



Engenharia Eletrotécnica e de Computadores

Fase 1

Estruturas de Dados Avançadas

2021-2022

Instituto Politécnico do Cávado e do Ave
João Sampaio 18611



INSTITUTO POLITÉCNICO
DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

Índice

Introdução	6
1.Propósitos e Objetivos.....	7
2.Estruturas de Dados	8
2.1.Ficheiro (leitura/armazenamento).....	11
2.2.Ficheiro (Alocação de memória).....	12
2.3.Criação de um nodo/operação e a sua inserção na lista ligada.....	13
2.4.Apagar um nodo/operação da lista	14
2.5.Modificar um elemento da lista	16
2.6.Cálculo do tempo de operação	19
3.Testes Realizados.....	21
Conclusão	25
Bibliografia	26

Índice de Figuras

Figura 1 - Disposição da Lista Ligada	9
Figura 2 - Estrutura representativa de uma operação	10
Figura 3 - Disposição do ficheiro de leitura/armazenamento da lista	11
Figura 4 - Alocação de memória com malloc()	12
Figura 5 Ciclo de leitura do ficheiro	12
Figura 6 - Criação de um node	14
Figura 7 - Chamada da função que eliminará o node escolhido	14
Figura 8 - Remoção do node correspondente á cabeça da lista ligada	15
Figura 9 - Adição de uma máquina numa operação existente	16
Figura 10 Processo de eliminação da máquina	17
Figura 11 - Explicação gráfica do procedimento	17
Figura 12 - Eliminação total da operação	18
Figura 13 - Modificação do tempo de operação	18
Figura 14 - Cálculo do valor máximo de tempo de operação até concluir o job	19
Figura 15 - Cálculo do valor máximo de tempo de operação até concluir o job	20
Figura 16 - Menu Principal	21
Figura 17 - Escolha da operação a modificar	21
Figura 18 - Processo de remoção máquina 5 e máquina 4	22
Figura 19 - Lista final após a modificação	22
Figura 20 - Submenu da criação de uma operação	23
Figura 21 - Escolha das máquinas/tempo de operação	23
Figura 22 - Antes e depois da lista ligada	23
Figura 23 - Escolha da operação a apagar	24
Figura 24 - Lista após a remoção	24

Glossário

Node – elemento da lista ligada correspondente a uma operação.

EOF (end of file) – referência representativa do final de um ficheiro .txt.

Flexible Job Shop Problem (FJSSP) - extensão do problema clássico de escalonamento de job shop, que permite que uma operação seja processada por qualquer máquina de um determinado conjunto.

user input – valores inseridos pelo utilizador.

Introdução

O presente relatório tem a finalidade de demonstrar o funcionamento e lógica referentes ao trabalho prático proposto como avaliação da unidade curricular de Estruturas de Dados Avançadas.

O trabalho prático tem como objetivo sedimentar os conhecimentos relativos a definição e manipulação de estruturas de dados dinâmicas, na linguagem de programação C.

O objetivo principal consiste no desenvolvimento de uma solução digital para o problema de escalonamento denominado *Flexible Job Shop Problem* (FJSSP).

A implementação permitirá gerar uma proposta de escalonamento para a produção de um produto, dividindo-o em diferentes processos ou *jobs*, constituídos por diversas operações, que por sua vez, são constituídas por diferentes máquinas.

Cada máquina apresenta um tempo de operação distinto, onde a implementação deverá de ser capaz de avaliar a melhor opção, de maneira a minimizar o tempo necessário na produção do produto (*makespan*).

O trabalho pratico é dividido em duas fazes distintas.

1. Propósitos e Objetivos

Descrição Fase 1:

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

Na alteração/modificação de uma operação, o programa deve ser capaz de:

- Modificar o valor do tempo de operação de uma máquina á escolha, numa operação á escolha;
- Adicionar uma máquina numa operação;
- Remover uma máquina numa operação;
- Ser capaz de permitir modificar apenas máquinas que estão na operação;
- Não permitir o uso de máquinas repetidas na mesma operação;
- Não permitir que o tempo de operação de uma máquina seja nulo.
- Não permitir escolher uma máquina que não exista na operação.

2. Estruturas de Dados

Para a solução a implementar optei por uma lista ligada simples (estrutura de dados), de modo a criar uma lista de todas as operações inseridas em cada *job*.

Cada uma dessas operações tem como conteúdo as máquinas utilizadas e o seu tempo de operação, sendo permitido a modificação de todos os parâmetros.

A implementação permite:

- Criar, eliminar ou modificar toda a operação, desde as máquinas que podem ser utilizadas e os seus respetivos tempos de operação.
Limitações são aplicadas nos processos supracitados, de modo, por exemplo, a não permitir duas máquinas iguais na mesma operação.
- Ficheiro .txt usado para armazenar todas as alterações realizadas na lista, e também carregar o conteúdo no início do programa.
- Cálculo do tempo máximo, mínimo e médio necessário para concluir o *job*.

A seguinte imagem exemplifica a disposição da lista ligada.

Em cada operação existem dados referentes às máquinas e os seus tempos de operação, incluindo também o respetivo endereço da operação seguinte, de forma a criar a conexão entre a lista.

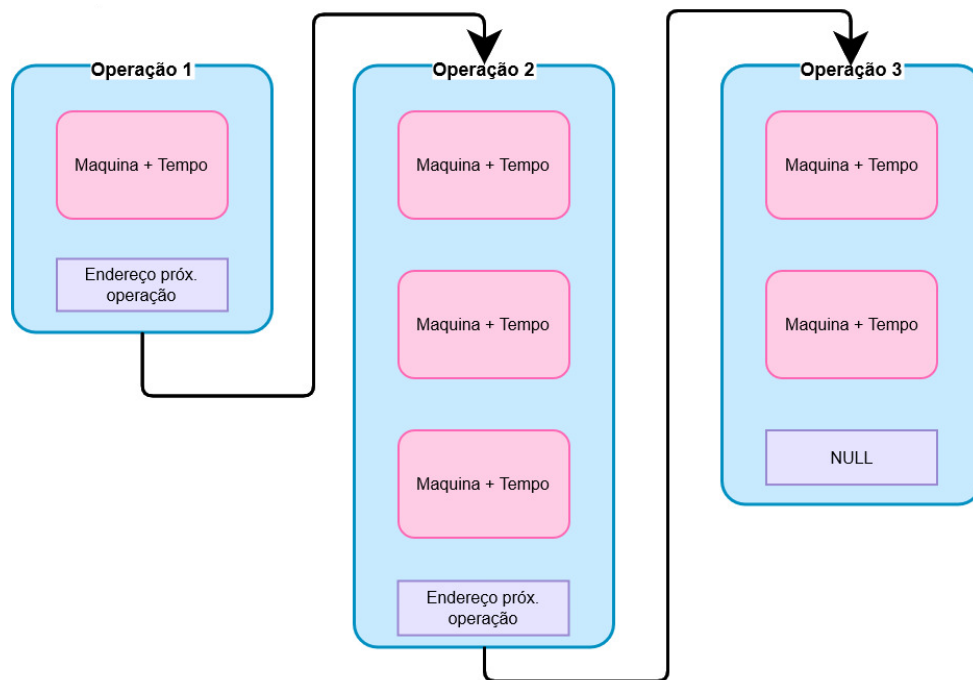


Figura 1 - Disposição da Lista Ligada

Os diferentes elementos de cada operação foram implementados através de apontadores usando alocação de memória dinâmica.

Esta abordagem permite ter em cada *node* (elemento da lista/operação), vários endereços para várias máquinas/tempo de operação, sem a necessidade de recorrer a memória estática como *arrays*, que iriam limitar o programa.

Ao último endereço da lista é atribuído o “valor” *NULL*, de forma a simbolizar o final da lista ligada.

Transcrevendo o diagrama anterior para código em linguagem C, construí a seguinte estrutura, que permitirá a criação das operações.

```
4
5  typedef struct operation
6  {
7      int counter; // to count the number of machines inside an operation
8      int* machineOperationTime;
9      int* machineNumber;
10     struct operation* next; // next position on the list
11 } operation;
```

Figura 2 - Estrutura representativa de uma operação

machineOperationTime – representa o tempo de operação.

machineNumber – representa a máquina.

next – representa o endereço do próximo *node* na lista ligada.

counter – variável auxiliar que permite contar a quantidade de máquinas, facilitando diversos processos como a leitura do ficheiro.

2.1. Ficheiro (leitura/armazenamento)

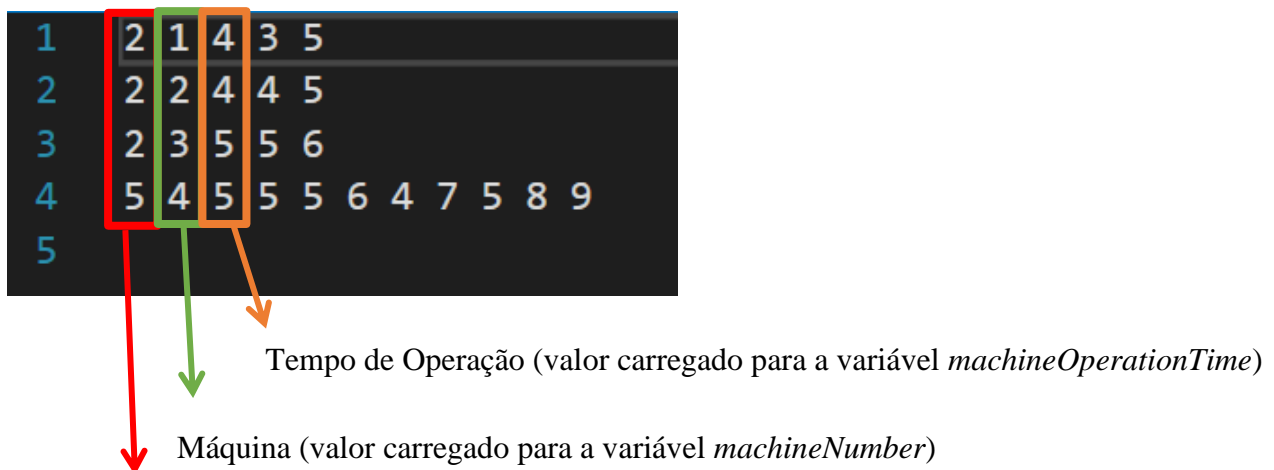
O processo de leitura do ficheiro e criação da lista ligada, no início da execução do programa, demonstrou-se um dos maiores desafios desta implementação. No entanto fui capaz de resolver o problema recorrendo a um método de leitura simplificado e funcional.

Sem comprometer a eficácia da implementação, organizei da maneira mais funcional possível o ficheiro, tendo como primeiro elemento da linha a quantidade de máquinas que fazem parte da operação.

Este valor inicialmente lido do ficheiro permite ao programa saber a quantidade de máquina que constituem a operação.

O valor será carregado na variável *counter*, incluída na estrutura de uma operação, permitindo recorrer a essa variável através de ciclos.

Em cada repetição do ciclo, é realizada uma alocação de memória para cada máquina e o seu respetivo tempo de operação.



Contador de máquinas dentro da operação (valor carregado para a variável *counter*)

Figura 3 - Disposição do ficheiro de leitura/armazenamento da lista

2.2. Ficheiro (Alocação de memória)

De forma a conseguir incluir mais que uma máquina e o seu respetivo tempo de operação, numa única operação, realizei uma alocação de memória dinâmica conforme o que fora gravado no ficheiro.

```
temporary->valueReadMachine = (int*)malloc(sizeof(int) * saveValue);
temporary->valueReadOpTime = (int*)malloc(sizeof(int) * saveValue);
```

Figura 4 - Alocação de memória com malloc()

Inicialmente existe uma declaração das variáveis e a sua respetiva alocação de espaço. Ao alocar espaço é necessário ter em conta o tipo de variável, neste caso, apontador para inteiro, e o seu tamanho, descrito na variável *saveValue*, que corresponde ao valor lido e carregado na variável *counter*.

De seguida, através de um ciclo, atribuímos os valores inseridos, ou extraídos do ficheiro, e inserimos nas respetivas variáveis. O ciclo irá possibilitar a resolução do objetivo que é colocar várias máquinas na mesma operação.

```
for (int i = 0; i < saveValue; i++) { // loops the amount of times needed to fill the operation
    fscanf(file, "%d ", &temporary->valueReadMachine[i]);
    fscanf(file, "%d ", &temporary->valueReadOpTime[i]);
}
// calls the function to create the list, with all the values and the counter
createOp = createNodeFile(head, temporary->valueReadMachine, temporary->valueReadOpTime, saveValue);
*head = insertAtTail(head, createOp, NULL);
createOp->next = NULL;
```

Figura 5 Ciclo de leitura do ficheiro

Após o ciclo da operação estar terminado, a função de criação do *node* é invocada. Após a criação da operação, a função retorna o *node* que foi criado, sendo recebido na variável *createOp* do tipo *operation** (tipo igual á estrutura).

Esse *node* é enviado como parâmetro na invocação da função de inserção na lista ligada, inserindo sempre no final da lista. Após finalizar sem erros, a cabeça da lista é retornada, permitindo a sua atualização. O endereço seguinte da lista será nulo, até outro nodo ser criado.

O processo decorre até terminar o ficheiro, ou seja, até EOF (*end of file*).

Na gravação do ficheiro apenas é necessário percorrer toda a lista ligada, escrevendo cada valor.

2.3. Criação de um nodo/operação e a sua inserção na lista ligada

Duas funções distintas foram implementadas de modo a criar um *node* na lista ligada.

Uma das funções apenas será utilizada quando é necessário realizar uma leitura do ficheiro, ou seja, no início da execução do programa (como fora explicado no tópico anterior). A outra permite a criação de *nodes* novos (operações novas) durante a execução do programa.

A decisão de separar em duas funções distintas tem o intuito de simplificar ao máximo o programa, de maneira a criar uma implementação de fácil compressão e sem comprometer a sua funcionalidade.

Novamente nesta segunda função, é utilizada uma variável que indicará qual é a quantidade de máquinas que irão fazer parte da operação, no entanto, desta vez, essa variável e todos os valores das máquinas e os seus respetivos tempos de operação, serão obtidos através de *user input*.

```

}else{
    printf("Choose the time of operation of machine: %d\n", inputMachine);
    scanf("%d", &inputOpTime);
    // adds the new node to the linked list
    node->machineNumber[i] = inputMachine; // adds a machine
    node->machineOperationTime[i] = inputOpTime; //adds a operation time fo
    counter++;
    node->counter = counter;
    previousMachine = inputMachine;
}

```

Figura 6 - Criação de um node

Após a leitura ou criação dos *nodes* pelo utilizador, o respetivo *node* criado não consta na lista ligada, ainda é necessário realizar a sua inserção e criar a ligação com o próximo elemento.

O programa tem a capacidade de realizar a inserção em qualquer posição, desde que seja especificado como parâmetro de entrada, ou seja escolhido através da interface gráfica com o utilizador. Por predefinição a inserção é realizada no final da lista ligada (exemplos no tópico Testes Realizados).

2.4. Apagar um nodo/operação da lista

De forma a eliminar um *node*/operação da lista, apenas é necessário chamar a função correspondente, enviando como parâmetro de entrada a cabeça da lista e o nodo a ser removido, que é posteriormente selecionado pelo utilizador (variável *option_2*), e encontrado na lista através da função de procura de *nodes* (*find_node()*).

```

if (scanf("%d", &option_2) > 0)
    if (option_2 > 0) {
        deleteNode(&head, find_node(head, option_2));
    }
    else break;

```

Figura 7 - Chamada da função que eliminará o node escolhido

Na função *deleteNode()*, inicialmente temos uma avaliação de qual posição é a escolhida. Se o nodo corresponder á cabeça da lista, a função está preparada para identificar a situação e eliminar a cabeça da lista. No entanto, não é possível simplesmente eliminar esse nodo, senão criaria problemas na lista ligada, faltando a ligação para as restantes posições.

De forma a colmatar esse problema, a cabeça da lista é transferida para a segunda posição, criando assim uma nova cabeça e atualizando o restante. Após essa atualização, o *node* que posteriormente estava na cabeça, é eliminado através da função *free()*.

Se o *node* escolhido não for o que está na cabeça da lista, o programa irá percorrer toda a lista ligada até encontrar o *node* em questão, e através da função *free()*, esse *node* será libertado deixando de ocupar espaço na memória.

```
if ((*head)->machineNumber == value->machineNumber)
{
    temporary = *head;    //backup to free the memory
    *head = (*head)->next;
    free(temporary);
}
```

Figura 8 - Remoção do node correspondente á cabeça da lista ligada

A função *free()* é sempre utilizada em casos que existe uma alocação de memória dinâmica e é necessário libertá-la.

2.5. Modificar um elemento da lista

Como operações de modificação, optei pelas seguintes:

- Modificar o valor do tempo de operação de uma máquina á escolha numa operação á escolha.
- Adicionar uma máquina numa operação.
- Remover uma máquina numa operação.

Adicionar uma nova máquina:

```
case 1: // add new item
    //adds the new machine and operation time, to the last position inside
    nodeToModify->machineNumber[nodeToModify->counter] = addMachine;
    nodeToModify->machineOperationTime[nodeToModify->counter] = addOpTime;
    nodeToModify->counter++;
    break;
```

Figura 9 - Adição de uma máquina numa operação existente

Na seguinte opção, o *node* enviado como parâmetro de entrada da função, que foi posteriormente escolhido pelo utilizador, é encontrado através da função *find_node()*.

Se uma operação tiver 2 elementos, ou seja, duas máquinas e os seus respetivos tempos de operação, a variável *counter* terá o valor 2 guardado. No entanto, as posições ocupadas nessa operação para as máquinas e tempos são [0] e [1], alocados dinamicamente.

Por essa razão apenas necessito de colocar como parâmetro dentro dos parenteses retos, a variável *counter*, que irá sempre corresponder ao espaço seguinte, neste caso [2]. O valor guardado em *counter* é atualizado após inseridos os valores escolhidos.

O que está descrito na imagem anterior, apenas será realizado na eventualidade da máquina que está a ser inserida, já não constar na operação. Se já existir, o programa não permitirá a sua colocação.

Remover uma máquina existente:

```

if (scanf("%d", &option) > 0) {
    while (counter < nodeToModify->counter) { // save the node in a buffer /
        bufferMachine[counter] = nodeToModify->machineNumber[counter];
        bufferOpTime[counter] = nodeToModify->machineOperationTime[counter];
        counter++;
        if (nodeToModify->machineNumber[counter] == option) { // saves the p
            pos = counter;
        }
    }
    // starts on the position to delete, and reorganizes the arrays
    for (counter = pos; counter < nodeToModify->counter; counter++) {
        bufferMachine[counter] = bufferMachine[counter + 1];
        bufferOpTime[counter] = bufferOpTime[counter + 1];
    }
}
nodeToModify->counter = counter - 1; // decrease one number on the operation
counter = 0;
while (counter < nodeToModify->counter) { // fill the node again without the
    nodeToModify->machineNumber[counter] = bufferMachine[counter];
    nodeToModify->machineOperationTime[counter] = bufferOpTime[counter];
    counter++;
}

```

Figura 10 Processo de eliminação da máquina

O utilizador escolhe inicialmente uma máquina que irá ser identificada dentro da operação, e o seu valor juntamente com o valor tempo de operação, serão copiados para dois buffers (*arrays*).

Após a finalização dessa etapa, executo uma operação que irá retirar o valor da posição onde se encontra a máquina a ser removida, substituindo o valor presente nessa posição, pelo valor na posição seguinte do *array*.

O valor na variável *counter* irá ser decrementado, devido á eliminação de uma máquina, e todos os elementos do array são copiados novamente para o *node*.

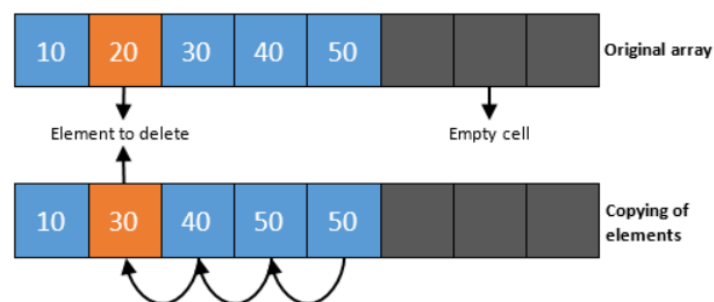


Figura 11 - Explicação gráfica do procedimento

Como característica adicional, o programa é capaz de identificar quando a operação ficou sem máquinas disponíveis, resultando na eliminação por completo da operação.

```
else { // delete the operation since there is no
    printf("Last machine available deleted.\n");
    printf("Operation deleted.\n");
    deleteNode(head, nodeToModify); // function c
}
```

Figura 12 - Eliminação total da operação

Modificar o tempo de operação:

É escolhida uma máquina existente na operação, sendo o valor do seu tempo de operação atualizado, conforme o que for escolhido pelo utilizador.

```
if (nodeToModify->machineNumber[i] == option) { // identify the machine chosen
    printf("\nPrevious operation time: %d\n", nodeToModify->machineOperationTime[i]);
    printf("New operation time: ");
    if(scanf("%d", &option) > 0) // reutilizing the option variable
        nodeToModify->machineOperationTime[i] = option; // update the operation time of sa
}
```

Figura 13 - Modificação do tempo de operação

2.6. Cálculo do tempo de operação

De forma a realizar o cálculo do tempo de operação no *job*, utilizei uma abordagem simples.

Para realizar o cálculo do valor máximo e mínimo apenas recorri ao uso de ciclos. Um ciclo para percorrer toda a lista e um ciclo para poder atualizar uma variável.

Sempre que o valor máximo é encontrado na operação, ele é guardado e mais tarde somado (ao valor da operação anterior), de forma a obter o valor máximo de tempo que teria de decorrer até terminar o *job*.

Isto aconteceria se o sistema optasse pelas máquinas com mais tempo de operação por cada operação.

Para o tempo mínimo, o contrário é aplicado ao código representado na imagem seguinte, atualizando sempre que encontrar o valor mínimo na operação.

```
while (temporary != NULL) {  
    for (int i = 0; i < temporary->counter; i++) {  
        if (i == 0) {  
            max = temporary->machineOperationTime[i];  
        }  
        else if (max <= temporary->machineOperationTime[i]) {  
            max = temporary->machineOperationTime[i];  
        }  
    }  
    sum += max;  
    temporary = temporary->next;  
}
```

Figura 14 - Cálculo do valor máximo de tempo de operação até concluir o job

Para o cálculo da média de valores apenas é necessário realizar um somatório de todos os tempos de operação em todas as operações, e dividir pela quantidade de máquinas no *job* inteiro.

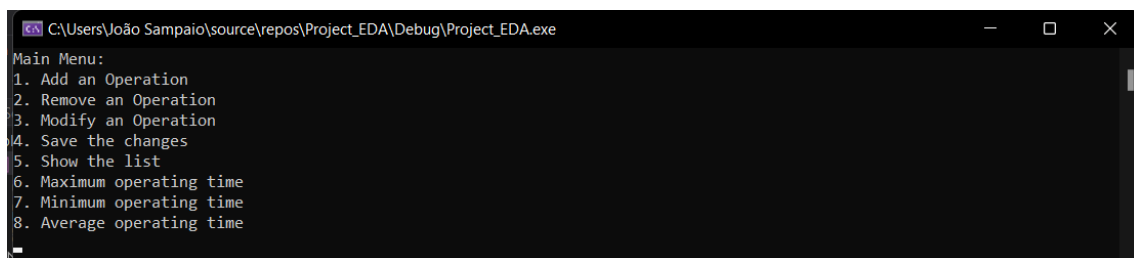
```
while (temporary != NULL) {  
    for (int i = 0; i < temporary->counter; i++) {  
        sum += temporary->machineOperationTime[i];  
    }  
    count += temporary->counter;  
    temporary = temporary->next;  
}
```

Figura 15 - Cálculo do valor máximo de tempo de operação até concluir o job

3. Testes Realizados

De modo a facilitar a compreensão do funcionamento da solução implementada, apresento os seguintes exemplos:

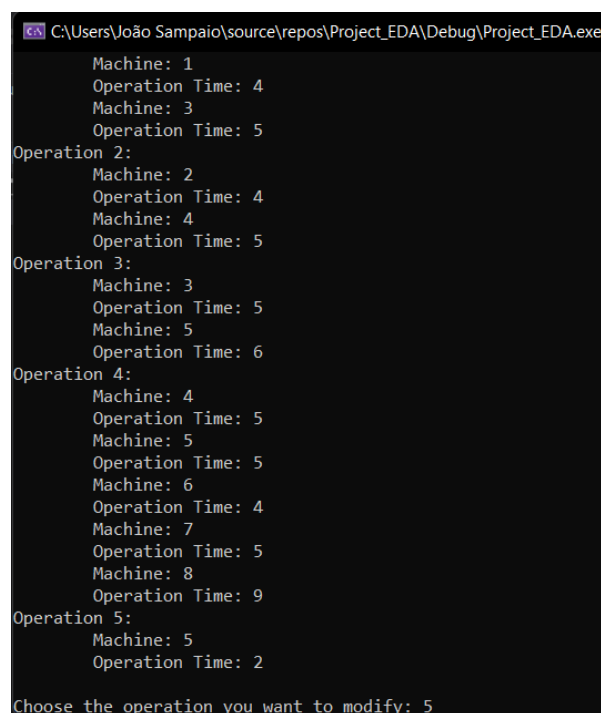
O menu principal apresenta todas as possibilidades de operações a realizar, de modo a criar um *job*. Para aceder aos submenus de cada operação, é necessário fornecer ao programa o *input* do número referente à operação desejada.

A screenshot of a Windows command prompt window titled "C:\Users\João Sampaio\source\repos\Project_EDA\Debug\Project_EDA.exe". The window displays the "Main Menu:" with a list of eight options: 1. Add an Operation, 2. Remove an Operation, 3. Modify an Operation, 4. Save the changes, 5. Show the list, 6. Maximum operating time, 7. Minimum operating time, and 8. Average operating time. A cursor is visible at the bottom of the list.

```
C:\Users\João Sampaio\source\repos\Project_EDA\Debug\Project_EDA.exe
Main Menu:
1. Add an Operation
2. Remove an Operation
3. Modify an Operation
4. Save the changes
5. Show the list
6. Maximum operating time
7. Minimum operating time
8. Average operating time
```

Figura 16 - Menu Principal

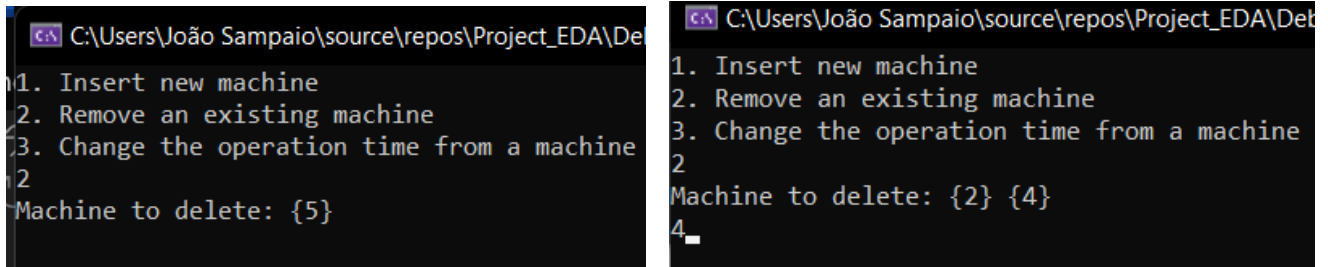
Escolhendo a modificação de uma operação, acedemos ao menu de modificação, onde toda a lista é apresentada e como *input* o utilizador escolherá a operação que pretende modificar.

A screenshot of the same Windows command prompt window, now displaying the "Operation 1:" through "Operation 5:" menu. Each operation entry lists the machine number and the operation time. At the bottom, it prompts the user to "Choose the operation you want to modify: 5".

```
C:\Users\João Sampaio\source\repos\Project_EDA\Debug\Project_EDA.exe
Machine: 1
Operation Time: 4
Machine: 3
Operation Time: 5
Operation 2:
Machine: 2
Operation Time: 4
Machine: 4
Operation Time: 5
Operation 3:
Machine: 3
Operation Time: 5
Machine: 5
Operation Time: 6
Operation 4:
Machine: 4
Operation Time: 5
Machine: 5
Operation Time: 5
Machine: 6
Operation Time: 4
Machine: 7
Operation Time: 5
Machine: 8
Operation Time: 9
Operation 5:
Machine: 5
Operation Time: 2
Choose the operation you want to modify: 5
```

Figura 17 - Escolha da operação a modificar

Como exemplo, irei executar um processo de remoção de uma máquina na operação 2 e na operação 5, removendo a máquina 4 e a 5, respetivamente.

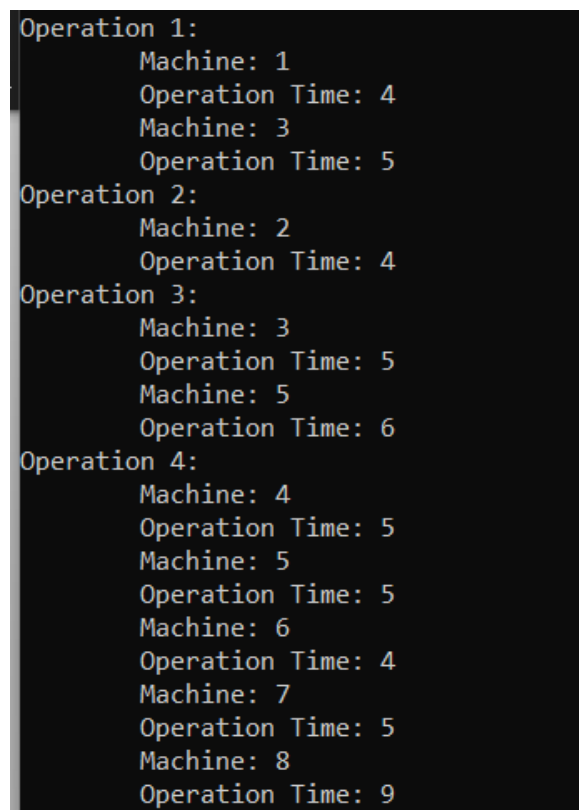


The figure consists of two side-by-side terminal windows. Both windows show a list of three operations: '1. Insert new machine', '2. Remove an existing machine', and '3. Change the operation time from a machine'. In the left window, after operation 2, the prompt 'Machine to delete: {5}' is shown. In the right window, after operation 2, the prompt 'Machine to delete: {2} {4}' is shown, and the cursor is on line 4.

```
C:\Users\João Sampaio\source\repos\Project_EDA\De
1. Insert new machine
2. Remove an existing machine
3. Change the operation time from a machine
2
Machine to delete: {5}

C:\Users\João Sampaio\source\repos\Project_EDA\De
1. Insert new machine
2. Remove an existing machine
3. Change the operation time from a machine
2
Machine to delete: {2} {4}
4
```

Figura 18 - Processo de remoção máquina 5 e máquina 4



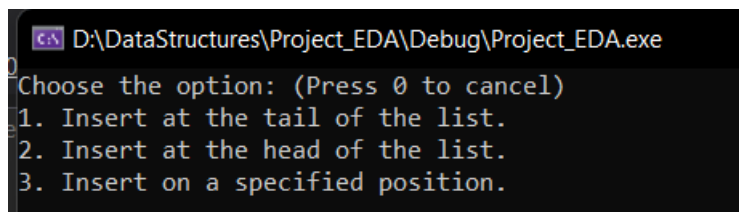
The terminal window displays a list of operations grouped by their type. Operation 1 has two entries for machines 1 and 3. Operation 2 has one entry for machine 2. Operation 3 has two entries for machines 3 and 5. Operation 4 has five entries for machines 4, 5, 6, 7, and 8.

```
Operation 1:
    Machine: 1
    Operation Time: 4
    Machine: 3
    Operation Time: 5
Operation 2:
    Machine: 2
    Operation Time: 4
Operation 3:
    Machine: 3
    Operation Time: 5
    Machine: 5
    Operation Time: 6
Operation 4:
    Machine: 4
    Operation Time: 5
    Machine: 5
    Operation Time: 5
    Machine: 6
    Operation Time: 4
    Machine: 7
    Operation Time: 5
    Machine: 8
    Operation Time: 9
```

Figura 19 - Lista final após a modificação

Como podemos verificar, as máquinas já não fazem parte da lista de operações.

Exemplo de criação de um node:

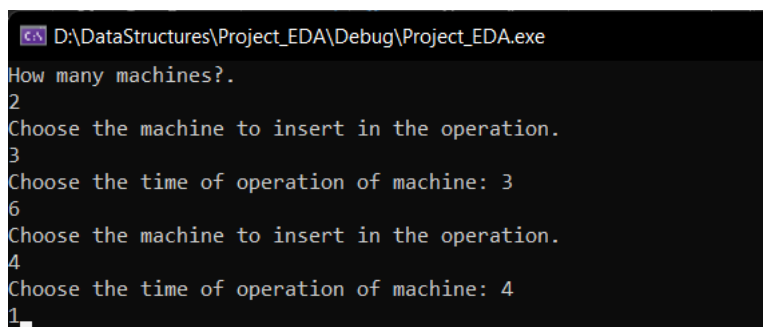


```
D:\DataStructures\Project_EDA\Debug\Project_EDA.exe
Choose the option: (Press 0 to cancel)
1. Insert at the tail of the list.
2. Insert at the head of the list.
3. Insert on a specified position.
```

Figura 20 - Submenu da criação de uma operação

Irei colocar a nova operação na 3^o posição da lista.

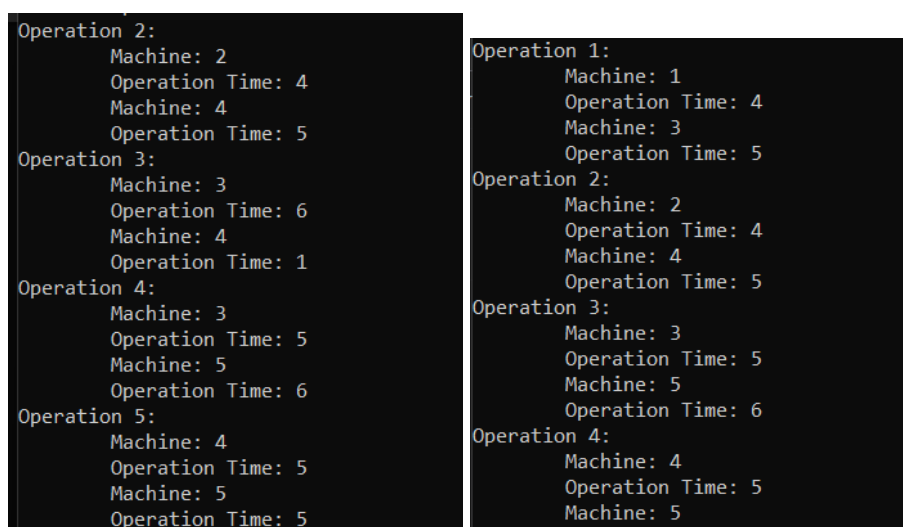
Após escolher a 3^o posição, o programa necessita que o utilizador indique quantas máquinas serão utilizadas, o seu valor e o tempo de operação.



```
D:\DataStructures\Project_EDA\Debug\Project_EDA.exe
How many machines?..
2
Choose the machine to insert in the operation.
3
Choose the time of operation of machine: 3
6
Choose the machine to insert in the operation.
4
Choose the time of operation of machine: 4
1
```

Figura 21 - Escolha das máquinas/tempo de operação

Como podemos comparar, o novo *node* foi adicionado na 3^a posição, reordenando a lista.



```
Operation 2:
  Machine: 2
  Operation Time: 4
  Machine: 4
  Operation Time: 5
Operation 3:
  Machine: 3
  Operation Time: 6
  Machine: 4
  Operation Time: 1
Operation 4:
  Machine: 3
  Operation Time: 5
  Machine: 5
  Operation Time: 6
Operation 5:
  Machine: 4
  Operation Time: 5
  Machine: 5
  Operation Time: 5

Operation 1:
  Machine: 1
  Operation Time: 4
  Machine: 3
  Operation Time: 5
Operation 2:
  Machine: 2
  Operation Time: 4
  Machine: 4
  Operation Time: 5
Operation 3:
  Machine: 3
  Operation Time: 5
  Machine: 5
  Operation Time: 6
Operation 4:
  Machine: 4
  Operation Time: 5
  Machine: 5
  Operation Time: 5
```

Figura 22 - Antes e depois da lista ligada

Exemplo de remoção de uma operação:

Utilizando a operação criada anteriormente, procedemos á sua eliminação.

```
Operation Time: 9
Choose the operation to delete.(Press 0 to go back)
3
```

Figura 23 - Escolha da operação a apagar

Após a seleção, o *node* adicionado á lista no exemplo anterior foi removido, e a lista voltou a ser reordenada.

```
Operation Time: 5
Operation 2:
Machine: 2
Operation Time: 4
Machine: 4
Operation Time: 5
Operation 3:
Machine: 3
Operation Time: 5
Machine: 5
Operation Time: 6
Operation 4:
Machine: 4
Operation Time: 5
Machine: 5
Operation Time: 5
Machine: 6
Operation Time: 4
Machine: 7
```

Figura 24 - Lista após a remoção

Para todos os exemplos anteriores, considereei como lista ligada, a tabela apresentada no enunciado do trabalho pratico.

Conclusão

A implementação da primeira fase do trabalho prático foi bastante enriquecedora, visto que possibilitou colocar em prática os conhecimentos obtidos ao longo da primeira parte das aulas da unidade curricular de Estruturas de Dados Avançadas, e também consolidar os conhecimentos adquiridos em Programação Imperativa.

Fui capaz de aprimorar as minhas capacidades já adquiridas ao nível da programação em linguagem C de modo geral. Clarifiquei certos pontos, como por exemplo apontadores, e consegui chegar mais além, colocando em prática métodos de criação de listas dinâmicas de dados e alocação de memória.

Bibliografia

Link repositório GitHub:

https://github.com/sampaio00joao/Project_EDA_18611 - atualizado pela
última vez a 18/04/2022