Pohlepni algoritmi

Lemajić, Tamara

Undergraduate thesis / Završni rad

2014

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Humanities and Social Sciences / Sveučilište Josipa Jurja Strossmayera u Osijeku, Filozofski fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:142:565707

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

FILOZOFSKI FAKULTET
SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

Repository / Repozitorij:

FFOS-repository - Repository of the Faculty of Humanities and Social Sciences Osijek

zir.nsk.hr

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište J.J. Strossmayera u Osijeku

Filozofski fakultet

Preddiplomski studij Informatologije

Tamara Lemajić

Pohlepni algoritmi

Završni rad

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

Osijek, 2014.

Sažetak

Rad se bavi općenito algoritmima s posebnim naglaskom na pohlepne algoritme. Ukratko se predstavlja povijest nastanka algoritma. Naime, sam pojam algoritma nastao je u čast arapskom matematičaru al Khowarizmiju koji je vjerovao kako se svaki matematički problem može rastaviti na manje dijelove i na taj način ubrzati njegovo rješavanje. Upravo ta ideja dovodi do onoga što se danas naziva algoritmom. Također, navodi se kada i gdje se pronalaze prvi zapisi algoritama. Da bi se precizno izrazili koraci od kojih se sastoji algoritam mogu se koristiti prirodan jezik, pseudo jezik te pravi programski jezik. Rad donosi i klasifikaciju algoritama prema načinu implementacije na rekurzivne ili iterativne algoritme, serijske ili paralelne, determinističke ili stohastičke te na točne ili približne algoritme. Algoritme je, također, moguće podijeliti prema metodologiji dizajna. Prema metodologiji dizajna algoritmi se dijele na algoritme „na silu“, „podijeli i vladaj“ algoritme, dinamičke, pohlepne te na algoritme za sortiranje i prebrojavanje. Tehnikom pohlepnog pristupa, rješenje zadanog algoritma se konstruira u nizu koraka. U svakom se koraku bira mogućnost koja je lokalno optimalna u nekom smislu. Zamisao je da će nas takvi optimalni koraci dovesti do globalnog optimalnog rješenja. No, za mnoge probleme pohlepni algoritam ne uspije proizvesti optimalno rješenje.

Ključne riječi: algoritam, pohlepni algoritam, vremenska složenost algoritma

Sadržaj

Sažetak 1
1. Uvod 3
2. Povijest nastanka algoritma 4
3. Određenje pojma algoritma 4
3.1. Zapis algoritama 6
3.2. Klasifikacija algoritama 7
3.3. Analiza i složenost algoritma 9
4. Pohlepni algoritam 9
4.1. Specifičnosti pohlepnih algoritama 12
4.2. Čisti i ortogonalni pohlepni algoritam 13
4.3. Primjeri korištenja pohlepnog algoritma 13
4.4. Slučajevi neuspjeha 16
5. Zaključak 17
Literatura 18

1. Uvod

U radu se opisuju algoritmi s posebnim naglaskom na pohlepne algoritme. Drugo poglavlje rada donosi pregled povijesnog razvoja algoritma. Naime, riječ algoritam dolazi od latinskog prijevoda imena arapskog matematičara al Khowarizmija koji se bavio trigonometrijom, astronomijom, zemljopisom, kartografijom, a smatra se ocem algebre jer je definirao osnovna pravila rješavanja linearnih i kvadratnih jednadžbi. Vjerovao je kako se svaki matematički problem može rastaviti na manje cjeline i na taj način ubrzati njegovo rješavanje. Upravo ta ideja je dovela do onoga što se danas naziva algoritam.

U trećem poglavlju opisuje se i objašnjava određenje pojma te analiza i složenost algoritma. Također, treće poglavlje donosi i klasifikaciju algoritama prema načinu implementacije na rekurzivne ili iterativne algoritme, serijske ili paralelne, determinističke ili stohastičke te na točne ili približne algoritme. Algoritme je, također, moguće podijeliti prema metodologiji dizajna. Prema metodologiji dizajna algoritmi se dijele na algoritme „na silu“, „podijeli i vladaj“ algoritme, dinamičke, pohlepne te na algoritme za sortiranje i prebrojavanje. Svaki od navedenih algoritama ima zanimljiva određena svojstva na koja se ukazuje u radu.

Četvrto poglavlje rada detaljnije opisuje pohlepne algoritme, njihovu definiciju te način rada kao i njihove specifičnosti. Nadalje, četvrto poglavlje rada sadrži primjere korištenja pohlepnoga algoritma. Naime, tehnikom pohlepnog pristupa, rješenje zadanog algoritma se konstruira u nizu koraka. U svakom se koraku bira mogućnost koja je lokalno optimalna u nekom smislu. Zamisao je da će nas takvi optimalni koraci dovesti do globalnog optimalnog rješenja. No, za mnoge probleme pohlepni algoritam ne uspije proizvesti optimalno rješenje. Na kraju rada donose se najvažniji zaključci vezani uz pohlepne algoritme.

2. Povijest nastanka algoritma

Riječ algoritam nastala je u čast arapskom matematičaru Muhammed ibn Musa al Khowarizmi iz Bagdada.[1]Sama riječ algoritam nastala je evolucijom upravo njegova imena, Al Khowarizmi. U starim se latinskim knjigama o matematici pronalazi pojam *algorizmi* pokraj svakog pravila, pa se s vremenom taj pojam pretvorio u *algoritam*.[2] Al Khowarizmi se bavio astronomijom, trigonometrijom, zemljopisom, kartografijom, a također se smatra i ocem algebre jer je definirao osnovna pravila rješavanja kvadratnih i linearnih jednadžbi. Njegovi radovi su osnova razvoja mnogih matematičkih i prirodnih disciplina, a također i računarstva.[3] Živio je oko sedamdeset godina i pripada među najcjenjenije matematičare svih vremena. Upravo zahvaljujući njemu služimo se nekim osnovnim pravilima, poput prebacivanja s jedne strane jednakosti na drugu, prilikom rješavanja jednadžbi. Također, Al Khowarizmi je vjerovao kako se svaki matematički problem može rastaviti na manje cjeline i na taj način olakšati i ubrzati njegovo rješavanje. Ta je ideja bila začetnik onoga što se danas naziva algoritam.[4] Na samome početku taj se pojam koristio samo kada je u pitanju bila matematika, no s pojavom računala u 20. stoljeću pojam se proširio na računalstvo i označio presudan korak u razvoju računalnih programa.

Prvi zapis algoritma prilagođen računalu pripada Adi Byron iz 1842., a računao je Bernoullijeve brojeve. Računalo za koje je napisan bio je analitički stroj koji je zamislio, ali nikad u potpunosti proveo u djelo, Charles Babbage. Analitički stroj je trebao biti prvo programibilno računalo, sastavljeno u potpunosti od mehaničkih dijelova. Mehanički dijelovi i fizička glomaznost su glavni razlozi zašto nikad nije završen.[5]

3. Određenje pojma algoritma

Pojam algoritma pripada osnovnim pojmovima matematike. Algoritmom se smatra točan propis o izvršenju nekog sustava operacija za rješavanje svih zadataka određenim redoslijedom nekog zadanog tipa.[6] Ova definicija ne opisuje matematičku definiciju algoritma, već samo razjašnjava njezin smisao. Postoje razne definicije algoritma te se niti za jednu ne može reći da je prava jer sve one na svoj način definiraju algoritam. Jedan primjer definicije je da je algoritam konačan skup preciznih, razumljivih i jednoznačnih instrukcija koje učinkovito i djelotvorno dovode do

[1]Usp. Obradović, M. Diofantske jednadžbe: propozicije-ekspozicije-algoritmi: za učenike srednjih škola. Zagreb : Naklada Ljevak, 2000., str. 20.
[2]Usp. Hoško, T.; Slamić, M. Strukture podataka i algoritmi: Priručnik. Zagreb : Algebra, 2009., str. 8.
[3]Usp. Kiš, M., Englesko-hrvatski i hrvatsko-engleski informatički rječnik, Zagreb, Naklada Ljevak, 2000., str. 36.
[4]Usp. Hoško, T.; Slamić, M. Nav.dj.,str. 8.
[5]Usp. Kiš, M. Nav. dj., str. 36.
[6]Usp. Trahtenbrot, B. A. Što su algoritmi: algoritmi i računski automati. Zagreb: Školska knjiga, 1978., str. 13.

rješenja u konačnom vremenu.[7] Razumijevanje definicije algoritma je ključno u razumijevanju samoga algoritma te je zbog toga potrebno istaknuti neke ključne dijelove definicije algoritma. Dakle, instrukcije algoritma moraju biti jednoznačne, tj. jedna te ista instrukcija bi trebala biti uvijek shvaćena na isti način. Također, moraju biti i precizne zbog samoga kreiranja algoritma. Broj instrukcija nekog algoritma mora biti konačan da bi se uopće mogao zapisati. Nadalje, algoritam mora biti djelotvoran, tj. u konačnici mora dati rezultat. Isto tako mora biti i učinkovit u korištenju raspoloživih resursa poput vremena, memorije, itd. Vrijeme potrebno za izvršavanje algoritma mora biti konačno da bi algoritam imao smisla. Svaki algoritam sadrži još i skup ulaznih podataka te rješenje kao izlazni podatak. Također, algoritam mora dati rješenje za sve moguće ulazne podatke, tj. nije dovoljno da funkcionira samo za neke od mogućih ulaznih podataka.[8] U matematici se niz zadataka određenog tipa smatra riješenim kada je postavljen algoritam za njegovo rješavanje. Na primjer, u algebri su sagrađeni algoritmi koji iz algebarske jednadžbe zadane koeficijentima omogućuju potpuno automatsko određivanje broja njezinih različitih koraka (s pripadnim višestrukostima) kao i njihovo izračunavanje s unaprijed zadanom točnošću. Pronalaženje algoritma prirodni je cilj matematike.[9] Ne samo u matematici već i u drugim raznovrsnim područjima ljudske djelatnosti, susreću se procesi koji teku po strogo određenom formalnom propisu, tj. algoritmu. Na primjer u knjigovodstvu i planiranju analiza ulaznih podataka, njihova obrada i sastavljanje materijalnih bilanca za dobivanje optimalnih rezultata provodi se u drugom lancu elementarnih operacija mnogih tipova u strogoj suglasnosti s posebnim uputama i shemama.[10] Također, klasični primjeri algoritma su i Euklidov algoritam za nalaženje najveće zajedničke mjere dva prirodna broja, što je ujedno i prvi formalni algoritam u povijesti, te primjeri algoritama kao recepata za pripremanje jela i pića.[11]

Konstrukcija algoritma za zadatke određenog tipa, kada je moguća, povezana je općenito sa suptilnim i složenim rasuđivanjem koje zahtjeva visoku stručnost i veliku domišljatost. No, od onog trenutka kada je algoritam već sagrađen, proces rješavanja pripadnih zadataka postaje takav da ga potpuno može izvršiti i osoba koja ništa ne zna o biti samoga zadatka. Samo se zahtjeva da ta osoba može izvršavati elementarne operacije iz kojih se sastoji proces i da se savjesno i točno pridržava postavljenog propisa, tj. algoritma.[12]

[7] Usp. Nav.dj. Hoško, T.; Slamić, M., str. 9.
[8] Isto.
[9] Usp. Trahtenbrot, B. A. Nav. dj., str. 13.
[10] Isto. Str. 14.
[11] Usp. Singer, S. Složenost algoritama. Zagreb, 2005., str. 1.
[12] Isto, str. 14.

3.1. Zapis algoritama

Vezano uz algoritme često se postavlja pitanje zapisa koji se koristi za opis postupaka koji čini određeni algoritam. Da bi se precizno izrazili koraci od kojih se sastoji algoritam mogu se koristiti, po redoslijedu rastuće preciznosti, prirodan jezik, pseudo jezik te pravi programski jezik. No, nažalost, lakoća samoga izražavanja ide u obrnutom redoslijedu.[13] Algoritamski postupak, u većini slučajeva, obuhvaća komplicirane ideje pa je preporučljivo koristiti zapis koji je prirodniji i lakši za razumijevanje nego što je to, na primjer, odgovarajući strojni program za RAM ili čak računalni program na programskom jeziku visoke razine.[14] Zbog toga, jedini preduvjet za zapis algoritma je da specifikacija odgovarajućeg postupka omogućuje precizan niz instrukcija koje računalo treba izvršiti, tj. moguće je koristiti običan jezik za algoritamske korake. Međutim, u složenijim slučajevima potrebno je koristiti programske konstrukcije kako bi se nedvosmisleno izrazili koraci algoritma. Nadalje, potrebno je obratiti pozornost na to da sami pojam algoritma nije isto što i konkretniji pojam računalnog programa. Računalni program se izvršava na računalu te se zbog toga piše na programskom jeziku strogo poštujući pravila toga jezika.[15] Kod zapisa programskoga jezika bitno je svako slovo i svaki simbol. Upravo zbog toga, ako se želi napisani program izvršiti na računalu, ne smije postojati niti jedna greška prilikom pisanja. Za razliku od programskoga jezika, algoritmi ne moraju pratiti tako stroga pravila jer se oni ne izvršavaju na računalima. „Algoritmi se mogu zamisliti da se izvršavaju na jednoj vrsti idealiziranog računala s neograničenom memorijom. Oni su ustvari matematički objekti koji se mogu analizirati da bi se prepoznali suštinski dijelovi složenih problema.“[16] Stoga, vrlo je bitno ne uzimati u obzir nevažne detalje programskoga jezika. Također, još je jedna bitna razlika između algoritama i programskoga jezika. Ta razlika je da kod algoritama nisu bitna pitanja softverskog inženjerstva. Na primjer, da bi se sama bit algoritma prenijela što jasnije i jezgrovitije, obrada grešaka kod algoritama se obično zanemaruje.[17]

Algoritmi su uglavnom male veličine ili se sastoje od kratkih dijelova naredbi prožetih tekstom koji objašnjava glavne ideje ili naglašava ključne točke. Ovo je uglavnom prvi korak za rješavanje problema na računalu budući da sitni, nevažni detalji programskog jezika mogu samo smetati za prikazivanje pravoga problema. No, ovo ne znači da algoritam smije sadržavati bilo

[13]Usp. Živković, D. Uvod u algoritme i strukture podataka. Beograd: Univerzitet Singdinum, 2010., str. 10.
[14] Isto, str. 10.
[15] Isto, str. 11.
[16]Živković, D. Uvod u algoritme i strukture podataka. Beograd: Univerzitet Singdinum, 2010., str. 11.
[17]Usp. Isto.

kakve nedvosmislenosti u vezi s vrstom naredbi koje treba izvršiti ili s njihovim točnim redoslijedom izvršavanja.[18]

3.2. Klasifikacija algoritama

Algoritam je, kao što je već spomenuto, nekakav postupak za dobivanje konačnoga rješenja i sastoji se od niza koraka koji se izvode. Algoritme je moguće klasificirati po raznim kriterijima, a jedan od načina klasifikacije je prema načinu implementacije. Algoritmi se prema načinu implementacije dijele na rekurzivne ili iterativne algoritme, serijske ili paralelne, deterministične ili stohastičke te na točne ili približne algoritme.[19]

Rekurzivni algoritmi su oni algoritmi koji pozivaju sami sebe, primjerice algoritam za pronalazak n-tog člana Fibonnacijevog niza brojeva.[20] Iterativni algoritmi su oni algoritmi koji ne pozivaju sami sebe već se oslanjaju na konstrukte poput petlji (iteracija) i dodatne strukture podataka, kao što je red, da bi riješili problem.[21] Bitno je spomenuti da se svaki rekurzivni algoritam može pretvoriti u iterativni, a svaki iterativni u rekurzivni algoritam.[22] S obzirom na činjenicu da većina današnjih računala sadrži jedan procesor, koriste se serijski algoritmi koji obavljaju naredbe jednu po jednu, te ih na taj način dio po dio rješavaju.[23] Suprotno njima su paralelni algoritmi koji sa sve većim probojem višeprocesorskih računala dobivaju sve veću važnost. Paralelni algoritmi koriste mogućnost višeprocesorskog sustava na taj način da problem podijele na više malih problema koje svaki procesor rješava zasebno te se zatim rezultati spajaju. Paralelni algoritmi uz resurse potrebne za obradu podataka također imaju i malu potrošnju resursa na komunikaciju između više procesora. Algoritmi za sortiranje su jedan od primjera algoritama koje je moguće znatno poboljšati upotrebom paralelnih procesora, no, postoje algoritmi koji bi se primjenom paralelnih metoda samo više zakomplicirali.[24] Nadalje, deterministički algoritmi su oni algoritmi koji će pri svakom izvršavanju u bilo kojim uvjetima od istoga unosa doći do istoga izlaza na način da svaki put slijedi jednak niz naredbi. Za razliku od determinističkih algoritama, stohastički će barem jednom u jednom dijelu izvršavanja donijeti

[18] Isto, str. 11.
[19] Složenost algoritama. URL: https://docs.google.com/document/d/1vwl2dLinUJfcwc8VPpig9FSOaM9yt7QiEhZ05VJ9ijc/edit (2014-07-30)
[20] Isto.
[21] Usp. Kiš, M. Nav. dj., str. 36.
[22] Složenost algoritama. URL: https://docs.google.com/document/d/1vwl2dLinUJfcwc8VPpig9FSOaM9yt7QiEhZ05VJ9ijc/edit (2014-07-30)
[23] Isto.
[24] Usp. Kiš, M. Nav. dj., str. 36.

neku odluku slučajnim odabirom.[25] Iako velika većina algoritama traži točan rezultat, što u većini slučajeva i daju, postoje i programi koji su dizajnirani da pribave približno rješenje jer je točno rješenje nemoguće pronaći.[26]

Algoritme je, također, moguće podijeliti prema metodologiji dizajna. Prema metodologiji dizajna algoritmi se dijele na „brute force“ („na silu“) algoritme, „podijeli i vladaj“ algoritme, dinamičke, pohlepne algoritme te na algoritme za sortiranje i prebrojavanje, tzv. „search and enumeration“.[27] „Brute force“algoritmi „čistom silom“ isprobavaju sve mogućnosti i traže odgovarajuće rješenje. Također, smatraju se najneučinkovitijim algoritmima. Nadalje, kod algoritama „podijeli pa vladaj“, tzv. „divide and conquer“, problem se dijeli na više istih, manjih problema na način da se rješenje polaznog problema može relativno lako konstruirati iz rješenja manjih problema. Odnosno, podjela ide tako dugo dok se ne dođe do malog problema kojeg je jednostavno riješiti, a dobiveni algoritam je rekurzivan jer se svaki od manjih problema i dalje dijeli na manje probleme. Nadalje, jedna od varijacija „podijeli pa vladaj“ algoritama je „smanji pa vladaj“ metoda. Ovdje je riječ o algoritmu u kojem rješenje ne ovisi o svim manjim problemima već samo o jednome. Kao primjer toga navodi se binarno pretraživanje. Pohlepni algoritmi, tzv. „greedy“, su algoritmi koji u svakom koraku biraju lokalno najbolje rješenje, u nadi da će tako iznaći globalni optimum. Ovi algoritmi često ne daju najbolje rješenje već brzu približnu vrijednost najboljeg rješenja. Algoritmi sortiranja služe za brzo sortiranje podataka, npr. niza brojeva.

Svaki od navedenih algoritama ima određena svojstva. Neka od svojstava algoritama su njihova diskretnost, tj. mogućnost da u odvojenim koracima izvode diskretne operacije koje vode prema konačnome cilju. Konačnost je također jedno od bitnih svojstava. Konačnost označava sposobnost algoritma da nakon konačnog broja koraka daje izlazne podatke, odnosno rezultate. Nadalje, bitno svojstvo je determinantnost, tj. kada za iste ulazne podatke algoritam uvijek daje iste rezultate. Također, bitno svojstvo je i masovnost,odnosno da je algoritam primjenjiv na veći broj ulaznih vrijednosti.[28]

[25]Složenost algoritama. URL: https://docs.google.com/document/d/1vwl2dLinUJfcwc8VPpig9FSOaM9yt7QiEhZ05VJ9ijc/edit (2014-07-30)
[26] Isto.
[27]Usp. Kiš, M. Nav. dj., str. 36.
[28] Isto.

3.3. Analiza i složenost algoritma

Analiza algoritma iznimno je važna u programiranju i dizajniranju algoritama. Kako postoji više načina koji će dovesti do istoga rješenja, potrebno je odabrati onaj najučinkovitiji. Kako bi se odredila učinkovitost algoritma najčešće se gleda vremenska i prostorna složenost. „Složenost algoritma je cijena korištenja (izvođenja) tog algoritma za rješavanje jednog od tih problema iz te klase.“[29] Složenost algoritma obično se mjeri u vremenu izvršavanja, potrebnoj memoriji ili nekim sličnim relevantnim pojmovima poput broja procesora za višeprocesorska računala.[30] Složenost algoritma neformalno se može definirati i kao maksimalan broj operacija potrebnih za izvršavanje algoritma. Ovdje je prvo uvedena pretpostavka da su sve osnovne operacije iste složenosti s obzirom da nam je potreban samo njihov broj. S druge strane, broj operacija svakako ovisi o samome ulazu. Upravo iz tog razloga, kada se ispituje složenost algoritma, treba razmatrati „najgori mogući slučaj“. Vremenska složenost je vrijeme koje je potrebno za izvođenje određenog algoritma. Također, ne mjeri se u sekundama kao što je za očekivati, već se mjeri u nekim osnovnim mjernim jedinicama poput aritmetičkih operacija. Nadalje, prostorna složenost predstavlja potrebnu memoriju za izvođenje algoritma. Manje je ograničavajuća jer algoritam istu memorijsku lokaciju može koristiti više puta tijekom izvođenja.[31]

4. Pohlepni algoritam

Pohlepni algoritam je jedan od najjednostavnijih algoritama kombinatorne optimizacije. Sadrži optimalno rješenje problema radeći slijed izbora, tj., za svaku točku odluke u algoritmu odabire se izbor koji je u tome trenutku najbolji.[32] Pohlepna paradigma se često koristi u kombinatornoj teoriji i praksi optimizacije. Ova pojava može biti objašnjena činjenicom da je naširoko pretpostavljeno da pohlepni algoritam rijetko pruža optimalna rješenja, a često pruža neku vrstu približne vrijednosti, odnosno, pruža rješenja koja su znatno bolja od onih lošijih.[33] Pohlepni algoritmi su uglavnom jednostavnog oblika. Koriste se većinom za rješavanje problema optimizacije, kao na primjer nalaženja minimalnog razapinjućeg stabla grafa, nalaženja najkraćeg puta u grafu te nalaženja najboljeg redoslijeda izvođenja zadanih poslova.[34]Apstraktna formulacija takvoga problema optimizacije standardno sadrži sljedeće karakteristične elemente:

---

[29]Singer, S. Nav dj., str. 1.
[30] Isto.
[31] Usp. Ilić, A. Analiza algoritma, str. 2. URL: http://www.pmf.ni.ac.rs/pmf/predmeti/7015/vezbe/slozenost_algoritama.pdf (2014-07-30)
[32]Usp. Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. Introduction to algorithms. The MIT Press, 2001., str. 329.
[33]Usp. Bang-Jensen, J.; Gutin, G.;Yeo, A. When the greedy algorithm fails: Discrete Optimization //Science Direct 1/2, 15(2004), str. 121.
[34]Usp. Singer, S. Nav dj., str. 31.

„Skup C svih mogućih raspoloživih ili dozvoljenih kandidata, na primjer bridovi grafa ili poslovi koje treba obaviti. Nadalje, funkciju koja provjerava daje li određeni skup S izabranih kandidata rješenje zadanog problema ignorirajući pitanje optimalnosti tog rješenja. Drugim riječima, funkcija *rješenje* daje odgovor da ili ne, odnosno ima vrijednosti tipa *boolean.* Na primjer ova funkcija provjerava je li neki skup bridova put između dva zadana vrha u grafu. Također, funkciju cilja koja daje vrijednost svakog rješenja problema. To je funkcija koju optimiziramo, na primjer ukupna duljina razapinjućeg stabla, duljina puta između dva vrha, ukupno vrijeme potrebno za izvođenje poslova u zadanom poretku.“[35] U ovim terminima, vidljivo je da je rješavanje problema optimizacije traženje skupa kandidata koji predstavlja rješenje problema i optimizira vrijednost same funkcije cilja.

Obično se pretpostavlja da polazni problem ima barem jedno rješenje, odnosno, da postoji podskup S ⊆ C koji je rješenje problema. S obzirom na to, problem optimizacije je korektno definiran i, također, ima rješenje. Iz navedenoga, vidljivo je da se rješenje problema svodi na traženje, tj. izbor pravog podskupa S ⊆ C gdje se obično, traženi skup S konstruira element po element, tj. dodavanjem i izbacivanjem kandidata vodeći računa o samoj funkciji cilja. Pohlepni algoritam je pokušaj brzoga nalaženja rješenja, tj. izbora kandidata za traženi skup S. U takvome algoritmu nalaze se također i još neki dodatni karakteristični objekti, tj. skup onih kandidata koje smo već iskoristili, funkcija koja provjerava je li određeni skup kandidata dopustiv, tj. je li moguće taj skup dopuniti tako da se dobije bar jedno rješenje problema koje nije nužno i optimalno, te funkciju izbora (selekcije) koja u svakom trenutku daje najperspektivnijeg, još nekorištenoga kandidata.[36] U navedenim terminima rad pohlepnog algoritma može se opisati na sljedeći način gdje algoritam napreduje korak po korak:

Na samome početku, skup *S* izabranih kandidata je prazan, tj. $S = \emptyset$. U svakom koraku, skupu *S* se pokušava dodati najboljeg preostalog, tj. neiskorištenog kandidata kojega se označuje sa *x*. Prijedlog tog kandidata daje funkcija izbora, tj. *x* je kandidat na kojem se dostiže maksimum vrijednosti *izbor*(*x*). Ako skup $S \cup \{x\}$ do tada izabranih kandidata, zajedno s upravo navedenim, nije više dopustiv, odbacujemo *x*. Tog se odbačenog kandidata nikada više ne provjerava, tj. ne vraća ga se među neiskorištene kandidate, već se prebacuje u odbačene kandidate. Nadalje slijedi da svaki kandidat može biti provjeren samo jednom, što daje brzinu pohlepnom algoritmu. Također, ako je $S \cup \{x\}$ dopustiv, dodaje se kandidat *x* skupu S, tj. on postaje stvarno izabran. Svaki put kada se poveća skup S, provjerava se je li novi S i rješenje

[35]Isto, str. 31.
[36]Isto, str. 32.

problema. U slučaju da jest, stajemo, a u protivnome idemo na novi korak, tj. izbor sljedećeg kandidata *x* ako takav postoji. Nadalje, sasvim općenito, u svakome trenutku skup C svih kandidata je disjunktna unija sljedeća tri podskupa:

C0 = skup neiskorištenih kandidata,
S = skup izabranih (prihvaćenih) kandidata,
D = skup odbačenih kandidata.
Na početku je C0 = C, a S = D = ∅. Funkcija *izbor* je definirana na skupu C0.
U pohlepnom algoritmu skup C0 stalno se smanjuje, a skupovi S i D rastu. Za pojednostavljenje zapisa algoritma skup C0 se pamti u skupu C, a skup D uopće se ne pamti jer se ti kandidati više ne provjeravaju.[37] (Vidi Sliku 1.)

```
procedure greedy ( C : skup ; var S : skup ; var OK : boolean ) ;
        { C je na početku skup svih raspoloživih kandidata. }
        { U skupu S akumuliramo rješenje problema. }
        { OK kaže da li je nađeni S rješenje. }
begin
   S := ∅ ;
   while not rješenje(S) and (C ≠ ∅) do
      begin
      x := element iz C koji maksimizira vrijednost izbor(x) ;
      C := C \ {x} ;
      if dopustiv(S ∪ {x}) then S := S ∪ {x} ;
      end ;
   OK := rješenje(S) ;
end ; { greedy }
```

Slika 1. Prikaz općeg oblika pohlepnog algoritma.[38]

Iz navedenoga vidljivo je da pohlepni algoritam ne mora naći rješenje problema. Čak i ako ga nađe, to rješenje ne mora biti optimalno jer pohlepni algoritam nigdje ne koristi funkciju cilja, već samo funkciju izbora. Da bi rješenje nađeno na ovakav način bilo i optimalno rješenje, potrebno je dokazati da pohlepni algoritam korektno rješava problem optimizacije.[39]

Funkcija izbora najčešće je bazirana na funkciji cilja. Također, navedene funkcije mogu biti i identične. Međutim, ima problema za koje postoji nekoliko mogućnosti za funkciju izbora, a funkcija cilja je, naravno, ista. Ponekad se treba izabrati prava funkcija izbora kako bi pohlepni

[37] Isto.
[38] Isto, str. 33.
[39] Isto, str. 32.

algoritam radio korektno, tj. da bi zaista bio u mogućnosti riješiti problem optimizacije. Osim toga, različitim funkcijama izbora moguće je dobiti različite korektne algoritme za isti problem.[40] „Lako se vidi zašto ove algoritme zovemo pohlepnim. U svakom koraku algoritam uzima najbolji zalogaj koji može progutati, ne vodeći računa o budućnosti (ili prošlosti). On nikada ne mijenja odluku - kad je kandidat jednom prihvaćenu rješenje, on tamo i ostaje (bio dobar ili ne). Također, ako je jednom odbačen, to je zauvijek, nikada više neće biti provjeren.“[41] Upravo zbog navedenog, pohlepni algoritmi su brzi jer se svakog kandidata provjerava najviše jednom, a za uzvrat u nekim problemima se ne dobiva optimalno rješenje.

4.1. Specifičnosti pohlepnih algoritama

Što se tiče specifičnosti pohlepnih algoritama, one mogu biti objašnjene pomoću pet komponenti koje posjeduju pohlepni algoritmi:

1. Skup kandidata, od kojih je stvoreno rješenje;
2. Funkcija odabira koja bira najboljeg kandidata koji će biti dodan rješenju;
3. Funkcija izvedivosti koja se koristi kako bi se utvrdilo može li se kandidata koristiti za doprinose rješenju;
4. Objektivna funkcija koja određuje vrijednost u rješenju ili djelomičnom rješenju;
5. Funkcija rješenja koja će naznačiti kada je otkriveno cjelovito rješenje;[42]

Nadalje, pohlepni algoritmi proizvode dobra rješenja na nekim matematičkim problemima, ali ne i na ostalima. Većina problema na kojima su uspješni imat će dva svojstva: svojstvo pohlepnog izbora i optimalna podstruktura. Što se tiče svojstva pohlepnog izbora, može se napraviti bilo koji izbor koji se čini najboljim u tom trenutku, a zatim rješavati probleme koji se javljaju kasnije. Izbor napravljen od strane pohlepnog algoritma može ovisiti o izborima koji su napravljeni do sada, ali ne i na budućim izborima ili svim rješenjima problema. To neprestano stvara jedan pohlepni izbor za drugim smanjujući svaki zadani problem u jedan još manji. Drugim riječima, pohlepni algoritam nikada ne preispituje svoje odluke. Nadalje, kod optimalne podstrukture problem prikazuje optimalnu podlogu ako optimalno rješenje za problem sadrži optimalna rješenja za podprobleme.[43]

[40] Isto.
[41] Isto, str. 34.
[42] Greedy algorithm. URL: https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Greedy_algorithm.html (2014-08-26)
[43] Isto.

4.2. Čisti i ortogonalni pohlepni algoritam

Pohlepni algoritmi su proučavani u statistikama i teoriji obrade signala još 1980-ih godina. No, aktivna istraživanja generalne teorije pohlepnih algoritama započela su tek 1990-ih godina. Pohlepni algoritmi se mogu okarakterizirati kao „kratkovidni" i kao „nepopravljivi". Savršeni su samo za probleme koji imaju optimalnu podlogu. Unatoč tome, najprikladniji su za jednostavne probleme poput vraćanja ostatka novca. Međutim, bitno je imati na umu da se pohlepni algoritam može koristiti kao algoritam izbora za određivanje prioritetnih opcija unutar potrage ili grane algoritma. S tim u svezi samo će se spomenuti da postoji nekoliko tipova pohlepnog algoritma, npr. čisti pohlepni algoritam i ortogonalni pohlepni algoritam.[44]

4.3. Primjeri korištenja pohlepnog algoritma

U mnogim algoritmima optimizacije potrebno je napraviti niz izbora. Strategiju dinamičkog programiranja često koristimo pri rješavanju takvih problema, gdje se optimalno rješenje traži primjenom rekurzije računajući od najmanjih problema prema složenijim. Nažalost, strategija dinamičkog programiranja nekad dovodi do neefikasnih algoritama, najčešće zbog predugih vremenskih izvršavanja. Stoga, u takvim slučajevima alternativna tehnika koju koristimo je strategija pohlepnog algoritma.[45] Ova strategija najčešće dovodi do jednostavnih i veoma brzih algoritama, ali nažalost nije toliko učinkovita kao dinamičko programiranje. Tehnika pohlepnog pristupa ne daje uvijek optimalno rješenje, no iako ne daje optimalno rješenje, često vodi na novu strategiju razvoja algoritma koji je učinkovitiji od prijašnjeg. Tehnikom pohlepnog pristupa, rješenje zadanog algoritma se konstruira u nizu koraka. U svakom se koraku bira mogućnost koja je lokalno optimalna u nekom smislu. Zamisao je da će nas takvi optimalni koraci dovesti do globalnog optimalnog rješenja. Također, postoje problemi kod kojih je dokazano da pohlepni algoritam pruža optimalno rješenje. Jedan od takvih problema je problem rasporeda zadataka. Postavlja se pitanje kako rasporediti zadatke, a da prosječno vrijeme izvršavanja zadatka bude minimalno. Općenito, može se dokazati da će prosječno vrijeme završavanja biti najmanje ukoliko se najprije izabere aktivnost koja se prva završava, a nadalje se od svih aktivnosti koje se ne preklapaju s prvom aktivnošću izabere ona koja završava prva i taj se algoritam ponavlja u skladu s pohlepnim algoritmom. Kako bi bilo objašnjeno što je to pohlepni pristup razradit će se na nekoliko sljedećih primjera:

---

[44] Isto.

[45] PMF-MO studentske stranice. URL: http://web.studenti.math.pmf.unizg.hr/~markob/ostalo/maturalna.pdf (2014-07-24)

Primjer 1. Trgovac vraća kupcu iznos od 62 kn, na raspolaganju su mu novčanice od 50, 20, 10, 5 i 1 kn. Većina će ljudi instinktivno vratiti jednu novčanicu od 50 kn, jednu od 10 kn i dvije od 1 kn. Takav algoritam vraća određen iznos uz najkraću moguću listu novčanica. Tj., izabere se najveća novčanica koja ne prelazi ukupnu sumu, stavlja se na listu za vraćanje, oduzme se od ukupnog iznosa te se postupak ponavlja sve dok se ne vrati određen iznos. Ovakva strategija dovodi do najefikasnijeg rješenja, ali samo zahvaljujući specijalnim svojstvima određenih novčanica.[46]

Međutim, obratimo pozornost na sljedeći primjer kad pohlepni pristup ne funkcionira. Naime, što ako je potrebno vratiti 15 kn pomoću novčanica od 11, 5 i 1 kn? Pohlepni algoritam prvo vraća 11 kn i tada mu preostaje samo da vrati četiri novčanice od 1 kn. Znači ukupno 5 novčanica, dok bi optimalno rješenje bilo vratiti 3 puta po 5 kn.[47]

Primjer 2. Pretpostavimo da se morate voziti od Islamabada do Lahora. Na početku je vaš spremnik pun goriva. Vaš spremnik kada je pun sadrži dovoljno litara goriva kako biste putovali *m* milja i imate kartu koja pokazuje udaljenost između benzinskih crpki na vašem putu. Neka d1 < d2 < … < dn budu lokacije svih benzinskih crpki na vašem putu, a oznaka d udaljenost od Islamabada do benzinske crpke. Udaljenost među susjednim benzinskim crpkama je najviše *m* milja. Također, udaljenost između zadnje benzinske crpke i Lahora je najviše *m* milja. Vaš cilj je što manje puta stati zbog goriva tijekom vašeg putovanja. Koristit će se pohlepni algoritam u obliku pseudo koda kako bi se utvrdilo na kojoj benzinskoj crpki se treba stati:[48]

```
current_distance = 0
current_stop = 0
stops = [ ]
while current != lahore:
  next_stop = 0
while distance(next_stop) - current_distance <= m:
  next_stop = next_stop + 1
  next_stop = next_stop - 1

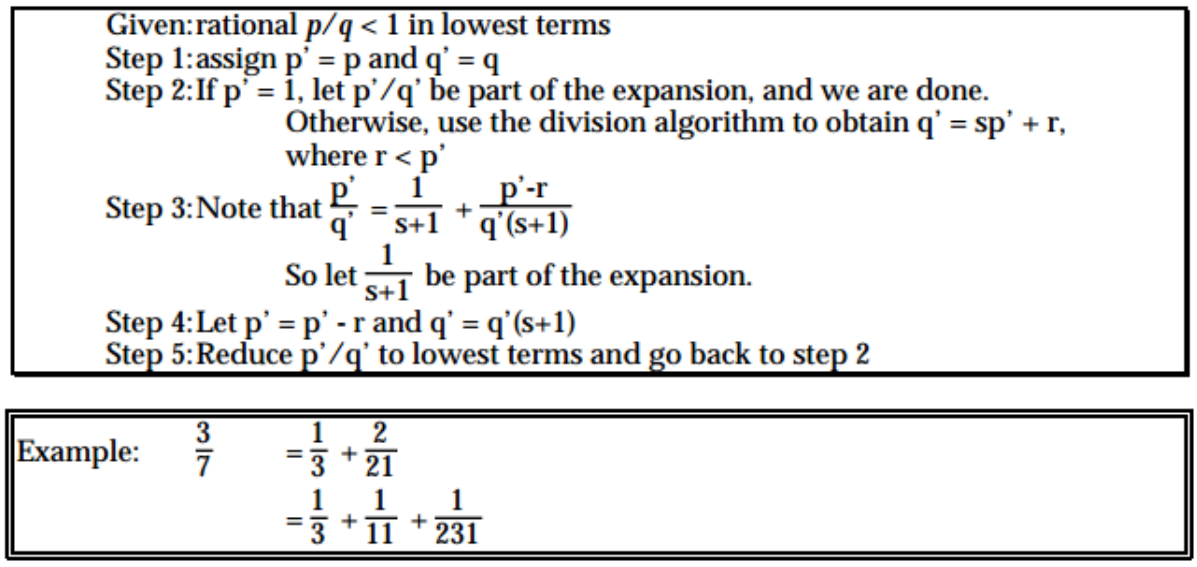

  current_stop = next_stop
```

[46] PMF-MO studentske stranice. URL: http://web.studenti.math.pmf.unizg.hr/~markob/ostalo/maturalna.pdf (2014-07-24)
[47] Isto.
[48] Stack Overflow. URL:http://stackoverflow.com/questions/17849059/greedy-algorithm-pseudo-code (2014-07-24)

```
current_distance = distance(current_stop)
  add next_stop to stops
return stops
```

Postavlja se pitanje je li rješenje optimalno. Ovaj algoritam počinje kod Islamabada i uzastopno pokušava ići koliko je god moguće do trenutka nestajanja goriva. Iz koda se može utvrditi kako je ovo optimalno rješenje, tj. korištenjem uvođenja možemo vidjeti je li pohlepni algoritam najdalje što može biti nakon prvog zaustavljanja te je li nakon n-tog stajališta najdalje što može dati stajalište n – 1, tada pohlepni algoritam mora biti najdalje što može biti za sva zaustavljanja na putu. Iako ovaj algoritam ima složenost O(n), on vraća optimalno rješenje računski.[49]

Primjer 3. je korištenje pohlepnih algoritama u matematici za egipatske razlomke. Opisani su od strane Fibonaccija, za pretvaranje racionalnih brojeva u egipatske razlomke. Egipatski razlomak je predstavljanje nesvodljivog razlomka kao sume jediničnih razlomaka, kao npr. 5/6=1/2 + 1/3. Kao što ime označava, ovo predstavljanje se koristi još iz vremena starog Egipta, ali prva objavljena metoda za konstruiranje takvih proširenja objašnjena je u Liber Abaciju od strane Leonarda iz Pise (Fibonacci).[50] Naziva se pohlepni algoritam jer u svakom koraku algoritam pohlepno bira najveći mogući jedinični razlomak koji se može iskoristiti u bilo kojem predstavljanju preostalih razlomaka. (Vidi Sliku 2.)



Slika 2. Korištenje pohlepnih algoritama u matematici za egipatske razlomke.[51]

[49]Stack Overflow. URL:http://stackoverflow.com/questions/17849059/greedy-algorithm-pseudo-code (2014-07-24)
[50]Usp. Dunton, M.; Grimm, R. E. Fibonnaci on egyptian fractions. // Fibonacci Quart., 4 (1966), str. 339.
[51] Gong, K. Egyptian Fractions.UC Berkley, 1992. Greedy algorithm.URL:
https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Greedy_algorithm.html (2014-08-26)

4.4. Slučajevi neuspjeha

Za mnoge probleme pohlepni algoritam ne uspije proizvesti optimalno rješenje, također, događa se da proizvede jedinstveno najgore moguće rješenje. Jedan od primjera je novac, tj. primjer sa samo 25, 10 i 4 centa. U ovome slučaju pohlepni algoritam ne bi bio u stanju usitniti 41 cent jer koristi jednu kovanicu od 25 centa i jednu od 10 centa te nije moguće koristiti kovanice od 4 centa zbog ravnoteže preostalih 6 centa, dok su osobe ili više sofisticiraniji algoritmi u mogućnosti napraviti promjenu, tj. usitniti 41 cent s pomoću kovanice od 25 centa i četiri kovanice od 4 centa.[52]

[52]Greedy algorithm. URL: https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Greedy_algorithm.html (2014-08-26)

5. Zaključak

Pohlepni algoritmi su lagani za provedbu i brzo rade, tj. smanjuju prostor pretraživanja, a također smanjuju i vremensku složenost. Naime, pohlepna strategija u svakom trenutku usmjerava na trenutno najbolje rješenje ne uzimajući u obzir da to rješenje ne mora voditi globalnom optimumu. Upravo zbog toga, pohlepni algoritmi ne moraju uvijek naći optimalno rješenje, ali su značajno brži od drugih algoritama. Budući da se pohlepni algoritmi relativno brzo izvršavaju i daju granice optimalnog rješenja, također omogućavaju da se drugim algoritmima smanji prostor pretrage rješenja. Dodatna prednost pohlepne strategije je da, ukoliko i ne pronađe optimalno rješenje, često vodi na novu strategiju razvoja algoritma koja rezultira učinkovitijim rješenjima ili tehnikom koja brzo pronalazi približno dobro rješenje. Iako postoje neki standardizirani problemi, većina problema koja je rješiva ovom metodom traži heuristiku. Važno je napomenuti da ne postoji generalni predložak kako primijeniti pohlepni algoritam kod određenog problema, no specifikacija problema može dati dobar uvid.

Literatura

1. Bang-Jensen, J.; Gutin, G.;Yeo, A. When the greedy algorithm fails: Discrete Optimization //Science Direct 1/2, 15(2004), str. 121-127. URL:http://www.sciencedirect.com/science/article/pii/S1572528604000222 (2014-07-27)
2. Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. Introduction to algorithms. The MIT Press, 2001.
3. Dunton, M.; Grimm, R. E. Fibonnaci on egyptian fractions. // Fibonacci Quart., 4 (1966), str. 339–354.
4. Gong, K. Egyptian Fractions. UC Berkley, 1992. Greedy algorithm.URL: https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Greedy_algorithm.html (2014-08-26)
5. Hoško, T.; Slamić, M. Strukture podataka i algoritmi: Priručnik. Zagreb : Algebra, 2009.
6. Ilić, A. Analiza algoritma. URL: http://www.pmf.ni.ac.rs/pmf/predmeti/7015/vezbe/slozenost_algoritama.pdf (2014-07-30)
7. Kiš, M. Englesko-hrvatski i hrvatsko-engleski informatički rječnik, Zagreb, Naklada Ljevak, 2000.
8. Obradović, M. Diofantske jednadžbe: propozicije-ekspozicije-algoritmi: za učenike srednjih škola. Zagreb : Naklada Ljevak, 2000.
9. PMF-MO studentske stranice. URL: http://web.studenti.math.pmf.unizg.hr/~markob/ostalo/maturalna.pdf (2014-07-24)
10. Singer, S. Složenost algoritama. Zagreb, 2005.
11. Složenost algoritama. URL: https://docs.google.com/document/d/1vwl2dLinUJfcwc8VPpig9FSOaM9yt7QiEhZ05VJ9ijc/edit (2014-07-30)
12. Stack Overflow. URL:http://stackoverflow.com/questions/17849059/greedy-algorithm-pseudo-code (2014-07-24)
13. Trahtenbrot, B. A. Što su algoritmi: algoritmi i računski automati. Zagreb: Školska knjiga, 1978.
14. Živković, D. Uvod u algoritme i strukture podataka. Beograd: Univerzitet Singdinum, 2010.