SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET

HANOJSKI TORNJEVI

Ivan Anđelić

Mentor:
prof.dr.sc. Slavomir Stankov

Neposredni voditelj:
dr.sc. Ani Grubišić

Split, rujan 2013.

Sadržaj

1 Uvod

Stvaranje zagonetke hanojskih tornjeva se pripisuje francuskom matematičaru Lucasu. jedan je od pionira teorije brojeva, najbitniji djelovi njegovog rada tiču se prostih brojeva i faktorizacije.

Problem hanojskih tornjeva se prvi put pojavljuje 1883. godine, a pripisan je N. Claus de Siam-u što je anagram od Lucas d'Amiens. Problem je opisan u njegovom djelu „Récréations mathematiques ". Navodno je problem inspiriran hindu legendom koja glasi otprilike ovako:

„U hramu Benares ispod kupole koja označava centar svjeta, tri dijamantne igle, svaka lakat visoka i široka kao tjelo pčele, stoje na postolju od bakra. Pri stvaranju svjeta Bog je stavio 64 diska od čistog zlata na jednu iglu, najveći disk na dno i ostale po veličini poredao na njega, sve do najmanjeg diska na vrhu. Svečenicima u hramu je dao zadatak da neprestano pomiču diskove, sve dok ih ne poslože na drugu iglu u originalnom poretku. Pravila za pomicanje diskova su jednostavna: pomiče se jedan po jedan disk i nikad se ne stavlja veći disk na manji. Po legendi kada svečenici završe prebacivanje diskova po zadanim uvjetima, hram, i sve na Zemlji će se pretvoriti u prašinu i sa udarom groma svemir će prestati postojati.„

Hoće li svijet prestati postojati kada svećenici iz legende premjeste sve tornjeve neznamo, ali možemo pokušati izračunati koliko bi im vremena trebalo da to naprave.

U 2. poglavlju detaljnije ću opisati problem i njegovog tvorca, nakon toga u 3. poglavlju sljedi opis rješenja problema rekurzivno i iterativno, pa programska implementacija rješenja u 4. poglavlju. U petom poglavlju ukratko ću predstaviti varijacije problema hanojskih tornjeva i za kraj zaključak i literatura u poglavljima 5 i 6.

2 Opis problema

Hanojski tornjevi je zagonetka koju je osmislio francuski matematičar Lucas 1883. godine, sastoji se od 3 tornja, na jednom se nalazi n diskova, poredanih po veličini, na dnu je najveći, na vrhu najmanji. Cilj je prebaciti sve diskove na neki od preostalih tornjeva, tako da oni zadrže originalni poredak, koristeći 2 jednostavna pravila:

1. Prebacuje se jedan po jedan disk
2. Nesmije se staviti veći disk na manji

2.1 Tko je bio Édouard Lucas?

Slika 1, Edouard Lucas

Édouard Lucas (Slika 1), punim imenom François Édouard Anatole Lucas je francuski matematičar rođen 4. travnja 1842. godine u Amienu, glavnom gradu regije Pikardije u kojem se i školovao. Radio je kao asistent u pariškoj zvjezdarnici do Francusko-Pruskog rata u kojem je služio kao topnički časnik. Nakon francuskog poraza u Parizu upisuje studij matematike i kasnije postaje profesor. Imao je reputaciju dobrog i zabavnog učitelja, koji je svoje učenike zabavljao matematičkim zagonetkama.

Kao jedna od tih zagonetki nastaje i problem Hanojskih tornjeva, kojeg je prvi put formulirao 1883. godine, kada je bio pripisan N. Claus de Siam-u što je anagram od Lucas d'Amiens. Zagonetku je i opisao u svom radu „Récréations mathematiques“.

Osim smišljanja zagonetki doprinuo je razvoju matematike i svojim radom u teoriji brojeva, posebno prostim brojevima i faktorizaciji. Proučavajući Fibonaccijev niz, niz brojeva 0,1,1,2,3,5,8,13,21,34,55 … gdje je svaki broj osim prva dva suma dva prethodna broja, koji je zadan sa $F_0 = 0$, $F_1 = 1$, $F_{n+1} = F_{n-1} + F_n$, i drugi niz brojeva, 2,1,3,4,7,11,18,29,47,76,123, sa sličnim svojstvima koji je zadan sa : $L_0=1, L_1=1$ i $L_n = L_{n-1} + L_{n-2}$, koji se u njegovu čast zove Lucasov niz, razvio je modernu metodu za testiranje jeli neki Mersennov broj prost, Mersennovi brojevi su oblika $2^n – 1$ gdje je n broj veći od 2 koji je i sam prost. Koristeći svoju metodu uspio je dokazati da je $2^{127} - 1$ prost broj i tako je otkrio novi Mersennov prost broj, jedini takav broj otkriven u vuše od 100 godina i najveći koji je ikad izračunat bez korištenja pomoći računala. Lucasova smrt je rezultat bizarne nesreće, na godišnjem kongresu za napredak znanosti, komadić tanjura koji se razbio mu je porezao obraz, rana se kasnije inficirala i umro je od bakterijske infekcije kože (*ThinkQuest, Edouard Lucas*).

3 Rješenje problema

Ako pažljivo promatramo poteze koji su nas doveli do rješenja, nakon što nekoliko puta rješimo problem hanojskih tornjeva, vidimo da se pojavljuje uzorak pri rješavanju.

Pretpostavimo da počinjemo sa N brojem diskova. Toranj na kojem se diskovi nalaze označimo slovom A, toranj koji nam je krajnji cilj označimo slovom B, a preostali toranj, koji zovemo pomoćni toranj označimo slovom C. Prvo odredimo „parnost“ diskova. Ako je broj diskova paran, najmanji disk i svaki drugi ispod njega zovemo parni diskovi a ostale neparni, a ako je broj diskova neparan, onda je najmanji disk neparan i svaki drugi ispod njega je također neparan, a ostali su parni. Svaki disk iste parnosti se uvijek kada dođe na red za pomicanje kreće istim uzorkom.

- Ako je disk paran redosljed svakog pojedinačnog diska je A-> C-> B ->A
- Ako je disk neparan redosljed svakog pojedinačnog diska je A-> B-> C ->A

Slijedeći ova pravila pomicanja, sve što trbamo odrediti je koji disk ćemo pomaknuti, a to je jednostavno, samo pomaknemo najmanji disk koji nije bio pomaknut u prošlom potezu.

Ako ovaj pristup primjenimo pri programskom rješavanju problema, dobijemo poprilično jednostavan iterativni algoritam, koji daje točno rješenje u najmanjem broju poteza. Relativno je jednostavan za razumjeti, a ako ga razumijemo, nije ga teško implementirati. Većina iterativnih rješenja ovog problema su kompliciranija od ovdje navedenog, pa ostaje pitanje postoji li neki drugi način za rješavanje problema? Odgovor je, naravno, da postoji. Odgovor će nam postati jasnji nakon što pokušamo matematički analizirati problem.

3.1 Matematička analiza problema

Za početak, želimo pronaći formulu koja nam za određeni broj diskova (označimo broj diskova sa n) daje najmanji broj poteza koji nam trebaju da bi rješili problem. Ako krenemo brojati korake za različit broj diskova dobijemo:

- Za n = 1, $K_n = 1$
- Za n = 2, $K_n = 3$
- Za n = 3, $K_n = 7$
- Za n = 4, $K_n = 15$

Iz ovoga možemo naslutiti uzorak ponavljanja. Ako povećamo broj diskova za 1 kroj koraka potrebnih za rješavanje problema se poveća za 2 puta prethodni broj koraka + još jedan korak, dakle broj koraka potrebnih za rješavanje problema u ovisnosti o broju diskova možemo izraziti kao:

$$T_n = 2 \times T_{n-1} + 1, n > 0$$

Ako dodamo početnu vrijednost $T_n = 0$ vidimo da smo dobili rekurzivnu formulu. Kako možemo izraziti problem pomoću rekurzije, to znači da ga možemo i programski rješiti pomoću rekurzije. (Skorks (29. 03. 2010.))

3.2 Rekurzivno rješenje

„Moć rekurzije leži u mogućnosti definiranja beskonačnog broja objekata sa konačno linija programskog koda. Sljedeći isti princip beskonačan broj naredbi (računanja) se može opisati sa konačnim programom“ (*Wirth, Niklaus, 1976.*).

Rekurzija u računarstvu je metoda u kojoj rješenje problema ovisi o rješenju manjeg ili jednostavnijeg dijela tog problema. Ovaj osnovni pristup se može primjeniti na puno vrsta problema i jedan je od osnovnih ideja u računarstvu. Većina programskih jezika implementira rekurziju tako da dozvoli da funkcija zove sama sebe iz programskog koda unutar same funkcije. Neki funkcijski programski jezici ne definiraju nikakve rogramske petlje nego se oslanjaju isključivo na rekurziju za obavljanje dijela koda koji se ponavlja. Računalna teorija je dokazala da takvi programi mogu rješiti jednake probleme kao i jezici koji imaju definirane petlje.(*Skorks, 2010.)*

3.2.1 Rekurzivne funkcije i algoritmi

Jedna od osnovnih metoda u rješavanju problema pomoću računala je podjeliti problem u manje ili jednostavnije probleme koji su istog tipa kao i originalan problem, rješiti te manje probleme i kombinirnjem rezultata dobiti rješenje početnog problema. Ova metoda se često naziva „Podjeli pa vladaj“. Kada se ova metoda koristi sa tablicama u koje se spremaju rješenja tih manjih problema (da bi se izbjeglo moguće ponovno računanje istih rezultata koje donosi trošak procesorskog vremena), dobijemo koncept koji nazivamo dinamičko programiranje. Definicija rekurzivne funkcije ima jedan ili više ulaznih podataka za koje funkcija vraća rezultat trivijalno (bez ponavljanja) i jedan ili više ulaznih podataka za koje je potrebno ponovno pozivanje same funkcije. Na primjer faktorijeli se rekurzivno mogu definirati sa jednadžbama:

$$0! = 1, \qquad n! = n \times (n-1)!$$

Nijedna funkcija sama za sebe nije potpuna definicija faktorijela, prva je osnovni korak a druga je rekurzivni korak. Zadatak rekurzivnog koraka je da složene ulazne podatke pojednostavni. U pravilno napisanoj rekurzivnoj funkciji ulazni problem će se sa svakim pozivanjem pojednostavnjivati do onog trenutka kada postsne jednak osnovnom slučaju i tada se rekurzija prekida. Za neke funkcije npr:

$$e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} \dots$$

ne postoji osnovni slučaj koji je impliciran ulaznim podacima, u tom slučaju se dodaje dodatni parametar koji će poprimanjem unaprijed određene vrijednosti prekinuti rekurziju (*Wikipedia, 2013. , Recursion, Comupter science*).

3.2.2 Tipovi rekurzije

Kako postoji više rezličitih tipova rekurzije, u ovom poglavlju ću ukratko opisati nekoliko različitih tipova rekurzija.

3.2.2.1 Jednostruka i višestruka rekurzija

Rekurzija koja sadrži samo jedan poziv same sebe naziva se jednostruka rekurzija, dok se rekurzija koja sadrži više poziva same sebe naziva višestrukom. Jednostruka rekurzija je najčešće puno efikasnija od višestruke rekutzije i u najvećem broju slučajeva može biti zamjenjena iterativnom metodom kojoj vrijeme izvršavanja i količina potrebne memorije rastu linearno. Vešestruke rekurzije najčesšće zahtjevaju vrijeme izvođenja i količinu memorije koja se povećava eksponencijalno.

3.2.2.2 Indirektna rekurzija

Večina osnovnih primjera rekurzije su primjeri direktne rekurzije, gdje funkcija poziva samu sebe. Indirektna rekurzija je rekurzija u kojoj funkcija poziva drugu funkciju koja onda poziva početnu funkciju, npr. F poziva G, a G poziva F, ako su F i G različite funkcije je primjer indirektne rekurzije. Ulančavanje više od dvije funkcije je također moguće.

3.2.2.3 Anonimna rekurzija

Ovo je tip rekurzije u kojem se funkcija ne poziva ekpslicitno po imenu,nego se implicitno poziva neka funkcija ovisno o kontekstu npr. poziva „funkciju koja se trenutno izvršava“ ili „funkciju koja je pozvala trenutnu funkciju“.

3.2.3 Rješenje problema Hanojskih tornjeva

Pri rekurzivnom rješavanju problema potrebno je iskoristiti informacije koje smo dobili matematičkom analizom problema. Ako rekurzivnu relaciju na pišemo u drugačijem obliku:

$$T_n = T_{n-1} + 1 + T_{n-1}$$

vidimo da, da bi rješili problem za n diskova prvo moramo pomaknuti n-1 diskova, zatim 1 disk, i onda opet n-1 disk. Dakle, moramo pomaknuti n-1 disk na pomoćni toranj, pomaknuti 1 disk na kraj i potom opet pomaknuti n-1 disk na kraj. Sljedeći zaključke matematičke analize dolazimo do rješenja koje se vrlo jednostavno zapiše i implementira.

4 Programska implementacija problema Hanojskih tornjeva

Za demonstraciju problema Hanojskih tornjeva izradio sam web aplikaciju u pomoću koje korisnik može sam pokušati rješiti problem ili pokrenuti algoritam koji ilustrira rekurzivno rješavanje problema.

4.1 Alati koje sam koristio

U izradi internet aplikacije sam koristio više različitih tehnologija. Ukratko ću opisati njihovu povijest i razvoj do danas.

4.1.1 HTML

HTML je kratica za HyperText Markup Language, što znači prezentacijski jezik za izradu web stranica i ostalog sadržaja koji se može prikazati u internet pregledniku. Svrha internet preglednika je čitanje i interpretiranje HTML dokumenata u internet stranice koje krajnji korisnici vide. Razvoj HTML-a počinje početkom 1990-ih, u CERN-u. Tim Berners-Lee je 1989.g radio u CERN-u na odjelu informacijsku tehnologije kada je smislio ideju iz koje se kasnije rodio HTML standard. Ideja je nastala iz potrebe da se znanstvenicima koji surađuju sa CERN-om iz svih djelova svijeta omogući da urede, organiziraju i prikupe sve informacije koje su im potrebne. Ali umjesto da te informacije pohrani kao dokumente koje bi onda svatko trebao snimati na svoje računalo, i povezivati, on je predložio da pokušaju povezati djelove teksta iz jednog dokumenta sa drugim dokumentom. To je značilo da bi mogli, dok čitate jedan dokument, mogli brzo prikazati drugi dokument koji sadrži bitne informacije, grafove ili bilo koji drugi oblik informacija. Takvo povezivanje dokumenata bi stvorilo „Web“, mrežu koja bi se čuvala u elektronskom obliku na računalima diljem svijeta. Prvi takav prototip Tim je izradio već 1980 za osobnu upotrebu i nazvao ga je Enquire. Prvi prototip za pretraživanje Web-a je nastao 1990 za NeXT računalo. Činjenica da je Web nastao početkom devedesetih nije nikakvo iznenađenje. Jedna od bitnijh stvari za razvoj weba je razvoj DNS-a. Taj novi, jednostavni način za imenovanje računala na internetu je omogućio lakši i brži pritup traženim podacima. Kasnije inačice HTML-a su postale standard za prikaz podataka na internetu. Najnovija revizija HTML standarda, HTML5, je nastalo sa ciljem da poboljša podršku za najnovije tipove medijskih datoteka, ostajući pri tom i dalje razumljiv kako ljduima, tako i kompjuterima. Dodana je podrška za audio i video elemente, kao i vektorsku

grafiku i matematičke formule, sve sa ciljem da se olakša stvaranje modernijih internet stranica (*Ragget, 1998, Wikipedia, HTML5)*.

4.1.2 JavaScript

JavaScript je skriptni programski jezik koji se izvršava u internet pregledniku. Sintaksa JavaScripta je nastala pod utjecajem programskog jezika C i Java, JavaScript podržava više paradigmi, pa se, uz ostale, može koristiti i objektno orjentirana paradigma. Osim u internet preglednicima raste primjena JavaScripta i izvan internet stranica, na primjer u PDF dokumentima (*Wikipedia, JavaScript*). Kratka povijest JavaScripta:

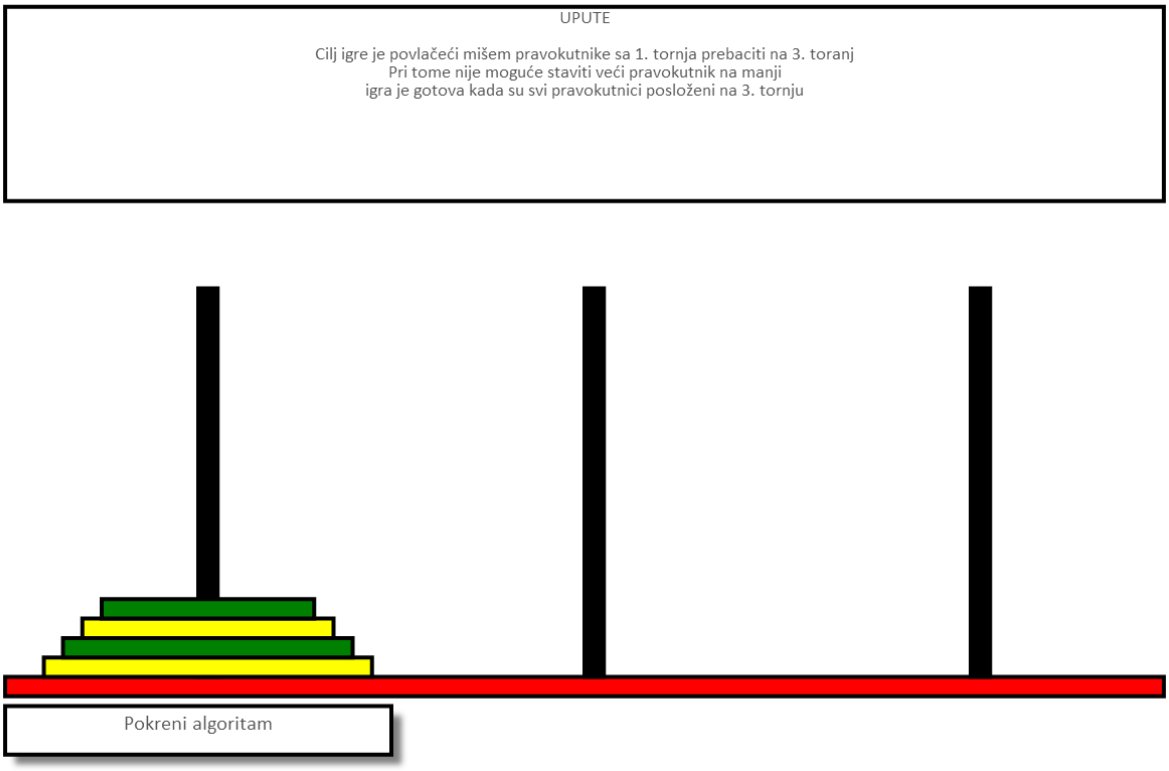
Javascript je nastao u 10 dana tokom svibnja 1995. Napisao ga je Brendan Eich, tada zaposlenik Netscapea a sada Mozille. Originalno ime mu je bilo Mocha, koje je odabrao osnivač Netscape-a Marc Andreessen. U rujnu 1995 ime je promjenjeno u LiveScript, a onda u listopadu iste godine u javaScript. To je bio i pomalo makretinški potez jer je Java tada bila popularna. 1996-1997 JavaSript je standardiziran i prilagođen proizvođačima internet preglednika. Prosec standardizacije se nastavlja u ciklusima 1998 i 1999, i ta verzija je osnova današnjeg JavaScripta. 2005 godine, JavaScript je revolucioniran, nastaje Ajax, set tehnologija, čiji kostur je JavaScript koji omogučuje stvaranje internet aplikacija u kojima se podatci moglu napuniti u pozadini, bez da se cijela stranica ponovo učitava, što je poslužilo javaScriptu da dobije na popularnosti i da nastanu mnoge biblioteke koje omogučuju njegovo lakše korištenje (*World Wide Web Consortium, A Short History of JavaScript*).

4.1.3 Kinetic.js

Kinetic.js je skup biblioteka za JavaScript koje se koriste u HTML5 elementu „canvas“ (platno) ii omogućavaju jednostavno korištenje animacija, tranzicija i kontroliranja događaja na nastalim objektima za mobilne i dekstop aplikacije (*KineticJS*).

4.2 Igra u kojoj korisnik sam rješava problem Hanojskih tornjeva

Kada se pokrene aplikacija u internet pregledniku dobije se prozor kao na slici:



Slika 2, početni izgled internet alikacije

Igra se igra sa četiri diska, posložena na prvom tornju, i cilj ih je pomaknuti na zadnji toranj. Sukladno pravilima problema, jedini disk koji se može pomicati je najgornji disk na nekom tornju. Korisnik mišem može odvući disk sa prvog tornja na neki od drugih tornjeva ili ga vratiti na toranj odakle ga je uzeo. Ako pri prebacivanju na drugi toranj krši pravila igre, tj. pokuša staviti veći disk na manji disk će se vratiti na svoju početnu poziciju. Ako je potez ispravan igrač nastavlja dalje pomicati diskove sve dok ih sve ne prebaci na posljednji toranj.

Na kraju igre svi se diskovi vrate na početni toranj i u pravokutniku na vrhu se ispiše poruka korisniku, koja ga obavijesti da je upsješno završio igru i ispiše mu njegov broj poteza i minimalni broj poteza.Pritiskom na „Pokreni algoritam“ korisnik u svakom trenutku može prekinuti dvoju igru i pokrenuti algoritam koji sam pomiče diskove odgovarajućim redosljedom i u najmanjem broju poteza završava igru. Nakon što se algoritam završi diskovi

se vraćaju na početne pozicije i korisnik može ponovno započeti igru. Dok se algoritam izvršava korisniku je onemogućeno samostalno pomicanje diskova.

4.3 Implementacija rekurzivnog algoritma

Princip rekurzivnog algoritma za problem hanojskih tornjeva je jednostavan. Ako uvedemo sljedeće oznake:

- Početni toranj -> A
- Krajnji toranj -> B
- Pomoćni toranj C
- Broj diskova -> N

Da bi pomakli N diskova na B, moramo prvo pomaknuti N-1 disk na C, zatim pomaknuti N-ti disk na C pa pomaknuti N-1 disk na B. Rekurzivni algoritam koji sam implementirao u svom rješenju je :

Ako je N=1

Pomakni toranj sa početnog tornja na krajnji toranj

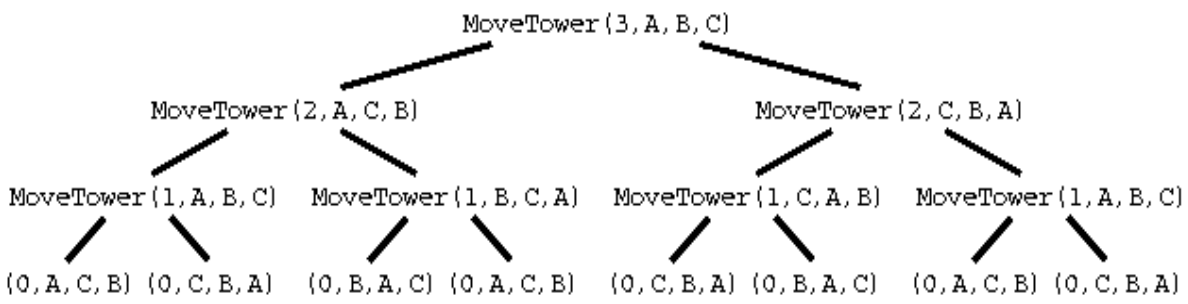
Inače

Pomakni n-1 toranj sa početnog na pomočni

Pomakni n-ti toranj sa početnog na krajnji

Pomakni n-1 toranj sa pomoćnog na krajnji

Vidimo da se u svakom pozivu funkcije u „else“ bloku funkcija zove rekurzivno dva puta. Dakle, ako grafički prikažemo pozive funkcija, dobijemo binarno stablo, u kojem svaki čvor ima dvoje djece. Npr, za N= 3 dobijemo stablo poziva funkcija koje izgleda kao na slici:



Slika 3

Redosljed pomicanja tornjeva koji se dobije ovakvim algoritmom za N= 3 je sljedeći:

1. A -> B (pomakni najgornji disk sa A na B)
2. A -> C (pomakni najgornji disk sa A na C)
3. B -> C (pomakni najgornji disk sa B na C)
4. A -> B (pomakni najgornji disk sa A na B)
5. C -> A (pomakni najgornji disk sa C na A)
6. C -> B (pomakni najgornji disk sa C na B)
7. A -> B (pomakni najgornji disk sa A na B)

U svom završnom radu koristio sam 4 diska, pa je rješenje za 4 diska: (*)

1. A -> C (pomakni najgornji disk sa A na C)
2. A -> B (pomakni najgornji disk sa A na B)
3. C -> B (pomakni najgornji disk sa C na B)
4. A -> C (pomakni najgornji disk sa A na C)
5. B -> A (pomakni najgornji disk sa B na A)
6. B -> C (pomakni najgornji disk sa B na C)
7. A -> C (pomakni najgornji disk sa A na C)
8. A -> B (pomakni najgornji disk sa A na B)
9. C -> B (pomakni najgornji disk sa C na B)
10. C -> A (pomakni najgornji disk sa C na A)
11. B -> A (pomakni najgornji disk sa B na A)
12. C -> B (pomakni najgornji disk sa C na B)
13. A -> C (pomakni najgornji disk sa A na C)
14. A -> B (pomakni najgornji disk sa A na B)
15. C -> B (pomakni najgornji disk sa C na B)

4.3.1 Procjena vremena potrebnog za izvršavanje

Analizom poziva funkcije i ispisa rješenja u obliku (*) dobijemo da za N = 4 broj izvršenih naredbi iznosi $30*2^N$. Ako krenemo pretpostavke da CPU izvršava 1 naredbu po ciklusu, i da radi na frekvenciji od 2.99 Ghz, dobijemo da mu je vrijeme izvršavanja jednog ciklusa 1/2.99E9. Sljedeći tu pretpostavku dobijemo vrijeme izvršavanja u ovisnosti o broju tornjeva. Ako to zapišemo tablično dobijemo prikaz kao na slici:

	A	B	C	D	E
33					
34			**Processor Speed**	2.992478	GHz
35			**Cycle length**	3.34171E-10	sec
36					
37					
38		**N**	**30 x 2^N**	**Total CPU time**	
39					
40		5	960	3.20804E-07	
41		6	1920	6.41609E-07	
42		7	3840	1.28322E-06	
43		8	7680	2.56643E-06	
44		9	15360	5.13287E-06	
45		10	30720	1.02657E-05	
46		11	61440	2.05315E-05	
47		12	122880	4.1063E-05	
48		13	245760	8.21259E-05	
49		14	491520	0.000164252	
50		15	983040	0.000328504	
51		16	1966080	0.000657007	
52		17	3932160	0.001314015	
53		18	7864320	0.002628029	
54		19	15728640	0.005256059	
55		20	31457280	0.010512117	
56		21	62914560	0.021024235	
57		22	125829120	0.04204847	
58		23	251658240	0.084096939	
59		24	503316480	0.168193878	
60		25	1006632960	0.336387756	
61		26	2013265920	0.672775512	
62		27	4026531840	1.345551025	
63		28	8053063680	2.69110205	
64		29	16106127360	5.3822041	
65		30	32212254720	10.7644082	
66		31	64424509440	21.5288164	
67		32	1.28849E+11	43.0576328	
68		33	2.57698E+11	86.1152656	
69		34	5.15396E+11	172.2305312	
70		35	1.03079E+12	344.4610624	
71		36	2.06158E+12	688.9221248	
72		37	4.12317E+12	1377.84425	
73		38	8.24634E+12	2755.688499	
74		39	1.64927E+13	5511.376998	
75		40	3.29853E+13	11022.754	
76					

Slika 4, vrijeme izvršavanja algoritma u ovisnosti o broju diskova

Vidimo da vrijeme izvršavanja dođe blizu jedne sekunde za N = 26, N = 27. Pogledajmo koliko bi ovakvom procesoru trebalo da rješi problem za 64 diska iz legende o budističkim svećenicima sa početka rada.

	A	B	C	D	E	F	G	H	I
1									
2			**Processor Speed**	2.992478	GHz				
3			**Cycle length**	3.34171E-10	sec				
4									
5		**N**	**30 x 2^N**	**Total CPU time**					
29		27	4026531840	1.345551025	seconds				
30		28	8053063680	2.69110205					
31		29	16106127360	5.3822041					
32		30	32212254720	10.7644082					
33		31	64424509440	21.5288164					
34		32	1.28849E+11	43.0576328					
35		33	2.57698E+11	86.1152656					
36		34	5.15396E+11	172.2305312					
37		35	1.03079E+12	344.4610624					
38		36	2.06158E+12	688.9221248					
39		37	4.12317E+12	1377.84425					
40		38	8.24634E+12	2755.688499					
41		39	1.64927E+13	5511.376998	1.53093806	hours			
42		40	3.29853E+13	11022.754	3.06187611				
43		41	6.59707E+13	22045.50799	6.12375222				
44		42	1.31941E+14	44091.01599	12.2475044				
45		43	2.63883E+14	88182.03197	24.4950089	1.02062537	days		
46		44	5.27766E+14	176364.0639	48.9900178	2.04125074			
47		45	1.05553E+15	352728.1279	97.9800355	4.08250148			
48		46	2.11106E+15	705456.2558	195.960071	8.16500296			
49		47	4.22212E+15	1410912.512	391.920142	16.3300059			
50		48	8.44425E+15	2821825.023	783.840284	32.6600118			
51		49	1.68885E+16	5643650.046	1567.68057	65.3200237			
52		50	3.3777E+16	11287300.09	3135.36114	130.640047			
53		51	6.7554E+16	22574600.18	6270.72227	261.280095			
54		52	1.35108E+17	45149200.37	12541.4445	522.560189	1.43167175	years	
55		53	2.70216E+17	90298400.74	25082.8891	1045.12038	2.8633435		
56		54	5.40432E+17	180596801.5	50165.7782	2090.24076	5.72668701		
57		55	1.08086E+18	361193603	100331.556	4180.48152	11.453374		
58		56	2.16173E+18	722387205.9	200663.113	8360.96303	22.906748		
59		57	4.32346E+18	1444774412	401326.226	16721.9261	45.8134961		
60		58	8.64691E+18	2889548824	802652.451	33443.8521	91.6269921		
61		59	1.72938E+19	5779097647	1605304.9	66887.7043	183.253984	1.83253984	centuries
62		60	3.45876E+19	11558195294	3210609.8	133775.409	366.507968	3.66507968	
63		61	6.91753E+19	23116390589	6421219.61	267550.817	733.015937	7.33015937	
64		62	1.38351E+20	46232781178	12842439.2	535101.634	1466.03187	14.6603187	
65		63	2.76701E+20	92465562355	25684878.4	1070203.27	2932.06375	29.3206375	
66		64	5.53402E+20	1.84931E+11	51369756.9	2140406.54	5864.1275	58.641275	
67									
68									

Slika 5, vrijeme izvršavanja algoritma u ovisnosti o broju diskova

Vidimo da se za 64 diska vrijeme izvršavanja poveća na 58.64 stoljeća. A brzina pomicanja diskova je $2.99*10^9$ poteza po sekundi. Ako pretpostavimo da svećenici mogu pomicati jedan disk po sekundi i da nikada ne naprave pogrešku trbalo bi im $58.64* 2.99*10^9$ stoljeća.

5 Varijacija problema Hanojskih tornjeva

Iako je problem Hanojskih tornjeva poprilično jednostavan, neke njegove izvedenice su pono kompliciraniji problemi, primjere nekih od njih ću ukratko opisati u ovom poglavlju.

5.1 Najveći broj poteza bez ponavljanja

Ako umjesto najmanjeg broja poteza za rješavanje problema Hanojskih tornjeva poželimo pronaći najveći broj poteza bez da ponavljamo poredak diskova. Jedan od načina za rješiti ovu varijaciju igre je dan sljedećim algoritmom:

Ako je N=0

Kraj

Inače

Pomakni N-1 diskova sa početnog tornja na pomoćni

Pomakni posljednji disk sa početnog tornja na krajnji

Pomakni N-1 diskova sa pomočnog na početni

Pomakni posljednji disk sa pomoćnog na krajni toranj

Pomakni N-1 diskova sa početnog na krajnji toranj

Ako predstavimo položaje N diskova na 3 tornja kao N-znamenkasti broj od znamenki 0.1,2 vidimo da postoji 3^N načina za to, dakle možemo napraviti $3^N - 1$ poteza bez da se oni ponavljaju. Znamo da su rješenja koja algoritam daje jednistvena i da se stanja diskova ne ponavljaju inače bi došlo do beskonačne petlje i algorizam nikad nebi stao.

5.2 Rješenje problema za M tornjeva

Ako povećamo broj tornjeva logično je da bi se najmanji broj poteza potrebnih za završavanje igre trebao smanjiti. Možemo za svaki od M>3 tornjeva ignorirati ostale tornjeve i rješiti ga koristeći samo 3 tornja ali to nam ne daje optimalno rješenje.

Verzija problema sa 4 tornja se prvi put pojavljuje 1909.g. Nakon nje se pojavljuju mnoge N,M verzije sa N diskova i M tornjeva. Kod povećanja broja tornjeva očito je da će se smanjiti broj pozela al problem nastaje kod određivanja optimalnog broja poteza. 1941.g „ American Mathematical Monthly “ izdaje 2 algoritma, Frameov i Stewartov koji rješavaju opći N,M problem ali njedan nije dokazao da je algoritam opptimalan, tj. da se problem nemože rješiti i sa manje poteza.Pseudokod algoritma je sljedeći:

T(N,M)-> minimalni broj poteza za pomicanje N diskova koristeći M tornjeva

1. Za neki K, 1 <= K <=N premjesti gornjih k diskova na neki toranj koji nije početni ili krajnji, koristeći T(N,M)poteza
2. Bez korištenja tornja koji sada sadrži k diskova prebaci preostalih N-K diskova na odredište, koristeći T(N-K,M-1) poteza
3. Premjesti gornjih K diskova na odredište koristeći T(K,M) poteza

6 Zaključak

Problem hanojskih tornjeva je star preko 100 godina, i do sada su pronađena mnoga njegova rješenja, od kojih sam ja neka pokazao u ovom radu. Ali zahvaljujući specifičnosti problema, za kojeg postoje razna rješenja, često se koristi u obrazovanju za demonstraciju rješavanja problema rekurzijom, jer je intuitivno rješenje očito a opet je dovoljno složen da bi se na njemu mogao pokazati princip korištenja rekurzivnih algoritama. Alalizom algoritma vidimo da se broj poteza a time i vremena izvođenja algoritma ekponencijalno povećava povećanjem broja diskova. Tako za veliki N dobijemo problem koji nije rješiv u realnom vremenu. Problemi poput ovoga mogu psolužiti za testiranje procesorske brzine i tako nam služe u ispitivanju novih tehnologija i arhitektura u razvoju procesora.

Za one željene većih logičkih izazova promjenom samo jednog parametra u problemu (povećanjem broja tornjeva) može se dobiti problem koji još nema rješenje za koje je dokazano da je optimalno,pa problem u općenitom obliku može poslužiti i kao puno kompleksniji istraživački rad.

7 Literatura

Raggett D. (1998) : Raggett on HTML 4, Addison Wesley, ISBN 0-201-17805-2

Wirth, Niklaus (1976.) : Algorithms + Data Structures = Programs, Prentice-Hall. p. 126.

ThinkQuest (1999.), Edouard Lucas, http://library.thinkquest.org/27890/biographies2.html

Skorks (29. 03. 2010.), Solving The Towers Of Hanoi Mathematcially And Programmatically – The Value Of Recursion, http://www.skorks.com/2010/03/solving-the-towers-of-hanoi-mathematically-and-programmatically-the-value-of-recursion/

Bogomolny A., Tower of Hanoi, the Hard Way, http://www.cut-the-knot.org/recurrence/LongHanoi.shtml

Wikipedia (9. 9. 2013.), Tower of Hanoi, http://en.wikipedia.org/wiki/Tower_of_Hanoi

Wikipedia (14. 9. 2013.), HTML5, http://en.wikipedia.org/wiki/HTML5

Wikipedia (15. 9. 2013.), Java Script, http://en.wikipedia.org/wiki/JavaScript

Wikipedia (29. 9. 2013), Recursion(Comupter science), http://en.wikipedia.org/wiki/Recursion_(computer_science)

World Wide Web Consortium (27. 06. 2012.), A Short History of JavaScript, http://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript

Kinetic JS, www.kineticjs.com

Theibaut D. (25. 11. 2012), Analysis of The Tower of Hanoi, http://cs.smith.edu/dftwiki/index.php/CSC231_Analysis_of_the_Towers_of_Hanoi

Tracing our Towers solution, http://www.cs.cmu.edu/~cburch/survey/recurse/hanoiex.html