Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Ingineria Software și Automatică

**Proiect de curs**

Disciplina: Analiza și Proiectarea Algoritmilor

**Tema:** Implementarea algoritmului Dijkstra pentru soluționarea a unei probleme de transport.

Implementation of the Dijkstra algorithm for solving a transport problem

Au elaborat: st. gr. TI-207

Bodarev Victor

Bunescu Gabriel

Rusu Cătălin

A verificat: lect. univ. Andrievschi – Bagrin Veronica

Chișinău, 2021

Cuprins

[Introducere 3](#_Toc91083100)

[1. Formularea unei probleme prin abstractizare unei situații. 4](#_Toc91083101)

[2. Definirea datelor problemei și stabilirea relațiilor între date și soluția problemei. 5](#_Toc91083102)

[3. Alegerea algoritmului pentru rezolvarea problemei. 6](#_Toc91083103)

[4. Exemplu de lucru al algoritmului: 7](#_Toc91083104)

[5. Argumentarea alegerii a algoritmului. 15](#_Toc91083105)

[6. Analiza complexității algoritmilor. 16](#_Toc91083106)

[7. Dezvoltarea unei interfețe grafice pentru aplicație software. 17](#_Toc91083107)

[Concluzie: 20](#_Toc91083108)

[Bibliografie: 21](#_Toc91083109)

# Introducere

Procesul dezvoltării unui algoritm, pornind de la definirea unei probleme, impune atât verificarea corectitudinii și analiza detailată a complexității algoritmului, cât și analizarea problemei din perspectivă computațională, în scopul determinării calității procesului de rezolvare.

Etapele menționate, esențiale pentru garantarea calității și performanțelor soluției, constituie obiectivul întregului proiect de curs și care pe parcurs sunt prezentate succint în acest proiect.

Calea de la problemă la algoritm și principalele elemente ce trebuie precizate în proiectul de curs care sunt enumerate mai jos:

* Există o procedură efectivă care calculează soluția problemei?
* Din ce clasă de complexitate face parte problema?
* Algoritmul implementat corespunde specificației problemei?
* Ce cantitate de resurse de calcul, în special spațiu de memorie și timp de calcul, consumat de către algoritm?

Specificarea unei proceduri efective, folosind o notație convențională, conform particularităților computaționale ale unui tip de mașină de calcul este un algoritm.

Un algoritm acceptat de o mașină de calcul ca o mulțime finită de instrucțiuni cu semnificație precisă, care poate fi executat de către mașină de calcul în timp finit pentru a calcula soluția unei probleme reprezentate în mod finit.

## Formularea unei probleme prin abstractizare unei situații.

În fiecare zi noi efectuăm cumpărături online, în același moment zi de zi milioane de oameni fac același lucru, ceea ce înseamnă că fac cumpărături fără a ieși din casă. Acest lucru este foarte simplu și comod pentru consumatori, tot ce este necesar să facem este alegerea produselor dorite , să le achităm și să așteptăm până când ne va fi produsul livrat.

Livrarea produselor se face prin intermediul aerian și terestru, pentru cei care oferă astfel de servicii este cu mult mai complicat. Ca de exemplu se poate de imaginat că o companie care oferă servicii de livrare timp de 24 de ore trebuie să prelucreze și să livreze circa de 50 mii de produse. După ce a fost procesată tranzacția, cererea de procesare a produselor este transmisă către depozit, unde acesta este procesat și transmis către cumpărător.

Dacă să ne imaginăm un depozit are o intrare și o ieșire. La intrare avem marfă nesortată care provine de la furnizor, iar la ieșire avem produsul deja procesat și care este gata pentru livrare către cumpărător. Pentru ca procesul să fie optimizat unele companii utilizează roboți care îndeplinesc careva operații elementare. Spre exemplu pentru produsele noi se efectuează următorii pași:

1. Identificarea produsului.
2. Transportarea produsului către raftul unde va fi depozitat.
3. Plasarea produselor pe rafturi pentru transportare.

O dată cu apariția cereri, se efectuează așa operații ca:

1. Luarea produsului de pe raft pentru transportare.
2. Plasarea produsului la punctul de transport.
3. Transportarea produsului către cumpărător.

Lucrarea dată de an la obiectul Analiza și Proiectarea Algoritmilor are ca scop optimizarea a pasului 3 – transportarea produsului către consumator. Se poate de menționat faptul că acest proces trebuie de realizat cu cheltuieli foarte mici ceea ce ar însemna cheltuieli minime. Luând în considerație că produsele sunt transportate prin diferite medii de transport, noi am decis să optimizăm procesul de transportare a produsului din depozit către consumător prin transportarea lui pe calea cea mai scurtă.

Pentru realizarea acestui proces avem nevoie de a implementa un algoritm care ar găsi această scurtă cale, în dependență de organizare a depozitului. Reieșind din aceste considerente pentru a determina cea mai optimală cale pentru transport noi ne-am propus să implementăm algoritmul de căutarea a drumului minim și anume algoritmul Dijkstra.

## Definirea datelor problemei și stabilirea relațiilor între date și soluția problemei.

Primul pas spre rezolvarea unei probleme de explorare este definirea exactă a problemei propuse. Problema este bine definită dacă cunoaștem 4 parametri:

1. **Starea inițială a problemei** – reprezentarea configurației de plecare.
2. **Funcția de expandare a nodurilor** – în caz general este o listă de perechi (acțiune, stare, rezultat). Astfel, pentru fiecare stare se enumeră toate acțiunile posibile precum și starea care va rezulta în urmă aplicării respectivei acțiuni.
3. **Predicarea pentru starea finală** – funcție care returnează *true* dacă o stare este starea de scop și *false* altfel.
4. **Funcția de cost** – atribuie o valoare numerică fiecărei căi generate în procesul de exploatație a acestuia. De obicei se folosește o funcție de cost pentru fiecare acțiune sau tranziție, atribuim astfel, o valoare fiecărui arc din graf.

Sarcina algoritmilor de căutare este de a găsi o cale din stare inițială într-un scop. Dacă algoritmul găsește o soluție atunci când mulțimea soluțiilor este nevidă putem spune că algoritmul este **complet**. Dacă algoritmul găsește și calea de cost minim către starea finală spunem că algoritmul este **optim**.

Algoritmii euristici de căutare sunt algoritmi care lucrează pe grafuri, și folosesc informație suplimentară, necontinuă în definirea problemei, prin care se efectuează procesul de găsire a unei soluții.

În cadrul explorării stărilor fiecare algoritm generează un arbore, care în rădăcină va conține starea inițială. Fiecare nod al arborelui va conține următoarele informații:

1. **Starea continuu** – stare (nod)
2. **Părintele nod** – π (nod)
3. **Costul** – costul drumului de la starea inițială până la nod – *g*

De asemenea, pentru fiecare nod definim o funcție de evaluare f care indică cât de promițător este un nod în perspectiva găsirii unui drum către soluție (dacă f este mai mic cu atât drumul este mai favorabil).

**Datele de intrare** sunt valorile pe care un program care le primește dintr-o sursă oarecare. În cazul nostru ca date de intrare servește un grid, din care inițial poate fi citită stările inițiale (nodul de plecare), starea finală (nodul destinației) și nodurile ce reprezintă obstacole.

**Datele de ieșire** reprezintă valorile pe care programul le va genera ca urmare a executării instrucțiunilor sale și le transmite utilizatorilor. Algoritmul Dijkstra care a fost implementat pe parcurs furnizează ieșirea drumului minim, sub forma unei liste în care se păstrează coordonatele nodurilor care alcătuiesc acest drum în grid-ul de intrare.

## Alegerea algoritmului pentru rezolvarea problemei.

Găsirea drumului minim este o problemă des întâlnită în programare, în special în dezvoltarea jocurilor și a inteligenței artificiale. Un algoritm este un șir de pași care transformă datele de intrare în date de ieșire, este un instrument de rezolvare a problemelor de calcul bine definite.

În continuare vom analiza algoritmul ales pentru realizarea acestui proiect și soluționarea problemei noastre:

* Algoritmul Dijkstra.

Algoritmul Dijkstra pornește de la nodul de start și verifică pe toți vecinii săi în scopul determinării celui mai aproape (distanța până la un vecin să fie minimă). După identificarea acestui nod, algoritmul se mută la el, repetând aceeași pași, până când algoritmul atinge nodul de destinație. Complexitatea a algoritmului Dijkstra pe care noi l-am implementat este O (E log V), ea este datorată integrării a unei stucturi de date ca „Binary heap”, care permite accelerarea accesării datelor.

**Algoritmul:**

1. Se creează o listă cu distanțe, o listă cu noduri anterioare, o listă cu nodurile vizitate și nod curent.
2. Toate valorile din listă cu distanțe care sunt inițializate cu valoarea infinită, cu excepția nodului de start, unde este setat valoarea 0.
3. Toate valorile din listă cu nodurile vizitate sunt setate false ceea ce înseamnă că nu au fost vizitate.
4. Toate valorile din listă cu nodurile anterioare au fost inițializate cu valoarea -1.
5. Nodul start este setat ca un nod curent.
6. Nodul curent se va marca ca nod vizitat.
7. Distanțele se actualizează distanțele, ca bază se iau nodurile care pot fi vizitate din nodul curent.
8. Se actualizează nodul curent la nodul nevizitat care pot fi vizitate prin calea cea mai scurtă de la nodul start.
9. Se efectuează repetarea de la punctul 6 până când toate nodurile vor fi vizitate.

## Exemplu de lucru al algoritmului:

Avem de exemplu un grid de mărimea 5x5, în total vom avea 25 de noduri în interiorul căruia sunt 3 noduri care sunt setate ca obstacole. Scopul este de a găsi cel mai scurt drum pornind din nodul care are coordonatele (0, 0) care reprezintă nodul start și când va ajunge în nodul final va avea coordonatele (4, 2).

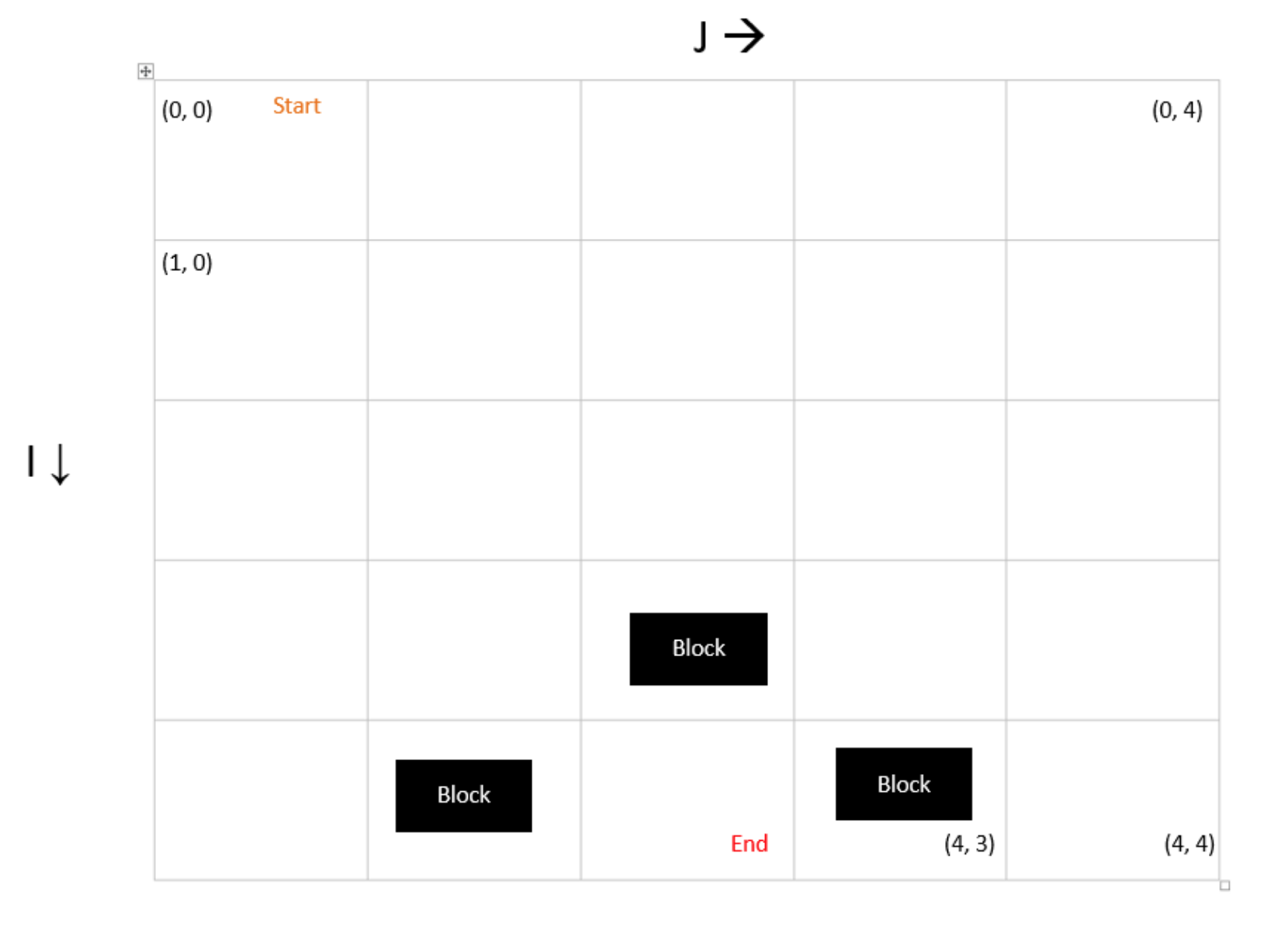
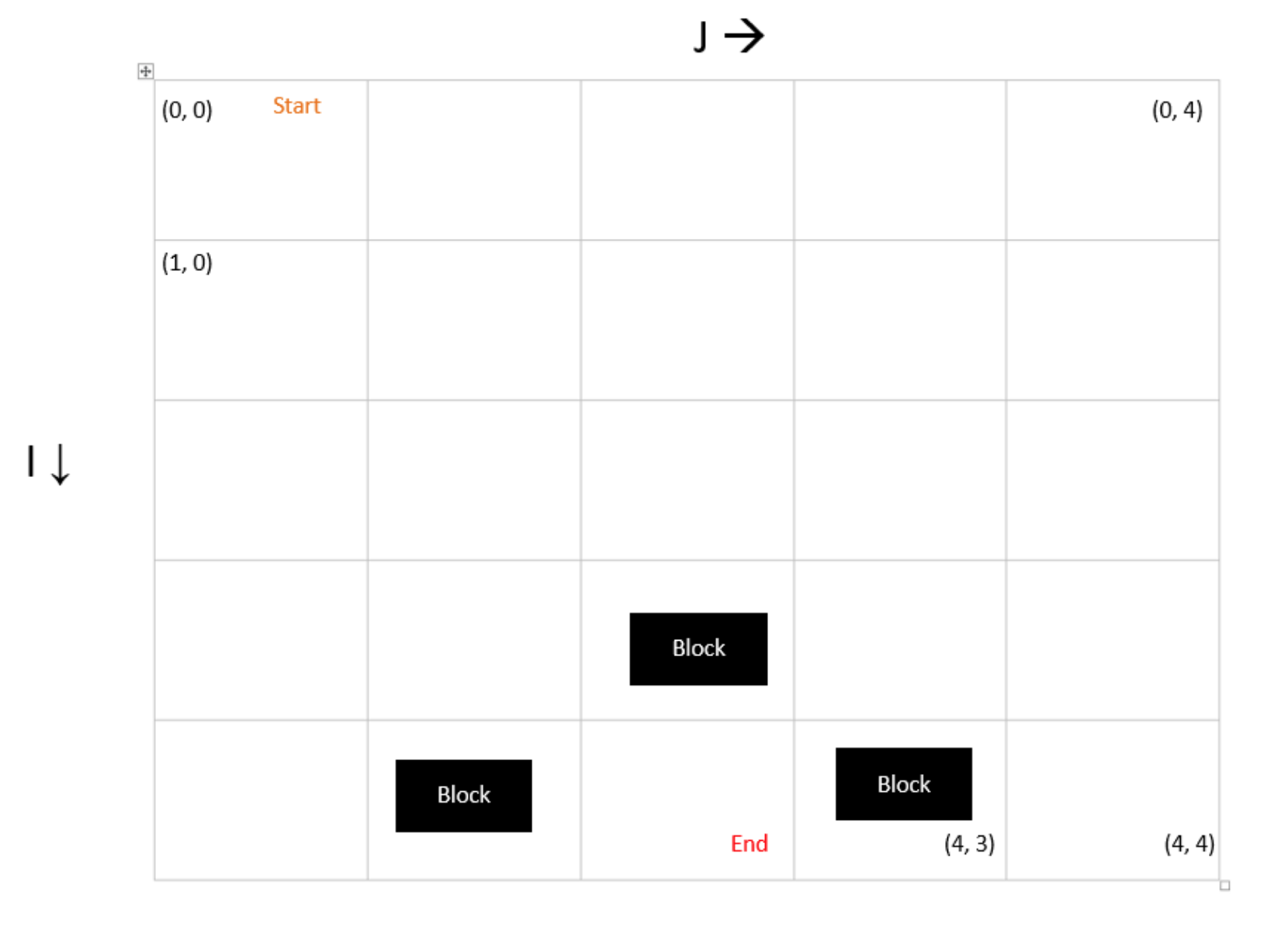


Figura 1.1. Exemplu de lucru al algoritmului.

În exemplul de mai sus sunt folosite distanțe euclidiene. Atunci, dacă ne mișcăm pe spațiul vertical sau orizontal nodul va avea valoarea 1, iar dacă ne vom mișca pe căile de pe diagonală atunci nodurile vor avea valoarea 1 și 4. De la început vom defini distanța de start ca fiind egală cu 0, iar restul distanțelor către restul nodurilor să fie infinit.



Infinity

Distanta 1.4

Infinit

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Figura 1.2 Exemplu de lucru al algoritmului.

Pornind de la nodul de start setăm distanțele minime pentru nodurile învecinate. Distanța pe orizontală este 1, iar întrucât 1 este mai mic decât ∞, valoarea setată este 1 pentru nodurile la care putem atinge pe verticală sau orizontală. Distanța pe diagonală este 1 și 4, iar întrucât 1.4 este mai mic decât ∞, iarăși setăm valoarea 1.4 pentru nodurile pe care noi le putem atinge pe diagonală. Între timp noi adăugăm nodurile care sunt vizitate într-o coadă de prioritate, prioritar de către valoarea cea mai mică a distanței. La fel noi definim părinți pentru fiecare nod din care se va lua valoarea minimă.

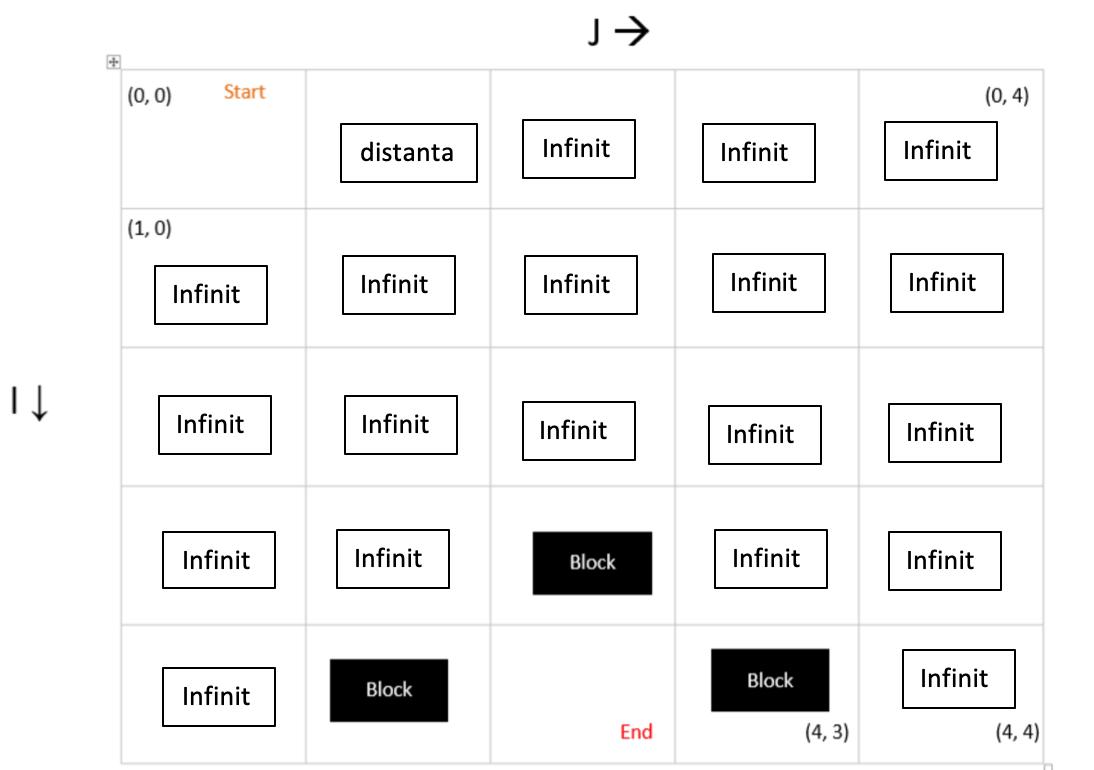
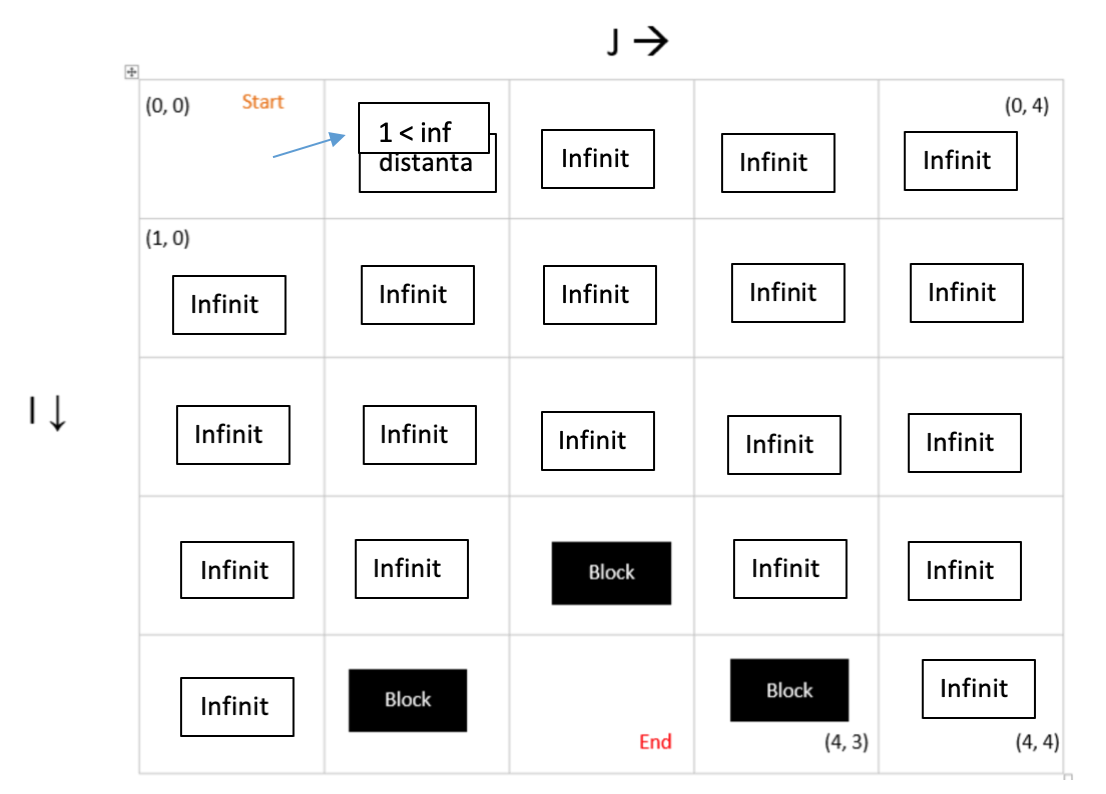


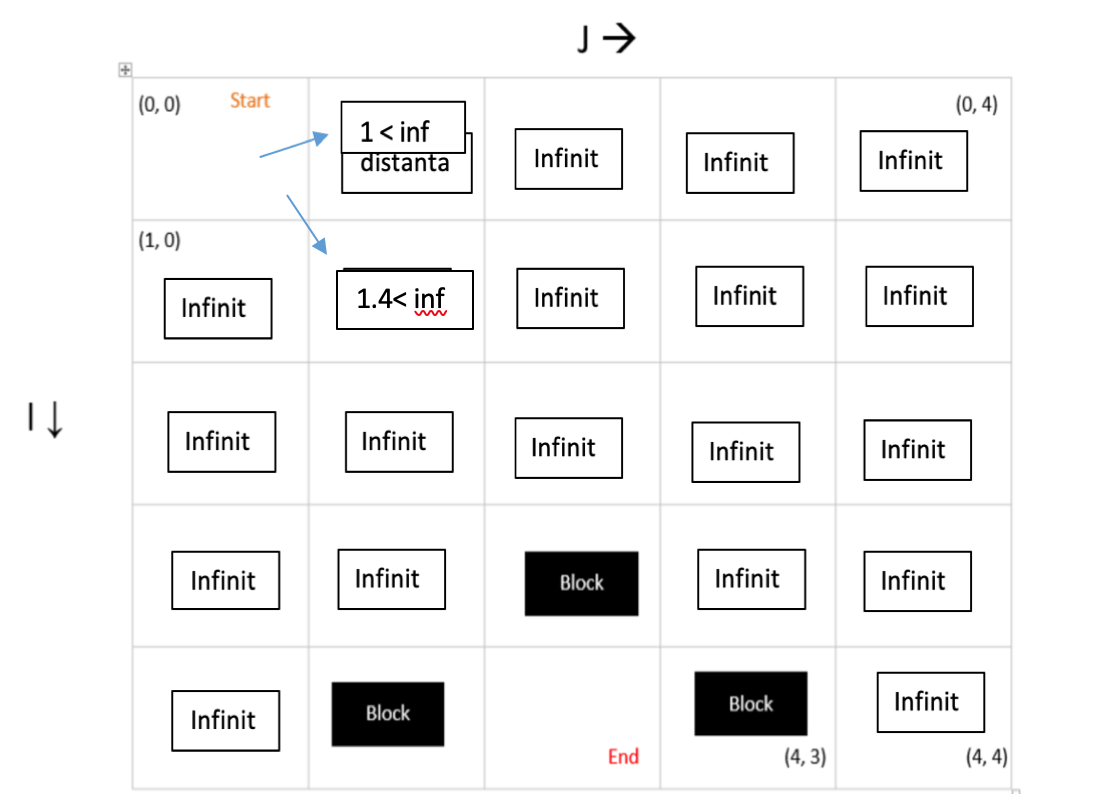
Figura 1.3 Exemplu de lucru al algoritmului.



1 < inf

1.4< inf

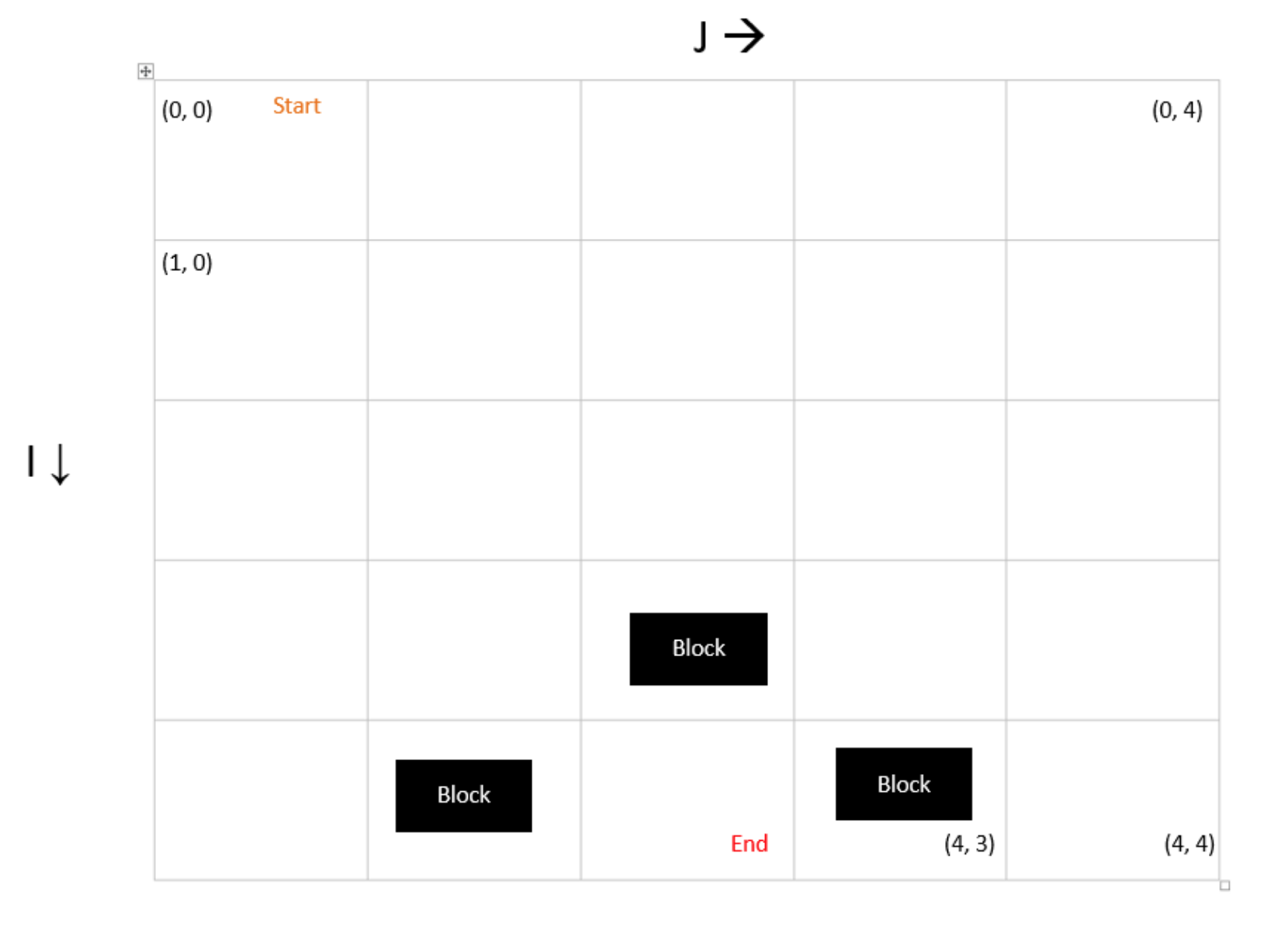
Figura 1.4 Exemplu de lucru al algoritmului



1<inf

Figura 1.5 Exemplu de lucru al algoritmului.

După ce am găsit toate distanțele către nodurile învecinate nodul start, el va fi marcat ca nod vizitat, pentru a nu a mai fi verificat.



Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Dist.1

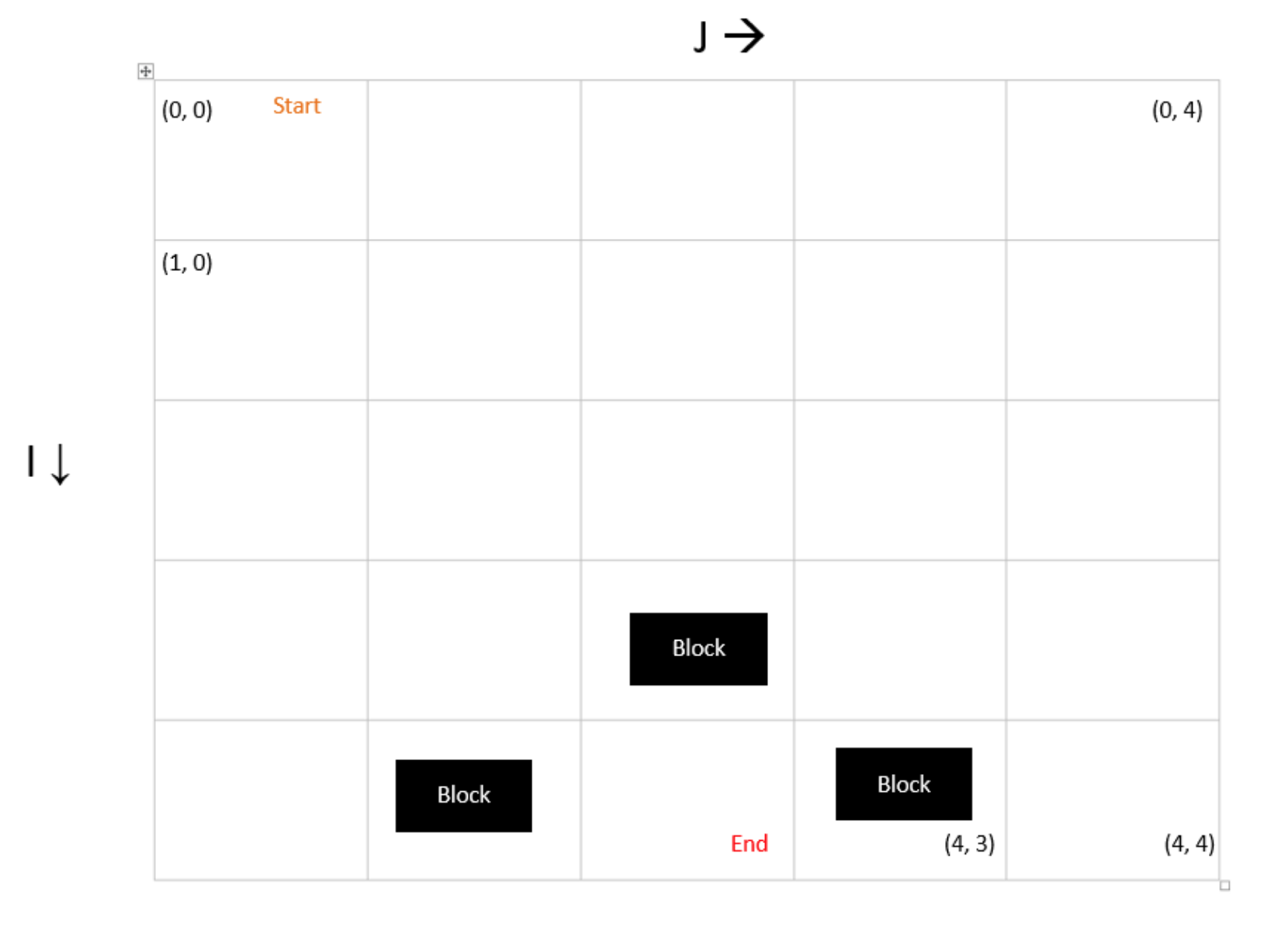
Dist.1.4

Dist.1

vizitat

Figura 1.6 Exemplu de lucru a algoritmului

După pasul dat am extras nodul din topul cozii de prioritate, și vom modifica toate valorile învecinate nodului respectiv. Aici adăugăm distanța către nodul care este setat ca nod curent la valorile găsite anterior pentru noduri.



Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Infinity

Dist.1

2.4<1

Dist.1.4

2<1.4

Dist.2.4

2<inf

Dist.2

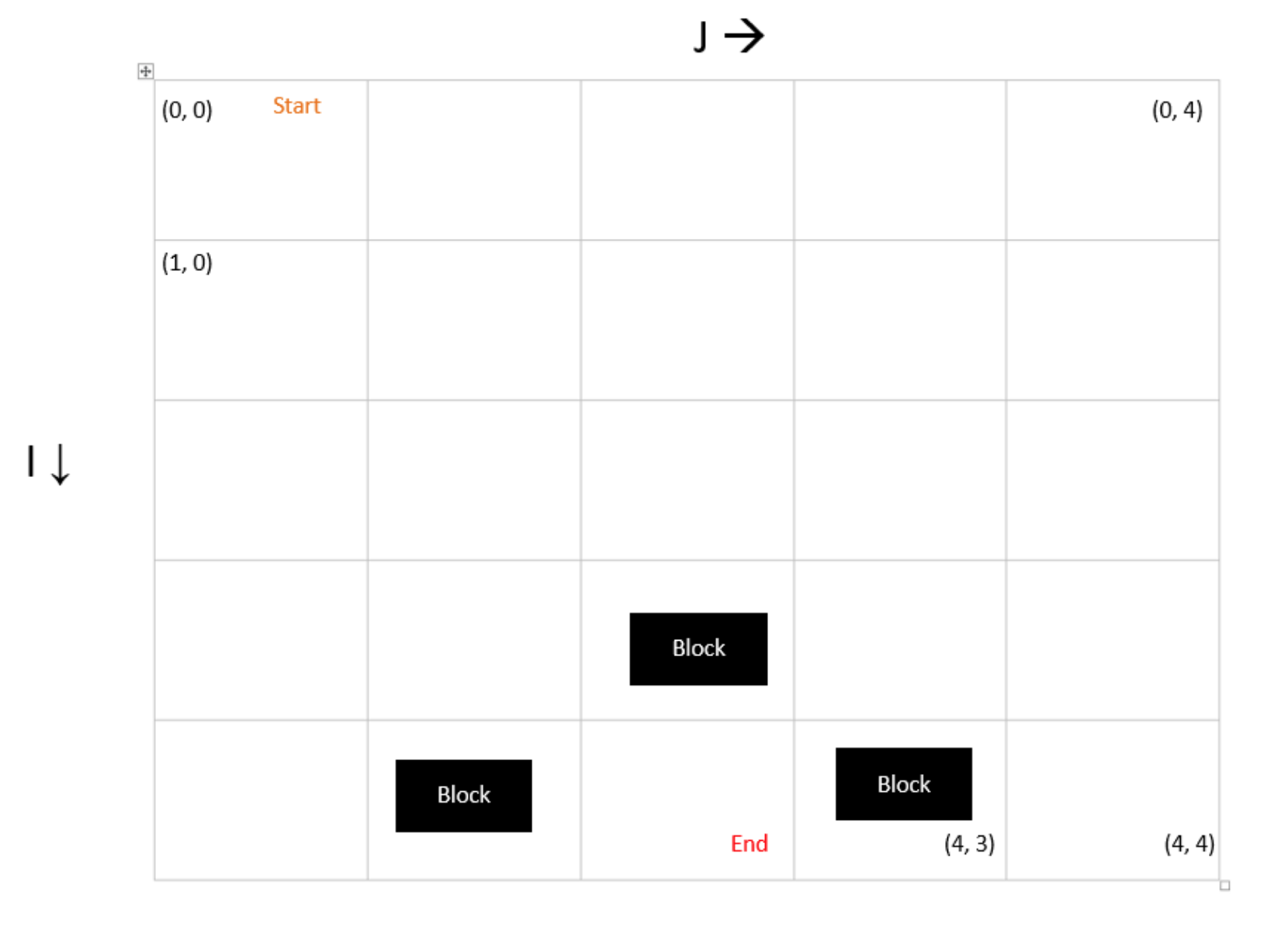
2<inf

Dist.1

vizitat

Figura 1.7 Exemplu de lucru al algoritmului.

La sfârșit celui de al doilea nod avem valorile și părinți următori:



Dist.2

Dist.2.4

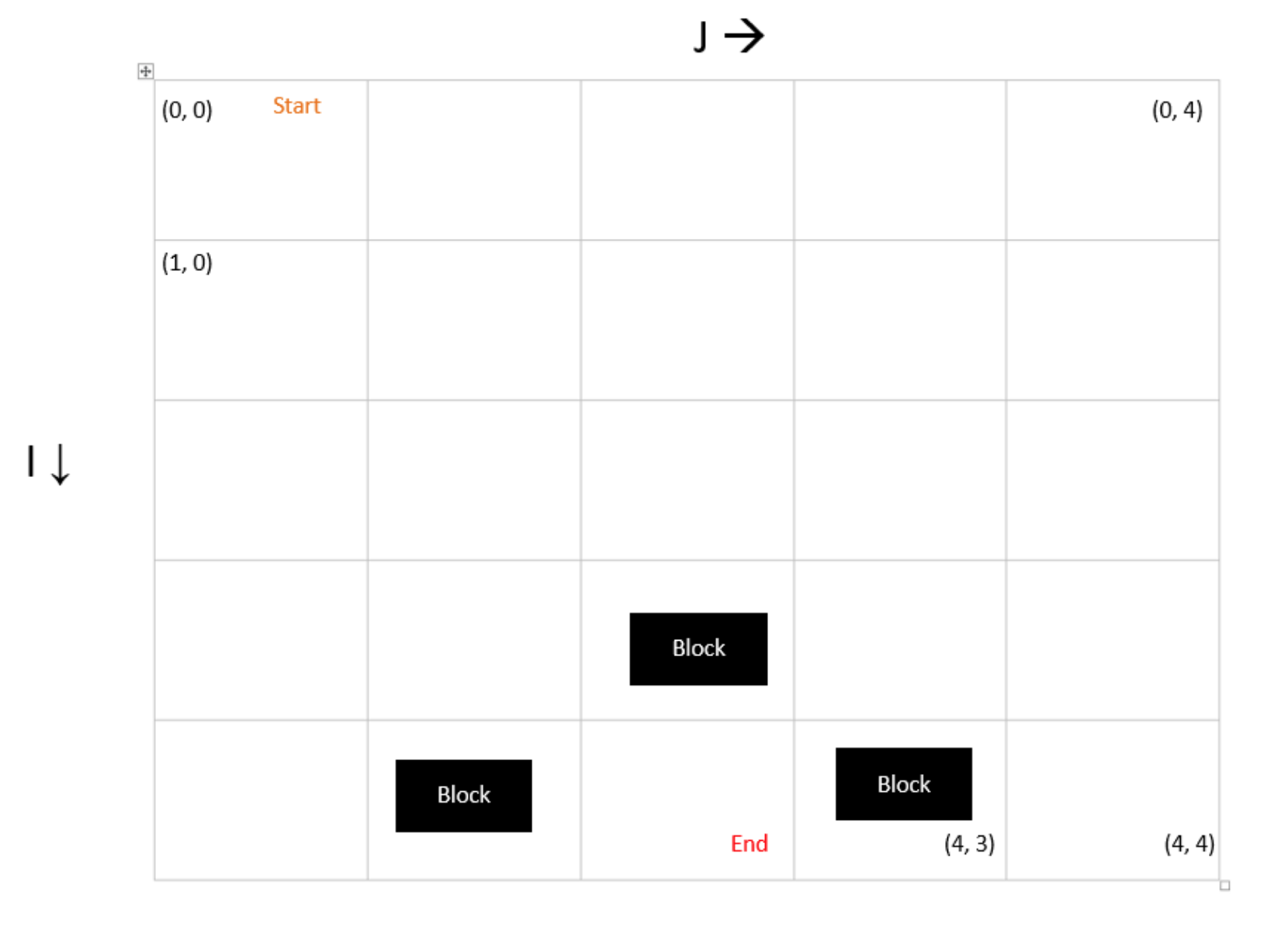
Dist.1.4

Dist.1

vizitat

vizitat

Figura 1.8 Exemplu de lucru al algoritmului.

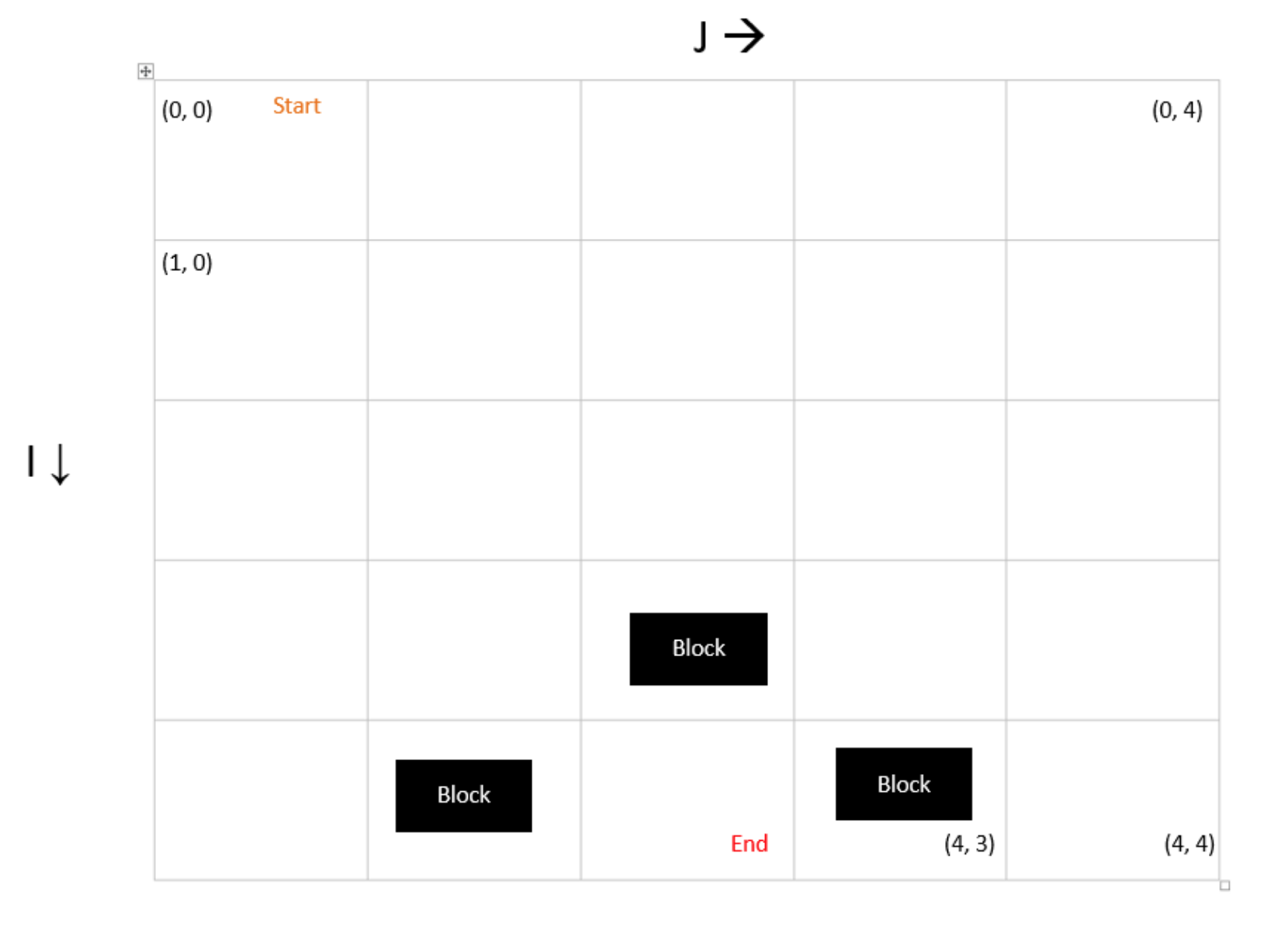


vizitat

vizitat

vizitat

Figura 1.9 Exemplu de lucru al algoritmului.



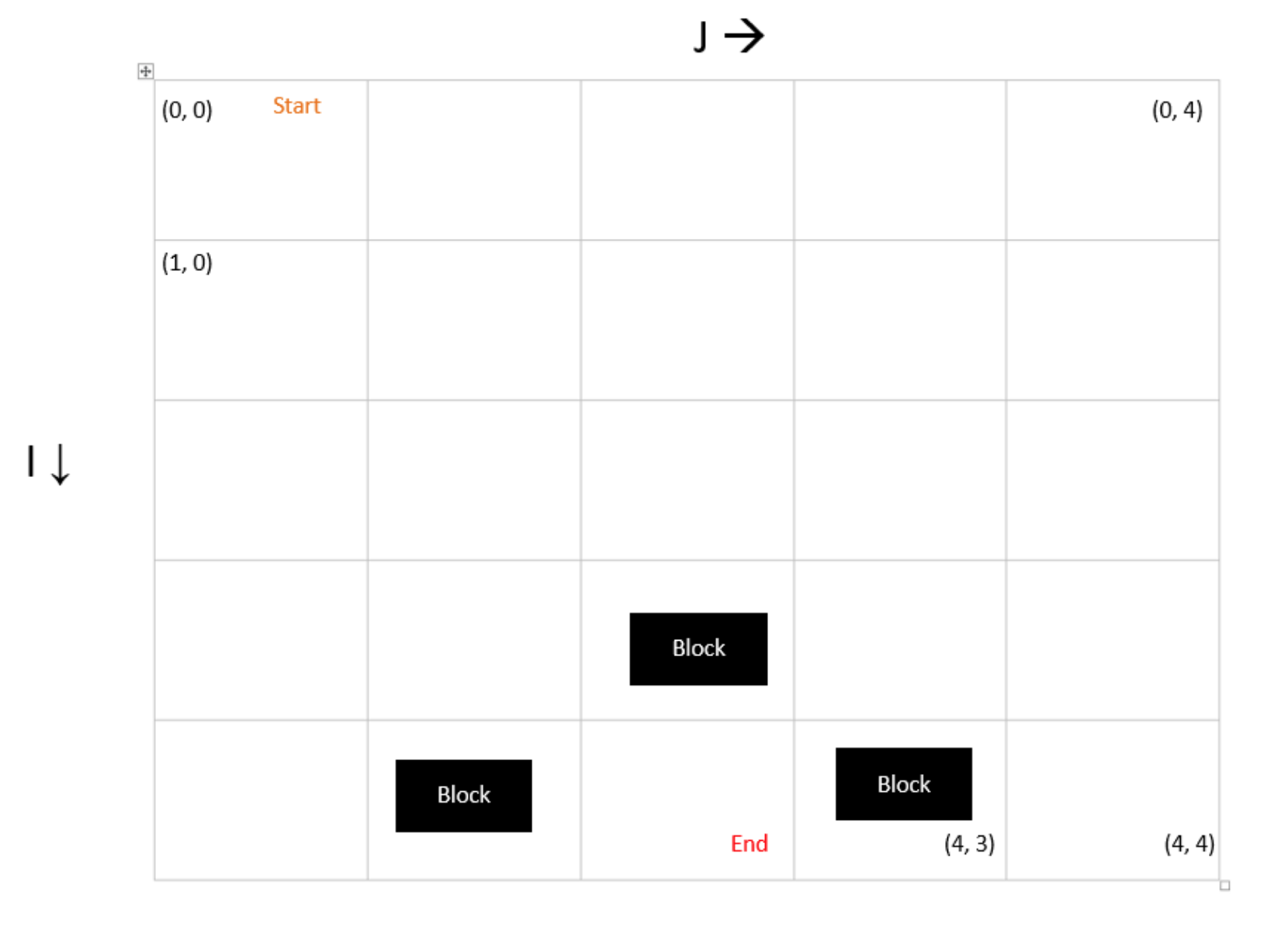
vizitat

vizitat

vizitat

vizitat

Figura 1.10 Exemplu de lucru al algoritmului.



vizitat

vizitat

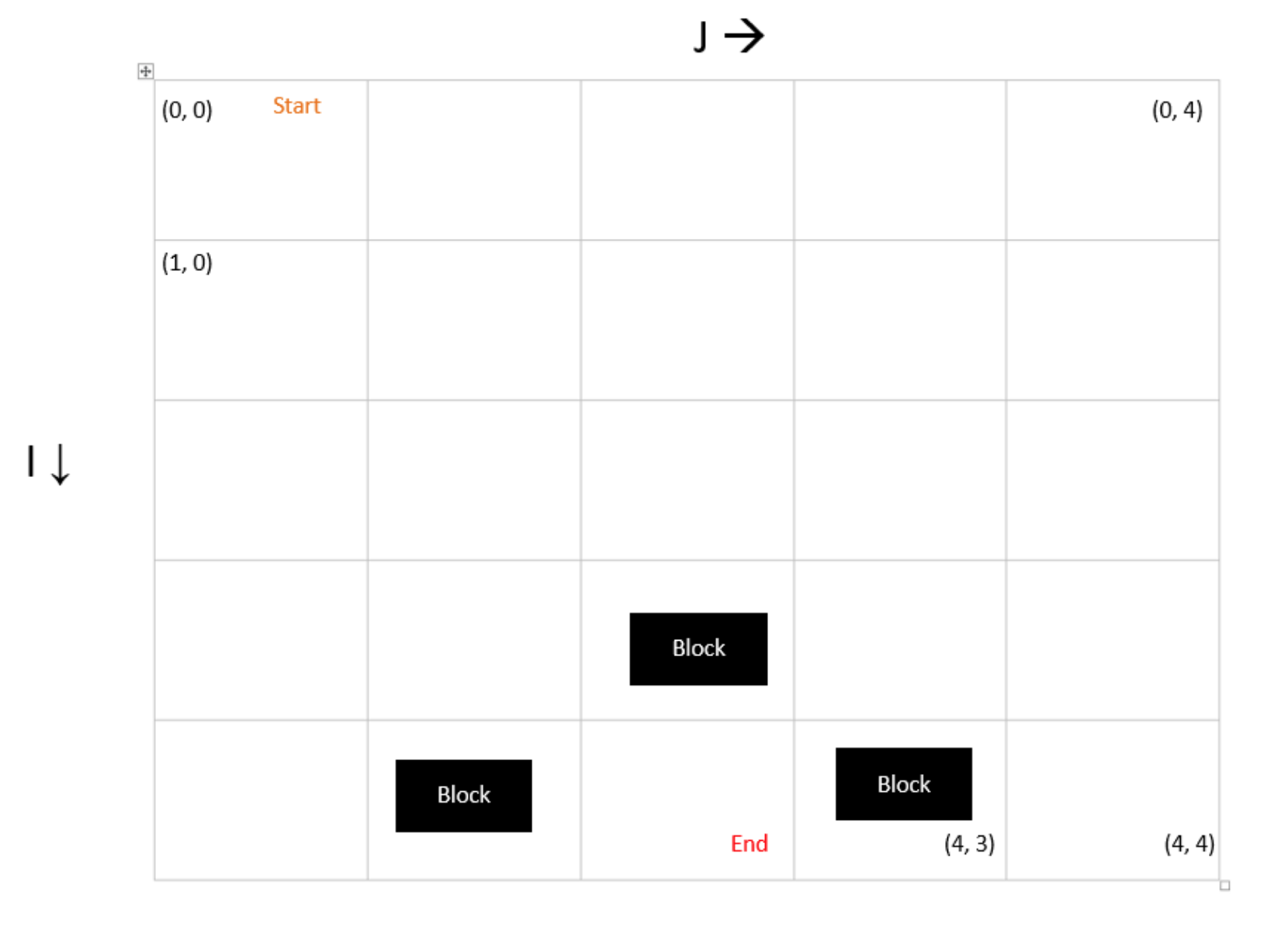
vizitat

vizitat

vizitat

vizitat

Figura 1.11 Exemplu de lucru al algoritmului.



vizitat

vizitat

vizitat

vizitat

vizitat

vizitat

vizitat

vizitat

vizitat

vizitat

vizitat

Figura 1.12 Exemplu de lucru al algoritmului.

În final noi observăm că după ce algoritmul folosit a ajuns la nodul de destinație, urmărind părinții nodurilor care deja au fost vizitate, se construiește drumul de la nodul destinației către nodul de start, unde deja distanța a fost găsită.

Rezultatul algoritmului după ce a fost găsit drumul cel mai scurt se poate de analizat în figura 1.13.



Figura 1.13 Rezultatul final după efectuarea lucrului a algoritmului.

## Argumentarea alegerii a algoritmului.

Algoritmii pentru determinarea drumului minim sunt algoritmii care au multiple aplicații în practică și pot reprezenta clasa de algoritmi bazat pe grafuri cel mai de ele sunt utilizate pentru:

1. Rutarea în cadrul unei rețele (de calculatoare etc).
2. Stabilirea unei agende de zbor.
3. Găsirea a drumului minim dintre două locații (Google Maps, GPS).
4. Asigurarea unui server de fișiere în funcție de matricele definite pe fiecare linie de comunicație.
5. Pentru jocuri.

Din multitudinea de algoritmi prin care am putea soluționa problema care este propusă spre rezolvare la un moment noi am ajuns să la ceea că a fost necesar pentru a alege un algoritm efectiv am avut de ales între doi algoritmi anume A\* și Dijkstra, din algoritmi dați noi am ales algoritmul Dijkstra.

În continuare vom prezenta avantajele și dezavantajele utilizării fiecăruia dintre ei.

1. Algoritmul A\*.

Complexitate: O(|E|)

1. Avantaje:

* Este un algoritm euristic – A\* se extinde asupra unui nod dor în cazul în care acesta este promițător. Algoritmul pune accentul pe faptul de a găsi nodul destinație cât mai repede posibil.
* Este complet – A\* este un algoritm complet ceea ce înseamnă că el va găsi mereu o soluție, nu neapărat optimală dacă aceasta există.
* Poate fi transformat în alți algoritmi – A\* poate fi transformat ușor în alt algoritm de găsire a drumului minim, modificând funcția euristică de evaluare a nodurilor.

1. Dezavantaje:

* Nu este eficient pentru cazul când avem mai multe noduri țintă – dacă avem mai multe noduri țintă și nu se știe care este cel mai apropiat de cel principal A\* nu este foarte optim. Acest lucru se datorează faptului că trebuie să fie

rulat de mai multe ori (o dată pentru fiecare nod țintă) pentru a ajunge la toate.

1. Algoritmul Dijkstra.

Complexitate: O (E log V).

1. Avantaje:

* Este un algoritm neinformat – Dijkstra este un algoritm neinformat. Aceasta înseamnă că nu este necesar să cunoaștem nodul țintă în prealabil.
* Este bun pentru cazul în care avem mai multe noduri țintă – Deoarece Dijkstra preia arcele cu cel mai mic cost la fiecare pas, acoperă de obicei o suprafață mare a grafului. Acest lucru este util în special atunci când avem mai multe noduri țintă, dar nu știm care dintre acestea este cel mai apropiat.

1. Dezavantaje:

* Eșuează pentru arce cu ponderea negativă – algoritmul Dijkstra nu poate evalua arce cu greutăți negative.

Concluzionând putem afirma că pentru problema propusă algoritmul A\* este unul mai optimal, dar Dijkstra ar fi un algoritm mai universal și mai ușor pentru înțelegere.

## Analiza complexității algoritmilor.

Timpul de execuție:

La algoritmul Dijkstra o dată cu creșterea mărimii a problemei, timpul de execuție crește liniar. În medie pentru un grid cu mărimea de exemplu 1024 x 1024 de noduri, algoritmul furnizează cel mai scurt drum în 2,3 secunde. Mai jos putem să analizăm un tabel unde ne arată timpul de execuție pentru diferite mărimi ale grid-ului.

Tabelul 1. Timpul de execuție pentru algoritmul Dijkstra.

|  |  |
| --- | --- |
| Mărime grid (noduri) | Timpul de execuție (ms) |
| 64x64 | 6 |
| 128x128 | 25 |
| 256x256 | 120 |
| 512x512 | 515 |
| 1024x1024 | 2362 |

În continuare mai jos este prezentat un grafic care ne arată timpul de execuție a algoritmului Dijkstra.

Figura 2 Timpul de execuție a algoritmului Dijkstra

## Dezvoltarea unei interfețe grafice pentru aplicație software.

Pentru a elabora algoritmul lui Dijkstra am decis să folosim **HyperText Markup Language** (**HTML**), ca baza pentru crearea unei Web pagini, și care  este o formă de marcare orientată către prezentarea documentelor text pe o singura pagină, utilizând un software de redare specializat, numit agent utilizator HTML, cel mai bun exemplu de astfel de software fiind browserul web. HTML furnizează mijloacele prin care conținutul unui document poate fi adnotat cu diverse tipuri de metadate și indicații de redare. Indicațiile de redare pot varia de la decorațiuni minore ale textului, cum ar fi specificarea faptului că un anumit cuvânt trebuie subliniat sau că o imagine trebuie introdusă, până la scripturi sofisticate, hărți de imagini și formulare. Metadatele pot include informații despre titlul și autorul documentului, informații structurale despre cum este împărțit documentul în diferite segmente, paragrafe, liste, titluri etc. și informații cruciale care permit ca documentul să poată fi legat de alte documente pentru a forma astfel [hiperlink](https://ro.wikipedia.org/wiki/Hiperlink)-uri (sau [web](https://ro.wikipedia.org/wiki/Web)-ul). În acesta am inclus o colecție de clase **CSS** predefinite, care le-am utilizat pentru formatarea elementelor documentului [HTML](https://ro.wikipedia.org/wiki/HTML). Colecția dată de clase este descrisă în framework-ul **MaterializeCSS**, care este creată și proiectat de Google, Material Design fiind un limbaj de design care combină principiile clasice ale designului de succes împreună cu inovația și tehnologia. Scopul Google este să dezvolte un sistem de design care să permită o experiență unificată a utilizatorului pentru toate produsele pe orice platformă.

Ca backend, adică algoritmizarea paginii Web, descrierea structurii de date, și a datelor la general am folosit **JavaScript**, care este un limbaj structurat, de programare orientată pe obiecte, bazat pe un stil de programare orientată pe obiecte în care reutilizarea comportamentului (cunoscută sub numele de moștenire ) se realizează printr-un proces de reutilizare a obiectelor existente care servesc drept prototipuri . Acest model poate fi cunoscut și ca programare prototip , orientată pe prototip, fără clasă sau bazată pe instanțe .

În ciuda numelui și a unor similarități în sintaxă, între JavaScript și limbajul Java nu există nicio legătură. Ca și [Java](https://ro.wikipedia.org/wiki/Java), JavaScript are o sintaxă apropiată de cea a [limbajului C](https://ro.wikipedia.org/wiki/Limbajul_de_programare_C), dar are mai multe în comun cu [limbajul Self](https://ro.wikipedia.org/w/index.php?title=Limbajul_Self&action=edit&redlink=1) decât cu [Java](https://ro.wikipedia.org/wiki/Limbajul_Java).

Una din caracteristicile principale a acestui limbaj de programare este declararea variabilelor fără indicare tipului de date, acesta fiind inițializat automat. Programatorul indicând doar capacitatea acestuia de redeclarare, redifinire sau tipul de amploare(global, local).

Pentru accelerarea încărcării paginii, și ușurarea procesului de interacționare între mai multe fișiere JavaScript, am utilizat WebPack, care este un pachet de module static pentru aplicațiile JavaScript moderne, care procesează aplicația, și construiește intern un grafic de dependență din unul sau mai multe puncte de intrare(interacționarea fișierelor .js) și apoi combină fiecare modul de care are nevoie proiectul dvs. într-unul sau mai multe pachete, care sunt active statice din care difuzează conținutul. Generalizând, acest package colectează toate datele și metodele într-un fișier care bazic este numit bundle, care stabilește relațiile dintre fișiere, astfel putem apela doar un fișier .js, fără apariția ambiguităților de importare.

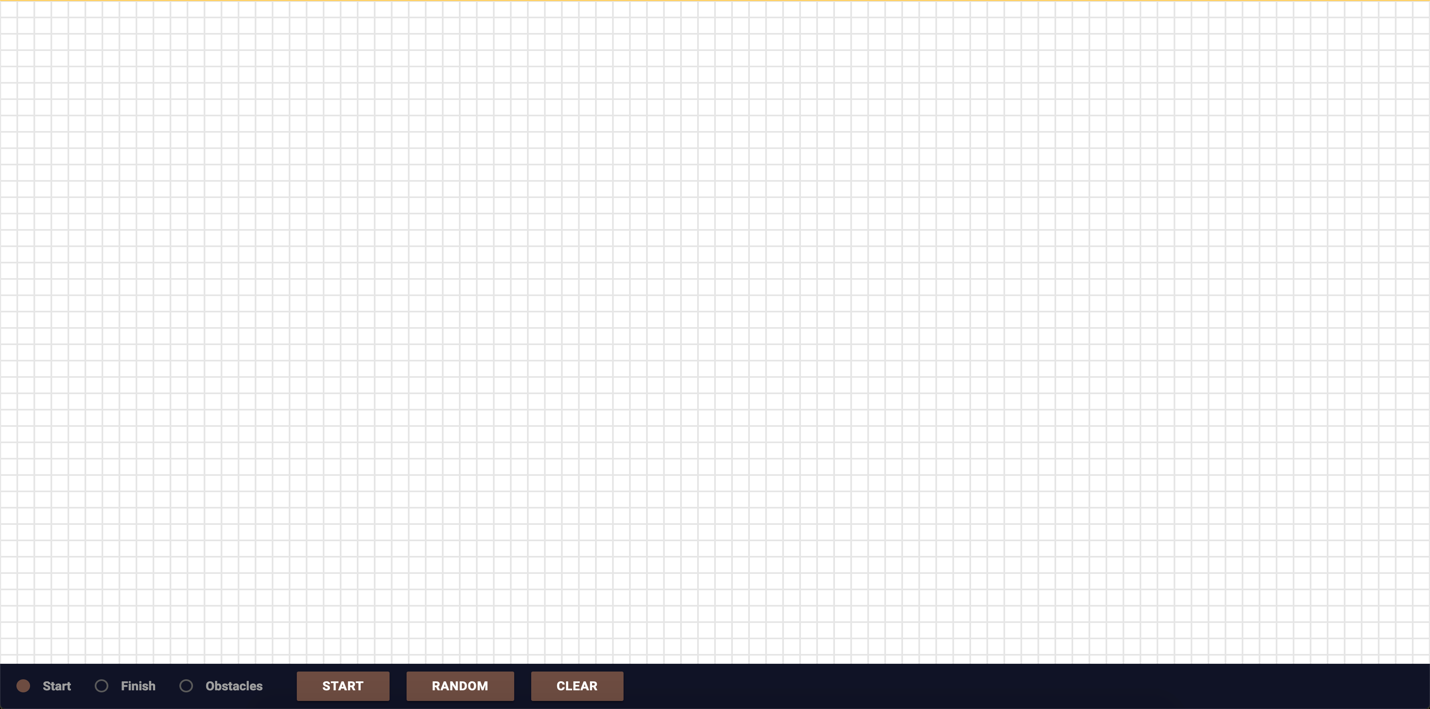


Figura 3 Interfața algoritmului.



Figura 4 Exemplu de lucru al algoritmului.

## Concluzie:

Pe parcursul elaborării acestui proiect de an noi am studiat aprofundat despre algoritmul Dijkstra, comparândul analitic cu algoritmul A\* pe parcursul elaborării planului pentru realizarea a acestui proiect.

Primul pas a fost formularea problemei care pe parcurs va fi rezolvată de către proiectul nostru, următorul nostru scop a fost definirea datelor problemei formulate și stabilirea relației dintre datele problemei și soluția acesteia, următorul pas a fost alegerea algoritmului pentru soluționarea problemei date, cum a fost descris în raport noi am avut de ale dintre 2 algoritmi, Dijkstra și A\*, astfel după analiza fiecărui algoritm, stabilirea avantajelor și dezavantajelor lor, noi am ales algoritmul Dijkstra, pentru că ar fi un algoritm mai universal și mai simplist pentru înțelegerea lui.

Comparând acești 2 algoritmi și alegândul pe cel al lui Dijkstra, am decis să soluționăm problema de bază a algoritmului A\*, și anume identificarea drumului minim dintre 2 puncte distincte.

După efectuarea a acestor scopuri noi am început implementarea algoritmului întru-un limbaj de programare, și acesta a fost limbajul de programare JavaScript, după implementare am trecut la dezvoltarea a interfeței grafice pentru aplicația noastră ca instrumente pentru realizarea a acestui pas am folosit același JavaScript, împreună cu HTML și CSS.

Pe parcursul implimentării aplicației descrise mai sus, am obținut mai multe fișiere .js, pentru interacționarea rapidă și eficientă a cărora am utilizat utilita WebPack. Iar pentru a ușura procesul de stilizare a interfeței grafice am utilizat o adunare de clase predifinite.

## Bibliografie:

1. <https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm>
2. [https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)
3. <https://brilliant.org/wiki/dijkstras-short-path-finder/>
4. <https://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/>
5. [Binary heap - Wikipedia](https://en.wikipedia.org/wiki/Binary_heap#:~:text=A%20binary%20heap%20is%20a,way%20of%20implementing%20priority%20queues.&text=Heap%20property%3A%20the%20key%20stored,according%20to%20some%20total%20order.)
6. [JavaScript | MDN (mozilla.org)](https://developer.mozilla.org/ru/docs/Web/JavaScript)
7. [HTML | MDN (mozilla.org)](https://developer.mozilla.org/ru/docs/Web/HTML)
8. <https://medium.com/basecs/finding-the-shortest-path-with-a-little-help-from-dijkstra-613149fbdc8e>
9. <https://brilliant.org/wiki/a-star-search/>
10. <https://levelup.gitconnected.com/dijkstras-shortest-path-algorithm-in-a-grid-eb505eb3a290>
11. <https://stackoverflow.com/questions/26126088/using-dijkstras-algorithm-for-pathfinding-on-a-grid>
12. <https://www.quora.com/What-are-advantages-and-disadvantages-of-Dijkstras-algorithm-in-data-structures>
13. <https://materializecss.com/>
14. <https://webpack.js.org/guides/>