Projeto Prático de Introdução aos Algoritmos (GAC124)

Grupo:

Nome	Número de matrícula
Alexandre de Castro Nunes Borges Filho	202410426
Ana Clara Moreira Viana	202410424
Maycon Henrique Soares de Sousa	202410422

Introdução

Na disciplina de Introdução aos Algoritmos (GAC124), foi solicitado aos discentes que, em grupos formados por duas a três pessoas, construíssem um sistema de cadastro de arquivos com ordenação.

A descrição do projeto prático, apresentada oficialmente às turmas no final do mês de julho, detalha as características, requisitos e restrições para seu desenvolvimento. Suas características incluem, por exemplo, a utilização do redimensionamento de vetores, operação necessária para o atendimento de um dos requisitos do projeto: a opção de alterar o número de dados do sistema por meio de inserções e remoções. Por sua vez, uma das restrições apontadas no documento se refere ao método aplicado para a ordenação das informações, outro importante requisito do trabalho. Na construção do sistema, os estudantes poderiam utilizar um dos métodos eficientes de ordenação: *quick sort*, *merge sort* ou *shell sort*.

Em suma, o objetivo deste trabalho é a implementação de um sistema de cadastro com armazenamento dos dados em arquivos binários. Além disso, tal sistema deve apresentar recursos para busca e alteração das informações contidas no arquivo.

Metodologia

A base de dados: características e forma de elaboração

Além da construção do sistema de cadastro, os discentes deveriam elaborar a própria base de dados de exemplo a ser manipulada no programa. Tal base deve atender às instruções explicitadas no documento da descrição do projeto prático, numeradas a seguir.

1. O arquivo precisa ser constituído de, no mínimo, cinco informações.
2. As informações devem apresentar, no mínimo, dois tipos diferentes, com pelo menos um tipo textual e um numérico.

3. Pelo menos um dos campos deve compreender *strings* com espaços em seu conteúdo.

As bases foram informadas pelos grupos em um fórum no Campus Virtual. Cada grupo deveria escolher um determinado tema e indicar as informações utilizadas com o nome e o tipo da variável.

Nosso grupo optou pelo tema intitulado “Guilda de Heróis”, formado por cinco campos, descritos abaixo.

1. ID: o identificador do membro da guilda. Consiste de um vetor de *char* com dez números.
2. Nome: o nome do membro da guilda. É um tipo textual, uma *string* com espaços.
3. *Rank*: nível do membro da guilda no *ranking* de jogadores. Composto por um único caractere, tipo *char*, o *rank* indica a quarta parte - ‘A’, ‘B’, ‘C’ ou ‘D’ - à qual o membro pertence na hierarquia. ‘A’ representa a porção superior do *ranking*.
4. Porcentagem de sucesso: valor que indica o desempenho do membro da guilda. Consiste de um número do tipo *float*.
5. Atividade: expressa se o membro da está ativo ou inativo. É a única variável do tipo *bool* utilizada em nosso projeto.

Salientamos que, para fins de otimização da produção da base de dados, criamos um código que gera, de forma aleatória, dados a serem preenchidos em nosso projeto. Apesar de que os nomes foram, em sua maioria, preenchidos manualmente, o ID, *rank* e taxa de sucesso de cada entrada foram criados a partir da função *rand* da biblioteca *cmath*.

O desenvolvimento colaborativo: contribuições do GitHub

A fim de gerenciarmos diferentes versões do código, optamos por criar contas no GitHub na primeira fase de desenvolvimento do projeto.

Em uma reunião no Google Meet, um dos integrantes do grupo que já tinha experiência com a utilização do GitHub orientou os outros membros na criação das contas, instalação e configuração geral para uso da plataforma.

Ao longo de todo o processo de construção do sistema, conseguimos manter um histórico de versões do código no GitHub, o que permitiu a identificação de erros, novas formas de implementação de funcionalidades, entre outras vantagens exibidas da plataforma.

Em síntese, cada membro ficou responsável pelo desenvolvimento de módulos específicos do código e, com o suporte do GitHub, as diferentes partes foram sendo mescladas ao longo do tempo.

O código: estrutura e conteúdo

O código construído é formado por diversos módulos que executam tarefas específicas.

Leitura do arquivo binário

O programa fornece dois tipos de leituras, tanto para arquivos tipados (ou binários) como para arquivos separados por vírgula (.csv).

Para a leitura dos arquivos binários, diferentemente do processo executado para o formato .csv (em que acontece um redimensionamento do vetor durante a leitura), utilizamos o tamanho total do arquivo dividido pelo tamanho do registro (levando em consideração que a entrada *string* foi substituída, tanto para a leitura quanto para o salvamento de arquivos tipados, por um *array* de caracteres) para determinar o tamanho do vetor alocado dinamicamente.

Em seguida, lê-se cada entrada do registro para cada posição do vetor de acordo com seu tamanho, tomando um cuidado especial para a conversão da fileira de caracteres (salvos em um *array* de 50 posições) em *string*, em que foram utilizados números para representar espaços e posições a serem ignoradas.

Leitura do arquivo .csv

No caso da leitura dos arquivos com a extensão .csv, em que o redimensionamento do vetor acontece durante o processo de leitura de dados, foi utilizada a biblioteca *sstream* para auxiliar no processo de leitura de dados.

Ao utilizar a função *getline* para ler linha por linha (em que cada uma representa uma posição de nosso futuro vetor), cada uma destas linhas foi transferida para uma *string* temporária para ser lida através da *String Stream*.

Desta maneira, foi possível controlar durante o *loop* o tamanho e o redimensionamento do vetor. Por sua vez, após determinada a disponibilidade de espaço no vetor para a linha a ser lida, extrai-se, através da biblioteca *sstream* e do uso de *getline*, cada entrada separada por vírgulas a ser inserida em nosso registro.

Ordenação dos dados

A ordenação dos dados foi feita a partir da utilização do *quick sort*. O *quick sort* é um algoritmo recursivo sustentado na técnica de dividir-e-conquistar.

Nesse método de ordenação, o processo de divisão é iniciado com a seleção de um pivô; no caso do código que implementamos, este é o último elemento do vetor. Em seguida, os elementos do vetor são rearranjados de forma que todos os elementos à esquerda são maiores ou iguais ao pivô; aqueles à direita, maiores que o pivô. Dá-se o nome de “partição” a essa fase de troca de elementos.

A fase de “conquistar” acontece recursivamente ao chamar a função para o subgrupo de elementos da esquerda e da direita do vetor. Após um certo número de chamadas dessa função, tem-se a fase de “combinar”. É uma fase simples e rápida, já que todos os elementos à direita e à esquerda do vetor já estão ordenados adequadamente.

Busca e remoção de informações

O sistema possui funções que permitem a busca e remoção de membros, utilizando o ID e o nome dos mesmos. A busca é realizada por meio de uma busca binária, o que garante eficiência na localização dos dados desejados. A busca binária, conhecida por sua eficiência em listas ordenadas, verifica se o membro existe na base de dados e retorna as informações correspondentes, caso seja encontrado. Essa técnica é aplicada tanto para a busca por nome quanto por ID, assegurando que a localização dos membros seja rápida e precisa.

A função de remoção de membros é executada com base no ID. Ao encontrar o ID correspondente utilizando a busca binária, o sistema procede à remoção do membro da guilda. Esse processo é crucial para manter a integridade e atualização da lista de membros.

Inserção de informações

Além das funcionalidades de busca e remoção, o sistema também inclui uma função dedicada à inserção de novos membros na guilda. Essa função permite adicionar novos membros ao sistema, garantindo que eles sejam inseridos de maneira ordenada para manter a eficácia da busca binária. Ao inserir um novo membro, a função verifica a posição correta na lista, de acordo com o ID e o nome, e realiza a inserção de forma a não comprometer a ordem existente. Dessa forma, o sistema assegura que a lista de membros esteja sempre pronta para realizar buscas eficientes e manter a organização interna da guilda.

Edição de membros

Em nosso programa, também disponibilizamos a opção de editar membros existentes, ação realizada através da busca de seu índice (a edição só irá ocorrer se o membro for encontrado). Após feita a busca e a captura de seu índice atual, é feita a edição de suas informações.

Display dos dados

Na tela principal do sistema, oferecemos ao usuário duas opções de visualização do arquivo: 1) arquivo inteiro e 2) trecho específico do arquivo. Ao escolher visualizar uma seção do arquivo, pedimos que o usuário informe o início e o fim do trecho. Essas informações são armazenadas em duas variáveis que são posteriormente utilizadas na função *displaySection*. Essa função consiste em uma estrutura de repetição *for* modificada: o contador ‘i’ recebe o valor inserido como início e percorre

as linhas somente até o limite informado como o final da visualização. Caso o usuário insira um valor de "fim" maior que o tamanho do vetor, ele é definido para o tamanho máximo disponível no vetor.

Salvamento

Assim como ocorre no processo de leitura de dados, fornecemos ao usuário duas opções: poder realizar o salvamento no formato de arquivo separado por vírgula (.csv) ou tipado (.dat).

Para o salvamento de arquivos em .csv, inicialmente, são inseridos no arquivo identificadores para cada entrada. Em seguida, cada entrada do registro é escrita para cada posição do vetor, adicionando uma vírgula entre a escrita de cada tipo de dado e, durante a troca da posição a ser acessada no vetor, quebras de linhas.

Já no salvamento na opção .dat, cada posição do registro é inserida no arquivo, para cada posição do vetor, em ordem de acordo com o espaço ocupado pelo tipo de variável em que ele foi utilizado. Porém, foi tomado um cuidado extra na hora de salvar a *string* para garantir um tamanho constante para futuras leituras. Desta forma, a *string* de cada posição do vetor é previamente convertida para um *array* de caracteres com 50 posições antes de serem salvos no arquivo, com espaços e posições não utilizadas sendo sinalizadas por números.

Conclusão

Resultados

O código construído pelo grupo atende aos requisitos descritos pelos docentes e executa a lista de funcionalidades exigidas. Por meio do programa, o usuário é capaz de:

1) Abrir o arquivo tipado;
2) Importar e exportar o arquivo para o formato .csv;
3) Realizar alterações no arquivo (modificar, inserir, remover e ordenar os dados);
4) Realizar a busca de um registro (por ID e por nome);
5) Visualizar o arquivo inteiro ou um trecho de interesse;
6) Salvar o arquivo.

Acertos

Consideramos como acertos as seguintes ações:

1) Utilização de uma plataforma de hospedagem e construção colaborativa do código: o uso do GitHub facilitou a colaboração entre os membros do grupo

na criação do programa, permitindo o acesso a diferentes versões do código e o compartilhamento de informações;

2) Divisão de tarefas: cada membro do grupo pôde focar em resolver problemas específicos do projeto, o que tornou o processo mais eficiente;

3) Implementação de módulos: a organização do programa em unidades modulares não somente favoreceu a legibilidade do código, mas também tornou mais fácil sua manutenção;

4) Ajuda mútua constante: a divisão de tarefas não implicou a ausência de cooperação entre os membros na realização de suas respectivas atribuições. Pelo contrário, ao encontrar dificuldades, um membro contatava outro em busca de auxílio.

Dificuldades e/ou erros

A seguir, as dificuldades encontradas pelos integrantes durante o desenvolvimento do projeto prático.

Alexandre

Durante o processo de leitura e salvamento de arquivos tipados, foi necessário descobrir como este processo é feito e também entender a lógica da leitura e escrita de blocos de informações de acordo com o tamanho da variável sendo utilizada. Por meio de pesquisas, constatou-se que era necessário o uso de um *array* de caracteres para poder manter este processo consistente.

Também durante a leitura, para evitar possíveis erros com a leitura de arquivos não existentes ou de arquivos não adequados para o programa, aprender como verificar a existência de um arquivo a ser lido para diminuir a chance de leitura indevida de dados de arquivos não adequados representou outro obstáculo.

Uma outra dificuldade foi realizar a leitura e salvamento de arquivos tipados, principalmente na forma em que o trabalho exigia que as leituras fossem feitas: com um tamanho mínimo e redimensionamento de posições realizadas, em nosso projeto, de cinco a cinco. O método que, para mim, foi mais intuitivo, foi realizar a leitura de linha por linha, controlando, desta forma, qual posição a ser acessada em nosso vetor, para depois realizar a leitura e conversão para cada *string* lida em seu devido tipo de variável no registro.

Ana Clara

Minha maior dificuldade foi a compreensão do funcionamento do *quick sort*. Nosso grupo iniciou o desenvolvimento do projeto não muito tempo após suas instruções terem sido divulgadas e, nessa época, os métodos de ordenação ainda não haviam sido explicados aos estudantes. Assim, tive de fazer pesquisas e assistir a alguns

vídeos sobre o assunto para que conseguisse implementar essa funcionalidade no código. Mesmo assim, como o *struct* era um conceito não muito intuitivo para mim, foi necessária a ajuda de um integrante do grupo na transposição do código que eu havia criado, que utilizava vetores simples, para o programa referente ao projeto prático.

Acredito que, devido a dificuldades com certos conteúdos da disciplina (como o redimensionamento de vetores), posso dizer que, para mim, o projeto prático apresentou uma complexidade considerável. Diferentemente dos demais integrantes do grupo, eu não possuía uma grande bagagem de conhecimentos da área da programação. Por isso, diversas funcionalidades do código que, para eles, eram relativamente intuitivas, para mim, eram extremamente complicadas.

Por fim, posso afirmar que, apesar dos problemas apontados, o desenvolvimento do trabalho em grupo constituiu um processo enriquecedor. Pude aprender bastante com os colegas e exercitar minha autoconfiança.

Maycon

A implementação da busca binária foi um desafio e tanto, especialmente porque é necessário considerar mais de um critério de busca, como o nome e o ID do membro. A busca binária exige que a lista esteja ordenada. Qualquer falha na lógica de busca pode levar a resultados incorretos, dificultando a localização precisa dos membros na lista.

Além disso, a remoção de membros com base no ID me forçou a ser cuidadoso para garantir que todos os dados associados ao membro fossem removidos corretamente. A lógica de remoção não foi implementada de maneira correta no início e a persistência de registros inconsistentes causou inúmeros problemas em outras partes do sistema. Após algumas horas de trabalho em dias variados, o problema foi superado.

Outro aspecto que foi desafiador para mim foi a necessidade de realizar testes de validação das funcionalidades. Testar combinações de nomes e IDs para garantir que as operações de busca, inserção e remoção funcionassem corretamente foi uma tarefa complexa. A cada nova ideia de teste que surgia, um erro na lógica surgia em paralelo; assim, por vezes foi frustrante.

Por fim, considerando que o número de membros na guilda poderia crescer, a performance das operações foi uma preocupação presente durante todo o processo. Manter o sistema performático para garantir uma melhor experiência dos professores e possíveis usuários, bem como não comprometer a integridade dos dados, foram elementos que aumentaram a dificuldade do processo.