Relatório Trabalho Prático

Estruturas de Dados Avançadas

Pedro Miguel Dias Leiras, 18550

01/06/2022

Professor: Luís Ferreira

Índice

Índica de Figuras

Introdução

O presente relatório ilustra quais os propósitos e objetivos do trabalho prático, quais estruturas de dados utilizadas na sua resolução e testes realizados, no âmbito da UC de Estruturas de Dados Avançadas do curso Engenharia de Desenvolvimento de Jogos Digitais do Instituto Politécnico do Cávado e Ave.

Propósitos e Objetivos

Este trabalho prático visa o reforço e a aplicação dos conhecimentos adquiridos ao longo do semestre, pretendendo-se sedimentar os conhecimentos relativos a definição e manipulação de estruturas de dados dinâmicas na linguagem de programação C.

A essência deste trabalho reside no desenvolvimento de uma solução digital para o problema de escalonamento denominado *Flexible Job Shop Problem* (FJSSP). A solução a implementar 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 (*makespan*).

Numa primeira fase a solução deve satisfazer as seguintes alíneas:

- 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.

Numa segunda fase:

- 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;
- Armazenamento/leitura do ficheiro de um *process plan*;
- Inserção de um novo job;
- Remoção de um job;
- Inserção de uma nova operação num job;
- Remoção de uma determinada operação de um job;
- Edição das operações associadas a um job;
- Geração de uma solução para o FJSSP, apresentando a distribuição das operações pelas várias máquinas, e determinando o menor *makespan* (unidades de tempo necessárias para realização de todos os jobs);

Estruturas de Dados

Fase 1

Para contemplar a resolução da primeira fase as seguintes estruturas de dados foram criadas:

Machine: Estrutura utilizada para armazenar os dados de uma máquina, que contém o seu código (cod) e o tempo (time).

```
typedef struct Machine {
    int cod;    /**< Codigo da Maquina*/
    int time;   /**< Tempo para executar a operacao*/
}Machine;
```

Figura 1 - Estrutura Máquina

MachinesList: Lista que armazena estruturas do tipo "Machine" (máquinas).

```
typedef struct MachinesList {
    struct Machine machine;              /**< toda a informacao da Maquina */
    struct MachinesList* nextMachine;    /**< Ligacao a outra Maquina*/
}MachinesList;
```

Figura 2 - Lista de Máquinas

Operation: Estrutura utilizada para armazenar os dados de uma operação, que contém o seu código (cod) e uma lista de máquinas (MachinesList).

```
typedef struct Operation {
    int cod;                             /**< Codigo da Operacao*/
    struct MachinesList* machines;       /**< Todas as Maquinas da Operacao*/
}Operation;
```

Figura 3 - Estrutura Operação

OperationList: Lista que armazena estruturas do tipo “Operation” (operações).

```
typedef struct OperationsList {
    struct Operation operation;                 /**< toda a informacao da Operacao */
    struct OperationsList* nextOperation;       /**< Ligacao a outra Operacao*/
}OperationsList;
```

Figura 4 - Lista de Operações

Job: Estrutura utilizada para armazenar os dados de um “job”, que contém o seu código (cod) e uma lista de operações (OperationList).

```
typedef struct Job {
    int cod;                                    /**< Codigo do Job*/
    struct OperationsList* operations;          /**< Todas as Operacoes do Job*/
}Job;
```

Figura 5 - Estrutura Job

ProcessPlan: Lista que armazena estruturas do tipo “Job” (*process plan*).

```
typedef struct ProcessPlan {
    struct Job job;                         /**< toda a informacao do Job */
    struct ProcessPlan* nextJob;            /**< Ligacao o outro Job*/
}ProcessPlan;
```

Figura 6 - Lista de Jobs

OperationsMath: Estrutura utilizada para armazenar o resultado de cálculos sobre operações, que contém o resultado do cálculo (res) e a lista das operações utilizadas (“OperationList”).

```
typedef struct OperationsMath {
    int res;
    struct OperationsList* operations;
}OperationsMath;
```

Figura 7 - Estrutura OperationsMath

Fase 2

Na fase 2, as estruturas foram mantidas sendo alterada a estrutura ProcessPlan de uma lista ligada para uma árvore, figura 8.

```
typedef struct Node {
    Job job;                    /**< toda a informacao do Job */
    struct Node* leftTree;      /**< Ligacao o Job a esquerda*/
    struct Node* rightTree;     /**< Ligacao o Job a direita*/
}Node, * PtrNode;
```

Figura 8 - Árvore do ProcessPlan

Foi implementada uma estrutura auxiliar para guardar toda a informação da arvore num ficheiro binário, figura 9.

```
typedef struct NodeInfo {
    int codJob;
    int codOperation;
    Machine machine;
}NodeInfo;
```

Figura 9 - Estrutura NodeInfo

Testes Realizados

Fase 1

Para criar as estruturas do tipo “Machine”, “Operation” e “Job” foram criadas funções que recebem como parâmetro os dados e retornam as respetivas estruturas.

Como podemos ver na figura abaixo, temos um exemplo dessas funções que cria a estrutura do tipo “Machine”.

```
Machine* CreateMachine(int cod, int time) {
    Machine* aux = (Machine*)calloc(1, sizeof(Machine));
    aux->cod = cod;
    aux->time = time;

    return aux;
}
```

Figura 10 - Função "CreateMachine"

Para testar o funcionamento desta função, foram enviados como parâmetro os dados em “hard code” de máquinas, como podemos ver na figura 9 e na figura 10 o retorno das funções.

```
Machine* M1 = CreateMachine(1, 4);
Machine* M2 = CreateMachine(2, 4);
Machine* M3 = CreateMachine(3, 5);
Machine* M4 = CreateMachine(4, 5);
```

Figura 11 - Utilização da função "CreateMachine"

Name	Value	Type
M1	0x00e39630 {cod=1 time=4 }	Machine *
M2	0x00e39668 {cod=2 time=4 }	Machine *
M3	0x00e392b8 {cod=3 time=5 }	Machine *
M4	0x00e392f0 {cod=4 time=5 }	Machine *
Add item to watch		

Figura 12 - Retornos da função "CreateMachine"

Para testar a introdução destas funções nas listas foi implementado funções para cada estrutura como podemos ver um exemplo, na figura abaixo.

```
ProcessPlan* InsertJobInProcessPlan(ProcessPlan* h, Job* newJob) {
    if (newJob == NULL) {
        return h;
    }

    ProcessPlan* aux = CreateProcessPlanNode(newJob);

    if (CheckJobExists(h, aux->job.cod)) {
        return h;
    }

    if (h == NULL) {
        h = aux;
    }else {
        ProcessPlan* tmp = h;
        while (tmp->nextJob != NULL) {
            tmp = tmp->nextJob;
        }
        tmp->nextJob = aux;
    }

    return h;
}
```

Figura 13 - Função "InsertJobInProcessPlan"

O exemplo da figura 11 é relativo à inserção de um job, figura 12, na lista de jobs (process plan). Após criar um job (para efeitos de teste este job será utilizado nos testes seguintes com o nome de” Job1”) e iniciar a lista, os dados foram enviados por parâmetro e o resultado obtido foi o esperado, figura 13.

Job1	0x0137def0 {cod=1 operations=0x0137db80 {operation={cod=1 machines=0x0137dad8 {machine={cod=1 time=4 } ...} } ...} }
cod	1
operations	0x0137db80 {operation={cod=1 machines=0x0137dad8 {machine={cod=1 time=4 } nextMachine=0x0137db10 {machine=...} } } ...}
operation	{cod=1 machines=0x0137dad8 {machine={cod=1 time=4 } nextMachine=0x0137db10 {machine={cod=2 time=4 } ...} } }
cod	1
machines	0x0137dad8 {machine={cod=1 time=4 } nextMachine=0x0137db10 {machine={cod=2 time=4 } nextMachine=0x00000000 <NULL> } }
machine	{cod=1 time=4 }
cod	1
time	4
nextMachine	0x0137db10 {machine={cod=2 time=4 } nextMachine=0x00000000 <NULL> }
machine	{cod=2 time=4 }
cod	2
time	4
nextMachine	0x00000000 <NULL>
nextOperation	0x00000000 <NULL>

Figura 14 - Exemplo de Job

p	0x0137df28 {job={cod=1 operations=0x0137db80 {operation={cod=1 machines=0x0137dad8 {machine={cod=1 time=...} ...} } ...} } ...}
job	{cod=1 operations=0x0137db80 {operation={cod=1 machines=0x0137dad8 {machine={cod=1 time=4 } nextMachine=...} } ...} }
cod	1
operations	0x0137db80 {operation={cod=1 machines=0x0137dad8 {machine={cod=1 time=4 } nextMachine=0x0137db10 {machine=...} } } ...}
operation	{cod=1 machines=0x0137dad8 {machine={cod=1 time=4 } nextMachine=0x0137db10 {machine={cod=2 time=4 } ...} } }
cod	1
machines	0x0137dad8 {machine={cod=1 time=4 } nextMachine=0x0137db10 {machine={cod=2 time=4 } nextMachine=0x00000000 <NULL> } }
machine	{cod=1 time=4 }
cod	1
time	4
nextMachine	0x0137db10 {machine={cod=2 time=4 } nextMachine=0x00000000 <NULL> }
machine	{cod=2 time=4 }
cod	2
time	4
nextMachine	0x00000000 <NULL>
nextOperation	0x00000000 <NULL>
nextJob	0x00000000 <NULL>

Figura 15 - Exemplo de Lista de Jobs

De forma a testar a adição e remoção de operações de um job foram implementadas as respetivas funções, figura 14, onde podemos observar na figura 15, uma nova operação, na figura 16 um job antes de ser adicionada essa mesma operação e por fim na figura 17 a representação do job após adição da operação.

```
p = AddNewOperationToJob(p, 1, O2);
p = AddNewOperationToJob(p, 1, O3);
```

Figura 16 - Função para adicionar operação num job

O2	0x00b8fea8 {cod=2 machines=0x00b8fff8 {machine={cod=3 time=5 } nextMachine=0x00b8fe70 {machine={cod=...} ...} } }
cod	2
machines	0x00b8fff8 {machine={cod=3 time=5 } nextMachine=0x00b8fe70 {machine={cod=4 time=5 } nextMachine=0x00000000 <NULL> } }
machine	{cod=3 time=5 }
cod	3
time	5
nextMachine	0x00b8fe70 {machine={cod=4 time=5 } nextMachine=0x00000000 <NULL> }
machine	{cod=4 time=5 }
cod	4
time	5
nextMachine	0x00000000 <NULL>

Figura 17 - Exemplo de operação

Name	Value
Job1	0x00b8ff88 {cod=1 operations=0x00b8fd30 {operation={cod=1 machines=0x00b8fc88 {machine={cod=1 time=4 } …} } …} }
cod	1
operations	0x00b8fd30 {operation={cod=1 machines=0x00b8fc88 {machine={cod=1 time=4 } nextMachine=0x00b8fcc0 {machine=…} } } …}
operation	{cod=1 machines=0x00b8fc88 {machine={cod=1 time=4 } nextMachine=0x00b8fcc0 {machine={cod=2 time=4 } …} } }
cod	1
machines	0x00b8fc88 {machine={cod=1 time=4 } nextMachine=0x00b8fcc0 {machine={cod=2 time=4 } nextMachine=0x00000000 <NULL> } }
machine	{cod=1 time=4 }
nextMachine	0x00b8fcc0 {machine={cod=2 time=4 } nextMachine=0x00000000 <NULL> }
nextOperation	0x00000000 <NULL>

Figura 18 – Job1 antes de adicionar operação

Job1	0x00b8ff88 {cod=1 operations=0x00b8fd30 {operation={cod=1 machines=0x00b8fc88 {machine={cod=1 time=4 } …} } …} }
cod	1
operations	0x00b8fd30 {operation={cod=1 machines=0x00b8fc88 {machine={cod=1 time=4 } nextMachine=0x00b8fcc0 {machine=…} } } …}
operation	{cod=1 machines=0x00b8fc88 {machine={cod=1 time=4 } nextMachine=0x00b8fcc0 {machine={cod=2 time=4 } …} } }
cod	1
machines	0x00b8fc88 {machine={cod=1 time=4 } nextMachine=0x00b8fcc0 {machine={cod=2 time=4 } nextMachine=0x00000000 <NULL> } }
machine	{cod=1 time=4 }
nextMachine	0x00b8fcc0 {machine={cod=2 time=4 } nextMachine=0x00000000 <NULL> }
nextOperation	0x00b90068 {operation={cod=2 machines=0x00b8fff8 {machine={cod=3 time=5 } nextMachine=0x00b8fe70 {machine=…} } } …}
operation	{cod=2 machines=0x00b8fff8 {machine={cod=3 time=5 } nextMachine=0x00b8fe70 {machine={cod=4 time=5 } …} } }
cod	2
machines	0x00b8fff8 {machine={cod=3 time=5 } nextMachine=0x00b8fe70 {machine={cod=4 time=5 } nextMachine=0x00000000 <NULL> } }
nextOperation	0x00000000 <NULL>

Figura 19 – Job1 após adição de uma operação

Para remover, foi introduzido o código do Job1, e o código de uma operação presente no Job1, neste caso “1”. Na figura 19, temos ilustrado o Job1 antes da remoção e na figura 20 após a remoção da operação com o código 1.

```
p = RemoveOperationFromJob(p, 1, 1);
```

Figura 20 - Função para remover operação num job

Name	Value
Job1	0x0129dbe8 {cod=1 operations=0x0129dae0 {operation={cod=1 machines=0x0129da38 {machine={cod=1 time=4 } …} } …} }
cod	1
operations	0x0129dae0 {operation={cod=1 machines=0x0129da38 {machine={cod=1 time=4 } nextMachine=0x0129da70 {machine=…} } } …}
operation	{cod=1 machines=0x0129da38 {machine={cod=1 time=4 } nextMachine=0x0129da70 {machine={cod=2 time=4 } …} } }
cod	1
machines	0x0129da38 {machine={cod=1 time=4 } nextMachine=0x0129da70 {machine={cod=2 time=4 } nextMachine=0x00000000 <NULL> } }
nextOperation	0x0129dcc8 {operation={cod=2 machines=0x0129dc58 {machine={cod=3 time=5 } nextMachine=0x0129de50 {machine=…} } } …}
operation	{cod=2 machines=0x0129dc58 {machine={cod=3 time=5 } nextMachine=0x0129de50 {machine={cod=4 time=5 } …} } }
nextOperation	0x0129dc20 {operation={cod=3 machines=0x0129de88 {machine={cod=1 time=4 } nextMachine=0x0129dec0 {machine=…} } } …}
operation	{cod=3 machines=0x0129de88 {machine={cod=1 time=4 } nextMachine=0x0129dec0 {machine={cod=3 time=5 } …} } }
nextOperation	0x00000000 <NULL>

Figura 21 - Job1 antes de remover operação

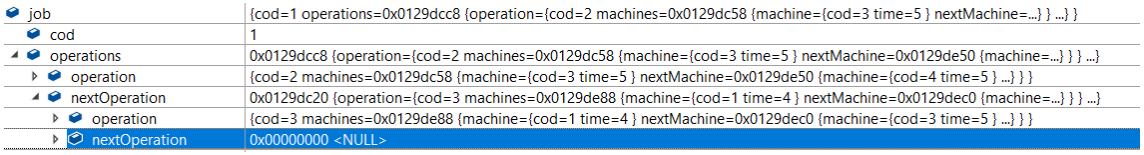
job	{cod=1 operations=0x0129dcc8 {operation={cod=2 machines=0x0129dc58 {machine={cod=3 time=5 } nextMachine=…} } …} }
cod	1
operations	0x0129dcc8 {operation={cod=2 machines=0x0129dc58 {machine={cod=3 time=5 } nextMachine=0x0129de50 {machine=…} } } …}
operation	{cod=2 machines=0x0129dc58 {machine={cod=3 time=5 } nextMachine=0x0129de50 {machine={cod=4 time=5 } …} } }
nextOperation	0x0129dc20 {operation={cod=3 machines=0x0129de88 {machine={cod=1 time=4 } nextMachine=0x0129dec0 {machine=…} } } …}
operation	{cod=3 machines=0x0129de88 {machine={cod=1 time=4 } nextMachine=0x0129dec0 {machine={cod=3 time=5 } …} } }
nextOperation	0x00000000 <NULL>

Figura 22 - Job1 após remover operação

De forma a alterar uma operação num job foi escolhida a operação com o código 3, do Job1, para alterar a máquina com o código 3, para o código 5 e tempo 10, figura 21.

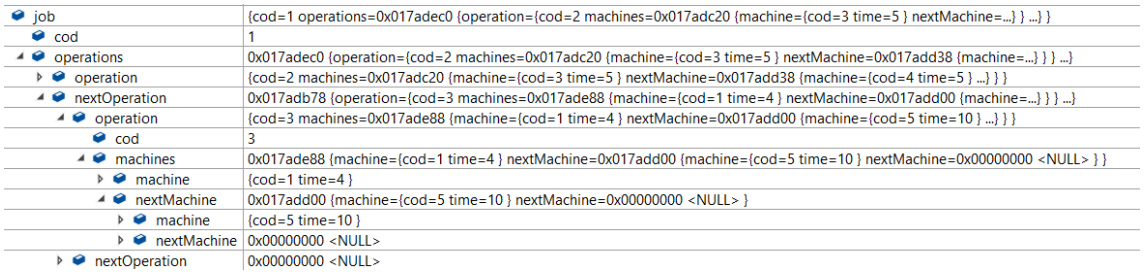
```
p = ChangeOperationInJob(p, 1, 3, 3, 5, 10);
```

Figura 23 - Função para alterar operação num job



job	{cod=1 operations=0x0129dcc8 {operation={cod=2 machines=0x0129dc58 {machine={cod=3 time=5 } nextMachine=...} } ...} }
cod	1
operations	0x0129dcc8 {operation={cod=2 machines=0x0129dc58 {machine={cod=3 time=5 } nextMachine=0x0129de50 {machine=...} } } ...}
operation	{cod=2 machines=0x0129dc58 {machine={cod=3 time=5 } nextMachine=0x0129de50 {machine={cod=4 time=5 } ...} } }
nextOperation	0x0129dc20 {operation={cod=3 machines=0x0129de88 {machine={cod=1 time=4 } nextMachine=0x0129dec0 {machine=...} } } ...}
operation	{cod=3 machines=0x0129de88 {machine={cod=1 time=4 } nextMachine=0x0129dec0 {machine={cod=3 time=5 } ...} } }
nextOperation	0x00000000 <NULL>

Figura 24 - Job1 antes da alteração

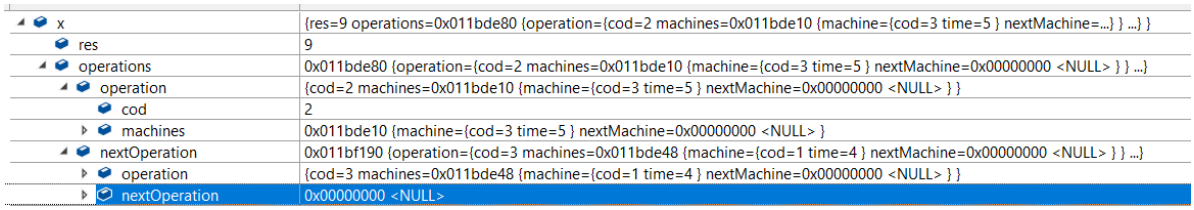


job	{cod=1 operations=0x017adec0 {operation={cod=2 machines=0x017adc20 {machine={cod=3 time=5 } nextMachine=...} } ...} }
cod	1
operations	0x017adec0 {operation={cod=2 machines=0x017adc20 {machine={cod=3 time=5 } nextMachine=0x017add38 {machine=...} } } ...}
operation	{cod=2 machines=0x017adc20 {machine={cod=3 time=5 } nextMachine=0x017add38 {machine={cod=4 time=5 } ...} } }
nextOperation	0x017adb78 {operation={cod=3 machines=0x017ade88 {machine={cod=1 time=4 } nextMachine=0x017add00 {machine=...} } } ...}
operation	{cod=3 machines=0x017ade88 {machine={cod=1 time=4 } nextMachine=0x017add00 {machine={cod=5 time=10 } ...} } }
cod	3
machines	0x017ade88 {machine={cod=1 time=4 } nextMachine=0x017add00 {machine={cod=5 time=10 } nextMachine=0x00000000 <NULL> } }
machine	{cod=1 time=4 }
nextMachine	0x017add00 {machine={cod=5 time=10 } nextMachine=0x00000000 <NULL> }
machine	{cod=5 time=10 }
nextMachine	0x00000000 <NULL>
nextOperation	0x00000000 <NULL>

Figura 25 - Job1 após a alteração

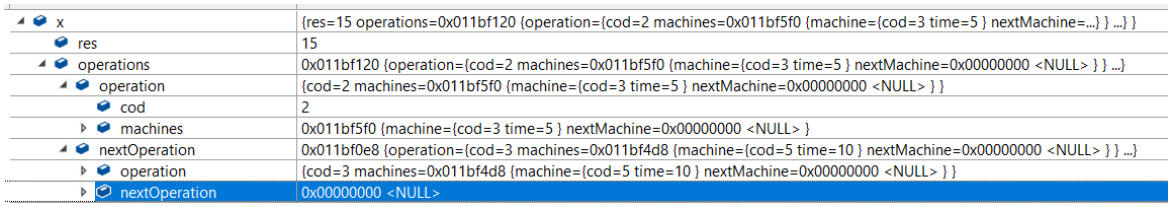
Para calcular o tempo mínimo e máximo de um job, foi utilizado o Job 1 da figura 23.

Podemos ver o resultado do tempo mínimo na figura 24, o máximo na figura 25 e as listagens das operações, respetivamente, na figura 26.



x	{res=9 operations=0x011bde80 {operation={cod=2 machines=0x011bde10 {machine={cod=3 time=5 } nextMachine=...} } ...} }
res	9
operations	0x011bde80 {operation={cod=2 machines=0x011bde10 {machine={cod=3 time=5 } nextMachine=0x00000000 <NULL> } } ...}
operation	{cod=2 machines=0x011bde10 {machine={cod=3 time=5 } nextMachine=0x00000000 <NULL> } }
cod	2
machines	0x011bde10 {machine={cod=3 time=5 } nextMachine=0x00000000 <NULL> }
nextOperation	0x011bf190 {operation={cod=3 machines=0x011bde48 {machine={cod=1 time=4 } nextMachine=0x00000000 <NULL> } } ...}
operation	{cod=3 machines=0x011bde48 {machine={cod=1 time=4 } nextMachine=0x00000000 <NULL> } }
nextOperation	0x00000000 <NULL>

Figura 26 - Tempo mínimo para realizar Job1



x	{res=15 operations=0x011bf120 {operation={cod=2 machines=0x011bf5f0 {machine={cod=3 time=5 } nextMachine=...} } ...} }
res	15
operations	0x011bf120 {operation={cod=2 machines=0x011bf5f0 {machine={cod=3 time=5 } nextMachine=0x00000000 <NULL> } } ...}
operation	{cod=2 machines=0x011bf5f0 {machine={cod=3 time=5 } nextMachine=0x00000000 <NULL> } }
cod	2
machines	0x011bf5f0 {machine={cod=3 time=5 } nextMachine=0x00000000 <NULL> }
nextOperation	0x011bf0e8 {operation={cod=3 machines=0x011bf4d8 {machine={cod=5 time=10 } nextMachine=0x00000000 <NULL> } } ...}
operation	{cod=3 machines=0x011bf4d8 {machine={cod=5 time=10 } nextMachine=0x00000000 <NULL> } }
nextOperation	0x00000000 <NULL>

Figura 27 - Tempo máximo para realizar Job1

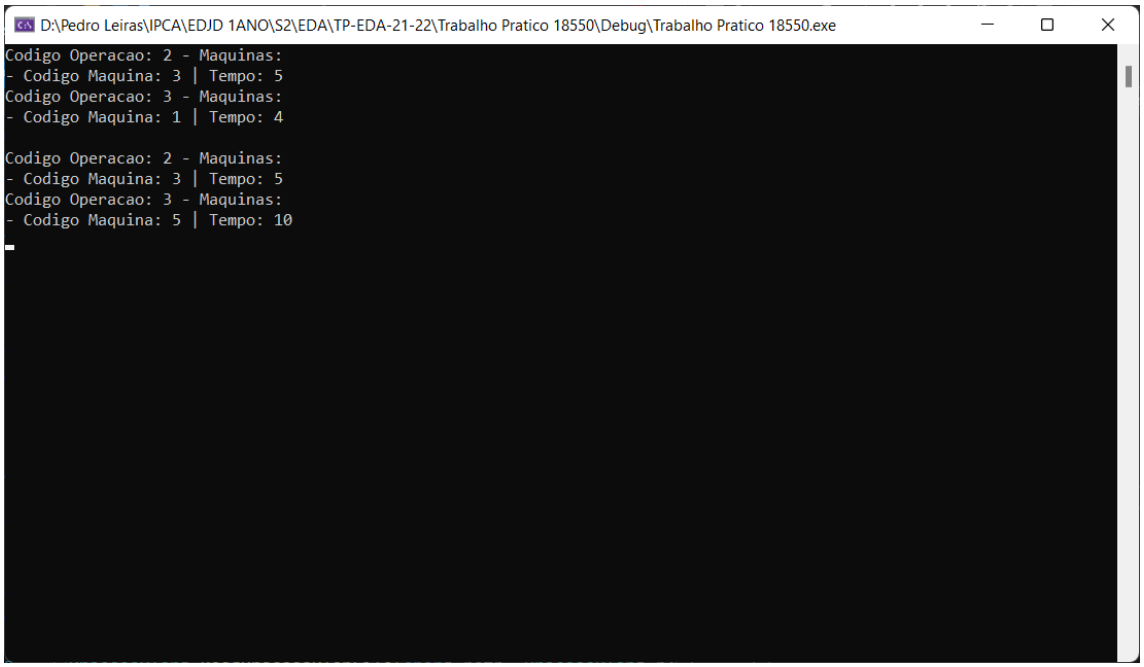


Figura 28 - Listagem das operações com o tempo mínimo e máximo, respetivamente

Por fim, para determinar a médio de tempo de uma operação, foi escolhida a operação com o código 5 do Job1, figura 27.

```
y = GetAverageOperationTime(p, 1, 5);
```

Figura 29 - Função para calcular a média de uma operação

Tendo a operação com o código 3, duas máquinas com tempos de 4 e 10 o resultado da média será 7, como exemplificado na figura 28.

Name	Value
y	7.00000000

Figura 30 - Resultado do cálculo da média

Fase 2

Na fase 2, os dados foram carregados a partir de um ficheiro binário e as funções utilizadas na fase 1 foram reaproveitadas, alterando apenas o parâmetro de entrada e saída de lista para árvore, não apresentado quaisquer erros.

Conclusão

A realização deste trabalho ajudou-me a melhor e a cimentar os meus conhecimentos de programação, C e listas ligadas, contudo nem todos os objetivos propostos para a primeira fase foram cumpridos e como tal serve de impulso para continuar a aprender mais e a melhorar, de forma a ultrapassar as dificuldades sentidas para conseguir completar todos os objetivos na segunda fase.

Na segunda fase as dificuldades da primeira fase foram superadas, porém ficou um ponto do enunciado por resolver, devido a dificuldades na compreensão do mesmo.