Trabalho Prático 1: Acesso ao Ensino Superior em Arendelle

Guilherme de Abreu Lima Buitrago Miranda
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil
guilhermemiranda@dcc.ufmg.br

1. Introdução

O Trabalho Prático 1 da disciplina de Estrutura de Dados consiste num sistema para seleção de alunos para o Ensino Superior em *Arendelle*, semelhante ao SISU que existe no Brasil. Dados cursos e vagas, os alunos devem ser classificados de acordo com a nota alcançada e os cursos almejados.

Para isso, foi desenvolvido um código na linguagem C++ que, além de classificar os alunos nos respectivos cursos, permite a visualização da lista de espera e se atenta a critérios de desempate em caso de alunos com notas iguais, dando preferência ao estudante que colocou o curso almejado como primeira opção.

A solução computacional faz uso do paradigma de orientação a objetos e implementa como estrutura de dados a lista simplesmente encadeada.

2. Implementação

A solução para o dado problema foi pensada da seguinte forma: Após a leitura dos dados (detalhada posteriormente), os alunos são alocados nas listas dos seus respectivos cursos desejados, atentando-se cuidadosamente aos critérios de desempate.

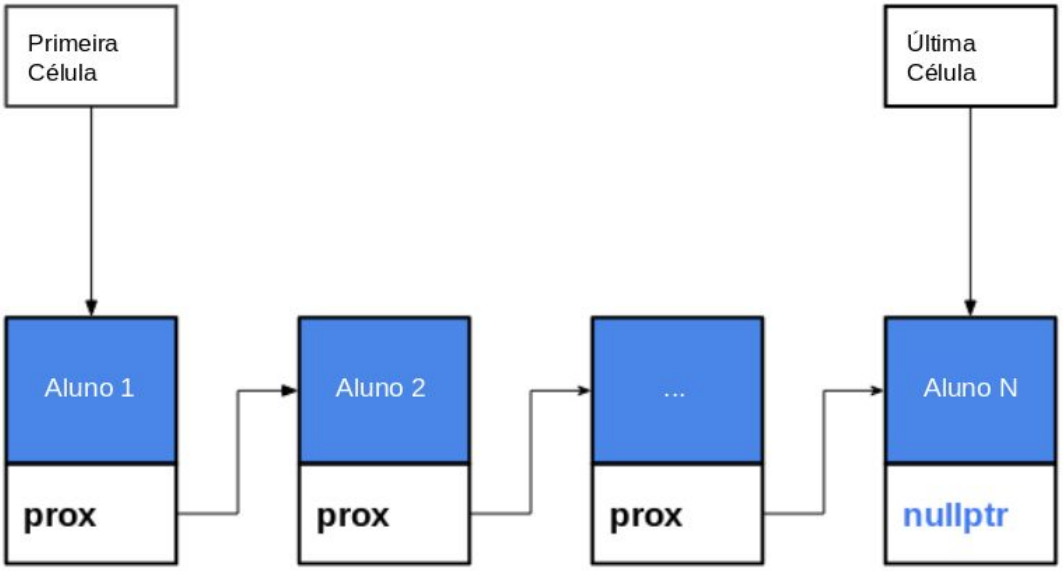
Em sequência, existe uma verificação para cada aluno, observando se o mesmo está selecionado em sua opção de curso número um. Se tal informação é verídica, o mesmo é apagado da lista de estudantes do curso desejado como opção número dois.

Por fim, é feita a impressão dos alunos, separando-os em “Classificados” e “Lista de espera” de acordo com seus cursos, assim como a nota de corte destes.

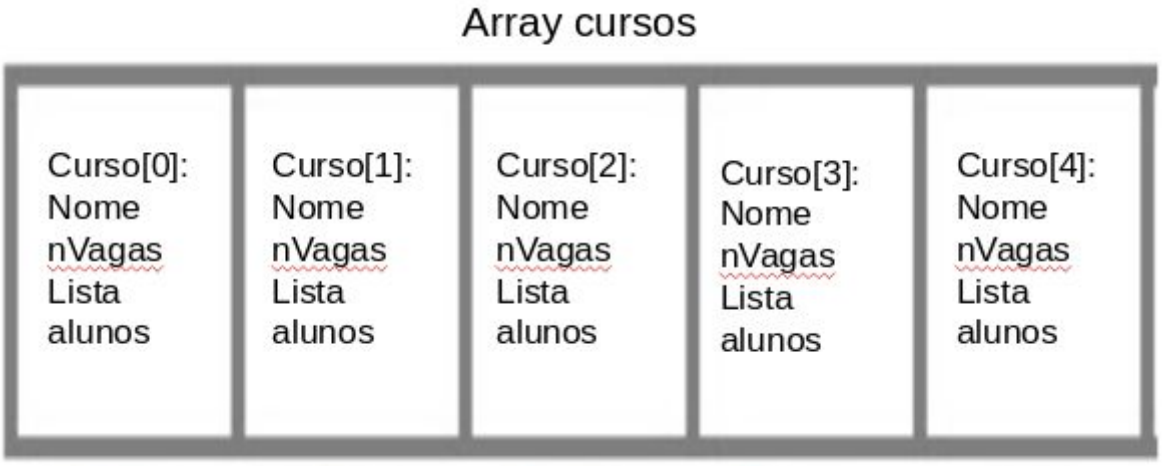
2.1. Estrutura de Dados

Para a resolução do trabalho foi implementada uma classe “lista.cpp” que trabalha com uma lista simplesmente encadeada de células. As células são uma struct que tem um atributo do tipo “aluno” e um apontador para a próxima célula.

Os alunos são uma struct com nome, nota, opção 1 e 2 e, por fim, dois atributos booleanos que dizem se o aluno foi inserido na opção 1 e na opção 2. Dessa forma, a lista com todos os alunos recebidos na entrada do programa ficará da seguinte forma:



Além disso, existe um struct curso que armazena o nome do curso, o número de vagas e uma lista de alunos. Assim, é criado um array de cursos, fazendo com que a estrutura de dados principal do programa se disponha da seguinte forma:



2.2. Procedimentos e Funções

Os principais procedimentos e funções utilizados para a resolução do problema são os declarados na classe Lista. Além do construtor, destrutor e de uma função que retorna se a lista está vazia, a classe conta com um procedimento que insere um novo aluno na lista de forma ordenada pelo critério de notas decrescentes. Ou seja, inserindo os alunos aplicando esse procedimento, tem-se uma lista em que o primeiro elemento tem a maior nota, o segundo elemento tem a segunda menor nota e, por fim, o último elemento tem a menor nota.

Ademais, a função busca retorna o índice de um aluno passado como atributo, podendo retornar o valor -1 (caso o aluno não esteja na lista) ou variar entre 0 (primeira posição da lista) e n-1 (última posição de uma lista de tamanho n).

O procedimento mais difícil e de maior complexidade a ser implementado foi o insereCursos, que passa por todos os alunos, inserindo os mesmos em cada uma das opções desejadas. A complicação encontra-se no momento em que dois ou mais alunos estão em situação de empate de notas. Na ocasião em que isso acontece, utiliza-se um outro while, passando pelos alunos até que o empate seja encerrado e fazendo a inserção conforme critério de desempate (se as notas forem as mesmas, o aluno que deseja aquele curso como primeira opção tem preferência frente a alunos que desejam o curso como segunda opção).

Posteriormente, existe o procedimento removeClassificados, que analisa se o aluno está entre os classificados na sua primeira opção de curso e, caso seja verdadeiro, retira o aluno da lista do curso de opção 2 com o procedimento “deleta”.

Por fim, existe o procedimento imprimeNomes que imprime os classificados e a lista de espera conforme a saída de dados esperada (detalhada posteriormente). Além disso, esse método chama a função buscaCorte, que retorna o ponto de corte (nota do último aluno classificado) do curso em questão.

2.3. Entrada e Saída de Dados

A entrada de dados, segundo o enunciado fornecido, deve seguir a seguinte estrutura:

Entrada
(qtd. de cursos) (qtd. de alunos) (nome do curso 0) (qtd. de vagas do curso 0) (nome do curso 1) (qtd. de vagas do curso 1) ... (nome do curso $N-1$) (qtd. de vagas do curso $N-1$) (nome do aluno 0) (nota do aluno 0) (1ª opção) (2ª opção) (nome do aluno 1) (nota do aluno 1) (1ª opção) (2ª opção) ... (nome do aluno $M-1$) (nota do aluno $M-1$) (1ª opção) (2ª opção)

Entrada
2 5 Mineracao de Gelo 2 Engenharia Metalurgica 1 Olavo das Neves 496.00 0 1 Gothi Gelatto 490.10 0 1 Gerda Ferreira 556.79 1 0 Hans Westergaard 418.09 1 0 Kristoff Bjorgman 725.66 0 1

Dessa forma, a primeira parte da entrada (até a quantidade de alunos) são lidos de maneira linear, enquanto a leitura dos cursos e suas vagas, assim como dos alunos, suas notas e opções, são feitas dentro de uma estrutura de repetição.

Após todo o processamento previamente explicado, o método imprimeNomes gera a saída, que segue o seguinte modelo:

Saída
(nome do curso 0) (nota de corte) Classificados (nome do primeiro aluno) (nota do aluno) (nome do segundo aluno) (nota do aluno) ... (nome do último aluno) (nota do aluno) Lista de espera (nome do primeiro aluno) (nota do aluno) (nome do segundo aluno) (nota do aluno) ... (nome do último aluno) (nota do aluno) ...

Saída
Mineracao de Gelo 496.00 Classificados Kristoff Bjorgman 725.66 Olavo das Neves 496.00 Lista de espera Gothi Gelatto 490.10 Hans Westergaard 418.09 Engenharia Metalurgica 556.79 Classificados Gerda Ferreira 556.79 Lista de espera Gothi Gelatto 490.10 Hans Westergaard 418.09

2.4. Compilador e Ambiente

Para a compilação e execução de testes foi utilizada uma máquina com a versão 4.19.36 do Kernel Linux e o g++/gcc versão 8.3.0 instalados. Não obstante, a compilação foi feita com o comando make no terminal (utilizando o arquivo makefile disponibilizado junto do enunciado e também em anexo) e os testes de entrada e saída foram feitos redirecionando os arquivos .in e .out pelo terminal.

3. Complexidade

3.1. Complexidade de Tempo

Atentando à complexidade de tempo da solução desenvolvida, devemos analisar o pior caso possível, ou seja, o caso em que é feito o maior número de verificações e/ou iterações. Em relação aos cursos, o melhor, pior e o caso médio são os mesmos, já que todo processamento é linear, portanto, m.

Já em relação aos alunos, em caso de empate, a complexidade é aumentada. Dessa forma, o pior caso possível é aquele em que todos os n alunos têm notas iguais. Quando isso acontece, apesar das n operações para leitura, no momento de inserção na lista (tanto na lista com todos os alunos, tanto nas listas individuais de cada curso), o aluno será comparado até o último, o que nos dá uma progressão aritmética que se comporta da seguinte maneira: 1 + 2 + ... + (n-2) + (n-1) = [(n-1)n]/2 operações.

Assim, temos a complexidade de tempo do algoritmo: $f(n, m) = O(n^2 + m)$.

3.2. Complexidade de Espaço

Com relação à complexidade de espaço, é fundamental notarmos que o algoritmo trabalha com duas grandes estruturas de dados: uma lista com todos os alunos (independente dos cursos desejados) de tamanho n e um array de m cursos. Na segunda, para cada curso, existe também uma lista de alunos, contudo, um aluno estará em no mínimo uma e no máximo duas listas no array cursos.

Por conseguinte, analisando o pior caso possível (cada aluno está em duas listas ao mesmo tempo), a complexidade de espaço do algoritmo será dada por uma lista com n alunos, um array de m cursos e uma listas internas ao array com tamanho 2n. Assim, $f(n, m) = 3n + m = O(n + m)$.

4. Conclusão

O desenvolvimento do algoritmo proposto foi de fundamental importância para consolidar os conhecimentos adquiridos durante as aulas da disciplina de Estrutura de Dados, sobretudo a respeito de listas encadeadas.

A maior dificuldade encontrada, além da manipulação de ponteiros e erros de segmentação, foi buscar uma maneira para não ter que verificar a todo momento os cursos e remover os selecionados na primeira opção de sua segunda. A priori, imagina-se uma solução recursiva (e nada eficiente) para o problema, mas, após bastante reflexão a respeito do problema, ordenar os alunos de forma decrescente de nota mostrou-se a opção mais fácil e eficiente, eliminando o problema e tratando apenas casos de empate.

Isto posto, o trabalho mostrou-se bastante proveitoso para treinar a lógica de programação e melhorar os conhecimentos profundos a respeito de uma importante estrutura de dados.

5. Referências

- A Tour of C++. Bjarne Stroustrup. Addison-Wesley Professional, 2013. 1st Edition.
- MS LATHA SHILVANTH. Java: For Programming (2018). Createspace independent Publishing Platform. ISBN-10: 1985254603
- Cormen , T., Leiserson, C, Rivest R., Stein, C. Introduction to Algorithms, Third Edition, MIT Press, 2009. Versão Traduzida:Algoritmos – Teoria e Prática 3a. Edição, Elsevier, 2012.