Centro Universitário da FEI

Programa de Mestrado e Doutorado em Engenharia Elétrica

PEL 201 – Algoritmos Computacionais

**Relatório 3**

**Algoritmos de grafos: PRIM e Dijkstra**

Claudio Aparecido Borges Junior

Matrícula: 120122-7

**PRIM**

O algoritmo de PRIM tem como finalidade buscar uma árvore geradora mínima em um grafo conectado, não orientado e ponderado. Uma árvore geradora mínima de um grafo é um subconjunto do grafo onde soma dos valores de todas as arestas desse subconjunto é minimizado e todas as vértices estão conectadas. Ele é um algoritmo guloso.

O algoritmo de PRIM é inicializado recebendo a representação de um grafo e um vértice de origem; então é criado um conjunto contendo todos os vértices do grafo e atribuído o valor infinito para cada vértice desse conjunto. Enquanto o conjunto de vértices contiver algum vértice, o vértice com menor valor é extraído (v), e para cada aresta (a) é verificado se ela atinge um vértice válido (v’), ou seja, um vértice ainda não processado. Se o vértice (v’) for valido, é verificado se a aresta (a) possui um valor menor do que o valor do vértice (v’), se sim, o valor do vértice (v’) é alterado e passa a ter o vértice original (v) como pai.

Abaixo é apresentado a implementação algoritmo de PRIM utilizando *Python 2.7.* Foi utilizado um fila *heap* para armazenar o vetor com menor valor.

1 INF = 999

2

3 def mst\_prim(adj\_l, root):

4 n = len(adj\_l)

5

6 parents = [None for i in range(n)]

7 not\_visited = set(range(n))

8

9 h\_weights = [(INF, i) for i in range(n)]

10 heapq.heapify(h\_weights)

11

12 heapq.heappush(h\_weights, (0, root))

13 while not\_visited:

14 w\_min, v\_min = heapq.heappop(h\_weights)

15 if v\_min not in not\_visited:

16 continue

17

18 for v, w in adj\_l[v\_min]:

19 if v in not\_visited and w < w\_min:

20 heapq.heappush(h\_weights, (w, v))

21 parents[v] = v\_min

22 not\_visited.discard(v\_min)

23

24 return parents

**DIJKSTRA**

O algoritmo de Dijkstra tem como finalidade buscar o menor caminho entre duas vértices em um grafo conectado, orientado ou não e com pesos positivos. Ele é um algoritmo guloso.

O algoritmo de Dijkstra é inicializado recebendo a representação de um grafo e um vértice de origem e destino; então é criado um conjunto contendo todos os vértices do grafo e atribuído o valor infinito para cada vértice desse conjunto. Enquanto o conjunto de vértices contiver algum vértice, o vértice com menor valor é extraído (v), e para cada aresta (a) é verificado se ela atinge um vértice válido (v’), ou seja, um vértice ainda não processado. Se o vértice (v’) for valido é feito o relaxamento de cada aresta (a). O relaxamento é o processo de estimar a menor distância de um vértice a partir da origem, ou seja, verificar se o valor acumulado do vértice (v’) é menor do que a aresta (a) e o vértice acumulado original (v), se sim, o valor do vértice acumulado (v’) é alterado e passa a ter o vértice original (v) como pai.

Abaixo é apresentado a implementação algoritmo de Dijkstra utilizando *Python 2.7.* Foi utilizado uma fila *heap* e também um vetor auxiliar *weights* representando o relaxamento.

1 def dijkstra(adj\_l, src):

2 n = len(adj\_l)

3

4 parents = [None for i in range(n)]

5 weights = [INF for i in range(n)]

6 not\_visited = set(range(n))

7

8 weights = [INF for i in range(n)]

9 h\_weights = [(INF, i) for i in range(n)]

10 heapq.heapify(h\_weights)

11

12 weights[src] = 0

13 heapq.heappush(h\_weights, (0, src))

14 while not\_visited:

15 w\_min, v\_min = heapq.heappop(h\_weights)

16 if v\_min not in not\_visited:

17 continue

18

19 for v, w in adj\_l[v\_min]:

20 if v in not\_visited and w + w\_min < weights[v]:

21 heapq.heappush(h\_weights, (w + w\_min, v))

22 weights[v] = w + w\_min

23 parents[v] = v\_min

24 not\_visited.discard(v\_min)

25

26 return parentes

**COMPARATIVO ENTRE PRIM e DIJKSTRA**

Ambos algoritmos são semelhantes em sua implementação, porém diferentes em seu propósito. O algoritmo de PRIM busca encontrar a árvore geradora mínima, enquanto o Dijkstra busca encontrar a menor distância entre duas vértices. Ambos algoritmos utilizam uma abordagem gulosa.

Considerando sua implementação, o relaxamento é a principal diferença entre eles. O algoritmo de PRIM utiliza somente o valor da aresta para decidir qual o valor do vértice, enquanto o Dijkstra utiliza o relaxamento, isto é, sua decisão é com base na distância percorrida desde a origem até um dado vértice. Para o relaxamento é necessário um vetor auxiliar contendo a menor distância de uma dada vértice até a origem.

**COMPARATIVO DE DESEMPENHO DE TEMPO**

Os testes foram feitos com grafos de 1000 vértices variando a quantidade de arestas por vértice. O primeiro ponto no gráfico é referente a 1 aresta por vértice, seguindo por 199 arestas até 999. Um grafo em sua densidade máxima é o que possui 999 arestas quando há 1000 vértices, ou seja, todos os vértices estão a ligados a todos os outros vértices. Um grafo onde 1 vértice é ligada a apenas uma outra vértice é considerado esparso.

|  |  |  |
| --- | --- | --- |
| **Arestas em casa Vértice** | **Prim** | **Dijkstra** |
| 1 (0,01%) | 0,003439581 | 0,001953363 |
| 199 (19,92%) | 0,070094667 | 0,06342449 |
| 399 (39,94%) | 0,153057893 | 0,112835012 |
| 599 (59,96%) | 0,236000522 | 0,173570339 |
| 799 (79,98%) | 0,323083186 | 0,265071172 |
| 999 (100,00%) | 0,423409603 | 0,330884427 |