Trabalho 1

Flávio Marcílio de Oliveira

1. Modelagem Computacional do Problema

O problema consiste em uma versão do problema do Casamento Estável, em que visitantes serão alocados a bicicletas. A lista de preferência dos visitantes é dada e a lista de preferência das bicicletas é dada pela menor distância da bicicleta para o visitante que deve ser determinada.

Para determinar a distância entre as bicicletas e os visitantes, o mapa da região é modelado como um grafo não direcionado e não ponderado com os vértices representando os diversos pontos do percurso e as arestas representando os trechos disponíveis para o percurso. A distância entre dois pontos adjacentes é considerada uma unidade de medida.

Uma vez determinado o grafo com a respectiva numeração dos nós, os nós correspondentes aos visitantes e às bicicletas são identificados. Com essa identificação, o algoritmo de Busca em Largura (BFS) é utilizado para determinar a distância mínima entre os visitantes e as bicicletas.

Com a lista de preferência dos visitantes para cada bicicleta e a lista de distância mínima das bicicletas para cada visitante, o algoritmo de Gale-Shapley é utilizado para resolver o problema da alocação de cada visitante para cada bicicleta, sendo que o visitante tem a iniciativa da escolha de forma a priorizar a ordem de preferência.

2. Descrição da Solução

a. Estruturas de Dados

struct percurso para armazenar a distância entre dois nós;

map<int, int> visitantes e bicicletas para guardar a relação dos visitantes e bicicletas e seus respectivos nós de localização no grafo; Foi utilizado o número referente ao código ASCII dos visitantes (97 a 106) e bicicletas (48 a 57).

vector<int> adj para representar o grafo por uma lista de adjacência;

vector<pair<int,int>> prefVis e prefBike para armazenar a lista de preferência dos visitantes e das bicicletas. O *pair* é utilizado para guardar o ID da bicicleta e o grau de preferência do visitante para esta bicicleta no caso da *prefVis* e, no caso da *prefBike*, o ID do visitante e a distância da bicicleta a este visitante;

vector<int> comprimento para calcular a distância de um nó a outro;

vector<bool> visited para identificar os nós visitados no algoritmo de Busca em Largura.

vector<percurso> percursos para armazenar todas as distâncias entre visitantes e bicicletas;

visList matriz bidimensional de preferências dos visitantes, onde visLis[i][j] = b indica que a bicicleta b está na j-ésima posição de preferência do visitante i.

bikList matriz bidimensional de distâncias, onde bikList[i][j] = v indica que o visitante v é o j-ésimo mais próximo da bicicleta i.

visAlocado e *bikAlocada* vetor para representar a alocação;

visProxEscolha que representa a lista de escolha dos visitantes.

b. Algoritmos

bfs(s) - Algoritmo de busca em largura que calcula o caminho mais curto do nó s a todos os outros nós do grafo dado;

compareVis - Função de comparação utilizada para criar a lista de preferência dos visitantes de acordo com o grau de preferência e levando em consideração o critério de desempate para o mesmo grau de preferência para duas bicicletas;

compareBik - Função de comparação utilizada para criar a lista de preferência para as bicicletas considerando as distância para os visitantes e levando em conta o critério de desempate para a mesma distância para dois visitantes;

main - Programa principal

No programa principal o arquivo de entrada é lido armazenando o mapa em um vetor de strings. A ordem de preferência dos visitantes é lida e criada utilizando o algoritmo sort para ordenação segundo os critérios dados utilizando a função *compareVis*. A matriz de preferências *visList* é populada utilizando o vetor de preferências dos visitantes ordenado e a lista de alocação para cada visitante é inicializada como 0 marcando que ninguém foi alocado. Em seguida, o grafo é construído utilizando a representação por lista de adjacências verificando a existência de obstáculos no vetor de strings lido, percorrendo cada caractere em cada linha. A verificação da localização dos visitantes e bicicletas é feita nesta etapa, criando o mapeamento para identificação em quais nós estão localizados. Os nós foram numerados segundo a ordem mostrada no exemplo abaixo (mapa 4 x 4):

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Após a criação do grafo, o algoritmo de Busca em Largura (bfs) é rodado para cada visitante existente. Com a lista de distâncias encontrada para cada visitante em relação às bicicletas é criado um vetor para representar as distâncias para cada bicicleta e depois este vetor é ordenado utilizando a função *compareBik* para criar a lista de preferências das bicicletas.

O passo final é rodar o algoritmo de Gale-Shapley para fazer a alocação de cada visitante para cada bicicleta, onde cada visitante escolhe a bicicleta na sua ordem de preferência. O pseudocódigo do programa desenvolvido é apresentado abaixo.

```
num ← Número de visitantes e bicicletas
linhas ← quantidade de linhas do mapa
colunas ← quantidade de colunas do mapa
// Lendo o mapa
para i de 0 até linhas faça:
    aux[i] ← conjunto de caracteres da linha
// Lendo as preferências dos visitantes
para i de 0 até num faça:
    para j de 0 até num faça:
        prefVis[i] ← (j, grau de preferência)
    ordene(prefVis[i])
// Gale-Shapley - Inicializando a matriz de preferências dos visitantes
para i de 1 até num faça:
    Inicializa todos visitantes como não alocado
    Define a primeira escolha de visitante cada visitante
    para j de 0 até num faça:
        visList[i][j] ← ID da bicicleta que está na j-ésima posição de preferência do visitante i
// Criando o grafo com Listas de Adjacência
para i de 0 até linhas faça:
    para j de 0 até colunas faça:
        se aux[i][j] é diferente de '-' faça:
            se i é maior que 0 faça:
                // Criando o mapeamento de visitantes e bicicletas
                vis ← (int)aux[i][j];
                bic ← (int)aux[i][j];
                se vis é maior ou igual a 97 e vis é menor ou igual a 106 faça:
                    visitantes[i * colunas + j] ← vis;
                se bic é maior ou igual a 48 e bic é menor ou igual a 57 faça:
                    bicicletas[i * colunas + j] ← bic;
                // Criando a Lista de Adjacência
                se aux[i - 1][j] é diferente de '-' faça:
                    adj[i * colunas + j] ← colunas * (i - 1) + j;
            se i é menor que linhas - 1 faça:
```

```
                // Criando o mapeamento de visitantes e bicicletas
                vis ← (int)aux[i][j];
                bic ← (int)aux[i][j];
                se vis é maior ou igual a 97 e vis é menor ou igual a 106 faça:
                    visitantes[i * colunas + j] ← vis;
                se bic é maior ou igual a 48 e bic é menor ou igual a 57 faça:
                    bicicletas[i * colunas + j] ← bic;
                // Criando a Lista de Adjacência
                se aux[i + 1][j] é diferente de '-' faça:
                    adj[i * colunas + j] ← colunas * (i + 1) + j;
            se j é maior que 0 faça:
                // Criando o mapeamento de visitantes e bicicletas
                vis ← (int)aux[i][j];
                bic ← (int)aux[i][j];
                se vis é maior ou igual a 97 e vis é menor ou igual a 106 faça:
                    visitantes[i * colunas + j] ← vis;
                se bic é maior ou igual a 48 e bic é menor ou igual a 57 faça:
                    bicicletas[i * colunas + j] ← bic;
                // Criando a Lista de Adjacência
                se aux[i][j-1] é diferente de '-' faça:
                    adj[i * colunas + j] ← colunas * i + j - 1
            se j é menor que colunas - 1 faça:
                // Criando o mapeamento de visitantes e bicicletas
                vis ← (int)aux[i][j];
                bic ← (int)aux[i][j];
                se vis é maior ou igual a 97 e vis é menor ou igual a 106 faça:
                    visitantes[i * colunas + j] ← vis;
                se bic é maior ou igual a 48 e bic é menor ou igual a 57 faça:
                    bicicletas[i * colunas + j] ← bic;
                // Criando a Lista de Adjacência
                se aux[i][j+1] é diferente de '-' faça:
                    adj[i * colunas + j] ← colunas * i + j + 1
```

```
// Calculando as distâncias entre cada visitante e as bicicletas
para cada visitante faça:
    bfs(visitante)
// Criando a estrutura para facilitar a criação da lista de preferência das biciletas
para i de 0 até total de caminhos faça:
    prefBike[ID bicicleta] ← (visitante, distancia)
// Criando a Ordem de Preferência das bicicletas
para i de 0 até num faça:
    ordene(prefBike[i])
// Gale-Shapley - Inicializando a matriz de preferências para as bicicletas
para i de 1 até num faça:
    Inicializa todas as bicicletas como Não Alocada
    para j de 0 até num faça:
        bikList[i][j] ← ID do visitante que tem a j-ésima menor distância da bicileta i
// Algoritmo de Gale-Shapley
enquanto existir visitante livre faça:
    escolhe uma bicicleta na lista de preferência do visitante
    se a bicicleta escolhida não estiver alocada faça:
        Aloca a bicicleta para o visitante
        Aloca o visitante para a bicicleta
    se a bicicleta estiver alocada faça:
        Verifica se o visitante está mais perto do que o que foi alocado para a bicicleta
        se o visitante está mais perto faça:
            Aloca a bicicleta com o novo visitante e desaloca o antigo
    Encontra um novo visitante livre
Apresenta o resultado da alocação
```

3. Análise de Complexidade de Tempo Assintótica

Considerando m nós, n arestas e v visitantes.

A criação do grafo tem complexidade O(m), pois é necessário percorrer todas as linhas e todas as colunas do mapa, sendo m = linhas * colunas;

A complexidade da Busca em largura é O(m+n), como é rodado para v visitantes então a etapa de calcular a distância para todos os visitantes será O(vm + vn).

A complexidade do Gale-Shapley é $O(v^2)$.

Todo o pré-processamento realizado - criação das listas de preferências, inicialização das matrizes - é no máximo $O(v^2)$.

Portanto, a complexidade global do programa será $O(vm + vn + m + v^2)$.

4. Conclusão

A implementação foi desenvolvida utilizando um grafo representado por lista de adjacência por usar a memória de maneira mais eficiente.

Os algoritmos tanto de Busca em Largura quanto Gale-Shapley seguiram conforme apresentado em sala de aula buscando melhor eficiência.

Os resultados foram obtidos mostrando a corretude da implementação em solucionar o problema proposto.