

Razvoj suca temeljenog na umjetnoj inteligenciji za igru Boggle

Buhiniček, Filip

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: <https://urn.nsk.hr/urn:nbn:hr:168:504839>

Rights / Prava: [In copyright](#)/[Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2025-03-21**



Repository / Repozitorij:

[FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory](#)



SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1663

**RAZVOJ SUČA TEMELJENOG NA UMJETNOJ
INTELIGENCIJI ZA IGRU BOGGLE**

Filip Buhiniček

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1663

**RAZVOJ SUČA TEMELJENOG NA UMJETNOJ
INTELIGENCIJI ZA IGRO BOGGLE**

Filip Buhiniček

Zagreb, lipanj 2024.

ZAVRŠNI ZADATAK br. 1663

Pristupnik: **Filip Buhiniček (0036539445)**
Studij: Elektrotehnika i informacijska tehnologija i Računarstvo
Modul: Računarstvo
Mentor: prof. dr. sc. Ivica Botički

Zadatak: **Razvoj suca temeljenog na umjetnoj inteligenciji za igru Boggle**

Opis zadatka:

U okviru završnog rada potrebno je implementirati programskog robota koji će djelovati kao sudac u društvenoj igri Boggle. Potrebno je izraditi heuristički algoritam pronalaska valjanih riječi u rječniku. Nakon izrade algoritma, potrebno ga je optimizirati i testirati kako bi se osiguralo da pronalazi sve riječi na ploči igre Boggle. Koristeći navedeni algoritam potrebno je trenirati programskog robota za ulogu suca u igri. On treba na brz, učinkovit i efikasan način pronaći riječi koje se nalaze na ploči, usporediti ih s riječima iz rječnika te izdvojiti samo valjane riječi. U sklopu rada potrebno je izvršiti i testiranje algoritma u cilju ocjene performansi.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

1. Uvod	1
2. Teorijska pozadina.....	2
2.1. Umjetna inteligencija.....	3
2.1.1. Kratka povijest umjetne inteligencije	3
2.1.2. Umjetna inteligencija u društvenim igrama	4
2.2. Društvena igra Boggle	5
2.2.1. Pravila i mehanika igre Boggle	5
2.2.2. Izazovi u pronalaženju riječi.....	5
2.2.3. Programska rješenja na temu igre Boggle	6
2.3. Algoritmi	7
2.3.1. Povijest algoritama	7
2.3.2. Vrednovanje uspješnosti algoritama.....	8
2.3.3. Vrste algoritama	9
2.4. Trie strukture	11
2.4.1. Primjena trie struktura	11
3. Analiza algoritama.....	12
3.1. Početni algoritam.....	13
3.2. Algoritam nastao djelomičnom optimizacijom	15
3.3. Optimizirani algoritam	17
3.4. Sažetak analize algoritama	18
4. Analiza grafova.....	19
4.1. Usporedba algoritama nad pločom 4x4	19
4.2. Usporedba algoritama kroz razne dimenzije ploče.....	21
4.3. Sažetak analize grafova	23

5. Demo korištenja djelomično optimiziranog i optimiziranog algoritma na ploči igre Boggle dimenzija 4x4.....	24
6. Zaključak	28
Literatura	29
Popis Slika	31
Sažetak.....	32
Summary.....	33

1. Uvod

Razvoj umjetne inteligencije i strojnog učenja omogućio je primjenu sofisticiranih algoritama u raznim područjima kao što su zdravstvo, edukacija, automobilska industrija, marketing, financije, ali i u području društvenih igara [1]. U okviru ovog završnog rada fokusirat ćemo se na razvoj algoritma koji će djelovati kao sudac u društvenoj igri Boggle. Cilj je izraditi algoritam koji može brzo te efikasno pronaći sve valjanje riječi na ploči igre, validirajući ih prema unaprijed definiranom rječniku.

Prvi korak u ovom procesu bio je pronaći rječnik koji će se koristiti za validaciju pronađenih riječi tijekom pretraživanja ploče. Zatim smo izradili početni algoritam koji pretražuje sve moguće opcije na ploči, kako bismo mogli identificirati njegove mane i krenuti u daljnju optimizaciju.

Sljedeći korak bio je iterativni proces optimizacije početno algoritma kako bi se postigla što veća brzinu pretraživanja te istovremeno skratili memorijski zahtjevi. Ovo ćemo postići koristeći strukture podataka predviđene za traženje podudaranja među riječima, a to su Trie strukture, te koristiti heurističke pristupe poput podrezivanja grana.

Kako bismo procijenili performanse konačno razvijenog algoritma, temeljito ćemo ga testirati uspoređujući ga s početnim algoritmom te algoritmom nastalim usred optimizacije. Analize i testiranje provesti ćemo i nad pločama raznih dimenzija kako bi osigurali skalabilnost te pouzdanost neovisno o raznim utjecajima igre i ploče.

Cilj ovog rada je pružanje uvida u proces razvoja i implementacije umjetne inteligencije za specifične zadatke kao što je pronalaženje valjanih riječi u igri Boggle. Rad će također demonstrirati praktične aspekte primjene heurističkih algoritama i optimizacijskih problema u stvarnim problemima.

2. Teorijska pozadina

U ovom poglavlju obradit ćemo sve teorijske teme koje su ključne za razumijevanje i provedbu ovog rada.

Kroz povijest i korištenje umjetne inteligencije u društvenim igrama, detaljno istraživanje različitih aspekata igre Boggle, uključujući njezinu povijest, pravila i mehaniku, te računalne algoritme i strukture podataka, osigurat ćemo čvrstu osnovu za analizu i implementaciju programskih rješenja.

Prvo ćemo se osvrnuti na povijest i značaj umjetne inteligencije, te razne implementacije umjetne inteligencije u područjima igara.

Nakon toga ćemo se osvrnuti na povijest i razvoj igre Boggle, što će nam pružiti uvid u njezin značaj i evoluciju kroz vrijeme. Zatim ćemo detaljno razmotriti pravila i mehaniku igre, što će biti ključno za razumijevanje izazova s kojima se suočavaju igrači i programi prilikom traženja riječi na ploči. Zajedno s time spomenut ćemo i raznih programskih implementacija rješenja za Boggle

Zatim ćemo istražiti računalne algoritme koji se najčešće koriste za rješavanje igre Boggle. Ova analiza uključivat će pregled osnovnih algoritama pretraživanja, poput dubinskog pretraživanja (Depth-First Search – DFS). Također ćemo razmotriti različite optimizacijske tehnike koje se primjenjuju kako bi se poboljšala efikasnost algoritama.

Konačno, pogledat ćemo strukture podataka koje su optimalne za korištenje u pretraživanjima koja uključuju podudaranja riječi, a to su trie strukture.

Kroz ovo poglavlje, čitatelj će steći sveobuhvatno razumijevanje teorijskih osnova koje su neophodne za uspješnu provedbu ovog rada

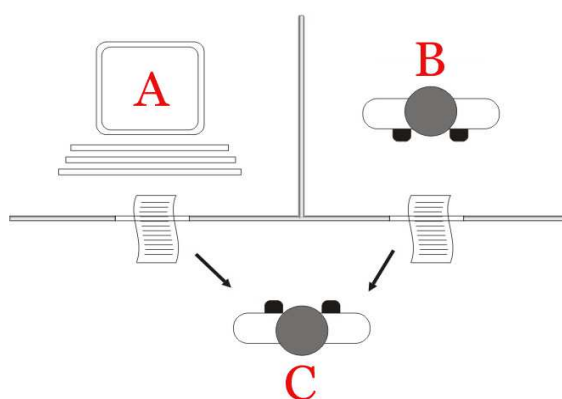
2.1. Umjetna inteligencija

Umjetna inteligencija predstavlja sposobnost računala ili robota da obavlja zadatke koji obično zahtijevaju ljudsku inteligenciju. Neki primjeri takvih zadataka bili bi prepoznavanje govora, donošenje odluka na temelju poznatih informacija, prevođenje jezika ili pak vizualno prepoznavanje. Te zadatke svrstavamo u široko područje tehnika kao što su duboko učenje, strojno učenje pretraživanje prostora stanja te logiku [2].

2.1.1. Kratka povijest umjetne inteligencije

Umjetna inteligencija nije nastala jučer, već ona ima duboke korijene koji sežu u prošlost sve do prvih pokušaja modeliranja ljudskog razmišljanja i inteligencije pa sve do razvoja složenih algoritama i sustava kakvi se danas koriste.

Prvi radovi u ovome području datiraju sa sredine 20. stoljeća. Davne 1943. godine, Warren McCulloch i Walter Pitts predstavljaju prvi model umjetnih neurona koja otvara put istraživanjima umjetnih neuronskih mreža. Kasnije, Alan Turing postavlja temelje za razmišljanje o inteligenciji računala te uspoređivanju računala s čovjekom. To čini predloženim Turingovim testom u svojem radu *Computing Machinery and Intelligence*. Njime bi se mjerila inteligencija računala te mogu li zavarati osobu prilikom testova [3].



Slika 2.1: Turingov test

Ključni događaj u području umjetne inteligencije dogodio se 1956. godine na konferenciji u Dartmouthu, koju su organizirali imena poput John McCarthy, Marvin Minskog, Nathaniel

Rocherstera i Claude Shannona. Ovom konferencijom označava se formalno rođenje umjetne inteligencije kao polja proučavanja i postavljaju se temelji za daljnja istraživanja i razvoj [2].

Brojne prekretnice i inovacije su obilježile razvoj umjetne inteligencije do današnjih dana, počevši od predstavljanja perceptrona, prve umjetne neuronske mreže koja je sposobna učiti iz ulaznih podataka. Dalje dolazi do razvoja ekspertnih sustava te sve složenijih tehnika strojnog učenja, od kojih je jedna duboko učenje.

Danas, umjetna inteligencija je sveprisutna u svakodnevnom životu, od glasovnih asistenata, preporuka na društvenim mrežama, autonomnih vozila i razvoja u medicini i drugim područjima [2].

2.1.2. Umjetna inteligencija u društvenim igrama

U području umjetne inteligencije u društvenim igrama, dva bitna uspjeha su kada je računalni program uspio pobijediti ljudske prvake u šahu i igri Go.

Deep Blue

Računalo koje je pobijedilo tada aktualnog prvaka svijeta u šahu, Garya Kasparova, 1997. godine bilo je Deep Blue. Deep Blue je superračunalo nastalo pod okriljem tvrtke *International Business Machines Corporation (Međunarodna korporacija za poslovne strojeve)*, poznate kao *IBM*. Računalo je sa svoja 32 procesora moglo evaluirati 200 milijuna šahovskih poteza u sekundi te time pobijedilo prvaka u meču od 6 partija [5].

AlphaGo

2016. godine, *Googleov DeepMind* stvorio je program AlphaGo koji je u dvoboju protiv tada svjetskog prvaka u igri Go, Lee Sedola, pobijedio s 4:1. Igra Go je kompleksna igra koja zahtjeva puno strategije te ima ogroman broj mogućih konfiguracija ploče (10^{170}), čime nadmašuje kompleksnost šaha za vrijednost od 10^{100} . Program se sastoji od dvije duboke neuronske mreže od kojih jedna izabire sljedeći potez, a druga predviđa pobjednika igre [6].

Ovi uspjesi imali su utjecaj na širu javnost te potaknuli daljnji razvoj i istraživanja u području umjetne inteligencije u raznim domenama života.

2.2. Društvena igra Boggle

Prvotno ime društvene igre Boggle bilo je *Find-a-Word*, a osmislio ju je izumitelj Allan Turoff koji je svoju ideju predstavio tvrtki *Parker Brothers*. Početnu ideju tvrtka je odbila te čekala sve do 1972. godine kada je preimenovala Turoffovu igru u Boggle i plasirala je na tržište. Od tada, igra je doživjela mnoge nove verzije te se etablirala kao jedna od glavnih obiteljskih igara [7].

2.2.1. Pravila i mehanika igre Boggle

Pravila igre su vrlo jednostavna i to ju čini vrlo poželjnom. Da bi igra započela, potrebno je samo zatresti ploču s kockicama na kojima se nalaze slova abecede, pokrenuti štopericu na 3 minute i naći što je više moguće riječi na ploči.

2.2.2. Izazovi u pronalaženju riječi

Sama ploča igre Boggle ima dimenzije 4x4, ali ne može se pretražiti na bilo koji način. Pravila nalažu da se riječi moraju slagati u nizu, krećući se u bilo kojem smjeru - dijagonalno, vertikalno ili horizontalno – pri čemu se svaka kockica smije koristiti samo jedanput u jednoj riječi. Zbog pritiska vremena i mnogo opcija na ploči, vrlo je izazovno pronaći polovicu svih mogućih riječi na ploči, dok je pronalaženje svih riječi izuzetno teško. Nečesto se dogodi da igrači u želji brzine pronađu i riječi koje ne postoje. Dodatno, kako bi pretraživanje riječi bilo još kaotičnije i složenije, postoje varijante igre poput *Big Boggle* i *Super Big Boggle*. Ove varijante imaju povećane dimenzije ploče od 5x5, odnosno 6x6, što omogućava pronalaženje još većeg broja riječi i stvaranje dužih riječi. Vizualni prikaz ploča možete vidjeti na slici 2.2.



Slika 2.2: Različita izdanja igre Boggle

2.2.3. Programska rješenja na temu igre Boggle

Zbog složenosti pretraživanja i velikog broja mogućih kombinacija, mnogi programeri su razvili računalna rješenja za pronalaženje svih validnih riječi na Boggle ploči. Jedno od najpoznatijih programskih rješenja napisao je Troy Downing, a njegovo rješenje dostupno je na ovoj [poveznici](#) [11].

Pored Downlinga, mnogi drugi autori su također razvili svoja rješenja, koristeći različite algoritme i tehnike. Postoji mnogo različitih implementacija u raznim programskim jezicima, uključujući Python, C++, Javu i JavaScript.

Ova raznolika programska rješenja predstavljaju odličan alat za razumijevanje algoritama pretraživanja i struktura podataka.

2.3. Algoritmi

Algoritam je konačan slijed precizno definiranih koraka ili instrukcija koji se koriste za rješavanje problema ili izvođenje određenog zadatka. U programiranju ih koristimo kako bi omogućili računalima da izvršavaju složene operacije na učinkovit način [12].

Osnovne karakteristike svakog algoritma:

- Preciznost - kojom se jasno definira svaki korak algoritma tako da ne bude dvosmislen
- Konačnost – kojom se definira hoće li algoritam završiti u konačnom broju koraka
- Efikasnost – kojom se definira koliko resursa (vremena i prostora) nam je potrebno i kako ju algoritam iskorištava da bi došao do krajnjeg rezultata

2.3.1. Povijest algoritama

Algoritmi su temelj računarstva i informatike i njihova prisutnost se proteže kroz tisućljeća ljudske povijesti. Njihova povijest započinje još s antičkim matematičarima Euklidom i Arhimedom, koji su razvili temeljne algoritme za matematiku i geometriju. Tijekom srednjeg vijeka, Leonardo Fibonacci daje i svoj doprinos u razvoju algoritama s Fibonaccijevim nizom, koji ima široku primjenu u područjima matematike i računarstva.

Daljnijim razvojem moderne matematike s vođama razvoja u obliku Isaaca Newtona i Gottfrieda Wilhelma Leibniza, ubrzo dolazimo i do pionira u razvoju računalnih strojeva, Charles Babbageu, koji stvara prve korake prema algoritmima kakve danas poznajemo. Tijekom 20. stoljeća, algoritmi prelaze u središnji dio računalne znanosti, posebice radom Alana Turinga s razvojem Turingovog stroja te Johna von Neumanna svojim razvojem arhitekture modernih računala, čime stvara uvjete za izvođenje složenih algoritama.

Danas algoritmi imaju ključnu ulogu u gotovo svim aspektima tehnologije. Koriste se u računalnoj znanosti, umjetnoj inteligenciji, strojnom učenju, komunikacijama i mnogim drugim područjima. Stalnom primjenom dolazi i do ubrzanog razvoja čime konstantno dolazimo do novih otkrića i inovacija u računarstvu i tehnologiji [13].

2.3.2. Vrednovanje uspješnosti algoritama

Da bismo ocijenili kvalitetu algoritma, važno je osigurati da su neke ključne karakteristike ispunjene.

Potpunost

Algoritam se smatra potpunim kada za problem za koji je namijenjen pronađe ispravno rješenje kada rješenje postoji. Kada bi ovo svojstvo nedostajalo, nemamo garanciju kada pokrećemo algoritam s podacima, da ćemo dobiti neko rješenje ne znajući je li to zbog nedostatka rješenja ili mane algoritma

Optimalnost

Optimalnost je svojstvo kojem kada algoritam pokrenemo i on pronađe neko rješenje, tada sa sigurnošću znamo da je to rješenje i najbolje rješenje koje tražimo. Nekada nam nije bitno je li pronađeno rješenje najbolje, bilo to u smislu cijene, brzine ili efikasnosti, no kada je potrebno pronaći najbolje, tada algoritam mora zadovoljiti svojstvo optimalnosti.

Vremenska složenost

Vremensku složenost algoritma promatramo u odnosu na količinu vremena koju algoritam zahtjeva da bi se izvršio. Tu količinu obično uspoređujemo u ovisnosti o ulaznim podacima. Kada imamo veliki ulaz i zahtjevne probleme, bitno je da ih performantno pretražimo i izvršimo algoritam nad njima da u što manjem vremenu, čime koristimo i što manje vremenskih resursa. Ovu složenost najčešće notiramo u obliku O-veličine, što predstavlja gornju granicu rasta vremena izvršavanja algoritma u najgorem slučaju.

Prostorna složenost

Prostorna složenost je slična vremenskoj složenosti, s razlikom da se odnosi na količinu memorijskog prostora koju algoritam zahtjeva tijekom svog izvršavanja. Ponekad smo ograničeni prostorom koji posjedujemo i tada trebamo kvalitetno posložiti algoritam kako ne bi koristili preveliku količinu memorijskog prostora za problem koji zahtjeva mnogo manje. Notiranje je identično kao i prilikom vremenske složenosti, s razlikom da O-veličina predstavlja gornju granicu rasta memorijskog zauzeća prostora tijekom izvršavanja algoritma u najgorem slučaju [13].

2.3.3. Vrste algoritama

Algoritme možemo klasificirati u različite vrste ovisno na koji način rješavaju probleme i strategije koje koriste tijekom izvođenja. Dvije osnovne kategorije s kojima ćemo se detaljnije upoznati su:

- **Algoritmi pretraživanja stanja**

Algoritmi pretraživanja stanja su osnovna klasa algoritama koji se koriste za pronalaženje rješenja u problemima gdje je prostor mogućih rješenja moguće strukturirati u obliku grafa ili stabla. Glavna karakteristika je sistematsko pretraživanje prostora od kojih svaki od glavnih predstavnika ima različitu strategiju kako prolazi kroz stabla generirana od prostora rješenja.

Glavni predstavnici su:

1. Pretraživanje u dubinu (DFS) – Njegova strategija je prvo pretražiti prostor u dubinu prije povratka u čvor roditelja. Kada ga vrednujemo možemo reći da nije potpun algoritam, ali ni optimalan. To označava da će nekada može pronaći rješenje, ali ono ni tad ne treba biti najbolje, ali postoje i slučajevi kada uđe u beskonačnu petlju i nikada se ne izvrši do kraja. Vremenska složenost mu je $O(b^m)$ gdje je b faktor grananja stabla, to jest prosječan broj djece jednog čvora roditelja, a m maksimalna dubina stabla. Prostorna složenost je $O(b \cdot m)$ [14].
2. Pretraživanje u širinu (BFS) – Njegova strategija je pretraživati stablo razinu po razinu putem spremajući svu djecu proširenih čvorova na stog. Ovime, za razliku od dubinskog pretraživanja, možemo garantirati da će rješenje uvijek biti pronađeno, ali i isto tako da će ono biti najbolje moguće. Time je pretraživanje u širinu zadovoljilo uvjete potpunosti i optimalnosti. Vremenska i prostorna složenost su mu jednake i iznose $O(b^{d+1})$ gdje b označava faktor grananja stabla, a d dubinu optimalnog rješenja. Razlog zašto uz d dodajemo još jedinicu slijedi iz širenja i spremanja čvorova djece tijekom prolaska čvorom [14].
3. Iterativno pretraživanje – Njegova strategija je kombinacija pretraživanja u dubinu i pretraživanja u širinu. Počinje s malom dubinom pretraživanja i postepeno ju povećava sve dokle ne dođe do rješenja. Na ovaj način nadogradili smo pretraživanje u dubinu svojstvima optimalnosti i potpunosti, a isto tako i smanjili vremensku i prostornu složenost tako da one više ne ovise o

maksimalnoj dubini stabla, već o optimalnoj dubini stabla, to jest o dubini na kojoj se nalazi najbolje rješenje. Tada postizemo vrijednosti $O(b^d)$ za vremensku složenost i $O(b*d)$ za prostornu [14].

- **Heuristički algoritmi**

Heuristički algoritmi su dizajnirani za pronalaženje približnih rješenja u razumnom vremenu. U svojim pretraživanjima uz standardne tehnike algoritama pretraživanja stanja, oni koriste heurističke metode ili pravila, koja ne moraju nužno biti optimalna, ali pomažu u približavanju prema željenom cilju u problemima kada bi algoritmi pretraživanja stanja bili prespori i neefikasni. Kada spominjemo heurističke funkcije, tada ih možemo obilježiti s dva svojstva, a to su optimističnost i konzistentnost. Kako bi heuristička funkcija bila optimistična, tada ona nikad ne precjenjuje stvarnu vrijednost do cilja, to jest da je vrijednost funkcije uvijek niža ili jednaka stvarnoj cijeni. Konzistentnost je nadogradnja na optimističnost kada za svaki čvor u koji stignemo znamo da je to najefikasniji put do tog čvora te na ovaj način vrlo brzo prelazimo kroz stablo sve do željenog cilja.

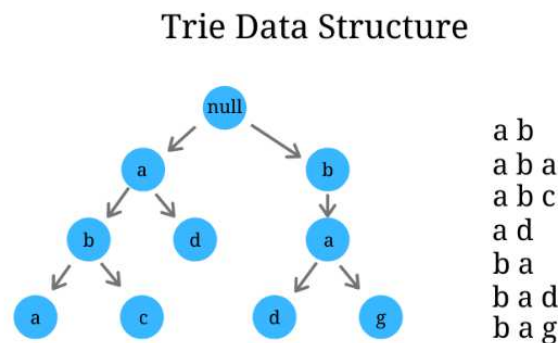
Glavni predstavnik ovih algoritama je:

1. Algoritam A*– Njegova strategija je kombinacija strategije najboljeg prvog poteza zajedno sa heurističkom funkcijom procjene najmanje cijene do cilja. On je uvijek potpun algoritam, no nije uvijek optimalan. Optimalnost ovisi o svojstvu optimističnosti kojom ne precjenjuje stvarne troškove. Vremenska i prostorna složenost su mu jednake, no one ovise o složenosti heurističke funkcije te strukturi grafa. Iznos je $O(\min(b^{d+1}, b|S|))$ gdje je b faktor grananja stabla, d optimalna dubina rješenja i $|S|$ ukupan broj stanja u pretraživanju [15].

2.4. Trie strukture

Trie strukture su poseban tip stabala. Koristimo ih kada nam je potrebno učinkovito pohranjivanje i pretraživanje skupa podataka. Najčešći podaci koje ćemo spremati u njih su tekstni nizovi. Sama riječ trie zapravo je dio engleski riječi *retrieval*, značenja dohvaćanje, kojom se naglašava glavna svrha ovih struktura, a to je brzo i efikasno dohvaćanje podataka.

Sama strukture sastoje se od čvorova koji predstavljaju pojedinačne elemente tekstnog niza. Svaki čvor može imati više djece, pri čemu svako dijete predstavlja novi pojedinačni dio tekstnog niza. Na ovaj način trie strukture omogućuju brzo i efikasno pretraživanje i dohvaćanje nizova.



Slika 2.3: Trie Struktura

2.4.1. Primjena trie struktura

Ove strukture imaju brojne primjene koje zbog svojih specifikacija i potreba mogu vrlo efikasno pretraživati trie strukture. Bitna sličnost svima je što su podaci u obliku tekstnih nizova.

Neke od primjena su sljedeće:

- Pretraživanje riječi
- Automatsko završavanje riječi
- Rad s prefiksima

Svaka od ovih primjena treba brzo i efikasno pretraživanje tekstnih nizova, a trie strukture točno to i omogućavaju. Prolazimo od korijena do velikih dubina vrlo brzo zbog čega je vremenska složenost vrlo povoljna, a isto tako se vrlo lagano mogu dodati nove riječi i tekstni nizovi, a da memorijska složenost bude vrlo efikasno iskorištena.

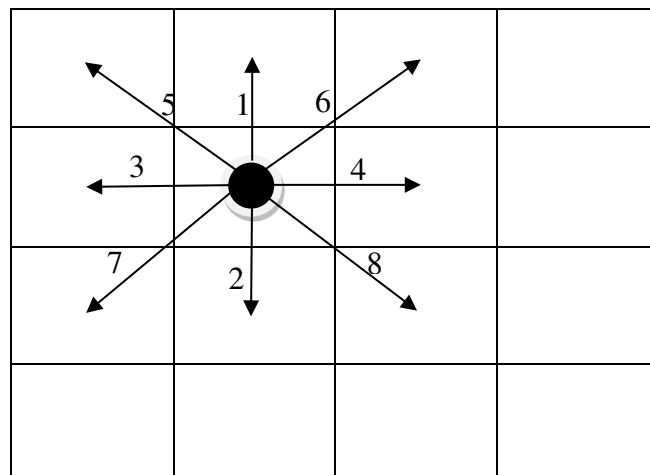
3. Analiza algoritama

Prije nego krenemo na samu analizu algoritama, potrebno je proći kroz rječnik koji je korišten za uspoređivanje validnosti riječi i pretraživanje ploče. Datoteku koja ja korištena sastavio je dr. Goran Igaly, viši predavač na PMF-Matematičkom odsjeku Sveučilišta u Zagrebu. On je sastavio ovaj rječnik s ciljem da obuhvati sve riječi koje se javljaju u tekstovima pisanim na hrvatskom jeziku [18].

Kako pravila igra Boggle navode da se u validne riječi samo one u nominativu, te za glagole u infinitivu, provedeno je filtriranje po rječniku te na taj način strukturirano rječnik koji predstavlja temelj za daljnju analizu i implementaciju algoritama.

Uz rječnik potrebno je i implementirati izgled ploče. Sama ploča je kvadratnih dimenzija, i standardna ploča ima visinu i širinu od 4 polja, od kojih je u svakom polju smješteno jedno slovo hrvatske abecede. Unutar ploče moguće je napraviti 8 različitih pomaka.

Osnovni koraci su gore(1), dolje(2), lijevo(3), desno(4), te su mogući i dijagonalni koraci gore-lijevo(5), gore-desno(6), dolje-lijevo(7) i dolje-desno(8).



Slika 3.1: Moguće kretnje u igri Boggle

U nastavku u sljedećim potpoglavljima proći će se kroz analizu triju algoritama nastalih u različitim stadijima optimizacije, od početnog sve do optimiziranog algoritma.

3.1. Početni algoritam

Za početni algoritam sastavljen je algoritam koji prolazi kroz sve mogućnosti po ploči u rekursivnoj metodi *korak*. Nakon svakog koraka bilježi se odabrani pokret u maski za praćenje posjećenih pozicija kako bi se spriječilo posjećivanje istog polja dva puta u jednom rekursivnom nizu. Samu rekursivnu metodu zovemo nad svakim poljem ploče počevši od pozicije (0,0).

```
korak(ploča, maska, pozicija, riječ)
    postavi x i y na koordinate pozicija

    izaberi stranu u koju se pomičeš:
        ažuriraj masku pomoću x,y vrijednosti
        riječ nadogradi sa slovom na poziciji
        korak(ploča, ažurirana maska, nova pozicija, nadograđena
        riječ)

    broj generiranih riječi ++
    ako riječ validna:
        spremi u listu validnih riječi
    broj validnih riječi ++
```

Pseudokod 1 – Početni algoritam

Vremenska složenost

Sama vremenska složenost ovog algoritma je eksponencijalna. Najgora vrijednost vremenske složenosti dobivamo kada pogledamo najveći smjer kretanja koje možemo postići s neke pozicije i on je jednak 8, te broj pozicija koje postoje na ploči, a on za dimenzije ploče n iznosi n^2 . Tako dolazimo do najgore vrijednosti vremenske složenosti koja je $O(8^{n^2})$.

Kada bismo računali prosječnu vrijednost broja mogućih poteza u zavisnosti od pozicije na ploči dimenzija većih od 2, tada imamo tri različite vrste pozicija:

- Rubne pozicije: Imamo 4 rubne pozicije i svaka ima po 3 moguća kretanja, jedno dijagonalno, jedno okomito i jedno vodoravno

- Pozicije uz rub koje nisu rubne: Za dimenziju ploče n , postoje 2 reda i 2 stupca koji svaki imaju po 2 rubne pozicije uključene pa njih ne želimo ovdje ubrajati te ostale pozicije koje imaju 5 mogućih kretanja tako da tu dobivamo zbroj $4 * (n-2)$
- Unutrašnje pozicije: Imaju 8 mogućih kretanja i ovih pločica je n^2 minus ostale dvije vrste pozicija pa iz toga slijedi $(n^2 - 4 - 4 * (n - 2)) * 8$

Na ovaj način dolazimo do jednadžbe (1).

$$\frac{4*3+(4*(n-2))*5+(n^2-4-4*(n-2))*8}{n^2} \quad (1)$$

Za velike vrijednosti varijable n , ova vrijednost će se asimptotski približavati broju 8 te tako vremenska složenost teži ka $O(8^{n^2})$.

Prostorna složenost

Prostornu složenost određujemo na temelju sljedećih čimbenika:

- Maska – dvodimenzionalna maska istih dimenzija kao i ploča koristi $O(n^2)$ gdje je n ukupan broj pozicija na ploči
- Rekurzije – Dubina rekurzija može otići sve do broja pozicija na ploči, što zahtjeva dodatnih $O(n^2)$ prostora
- Riječ – Isto tako maksimalna duljina generirane riječi može biti $O(n^2)$

Svaki od čimbenika pridonosi sa složenosti $O(n^2)$ te njihove vrijednosti zbrajamo i time dobivamo složenost $O(3*n^2)$

Ovaj algoritam će uvijek pronaći sve validne riječi na ploči, no efikasnost se vrlo brzo može vidjeti kako pada zbog eksponencijalne vremenske složenosti.

3.2. Algoritam nastao djelomičnom optimizacijom

Drugi algoritam nastao je djelomičnom optimizacijom. Kako je prvi algoritam trošio jako puno vremena u rekurzijama koje ne pronalaze nove validne riječi, potrebno je bilo koristiti tehniku podrezivanja grana, te kada više neka grana rekurzije nije imala mogućnosti pronaći nove validne riječi, prekinuti njeno pretraživanje. Isto tako, nakon što smo nadogradili trenutnu riječ, nema smisla više promatrati podudaranja nad cijelim rječnikom, već je dovoljno profiltrirati i reducirati na samo one riječi koje imaju prefiks jednak trenutnoj riječi.

```
korak(ploča, maska, pozicija, riječ, rječnik)
    broj generiranih riječi ++
    ako riječ validna:
        spremi u listu validnih riječi
        broj validnih riječi ++

    ako rječnik prazan:
        vrati

    postavi x i y na koordinate pozicija

    izaberi stranu u koju se pomičeš:
        ažuriraj masku pomoću x,y vrijednosti
        riječ nadogradi sa slovom na poziciji
        novi rječnik = reducirani rječnik po trenutno generiranoj
        riječi
        korak(ploča, ažurirana maska, nova pozicija, nadograđena
        riječ, novi rječnik)
```

Pseudokod 2 – Djelomično optimizirani algoritam

Vremenska složenost

Kao i u gornjem algoritmu, najgori slučaj će biti $O(8^{n^2})$, no u stvarnosti, prosječan broj će biti mnogo manji. Teško je točno procijeniti prosječnu vrijednost zbog slučajnosti u generiranju slova na ploči, no ona će biti mnogo manja od najgore vrijednosti. To dolazi kada na primjer, kada je dimenzija ploče 4 i skratimo jednu granu u rekurziji dubine 4, tada skraćujemo broj grana za broj poteza od te pozicije, pa nadalje rekurzivno od svake sljedeće

pozicije njen broj pozicija te na kraju time skratimo veliki postotak grana koje ne bi dale validne riječi na izlazu.

Prostorna složenost

Najgora prostorna složenost je isto tako kao i u početnom algoritmu $O(3^n)$, no isto kao i za vremensku složenost, on je mnogo manji. Ovdje je razlika da ćemo rijetko kada generirati velik broj rekurzija u lancu zbog filtriranja rječnika. U prvom koraku će rječnik uvijek biti potpun, no već s novim korakom, on se smanji na vrlo mali broj u odnosu na početak jer će filtrirati samo po početnom slovu, koji prosječno zauzima jednu tridesetinu ukupnog prostora rječnika.

3.3. Optimizirani algoritam

Posljednji algoritam koristi trie strukturu kako bi se riječi učinkovitije pretraživale i validirale. Rječnik korišten u prethodnim algoritama unaprijeđen je pretvaranjem u trie strukturu. Tako je svaka riječ u strukturi zapravo predstavljena nizom čvorova. Svaki čvor sadrži jedno slovo, podstablo slova koja mogu doći poslije ovog čvora te varijablu koja označuje je li trenutni niz slova validna riječ.

```
korak_trie(ploča, maska, pozicija, riječ, trie_stablo)
    broj generiranih riječi ++

    ako riječ u trie_stablu:
        spremi u listu validnih riječi
        broj validnih riječi ++
    inače:
        vrati

    postavi x i y na koordinate pozicija

    izaberi novi smjer kretanja:
        ažuriraj masku pomoću x,y vrijednosti
        riječ nadogradi sa slovom u trenutnom čvoru
        korak_trie(ploča, ažurirana_maska, ažurirana pozicija,
nadograđena_riječ, trie)
```

Pseudokod 3 – Optimizirani algoritam

Vremenska složenost

U najgorem slučaju, vremenska složenost ostaje $O(8^{n^2})$, ali prosječna složenost je mnogo manja zbog upotrebe trie strukture. Trie struktura omogućava brzo podrezivanje grana pretrage. Svaki čvor u trie strukturi sadrži informaciju o tome je li trenutni niz slova validan te ima li nastavak. Ovim pristupom, algoritam brzo prekida rekurziju kad se dođe do niza bez validnih riječi, smanjujući tako broj nepotrebnih pretraga.

Prostorna složenost

Najgora prostorna složenost je ponovno $O(3*n^2)$, kao i u početnom i djelomično optimiziranom algoritmu. Međutim, prosječna prostorna složenost je značajno manja. U djelomično optimiziranom algoritmu, rječnik se smanjivao svakim korakom dublje u rekurziju. U ovom algoritmu, svaki korak uzima samo podstablo trenutnog čvora u trie strukturi, što smanjuje memorijske potrebe tijekom pretrage. Trie struktura također koristi manje memorije za pohranu jer dijeli zajedničke prefikse među riječima, što dodatno optimizira prostornu složenost.

3.4. Sažetak analize algoritama

Analizom ova tri algoritma, koji imaju glavnu logiku jednaku, no razlikuju se u sitnim implementacijskim promjenama, možemo vidjeti koliko je važno pronaći ispravnu strukturu podataka te strategiju kako će naš algoritam prolaziti kroz podatke, imajući na umu svojstva kao što su vremenska i prostorna složenost. Ova svojstva će uvelike pokazati krećemo li se u dobrom smjeru implementacije.

Kroz dva manja koraka u provedbi strategije algoritma, uspjeli smo kreirati mnogo učinkovitiji algoritam nego na početku. Iako se najgora vremena prostorne i vremenske složenosti nisu mijenjale kroz implementacije, praktična učinkovitost značajno je porasla.

4. Analiza grafova

U ovom poglavlju, iskazat ćemo rezultate usporedbi algoritama napravljenih za pretraživanje ploče igre Boggle. Usporedbu ćemo provesti nad djelomično optimiziranim algoritmom te optimiziranim algoritmom. Usporedba će istražiti ponašanje ovih algoritama na osnovnoj ploči društvene igre Boggle koja je dimenzija 4x4, te nakon toga samu skalabilnost tih algoritama i kako će povećanje dimenzija utjecati na njih.

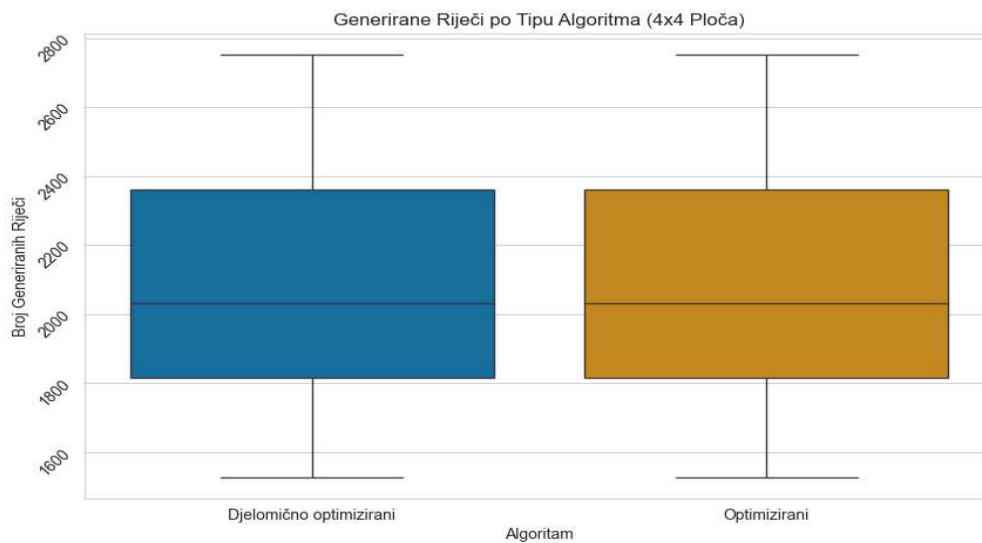
Podaci koji će biti prikazani dobiveni su kroz mnoštvo generiranja raznih ploča te izbora najboljih uzoraka kako bi analiza bila što bliže stvarnim podacima.

4.1. Usporedba algoritama nad pločom 4x4

Analizirajmo performanse djelomično optimiziranog i optimiziranog algoritma za pretraživanje ploče igre Boggle dimenzija 4x4.

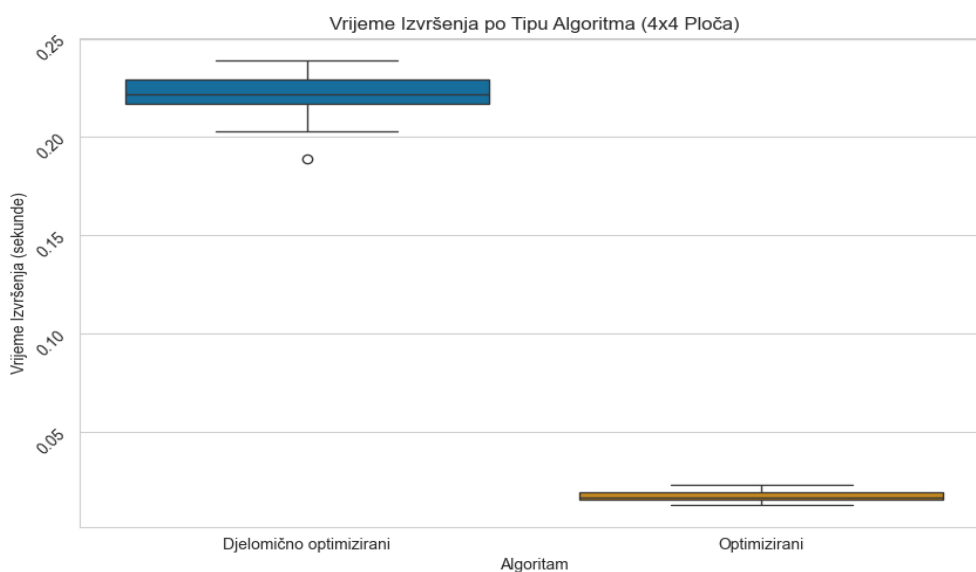
Prva slika (Slika 4.1), prikazuje podatke usporedbe broja generiranih riječi po tipu algoritma.

Slika jasno prikazuje kako su vrijednosti ova dva algoritma identična, što objašnjavamo tako da na isti način i u istim slučajevima odlučuju za podrezivanje grana. Ovdje ćemo spomenuti i početni algoritam. On prolazi kroz sve mogućnosti na ploči te tako svakim pokretanjem uvijek generira jednak broj riječi, što je u slučaju ploče dimenzija 4x4 više od 12 milijuna riječi, a vrijeme potrebno za izvršavanje u prosjeku 106 sekundi. Već prelaskom na ploču 5x5 algoritam ne završava nakon što je prođe više od 24 sata izvršavanja. Iz ovih razloga ovog algoritma nema na prikazima grafova, jer nije usporediv s druga dva.



Slika 4.1: Graf - Generirane Riječi po Tipu Algoritma (4x4)

Druga slika (Slika 4.2) , prikazuje podatke usporedbe vremena izvršavanja ovisno o tipu algoritma i ovdje jasno možemo vidjeti razliku u njima. Optimizirani algoritam koji koristi trie strukturu, koja je specijalizirani tip strukture za slučajeve kada trebamo ispitati podudaranja i prefikse riječi, mnogo brže uspijeva savladati jednaku količinu riječi koje generira.

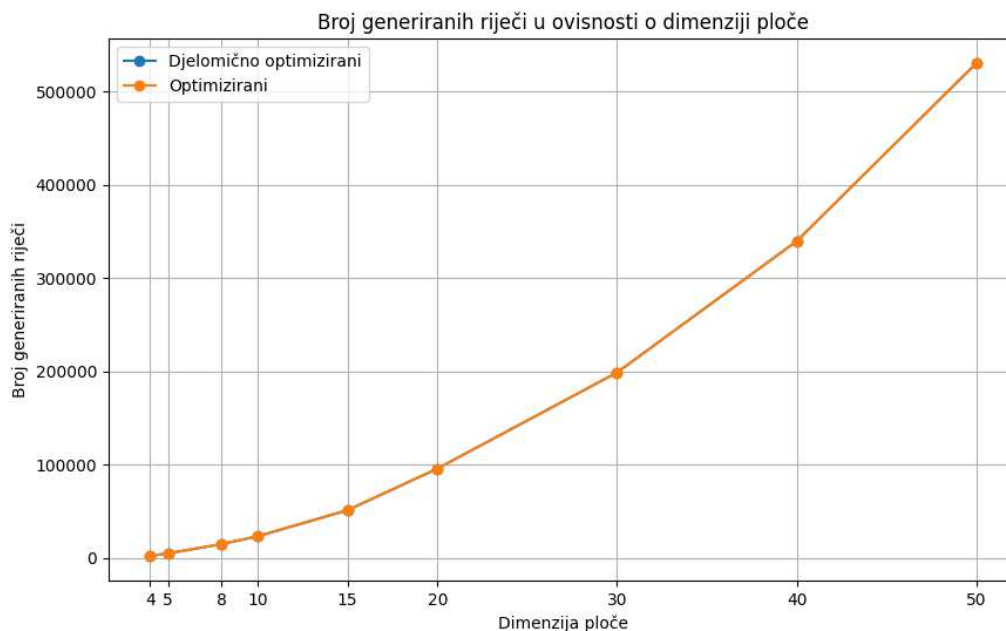


Slika 4.2: Graf – Vrijeme Izvršenja po Tipu Algoritma (4x4)

4.2. Usporedba algoritama kroz razne dimenzije ploče

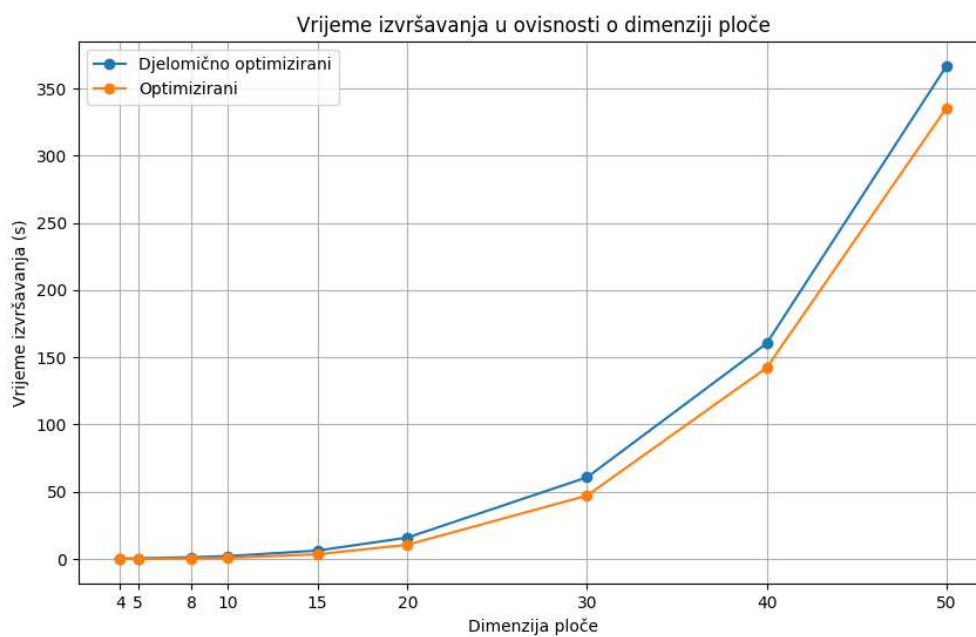
U ovom dijelu analizirat ćemo performanse djelomično optimiziranog i optimiziranog algoritma kroz različite dimenzije ploče kako bismo vidjeli skalabilnost algoritama.

Na Slici 4.3 prikazani su podaci usporedbe broja generiranih riječi po tipu algoritma kroz različite dimenzije ploče. Možemo vidjeti kako s povećanjem dimenzionalnosti ploče eksponencijalno raste i broj generiranih riječi, ali također vidimo da oba algoritma imaju identične vrijednosti u svakom trenutku. To dolazi iz jednakog pristupa podrezivanju grana prilikom prolaska ploče.

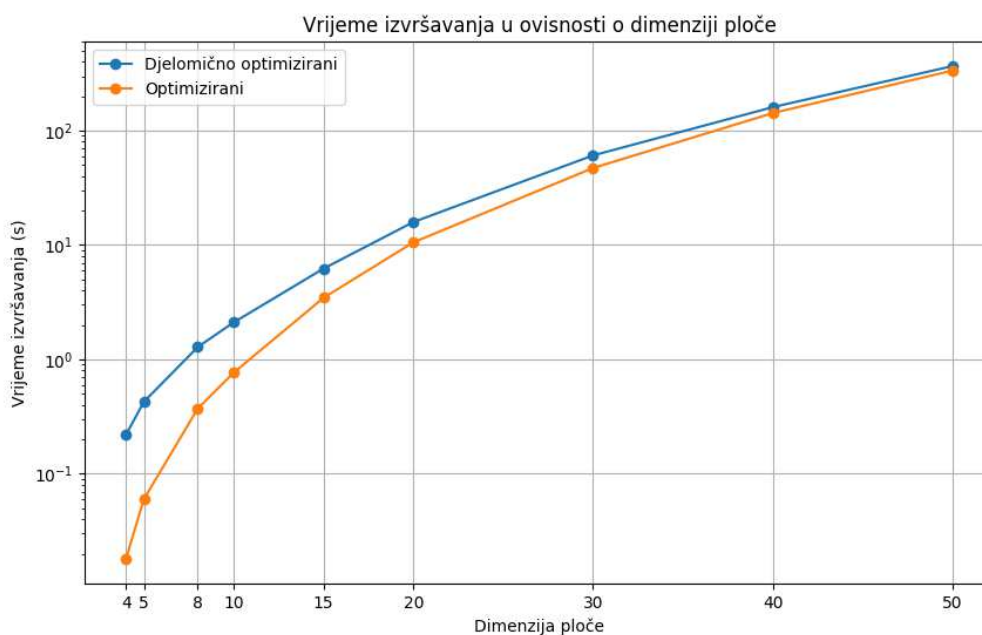


Slika 4.3: Graf – Broj generiranih riječi u ovisnosti o dimenziji ploče

Sljedeće dvije slike (Slika 4.4 i Slika 4.5) prikazuju ovisnost vremena izvršavanja u odnosu na dimenzije ploče. Ovdje vidimo nastavak trenda sa Slike 4.2 i usporedbe algoritama nad pločom dimenzija 4x4. Optimizirani algoritam je u svakom trenutku brži od djelomično optimiziranog algoritma, što se jasno može vidjeti za početne dimenzije ploče na Slici 4.5, gdje je korištena logaritamska skala kako bi lakše prikazali razlike prilikom manjih vrijednosti dimenzija ploče, dok Slika 4.4 prikazuje razlike za veće dimenzije ploče.

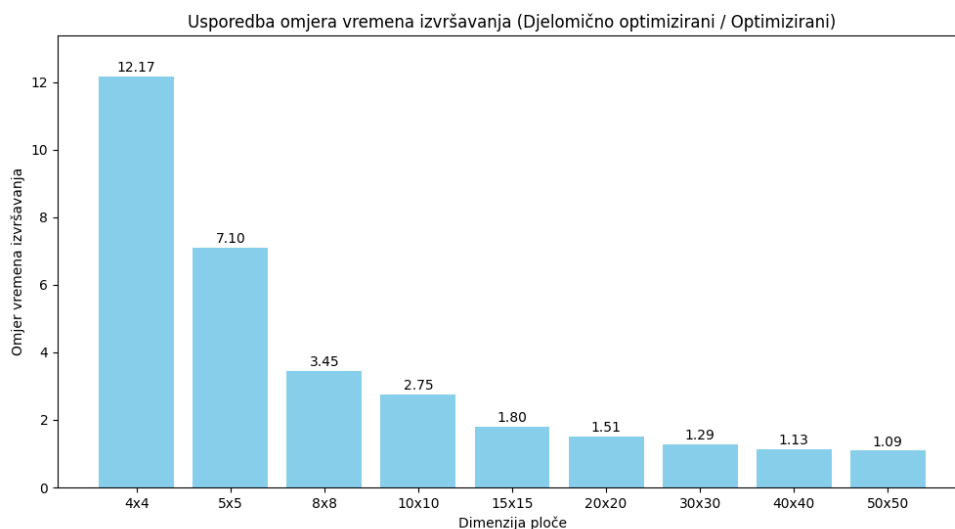


Slika 4.4: Graf – Vrijeme izvršavanja u ovisnosti o dimenziji ploče



Slika 4.5: Graf – Vrijeme izvršavanja u ovisnosti o dimenziji ploče - logaritamski

U grafu 4.5 prikazani su omjeri vremena izvršavanja za djelomično optimizirani algoritam i optimizirani algoritam dobiveni eksperimentalnim podacima. Sami omjeri bili bi više prema strani optimiziranog algoritma u stvarnim podacima, no generiranjem slučajnih ploča gubi se na smislenosti samih ploča i dolazi do gubitaka u informaciji.



Slika 4.6: Usporedba omjera vremena izvršavanja (Djelomično optimizirani / Optimizirani)

4.3. Sažetak analize grafova

U analizi smo pokazali da optimizirani algoritam, koji koristi trie strukturu, nadmašuje djelomično optimizirani algoritam u pogledu vremena izvršavanja pri svakoj dimenziji ploče, dok oba algoritma generiraju isti broj riječi u bilo kojem slučaju.

Optimizirani algoritam pokazuje superiorne performanse, posebno pri većim dimenzijama ploče, zbog učinkovitijeg pretraživanja i podrezivanja grana, čime dobiva na brzini izvođenja.

Ova analiza jasno ilustrira prednosti optimizacije i upotrebe specijaliziranih struktura podataka za rješavanje kompleksnih problema poput pretraživanja u igri Boggle.

5. Demo korištenja djelomično optimiziranog i optimiziranog algoritma na ploči igre Boggle dimenzija 4x4

U ovom poglavlju se nalazi demonstracija rada djelomično optimiziranog i optimiziranog algoritma.

Njihovo generiranje riječi je jednako, i prolazit će uvijek kroz iste grane na jednak način, no drugačije će spremati podatke pa će se taj dio prikazati njihovim trenutnim stanjem po koracima. Sastavljen je primjerak ploče s nekoliko validnih riječi, no s pomoću ove ploče proći će se kroz sve funkcionalnosti algoritama.

Sadržaj rječnika: antena, auto, arka, auti, autor, automat, alka, alkar

Trie struktura:

a → an → ant → ante → anten → **antena**
→ au → aut → **auto** → autom → automa → **automat**
→ **autor**
→ **auti**
→ ar → ark → **arka**
→ al → alk → **alka** → **alkar**

Legenda:

- **Plava pozadina** – trenutna pozicija
- **Crvena pozadina** – prođena pozicija u rekurziji
- Preferencije kretanja: 1. Lijevo, 2. Desno, 3. Gore, 4. Dolje
5. Gore-lijevo, 6. Gore-desno, 7. Dolje-lijevo, 8. Dolje-desno

1. Korak – Početno stanje u (0,0)

a	u	t	a
ć	đ	o	r
f	g	h	k
i	k	l	c

Stanje rječnika: [antena, auto, arka, auti, autor, automat, alka, alkar]

Trie struktura: [Čvor **a**]

Validno: Ne

Validne riječi: []

2. Korak – Pomak u desno

a	u	t	a
ć	đ	o	r
f	g	h	k
I	k	l	c

Stanje rječnika: [auto, auti, autor, automat]

Trie struktura: [Čvor **au**]

Validno: Ne

Validne riječi: []

3. Korak – Pomak u desno

a	u	t	a
ć	đ	o	r
f	g	h	k
i	k	l	c

Stanje rječnika: [auto, auti, autor, automat]

Trie struktura: [Čvor **aut**]

Validno: Ne

Validne riječi: []

4. Korak – Pomak desno → prazni rječnik i trie struktura, idemo korak unazad

a	u	t	a
ć	đ	o	r
f	g	h	k
I	k	l	c

Stanje rječnika: []

Trie struktura: []

Validno: Ne

Validne riječi: []

5. Korak – Vraćanje korak unazad

a	u	t	a
ć	đ	o	r
f	g	h	k
i	k	l	c

Stanje rječnika: [auto, auti, autor, automat]

Trie struktura: [Čvor **aut**]

Validno: Ne

Validne riječi: []

6. Korak – Pomak dolje → Pronađena validna riječ

a	u	t	a
ć	đ	o	r
f	g	h	k
i	k	l	c

Stanje rječnika: [auto, autor, automat]

Trie struktura: [Čvor **auto**]

Validno: Da

Validne riječi: [auto]

7. Korak – Pomak lijevo → Prazan rječnik i trie struktura, korak nazad

a	u	t	a
ć	đ	o	r
f	g	h	k
i	k	l	c

Stanje rječnika: []

Trie struktura: []

Validno: Ne

Validne riječi: [auto]

8. Korak – Vraćanje korak unazad

a	u	t	a
ć	đ	o	r
f	g	h	k
i	k	l	c

Stanje rječnika: [auto, autor, automat]

Trie struktura: [Čvor **auto**]

Validno: Da

Validne riječi: [auto]

9. Korak – Pomak desno → Pronađena validna riječ

a	u	t	a
ć	đ	o	r
f	g	h	k
i	k	l	c

Stanje rječnika: [autor]

Trie struktura: [Čvor **autor**]

Validno: Da

Validne riječi: [auto, autor]

10. Korak – Pomak dolje → Prazan rječnik i trie struktura, korak nazad

a	u	t	a
ć	đ	o	r
f	g	h	k
i	k	l	c

Stanje rječnika: []

Trie struktura: []

Validno: Ne

Validne riječi: [auto, autor]

11. Korak – Vraćanje korak unazad

a	u	t	a
ć	đ	o	r
f	g	h	k
i	k	l	c

Stanje rječnika: [autor]

Trie struktura: [Čvor **autor**]

Validno: Da

Validne riječi: [auto, autor]

Završavamo s demonstracijom iz razloga što će sljedeći koraci samo generirati prazne korake rječnika i vraćanje unatrag sve dokle ne prođu sve kombinacije grana koje izlaze iz početnog čvora na poziciji (0,0).

Nakon što algoritmi završe sa početnom pozicijom, ona se pomiče na sljedeće slovo i ponovno za njega prolazi sve mogućnosti.

6. Zaključak

Na temelju analize provedenih algoritama za pretraživanje u igri Boggle, možemo donijeti nekoliko ključnih zaključaka.

Kroz postupnu optimizaciju algoritama, korištenjem trie strukture, postignuta su značajna poboljšanja u vremenskoj složenosti. Oba optimizirana algoritma generiraju isti broj riječi, što je rezultat sličnih strategija podrezivanja grana. Međutim, bitno je istaknuti kako optimizirani algoritam pokazuje superiorne performanse pri većim dimenzijama ploče te na taj način ilustrira prednosti optimizacije i upotrebe specijaliziranih struktura podataka pri rješavanju problema pretraživanja riječi, kao što je to problem u igri Boggle.

Različite strategije pretraživanja, uključujući dubinsko i pretraživanje u širinu, pokazale su se ključnima za optimizaciju. Uvođenje heurističkih metoda, kao što je podrezivanje grana, omogućilo je daljnje smanjenje složenosti te na taj način približilo vrijeme izvršavanja algoritma željenom cilju provođenja algoritma u razumnom vremenskom okviru.

Zaključno, analiza provedenih optimizacija i usporedbi algoritama jasno pokazuje važnost korištenja naprednih struktura podataka i strategija pretraživanja za postizanje boljih performansi u igrama poput igre Boggle. Optimizirani algoritam ne samo da je brži, već je i skalabilniji, što ga čini prikladnim za šire korištenje u sličnim problemima pretraživanja riječi. Ovi nalazi mogu poslužiti kao temelj za daljnja istraživanja i unapređenja algoritama u području umjetne inteligencije i strojnog učenja.

Literatura

- [1] *Što je umjetna inteligencija i kako se upotrebljava?*, Europski parlament, (2020, rujan), Poveznica: <https://www.europarl.europa.eu/topics/hr/article/20200827STO85804/sto-je-umjetna-inteligencija-i-kako-se-upotrebljava>; pristupljeno 2. lipnja 2024.
- [2] *Što je umjetna inteligencija?*, Poveznica: <https://www.umjetnaintelijai.com/sto-je-umjetna-inteligencija/> pristupljeno 2. lipnja 2024
- [3] *A proposal for the Dartmouth Summer Research Project on Artificial Intelligence*, (1955, kolovoz), Poveznica: <https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1904> pristupljeno 3. lipnja 2024
- [4] *Turingov test*, Poveznica: https://en.wikipedia.org/wiki/Turing_test pristupljeno 2. lipnja 2024
- [5] *Deep Blue*, IBM, Poveznica: <https://www.ibm.com/history/deep-blue> pristupljeno 3. lipnja 2024.
- [6] *AlphaGo*, Google, DeepMind, Poveznica: <https://deepmind.google/technologies/alphago/> pristupljeno 3. lipnja 2024.
- [7] *Boggle from Parker Brothers (1972)*, autor: Todd Coopee, (2023, rujan), Poveznica: <https://toytale.ca/boggle-from-parker-brothers-1972/> pristupljeno 2. lipnja 2024.
- [8] *Boggle Word Game*, Steam Rocket, Poveznica: <https://www.steamrocket.co.uk/products/boggle-word-game-hasbro> , pristupljeno 2. lipnja 2024.
- [9] *Big Boggle Board Game by University Games*, Poveznica: <https://www.walmart.com/ip/Big-Boggle-Board-Game-by-University-Games/33359295> , pristupljeno 2. lipnja 2024
- [10] *Super Big Boggle Game*, Poveznica: https://www.amazon.com/Winning-cuadr%C3%ADcula-pulgadas-historia-temporizador/dp/B008GCT8ZM?th=1&language=en_US¤cy=USD pristupljeno: 2. lipnja 2024.
- [11] *Boggle Solver*, Autor: Troy Downing, Poveznica: <https://troydownling.io/programming/boggle-solver/> pristupljeno: 10. travnja 2024.
- [12] *What is Algorithm / Introduction to Algorithms*, Poveznica: <https://www.geeksforgeeks.org/introduction-to-algorithms/> pristupljeno 3. lipnja 2024.
- [13] *History of Algorithms with Timeline*, autor: Shubam Gautam, Poveznica: <https://www.enjoyalgorithms.com/blog/history-of-algorithms> pristupljeno: 4. lipnja 2024.
- [14] *Introduction to Algorithms, Third Edition*, Poveznica: https://pd.daffodilvarsity.edu.bd/course/material/book-430/pdf_content pristupljeno 4. lipnja 2024.

- [15] *A* Search Algorithm*, Poveznica: <https://www.geeksforgeeks.org/a-search-algorithm/> pristupljeno 4. lipnja 2024
- [16] *Trie Data Structure / Insert and Search*, Poveznica: <https://www.geeksforgeeks.org/trie-insert-and-search/> pristupljeno 4. lipnja 2024.
- [17] *What is Trie Data Structure? Why do you need it?*, Poveznica: <https://bootcamp.uxdesign.cc/what-is-trie-data-structure-why-do-you-need-it-c11dbcdfa75b> pristupljeno 4. lipnja 2024.
- [18] *Rječnik-hrvatskih-jezika*, autor: Goran Igaly, Poveznica: <https://github.com/gigaly/rjecnik-hrvatskih-jezika> pristupljeno 10. travnja 2024.

Popis Slika

Slika 2.1: Turingov test	3
Slika 2.2: Različita izdanja igre Boggle	6
Slika 2.3: Trie Struktura.....	11
Slika 3.1: Moguće kretnje u igri Boggle	12
Slika 4.1: Graf - Generirane Riječi po Tipu Algoritma (4x4)	20
Slika 4.2: Graf – Vrijeme Izvršenja po Tipu Algoritma (4x4)	20
Slika 4.3: Graf – Broj generiranih riječi u ovisnosti o dimenziji ploče	21
Slika 4.4: Graf – Vrijeme izvršavanja u ovisnosti o dimenziji ploče	22
Slika 4.5: Graf – Vrijeme izvršavanja u ovisnosti o dimenziji ploče - logaritamski ...	22
Slika 4.6: Usporedba omjera vremena izvršavanja (Djelomično optimizirani / Optimizirani).....	23

Sažetak

Razvoj suca temeljenog na umjetnoj inteligenciji za igru Boggle

Ovaj rad istražuje primjenu umjetne inteligencije u igri Boggle, pružajući pregled povijesti umjetne inteligencije, njezine upotrebe u društvenim igrama te različitih vrsta algoritama. Detaljno se analizira početni algoritam, optimizira i uspoređuje s drugim algoritmima kako bi se istaknule razlike i važnost optimizacije. Kao rezultat, razvijen je algoritam koji u razumnom vremenu može pronaći sve valjane riječi u igri Boggle.

Ključne riječi: umjetna inteligencija, Boggle, algoritmi, optimizacija, trie struktura, analiza

Summary

Development of an Artificial Intelligence-Based Solver for the Game of Boggle

This paper explores the application of artificial intelligence in the game of Boggle, providing an overview of the history of artificial intelligence, its use in social games, and different types of algorithms. The initial algorithm is analyzed in detail, optimized, and compared with other algorithms to highlight differences and the importance of optimization. As a result, an algorithm has been developed that can find all valid words in the game of Boggle in a reasonable amount of time.

Keywords: artificial intelligence, Boggle, algorithms, optimization, trie structure, analysis