Sveučilište J.J. Strossmayera u Osijeku

Filozofski fakultet

Preddiplomski studij Informatologije

Milan Balać

# Rekurzivni algoritmi

Završni rad

Mentor: izv. prof. dr. sc. Gordana Dukić
Sumentor: dr. sc. Anita Papić, viša asistentica

Osijek, 2014.

## Sažetak:

U radu je obrađena tema rekurzije i rekurzivnih algoritama. Najprije se pojašnjava pojam algoritma, te se opisuje povijest njegova nastanka kao i al Khowarizmijeva ideja na kojoj su zasnovani algoritmi. Uz to se navode tri definicije koje prikazuju ključne karakteristike svih algoritama, neovisno o području njihove primjene. Pojašnjena je analiza algoritama te što je potrebno uzeti u obzir prilikom analize i što se njome postiže. Zatim se prolazi klasifikacija algoritama, najopćenitija prema tipovima, te nešto složenije klasifikacije algoritama prema implementaciji, metodologiji dizajna i kompleksnosti. Nadalje, obrađuje se pojam rekurzije, te je prikazan jednostavan primjer rekurzivnog algoritma kako bi se razumjelo što rekurzija radi, te kako bi kasniji algoritmi bili jasniji. Također su navedeni osnovni principi i dijelovi rekurzije koje je moguće vidjeti kod svakog od navedenih rekurzivnih algoritama. U idućem poglavlju navedeno je pet primjera rekurzivnih algoritama, a to su algoritmi za izračunavanje faktorijela, Fibonaccijeve brojeve, rješenje problema 8 kraljica, ispunjavanje sudokua te Hanojske tornjeve. Za svaki algoritam pojašnjen je najprije problem koji rješava, prikazan primjer koda, te je navedeno jedno ili više mogućih rješenja. Zaključuje se kako rekurzija predstavlja dobar alat u rješavanju problema njihovim pojednostavljivanjem, te kako je kao metoda dovoljno jednostavna za uvođenje u svijet programiranja.

**Ključne riječi:** algoritmi, rekurzija, rekurzivni algoritmi

## Sadržaj:

# 1. UVOD

U radu se obrađuju rekurzivni algoritmi. S tim u svezi definiraju se pojmovi kao što je algoritam, rekurzija, iteracija i sl. Nadalje, u radu se predstavljaju neki odabrani poznatiji algoritmi kao što su faktorijeli, Fibonaccijevi brojevi, problem N kraljica, sudoku te Hanojski tornjevi koji se za rješavanje problema koriste metodom rekurzije.

U drugom poglavlju upoznaje se Ja'far Mohammed ibn Musa al Khowarizmi koji se smatra ocem algoritama te su opisane njegove osnovne ideje na kojima se zasnivaju algoritmi. Predstavlja se algoritam, koji je u svojoj osnovi određeni broj koraka prema rješenju nekog problema. Zatim se kroz tri definicije algoritma ukazuje na ključne karakteristike svih algoritama kao što su jednoznačnost, razumljivost i preciznost instrukcija, djelotvornost i učinkovitost algoritma te konačnost instrukcija i vremena potrebnog za izvršavanje algoritma. Treće poglavlje bavi se različitim načinima klasifikacije algoritama od najopćenitije podjele na računalne algoritme, algoritme za rješavanje problema, algoritme logičkog mišljenja, te recepte i upute, do složenijih klasifikacija algoritama prema implementaciji, metodologiji dizajna i kompleksnosti. Nadalje, u četvrtom poglavlju upoznaje se s pojmom rekurzije koja se koristi u rješavanju odabranih algoritama. Naime, rekurzija je metoda kojom se problem razbija na jednostavnije probleme koje je samim time lakše riješiti, te se rješavanjem tih malih probleme ide prema rješenju glavnog problema. Zatim se navode jednostruka rekurzija, višestruka rekurzija, indirektna rekurzija, te anonimna rekurzija koje predstavljaju različite tipove rekurzije. Potom se pojašnjava osnovna ideja rekurzije kao metode rješavanja problema i navode se glavne karakteristike rekurzivnih algoritama, a to su osnovni slučaj, napredak, pravilo projektiranja, pravilo neponavljanja te elegantnost koda, ali i duže izvođenje. U nastavku se na primjeru jednostavnog rekurzivnog algoritma dodatno pojašnjavaju definicije i karakteristike rekurzije, te se naposljetku provjerava ima li navedeni algoritam osnovne dijelove rekurzije. Peto poglavlje posvećeno je obradi pet navedenih algoritama, te je svaki od njih obrađen zasebno. Za svaki algoritam najprije je predstavljen i opisan problem, zatim je opisan kod algoritma i naposljetku se opisuje metoda i primjena algoritma na rješavanje problema. Najveća pažnja posvećena je Hanojskim tornjevima čije je rješenje najprije ispitano praktično a zatim pokretanjem algoritma na računalu.

Naposljetku, u zaključku se navode neke od osnovnih ideja navedenih kroz rad te se osvrće na rekurziju kao metodu koja svojom jednostavnošću i pristupačnošću može služiti kao dobar uvod u svijet programiranja.

# 2. POVIJEST I DEFINICIJA ALGORITAMA

Ocem algoritama smatra se perzijski matematičar Ja'far Mohammed ibn Musa al Khowarizmi[1]. On je svojim radom dao doprinos kako matematici, tako i geografiji, astronomiji te kartografiji. Al-Khowarizmi je smatrao kako se svaki matematički problem može „razbiti“, odnosno pojednostaviti raščlanjivanjem na manje korake i na taj način olakšati. Upravo to raslojavanje problema u manje, jednostavnije probleme je osnovni princip algoritama. Sama riječ „algoritam“ nastala je kombinacijom njegova prezimena i izraza „algorizmi“ koji se nalazio pored svakog aritmetičkog pravila u latinskim knjigama. Al-Khowarizmijevi principi najprije su pronašli svoje mjesto u matematici, no razvojem tehnologije dodijeljena im je važna pozicija i u računarstvu.[2] Moderno računarstvo je zasnovano na algoritmima – računalni programi su u svojoj suštini sačinjeni od njih, odnosno, rade tako što izvršavaju skupove instrukcija u unaprijed određenom slijedu. Uz to, al Khowarizmijevi principi „razbijanja“ problema, stari preko tisuću godina, i danas se primjenjuju u poboljšavanju brzine i efikasnosti računalnih programa.[3] Iz navedenoga da se zaključiti kako je algoritam skup koraka, odnosno instrukcija, koji vode rješenju nekog problema, no ne postoji uniformna definicija algoritama.

Algoritme je moguće promatrati iz različitih perspektiva, stoga i ne čudi mnoštvo definicija. Vrlo je vjerojatno kako će ih velik broj ljudi povezati s matematikom ili računarstvom, što nije greška, ali je ograničavajuće. Algoritmi predstavljaju korake koji vode rješenju nekog problema, otuda tolika primjenjivost i mnoštvo definicija jer se na probleme ne nailazi samo u matematici i računarstvu. Odabrane su tri definicije koje ilustriraju spektar perspektiva iz kojih je moguće promatrati algoritme, ali i omogućuju povlačenje paralela između definicija koje na prvi pogled opisuju različite pojmove. U prvoj definiciji algoritmi predstavljaju niz operacija koje se provode sustavno kako bi dale rezultat u ograničenom broju koraka. Nadalje, navodi se kako algoritmi ne moraju biti izraženi u određenom obliku ili jeziku navodeći recepte, upute za sastavljanje i sl. kao primjere algoritama.[4] Dok je prva definicija šira, druga je znatno uža i definira algoritme iz jedne perspektive, u ovom slučaju računarstva, kao bilo koju dobro definiranu računalnu proceduru koja uzima određenu vrijednost, ili skup vrijednosti, kao input, odnosno ulaz, i proizvodi vrijednost, ili skup vrijednosti, kao output, odnosno izlaz.[5] Treća definicija se može

[1] Hoško, Tomislav. Strukture podataka i algoritmi: priručnik. Zagreb: Algebra, 2009. str. 8
[2]Usp. Hoško, Tomislav. Nav. Dj. str. 8
[3] Usp. World of Computer Science. Detroit: Gale Group, cop. 2002. str. 14
[4] Usp. Isto
[5]Cormen, Thomas H. Introduction to Algorithms. Cambridge [etc.]: The MIT Press: McGraw-Hill Book Company, 2000. str. 1

promatrati kao kombinacija prethodne dvije: „algoritam je konačan skupa preciznih, razumljivih i jednoznačnih instrukcija koje djelotvorno i učinkovito dovode do rješenja u konačnom vremenu.“[6] Dakle, algoritmi ne sačinjavaju samo računalne programe nego se mogu promatrati i kao dio ljudskog misaonog procesa, odnosno koraka za neku radnju koju čovjek obavlja u svakodnevnom životu. Podudarnosti između recepta i računalnog algoritma moguće je pronaći i u raščlanjivanju zadnje navedene definicije na sedam ključnih dijelova. U definicije se navodi kako *instrukcije moraju biti jednoznačne*, odnosno moraju biti uvijek iste i izvoditi se na isti način; *instrukcije moraju biti razumljive*, jasne i ljudima i računalima; uz to, *instrukcije moraju biti precizne*; *algoritam mora biti djelotvoran*, dakle, mora dati neki rezultat; *algoritam mora biti učinkovit*, odnosno, mora dobro raspolagati memorijom i taktom procesora, kao i vremenom; nadalje, *broj instrukcija mora biti konačan kako bi se uopće mogao zapisati*; i u konačnici *vrijeme potrebno za izvršavanje mora biti konačno kako bi algoritam imao smisla*.[7] I dok je očito kako se upute najprije odnose na računalne algoritme nemoguće je u njima ne pronaći primjenjivost na svakodnevne radnje i probleme.

Promatraju li se ipak algoritmi striktno iz perspektive računarstva radi se o dinamičnim dijelovima računalnih programa, koji, sa strukturama podataka kao statičnim dijelovima, čine srž programa. Velik izazov u izradi nekog računalnog programa predstavlja odabir adekvatnog algoritma. Može se pretpostaviti kako postoje algoritmi koji obavljaju iste funkcije na različite načine, odnosno korištenjem više ili manje resursa, što traje duže ili kraće. Za odabir odgovarajućeg algoritma postoje dvije mogućnosti. Prva mogućnost bila bi isprobati sve varijante nekog algoritma praktično što je izrazito dugotrajan i mukotrpan proces. Druga, nešto lakša, mogućnost je algoritme analizirati, odnosno predvidjeti resurse potrebne za njihovo provođenje – memoriju, propusnost i sl., međutim, najčešće se analizira vrijeme.[8] Na ovaj način moguće je odabrati najpogodniji/najefikasniji algoritam, ili, u najgorem slučaju, izdvojiti bolje algoritme od lošijih i kroz idućih nekoliko koraka analize doći do onog koji je u određenoj situaciji najpogodniji.[9] Uz to, analiza omogućuje određen oblik klasifikacije algoritama, prema njihovoj efikasnosti. Međutim, podjela algoritama znatno je obuhvatnija i kompleksnija od samo jednog načina podjele, stoga je toj temi dodijeljeno zasebno poglavlje.

[6] Usp. Hoško, Tomislav. Nav. Dj. str. 9
[7] Usp. Hoško, Tomislav. Isto
[8] Usp. Cormen, Thomas H. Nav. Dj. str. 6
[9] Usp. Hoško, Tomislav. Nav Dj. str. 13

# 3. KLASIFIKACIJA ALGORITAMA

Jednostavnim nabrajanjem tipova algoritama poput rekurzivnih algoritama, „brute force“ („na silu“) algoritama, algoritama za unatražno pretraživanje, dinamičkih algoritama, pohlepnih algoritama itd. nije moguće steći znanje ili uvid u način njihova funkcioniranja. Stoga su osmišljene različite podijele, odnosno klasifikacije algoritama, svaka s određenim prednostima, koje ih svrstavaju u smislene cjeline. Jedna od podjela je na *računalne algoritme*, *algoritme za rješavanje problema*, *algoritme logičkog mišljenja*, te *recepte i upute*.[10] Radi se jednostavnoj podijeli koja algoritme svrstava u različite cjeline ovisno o području u kojem se primjenjuju. Znatno točnije i kompleksnije podjele algoritama bile bi podjela algoritama *prema implementaciji*, podjela algoritama *prema metodologiji dizajna* te podjela algoritama *prema kompleksnosti*.[11]

Klasifikacija algoritama prema implementaciji algoritme dijeli na *rekurzivne* ili *iterativne*, *logičke*, *serijske* ili *paralelne* te *deterministčke* ili *stohastičke*. **Rekurzivni algoritmi** pozivaju sami sebe dok ne dođu do rješenja, odnosno, od većeg problema stvaraju manje i jednostavnije potprobleme i tako rješavaju glavni. S druge strane, suprotno rekurzivnima, **iterativni** koriste petlje i stogove za rješavanje danih zadataka. Rekurzivni algoritmi mogu se konvertirati u iterativne i obrnuto, a ovisno o prirodi problema moguće je odabrati koji će se od dva načina rješavanja koristiti. Zatim, **logički** algoritmi polaze od pretpostavke da se algoritam može promatrati kao kontroliranu logičku dedukciju. Logička komponenta predstavlja činjenice koje se mogu koristiti u rješavanju problema, dok kontrola predstavlja način na koji se dedukcija primjenjuje na činjenice. **Serijski** algoritmi nose takav naziv jer većina računala ima samo jedan procesor te se radnje obavljaju jedna po jedna – serijski. S druge strane su **paralelni** koji koriste mogućnosti više procesora u računalu tako što probleme rascjepljuju na manje potprobleme od kojih svaki obrađuje jedan procesor. Nadalje, **deterministički** algoritmi rješavaju probleme unaprijed zadanim procesima dok **stohastički** do rješenja dolaze slučajnim odabirom u svakom koraku procesa.[12]

Drugi način klasifikacije algoritama je prema metodologiji dizajna, gdje svaka kategorija sadrži dodatne različite tipove algoritama. Algoritmi su podijeljeni na *„brute force“ („na silu“) algoritme*, *„podijeli pa vladaj“ algoritme*, *dinamičke algoritme*, *pohlepne algoritme*, *redukciju* tj.

[10] Usp. Witt, Pharaba. Different Kinds of Algorithms. URL: http://www.ehow.com/info_8292589_different-kinds-algorithms.html (2014-08-29)
[11] Usp. Classification of Algorithms. URL: http://www.scriptol.com/programming/algorithms-classification.php (2014-08-29)
[12] Usp. Classification of Algorithms. URL: http://www.scriptol.com/programming/algorithms-classification.php (2014-08-29)

*„transformiraj pa vladaj" algoritme, algoritme za sortiranje i pobrojavanje* te algoritme prema *probabilističkoj i heurističkoj paradigmi*.[13] **„Brute force" algoritmi** predstavljaju najneefikasnije algoritme. Oni se oslanjaju na snagu računala kako bi isprobali sve mogućnosti dok ne dođu do rješenja. Primjenjivi su uglavnom na probleme gdje je veličina inputa mala.[14] Zatim, **„podijeli pa vladaj" algoritmi** dijele problem na jednu ili više manjih instanci dok ne dođu do instance koja je lako rješiva. Jedna od varijanti ovih algoritama je *„smanji pa vladaj"* koji rješavaju identičan potproblem i primjenjuju to rješenje na veći problem.[15] **Dinamički algoritmi** primjenjuju se u slučajevima kad rješenje problema ovisi o rješenjima potproblema. Inače bi u svakom takvom slučaju bilo potrebno ponovno izračunavati vrijednosti zavisnih potproblema što je vremenski zahtjevno te se stoga koristi dinamičko programiranje gdje se „pamte" vrijednosti potproblema i mogu se koristiti kad god su potrebne čime se uklanja potreba za ponovnim izračunavanjem. Nadalje, **pohlepni** algoritmi u svakom koraku biraju trenutno najbolje rješenje i na kraju nude optimalno rješenje.[16] Zatim, **redukcija** poznata i pod nazivom **„transformiraj pa vladaj"** kojom se problem rješava tako što ga se transformira u drugi problem. Cilj ovih algoritama je pronaći najjednostavniju moguću transformaciju. **Algoritmi za sortiranje i pobrojavanje** koriste se između ostalog za pretraživanje grafova, i u ovu skupinu spadaju algoritmi za pretraživanje i unatražno pretraživanje. Konačno, prema probabilističkoj i heurističkoj paradigmi algoritme je moguće dodatno podijeliti na **probabilističke algoritme** koji neke izbore rade nasumično, **genetske algoritme** koji rješenja problema traže imitirajući evolucijske procese, stvarajući tako uzastopne generacije „rješenja" i emulirajući „opstanak najjačih", te **heurističke algoritme** koji daju približna rješenja problema u slučajevima kada pronalazak točnog rješenja nije praktičan.[17]

Algoritme je također moguće klasificirati prema kompleksnosti. Najčešće se u tu svrhu koristi linearna, logaritamska, polinomijalna te eksponencijalna kompleksnost. Linearni algoritmi su najmanje kompleksni i obično im je potrebno najmanje vremena za izvođenje. S druge strane su eksponencijalni koji su najkompleksniji, a negdje između se nalaze logaritamski i polinomijalni.[18]

---

[13] Usp. Isto
[14] Usp. Prabhu, Manohar. Algorithm: Types and Classification, 4.9.2011. URL: http://gonitsora.com/algorithm-types-and-classification/ (2014-08-29)
[15] Usp. Classification of Algorithms. URL: http://www.scriptol.com/programming/algorithms-classification.php (2014-08-29)
[16] Usp. Prabhu, Manohar. Algorithm: Types and Classification, 4.9.2011. URL: http://gonitsora.com/algorithm-types-and-classification/ (2014-08-29)
[17] Usp. Classification of Algorithms. URL: http://www.scriptol.com/programming/algorithms-classification.php (2014-08-29)
[18] Usp. World of Computer Science. Detroit: Gale Group, cop. 2002. str. 14

Potrebno je napomenuti kako su odabrane klasifikacije navedene kako bi se ilustrirala obuhvatnost pojma algoritma, te kako uz navedene postoji još pregršt različitih podijela. Rekurzivni algoritmi, kojima se ovaj rad bavi, nalaze se u klasifikaciji algoritama prema implementaciji međutim, prije bavljenja samim rekurzivnim algoritmima potrebno je pojasniti pojam rekurzije.

# 4. REKURZIJA

Ponekad je problem prevelik i čini se nerješiv, međutim, ako se taj problem može razdvojiti na manje potprobleme i njihovim rješavanjem dolaziti bliže, te u konačnici riješiti glavni problem radi se o rekurziji. Na rekurziju se može gledati kao na metodu rješavanja problema ili kao na programsku tehniku. Oba aspekta u svojoj se suštini odnose na isto – rješavanje većeg problema njegovim pojednostavljivanjem. Uz to, rekurzija se podudara i s osnovnim al Khowarizmijevim principima na kojima su nastali algoritmi. Rekurzija je, dakle, „algoritam koji poziva sam sebe, pri čemu do rješenja problema dolazimo rješavanjem potproblema.“[19]. Uz to, svaka funkcija koja unutar svoje definicije poziva samu sebe je rekurzivna.[20]

Postoji više tipova rekurzije, a to su: *jednostruka rekurzija* i *višestruka rekurzija, indirektna rekurzija* te *anonimna rekurzija*. *Jednostruka rekurzija* sadrži samo jedan poziv same sebe dok se *višestruka rekurzija* poziva više puta. Jednostruka rekurzija je često efikasnija od višestruke i može se zamijeniti iterativnom metodom kojoj vrijeme izvršavanja i količina potrebne memorije rastu linearno. S druge strane, višestruke rekurzije zahtijevaju memoriju i vrijeme izvršavanja koji se povećavaju eksponencijalno. I jednostruka i višestruka rekurzija bili bi primjeri *direktne* rekurzije, odnosno slučaja kad funkcija poziva samu sebe. Nasuprot njima nalazi se *indirektna rekurzija* u kojoj funkcija poziva drugu funkciju koja poziva početnu funkciju. Dakle, funkcija A poziva funkciju B koja poziva funkciju A. Mogući su i slučajevi u kojima su povezane više od dvije funkcije, na primjer – funkcija A poziva funkciju B, funkcija B poziva funkciju C, a funkcija C poziva funkciju A. Naposljetku, *anonimna rekurzija* predstavlja slučaj u kojem se funkcije ne pozivaju eksplicitno već implicitno, ovisno o kontekstu.[21] Razlike među navedenim tipovima

[19] Hoško, Tomislav. Nav. Dj. str. 9
[20] Usp. Baumgartner, Alfonzo; Poljak, Stjepan. Sortiranje podataka. // Osječki matematički list. - 5 (2005), 1. str. 22
[21] Recursion (computer science), 2014. URL: http://en.wikipedia.org/wiki/Recursion_%28computer_science%29 (2014-08-29)

rekurzije su očite, međutim i dalje se radi o istom principu – funkciji koja poziva samu sebe ili nizu povezanih funkcija koje se međusobno pozivaju.

Osnovna je ideja rekurzije riješiti algoritam funkcijom koja poziva samu sebe, pri tome, može se raditi o samo jednoj funkciji ili više njih koje se višestruko međusobno pozivaju. Zatim, rekurzivni algoritam mora imati završetak, odnosno osnovni problem čijim rješavanjem staje i izvođenje algoritma. Algoritam bez završetka je nerješiv jer bi se izvodio beskonačno. I konačno, može se koristiti struktura podataka stog za pohranu rezultata i povratak iz rekurzije. Uz to, navedene su i glavne karakteristike rekurzivnih algoritama. Prije svega mora postojati *osnovni slučaj* ili *rješenje* do čijeg se rezultata dolazi direktno bez rekurzije. Zatim, svaki sljedeći poziv mora doći bliže osnovnom slučaju, odnosno, mora doći do *napretka*. U suprotnom bi rekurzija bila beskorisna. Slijede *pravilo projektiranja* koje navodi kako svaki rekurzivni poziv mora biti funkcionalan, te *pravilo neponavljanja* koje tvrdi kako se ne treba dopustiti da se isti problem rješava odvojenim rekurzivnim pozivima. I u konačnici, rekurzivni programi su kraći, to jest elegantniji, ali njihovo izvođenje traje duže.[22]

Rekurzivni algoritam funkcionira na sljedeći način: ulaskom u algoritam najprije se postavlja pitanje je li dosegnut osnovni slučaj. Ako je odgovor potvrdan dolazi do povratka iz funkcije. Ukoliko osnovni slučaj nije dosegnut, smanjuje se osnovni izlaz i ponovno se poziva algoritam. Taj proces se ponavlja dokle god se ne počne vraćati s rezultatima jednostavnijih slučaja koji će, u konačnici, pomoći u rješavanju osnovnog slučaja.[23] Vrlo jednostavan primjer rekurzivnog algoritma moguće je ilustrirati kutijom s tri igračke koju je potrebno isprazniti. Uzme li se u obzir kako se radnje obavljaju serijski, iz kutije je moguće uzimati jednu po jednu igračku. Dakle, dokle god je broj igračaka u kutiji veći od nule potrebno je izvaditi jednu igračku, a vađenjem jedne igračke dolazi se bliže rješavanju glavnog problema, odnosno pražnjenja kutije. Počinje se osnovnim slučajem, odnosno pitanjem: „Kako isprazniti kutiju u kojoj su tri igračke?“. Odgovor na to pitanje bio bi: „Ako kutija nije prazna, uzimam jednu igračku i moram isprazniti kutiju u kojoj su dvije igračke.“. Zatim se ponovno postavlja pitanje koje se u ovom slučaju odnosi na potproblem: “Kako isprazniti kutiju u kojoj su dvije igračke?“. Odgovor je: „Ako kutija nije prazna, uzimam jednu igračku i moram isprazniti kutiju u kojoj je jedna igračka.“. Nadalje, postavlja se pitanje manjeg potproblema: „Kako isprazniti kutiju u kojoj je jedna igračka?“. Odgovor slijedi isti princip kao i posljednja dva: „Ako kutija nije prazna, uzimam jednu igračku i moram isprazniti kutiju u kojoj nema igračaka.“. Konačno, na pitanje „Kako isprazniti kutiju u

[22] Usp. Hoško, Tomislav. Nav. Dj. str. 23
[23] Usp. Isto. str. 24

kojoj nema igračaka?“, odgovor je „Kutija je prazna, problem je riješen.“. Navedeni problem, iako banalan, predstavlja jednostavan rekurzivni algoritam koji je, kao i u definiciji, pozivao samog sebe i riješio glavni problem rješavanjem potproblema. Isti algoritam moguće je generalizirati preoblikovanjem pitanja i odgovora. Pitanje bi bilo: „Kako isprazniti kutiju u kojoj je N igračaka?“, a odgovor: „Ako kutija nije prazna, uzimam jednu igračku i moram isprazniti kutiju s N-1 igračaka.“.

Navedeni algoritam također ima tri osnovna dijela rekurzivnog algoritma. Najprije je naveden osnovni slučaj čijim ispunjavanjem algoritam staje. U ovom slučaju radilo se o kutiji u kojoj je broj igračaka jednak nuli, odnosno u potpunosti je ispražnjena. Svaki korak predstavljao je napredak prema rješenju osnovnog slučaja, dakle, sa svakom izvađenom igračkom kutija je bila bliže pražnjenju. I konačno, koristio se rekurzivni poziv – isti algoritam se primjenjivao na jednostavnijim problemima.[24] Prikazani algoritam bio je ipak tu samo kako bi se na jednostavan način ilustriralo funkcioniranje rekurzije dok su bolji i poznatiji primjeri rekurzivnih algoritama znatno kompleksniji i zahtijevaju više promišljanja i objašnjavanja. Stoga je primjerima rekurzivnih algoritama posvećeno posebno poglavlje u kojem će svaki od njih biti predstavljen i pojašnjen u zasebnom potpoglavlju.

# 5. PRIMJERI REKURZIVNIH ALGORITAMA

Kako je već navedeno, u ovom poglavlju bit će prikazano i pojašnjeno nekoliko rekurzivnih algoritama. Potrebno je napomenuti kako su neki od njih napisani u pseudokodu, a neki u određenim programskim jezicima, što će biti naznačeno u tekstu o svakom od algoritama. Kodovi koji se nalaze kao prilozi uz svaki algoritam odabrani su zbog svoje jednostavnosti i mogućnosti shvaćanja rekurzije kroz njihovo objašnjavanje, te kako bi se mogla vidjeti primjena rekurzije na različite tipove problema. Kod svakog algoritma pojašnjen je najprije problem koji rješava, zatim je prikazan primjer koda koji je pojašnjen i u konačnici je ponuđeno jedno ili više rješenja navedenog problema.

[24] Usp. Recursion. URL: http://www.cs.utah.edu/~germain/PPS/Topics/recursion.html (2014-08-30)

## 5.1 Faktorijeli

Faktorijeli predstavljaju brojeve koji se dobiju kao umnožak svih prirodnih brojeva manjih od zadanog broja. Oznaka za faktorijel je „!“. Potrebno je napomenuti kako se, po dogovoru, uzima da je 0!=1.[25] Dakle rješenje, odnosno duži prikaz faktorijela broja 7 bio bi 7!=7*6*5*4*3*2*1, što bi dalo rješenje jednako 5040. Upravo faktorijeli predstavljaju jedan od najosnovnijih primjera rekurzivnih algoritama u većini slučajeva jer omogućuju izradu izrazito jednostavnog algoritma na kojem se bez većih poteškoća mogu primijetiti osnovni principi i rad rekurzije.

Primjer koda ovog algoritma pisan je u C programskom jeziku **(Vidi Prilog 1 Slika 1.)**. Radi se, dakle, o funkciji praktičnog naziva *factorial* koja ima jedan osnovni uvjet prilikom unosa broja N. Ako je uneseni broj jednak nuli vraća se rezultat 1, što je u skladu s navedenim dogovorom gdje je 0!=1. Za sve slučajeve gdje je broj veći od nule, manji ne može biti jer se radi o prirodnim brojevima, odnosno, pozitivnim, cijelim brojevima, provodi se naredba *N * factorial(N-1)*. Funkcija se tada izvodi, odnosno ponovno poziva, dok se smanjivanjem ne dođe do nule kada se kao rezultat dobiva broj 1. Dakle, za faktorijel broja 5 algoritam bi učinio sljedeće: 5 nije jednako nuli, provodi se iduća naredba gdje je rezultat 5 * faktorijel(5-1), odnosno, 5 * 4!. Zatim kroz isti proces prolazi broj 4. 4 nije jednak nuli, dobiva se rezultat 4 * 3!, odnosno, u globalu to izgleda kao 5 * 4 * 3!. Isto se događa i s brojem 3, veći je od nule, 3 * 2! ili 5 * 4 * 3 * 2!. Nadalje, s brojem 2 dobivamo 2 * 1!, to jest, 5 * 4 * 3 * 2 * 1!. Broj 1 i dalje nije jednak nuli, međutim, ponovnim obavljanjem naredbe dobiva se 1 * 0! te će se sljedećim pozivom kao rezultat vratiti 1 jer je postignut uvjet gdje je N jednak nuli. Dakle, konačno rješenje faktorijela broja 5 bilo bi 5 * 4 * 3 * 2 * 1 * 1 čijim se množenjem dobiva 120.

Navedeni algoritam je pomoću rekurzivnih poziva dobio rješenje faktorijela broja 5 računajući faktorijele za svaki broj zasebno i kombinirajući ih naposljetku u konačno rješenje. Isto se može učiniti i s bilo kojim drugim prirodnim brojem, proces će biti isti – tražiti faktorijele za svaki broj, odnosno rješavati potprobleme, te ih u konačnici iskoristiti za rješenje glavnog slučaja. I dok u slučaju broja 5 algoritam ne predstavlja nikakvu veliku prednost u odnosu na računanje „na pamet“, lako je zaključiti kako bi bio od velike pomoći kada bi se računali faktorijeli višeznamenkastih brojeva.

[25] Usp. Faktorijel.// Proleksis enciklopedija online. Leksikografski zavod Miroslav Krleža, lipanj 2012. URL: http://proleksis.lzmk.hr/4865/ (2014-08-31)

## 5.2 Fibonaccijevi brojevi

Poznatiji pod nadimkom Fibonacci, Leonardo Pisano ili Leonardo iz Pise, rođen je u Italiji, a školovao se u Sjevernoj Africi gdje je njegov otac imao diplomatski položaj. Kroz putovanja s ocem Fibonacci je učio matematiku vidjevši prednost sustava koji se koristio u zemljama koje su posjetili. Po svom povratku u Italiju, izdao je mnoga djela, od kojih dobar dio nije sačuvan budući da se radilo o dobu prije tiskarskog stroja kada je svaka kopija morala biti prepisana. Međutim, očuvano je jedno od njegovih najpoznatijih djela, Liber abaci, koje je zasnovano na znanju aritmetike i algebre koje je Fibonacci akumulirao na svojim putovanjima. Radilo se prije svega o knjizi koja je dovela arapske brojeve u Europu u kojoj se, do tad, i dalje koristio rimski brojevni sustav. Nadalje, u trećem dijelu knjige iznesen je problem koji je predstavio Fibonaccijeve brojeve i niz po kojima je i danas poznat. Radilo se o sljedećem problemu: par zečeva okružen je zidinama. Koliko parova zečeva će dobiti prvi par u godinu dana, ako svaki mjesec svaki par dobije novi par koji će nakon dva mjeseca biti spreman za reprodukciju? Rješavanjem ovog problema dobije se niz 1, 1, 2, 3, 5, 8, 13, 21, 34... gdje je svaki broj jednak zbroju prethodna dva.[26] Elementi tog niza nazivaju se Fibonaccijevi brojevi, a niz se može definirati rekurzivnom relacijom $\mathbf{F_n=F_{n-1} + F_{n-2}}$.[27] Uključi li se ta relacija u algoritam, moguće je tražiti vrijednost za bilo koji broj.

Ovaj algoritam također je zapisan u C programskom jeziku **(Vidi Prilog 1 Slika 2.)**. Nešto je kompleksniji od prijašnjeg primjera te ima četiri različita uvjeta. Prvi uvjet je da se za svaki broj jednak nuli izbaci vrijednost 0, zatim ako je broj jednak broju 1, izbacuje se rezultat 1. U slučaju da je broj pozitivan i veći od 1 primjenjuje se navedena relacija, a ako je broj negativan, odnosno, manji od nule dobije se vrijednost -1 i program javlja grešku. Potrebno je, naravno, usmjeriti pozornost na pozitivne brojeve veće od 1. Uzme li se da je vrijednost n jednaka broju 5, dakle veća od 0 i 1, na broj se primjenjuje formula čime se najprije izračunava vrijednost za fib(5) koja je jednaka fib(5 – 1) + fib(5 – 2), to jest fib(4) + fib(3). Zatim dolazi do rekurzivnog poziva gdje se izračunavaju vrijednosti za fib(4) i fib(3) te algoritam počinje eksponencijalno rasti, odnosno granati se, što je upravo jedan od nedostataka primjene rekurzije na ovaj problem. Vrijednost za fib(4) iznosi fib(3) + fib(2), a vrijednost za fib(3) iznosi fib(2) + fib(1). U ovom djelu algoritma vidljive su neke neadekvatnosti rekurzije, jer će se prilikom novog poziva za računanje nove četiri vrijednosti ponovno računati vrijednost za fib(3) koja je u prijašnjem koraku već izračunata.

[26] Usp. O'Connor, J. J.; Robertson, E. F. Leonardo Pisano Fibonacci, 1998. URL: http://www-history.mcs.st-and.ac.uk/Biographies/Fibonacci.html (2014-08-31)
[27] Usp. Jurić, Ljerka; Velić, Helena. Fibonaccijev brojevni sustav, 2009. URL: http://e.math.hr/math_e_article/br16/jukic_velic (2014-09-01)

Zanemare li se ipak nepotrebni koraci, algoritam nastavlja dalje s ponovnim pozivanjem dok ne dođe do fib(1) i fib(0) čije vrijednosti iznose 1 i 0. Posljednji korak u izračunavanju vrijednosti broja 5 izgleda ovako: (((fib(1) + fib(0)) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1)). Zamijene li se fib(1) i fib(0) svojim vrijednostima, dobiva se rješenje fib(5)= 1 + 0 + 1 + 1 + 0 + 1 + 0 + 1 = 5.[28] Eksponencijalni rast algoritma, višestruko računanje istih vrijednosti te preskočene korake moguće je vidjeti na priloženoj slici stabla **(Vidi Prilog 1 Slika 3.)** koje je izrađeno upravo za potrebe ilustracije problema rekurzije.

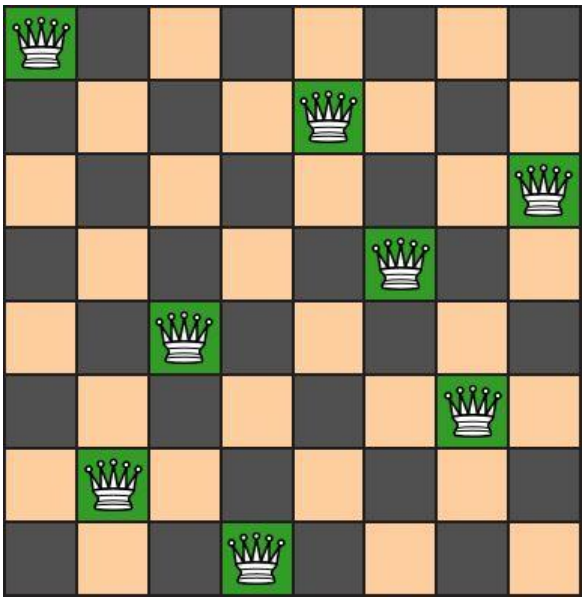
## 5.3 Problem N kraljica

Jedan od poznatijih rekurzivnih algoritama jest problem 8 kraljica. Predstavljen je 1848. godine od strane šahista Maxa Bezzela, te su se kroz godine mnogi matematičari, uključujući i Gaussa, bavili ovim problemom pokušavajući pronaći rješenje. Prva rješenja predstavio je Franz Nauck 1850. Godine, a uz to je i generalizirao, ali i proširio problem na „problem N kraljica“ omogućujući tako, smanjenje, odnosno povećanje ploče i broja kraljica te otežavanje, odnosno olakšavanje zadatka.[29] Za potrebe objašnjavanja ovog problema i algoritma koristit će se originalni zadatak s 8 kraljica. Cilj je, dakle, na šahovskoj ploči veličine 8*8 polja pronaći položaj 8 kraljica tako da se one međusobno ne napadaju. Ono što problem čini znatno težim je način, odnosno mogućnosti kretanja kraljice kao figure u šahu. Za razliku od ostalih figura kraljica se može kretati u 8 različitih smjerova – horizontalno, vertikalno te dijagonalno. Uz to, kraljičin doseg ograničen je samo rubovima ploče. Slažući kraljice jednu po jednu na ploču potrebno je uvijek u obzir uzimati smjerove napada prethodnih kraljica, iz tog razloga koristi se metoda „*backtracking*“ (metoda unatražnog pretraživanja). Unatražno pretraživanje omogućuje isprobavanje različitih kombinacija postavljanja svake kraljice, korak po korak, i postepeno približavanje rješenju. Dogodi li se u jednom od koraka greška, algoritam se vraća jedan ili više koraka unatrag i isprobava druge kombinacije.[30]

Algoritam koji se analizira zapisan je u pseudokodu **(Vidi Prilog 1 Slika 4.)**. Najprije se deklarira osnovni slučaj i njegovo rješenje – ako je kraljica uspješno postavljena u posljednji, osmi

[28] Usp. Fibonacci – Iterative and Recursive algorithms, 2012. URL: http://geekyfry.com/interview/tech-it-interview/algo-ds/fibonacci-iterative-and-recursive-algorithms/ (2014-09-01)
[29] Usp. Eight Queens Puzzle, 2014. URL: http://en.wikipedia.org/wiki/Eight_queens_puzzle (2014-09-02)
[30] Usp. Hoško, Tomislav. Nav. Dj. str. 26

red, algoritam staje. U protivnom postavlja prvu kraljicu u prvo polje prvog reda i pamti koje kolone i dijagonale može napasti. Svaku novu kraljicu postavlja u red niže i provjerava je li ta kraljica legalna u odnosu na ostale, to jest može li ostati na tom polju. Zatim pamti kolone i dijagonale koje ta kraljica napada čime je završen proces za još jednu kraljicu te dolazi do novog rekurzivnog poziva i algoritam počinje od vrha. Principi rekurzije jasno su vidljivi, osnovni problem je postavljanje svih osam kraljica dok su potproblemi podijeljeni u postavljanje svake kraljice pojedinačno. Isti proces prolaze sve kraljice. Dogodi li se u nekom trenutku da je nemoguće postaviti jednu od kraljica jer se uvijek nalazi u liniji napada neke od prethodno postavljenih, algoritam briše vrijednosti napada, te se metodom unatražnog pretraživanja vraća na prethodnu kraljicu i pokušava ju postaviti na neko drugo polje. Također se može dogoditi da ni prethodna kraljica ne može biti postavljena nigdje drugdje osim na polje koje očito proizvodi probleme za iduću kraljicu. U tom slučaju se algoritam vraća još više unatrag. Nije nimalo začuđujuće ako se algoritam, posebno u završnim koracima, vrati po nekoliko kraljica unatrag kako bi isprobao druge mogućnosti. Jednom kada je svih osam kraljica uspješno postavljeno, algoritam obavlja zadnju rekurziju te staje kod prvog uvjeta. Moguća su 92 rješenja navedenog problema, na slici je prikazan jedan od slučaja kada je prva kraljica postavljena u prvo polje prvog reda **(Slika 5.)**[31].



Slika 5. Jedno od mogućih rješenja problema 8 kraljica

[31]Eight Queens, 2013. URL: http://www.datagenetics.com/blog/august42012/ (2014-09-05)

## 5.4 Sudoku

Poznat još i kao Rubikova kocka 21. stoljeća, sudoku je najprije postao popularan u Japanu 1986. godine, a dvadesetak godina kasnije, 2005. godine, postiže međunarodnu popularnost.[32] Jedan sudoku problem sastoji se od mreže s 9*9 kućica koja je sačinjena od 9 manjih podmreža od kojih je svaka velika 3*3 kućice. Može se dogoditi da su neke kućice unaprijed ispunjene brojevima od 1 do 9 dok su druge prazne. Cilj je popuniti sva prazna polja brojevima od 1 do 9 tako da se u svakom redu, koloni i mreži od 3*3 kućice nalazi svih 9 mogućih brojeva, bez ponavljanja.[33] Koristeći rekurzivni algoritam moguće je rješavati sudoku probleme. Navedeni algoritam koristi „*brute force"* pristup te isprobava svaku moguću kombinaciju brojeva za svaku kućicu.[34]

Ovaj je algoritam također zapisan u pseudokodu **(Vidi Prilog 1 Slika 6.)**. Počinje iz prve kućice u gornjem lijevom kutu te se pomiče kolonu po kolonu nakon čega se spušta u idući red i obavlja istu funkciju. Brute force pristupom isprobava svih devet brojeva za kućicu jedan po jedan te isprobava može li se takvom kombinacijom sudoku ispuniti do kraja i riješiti. Rekurzija je ovdje jasno vidljiva, osnovni problem upisati je ispravne brojeve u sve kućice, odnosno, njih 81, a potproblemi su unošenje ispravnih brojeva u svaku kućicu, odnosno, red, kolonu i kvadrat. Upisani broj testira se za svaki red, kolonu i kvadrat, te, ukoliko je sve u redu, algoritam nastavlja s provjeravanjem mogućnosti rješavanja sudokua danom kombinacijom. Ne bude li algoritam mogao nastaviti dalje zbog greške u unosu, metodom unatražnog pretraživanja bilo bi moguće vratiti se jednu ili više kućica unatrag te isprobavati druge kombinacije. Jednom kada su sve kućice ispravno popunjene, algoritam posljednjim pozivom dolazi do prvog uvjeta koji je ispunjen te javlja uspjeh.[35]

[32]Usp. Lynce, Ines; Ouaknine, Joël. Sudoku as a SAT Problem, 2006. Str. 1. URL: http://ldc.usb.ve/~meza/ci-5651/e-m2008/articulosYsoftware/SudokuAsSAT.pdf (2014-09-02)
[33] Usp. Isto. str. 2
[34] Usp. Recursion. URL: http://www.cs.utah.edu/~germain/PPS/Topics/recursion.html (2014-09-02)
[35] Usp. Isto

## 5.5 Hanojski tornjevi

Posljednji algoritam u ovom radu je jedan od poznatijih primjera rekurzije i algoritam na čijoj se obradi provelo najviše vremena. Radi se o problemu Hanojskih, odnosno Brahminih tornjeva. Uz ovaj problem vezana je i sljedeća legenda: u Varanasijevom hramu, ispod kupole koja označava centar svijeta, stoji mjedena ploča u koju su ugrađene tri dijamantne igle, svaka visine jednog lakta i debljine tijela pčele. Na jednoj od igala, za vrijeme stvaranja svijeta, Bog je postavio 64 diska od čistog zlata, najveći na mjedenu ploču, a na njega diskove koji se sve više smanjuju prema vrhu. To je Brahmin toranj. Danonoćno, bez stajanja, svećenici prenose diskove s jedne dijamantne igle na drugu prema stalnim i neizmjenjivim Brahminim zakonima koji nalažu da svećenik ne smije micati više od jednog diska i da disk mora postaviti na iglu tako da ispod njega nema manjih diskova. Kada 64 diska budu prenijeta s igle na koju ih je postavio Bog, na jednu od drugih igala, toranj, hram i brahmani raspast će se u prašinu, a svijet će nestati uz prasak groma.[36] Ovu je legendu Edouard Lucas 1883. godine popularizirao i pretvorio u zagonetku.[37] Problem Hanojskih ili Brahminih tornjeva je sljedeći: postoje tri štapa, A ili izvor, B kao pomoćni štap i C kao odredište. Na štapu A nalazi se n diskova različite veličine koji od dna prema vrhu idu veći prema manjim. Cilj je u što manje poteza, koristeći pomoćni štap, sa štapa izvora prenijeti diskove na štap odredište, jedan po jedan, bez da se veći disk ikad nađe iznad manjeg.[38] Ovaj problem je također rješiv rekurzivnim algoritmom, no ,usporedbe radi, problemu se najprije pristupilo praktično, a tek onda pokretanjem algoritma na računalu. U oba slučaja broj diskova na štapu A bio je 4.

Prilikom praktičnog rješavanja problema, na komad papira nacrtan je početni položaj štapova i diskova na štapu A. Svaki novi korak bio je nacrtan zasebno. U prvih nekoliko pokušaja potraga za rješenjem bila je neuspješna. Međutim, vježbom i s povećanom pažnjom, diskovi su uspješno prebačeni sa štapa A na štap C držeći se pritom svih propisanih pravila. Broj potrebnih poteza iznosio je 15 ne računajući početnu poziciju diskova. Radi provjere točnosti rješenja na računalu je pokrenut rekurzivni algoritam kako bi se provjerilo je li 4 diska moguće premjestiti na štap C u manjem broju poteza.

[36] Usp. Odifreddi, Piergiorgio; Cooper, S. Barry. Recursive Functions.// Stanford Encyclopedia of Philosophy. 2012. URL: http://plato.stanford.edu/entries/recursive-functions/ (2014-09-02)
[37] Usp. Palmer, Daniel W. EXPLORING RECURSION WITH VARIATIONS ON THE TOWERS OF HANOI. Str. 1. URL: http://www.ccscjournal.willmitchell.info/Vol12-96/No2a/Daniel%20W%20Palmer.pdf (2014-09-02)
[38] Usp. Hoško, Tomislav. Nav. Dj. str. 27

Kod je zapisan u C programskom jeziku **(Vidi Prilog 1 Slika 7.)**. Radi se o u potpunosti funkcionalnom kodu koji, između ostalog otvara poseban prozor za unos broja diskova i računanje rješenja. Broj diskova je u algoritmu deklariran kao integer što bi značilo da pripada skupu pozitivnih, cijelih brojeva. Naredba *prntf* na ekranu ispisuje „Enter the number of discs:“ odnosno, traži unošenje broja diskova. Unošenjem broja i pritiskom tipke Enter *&number* dodjeljuje vrijednost upisanog broja varijabli *number_of_ discs*, to jest, broju diskova. Zatim se deklarira funkcija *Towers* čiji su ulazni parametri broj diskova te konstante A, B i C. Potrebno je napomenuti kako su u navedenom algoritmu štapovi B i C zamijenili uloge te je štap B odredište a štap C pomoćni štap, međutim, to ne pravi bitnu razliku niti će imati utjecaja na rješenje. Time se dolazi do prvog uvjeta – ako je broj diskova jednak 1 ispisuje se Moving A to B, odnosno, taj se jedan disk prebacuje s izvora na odredište i problem je riješen. U slučaju da je broj diskova veći od 1 dolazi se do dijela gdje se problem dijeli na N potproblema ovisno o broju diskova. Problem se tada rješava rekurzivno – pomiču se svi diskovi osim najvećeg, dakle od 1 do N-1, zatim se pomiče najveći disk N, te se na kraju diskovi od 1 do N-1 postavljaju na najveći disk. Algoritam je pokrenut u aplikaciji Dev-C++, te je pod broj diskova unesen broj 4. Pokretanjem algoritma izlistan je popis od 15 poteza što je bilo jednako broju poteza prilikom praktičnog rješavanja problema. Prateći taj popis poteza moguće je reproducirati rješenje **(Vidi Prilog 1 Slika 8.)**.

Isti algoritam moguće je primjenjivati za bilo koji broj diskova, tako je za 5 diskova potrebno minimalno 31 potez, za 6 diskova to bi bilo 63 poteza, 7 diskova zahtijevalo bi 127 itd. No ostaje pitanje koliko je poteza potrebno da se pomakne 64 diska? Odgovor je zapanjujućih $2^{64}$ – 1 poteza. Koliko god nerealno zvučalo, to znači da bi, ukoliko bi se svake sekunde, svakog dana, svake godine, učinio jedan pravi potez, za rješenje problema za 64 diska, bilo potrebno otprilike 58 milijardi stoljeća.[39] Može se, dakle, zaključiti kako će svijet biti siguran jedno izvjesno vrijeme, barem kad je legenda u pitanju.

[39] Usp. Odifreddi, Piergiorgio; Cooper, S. Barry. Nav. Dj.

# 6. ZAKLJUČAK

Algoritmi u svojoj osnovi predstavljaju korake do rješenja nekog problema, stoga i ne čudi njihova velika primjenjivost koja seže dalje od samo matematike i računarstva. Držeći se osnovnih principa na kojima su nastali algoritmi, rekurzija se pokazala kao moćan alat u rješavanju raznih zadataka, kako matematičkih, tako i logičkih. S obzirom da se obrađuje vrlo rano na kolegijima vezanim uz programiranje, razumjeti način na koji funkcionira, te algoritme koji se njome koriste kao metodom, nije posebno teško. Algoritmi navedeni u radu predstavljaju samo dio problema na koje je moguće primijeniti rekurziju. No, njihovim svladavanjem produbljuje se ne samo razumijevanje rekurzije kao metode, već i razumijevanje njezine primjene čime se olakšava i bilo koja buduća primjena na nove i kompleksnije probleme. Zanimljiva je i mogućnost primjene svih navedenih koncepata na rješavanje zadataka u stvarnom okruženju. Dubljim shvaćanjem značenja razbijanja problema na manje i jednostavnije zasigurno će se ubrzati i olakšati rješavanje istih. Kao jedna od osnovnijih metoda, rekurzija može služiti i kao dobar uvod u programiranje te dati određenu osnovu za razumijevanje anatomije algoritma. Uz sve mane, njezina primjenjivost i dalje je velika, te će zasigurno još generacije programera uvoditi u svijet algoritama.

# PRILOG 1

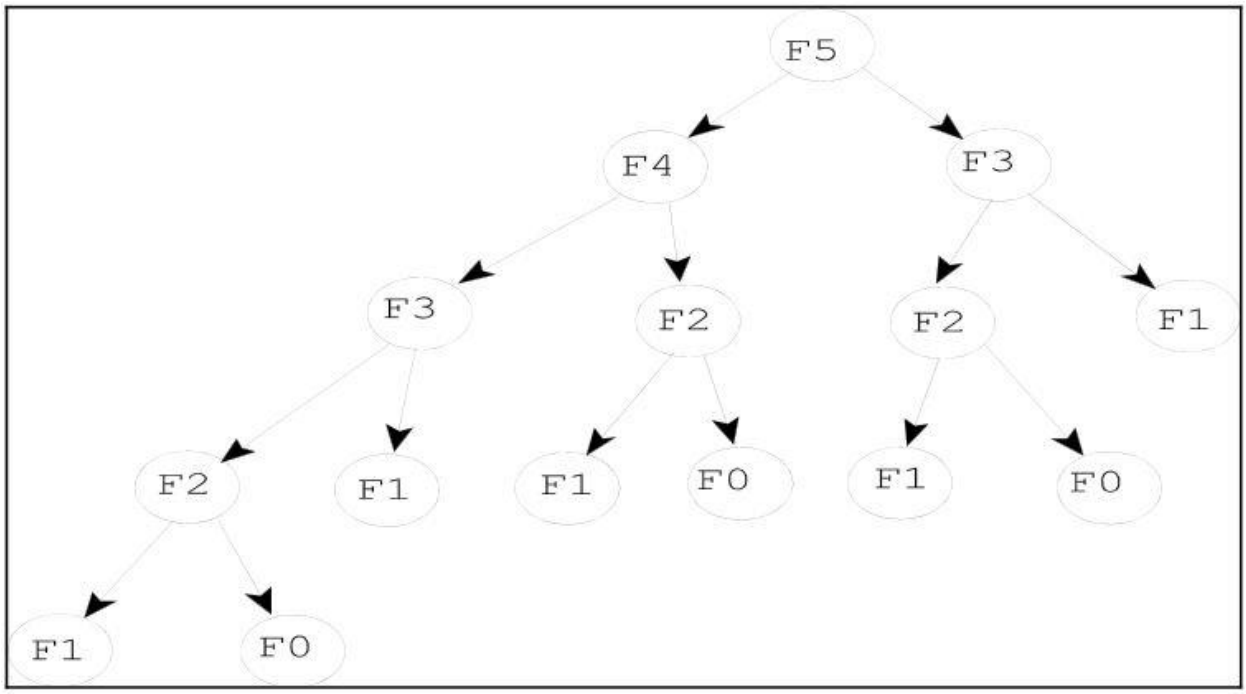
```
1   int
2       factorial ( int N )
3       {
4          if ( N == 0 )
5          {
6            return 1;
7          }
8          else
9          {
10           return N * factorial(N-1);
11         }
12      }
```

Slika 1. Rekurzivni algoritam za izračunavanje faktorijela[40]

```
1   int fib(int n) {
2     if (n == 0)
3     {
4       return 0;
5     }
6     else if (n == 1)
7     {
8       return 1;
9     }
10    else if (n > 1)
11    {
12      return fibo(n-1) + fibo(n-2); /* this line is dangerous,
13                                       makes the algo exponential */
14    }
15    else
16    {
17      return -1; /* Error in input */
18    }
19  }
```

Slika 2. Rekurzivni algoritam za Fibonaccijeve brojeve[41]

[40]Recursion. URL: http://www.cs.utah.edu/~germain/PPS/Topics/recursion.html (2014-08-30)
[41]Fibonacci – Iterative and Recursive algorithms, 2012. URL: http://geekyfry.com/interview/tech-it-interview/algo-ds/fibonacci-iterative-and-recursive-algorithms/ (2014-09-01)




Slika 3. Eksponencijalni rast algoritma prikazan stablom[42]

```
1   void place(int row) {
2       if(row==8)
3           celebrateAndStop();
4       else {
5           for( queen[row] = all possible vals ) {
6               check if new queen legal;
7               record columns and diagonals it attacks;
8               // recurse
9               place(row+1);
10              // if reach here, have failed and need to backtrack
11              erase columns and diagonals it attacks;
12      }
13  }
```

Slika 4. Rekurzivni algoritam za problem 8 kraljica[43]

[42]Cargal, J.M.. Recursive algorithms // Discrete Mathematics for Neophytes: Number Theory, Probability, Algorithms, and Other Stuff / J.M. Cargal. str. 1-10.
URL:http://www.cargalmathbooks.com/5%20Recursive%20Algorithms.pdf (2014-09-05)
[43]Recursion. URL: http://people.cs.clemson.edu/~goddard/texts/cpsc212/9.pdf (2014-09-05)

```
function solve_sudoku ( current_box_id )

    if all the boxes are filled
        return true; // solved!
    end

    // ignore already "filled" squares (assume they are correct!)
    if the current box is filled
        return the result of: solve_sudoku( next box );
    end

    // hypothesize all 9 possible numbers
    for each possible number from 1 to 9
        if that number is 'valid' (okay to put in box)  // test row,col,square
            try that number in the box
            if you can "solve_sudoku" for the rest of the puzzle
                return true; // success!
            end
        end
    end

    return false; // failure

end
```

Slika 6. Rekurzivni algoritam za ispunjavanje sudokua[44]

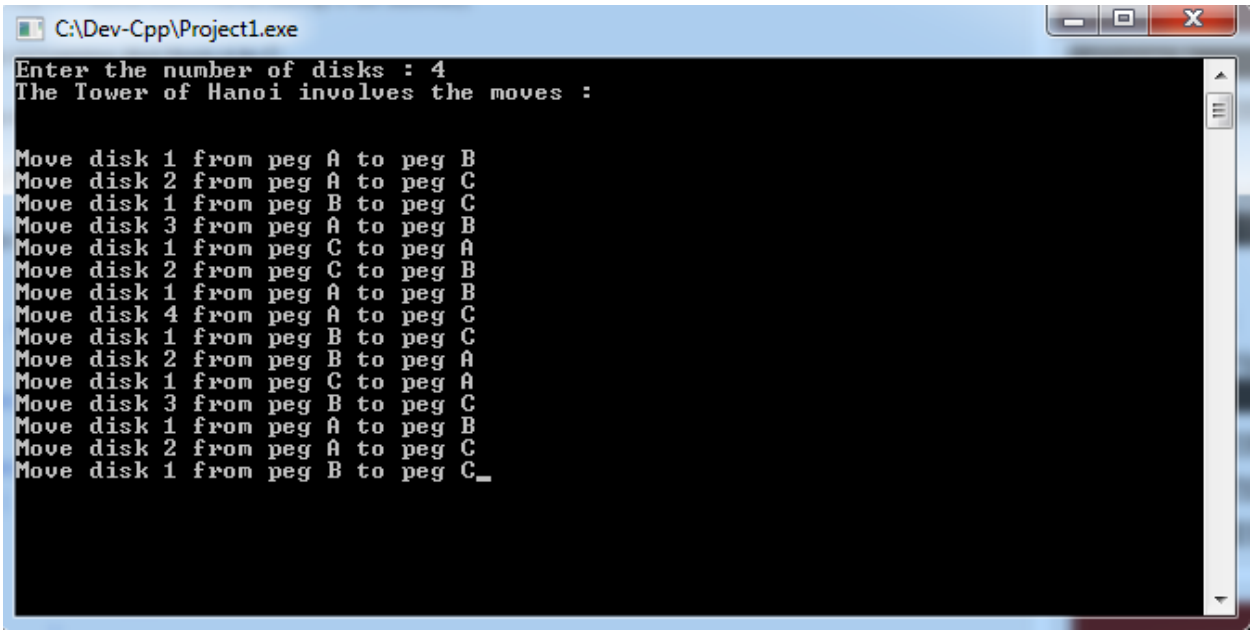
[44]Recursion. URL: http://www.cs.utah.edu/~germain/PPS/Topics/recursion.html (2014-08-30)

```c
#include "stdio.h"

void towers(int,char,char,char);

void towers(int n,char frompeg,char topeg,char auxpeg)
    { /* If only 1 disk, make the move and return */
      if(n==1)
        { printf("\nMove disk 1 from peg %c to peg %c",frompeg,topeg);
          return;
        }
      /* Move top n-1 disks from A to B, using C as auxiliary */
      towers(n-1,frompeg,auxpeg,topeg);
      /* Move remaining disks from A to C */
      printf("\nMove disk %d from peg %c to peg %c",n,frompeg,topeg);
      /* Move n-1 disks from B to C using A as auxiliary */
      towers(n-1,auxpeg,topeg,frompeg);
    }
main()
    { int n;
      printf("Enter the number of disks : ");
      scanf("%d",&n);
      printf("The Tower of Hanoi involves the moves :\n\n");
      towers(n,'A','C','B');
      return 0;
    }
```

Slika 7. Rekurzivni algoritam za rješavanje problema Hanojskih tornjeva[45]




Slika 8. Rješenje problema Hanojskih tornjeva za 4 diska

[45]Saurabh. C code for towers of Hanoi, 2013. URL: http://programminggeeks.com/c-code-for-towers-of-hanoi/ (2014-09-05)

# LITERATURA


1. Baumgartner, Alfonzo; Poljak, Stjepan. Sortiranje podataka. // Osječki matematički list. - 5 (2005), 1. URL: http://hrcak.srce.hr/index.php?show=clanak&id_clanak_jezik=6637 (2014-08-29)
2. Cargal, J.M.. Recursive algorithms // Discrete Mathematics for Neophytes: Number Theory, Probability, Algorithms, and Other Stuff / J.M. Cargal. str. 1-10. URL: http://www.cargalmathbooks.com/5%20Recursive%20Algorithms.pdf (2014-09-05)
3. Classification of Algorithms. URL: http://www.scriptol.com/programming/algorithms-classification.php (2014-08-29)
4. Cormen, Thomas H. Introduction to Algorithms. Cambridge [etc.]: The MIT Press: McGraw-Hill Book Company, 2000.
5. Eight Queens Puzzle, 2014. URL: http://en.wikipedia.org/wiki/Eight_queens_puzzle (2014-09-02)
6. Faktorijel.// Proleksis enciklopedija online. Leksikografski zavod Miroslav Krleža, lipanj 2012. URL: http://proleksis.lzmk.hr/4865/ (2014-08-31)
7. Fibonacci – Iterative and Recursive algorithms, 2012. URL: http://geekyfry.com/interview/tech-it-interview/algo-ds/fibonacci-iterative-and-recursive-algorithms/ (2014-09-01)
8. Hoško, Tomislav. Strukture podataka i algoritmi: priručnik. Zagreb: Algebra, 2009.

9. Jurić, Ljerka; Velić, Helena. Fibonaccijev brojevni sustav, 2009. URL: http://e.math.hr/math_e_article/br16/jukic_velic (2014-09-01)
10. Lynce, Ines; Ouaknine, Joël. Sudoku as a SAT Problem, 2006. Str. 1. URL: http://ldc.usb.ve/~meza/ci-5651/e-m2008/articulosYsoftware/SudokuAsSAT.pdf (2014-09-02)
11. Muhammad ibn Mūsā al-Khwārizmī, 2014. URL: http://en.wikipedia.org/wiki/Mu%E1%B8%A5ammad_ibn_M%C5%ABs%C4%81_al-Khw%C4%81rizm%C4%AB (2014-08-29)
12. O'Connor, J. J.; Robertson, E. F. Leonardo Pisano Fibonacci, 1998. URL: http://www-history.mcs.st-and.ac.uk/Biographies/Fibonacci.html (2014-08-31)
13. Odifreddi, Piergiorgio; Cooper, S. Barry. Recursive Functions.// Stanford Encyclopedia of Philosophy. 2012. URL: http://plato.stanford.edu/entries/recursive-functions/ (2014-09-02)
14. Palmer, Daniel W. EXPLORING RECURSION WITH VARIATIONS ON THE TOWERS OF HANOI. Str. 1. URL: http://www.ccscjournal.willmitchell.info/Vol12-96/No2a/Daniel%20W%20Palmer.pdf (2014-09-02)
15. Prabhu, Manohar. Algorithm: Types and Classification, URL: http://gonitsora.com/algorithm-types-and-classification/ (2014-08-29)
16. Recursion. URL: http://people.cs.clemson.edu/~goddard/texts/cpsc212/9.pdf (2014-09-05)
17. Recursion. URL: http://www.cs.utah.edu/~germain/PPS/Topics/recursion.html (2014-08-30)
18. Recursion (computer science), 2014. URL: http://en.wikipedia.org/wiki/Recursion_%28computer_science%29 (2014-08-29)
19. Saurabh. C code for towers of Hanoi, 2013. URL: http://programminggeeks.com/c-code-for-towers-of-hanoi/ (2014-09-05)
20. Witt, Pharaba. Different Kinds of Algorithms. URL: http://www.ehow.com/info_8292589_different-kinds-algorithms.html (2014-08-29)
21. World of Computer Science. Detroit: Gale Group, cop. 2002.