UNIVERSIDADE DE BRASILIA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA: Estrutura de Dados **TURMA E**
1º SEMESTRE 2016

Alunos: Rodrigo Demetrio Palma – 15/0147384
Christian Luís Marcondes Costa – 15/0153538

Apresentação

Elevadores são encontrados em diversos lugares como prédios residenciais, empresas, fábricas e foram desenvolvidos para acelerar a locomoção das pessoas. Estas máquinas operam a partir de regras estabelecidas por um algoritmo, o qual procura-se maior eficiência. Para medir a eficiência de um elevador, submete-se ele a diversas chamadas aleatórias e calcula-se o tempo gasto para realizá-las.

Descrição sucinta sobre o desenvolvimento do trabalho

Este trabalho foi desenvolvido de maneira a aperfeiçoar as habilidades em programação, em especial o uso de TAD's, por isso resolveu-se fazer o método **FIFO** (First in, first out) o qual é um dos menos eficientes, porém funcional.

Descrição dos módulos e sua interdependência

O projeto contém 4 módulos:

1. *main.c* - onde está o programa principal e sua lógica;
2. *funções.c* - contendo as funções utilizadas no programa principal;
3. *funções.h* - onde estão declarados os Tipos Abstratos de Dados (TAD's) e os protótipos das funções que foram usados no programa;
4. *makefile* - onde estão as regras para a compilação do programa executável, a partir do utilitário make.

Descrição dos TADs e as estruturas de dados utilizadas

O programa foi feito a partir do TAD Lista, que possui variáveis do tipo inteiro as quais representam os dados de uma pessoa - ID, andar atual, andar destino, hora da chamada e tempo de espera pelo elevador (wait_time) - e uma variável do tipo ponteiro, que apontará sempre para o próximo item da lista.

A lista utilizada engloba as seguintes funções e suas utilidades:

1. Inicia - Inicia uma lista vazia;
2. Aloca_Vetor - Esta função aloca dinamicamente um espaço do tamanho de um item da lista e adiciona os dados de um vetor;
3. InsereFim_Vetor - Insere os dados de um vetor no final da lista;
4. Imprime_lista - Imprime os itens de uma lista;
5. ChecaTempo - Incrementa o tempo de espera do elevador em zepslon;
6. BubbleSort - Ordena um vetor em ordem crescente;
7. Libera - Desaloca os itens da lista;

Descrição do formato de entrada dos dados

O formato de entrada dos dados deve ser a partir de um arquivo texto (.txt) onde as entradas estarão separadas por um espaço e digitadas na seguinte ordem:

1. ID;
2. Hora da chamada;
3. Andar inicial;
4. Andar destino;

Se por acaso o usuário não digitar um arquivo texto, o programa gerará um arquivo "*rotina.txt"* com algumas entradas aleatórias, onde ele poderá ver o programa funcionando.

Descrição do formato de saída dos dados

Os dados de saída serão mostrados ao usuário via monitor, de modo que cada pessoa, tem associado a ela, um número *zepslon* que representa o tempo que ela esperou para chegar ao destino.

Explicação sobre como utilizar o programa

Para utilizar o programa é necessário ter os arquivos *main.c*, *funcoes.c*, *funcoes.h* e o *makefile.txt*. Para o funcionamento do programa é necessário que todos esses arquivos estejam no mesmo diretório. Feito isso, o usuário precisa abrir o diretório que contém todos os arquivos pelo terminal e digitar *make,* e após isso o usuário deve digitar *./elevador rotina.txt* onde rotina.txt é o nome do txt que é dado ao arquivo de texto que será usado para execução do programa. Feito isso o programa executará,

mostrando os dados pedidos na tela e gerando um arquivo .txt no diretório do programa**.** Não é necessário digitar o rotina.txt para a execução do programa.

Estudo da complexidade do programa

Uma das funções usadas no programa é o *bubblesort* que recebe como argumentos a variável *vetor* e *Npessoas*. Essa função é de complexidade n^2.

A função *insereFim_vetor* recebe dois argumentos, sendo um do tipo *node** e outra do tipo *node*. Essa função possui complexidade de n. Da mesma forma, a função *imprime_lista* é de complexidade n.

A condicional *if (argv[1] == NULL)* na linha 42 contém *for(i = 0; i < Npessoas; i++)* na linha 54. O mesmo possui complexidade de n. Além disso temos o *for(aux = pessoas; aux != NULL; aux=aux->prox)* na linha 115 que é de complexidade *n*m* tendo em vista que *m* é o número de andares a ser percorrido pelo elevador e *n* é o número de membros na lista.

A função *checatempo* também recebe dois argumentos, um tipo *node** e um tipo *inteiro* . Em seguida ela utiliza um laço que percorre uma lista do início ao fim checando se o tempo decorrido é maior que a hora da chamada. Está função tem complexidade n.

As demais funções são de ordem de complexidade de *O(1).*

Com isso podemos concluir que a complexidade do programa é de n^2 já que a parte mais complexa é de ordem de n^2.

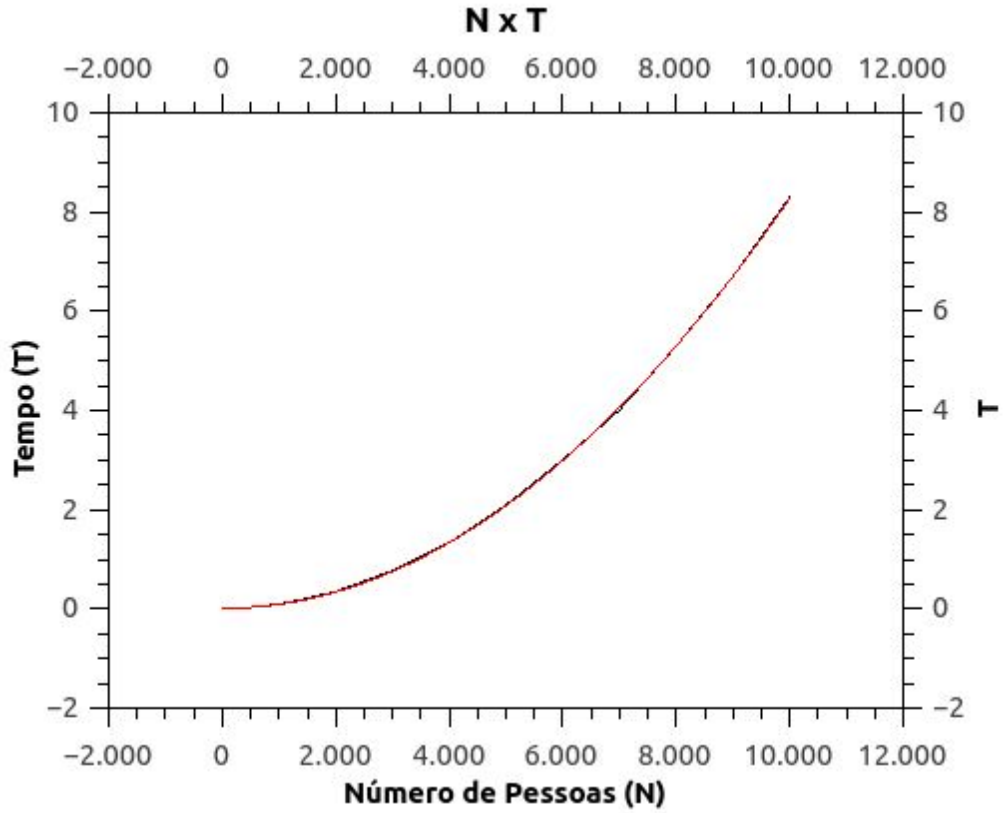
Listagem dos testes executados

Foram feitas várias medidas com o número fixo de andares igual a 50. Os resultados estão mostrados abaixo:

Número de pessoas	Tempo de Execução (s)
10	0
50	0
100	0.004
250	0.012
500	0.0304
1000	0.0968
1500	0.212
2000	0.3488
3000	0.7672
4000	1.3328
5000	2.0944
6000	3.0024
7000	4.006
8000	5.2936

9000	6.6958
10000	8.3016

Em seguida foi feito um gráfico de T x N para visualizar o crescimento da função:



Utilizando uma regressão quadrática encontrou-se a seguinte equação do segundo grau:

$$T = 0,00904 + 0,00000104 * n + 8,25 * 10^{-8} * n^2$$

Conclusão

Pode-se concluir que, apesar de utilizar-se uma lógica simples como a *FIFO*, os resultados são aceitáveis, quando os valores de entrada são razoavelmente pequenos. Além disso, foram realizados vários testes para garantir um bom funcionamento do programa.