Rapport för A-labb

Equilibrium Mobile

Axel Rooth

# Uppgiftsbeskrivning

Uppgiften var att undersöka om en skulptur var i jämvikt eller inte. Skulpturen består av ett antal stavar som hänger under varandra horisontellt via trådar. Längst ner i denna trädstruktur sitter vikter. Dessa vikter kan väga olika mycket så att skulpturen kommer i obalans. Lösningen på detta problem blir därmed att undersöka hur många vikter som ska behövas förändras för att skulpturen ska hamna i jämvikt.

# Algoritmbeskrivning

Algoritmen går ut på att undersöka hur stor varje vikt är i trädet. För att parsa inputsträngen läggs varje symbol och nummer in i en kö. Denna kö gås sedan igenom med hjälp av syntaxen för trädets struktur i form av en rekursiv algoritm, där funktionerna heter readExpression och readWeight. När en symbol hittas kallas readExpression och när ett numeriskt värde kallas readWeight. I readWeight undersöker man om nästkommande symbol även är ett tal i en loop. Om detta är fallet så adderas detta värde på strängen och tas bort från kön. När detta har gjort kallas append2dict för att först beräkna den totala vikten skulle bli om trädet var i jämvikt med denna specifika vikt. Detta totalvärde sparas sedan som en nyckel i en dict och nyckelns värde adderas med 1. Det vill säga, om det visar sig att totalvärdet redan finns i dict kommer värdet på nyckeln bli större. När strängen är slut returneras antalet vikter som uppkommit samt dict:en med nycklar med olika totalvikter. Sedan används pythons inbyggda max() funktion för att hitta den totalvikt som har störst värde. Detta värde tas och subtraheras med det totala antal vikter (totala antal vikter – max\_matchningar) vilket blir hur många vikter man minst måste ändra på för att systemet ska vara i jämvikt.

# Testdata

|  |  |
| --- | --- |
| Indata | Utdata |
| 3  [[3,7],6]  40  [[2,3],[4,5]] | 1  0  3 |

# Tidskomplexitet

Först och främst sätts varje symbol till en kö. Tiden det tar att göra denna procedur är O(n), där n är antalet symboler. Vidare gås varje symbol igenom i den rekursiva algoritmen vilket även detta tar O(n) tid. Att lägga till totalvikten i en dict tar O(1). Däremot kan totalvikten ta lång tid att beräkna då vikternas värde kunde vara upp emot 109 vilket kan försvåra kattis minneshantering något. Pythons inbyggda metod max() är extremt effektiv samt att ta ut en nyckel från en dict tar även det O(1). Resultatet blir därmed att algoritmens tidskomplexitet blir O(2n) vilket slutligen kan sammanfattas som O(n).

# Appendix – kod







