UNIVERSITETI I PRISHTINES FAKULTETI I SHKENCAVE MATEMATIKO-NATYRORE

Tema: Algoritmi i Dijkstra's

Lënda: Inteligjenca Artificiale

Punoi: Florian Shabani, Jon Sadikaj, Syard Dauti

Përmbajtja:

Abstrakti: Algoritmi i Dijkstrës (Dÿkstra) është algoritëm i gjetjes së rrugës më të shkurtër mes një nyje dhe pjesës tjetër të grafit (SSSP Single Source Shortest Path), në grafe me masë jo negative të brinjëve. Thjeshtësia dhe shpjejtësia e algoritmës kanë bërë që ky algoritëm që nga koha e zbulimit të gjejë përdorime të shumta në fusha të ndryshme të shkencës kompjuterike. Në këtë punim, Ne analizojmë dhe përshkruajmë cka e dallon algoritmin e Dijkstra's.

1. Historiku i zhvillimit te Algoritmit te Dijkstras

Algoritmi i Dijkstras është një algoritëm për gjetjen e shtigjeve më të shkurtra midis nyjeve në një grafik të peshuar. Ai u zhvillua nga shkencëtari holandez i kompjuterave Edsger ë. Dijkstra në vitin 1956 dhe konsiderohet të jetë një nga algoritmet themelore në shkencën kompjuterike.

Algoritmi fillimisht u krijua për të zgjidhur problemin e shtegut më të shkurtër me një burim të vetëm, i cili përfshin gjetjen e shtegut më të shkurtër nga një nyje e vetme burimore në të gjitha nyjet e tjera në grafik. Punimi origjinal i Dijkstra's mbi algoritmin u titullua "Një shënim mbi dy probleme në lidhje me grafikët" dhe u botua në revistën Numerische Mathematik në vitin 1959.

Algoritmi i Dijkstra's përdor një radhë prioritare për të mbajtur gjurmët e nyjeve të ardhshme për t'u vizituar dhe distancat në ato nyje. Në çdo hap, ai zgjedh nyjen me distancën më të ulët nga nyja burimore dhe përditëson distancat me fqinjët e saj. Algoritmi vazhdon derisa të ketë vizituar të gjitha nyjet ose derisa të arrihet nyja e destinacionit.

Me kalimin e kohës, algoritmi i Dijkstra's është përmirësuar dhe zgjeruar në mënyra të ndryshme. Për shembull, algoritmi tani mund të trajtojë grafikët me pesha negative duke përdorur algoritmin Bellman-Ford për të zbuluar ciklet negative. Për më tepër, variacionet e algoritmit janë zhvilluar për të zgjidhur lloje të tjera të problemeve të rrugës më të shkurtër, siç është problemi i rrugës më të shkurtër me të gjitha çiftet.

Në përgjithësi, algoritmi i Dijkstra's ka pasur një ndikim të rëndësishëm në shkencën kompjuterike dhe është përdorur në një gamë të gjerë aplikacionesh, nga rutimi në rrjetet kompjuterike deri tek gjetja e shtigjeve në video lojëra.

2. Implementimi konkret i algoritmit te Dijkstras

Implementimi i algoritmit te Dijkstra's bëhet ashtu që fillimisht në grafin e peshuar nyjes fillestare i caktohet kostoja zero kurse të gjithë nyjeve tjera ju caktohet kostoja infinit. Figura 1 paraqet hapin fillestar që merret në implementimin e algoritmit të Dijkstra's, ku si nyje fillestare perdoret nyja A. Nëse me S marrim bashkësine e kulmeve ku distanca minimale nga nyja fillestare është llogaritur, fillimisht kemi se S eshte boshe, për arsye se nuk ka ndonje distance te llogaritur.

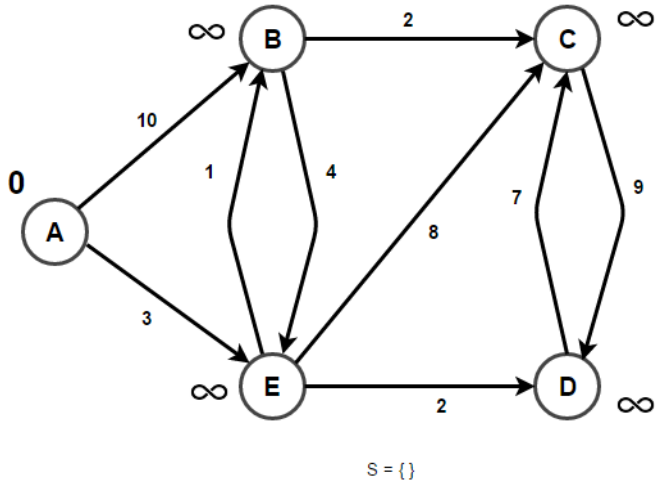

Fig 1.Inicializimi i nyjeve

Pas kësaj, llogaritet distanca ndërmjet nyjes fillestare, në këtë rast nyjes A dhe kulmeve fqinje të A-se dhe nëse kostoja e nyjeve të llogaritura është më e vogel, atëherë kostoja e atyre nyjeve rivendoset në kosto minimale. Figura 2, paraqet kulmet fqinje të A me kostot e tyre minimale. Meqë A është eksploruar ajo vendoset tek bashkesia S.

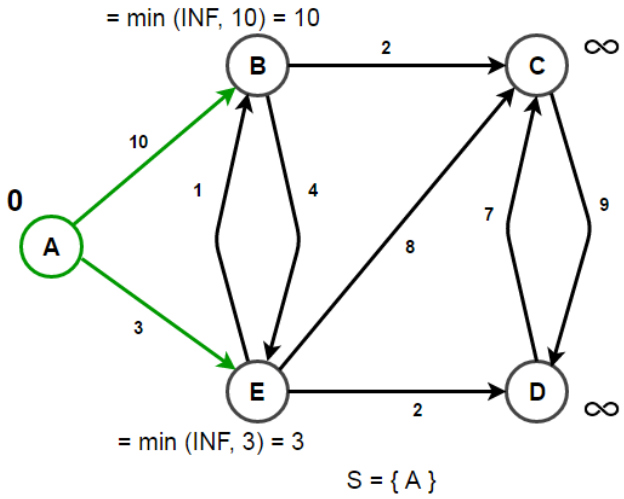

Fig 2. Llogaritja e kostove te nyjeve fqinje te A-se

Pas përpunimit të të gjitha skajeve të A-se ne kalojmë në eksplorimin e nyjes e cila ka koston minimale. Në këtë rast meqë E ka koston minimale gjejme koston e kësaj nyje me nyjet e lidhura me të. Si shihet nga figura 2 fqinjët e nyjes E janë B, C dhe D, dhe me llogaritjen e kostove ndërmjet këtyre nyjeve, gjejmë se kostoja për të shkuar deri tek nyja B nga nyja E është 4, dhe meqë kostoja momentale e nyjes B është 10 dhe kjo nuk është kosto minimale, atëherë rikthejmë koston e B në 4, kurse kostoja e nyjes C dhe D meqë kanë kosto të pakufishme vendoset në 8 dhe në 2 perkatësisht. Nyja E i shtohet bashkësise S si nyje e eksploruar. Figura 3, paraqet eksplorimin e nyjes E.

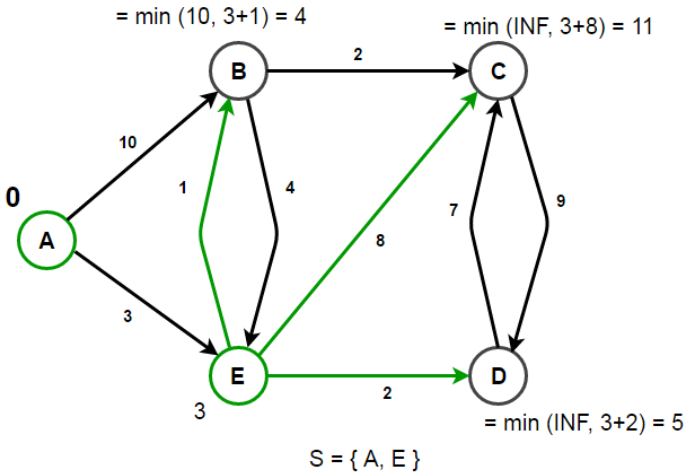

Fig 3.Eksplorimi i nyjes me koston më të vogël pas nyjes A

Algoritmi vazhdon kështu deri në eksplorimin e të gjitha nyjeve. Figura 4 paraqet eksplorimin e të gjithë grafikut ku bashkesia S ka radhitjen: A, E, B, D, C.

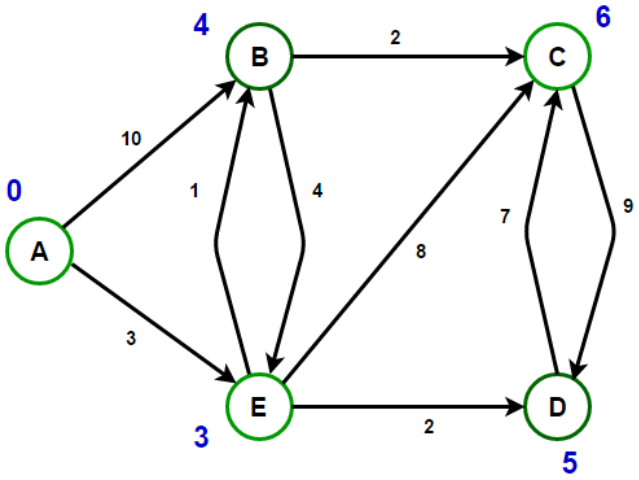

Fig 4. Eksplorimi i gjithe grafikut dhe caktimi i kostove të nyjeve

3. Pseudokodi i algoritmit të Dijkstra's

```
1:   function Dijkstra(Graph, source):
2:       for each vertex v in Graph:                  // Initialization
3:           dist[v] := infinity                      // initial distance from source to vertex v is set to infinite
4:           previous[v] := undefined                 // Previous node in optimal path from source
5:       dist[source] := 0                            // Distance from source to source
6:       Q := the set of all nodes in Graph           // all nodes in the graph are unoptimized - thus are in Q
7:       ëhile Q is not empty:                        // main loop
8:           u := node in Q ëith smallest dist[ ]
9:           remove u from Q
10:          for each neighbor v of u:                // ëhere v has not yet been removed from Q.
11:              alt := dist[u] + dist_betëeen(u, v)
12:              if alt < dist[v]                     // Relax (u,v)
13:                  dist[v] := alt
```

```
14:                    previous[v] := u
15:          return previous[ ]
```

4. Vlerësimi i algoritmit te Dijkstras

Restrictions		SSSP Algorithm	
Graph	Weights	Name	Running Time $O(\cdot)$
General	Unweighted	BFS	$\|V\| + \|E\|$
DAG	Any	DAG Relaxation	$\|V\| + \|E\|$
General	Non-negative	Dijkstra	$\|V\| \log \|V\| + \|E\|$
General	Any	Bellman-Ford	$\|V\| \cdot \|E\|$

Algoritmi i Dijkstra's në dallim me algoritmet e tjera:

Në krahasim me algoritmet e tjera të kërkimit, siç është për shembull DFS dhe BFS, algoritmi i Dijkstras është më i specializuar dhe i përshtatshëm për kërkimin e rrugës më të shkurtër në një graf me peshë. Për dallim nga DFS dhe BFS, ai ndjek një strategji të shumësive minimale, duke qenë i orientuar drejt gjetjes së rrugës më të shkurtër në mes të dy pikave.

Në krahasim me algoritmin e Bellman-Ford, algoritmi i Dijkstras është më efikas dhe më shpejtë për grafe me peshë të vogël dhe të mesme, pasi që nuk ndjek një strategji ekuilibrimi dhe nuk llogaritë rrethime të vazhdueshme të gjithë grafikun.

Megjithatë, në krahasim me algoritmin e Bellman-Ford, algoritmi i Dijkstras ka disa kufizime: nuk mund të përdoret për grafe me peshë negative dhe nuk do të jep zgjidhje optimale në këto raste. Gjithashtu, në krahasim me algoritmin e A* (prononcuar "A-star"), algoritmi i Dijkstras nuk përdor heuristikë, që mund të jetë më efikase në disa raste.

Priority Queue Q' on n items	Q Operations $O(\cdot)$			Dijkstra $O(\cdot)$ $n = \|V\| = O(\|E\|)$
	`build(X)`	`delete_min()`	`decrease_key(id, k)`	
Array	n	n	1	$\|V\|^2$
Binary Heap	n	$\log n_{(a)}$	$\log n$	$\|E\| \log \|V\|$
Fibonacci Heap	n	$\log n_{(a)}$	$1_{(a)}$	$\|E\| + \|V\| \log \|V\|$

Kompleksiteti i Dijkstra's dallon nga lloji i strukturës të së dhënave që përdoret për ta implementuar.

Implementimi me array jep kompleksitet O(V2), ku në grafet e dendura është kompleksitet ideal. Ndërsa implementimi me Binary heap jep një kompleksitet O(|E|log|V|), kompleksiteti ideal për grafet e rralla.

Por, për të marrë më të mirën e të dy rasteve, është zbuluar një strukturë e të dhënav specifikisht për algoritmit e Dijkstra's, që quhet Fibonacci Heap. Me këtë struktur të të dhënav, kompleksiteti i algoritmit mbërrin O(|E| + |V|log|V|). Mirëpo kjo strukturë është e komplikuar dhe jep konstante të mëdha, gjë që e bën jo të përshtatshme për implementim, sidomos në programe me problem klasën e grafeve të vogla.

5. Implementimi në Java i algoritmit të Dijkstras

Dijkstra.java > ShortestPath

```java
// A Java program for Dijkstra's single source shortest path
// algorithm. The program is for adjacency matrix
// representation of the graph
import java.io.*;
import java.lang.*;
import java.util.*;

class ShortestPath {
    // A utility function to find the vertex with minimum
    // distance value, from the set of vertices not yet
    // included in shortest path tree
    static final int V = 9;
    int minDistance(int dist[], Boolean sptSet[])
    {
        // Initialize min value
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < V; v++)
            if (sptSet[v] == false && dist[v] <= min) {
                min = dist[v];
                min_index = v;
            }

        return min_index;
    }

    // A utility function to print the constructed distance
    // array
    void printSolution(int dist[])
    {
        System.out.println(
            "Vertex \t\t Distance from Source");
        for (int i = 0; i < V; i++)
            System.out.println(i + " \t\t " + dist[i]);
    }

    // Function that implements Dijkstra's single source
    // shortest path algorithm for a graph represented using
    // adjacency matrix representation
    void dijkstra(int graph[][], int src)
    {
        int dist[] = new int[V]; // The output array.
                                 // dist[i] will hold
        // the shortest distance from src to i

        // sptSet[i] will true if vertex i is included in
        // shortest path tree or shortest distance from src
        // to i is finalized
        Boolean sptSet[] = new Boolean[V];

        // Initialize all distances as INFINITE and stpSet[]
        // as false
        for (int i = 0; i < V; i++) {
            dist[i] = Integer.MAX_VALUE;
            sptSet[i] = false;
        }

        // Distance of source vertex from itself is always 0
        dist[src] = 0;

        // Find shortest path for all vertices
        for (int count = 0; count < V - 1; count++) {
            // Pick the minimum distance vertex from the set
            // of vertices not yet processed. u is always
            // equal to src in first iteration.
            int u = minDistance(dist, sptSet);
```

```
67
68              // Mark the picked vertex as processed
69              sptSet[u] = true;
70
71              // Update dist value of the adjacent vertices of
72              // the picked vertex.
73              for (int v = 0; v < V; v++)
74
75                  // Update dist[v] only if is not in sptSet,
76                  // there is an edge from u to v, and total
77                  // weight of path from src to v through u is
78                  // smaller than current value of dist[v]
79                  if (!sptSet[v] && graph[u][v] != 0
80                      && dist[u] != Integer.MAX_VALUE
81                      && dist[u] + graph[u][v] < dist[v])
82                      dist[v] = dist[u] + graph[u][v];
83          }
84
85          // print the constructed distance array
86          printSolution(dist);
87      }
```

6. Konkluzione

Si përfundim, algoritmi i Dijkstra's është një algoritëm themelor dhe i përdorur gjerësisht për gjetjen e rrugëve që ka pasur një ndikim të rëndësishëm në shkencën kompjuterike dhe në shumë aplikime praktike. Aftësia e tij për të gjetur shtegun më të shkurtër në një grafik të peshuar e ka bërë atë një mjet kyç në problemet e rrugëtimit dhe optimizimit. Për më tepër algoritmi është zgjeruar dhe përmirësuar në mënyra të ndryshme për të trajtuar skenarë më kompleks. Megjithëse u zhvillua fillimisht mbi 60 vjet më parë, algoritmi i Dijkstra mbetet një komponent jetik i informatikës moderne dhe ndikimi i tij ka të ngjarë të vazhdojë të ndihet për shumë vite në vijim.