UNIVERSITETI I PRISHTINËS "HASAN PRISHTINA"
FAKULTETI I INXHINIERISË ELEKTRIKE DHE KOMPJUTERIKE
DEPARTAMENTI: INXHINIERI KOMPJUTERIKE

Lënda: Dizajni dhe Analiza e Algoritmeve

Tema: Algoritmi i Dijkstra's

Studentët: Gresa Salihu
Leonita Nika
Mihrije Kadriu

Mentorë: Prof.Ass.Dr. Avni Rexhepi
Msc. Dardan Shabani

Prishtinë, Maj 2021

Përmbajtja

1. Hyrje 3
2. Grafet 3
2.1 Zbatimet e grafeve 4
3. Algoritmi i Dijkstra's 4
3.1 Kompleksiteti kohor 11
3.1.1 Cka është kompleksiteti i algoritmit? 11
3.1.2 Kompleksiteti kohor i algoritmit Dijkstra 11
3.2 Krahasimet e Algoritmit te Dijkstra's me algoritmet tjera 13
3.2.1 Krahasimi i Algoritmit Dijkstra's me algoritmin Bellman- Ford 13
3.2.2 Krahasimi i Algoritmit Dijkstra's me algoritmin Dijkstra-Prim 13
3.3 Pseudokodi 13
3.4 BllokDiagrami 15
3.5 Disavantazhi i Algoritmit Dijkstra 16
3.6 Aplikimi i algoritmit Dijkstra 17
3.7 Implementimi i algoritmit 17
3.7.1 Kodi i aplikacionit 17
3.7.2 Ekzekutimi i aplikacionit 22
4. Referencat 24

1. Hyrje

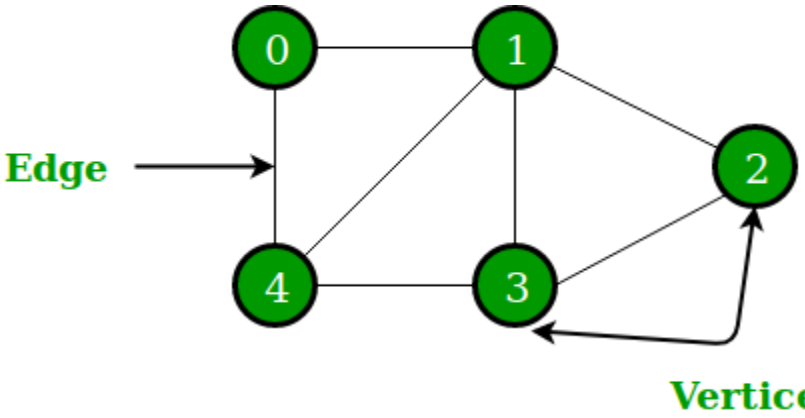
Qëllimi i projektit tonë është përshkrimi i detajuar i Algoritmit Dijkstra. Ky algoritëm është një nga algoritmet më të popullarizuar për zgjidhjen e shumë problemeve të rrugës më të shkurtër me burim të vetëm që ka peshë jo-negative të skajit në një graf.

Si fillim që t'a kuptojmë më mirë këtë algoritëm le t'i shqyrtojmë disa nga termet kryesore të grafeve.

2. Grafet

Grafi është një strukture matematikore që përbehet nga një bashkësi kulmesh dhe brinjësh që lidhin këto kulme. Formalisht brinjët trajtohen si cifte kulmesh, një brinjë *(v, w)* lidh kulmin *v* me kulmin *w*. Formalisht:

- *V* është bashkësia e kulmeve
- *E* është nënbashkësi e *V x V*



Shkruajmë **G = (V, E)** për të treguar qe G është një graf me bashkësi kulmesh bashkësinë V dhe me bashkësi brinjësh bashkësinë E. Grafi G = (V, E) është i paorientuar atëherë dhe vëtëm atëherë kur kur nqs për të gjitha kulmet *v, w* ϵ *V*, *(v, w)* ϵ *E* kemi *(w, v)* ϵ *E*. Nga perkufizimi duket që grafet e paorientuara janë raste të vecanta të grafeve të orientuar.

Bashësia *V* dhe rrjedhimisht edhe bashkësia *E* janë të pafundme. Grafe të tilla gjejnë perdorim në shumë fusha, megjithatë do të trajtojmë vetëm bashkësi të fundme *V*. Mqs *E* është nënbashkësi e *VxV* atëherë edhe *E* është e fundme.

Nqs $v \in V$ është një kulm në një graf të orientuar G=(V, E) atëherë:

- Rendi hyrës in(v) i kulmit v është numri i brinjëve hyrëse tek v, dmth numri i brinjëve të trajtës (v, u).
- Rendi dalës out(v) i kulmit v është numri i brinjëve dalëse nga v, dmth numri i brinjëve (v, u).

Në një graf të paorientuar, rendi i një kulmi v është numri i brinjëve $e \in E$ për të cilat njeri kulm është v. Dy kulme janë *fqinje* nëse ka një brinjë që i lidh ato.[2]

Grafet janë modele matematikore që mund të përdoren në probleme të ndryshme.

2.1 Zbatimet e grafeve

Hartat e linjave ajrore. Kulmet përfaqësojnë aeroporte dhe nga kulmi A tek kulmi B ka nje brinjë nëse ka një fluturim direkt nga një aeroport i A tek një aeroport i B.[1]

Qarqet elektrike. Kulmet përfaqësojnë dioda, tranzistore, celësa, etj dhe brinjët përfaqësojne telat që i lidhin ato.

Rrjetat kompjuterike. Kulmet përfaqësojnë kompjutera dhe brinjët përfaqësojnë lidhjet e rrjetit ndërmjet tyre.

Worl Wide Web. Kulmet janë faqet e internetit ndërsa brinjet janë hyperlinks që lidhin faqet me njëra tjetren.

3. Algoritmi i Dijkstra’s

Në lëmin e teknologjisë përdoren lloje të ndryshme të algoritmeve për gjetjen e rrugës më të shkurtë në mes dy nyjave të një grafi.Ne e kemi marrë si detyrë për të shpjeguar në detaje njëren nga këto algoritme: Algoritmin Dijkstra. Algoritmi Dijkstra është një algoritëm që mundëson gjetjen e rrugëve më të shkurtra midis nyjeve të një grafi. Ky algoritëm u konceptua nga shkenctari Edsger W. Dijkstra në 1956 dhe u botua tre vjet më vonë.

Në figurën më poshtë është paraqitur një graf (të lidhur dhe me peshë) për zbatimin e algoritmit të Dijkstra-s ashtu që të fitojmë rrugën më të shkurtë nga nyja C deri tek nyja E.

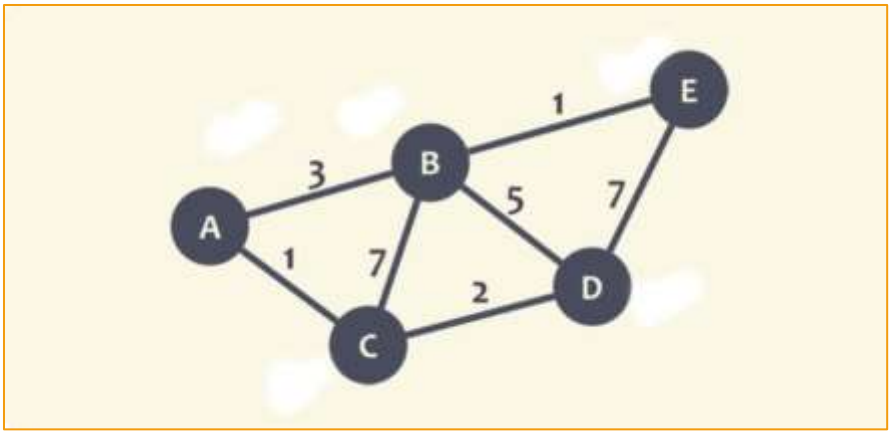


Figure 1 Hapi 1 i Algoritmit [8]

1. Gjatë ekzekutimit të algoritmit, secila nyje do të shënohet me distancën e saj minimale deri në nyjen C pasi kemi zgjedhur nyjen C.

   Në këtë rast, distanca minimale është 0 për nyjen C. Gjithashtu, për pjesën tjetër të nyjeve, pasi nuk e dimë këtë distancë, ato do të shënohen si pafundësi (∞), përveç nyjes C (aktualisht e shënuar si pikë e kuqe)

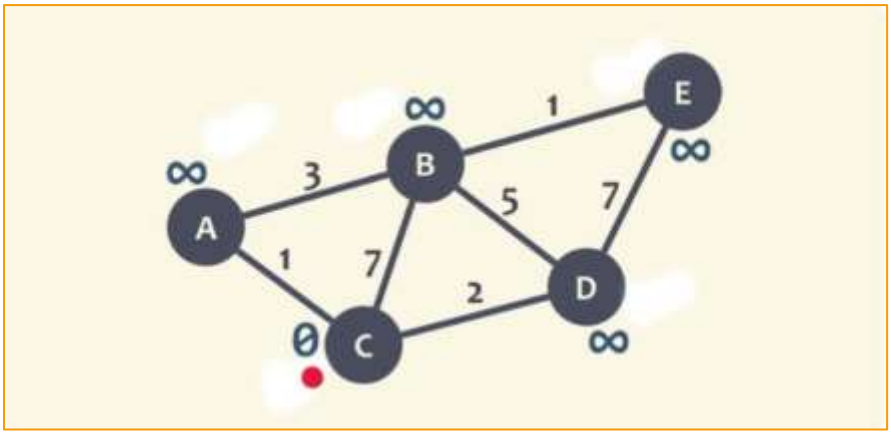


Figure 2 Hapi 2 i Algoritmit

2. Tani do të kontrollohen fqinjët e nyjes C, pra nyja A, B dhe D. Ne fillojmë me B, këtu do të shtojmë distancën minimale të nyjes aktuale (0) me peshën e deges (7) që lidhte nyja C në nyjen B dhe merrni 0+ 7 = 7.

Tani, kjo vlerë do të krahasohet me distancën minimale të B (pafundësi) dhe pasi 7 është më e vogël se pafundësia shënojmë atë vlerë si vlerën më të vogël në nyjën B .

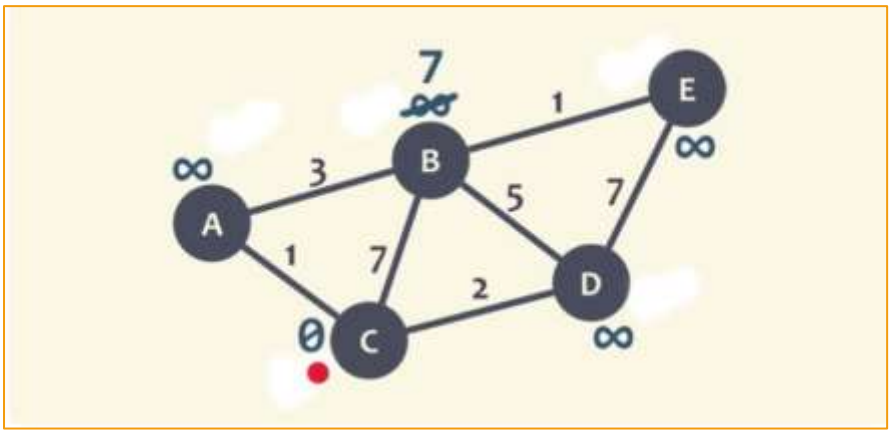


Figure 3 Hapi 3 i Algoritmit

3. Tani, i njëjti proces kontrollohet me fqinjin A. Shtojmë 0 me 1 (pesha e degës që lidh nyjen C me A) dhe marrim 1. Përsëri, 1 krahasohet me distancën minimale të A (pafundësi), dhe shënohet vlera më e ulët(pra 1).

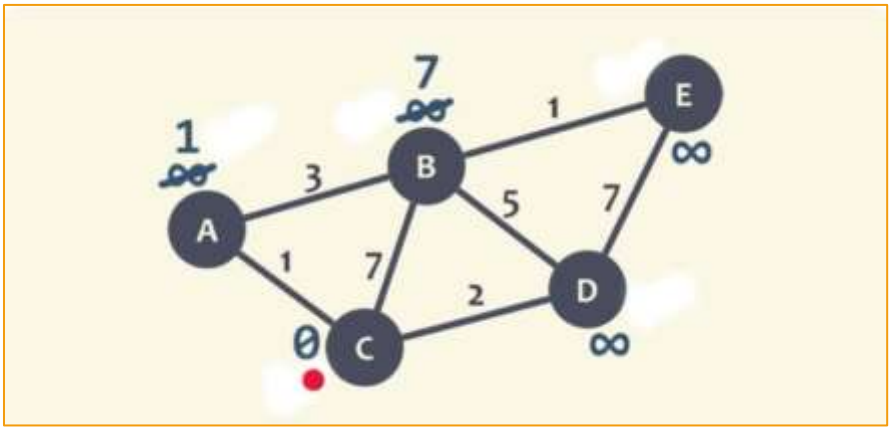


Figure 4 Hapi 5 i Algoritmit

E njëjta gjë përsëritet me nyjen D, dhe shënohet 2 si vlera më e ulët në D.

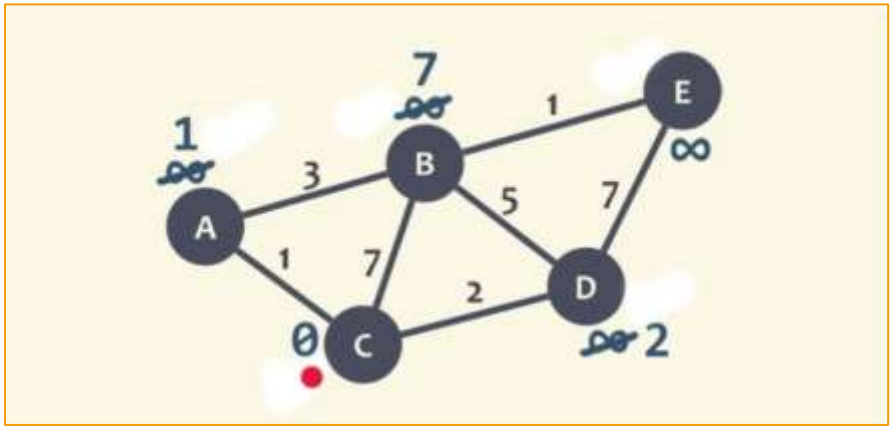


Figure 5 Hapi 6 i Algoritmit

Meqenëse, të gjithë fqinjët e nyjës C janë kontrolluar, e shënojmë nyjen C, shënohet si të vizituar me një shenjë të gjelbër.

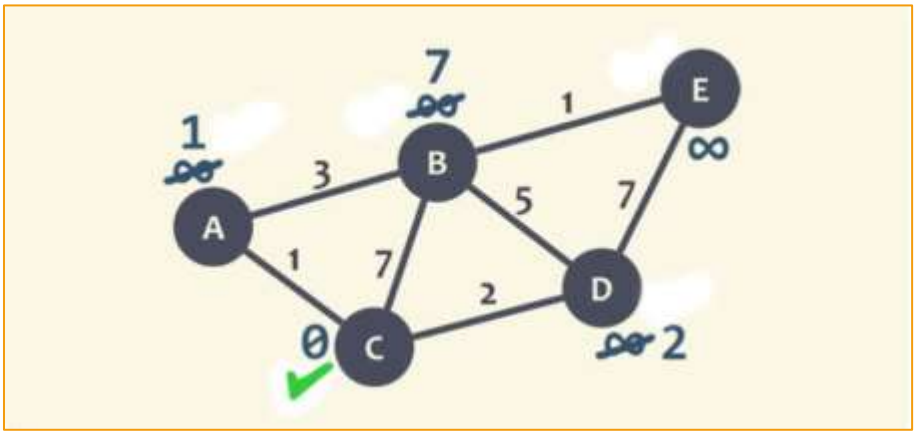


Figure 6 Hapi 7 i Algoritmit

4. Tani, ne do të zgjedhim nyjen e re aktuale ashtu që nyja duhet të mos vizitohet me distancën minimale më të ulët, ose nyjen me numrin më të vogël dhe asnjë shenjë kontrolli. Këtu, nyja A është e pavizituara me distancë minimale 1, e shënuar si nyje aktuale me pikë të kuqe.

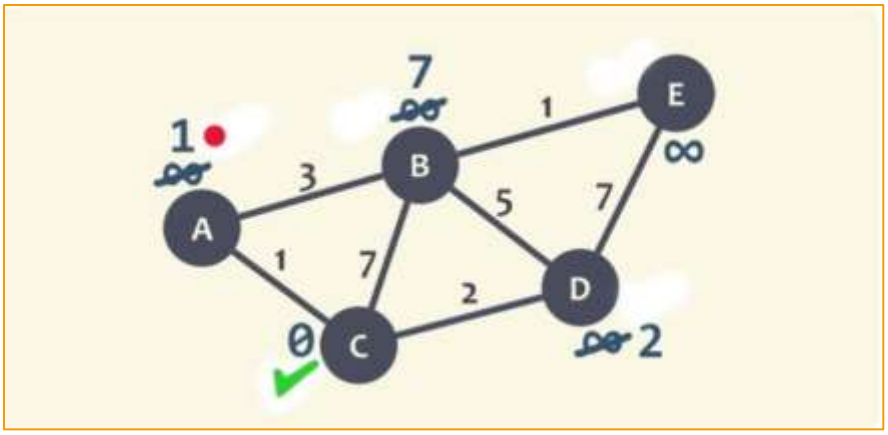


Figure 7 Hapi 8 i Algoritmit

5. Ne përsërisim algoritmin, duke kontrolluar fqinjin e nyjes aktuale ndërsa injorojmë nyjen e vizituar, kështu që vetëm nyja B do të kontrollohet.
   Për nyjen B, shtojmë 1 me 3 (pesha e skajit që lidh nyjen A me B) dhe fitojmë 4. Kjo vlerë, 4, do të krahasohet me distancën minimale të B, 7 dhe do të shënojmë vlerën më të ulët në B si 4 .

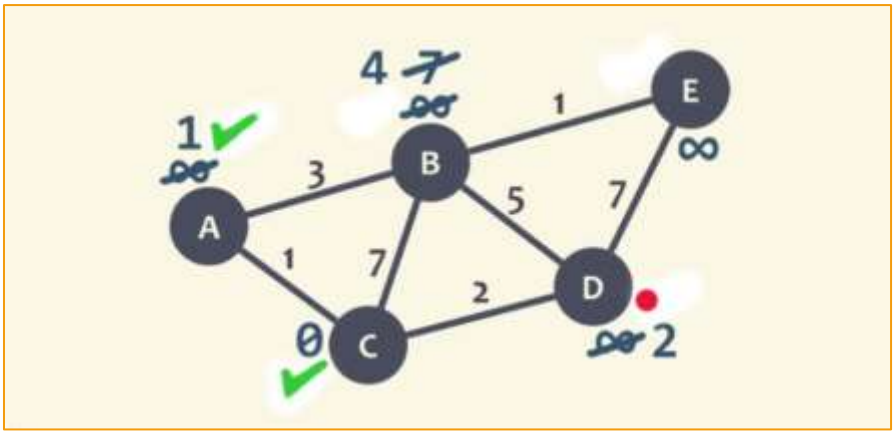


Figure 8 Hapi 9 i Algoritmit

Për nyjen B, shtojmë 2 në 5, marrim 7 dhe e krahasojmë atë me vlerën minimale të distancës B, meqenëse 7> 4, kështu që lini vlerën më të vogël të distancës në nyjen B si 4.

Për nyjen E, marrim 2+ 7 = 9, dhe e krahasojmë atë me distancën minimale të E-së që është pafundësia, dhe vlerën më të vogël e shënojmë si nyje E si 9. Nyja D shënohet si e vizituar me një shenjë të gjelbër.

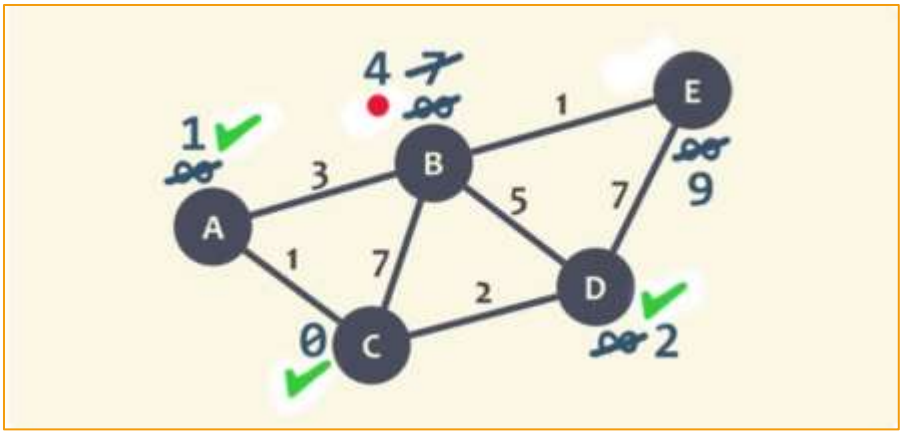


Figure 9 Hapi 10 i Algoritmit

6. Nyja aktuale është vendosur si nyja B, këtu duhet të kontrollojmë vetëm nyjen E pasi është e pavizituar dhe nyja D vizitohet. Marrim 4+ 1 = 5, e krahasojmë me distancën minimale të nyjës.

   Pasi 9> 5, lejme vlerën më të vogël në nyjën E si 5.

   Ne shënojmë D si nyje të vizituar me një shenjë të gjelbër, dhe nyja E vendoset si nyja aktuale.

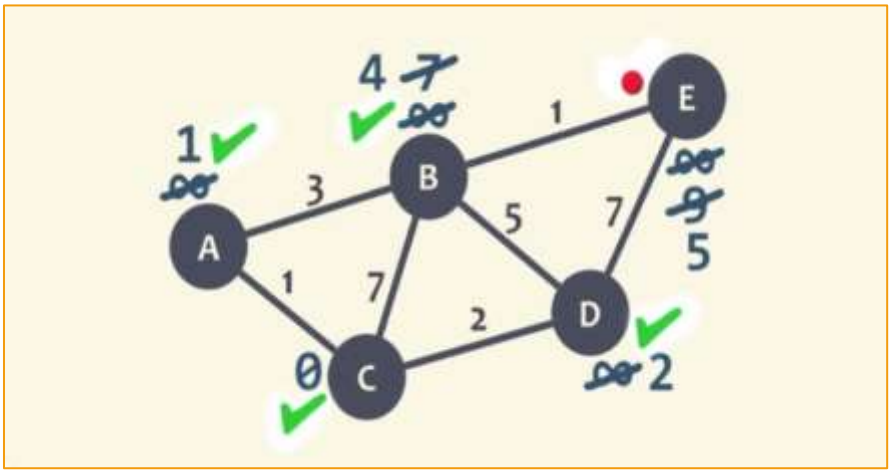


Figure 10 Hapi 11 i Algoritmit

7. Meqenëse nuk ka ndonjë fqinjë të pa vizituar, kështu që nuk ka ndonjë kërkesë për të kontrolluar ndonjë gjë. Nyja E shënohet si një nyje e vizituar me një shenjë të gjelbër.

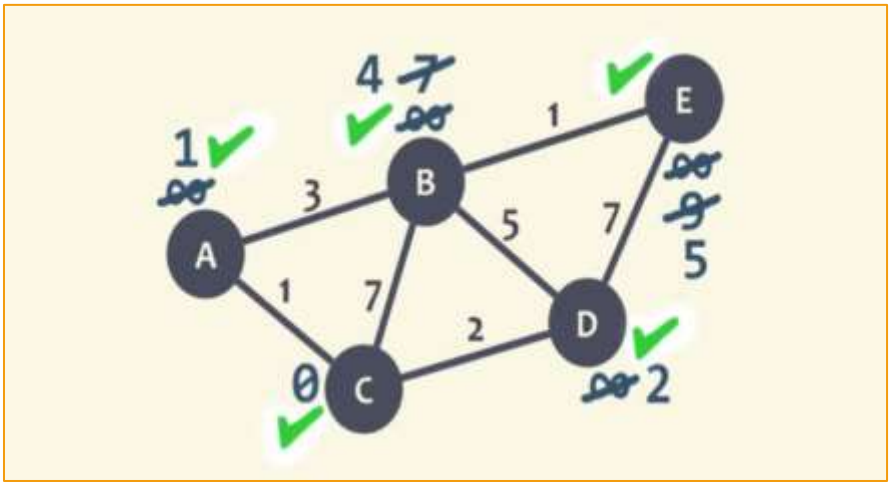


Figure 11 Hapi 12 i Algoritmit

Pra, ne kemi mbaruar pasi nuk ka nyje të pavizituara. Distanca minimale e secilës nyje tani përfaqëson distancën minimale të asaj nyje nga nyja C

Me këtë algoritëm në menyrë shumë të thjeshtë përftojmë distancën më të shkurtër prej një nyje deri te të gjitha nyjat e tjera në graf, mirëpo, në qoftë se na nevojiten vlerat e distancave më të shkurtra ndërmjet cdo dy nyjave të ndryshme në graf, do të nevojitet që këtë algoritëm ta përserisim aq here sa ka nyje në graf.

3.1 Kompleksiteti kohor

3.1.1 Cka eshte kompleksiteti i algoritmit?

Kompleksiteti algoritmik shënohet me O ("Big O notation" sic thuhet në anglisht) dhe është një funksion matematikor që varet nga një parameter, ky parameter sic mund ta paramendoni është numri i elementve që hyn në algoritëm. Do e shënojmë këtë numër me shkronjën n.

Në algoritmikë gjejmë shpesh shkallë te kompleksitetit të tipit :

Shënimi	Tipi i KOMPLEKSITETIT
$O(1)$	Kompleksiteti konstant (i pamvarur nga madhësia e të dhënave)
$O(log(n))$	Kompleksiteti logaritmik
$O(n)$	Kompleksiteti linear
$O(nlog(n))$	Kompleksiteti gati-linear
$O(n^2)$	Kompleksiteti kuadratik
$O(n^3)$	Kompleksiteti kubik
$O(n^p)$	Kompleksiteti polinomial
$O(n^{\log(n)})$	Kompleksiteti gati-polinomial
$O(2^n)$	Kompleksiteti ekspononcial
$O(n!)$	Kompleksiteti faktor

Figure 12 Tipet e kompleksitetit

Në figurën e mëposhtme tregohet secili kompleksitet. Vërejmë se algoritmat në të djathtë të grafikut që janë të shënuar me ngjyrë: jeshile, verdhë kanë një kompleksitet algoritmik më të mirë se sa ata në të majtë që janë më të kuqe. Psh një algoritëm që ka kompleksitetin O(n!) nuk mund të vihet asnjëherë në praktikë sepse kërkon shumë kohë për të mbaruar. [9]

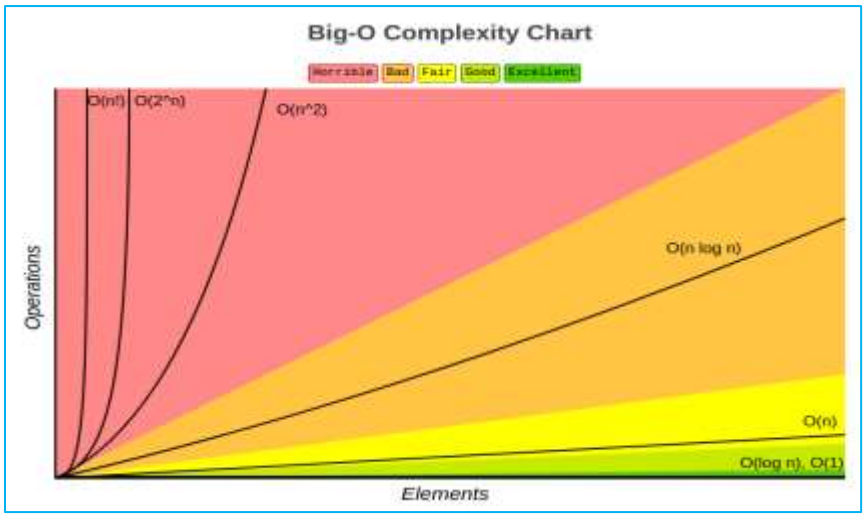


Figure 13 Big-O

3.1.2 Kompleksiteti kohor i algoritmit Dijkstra

Kompleksiteti i kohës në shkencat kompjuterike paraqet kompleksitetin kompjuterik që përshkruan sasinë e kohës që duhet për tu ekzekutuar një algoritëm. Zakonisht konsiderohen 2 rastet kompleksiteti i rastit më të keq, që është sasia maksimale e kohës që kërkohet për inputet e një madhësie të caktuar dhe rasti më pak i zakonshëm që është kompleksiteti i rasteve mesatare pra mesatarja e kohës së marrë në inputet e një madhësie të caktuar. Ndërsa termi kompleksiteti i rastit më të mirë përdoret për të përshkruar sjelljën e një algoritmi në kushte optimale. [7]

Algoritmi Dijkstra përdor një strukturë të të dhënave për ruajtjen e pjesëve të rënditura sipas distancës që nga fillimi!.Përderisa algoritmi orgjinal e përdor min-priority queue dhe ekzekutohet në kohën **Θ((|V| +|E|) log |V|)**, ku V paraqet numrin e nyjeve ndërsa E numrin e degëve. Po ashtu mund të implementohet në **Θ(|V|²)** duke përdorur një varg. Një ide tjeter për këtë algoritëm u propozua duke përdorur Fibonacci heap min-priority queue që mundësoj kompleksitetin e kohës së ekzekutimit në **Θ((|E| +|V|) log |V|)**, kjo paraqet lidhjen më të mirë.

Nëse përdoret standard binary heap, atëherë kompleksiteti është **O (| E | log | E |)** ,

| E | log | E | termi vjen nga | E | azhurnimet për grumbullimin standard.

Nëse grupi i përdorur është një radhë prioritare,atëherë kompleksiteti është **O (| E | + |V|^2)**

- Kompleksiteti kohor i rastit më të keq: **Θ(E+V log V)**
- Kompleksiteti kohor i rastit mesatar : **Θ(E+V log V)**
- Kompleksiteti kohor i rastit më të mirë : **Θ(E+V log V)**
- Kompleksiteti hapesinor: **Θ(V)**

Nëse nuk perdoret priority queue kompleksiteti kohor do të jetë : **Θ(E+V^2)** .

3.2 Krahasimet e Algoritmit te Dijkstra's me algoritmet tjera

3.2.1 Krahasimet e Algoritmit te Dijkstra's dhe algoritmit Bellman – Ford

Algoritmi i Dijkstra's	Algoritmi Bellman-Ford
Algoritmi I Dijkstra's ka disavantazh në grafet me peshë negative.	Algoritmi I Bellman – Ford ka përparesi kur kemi të bëjmë me peshë negative në graf.
Rezultati përmban kulmet që përmbajnë informacion të plotë në lidhje me grafin, jo vetëm kulmet me të cilët janë të lidhur.	Rezultati përmban kulmet e cila përmban informacionin në lidhje me kulmet e tjera me të cilat janë lidhur.
Merr më pak kohë se Algoritmi Bellman-Ford. Komplekstiteti kohor është O(ElogV).	Merr më shumë kohe se algoritmi I Dijkstra's. Kompleksiteti kohor eshte O(VE)
Merret qasje Greedy për të zbatuar algoritmin.	Për implementimin e algoritmit merret qasja e Programimit Dinamik.

[10]

3.2.2 Krahasimet e Algoritmit te Dijkstra's dhe algoritmit Dijkstra - Prim

Algoritmi I Dijkstra's	Algoritmi Dijkstra-Prim
Gjen rrugën më të shkurtër	Gjen MST(Minimum Spanning Tree)
Mund te punojë në dy lloje të grafeve, në grafet e orientura dhe në grafe të paorientuara.	Punon vetëm me grafet e paorientuara.
Duhet të përcaktohet nyja fillestare dhe destinacioni.	Nuk duhet të përcakotohet nyja fillestare.
Ky algoritëm ka disavantazh rastet e degëve me peshë negative në graf.	Mund të punojë edhe kur kemi raste me degë me peshë negative në graf.

3.3 Pseudokodi

Në algoritmin e mëposhtëm të pseudokodit, kodi *u* ← kulm në Q me min dist [u], kërkon kulmin *u* në bashkësinë e kulmit *Q* që ka vlerën më të vogël dist *[u]* gjatësia (u, v) kthen gjatësinë e bashkimit të skajit (dmth. distancën midis) dy nyjeve fqinje *u* dhe *v*. Ndryshorja alt në rreshtin 18 është gjatësia e shtegut nga nyja rrënjë në nyjen fqinje v nëse do të kalonte përmes u. Nëse kjo rrugë është më e shkurtër se rruga më e shkurtër aktuale e regjistruar për v, ajo rrugë aktuale zëvendësohet me këtë rrugë alt. Vargu prev(previous) është i mbushur me një tregues në nyjën "next-hop" në grafin burimor për të marrë rrugën më të shkurtër deri tek burimi.

```
function Dijkstra(Graph, source):

    create vertex set Q

   for each vertex v in Graph:
       dist[v] ← INFINITY
       prev[v] ← UNDEFINED
       add v to Q
   dist[source] ← 0

   while Q is not empty:
       u ← vertex in Q with min dist[u]

       remove u from Q

      for each neighbor v of u:           // only v that are still in Q
          alt ← dist[u] + length(u, v)
           if alt < dist[v]:
              dist[v] ← alt
              prev[v] ← u

 return dist[], prev[]
```

3.4 BllokDiagrami

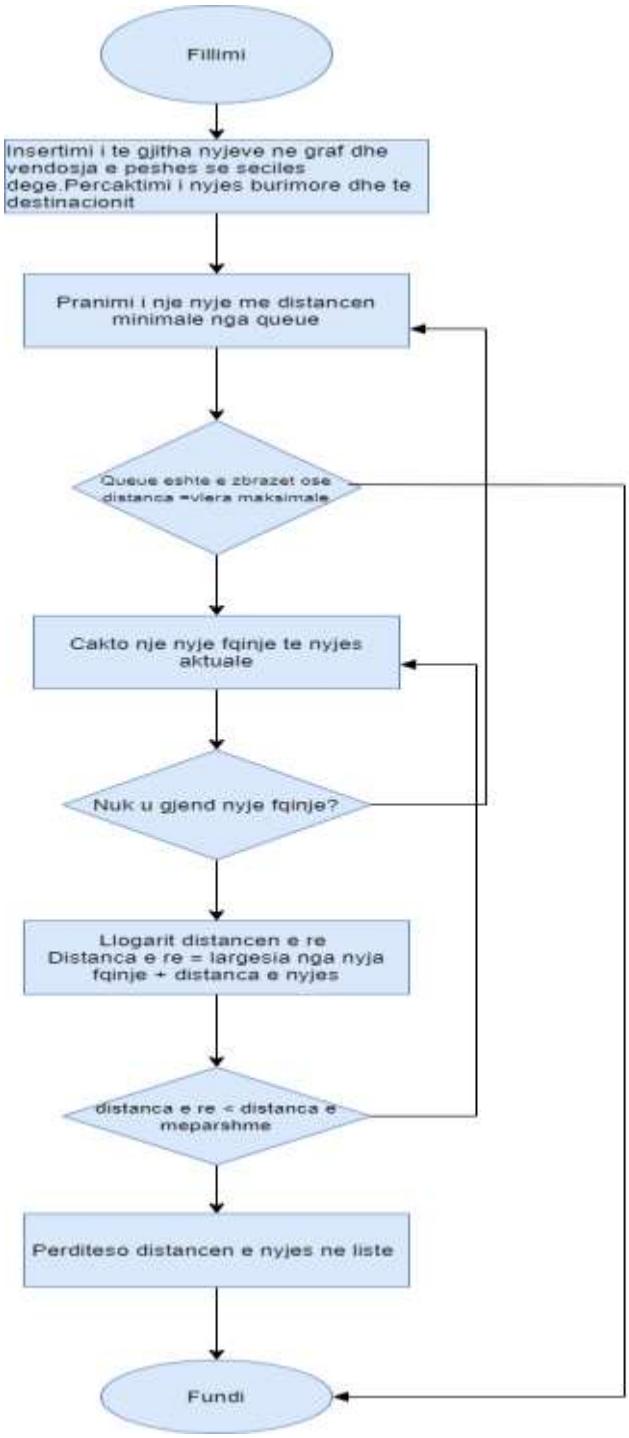


Figure 14 BllokDiagrami

3.5 Disavantazhi i Algoritmit Dijkstra

Siç e dimë vetia themelore e përdorur në Dijkstra është mbledhja e dy numrave pozitivë, prandaj, ky algoritëm mund të çojë në përgjigje të gabuar në rastin e grafikut që përmban skaje negative.

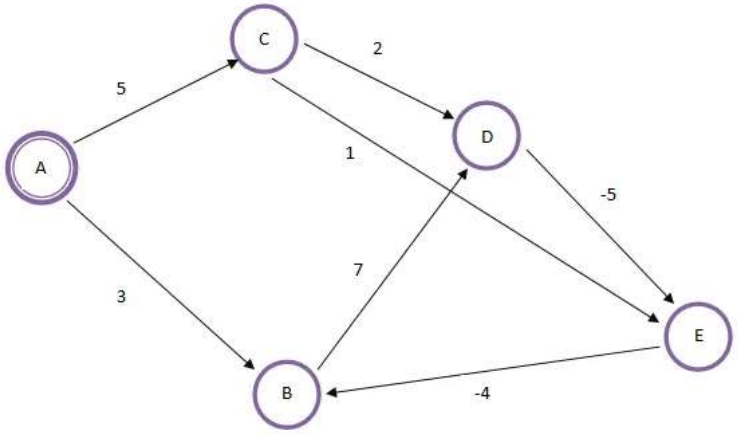


Figure 15 Graf qe permban dege me peshe negative

Nyja fillestare: A
Dijkstra do të llogarisë 3 si distancë minimale për të arritur B nga A.

Por ne mund ta shohim qartë A-> C-> E-> B rruga do të kushtojë 2 për të arritur B nga A.

3.6 Aplikimi i algoritmit Dijkstra

- ***Shërbimet e Hartësimit Dixhital në Google Maps*** *:* Algoritmi i Dijkstra përdoret për të gjetur distancën minimale midis dy vendndodhjeve përgjatë shtegut .

- ***Aplikacionet e Rrjeteve Sociale*** *:* Algoritmi standard Dijkstra mund të zbatohet duke përdorur rrugën më të shkurtër midis përdoruesve të matur përmes 'handshakes' ose lidhjeve mes tyre.

- ***Rrjeti Telefonik***: Dijkstra mund të përdoret edhe në rrjetet telefonike për gjetjen e rrugës më të shkurtër gjë që ndikon në bartjen e sasisë së informacioneve në linjat përkatëse.

- ***IP Routing*** *:* Algoritmi i Dijkstra përdoret gjerësisht në protokollet e rutimit të kërkuara nga ruterat për të azhurnuar tabelën e tyre të përcjelljes. Algoritmi siguron shtegun më të shkurtër të kostos nga routeri burimor te rrugëzuesit e tjerë në rrjet.

- ***Agjenda e fluturimit*** *:* Ky algoritëm përdoret në rastet kur agjenti dëshiron të përcaktojë kohën më të hershme të mbërritjes për destinacionin e dhënë një aeroport origjine dhe kohën e fillimit.

3.7 Implementimi i algoritmit

3.7.1 Kodi i aplikacionit

```
package dijkstra;

import models.Edge;
import models.Graph;
import models.Node;

import java.util.*;

import dijkstra.AlgoritmiDijkstra.NodeComparator;

public class AlgoritmiDijkstra {
    private boolean safe = false;
    private String message = null;

    private Graph graph;

    private Map<Node, Node> predecessors;
    private Map<Node, Integer> distances;
```

```java
private PriorityQueue<Node> unvisited;
private HashSet<Node> visited;

public class NodeComparator implements Comparator<Node>  {

    @Override
    public int compare(Node node1, Node node2) {
        int distanca = distances.get(node1) - distances.get(node2);
        return distanca;
    }
};

 public AlgoritmiDijkstra(Graph graph){
    this.graph = graph;
    predecessors = new HashMap<>();
    distances = new HashMap<>();
    for(Node node : graph.getNodes()){
        distances.put(node, Integer.MAX_VALUE);
    }

    visited = new HashSet<>();

    safe = evaluate();
}

private boolean evaluate(){
    //nese nuk caktohet nyja burimore
    if(graph.getSource()==null){
        message = "Duhet te caktohet nyja burimore";
        return false;
    }
    //nese nuk caktohet nyja e destinacionit
    if(graph.getDestination()==null){
        message = "Duhet te caktohet nyja e destinacionit";
        return false;
    }
    //nese mbet ndonje nyje e palidhur
    for(Node node : graph.getNodes()){
        if(!graph.isNodeReachable(node)){
            message = "Cdo nyje duhet te jete e lidhur ne graf";
            return false;
        }

    }
```

```
        return true;
    }

    public void run() throws IllegalStateException {
        if(!safe) {
            throw new IllegalStateException(message);
        }

        unvisited = new PriorityQueue<>(graph.getNodes().size(), new NodeComparat
or());

        Node source = graph.getSource();
        distances.put(source, 0);
        visited.add(source);


        for (Edge neighbor : getNeighbors(source)){
            Node adjacent = getAdjacent(neighbor, source);
            distances.put(adjacent, neighbor.getWeight());
            predecessors.put(adjacent, source);
            unvisited.add(adjacent);
        }
        //perderisa nuk i kemi vizituar te gjitha nyjet
        while (!unvisited.isEmpty()){
            Node current = unvisited.poll();
            updateDistance(current);
            unvisited.remove(current);
            visited.add(current);

        }
        System.out.println("Distanca nga  "+graph.getSource()+ " ne  " +graph.get
Destination()+ " eshte:" +distances.get(graph.getDestination()));


        for(Node node : graph.getNodes()) {
            node.setPath(getPath(node));
            updateDistance(node);
        }
        graph.setSolved(true);

    }


   //funksioni qe perdoret per te caktuar distancen e rruges me te shkurte
    private void updateDistance(Node node){
```

```
        int distance = distances.get(node);
        for (Edge neighbor : getNeighbors(node)){
            Node adjacent = getAdjacent(neighbor, node);
            if(visited.contains(adjacent))
                continue;

            int current_dist = distances.get(adjacent);

            int new_dist = distance + neighbor.getWeight();

            if(new_dist < current_dist) {
                distances.put(adjacent, new_dist);
                predecessors.put(adjacent, node);
                unvisited.add(adjacent);
            }


        }
    }

    // Metoda per me e marr nyjen fqinje nyjes me degen perkatese qe vjen si inpu
t
    private Node getAdjacent(Edge edge, Node node) {
        if(edge.getNodeOne()!=node && edge.getNodeTwo()!=node)
            return null;

        return node==edge.getNodeTwo()?edge.getNodeOne():edge.getNodeTwo();
    }
    // Metoda per me i marr deget fqinje te nyjes perkatese qe vjen si input
    private List<Edge> getNeighbors(Node node) {
        List<Edge> neighbors = new ArrayList<>();

        for(Edge edge : graph.getEdges()){
            if(edge.getNodeOne()==node ||edge.getNodeTwo()==node)
                neighbors.add(edge);
        }

        return neighbors;
    }




    // funksioni qe mundeson marrjen e distances se nje nyjeje
    public Integer getDistance(Node node){
```

```
        return distances.get(node);
    }
   // per te mundesuar marrjen e rruges se destinacionit
    public List<Node> getDestinationPath() {
        return getPath(graph.getDestination());
    }

    //metoda qe mundson marrjen e path te nyjes aktuale
    public List<Node> getPath(Node node){
        List<Node> path = new ArrayList<>();

        Node current = node;
        path.add(current);
        while (current!=graph.getSource()){
            current = predecessors.get(current);
            path.add(current);




        }


        return path;
    }

}
```

3.7.2 Ekzekutimi i aplikacionit

Pas ekzekutimit të aplikacionit do të fitojmë një frame ku janë të vendosur tre butona dhe ku në të cilën më pas mund të vizatojm grafin. Me shtypjen e butonit të parë mundësohet resetimi apo kthimi në gjendje fillestare, ndërsa me shtypjen e butonit të dytë mundësohet gjetja e rrugës më të shkurtër në mes dy nyjeve që ne i caktojmë, dhe me shtypjen e butonit *i* do të paraqitet në mënyrë të detajuar një dritare që na informon se si mund të krijojmë nyje, të i zhvendosim apo të i fshijme ato.

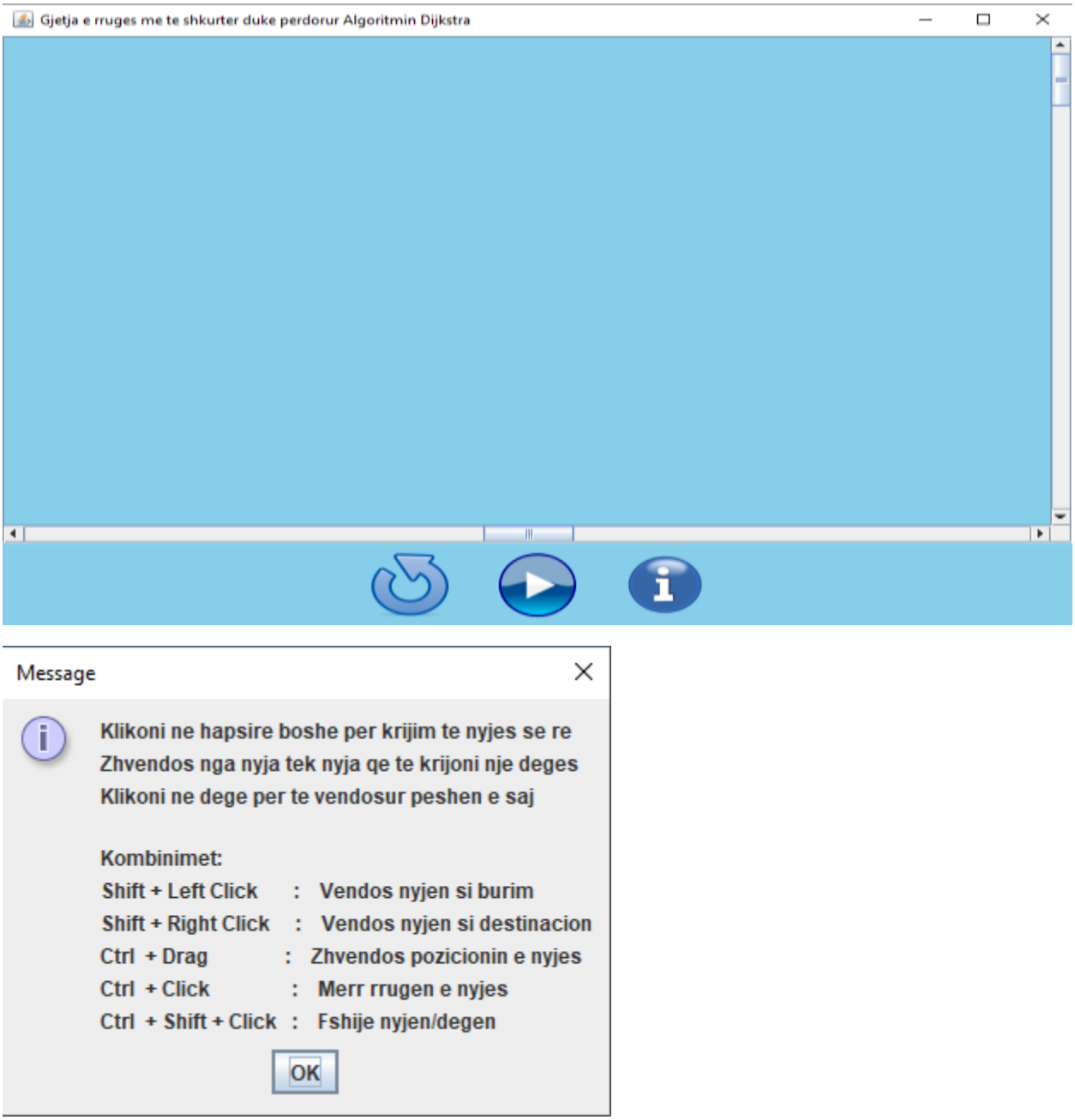


Figure 16 Informata rreth perdorimit te applikacionit

Pas vendosjës së nyjeve dhe degëve, duke përfshirë këtu mundësinë për vendosjen e peshës së degëve dhe percaktimit të nyjës fillestare dhe nyjës së destinacionit, me shtypjen e butonit(run) algoritmi Dijkstra do të ekzektuohet dhe do t'a gjejë distancën më të shkurtër mes nyjeve të përcaktuara.

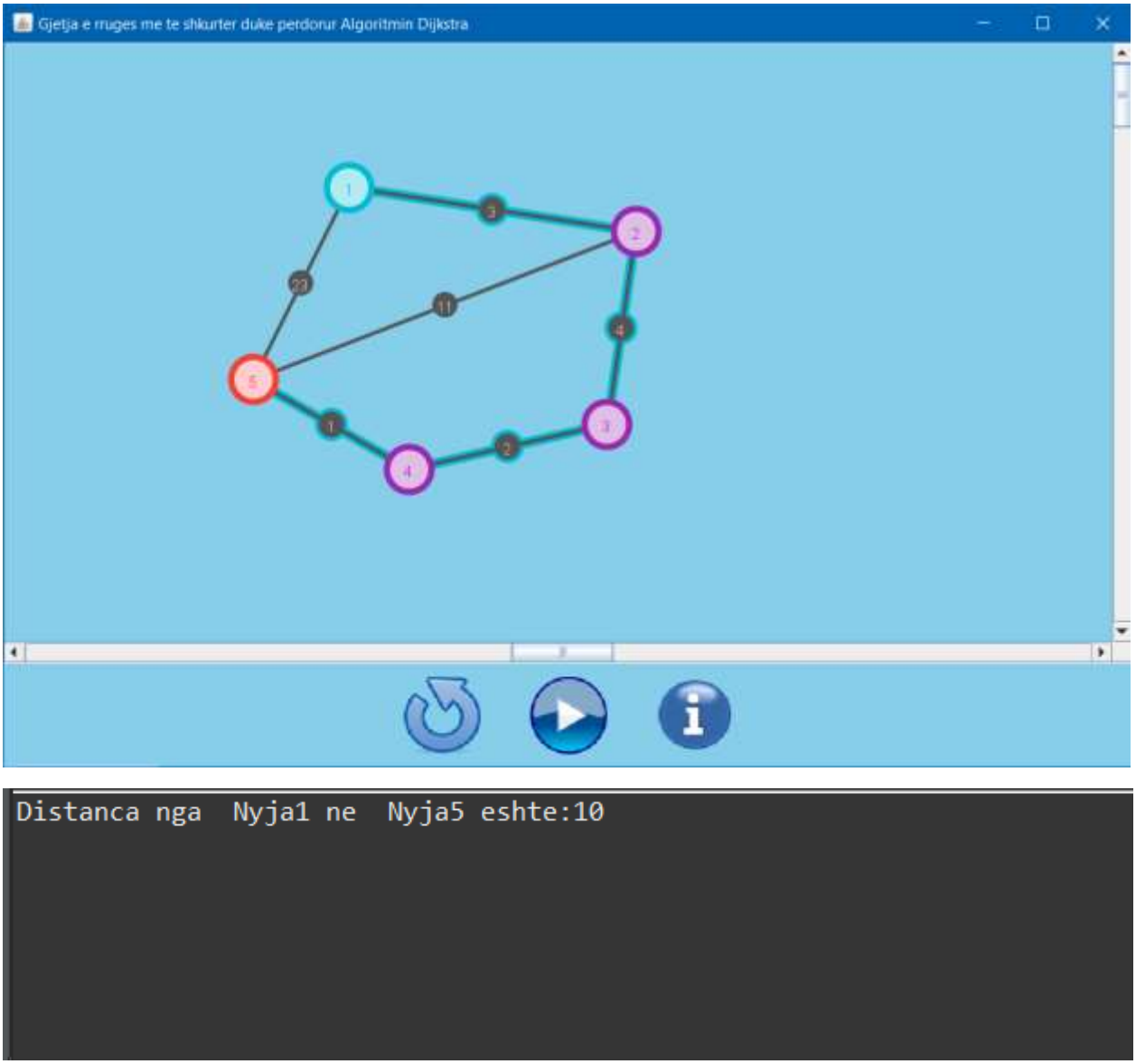


Figure 17. Ekzekutimi i applikcaionit

4.Referencat

[1] https://issuhub.com/view/index/4567 Learning Approach - Jeffrey J. McConnell, JONES AND BARTLETT PUBLISHERS

[2] MATEMATIKA III K(ME MATEMATIKE DISKRETE) - Prof. Asc. Dr. Qefsere Doko - Gjonbalaj

[3] https://issuu.com/klodian2/docs/algoritmi_e_programim_java

[4] https://sites.google.com/a/fshn.edu.al/leogena-zhaka/kurset/departamenti-i-informatikes/informatike/algoritmikee-neen-c/leksion-06

[5] https://iq.opengenus.org/dijkstras-algorithm-finding-shortest-path-between-all-nodes/

[6] https://github.com/iamareebjamal/dijkstra-gui

[7] https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[8] https://www.analyticssteps.com/blogs/dijkstras-algorithm-shortest-path-algorithm

[9] https://kristogodari.com/blog/shqip/kompleksiteti-algoritmik-i-strukturave-ne-te-cilat-ruhen-te-dhenat-informatike/

[10] https://www.geeksforgeeks.org/what-are-the-differences-between-bellman-fords-and-dijkstras-algorithms