Flexible Job Shop Problem

Parte 2

Trabalho Prático

Instituto Politécnico do Cavado e do Ave

Licenciatura em Engenharia em Desenvolvimento de Jogos Digitais

Estrutura de Dados Avançados

Ademar Valente nº23155

Docente: Luís Ferreira

Maio 2022

Índice

Introdução

O presente relatório procura descrever da forma mais sucinta o processo de realização do segundo conjunto de funcionalidades pedido no trabalho prático lançado pela docência no âmbito de realização da disciplina de Estrutura de Dados Avançados; inserida no curso de Licenciatura de Engenharia em Desenvolvimento dos Jogos Digitais.

O desafio continua a ser o de implementar uma solução digital que permita gerar uma proposta de escalonamento para a realização de um determinado produto, que para ser criado utiliza um conjunto específico de máquinas e uma consequente execução com um determinado tempo de execução. Este processo visa minimizar o tempo perdido entre a execução dos diferentes processos e intervenientes nele inseridos. O enunciado do projeto identifica sob a forma de quadro o organigrama de organização da empresa na construção dos seus produtos.

Process Plan	Operation						
	O 1	**O 2**	**O 3**	**O 4**	**O 5**	**O 6**	**O 7**
$pr_{1,2}$	(1,3) [4,5]	(2,4) [4,5]	(3,5) [5,6]	(4,5,6,7,8) [5,5,4,5,9]			
$pr_{2,2}$	(1,3,5) [1,5,7]	(4,8) [5,4]	(4,6) [1,6]	(4,7,8) [4,4,7]	(4,6) [1,2]	(1,6,8) [5,6,4]	(4) [4]
$pr_{3,3}$	(2,3,8) [7,6,8]	(4,8) [7,7]	(3,5,7) [7,8,7]	(4,6) [7,8]	(1,2) [1,4]		
$pr_{4,2}$	(1,3,5) [4,3,7]	(2,8) [4,4]	(3,4,6,7) [4,5,6,7]	(5,6,8) [3,5,5]			
$pr_{5,1}$	(1) [3]	(2,4) [4,5]	(3,8) [4,4]	(5,6,8) [3,3,3]	(4,6) [5,4]		
$pr_{6,3}$	(1,2,3) [3,5,6]	(4,5) [7,8]	(3,6) [9,8]				
$pr_{7,2}$	(3,5,6) [4,5,4]	(4,7,8) [4,6,4]	(1,3,4,5) [3,3,4,5]	(4,6,8) [4,6,5]	(1,3) [3,3]		
$pr_{8,1}$	(1,2,6) [3,4,4]	(4,5,8) [6,5,4]	(3,7) [4,5]	(4,6) [4,6]	(7,8) [1,2]		

Fig.1 – Tabela/Problema base do Projeto de trabalho.

O trabalho enunciado divide-se em duas partes, com requisitos diferentes, que procuram solucionar problemas de forma evolutiva, ordenada e dinâmica em termos de aprendizagem. O presente relatório ser o propósito de expor as soluções para os problemas propostos na parte 2 do trabalho prático, que são:

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 do ficheiro 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. 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*).

Propósitos e Objetivos

A realização do presente trabalho serve o propósito principal de continuidade de aprendizagem na **linguagem de programação C** iniciada na disciplina de Programação Imperativa do semestre passado.

Dadas as dificuldades sentidas anteriormente, os objetivos são os de tirar um aproveitamento máximo na aprendizagem, por forma a ir minimizando as lacunas que ainda são bastante consideráveis à data presente; bem como ir construindo com solidez uma base que me permita gerar soluções para os problemas que me vão sendo propostos e dominar de forma crescente a linguagem especifica da elaboração dos trabalhos (neste caso, a linguagem C).

Para além disso, é objetivo adicional ao trabalho proposto o de adquirir conhecimentos relativos a outras ferramentas de trabalho propostas pela docência para a realização deste trabalho. Passa pelos objetivos os de avaliar, armazenar conhecimentos e adquirir competências no manuseamento do ***VisualStudio***, ***Git***, ***GitHub*** e ***Doxigen***, intervenientes no processo de realização deste projeto.

Estruturas de Dados

A estrutura de dados do presente projeto assenta num conjunto de estruturas dinâmicas (***structs***). As estruturas adotadas estão interligadas entre si por forma a demonstrarem cada trabalho especificamente, apontando cada uma delas para a seguinte. Associado a este tipo de estruturação houve a necessidade de uma criar **tabela Hash** em forma de Array de maneira a ligar todas as funcionalidades das diferentes listas utilizadas.

Para além disso foi ainda inicializada uma estrutura capaz de dar resposta ao problema 8 do enunciado, não tendo sido, no entanto, concluída.

De forma resumida, o trabalho possui seis listas ligadas.

Primeiramente a estrutura Trabalho (***Job***), caracterizada por um identificador. Nesta estrutura (tal como em todas as outras) o Trabalho aponta para um próximo que possa existir (***next**).

```
typedef struct Job
{
	int id;
	struct Job* next;
} Job;
```

De forma consecutiva foi criada a estrutura Operação (***Operation***), com um identificador próprio, o trabalho ao qual esta se refere devidamente identificado, e a ordem na qual esta vai ser realizada dentro do próprio Trabalho.

```
typedef struct Operation
{
	int id;
	int jobID;
	int position;
	struct Operation* next;
} Operation;
```

Seguidamente, e pelo que foi proposto pelo enunciado, a estrutura Máquina (***Machine***), responsável por realizar as várias operações que constituem um trabalho. Cada máquina tem na sua estrutura um identificador próprio e a referência de estar ou não a ser utilizada no momento.

```
typedef struct Machine
{
	int id;
	bool isBusy;
	struct Machine* next;
} Machine;
```

Finalizando, chegamos à estrutura Execução (***Execution***). Foi o mecanismo utilizado para fazer a interligação dinâmica entre a utilização de uma máquina, numa determinada operação, num determinado intervalo de tempo. Dessa forma encerra precisamente na sua estrutura o identificador da operação, o identificador da máquina e o tempo necessário para esta executar o que a operação pede.

```
typedef struct Execution
{
	int operationID;
	int machineID;
	int runtime;
	struct Execution* next;
} Execution;
```

Através das execuções foi introduzida a tabela Hash que dá apoio à aplicação para organizar as várias listas utilizadas para responder ao problema. Primeiro definiu-se uma estrutura para representar cada posição na tabela Hash das execuções, com um apontador para o primeiro elemento de cada posição da tabela, e a quantidade de execuções que cada posição da tabela tem.

Em seguida é enunciada tabela Hash propriamente dita, composta por um **Array** em cada posição do Array vai apontar para uma lista especifica.

```
typedef struct {
	Execution* start;
	int numberOfExecutions;
} ExecutionNode;


extern ExecutionNode* executionsTable[HASH_TABLE_SIZE];
```

Finalmente, foi também introduzido o esboço do que pode ser uma solução à funcionalidade 8. Criou-se uma estrutura (***Cell***) composta por um identificador do trabalho e da operação, bem como o intervalo de tempo que a máquina (presente no eixo Y do Array onde depois os dados são mostrados) demora a realizar a tarefa, com tempo inicial e final apresentado.

```
typedef struct Cell
{
	int jobID;
	int operationID;
	int initialTime;
	int finalTime;
} Cell;

extern Cell plan[NUMBER_MACHINES][MAX_TIME];
```

Armazenamento e leitura de ficheiros (Process plan e alterações/actualizações ao mesmo)

Uma das funcionalidades pedidas no problema diz respeito ao armazenamento dos dados iniciais bem como de onde as alterações feitas em código ficam. Após ter sido debatida a questão em aula foi decidido na elaboração deste trabalho guardar o process plan inicial em ficheiro **.csv** e as alterações efetuadas armazenadas em memória. Inicialmente foi criado um ficheiro data.c, que se mantém no projeto sem funcionalidade (apenas documentado); por poder no futuro ser útil ao desenvolvimento deste projeto.

```
#define FILE_LINE_SIZE 50
#define JOBS_FILENAME_TEXT "text/jobs.csv"
#define MACHINES_FILENAME_TEXT "text/machines.csv"
#define OPERATIONS_FILENAME_TEXT "text/operations.csv"
#define EXECUTIONS_FILENAME_TEXT "text/executions.csv"

#define JOBS_FILENAME_BINARY "binary/jobs.bin"
#define MACHINES_FILENAME_BINARY "binary/machines.bin"
#define OPERATIONS_FILENAME_BINARY "binary/operations.bin"
#define EXECUTIONS_FILENAME_BINARY "binary/executions.bin"
```

As estruturas de dados para armazenar em ficheiros obedecem aos conteúdos de cada estrutura correspondente:

```
typedef struct FileJob
{
	int id;
} FileJob;

typedef struct FileOperation
{
	int id;
	int jobID;
	int position;
} FileOperation;
```

```
typedef struct FileMachine
{
	int id;
	bool isBusy;
} FileMachine;

typedef struct FileExecution
{
	int operationID;
	int machineID;
	int runtime;
} FileExecution;
```

Testes Realizados

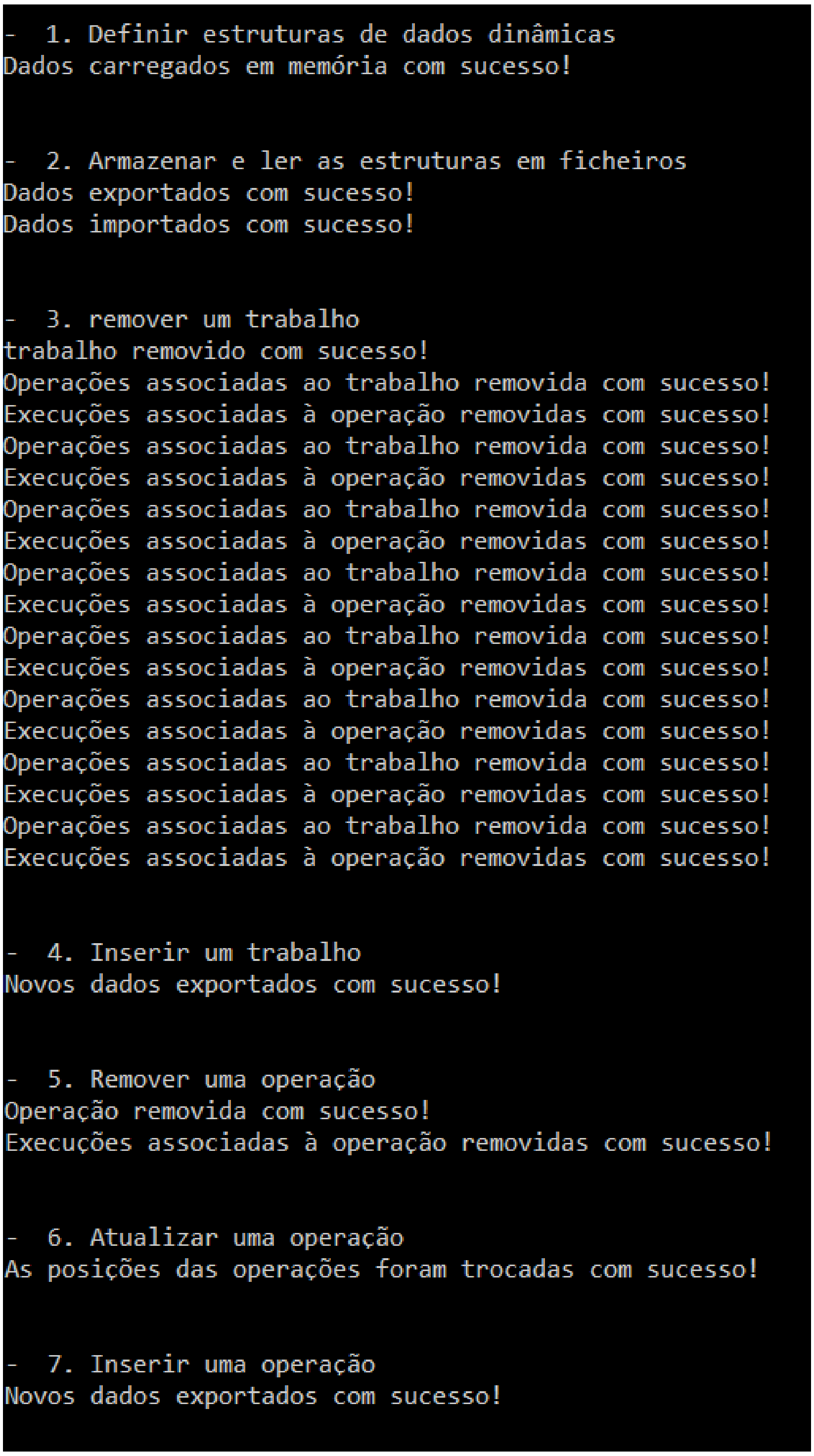
Os testes de software são uma componente bastante importante para qualquer empresa. Permitem otimizar uma aplicação na altura em que esta é gerada, e avaliar o seu grau de funcionalidade face ao público-alvo, seja ele o utilizador comum ou profissional habilitado.

Na tabela a seguir encontra-se a lista de requisitos, e a validação do funcionamento de cada um através de testes manuais ao software.

Fase 2	**Resultado**
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*)	

Fig.2 – Tarefas dos testes a realizar e sua respetiva concretização.

Compilando a aplicação, podemos observar que as funcionalidades pedidas no enunciado desta segunda parte foram concluídas com sucesso.



```
-  1. Definir estruturas de dados dinâmicas
Dados carregados em memória com sucesso!


-  2. Armazenar e ler as estruturas em ficheiros
Dados exportados com sucesso!
Dados importados com sucesso!


-  3. remover um trabalho
trabalho removido com sucesso!
Operações associadas ao trabalho removida com sucesso!
Execuções associadas à operação removidas com sucesso!
Operações associadas ao trabalho removida com sucesso!
Execuções associadas à operação removidas com sucesso!
Operações associadas ao trabalho removida com sucesso!
Execuções associadas à operação removidas com sucesso!
Operações associadas ao trabalho removida com sucesso!
Execuções associadas à operação removidas com sucesso!
Operações associadas ao trabalho removida com sucesso!
Execuções associadas à operação removidas com sucesso!
Operações associadas ao trabalho removida com sucesso!
Execuções associadas à operação removidas com sucesso!
Operações associadas ao trabalho removida com sucesso!
Execuções associadas à operação removidas com sucesso!
Operações associadas ao trabalho removida com sucesso!
Execuções associadas à operação removidas com sucesso!


-  4. Inserir um trabalho
Novos dados exportados com sucesso!


-  5. Remover uma operação
Operação removida com sucesso!
Execuções associadas à operação removidas com sucesso!


-  6. Atualizar uma operação
As posições das operações foram trocadas com sucesso!


-  7. Inserir uma operação
Novos dados exportados com sucesso!
```

Fig.3 - Apresentação da aplicação compilada.

Relativamente ao pedido no ponto 8 do enunciado, a formulação em base matemática a realização de todos os trabalhos no menor tempo possível tendo sempre em conta a premissa de que uma máquina só pode ser usada em exclusividade (ou seja, só faz uma tarefa integrante de uma operação de cada vez) não foi realizada.

A solução apresentada é a representação gráfica do que foi pedido, através da documentação dos dados necessários.

O exemplo documentado representa a realização da Operação 1 de todos os Jobs com os respectivos intervalos de tempo, em função das máquinas usadas para esse efeito (de 1 a 8).

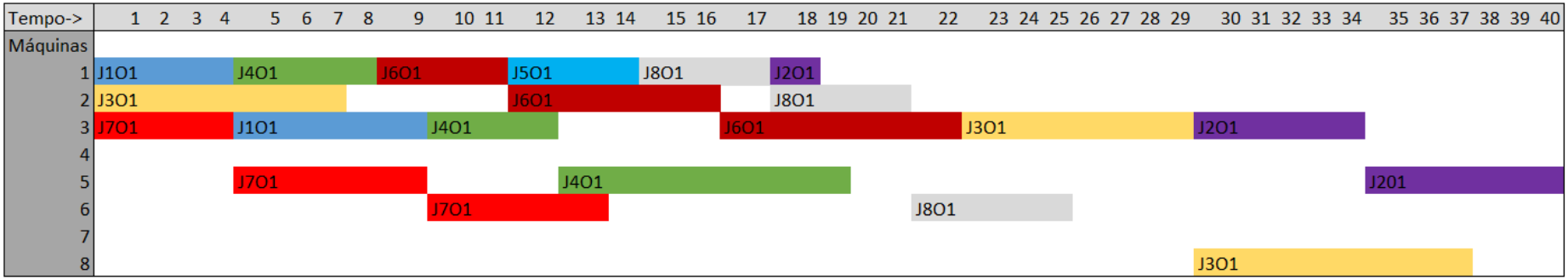


Fig.4 – Utilização das máquinas representado no decorrer do tempo.

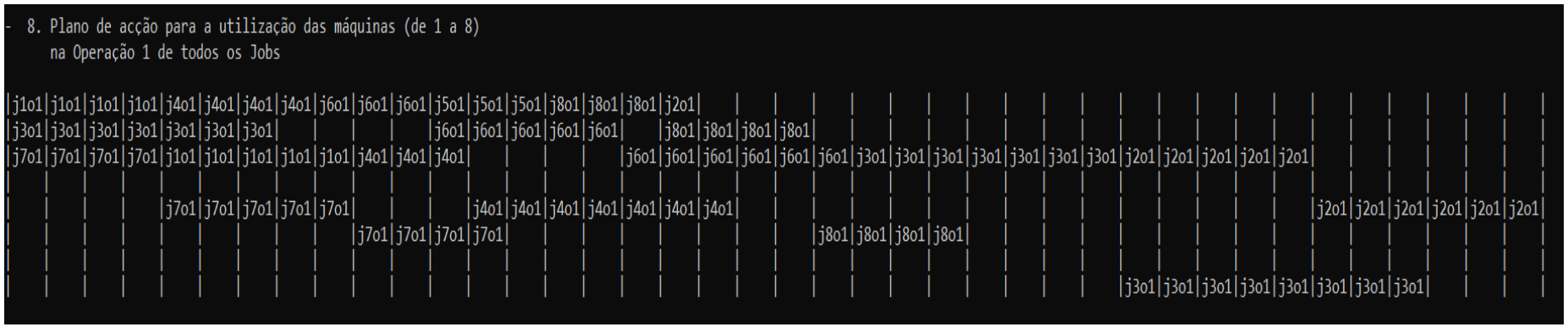


Fig.5 – Transcrição da figura acima para a aplicação.

Conclusão

Após a resolução da segunda parte deste trabalho, é possível tirar bastantes ilações sobre o que foi desenvolvido, positivas e negativas. Como ponto introdutório a esta conclusão é importante referir que o trabalho realizado pela docência nas aulas é ponto fundamental para a realização deste trabalho, por ter permitido que a compreensão e elaboração deste projeto tivesse menos entraves do que aqueles que se tinham no ponto inicial da disciplina.

Do ponto de vista pessoal, vinha do semestre anterior com muitas dificuldades em compreender e resolver problemas de programação. Essas dificuldades, apesar de ainda existirem, foram sendo diminuídas ao longo do processo de resolução do trabalho. Para isto contribuíram vários fatores. Para além do trabalho desenvolvido em aula (cuja importância foi referida acima), a própria estrutura de elaboração do trabalho contribuiu bastante para a evolução que senti a realizá-lo.

Por se tratar de um trabalho em que o Professor deu total liberdade para a realização do mesmo em grupo, tive a oportunidade de obter ajuda de várias pessoas e de receber auxílio de vários métodos de resolução do trabalho. Com isto associado ao pacote de conhecimento recebido nos exercícios práticos das aulas, tive a oportunidade de elaborar um código que, apesar de não estar perfeito, identifica o ponto de aprendizagem em que me encontro. Tive auxílio de várias colegas de turma para fazer o que fiz, colegas que estão em diferentes pontos de evolução na sua própria aprendizagem, e que me deram diferentes pontos de vista para o que eu escolhi que me identificasse como executor deste trabalho.

Para além disso, foi importante também perceber o grau de importância que tem uma documentação de código bem estruturada; tendo sido essa uma das minhas principais preocupações para a realização deste desafio.

Bibliografia

- Documentação fornecida pela docência via *Moodle*;

- https://www.doxygen.nl/download.html

- https://github.com

- https://visualstudio.microsoft.com

- Code analysis for C/C++ overview | Microsoft Docs

- PEREIRA, Alexandre. **C e Algoritmos.** 2ªEdição: Edições Sílabo, 2013.