# Simulador de Elevador

Bárbara Boechat, Juliana Araújo

Universidade Federal de São João Del Rei, Setembro 2016

## 1 Introdução

Esse trabalho tem como objetivo a implementação de um simulador orientado a eventos do controlador de elevador, para incitar problemas complexos e a suas soluções. O ambiente do elevador consiste no número de andares do prédio e sua capacidade máxima de passageiros, e possui dois eventos a serem considerados:

1. O andar de chamada.
2. O andar de destino do passageiro.

Esses dois parâmetros que irão ditar as variáveis interessantes no problema, como o tempo que um passageiro passa dentro do elevador, e o tempo que ele passa fora, esperando-o, além do número de vezes que a porta abriu e a distância percorrida pelo elevador. O que será explicado num próximo tópico.

## 2 Ambiente

- Editor de textos ATOM
- Linguagem de programação utilizada: Linguagem C
- Testes foram realizados num notebook com processador intel i5 de quarta geração,com 6 gb de memória RAM e uma máquina virtual com Ubuntu 16.04 LTS 32Bits instalado.
- Ambiente de desenvolvimento da documentação: TeXnicCenter 1.0 SRC-Editor de LATEX.
- Os testes foram realizados em um notebook com processador intel i5 de quarta geração,com 6 gb de memória RAM e sistema operacional Ubuntu 15.10 64-bit. Foi utilizado o editor de textos ATOM, e o compilador gcc para linux.

# 3 Especificações do problema

O ambiente do elevador é descrito pela capacidade máxima de pessoas que dentro dele cabem, e o número total de andares do prédio a serem percorridos. Cada passageiro possui dois dados básicos, que é o andar em que ele se encontra, e o andar para o qual ele está indo.

1. Sempre que o elevador é chamado, o passageiro é sujeito a verificação de sua capacidade máxima.
2. Se estiver cheio o atendimento o passageiro deve esperar até que haja a possibilidade de seu chamado ser atendido.
3. É considerada a velocidade constante.
4. Os tempos são expressos em uma unidade de tempo Jepslon. Para percorrer a distância de um andar a outro, é necessário 1 Jepslon e para 1 ou mais passageiros entrarem ou saírem do elevador também é necessário 1 Jepslon.

E para isso foram desenvolvidas duas estratégias para a implementação do elevador.

## 3.1 Entradas

As entradas para o problema são dadas de acordo a dois arquivos base. Sendo eles, o descritor do ambiente do elevador, contendo a capacidade máxima e a quantidade de andares do prédio:

| | |
|---|---|
| X | Capacidade |
| Y | Num. Andares |

O segundo arquivo possui o andar de chamada e o andar de destino dos passageiros:

| X | Y |
|---|---|
| Chamada | Destino |

# 4 Estrategia 1

Para auxiliar na implementação da primeira estrategia, uma fila simplesmente encadeada foi implementada. Tendo em vista a alocação dinâmica e a não necessidade de inserção e ou retirada no meio, fez-se então mais sentido o uso do FIFO (first in first out), pois a primeira estratégia consiste em colocar pessoas no elevador até que a capacidade máxima seja alcançada e depois esvaziá-lo, deixando-as seguindo sua ordem de entrada.

O processo é repetido até que todas as pessoas tenham sido pegas e deixadas em seu destino.

## 4.1 Fila

A fila dinâmica é uma estrutura de dados que permite a inserção de dados sem que haja o conhecimento prévio do N número de elementos a serem inseridos. Ela é composta por dois apontadores que trabalham como descritores dos eventos ocorridos. Um deles marca qual é o primeiro elemento, pois nesse estilo de implementação, o primeiro a ser inserido será o primeiro a ser retirado. E o outro descritor marca qual é o último elemento, assim sendo, quando os dois descritores marcarem o mesmo item sabe-se que há apenas um item na fila.

Além disso, possui também um apontador para ligar cada uma de suas células ou nodos ao próximo nodo, e um dado base, que é composto pelo numero do andar de chamada e andar de destino. [1]

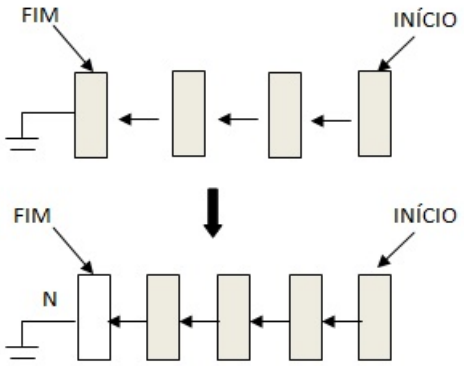


Figure 1: Lista Simp. Encadeada

## 4.2 Desenvolvimento

O programa está dividido entre pegar as pessoas em seus devidos andares e depois deixá-las em seu andar de destino. Quando uma pessoa desce, a próxima será a nova primeira. Os dados do elevador são obtidos da seguinte forma:

1. **Diferença (dif):** Variável auxiliar que calcula o módulo da diferença dos andares atual e o chamado;
2. **Ocupação do elevador (ocup):** Quando uma pessoa entra no elevador a variável é acrescentada, e quando o deixa, diminuída. Devida a lógica do programa do elevador obter sua capacidade máxima e depois esvaziá-lo, a ocupação sempre irá crescer até a capacidade, e depois decrescer até zero.
3. **Distancia total (distTotal):** Soma-se todas as diferenças, e obtêm quantos andares o elevador percorreu;

4. **Número de vezes que a porta foi aberta e fechada (porta):** Ao entrar ou sair uma pessoa, a variável porta é acrescentada. É importante dizer que a lógica desse algoritmo não considera o tempo de abrir e fechar a porta nem como tempo dentro, nem como tempo fora, para quem está entrando ou saindo do elevador;

Os dados que serão calculados em relação a cada pessoa estão em outra estrutura, que será alocada em um vetor (r) afim de facilitar sua visualização e utilização. Abaixo, as variáveis presentes nessa estrutura de resultado e a forma como foram calculadas:

1. **Tempo guardado da última ocupação (guardatFora):** Guarda o tempo da última pessoa que saiu do elevador. Isso será o tempo base de quem está fora;

2. **Tempo fora do elevador (tFora):** O tempo que a primeira pessoa passou fora do elevador será a diferença entre o andar atual e o que a pessoa está, mais o tempo guardado da última pessoa que saiu da ocupação anterior (guardatFora). Já as demais terão seu tempo calculado tendo como base o tempo que a última pessoa ficou fora do elevador, mais 1, que é o tempo dela sair do elevador, mais a diferença de andares, que é o tempo gasto para chegar até ela.

3. **Tempo dentro do elevador (tDentro):** É calculado em duas etapas, sendo a primeira no ato de pegar as pessoas em seu andar de chamada, e a segunda levá-las ao andar de destino.

    - Para a primeira parte, o tempo dentro é igual a diferença com o andar seguinte (tempo gasto para descer ou subir ao pegar os próximos passageiro) mais 1 (abrir e fechar da porta). Todos os passageiros anteriores devem receber o tempo gasto para se pegar o novo passageiro, para isso utiliza-se um laço de repetição.

    - Na segunda parte do processo, o tempo dentro da primeira pessoa será acrescido apenas o tempo gasto para chegar até ela. Já nas seguintes adiciona-se também o tempo de abrir e fechar a porta referente a essa pessoa descer.

# 5 Estrategia 2

## 5.1 Lista duplamente encadeada

A lista possui uma série de elementos, chamados também de nós, e é composta por diversas variáveis que guardam ou auxiliam nos cálculos necessários. Também possui dois ponteiros que ligam os nós uns aos outros. Sendo assim, essa lista é designada por "Duplamente encadeada", exatamente pelo fato de

possuir dois ponteiros de controle, ao contrário da simplesmente encadeada que possui apenas um ponteiro, que aponta para o próximo elemento da lista.

Possuindo os dois ponteiros, que marcam o nó anterior e o próximo nó é possível realizar diversas tarefas que seriam impossíveis em caso de encadeamento simples. Como por exemplo a inserção no meio da fila, e a retirada de qualquer nodo a qualquer momento. [2]

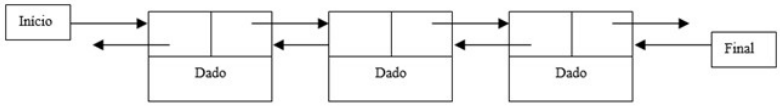


Figure 2: Duplamente encadeada

## 5.2 Desenvolvimento

A estratégia 2 busca aproximação com a realidade de um elevador e consiste em subi-lo até o último andar, pegando apenas os passageiros que irão subir e, em cada andar, verifica se alguém pode descer ou não. O processo inverso é feito na descida, ou seja, descer até o primeiro andar, pegando quem também irá descer, e sempre verificar se algum passageiro pode deixar o elevador naquele andar.

Enquanto a fila não estiver vazia, o elevador fará esse processo, já que quando alguém deixa o elevador é retirado da lista. Ele começará no andar 1 e subirá até o último, sempre verificando se alguém pode descer no andar em questão. Para fazer essa verificação, a variável *marcador* irá marcar 1 para a pessoa pega, então toda a fila será percorrida em busca de quem já foi pego e tem seu destino igual ao andar atual. Quando o passageiro descer, ele será removido da lista. Feito isso, haverá a tentativa de pegar alguém naquele andar.

A fila está ordenada em relação ao número de chamada do passageiro, a fim de facilitar esse processo. Por tanto, enquanto o ponteiro for diferente de nulo e o andar de chamada for igual ao atual, será verificado se lotação do elevador não foi atingida, se a pessoa está subindo (caso o elevador esteja no ato de subida), e se está fora do elevador. Sendo todas essas verificações verdadeiras, a pessoa entra no elevador.

Abaixo, as principais variável utilizadas para obter as informações necessárias, e como foram calculadas:

1. **Ocupação do elevador (ocup):** sempre que alguém deixa o elevador, é diminuída, e ao entrar é acrescida, caso a ocupação seja menor que a capacidade do elevador.

2. **Número de vezes que a porta foi aberta e fechada:** é acrescida sempre que alguém entra ou sai do elevador. Para que não seja aberta e fechada mais de uma vez no mesmo andar, há uma variável auxiliar (portaAberta) que guarda se a porta já foi aberta naquele andar. Após a mudança de andar, essa variável auxiliar é zerada.

3. **Distância total percorrida pelo elevador:** sempre que há mudança de andar, a distância é acrescida em uma unidade.

4. **Tempo de cada passageiro fora do elevador(tDentro):** há uma variável auxiliar (tForaTotal) que guarda o tempo de todas ações até o momento. Sempre que a porta é aberta ou há mudança de andar, essa variável é acrescida. Sendo assim, quando o passageiro entra no elevador o tempo que ele passou fora é exatamente o valor da variável *tForaTotal* naquele momento.

5. **Tempo de cada passageiro dentro do elevador (tDentro):** também é calculado de forma simples e com ajuda da variável que guarda todo o tempo total (tForaTotal). Quando o passageiro é removido do elevador, o seu tempo dentro será o tempo total menos o tempo que esse passageiro passou fora.

Vale ressaltar que mais uma vez, a porta não é contada como tempo dentro ou fora daquele passageiro no elevador. Os cálculos são os mesmos para o ato de descer o elevador. O andar é acrescido (subida) ou diminuído (descida) logo depois de verificar se alguém pode sair ou entrar no elevador.

# 6 Análise de resultados

Os dados distribuídos nos gráficos abaixo foram obtidos da média de execução de sete entradas diferentes aleatórias. O ambiente do elevador se manteve o mesmo a fim de realizar uma comparação eficaz. A capacidade do elevador foi de 15 pessoas para um prédio de 50 andares. Os testes foram feitos para 10, 100 e 1000 pessoas.

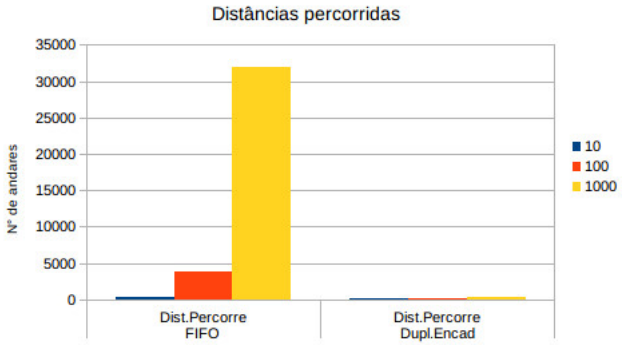


Figure 3: Distância Percorrida

A diferença entre a fila e a lista duplamente encadeada fica evidente neste gráfico, em que é possível perceber a eficiência da estratégia 2, que precisou

percorrer uma quantidade de andares significativamente menor para cumprir o mesmo objetivo da estratégia 1.

O fator principal dessa eficiência é a verificação dos passageiros, se poderiam entrar ou sair do elevador em cada andar, ao invés de apenas buscá-los e deixá-los seguindo sua ordem de chegada.

Isso foi possível pois a lista duplamente encadeada permitiu percorrer os dados em qualquer direção, e retirar os passageiros em qualquer posição. Outro fator importante foi a ordenação dos elementos, já que não foi preciso percorrer toda a fila em busca de alguém que solicitava o elevador naquele andar, para efetuar sua entrada. Todas essas técnicas impactaram no resultado final obtido pela estratégia 2.

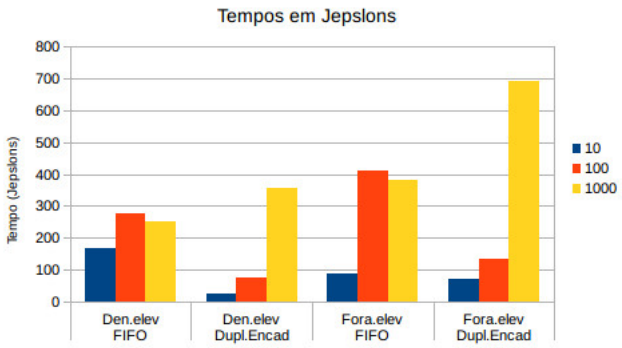


Figure 4: Tempos em Jepslons

A lógica mais aproximada da realidade de um elevador na estratégia 2 também teve impacto no tempo de espera dos passageiros para chegarem ao seu destino. Tendo o elevador percorrido uma distância menor, em relação a estratégia 1, consequentemente, o tempo de espera de quem está dentro e fora do elevador também será menor. Porém essa lógica é válida apenas para a entrada pequena e média. Para a entrada maior, de 1000 passageiros, que já foge da realidade, os tempos da estratégia 2 foram maiores.

Apesar da estratégia 2 previamente ordenar seus elementos, e ainda precisar percorrer todo a lista em cada andar, a fila teve maior tempo de usuário. Isso porque a estratégia 1 se torna muito dependente da capacidade do elevador, e seu número de execuções depende da mesma. Ainda, há maior complexidade em se calcular os tempos dos passageiros, necessitando de um maior número de laços de repetição.

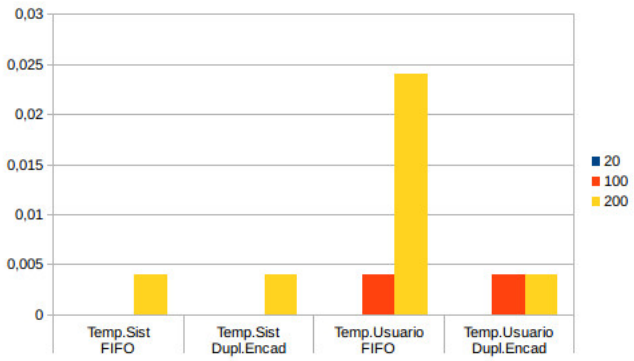


Figure 5: Tempos de sistema e usuário

O número de vezes que a porta abriu e fechou para as diferentes estratégias não foi o esperado para dados maiores. Na estratégia 1, a porta era aberta sempre que um passageiro entrava ou saia, independentemente se já havia sido aberta naquele andar. Diferentemente da estratégia da lista duplamente encadeada, em que a porta foi aberta apenas uma vez no andar, quando necessário.

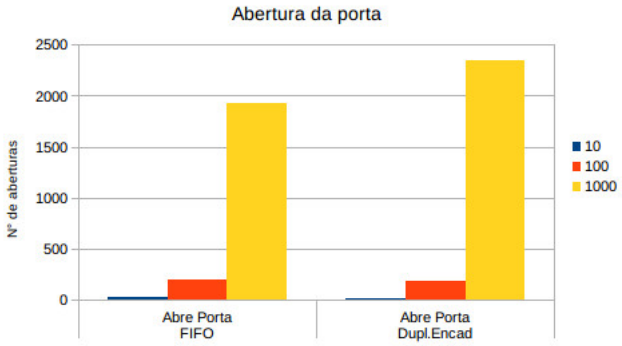


Figure 6: Abertura da porta

Isso trouxe resultados mais satisfatórios da estratégia 2, para casos menores. Porém, para a maior entrada, essa lógica não se aplicou, e o número de vezes que a porta se abriu e fechou foi praticamente o mesmo para as duas estratégias.

# 7 Análise de complexidade

## 7.1 Estratégia 1

A estratégia 1 é muito dependente da capacidade do elevador, já que ele é ocupado até sua capacidade total e depois desocupado até restar ninguém. Deste modo, a análise de complexidade será feita considerando não apenas o número de passageiros na entrada (n), mas também a capacidade do elevador (c).

Primeiramente há um laço de repetição que garante que toda a fila será percorrida, até o último elemento (n). A partir daí, o código se divide em duas partes: buscar os passageiros até a capacidade total do elevador, e depois deixá-los.

Na primeira parte, é feito um laço de repetição que verifica se a ocupação ainda é menor que a capacidade, e se o último elemento não é nulo. Então pega-se o primeiro passageiro. Outro laço aninhado terá a mesma condição e então as demais pessoas entrarão no elevador. Dentro deste, é necessário outro laço de repetição para adicionar o tempo de cada passageiro pego aos demais que já estão do elevador, portanto a condição de parada também é a capacidade máxima. A complexidade da primeira parte pode ser representada por:

n * c * c *c = O(n$(c)^3$)

A segunda parte irá se diferenciar apenas em um laço adicional, porém o mesmo não se encontra em último nível. As demais operações executadas são de custo 1, portanto a complexidade do algoritmo será O(n$(c)^3$), sendo n o número de passageiros e c a capacidade do elevador.

## 7.2 Estratégia 2

A estratégia 2 se divide em duas partes: subir até o último andar, e descer até o primeiro. Isso é feito até que a fila esteja vazia, por isso o primeiro laço de repetição irá percorrer todos os elementos da fila no pior caso.

Dentro desse laço, há mais dois aninhados: o laço responsável por percorrer toda a fila em busca de alguém que possa descer, que irá sempre até seu final, e outro que buscará quem está naquele andar e pode ser pego. Esse último laço terá seu pior caso raro, em que todas as pessoas estão no mesmo andar.

O restante das operações tem custo 1, e isso irá se repetir para o ato de descida. Portanto, a complexidade desse algoritmo é O($n^3$), sendo n o número de passageiros da entrada.

# 8 Conclusão

A implementação do simulador de elevador de duas maneiras diferentes possibilitou a visualização de uma diferença notável entre os tempos de espera observados.

Como é mostrado na Figura 3, o método FIFO é o menos eficiente, pois não privilegia as necessidades momentâneas do passageiro. O algoritmo tem um

objetivo a ser atingido, que é esvaziar o elevador,e assim desrespeita completamente qualquer possibilidade de um passageiro deixar o elevador antes disso acontecer.

Enquanto o segundo método, por usar uma fila, e permitir a retirada de elementos a qualquer momento, avalia se há um ou mais passageiro para descer, mesmo não tendo lotado sua capacidade máxima, e verifica se há uma ou mais pessoas a embarcarem em um mesmo andar, tornando-se assim mais eficiente.

Além disso, esses simuladores tratam também da observação de um mesmo evento sobre duas perspectivas diferentes. Sendo assim, um mesmo problema pode possuir inúmeras soluções, e não necessariamente a simplicidade é o melhor caminho, como foi provada pela primeira estratégia que faz uso do método FIFO.

# References


[1] UFMG DCC. *Estruturas de dados básicas*. 2016.

[2] C Progressivo. *Estruturas*. 2015.