IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Programação Avançada

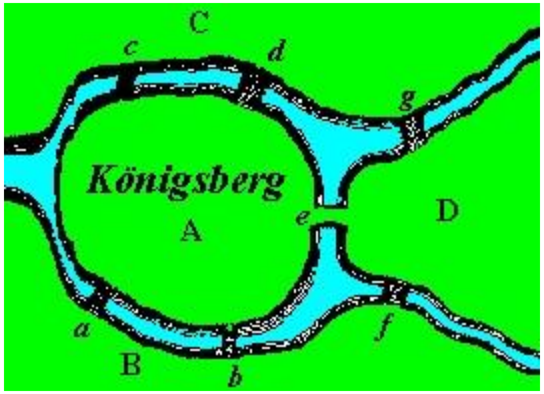
Implementação do TAD –Graph
Programação Avançada – 2020-21

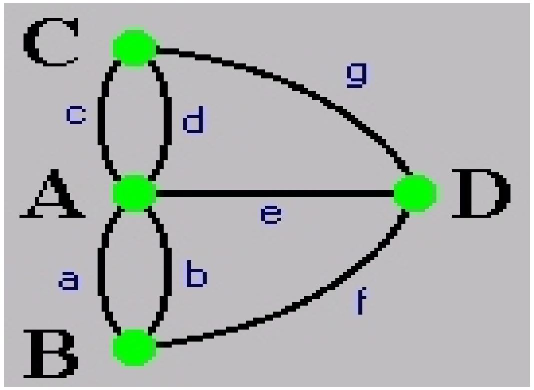
Bruno Silva, Patrícia Macedo

Sumário

- Noção de Caminho Mais Curto ou de Menor Custo
- Algoritmo Dijsktra
- Algoritmo do Caminho mais Curto
- Implementação dos Algoritmos no TAD Graph

Procura do Caminho mais curto





Qual o caminho mais curto para ir de A-D ?

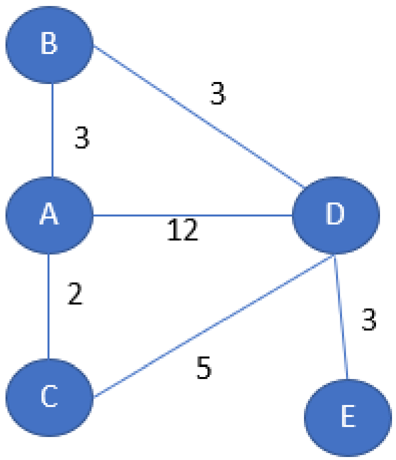
Se as pontes não tiverem uma distância, ou custo associada, diremos que o caminho mais curto, é ir de A a D pela ponte *e*.

Se as pontes tiverem portagem.

- *a* ,*d*, *g*, *f* - 2 Euros
- *c*, *b* – 3 euros
- *e* – 10 euros

Neste caso o caminho de menor custo é seguir pelas pontes *d*, *g* para chegar de A a D.

Procura do Caminho mais curto



- Qual o caminho de menor custo entre A e E?

- Qual o caminho de menor curso (que percorre um menor numero de arestas), entre A e E ?

Procura do Caminho mais curto /menor custo

1. Qual o caminho mais curto (ou de menor valor) entre dois vértices de um graph?

   - O caminho mais curto entre o ponto A e o ponto B, é aquele em que a soma do valor das arestas percorridas é menor.

2. Como se determina o valor de uma aresta ?

   - Caso estejamos perante um graph valorado usa-se o valor de associado a cada aresta para determinar o caminho de menor valor entre dois pontos.
   - Nos restantes casos, considera-se que cada aresta percorrida tem o valor 1. O caminho mais curto é aquele que percorrer menos arestas.

Algoritmo Dijkstra

- **Input:**
  - Um graph valorado G = (V, E), w : E -> R (cada aresta tem um valor associado / peso)
  - um nó de partida s.
- **Output:**
  - <u>Caminho mais curto de s para todos os outros nós do graph.</u>

O custo (ou distância, ou peso) de um caminho é igual ao somatório do custo das arestas que constituem o caminho, ou infinito se não existir caminho.

Algoritmo Dijkstra – Descrição informal

1. É atribuída uma distância para todos os pares de vértices. Todos os vértices são inicializados com uma distância infinita, menos para o vértice de origin, que é inicializado a zero.

2. Marque todos os vértices como **não visitados** e defina o vértice inicial como vértice corrente.

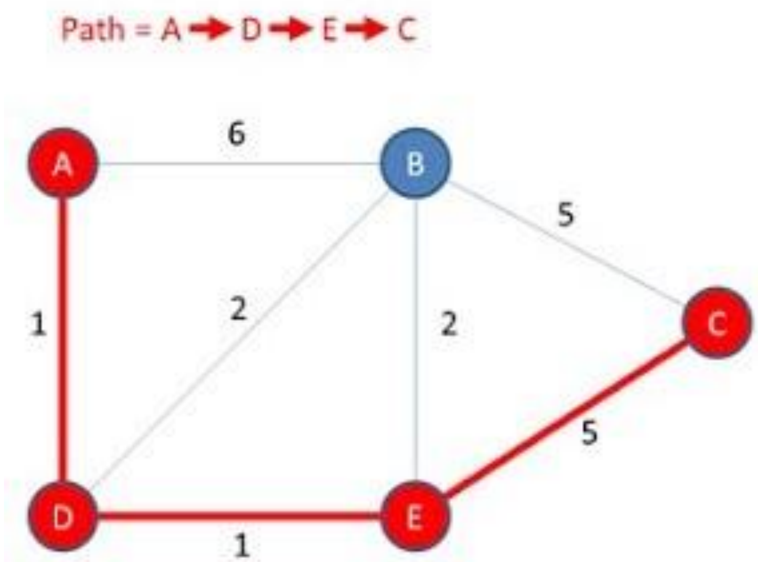
3. Para este vértice corrente, considere todos os seus vértices vizinhos não visitados e calcule a distância a partir do vértice. Se a distância for menor do que a definida anteriormente, substitua a distância pela nova distância calculada.

4. Quando todos os nós adjacentes do vértice corrente forem visitados, marque-o como visitado, o que fará com que ele não seja mais analisado (sua distância é mínima e final).

5. Selecione para vértice corrente o vértice não visitado com a menor distância (a partir do vértice inicial) e continue a partir do passo 3**, até todos os nós já terem sido visitados**.

Ver o Algoritmo em funcionamento

- https://www.youtube.com/watch?v=pVfj6mxhdMw



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

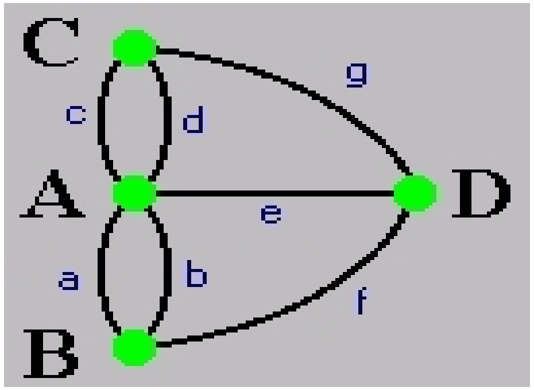
Algoritmo Dijkstra – Pseudocódigo

```
Dijkstra
 Input - (graph,origin)
 Output – costs[] e predecessors[]
 BEGIN
   FOR EACH VERTEX v in graph
        costs[v] <- Infinit
        predecessor[v] <- -1
   END_FOR
   costs[origin] <- 0
   S <- {all vértices of graph}
   WHILE (S IS NOT EMPTY) DO:
          u <- findLowerVertex(graph, costs[])  é o vértice do graph com menor custo
          IF (costs[u] = Infinit) RETURN
             remove(S,u)
          FOR EACH adjacent vertex v of vertex u DO:
             cost <- costs[u] + cost between u and v;
             IF (cost < costs[v]) THEN
                  costs[v] <- cost;
                  predecessor[v] <- u;
             END_IF
          END_FOR
    END_WHILE
END Disjkstra
```

Implementação do algoritmo Dijkstra

Considerações Iniciais

- Consideramos que temos a situação do problema das Pontes de Konigsberg.
- Os vértices são instanciados pela classe Local
- As arestas são instanciadas pela classe Bridge (representa as pontes)




```java
public class Local {
    private String name;

    @Override
    public String toString() {
        return "Local{" +
                name +
                '}';
    }

    public Local(String name) {
        this.name = name;
    }
}
```

```java
public class Bridge {
    private String name;
    private int cost;

    public Bridge(String name, int cost) {
        this.name = name;
        this.cost = cost;
    }

    public String getName() {
        return name;
    }

    public int getCost() {
        return cost;
    }
}
```

Implementação do algoritmo Dijkstra

```
private void dijkstra(Vertex<Local> orig,
                      Map<Vertex<Local>, Double> costs,
                      Map<Vertex<Local>, Vertex<Local>> predecessors) {

    costs.clear();
    predecessors.clear();
    List<Vertex<Local>> unvisited = new ArrayList<>();

    for (Vertex<Local> v : graph.vertices()) {
        costs.put(v, Double.POSITIVE_INFINITY);
        predecessors.put(v, null);

        unvisited.add(v);
    }
    costs.put(orig, 0.0);
    while(!unvisited.isEmpty()) {
        Vertex<Local> lowerCostVertex = findLowerCostVertex(unvisited, costs);
        unvisited.remove(lowerCostVertex);
        for (Edge<Bridge, Local> incidentEdge : graph.incidentEdges(lowerCostVertex)) {
            Vertex<Local> opposite = graph.opposite(lowerCostVertex, incidentEdge);
            if( unvisited.contains(opposite) ) {
                double cost = incidentEdge.element().getCost();
                double pathCost = costs.get(lowerCostVertex) + cost;
                if( pathCost < costs.get(opposite) ) {
                    costs.put(opposite, pathCost);
                    predecessors.put(opposite, lowerCostVertex);
                }
            }
        }
    }

}
```

- **input: local de** orig variáveis de entrada
- **outputs:** costs e predecessor vão ser o resultado da execução do algoritmo .Usamos **Map** em vez de vetores, pos os vértices não são numerados, mas identificados pelo seu rotulo

Implementação do algoritmo calcula caminho de menor custo

- O algoritmo do cálculo do caminho de menor custo, usa o algoritmo Dijkstra.
- Assim a execução do algoritmo de Dijkstra é um **passo** do algoritmo para cálculo de caminho de menor custo e do seu valor.

```
Minumum_Cost_Path

 Input - (graph,origin,destination)
 output – paths[] e cost

  vOri <- pesquisa_vertice(graph,origin)
  vDst <- pesquisa_vertice(graph,destination)
  Disjktra(graph,vOri, costs[], predecessors[])
  path<-[]
  v <- vDst

  WHILE v ≠ vOri DO
         path<-insert(path, 0,v)
         v<-predecessors[v]
  END_WHILE
  path<-insert(path, 0,v)
  cost<-costs[vDst]
END Minumum_Cost_Path
```

costs[] e **predecessor[]** vão ser o resultado da execução do algoritmo Dijkstra

Implementação do algoritmo calcula caminho de menor custo

```java
public int minimumCostPath(String origName, String dstName, List<Local> path) {

    Vertex<Local> vOrig = findLocal(origName);
    Vertex<Local> vDst= findLocal(dstName);

    if( vOrig==null) throw new IllegalArgumentException("orig does not exist");
    if(vDst==null) throw new IllegalArgumentException("dst does not exist");
    if(path == null ) throw new IllegalArgumentException("path list reference is null");

    Map<Vertex<Local>, Double> costs = new HashMap<>();
    Map<Vertex<Local>, Vertex<Local>> predecessors = new HashMap<>();

    dijkstra(vOrig, costs, predecessors);

    path.clear();

    boolean complete = true;
    Vertex<Local> actual = vDst;
    while( actual != vOrig) {
        path.add(0, actual.element());
        actual = predecessors.get(actual);
        if( actual == null) {
            complete = false;
            break;
        }
    }
    path.add(0, vOrig.element());
    if(!complete) {
        path.clear();
        return -1;
    }
    return costs.get(vDst).intValue();
}
```

ADT Graph | Exercícios de implementação 💻

Continuando com o código iniciado na aula anterior sobre implementação do ADT Grafo.
Faça download do package model
https://github.com/pa-estsetubal-ips-pt/Model_Bridges.git
e integre-o no projeto do ADT Graph que tem vindo a trabalhar.

1. Analise a classe Brige Manager e execute o main de teste disponibilizado – MainTest.
2. Presentemente o método do caminho mais curto indica-nos apenas os locais que o constituem. No caso de haverem pontes paralelas (arestas paralelas) , é importante saber quais as pontes que compõe o caminho

Altere os métodos dijkstra e minimumCostPath de forma a este último devolver também a listas das pontes (Edges) que compõe o caminho de menor custo.

Se as pontes tiverem portagem.

- a ,d, g, f - 2 Euros
- c, b – 3 euros
- e – 10 euros

Neste caso o caminho de menor custo é seguir pelas pontes d, g para chegar de A a D.