RELATÓRIO DE TRABALHO PRÁTICO

FASE 2

PEDRO CUNHA NEVES

ALUNO Nº 21141

Trabalho realizado sob a orientação de:
Luís Ferreira e João Silva

Estruturas de Dados Avançadas

Licenciatura em Engenharia de Sistemas Informáticos

Barcelos, Maio de 2022

Índice

Introdução

- Este trabalho, teve como objetivo o desenvolvimento de aplicação em linguagem C, utilizando estruturas de dados dinâmicas.
- O trabalho foi dividido em duas partes, que correspondentes a diferentes etapas: Definição de estrutura de um job e operações, Definição de um conjunto finito de jobs e inserção dos mesmos.
- Na aplicação desenvolvida na primeira parte, o objetivo é definir um Txt e a realização de operações, tendo que aplicar listas ligadas multidimensionais desenvolvidas em linguagem de programação C.

Propósitos e Objetivos

- No âmbito da unidade curricular de Estruturas de dados avançadas, do segundo semestre do primeiro ano do curso de Engenharia de Sistemas Informáticos do Instituto Politécnico do Cavado e do Ave, no ano letivo 2021/2022 foi-nos proposto a elaboração de um trabalho prático que está dividido em duas partes.
- Este trabalho surge no contexto da avaliação contínua composta por um trabalho prático. O objetivo principal deste trabalho é levar ao melhor entendimento dos conhecimentos adquiridos ao longo do semestre e a melhor aplicação dos mesmos.
- Assim, com este trabalho pretendemos sedimentar mais os nossos conhecimentos sobre os temas abordados nas respetivas aulas.
- O trabalho realizado individual tem como objetivo a elaboração de um programa em C para o problema de escalonamento denominado Flexible Job Shop Problem (FJSSP).

Estrutura de Dados

O que é o Escalonamento em *Flexible Job Shop?*

- O Flexible Job Shop scheduling (FJSS) ou Flexible Job Shop problem (FJSP) é uma das variantes do JSP que permite que uma determinada operação seja realizada a partir de qualquer máquina de um determinado conjunto de máquinas.

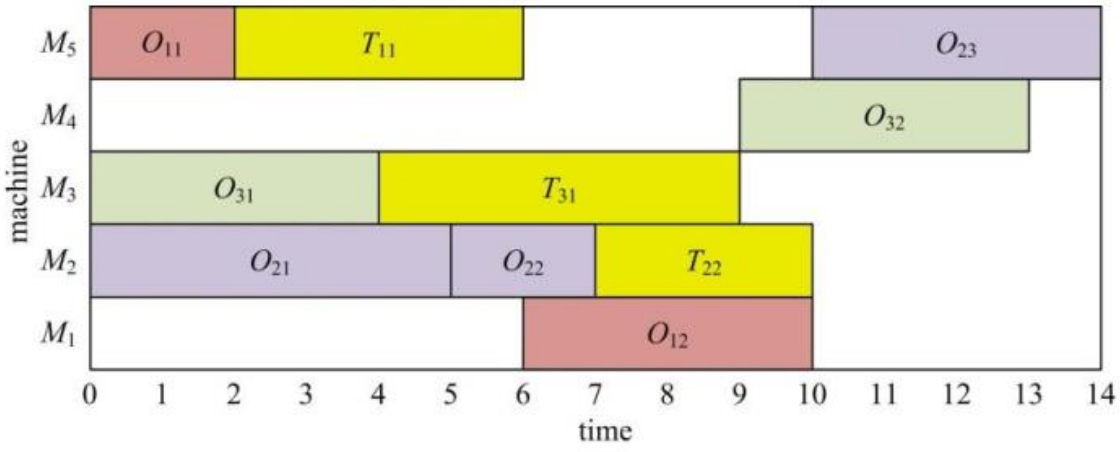
- O problema encontra-se na atribuição de cada operação a cada máquina e a sua ordenação, de forma que o tempo máximo de duração para conclusão de uma operação (makespan) seja minimizado.

- O FJSP pode ser descrito por n operações para serem processados em m máquinas. Cada operação necessita ser executada, e cada uma destas deve ser executada num tempo fixo numa determinada máquina.

- Existem várias restrições que devem ser consideradas, nomeadamente:

    1. Cada trabalho é constituído por um conjunto de operações.
    2. Cada operação só poderá passar numa máquina uma única vez.
    3. Têm que ser respeitadas as diferentes precedências de operações que poderão ocorrer.
    4. Cada máquina só pode processar uma operação de cada vez.

Exemplo:



Structs

Processo (job):

```
typedef struct Processo
{
    int id; //Contem o ID
    struct OperationExecution* OperationExecutions;
    struct Processo* next; //Proximo job/processo
}Processo;
```

- Estrutura para armazenar um processo;
- Um processo contêm um id e contêm um apontador para o próximo processo e uma apontador para a posição da operação;

OperationExecution:

```
typedef struct OperationExecution
{
    int operationID;
    int machineID;
    int usageTime;
    struct OperationExecution* next;
}OperationExecution;
```

- Estrutura para armazenar a execução de uma operação;
- Uma execução de uma operação contêm o id da operação(operationID),o id da maquina(maquinaID),a unidade de tempo(usageTime) e contêm um apontador para a próxima execução de uma operação;

Métodos

Processo:

```
//Inserir novo Processo
Processo* novoProcesso(int id);
//Inserir um processo no Inicio
Processo* inserirProcessoNoInicio(Processo* head, Processo* ProcessoDeInicio);
//Escrever para o ficheiro processos
bool escreverProcessos(char fileName[], Processo* head);
//Ler ficheiro de processos
Processo* lerProcessos(char fileName[]);
//Mostrar ficheiro de processos
bool mostrarProcessos(Processo* head);
//Atualizar um processo
Processo* atualizarProcesso(Processo* head, Processo* ProcessoParaAtualizar, int id);
//Eliminar um processo
bool eliminarProcesso(Processo** head, int id);
//Procurar um processo
bool procurarProcesso(Processo* head,int id);
//Procurar um processo
Operation* procurarOperationporID(Operation* head, int id);
//Apagar processos na memoria
bool apagarProcessos(Processo* head);
```

OperationExecution:

```
//Inserir nova execução de uma operação
OperationExecution* novoOperationExecution(int operationID,int machineID,int usageTime);
//Inserir uma execução de uma operação no Inicio
OperationExecution* inserirOperationExecutionNoInicio(OperationExecution* head,OperationExecution* operationExecutionDeInicio);
//Eliminar uma execução de uma operação
bool eliminarOperationExecution(OperationExecution** head,int operationID);
//Escrever para o ficheiro das execuções das operações
bool escreverOperationExecution(char fileName[],OperationExecution* head);
//Ler o ficheiro das execuções das operações
OperationExecution* lerOperationsExecution(char fileName[]);
//Mostrar o ficheiro das execuções das operações
bool mostrarOperationExecution(OperationExecution* head);
//Procurar uma execução de uma operação
bool procurarOperationExecution(OperationExecution** head,int operationID);
//Apagar execuções das operações na memoria
bool apagarOperationExecution(OperationExecution* head);
```

Extras:

```
//Lista o numero total de processos
int getListCount(Processo* head);
//Carrega os dados das listas
void loadData(Processo** processos,Maquina** maquinas,Operation** operations,OperationExecution** operationsExecution,ProcessoPart** processoparts);
//Calcular a media do tempo
int avgOperationExecution(OperationExecution* head,int operationID);
//Calcular o caminho mais curto para completar um job
bool TempoMinimoDaOperacao(char fileName[],Processo** head1,OperationExecution** head,int id,int operationID);
//Calcular o caminho mais longo para completar um job
int TempoMaximoDaOperacao(OperationExecution* head,int operationID);
// Maior Operation
int maiorOperation(OperationExecution* head);
//Eliminar Processo
bool eliminarProcessoOperationExecution(Processo** head,int id);
//Eliminar Processo e Operation
bool eliminarProcessoOperationExecutionOp(Processo** head,OperationExecution** head1,int id,int operation);
//Eliminar Processo e Operation
bool procurarProcessoOperationExecutionOp(Processo** head1,OperationExecution** head,int id,int operationID);
//Procurar Maquina
bool procurarProcessoOperationExecutionOpMaquina(Processo** head1,OperationExecution** head,int id,int operationID,int machineID);
//Atualizar uma operação
bool atualizarProcessoOperationExecution(Processo** head1,OperationExecution** head,int id,int operationID,int update);
//Atualizar uma maquina
bool atualizarProcessoOperationExecutionMaquina(Processo** head1,OperationExecution** head,int id,int operationID,int machineID,int update);
//Atualizar tempo de uma maquina
bool atualizarProcessoOperationExecutionTempo(Processo** head1,OperationExecution** head,int id,int operationID,int machineID,int update);
```

Conclusão

- Após terminar a realização do trabalho, consegui pôr em prática os conhecimentos dados nas aulas, embora tenha deparado com bastantes dificuldades.
- Apesar de todas as dificuldades encontradas neste trabalho, consegui superar algumas com sucesso.
- Este trabalho consegue-nos demonstrar um pouco o nível de dificuldade que podemos ter no exterior a nível profissional.
- Contudo, este trabalho vem me trazer mais experiência e conhecimento da disciplina, preparando-nos assim para a realização de mais trabalhos no futuro com outros programadores/empresas.

Bibliografia

- https://github.com/PedroNeves21141/EDA
- https://stackoverflow.com/
- https://elearning1.ipca.pt/2122/my/