Greining Reiknirita

TÖL403G

Skilaverkefni 3

Verkefnishöfundar:
Guðmundur Már Gunnarsson
Skarphéðinn Þórðarsson
Sigurður Skúli Sigurgeirsson

Kennari:
Páll Melsted

6. apríl 2014

1 Keyrsla og virkni

Öll forritun fór fram í Java og notast við java version 1.6.0_27 fyrir Windows við keyrslu og þýðingu þess. Til þess að þýða forritið þarf einungis að sækja skjalið "IntervalTree.javaǫg þýða það með "javac IntervalTree.java"Forritið er aðgengilegt á https://github.com/gudmundurmar/SKSKG Þar er einnig að finna Stopwatch.java sem notast var við til þess að taka timann á forritinu en einnig uppgefin inntök inntök og úttök s1-s3 sem notast var við til þess að keyra forritið líkt og gefið var upp í verkefnislýsingunni.
Þýðing: `$javac IntervalTree.java`
Keyrsla: `$java IntervalTree < s1.in | diff -w s1.out - | wc -l`

2 Uppbygging

Tréð er byggt upp þannig að fyrsta bilið verður rótin. Ef næsta inntak skarast ekki við bilið í nóðunni og neðri mörk bilsins eru stærri en efri mörk rótarinnar fer það hægra megin, annars fer það vinstra megin í tréð. Ef bilið skarast aftur á móti við rótina fer það í tengdann lista hjá rótinni. Það sama á við um rótina og aðrar nóður í trénu. Ef bilið skarast bætist það við nóðuna í tengdum lista. Þessi lausn hefur þann vankant að ef við byrjum á því að bæta stórum bilum í tréð getum við lent í því að tréð okkar verði bara einn tengdur listi. Upprunalega vildum við notast við median bilsins til þess að raða upp í tréð en þar sem að við fengum það ekki til þess að virka rétt, vildum við frekar hafa rétta lausn sem tæki mögulega aðeins lengri tíma frekar en ranga lausn sem tæki minni tíma. Við hugsuðum líka um kosti og galla þess að notast við rauð-svört tré til þess að halda leitartíma í lágmarki en af því að við erum að við vitum ekki hvort við erum að leita oftar en við bætum í tréð og eyðum úr því, gátum við ekki sagt til um hvort við myndum græða meira á því eða hreinlega tapa á því að innleiða rautt-svart tré í lausnina okkar.

2.1 Yfirklasinn - IntervalTree

```
public class IntervalTree {

  static ArrayList<int[]> closedIntervalList = new ArrayList<int[]>();
  Node root; //rót trésins

  //FG: Lokuðu bilin eru geymd í tré sem hefur rót í root.
  //Öll bil sem prenta skal út eru geymd í closedIntervalList

  public IntervalTree() { root = null;  }

  //Notkun: tree.insert(a,b);
  //Fyrir: a og b eru heiltölur, a < b
  //Eftir: búið er að bæta bilinu [a,b] í tréð
  public void insert(int a, int b){}

  //Notkun: tree.delete(a,b);
  //Fyrir: a og b eru heiltölur, a <= b
  //Eftir: Ef [a,b] var í trénu þá er búið að eyða því
  public void delete(int a, int b){}

  //Notkun: tree.intersects(a,b,root);
  //Fyrir: a og b eru heiltölur, a <= b, root er nóða
  //Eftir: búið er að finna öll bil sem skarast á við [a,b] í trénu með rótina root
  public int intersects(int a, int b, Node node){}
```

```
  //Notkun: tree.intersects(a,b);
  //Fyrir: a og b eru heiltölur, a <= b
  //Eftir: búið er að finna og prenta út öll bil sem skarast á við bilið [a,b]
  public void intersects(int a, int b) {}

  //Notkun: tree.contains(a,b,root);
  //Fyrir: a og b eru heiltölur, a <= b, root er nóða
  //Eftir: búið er að finna öll bil sem innihalda[a,b] í trénu með rótina root
  public boolean contains(int a, int b, Node node){}

  //Notkun: tree.contains(a,b);
  //Fyrir: a og b eru heiltölur, a <= b, root er nóða
  //Eftir: búið er að finna og prenta öll bil sem innihalda[a,b]
  public void contains(int a, int b){}

  //Notkun: tree.point(a);
  //Fyrir: a er heiltala
  //Eftir: búið er að finna og prenta öll bil sem innihalda a
  public void point(int a){}

  //Notkun: printFoundIntervals()
  //Fyrir:  Ekkert
  //Eftir:  Búið er að prenta öll þau bil sem leitað var að
  public void printFoundIntervals(){}

  //Notkun: tree.deleteNode(node)
  //Fyrir: node er nóða
  //Eftir: Búið er að fjarlægja node úr trénu
  public void deleteNode(Node node){}

  //Notkun: tree.query(q);
  //Fyrir: q er strengur sem inniheldur fyrirspurn fyrir Interval tréð
  //Eftir: Búið er að framkvæma fyrirspurnina í q.
  public void query(String query) {}

  //nóðurnar í trénu
  static class Node
  { /*Sjá næstu földunarhæð hér að neðan*/  }

  public static void main(String[] args)
  {
    //Býr til nýtt tré sem það síðan
    //byggir upp með inntaki sem lesið er inn af standard input
  }
}
```

2.2 Fyrsta földun - Node

```
//nóðurnar í trénu
  static class Node
  {
    int lower; //neðri mörk
    int higher; //efri mörk
```

```
    Node left; //vinstra barn
    Node right; //hægra barn
    Node parent; //foreldri
    Link intervals; //bilin sem skarast á við bilið í nóðunni

    //FG: Lokuðu bilin í nóðunni eru geymd í tengda listanum intervals, hægra barn
    //nóðurnar er í right og vinstra barn í left. Foreldri nóðunnar er parent.
    //lower er lægri endi fyrsta lokaða bilsins sem var sett í nóðuna og higher
    //er hærri endi lokaðs bilsins.

    static class Link
    {
      //Sjá næstu földun í skjalinu
    }

    //Notkun: node.insertInterval(a,b);
    //Fyrir: a og b eru heiltölur, a < b
    //Eftir: búið er að setja bilið [a,b] á réttan stað í intervals
    void insertInterval(int a, int b){}

    //Notkun: node.findIntersections(a,b);
    //Fyrir: a og b eru heiltölur, a < b
    //Eftir: búið er að finna öll bil sem skarast á við bilið [a,b] í nóðunni node
    int findIntersections(int a,int b){}

    //Notkun: node.findContains(a,b);
    //Fyrir: a og b eru heiltölur, a < b
    //Eftir: búið er að finna öll bil sem innihalda [a,b] í nóðunni node
    boolean findContains(int a,int b){}

    //Notkun: node.deleteInterval(a,b);
        //Fyrir: a og b eru heiltölur, a < b
        //Eftir: Búið er að fjarlægja lokaða bilið [a,b] ef það var geymt í nóðunni.
    void deleteInterval(int a, int b){}

  }
```

2.3 Önnur földun - Link

```
static class Link
    {
      Link next;
      int lower;
      int higher;

      //FG: next bendir á næsta hlekk í tengda listanum, lower er lægri endi lokaðs bils
      // og higher er hærri endi lokaðs bils.

      //Notkun: link.compareTo(a,b);
      //Fyrir: a og b eru heiltölur, a < b
      //Eftir: Skilar 1 ef [lower,higher] < [a,b], 0 ef þau eru jöfn og -1 annars
      int compareTo(int a, int b){}
    }
```

3 Forritið í heild

```
# -*- coding: cp1252 -*-
from priority_dict import priority_dict
from bisect import bisect_left
import time
import sys

class verk3:
    def __init__(self):
        self.notSpan = []
        self.Span = []
        self.wholeNet = []
        self.tree = []

    def inputToDict(self,filename):
        dict = {}

        length = int(sys.stdin.readline())
        file = sys.stdin.readlines()
        for i in range(0, length):
            dict[str(i)] = [float("inf"), None, []]
        for line in file:

            split = line.split(" ")
            dict[split[0]].append([split[1], split[2]])
            dict[split[1]].append([split[0], split[2]])
            self.wholeNet.append([int(split[2]),split[0],split[1], False])

        self.wholeNet.sort()
        weight = self.MSTPRIM(dict,0, dict['0']);
        print weight
        #print dict
        self.NotMinPRIM(dict, weight)




    def MSTPRIM(self,G,w,r):

        r[0] = 0

        Q =  priority_dict(G);
        while Q:
            Q._rebuild_heap()

            u = Q.pop_smallest()

            if not(G[u][1] is None):
                if(int(u) < int(G[u][1])):
                    self.Span.append([int(G[u][0]), u,G[u][1], False])
                    #notSpan[:] = (name for name in notSpan if name != [int(G[u][0]), u,G[u][1]])
```

```
                    #notSpan.remove([int(G[u][0]), u,G[u][1]])
                else:
                    self.Span.append([int(G[u][0]), G[u][1], u, False])
                    #notSpan[:] = (name for name in notSpan if name != [int(G[u][0]), G[u][1], u])
                    #notSpan.remove([int(G[u][0]), G[u][1], u])
            w += int(G[u][0])

            self.tree.append(u)

            for v in range(3,len(G[u])):
                ver = G[u][v]
                if ver[0] in Q:
                    if float(ver[1]) < float(Q[ver[0]][0]):
                        Q[ver[0]][1] = u
                        G[ver[0]][1] = u

                        if float(G[u][0]) == float(0):
                            G[u][0] = float(ver[1])

                        #setur nyja besta
                        Q[ver[0]][0] = float(ver[1])
                        G[ver[0]][0] = float(ver[1])


        for i in range(1,len(self.tree)):
            G[str(G[self.tree[i]][1])][2].append(self.tree[i])

        return w


    def NotMinPRIM(self,G,w):

        self.Span.sort()
        for e in self.Span:
            res = binary_search(self.wholeNet, e)
            if not(res == -1):
                self.wholeNet[res][3] = True


        for e in range(0,len(self.wholeNet)):
            cur = len(self.wholeNet)-1-e
            if not(self.wholeNet[cur][3]):
                self.notSpan.append(self.wholeNet[cur])


        cnt = 0
        nrSecond = 1
        weight = 0
        shortest = {}


        for i in range(1,len(self.tree)):

```

```
104             node = self.tree[i]
105 
106             context = doubleBFS(G, node, G[node][1])
107 
108             for j in range(0,len(self.notSpan)):
109                 cur = len(self.notSpan)-1-j
110                 if self.notSpan[cur][1] in context and self.notSpan[cur][2] in context:
111                     continue
112                 elif self.notSpan[cur][1] in context or self.notSpan[cur][2] in context:
113                     weight = w-G[node][0]+self.notSpan[cur][0]
114                     break
115 
116             if(int(node) < int(G[node][1])):
117                 shortest[cnt] = [int(node), int(G[node][1]), int(weight)]
118             else:
119                 shortest[cnt] = [int(G[node][1]), int(node), int(weight)]
120             cnt += 1
121 
122 
123         Q =  priority_dict(shortest);
124 
125         while Q:
126             u = Q.smallest()
127             print str(Q[u][0])+" "+str(Q[u][1])+" "+str(Q[u][2])
128             u = Q.pop_smallest()
129 
130 def doubleBFS(G,u,v):
131     contextU = [u]
132     contextV = [v]
133 
134     treeU = [u]
135     treeV = [v]
136 
137     while treeU and treeV:
138         curU = treeU.pop()
139         for adjU in G[curU][2]:
140             treeU.append(adjU)
141             contextU.append(adjU)
142         curV = treeV.pop()
143         for adjV in G[curV][2]:
144             treeV.append(adjV)
145             contextV.append(adjV)
146 
147 
148     if len(contextU) < len(contextV):
149         return contextU
150     else:
151         return contextV
152 
153 
154 def binary_search(a, x, lo=0, hi=None):   # can't use a to specify default for hi
155     hi = hi if hi is not None else len(a) # hi defaults to len(a)
```

```
157     pos = bisect_left(a,x,lo,hi)           # find insertion position
158     return (pos if pos != hi and a[pos] == x else -1) # don't walk off the end
159
160
161 if __name__ == '__main__':
162     start_time = time.time()
163     V3 = verk3()
164     V3.inputToDict("10k.in")
165     #print time.time() - start_time, "seconds"
```