Algoritmi

9. april 2012

Kazalo

Poglavje 1

Podatkovne strukture

1.1 Binarno drevo

Binarno drevo

```
class Node:
    def __init__(self,data,left=None,right=None):
        self.data=data
        self.left=left
        self.right=right
```

1.2 Ulomki

Ulomki

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

Poglavje 2

Teorija stevil

2.1 Praštevila

Preprosti algoritem za preverjanje praštevilskosti

```
def primes(n):
    def prime(i, primes):
        for prime in primeSet:
            if not (i == prime or i % prime):
                return False
        primes.add(i)
        return i
    primeSet = set([2])
    i, p = 2, 0
    while True:
        if prime(i, primeSet):
            p += 1
            if p == n:
                return primeSet
        i += 1
```

2.2 Največji skupni delitelj in najmanjši skupni večkratnik

2.2.1 Evklidov algoritem

Evklidov algoritem za iskanje največjega skupnega delitelja

```
def gcd(a,b):
    while b: a,b = b,a%b
    return a

def lcm(a,b):
    return a*b / gcd(a,b)
```

2.3 Fibonaccijevo zaporedje

Algoritem za izračun n-tega števila Fibonaccijevega zaporedja

```
#red 2
memo = {0:0, 1:1}
def fib(n):
    if not n in memo:
        memo[n] = fib(n-1) + fib(n-2)
    return memo[n]

#red k
memok = {}
def fib(n,k):
    if not k in memok:
        memok[k] = dict(zip(range(k), [0]*(k-1)+[1]))
    if not n in memok[k]:
        memok[k][n] = sum([fib(n-i-1, k) for i in range(k)])
    return memok[k][n]
```

Poglavje 3

Iskalni algoritmi

3.1 Binarno iskanje

Algoritem binarnega iskanja oz. bisekcije v urejeni tabeli

```
from bisect import bisect_left

def binary_search(a, x, lo=0, hi=None):
    hi = hi if hi is not None else len(a)
    pos = bisect_left(a,x,lo,hi)
    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.

Psevdokoda 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

Psevdokoda Ford-Fulkersonovega algoritma za iskanje maksimalnega pretoka skozi graf

```
0  vse pretoke nastavi na 0
1  označi izvorno vozliče
2  iz označenih vzemi poljubno vozlišče in ga zaznamuj kot obiskanega
3  označi vse neobiskane sosede 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 in popravi pretoke s shranjenim in pojdi na 1
7  če ni več označenih vozlišč, zaključi algoritem.
```

4.3 Topološko urejanje

Psevdokoda 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 Dijsktrov algoritem

Psevdokoda Dijkstovega algoritma za iskanje najcenejših poti

```
0 izberi začetno vozlišče, označi ga kot obiskanjega 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 soseda 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

Računska geometrija

5.1 Najbližji par v množici točk

Preprost algoritem za iskanje najbližjega para točk v množici točk s pometanjem

```
begin@
static int[][] tocke;

private static int[] closestPair() {
    //uredi točke po Y
    sort(tocka);

    int minTocka1 = 0, minTocka2 = 1;
    double minRazd = razdalja(minTocka1, minTocka2);
    //ugotovi najkrajo razdaljo med sosednjimi točkami po dimenziji Y
    //ta korak ni nujen, ampak lahko pohitri računanje, ker prej dobimo
    //boljšo trenutno vrednost za najkrajšo razdaljo
    for(int i = 1; i < tocke.length-1; i++) {
        int t1 = i, t2 = i+1;
        if(((tocke[t2][1]-tocke[t1][1]) >= minRazd)
        ||(Math.abs(tocke[t2][0]-tocke[t1][0]) >= minRazd)) continue;

        double tmpRazd = razdalja(t1, t2);
        if(tmpRazd < minRazd) {
            minRazd = tmpRazd;
            minTocka1 = t1;
            minTocka2 = t2;
        }
    }

    //preišče okno točk od t1+1...t2 v katerem so lahko točke,
```

```
    //ki so bližje od trenutne vrednosti za najkrajšo razdaljo
    int t1 = 0, t2 = 2;
    do {
        do {
            double tmpRazd = razdalja(t1, t2);
            if(tmpRazd < minRazd) {
                minRazd = tmpRazd;
                minTocka1 = t1;
                minTocka2 = t2;
            }
            do t2++; while((t2 < tocke.length)
                &&(Math.abs(tocke[t2][0]-tocke[t1][0]) >= minRazd)
                &&((tocke[t2][1]-tocke[t1][1]) < minRazd));
        } while((t2 < tocke.length)&&((tocke[t2][1]-tocke[t1][1]) < minRazd));
        do {
            t1++;
            t2 = t1+2;
        } while((t2 < tocke.length)&&((tocke[t2][1]-tocke[t1][1]) >= minRazd));
    } while(t2 < tocke.length);

    return new int[]{minTocka1, minTocka2};
}

private static double razdalja(int t1, int t2) {
    return Math.sqrt((long)(tocke[t1][0]-tocke[t2][0])*(tocke[t1][0]-tocke[t2][0])
        + (long)(tocke[t1][1]-tocke[t2][1])*(tocke[t1][1]-tocke[t2][1]));
}
//@end
```

Poglavje 6

Random stuff

6.1 Reševanje enačb

Algoritem za reševanje linearnih enačb

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