Trabalho - Estrutura de Dados

Gabriel Victor Carvalho Rocha

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

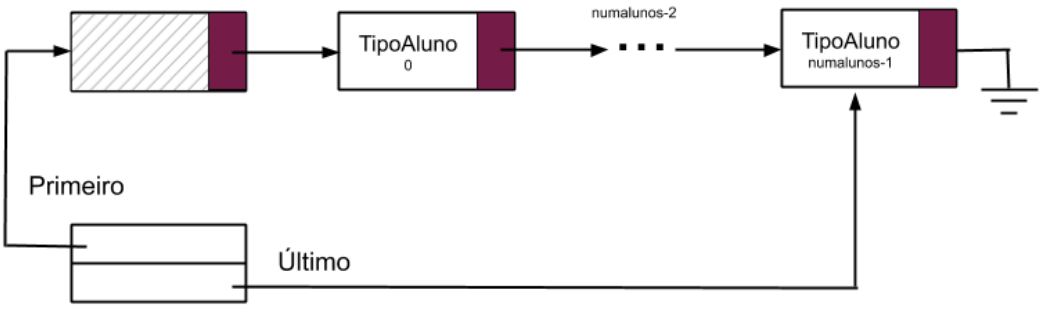
1. Introdução

O intuito do trabalho é realizar a classificação dos alunos de Arendelle a partir de suas notas e das opções de cursos disponíveis. O programa é feito em C e lê um arquivo .in por meio do stdin (./prog <file.in ...), armazena o número de cursos e o número de alunos em variáveis, depois armazena e insere todos os cursos e suas vagas em uma lista encadeada e o mesmo ocorre para os nomes, opções e nota do aluno. Após isso, é feito a ordenação das notas em ordem crescente em uma nova lista, e por fim, é feita a classificação dos alunos.

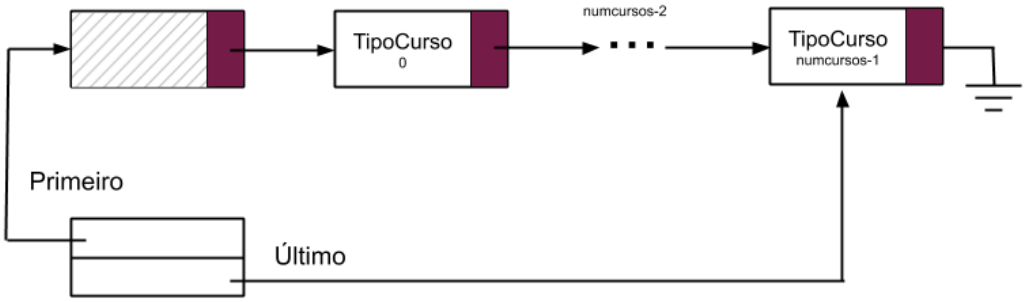
2. Implementação

Foi utilizada a estrutura de dados do tipo lista encadeada (que possui TipoAluno, TipoCurso e TipoClassificacao em sua célula) para os alunos, cursos e a classificação final. Para a implementação, o compilador usado foi o GCC.

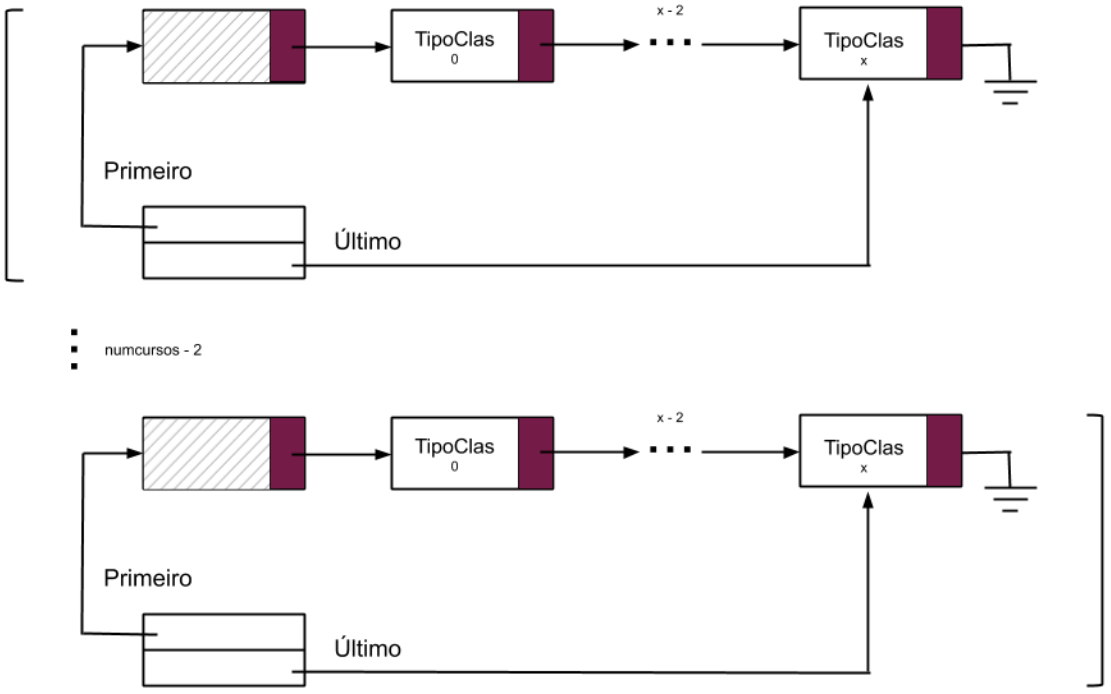
Para os alunos, a lista encadeada é a representada abaixo:



Para os cursos:



E para a classificação, foi criado um vetor de lista encadeada com tamanho n-cursos:



Funcionamento das principais funções:

1. void faz_lista_vazia(TipoLista *Lista);

   Cria uma lista vazia, alocando uma nova célula que será apontada pelo apontador Primeiro e Ultimo.

2. int verifica_lista_vazia(TipoLista *Lista)

   Verifica se a lista está vazia, retornando 1 quando o apontador Primeiro aponta para o mesmo lugar que o apontador Ultimo. Caso contrário, retorna 0.

3. void insere_curso(TipoLista *Lista, NomeCurso *nome, NumVagas vagas);

   Cria uma nova célula na lista que é apontada pelo apontador Ultimo, é colocado o nome do curso no ItemCurso.Curso e é colocado o número de vagas em ItemCurso.Vagas e apontamos o Prox dessa célula para NULL.

4. void insere_aluno(TipoLista *Lista, NomeAluno *nome, NotaAluno nota, Opcao1 op1, Opcao2 op2);

   Cria uma nova célula na lista que é apontada pelo apontador Ultimo, é colocado o nome do aluno no ItemAluno.Nome, sua nota em ItemAluno.Nota, e suas opções 1 e 2 em ItemAluno.Op1 e ItemAluno.Op2 e apontamos o Prox dessa célula para NULL.

5. void insere_classificado(TipoLista *Lista, NomeAluno *nome, NotaAluno nota);

   Cria uma nova célula na lista que é apontada pelo apontador Ultimo, é colocada as informações passadas para essa nova célula e por fim o apontador Prox da mesma aponta para NULL.

6. void desempate_classificado(TipoLista *Lista, Apontador Anterior, Apontador Alu);

   Essa função atua quando a nota do aluno inserido anteriormente é a mesma que a nota do que está inserindo atualmente e se o anterior tiver sido colocado na classificação por meio da opção 2 e for a opção 1 do atual, o atual ganha prioridade, sendo inserido na frente do anterior.

7. void retira_aluno(TipoLista*Lista, Apontador Anterior);

   Recebe por parâmetro o Apontador do aluno **anterior** do que se deseja retirar, cria um Apontador Elemento que **aponta pro**

aluno que será retirado e seu **anterior** recebe o Prox do Elemento. Caso esse Prox seja NULL, o Apontador Ultimo da lista aponta para o Apontador Anterior e é dado free no Elemento.

8. void insere_decrescente(TipoLista *Lista, TipoLista *Nova);

Recebe por parâmetro a lista com os dados (Lista) e uma lista que receberá em ordem decrescente (Nova). É criado um Apontador E que apontado para o primeiro aluno e um Apontador Anterior que recebe a célula cabeça. É executado um 2 whiles aninhados, onde o mais interno irá percorrer por todos alunos procurando a maior nota e após sair, insere-se o aluno com maior nota na lista Nova, esse mesmo aluno é retirado da Lista, então voltamos o E e o Anterior ao início para percorrer novamente a Lista até achar a maior nota novamente. Isto ocorre até a Lista se tornar vazia.

9. void faz_classificados(TipoLista *Alunos, TipoLista *Cursos, int numcursos);

Recebe por parâmetro a lista de Alunos (em ordem decrescente), a lista de Cursos e o número de cursos. Para controlar as vagas, cortes e a quantidade de alunos na lista de espera, é criado um vetor de tamanho numcursos para cada um deles. Também temos um vetor de TipoLista dos Classificados de tamanho numcursos para inserir a classificação de cada curso. E para os casos que possam haver empates, temos Apontadores para os Alunos anteriores (AluAnterior) e um vetor de tamanho numcursos do tipo Apontador que irá apontar para o último aluno inserido na Classificação de cada curso(ClasAnterior). Fazemos um for inicial que preenche esses vetores corretamente: vagas[i] recebe a quantidade de vagas do curso correspondente e como será necessário passar por referência depois e vagas[] precisa ser alterado para as condições a frente, é criado uma cópia vagascopia[]. Os vetores cortes[i] e alunos_espera[i] são preenchidos com 0. É realizado então um while para percorrer todos esses alunos e verificar suas opções e suas vagas:

- **1º Caso** (entrar no if): O aluno é inserido em Classificados[Op1 do aluno], e vagas[Op1 do aluno] receberá vagas[Op1 do aluno]-- e caso a vaga for 0 após o decremento, a nota do aluno inserido é o corte. Por fim, quando ocorrer a primeira classificação, iniciamos o apontador

ClasAnterior[Op1 do aluno] e nas vezes posteriores ele recebe o Prox.

- **2º Caso** (entrar no else if): Se o aluno não entrou na Op1, e se vagas[Op2 do aluno] possuir vagas, o aluno é inserido na Classificados[Op2 do aluno], e vagas[Op2 do aluno] receberá vagas[Op2 do aluno]--, mas ele também entra na lista de espera da Op1, então é inserido na Classificação[Op1 do aluno] (**classificados e lista de espera será controlado na hora de imprimir**) e alunos_espera[Op1 do aluno]++. Caso vaga[Op2 do aluno] for 0 após o decremento, a nota do aluno inserido é o corte. E cada vez que isso ocorre, o ClasAnterior[Op1 do aluno] e ClasAnterior[Op2 do aluno] recebem seus Prox.

- **3º Caso** (entrar no else): Se o aluno não entrou em nenhuma opção por falta de vagas, temos dois casos:

  - Caso o aluno possuir a mesma nota do aluno anterior e além disso o aluno anterior tiver sido classificado pela opção 2 e essa ser a opção 1 do atual, é usado a função desempate_classificado() onde o atual é inserido na frente do anterior.
  - Senão, o aluno atual é inserido normalmente em Classificados[Op1 do aluno] e o ClasAnterior recebe seu Prox.

  E depois dos casos, o aluno é inserido em Classificados[Op2 do aluno] e é dado alunos_espera++ para os 2 cursos. Por fim, imprimimos com a função que será explicada a seguir.

10. void imprime_classificacao(TipoLista *Cursos, TipoLista *Classificados, int numcursos, int *vagas, float *cortes, int *alunos_espera);

A função recebe a lista de Cursos, a lista com os Classificados, o número de cursos, um vetor com as vagas de cada curso, um vetor com os cortes de cada curso e um vetor com a quantidade de alunos na lista de espera de cada curso.

Temos o primeiro for que percorrerá cada Curso, imprimindo seu nome e o corte correspondente. No segundo for é imprimido os alunos classificados e suas notas, onde é imprimido enquanto as vagas forem maior q o iterador, mas também existe um **and** caso não haja mais alunos (no caso de sobra de vagas). Já no terceiro for, é imprimido os nomes e as notas dos alunos que estão na lista de espera do curso enquanto o alunos_espera for maior q o iterador.

3. Análise de Complexidade

Análise da complexidade de tempo e memória das principais funções, sendo m o número de cursos e n o número de alunos:

void faz_lista_vazia(TipoLista *Lista):
Tempo = a função é constante, O(1);
Espaço = a função é constante, O(1).

void insere_curso(TipoLista *Lista, NomeCurso *nome, NumVagas vagas):
Tempo = a função é constante, O(1);
Espaço = a função é constante, O(1).

void insere_aluno(TipoLista *Lista, NomeAluno *nome, NotaAluno nota, Opcao1 op1, Opcao2 op2):
Tempo = a função é constante, O(1);
Espaço = a função é constante, O(1).

void insere_classificado(TipoLista *Lista, NomeAluno *nome, NotaAluno nota):
Tempo = a função é constante, O(1);
Espaço = a função é constante, O(1).

void desempate_classificado(TipoLista *Lista, Apontador Anterior, Apontador Alu):
Tempo = a função é constante, O(1);
Espaço = a função é constante, O(1);

void retira_aluno(TipoLista *Lista, Apontador Anterior)
Tempo = a função é constante, O(1);
Espaço = a função é constante, O(1).

void insere_decrescente(TipoLista *Lista, TipoLista *Nova)
Tempo = O conjunto de whiles realizam m comparações, após a retirada do primeiro aluno, realizam m-1 comparações e assim por diante, sendo o somatório de m até 1 e temos: $m(m+1)/2 = m^2 + m / 2$ que possui complexidade $O(m^2)$;
Espaço = Inserimos m alunos com a função insere_aluno(), que possui custo constante e retiramos com retira_aluno() também constante, tendo então 2m, e sua complexidade é O(m).

void imprime_classificacao(TipoLista *Cursos, TipoLista *Classificados, int numcursos, int *vagas, float *cortes, int *alunos_espera)
Tempo = Imprimimos 3 vezes a quantidade de cursos, sendo primeiro o nome e a nota de corte do curso, depois “Classificados” e por fim “Lista de espera”, resultando em 3m. Como cada aluno só pode estar no máximo em 2 cursos diferentes, temos que o custo é 2n, totalizando no final 3m + 2n, possuindo complexidade O(m + n);
Espaço = Em questão de espaço, a função é constante, O(1).

void faz_classificados(TipoLista *Alunos, TipoLista *Cursos, int numcursos)
Tempo = Temos de início a criação de n listas vazias, resultando em custo n, depois de iniciar as listas, há mais um for para colocar os valores dos

vetores devidamente com custo n. Após isso, o loop while seguinte rodará m vezes (irá percorrer todos alunos verificando suas opções). Posteriormente, é chamada a função imprime_classificados() que possui custo m + n e por fim temos um for para executar o comando free n vezes no vetor de Apontador criado anteriormente, totalizando 2m + 4n, possuindo complexidade O(m + n);

Espaço = Em questão de espaço, alocamos memória no primeiro for n vezes na criação das listas vazias, tendo custo n. No loop while, no pior caso iremos alocar 2m (casos else if e else) e por fim, temos custo constante na função de imprimir, resultando em O(m + n).

int main()

Tempo = É criada 3 listas vazias de custo constante no início, depois temos um for de custo 4n (2 fgets, 1 scanf e o uso da função insere_curso()), após isso temos outro for de custo 6m (2 fgets, 3 scanfs e o uso da função insere_curso()). Além disso há o custo m^2 da função insere_decrescente() e por fim, custo m + n da função faz_classificados() totalizando $4n + 6m + m^2 + m + n = m^2 + 7m + 5n$, possuindo complexidade final de $O(m^2 + n)$;

Espaço = Como no caso anterior, é criada 3 listas vazias que possuem custo constante, o custo do for da leitura de cursos é 4n e o de leitura de alunos é 6m, após isso a função insere_decrescente() possui custo m e por fim, faz_classificados() possui custo m + n, totalizando no final 4n + 6 m + m + n + m = 8m + 6n, possuindo complexidade de O(m + n).

4. Conclusão

O programa final, quando executado, possui como saída o resultado esperado dos testes fornecidos, possui uma complexidade assintótica polinomial (em que a complexidade da função de ordenar de forma decrescente teve maior peso), o que torna o algoritmo eficiente. Houveram algumas dificuldades ao longo do trabalho como: para ler a string do arquivo após dar scanf em um valor numérico (int e float) foi necessário usar duas vezes o comando fgets, pois primeiramente era pego a quebra de linha (‘\n’) e só após isso era pego a string desejada; na hora de copiar os array de chars também tive problemas, tendo que usar a biblioteca string.h e usar a função strcpy(). Outro problema foi na criação de apenas um tipo de lista, contendo os itens das 3 listas utilizadas (aluno, curso

e classificados) que quando é dado malloc é alocado memória para os 3 itens, mas apenas 1 dos item é utilizado.

5. Bibliografia

ZIVIANI, N., Projeto de Algoritmos com Implementações em Pascal e C. 3ª Edição, Cengage Learning, 2011

.

STACK OVERFLOW. **Ponteiro de char**. Disponível em: <https://pt.stackoverflow.com/questions/136628/por-que-uma-atribui%C3%A7%C3%A3o-de-string-em-c-n%C3%A3o-funciona>. Acesso em: 24 de abril. 2019.

C PROGRESSIVO. **Como copiar uma string em C**. Disponível em: <https://www.cprogressivo.net/2013/03/strcpy-Como-copiar-uma-String-em-C.html>. Acesso em: 24 de abril. 2019.

CASAVELLA, Eduardo. **A biblioteca string.h**. Disponível em: <http://linguagemc.com.br/a-biblioteca-string-h/>. Acesso em: 24 de abril. 2019.