Algoritmi u nastavi informatike u gimnazijama

Milišić, Ivan

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Ivan Milišić

ALGORITMI U NASTAVI INFORMATIKE U GIMNAZIJAMA

Diplomski rad

Voditelj rada:
doc. dr. sc. Goranka Nogo

Zagreb, 2018

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem se svojoj obitelji na podršci tijekom studiranja, majci Milki, ocu Stjepanu te sestrama Ani, Katarini i Josipi. Posebno zahvaljujem sestri Marijani na ogromnoj motivaciji za vrijeme prve dvije godine studiranja.
Hvala i prijateljima koji su me bodrili tijekom studija, Domagoju, Mateju, Toniju i Jerki, odnosno glazbenom sastavu "Treskal'ca fajf".
Veliko hvala Matei Radošević na velikoj brizi i podršci koju mi je pružila za vrijeme diplomskog studija.
Na koncu, neizmjerno hvala i doc. dr. sc. Goranki Nogo na strpljenju i pomoći pri izradi diplomskog rada.

Sadržaj

Uvod

Ukratko, u radu ćemo opisati koliko se pozornosti pridaje algoritamskom mišljenju u važećim dokumentima, prikazati koji poznati algoritmi se obrađuju u gimnazijama te kritički komentirati trenutno stanje.

Kroz prvo poglavlje ćemo proučiti samu bit algoritma. Osvrnut ćemo se na definiciju algoritma, a onda i na svojstva koja algoritam mora zadovoljavati. Objasnit ćemo koje osobine mora imati "dobar" algoritam te načine na koje se algoritam može predstaviti. Bit će prikazani i primjeri algoritama kako u stvarnom svijetu tako i u matematici i informatici.

Drugo poglavlje govorit će o tome u kojem kontekstu se sve algoritamsko mišljenje spominje u *Hrvatskom nacionalnom obrazovnom standardu (HNOS-u)*, *Nacionalnom okvirnom kurikulumu za predškolski odgoj i obrazovanje te opće obvezno i srednjoškolsko obrazovanje (NOK-u)*, a onda i u standardu računalne znanosti *(CSTA K-12 Computer Science Standard)* koji se koristi u SAD-u i mnogim drugim državama. Usporedit ćemo date dokumente s trenutnim prijedlogom novog kurikuluma u Hrvatskoj.

U trećem poglavlju ćemo imati kratak pregled algoritama koji se spominju u osnovnoj školi kako bi imali bolji uvid u predznanje učenika u samom području algoritama i algoritamskog mišljenja.

Četvrto poglavlje će biti nastavak trećeg poglavlja. Ovdje ćemo proći algoritme koji se obrađuju u srednjim školama, preciznije gimnazijama, kao što i sam naslov rada govori. Također, kao i u prošlom poglavlju navest ćemo neke loše ali i dobre strane trenutnog stanja obrade algoritama i korištenja algoritamskog mišljenja u gimnazijama kao važnog dijela odgojno-obrazovnog procesa.

Poglavlje 1

Algoritam

1.1 Pojam algoritma

Na početku se odmah pitamo, što je to algoritam? U srednjim školama ne navodimo formalnu definiciju algoritma nego algoritam objašnjavamo pomoću nekih srodnih riječi ili rečenica. Najjednostavnije rečeno, algoritam je metoda, postupak ili pravilo za rješavanje nekog konkretnog problema ili klase problema. Riječ, od svih navedenih, koja bi najbolje opisala algoritam je *postupak*. Algoritam mora imati konačno mnogo koraka pa zato nije baš najbolje koristiti riječ metoda jer u matematici metodom nazivamo postupke koji uključuju beskonačno mnogo koraka i tek na limesu daju rješenje. Svaki pojedini korak algoritma je instrukcija (naredba) koju treba napraviti (izvršiti). Obično se naredbe zadaju tako da svaka sljedeća naredba piše ispod one prethodne pa se tim redoslijedom i izvršavaju. Izvođenje tog niza naredbi treba biti objektivan proces te se kao takav treba moći i reproducirati.

Postoje mnoga objašnjenja samog pojma algoritma kako u praksi tako i u nastavi pa se često susrećemo s nekim pogrešnim opisima algoritma kao što su: *"Algoritam je skup postupaka..."*, a *"skup postupaka"* bi prije opisalo više algoritama, zatim imamo *"Algoritam je konačan slijed dobro definiranih naredbi..."* bez da točno znamo što je to dobro definirana naredba, *"Algoritam je niz preciznih uputa..."* opet bez opisa što je to precizna uputa, itd. Svakako, algoritam ne bi smjeli opisivati pomoću pojmova koji nisu potpuno jasni ili nisu prije toga definirani.

Najbolji komentar za opis pojma algoritma koji bi trebali koristiti u nastavi bi bio: "Algoritam je konačan niz koraka koje treba napraviti za rješavanje nekog problema ili postizanje nekog cilja." Tako da ne bi bilo dvojbe oko nejasnih pojmova u samom opisu.
Dakle, ne navodimo strogu definiciju algoritma nego algoritam opisujemo pomoću srodnih i preciznih pojmova koje ne treba dodatno definirati.
Slijedi primjer postupka iz svakodnevnog života koji odgovara takvom poimanju algoritma:

Primjer 1:

Upute za korištenje aparata za kavu:

- ubacite kovanice
- odaberite napitak
- odaberite razinu šećera
- pričekajte dok se napitak pripremi
- poslužite se
- uzmite ostatak novca (ukoliko ga ima).

1.2 Svojstva algoritma

Kako bi se neki postupak mogao nazvati algoritmom, postoje svojstva koje svaki algoritam mora zadovoljavati. Ta svojstva su: ulaz, izlaz, konačnost, definiranost i nedvosmislenost (određenost) te efikasnost (efektivnost).

Ulaz: Svaki algoritam mora imati nula, jedan ili više, ali konačno mnogo ulaznih parametara prije nego što počne izvršavanje algoritma. Ulazne vrijednosti definiraju zadatak problema kojeg treba riješiti. Svaka od ulaznih vrijednosti uzima se iz unaprijed zadanog skupa dozvoljenih vrijednosti. Taj skup zovemo područje definicije problema. Ukoliko imamo algoritam koji računa volumen kvadra, moramo mu zadati duljinu, širinu i visinu kvadra. Sva tri ulaza su elementi skupa pozitivnih realnih brojeva.

Izlaz: Algoritam mora davati nekakav rezultat tj. povratnu informaciju. Ukoliko algoritmom ništa ne saznajemo, algoritam nema svrhe.

Konačnost: Također, algoritmima smatramo samo konačne postupke pa postupke koji imaju beskonačno koraka ili postupke koji imaju niz konačnih koraka koji se ponavljaju beskonačno mnogo puta ne smatramo algoritmima.

Definiranost i nedvosmislenost (određenost): Izvođenje niza naredbi mora biti objektivan proces, koji se mora moći reproducirati. Mora vrijediti da za izlaz dobivamo uvijek konkretan i neslučajan rezultat, tj. da uz iste ulazne parametre moramo dobiti uvijek isti konačni rezultat. Stvar možemo poistovjetiti s eksperimentima u kemiji. Očekuje se da će se neka tvar ponašati isto u jednakim uvjetima te da će se svakim eksperimentom dobiti jednak rezultat.

Efikasnost (efektivnost): Probleme rješavamo s ciljem da dobijemo tj. izračunamo rješenje. Smatramo da algoritam mora biti izvršen u prihvatljivom vremenu te da utrošak

resursa ne bude toliko velik da se sam algoritam ne isplati koristiti za određeni tip problema. Što je to prihvatljivo vrijeme ovisi o problemu i potrebi. Nekad to može biti par sekundi, ali i par mjeseci i godina. No postoji i velika klasa problema za koje postoji efikasno rješenje samo za male zadaće. (vidi [21])

Naravno, kako u stvarnom svijetu možemo probleme riješiti na puno različitih načina tako i za algoritme vrijedi da ne postoji jedinstven način za rješavanje nekog tipa problema. To načelo zove se načelo ekvifinaliteta i govori kako za rješavanje nekog problema ne postoji jedinstveni algoritam. U današnjem vremenu s pojmom algoritma se najviše susrećemo u matematici i informatici pa ćemo prikazati i primjer rješenja klasičnog problema na dva različita načina:

Primjer 1: Implementacije dva različita algoritma za izračunavanje n faktorijela u programskom jeziku Python.

1. Iterativni algoritam:

```
def factorial(n):
    r = 1
    i = 2
    while i <= n:
        r *= i
        i += 1
    return r
```

2. Rekurzivni algoritam:

```
def factorial(x):
    if x <= 1:
        return 1
    else:
        return x * factorial(x-1)
```

Svaki od ova dva algoritma je točan ali se bitno razlikuju po učinkovitosti tj. po brzini izvršavanja i količini resursa koje koriste pri izračunavanju.

Vidjeli smo da smo u Primjeru 1 imali algoritam koji je zapisan riječima dok smo u Primjeru 2 imali algoritam zapisan programskim kodom pa ćemo se osvrnuti i na činjenicu da postoji više vrsta zadavanja algoritma. Algoritam možemo zadati na barem 4 različita načina:

- tekstualno

- grafički
- pseudokodom ili kodom
- strukturogramom.

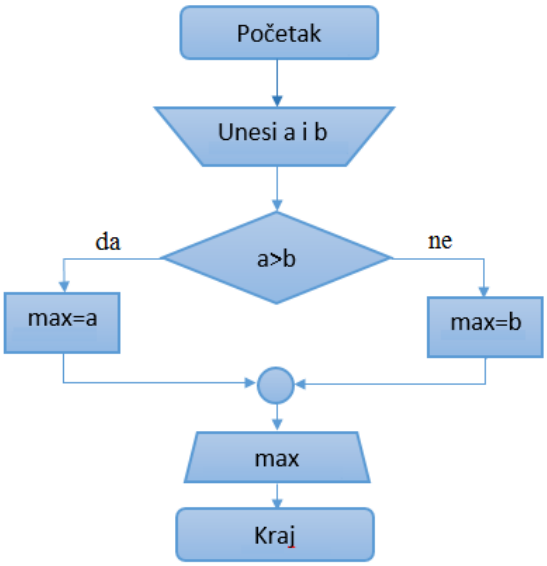
Tekstualno:
Moraju se koristiti precizne rečenice govornog jezika. Uglavnom se upotrebljava tamo gdje se osobe prvi put susreću s pojmom algoritma. Dobra strana ovog načina je to što je razumljiv za širi krug ljudi, a negativna strana je to što je govorni jezik često neprecizan pa izvršitelj može na više različitih načina shvatiti jednu instrukciju. Primjer: Algoritam za aktivaciju aparata za gašenje požara.

- donesite aparat na mjesto požara
- izvucite osigurač pokretne ručice na ventilu
- dlanom udarite pokretnu ručicu ventila do kraja
- pričekajte 5 sekundi
- uperite diznu ventila prema požaru i pritisnite pokretnu ručicu do kraja
- mlazom praha pokrivajte zapaljene površine sve dok se požar ne ugasi.

Grafički:
U grafičkom zadavanju algoritma koriste se određeni grafički simboli od kojih svaki predstavlja neku aktivnost u algoritmu. Ideja potiče iz teorije grafova pa se algoritam prikazuje u obliku usmjerenog grafa u kojem čvorovi predstavljaju aktivnosti koje se obavljaju u algoritmu. Strelicama su označene sljedeće aktivnosti koje se trebaju obaviti.
Blok dijagram je grafički način prikazivanja algoritma skupom simbola koji označuju pojedine radnje, a njihov raspored i povezanost određuju slijed postupaka. Slika 1.1 prikazuje primjer dijagrama toka za ispisivanje većeg od dva broja.

Pseudokod ili kod:
Pseudokod je algoritam opisan riječima u obliku niza naredbi koje se nalaze jedna ispod druge. Slika 1.2 prikazuje primjer pseudokoda za turnirsku selekciju. U Primjeru 1 smo vidjeli kako izgleda algoritam opisan kodom u Pythonu.

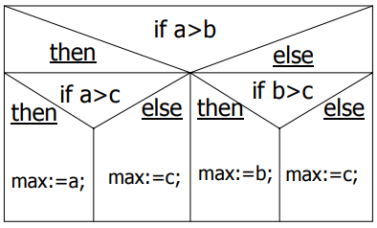
Strukturogram:
Strukturogram je kombinacija grafičkog prikaza i pseudokoda. Koristi se kao dokumentacija za već završene programe. Algoritam se piše tako što geometrijski nadopunjujemo sliku gdje svaki blok znači jedan dio instrukcija. Slika 1.3 prikazuje strukturogram gdje tražimo maksimalni element od tri zadana broja.



Slika 1.1: Dijagram toka

broj trenutnih članova = 1;
while *broj trenutnih članova* $\leq \lambda$ **do**
    odaberi nasumičnih k jedinki;
    odaberi najbolju među njima s obzirom na vrijednost funkcije cilja, označi ju s i;
    roditelji[broj trenutnih članova] = i;
    broj trenutnih članova = broj trenutnih članova + 1;
end

Slika 1.2: Pseudokod za turnirsku selekciju



Slika 1.3: Strukturogram za traženje najvećeg od 3 elementa

Poglavlje 2

Algoritamsko mišljenje

2.1 Što je algoritamsko mišljenje

Kada govorimo o algoritmima općenito neizbježno je spomenuti jedan pojam koji je usko vezan uz algoritme i čija su posljedica zapravo algoritmi. Govorimo o pojmu algoritamskog mišljenja ili algoritamskog pristupa problemu. U nekim dokumentima koristi se i termin računalno razmišljanje. Algoritamski način mišljenja *(eng. Computational Thinking - CT)* je pristup rješavanju problema koji može biti automatiziran te kao takav proveden i pomoću računala. Primjenjivo je u mnogim situacijama kao što su rješavanje problema iz svakodnevnog života, problema u medicini, problema iz područja STEM, ekonomije itd. Algoritamsko mišljenje obuhvaća nekoliko istaknutih koraka:

- **formuliranje problema:** za zadani problem potrebno je prvo izgraditi matematički model
- **pronalaženje, analiziranje i implementacija mogućih rješenja:** nakon što formuliramo problem, potrebno je razmisliti na koje načine se dati problem može riješiti te pronaći najefikasniji način za rješavanje. U matematičkom i informatičkom smislu potrebno je onda osmisliti implementaciju rješenja, a onda je i provesti.
- **poopćavanje:** nakon što se provede implementacija, potrebno je promisliti u kakvim uvjetima rješenje funkcionira, je li ga moguće primijeniti u više situacija te treba li nešto promijeniti ili nadodati ako postoji neki rubni slučaj za koji rješenje eventualno ne bi funkcioniralo.

Algoritamsko mišljenje kao takvo nam daje odgovore i na mnoga pitanja pri suočavanju s problemom kao što su ”koliko je problem težak?”, ”koji je najbolji način za njegovo rješavanje?”, ”može li se problem svesti na više manjih problema?” itd.

Iako je algoritamsko mišljenje puno općenitiji pojam ono se najviše veže za računarstvo, odnosno programiranje. Algoritamska strategija rješavanja problema koristi zapravo osnovne tehnike i koncepte računarstva *(Computer Science)*.
Gore navedeno u skladu je s preporukama udruga:

- *International Society for Technology in Education - ISTE*
- *Computer Science Teachers Association - CSTA*.

2.2 Algoritamsko mišljenje u važećim dokumentima

Kako se u ovom radu bavimo algoritmima u gimnazijama, malo detaljnije ćemo prikazati u kojem kontekstu se algoritamsko mišljenje spominje u dokumentima *Hrvatski nacionalni obrazovni standard (HNOS)* [2] i *Nacionalni okvirni kurikulum (NOK)* [4]. Riječ je o starim dokumentima i u ovom radu se na njima nećemo detaljnije zadržavati.

Algoritamsko mišljenje se javlja već jako rano pa tako i u *HNOS*-u za osnovne škole već imamo spominjanje istog. Kod izbornog predmeta informatika u *HNOS*-u piše sljedeće:

"Predmet Informatika treba omogućiti učenicima upoznavanje s informacijskom i komunikacijskom tehnologijom.
Nastavni sadržaji iz područja informacijske i komunikacijske tehnologije moraju učenicima omogućiti: stjecanje umijeća uporabe današnjih računala i primjenskih programa (vještine), upoznavanje s osnovnim načelima i idejama na kojima su sazdana računala odnosno informacijska i komunikacijska tehnologija (temeljna znanja) te razvijanje sposobnosti za primjene informacijske i komunikacijske tehnologije u različitim primjenskim područjima (rješavanje problema).
Umijeća, temeljna znanja i rješavanje problema tri su sastavnice obrazovnoga procesa koje se mogu razmatrati i djelomično odvojeno, ali tek njihovo međusobno prožimanje dat će učenicima dobru podlogu za buduće cjeloživotno učenje.
Podrobni opis umijeća, temeljnih znanja i rješavanja problema nalazi se u dokumentu Znanja i umijeća iz informacijske i komunikacijske tehnologije koje treba steći tijekom cjelokupnoga školovanja koji je sastavni dio HNOS-a.
U okviru nastavnog predmeta učenici moraju naučiti djelotvorno upotrebljavati računala i biti sposobni ugraditi osnovne zamisli ***algoritamskog načina razmišljanja*** *u rješavanje svakodnevnih problema.*
Stoga, nastavni program mora osposobiti učenike:

- *za rješavanje problema;*

- *za komuniciranje posredstvom različitih medija;*
- *za prikupljanje, organiziranje i analizu podataka te za njihovu sintezu u informacije;*
- *za razumijevanje i kritičku ocjenu prikupljenih informacija;*
- *za donošenje zaključaka na temelju prikupljenih informacija;*
- *za timski rad pri rješavanju problema".* (vidi [2])

U *Nacionalnom okvirnom kurikulumu (NOK-u)* algoritamsko mišljenje nalazimo u četvrtom ciklusu za gimnazije u poglavlju *Tehničko i informatičko područje*. U odgojno-obrazovnim ciljevima tehničkog i informatičkog područja među ostalim ciljevima također stoji:

*"razviti **algoritamski način razmišljanja**, steći vještine i sposobnosti primjene računala pri rješavanju problema u različitim područjima primjene".* (vidi [4])

S obzirom na to da u današnjem vremenu ne možemo funkcionirati bez tehnologije loše je što se algoritamsko mišljenje javlja među odgojno obrazovnim ciljevima samo u gimnazijama. Informatička pismenost je jako važna u današnjem svijetu te je možemo čak usporediti i s pismenosti u jeziku pa je neophodno govoriti o algoritamskom razmišljanju i u školama koje nisu gimnazije.

Kao dobar primjer dokumenta navodimo standard računalnih znanosti *CSTA K-12 Computer Science Standard* [1]. Malo detaljnije je opisano što se podrazumijeva pod pojmom algoritamskog mišljenja, tj. precizno su opisana učenička postignuća koji se očekuju od učenika nakon što završi svaku od razina. U kurikulumu *K-12* postoje 3 glavne razine:

1. Razina 1: *(Level 1)* Razredi 1-6 *(Grades K1-6)*
2. Razina 2: *(Level 2)* Razredi 6-9 *(Grades K6-9)*
3. Razina 3: *(Level 3)* Razredi 9-12 *(Grades K9-12).*

Više detalja može se naći u [1].

Detaljnije ćemo promotriti prijedlog novog kurikuluma *Nacionalni kurikulum nastavnog predmeta informatika (PRIJEDLOG)* [3]. Smatramo da je riječ o dobro napisanom dokumentu koji korespondira s aktualnim dokumentima u svijetu.

Prema prijedlogu novog kurikuluma, četiri su domene kojima bi se trebali realizirati ciljevi predmeta informatika:

- A. informacije i digitalna tehnologija
- B. računalno razmišljanje i programiranje
- C. digitalna pismenost i komunikacija
- D. e-društvo.

Kako se prvenstveno bavimo algoritamskim mišljenjem (računalnim razmišljanjem) pogledajmo što stoji pod opisom te domene:

"Razvijanje računalnog razmišljanja njeguje pristup rješavanju problema koji je primjenjiv na računalu. Takvim pristupom učenici nisu samo korisnici različitih alata nego postaju i njihovi stvaratelji. Razvijaju se vještine logičkoga zaključivanja, modeliranja, apstrahiranja te rješavanja problema. Računalno razmišljanje univerzalna je vještina koja potiče preciznost i sustavnost, a može se primijeniti u različitim područjima i u svakodnevnome životu. Apstrakcija kao temeljni koncept računalnoga razmišljanja potiče uporabu metakognitivnih vještina te omogućuje rad na složenim problemima razdvajajući ih u više jednostavnih problema. Kvalitetnim informatičkim obrazovanjem koje se temelji na računalnom razmišljanju i kreativnosti omogućuje se razumijevanje i mijenjanje svijeta koji nas okružuje. Rješavanje nekog problema izradom računalnoga programa uključuje standardne postupke razvoja programa, ali i inovativnost, poduzetnost te preuzimanje inicijative pri izradi dizajna i razvoja novih modela i proizvoda primjenom računalne tehnologije. Programiranje razvija samopouzdanje, upornost i preciznost u ispravljanju pogrešaka, sposobnost komunikacije i zajedničkoga rada usmjerenoga prema postizanju određenoga cilja." (vidi [3])

Detaljno su razrađeni odgojno-obrazovni ishodi, razine ishoda, razine usvojenosti te preporuke za ostvarenje odgojno-obrazovnih ishoda po razredima i domenama. Što se tiče srednjih škola razrađeni su ishodi po vrstama škola:

- Opće, jezične, klasične i prirodoslovne gimnazije te opće obrazovni dio strukovne škole 2 x 70 sati godišnje - 1. i 2. razred
- Prirodoslovno-matematičke gimnazije 4 x 70 sati godišnje (a,c i d program) - 1. - 4. razred
- Prirodoslovno-matematičke gimnazije 4 x 105 sati godišnje (b program) - 1. - 4. razred.

Izdvojit ćemo samo ishode prema vrsti škole.

Opće, jezične, klasične i prirodoslovne gimnazije te opće obrazovni dio strukovne škole:
Nakon dvije godine učenja informatike u domeni *računalno razmišljanje i programiranje* učenik:

- analizira problem, definira ulazne i izlazne vrijednosti te uočava korake za rješavanje problema
- primjenjuje jednostavne tipove podataka te argumentira njihov odabir, primjenjuje različite vrste izraza, operacija, relacija i standardnih funkcija za modeliranje jednostavnoga problema u odabranome programskom jeziku
- razvija algoritam i stvara program u odabranome programskom jeziku rješavajući problem uporabom strukture grananja i ponavljanja.
- analizira osnovne algoritme s jednostavnim tipovima podataka i osnovnim programskim strukturama i primjenjuje ih pri rješavanju novih problema
- u zadanome problemu uočava manje cjeline, rješava ih te ih potom integrira u jedinstveno rješenje problema
- rješava problem primjenjujući jednodimenzionalnu strukturu podataka
- u suradnji s drugima osmišljava algoritam, implementira ga u odabranome programskom jeziku, testira program, dokumentira i predstavlja drugima mogućnosti i ograničenja programa.

Prirodoslovno-matematičke gimnazije 4 x 70 sati godišnje:
U domeni *računalno razmišljanje i programiranje* nakon četiri godine uz ishode navedene za opće, jezične, klasične i prirodoslovne gimnazije te opće obrazovni dio strukovne škole učenik još:

+ analizira sortiranje podataka kao važan koncept za rješavanje različitih problema
+ vizualizira i grafički prikazuje problem
+ rješava problem primjenjujući složene tipove podataka definirane zadanim programskim jezikom
+ rješava problem primjenjujući rekurzivnu funkciju
+ uspoređuje različite algoritme sortiranja i pretraživanja podataka

+ vrednuje algoritme prema njihovoj vremenskoj složenosti
+ analizira tradicionalne kriptografske algoritme i opisuje osnovnu ideju modernih kriptografskih sustava
+ definira problem iz stvarnoga života i stvara programsko rješenje prolazeći sve faze programiranja, predstavlja programsko rješenje i vrednuje ga
+ osmišljava objektni model s pripadnim složenim strukturama podataka i implementira ga u zadanome programskom jeziku
+ rješava problem koristeći se apstraktnim strukturama podataka
+ stvara aplikaciju s grafičkim korisničkim sučeljem za rješavanje problema iz stvarnoga života
+ se koristi modeliranjem i simulacijom za predstavljanje i razumijevanje prirodnih fenomena
+ definira problem iz stvarnoga života i stvara programsko rješenje prolazeći sve faze programiranja, predstavlja programsko rješenje i vrednuje ga.

Prirodoslovno-matematička gimnazija 4 x 105 sati godišnje:
U domeni *računalno razmišljanje i programiranje* osim ishoda navedenih za prirodoslovno-matematičke gimnazije 4 x 70 sati godišnje, učenik još:

+ razlikuje složene tipove podataka u zadanome programskom jeziku te pri rješavanju problema koristi se funkcijama i metodama definiranima nad njima
+ se koristi različitim programskim paradigmama za rješavanje problema iz stvarnoga života.

Više detalja može se naći u [3].

Poglavlje 3

Algoritmi u osnovnoj školi

U ovom poglavlju ćemo opisati koji algoritmi se obrađuju u osnovnoj školi kao uvod u algoritme koji se obrađuju u srednjoj školi. Kako smo u prvom poglavlju opisali, algoritam je konačan niz koraka koje treba napraviti za postizanje nekog cilja. To znači da se tu ubrajaju i algoritmi za ispravno spajanje računala, ispravno fizičko rukovanje računalom, otvaranje i spremanje podataka te slične aktivnosti, no time se nećemo baviti u ovom radu. Naglasak će svakako biti na algoritmima koji se odnose na samo programiranje u programskim jezicima koji se uvode u osnovnoj školi.

3.1 Algoritmi u petom razredu

U ovom poglavlju ćemo se osvrnuti u kojem kontekstu se algoritmi spominju u 3 udžbenika petog razreda koji su trenutno aktualni u osnovnim školama, a to su udžbenici:

1. *NIMBUS OBLAK 5, udžbenik informatike s e-podrškom za peti razred osnovne škole* [10]

2. *Informatika+ 5, udžbenik iz informatike za 5. razred osnovne škole* [17]

3. *Moj PORTAL 5, udžbenik informatike za 5. razred osnovne škole* [9].

Odmah u petom razredu imamo uvod u algoritam te spominjanje pojma algoritma u tri različita oblika: *"Algoritam je postupak kojim se opisuje točan redoslijed kojim obavljamo neki posao."* (vidi [17]), *"Algoritam je detaljno opisan logički slijed radnji koje vode do rješenja. S istim ulaznim podatcima uvijek daje isto rješenje. Pisan je jednostavnim i čovjeku razumljivim rječnikom."* (vidi [10]) te *"Precizno napravljen plan izvedbe nekog slijeda postupaka nazivamo ALGORITAM"* (vidi [9]). Odmah uočavamo da nismo baš zadovoljni opisom pojma algoritma u niti jednom udžbeniku. Opet se spominju izrazi kao

što su *"...detaljno opisan..."*, *"algoritam je postupak..."* te *"...nekog slijeda postupaka..."* što ukazuje da je algoritam skup više postupaka kao što smo spominjali u prvom poglavlju. Spominju se *algoritmi slijeda* te *algoritmi grananja* što također nisu ispravni termini. Algoritmi se poistovjećuju s naredbama. Dani su primjeri algoritama iz stvarnog života te par jednostavnih primjera u programskim jezicima LOGO te Small Basic-u i QBasic-u.

Prikazat ćemo jedan zanimljiv primjer iz stvarnog života naveden u jednom od udžbenika:

Primjer 2: Algoritam za pisanje zadaće iz matematike: (vidi [17])

početak
potraži bilježnicu i udžbenik
otvori bilježnicu
pripremi našiljenu olovku i gumicu
pronađi zadatke koji su za zadaću
pročitaj ih
ako nešto ne razumiješ onda
prouči iz bilježnice što ste učili u školi
pronađi u udžbeniku slične riješene zadatke
riješi zadane zadatke
provjeri na kraju udžbenika jesi li točno riješio zadatke
ako imaš pogrešaka, **onda**
ispravi pogreške
ako ne znaš ispraviti, **onda** sutra pitaj učiteljicu
inače
zadaća je gotova
kraj

U ovom primjeru iz stvarnog života vidimo indirektno uvođenje forme programskog jezika korištenjem uvlaka tako da učenik jasno vidi koji dio spada pod koju granu u grananju.
Od primjera u programskom jeziku LOGO koriste se vrlo jednostavni primjeri crtanja pravilnih geometrijskih likova i slično, a od primjera u jezicima Small Basic i QBasic imamo također samo primjere učitavanja i ispisivanja te korištenja naredbe grananja. Još je opisan grafički prikaz algoritma, tj. obrađen je dijagram toka izvođenja nekog programa no o tome nećemo puno govoriti u ovom radu.

3.2 Algoritmi u šestom razredu

Udžbenici koje smo za ovo poglavlje koristili su:

1. *NIMBUS OBLAK 6, udžbenik informatike s e-podrškom za šesti razred osnovne škole* [11]
2. *Informatika+ 6, udžbenik iz informatike za 6. razred osnovne škole* [18]
3. *Moj PORTAL 6, udžbenik informatike za 6. razred osnovne škole* [8].

U opisanim udžbenicima, kao i u petom razredu obrađuju se programski primjeri u jezicima LOGO te Small Basic i QBasic. Što se tiče samog programiranja prvo slijedi ponavljanje onoga što se radilo u petom razredu, a zatim slijede primjeri korištenja naredbi za kontrolu toka.
Ono na čemu je naglasak u šestom razredu je primjena **naredbi za ponavljanje (petlji)**. Opis naredbi za ponavljanje u udžbenicima nije baš ispravan, a glasi: *"Algoritam petlje je postupak u kojemu zadane naredbe treba ponoviti konačan broj puta."* (vidi [18]). Koristi se termin *algoritam petlje* što ne postoji.
Kroz primjere u programskom jeziku LOGO vidimo da je petlju FOR zgodno koristiti za crtanje pravilnih likova određenih ulaznim varijablama. Uzmimo za primjer crtanje pravilnih n-terokuta ili zvijezde sa n krakova.
Slijedi jedan od težih primjera korištenja FOR-petlje u kojem imamo dvije ugniježđene FOR-petlje te ispisivanja i objašnjavanja koraka.

Primjer 3: Napiši program koji će ispisati tablicu množenja do vrijednosti ulazne varijable `X`.

Rješenje:

```
INPUT "Upisi broj do kojeg zelis tablicu mnozenja:", X
FOR i=1 TO X
FOR j=1 TO X
PRINT i; "*"; j; "="; i*j
NEXT j
NEXT i
```

Petlja s varijablom `i` naziva se vanjska petlja. Njezina se vrijednost mijenja od `1` do `X`. `X` je ulazna varijabla i unosimo je proizvoljno. Petlja s varijablom `j` naziva se unutarnja (ugniježđena) petlja i poprima vrijednosti od `1` do `X`.

Vanjska petlja mijenja i određuje vrijednost prvog faktora, a unutarnja drugog faktora. Uz pomoć tih vrijednosti izračunavaju se rezultati za sve kombinacije brojeva. Pogledajmo tablicu i bit će nam puno jasnije.

X	I	J	NAREDBA	ISPIS
3	1	1	`PRINT i; "*"; j; "="; i * j`	1*1=1
3	1	2	`PRINT i; "*"; j; "="; i * j`	1*2=2
3	1	3	`PRINT i; "*"; j; "="; i * j`	1*3=3
3	2	1	`PRINT i; "*"; j; "="; i * j`	2*1=2
3	2	2	`PRINT i; "*"; j; "="; i * j`	2*2=4
3	2	3	`PRINT i; "*"; j; "="; i * j`	2*3=6
3	3	1	`PRINT i; "*"; j; "="; i * j`	3*1=3

Primjećujemo da je program izvršio sve korake i točno ispisao umnoške. (vidi [11])

3.3 Algoritmi u sedmom razredu

Za sedmi razred obrađeni su sljedeći udžbenici:

1. *NIMBUS OBLAK 7, udžbenik informatike s e-podrškom za sedmi razred osnovne škole* [12]
2. *Informatika+ 7, udžbenik iz informatike za 7. razred osnovne škole* [19]
3. *Moj PORTAL 3.0 7, udžbenik informatike u sedmom razredu osnovne škole* [6].

Što se tiče samog programiranja, malo se razlikuje gradivo u udžbeniku *Informatika+ 7* u odnosu na ostala dva udžbenika. Programski jezici koji se obrađuju u udžbeniku *Informatika+ 7* su Python i Basic dok se u druga dva udžbenika obrađuju LOGO i Basic. Uglavnom se utvrđuje gradivo iz petog i šestog razreda. Rade se zadaci u kojima se ponavljaju **naredbe za kontrolu toka** i **naredbe za ponavljanje** koje se koriste u koordinatnoj grafici te pri crtanju likova, kružnica i slično.
U udžbeniku *Moj PORTAL 3.0 7* imamo kao izbornu temu ”Osnove rekurzivnog programiranja”. U udžbeniku stoji sljedeće:

"Mnoge se radnje (postupci) u svakodnevnom životu ponavljaju istim slijedom. U programiranju rabimo petlje ako pojedine naredbe želimo provesti nekoliko puta: REPEAT, FOR itd. Petlje skraćuju pisanje programa u kojima se programske naredbe ponavljaju.

*U ovom poglavlju učit ćemo o **petlji** koju rabimo u složenijim zadacima - tzv. **rekurziji**. U rekurziji cijeli program postaje dio petlje tako što program poziva samog sebe. Takav program nazivamo **rekurzivnim programom**."* (vidi [6])

Odmah ćemo se kritički osvrnuti na sami opis rekurzije. Ovaj nam opis kaže da je rekurzija zapravo petlja, što nije istina. Rekurzija je funkcija (procedura) koja u definiciji poziva samu sebe dok je petlja naredba u računalnom programu koja omogućuje da se dijelovi programa izvrše više puta. Dakle, rekurziju ne smijemo poistovjetiti s petljom.
U udžbeniku se radi primjer ispisa *n* prirodnih brojeva pomoću rekurzije. Riječ je o nerekurzivnom problemu.
Još jedna nelogična stvar je da se prvo obrađuje rekurzija, a zatim se objašnjava pojam funkcije.
Sljedeći pojam opisan u ovom poglavlju je tzv. **repna rekurzija**. Pogledajmo sljedeći primjer:

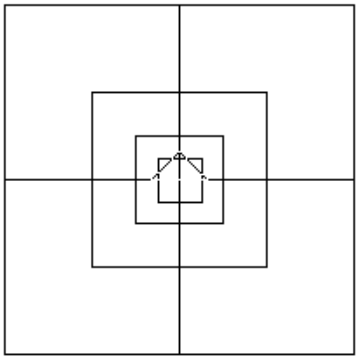
Primjer 4: Nacrtat ćemo kvadrat u kvadratu u sva četiri kvadranta koordinatnog sustava. Program ćemo nazvati `META :D :N`. Varijablom `:D` određena je početna duljina stranice najvećeg kvadrata, a varijablom `:N` broj kvadrata.
(U prijašnjem zadatku smo definirali rekurzivnu funkciju `KVADRAT` koja crta kvadrat.)

```
TO KVADRAT :D :S
   IF :S=0 [STOP]
   FD :D RT 90
   KVADRAT :D :S-1
END
```

PROGRAM (META 100 4)

```
TO META :D :N
   IF :N=0 [STOP]
   REPEAT 4[KVADRAT :D 4 RT 90]
   META :D/2 :N-1
END
```

ZADANI LIK



Ako se rekurzivni poziv nalazi na kraju programa riječ je o tzv. **repnoj rekurziji**. Nakon svakoga rekurzivnog poziva duljina stranice smanjuje se na polovicu, a broj preostalih kvadrata za jedan. (vidi [6])

3.4 Algoritmi u osmom razredu

Korišteni udžbenici za ovo poglavlje su:

1. *NIMBUS OBLAK 8, udžbenik informatike s e-podrškom za osmi razred osnovne škole* [13]
2. *Informatika+ 8, udžbenik iz informatike za 8. razred osnovne škole* [20]
3. *Moj PORTAL 3.0 8, udžbenik informatike u osmom razredu osnovne škole* [7].

U osmom razredu gradivo se razlikuje od udžbenika do udžbenika.

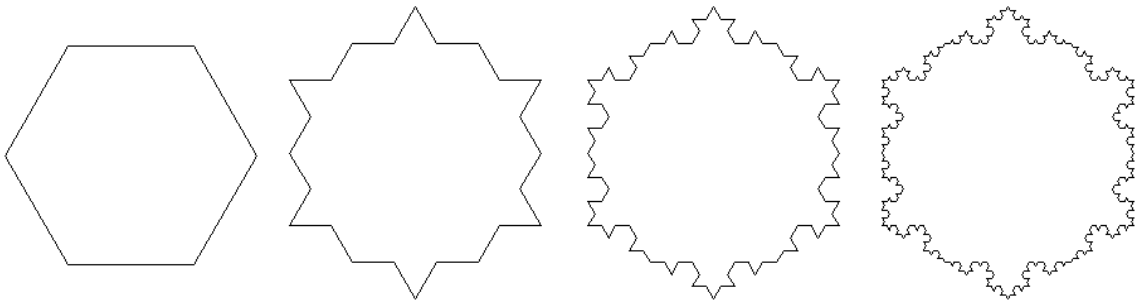
U udžbeniku *NIMBUS OBLAK 8* imamo prvo programiranje u programskom jeziku LOGO što uključuje ponavljanje gradiva 7. razreda i logičke petlje. Ovo je jedan od tri udžbenika u kojem pod obvezno gradivo spada *Rekurzivno programiranje* no obrađena su samo dva primjera rekurzivnog programa. Od programiranja u Basic-u ponovljeno je gradivo 7. razreda te su obrađeni još potprogrami i funkcije.

U udžbeniku *Moj PORTAL 3.0 8* imamo izbornu temu *Primjeri rekurzivnog programiranja* u kojem su obrađeni malo bolji primjeri korištenja rekurzivnog algoritma. Kad je riječ o fraktalima prikazani su primjeri rekurzivnog algoritma za iscrtavanje *simetričnog ili*

binarnog stabla te *Kochove pahuljice*. Pogledajmo sljedeći primjer.

Primjer 5: Napisat ćemo program koji će, krenuvši od mnogokuta, nacrtati Kochovu krivulju određene zadane razine.

`:n` - razina krivulje
`:m` - broj stranica pravilnog mnogokuta
`:a` - duljina stranice pravilnog mnogokuta



`Koch 0 6 100` `Koch 1 6 100` `Koch 2 6 100` `Koch 3 6 100`

U početnoj nultoj razini prikazan je zadani geometrijski lik. U prvoj razini svaka stranica geometrijskog lika zamijenjena je krivuljom. U drugoj razini svaka stranica krivulje zamijenjena je krivuljom. I tako redom do broja zadanih razina.

PROGRAM:

```
TO koch :n :m :a
  CS
  RT 90
  REPEAT :m [krivulja :n :a RT 360/:m]
END
```

OBJAŠNJENJE:

Glavni program kojim omogućujemo crtanje mnogokuta `:m`. Na početku programa kornjaču okrećemo za 90 stupnjeva kako bi početna stranica bila vodoravna s prozorom.
Pri crtanju mnogokuta umjesto crtanja stranica naredbom **`FD`** `:a` pozivamo proceduru **`krivulja`** `:n :a`, koja će nacrtati stranice mnogokuta u obliku krivulje.

PROGRAM:

```
TO krivulja :n :a
  IF :n=0[FD :a STOP]
  krivulja :n-1 :a/3 LT 60
  krivulja :n-1 :a/3 RT 120
  krivulja :n-1 :a/3 LT 60
  krivulja :n-1 :a/3
END
```

OBJAŠNJENJE

Uvjet kojim određujemo kraj rekurzije `:n=0` Samo u početnoj točki crtamo isječak krivulje `FD :a`.
Svaki isječak krivulje crtamo rekurzivnim pozivom programa s umanjenom veličinom stranice `:a/3`. Nakon povratka iz rekurzivnog poziva postavljamo kut crtanja idućeg isječka krivulje `LT 60`.
Postupak se ponavlja za svaki isječak. (vidi [7])

Ostatak gradiva koje se obrađuje u ovom udžbeniku je grafika te procedure i funkcije.

U udžbeniku Informatika+ 8 se obrađuje programiranje u programskim jezicima Python i Basic za razliku od druga dva udžbenika. Prvi put u udžbenicima Informatika+ se spominju rekurzivne procedure. U ovom udžbeniku stoji: *"Procedura koja u sebi ima naredbu koja poziva tu istu proceduru zove se **rekurzivna procedura** ili **rekurzija**."* (vidi [20]) što relativno dobro opisuje rekurziju.
Primjeri su slični kao oni u prethodnom udžbeniku.
Od gradiva se obrađuju još grafičke kontrole te primjena programiranja u drugim znanostima.

Poglavlje 4

Algoritmi u srednjoj školi

U ovom poglavlju ćemo se bazirati na algoritme koji se obrađuju u srednjim školama, odnosno gimnazijama. Prvo ćemo se kratko osvrnuti na algoritme koji se obrađuju u općim i jezičnim gimnazijama, a onda na algoritme u prirodoslovnim gimnazijama tj. gimnazijama s pojačanom nastavom matematike i informatike.

4.1 Algoritmi u općim i jezičnim gimnazijama

Opće i jezične gimnazije obrađujemo u istom poglavlju zato što je fond sati nastave informatike jednak tj. nastava informatike se izvodi u 2 sata vježbi te jedan sat predavanja tjedno, jednu školsku godinu - prvu u općim i drugu u jezičnim
Korištena je sljedeća literatura za ovo poglavlje:

1. *INFORMATIKA 1, udžbenik za prvi razred prirodoslovno matematičkih i općih gimnazija te drugi razred klasičnih i jezičnih gimnazija* [16].

Samo ćemo napomenuti da se algoritmi eksplicitno ne obrađuju te nema spomenutih poznatijih algoritama u ovom udžbeniku. Razlog tomu, kao što je i spomenuto, je premala satnica te izvođenje informatike samo jednu školsku godinu.

4.2 Algoritmi u prirodoslovnim gimnazijama

U nastavi informatike za prirodoslovne gimnazije literatura nije određena za svaki razred posebno nego imamo različite vrste literature: literatura za sve četiri godine u jednom udžbeniku, literatura za prvi i drugi razred u dva različita udžbenika bez literature za treći i četvrti razred od istog izdavača i sl. Zato ćemo opisati algoritme za srednju školu u jednom poglavlju bez podjela na razrede.

Za ovo poglavlje koristili smo sljedeću literaturu:

1. *RJEŠAVANJE PROBLEMA PROGRAMIRANJEM U PYTHONU* [14]
2. *INFORMATIKA 2, udžbenik iz informatike za 2. razred prirodoslovno-matematičkih gimnazija* [5]
3. *NAPREDNO RJEŠAVANJE PROBLEMA PROGRAMIRANJEM U PYTHONU* [15].

Programski jezik koji se koristi u ovim udžbenicima je Python. Programski jezik Logo se na neki način stopio s Pythonom jer u njemu također imamo modul za crtanje *turtle* pa je jednostavnije radi sintakse koristiti samo Python. Većina algoritama koji se obrađuju u ovim udžbenicima koriste strukturu liste. Osvrnut ćemo se dakle na algoritme sortiranja, pretraživanja i još neke poznatije algoritme kao što su Hornerov i Euklidov algoritam.

Algoritmi za sortiranje

U udžbeniku *INFORMATIKA 2* opisano je sortiranje na sljedeći način: *"Jedan od važnijih osnovnih algoritama jest onaj za sortiranje liste. Sortirati listu znači poredati njezine elemente po nekom kriteriju."*. Zatim imamo objašnjenje kakvo sortiranje imamo:

- ***uzlazno*** *- prvi je element nakon sortiranja najmanji, a zadnji je najveći od svih*
- ***silazno*** *- prvi je element nakon sortiranja najveći od svih, a zadnji najmanji.*

"Napišimo najprije funkciju koja za zadanu listu L određuje njezin najmanji element. Važno nam je i na kojem se mjestu u listi taj element nalazi, pa ćemo napraviti funkciju koja vraća najmanji element i njegov indeks u listi."

```
def najmanji(L):
    min_i = 0; min = L[0]
    for i in range(1,len(L)):
        if L[i] < min:
            min = L[i]
            min_i = i
    return min, min_i
```

"Logika je vrlo jednostavna - za početak pretpostavimo da se najmanji element liste nalazi na indeksu 0. Zatim prolazimo kroz sve ostale elemente i uspoređujemo ih s dosad pronađenim najmanjim elementom. Ako naiđemo na neki element koji je manji od dosad najmanjega, zapamtimo njega i njegov indeks kao nove najmanje. Vrlo je zgodno što taj

program radi i za liste koje sadrže brojeve i za liste koje sadrže stringove."
"Do algoritma za sortiranje dijeli nas još jedan mali korak, a to je zamjena vrijednosti dvaju elemenata u listi. Napišimo prvo program koji dovodi najmanji element u listi na prvo mjesto:"

```
min, min_i = najmanji(L)
temp = L[0]
L[0] = L[min_i]
L[min_i] = temp
```

"Funkcijom `najmanji` pronašli smo indeks `min_i` na kojemu se nalazi najmanji element u listi. Nakon toga želimo zamijeniti taj element i element koji se nalazi na indeksu `0`. Prvo smo spremili vrijednost `L[0]` u privremenu varijablu `temp`, a zatim smo `L[0]` "pregazili" s `L[min_i]`. U ovome trenutku u listi se nalaze dva jednaka elementa (min) na indeksima `0` i `min_i`. No stara vrijednost koja se prije nalazila u `L[0]` zapamćena je u varijabli `temp`, pa je možemo spremiti u `L[min_i]`." (vidi [5])

Navedeno je još da se zamjena elemenata liste u programskome slengu naziva **swap** što je i potkrijepljeno implementacijom u Pythonu.

"Sada imamo sve sastavne elemente za sortiranje liste. Ideja je sljedeća: pronaći ćemo najmanji element u listi i dovesti ga na indeks `0` zamjenom, kao što smo to napravili gore. Nakon toga ćemo u ostatku liste (od indeksa `1` pa nadalje) ponovno pronaći najmanji element, dovesti ga zamjenom na indeks `1` i tako dalje."

```
for i in range(0, len(L)):
    min, min_i = najmanji(L[i:])
    min_i = min_i + i
    temp = L[i]; L[i] = L[min_i]; L[min_i] = temp
```

*"Gornji algoritam za sortiranje nazivamo **selection sort**. Najčešće ga pišemo ovako:*

```
for i in range(0, len(L)):
    for j in range(i+1, len(L)):
        if L[i] > L[j]:
            temp = L[i]; L[i] = L[j]; L[j] = temp
```

"Unutarnja petlja po varijabli `j` ima ulogu traženja najmanjega elementa u podlisti `L[i:]`. Čim pronađemo neki novi najmanji element, odmah ga dovodimo na indeks `i`." (vidi [5])

Nakon ovog je postavljeno jedno važno metodičko pitanje koje se samo nameće, a do-

bro ga je postaviti prvo učenicima da sami razmisle: *"Kako ćemo promijeniti algoritam da sortira silazno? Vrlo jednostavno - samo* `if L[i] > L[j]` *zamijenimo s* `if L[i] < L[j]`*."* (vidi [5])

Sljedeći algoritam za sortiranje koji se spominje je **bubble sort**.
"On se zasniva na sljedećoj ideji: ako lista nije sortirana onda postoje neka dva susjedna elementa koja su u "krivom" poretku. Na primjer, ako listu `L=[2, 5, 8, 4, 9]` *sortiramo uzlazno, onda uočavamo susjedne elemente* `8` *i* `4` *koji su u krivome poretku. Kada pronađemo takve elemente, zamijenimo ih i time dovodimo listu u stanje koje je bliže sortiranomu:* `L=[2, 5, 4, 8, 9]`*. Ponovno možemo uočiti elemente* `5` *i* `4` *u krivome poretku. Njihovom zamjenom dobivamo listu* `L=[2, 4, 5, 8, 9]`*, koja je sortirana jer više ne postoje susjedni elementi koji su u krivome poretku.*
U donjem kodu unutarnja FOR-petlja prolazi kroz cijelu listu u potrazi za susjednim elementima u krivome poretku. Ako pronađemo takve elemente, u varijabli `postoje_krivi` *bilježimo tu činjenicu i zamjenjujemo ih. Ako takvi elementi ne postoje (varijabla* `postoje_krivi` *zadrži vrijednost* `False` *nakon prolaska unutarnjom petljom), onda je lista sortirana i prekidamo vanjsku WHILE-petlju."* (vidi [5])

```
postoje_krivi = True
while postoje_krivi:
   postoje_krivi = False
   for i in range(0, len(L)-1):
      if L[i] > L[i+1]:
           postoje_krivi = True
           temp = L[i]
           L[i] = L[i+1]
           L[i+1] = temp
```

Navedeno je još kako se prvim prolaskom najveći element dovodi na kraj liste te svakim sljedećim prolaskom možemo gornju granicu unutarnje petlje smanjivati za jedan. Time je algoritam nešto ubrzan jer ne moramo uspoređivati elemente koji sigurno neće rezultirati zamjenom.

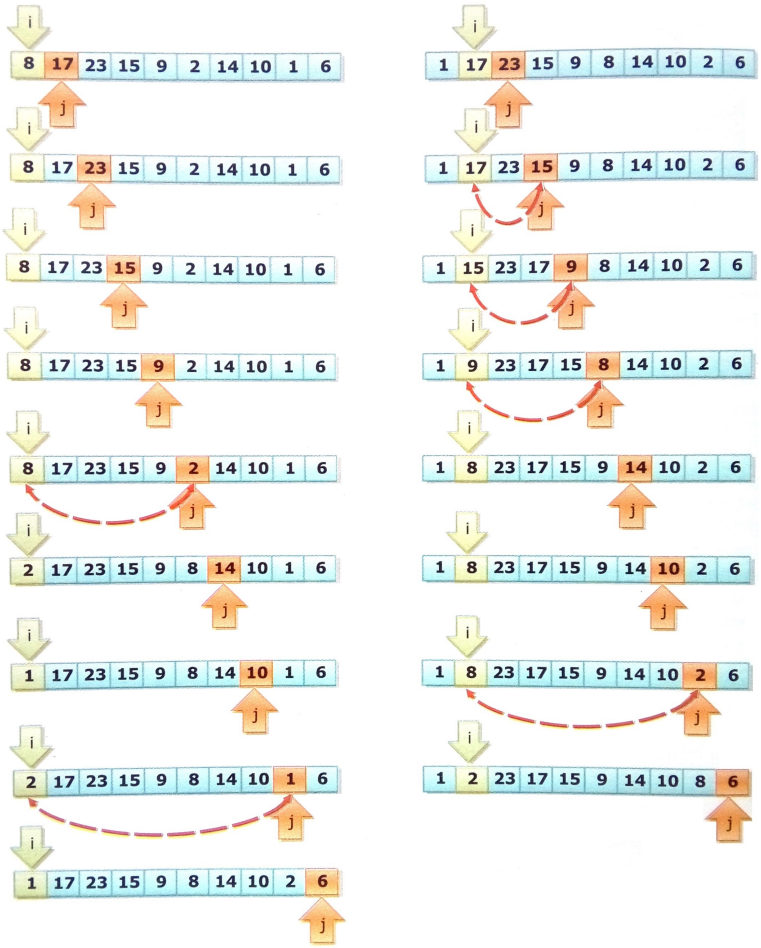
```
postoje_krivi = True; gornja_granica = len(L)
while postoje_krivi:
   postoje_krivi = False; gornja_granica = gornja_granica-1
   for i in range(0, gornja_granica):
      if L[i] > L[i+1]:
           postoje_krivi = True
           temp = L[i]
           L[i] = L[i+1]
```

```
            L[i+1] = temp
```

Kao i kod algoritma *selection sort*, promjenom uvjeta u IF naredbi smo mogli sortirati listu silazno.

U udžbeniku *NAPREDNO RJEŠAVANJE PROBLEMA PROGRAMIRANJEM U PYTHONU* nalaze se još neki algoritmi za sortiranje pored prethodna dva. To su sortiranje zamjenom elemenata (eng. *exchange sort*), sortiranje umetanjem (engl. *insertion sort*), sortiranje sjedinjavanjem (engl. *merge sort*), brzo sortiranje po Hoareu (engl. *quick sort*) te sortiranje razvrstavanjem u pretince (engl. *bucket sort*). Pogledajmo detaljnije prva tri algoritma te njihove opise u udžbeniku:

*"Sortiranje zamjenom elemenata (**exchange sort**) u prvi čas nam se nameće kao najjednostavniji način sortiranja elemenata liste. Radi se o tome da najprije uzmemo prvi element i uspoređujemo ga sa svim ostalim elementima. Svaki put kada naiđemo na element koji je manji, zamjenjujemo im mjesta. Nakon nakon što smo usporedili prvi element liste sa svim ostalim elementima liste, napravili smo jedan (prvi) **prolaz**. Nakon prvog prolaza na prvom mjestu liste bit će najmanji element. U drugom ćemo prolazu to isto učiniti s drugim elementom i tako prijeći cijelu listu za drugi element itd. Element koji unutar jednog prolaza uspoređujemo sa svim elementima koji se nalaze iza njega zvat ćemo **ključnim elementom**.*
Označimo s `i` i `j` pokazivače koji će pokazivati na elemente naše liste. Na samom početku u prvom prolazu pokazivač `i` će imati vrijednost `0`, a mi ćemo promatrati element s indeksom `0`. U ovom prolazu to je ključni element. Pokazivač `j` će početno pokazivati na prvi susjedni element s indeksom `1`, odnosno mi ćemo promatrati vrijednost te varijable.
Naš ključni element će prvo imati vrijednost `8`, a varijabla s indeksom `1` u tom trenutku ima vrijednost `17`, poredak tih dviju vrijednosti je u tom trenutku u skladu s našim zahtjevom te jednostavno povećavamo indeks `j`. Taj postupak ćemo ponavljati sve dok pokazivač `j` ne dosegne element s vrijednošću `2` koji ima indeks `5`. Kako je `8` veće od `2` zamijenit će se vrijednosti tih dvaju elemenata. Sada naš ključni element ima vrijednost `2`, a pokazivač `j` će nastaviti pokazivati na sljedeće vrijednosti sve do posljednjeg elementa liste. Sljedeća zamjena dogodit će se kada pokazivač `j` dosegne vrijednost `1`. Nakon što pokazivač `j` dođe do kraja liste, čineći pritom devet usporedbi, na prvom se mjestu nalazi element s najmanjom vrijednošću. Preostaje nam sortirati ostatak liste u kojoj je ostalo devet elemenata. To ćemo učiniti tako da pokazivač `i` postavimo na prvi element preostale liste (`i = 1`). Slijedi osam usporedbi da bismo pronašli najmanji element u listi i postavili ga na to drugo mjesto. Postupak nastavljamo sve dok kazaljka `i` ne dođe na pretposljednje, a `j` na posljednje mjesto liste te će nakon toga lista biti uzlazno sortirana." (vidi [15])



Slika 4.1: Sortiranje zamjenom: prvi i drugi prolaz (vidi [15])

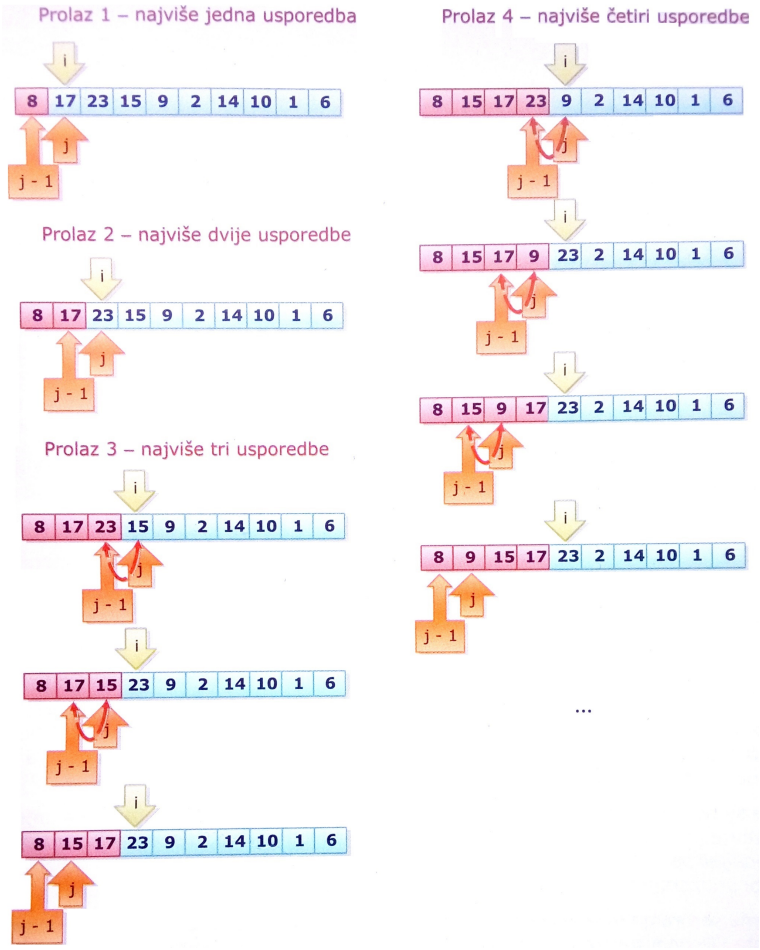
Na osnovi ovog opisa napišimo funkciju za sortiranje zamjenom elemenata:

```
def sort_zamjenom(lista):
    for i in range(len(lista) - 1):
        for j in range(i + 1, len(lista)):
            if lista[i] > lista[j]:
                lista[i], lista[j] = lista[j], lista[i]
    return lista
```

Sljedeći algoritam za sortiranje je *selection sort* no njega smo već vidjeli u prethodnom udžbeniku. Još je prokomentirano da je razlika od algoritma *exchange sort* što se kod algoritma *selection sort* ne obavlja zamjena kada se u koraku pojavi manji element od ključnog,

već se samo pamti indeks najmanjeg elementa u tom prolazu i zamjena se radi tek na kraju tog prolaza.

Sljedeće na red dolazi sortiranje umetanjem (***insertion sort***):
"Sortiranje umetanjem svodi se na postupno stvaranje dviju lista. u prvom (lijevom) dijelu liste nalaze se sortirani elementi liste, dok su u drugom dijelu nesortirani elementi liste. Na početku se sortirani dio sastoji samo od prvog elementa. Kao i kod dosadašnjih načina sortiranja, prolazit ćemo kroz elemente liste i u svakom prolazu uzeti samo jedan, i to prvi element iz nesortiranog dijela liste te ga pomicati u lijevo dok ne dođe na pravo mjesto u sortirani dio liste (slika 4.2). Indeks `i` *pokazuje na prvi element nesortirane liste.*



Slika 4.2: Sortiranje umetanjem: prva četiri prolaza (vidi [15])

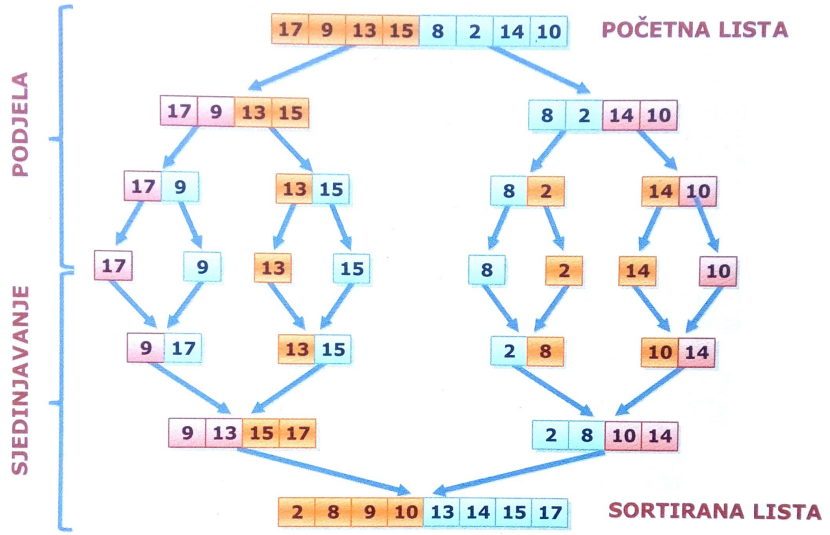
Indeks `j` *na početku svakog prolaza postavlja se na prvi element nesortirane liste koji pos-*

taje ključni element tog prolaza. Ako je njegova vrijednost manja od elementa lijevo od njega, oni trebaju zamijeniti mjesta. Takva se zamjena ključnog elementa s njegovim lijevim susjedom nastavlja sve dok ne naiđe na manju vrijednost elementa ili ne dosegne lijevi kraj liste (`j = 0`). Taj algoritam ostvaren je sljedećom funkcijom:" (vidi [15])

```
def sort_umetanjem(lista):
    for i in range(1, len(lista)):
        j=i
        while j > 0 and lista[j] < lista[j - 1]:
            lista[j - 1], lista[j] = lista[j], lista[j - 1]
            j -= 1
    return lista
```

Još je iskomentirana složenost algoritma. Algoritam će u najboljem slučaju imati linearnu a u najgorem kvadratnu složenost tj. složenost algoritma je $O(n^2)$.

Sljedeći algoritam za sortiranje koji se pojavljuje je sortiranje sjedinjavanjem (engl. ***merge sort***). U udžbeniku se ovaj algoritam uvodi rješavanjem primjera tj. zadatka gdje treba napisati funkciju čiji će parametri biti dvije sortirane liste, a oni će vraćati sortiranu listu koja se dobiva spajanjem dviju zadanih lista. Taj nam je primjer potreban jer ovaj algoritam rekurzivno spaja (sortira) liste koje dobije rastavljanjem početne liste na sve manje podliste:



Slika 4.3: Sortiranje sjedinjavanjem (vidi [15])

"Kod sortiranja sjedinjavanjem dijelit ćemo osnovnu listu na sve manje liste sve dok se one ne svedu na liste s jednim članom a potom ćemo te liste spajati u sortirane liste. Postupak je ilustriran slikom 4.3. Zbog preglednijeg prikaza odabrana je lista s 8 elemenata.

```
def sor_merge(lista):
    if len(lista) == 1:
        return lista            #1
    m = len(lista) // 2         #2
    lijevo = lista[:m]          #3
    desno = lista[m:]           #4
    sort_merge(lijevo)          #5
    sort_merge(desno)           #6
    i, j = 0, 0
    for k in range(len(lijevo) + len(desno)):
        if i < len(lijevo) and j < len(desno):
            if lijevo[i] < desno [j]:
                lista[k] = lijevo[i]
                i += 1
            else:
                lista[k] = desno[j]
                j += 1
        elif i < len(lijevo):
            lista[k] = lijevo[i]
            i += 1
        else:
            lista[k] = desno[j]
            j += 1
    return lista
```

Postupak podjele na podliste je jednostavan - svodi se na igru s indeksima. Rekurzivna funkcija objedinjuje podjelu i sjedinjavanje generiranih podlista.
*Kao što vidimo iz programa, funkcija neće obavljati ništa kada ulazna lista ima duljinu jedan, tj. kada se sastoji samo od jednog elementa (**#1**). Ako je duljina veća od jedan, lista će biti podijeljena na dva dijela: `lijevu` i `desnu` podlistu (**#2, #3**) na koje će se primijeniti rekurzivna funkcija sortiranja (**#5, #6**). Nakon toga slijedi sjedinjavanje sortiranih podlista (od **#7** do **#8**).*
U postupku se najprije u fazi podjele lista dijeli na podliste jednakih duljina. Podjela će svesti početnu listu na osam podlista duljine jedan. Nakon toga slijedi faza sjedinjavanja." (vidi [15])

Još je komentirano da je vrijeme izvođenja algoritma u svakom koraku sjedinjavanja $O(n)$ i da će takvih podjela biti $\log n$, pa je stoga složenost cijelog algoritma $O(n \log n)$.

Vidimo da su algoritmi bolje opisani i predočeni nego u prethodnom udžbeniku. Za svaki od algoritama imamo i ilustraciju slikom te podosta opširan opis samog algoritma te njegovog načina rada za razliku od prethodnog udžbenika.
Donosimo još implementaciju u Pythonu i kratak opis algoritama *quick sort* te *bucket sort*:

Quick sort:
"Algoritam brzog sortiranja također je rekurzivan kao i sortiranje sjedinjavanjem, ali se početna lista ne dijeli na jednako velike podliste, već se događa sljedeće:

- *na proizvoljni način odabire se jedan od elemenata za ključni element* `ključni_el`
- *kreiraju se dvije podliste: podlista onih elemenata koji su manji ili jednaki ključnom elementu (ta podliste uključuje i* `ključni_el`*) te podlista onih elemenata koji su veći od ključnog elementa*
- *te se dvije podliste zatim rekurzivno sortiraju.*

Djelovanje algoritma ilustrirano je slikom 4.4. U prvom koraku algoritma možemo u načelu odabrati bilo koji element za ključni element `ključni_el`*. Ovdje ćemo zbog jednostavnijeg slikovnog prikaza odabrati uvijek srednji element. Na našoj slici to je u 1. razini rekurzije element s s vrijednošću* `8`*.*

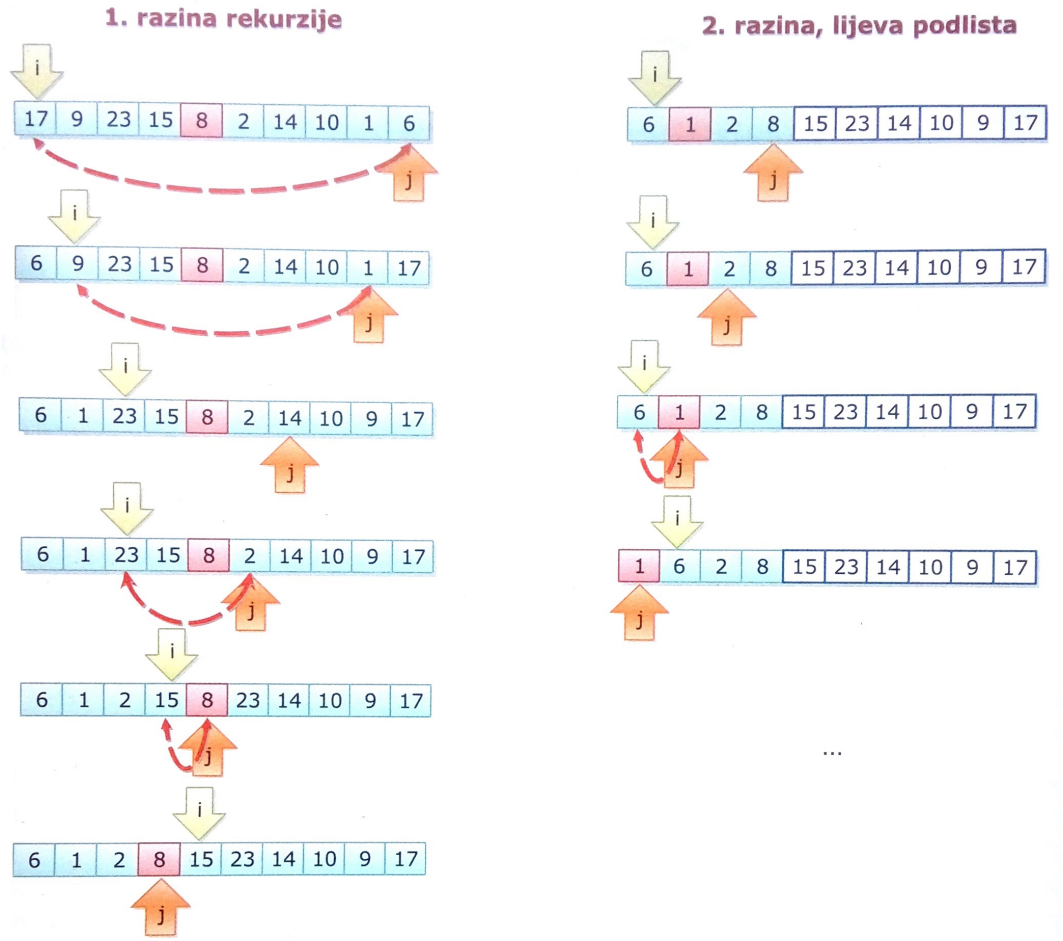
Nakon toga se postavljaju dva pokazivača:

- *prvi pokazivač* `i`*, koji se od početka liste kreće udesno - tražit ćemo prvi element koji je veći ili jednak ključnom*
- *drugi pokazivač* `j` *koji se od kraja čiste kreće ulijevo - tražit ćemo element koji je manji ili jednak ključnom.*

Kada lijevo i desno od ključnog elementa nađemo takva dva elementa, zamjenjujemo im mjesta i nastavljamo traženje.

Na slici 4.4 možemo uočiti da smo time dobili dvije tražene podliste. Prva sadrži elemente s vrijednostima manjim ili jednakim od ključnog elementa, a druga elemente s vrijednostima većim od vrijednosti ključnog elementa.

Te se dvije liste mogu podvrgnuti jednakom postupku koji možemo pratiti na temelju prikaza na slici 4.4." (vidi [15])



Slika 4.4: Algoritam brzog sortiranja (vidi [15])

Opisani postupak preveden u programski oblik prikazan je sljedećom funkcijom:

```
def sort_quick(lista, ind_p, ind_k):
    sred = lista[(ind_p + ind_k) // 2]
    i = ind_p
    j = ind_k
    while i <= j:
        while lista[i] < sred:
        i += 1
        while lista[j] > sred:
```

```
            j -= 1
        if i <= j:
            lista[i], lista[j] = lista[j], lista[i]
            i += 1
            j -= 1
    if ind_p < j:
        sort_quick(lista, ind_p, j)
    if ind_k > i:
        sort_quick(lista, i, ind_k)
    return lista
```

Još je izvedeno kako opisani algoritam ima vremensku složenost u najgorem slučaju $O(n^2)$, a u najboljem slučaju $O(n \log n)$. Ima veliku prednost jer se cijeli postupak obavlja na jednoj listi što je veoma važno pri sortiranju velikih lista.

Bucket sort:
Bucket sort je algoritam koji listu pretvara u rječnik kako bi se vrijeme sortiranja smanjilo. U rječniku je traženje elementa složenosti $O(1)$ pa bi to smanjilo cijelu složenost algoritma. Prikazani su problemi pretvorbe liste u rječnik jer kada ubacimo više istih vrijednosti u rječnik, u rječniku ostaje samo jednom zapisana vrijednost.
Tako će u rječniku svaka vrijednost iz liste biti ključ koji će imati vrijednost onoliku koliko se puta element nalazio u listi. Na kraju rječnik treba pretvoriti natrag u listu.
Implementacija u Pythonu:

```
def sort_u_pretince(lista):
    rjecnik = {}
    for t in lista:
        rjecnik[t] = rjecnik.get(t, 0) + 1
    sortirano = []
    for k in range(min(rjecnik), max(rjecnik) + 1):
        if k in rjecnik:
            sortirano.extend([k] * rjecnik[k])
    return sortirano
```

U udžbeniku je još komentirano kako je složenost algoritma linearna tj. $O(n)$.

Algoritmi za pretraživanje

Od algoritama za pretraživanje uveden je prvo učeniku intuitivan način pretraživanja. Želimo li znati postoji li neki element u listi, pretražujemo od prvog do zadnjeg člana te svaki od njih uspoređujemo s traženim elementom. Takav način pretraživanja nazivamo

linearno pretraživanje. Nakon toga je dan primjer gdje trebamo naći broj u telefonskom imeniku. Tada učenici shvate da neće na isti način pretraživati rječnik, tj da neće uspoređivati sa svakim imenom od početka dok ne nađu odgovarajuće traženo ime. U udžbeniku piše sljedeće:
"Napravimo jednu usporedbu sa stvarnim životom: Zamislite listu `L` *kao telefonski imenik u kojem se prezimena ne nalaze navedena po abecedi, nego nekim proizvoljnim redoslijedom. Kako biste pronašli telefonski broj neke osobe x? Nema druge opcije nego krenuti redom, od prvoga imena na prvoj stranici imenika do posljednjega imena na posljednjoj stranici: morali bismo uspoređivati ime tražene osobe sa svakim imenom u telefonskome rječniku.*
Srećom, telefonski imenici (kao i oni razredni) sortirani su po prezimenima i to nam jako olakšava i ubrzava traženje. Otvorimo li imenik u sredini, moguće su tri situacije.

1. *Imali smo sreće i otvorili smo imenik točno na imenu koje tražimo. Pretraga je završena!*

2. *Ime koje tražimo abecedno je manje od imena koje smo otvorili. To znači da se ime koje tražimo nalazi lijevo od toga mjesta. Zato možemo ponovno pretraživati na isti način (raspolavljanjem) u lijevoj polovici imenika.*

3. *Ime koje tražimo abecedno je veće od imena koje smo otvorili. To znači da se ime koje tražimo nalazi desno od toga mjesta. Zato možemo ponovno pretraživati na isti način (raspolavljanjem) u lijevoj polovici imenika."* (vidi [5])

Nakon što je naveden primjer s jednom listom na kojem je prikazan postupak binarnog traženja imamo i implementaciju u Pythonu:

```
d = 0; g = len(L)-1; indeks = -1
while d <= g:
   s=(d+g)//2
   if x == L[s]:
      indeks = s
      break
   elif x < L[s]:
      g=s-1
   elif x > l[S]:
      d = s+1
if indeks == -1:
   print(x, 'se ne nalazi u listi L.')
else:
   print(x, 'se ne nalazi na indeksu', indeks, 'u listi L.')
```

*"Takav način pretraživanja zovemo **binarno pretraživanje**. Ono funkcionira samo u sortiranim listama! Binarno pretraživanje puno je brže od linearnoga jer ima bitno manji broj uspoređivanja. Ako lista* `L` *ima N elemenata, onda linearno pretraživanje u najgoremu slučaju treba N usporedbi. Može se pokazati da binarno pretraživanje treba manje od* $log_2(N) + 1$ *usporedbi (isprobajte sami!). Na primjer, ako lista ima milijun elemenata, onda binarno uspoređivanje treba samo* $log_2(1000000) + 1 \approx 21$ *usporedbi elemenata liste* `L` *s traženim x."* (vidi [5])
Mislimo da su algoritmi dobro opisani te da je prije samog opisa dana primjerena motivacija.

Hornerov algoritam

U udžbeniku *RJEŠAVANJE PROBLEMA PROGRAMIRANJEM U PYTHONU* imamo opisan Hornerov algoritam. Hornerov algoritam služi nam za računanje vrijednosti polinoma u nekoj točki.
Kada imamo zadatak napisati funkciju koja računa vrijednost polinoma u točki x_0, gdje su nam zadani koeficijenti polinoma u listi p te točka x_0, vrlo intuitivno rješenje je sljedeća implementacija:

```
def vrijednost_polinoma(p, x0):
    v = 0
    for i in range(len(p)):
        v = v + p[i] * x0 ** i
    return v
```

No, ovom implementacijom i nismo baš zadovoljni. U udžbeniku je opisano i zašto:

"Općenito, kod računanja vrijednosti polinoma n-tog stupnja u točki x_0 *imamo ukupno:* $n + (n-1) + (n-2) + ... + 1 = \frac{n(n-1)}{2}$ *množenja i n zbrajanja. Međutim, zapišemo li polinom malo drugačije, broj množenja mogao bi se uvelike smanjiti. Promotrimo polinom:* $p(x) = 3x^3 + x^2 + 2x + 4$. *Ovaj polinom bismo mogli zapisati drugačije da iz prva tri člana 'izlučimo' x:* $p(x) = (3x^2 + x + 2) \cdot x + 4$ *te još jednom iz prva dva člana 'izlučimo' x:* $p(x) = ((3 \cdot x + 1) \cdot x + 2) \cdot x + 4$. *Prebrojimo li ovdje operacije množenja, ustanovit ćemo da ih je ukupno tri i isto toliko je operacija zbrajanja. Općenito, polinom n-tog stupnja možemo raspisati kao:* $p(x) = (...((a_n x + a_{n-1}) \cdot x + a_{n-2}) \cdot x + ... + a_1) \cdot x + a_0$ *te imamo ukupno n množenja i n zbrajanja. Uzmemo li u obzir da je množenje "složena" operacija, dolazimo do zaključka da smo ovime dosta "pojednostavnili" izvođenje algoritma. Ovakav način računanja vrijednosti polinoma u točki nazivamo **Hornerov algoritam**.*

Preostaje nam još da implementiramo ovaj algoritam u Pythonu. Dakle, zadana je lista koeficijenata polinoma n-tog stupnja: `p=[`a_0, a_1, ... , a_{n-1}, a_n`]` *te točka* x_0. *Za naš polinom u prilagođenom obliku zadana je lista koeficijenata:* `p=[4, 2, 1, 3]` *i točka* x_0`=3`. *Ilustrirajmo postupak računanja vrijednosti polinoma u točki.*

- *Računanje započinjemo vrijednošću* 3 *- zadnji element liste.*
- *U sljedećem koraku vrijednost* 3 *množimo sa* $x_0 = 3$ *i dodajemo sljedeći element liste:* $3 \cdot 3 + 1 = 10$.
- *U sljedećem koraku prethodnu vrijednost množimo sa* $x_0 = 3$ *i dodajemo sljedeći element liste:* $(3 \cdot 3 + 1) \cdot 3 + 2 = 10 \cdot 3 + 2 = 32$.
- *U zadnjem koraku prethodni rezultat množimo s 3 i dodajemo sljedeći element liste:* $32 \cdot 3 + 4 = 100$.

Na osnovi prethodnih razmatranja vidimo da računanje započinjemo prethodnom vrijednošću $v = a_n$. *U sljedećem koraku dobivamo:* $v = v \cdot x + a_{n-1}$ *te postupak nastavljamo do* $v = v \cdot x + a_0$. *Budući da računanje započinjemo sa zadnjim elementom liste koeficijenata, bilo bi zgodno listu okrenuti te u tom slučaju zapis ovog algoritma u Pythonu ima sljedeći oblik:"* (vidi [14])

```
def Horner(p, x0):
    v = 0
    p = p[::-1]
    for i in range(len(p)):
        v = v * x0 + p[i]
    return v
```

Euklidov algoritam

U istom udžbeniku možemo pronaći i **Euklidov algoritam**.
"Jedan od češćih problema u numeričkoj matematici jest računanje ***najvećeg zajedničkog djelitelja dvaju prirodnih brojeva***. *Najveći zajednički djelitelj (****mjera****) dvaju prirodnih brojeva* `a` *i* `b`, *u oznaci* `nzd(a, b)` *jest najveći broj koji dijeli brojeve* `a` *i* `b`. *Primjerice:* `nzd(6, 9)=3`, `nzd(18, 24)=6`, `nzd(7, 15)=1` *i sl. Za brojeve čija je mjera* 1 *reći ćemo da su* ***relativno prosti***. *Jedan od načina računanja najvećeg zajedničkog djelitelja dvaju brojeva je tzv.* ***Euklidov algoritam***. *Prvo ćemo promotriti malo modificiranu inačicu ovog algoritma koja kaže:*

1. *Ako su dva broja jednaka, onda je njihov najveći zajednički djelitelj upravo taj broj, primjerice* `nzd(6, 6)=6`.

2. *Ako brojevi nisu jednaki, onda je njihov zajednički djelitelj jednak najvećem zajedničkom djelitelju brojeva prema principu: veći broj - manji broj i manji broj. Primjerice,* `nzd(18, 24)=nzd(18, 6)` *- naime, brojevi* `18` *i* `24` *očito nisu jednaki pa je njihov zajednički djelitelj jednak najvećem zajedničkom djelitelju brojeva* `24 - 18 = 6(veći - manji)` *i* `18(manji)`*".* (vidi [14])

```
m = int(input('Prvi broj: '))
n = int(input('Drugi broj: '))
a, b = m, n
while m != n:
   if m > n:
      m -= n
   else:
      n -= m
print('nzd brojeva {0} i {1} je {2}'.format(a, b, n))
```

"Kao što smo naveli, inačica Euklidova algoritma malo je modificirana. Sljedeća inačica je:

1. *Ako je jedan od brojeva jednak nuli, onda je nzd (najveći zajednički djelitelj) drugi broj, primjerice* `nzd(6, 0)=0`*.*

2. *Ako brojevi nisu jednaki, onda je njihov nzd isti kao nzd brojeva: veći* `%` *manji i manji. Primjerice,* `nzd(18, 42)=nzd(18, 6)` *- naime, brojevi* `18` *i* `42` *očito nisu jednaki pa je njihov nzd jednak nzd brojeva* `42 % 18 = 6(veći % manji)` *i* `18(manji)`*".*

Osim promjene naredbe unutar tijela petlje, ovdje ćemo trebati promijeniti i uvjet izvođenja petlje. Naime, petlja se ne prestaje izvoditi kada brojevi postanu jednaki, već kada jedan od brojeva postane 0. Dakle, petlju ćemo izvoditi dok su oba broja veća od 0. Uvjet kojim ćemo to provjeriti za brojeve `m` *i* `n` *je sljedeći:* `m != 0` **`and`** `n != 0`*. Nakon petlje trebamo još provjeriti koji od brojeva nije jednak 0 te taj broj ispisati:*

```
m = int(input('Prvi broj: '))
n = int(input('Drugi broj: '))
a, b = m, n
while m != 0 and n != 0:
   if m > n:
      m %= n
   else:
      n %= m
if m > 0:
```

```
    print('Mjera brojeva {0} i {1} je {2}'.format(a, b, m))
else:
    print('Mjera brojeva {0} i {1} je {2}'.format(a, b, n))
```

Uvjet: `m != 0 and n != 0` *mogli smo zapisati i nešto "elegantnije". Naime, ako je jedan od brojeva jednak 0, onda je njihov umnožak jednak 0. Dakle, oba će broja biti različita od 0 ako je njihov umnožak različit od 0.*
Sličnu dosjetku možemo napraviti i kod ispisa. Budući da je jedan od brojeva jednak 0, onda je zbroj tih dvaju brojeva jednak broju koji nije 0, što je upravo onaj broj koji tražimo:" (vidi [14])

```
m = int(input('Prvi broj: '))
n = int(input('Drugi broj: '))
a, b = m, n
while m * n != 0:
    if m > n:
        m %= n
    else:
        n %= m
print('Mjera brojeva {0} i {1} je {2}'.format(a, b, m + n))
```

Rekurzivni algoritmi

U udžbeniku *NAPREDNO RJEŠAVANJE PROBLEMA PROGRAMIRANJEM U PYTHONU* je pomnije opisan postupak izgradnje rekurzivnog algoritma. Opisani su pojmovi kao što je *rekurzivni pristup* i *rekurzivna relacija* te se govori o uvjetima prekida rekurzivnog algoritma. Od rekurzivnih algoritama prikazali smo već neke algoritme za sortiranje i traženje te su u knjizi još navedeni neki primjeri rekurzivnih algoritama kojima računamo umnožak prvih n prirodnih brojeva, Fibonaccijeve brojeve te druge jednostavne algoritme koji se mogu rekurzivno implementirati kao što je algoritam za prebacivanje tornja od n diskova sa štapa X na štap Y uporabom pomoćnog štapa Z (Hanojski tornjevi), kombinatorni algoritmi i slično.
Moramo napomenuti da su u posebnoj cjelini rekurzivni algoritmi jedino opisani u ovom udžbeniku kojeg primjenjuje mali broj prirodoslovno-matematičkih gimnazija u praksi jer je gradivo napredno i potrebno je određeno predznanje kako bi se mogao uspješno savladati sadržaj.

Zaključak

Svi algoritmi opisani u ovom poglavlju su metodički dobro obrađeni. Nemamo primjedbi na implementaciju.
Naglašavamo da nije analizirana složenost nekih algoritama (Euklidov).
Smatramo da je sintagma ”teorija brojeva” primjerenija od ”numerička matematika” (str. 35).

Bibliografija

[1] *CSTA K-12 Computer Science Standards*, http://csta.hosting.acm.org/csta/csta/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf, posjećena 2018-31-01.

[2] *HNOS - Hrvastki nacionalno obrazovni standard*, https://www.kif.unizg.hr/_download/repository/Nastavni_plan_i_program_za_OS_%28HNOS%29%5B1%5D.pdf, posjećena 2018-31-01.

[3] *Nacionalni kurikulum nastavnog predmeta informatika (PRIJEDLOG)*, http://mzos.hr/datoteke/15-Predmetni_kurikulum-Informatika.pdf, posjećena 2018-31-01.

[4] *NOK - Nacionalni okvirni kurikulum*, http://www.azoo.hr/images/stories/dokumenti/Nacionalni_okvirni_kurikulum.pdf, posjećena 2018-31-01.

[5] N. Dmitrović, S. Grabusin, Z. Bujanović, *INFORMATIKA 2, udžbenik iz informatike za 2. razred prirodoslovno-matematičkih gimnazija*, SysPrint, Zagreb, 2014.

[6] M. Babić, F. Glavan, M. Stančić, B. Vejnović, Z. Dimovski, *Moj PORTAL 3.0 7 udžbenik informatike u sedmom razredu osnovne škole*, Školska knjiga, Zagreb, 2014.

[7] M. Babić, F. Glavan, M. Stančić, S. Leko, B. Vejnović, Z. Dimovski, *Moj PORTAL 3.0 8 udžbenik informatike u osmom razredu osnovne škole*, Školska knjiga, Zagreb, 2014.

[8] M. Stančić, B. Vejnović, Z. Dimovski, *Moj PORTAL 6 udžbenik informatike za 6. razred osnovne škole*, Školska knjiga, Zagreb, 2011.

[9] _____, *Moj PORTAL 5 udžbenik informatike za 5. razred osnovne škole*, Školska knjiga, Zagreb, 2013.

[10] S. Svetličić, L. Kralj, N. Hajdinjak, D. Rakić, B. Floriani, *NIMBUS OBLAK 5, udžbenik informatike s e-podrškom za peti razred osnovne škole*, Profil, Zagreb, 2014.

[11] _____, *NIMBUS OBLAK 6, udžbenik informatike s e-podrškom za šesti razred osnovne škole*, Profil, Zagreb, 2014.

[12] _____, *NIMBUS OBLAK 7, udžbenik informatike s e-podrškom za sedmi razred osnovne škole*, Profil, Zagreb, 2014.

[13] _____, *NIMBUS OBLAK 8, udžbenik informatike s e-podrškom za osmi razred osnovne škole*, Profil, Zagreb, 2014.

[14] P. Brođanac, L. Budin, Z. Markučič, S. Perić, *RJEŠAVANJE PROBLEMA PROGRAMIRANJEM U PYTHONU*, Element, Zagreb, 2012.

[15] _____, *NAPREDNO RJEŠAVANJE PROBLEMA PROGRAMIRANJEM U PYTHONU*, Element, Zagreb, 2014.

[16] _____, *INFORMATIKA 1, udžbenik za prvi razred prirodoslovno matematičkih i općih gimnazija te drugi razred klasičnih i jezičnih gimnazija*, Školska knjiga, Zagreb, 2015.

[17] V. Galešev, I. Kniewald, G. Sokol, B. Bedenik, K. Repek, *Informatika+ 5, udžbenik iz informatike za 5. razred osnovne škole*, SysPrint, Zagreb, 2014.

[18] _____, *Informatika+ 6, udžbenik iz informatike za 6. razred osnovne škole*, SysPrint, Zagreb, 2014.

[19] _____, *Informatika+ 7, udžbenik iz informatike za 7. razred osnovne škole*, SysPrint, Zagreb, 2014.

[20] _____, *Informatika+ 8, udžbenik iz informatike za 8. razred osnovne škole*, SysPrint, Zagreb, 2014.

[21] S. Singer, *Složenost algoritama*, `https://web.math.pmf.unizg.hr/~singer/oaa/skripta/00.pdf`, posjećena 2018-31-01.

Sažetak

U ovom radu smo opisali koliko se pozornosti pridaje algoritamskom mišljenju u važećim dokumentima, prikazali koji poznati algoritmi se obrađuju u gimnazijama te kritički komentirali trenutno stanje.

Kroz prvo poglavlje smo proučili samu bit algoritma. Osvrnuli smo se na definiciju algoritma, a onda i na svojstva koja algoritam mora zadovoljavati. Objasnili smo koje osobine mora imati "dobar" algoritam te načine na koje se algoritam može predstaviti. Bili su prikazani i primjeri algoritma kako u stvarnom svijetu tako i u matematici i informatici.

Drugo poglavlje govorilo je o tome u kojem kontekstu se sve algoritamsko mišljenje spominje u *Hrvatskom nacionalnom obrazovnom standardu (HNOS-u)*, *Nacionalnom okvirnom kurikulumu za predškolski odgoj i obrazovanje te opće obvezno i srednjoškolsko obrazovanje (NOK-u)*, a onda i u standardu računalne znanosti *(CSTA K-12 Computer Science Standard)*. Usporedili smo navedene dokumente s trenutnim prijedlogom novog kurikuluma u Hrvatskoj i vidjeli da trenutni prijedlog dobro korespondira s aktualnim dokumentima u svijetu.

U trećem poglavlju imali smo kratak pregled algoritama koji se spominju u osnovnoj školi kako bi imali bolji uvid predznanja učenika u samom području algoritama i algoritamskog mišljenja prije srednje škole.

Četvrto poglavlje nadovezalo se na treće. Ovdje ćemo prošli algoritme koji se obrađuju u srednjim školama, preciznije gimnazijama, kao što i sam naslov rada govori. Naveli smo neke loše ali i dobre strane trenutnog stanja obrade algoritama i korištenja algoritamskog mišljenja u gimnazijama kao važnog dijela odgojno-obrazovnog procesa.

Summary

This thesis describes the amount of attention given to algorithmic thinking in valid documents, shows which known algorithms are being processed in gymnasium and offers a critical commentary on the current state.

The first chapter deals with the very essence of an algorithm and defines the algorithm itself as well as the properties it has to satisfy. It also offers an explanation of what attributes a ”good” algorithm should have and how it can be presented. Finally, it offers examples not only of real world algorithms but also of those in mathematics and computing.

The second chapter refers to the contexts in which algorithmic thinking is mentioned in the *Croatian National Educational Standard (HNOS)*, in *The textbook of the National Framework Curriculum for Pre-school Education and General Compulsory and Secondary Education (NOK)*, as well as in the *Computer Science Standard CSTA K-12*. It offers a comparison of the aforementioned documents with the current proposal of the new Croatian curriculum, and brings to the conclusion that the latter corresponds well to the current documents of the sort in the world.

The third chapter offers a brief overview of the algorithms mentioned in elementary schools with the aim of providing a better insight into the knowledge of algorithms and the level of algorithmic thinking present before high school.

The fourth chapter is an extension of the previous one. It analyses algorithms studied in middle schools, i.e. in gymnasiums, as the title of the thesis itself suggests. It lists some positive and some negative aspects of the current state of processing algorithms and of the use of algorithmic thinking in gymnasiums as a very important part of the educational process.

Životopis

Zovem se Ivan Milišić. Rođen sam 15. studenog 1992. godine u Gornjem Vakufu (Bosna i Hercegovina). Zbog ratnih neprilika sam proveo četiri godine djetinjstva u Austriji. Godine 1997. vraćam se s obitelji u Gornji Vakuf-Uskoplje, gdje sam 2007. godine završio Osnovnu školu "Uskoplje". Iste godine upisujem Opću gimnaziju "Uskoplje" u Uskoplju, koju sam završio 2011. godine. Nakon toga se upisujem na Matematički odsjek Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu, gdje 2015. godine završavam preddiplomski sveučilišni studij Matematika; smjer: nastavnički. Završetkom tog studija 2015. godine, na istom fakultetu upisujem diplomski sveučilišni studij Matematika i informatika; smjer: nastavnički.