Rješavanje problema novčića na zvijezdi

Bekavac, Tonka

Undergraduate thesis / Završni rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split, University of Split, Faculty of science / Sveučilište u Splitu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:166:201595

Rights / Prava: Attribution-NonCommercial-NoDerivatives 4.0 International/Imenovanje-Nekomercijalno-Bez prerada 4.0 međunarodna

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Repository of Faculty of Science







SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

RJEŠAVANJE PROBLEMA NOVČIĆA NA ZVIJEZDI

Tonka Bekavac

Split, rujan 2017.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

RJEŠAVANJE PROBLEMA NOVČIĆA NA ZVIJEZDI

Tonka Bekavac

SAŽETAK

Poopćenje problema novčića na zvijezdi je problem pronalaska maksimalnog broja postavljenih novčića u bilo kojem neusmjerenom grafu, ne samo na zvijezdi. Pošto se rješenje problema može opisati šetnjama na grafu uz određena pravila, a graf je jedna od najčešćih matematičkih struktura za opisivanje prostora stanja, onda možemo koristiti algoritme pretraživanja. Prikazan je postupak rješavanja korištenjem algoritama pretrage. Algoritmi pretraživanja implementirani su u programskom jeziku Python. Rezultati njihovog izvršavanja su analizirani i objašnjena je složenost implementiranih algoritama.

Ključne riječi: problem novčića na zvijezdi, algoritmi pretraživanja

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 30 stranica, 12 grafičkih prikaza, 3 tablice i 10 literaturnih navoda. Izvornik je na hrvatskom jeziku

Mentor: **Doc.dr. sc. Branko Žitko**

Ocjenjivači: **Doc.dr. sc. Branko Žitko**
Doc.dr. sc. Ani Grubišić
Divna Krpan, *predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: **Rujan, 2017**

Basic documentation card

Bachelor thesis

University of Split
Faculty of Science
Department of informatics
Ruđera Boškovića 33, 21000 Split, Croatia

SOLVING COINS ON A STAR PROBLEM

Tonka Bekavac

ABSTRACT

Generalization of coins on a star problem is problem of finding maximum number of coins on any undirected graph, not only on a star. Since a problem solution can be described by the walks on the graph with certain rules and the graph is one of the most common mathematical structures for describing the state space, then we can use search algorithms. The solving process is displays using search algorithms. Search algorithms are implemented in Python. The result of their execution are analyzed and the complexity of implemented algorithms is explained.

Keywords : Coins on a star problem, search algorithms

Thesis deposited in library of Faculty of Science, University of Split

Thesis consists of: 30 pages, 12 figures, 3 tables and 10 references. Original language: Croatian

Supervisor: **Branko Žitko, Doc,Ph.D.**

Reviewers: **Branko Žitko, Doc,Ph.D.**
Ani Grubišić, Doc,Ph.D
Divna Krpan, Lecturer of Faculty of Science, University of Split

Thesis accepted: **September, 2017**

Sadržaj

Uvod ... 1

1. Problem novčića na zvijezdi ... 2

1.1. Povijest problema ... 2

1.2. Slični problemi ... 2

1.2.1. NP problemi ... 4

1.3. Matematički pojmovi vezani uz problem ... 4

1.4. Algoritmi za rješavanje problema ... 6

1.4.1. Iscrpno pretraživanje ... 6

1.4.2. Unatražno pretraživanje ... 7

1.4.3. *Greedy* algoritam ... 9

2. Algoritamsko rješavanje problema ... 11

2.1. Razumijevanje problema ... 11

2.2. Matematička formulacija problema ... 12

2.3. Oblikovanje rješenja problema ... 13

2.3.1. Iscrpno pretraživanje za problem novčića na zvijezdi ... 13

2.3.2. Unatražno pretraživanje za problem novčića na zvijezdi ... 15

2.3.3. *Greedy* algoritam za problem novčića na zvijezdi ... 16

2.3.4. Primjer rješavanja problema *Greedy* strategijom ... 18

2.4. Implementacija rješenja problema novčića na zvijezdi u Pythonu ... 21

2.5. Analiza izvršavanja implementiranih algoritama ... 24

Zaključak ... 27

Literatura ... 28

Sažetak ... 29

Summary ... 30

Uvod

Problem novčića na zvijezdi (engl. *Coins on a star*), je logička slagalica, u kojoj je cilj da se na što veći broj vrhova postavi novčić, poštujući pri tom određena pravila. Poznata je stoljećima. Postoje različite verzije ove slagalice kao na primjer *Octagram puzzle* koja je također poznata stoljećima, te Guarinijeva slagalica (engl. *Guarini puzzle*). Igra se odvija na ploči koju simbolizira zvijezda s osam vrhova i osam bridova. Iz zadanog početnog stanja kreće rješavanje igre, te se do rješenja dolazi sljedeći samo nekoliko jednostavnih pravila, s kojima ćemo se upoznati kasnije. Pomoću tih pravila određuje se i maksimalan broj novčića koje je moguće postaviti.

Cilj ovog rada je opisati matematičke probleme vezane uz problem novčića na zvijezdi te detaljno opisati postupak rješavanja. Za početak ćemo pogledati kako je uopće nastala slagalica te se ukratko upoznati s povijesnim razvojem slagalice. Nakon toga, upoznat ćemo neke probleme slične slagalici. Nadalje, upoznat ćemo detaljno matematička pravila potrebna za razumijevanje slagalice. Na kraju prvog dijela teorijski su opisani algoritmi koji se koriste za rješavanje ovog problema.

U drugom dijelu se pristupa algoritamskom rješenju problema. Prvo se oblikuje problem, tj. definiraju se ulazni i izlazni parametri te potrebna ograničenja. Nakon što se matematički formulira problem, potrebno je oblikovati algoritme za rješavanje. Algoritmi su implementirani u programskom jeziku Python te se analizira njihovo izvršavanje i rezultati izvršavanja nad određenim problemima.

1. Problem novčića na zvijezdi

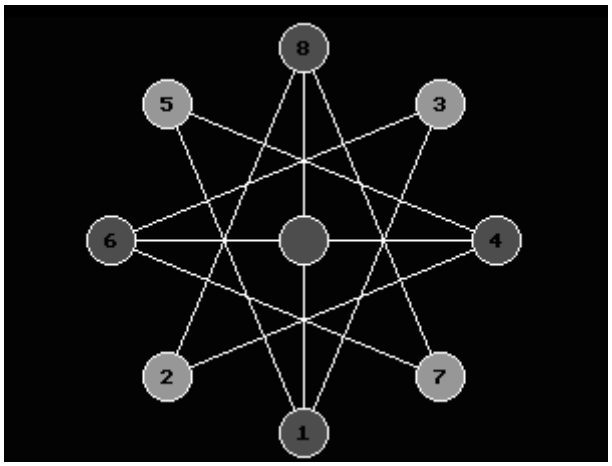
1.1. Povijest problema

Zvijezda s osam krakova, kraće se zove Oktogram (engl. *Octagram*). Općenito, oktogram je bilo koji osmostrani poligon. Većinu igara odnosno slagalica, geometrijskog tipa osmislio je Henry Dudeney, engleski matematičar i jedan od vodećih svjetskih sastavljača zagonetki, pa je stoga i problem novčića na zvijezdi izvedenica jedne od Dudney-evih slagalica [10]. Slagalica je poznata stoljećima, te nema svog autora, već je nastala kroz vrijeme kao izvedenica poznatih sličnih slagalica, o kojima će se govoriti u nastavku.

Logičku slagalicu je prvi napravio Charles Lutwidge Dodgson [8], koji je poznat po nadimku Lewis Carroll. Slagalice se često stvaraju kao oblik zabave, ali se mogu pojaviti i iz ozbiljnih matematičkih ili logičkih problema. U takvim slučajevima, njihovo rješenje može biti značajan doprinos matematičkim istraživanjima. Rješavanje slagalica često zahtijeva prepoznavanje uzoraka i pridržavanje određene vrste pravila.

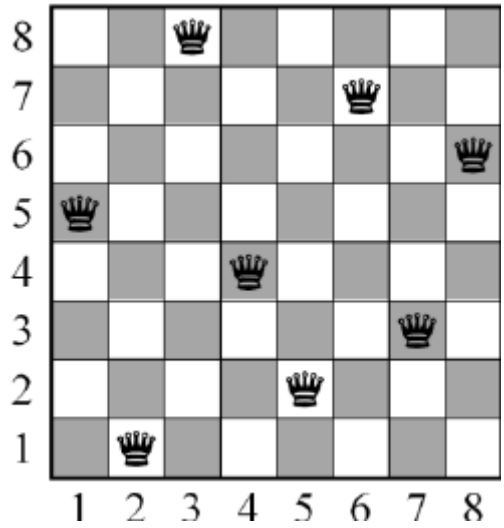
1.2. Slični problemi

Problem postavljanja novčića na zvijezdu s osam krakova je usko povezan s problemom, odnosnom igrom Oktogram (engl. *Octogram*) [9]. Kod problema postavljanja novčića na zvijezdu, novčić se može postaviti na bilo koji od osam krakova zvijezde, koji su numerirani uzastopno počevši od jedan pa sve do osam. Kod igre Oktogram brojevi su slučajno raspoređeni po vrhovima, te postoji dodatno točka u sredini, na koju se privremeno može premjestiti određeni broj. S obzirom na to da su brojevi na vrhovima raspoređeni slučajnim redoslijedom, cilj igre je da brojevi budu poredani uzastopno od jedan do osam u smjeru kazaljke na satu. Dakle, možemo reći da je početno stanje problema novčića na zvijezdi, ujedno i rješenje igre Oktogram. Na slici je prikazano početno stanje igre Oktogram.



Slika 1.1 Početno stanje igre Oktogram

Sličan problem problemu postavljanja novčića na osmerokraku zvijezdu je i šahovska zagonetka Osam kraljica (engl. *Eight queen puzzle*), zbog sličnog načina rješavanja. Kao i kod problema postavljanja novčića na vrhove zvijezde, kod ovog problema također ima mnogo kombinacija mogućih rješenja, stoga ga je najlakše riješiti pomoću jednostavnih pravila odnosno pohlepnog algoritma, koji odabire najbolji mogući potez i ne provjerava sva rješenja. Problem Osam kraljica je problem postavljanja osam kraljica na šahovsku ploču dimenzija 8x8 tako da dvije kraljice ne ugrožavaju jedna drugu. Rješenje zahtijeva da nikoje dvije kraljice ne budu u istom redu, stupcu ili dijagonali. Na slici je prikazano jedno od mogućih rješenja problema.



Slika 1.2 Rješenje problema Osam kraljica

1.2.1. NP problemi

U kombinatornoj optimizaciji često se pojavljuju NP problemi (engl. *non-polynomial problems*). Riječ je o problemima koji se ne mogu rješavati u polinomskom vremenu t = Nk, gdje je N dimenzija problema, a k konstanta. Klasični algoritam koji rješava takve probleme je algoritam iscrpne pretrage koji provjeri sva moguća rješenja da bi našao pravo, ali je ne izvediv za velike N. Problem novčića na zvijezdi između ostalog također ćemo riješiti algoritmom iscrpne pretrage, te dokazati da je algoritam izvodljiv samo za mali N. Opća složenost algoritma je $O((n-1)!)$.

1.3. Matematički pojmovi vezani uz problem

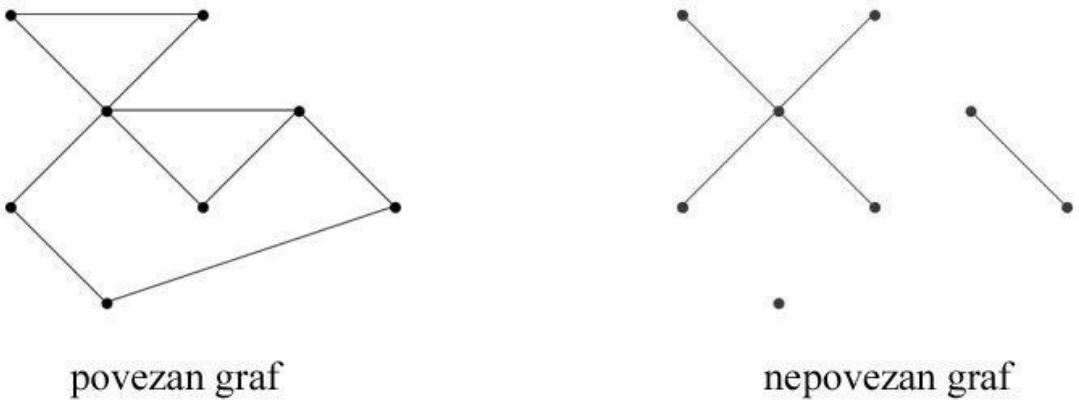
Za problem novčića na zvijezdi koriste se definicije teorije grafova kako bi postavili matematičku podlogu problema. Za početak je definiran graf općenito.

Definicija 1. ***Graf*** *je uređeni par G=(V, E), gdje je* $\emptyset \neq V = V(G)$ *skup* ***vrhova*** *(engl. Vertex),* $E = E(G)$ *skup* ***bridova*** *(engl. Edge) disjunktni s V, a svaki brid* $e \in E$ *spaja dva vrha* $u,v \in V$ *koji se zovu* ***krajevi*** *od e.*

S obzirom na to da je zvijezda graf, i to povezani graf, slijedi definicija povezanog grafa.

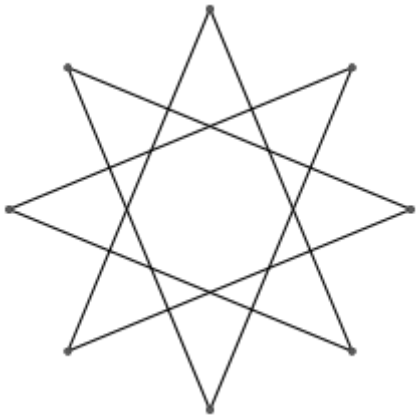
Definicija 2. *Graf je* ***povezan*** *ako se ne može prikazati kao unija neka dva grafa. U suprotnom kažemo da je graf* ***nepovezan****. Svaki se nepovezani graf dakle može prikazati kao unija povezanih grafova. Svaki član te unije zovemo* ***komponenta povezanosti.***

Na slici 1.3 je primjer povezanog i nepovezanog grafa.



Slika 1.3 Primjer povezanog i nepovezanog grafa

Zvijezda s osam vrhova i osam bridova je povezan graf prikazan na slici 1.4.



Slika 1.4 Zvijezda s 8 vrhova i 8 bridova

S obzirom na to da se do rješenja problema novčića na zvijezdi dolazi jednostavnim šetnjama po grafu, u nastavku slijede definicije šetnje u grafu.

Definicija 3. ***Šetnja*** *u grafu G je niz* $W := v_0e_1v_1e_2...e_kv_k$*, čiji su članovi naizmjence vrhovi* v_i *i bridovi* e_i*, tako da su krajevi brida* e_i *vrhovi* v_{i-1} *i* v_i*, gdje je* $1 \leq i \leq k$*. Kažemo da je* v_0 *početak, a* v_k *kraj šetnje W te da je W* ***šetnja od*** v_0 ***do*** v_k*. Vrhovi* $v_1, v_2, ..., v_{k-1}$ *su* ***unutarnji vrhovi šetnje****, a broj k se zove* ***duljina šetnje*** *W.*

1.4. Algoritmi za rješavanje problema

U matematici, računarstvu i lingvistici te srodnim disciplinama, algoritam predstavlja konačan slijed dobro definiranih naredbi za uspješno rješavanje zadatka, koji će za dano početno stanje nakon određenog broja iteracija vratiti točno rješenje. Za rješavanje problema novčića na zvijezdi korišteni su sljedeći algoritmi : iscrpni algoritam (engl. *Brute-force*), unatražno pretraživanje (engl. *backtracking*) te *Greedy* algoritam. Algoritmi za ulaz primaju neusmjereni graf u obliku rječnika, koji određuje način na koji je povezan graf. Ključevi u rječniku predstavljaju vrhove grafa, a vrijednosti rječnika skup vrhova incidentnih s vrhom iz ključa.

Pošto se rješenje problema novčića na zvijezdi može opisati šetnjama na grafu uz određena pravila, a graf je jedna od najčešćih matematičkih struktura za opisivanje prostora stanja, onda možemo koristiti algoritme pretraživanja.

1.4.1. Iscrpno pretraživanje

U računalnoj znanosti, iscrpno pretraživanje je općenita tehnika rješavanja problema, koja se sastoji od sustavnog nabrajanja svih mogućih kandidata za rješenje i provjere odgovara li svaki kandidat rješenju problema. Algoritam iscrpne pretrage će uvijek pronaći rješenje problema ako rješenje postoji, ali kako raste veličina problema algoritam postaje neefikasan. Algoritam iscrpnog pretraživanja se koristi kada je veličina problema ograničena.

Pseudokod iscrpnog pretraživanja je prikazan algoritmom 1.

Algoritam 1. Iscrpno pretraživanje

Input : problem P

Output : kandidati za rješenje problema c

1. c ← *first* (*P*)
2. **while** c ≠ λ **do**
3. **if** *valid* (*P, c*)
4. *output* (*P, c*)
5. c ← *next* (*P, c*)
6. end **while**

Da bi se primijenio algoritam iscrpne pretrage za određeni problem, trebaju se definirati četiri procedure : *first, next, valid* i *output*. Ove procedure kao ulazni parametar uzimaju problem *P*, te rade sljedeće :

1. *first (P)* – generira prvog kandidata za problem *P*
2. *next (P, c)* – generira sljedećeg kandidata za problem *P* temeljem trenutnog kandidata *c*
3. *valid (P, c)* – provjerava je li kandidat *c* rješenje problema *P*
4. *output (P, c)* - ispisuje kandidata *c* kao rješenje problema *P*

Procedura *next* mora javiti kada više nema kandidata za problem *P*. Prikladan način da javi da više nema kandidata je da se vrati prazni kandidat kojeg označavamo s λ. Na jednak način, i procedura *first* vraća λ ako nema prvog kandidata.

Pogledajmo jedan primjer iscrpnog pretraživanja jednog samog broja u nizu od parova brojeva i jednog samog broja. Instanca problema *P* je niz parova brojeva među kojima je i jedan sami broj. Algoritam uzima prvi element niza i uspoređuje ga sa svim ostalim elementima. Dakle, procedura *first* (*P*) će vratiti prvog kandidata za sami broj u nizu ako postoji, inače vraća λ. Sljedećeg kandidata vratit će funkcija *next* (*P, c*), a kada dođe do zadnjeg kandidata, vratit će λ. Procedura *valid* (*P,c*) će vratiti istinu ako i samo ako je *c* ujedno i jedini sami broj u zadanom nizu. Procedura *output* ispisuje kandidata *c* ako je kandidat *c* ujedno rješenje.

1.4.2. Unatražno pretraživanje

Unatražno pretraživanje je općeniti algoritam koji se temelji na sistematskom pretraživanju mogućih rješenja nekog računalnog problema. Kroz postupak pronalaženja pravih rješenja, algoritam stvara moguće kandidate dok istodobno uvjetovano odbacuje potencijalne kandidate rješenja koji ne mogu više na nikakav način stvoriti valjano rješenje. Samim time broj mogućih rješenja se smanjuje što omogućava brži pronalazak pravih rješenja u odnosu na iscrpni algoritam.

Pozitivna strana uporabe algoritma unatražnog pretraživanja jest to da se postavljanje početnih uvjeta dopušta prije izvođenja algoritma, čime se u samom početku smanjuje broj mogućih rješenja. Glavni zahtjev algoritma je taj da je postojanje hijerarhije u sistematskoj proceduri traženja rješenja obavezno, tako da se skup rješenja koji ne zadovoljava određeni uvjet odbaci prije nego se stvori potencijalno konačno rješenje. Zbog toga ispitivanje i stvaranje rješenja prati model stablastog prikaza.

U algoritmu 2. prikazan je pseudokod algoritma unatražnog pretraživanja, te je prikazana ideja odbacivanja kandidata i vraćanja unatrag.

Algoritam 2. Unatražno pretraživanje

Input : problem *P* i kandidat *c*

```
1.  function backtracking (P, c)
2.  if reject (P, c) then
3.      return
4.  if accept (P, c) then
5.      output (P, c)
6.  s ← first (P, c)
7.  while s ≠ λ do
8.      backtracking (P, s)
9.      s ← next (P, c)
10. end while
```

U algoritmu 2 koristi se šest pomoćnih procedura koje primaju instancu problema *P*:

1. *reject (P,k)* – odbacuje kandidata *k* koji ne vodi do rješenja problema *P*
2. *accept (P,k)* – prihvaća kandidata *k* za rješenje problema *P*
3. *root (P)* – generira prvog kandidata za problem *P*
4. *first (P,k)* – generira prvog kandidata za problem *P* na temelju trenutnog kandidata *k*
5. *next (P,k)* – generira sljedećeg kandidata za problem *P* na temelju trenutnog kandidata *k*
6. *output (P,k)* – ispisuje kandidata *k* kao rješenje problema *P*

Osnovni algoritam unatražnog pretraživanja je rekurzivan i sastoji se od odbacivanja kandidata ako ne vodi do rješenja, čime se rekurzija zaustavlja i traži se novi prvi kandidat. Zatim se rekurzivno provjerava je li rješenje novi kandidat, a ako se ne odbaci, onda se generira sljedeći kandidat.

Unatražno pretraživanje možemo pojasniti na problemu pretraživanja svih parova prirodnih brojeva čija je razlika jednaka prirodnom broju 7. Procedura *root* treba generirati prvog kandidata za problem, te u ovom primjeru vraća praznu n-torku (). Procedura *first* generira prvog kandidata na temelju trenutnog. Kako nemamo trenutnog kandidata, procedura nadodaje 1 za zadnjeg člana, odnosno prvi kandidat je (1). Zatim će prvi kandidat od (1) biti (1, 1). S obzirom na to da kandidati moraju imati isključivo dva člana, procedura *first* će za prvog kandidata od (1, 1) vratiti λ. Generiranje sljedećeg kandidata povećava prvog člana para za jedan, ako je njihova razlika manja od 7. Dakle, sljedeći kandidat od (1, 1) je (2, 1), a sljedeći kandidat od (2, 1) je (3, 1). Postupak povećavanja se nastavlja sve dok razlika između dva člana ne bude jednaka prirodnom broju 7.

1.4.3. *Greedy* algoritam

Greedy algoritmi su vrlo jednostavni, te su kratkovidni u svom pristupu. *Greedy* algoritam je sličan algoritmu dinamičkog programiranja, ali razlika je u tome što rješenja potproblema ne moraju biti poznata u svakoj fazi, te umjesto toga odabir može biti pohlepan i odabrati ono što u datom trenutku izgleda najbolje.

Za izradu rješenja na optimalan način, algoritam treba sadržavati dva sljedeća skupa:

1. Prvi koji sadrži odabrane kandidate
2. Drugi koji sadrži odbijene kandidate

Greedy algoritam čini dobre trenutne odluke u nadi da će one rezultirati optimalnim i izvodljivim rješenjem. Koristi četiri procedure :

1. Proceduru koja provjerava pruža li odabrani kandidat rješenje
2. Proceduru koja provjerava izvodljivost
3. Selekcijsku proceduru koja izvještava koji je kandidat najobećavajućiji
4. Objektivnu proceduru koja se ne pojavljuje eksplicitno, ali koja daje vrijednost rješenja

Optimizacija problema se odvija uzevši u obzir problematiku te skup prepreka i objektivnu funkciju, pronalazeći izvodljivo rješenje za danu instancu problema za koju objektivna procedura ima optimalno rješenje.

Algoritmom 3. predstavljen je pseudokod za navedeni algoritam.

Algoritam 3. Greedy algoritam

Input: skup a koji sadrži n elemenata

```
1. function Greedy (a, n)
2. solution ← 0
3. for i to n do
4.         x ← Select (a)
5.         if Feasible (solution, x) then
6.                 solution ← Union (solution, x)
7. return solution
```

Greedy algoritam se najjednostavnije može pojasniti na primjeru problema trgovačkog putnika [6]. Heuristika *Greedy* algoritma za rješavanje tog problema je sljedeća : „Na svakoj razini, posjeti onaj ne posjećeni grad koji je najbliži trenutnom gradu.“ Rješenje možda neće biti, a i ne mora biti, nabolje rješenje, ali je svakako najjednostavnije i najsmislenije što se broja koraka tiče.

2. Algoritamsko rješavanje problema

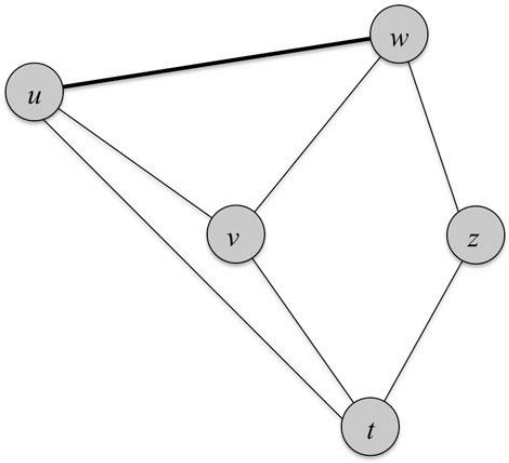
Rješenju problema slagalice se pristupa algoritamski, što znači da se prvo pojašnjava sami problem, tj. ulazni i izlazni parametri. Nakon što smo opisali problem, možemo oblikovati rješenje i prilagoditi algoritam. Naime, koristit će se poopćenje za zvijezdu u kojem je moguće dobiti rješenje za bilo koji povezan graf. Implementacija algoritama je napravljena u programskom jeziku Python, te su navedeni specifični koraci algoritma.

2.1. Razumijevanje problema

Kao što smo naveli ranije, problem novčića na zvijezdi je problem koji za dani ulaz vraća slijed poteza koji su doveli do postavljanja novčića na maksimalan broj vrhova. Slagalica je zapravo i određena samo početnim stanjem, te se ne zahtijeva nikakav daljnji ulaz.

Ulazni parametar je rječnik čiji su ključevi vrhovi, a vrijednosti skup vrhova incidentnih s vrhom iz ključa. Vrhove ćemo označavati malim slovima a,b,c. Bridove ćemo pokazati kao dva povezana vrha, na primjer c-d je oznaka za brid koji povezuje vrh c s vrhom d. Dva brida c-d i d-c smatraju se jednakim.

Na slici je prikazan graf s vrhovima t,u,v,w,t i bridovima u - w, u – v, v – w, v - t,t - z, z-w, u – t.

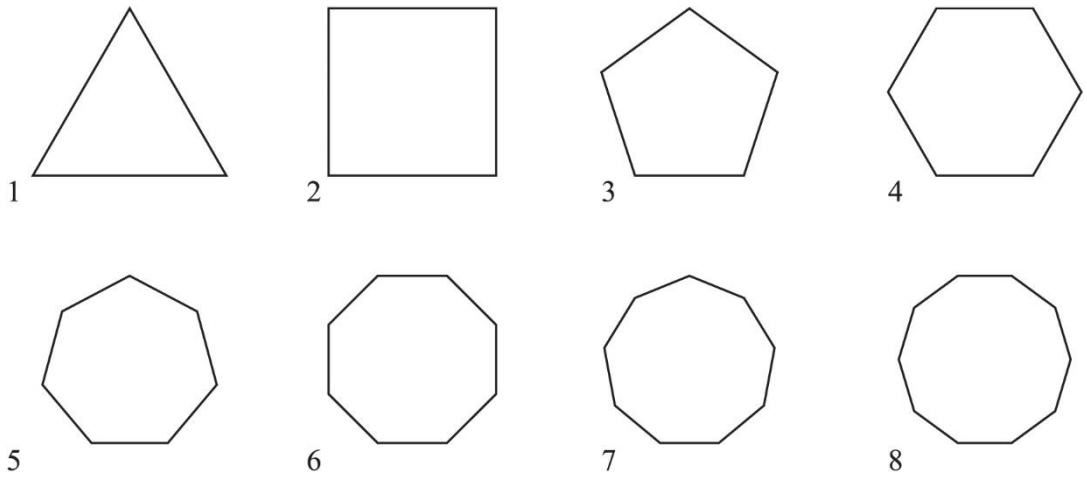


Slika 2.1 Graf s 5 vrhova i 7 bridova

Izlaz iz problema je slijed poteza koji su doveli do postavljanja novčića na maksimalan broj vrhova. Iako se dva brida smatraju jednaka, na primjer bridovi u – t i t – u se smatraju jednim bridom, za problem novčića na zvijezdi je jako bitan smjer veze između dva vrha. Naime, ako krenemo iz vrha t i postavimo novčić na vrh u, više nismo u mogućnosti šetati po bridu t - u, te isto tako nismo u mogućnosti više pomicati novčić s vrha u. Smjer veze je bitan jer se *Greedy* algoritam oslanja samo na njega, dakle, ako nastavimo, tj imamo novčić postavljen na vrhu u, idući potez nam je inicijalno staviti novčić u vrh z i kliznuti bridom do vrha t, te ga tamo i postaviti. Da smo nekim slučajem pogriješili, i postavili novčić s vrha t na vrh u, a brid zapisali kao u – t, smatralo bi se da je na vrhu t već postavljen novčić i ne bi bili u mogućnosti izvršiti potez z – t.

2.2. Matematička formulacija problema

Ulazni parametar za problem novčića na zvijezdi može se predstaviti kao neusmjereni graf G = (V, E) gdje je V skup vrhova (engl. *Vertex*), a E skup bridova (engl. *Edge*) grafa G. Na slici su prikazani neki od mogućih grafova kao ulaz u problem.



Slika 2.2 Mogući ulazi u problem novčića na zvijezdi

Neka je $\{v_1, v_2, ..., v_n\}$ skup vrhova, te neka su dvočlani skupovi oblika {u, v} elementi skupa E gdje je $u,v \in V$.

Rješenje problema za dani ulazni graf G je niz uređenih parova (c_i, c_j) koji predstavljaju potez, gdje je c_i prazni vrh na kojeg se stavio novčić, a c_j vrh na koji se dovukao novčić iz vrha ci, za koje vrijedi sljedeće :

— $c_i \in V$

— (c_i, c_j) – dva susjedna vrha povezana bridom, predstavljaju potez
— (v_1, ..., v_n) - uređena n-torka vrhova
— Neka je n ukupan broj vrhova. Ograničenje rješenja problema je to što se novčić ne može postaviti na svih n vrhova, nego na n – 1. Stoga je rješenje problema broj n – 1.

2.3. Oblikovanje rješenja problema

Rješenje problema novčića na zvijezdi je algoritam koji za zadani graf vraća slijed poteza koji su doveli do postavljanja novčića na maksimalan broj vrhova. S obzirom na to da su algoritmi pretraživanja pogodni za traženje optimalnog rješenja, sukladno s tim odabrali smo dva algoritma pretraživanja. Najjednostavniji algoritam je iscrpno pretraživanje koji generira sva moguća rješenja. Zatim se oblikuje algoritam unatražnog pretraživanja koji prilikom generiranja poteza odbacuje onaj potez koji ne vodi do rješenja. Kao posljednji algoritam oblikuje se *Greedy* algoritam koji ne spada u skupinu algoritama pretraživanja, već je jedna vrsta pohlepnog algoritma koji bira najbolje poteze u datom trenutku.

2.3.1. Iscrpno pretraživanje za problem novčića na zvijezdi

Kod iscrpnog pretraživanja, generiraju se sve moguće permutacije vrhova zadanog povezanog grafa i za svaku se provjerava poštuje li zadana pravila, te daje li rješenje. U algoritmu se koriste permutacije bez ponavljanja, jer se svi vrhovi moraju naći u rješenju. Algoritam iscrpnog pretraživanja uzima proizvoljan vrh iz skupa vrhova, te generira preostale vrhove, koji zajedno s prethodno uzetim proizvoljnim vrhom čine kandidata za rješenje. Nakon toga provjerava se je li skup bridova koji je kandidat za rješenje ujedno i rješenje algoritma. Inicijalno rješenje se postavlja na praznu listu, te nakon što se provjeri da kandidat daje rješenje, u rješenje se sprema uređeni par koji se sastoji od početnog vrha i kandidata koji čine rješenje.

Algoritam 4. Iscrpno pretraživanje

Input : povezan graf G = *(V, E)*

Output : [m/2] - maksimalan broj vrhova na koje je postavljen novčić, $((v_1, v_2),...,(v_{m-1}, v_m))$: $v_{i-1}, v_i \in V$ maksimalan vektor poteza

funkcija bruteforce(G)

moves ← $\bigcup_{\{u,v\}\in E}\{(u,v),(v,u)\}$

max_num ← 0

max_candidate ← $\vec{0}$

for each candidate ∈ permutacije(moves) **do**

visited ← ( )

candidates ← [ ]

for each i ∈ candidate **do**

candidate_next ← i

if candidate_next$_0$ ∉ visited **and** candidate_next$_1$ ∉ visited **then**

visited ← visited ∪ candidate_next$_1$

candidates ← candidates ∪ candidate_next

if [m/2] + 1 > max_num **then**

max_num ← [m/2] + 1

max_candidate ← candidates

return max_candidate, max_num

Složenost ovog algoritma iscrpne pretrage u svim slučajevima je *O*((n-1)!).

Pogledajmo jedan jednostavni primjer algoritma iscrpne pretrage. Algoritam generira sve permutacije od svih poteza. Primjera radi uzmimo da je jedna permutacija ((2, 7), (4, 7), (8, 3), (6, 1), (5, 2), (8, 5), (1, 4), (3, 8), (6, 3), (7, 4), (1, 6), (3, 6), (2, 5), (7, 2), (4, 1), (5, 8)). Ulaskom u petlju, prvi kandidat bi bio (2, 7). Slijedi provjera jesu li vrhovi 2 i 7 već posjećeni. S obzirom na to da je to prvi kandidat, lista posjećenih vrhova je prazna, stoga ti vrhovi nisu posjećeni i kandidat (2, 7) se sprema u listu rješenja, a vrh 7 u listu posjećenih vrhova. Zatim slijedi ista provjera za idući kandidat koji je (4, 7), međutim vrh 7 je posjećen, pa se prelazi na idućeg kandidata (8, 3), izvršavaju se provjere i na kraju se i on sprema u listu mogućih rješenja. Prethodno opisan postupak se izvršava za svakog kandidata svake permutacije, te se rješenje formira na način da se uzme jedna bilo koja najduža permutacija poteza. Jedno od mogućih rješenja je : [(5, 8), (2, 5), (6, 3), (1, 6), (7, 2), (4, 1), (7, 4)]

2.3.2. Unatražno pretraživanje za problem novčića na zvijezdi

Kod unatražnog pretraživanja, imamo jednu pomoćnu funkcije. Funkciju *rješenje* je rekurzivna funkcija koja zapravo pronalazi rješenje. Kao ulazne parametre, funkcija *rješenje* prima dvije prazne liste od kojih jedna predstavlja listu rješenja, a druga listu posjećenih vrhova.

Algoritam 5. Unatražno pretraživanje

Input : povezan graf G = *(V, E)*

Output : $((v_1, v_2),...,(v_{m-1}, v_m))$: $v_{i-1}, v_i \in V$ maksimalan vektor poteza

```
funkcija Backtracking(graph)
    all_points ← ∪ {v0, vm-1} ∈ V
    funkcija rjesenje(coins, max_rs)
        if len(coins) > len(max_rs) then
            max_rs ← coins
        if len(max_rs) >=  (len(graph) – 1) then
            return max_rs
    for c ∈ coins do
        p_coins ← ci
    for candidate ∈ permutacije (all_points – p_coins) do
        for p ∈ graph_candidate – p_coins do
            rs ← rjesenje(coins + [(candidate, p)] , max_rs)
            if len(rs) > len(max_rs) then
                max_rs ← rs
    return max_rs
return (rjesenje([ ], [ ]))
```

Pojasnimo algoritam na jednostavnom primjeru. Nakon što se postave svi vrhovi kandidati, funkcija *rjesenje* radi sljedeće : Za svakog kandidata iz liste permutacija vrhova kandidata provjera čini li rješenje zajedno s njegovim incidentnim vrhom, te ukoliko čini, rekurzivno ponavlja postupak sve dok se ne pronađe V – 1 kandidata. Jedno od mogućih rješenja bilo bi : [(1, 4),(6, 1), (2, 7), (5, 2), (8, 5), (3, 6), (3, 8)], koje je dobiveno na način : prvi kandidat iz liste permutacija kandidata je vrh 1, a vrh njemu incidentan je vrh 3, te oni zajedno mogu činiti rješenje. Funkcija se ponovno poziva, ali ovog puta u listi posjećenih ima uređeni par (1, 4) i kandidata 4, pa je sada prvi kandidat iz liste permutacija kandidata 6, a njemu incidentan ne posjećeni vrh je 1. Uređeni par (6, 1) također čini rješenje i sprema se u listu rješenja i u listu posjećenih. Postupak se rekurzivno ponavlja.

S obzirom na to da je algoritam rekurzivan, postoji $O(N^2)$ raznih mogućnosti, a svaki potez je zahtjeva $O(1+2+...+n)$ operacija za provjeru. Dakle, ukupna složenost je $O(N^{N+2})$.

2.3.3. *Greedy* algoritam za problem novčića na zvijezdi

Ulaz u *Greedy* algoritam je povezan graf G, dok je izlaz slijed poteza koji su doveli do postavljanja maksimalnog broja novčića. Na početku se, početni vrhovi postavljaju inicijalno na 1 i 2, te se inicijalizira prazan skup koji predstavlja sve vrhove na koje je već postavljen novčić. Zatim se rješenje postavlja na nulu. Procedura *Greedy* generira sve kandidate sve dok svi mogući kandidati nisu posjećeni. Zatim provjerava nalazi li se kandidat u skupu posjećenih vrhova, te ako nije, provjera je li taj kandidat trenutno najbolja opcija. Ako jest, u rješenje se sprema taj kandidat. Rješenje je zapravo i izlaz iz problema. Ukoliko je kandidat u skupu posjećenih vrhova, uzima se idući kandidat i na njega se primjenjuju pravila iz *Greedy* strategije koja je opisana ranije.

Algoritam 6. *Greedy* algoritam

Input : povezan graf G = *(V, E)*

Output : [m/2] - maksimalan broj vrhova na koje je postavljen novčić, $((v_1, v_2),...,(v_{m-1}, v_m))$: $v_{i-1}, v_i \in V$ maksimalan vektor poteza

funkcija Greedy (G)

x ← 1

y ← 2

direction ← 0

count ← 0

if $G_{x,0} \neq y$ **then**

    putanja ← 1

R ← [ ]

visited ← ()

while True **do**

    visited ← visited ∪ x

    candidate ← $G_{x,direction}$

    if candidate ∉ visited **then**

        candidate ← $G_{x,direction}$

    if candidate ∈ visited **then**

        if direction = 1 **then**

            candidate ← $G_{x,0}$

        if direction ≠ 1 **then**

            candidate ← $G_{x,1}$

    candidate_next ← $G_{candidate,\ direction}$

    if candidate_next = 1 **then**

        candidate_next ← $G_{candidate,\ direction}$

    if candidate_next ≠ 1 **then**

        if candidate_next ∈ visited **then**

            if direction = 0 **then**

                candidate_next ← $G_{candidate,\ 1}$

            if putanja ≠ 0 **then**

                candidate_next ← $G_{candidate,\ 0}$

    x ← candidate

if x ∈ visited **then**

zaustavi

if x ∉ visited **then**

count ← count + 1

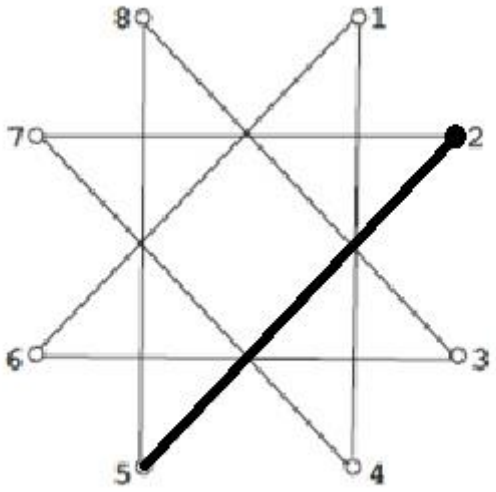
R ← R ∪ (candidate_next, candidate)

vrati R, count

Greedy algoritam u najboljem slučaju ima složenost $O(1)$ i to kada je graf prazan, a u svim ostalim slučajevima ima složenost $O(N^2)$. Povezujući jedan vrh s ostalima, imamo složenost $O(N)$. Najbliži susjed vrhu v je najbliži vrh v. Dakle, treba spojiti n vrhova s ostalim vrhovima, pa se na taj način dođe do $O(N^2)$.

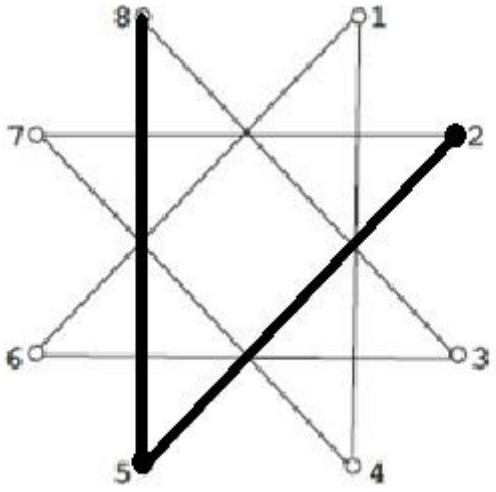
2.3.4. Primjer rješavanja problema *Greedy* strategijom

Ulaz u originalni problem je jedan neusmjereni graf u obliku zvijezde prikazan na slici 1.4. Za početak se uzima jedan proizvoljni vrh od osam mogućih vrhova, te se na njega privremeno postavlja novčić. Primjera radi, uzima se proizvoljni vrh 5, koji se spaja s jednim od dva povezana vrha, također drugi vrh se bira proizvoljno, u ovom slučaju biramo vrh 2. Trenutna povezanost je prikazana na slici 2.3.



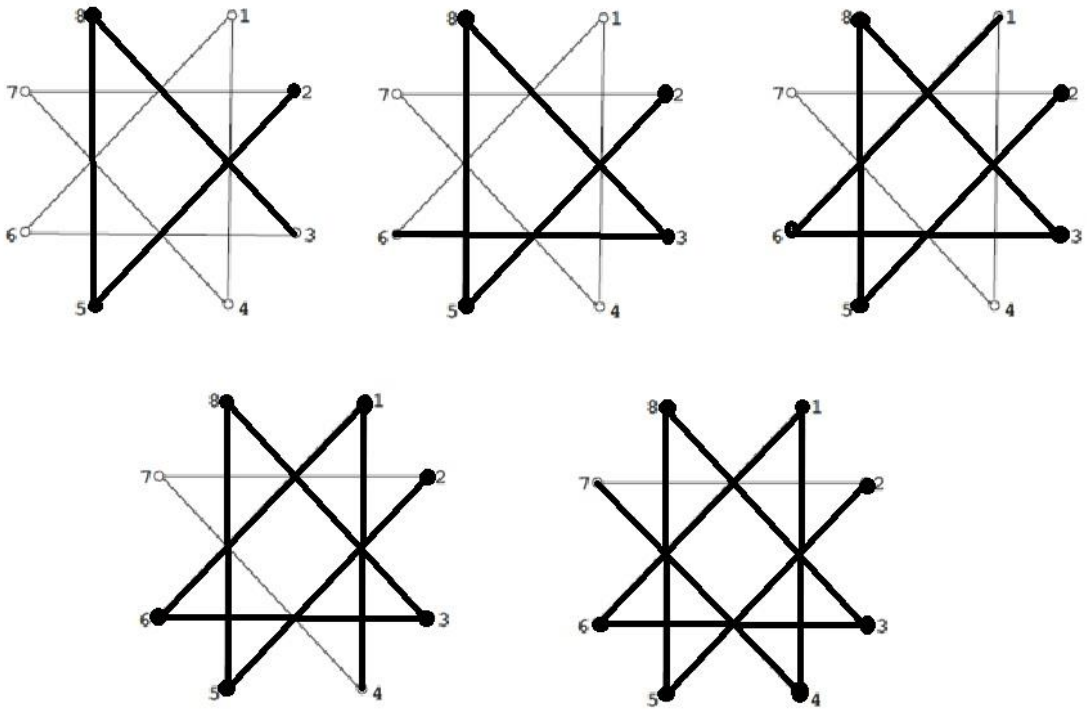
Slika 2.3 Prikaz povezanosti nakon prvog koraka

Nakon što smo postavili novčić na vrh 2, više nismo u mogućnosti taj novčić pomicati niti ijedan drugi novčić inicijalno postavljati na taj vrh. Također, i brid koji spaja vrhove 5 i 2 je zauzet brid. Iduće što trebamo napraviti, što nam nalaže *Greedy* strategija je da sljedeći novčić inicijalno postavimo na preostali vrh koji je povezan s vrhom 5, te ga preko brida postavimo na vrh 5. Ta radnja prikazana je slikom 2.4.



Slika 2.4 Prikaz povezanosti nakon dva koraka

Sada imamo dva vrha i dva brida koja su zauzeta, te se za ostatak moramo poslužiti s dozvoljenim, slobodnim vrhovima i bridovima. Nakon toga, idući novčić se postavlja po istom principu po kojem se i postavio na vrh 5. Dakle, novčić se inicijalno postavlja na vrh 3 te putem brida se postavi na vrh 8. Općenito, novčić se postavlja na onaj vrh na kojem je prethodni bio privremeno postavljen. Ostatak rješenja prikazan je slikom 2.5.



Slika 2.5 Prikaz rješenja problema po koracima

Ono što se da primijetiti je da na posljednjoj slici koja predstavlja kraj i rješenje problema, vidimo jedan vrh i jedan brid koji nisu zauzeti. Ako bolje promotrimo problem i pravila rješavanja, očito je da ne možemo postaviti novčić na baš sve vrhove, jer za posljednji nemamo više ni jedan slobodan vrh s kojeg bi krenuli.

Dakle, jedno od mogućih rješenja problema koje se dobiva Greedy-jevom strategijom, ujedno rješenje ovog primjera jest : $5 \rightarrow 2$, $8 \rightarrow 5$, $3 \rightarrow 8$, $6 \rightarrow 3$, $1 \rightarrow 6$, $4 \rightarrow 1$, $7 \rightarrow 4$, odnosno sedam, jer se na sedam vrhova postavio novčić.

2.4. Implementacija rješenja problema novčića na zvijezdi u Pythonu

Svi algoritmi koji su korišteni za rješavanje problema novčića na zvijezdi implementirani su u programskom jeziku Python. Struktura podataka koja se koristi za implementaciju ulaznog parametra, odnosno proizvoljnog povezanog grafa je rječnik. Ključevi u rječniku predstavljaju vrhove ulaznog grafa, a vrijednosti tih ključeva predstavljaju skup vrhova incidentnih s vrhom ključa.

Rječnik za graf u obliku zvijezde u originalnom problemu izgleda na sljedeći način :

{ 1: [4, 6], 2: [5, 7], 3: [6, 8], 4: [1, 7], 5: [2, 8], 6: [1, 3], 7: [2, 4], 8: [3, 5] }.

Implementacije algoritama iscrpne pretrage u Pythonu je prikazana je kodom 2.1.

```
def brute_force(graph):
    moves=set()
    for u, neighbors in graph.items():
        for v in neighbors:
            moves.add((u, v))

    max_num = 0
    max_candidate=()

    for candidate in permutations(moves):
        visited = set()
        candidates = []
        for i in candidate:
            cand_next=(i)
            if cand_next[0] not in visited and cand_next[1] not in visited:
                visited.add(cand_next[1])
                candidates.append(cand_next)
                cand_num=len(candidates)
                if cand_num > max_num:
                    max_num = cand_num
                    max_candidate = candidates

    return max_candidate, max_num
```

Kôd 2.1 Implementacija algoritma iscrpnog pretraživanja

Dakle, generiraju se sve permutacije iz skupa mogućih poteza, te se iz svake permutacije generiraju kandidati, koji u konačnici čine rješenje. Nakon što se izgeneriraju sve permutacije, i sve liste mogućih rješenja, za konačno rješenje uzima se bilo koja najduža lista.

Kodom 2.2 prikazan je algoritam unatražnog pretraživanja.

```
def Backtracking(graph):

    ALL_POINTS = set(list(graph)[:-1])
    def rjesenje(coins, max_rs):
        if len(coins) > len(max_rs):
            max_rs = coins

        if len(max_rs) >= len(graph)-1:
            return max_rs

        p_coins = {c[1] for c in coins}
        for candidate in permutation(list(ALL_POINTS - p_coins)):
            for p in graph[candidate] - p_coins:
                rs = rjesenje(coins+[(candidate, p)], max_rs)
                if len(rs) > len(max_rs):
                    max_rs = rs

        return max_rs

    return(rjesenje([],[]))
```

Kôd 2.2 Implementacija algoritma unatražnog pretraživanja

Kod algoritma unatražnog pretraživanja, srž algoritma je rekurzivna funkcija *rjesenje*, koja u konačnici i pronalazi rješenje. U funkciji *rjesenje* generiraju se svi kandidati koji čine moguće rješenje, a rješenje se postiže onda kada je broj kandidata jednak V – 1, odnosno broju vrhova ulaznog grafa umanjen za jedan.

Kodom 2.3 prikazan je *Greedy* algoritam.

```
def Greedy(graph):
    x=1
    y=2
    direction = 0
    count = 0
    if graph[x][0] != y:
        direction = 1
    R =[]
    visited = set()
    while (True):
        visited.add(x)
        candidate = graph[x][direction]
        if candidate not in visited:
            candidate = graph[x][direction]
        else:
            if direction == 1:
                candidate = graph[x][0]
            else:
                candidate = graph[x][1]
        candidate_next = graph[candidate][direction]
        if candidate_next == 1:
            candidate_next = points[candidate][direction]
        else:
            if candidate_next in visited:
                if direction == 0:
                    candidate_next = graph[candidate][1]
                else:
                    candidate_next = graph[candidate][0]

        x = candidate
        if x in visited:
            break
        else:
            count += 1
        R.append((candidate_next, candidate))

    return R, count
```

Kôd 2.3 Implementacija *Greedy* algoritma

Greedy algoritam implementiran je doslovno po principu *Greedy* strategije, koja je opisana prethodno. Inicijalno se za početni vrh uzima proizvoljni vrh iz skupa vrhova zadane zvijezde. Za *Greedy* algoritam u potpunosti je nebitno s kojim se vrhom krene, algoritam uvijek daje rješenje. Nakon što se novčić trenutno postavi na proizvoljno odabrani vrh, slučajnim odabirom odabire jednog od dva povezana vrha, na njega postavlja novčić i sprema ga u listu posjećenih vrhova. Tu završava slučajno biranje vrhova i kreće se po zadanoj heuristici *Greedy* algoritma. Nakon što je novčić postavljen, u listi posjećenih nalazi se samo jedan vrh, ali samim tim posjećena su dva brida koja više nisu upotrebljiva, što znači da se idući novčić mora postaviti na mjesto gdje je bio trenutno postavljen prethodni. Kako s vrha na kojem je već postavljen ne

možemo krenuti niti na njega više postaviti novčić, kreće se preostalog povezanog vrha, i novčić se uistinu postavlja na vrh na koji je privremeno bio postavljen prethodni. To se izvršava za svaki vrh.

2.5. Analiza izvršavanja implementiranih algoritama

Algoritmi implementirani za rješavanje problema novčića na zvijezdi su algoritmi pretrage. Samim time jasno je da će povećanjem broja vrhova i bridova ulaznog grafa porasti kako vrijeme izvršavanja algoritma tako će i prostorna složenost. Povećanjem broja vrhova raste i broj kandidata, pa tako raste i broj provjera.

Broj vrhova grafa	Broj mogućih poteza	Broj provjerenih mogućnosti	Broj pronađenih rješenja	Vrijeme utrošeno na provjere u sekundama
0	0	0	0	0.0
1	2	2	2	0.0
2	4	13	3	0.0
3	6	59	12	0.0
4	8	103	10	0.0
5	10	1 890	750	0.016
6	12	1 956	516	0.016
7	14	14 560	3 388	0.14
8	16	48 144	1 024	0.47
9	18	743 094	95 724	7.1
10	20	3 874 480	148 420	3.6e+01

Tablica 2.1 Prikaz rezultata mjerenja za algoritam unatražnog pretraživanja

U tablici 2.1 prikazani su rezultati mjerenja vremenske složenosti algoritma unatražnog pretraživanja. Iz tablice se može zaključiti da između dva grafa s različitim brojem vrhova, ako je broj tih vrhova malen, i njihova razlika mala, vremenska složenost je jednaka, odnosno povećanje je ne znatno. Ali, što raste broj vrhova, samim time raste i vremenska složenost, te postaje sve očitija.

U tablicama 2.2 i 2.3 prikazani su rezultati mjerenja vremenske složenosti za ostala dva algoritma, preciznije u tablici 2.2 se nalaze rezultati mjerenja algoritma iscrpne pretrage, a u tablici 2.3 *Greedy* algoritma.

Broj vrhova ulaznog grafa	Vrijeme u sekundama
1	0.00015
2	0.00019
3	0.00029
4	0.00079
5	0.00430
6	0.01736
7	0.46822
8	14.1499
9	59.3174

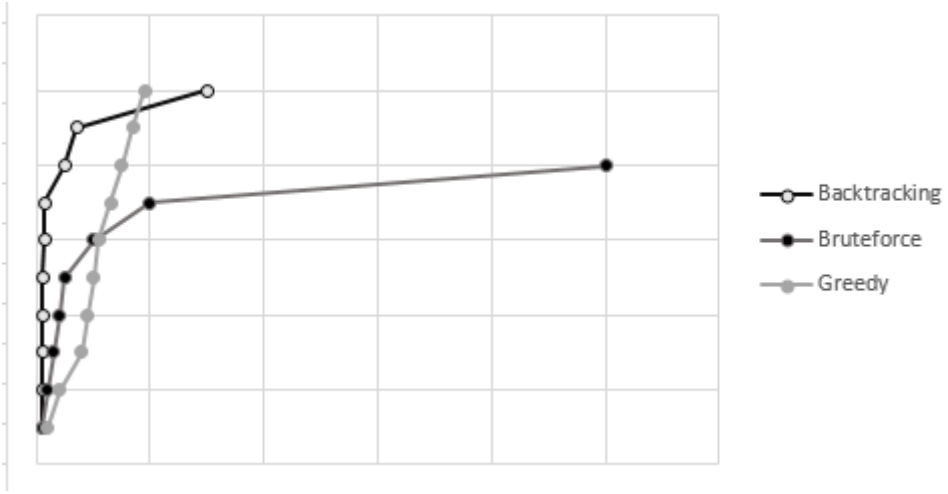
Tablica 2.2 Prikaz rezultata mjerenja vremena algoritma iscrpne pretrage

Broj vrhova ulaznog grafa	Vrijeme potrebno za izvršavanje programa u sekundama
1	0.015
2	0.036
3	0.054

4	0.074
5	0.095
6	0.12
7	0.14
9	0.16
8	0.18
10	0.21

Tablica 2.3 Prikaz rezultata mjerenja vremenske složenosti za *Greedy* algoritam

Na slici 2.6 prikazan graf vremenske složenosti za sva tri algoritma, s kojeg se može bolje vidjeti da vremenska složenost raste proporcionalno s povećanjem broja vrhova



Slika 2.6 Graf rasta vremenske složenosti povećanjem broja vrhova

Zaključak

Tema ovog završnog rada je bila opisati problem i način rješavanja problema novčića na zvijezdi. Problemi poput problema novčića na zvijezdi poznata su stoljećima. Problem je usko vezan s teorijom grafova iz matematičkog aspekta pa je problem osim što je zanimljiv mnogim informatičarima zbog mnogobrojnih načina rješavanja, zanimljiv i matematičarima zbog povezanosti s teorijom grafova.

U ovom radu opisan je postupak rješavanja problema novčića na zvijezdi za bilo koji povezani graf. Oblikovana su tri algoritma, od kojih su dva algoritma algoritmi pretraživanja, a jedan je oblik pohlepnog algoritma. Svi algoritmi za ulaz primaju rječnik koji predstavlja način povezanosti grafa, a za izlaz vraća broj vrhova na koje se postavio novčić. Algoritmi su implementirani u programskom jeziku Python

Literatura

[1] Wikipedia, Octagram, 2017, URL https://en.wikipedia.org/wiki/Octagram

[2] Wikipedia, Eight queens puzzle, 2017, URL https://en.wikipedia.org/wiki/Eight_queens_puzzle

[3] Darko Veljan. *Kombinatorika s teorijom grafova,* Školska knjiga, Zagreb, 1989

[4] Wikipedia, Brute-force search, 2017, URL https://en.wikipedia.org/wiki/Brute-force_search

[5] Wikipedia, Backtracking, 2017, URL https://en.wikipedia.org/wiki/Backtracking

[6] Wikipedia, Greedy algorithm, 2017, URL https://en.wikipedia.org/wiki/Greedy_algorithm

[7] Wikipedia, NP (complexity), 2017, URL https://en.wikipedia.org/wiki/NP_(complexity)

[8] Logic puzzle, 2017, URL https://en.wikipedia.org/wiki/Logic-puzzle

[9] Octogram, Acorn electron world, 2014, URL http://www.acornelectron.co.uk/eug/71/a-octa.html

[10] Henry Dudeney, 2017, URL https://hr.wikipedia.org/wiki/Henry_Dudeney

Sažetak

Poopćenje problema novčića na zvijezdi je problem pronalaska maksimalnog broja postavljenih novčića u bilo kojem neusmjerenom grafu, ne samo na zvijezdi. Pošto se rješenje problema može opisati šetnjama na grafu uz određena pravila, a graf je jedna od najčešćih matematičkih struktura za opisivanje prostora stanja, onda možemo koristiti algoritme pretraživanja. Prikazan je postupak rješavanja korištenjem algoritama pretrage. Algoritmi pretraživanja su implementirani u Pythonu. Rezultati njihovog izvršavanja su analizirani i objašnjena je složenost implementiranih algoritama.

Ključne riječi: Problem novčića na zvijezdi, algoritmi pretraživanja.

Summary

Generalization of coins on a star problem is problem of finding maximum number of coins on any undirected graph, not only on a star. Since a problem solution can be described by the walks on the graph with certain rules and the graph is one of the most common mathematical structures for describing the state space, then we can use search algorithms. The solving process is displayed using search algorithms. Search algorithms are implemented in Python. The results of their execution are analyzed and the complexity of implemented algorithms is explained.

Keywords: Coins on a star problem, search algorithms.