Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
DCC059 - Teoria dos Grafos Semestre 2016-3

Relatório

Helder Linhares Bertoldo dos Reis

Professor: Stênio Sã Rosário F. Soares

Relatório da parte 1 do trabalho de Teoria dos Grafos, parte integrante da avaliação da disciplina.

Juiz de Fora

Dezembro de 2016

Sumário

1 Introdução

- Este relatório visa explicar sobre os tópicos implementados que foram solicitados na primeira parte do trabalho de Teoria dos Grafos.

2 Descrição do Problema

- Desenvolver um TAD ou uma classe que implemente um conjunto de funcionalidades sobre grafos.

2.1 Estruturas de dados utilizadas

Para o desenvolvimento do algoritmo foram utilizadas as seguintes Estruturas de Dados implementadas em C++ 11:

- Tabela Hash
- Vector
- List
- MultiSet
- Pair

2.2 Estrutura do Grafo

O grafo foi modelado como sendo um conjunto de vértices, sendo que cada vértice tem um conjunto de adjacentes.

2.2.1 Tabela Hash

Conforme foi solicitado o grafo foi implementado utilizando tabela hash. O objetivo dessa implementação é a partir de uma chave simples, fazer uma busca rápida e obter o valor desejado.

A tabela hash dos Vértices recebe como tamanho inicial:

$$\frac{OrdemdoGrafo}{2}$$

A função de espalhamento segue o seguinte critério:

id do vértice % tamanho da tabela hash dos vértices

Também foi implementada tabela hash para os adjacentes. A tabela hash dos Adjacentes recebe como tamanho inicial:

$\frac{NumerodeArestas}{OrdemdoGrafo}$

Tendo como função de espalhamento:

id do Vertice % tamamanho da tabela hash dos adjacentes

3 Principais Algoritmos

Os algoritmos que devem ser destacados por sua maior complexidade são: Busca em Largura, Busca em Profundidade, Algoritmos de Caminho Mínimo de Dijkstra e Floyd, Algoritmos para Árvore Geradora Mínima de Prim e Kruskal.

3.1 Busca em Largura

Algoritmo 1: Busca em Largura

Entrada: Um Grafo G e um nó fonte s

Saída: Uma sequência de vértices alcançados pela busca em largura

```
início
    para cada u ∈ V[G] − {s} faça
        u.cor = BRANCO
        u.d = ∞
        u.π = NIL
    fim
    s.cor = CINZA
    s.d = 0
    Q = ∅
    ENFILEIRA(Q, s)
    Enquanto Q ≠ ∅ faca
        u = DESENFILEIRA(Q)
        para cada v = Adj[u] faça
            if v.cor == BRANCO then
                v.cor == CINZA
                v.d == u.d + 1
                v.π = u
                ENFILEIRA(Q, v)
            end
        fim
        u.cor = PRETO
    fim-enquanto
fim
```

3.2 Busca em Profundidade

Algoritmo 2: Busca em Profundidade

Entrada: Um Grafo G e um nó fonte s

Saída: Uma sequência de vértices alcançados pela busca em profundidade

início
    para *cada* $u \in V[G]$ **faça**
        $u.cor = BRANCO$
        $u.\pi = NIL$
    fim
    $aux - buscaProfundidade(G, s)$
fim

Algoritmo 3: aux-buscaProfundidade

Entrada: Um Grafo G e um nó fonte u

Saída: Uma sequência de vértices alcançados pela busca em profundidade

início
    $u.cor = CINZA$
    para *cada* $v \in G.Adj[u]$ **faça**
        if $v.cor == BRANCO$ **then**
            $v.\pi = u$
            $aux - buscaProfundidade(G, v)$
        end
    fim
    $u.cor = PRETO$
fim

3.3 Algoritmo de Dijkstra

Algoritmo 4: Algoritmo de Dijkstra

Entrada: Um Grafo G com arestas ponderadas, v1 um vértice origem e v2 um vértice destino

Saída: A menor distância entre v1 e v2

```
início
    para i ← 1 ate n faça
        dt[i] ← ∞
        rot[i] ← 0
    fim
    dt[v1] ← 0
    rot[v1] ← ∞
    A ← {N}
    F ← ∅
    Enquanto F ≠ N faca
        r ← j ∈ A, tal que dt[j] é mínimo dentre os elementos de A
        F ← F ∪ {r}
        A ← A − {r}
        V ← V − F
        para i ∈ V faça
            p ← min[dt[i], (dt[r] + dn)]
            if p ≺ dt[i] then
                dt[i] ← p
                rot[i] ← r
            end
        fim
    fim-enquanto
fim
```

3.4 Algoritmo de Floyd

Algoritmo 5: Algoritmo de Floyd

Entrada: Um Grafo G com arestas ponderadas, v1 um vértice origem e v2 um vértice destino

Saída: A menor distância entre v1 e v2

```
início
    MatrizL ← PesosdasArestas
    para k ← 1 ate n faça
        para i ← 1 ate n faça
            para j ← 1 ate n faça
                if Iij > (Iik + Ikj) then
                    Iij ← (Iik + Ikj)
                end
            fim
        fim
    fim
fim
```

3.5 Algoritmo de Prim

Algoritmo 6: Algoritmo de Prim

Entrada: Um Grafo G com arestas ponderadas, um vértice fonte s

Saída: Uma árvore geradora mínima

```
início
    T ← {i}
    V ← N − {i}
    Tmin ← ∅
    Enquanto T ≠ N faca
        Encontrar a aresta (j, k) ∈ M tal que j ∈ T, k ∈ V e djk é mínimo
        T ← T ∪ {k}
        V ← v − {k}
        Tmin ← Tmin ∪ (j, k)
    fim-enquanto
fim
```

3.6 Algoritmo de Kruskal

Algoritmo 7: Algoritmo de Kruskal

Entrada: Um Grafo G com arestas ponderadas, um vértice fonte s

Saída: Uma árvore geradora mínima

início

  Ordene as arestas em ordem crescente de pesos dij no vetor

  $H = [hi], i = 1, 2, ..., m$

  $T \leftarrow h1$

  $V \leftarrow N - \{i\}$

  $i \leftarrow 2$

  Enquanto $T \neq N$ **faca**

    if $T \cup hi$ *é um grafo aciclico* **then**

      $T \leftarrow T \cup hi$

    end

    $i \leftarrow i + 1$

  fim-enquanto

fim

4 Dificuldades Encontradas

A maior complexidade do trabalho talvez tenha sido desenvolver a estrutura do grafo utilizando tabela hash. Tanto de modo a desenvolver a estrutura em si, quanto adaptar os algoritmos para funcionar nela.

Dos algoritmos o que mais trouxe dificuldades foi o algoritmo de Dijkstra.