Trabalho Prático 1 de Algoritmos 1

Nome: Artur Gaspar da Silva
Matrícula: 2020006388

Modelagem computacional do Problema:

Desejamos encontrar um pareamento entre visitantes e bicicletas que seja justo e que não emparelhe dois visitantes a uma mesma bicicleta nem duas bicicletas a um mesmo visitante (ou seja, válido). Para isso, nós modelamos o problema como um grafo bipartido completo, em que de um lado estão os visitantes e de outro estão as bicicletas, além de cada visitante possuir uma ordem de preferência de todas bicicletas e cada bicicleta possuir uma ordem de preferência de todos visitantes. Portanto, se um visitante ver duas bicicletas quaisquer, ele vai preferir escolher alguma, e se toda bicicleta "ver" dois visitantes, ela vai preferir escolher algum deles.

No entanto, nosso problema não é dado diretamente dessa forma. Precisamos primeiro converter a entrada. O mapa da lagoa é usado para computar as preferências das bicicletas (um visitante mais próximo é preferido a um mais distante, e em caso de empate, o visitante com menor ID é preferido). Além disso, a lista de "preferências" dos visitantes que nos é fornecida nem sempre é uma ordem total no conjunto das bicicletas, pois um visitante pode possuir mesma "preferência" por duas bicicletas. Por isso temos o critério de desempate por ID da bicicleta, e podemos montar a verdadeira lista de preferências dos visitantes. Portanto, note que essa modelagem pode ser (e foi) feita a partir da entrada fornecida.

Estruturas de Dados e Algoritmos utilizados na resolução:

Primeiramente, apresentamos o pseudocódigo do nosso programa (procedimentos auxiliares declarados abaixo do principal):

TP01(mapa da lagoa, preferências iniciais dos visitantes):

Ordenar preferências dos visitantes de forma que caso um visitante inicialmente tenha mesmo nível de preferência por duas bicicletas, na nova ordem ele vai preferir a bicicleta com menor ID (e nos outros casos a velha ordem é mantida).

Inicializar matriz M

Para toda bicicleta *B* no mapa:

L← **BFS_pares**(mapa da lagoa, posição de *B*)

Ordenar L de forma que os com menor distância sejam priorizados, e em caso de empate os de menor ID.

Colocar na matriz M na posição i, *B* a posição do visitante i na ordem de preferências da bicicleta *B*

Retornar **Gale_Shapley**(lista de preferências das bicicletas, lista de preferências dos visitantes)

BFS_pares(mapa da lagoa, posição no mapa P):

Inicializar fila de visitação F já com a posição P
Inicializar matriz de marcação de posições já visitadas H
Inicializar lista L vazia
Enquanto F não está vazia:
Remover a primeira posição K da fila F
Marcar na matriz H que K já foi visitada
Se a posição H está ocupada por uma pessoa Ç no mapa da lagoa:
Inserir a distância da bicicleta até H, e o ID de Ç na lista L
Para todas posições G para onde podemos em um passo a partir de H:
Inserir G na fila F

Para implementar esse algoritmo, utilizou-se estruturas da Standard Template Library (STL) do C++. Mais especificamente, utilizamos filas, vetores, pares e tuplas (*queue*, *vector*, *pair* e *tuple*):

- Lemos a entrada utilizando *cin* e *cout*;
- Guardamos o mapa da lagoa numa matriz de caracteres (criada usando um *vector* de *vector* de *char*);
- Guardamos as prioridades dos visitantes como um vetor de filas (a entrada *i* do vetor indica a fila de para qual bicicleta o visitante *i* ainda pode propor emparelhamento);
- Guardamos as prioridades das bicicletas numa matriz (a entrada *i, j* da matriz indica qual a posição do visitante i na ordem de prioridades da bicicleta *j*);
- Guardamos o emparelhamento resultante num vetor (a entrada *i* do vetor indica com quem a);

Utilizamos também algumas estruturas intermediárias durante a execução do programa:

- Um vetor de pares de inteiros que guarda os pares {-prioridade, id da bicicleta}, que ordenamos depois de preenchido para obter as reais preferências dos visitantes;
- Um vetor de pares de inteiros que guarda os pares {distância, id do visitante}, que ordenamos depois de preenchido para obter as preferências das bicicletas;
- Uma fila de tuplas de três inteiros {coordenada x, coordenada y, distância da origem} que usamos durante a BFS auxiliar para preencher o vetor do item acima;
- Uma matriz de booleanos usada durante a BFS auxiliar para marcar quais posições já foram visitadas na BFS;
- Uma fila de visitantes ainda não emparelhados, usada durante a execução do *Gale-Shapley*.
- Um vetor de pares {ID do visitante, ID da bicicleta} emparelhados ao final, que ordenamos depois de preenchido para termos a resposta na ordem desejada pela especificação do problema.

Por fim, podemos destacar que utilizamos a função *sort* do STL (*algorithm*) para realizar as ordenações necessárias, e *assert* (de *cassert*) para manter robustez no programa e facilitar a identificação de erros caso alterações sejam feitas.

Análise de complexidade:

Podemos dividir nosso programa nas seguintes etapas:

- *Leitura da entrada:*

Para essa etapa, precisamos ler 3 números pros tamanhos da matriz (*l* linhas e *c* colunas) e quantidade de visitantes (*n*), depois precisamos ler *lc* caracteres da lagoa e depois n^2 números, que são as prioridades dos estudantes. Isso significa que leremos $\mathbf{O(lc+n^2)}$ caracteres. Como cada caractere pode ser lido em tempo constante, a complexidade de tempo dessa etapa é de $\mathbf{O(lc+n^2)}$.

- *Processamento das preferências dos visitantes pras suas preferências reais:*

Para realizar esse processamento nós colocamos os pares de preferências iniciais dos visitantes e id da bicicleta num vetor (**O(n)** pares, cada um colocado em tempo constante) e ordenamos este vetor (custo **O(n log(n))** pois o vetor tem tamanho *n*), depois colocamos o resultado ordenado na fila do visitante correspondente (**O(n)** pois a adição de cada bicicleta na fila gasta tempo constante). Isso totaliza complexidade de tempo **O(n log(n))** nessa etapa.

- *Processamento do mapa da lagoa para obter as preferências das bicicletas:*

Essa é a etapa mais custosa do nosso programa para os casos extremos, devido ao fato de que o mapa da lagoa pode ser muito maior que o número de visitantes e bicicletas. Para cada bicicleta (são **O(n)** bicicletas), executamos uma BFS que anda por toda a parte acessível do mapa para descobrir as prioridades das bicicletas, gastando **O(lc)**. Ou seja, **O(nlc)** no total. Para encontrar as bicicletas também andamos por toda a matriz, mas isso adiciona apenas mais uma passada na matriz, não alterando a complexidade.

- *Execução do Gale-Shapley:*

Durante o algoritmo de Gale-Shapley um visitante nunca tenta emparelhar duas vezes com a mesma bicicleta. Isso significa que teremos no máximo n^2 tentativas de emparelhamento. Como quando não há mais tentativas válidas o algoritmo termina e cada tentativa de emparelhamento toma tempo constante, essa etapa possui complexidade assintótica de tempo $\mathbf{O(n^2)}$.

- *Ordenação da saída:*

Após a execução do Gale-Shapley, teremos n pares de estudantes e bicicletas emparelhados, e para ordená-los no formato de saída gastamos **O(n log(n))** de tempo.

- *Impressão da saída:*

Finalmente, imprimimos a saída em tempo **O(n)** pois a impressão de cada par emparelhado gasta tempo constante.

Ou seja, no total o algoritmo possui complexidade de tempo $\mathbf{O(lc+n^2)+O(n\ log(n))+O(nlc)+O(n^2)+O(n\ log(n))+O(n) = O(nlc)}$, lembrando que *n* é o número de visitantes que é igual ao número de bicicletas, *l* é a quantidade de linhas na matriz do mapa da lagoa, e *c* é a quantidade de colunas na matriz do mapa.