## Lista #7 - Parcial Partição Dinâmica e Grafos

Data de entrega: 20 de junho de 2017

Modifique os arquivos que acompanham este enunciado, a fim de implementar as questões pedidas, e envie-os de volta zipados com nome no padrão <numero\_matricula>.zip por email para profs-eda@tecgraf.puc-rio.br, com o assunto [EDA] Lista 7. Atenção: Crie um arquivo contendo a função main do seu programa para testar suas implementações, mas envie SOMENTE os arquivos e as classes solicitadas.

- 1. A estrutura Union-find (ou Disjoint-set), vista em aula, armazena conjuntos de elementos, divididos em subconjuntos disjuntos. Com a operação de union é possível unir dois conjuntos em um só. Com a operação de find, é possível identificar a qual conjunto pertence um determinado elemento. Utilizando essa estrutura, fica fácil construir um labirinto: a idéia é partir de uma grade 2d de tamanho  $m \times n$ , onde cada célula é inicialmente um subconjunto de um único elemento. A partir daí, grupos vizinhos vão sendo unidos (paredes vão sendo derrubadas) até que reste somente um único grupo, quando então se torna possível ir de um ponto a qualquer outro do labirinto.
  - (a) Implemente uma classe UnionFind na estrutura definida no código abaixo. Os elementos armazenados são implicitamente definidos por índices.

A classe armazena um vetor de inteiros **parent**, que armazena o pai atual de cada elemento. Assim, por exemplo, se parent[2] = 9, significa que o pai do elemento 2 é o elemento 9.

No vetor **size**, deve ser armazenado o tamanho atual de cada conjunto. Assim, size[a] contém o tamanho do conjunto cuja raiz é a. Se a não é raiz, size[a] deve conter zero.

A variável **numSets** armazena a quantidade de conjuntos existentes. Ela é inicializada com o número de elementos inicial, e então decrementada sempre que dois conjuntos são unidos.

```
#ifndef UNIONFIND_H
#define UNIONFIND_H
```

```
#include <vector>
class UnionFind
public:
    //Inicializa a UnionFind com o numero de elementos
    UnionFind(int n);
    // Destrutor
    ~UnionFind();
    //Retorna a raiz do conjunto a que u pertence
    int find(int u);
    //Une os dois conjuntos que contem u e v
    void makeUnion(int u, int v);
    //Retorna o numero de conjuntos diferentes
    int getNumSets();
private:
    //Armazena o pai de cada elemento
    std::vector<int> parent;
    //Armazena o tamanho de cada conjunto
    std::vector<int> size;
    //Quantidade de conjuntos atual
    int numSets;
};
#endif
```

(b) Utilize a classe UnionFind para gerar um labirinto. A estrutura do labirinto pode ser armazenada num vetor de "paredes", conforme mostrado na Figura 1. As paredes da direita e de baixo de todas as células são armazenadas sequencialmente no vetor, contendo valor true quando existem, e false, quando não existem mais. Inicialmente, todas as paredes existem, como na Figura 1.

O algoritmo consiste em manter as células representadas em uma UnionFind, e sucessivamente sortear paredes para serem derrubadas, até que só reste um conjunto na UnionFind. Duas regras no entanto, devem ser respeitadas:

- As paredes de borda não podem ser excluídas
- Uma parede sorteada só pode ser excluída quando suas células vizinhas pertencerem a grupos diferentes.

0 0	1 <sup>2</sup>	2 4	3 6
4 8	5 10	6 <sub>12</sub>	7 14
8 16	9 18 19	10 20	11 22 23



Figure 1: Labirinto inicialmente completo e sua representação em vetor.

No fim, um resultado possível é mostrado na Figura 2. As paredes excluídas contém 0 (false) nas suas respectivas posições, e as bordas foram respeitadas.

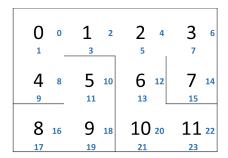




Figure 2: Possível labirinto gerado e sua representação em vetor.

Implemente a função createMaze, com a assinatura especificada abaixo. Esta função recebe como parâmetro de entrada a largura (m) e a altura (n) do labirinto, e também o vetor em que deve armazenar o resultado.

Três outras funções já estão implementadas e podem ser utilizadas:

```
int randomInt(int from, int to);
```

A função randomInt gera um número inteiro aleatório dentro do range especificado (inclusivo).

A função drawMaze desenha o labirinto no terminal. O labirinto da Figura 2 gera a saída:

```
+---+---+---+
|
+ +---+ + +
| | | |
+---+ + +---+
| | |
```

E a função printMaze imprime o labirinto como texto, no formato "célula parede\_direita parede\_inferior". O mesmo labirinto gera a saída:

```
0 0 0

1 0 1

2 0 0

3 1 0

4 0 1

5 1 0

6 1 0

7 1 1

8 0 1

9 1 1

10 0 1

11 1 1
```

2. (A definir) Algoritmos de grafo.