Trabalho Prático 1: Acesso ao Ensino Superior em Arendelle

Victor Hugo Silva Moura

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

victorhugosmoura@gmail.com

Abstract. *The following text describes the "Acesso ao Ensino Superior de Arendelle"'s problem and shows one of its possible solutions, using linked lists. Besides the solution, the complexity and its analysis are showed, as well as the implementation difficulties.*

Resumo. *O texto a seguir descreve o problema "Acesso ao Ensino Superior de Arendelle" e mostra uma de suas possíveis soluções, utilizando listas simplesmente encadeadas. Além da solução, é apresentada sua complexidade e a análise da mesma, bem como as dificuldades envolvidas na implementação.*

1. Introdução

O problema *Acesso ao Ensino Superior de Arendelle* consiste em classificar alunos para certos cursos de acordo com suas notas e opções de curso. Dado um conjunto de alunos com suas respectivas notas e opções de curso, e um conjunto de cursos, cada um com seu número de vagas, deve-se distribuir os alunos entre seus cursos de escolha, seguindo uma série de ordens específicas que serão citadas posteriormente. Ao final da execução do programa, os alunos precisam ter sua colocação garantida na primeira chamada ou na lista de espera de algum curso. A lista de classificados, lista de espera e a nota de corte, de cada curso, deverão ser impressos na tela.

Para classificar os alunos nos cursos, temos uma série de regras. Os alunos deverão estar em ordem decrescente de nota (para cada curso). Um aluno classificado em sua primeira opção de curso não deve ser colocado em nenhuma lista de espera, já um aluno classificado para sua segunda opção deverá ser colocado também na lista de espera da primeira opção. Caso o aluno não se classifique para nenhuma de suas opções, ele deve ser colocado na lista de espera de ambas. Se houver empate de nota entre alunos, primeiramente considera-se a ordem de opção (o aluno que escolheu o curso como primeira opção tem preferência), e após isso considera-se a ordem de chegada, ou seja, o aluno que se cadastrou primeiro no sistema fica com a vaga.

Para a resolução desse problema foi utilizada a estrutura de dados abstrata (TAD) lista encadeada. O programa utiliza diversas listas para organizar os alunos de acordo com os critérios necessários. A linguagem utilizada para este propósito foi a linguagem C.

2. Implementação

A organização das estruturas desse programa pode ser divida em duas partes: uma lista geral (que guarda todos os estudantes) e um array de cursos (cada um com seu nome, número de vagas e uma lista de estudantes). Essas são as duas estruturas principais, responsáveis por guardar os dados e distribuir os dados de forma correta. Sendo assim, vamos dividir a implementação em três partes principais: lista geral, array de cursos, solução final.

2.1. Lista Geral

No começo do programa todos os alunos são guardados na lista geral, sendo ordenados, em ordem decrescente, por nota. O método de ordenação utilizado é uma variação do Insertion Sort. À medida que os dados dos alunos são digitados, eles entram na lista de forma ordenada, porém, em caso de empate de notas, é mantida a ordem de chegada (importante para garantir um dos critérios de desempate). O modelo de lista (Figura 1) utilizado na lista geral é o mesmo utilizado para armazenar os alunos no array de cursos.

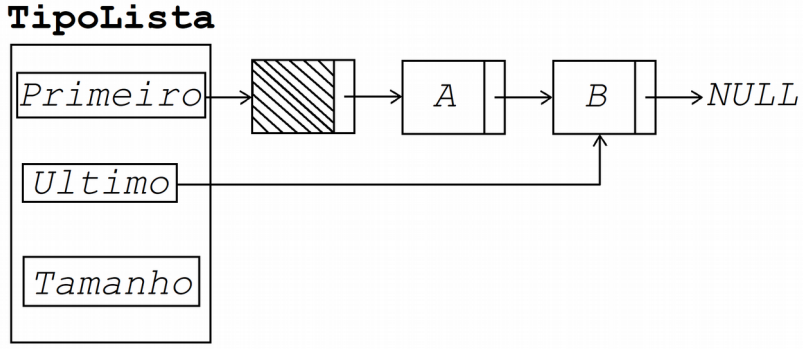

Figura 1. Modelo de lista encadeada utilizado no programa

Cada elemento da lista guarda dentro de si um ponteiro para o próximo elemento e um struct TipoItem (Figura 2) que representa um aluno. Esse struct é composto de: nome do aluno (string), nota do aluno (float), 1ª e 2ª opções de curso (int). Dessa forma, cada posição da lista geral guarda todas as informações disponíveis de aluno, facilitando o processo de distribuição deles posteriormente.

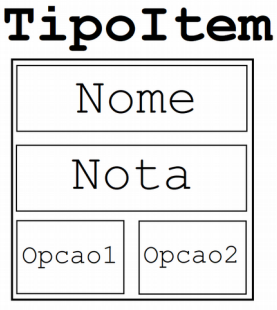

Figura 2. Modelo do struct TipoItem (fazer modelo do struct)

Falando mais detalhadamente sobre a ordenação, a função BuscaMenor é a principal função responsável por ordenar os itens de modo a garantir que a ordem de chegada seja mantida em caso de empate de notas. Essa função recebe um elemento TipoItem e a lista geral. A partir disso ela começa a percorrer a lista procurando o primeiro elemento com o atributo nota menor que a nota do item passado para ela. Caso encontre, a função

retorna um apontador para a posição desse item. Caso contrário, ela retorna um apontador para o final da lista.

2.2. Array de Cursos

Após ler todos os dados de alunos, é necessário distribuí-los entre os cursos que eles escolheram. Para isso foi criado o array de cursos. Cada posição desse array contém uma struct do tipo Curso (Figura 3) que guarda o nome do curso (string), o número de vagas desse curso (int), a nota de corte (float) e uma lista de alunos (TipoLista). Como foi dito na seção 2.1, a lista de alunos utiliza o mesmo modelo da lista geral.

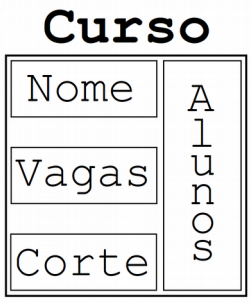

Figura 3. Modelo do struct Curso

A escolha de um array para armazenar os cursos se deve a três fatores:

1) acesso direto a qualquer curso por meio de um índice (Figura 4)
2) os cursos são numerados de 0 a n-1, assim como os arrays em C
3) o número de cursos é fixo e definido pelo usuário no começo do programa

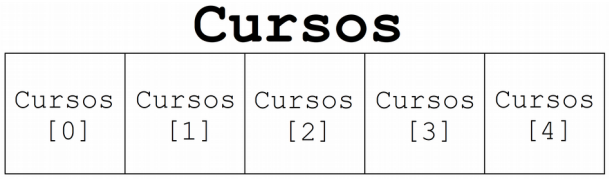

Figura 4. Representação do array de cursos e do acesso à cada item

O nome e o número de vagas de cada curso são guardados em suas respectivas posições do array imediamente após a leitura dos mesmos, a nota de corte é definida inicialmente como 0 e a lista de alunos é iniciada como uma lista vazia.

2.3. Solução Geral

Após detalhar as estruturas de dados utilizadas no programa, é possível explicar o processo de aquisição dos dados na entrada do programa, a organização e distribuição de alunos pelos cursos e a saída final, elementos que, juntos, compõe a solução final. Vamos tratar inicialmente da entrada.

A entrada de dados segue o padrão mostrado abaixo (veja Tabela 1), sendo *M* o número de alunos no total e *N* o número de cursos.

Tabela 1. Formatação dos dados de entrada do programa

Entrada	**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)	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
Saída	**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) ...	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

Após ler o número de cursos e a quantidade de alunos, o array de cursos é alocado dinamicamente, de acordo com o número de cursos e, assim, é possível percorrê-lo e fazer a leitura e distribuição correta de cada disciplina para sua posição correta no array (como citado na seção 2.2). Em seguida, é feita a leitura dos dados de alunos e sua ordenação no array geral (citado na seção 2.1).

Com isso pronto, começa o processo de distribuição dos alunos por seus cursos. Esse processo é feito na função Distribui que aproveita a ordenação dos alunos, feita anteriormente, para diminuir o número de verificações necessárias. Essa função percorre aluno por aluno do array ordenado, colocando ele em sua primeira opção de escolha. Para essa inserção, há três possibilidades:

1) Se ele é o primeiro a entrar no curso, e o número de vagas é > 0, então ele é inserido diretamente.

2) Caso o último aluno do curso, no momento, tenha uma nota maior que a do atual aluno, a inserção ocorre no fim e o atual aluno se torna o último.

3) Caso haja empate de notas, a lista inteira é verificada até que seja encontrado a primeira posição onde o atual aluno tem prioridade sobre outro com mesma nota (prioridade de opção de curso). O aluno é então inserido à frente do aluno que não tem essa prioridade. Caso não haja um caso desses na lista, o aluno é inserido no fim.

Após a inserção do aluno, é verificada sua posição na lista de seu primeiro curso. Se essa posição exceder o número de vagas do curso, ele também é inserido em sua segunda opção de curso, desta vez, diretamente no fim. Após cada uma dessas inserções, é conferido se o número de alunos do curso é igual ao número de vagas a fim de atribuir uma nota de corte ao curso. Essa nota de corte é a nota do aluno que está em último lugar no momento.

Todas essas operações são possíveis (e atendem aos requisitos de desempate) devido à ordenação feita inicialmente. Como ela leva em consideração a ordem de chegada, dois alunos com mesma nota e opções de curso entrarão nos cursos na mesma ordem determinada pela entrada do programa. Além disso, como as notas estão em ordem decrescente, o aluno que garantiu uma posição em sua primeira opção terá aquela posição fixa, ou seja, não há como outro aluno passar à sua frente, seja por nota ou pelos critérios de desempate (opção e ordem de chegada). Assim, a inserção desse aluno em sua segunda opção é facilitada.

Para a saída de dados, são utilizadas duas funções: ImprimeClassificacao e Imprime. A primeira é responsável por guiar a forma de impressão da segunda e gerar a saída no formato correto. Essa função percorre o array de cursos e, inicialmente, imprime o nome de curso seguido da nota de corte do mesmo. Após isso, vem a lista de classificados por meio de uma chamada para a função Imprime. Esta segunda função recebe a lista de alunos e o número de vagas do curso. Com isso, ela itera pela lista imprimindo, em sequência, o nome dos alunos que estão na lista. Um contador, iniciado em 0, é acrescido de 1 a cada aluno impresso e, ao chegar no número de vagas, aciona uma condição que então imprime “Lista de espera”, e dessa forma temos a lista de classificados e a lista de espera em uma só. Caso o curso tenha menos alunos que o número de vagas, a frase “Lista de espera” é impressa ao final da função.

O compilador utilizado durante toda a implementação foi o GNU C Compiler (GCC) versão 7.3.0 e o sistema operacional foi um Linux Ubuntu 18.04 64-bit. Todos os testes foram feitos nesse mesmo sistema e a entrada de dados foi feita de duas formas: digitada e redirecionada de um arquivo, ambas via terminal.

3. Complexidade

3.1. Complexidade de Espaço

Para analisar essa complexidade, o melhor parâmetro é o número de alocações. Existem 3 alocações principais que são feitas: a lista geral (que guarda os dados de todos os alunos), o array de cursos e a lista dentro de cada curso. Para facilitar essa análise, ficaremos no pior caso apenas.

No pior caso desse programa, todos os alunos ficam na lista de espera da primeira opção e da segunda também. Vamos considerar n como o número de alunos e m como o número de cursos. Sendo assim, o número de alocações para as listas dos cursos é $2n$ pois cada aluno aparecerá exatamente em duas listas diferentes. Para a lista geral, temos n alocações, uma alocação para cada aluno, e no array de cursos temos m alocações, uma alocação para cada curso. Juntando tudo, temos:

$$f(n, m) = 2n + n + m = 3n + m$$

$$f(n, m) = \mathbf{O(n + m)}$$

3.2 Complexidade de Tempo

Para analisar essa complexidade, uma série de operações foi analisada. Cada operação tinha uma relevância em determinada parte do código. Podemos então dividi-las em:

- Leitura/armazenamento (leitura inicial de dados)
- Comparações entre alunos (ordenação e distribuição dos alunos)
- Escrita/impressão (impressão final de dados)

Vamos considerar n como o número de alunos e m como o número de cursos. Para operações de leitura/armazenamento, ao começo do código temos a leitura de m cursos, com seus respectivos números de vagas, seguida da leitura de n alunos e suas notas. Assim temos $2m + 2n$ operações nesse passo.

Na parte de comparações entre alunos, temos inicialmente a ordenação inicial. O pior caso dessa ordenação ocorre quando todos os alunos têm a mesma nota ou quando a ordem de notas da entrada é decrescente. Para esses casos, o aluno que está entrando na lista deve ser comparado com todos os outros que já estão na lista. Fazendo o somatório de todas as operações necessárias, teremos $[(n-1)n]/2$ operações.

Para a distribuição, temos como pior caso a inserção de todos os alunos com mesma nota e mesmo curso principal, exceto pelo primeiro (o segundo curso deste deve ser igual ao primeiro dos outros). Nesse caso, a condição de empate é realizada para todos os $n-1$ alunos com mesma opção principal de curso. Para cada verificação de empate, deve-se percorrer toda a lista, comparando o elemento a ser inserido com todos os outros. Assim, temos: $1 + 2 + \ldots + (n-2) + (n-1) = [(n-1)n]/2$ operações (igual ao caso de ordenação).

Por fim, para a parte de escrita, temos que imprimir todos os m cursos e os alunos nas listas de cada curso. O custo de impressão de cada curso é $3m$ (nome do curso e nota de corte impressos na mesma linha, "Classificação" e "Lista de espera"). Para os alunos, no pior caso, é $2n$, pois cada alunos só pode aparecer em no máximo duas listas diferentes. Assim, temos que a complexidade desse caso é $2n + 3m$.

Juntando tudo, temos que a complexidade total é:

$$f(n, m) = 2m + 2n + (n^2 - n)/2 + (n^2 - n)/2 + 2n + 3m = n^2 + 3n + 5m$$

$$f(n, m) = \mathbf{O(n^2 + m)}$$

4. Conclusão

Com o auxílio de todas as estruturas de dados citadas anteriormente e da lógica envolvida na manipulação delas, a realização deste trabalho foi possível. Eventuais dificuldades sugiram durante o processo de implementação, sendo algumas delas:

- Problemas com a manipulação de ponteiros;
- Dificuldade em conseguir uma lógica que atendesse os requisitos de lista de espera, sem ter que revisitar todas as matérias para conferir isso;
- Problemas de alocar as estruturas em uma função separada e utilizá-las em outras partes do código.

A primeira e última dificuldades foram resolvidas utilizando os diagramas das estruturas (apresentados na seção 2) e outros necessários. A segunda dificuldade foi superada após diversas tentativas de lógicas diferentes e por meio de uma análise detalhada da entrada do programa.

O desenvolvimento do programa girou em torno da lista encadeada mostrada pelo professor em sala de aula. Sendo assim, a estrutura de lista utilizada no programa é uma leve modificação da estrutura vista em sala de aula. Foram feitas apenas algumas pequenas alterações para comportar o novo tipo de item que cada célula deveria guardar e para imprimir. Além disso, novas funções (quase todas envolvem busca de itens na lista) foram criadas para suprir algumas necessidades do programa.

5. Referências

ZIVIANI, Nivio. Projeto de Algoritmos: com implementações em PASCAL e C. 2 ed. São Paulo: Thomson, 2004

Insertion Sort. Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Insertion_sort>. Acesso em: 26 abr. 2019