UNIVERSIDADE FEDERAL DE MINAS GERAIS

Departamento de Ciência da Computação

Bacharelado em Sistemas de Informação

Jean George Alves Evangelista

Estruturas de Dados

Trabalho Prático 1: Acesso ao Ensino Superior em Arendelle

Belo Horizonte

Maio/2019

1 INTRODUÇÃO

O objetivo deste trabalho é realizar a implementação de um programa baseado no Sistema de Seleção Unificado (SISU) utilizando a linguagem de programação C/C++ e os conceitos vistos na disciplina Estruturas de Dados.

O programa primeiramente deve ler a quantidade de cursos e alunos. Em seguida, devem ser lidos os dados de cada curso (nome e quantidade de vagas ofertadas) e de cada candidato (nome, nota, opção de curso 1 e opção de curso 2). Como saída de ser mostrado a nota de corte, a lista de candidatos aprovados (em ordem decrescente de nota) e lista de candidatos na lista de espera (também em ordem decrescente de nota), para cada curso.

O que a versão final do programa faz é justamente o que foi descrito no parágrafo anterior. Após a leitura dos dados, os candidatos e cursos são armazenados em estruturas de dados do tipo lista encadeada (que também pode ser interpretada como uma fila, uma vez que os registros são inseridos somente no fim). Em seguida, a lista candidatos é ordenado por ordem decrescente de nota e os candidatos são alocados para seus respectivos cursos, sendo inseridos na lista de espera ou lista de aprovados do curso, conforme a regra estabelecida no enunciado do trabalho. Por fim, os dados são mostrados na tela.

2 IMPLEMENTAÇÃO

Optou-se por realizar a implementação na linguagem de programação C++, por conta da maior familiaridade com a mesma. O programa possui cinco classes: Candidato, Curso, No, Lista e MiniSisu. Tais classes são explicadas de forma detalhada abaixo.

Classes

Candidato: Tipo abstrato de dados que reúne em si os atributos de um candidato. Além dos atributos obrigatórios (*nome*, *nota* e as duas *opções de curso* escolhidas), possui um *id* para facilitar em uma possível ordenação, um atributo booleana denominado *aprovado* que diz se o candidato em questão já foi aprovado em algum curso e um atributo *sort* para realizar a ordenação.

Curso: Classe que reúne os atributos de um curso. Além de *nome* e *vagas disponíveis*, um curso possui um *id*, um *sort* para ordenação e duas listas de candidatos (*aprovados* e *de espera*).

No: Classe que representa cada nó da lista. Possui como atributos um *dado* e um ponteiro para o *próximo nó*, por se tratar de uma lista encadeada.

Lista: Possui como atributos dois *apontadores do tipo* nó, que demarcam a primeira e a última posição da lista. Possui dois métodos: para inserção e para ordenação. É importante salientar que numa classe que representa uma lista cabem inúmeros métodos, entretanto foram implementados somente os necessários para a realização do trabalho.

MiniSisu: Classe que representa o sistema em si. Armazena os atributos *lista de cursos* e *lista de candidatos*. Possui dois métodos para leitura dos dados, um método para

Uma observação interessante a ser feita é que tanto o *No* como a *Lista*, são classes genéricas, ou seja, foram implementadas utilizando *templates*. Implementar de tal forma se mostrou mais viável, uma vez que permite que sejam criadas listas

de qualquer tipo, desde os tipos primitivos como *int* ou *float* até tipos abstratos como um *Curso* ou *Candidato*.

Estrutura de Dados

A estrutura de dados utilizada foi uma lista encadeada, que guarda em si ponteiros para o primeiro elemento e para o último elemento. Abaixo segue uma figura ilustrativo deste tipo de lista.

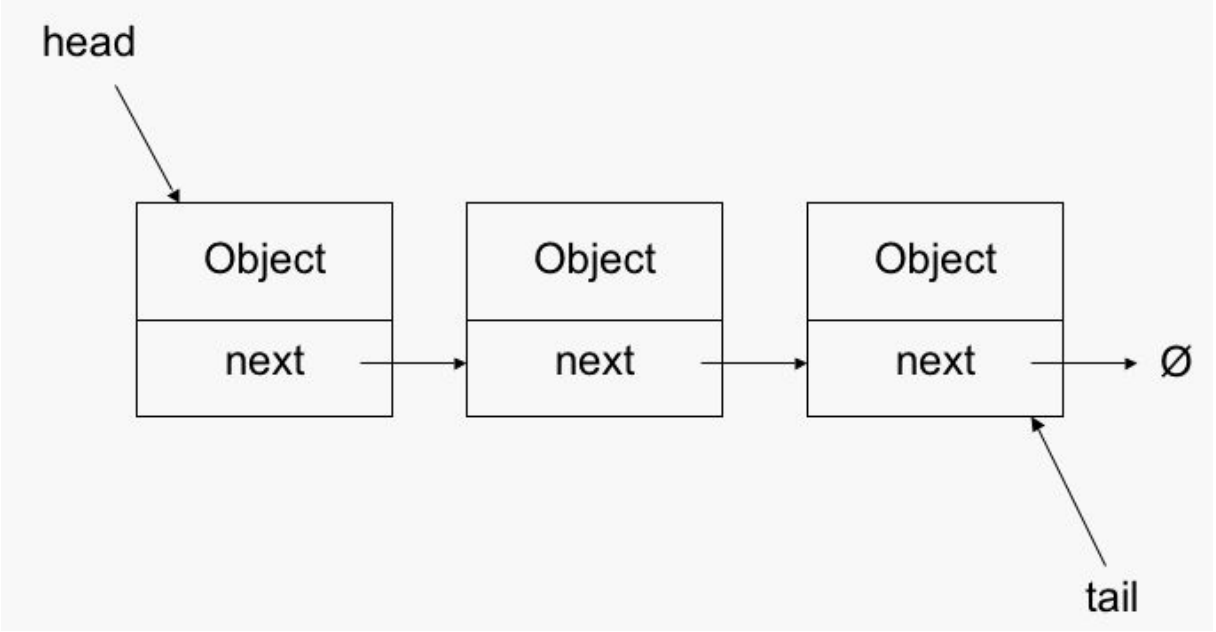

Figura 1. Representação de uma lista encadeada.

Head e *tail* são o primeiro e último nó da lista, respectivamente. *Object* é o dado que está sendo guardado em cada nó (cursos e candidatos para este trabalho). *Next* é um nó que referencia o próximo nó da lista.

Principais Métodos

Para a lista encadeada, foi necessário implementar apenas dois métodos: inserção e ordenação.

Inserção: Para facilitar, foi decidido que a inserção seria realizada somente no final da lista. Caso a lista esteja vazia o dado é inserido no primeiro nó e o último nó

é igualado ao primeiro. Já caso a lista não esteja vazia os dados são inseridos na última posição.

Ordenação: Para ordenar a lista de candidatos, optou-se pelo método da bolha por conta da sua fácil implementação. Este método consiste em comparar todos os elementos com todos, em laços *for* e realizar a troca quando necessário.

Também foram criados os métodos para *preenchimento* de cursos e de candidatos, que fazem as iterações necessárias e chamam a função *inserir* presente na classe *Lista*.

A função de *alocar candidatos* trata os dados relativos à primeira opção e à segunda opção de forma separada e sequencial. Desta forma, evita-se que um candidato seja alocado de maneira errada e que garante-se que as regras de desempate serão cumpridas. Primeiramente a função ordena a lista de candidatos. Em seguida a lista de cursos é percorrida e para cada curso percorre-se toda a lista de candidatos. Caso hajam vagas disponíveis, curso em questão é a primeira opção do candidato e o candidato ainda não foi aprovado em outro curso, este candidato é inserido na lista de aprovados do curso. Após percorrer todos os candidatos a lista de aprovados e a lista de espera são ordenadas, caso necessário. As iterações são repetidas: percorre-se todos os cursos e para cada curso percorre-se todos os candidatos, só que desta vez avalia-se segunda opção. A ordenação da lista de aprovados e da lista de espera é feita. Após a execução desta função tem-se as listas de aprovados e de espera de todos os cursos preenchidas corretamente.

Um método foi criado somente para *mostrar* o resultado final, no formato exigido no enunciado.

3 ANÁLISE DE COMPLEXIDADE

É possível realizar análise de complexidade de alguns métodos do programa:

ordenar(): A ordenação foi feita baseando-se no algoritmo *bubblesort*, que possui conhecida complexidade $O(n^2)$, um laço for dentro de outro laço for n vezes, sendo n o tamanho de elementos da lista.

alocarCandidatos(): Tal método percorre todos os cursos e para cada curso percorre todos os candidatos. Após cada término de iteração dos candidatos, é possível que o método *ordenar* seja chamado duas vezes. Todo esse processo é repetido duas vezes. Portanto a complexidade será: $2\ ((nm) + 2n^2))$, sendo n o número de cursos e m o número de candidatos. Nota-se que o termo que quem irá "mandar" na complexidade será nm se $m > n$ e n^2 caso contrário. Conclui-se que a complexidade no pior caso para a função será $O(max(m*n,n^2))$, isto é, será $O(m*n)$ ou $O(n^2)$.

mostrarResultado(): Este método percorre todos os cursos e para cada curso percorre sua lista de espera e lista de aprovados de candidatos. No melhor dos casos, onde todos os candidatos estão aprovados teremos como função de complexidade n, sendo n o número de candidatos. Já no pior dos casos teremos também candidatos na lista de espera. Não é difícil notar que a quantidade de candidatos na lista de espera nunca superará a quantidade de candidatos aprovados. Portanto a função de complexidade será O(n).

4 CONCLUSÃO

Foi possível realizar uma implementação funcional do programa descrito, que atenda aos requisitos pré-estabelecidos. Conclui-se que a implementação deste trabalho serviu como excelente forma de estudo e exercício dos conceitos vistos em sala de aula.

BIBLIOGRAFIA

[1] USFCA. **Linked Lists**. Disponível em: <https://www.cs.usfca.edu/~srollins/courses/cs112-f07/web/notes/linkedlists.html>. 03 mai. 2019.

[2] StackOverflow. Disponível em: <https://stackoverflow.com/>. 03 mai. 2019.

[3] C++ Language. Disponível em: <http://www.cplusplus.com/doc/tutorial/>. 03 mai. 2019.

[4] Geeks for Geeks. Disponível em: <https://www.geeksforgeeks.org/selection-sort/>. 03 mai. 2019.

[5] UFMG Virtual. Disponível em: <virtual.ufmg.br>. 03 mai. 2019.