UNIVERSIDADE ESTADUAL DE CAMPINAS
COLÉGIO TÉCNICO DE CAMPINAS – COTUCA

Projeto III - Estrutura de Dados

Relatório

Felipe Melchior de Britto RA:18200

Gabrielle da Silva Barbosa RA:18183

Objetivos do projeto: Criar um programa em Windows Forms em que um usuário possa escolher duas cidades, uma origem e um destino. O programa deverá mostrar todos os caminhos possíveis até ela, além do melhor caminho, levando em conta a quilometragem que será percorrida.

Desenvolvimento:

03/06:

- Começamos o projeto fazendo as classes Cidade e CaminhoEntreCidades, nas quais adicionamos todos os atributos necessários, além dos tamanhos e inícios destes, para que a leitura de arquivo pudesse ser realizada;
- Fizemos também métodos estáticos para leitura de um arquivo que retornavam objetos, de Cidade ou de CaminhoEntreCidades;
- Adicionamos as classes necessárias, sendo elas Arvore, ListaSimples, PilhaLista, etc… Utilizamos estas classes para a criação de variáveis de armazenamento de dados durante a execução do programa

05/06:

- Iniciamos a programação do formulário, começando pela leitura de arquivos. Para isso, fizemos o padrão, criamos um StreamReader, chamamos as classes para leitura de uma linha do arquivo e a utilizamos para o que precisávamos;
- Para a árvore binária de cidades lemos o arquivo CidadesMarte.txt, pois encontrava-se no melhor formato para tal;
- Para mostrar as cidades ao usuário lemos o arquivo CidadesMarteOrdenado.txt, já que já estava ordenado, facilitaria a construção da lógica do programa;
- Por fim, para os caminhos, lemos o arquivo CaminhosEntreCidades.txt e colocamos os caminhos encontrados em uma matriz de adjacências, de tamanho 23x23 (quantidadeDeCidades x quantidadeDeCidades).

06/06:

- Fizemos o código de backtracking recursivamente e para um caminho apenas, para entender de forma simples como a execução ocorreria;
- Para mudar de cidades utilizamos da matriz de adjacências para saber para onde percorrer, indo de 0 até a quantidade de cidades na árvore (23 no caso). No método, que chamamos de BuscarCaminhos e que inicialmente recebe de parâmetros idOrigem e idDestino, primeiramente verificamos sua saída, ou seja, se a “origem” é igual ao “destino”, e, se não, percorremos a matriz de adjacências. Se nenhum caminho for encontrado , como trata-se de backtracking, o método Retornar é chamado e, se dessa vez encontrar, chamamos o método de novo, com idOrigem como a próxima cidade, e empilhamos em uma pilha de caminho o idOrigem anterior;

```
private void BuscarCaminhos(int idOrigem, int idDestino)
{
    if (idOrigem == idDestino)
        caminho.Empilhar(idDestino);
    else
    {
        int c = -1;
        for (int i = 0; i < adjacencias.GetLength(0); i++)
        {
            if (adjacencias[idOrigem, i] != 0)
            {
                c = i;
                break;
            }
        }
        if (c == -1)
            Retornar(idOrigem, idDestino);
        else
        {
            caminho.Empilhar(idOrigem);
            BuscarCaminhos(c, idDestino);
        }
    }
}
```

- No método Retornar, a pilha de caminhos é desempilhada e o método BuscarCaminhos é chamado novamente.

```
3 references
private void Retornar(int idOrigem, int idDestino)
{
    int c = caminho.Desempilhar();
    BuscarCaminhos(c, idDestino);
}
```

10/06:

- Percebemos que nosso código estava resultando num loop, pois nada impedia que se passasse pela mesma cidade mais de uma vez e ficasse, assim, “rodando”. Para resolver isso adicionamos uma variável indiceInicial ao método BuscarCaminhos, assim ao invés de começar a percorrer a matriz sempre do 0, começava-se do índice mandado, que quando se retornava uma posição era da cidade anterior acrescido de um;
- Além disso, criamos um vetor de boolean do tamanho da quantidade de cidades (nisso não pensamos sozinhos , inicialmente queríamos fazer algo muito mais demorado e ineficaz, um vetor de int de cidades percorridas, os monitores nos deram a “dica” de utilizar lógica booleana), assim o vetor indexado do id da cidade estaria “true” caso ela já houvesse sido percorrida, permitindo-nos impedir de passar novamente por aquela cidade.

```
private void BuscarCaminhos(int idOrigem, int idDestino, int indiceInicial)
{
    if (idOrigem == idDestino)
        caminho.Empilhar(idDestino);
    else
    {
        int c = -1;
        for (int i = indiceInicial; i < adjacencias.GetLength(0); i++)
        {
            if (adjacencias[idOrigem, i] != 0 && cidadesPercorridas[i] == false)
            {
                c = i;
                break;
            }
        }
        if (c == -1)
            Retornar(idOrigem, idDestino);
        else
        {
            caminho.Empilhar(idOrigem);
            cidadesPercorridas[idOrigem] = true;
            BuscarCaminhos(c, idDestino, 0);
        }
    }
}
```

```
3 references
private void Retornar(int idOrigem, int idDestino)
{
    int c = caminho.Desempilhar();
    BuscarCaminhos(c, idDestino, idOrigem + 1);
}
```

11/06:

- Com o backtracking funcionando, começamos a pensar num modo de conseguir não apenas percorrer um caminho, e sim todos os possíveis;
- Criamos uma List de caminhos possíveis para armazená-los e fizemos um método Clone na classe PilhaLista, para que caso a pilha de caminho fosse esvaziada, a pilha guardada na List não seria alterada;
- Depois de muito tempo pensando, tivemos a ideia de utilizar o próprio método Retornar para achar outros caminhos, criamos uma variável boolean PercorreuTodosOsCaminhosPossiveis, que não sabíamos como verificar ainda e colocamos o Retornar dentro de um while, já que o próprio mandava buscar caminhos e sairia dele apenas quando houvesse chegado

novamente;

```
private void BuscarCaminhos(int idOrigem, int idDestino, int indiceInicial)
{
    if (idOrigem == idDestino)
    {
        caminho.Empilhar(idDestino);
        caminhosPossiveis.Add(caminho.Clone());
    }
    else
    {
        int c = -1;
        for (int i = indiceInicial; i < adjacencias.GetLength(0); i++)
        {
            if (adjacencias[idOrigem, i] != 0 && cidadesPercorridas[i] == false)
            {
                c = i;
                break;
            }
        }
        if (c == -1)
        {
            if (caminho.EstaVazia())
                percorreuTodosOsCaminhosPossiveis = true;
            else
                Retornar(idOrigem, idDestino);
        }
        else
        {
            cidadesPercorridas[idOrigem] = true;
            caminho.Empilhar(idOrigem);
            BuscarCaminhos(c, idDestino, 0);
        }
    }
}
```

- Percebemos outro erro, pois havia uma pequena quantidade de caminhos possíveis que estava sendo achada, e verificamos que isso era por causa do nosso vetor de boolean (ele guardava as cidades já percorridas, mas para outro caminho ser possível pode ser que se passe por cidades repetidas). Pensamos em “reinstanciar” o vetor, mas isso não funcionou também. Por fim, chegamos à ideia de, ao retornar uma posição, colocar false nela novamente no vetor, o que não daria erro justamente pelo uso do indiceInicial.

```
3 references
private void Retornar(int idOrigem, int idDestino)
{
    int c = caminho.Desempilhar();
    cidadesPercorridas[idOrigem] = false;
    BuscarCaminhos(c, idDestino, idOrigem + 1);
}
```

13/06:

- Chegamos a uma forma de verificar se já havia acabado os caminhos disponíveis, e esta foi, no método BuscarCaminhos, se a pilha de caminho estava vazia e não havia achado nenhum caminho possível. Este fato queria dizer que o Retornar já tinha voltado tanto que idOrigem era o original e não havia mais nenhum caminho possível a partir dele;
- Também fizemos um método responsável por pegar o melhor caminho, ou seja, o caminho de menor quilometragem, para isso íamos desempilhando os caminhos, verificando o valor das adjacências e somando, e, quando calculado o caminho todo, o comparávamos com o menor antes encontrado;
- Fizemos métodos para desenhar caminhos, cidades e a árvore de cidades (este último pegamos de um exercício em aula, então está praticamente igual).

14/06:

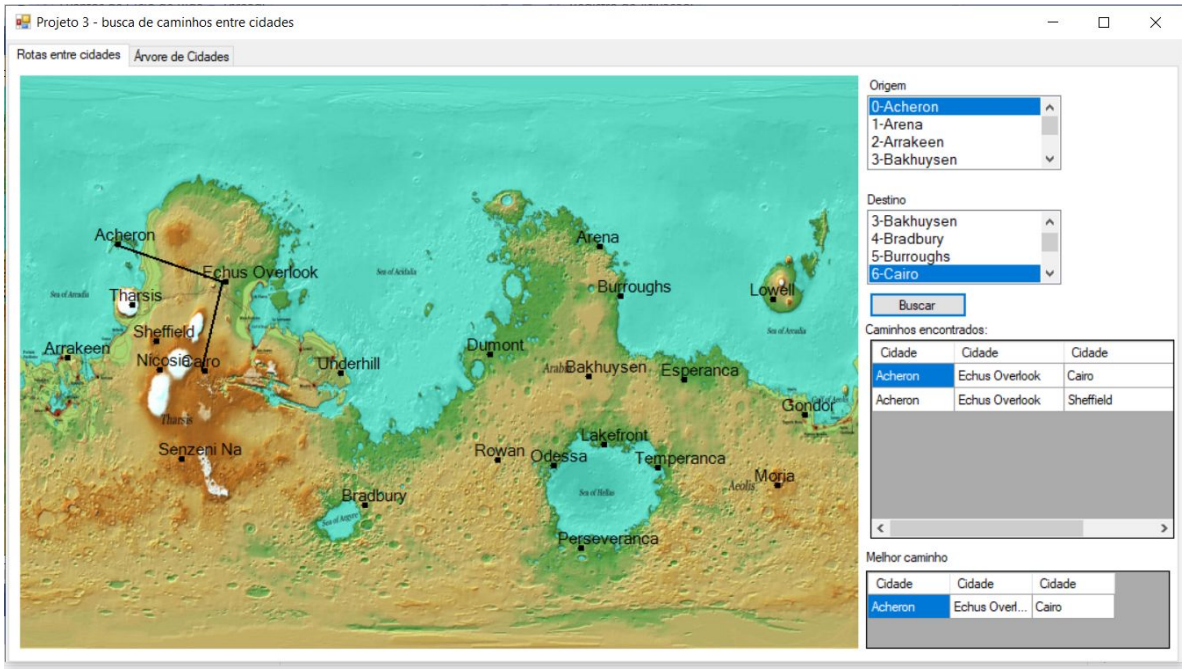
- Achamos uma boa ideia, ao invés de apenas mostrar o melhor caminho, ordenar toda a lista de possibilidades do melhor para o pior. Por conta de ordenarmos a lista, não precisamos mais do método que antes encontrava o melhor, pois este a partir de agora encontrava-se na primeira posição;
- Corrigimos um pequeno errinho referente ao desenho dos caminhos, pois a reta entre “Gondor” e “Arrakeen” e entre “Gondor” e “Senzen Na” passavam por trás do mapa, por assim dizer. Para verificar este fato fizemos uma conta entre a distância horizontal que as cidades tinham e a distância horizontal que teriam “por trás”, baseando-nos nas próprias coordenadas do mapa do formulário.

16/06:

- Descobrimos que estávamos guardando as adjacências lidas da origem para o destino e do destino para a origem, então tivemos que arrumar este problema e onde ele afetava, o que não foi tão difícil;
- Também verificamos se a lista de caminhos possíveis não estava vazia, para assim não dar erro ao mandarmos mostrá-la.

Conclusão:

programa rodando



Conseguimos finalizar o projeto e executá-lo por completo sem muitos problemas ou dificuldades. Foi de grande importância para a compreensão e aprendizado das atividades desenvolvidas durante as aulas, contribuindo para com nosso entendimento sobre árvores em estrutura de dados e a utilização de lógica e métodos recursivos.

Pudemos aperfeiçoar também nossos conhecimentos sobre backtracking, através da busca por caminhos. Este método de busca foi essencial para a realização da procura

Acabamos por aprender também um pouco mais sobre a utilização de imagens e sua manipulação na interface com o usuário para a criação visual de uma árvore de cidades e para a ilustração das cidades de Marte e o caminho entre elas.