Relatorio Grafos

João Pedro Giandoso

May 14, 2018

Apresentação da resolução da proposta de implementar o algoritmo de busca em largura com base na video aula, além de desenvolver o algoritmo de busca em profundidade de maneira iterativa. Após a implementação dos menos, devemos prepara-lo para ser executado utilizando os parametros da String args[], e construi-lo, sendo executavel totalmente através do arquvio jar referente.

1. Implementação BFS

Após assistir a video aula, foi bem intuitivo a implementação do metodo, na pratica apenas foi necessario traduzir pseudo-codigo corrigindo especificidades de java. A pilha FIFO utilizada foi a já implementada em java, ***java.utils.Queue***

```
public BFS(GrafoAbstrato pGrafo, int s) {
this.grafo = pGrafo;
this.vertices = grafo.getNumVertices();
cor = new Cores[vertices];
d = new int[vertices];
pai = new int[vertices];
for(int u =0; u < this.vertices; u++){
    cor[u] = BRANCO;
    d[u] = Integer.MAX_VALUE;
}
cor[s] = CINZA;
d[s] = 0;
pai[s] = -1;
Q.add(s);
int u;
while (Q.peek() != null) {
    u = Q.remove();
    List<Integer> adjU = grafo.getAdjacentes(u);
    for (int v : adjU) {
        if (cor[v] == BRANCO) {
```

```
                    cor[v] = CINZA;
                    d[v] = d[u] + 1;
                    pai[v] = u;
                    Q.add(v);
                }
            }
            cor[u] = PRETO;
        }
    }
```

2. Implementação DFS Iterativo

Para a implementação do DFS iterativo, a principio devemos preparar nossas estruturas auxiliares, após isso empilhamos e marcamos o tempo de inicialização do primeiro vertice, abrimos um laço que executará até a pilha estar vazia, esse laço tem a responsabilidade de iterar sobre a lista de adjacentes do vertice que esta no topo da pilha, porém sem removê-lo do topo, e caso o adjacente seja branco, ele é empilhado, pintado de cinza, tem seu tempo de inicialização adicionado, após isso, a lista de adjacentes que estamos iterando e alterada para a lista de adjacentes do vertice em questão, até nao ter mais vertices brancos, e o algoritmo com base na pilha consegue voltar distribuindo os tempos de finalização.

```
public DFS_iterativo(GrafoAbstrato pGrafo){
    this.grafo = pGrafo;
    this.vertices = grafo.getNumVertices();
    v = new Cores[vertices];
    d = new int[vertices];
    f = new int[vertices];
    for (int i = 0; i < vertices; i++) {
        v[i] = BRANCO;
    }
    tempo = 0;
    pilha.push(0);
    v[0] = CINZA;
    d[0] = ++tempo;

    while(!pilha.empty()){
        int aux = pilha.peek();
        List<Integer> adj = grafo.getAdjacentes(aux);
        for (int vertice : adj) {
            if(v[vertice] == BRANCO){
                pilha.push(vertice);
                v[vertice] = CINZA;
                d[vertice] = ++tempo;
                adj = grafo.getAdjacentes(vertice);
```

```
                }
            }
            int finalizado = pilha.pop();
            v[finalizado] = PRETO;
            f[finalizado] = ++tempo;
        }
    }
```

3. Implementação parametros

A implementação do mecanismo de execução através de parametros simplesmente executa um laço capturando todos os parametros, e com base nos parametros dados ao programa, é feita a execução.

```
        for (int i = 0; i < args.length; i+=2) {
            switch (args[i]) {
                case "-rep":
                    rep = args[i + 1].toLowerCase();
                    break;
                case "-f":
                    f = args[i + 1];
                    break;
                case "-csvorigem":
                    origem = args[i + 1];
                    break;
                case "-csvdestino":
                    destino = args[i + 1];
                    break;
                case "-verticeinicial":
                    vInicial = Integer.parseInt(args[i + 1]);
                    break;
                case "-grafo":
                    pGrafo = Integer.parseInt(args[i + 1]);
                    break;
            }
        }
        l = new Leitor(origem);
        switch (pGrafo) {
            case 1:
                grafo = l.buildGrafo_MA();
                break;
            case 2:
                grafo = l.buildGrafo_LA();
                break;
        }
        switch (rep) {
```

```
        case "bfs":
            BFS bfs = new BFS(grafo, vInicial);
            bfs.escreveCSV();
            break;
        case "dfs":
            switch (f) {
                case "i":
                    DFS dfs = new DFS(grafo);
                    dfs.escreveCSV(destino);
                    break;
                case "r":
                    DFS_iterativo dfs = new DFS_iterativo(grafo);
                    break;
            }
            break;
    }
```