Algoritmi in podatkovne strukture 1

Visokošolski strokovni študij Računalništvo in informatika

Algoritmi in problemi

Jurij Mihelič, UniLj, FRI

Algoritmi

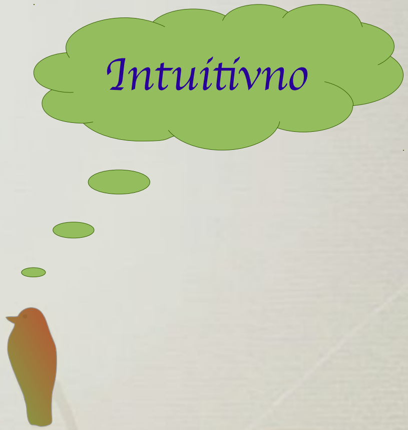
- Izvor izraza
  - al-Khwārizmī → al-gwaritmi → algoritmi

Sem
Muhammad ibn Mūsā al-Khwārizmī
Perzijski matematik iz 9. stoletja.

Opisal sem algoritme za seštevanje,
množenje, deljenje, kvadratni koren,
izračun decimalk π, itd. števil
v Arabskem (Indijskem) zapisu.

Algoritmi

- Algoritem je
  - jasen in nedvoumen
    - razumljiv in razumljiv na samo en način
  - mehaničen postopek
    - izvajanje postopka ne zahteva genialnosti,
  - ki za dani ***vhod***
  - vrne ustrezen ***izhod***.



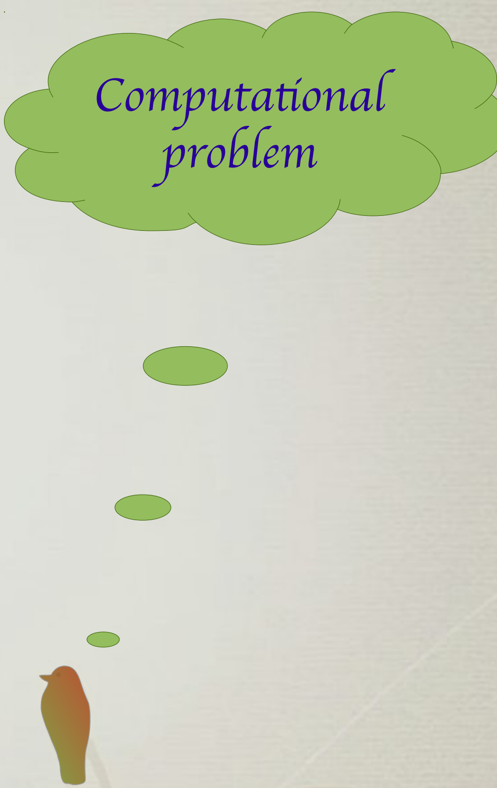
Algoritmi

- Algoritem je
  - jasen in nedvoumen mehaničen postopek
  - za reševanje ***računskega problema***.

  - Kaj je računski problem?
  - Kaj pomeni rešiti ga?

Računski problemi

- Rešiti računski problem pomeni,
  - za vse možne **naloge** problema
  - znati poiskati ustrezno **rešitev**.



- Računski problem
  - splošno in natančno opisuje želeni odnos
  - med ***nalogami*** in
  - njihovimi ***rešitvami.***

Računski problemi

- Vrste računskih problemov
  - *odločitveni* – rešitev je oblike da/ne
  - *preštevalni* – rešitev je število
  - *naštevalni* – rešitev je seznam
  - *iskalni* – rešitev je iskani objekt
  - *optimizacijski* – iskanje najboljše rešitve
    - *kriterijski* – iščemo le vrednost rešitve
    - *konstrukcijski* – rešitev želimo konstruirati
  - itd.

Računski problemi

- Formalna definicija računskega problema
  - množica vseh parov
    - naloga problema
    - in njena rešitev

Alan Turing, 1912 - 1954

- Formalna definicija algoritma
  - Turingov stroj, …, programski jeziki

- Church-Turingova teza
  - S Turingovim strojem se da izračunati vse, kar se sploh da izračunati.

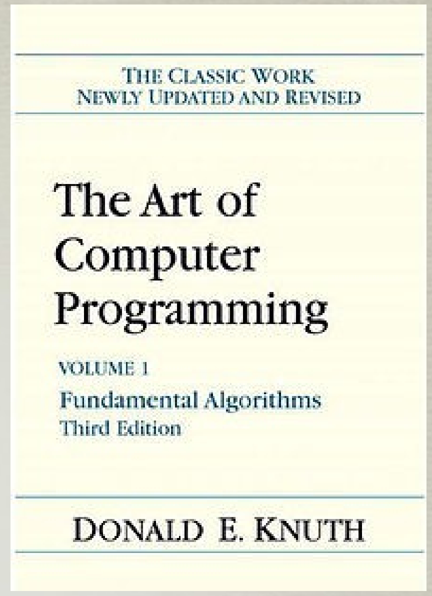
Snovanje algoritmov

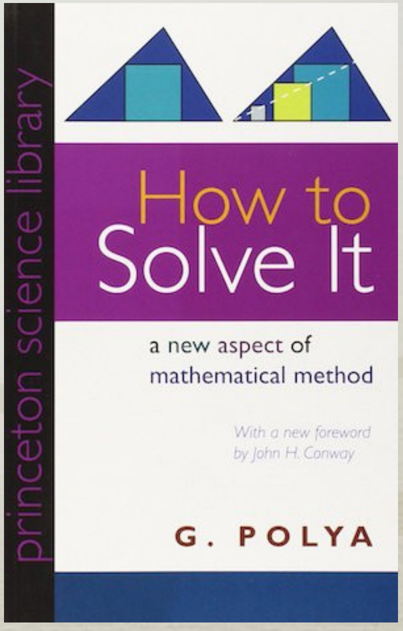
- Kako snovati algoritme?
- Predpogoj
  - dobro razumevanje problema
- Cilj
  - opis algoritma
- Kriteriji
  - učinkovitost
  - preprostost
  - implementabilnost

Zapisal sem Evklidov algoritem za največji skupni delitelj dveh števil.

Snovanje algoritmov

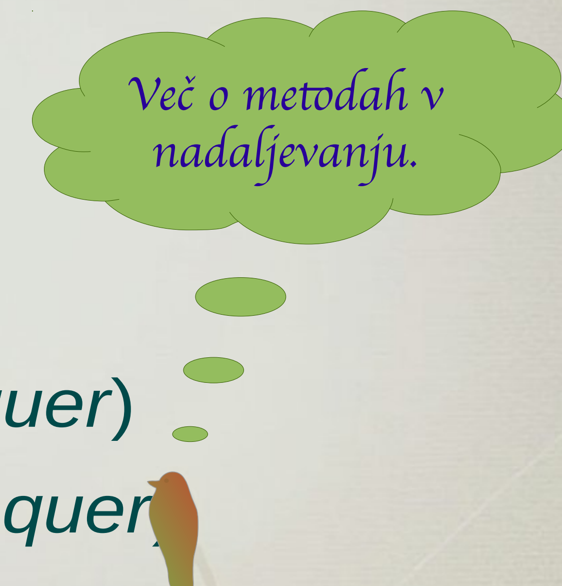
- Kako snovati algoritme?
  - Snovanje algoritmov je umetnost
    - TAOCP, Donald Knuth
  - Kdor hoče dobro pisati,
    mora veliko brati.
  - How to solve it?, George Polya
    1. Razumevanje problema
    2. Izdelava načrta
    3. Sledenje načrtu
    4. Pogled nazaj





Snovanje algoritmov

- Metode snovanja algoritmov
  - groba sila (*brute force*) in izčrpno preiskovanje (*exhaustive search*)
  - sestopanje (*backtracking*)
  - razveji in omeji (*branch & bound*)
  - požrešno (*greedy*)
  - deli in vladaj (*divide & conquer*)
  - zmanjšaj in vladaj (*reduce & conquer*)
  - pretvori in vladaj (*transform & conquer*)
  - dinamično programiranje (*dynamic programming*)
  - linearno programiranje (*linear programming*)
  - …



Snovanje algoritmov

- Opisni jeziki
  - naravni jezik
  - diagrami poteka
  - psevdokoda
  - programski jezik
  - strojna koda
  - itd.

Snovanje algoritmov

- Naravni jezik
  - nejasnost
  - dvoumnost
  - primeren za opis ideje

Prepisovalci bodo
javno obešeni
na oglasni deski.

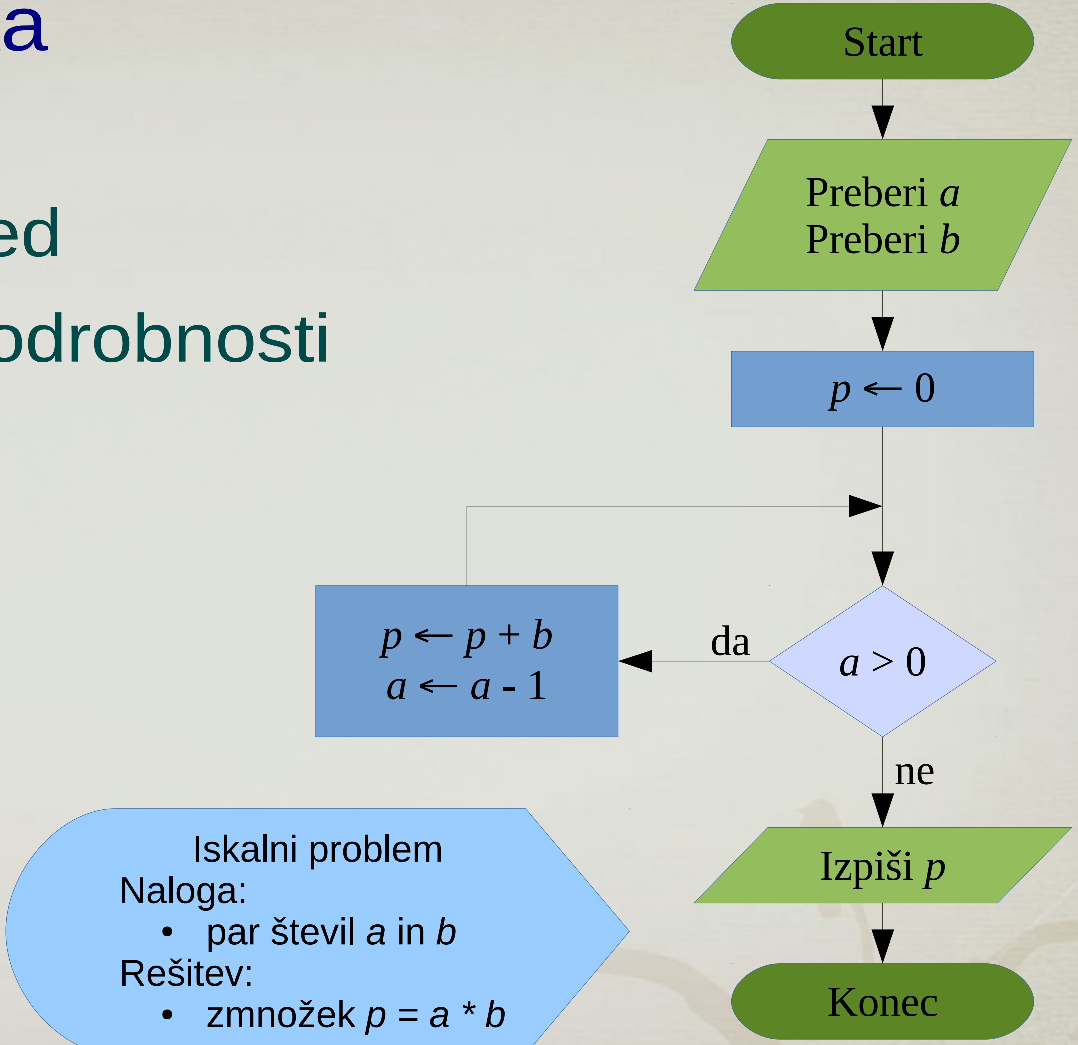
Dvojiško iskanje elementa
v urejeni tabeli
izvedemo tako, da s
primerjavo iskanega elementa
s sredinskim elementom tabele
ugotovimo ali je iskani element
v levi ali desni polovici.
Nato gremo iskat element
v ustrezni del tabele.

Odločitveni ali iskalni problem
Naloga:
- urejena tabela elementov

Rešitev:
- odgovor da/ne
- indeks iskanega elementa

Snovanje algoritmov

- Diagram poteka
  - grafični prikaz
  - globalen pogled
  - okoren opis podrobnosti



Snovanje algoritmov

- Psevdokoda
  - prenosljivost?
    - človek/človek
    - človek/stroj, stroj/stroj
  - poljubna natančnost opisa
  - uporaba matematičnih formul

Iskalni problem
Naloga:
- par števil *a* in *b*

Rešitev:
- zmnožek *p* = *a* * *b*

```
preberi a in b
p ← 0
while a > 0 do
    p ← p + b
    a ← a – 1
endwhile
izpiši p
```

Glej tudi: http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

```
Naj bo L seznam celih števil od 2 do N
Ponavljaj
    Naj bo X prvi neobdelani element v seznamu L
    Izpiši X
    Iz seznama odstrani vse večkratnike X
Dokler je X manjši od N
```

Iskalni problem
Naloga:
- število *N*

Rešitev:
- seznam praštevil

Snovanje algoritmov

- Programski jezik
  - realnost
  - ogromna izbira
  - algoritem lahko dejansko izvedemo

Iskalni problem
Naloga:
- par števil *a* in *b*

Rešitev:
- število g - gcd

```
.text
.global pgcd

pgcd:
    push    %ebp
    mov     %esp, %ebp
    mov     8(%ebp), %eax
    mov     12(%ebp), %ecx
    push    %edx
.loop:
    cmp     $0, %ecx
    je      .end
    xor     %edx, %edx
    div     %ecx
    mov     %ecx, %eax
    mov     %edx, %ecx
    jmp     .loop
.end:
    pop     %edx
    leave
    ret
```

```
int gcd(int a, int b) {
   if (b == 0) return a;
   return gcd(b, a % b);
}
```

```
gcd a 0 = a
gcd a b = gcd b (a `rem` b)
```

```
: gcd ( a b -- c )
    [ abs ] [ [ nip ] [ mod ] 2bi gcd ] if-zero ;
```

Vir programov: http://rosettacode.org/wiki/Greatest_common_divisor

Snovanje algoritmov

- Strojna koda
  - razumljiva le računalniku
  - hitro izvajanje


```
0000000 cf fa ed fe 07 00 00 [illegible] 00 00 80 02 00 00 00
0000010 0f 00 00 00 38 03 00 [illegible]
0000020 19 00 00 00 48 00 00 [illegible]
0000030 52 4f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000040 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
0000050 [illegible] 00 00 00 00 00 00
0000060 [illegible] 00 00 38 01 00 00
0000070 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000080 00 00 [illegible]
0000090 00 00 [illegible]
00000a0 07 00 00 00 05 00 00 00 03 00 00 00 00 00 00 00
00000b0 5f 5f 74 65 78 74 00 00 00 00 00 00 00 00 00 00
00000c0 5f 5f 54 45 58 54 [illegible] 00 00 00 00 00 00
00000d0 20 0f 00 00 01 00 00 00 77 00 00 00 00 00 00 00
00000e0 20 0f 00 00 04 00 00 00 00 00 00 00 00 00 00 00
00000f0 00 04 00 80 00 00 00 00 00 00 00 00 00 00 00 00
0000100 5f 5f 75 6e 77 69 6e 64 5f 69 6e 66 6f 00 00 00
0000110 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000120 98 0f 00 00 01 00 00 00 48 00 00 00 00 00 00 00
0000130 98 0f 00 00 02 00 00 00 00 00 00 00 00 00 00 00
0000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000150 5f 5f 65 68 5f 66 72 61 6d 65 00 00 00 00 00 00
0000160 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000170 e0 0f 00 00 01 00 00 00 18 00 00 00 00 00 00 00
0000180 e0 0f 00 00 03 00 00 00 00 00 00 00 00 00 00 00
0000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00001a0 19 00 00 00 48 00 00 00 5f 5f 4c 49 4e 4b 45 44
00001b0 49 54 00 00 00 00 00 00 00 10 00 00 01 00 00 00
00001c0 00 10 00 00 00 00 00 00 00 10 00 00 00 00 00 00
00001d0 d0 00 00 00 00 00 00 00 07 00 00 00 01 00 00 00
00001e0 00 00 00 00 00 00 00 00 22 00 00 80 30 00 00 00
00001f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
0000210 00 10 00 00 38 00 00 00 02 00 00 00 18 00 00 00
0000220 58 10 00 00 04 00 00 00 98 10 00 00 38 00 00 00
0000230 0b 00 00 00 50 00 00 00 00 00 00 00 00 00 00 00
0000240 00 00 00 00 03 00 00 00 03 00 00 00 01 00 00 00
0000250 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
0000280 0e 00 00 00 20 00 00 00 0c 00 00 00 2f 75 73 72
0000290 2f 6c 69 62 2f 64 79 6c 64 00 00 00 00 00 00 00
00002a0 1b 00 00 00 18 00 00 00 56 2d 90 64 27 5b 38 9c
00002b0 [illegible] 24 00 00 00 10 00 00 00
00002c0 [illegible] 2a 00 00 00 10 00 00 00
00002d0 00 00 00 00 00 00 00 00 28 00 00 80 18 00 00 00
00002e0 70 0f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00002f0 0c 00 00 00 38 00 00 00 18 00 00 00 02 00 00 00
0000300 01 01 ad 04 00 00 01 00 2f 75 73 72 2f 6c 69 62
0000310 2f 6c 69 62 53 79 73 74 65 6d 2e 42 2e 64 79 6c
0000320 69 62 00 00 00 00 00 00 26 00 00 00 10 00 00 00
0000330 38 10 00 00 08 00 00 00 29 00 00 00 10 00 00 00
0000340 40 10 00 00 00 00 00 00 2b 00 00 00 10 00 00 00
0000350 40 10 00 00 18 00 00 00 00 00 00 00 00 00 00 00
0000360 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
0000f20 55 48 89 e5 48 83 ec 10 89 7d f8 89 75 f4 81 7d
0000f30 f4 00 00 00 00 0f 85 0b 00 00 00 8b 45 f8 89 45
0000f40 fc e9 14 00 00 00 8b 7d f4 8b 45 f8 99 f7 7d f4
0000f50 89 d6 e8 c9 ff ff ff 89 45 fc 8b 45 fc 48 83 c4
0000f60 10 5d c3 66 66 66 66 2e 0f 1f 84 00 00 00 00 00
0000f70 55 48 89 e5 48 83 ec 10 bf 2a 00 00 00 be 17 00
0000f80 00 00 e8 99 ff ff ff be 00 00 00 00 89 45 fc 89
0000f90 f0 48 83 c4 10 5d c3 90 01 00 00 00 1c 00 00 00
0000fa0 00 00 00 00 1c 00 00 00 00 00 00 00 1c 00 00 00
0000fb0 02 00 00 00 20 0f 00 00 34 00 00 00 34 00 00 00
0000fc0 98 0f 00 00 00 00 00 00 34 00 00 00 03 00 00 00
0000fd0 0c 00 01 00 10 00 01 00 00 00 00 00 00 00 00 01
0000fe0 14 00 00 00 00 00 00 00 01 7a 52 00 01 78 10 01
0000ff0 10 0c 07 08 90 01 00 00 00 00 00 00 00 00 00 00
0001000 00 01 5f 00 05 00 03 5f 6d 68 5f 65 78 65 63 75
0001010 74 65 5f 68 65 61 64 65 72 00 26 67 63 64 00 2a
0001020 6d 61 69 6e 00 2f 02 00 00 00 03 00 a0 1e 00 03
0001030 00 f0 1e 00 00 00 00 00 a0 1e 50 00 00 00 00 00
0001040 fa de 0c 05 00 00 00 14 00 00 00 01 00 00 00 00
0001050 00 00 00 00 00 00 00 00 02 00 00 00 0f 01 10 00
0001060 00 00 00 00 01 00 00 00 16 00 00 00 0f 01 00 00
0001070 20 0f 00 00 01 00 00 00 1b 00 00 00 0f 01 00 00
0001080 70 0f 00 00 01 00 00 00 21 00 00 00 01 00 00 01
0001090 00 00 00 00 00 00 00 00 20 00 5f 5f 6d 68 5f 65
00010a0 78 65 63 75 74 65 5f 68 65 61 64 65 72 00 5f 67
00010b0 63 64 00 5f 6d 61 69 6e 00 64 79 6c 64 5f 73 74
00010c0 75 62 5f 62 69 6e 64 65 72 00 00 00 00 00 00 00
```

Snovanje algoritmov

- Semantični prepadi
  - ideja algoritma

    snovanje, metode snovanja
  - opis algoritma (algoritem)

    programiranje, implementacija
  - izvorna koda

    prevajanje / tolmačenje
  - strojna koda

    izvajanje
  - proces

Abstrakcija

Implementacija algoritmov

- Programiranje 1, 2, in 3 itd.
  - branje vhoda in izpis izhoda
  - aritmetične in logične operacije
  - nizi in tabele (polja, *array*)
  - odločitveni stavki
  - iteracija oz. zanke

Implementacija algoritmov

- Rekurzija
  - funkcija kliče samo sebe (lahko posredno)
  - podprta v večini jezikov (potrebuje sklad)
  - ustavljanje rekurzije
  - repna rekurzija

```
fun produkt(a, b) is
	if a == 0 then return 0
	return b + produkt(a – 1, b)
end
```

```
fun fib(n) is
    if n == 0 then return 0
    if n == 1 then return 1
    return fib(n - 1) + fib(n - 2)
```

```
fun hanoi(n, a, b, c) is
    if n == 0 then return
    hanoi(n-1, a, c, b)
    a -> b
    hanoi(n-1, c, b, a)
```

Implementacija algoritmov

- Rekurzija
  - repna rekurzija (*tail recursion*)

```
int gcd(int a, int b) {
   if (b == 0) return a;
   return gcd(b, a % b);
}
```

→

```
int gcd(int a, int b) {
    while (b > 0) {
        int c = a % b;
        a = b;
        b = c;
    }
    return a;
}
```

Implementacija algoritmov

- Razhroščevanje kode
  - `printf` metoda
  - sledenje programu (*trace*)
  - prekinitvena točka (*breakpoint*)
  - opazovanje (*watch*)
- Profiliranje in instrumentacija kode
  - ugotavljanje, koliko časa/pomnilnika/itd. porabijo posamezni deli programa
  - programu dodamo ukaze za merjenje

Implementacija algoritmov

- Izvajanje programa
  - prevajanje izvorne kode v strojno kodo
  - interpretiranje izvorne kode
- Izvedba algoritma
  - program zaženemo na ustreznem računalniku
  - izvajanje eksperimentov
  - znanstvena metoda: hipoteza
  - eksperimentalno ovrednotenje algoritma

Sled algoritma

- Sled algoritma
  - izpis podatkov tekom izvajanja, npr.:
    - spremenljivke, podatkovne strukture
    - št. korakov, globina rekurzije, itd.
- Izvajanje
  - simuliranje na papir
  - dejansko z računalnikom

Sled algoritma

- Evklidov algoritem

```
int gcd(int a, int b) {
   if (b == 0) return a;
   return gcd(b, a % b);
}
```

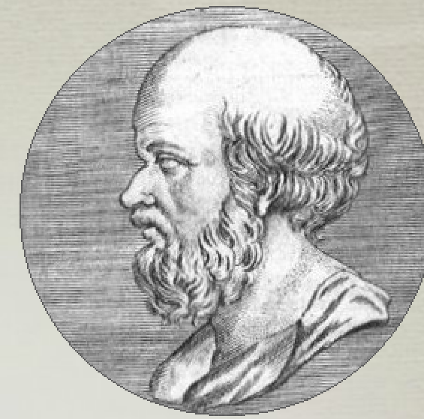
gcd(264, 72)

#	a	b	q	r
0	264	72	3	48
1	72	48	1	24
2	48	24	2	0
3	**24**	0		

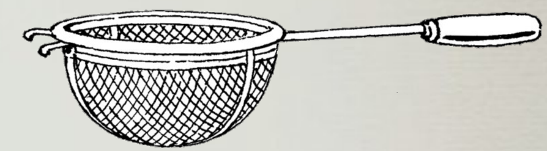
```
gcd(264, 72) =
gcd(72, 48) =
gcd(48, 24) =
gcd(24, 0) = 24
```

Sled algoritma

- Eratostenovo sito



Naj bo *L* seznam celih števil od 2 do *N*
Ponavljaj
    Naj bo *X* prvi neobdelani element v seznamu *L*
    Izpiši *X*
    Iz seznama odstrani vse večkratnike *X*
Dokler je *X* manjši od *N*



Eratostenovo sito za n = 30

#	izbrani X in seznam L																												
1	**2**	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
2	*2*	**3**		5		7		9		11		13		15		17		19		21		23		25		27		29	
3	*2*	*3*		5		7				11		13				17		19				23		25				29	
4	2	3		5		7				11		13				17		19				23						29	
5	2	3		5		7				11		13				17		19				23						29	
6	2	3		5		7				11		13				17		19				23						29	
7	2	3		5		7				11		13				17		19				23						29	
8	2	3		5		7				11		13				17		19				23						29	
9	2	3		5		7				11		13				17		19				23						29	
10	2	3		5		7				11		13				17		19				23						29	
11	2	3		5		7				11		13				17		19				23						29	

Pravilnost algoritmov

- Pravilnost algoritmov
  - Ali algoritem res dela tisto, kar mislimo, da dela?
  - glede na neko specifikacijo
- Delna pravilnost
  - vrne ***pravilno rešitev***, kadar jo vrne
  - lahko se zacikla
- Totalna pravilnost
  - ustavljivost
  - se za vse vhode ***ustavi***

Pravilnost algoritmov

- Dokazovanje pravilnosti
  - intuitivno razumevanje
  - preverjanje s testnimi primeri
  - formalni dokaz pravilnosti
  - avtomatsko formalno preverjanje

Pravilnost algoritmov

- Intuitivno razumevanje
  - vpogled v algoritem
  - nepopolno, subjektivno
  - podvrženo človeškim zmotam
  - sled algoritma je lahko v pomoč

Množenje s prištevanjem?

Preberi x in y
$p \leftarrow 0$
do
    $p \leftarrow p - y$
    $x \leftarrow x + 1$
while x > 0
Izpiši p

Pravilnost algoritmov

- Testni primeri
  - pravilnost pokažemo le za testne primere
  - lažje pokazati **nepravilnost** kot pravilnost
  - dober nabor testnih primerov
    - večja zanesljivost testiranja
    - uporaba robnih primerov
  - potrebna po izvajanju algoritma:
    - preverjanje sledi in izhoda
    - simulacija na papir / dejansko izvajanje
  - popolno testiranje je praktično nemogoče
    - nepopolnost in neobvladljivost
    - praktično nemogoče preveriti vseh možne vhode

Pravilnost algoritmov

- Testni primeri
  - zgled: največji skupni delitelj

```
int gcd(int a, int b) {
   if (b == 0) return a;
   return gcd(b, a % b);
}
```

Koliko let bi trajalo testiranje,
če vsako sekundo
preizkusimo milijardo (10^9) vhodov?

  - razumevanje problema
    - kaj je vhod?
    - koliko je različnih vhodov?

Vhod: dve 32-bitni števili

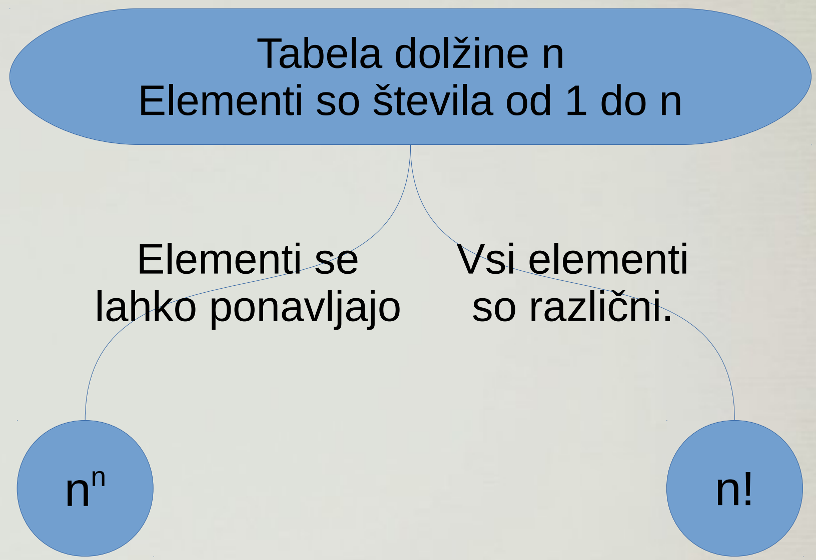
$$2^{32} \cdot 2^{32} = 2^{64} \approx 1{,}8 \cdot 10^{19}$$

Pravilnost algoritmov

- Testni primeri
  - zgled: urejanja seznama

Navadno izbiranje

```
fun selectionSort(a) is
    for i = 0 to n – 2 do
        m = i
        for j = i + 1 to n - 1 do
            if a[j] < a[m] then m = j
        swap(a, i, m)
    endfor
end
```



Pravilnost algoritmov

- Formalni dokaz pravilnosti
  - zanesljivost pravilnosti
  - zahtevnost dokazovanja
  - dokaz je lahko daljši
    kot algoritem
  - uporaba matematike
  - vsak »resen« algoritem
    mora imeti
    formalen dokaz pravilnosti

Pravilnost algoritmov

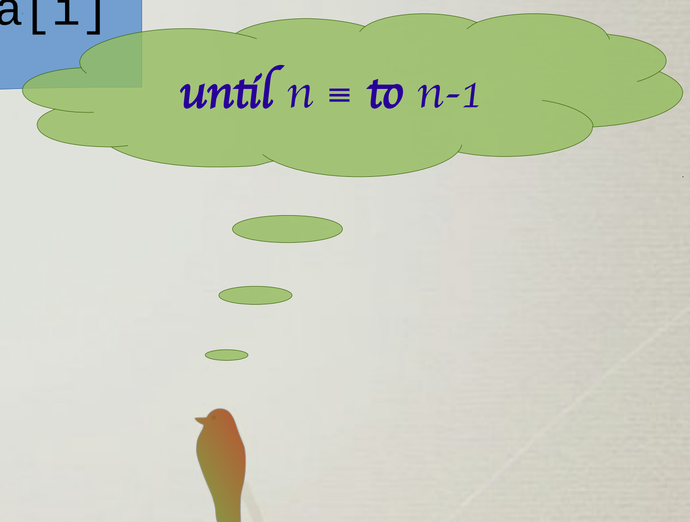
- Formalni dokaz pravilnosti
  - indukcija
    - metoda za dokazovanje neke trditve za vse n
      - po velikosti vhoda, po strukturi vhoda
    - hipoteza
      - trditev, ki jo želimo dokazati
    - induktivna predpostavka
      - predpostavka, da trditev velja za n
    - osnovni primer
      - dokaz trditve za nek majhen primer, npr. $n=0$
    - induktivni korak
      - iz majhnega primera na večjega, npr. $n \rightarrow n+1$

Pravilnost algoritmov

- Formalni dokaz pravilnosti
  - primer

Iskanje maksimuma v tabeli

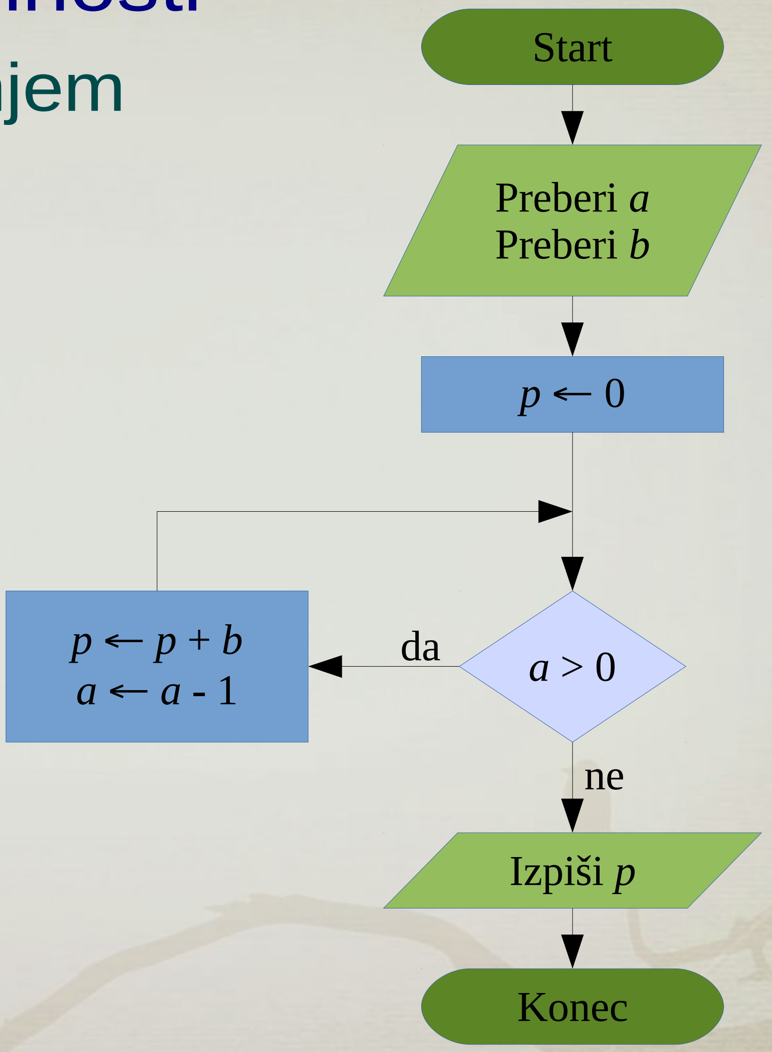
```
max = a[0]
for i = 1 until n do
    if a[i] > max then max = a[i]
```



  - dokaz pravilnosti
    - zančna invarianta
    - indukcija po dolžini tabele

Pravilnost algoritmov

- Formalni dokaz pravilnosti
  - množenje s prištevanjem



Pravilnost algoritmov

- Formalni dokaz pravilnosti
  - Evklidov algoritem
    - Neposredno uporabimo izrek
    - $\gcd(a, b) = \gcd(a \bmod b, b)$


```
fun gcd(a, b) is
   if b == 0 then return a
   return gcd(b, a % b)
```

Pravilnost algoritmov

- Avtomatsko formalno preverjanje
  - temelji na klasičnem formalnem dokazovanju
  - zahtevnost programiranja
  - dokaz pravilnost preko tipov
    - s tipi je moč računati
  - Curry-Howardova enakovrednost
    - tipi so izjave
    - programi so dokazi

```
module Stack

%default total

data Stack : List a -> Type where
  StNil : Stack Nil {a}
  StCons : {xs : List a} -> (x : a) ->  Stack xs -> Stack (x :: xs)

drop : {x : a} -> {xs : List a} -> Stack (x :: xs) -> Stack xs
drop (StCons y ys) = ys

peek : {x : a} -> {xs : List a} -> Stack (x :: xs) -> (y : a ** y = x)
peek (StCons y ys) = (y ** refl)

pop : {x : a} -> {xs : List a} -> Stack (x :: xs) -> (p : (a, Stack xs) ** (fst p) = x)
pop (StCons y ys) = ((y, ys) ** refl)

push : {xs : List a} -> (x : a) -> Stack xs -> Stack (x :: xs)
push x xs = StCons x xs

dup : {x : a} -> {xs : List a} -> Stack (x :: xs) -> Stack (x :: x :: xs)
dup (StCons y ys) = StCons y (StCons y ys)

swap : {x : a} -> {z : a} -> {xs : List a} -> Stack (x :: z :: xs) -> Stack (z :: x :: xs)
swap (StCons y (StCons w ws)) = (StCons w (StCons y ws))

isEmpty : {l : List a} -> Stack l -> Bool
isEmpty StNil = True
isEmpty (StCons x y) = False
```

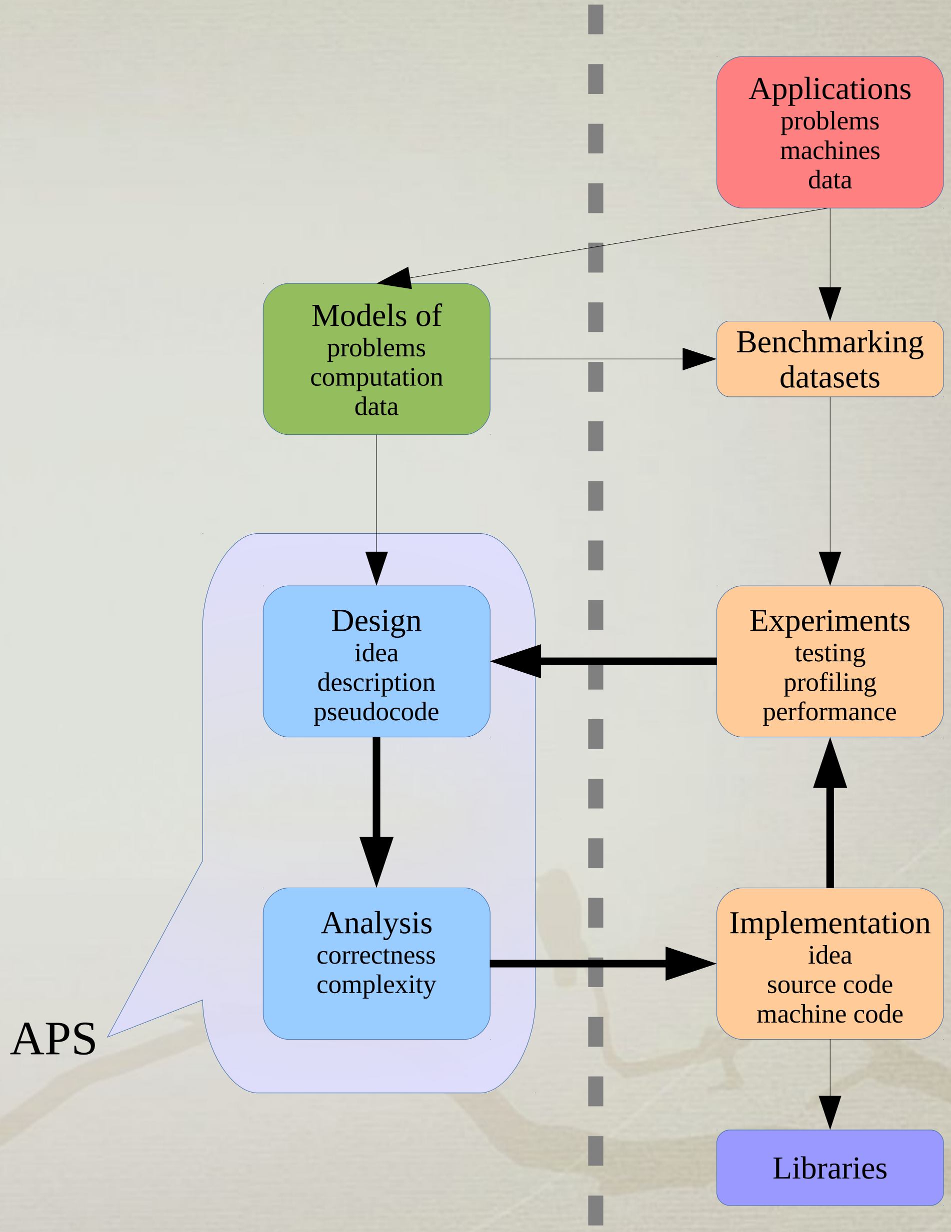
Vir: Blaž Repas, Preverjanje pravilnosti programov z odvisnimi tipi v programskem jeziku Idris

Zahtevnost algoritmov

- Zahtevnost algoritmov
  - Analiza algoritmov, teorija algoritmov
  - Katere vire potrebuje algoritem?
  - Koliko virov algoritem potrebuje?
  - Viri:
    - realni čas, št. korakov, št. operacij, poraba pomnilnika, št. odprtih datotek, poraba električne energije, itd.
  - Porabo virov navadno le ocenimo.
  - Ocena temelji na modelu računanja.

Algoritmika

- Veda o algoritmih
- Področja
  - Teorija algoritmov
  - Analiza algoritmov
  - Računska zahtevnost
  - Izračunljivost
  - Inženiring algoritmov
  - Inženiring programov
  - itd.



Povzetek

- Algoritem, računski problem
  - naloga, rešitev, vrste problemov
- Snovanje algoritmov
  - opis algoritma, metode snovanja
- Sled algoritma
- Pravilnost algoritmov
  - intuitivno, testiranje, formalno, avtomatsko
- Zahtevnost algoritmov
  - analiza zahtevnosti
- Implementacija algoritmov
  - rekurzija, eksperimenti