Breadth-First Search

Elvira Jahaj
Fakulteti i Shkencave Matematiko-Natyrore
Drejtimi:Shkenca Kompjuterike
Prizren, Kosovë 20000
elvira.jahaj@student.uni-pr.edu

Elsa Tafilaj
Fakulteti i Shkencave Matematiko-Natyrore
Drejtimi:Shkenca Kompjuterike
Klinë, Kosovë 32000
elsa.tafilaj@student.uni-pr.edu

Dafina Sopa
Fakulteti i Shkencave Matematiko-Natyrore
Drejtimi:Shkenca Kompjuterike
Kacanik, Kosovë 71000
dafina.sopa@student.uni-pr.edu

***Abstrakt* – Qellimi i këtij punimi është shpjegimi dhe kuptimi sa më i mirë i Breadth-First Search një algoritëm kerkimi. Kjo perfshin nevojën e krijimit të kwtij algoritmi dhe cka më saktësisht bën ky algoritëm.**

Fjalet kyçe:algorithm ,search , kosto optimale , kompleksitet,zgjidhje .

I. HYRJE

Algoritmet e kërkimit janë një nga fushat më të rëndësishme të Inteligjencës Artificiale. Ky projekt do të shpjegojë gjithçka rreth algoritmeve të kërkimit në AI. Në inteligjencën artificiale, teknikat e kërkimit janë metoda universale të zgjidhjes së problemeve. Agjentët racionalë ose agjentët e zgjidhjes së problemeve në AI përdorën kryesisht këto strategji ose algoritme kërkimi për të zgjidhur një problem specifik dhe për të ofruar rezultatin më të mirë. Agjentët për zgjidhjen e problemeve janë agjentë të bazuar në qëllime dhe përdorin paraqitje atomike. Në këtë temë do të mësojmë algoritme të ndryshme kërkimi për zgjidhjen e problemeve. Terminologjite e algoritmit tw kwrkimit janw:

Search- Kërkimi është një procedurë hap pas hapi për të zgjidhur një problem kërkimi në një hapësirë të caktuar kërkimi. Një problem kërkimi mund të ketë tre faktorë kryesorë:

Hapësira e kërkimit-përfaqëson një grup zgjidhjesh të mundshme, që mund të ketë një sistem.

Gjendja e fillimit- Është një gjendje nga ku agjenti fillon kërkimin.

Testi i qëllimit- Është një funksion i cili vëzhgon gjendjen aktuale dhe kthen nëse gjendja e qëllimit është arritur apo jo.

Pema e kërkimit- Një paraqitje peme e problemit të kërkimit quhet pema e kërkimit. Rrënja e pemës së kërkimit është nyja rrënjë e cila korrespondon me gjendjen fillestare.

Veprimet- I jep agjentit përshkrimin e të gjitha veprimeve të disponueshme.

Modeli i tranzicionit- Një përshkrim i asaj që bën çdo veprim, mund të përfaqësohet si një model tranzicioni.

Kostoja e rrugës- Është një funksion i cili i cakton një kosto numerike çdo shteg.

Zgjidhja- Është një sekuencë veprimi që çon nga nyja fillestare në nyjen e qëllimit.

Zgjidhja optimale- Nëse një zgjidhje ka koston më të ulët nga të gjitha zgjidhjet.

Bazuar në problemet e kërkimit, ne mund t'i klasifikojmë algoritmet e kërkimit në algoritme të kërkimit të painformuar dhe kërkim të informuar .

Nw kwtw projekt ne do tw tregojmw pwr algoritmin Breadth-first search i cili bwnw pjesw në algoritme të kërkimit të painformuar [1].

II. ÇKA ËSHTË BREADTH-FIRST SEARCH

Kur të gjitha veprimet kanë të njëjtën kosto, një strategji e përshtatshme është Breadth-First Search, në të cilin fillimisht zgjerohet nyja rrënjë, pastaj zgjerohen të gjithë pasardhësit e nyjes rrënjë, pastaj pasuesit e tyre, e kështu me radhë. Kjo është një strategji kërkimore sistematike, e cila për këtë arsye është kerkim i plotw edhe në hapësirat e gjendjes së pafundme. Ne mund të zbatojmë kërkimin e parë në gjerësi si një thirrje për BEST-FIRST-SEARCH ku funksioni i vlerësimit f(n) është thellësia e nyjës—d.m.th. numri i veprimeve që duhen për të arritur në nyje [2].

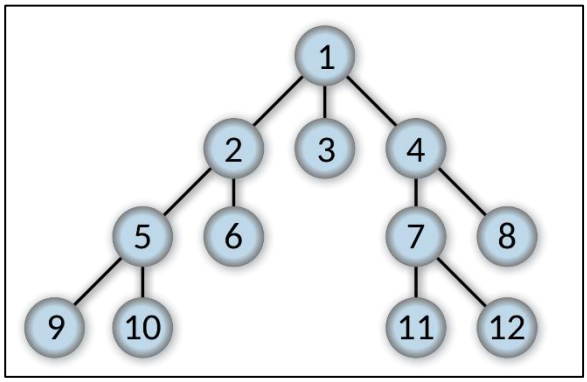

Figure 1.Breadth-first search

Algoritmi i kërkimit Breadth-First Search gjen gjithmonë një zgjidhje me një numër minimal veprimesh, sepse kur gjeneron nyje në thellësinë d, ai tashmë ka gjeneruar të gjitha nyjet në thellësinë d − 1, kështu që nëse njëri prej tyre do të ishte zgjidhje, do të kishte është gjetur. Kjo do të thotë se është kosto optimale për problemet ku të gjitha veprimet kanë të njëjtën kosto, por jo për problemet që nuk e kanë atë pronë. Është i plotë në të dyja rastet. Për sa i përket kohës dhe hapësirës, imagjinoni të kërkoni një pemë uniforme ku çdo shtet ka b pasues. Rrënja e pemës së kërkimit gjeneron b nyje, secila prej të cilave gjeneron b më shumë nyje, për një total prej b 2 në nivelin e dytë. Secila prej këtyre gjeneron b më shumë nyje, duke nxjerrë b 3 nyje në nivelin e tretë, e kështu me radhë. Tani supozojmë se zgjidhja është në thellësinë d [2]. Atëherë numri i përgjithshëm i nyjeve të gjeneruara është:

$$1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

III. PËRDORIMI I ALGORITMIT BFS

BFS (Breadth-First Search) mund të përdoret në një sërë aplikacionesh. Këto janë disa nga rastet më të zakonshme të përdorimit për BFS:

***Rruga më e shkurtër**:* BFS shpesh përdoret për të gjetur shtegun më të shkurtër midis dy nyjeve në një grafik ose pemë të papeshuar.

Përshkimi i një grafiku ose peme: BFS mund të përdoret për të përshkuar një grafik ose pemë në rendin e parë të gjerësisë. Kjo mund të jetë e dobishme për probleme të tilla si gjetja e të gjitha nyjeve që janë në një distancë të caktuar nga një nyje e caktuar, ose për zbulimin e cikleve në një grafik.

Crawling në ueb: BFS mund të përdoret për të zvarritur ueb-in, duke vizituar të gjitha faqet e internetit që janë të lidhura me një faqe të caktuar në një rend të parë. Kjo mund të jetë e dobishme për aplikacione të tilla si motorët e kërkimit ose scraping në internet.

Gjetja e komponentëve të lidhur: BFS mund të përdoret për të gjetur të gjithë komponentët e lidhur në një grafik të padrejtuar. Kjo mund të jetë e dobishme për aplikacione të tilla si analiza e rrjeteve sociale ose grupimi.

Gjetja e rrugës në lojëra: BFS mund të përdoret për gjetjen e shtigjeve në lojëra, ku qëllimi është të gjesh shtegun më të shkurtër midis dy pikave në një hartë të lojës. Duke paraqitur hartën e lojës si grafik, BFS mund të përdoret për të gjetur shtegun më të shkurtër midis dy pikave.

Në përgjithësi, BFS është një algoritëm i dobishëm për një gamë të gjerë aplikacionesh, veçanërisht ato që përfshijnë kërkimin ose kalimin e grafikëve ose pemëve [3].

IV. PLOTËSIA

Një metodë kërkimi përshkruhet si e plotë nëse është e garantuar të gjejë një gjendje qëllimi nëse ekziston. Kërkimi i parë në gjerësi është i plotë.

Plotësia është zakonisht një veti e dëshirueshme sepse ekzekutimi i një metode kërkimi që nuk gjen kurrë një zgjidhje nuk është shpesh i dobishëm. Nga ana tjetër, mund të ndodhë që kërkimi i të gjithë pemës së kërkimit nuk është i nevojshëm, ose thjesht nuk është

i mundur, në këtë rast një Mteodi që kërkon mjaft nga pema mund të jetë i mirë.
Një metodë që nuk është e plotë ka disavantazhet që nuk mund të besohet domosdoshmërisht nëse raporton se nuk ekziston asnjë zgjidhje [4].

V. OPTIMALITETI

Një metodë kërkimi është optimale nëse garantohet se do të gjejë zgjidhjen më të mirë që ekziston. Me fjalë të tjera, ajo do të gjejë rrugën drejt një qëllimi që përfshin marrjen e numrit më të vogël të hapave. Kjo nuk do të thotë se vetë metoda e kërkimit është efikase - mund të duhet shumë kohë për një metodë kërkimi optimale për të identifikuar zgjidhjen më të mirë, por pasi të ketë gjetur zgjidhjen, ajo është e garantuar të jetë më e mira.
Kërkimi me gjerësi të parë është një metodë kërkimi optimale. Për shkak se kërkimi i parë në gjerësi shqyrton të gjitha nyjet në një thellësi të caktuar përpara se të kalojë në thellësinë tjetër, nëse gjen një zgjidhje, nuk mund të ketë një zgjidhje tjetër më parë në pemën e kërkimit [4].

VI. KOMPLEKSITETI

Në shkencën kompjuterike, ***kompleksiteti kohor*** është kompleksiteti llogaritës që përshkruan sasinë e kohës që duhet për të ekzekutuar një algoritëm.
Meqenëse në rastin më të keq kërkimi i parë në gjerësi duhet të marrë në konsideratë të gjitha shtigjet për të gjitha nyjet e mundshme, kompleksiteti kohor i kërkimit të parë në gjerësi është O(|E| + |V|) ku |V| dhe |E| është kardinaliteti i grupit të kulmeve dhe skajeve përkatësisht.
Kompleksiteti është ky pasi çdo kulm dhe çdo skaj do të eksplorohet në rastin më të keq [5].
Kompleksiteti hapësinorë është një masë e sasisë së ruajtjes së punës që i nevojitet një algoritmi. Kjo do të thotë se sa memorie, në rastin më të keq, nevojitet në çdo pikë të algoritmit.
Meqenëse të gjitha nyjet e një niveli duhet të ruhen derisa të gjenerohen nyjet e tyre fëmijë në nivelin tjetër, kompleksiteti i hapësirës është proporcional me numrin e nyjeve në nivelin më të thellë.
Kompleksiteti hapësinorë mund të shprehet gjithashtu si O(|V|) ku |V| është kardinaliteti i grupit të kulmeve. Në skenarin më të keq, grafiku ka një thellësi prej 1 dhe të gjitha kulmet duhet të ruhen [6].

Table 1.Kompleksiteti i breadth-first search

Algorithm	**Complete**	**Optimal cost**	**Time**	**Space**
BFS	**Yes**	**Yes**	**O(b^d)**	**O(b^d)**

VII. ALGORITMI

INPUT: Trungu me nyjet

OUTPUT: Zgjidhja ose porosia

Step 1: Queue<Node> queue = new LinkedList<>();

Step 2: Set <Node> visited = new HashSet<>();

Step 3: queue.add(start)

Step 4: visited.add(start)

Step 5: while (!queuu.isEmpty()) do Step 6-7

Step 6: set Node curr = queue.poll()

Step 7: Output (curr.val + “”)

Step 8: for (Node neighbor : curr.neighbors) do Step 9-11

Step 9: If (!visited.contains(neighbor)) do Step 10-11

Step 10: queue.add(neighbor)

Step 11: visited.add(neighbor)

Step 12: Output ose porosia

STOP.

VIII. IMPLEMENTIMI NE JAVA

```
import java.util.*;
class Node {
        int value;
        List<Node> neighbors;
  public Node(int value) {
        this.value = value;
        this.neighbors = new ArrayList<>();
        }
 }
   public class BFS {
        public static void main(String[] args) {
        // Krijimi i nyjeve
        Node node1 = new Node(1);
        Node node2 = new Node(2);
        Node node3 = new Node(3);
        Node node4 = new Node(4);
        Node node5 = new Node(5);
        Node node6 = new Node(6);
        // Formimi i grafit
        node1.neighbors.add(node2);
        node1.neighbors.add(node3);
        node2.neighbors.add(node4);
        node3.neighbors.add(node4);
        node3.neighbors.add(node5);
        node4.neighbors.add(node6);
        node5.neighbors.add(node6);

        // Call BFS starting from node1
        bfs(node1);
        }
public static void bfs(Node start) {
        Queue<Node> queue = new LinkedList<>();
        Set<Node> visited = new HashSet<>();
        queue.add(start);
        visited.add(start);
        while (!queue.isEmpty()) {
           Node curr = queue.poll();
           System.out.print(curr.value + " ");
         for (Node neighbor : curr.neighbors) {
                if (!visited.contains(neighbor)) {
                queue.add(neighbor);
                visited.add(neighbor);
            }
          }
       }
    }
 }
```

IX. REFERENCAT

[1] "javaTpoint-BFS algorithm" . Available: https://www.javatpoint.com/breadth-first-search-algorithm.

[2] S. J. R. a. P. Norvig, Artificial Intelligence A Modern Approach Fourth Edition, Pearson Education, 2010.

[3] "geeksforgeeks-Applications of Breadth First Traversal". Available: https://www.geeksforgeeks.org/applications-of-breadth-first-traversal/.

[4] B. Coppin, Artificial Intelligence Illuminated, 2004: Jones and Bartlett Publishers.

[5] "VirtualLabs-Time and Space complexity of BFS". Available: Jones and Bartlett Publishers.

[6] "linuxhint-BFS Time Complexity" . Available: https://linuxhint.com/bfs-time-complexity/.