Teoria dos Grafos e Computabilidade Implementação 2

Fabio Antônio, Lucas Alkmim, Pedro Marçal, Pedro Ribeiro

10 de Setembro, 2024

Conteúdo

1 Introdução

1.1 Objetivo

O objetivo desse codigo é implementar e exibir possivéis aplicações dos algoritmos de Dijkstra, Min-Max e Max-Min em grafos.

2 Algoritmo de Dijkstra

2.1 Classes

A classe utilizada para a implementação do código foi a seguinte:

```
1
2  class Graph {
3
4  public:
5    vector<vector<int>> adj_matrix;
6
7    Graph(int n) {};
8    void add_edge(int u, int v, int p) {};
9    void print() {};
10 };
```

2.2 Função

A função é definida como:

```
1    void Dijkstra(Graph *g, int v) {};
```

recebe um grafo(g) e um vértice v(vértice de início) e calcula o menor caminho do vértice v para todos os outros vértices.

2.3 Criação de variavéis

```
1    int n = g->adj_matrix.size();
2    vector<int> dist_array(n, INT_MAX);
3    vector<bool> visited(n, false);
4    vector<int> prev(n, -1);
```

Nesse trecho inicializamos o vetor de distâncias com tamanho n e valor $\infty+$, o vetor booleano de verificação de visita a um vértice com tamanho n e valor false, o vetor de antecessores com tamanho n e valor -1 como também a fila de prioridade, definida como:

```
priority_queue<pair<int, int>> , vector<pair<int, int>>, greater<
  pair<int, int>>> pq;
```

Em que o tipo de dados será um par de inteiros(valor do vértice e posição do vértice na matriz de adjacência). Armazenados em um vetor de par de inteiros e sendo uma fila de prioridade mínima.

2.4 Execução

Respeitando o seguintes passos:

- Até que a fila de prioridade esteja vazia

```
while (pq.empty() == false) {

```

- Escolher o vértice mais "barato" dentre os vizinhos do vértice atual e remover vértice escolhido

```
    int path_vertex = pq.top().second;
    pq.pop();

```

- Atualizar o custo dos vizinhos desse vértice e atualizar na fila de prioridade

```

    for (int i = 0; i < n; i++) {
      if (g->adj_matrix[path_vertex][i] != 0 && visited[i] ==
    false) {
        int min_dist = dist_array[path_vertex] + g->adj_matrix
    [path_vertex][i];
        if (dist_array[i] > min_dist) {
          dist_array[i] = min_dist;
          prev[i] = path_vertex;
          pq.push({min_dist, i});
        }
      }

```

- Imprimir na tela os resultados

```
1
2   for (int i = 0; i < n; i++) {
3     cout << "Distancia ate o no " << i << " = " << dist_array[
      i] << endl;
4
5     cout << "Caminho: ";
6     cout << endl;
7
```

- e o caminho feito por cada um

```
1 void print_path(const vector<int> &prev, int v) {
2   if (prev[v] == -1) {
3     cout << v;
4     return;
5   }
6   print_path(prev, prev[v]);
7   cout << " -> " << v;
8 }
9
```

3 Aplicações