Algoritmi

10. november 2011

Kazalo

Poglavje 1

Podatkovne strukture

1.1 Dvojno-povezan seznam

Dvojno-povezan seznam

```
1  class Link<T> {
2      Link<T> p,n; // previous, next
3      T val; // value
4
5      public Link(Link<T> prev, T v, Link<T> next) { p=prev; val = v; n=next; }
6      public Link(T v) { this(null, v, null); }
7
8      public Link remove() { // remove self, return next
9          Link next = n;
10         if(p!=null) p.n = n;
11         if(n!=null) n.p = p;
12         n = null;
13         p = null;
14         return next;
15     }
16
17     public Link move(int k) { // move k places
18         Link<T> out = this;
19         boolean forward = true;
20         if(k<0) { k = -k; forward = false; } // move backwards
21         while(k --> 0 && out != null) if(forward) out = out.n; else out = out.p;
22         return out;
23     }
24
25     public void insert(Link<T> next) { // insert after
26         Link<T> nn = this.n;
27         if(nn!=null) {
28             next.n = nn;
29             nn.p = next;
30         }
31         next.p = this;
32         this.n = next;
33     }
34
35     public static <T> Link<T> createLinks(T... links) { // create links and return first
36         Link<T> first = new Link<T>(links[0]), next=first;
37         for(int i=1; i < links.length; i++) {
38             next.insert(new Link<T>(links[i]));
39             next = next.n;
40         }
41         return first;
42     }
43
44     public String toString() { return val.toString(); }
45 }
```

1.2 Binarno drevo

Binarno drevo

```
1  class Node<T extends Comparable<T>> {
2      Node<T> l,r; // left, right
3      Node<T> parent;
4      T val; // value
```

```
5
6       public Node(Node<T> left, T v, Node<T> right) { l=left; val = v; r=right; }
7       public Node(T v) { this(null, v, null); }
8
9       public void insert(T v) {
10          int cmp = v.compareTo(val);
11          if(cmp==0) {
12              return;
13          } else if(cmp < 0) {
14              if(l!=null) { l.insert(v); } else { l = new Node<T>(v); l.parent = this; }
15          } else {
16              if(r!=null) { r.insert(v); } else { r = new Node<T>(v); r.parent = this; }
17          }
18      }
19      public boolean contains(T v) {
20          int cmp = v.compareTo(val);
21          if(cmp==0) {
22              return true;
23          } else if(cmp < 0) {
24              if(l!=null) { return l.contains(v); } else { return false; }
25          } else {
26              if(r!=null) { return r.contains(v); } else { return false; }
27          }
28      }
29
30      public String getTreeString() {
31          return "("+
32              ((l!=null)?l.getTreeString():"_") + ",␣" +
33              toString() + ",␣" +
34              ((r!=null)?r.getTreeString():"_")
35          +")";
36      }
37
38      public String toString() { return val.toString(); }
39  }
```

1.3 Ulomki

Ulomki

```
1   class Fraction {
2       long num, denom;
3
4       public Fraction(long numerator, long denominator) {
5           num = numerator;
6           denom = denominator;
7           fix();
8       }
9
10      public Fraction negate() {
11          Fraction out = this.clone();
12          out.num = -out.num;
13          out.fix();
14          return out;
15      }
16      public Fraction add(Fraction b) {
17          Fraction ax = this.clone(), bx = b.clone();
18          long LCM = ax.denom*bx.denom / GCD(ax.denom,bx.denom),
19              LCMa = LCM/ax.denom,
20              LCMb = LCM/bx.denom;
21
22          ax.num *= LCMa; ax.denom *= LCMa;
23          bx.num *= LCMb; bx.denom *= LCMb;
24
25          ax.num += bx.num;
26          ax.fix();
27          return ax;
28      }
29      public Fraction add(int x) {
30          Fraction out = this.clone();
31          out.num += out.denom*x;
32          out.fix();
33          return out;
34      }
35      public Fraction remove(Fraction b) {
36          return this.add(b.negate());
37      }
38      public Fraction remove(int x) {
39          return add(-x);
40      }
41      public Fraction mul(long x) {
42          Fraction out = this.clone();
```

```
43          out.num *= x;
44          out.fix();
45          return out;
46      }
47      public Fraction mul(Fraction x) {
48          Fraction out = this.clone();
49          out.num *= x.num;
50          out.denom *= x.denom;
51          out.fix();
52          return out;
53      }
54
55      public void fix() { // an in-place method!
56          if(num==0) denom = 1;
57          if(num<0 && denom<0) {num = -num; denom = -denom; }
58          if(num>0 && denom<0) {num = -num; denom = -denom; }
59      }
60      public Fraction simplify() {
61          long GCD = GCD(num,denom);
62          num/=GCD;
63          denom/=GCD;
64          return this;
65      }
66      public long sign() {
67          if(num==0) return 0;
68          if((num<0 && denom<0) || (num>0 && denom>0)) return 1;
69          return -1;
70      }
71      public long wholePart() { return num/denom; }
72
73      @Override
74      public Fraction clone() { return new Fraction(num, denom); }
75
76      @Override
77      public String toString() { return num + "/" + denom; }
78
79      public int compareTo(Fraction frac) {
80          long t = this.num * frac.denom;
81          long f = frac.num * this.denom;
82
83          if(t>f) return 1;
84          else if(f>t) return -1;
85          else return 0;
86      }
87
88      public static long GCD(long a, long b) {
89          return BigInteger.valueOf(a).gcd(BigInteger.valueOf(b)).longValue();
90      }
91  }
```

Poglavje 2

Teorija stevil

2.1 Praštevila

Preprosti algoritem za preverjanje praštevilskosti

```
1   public static boolean isPrime(int a) {
2       if(a<=1) return false;
3
4       for(int i=2; i<=Math.sqrt(a); i++)
5           if(a%i==0) return false;
6
7       return true;
8   }
9
10  public static boolean isPrime2(int a) {
```

```
11      return BigInteger.valueOf(a).isProbablePrime(5);
12  }
```

2.2 Največji skupni delitelj in najmanjši skupni večkratnik

2.2.1 Evklidov algoritem

Evklidov algoritem za iskanje največjega skupnega delitelja

```
1   public static int GCD(int a, int b) {
2       if(a==0) return b; else
3       if(b==0) return a; else
4       if(a>b) return GCD(b, a%b); else
5       return GCD(a, b%a);
6   }
7
8   public static int GCD2(int a, int b) {
9       return BigInteger.valueOf(a).gcd(BigInteger.valueOf(b)).intValue();
10  }
```

2.2.2 Najmanjši skupni večkratnik

$$\text{lcm(a, b)} = \frac{a * b}{\text{gcd}(a, b)}$$

2.3 Fibonaccijevo zaporedje

Algoritem za izračun prvih n števil Fibonaccijevega zaporedja reda k

```
1   public static int[] Fib(int n, int k) {
2       // tabela vrednosti zaporedja (na začetku 0,0,...,0,1,0,...)
3       int[] out = new int[n];
4       out[k-1]=1;
5
6       // generiramo zaporedje naprej - nov element je vsota zadnjih k elementov
7       for(int i=k; i<n; i++)
8           for(int j=0; j<k; j++)
9               out[i] += out[i-j-1];
10
11      return out;
12  }
```

Algoritem za preverjanje, ali je število del Fibonaccijevega zaporedja reda k

```
1   public static boolean isFib(int a, int k) {
2       if (a==0) return true;
3
4       // tabela zadnjih k vrednosti zaporedja (na začetku 0,0,...,0,1)
5       int[] store = new int[k];
6       store[k-1]=1;
7       // vsota členov
8       int sum=1;
9
10      // kaže na mesto v tabeli, ki ga bomo prepisali
11      int pos=0;
12
13      while(a>sum) {
14          int oldSum = sum;
15          sum = 2*sum-store[pos%k];
16          store[pos%k] = oldSum;
17
18          pos++;
19      }
20
21      return (sum==a); // ali je a člen zaporedja
22  }
```

Poglavje 3

Iskalni algoritmi

3.1 Binarno iskanje

Algoritem binarnega iskanja oz. bisekcije v urejeni tabeli

```
1  public static int binarySearch(int a[], int key) {
2      int left = 0, mid, right = a.length-1;
3
4      while(left<=right) {
5          mid = (left+right)>>>1;            // sredina intervala
6          if(a[mid]<key) left = mid+1; else  // premaknemo spodnjo mejo
7          if(a[mid]>key) right = mid-1; else // premaknemo zgornjo mejo
8          return mid;                        // našli smo iskani element
9      }
10     return -1; // iskanega elementa ni v tabeli
11 }
12
13 public static int binarySearch2(int a[], int key) {
14     return Arrays.binarySearch(a, key);
15 }
```

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

```
1  static int[][] tocke;
2
3  private static int[] closestPair() {
4      //uredi točke po Y
5      sort(tocka);
6
7      int minTocka1 = 0, minTocka2 = 1;
8      double minRazd = razdalja(minTocka1, minTocka2);
9      //ugotovi najkrajo razdaljo med sosednjimi točkami po dimenziji Y
10     //ta korak ni nujen, ampak lahko pohitri računanje, ker prej dobimo
11     //boljšo trenutno vrednost za najkrajšo razdaljo
12     for(int i = 1; i < tocke.length-1; i++) {
13         int t1 = i, t2 = i+1;
14         if(((tocke[t2][1]-tocke[t1][1]) >= minRazd)
15         ||(Math.abs(tocke[t2][0]-tocke[t1][0]) >= minRazd)) continue;
16
17         double tmpRazd = razdalja(t1, t2);
18         if(tmpRazd < minRazd) {
19             minRazd = tmpRazd;
20             minTocka1 = t1;
21             minTocka2 = t2;
22         }
23     }
24
25     //preišče okno točk od t1+1...t2 v katerem so lahko točke,
26     //ki so bližje od trenutne vrednosti za najkrajšo razdaljo
27     int t1 = 0, t2 = 2;
28     do {
29         do {
30             double tmpRazd = razdalja(t1, t2);
31             if(tmpRazd < minRazd) {
32                 minRazd = tmpRazd;
33                 minTocka1 = t1;
34                 minTocka2 = t2;
35             }
36             do t2++; while((t2 < tocke.length)
37                 &&(Math.abs(tocke[t2][0]-tocke[t1][0]) >= minRazd)
38                 &&((tocke[t2][1]-tocke[t1][1]) < minRazd));
39         } while((t2 < tocke.length)&&((tocke[t2][1]-tocke[t1][1]) < minRazd));
40         do {
41             t1++;
42             t2 = t1+2;
43         } while((t2 < tocke.length)&&((tocke[t2][1]-tocke[t1][1]) >= minRazd));
44     } while(t2 < tocke.length);
45
46     return new int[]{minTocka1, minTocka2};
47 }
48
49 private static double razdalja(int t1, int t2) {
50     return Math.sqrt((long)(tocke[t1][0]-tocke[t2][0])*(tocke[t1][0]-tocke[t2][0])
51         + (long)(tocke[t1][1]-tocke[t2][1])*(tocke[t1][1]-tocke[t2][1]));
52 }
```

Poglavje 6

Random stuff

6.1 Možne vsote podmnožic

Algoritem za izračun vseh možnih vsot podmnožic neke množice

```
1  public static boolean[] subsetSums(int[] a) {
2      int M = 0;
3      for(int i : a) M += i;
4      boolean[] m = new boolean[M+1];
5      m[0]=true;
6
7      for(int i=0; i<a.length; i++)
8          for(int j=M; j>=a[i]; j--)
9              m[j] |= m[j-a[i]];
10
11     return m;
12 }
```

6.2 Hanojski stolpi

Rekurzivni algoritem za reševanje igre Hanojski stolpi

```
1  public static void hanoi(int n) { hanoi(n,'a','b','c'); }
2  private static void hanoi(int n, char source, char dest, char by) {
3      if(n==1) {
4          System.out.println(source+"␣->␣"+dest);
5      } else {
6          hanoi(n-1, source, by,   dest);
7          hanoi(1,   source, dest, by);
8          hanoi(n-1, by,     dest, source);
9      }
10 }
```

6.3 CRC varnostno kodiranje

Algoritem za izračun CRC varnostnih bitov

```
1      public static boolean[] CRC(boolean[] bits, boolean[] poly) {
2          // ustvarimo dovolj veliko tabelo za izračun in na začetek prilepimo vhodne bite
3          boolean[] tmpbits = new boolean[bits.length + poly.length];
4          System.arraycopy(bits, 0, tmpbits, 0, bits.length);
5
6          for(int pos=0; pos<bits.length; pos++) if(tmpbits[pos])
7              for(int i=0; i<poly.length; i++) if(poly[i])
8                  tmpbits[pos+i] = tmpbits[pos+i]^poly[i];
9
10         boolean[] CRCbits = new boolean[poly.length-1];
11         System.arraycopy(tmpbits, bits.length+1, CRCbits, 0, poly.length-1);
12         return CRCbits;
13     }
```

6.4 Levenshteinova razdalja med nizi

Vrste urejanj so:

- brisanje znaka: $d_L(\text{"}aaa\text{"},\text{"}aa\text{"}) = 1$
- vstavljanje znaka: $d_L(\text{"}aaa\text{"},\text{"}aaaa\text{"}) = 1$
- spremembo znaka: $d_L(\text{"}aaa\text{"},\text{"}aba\text{"}) = 1$

Algoritem za izračun Levensteinove razdalje med dvema nizoma

```
1  public static int levensteinDistance(String s1, String s2) {
2      int[][] d = new int[s1.length()+1][s2.length()+1];
3
4      for(int i=0; i<=s1.length(); i++) d[i][0] = i;
5      for(int j=0; j<=s2.length(); j++) d[0][j] = j;
6
7      for(int i=1; i<=s1.length(); i++) {
8          for(int j=1; j<=s2.length(); j++) {
9
10             int del = d[i-1][j]+1, // brisanje
11                 ins = d[i][j-1]+1, // vstavljanje
12                 sub = d[i-1][j-1]; // sprememba
13
14             if(s1.charAt(i-1) != s2.charAt(j-1)) sub++;
15
16             // shrani najmanjšo izmed razdalj
17             d[i][j] = Math.min(del, Math.min(ins, sub));
18         }
19     }
20
21     return d[s1.length()][s2.length()];
22 }
```