Algoritmo de dijkstra

algoritmo de dijkstra, que determina caminho mínimo até X para grafos ponderados

Como funciona

1 - Encontre o vértice mas "baratos" que você consegue achar com o menor custo possível
2 - Verifique se há um caminho mais barato para os vértices vizinhos, caso exista, atualize o custo
3 - Repita o processo até que todos os vértices tenham sido visitados
4 - Calcule o caminho final

Terminologia

- Peso: Custo associado a uma aresta
- Grafo ponderado: Grafo com pesos associados a arestas
- Grafo não ponderado: Grafo sem pesos associados a arestas
- Grafo ciclico: Grafo que possui um ciclo
- Graficos direcionados: Grafo que possui direção
- Grafos não direcionados: Grafo que não possui direção
- O algoritmo de dijkstra é utilizado para encontrar o caminho mais curto em grafos ponderados ou seja so funcionar para grafos ponderados e direcionados, a partir de um vértice de origem até um vértice de destino

Arestas com pesos negativos

O algoritmo de dijkstra não funciona com arestas com pesos negativos, pois ele não consegue determinar o caminho mais curto, pois ele sempre irá escolher o caminho com o menor custo, e se houver um caminho com custo negativo, ele irá escolher esse caminho, mesmo que exista um caminho mais curto com custo positivo

PROF

Implementação

```go
package main

import (
    "fmt"
    "math"
)

// Definição do grafo
var grafo = map[string]map[string]int{
    "A": {"B": 2, "C": 4},
    "B": {"C": 1, "D": 7},
    "C": {"D": 3, "E": 5},
    "D": {"E": 1, "F": 4},
    "E": {"F": 2},
```

```
    "F": {},
}

// Inicialização das distâncias
var distancia = map[string]int{
    "A": 0,
    "B": math.MaxInt64,
    "C": math.MaxInt64,
    "D": math.MaxInt64,
    "E": math.MaxInt64,
    "F": math.MaxInt64,
}

var visitados = make(map[string]bool)

func menor_distancia(nome string) {
    visitados[nome] = true
    for vizinho, peso := range grafo[nome] {
        if distancia[vizinho] > distancia[nome]+peso {
            distancia[vizinho] = distancia[nome] + peso
        }
    }
}

func main() {
    for len(visitados) < len(grafo) {
        // Encontrar o vértice com a menor distância que ainda não foi
visitado
        menor := math.MaxInt64
        vertice := ""

        for k, v := range distancia {
            if !visitados[k] && v < menor {
                menor = v
                vertice = k
            }
        }

        // Se nenhum vértice foi encontrado, significa que o restante é
inacessível
        if vertice == "" {
            break
        }

        menor_distancia(vertice)
    }

    // Exibir distâncias mínimas
    for k, v := range distancia {
        fmt.Printf("Distância de A até %s: %d\n", k, v)
    }
}
```