Relatorio do Trabalho de Implementação 2

Grupo: Caio Silas de Araujo Amaro(21.1.4111)

15 de Maio,2022

1 Implementação:

Começando pela implementação eu pensei e fazer duas struct uma que contém todos os casos e os médicos de plantão e outra que usa essa struct e contém o nume da upa.

```
struct caso
{
int grave;
int medio;
int normal;
int plantao;
};
//struct que contém uma struc de caso e o nome das upas
struct upa
{
Tcaso caso;
char nome [20];
};
```

Depois de fazer a alocação da struct e de preenchê-la, eu uso uma função ordena(merge Interativo) e também uso uma função compara ,para saber qual caso tem precedência e retorno 1 para o caso x e 2 para o caso y, já na função ordena eu tenho um if para caso o retorno seja 2 fazer a troca no vetor. Esse método foi escolhido pois após conversar com meus colegas sobre o trabalho acabei optando por ele, pois apesar do trabalho ter entradas curtas o merge consegue se sair muito bem e é um método de ordenação estável mesmo levando mais tempo para terminar sua execução.

```
int compara(Tupa x, Tupa y){
//casos graves sejam maiores
if (x.caso.grave != y.caso.grave){
if (x.caso.grave > y.caso.grave){
return 1;
}
}
//casos graves sejam iguais deve se comparar os casos medios
else if(x.caso.medio != y.caso.medio){
if (x.caso.medio > y.caso.medio){
return 1;
}
}
//casos grave e medios sejam iguais, deve-se comparar os normais
else if (x.caso.normal != y.caso.normal){
if (x.caso.normal > y.caso.normal){
return 1;
}
}
//casos todos sejam iguais, a preferencia e da upa que tem menos medicos de plantao
else if (x.caso.plantao != y.caso.plantao){
if (x.caso.plantao < y.caso.plantao){
return 1;
}
```

```
}
//casos tudo seja igual a preferencia e por ordem alfabetica
else if (strcmp(x.nome,y.nome) < 0){
return 1;
}
return 2;
}
//ordenando o vetor
```

//ordenando o vetor void merge(Tupa **v,int l,int m, int r) int tamE = m - l + 1; int i,j; int tamD = r - m; Tupa *vetorE = malloc((tamE)*sizeof(Tupa)); Tupa *vetorD= malloc((tamD)*sizeof(Tupa)); for (i = 0; i ¡ tamE; i++) vetorE[i] = (*v)[i + l]; for (j = 0; j ¡ tamD; j++) vetorD[j] = (*v)[m + j + 1]; i = 0; j = 0; int k; for (k = l; k ¡= r; k++) if (i == tamE) (*v)[k] = vetorD[j++]; else if (j == tamD) (*v)[k] = vetorE[i++]; else if (compara(vetorE[i],vetorD[j]) == 1) (*v)[k] = vetorE[i++]; else (*v)[k] = vetorD[j++];

free(vetorE); free(vetorD); //ordena por merge int mergeint(Tupa **v, int l, int r,int n) int b; int esq,dir; b = 1; while (b ¡ n) esq = 0; while ((esq + b) ¡ n) dir = esq + (2 * b); if (dir ¿ n) dir = n; merge(v,esq,(esq + b) - 1,dir - 1); esq = esq + (2 * b); b = b * 2; return 0;

Depois de feito todas essas funções eu as chamo no meu main na seguinte ordem e finalizo o meu código.

```
int main(){
int qnt;
//lendo a quantidade de upas que serao alocadas
scanf("%d",&qnt);
Tupa *upa = NULL;
//alocando a struct de upa
alocaUpa(&upa,qnt);
//preenchendo a struct
Completaupa(&upa,qnt);
//ordenando a struct
mergeint(&upa,0,qnt,qnt);
//printando apos ter feito a ordenacao e colocado por ordem de preferencia
printupa(&upa,qnt);
//liberando a struct que foi alocada
free(upa);
return 0;
}
```

2 Impressões Gerais:

A implementação do projeto foi pensada antes de começar o trabalho, apesar de usar um dos métodos mais simples de ordenação de vetores eu gostei bastante de implementar, além das funções que foram apresentadas também optei por adicionar mais algumas. Uma das maiores dificuldades que tive foi saber se o programa funcionaria dentro do tempo solicitado.

3 Análise:

Após analisar as saídas pode se concluir que o método se mostrou muito eficaz para a tarefa proposta.

4 Conclusão:

Ao concluir o trabalho pude notar que minha maior dificuldade foi a liberação de memória com a ajuda da ferramenta valgrind,conseguir encontrar e corrigir todos meus erros de liberação.