SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

**Matija Belec**

PRIMJENA REKURZIVNIH ALGORITAMA U PROGRAMIRANJU

ZAVRŠNI/DIPLOMSKI rad

Varaždin, 2016.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Belec

Matični broj: 39912/11

Studij: Informacijski sustavi

PRIMJENA REKURZIVNIH ALGORITAMA U PROGRAMIRANJU

ZAVRŠNI/DIPLOMSKI RAD

Mentor:

Prof.dr.sc. Danijel Radošević

Varaždin, rujan 2016.

**Sadržaj**

[1. Uvod 1](#_Toc460696763)

[2. Matematička indukcija 2](#_Toc460696764)

[3. „Školski“ primjeri rekurzija 3](#_Toc460696765)

[3.1. Faktorijel 3](#_Toc460696766)

[3.2. Fibonaccijev niz 3](#_Toc460696767)

[4. Algoritmi podijeli pa vladaj 5](#_Toc460696768)

[4.1. Algoritmi sortiranja 5](#_Toc460696769)

[4.1.1. Quick sort 5](#_Toc460696770)

[4.1.2. Merge sort 6](#_Toc460696771)

[4.2. Algoritmi pretraživanja 7](#_Toc460696772)

[4.2.1. Binarno pretraživanje 7](#_Toc460696773)

[5. Pohlepni algoritmi 9](#_Toc460696774)

[5.1. Vraćanje ostatka novca 9](#_Toc460696775)

[5.2. Rastavljanje broja na egipatske razlomke 9](#_Toc460696776)

[6. Metoda pretraživanja s vraćanjem 10](#_Toc460696777)

[6.1. Problem N-kraljica 10](#_Toc460696778)

[7. Literatura 11](#_Toc460696779)

1. Uvod

U ovom ću radu navesti nekoliko primjera rekurzivnih algoritama koje ću detaljno razraditi. Algoritmi su podijeljeni u nekoliko poglavlja ovisno o njihovoj namjeni. Za početak nešto ću reći o matematičkoj indukciji koja je temelj rekurzivnih algoritama.

Zatim ću navesti nekoliko jednostavniji rekurzivnih algoritama. Radi se o algoritmima koji se često stavljaju kao osnovni primjeri rekurzije u školstvu.

Zatim slijede algoritmi za pretraživanje i sortiranje koji koriste metodu „podijeli pa vladaj“. Oni su po svojoj naravi malo složeniji od klasičnih „školskih“ algoritama sa početka rada.

Na kraju dolaze i rekurzivni algoritmi s vraćanjem (*engl. backtracking*) gdje će se razraditi složeniji rekurzivni algoritmi.

1. Matematička indukcija

T.

1. „Školski“ primjeri rekurzija

Faktorijel broja je matematička funkcija kojom se izračunava umnožak brojeva od 1 do odabranog broja n. Na primjer, iznosi 120:

U nastavku slijede klasični jednostavniji primjeri rekurzija.

* 1. Faktorijel

Prvi klasičan primjer rekurzije u ovom radu je izračun faktorijela broja. Primjer jednostavnog programskog koda koji koristi rekurzivni pristup je prikazan sljedećim programskim kodom:

// g++ faktorijel.cpp -o faktorijel.out

#include <iostream>

long long faktorijel(unsigned int n) {

return (n>1 ? n \* faktorijel(n-1) : 1);

}

int main(){

int n;

do std::cin >> n; while(n<1);

std::cout << faktorijel(n) << std::endl;

return 0;

}

Također, faktorijel je moguće programski izračunati i na iterativni način kako slijedi u donjem primjeru programskog koda.

// g++ faktorijel\_iter.cpp -o faktorijel\_iter.out

#include <iostream>

int main(){

int n;

do std::cin >> n; while(n<1);

long long f = 1;

for(int i=2; i<=n; i++) {

f \*= i;

}

std::cout << f << std::endl;

return 0;

}

* 1. Fibonaccijev niz

Drugi jednostavniji primjer rekurzije jest fibonaccijev niz. Fibonaccijev niz određen je sljedećom matematičkom formulom:

Prema navedenoj formuli možemo jednostavno izraditi programski kod koji slijedi u nastavku.

// g++ fibo.cpp -o fibo.out

#include <iostream>

long long fibo(unsigned int n) {

return (n > 1 ? fibo(n-1) + fibo(n-2) : n);

}

int main(){

int n;

do std::cin >> n; while(n<0);

std::cout << fibo(n) << std::endl;

return 0;

}

1. Algoritmi podijeli pa vladaj

Metoda „podijeli pa vladaj“ (u nekim literaturama se spominje naziv „podijeli i ovladaj“) radi upravo to što se i iz samog naziva može isčitati: složeniji problem dijeli na više manjih problema od kojih se nadalje svaki zasebno riješava. Dobivanjem riješenja manjih problema, nadalje se dolazi do riješenja složenijeg (prvozadanog) problema.

* 1. Algoritmi sortiranja

U programskim algoritmima često se dolazi do upotrebe određene vrste pretraživanja ili sortiranja. Kod sortiranja polja se radi pretraživanje polja ili dijela polja unutar kojeg tražimo elemente koji su veći ili manji od prethodno određenog elementa sa kojim ih uspoređujemo sa ciljem da se nakon procesa sortiranja, u spomenutom polju elemenata nalaze isti elementi ali sortirani redom od najmanjeg do najvećeg (ili obrnuto). Postoji mnogo vrsta sortiranja među kojima su više poznatiji:

* Sortiranje izborom,
* Sortiranje umetanjem,
* Sortiranje zamjenom,
* Quick sort,
* Sortiranje spajanjem (Merge sort).

Dok su prvi spomenuti oblici sortiranja koji se uglavnom implementiraju iterativnim pristupom, zadnja dva spomenuta, quick sort i sortiranje spajanjem, su dva najpoznatija algoritma soritiranja koja koriste metodu „podijeli pa vladaj“.

U nastavku se nalaze algoritamske reprezentacije i programska riješenja za oba navedena alogoritma za sortiranje koji koriste navedenu metodu „podijeli pa vladaj“.

* + 1. Quick sort

Algoritam sortiranja quick sort je vrlo popularan u većini programskih jezika kao standardni način sortiranja elemenata unutar određenog polja, odnosno liste. Uglavnom se radi o jednoj od „poboljšanih“ varijanta quick sort-a gdje se polje, umjesto na standardnih 2 dijela, dijeli na 3 dijela. U ovom radu ću se ograničiti na klasičan algoritam quick sort sortiranja i koristit ću samostalno napisan programski kod u programskom jeziku C/C++ kao primjer programskog riješenja navedenog sortiranja.

Za početak, ukratko o samom quick sort sortiranju. Navedeno sortiranje koristi sljedeće opisani algoritam:

1. Odredi „pivot“ element,
2. Podijeli polje na dva dijela tako da se u prvi (lijevi) dio polja prebace svi elementi manji od vrijednosti „pivot“ elementa, dok se preostali elementi postave u drugi (desni) dio polja,
3. Ponovi korak 2 za lijevi dio polja, te zatim za desni dio polja sve dok svaki dio polja ima elemenata.

Nakon što se korak 2 ponovi nad svakim dijelom polja rekurzivno, te se u dijelu polja nađe samo jedan element, taj element je već sortiran i nalazi se na točnom indeksu na kojem se treba nalaziti u sortiranom polju. Time se nakon rekurzivnog prebacivanja manjih elemenata u lijevi dio polja, a preostalih u desni dio polja dobiva sortiranje „pivot“ elementa u svakom koraku.

U nastavku se nalazi primjer programskog riješenja.

void qs(int\* p, int a, int b) {

int i=a, j=b;

int m=p[a], s;

while(i<=j) {

while(p[i] < m) i++;

while(p[j] > m) j--;

if(i<=j) {

s=p[i]; p[i]=p[j]; p[j]=s;

i++; j--;

}

}

if(a<j) qs(p, a, j);

if(i<b) qs(p, i, b);

}

//qs(p, 0, n-1)

Na samom kraju programskog riješenja vidljiv je i sam poziv navedene funkcije qs. Prvi argument funkcije je polje (točnije pokazivač na prvi element polja), drugi argument je indeks prvog elementa polja (ili dijela polja koji se želi sortirati), te posljednji, treći argument je indeks zadnjeg elementa polja (ili dijela polja koji se želi sortirati).

* + 1. Merge sort

Drugi algoritam sortiranja koji koristi metodu „podijeli pa vladaj“ koji ću obraditi u ovom radu je sortiranje spajanjem (poznatiji pod engleskim nazivom „merge sort“).

* 1. Algoritmi pretraživanja

U algoritmima pretraživanja također se može naći i algoritama koji koriste metodu „podijeli pa ovladaj“. Iako se iterativni algoritmi mogu lako prebaciti u rekurzivne, postoji standardni algoritam pretraživanja naziva binarno pretraživanje koji se jednostavnije implementira rekurzivno nego iterativno (iako je iterativni pristup gledajući sa aspekta memorije svakako manje zahtjevan nego rekurzivni).

* + 1. Binarno pretraživanje

Specifičnost binarnog pretraživanja leži u klasičnom stilu metode „podijeli pa vladaj“. Radi se o algoritmu pretraživanja kojim se pronalazi traženi element u polju elemenata na način da se u svakom koraku polje podijeli na dva podjednaka dijela, te se zatim uspoređuje središnji element (element između dva dijela polja) sa traženim elementom. Ukoliko je traženi element jednak središnjem elementu tada je element pronađen u polju. Ukoliko je vrijednost traženog elementa manja od vrijednosti središnjeg elementa daljnja potraga za elementom nastavlja se nad prvim (lijevim) dijelom polja na isti princip kao i u prvom slučaju: odredi se središnji element i polje se time podijeli na dva dijela pa se radi navedena usporedba ponovo. Ukoliko je vrijednost traženog elementa veća od vrijednosti središnjeg elementa tada se traženje na isti princip ponavlja na drugom (desnom) dijelu polja. Takvo se rekurzivno traženje ponavlja sve dok se ne pronađe element ili dijelu polja ponestane elemenata što sugerira na to da traženi element u polju ne postoji.

Postoji nekoliko varijanti algoritma ovisno o željenoj implementaciji: algoritam može vratiti vrijednost tipa bool, odnosno podatak da li se traženi element nalazi u polju ili ne, a može se implementirati i verzija u kojoj se vraća vrijednost -1 ako traženi element ne postoji u polju, odnosno indeks elementa u polju ako je isti pronađen.

U nastavku se nalazi implementacija algoritma koja vraća indeks traženog elementa u polju ili -1 ukoliko traženi element ne postoji u polju.

int bs(int\* polje, int x, int i, int j) {

if(i>j) return -1;

int m = (j-i)/2 + i;

if(x == polje[m]) return m;

if(x < polje[m]) return bs(polje, x, i, m-1);

return bs(polje, x, m+1, j);

}

1. Pohlepni algoritmi

T.

* 1. Vraćanje ostatka novca

T.

* 1. Rastavljanje broja na egipatske razlomke

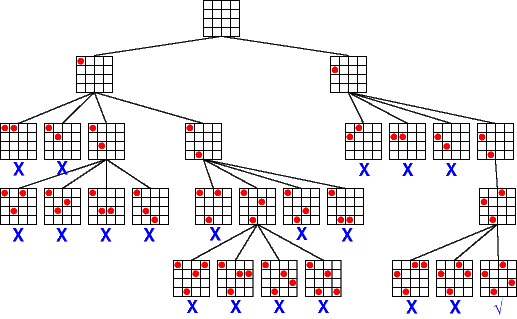
T.

1. Metoda pretraživanja s vraćanjem

T.

* 1. Problem N-kraljica

Problem N-kraljica možemo riješiti pomoću rekurzivnog algoritma s vraćanjem. U nastavku je vizualni prikaz kako se pomoću navedenog algoritma može tražiti riješenje problema za N=4 kraljice.



Slika rekurzivno pronalaženje pozicija kraljica za N=4

(http://ktiml.mff.cuni.cz/~bartak/constraints/images/backtrack.gif, 3.9.2016.)

U nastavku slijedi primjer programskog koda koji riješava navedeni problem.

1. Literatura
2. Barnett G, Del Tongo L (2008.) Data Structures and Algorithms
3. Drozdek A (2013.) Data Structures and Algorithms in C++
4. Knuth D E (?) Art of Computer Programming, Volume 3
5. Sedgewick R, Flajolet P (2013.) An Introduction to the Analysis of Algorithms.
6. Skiena S S (2008.) The Algorithm Design Manual