

Algoritmi

19. oktober 2012



This work is licensed under a Creative Commons
Attribution-NonCommercial-ShareAlike 3.0
Unported License

Poglavje 1

Podatkovne strukture

1.1 Binarno drevo

Binarno drevo

```
1  from math import *
2
3  class Node:
4      def __init__(self, val, l = None, r = None, parent = None):
5          self.val, self.l, self.r, self.parent = val, l, r, parent
6
7      def insert(self, val):
8          cmp = val-self.val
9          if cmp < 0:
10             if self.l is None:
11                 self.l = Node(val, parent = self)
12             else:
13                 self.l.insert(val)
14         elif cmp > 0:
15             if self.r is None:
16                 self.r = Node(val, parent = self)
17             else:
18                 self.r.insert(val)
19         else:
20             return
21
22     def contains(self, val):
23         cmp = val-self.val
24         if cmp < 0:
25             if self.l is None:
26                 return False
27             else:
28                 return self.l.contains(val)
29         elif cmp > 0:
30             if self.r is None:
31                 return False
32             else:
33                 return self.r.contains(val)
34         else:
35             return True
36
37     def getRoot(self):
38         return self if self.parent is None else self.parent.getRoot()
39
40     def getDepth(self, d = 0):
41         return max(
42             d if self.l is None else self.l.getDepth(d+1),
43             d if self.r is None else self.r.getDepth(d+1)
44         )
45
46     def getElts(self, out = []):
47         out.append(self.val)
48         if not self.l is None: out = getElts(self.l, out)
49         if not self.r is None: out = getElts(self.r, out)
50         return out
51
52     def __str__(self):
53         return '(' +
54             ('_' if self.l is None else str(self.l)) +
55             ',' +
56             str(self.val) +
57             ',' +
58             ('_' if self.r is None else str(self.r)) +
59             ')'
```

1.2 Ulomki

Ulomki

```
1 from fractions import Fraction
2 Fraction('3.1415926535897932').limit_denominator(1000)
```

Poglavje 2

Teorija števil

2.1 Praštevila

Preprosti algoritem za preverjanje praštevilstosti

```
1 def getPrimes(limit, primes = [2,3,7]):
2     '''Memoized prime generator'''
3     i = primes[-1] + 1
4     while i < limit:
5         for prime in primes:
6             if prime > i/2: continue
7             if i % prime == 0:
8                 break
9             else: primes.append(i)
10            i += 1
11
12     return primes
```

2.2 Največji skupni delitelj in najmanjši skupni večkratnik

2.2.1 Evklidov algoritem

Evklidov algoritem za iskanje največjega skupnega delitelja

```
1 def gcd(a,b):
2     while b: a,b = b,a%b
3     return a
4
5 def lcm(a,b):
6     return a*b / gcd(a,b)
```

2.3 Fibonaccijevo zaporedje

Algoritem za izračun n -tega števila Fibonaccijevega zaporedja

```
1 #red 2
2 memo = {0:0, 1:1}
3 def fib(n):
4     if not n in memo:
5         memo[n] = fib(n-1) + fib(n-2)
6     return memo[n]
7
8 #red k
9 memok = {}
10 def fib(n,k):
11     if not k in memok:
```

```

12     memok[k] = dict(zip(range(k), [0]*(k-1)+[1]))
13     if not n in memok[k]:
14         memok[k][n] = sum([fib(n-i-1, k) for i in range(k)])
15     return memok[k][n]

```

Poglavje 3

Iskalni algoritmi

3.1 Binarno iskanje

Algoritem binarnega iskanja oz. bisekcije v urejeni tabeli

```

1 from bisect import bisect_left
2
3 def binary_search(a, x, lo=0, hi=None):
4     hi = hi if hi is not None else len(a)
5     pos = bisect_left(a, x, lo, hi)
6     return (pos if pos != hi and a[pos] == x else -1)

```

Poglavje 4

Grafi

Graf formalno podamo kot dvojček vozlišč V in povezav E med vozlišči: $G = \langle V, E \rangle$.

Navadno uporabimo enega od naslednjih zapisov grafov:

- **Seznam sosednosti** – za vsako vozlišče hranimo seznam povezanih vozlišč (omogoča hrambo dodatnih podatkov v vozliščih)
- **Seznam pojavnosti** – za vsako vozlišče hranimo seznam njegovih povezav in za vsako povezavo njen seznam vozlišč (omogoča hrambo dodatnih podatkov v vozliščih in na povezavah)
- **Matrika sosednosti** – kvadratna matrika vozlišč, kjer $M_{i,j} \neq 0$ pomeni povezavo med vozliščema v_i in v_j . (omogoča hrambo ene vrednosti na povezavah)
- **Matrika pojavnosti** – matrika vozlišč, kjer $M_{i,j} \neq 0$ pomeni da ima vozlišče v_i povezavo e_j . (omogoča hrambo ene vrednosti na obeh koncih povezave)

4.1 Eulerjev in Hamiltonov graf

4.1.1 Eulerjev graf

Graf je Eulerjev, kadar vsebuje Eulerjev cikel – tak sprehod, ki vsako povezavo uporabi natanko enkrat in se na koncu vrne v izhodišče.

Pseudokoda Fleuryjevega algoritma za iskanje Eulerjevega cikla

```
0 izberi začetno vozlišče
1 prečkaj poljubno povezavo in jo odstrani (tu most prečkamo le če ni boljše izbire)
2 če je bil odstranjen most, odstrani vse točke ki so ostale izolirane
3 če je še kaka povezava, pojdi na 1
4 sicer zaključi algoritem.
```

4.1.2 Hamiltonov graf

Graf je Hamiltonov, kadar vsebuje Hamiltonov cikel – tak sprehod, ki vsako vozlišče obišče natanko enkrat in se na koncu vrne v izhodišče.

4.2 Iskanje maksimalnega pretoka skozi graf

Pseudokoda Ford-Fulkersonovega algoritma za iskanje maksimalnega pretoka skozi graf

```
0 vse pretoke nastavi na 0
1 označi izvirno vozlišče
2 iz označenih vzemi poljubno vozlišče in ga zaznamuj kot obiskanega
3 označi vse neobiskane sosedo do katerih obstaja nezasičena povezava
4 če je povečanje (ali zmanjšanje) toka po povezavi po kateri smo prišli v trenutno vozlišče ←
  manjše od shranjenega, ga shrani
5 če obstajajo označena vozlišča in ponor še ni označen, pojdi na 2
6 če je ponor označen, pojdi po poti od ponora nazaj, popravi pretoke s shranjenim; pojdi na 1
7 če ni več označenih vozlišč, zaključi algoritem.
```

4.3 Topološko urejanje

Pseudokoda algoritma za topološko urejanje grafa

```
0 naredi delovno kopijo grafa
1 poišči vozlišče brez vhodnih povezav
2 če takega vozlišča ni, zaključi algoritem z napako. (odkrili smo cikel)
3 sicer vozlišče označi z naslednjo oznako in ga odstrani iz grafa
4 če obstaja še kako vozlišče, pojdi na 1
5 sicer prenesi oznake na prvotni graf in zaključi algoritem.
```

4.4 Iskanje najcenejših poti

4.4.1 Dijkstraev algoritem

Pseudokoda Dijkstovega algoritma za iskanje najcenejših poti

```
0 izberi začetno vozlišče, označi ga kot obiskanega in nastavi njegovo trenutno razdaljo na 0
1 ostala vozlišča označi kot neobiskana, nastavi trenutne razdalje na neskončno in jih dodaj v ←
  vrsto neobiskanih
2 izračunaj dolžine poti od izbranega vozlišča do neobiskanih sosedov
3 če je dobljena dolžina do sosedu kje manjša od že znane, jo shrani
4 izbrano vozlišče označi kot obiskano in ga odstrani iz vrste neobiskanih (razdalja do tu je ←
  že minimalna)
5 če vrsta neobiskanih ni prazna, izberi vozlišče z najmanjšo trenutno razdaljo in pojdi na 3
6 sicer zaključi algoritem.
```

Poglavje 5

Random stuff

5.1 Reševanje enačb

Algoritem za reševanje linearnih enačb

```
1 def solve(eq,var='x'):  
2     eq1 = eq.replace("=", "-( "+") "  
3     c = eval(eq1,{var:1j})  
4     return -c.real/c.imag
```