. Todas as funções são dinâmicas, mas poderiam ser mais recursivas. Através de uma mera avaliação do código produzido, pode-se verificar que existe muito código repetido, que poderia facilmente ser corrigido através do desenvolvimento de funções recursivas. Outra grande falha é nas funções de remoção, em que ao dar delete de uma operação, ela é eliminada, deixa-se de ter acesso a ela, mas contudo, se ela tiver máquinas associadas, essas máquinas continuam presentes na memoria. Em desenvolvimentos futuros estes pontos apresentados irão ser corrigidos.

5 Bibliografia

Alexandre Pereira - C e Algoritmos 2ª Edição - Acedido a 1 de Março de 2022 em: Fomato Papel.

Luís Damas – Linguagem C 17ª Edição - Acedido a 1 de Março de 2022 em: Fomato Papel.