Universidade do Estado de Santa Catarina (UDESC)
Curso: Tecnologia em Análise e Desenvolvimento de Sistemas
Disciplina: Estrutura de dados (EDA0001)
Prof. Rui Jorge Tramontin Jr.

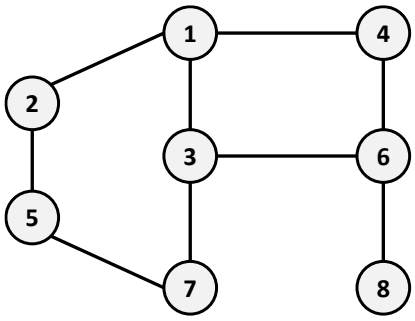
Trabalho 2: Implementando BFS com uma Fila

O objetivo deste trabalho é implementar o algoritmo BFS (*Breadth-first Search*) ou ***Busca em Largura*** em um grafo. Tal como a busca em profundidade, é um algoritmo usado para realizar uma busca ou travessia num grafo. Ele se inicia em algum vértice arbitrário do grafo e explora todos os vértices vizinhos (no mesmo nível), antes de se mover para os vértices no próximo nível de profundidade. Para isso, a implementação desse algoritmo utiliza uma **fila**.

O objetivo do algoritmo neste trabalho é implementar a busca no grafo, dados o vértice inicial A e o vértice final B. O algoritmo começa em A e faz a busca em largura até encontrar B. Uma aplicação desse tipo de algoritmo é que ele permite encontrar o **menor caminho entre A e B**, em termos de número de arestas percorridas. Nesse sentido, após a aplicação da busca, a saída deve ser a sequência de vértices visitados entre A e B.

Implementação

Um grafo pode ser representado por uma matriz de adjacência, na qual cada linha representa um vértice e cada vértice adjacente a ele (coluna) possui valor 1, ou 0 para os que não são adjacentes (adjacente aqui signica estar conectado). Por exemplo, considere o grafo a seguir e sua respectiva matriz de adjacência:



	1	2	3	4	5	6	7	8
1	0	**1**	**1**	**1**	0	0	0	0
2	**1**	0	0	0	**1**	0	0	0
3	**1**	0	0	0	0	**1**	**1**	0
4	**1**	0	0	0	0	**1**	0	0
5	0	**1**	0	0	0	0	**1**	0
6	0	0	**1**	**1**	0	0	0	**1**
7	0	0	**1**	0	**1**	0	0	0
8	0	0	0	0	0	**1**	0	0

A matriz pode ser facilmente implementada utilizando a biblioteca de matrizes dinâmicas feita como exercício em aula (Lista 2). Repare que, para o usuário, os vértices são numerados a partir de 1, enquanto que na linguagem C, os índices começam em 0.

As seguintes estruturas são necessárias para o algorirmo:

1. Matriz de adjacência: alocada dinamicamante, conforme entrada do usuário;
2. Vetor de *status* dos vértices: também alocado dinamicamente; indica se cada vértice foi visitado (1), ou se ainda não foi (0);

3. Vetor com o antecessor de cada vértice: semelhante ao vetor de *status*, mas serve para registrar qual é o antecessor de cada vértice visitado. Serve para determinar o caminho entre A e B;
4. Fila de inteiros: bilioteca de filas (vetor dinâmico);
5. Pilha de inteiros: bilioteca de pilhas (vetor dinâmico), usada para mostrar o caminho percorrido (após a busca, é preciso fazer o percurso na ordem invertida).

O algoritmo é implementado conforme o seguinte pseudocódigo:

```
 1. ENTRADA DE DADOS:
     - Matriz de adjacência (sua dimensão e seu conteúdo);
     - Vértice inicial do percurso (A);
     - Vértica final do percurso (B);
 2. INICIALIZA VETOR DE STATUS (VS): todos os índices com zero;
 3. INICIALIZA VETOR DE ANTECESSORES (VA): todos os índices com zero;
 4. INICIALIZA FILA F;
 5. VS[A] <- 1;              // Marca A como visitado,
 6. INSERE A em F;           // e insere em F para começar o algoritmo.
 7. ACHOU <- FALSO           // Variável booleana
 8. ENQUANTO F NÃO ESTIVER VAZIA E NÃO ACHOU FAÇA
 9.    REMOVE O VÉRTICE X DE F;       // Remove vértice.
10.    SE X = B ENTÃO                 // Se for o vértice final...
11.       ACHOU <- VERDADEIRO;        // Interrompe do laço.
12.    SENÃO
13.       PARA CADA VÉRTICE I ADJACENTE A X FAÇA
14.          SE VS[I] = 0 ENTÃO      // Se ainda não foi visitado...
15.             VS[I] <- 1;          // - marca I como visitado;
16.             VA[I] <- X;          // - marca X como antecessor de I;
17.             INSERE I EM F;       // - e insere para análise futura.
18.          FIMSE
19.       FIMPARA
20.    FIMSE
21. FIMENQUANTO
22. SE ACHOU ENTÃO
23.    INICIALIZA PILHA P;
24.    ENQUANTO X != 0 FAÇA
25.       EMPILHA X EM P;          // Empilha a partir de B...
26.       X <- VA[X];              // e todos os antecessores até A.
27.    FIMENQUANTO
28.    ENQUANTO P NÃO ESTIVER VAZIA FAÇA
29.       DESEMPILHA X DE P;
30.       MOSTRA X NA TELA;        // Desempilha e mostra em ordem.
31.    FIMENQUANTO
32. SENÃO
33.    ESCREVA "B não é alcançável a partir de A!";
34. FIMSE
```

A implementação desse algortimo é similar ao DFS, exceto em dois aspectos:

1. Usa uma fila (DFS usa uma pilha[1]);
2. Marca o vértice como descoberto **antes** de inserir na fila (no DFS, o vértice é marcado como descoberto somente **depois** de ser removido da pilha).

Entrada

A primeira linha da entrada contém um inteiro *N*, que representa o número de vértices do grafo. A matriz de adjacência terá, portanto, dimensão *N*. Em seguida, são lidas cada uma das *N* linhas da matriz de adjacência. Finalmente, temos como entrada o vértices A (inicial) e B (final).

Saída

A lista de vértices visitados, um em cada linha.

Exemplo

Exemplo de entrada	Exemplo de saída
8 0 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 5 6	5 7 3 6

Critérios de avaliação

- Execução correta e alinhamento com o que foi solicitado neste enunciado;
- Uso apropriado das funções dos *tipos abstratos de dados* (matriz e fila). Respeite o encapsulamento!

Informações importantes

- **Equipe:** 1 ou 2 alunos.
- **Entrega via Moodle.**

[1] A pilha também é usada aqui, mas não no algoritmo de busca em si. Ela é usada quando B é encontrado, pois é preciso percorrer na ordem inversa até A. A pilha vai permitir mostrar o caminho de A até B.