Greining reiknirita: Verkefni 3

Kennari: Páll Melsted

Bjarki Geir Benediktsson Bjarni Jens Kristinsson Tandri Gauksson

Skil: 6. apríl 2014

1 Lýsing á reikniritinu

Við byggðum lausn okkar á klasanum `KruskalMST.java`[1] sem er að finna á heimasíðu bókarinnar *Algorithms* eftir Sedgewick og Wayne. Smiður klasans tekur inn óstefnt vigtað net $G = (V,\ E,\ W)$ og notar reiknirit Kruskals til að finna minnsta spanntré mst og vigt þess w.

Við skrifuðum nýja aðferð `secondMSTweight( EdgeWeightedGraph G, Edge e )` sem að finnur vigt minnsta spanntré netsins $G_e = (V,\ E \setminus \{e\},\ W)$. Í `keyrsluforrit.java` finnum við fyrst mst og prentum vigt þess. Síðan tökum við hvern legg $e \in mst$ (raðað í vaxandi röð eftir fyrra hniti e) og finnum minnsta spanntré netsins G_e og prentum legginn e og vigt þess w_e á staðalúttak.

Við útfærðum `secondMSTweight( EdgeWeightedGraph G, Edge e )` svipað og smiðinn fyrir `KruskalMST`. Í staðinn fyrir að fara línulega í gegn um alla leggi V og kanna hvort þeir tengja saman samhengisþætti í skóginum fyrir netið þá byggjum við upp `UF` hlutinn með öllum leggjunum úr $mst \setminus \{e\}$. Þetta skilur okkur eftir með tvo samhengisþætti, eða tvö tré í skóginum, og þá förum við í vaxandi röð í gegn um E og könnum hvort að leggurinn sé e og ef ekki hvort hann tengir saman samhengisþættina tvo. Ef hann gerir það þá erum við komin með minnsta spannandi hluttré fyrir G_e.

`KruskalMST.java` notast við eftirfarandi klasa frá sömu höfundum (við eyddum öllum main() aðferðum úr klösunum en aðrar breytingar eru taldar upp á eftir hverjum og einum klasa):

- `EdgeWeightedGraph.java`[2]
  - Breyttum öllum línum sem að búa til nýjan hlut af taginu `Edge` til að nota heiltöluvigt í stað fleytitölu (sjá athugasemdir við `Edge.java`)
- `Queue.java`[3]
- `Edge.java`[4]
  - Breyttum vigtinni yfir í að vera heiltala í stað fleytitölu
  - toString() aðferðin gefur örlítið öðru vísi strengjaframsetningu á hnútunum
  - Bættum við toString2() til að prenta út hnút og vigt eins og er í .out skránum sem Páll gefur upp á heimasíðunni
  - Bættum við equals() aðferð
  - Bættum við klasann lexiCompare sem ber saman hnúta eftir orðabókaröðun
- `MinPQ.java`[5]
- `UF.java`[6]

[1] `http://algs4.cs.princeton.edu/43mst/KruskalMST.java.html`
[2] `http://algs4.cs.princeton.edu/43mst/EdgeWeightedGraph.java.html`
[3] `http://algs4.cs.princeton.edu/43mst/Queue.java.html`
[4] `http://algs4.cs.princeton.edu/43mst/Edge.java.html`
[5] `http://algs4.cs.princeton.edu/24pq/MinPQ.java.html`
[6] `http://algs4.cs.princeton.edu/15uf/UF.java.html`

- `Bag.java`[7]
- `Stack.java`[8]

2 Tímaflækja lausnarinnar

Hér er G upphaflega netið sem `KruskalMST` hluturinn er smíðaður með, V fjöldi hnúta þess og E fjöldi leggja.

Að búa til hlut af taginu `KruskalMST` tekur $O(E \log V)$ tíma skv. bókinni. Hluturinn er minnsta spanntré G og inniheldur $V-1$ leggi. Við köllum því $\Theta(V)$ sinnum á `secondMSTweight` og greinum aðferðina svona:

1. Fyrst þarf að smíða pq af taginu `MinPQ`. Innsetning er innistæðubundið $O(\log n)$ svo smíði og innsetning á E leggjum tekur $O(E \log E)$.
2. Að smíða hlut uf af taginu `UF` tekur V tíma. Að sameina samhengisdæmi uf tekur $O(\log V)$ tíma en það er gert V sinnum. Þessi partur forritsins tekur því $O(V + V \log V)$.
3. Þegar farið er í gegn um pq og hann tæmdur þarf í versta falli að skoða alla leggina eða E talsins.

Nú er fjöldi leggja E í mesta lagi $(V-1) + (V-2) + ... + 1 = \frac{(V-1)(V-2)}{2}$ þ.a. $E = O(V^2)$. Keyrslutími `secondMSTweight` er því skv. ofantöldu

$$O(E \log E + V + V \log V + E) = O(E \log V^2 + V + E \log V + E) = O(E \log V).$$

Keyrslutími `keyrsluforrit.java` er $T_{Kruskal} + (V-1)T_{secondMSTweight}$ eða

$$O(E \log V + (V-1) \cdot E \log V) = O(EV \log V) = O(E^{3/2} \log V).$$

[7] `http://algs4.cs.princeton.edu/13stacks/Bag.java.html`
[8] `http://algs4.cs.princeton.edu/13stacks/Stack.java.html`

3 Java kóði

3.1 KruskalMST.java

```
public class KruskalMST{
    // weight er summa e.weight() yfir öll e í mst.
    private int weight = 0;
    private Queue<Edge> mst = new Queue<Edge>();


    //~ Notkun: KruskalMST KMST = new KruskalMST(G);
    //~ Fyrir: G er hlutur af taginu EdgeWeightedGraph.
    //~ Eftir: KMST er minnsta spannandi tré fyrir G
    //      (minnsti spannandi skógur ef G er ekki samanhangandi).
    public KruskalMST(EdgeWeightedGraph G) {
        // Raða leggjum G eftir vigt.
        MinPQ<Edge> pq = new MinPQ<Edge>();
        for (Edge e : G.edges()) {
            pq.insert(e);
        }

        UF uf = new UF(G.V());
        while (!pq.isEmpty() && mst.size() < G.V() - 1) {
            // uf inniheldur samhengisþætti nets af stærð G.V()
            // með mengi leggja = mst.
            Edge e = pq.delMin();
            int v = e.either();
            int w = e.other(v);
            if (!uf.connected(v, w)) {
                uf.union(v, w);
                mst.enqueue(e);
                weight += e.weight();
            }
        }
    }

    //~ Notkun: w = KMST.secondMSTweight();
    //~ Fyrir: KMST inniheldur d og er minnsta spanntré fyrir G.
    //~ Eftir: w er vigt minnsta spannandi trés (eða skógar)
    //      netsins G þegar leggurinn d hefur verið fjarlægður.
    public int secondMSTweight( EdgeWeightedGraph G, Edge d ){
        // Raða leggjum G eftir vigt.
        MinPQ<Edge> pq = new MinPQ<Edge>();
        for (Edge e : G.edges()) {
            pq.insert(e);
        }

        // Finn samhengisþætti KMST án leggsins d.
        UF uf = new UF(G.V());
        for (Edge e : this.edges()) {
            if(e.equals(d)) continue;
            int v = e.either();
            int w = e.other(v);
            uf.union(v,w);
```

```
        }

        // Finn þann legg í G, annan en d, sem tengir saman
        // samhengisþættina tvo og hefur minnsta vigt.
        while (!pq.isEmpty()) {
            Edge e = pq.delMin();
            if(e.equals(d)) continue;
            int v = e.either();
            int w = e.other(v);
            if (!uf.connected(v, w)) {
                return weight - d.weight() + e.weight();
            }
        }
        // d er eini leggurinn í G sem tengir samhengisþættina.
        return weight - d.weight();
    }

    //~ Notkun: Iterable<Edge> E = KMST.edges();
    //~ Fyrir: KMST er hlutur af taginu KruskalMST.
    //~ Eftir: E inniheldur öll stök mst.
    public Iterable<Edge> edges() {
        return mst;
    }

    //~ Notkun: Iterable<Edge> E = KMST.sortedEdges();
    //~ Fyrir: KMST er hlutur af taginu KruskalMST.
    //~ Eftir: E inniheldur öll stök mst og er raðað vaxandi eftir fyrri hnút leggsins.
    public Iterable<Edge> sortedEdges() {
        MinPQ<Edge> sorted = new MinPQ<Edge>(new Edge.lexiCompare());
        for (Edge e : mst) {
            sorted.insert(e);
        }
        return sorted;
    }

    //~ Notkun: w = KMST.weight();
    //~ Fyrir: KMST er hlutur af taginu KruskalMST
    //~ Eftir: w er heildarvigt leggjanna í KMST.
    public int weight() {
        return weight;
    }
}
```

3.2 keyrsluforrit.java

```
public class keyrsluforrit {
    public static void main(String[] args) {
        //~ Skanni sem les inn tölurnar
        java.util.Scanner scanner = new java.util.Scanner(System.in);
        int n = Integer.parseInt(scanner.nextLine()); //vitum að fyrsta línan er bara ein tala
        EdgeWeightedGraph EWG = new EdgeWeightedGraph(n);

        //~ F: Búið er að lesa inn eina línu af staðalinntaki.
        while (scanner.hasNext()) {
```

```
            //~ I: Búið er að lesa inn eina eða fleiri línur af staðalinntaki
            //~    og gera viðeigandi aðgerðir með þær.
            String line = scanner.nextLine(); //tekur inntak fram að næsta enter
            String[] S = line.split(" "); //fylki af orðunum í línunni (skipt eftir bilum)
            EWG.addEdge( new Edge(  Integer.parseInt(S[0]),
                                    Integer.parseInt(S[1]),
                                    Integer.parseInt(S[2]) ));
        }
        //~ E: Búið er að lesa inn allar línur af staðalinntaki
        //~    og gera viðeigandi aðgerðir með þær.

        KruskalMST MST = new KruskalMST(EWG);
        System.out.println(MST.weight());
        for (Edge e : MST.sortedEdges()) {
            System.out.println(e.toString2() + " " + MST.secondMSTweight(EWG,e));
        }
    }
}
```

3.3 Edge.java

```
    //~ Notkun: b = e.equals(d);
    //~ Fyrir:
    //~ Eftir: b == true þþaa e og d tengi saman sömu hnúta og hafi sömu vigt.
    public boolean equals(Edge that) {
        if(this.compareTo(that) != 0) return false;
        int t = that.either();
        int u = that.other(t);
        return (t == v && u == w) || (t == w && u == v);

    }

    //~ Notkun:
    //~ Fyrir:
    //~ Eftir:
    public static class lexiCompare implements Comparator<Edge> {
        public int compare(Edge e1, Edge e2){
        int fyrra1 = e1.either();
        int fyrra2 = e2.either();
        if (Integer.compare(fyrra1,fyrra2)==0)
            return Integer.compare(e1.other(fyrra1),e2.other(fyrra2));
        else
            return Integer.compare(fyrra1,fyrra2);
        }
    }


    /**
     * Returns a string representation of the edge.
     * @return a string representation of the edge
     */
    public String toString() {
        return v+" "+w+" "+weight;
    }
```

```
/**
 * Returns a string representation of the edge.
 * @return a string representation of the edge
 */
public String toString2() {
    return v+" "+w;
}
```