

Estrutura de dados avançada

Mário pinto
(nrº 23506, regime diurno)

Orientação de

Identificação do aluno

Mário pinto

Aluno número 23506, regime diurno

Licenciatura em Engenharia em Sistemas Informáticos

Orientação

Resumo

Com este projeto de avaliação pretende-se sedimentar os conhecimentos introduzidos nas aulas da unidade curricular Estrutura de dados avançados.

Com este trabalho prático pretende-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.

Conteúdo

1	Introdução	1
1.1	Objetivos	1
2	GIT	2
2.1	Criar repositório	2
2.2	Iniciar o repositório	3
2.3	Adição da branch	3
2.4	Adição da branch	3
3	Estrutura de Dados e funções	5
3.1	struct utilizada	5
3.2	Assinatura no header	5
3.3	contabilizar	5
3.4	Inserir	5
3.5	Inserir	5
3.6	Remover	5
3.7	Procurar	5
3.8	Alterar	6
3.9	Listar	6
3.10	Listar	6
3.11	Puxar dados do ficheiro part1	6
3.12	Puxar dados do ficheiro part2	6
3.13	Menu	6
3.14	Calculo minimo	6
3.15	Calculo máximo	6
3.16	Calculo máximo	6
3.17	Calculo máximo	7
4	Programa em C	23

4.1	Função Main	23
4.2	Função Main	24
4.3	Função Main	24
5	Conclusão	27
6	Bibliografia	28

Lista de Figuras

2.1	Criação de repositório a partir do github	2
2.2	Primeiros passo para inicializar o repositório	3
2.3	Como se cria os branches	3
2.4	Github file system	4
3.1	struct	8
3.2	Assinaturas	9
3.3	Função quant	10
3.4	Função insert	11
3.5	Função insert	12
3.6	Função remove	13
3.7	Função search	14
3.8	Função change	14
3.9	Função list	15
3.10	Função list	16
3.11	Função pull	17
3.12	Função pull2	18
3.13	Função menu	19
3.14	Função calc	20
3.15	Função calcMax	20
3.16	Função Minop	21
3.17	Função Maxop	22
4.1	Função Main	23
4.2	Função Main2	25
4.3	Função Main3	26

1. Introdução

Nesta fase do trabalho é pedido para desenvolver uma solução digital para o problema de escalonamento denominado Flexible Job Shop Problem. Com este trabalho tencionamos aplicar tudo o aprendemos sobre estruturas e algoritmos.

1.1 Objetivos

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

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

2. GIT

Foi usado o GITHUB para fazer o controlo de versões do código.

2.1 Criar repositório

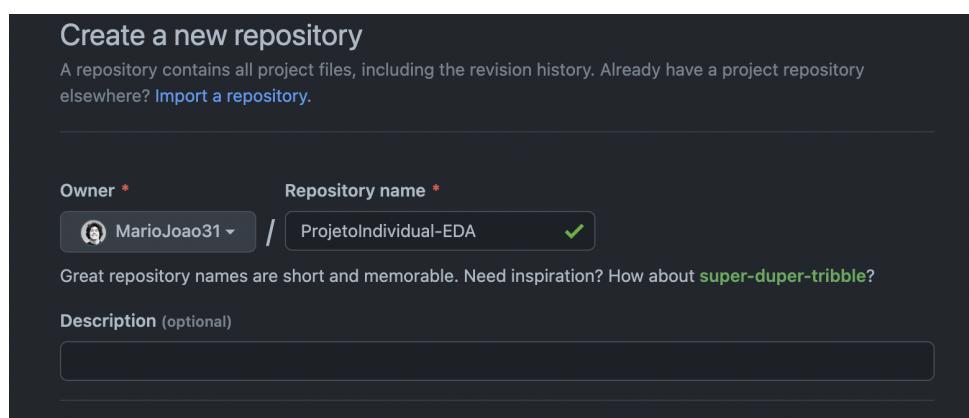


Figura 2.1: Criação de repositório a partir do github

2.2 Inicializar o repositório

Comando faz commit a tudo que fiz

```
apple ➤ ~/De/U/2/D/ProjetoIndividual ➤ git p master ?11 ➤ git add --all && git comm
it -m "add tudo que fiz"
apple ➤ ~/D/U/2/D/ProjetoIndividual ➤ 9% ➤ 11 GB
```

Figura 2.2: Primeiros passo para inicializar o repositório

2.3 Adição da branch

Os próximos comandos fazem uma branch nova e dão push aos meus commits

```
> git branch -M main
> git remote add origin https://github.com/MarioJoao31/ProjetoIndividual-EDA.git

> git push -u origin main
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 8 threads
Compressing objects: 100% (18/18), done.
Writing objects: 100% (19/19), 2.33 MiB | 3.64 MiB/s, done.
Total 19 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/MarioJoao31/ProjetoIndividual-EDA.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.

apple ➤ ~/De/U/2/D/ProjetoIndividual ➤ p main ➤ 14:20:05 ⌂
apple ➤ ~/D/U/2/D/ProjetoIndividual ➤ 9% ➤ 11 GB
```

Figura 2.3: Como se cria os branches

2.4 Adição da branch

A próxima foto é para amostrar como ficou tudo guardado no github.

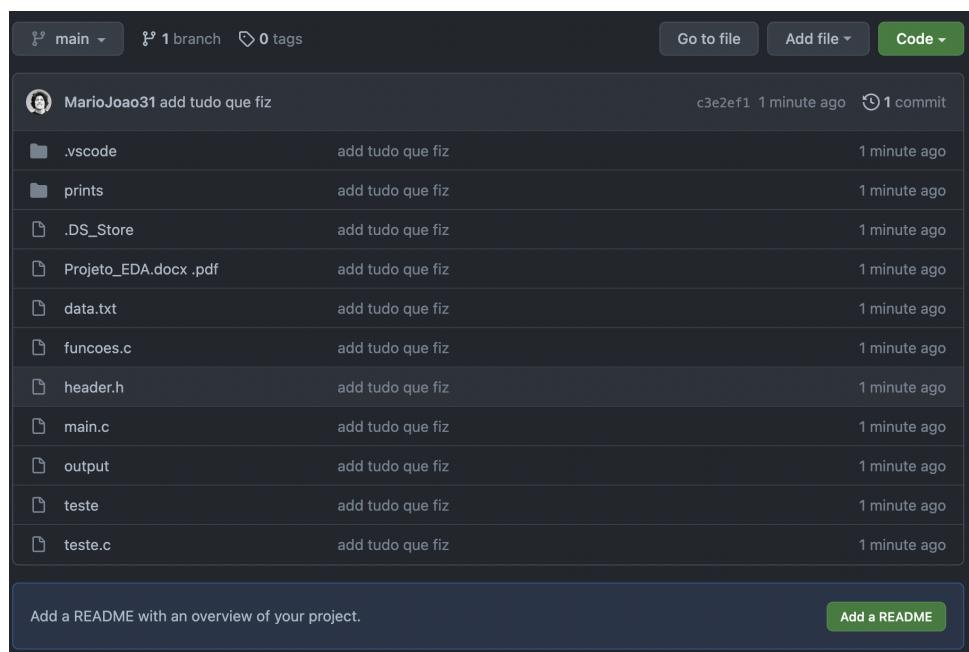


Figura 2.4: Github file system

3. Estrutura de Dados e funções

Eu decidi usar a estrutura desta maneira com id nas operações para ser mais fácil no futuro para adicionar mais operações e fazer alterações.

3.1 struct utilizada

3.2 Assinatura no header

3.3 contabilizar

Função para contabilizar as operações e jobs.

3.4 Inserir

Função para inserir um job com id das operações.

3.5 Inserir

Função para inserir múltiplas operações.

3.6 Remover

Função para remover múltiplas operações.

3.7 Procurar

Função para procurar uma operações.

3.8 Alterar

Função para alterar uma operações.

3.9 Listar

Função para listar jobs.

3.10 Listar

Função para listar operações.

3.11 Puxar dados do ficheiro part1

Função para adicionar no programa os dados do ficheiro.

3.12 Puxar dados do ficheiro part2

Função para adicionar no programa os dados do ficheiro.

3.13 Menu

Função para dar display ao menu.

3.14 Calculo minimo

Função para calcular a media mínima de um job.

3.15 Calculo máximo

Função para calcular a maxmimo de tempo.

3.16 Calculo máximo

Função para calcular o mínimo de tempo por operação.

3.17 Calculo máximo

Função para calcular o máximo de tempo por operação.

```
struct Job;
struct Operation;

You, 3 hours ago | 1 author (You)
typedef struct jp{
    int id;
    int operacao[7];
    struct jp *seguinte;
} Job;

You, 3 hours ago | 1 author (You)
typedef struct op{
    int id;
    int maquina[MAXMAQUINAS];
    int tempo[MAXTEMPO];
    int sizeMT;
    struct op *seguinte;
} Operation;
```

Figura 3.1: struct

```
//assinaturas

//JOBS
Job *inserirJobs(Job *jp, int id, int operacao[MAXOPERATION]);
void listarJobs(Job *jp);
int quantidadeJobs(Job *jp);
void guardarJobs(Job *jp);
void medMinJob(Job *jp, Operation *op);
void medMaxJob(Job *jp, Operation *op);

//OPERAÇÕES
Operation *inserirOperacoes(Operation *op, int id, int maq[MAXMAQUINAS], int temp[MAXTEMPO], int size);
Operation* procuraOperacoes(Operation* op, int id);
Operation* removerOperacoes(Operation * op, int id);
Operation* alteraOperacao(Operation* op, int id, int* maq, int* temp);
void listarOperations(Operation *op);
int quantidadeOperacoes(Operation * op);      You, 3 hours ago • add tudo que fiz ...
int minOperacao(Operation *op, int id);
int maxOperacao(Operation *op, int id);

//OUTROS
int menu();
Operation* pullFicheiro(Operation *op, int idCont);

#endif
```

Figura 3.2: Assinaturas

```
int quantidadeJobs(Job * jp) {
    int soma = 0;
    while (jp != NULL) {
        jp = jp -> proximo;
        soma++;
    }
    return (soma);
}

int quantidadeOperacoes(Operation * op) { You, 3 h
    int soma = 0;
    while (op != NULL) {
        op = op -> proximo;
        soma++;
    }
    return (soma);
}
```

Figura 3.3: Função quant

```
//INSERIR

Job* inserirJobs(Job * jp, int id, int* operacao){

    Job *jb = (Job*) malloc(sizeof(Job));

    if (jb!=NULL)
    {
        jb->id = id;
        //copiar array
        int sizearrayoperacao=sizeof(operacao);
        for(int h=0;h<sizearrayoperacao;h++){
            jb->operacao[h] = operacao[h];
        }

        jb->seguinte = jp;
        return(jb);
    }
    else return(jp);
}
```

Figura 3.4: Função insert

```

//operação
Operation* inserirOperacoes(Operation * op, int id, int* maq, int* temp, int size) {
    Operation *ot = (Operation*) malloc(sizeof(Operation));
    ot->id=id;

    // passa de um array para o outro /maquina/
    for(int i=0; i<size;i++){
        if(maq[i]!=0){
            ot->maquina[i]=maq[i];
        }
    }

    // passa de um array para o outro /tempo/
    for(int j=0; j<size;j++){
        if(temp[j]!=0){
            ot->tempo[j]=temp[j];
        }
    }

    ot->sizeMT=size;
    ot->seguinte=NULL;
    if(op == NULL){
        op= ot;
    }else{
        Operation* lastnode = op;
        while(lastnode->seguinte != NULL){
            lastnode = lastnode->seguinte;
        }
        lastnode->seguinte = ot;
    }
    return op;
}

```

Figura 3.5: Função insert

```
Operation* removerOperacoes(Operation * op, int id){

    if( op == NULL) return NULL;

    if(op->id == id){
        Operation* aux = op;
        op=op->seguinte;
        free(aux);
    }
    else{
        Operation *aux=op;
        Operation *auxAnt = aux;
        while(aux && aux->id != id){
            auxAnt=aux;
            aux = aux->seguinte;
        }
        if(aux != NULL){
            auxAnt->seguinte= aux->seguinte;
            free(aux);
        }
    }
    return op;
}
```

Figura 3.6: Função remove

```

Operation* procuraOperacoes(Operation* op, int id){
    if(op==NULL) return NULL;
    else{
        Operation* aux = op;
        while(aux != NULL){
            if(aux->id == id){
                return aux;
            }
            aux= aux->seguinte;
        }
        return NULL;
    }
}

```

Figura 3.7: Função search

```

Operation* alteraOperacao(Operation* op, int id,int* maq,int* temp){
    Operation* aux = procuraOperacoes(op, id);
    if(aux != NULL){
        // insere as maquinas de novo
        int sizearraymaq= sizeof(maq);
        // passa de um array para o outro /maquina/
        for(int i=0; i<sizearraymaq;i++){
            aux->maquina[i]=maq[i];
        }

        //insere o tempo de novo
        int sizearraytemp= sizeof(temp);
        // passa de um array para o outro /maquina/
        for(int i=0; i<sizearraytemp;i++){
            aux->tempo[i]=temp[i];
        }

    }
    return op;
}

```

Figura 3.8: Função change

```
void listarJobs(Job *jp){
    while (jp!=NULL)
    {
        printf("Job nº%d\n",jp->id );
        for(int j=0; j<7;j++){
            if(jp->operacao[j]==0) break;
            printf(" Lista de operações:%d\n", jp->operacao[j]);
        }
        jp = jp->seguinte;
    }
}
```

Figura 3.9: Função list

```

void listarOperations(Operation *op){
    while (op!=NULL)
    {
        printf("Operação nº%d\n",op->id );

        //lista as maquinas que se pode usar
        printf(" Lista de Maquinas:");
        for(int j=0; j<op->sizeMT;j++){
            if(j == op->sizeMT){
                printf("%d", op->maquina[j]);
                break;
            }else
                printf(", ", op->maquina[j]);
        }
        printf("\n");

        //lista o tempo que cada maquina demor
        printf("Tempo de cada Maquina:");
        for(int j=0; j<op->sizeMT;j++){
            if(op->tempo[j+1]==0){
                printf("%d", op->tempo[j]);
                break;
            }else
                printf(", ", op->tempo[j]);
        }
        printf("]\n");
        printf("Size:%d\n",op->sizeMT);

        op = op->seguinte;
    }
}

```

Figura 3.10: Função list

```
Operation* pullFicheiro(Operation *op, int idCont) {
    Operation *opP;
    FILE *f_JOB = fopen("data.txt","r");
    char symb ;
    unsigned char symbI;
    int i = 0, cont = 0, success = 0, arrayM[100], arrayT[100];

    You, 4 hours ago • add tudo que fiz ...
    if(f_JOB != NULL) {
        do {
            if((symb = getc(f_JOB)) != EOF) {
                i = cont = 0;
                Operation *ot=(Operation *)malloc(sizeof(Operation));

                //primeira linha le as maquinas
                while ((symb=getc(f_JOB))!='\n') {
                    symbI = (int) symb;
                    if(symbI >= '0' && symbI <='9') {
                        arrayM[i] = symbI - '0';
                        cont++;
                        i++;
                    }
                }

                i=0;

                //segunda linha le os tempos
                //BUG: tem de ter uma linha sem nada no ficheiro pq senao o
                while ((symb=getc(f_JOB))!='\n') {
                    symbI = (int) symb;
                    if(symbI >= '0' && symbI <='9') {
                        arrayT[i] = symbI - '0';
                        i++;
                    }
                }
            }
        }
    }
}
```

Figura 3.11: Função pull

```
        }

    }

    idCont++;
    ot->id = idCont;
    ot->sizeMT = cont;
    for(i=0; i < cont; i++) {
        ot->maquina[i] = arrayM[i];
        ot->tempo[i] = arrayT[i];
    }
    ot->seguinte = NULL;

    if(op == NULL){
        op= ot;
    }else{
        Operation* lastnode = op;
        while(lastnode->seguinte != NULL){
            lastnode = lastnode->seguinte;
        }
        lastnode->seguinte = ot;
    }

    success = 0;

    listarOperations(ot);
}

else{
    success = 1;
}

}
}while(success == 0);
}

return (op);
```

Figura 3.12: Função pull2

```
int menu(){
    int opcao;

    do{
        printf("-----MENU-----\n");
        printf("1 - Inserir job com operações\n");
        printf("2 - Quantidade de jobs\n");
        printf("3 - Listar jobs\n");
        printf("4 - Remover operação\n");
        printf("5 - Alterar operação\n");
        printf("6 - Listar operações\n");
        printf("7 - Guardar \n\n");
        printf("8 - Media minima por job\n");
        printf("9 - Media maxima por job\n");
        printf("10 - Pull dados do ficheiro\n");
        printf("11 - Inserir so operações\n");
        printf("0 - Sair\n");
        printf("Opção:");
        scanf("%d", &opcao);
    }
    while ((opcao>12) || (opcao<0));
    return(opcao);
}
```

Figura 3.13: Função menu

```
//CALCULOS
void medMinJob(Job *jp,Operation *op){
    int min;
    int incre=0;      You, 4 hours ago • add tudo que fiz ...
    while (jp!=NULL){

        printf("Job nº%d tempo minino:\n",jp->id );
        for(int j=0; j<8;j++){
            if(jp->operacao[j]==0) break;
            printf("Operação nº:%d\n",jp->operacao[j]);

            //inserir aqui a funcao de retornar a operacao min
            min = minOperacao(op,jp->operacao[j]);
            incre += min;
        }
        printf("O minimo de tempo é de %d minutos\n",incre+1);
        jp = jp->seguinte;
    }
}
```

Figura 3.14: Função calc

```
void medMaxJob(Job *jp,Operation *op){
    int max;
    int incre=0;
    while (jp!=NULL){

        printf("Job nº%d tempo maximo:\n",jp->id );
        for(int j=0; j<8;j++){
            if(jp->operacao[j]==0) break;
            printf("Operação nº:%d\n",jp->operacao[j]);

            //inserir aqui a funcao de retornar a operacao max
            max = maxOperacao(op,jp->operacao[j]);
            incre += max;
        }
        printf("O maximo de tempo é de %d minutos\n",incre-1);
        jp = jp->seguinte;
    }
}
```

Figura 3.15: Função calcMax

```
int minOperacao(Operation *op, int id){
    Operation* aux = procuraOperacoes(op, id);
    int min=100;
    int temp=0;
    //verifica se é null
    if(aux!=NULL){
        printf("%d",op->sizeMT);
        //le todos os tempos e guarda o maximo
        for(int i=0;i<op->sizeMT;i++){
            if(aux->tempo[i] < min){
                min=aux->maquina[i];
                temp=aux->tempo[i];
            }
        }
        printf("Maquina nº%d é a mais rápida\n",min+1);
        return (temp);
    }
}
```

Figura 3.16: Função Minop

```
int maxOperacao(Operation *op, int id){
    Operation* aux = procuraOperacoes(op, id);
    int max=0;
    int temp=0;
    //ler todos os tempos

    if(aux!=NULL){
        printf("%d", op->sizeMT);
        for(int i=0; i<op->sizeMT; i++){
            if(aux->tempo[i] > max){
                max=aux->maquina[i];
                temp=aux->tempo[i];
            }
        }
    }

    printf("Maquina nº%d é a mais rápida\n", max+1);
    return (temp);
}
```

Figura 3.17: Função Maxop

4. Programa em C

4.1 Função Main

Função main parte 1.

```
int main(){
    Job *jobs = NULL;
    Operation * operacoes = NULL;

    int opcao;
    int idCount=0;
    int qt=0;
    int aa[MAXOPERATION]={1,2,3};
    int bb[3]={1,2,3};
    int cc[5]={7,4,9};
    int rr[3]={2,6,5};
    int tt[3]={6,5,5};
    int kk[1]={3};
    int ii[1]={5};
    int hh[2]={6,6};

    do{
        opcao = menu();
        switch(opcao){
            case 1:
                //inserir Job
                printf("A inserir jobs...\n");
                jobs=inserirJobs(jobs,1,aa);

                operacoes= inserirOperacoes(operacoes,1,bb,cc,3);
                operacoes= inserirOperacoes(operacoes,2,rr,tt,3);
                operacoes= inserirOperacoes(operacoes,3,kk,ii,1);
        }
    }while(opcao!=5);
}
```

Figura 4.1: Função Main

4.2 Função Main

Função main parte 2.

4.3 Função Main

Função main parte 3.

```
        break;
case 2:
    printf("Quantidade: %d\n", quantidadeJobs(jobs));
    break;
case 3:
    listarJobs(jobs);
    break;
case 4:
    printf("##### remover operação #####\n\n");
    removerOperacoes(operacoes,2);
    break;
case 5:
    printf("##### Alterar Operação #####\n\n");
    alteraOperacao(operacoes,3, hh, bb);
    break;
case 6:
    //insere e depois lista
    //tenho de passar arrays no parametros para maquir
    listarOperations(operacoes);
    break;
case 7:
    //guardar ainda nao fiz
    //TODO:Ainda não fiz e nao funciona
    //
    break;
case 8:
    printf("##### tempo minimo do job #####\n");
    medMinJob(jobs,operacoes);
    break;
case 9:
    printf("##### tempo minimo do job #####\n");
    medMaxJob(jobs,operacoes);
    break;
```

Figura 4.2: Função Main2

```
case 10:  
    // vai ler ao ficheiro  
    printf("Puxa os dados do ficheiro\n");  
    operacoes = pullFicheiro(operacoes, idCount);  
    break;  
case 11:{  
    //ve quantas operacoes tem inseridas para ter o id correto  
    idCount=quantidadeOperacoes(operacoes);  
    idCount++;  
    //inserir operacoes e ver a quantidade de maquinas  
    system("clear");  
    printf("####Inserir operações###\n");  
    printf("Quantas maquinas:");  
    scanf("%d",&qt);  
  
    int maq[qt];  
    int temp[qt];  
    for(int i=0; i<qt; i++){  
        printf("\nMaquina numero:");  
        scanf("%d",&maq[i]);  
        printf("tempo da maquina %d:",i+1);  
        scanf("%d",&temp[i]);  
    }  
  
    operacoes=inserirOperacoes(operacoes,idCount,maq,temp,qt);  
  
    break;
```

Figura 4.3: Função Main3

5. Conclusão

Com este trabalho consegui aplicar o que aprendi nas aulas sobre os algoritmos e sobre as estruturas.

6. Bibliografia

<https://www.geeksforgeeks.org/maximum-and-minimum-in-an-array/>
<https://stackhowto.com/c-program-to-search-an-element-in-an-array/>
<https://www.geeksforgeeks.org/find-the-minimum-distance-between-two-numbers/>