DYPL - uppgift 3

Linus Harling
Nathalie Sandström

5 juni 2009

1 Algoritm - 12h

Vi gick igenom ett antal olika varianter, implementerade i pseudokod, anteckningar och Java. Från rent ”raka” lösningar utan rekursion, något som resulterade i en fantastisk mängd kod och invecklade `for` satser. Till åtminstone två varianter på rekursiva funktioner, med olika angreppssätt angående vilka parametrar som skulle skickas vidare. Hanteringen av datat gick dessutom igenom ett antal olika iterationer innan en handledare gav oss en hint om hur HashMap kunde utnyttjas. Efter det var det bara en fråga om att hitta rätt vinkel och klia sig i huvudet tillräckligt mycket för att få grepp om hur man skulle göra.

Programmetstrukturen bevarades i båda implementationerna, trots att man antagligen skulle kunna lösa uppgiften mer optimalt om man anpassat sig mer till språket. Ordlistan och listan över telefonnummer läses in och lagras i en `ArrayList` i Java och en `list` i Python. Ordlistan översätts sedan till en `HashMap` respektive `dict`, där nyckeln är ordet och värdet översättningen till siffror. I efterhand har vi fått höra att det kan vara mer effektivt att utnyttja översättningen som nyckel och lagra orden i värdet. Detta eftersom man då slipper en uppslagning för att göra själva jämförelsen.

2 Java - 4h

Java kändes ”Tryggare” eftersom det är mer traditionellt och på grund av vanan vid språket. Det var naturligt att börja implementera i Java eftersom mindre tankekraft behövde läggas på programkonstruktioner. Vanan vid språket gjorde att arbetet med att förstå rekursionen gick ganska lätt i jämförelse med om vi hade suttit med ett mer obekant språk.

Mycket väl dokumenterat API, väldigt lätt att hitta funktioner som är användbara. Samtidigt kan det vara lite överväldigande, eftersom varje typ har en helt ohygglig mängd funktioner för att klara av samma saker Python har inbyygt.

Hantering av typer kändes bökigare än i Python (naturligtvis), med fler konverteringar, exempelvis `Integer.parseInt`, `String.valueOf` etc. En version av implementationen hade till exempel följande mindre läsbara rader:

```
combos[Integer.parseInt(String.valueOf(number.charAt(index)))]
...
str.contains((new Character(word.charAt(index))).toString())
```

Mer naturlig variabel- och metodhantering; self.¡något¿ känns lätt onaturligt efter att man vant sig vid Javas och C:s syntax. Java har dock lite bökigare programkontroll: Eftersom Python är ett skript-språk är det bara skriva som man skulle gjort i terminalen eller interpretatorn. Inget `static void main()` etc.

3 Python - 2h

Själva implementationen av algoritmen gick snabbare, delvis på grund av att vi stött på de flesta problemen innan. Vi visste med andra ord precis vad vi skulle skriva. Pythons hantering av fil-inläsning, avsaknad av kontrolltecken som `{ }` med vänner hjälpte naturligtvis också. Loopar, iteratorer, hantering av datasamlingar som dictionaries och maps väldigt mycket behagligare än Java. Överhuvudtaget fick vi intrycket att Python är betydligt naturligare att programmera i. Java har ett antal mycket ”stiliga” abstraktioner, saker som tilltalar på ett tekniskt plan. Exempel är konceptet med strömmar, att all inläsning och utskrift sker via strömmar är både snyggt och smart, men mängden kod som behöver skrivas för att utnyttja dem blir lätt frustrerande. Inläsning av filer kräver nästan dubbelt så mycket kod i Java som i Python, mycket på grund av just strömmarna. Överhuvudtaget märks det ganska tydligt att Java är utvecklat mot en starkt tekniskt fokuserad bakgrund, allt är väl planerat och strukturerat men kanske inte så programmerarvänligt. Jämför:

```
def loadFromFile(self):
wordFile = open("newdict.txt")
wordLine = wordFile.readline().lower().strip()
while wordLine:
self.wordsToNumber(wordLine)
wordLine = wordFile.readline().lower().strip()
wordFile.close()
```

Med:

```
private ArrayList<String> words = new ArrayList<String>();

public void readFromFile(){
String str = "";
try{
BufferedReader br = new BufferedReader(new FileReader("newdict.txt"));
while ((str = br.readLine()) != null){
words.add(str.toLowerCase());
numberRepresentation.put(str.toLowerCase(), wordsToNumbers(str.toLowerCase()));
}
}
catch(FileNotFoundException fileNotFoundException){
System.out.println("File not found");
}
catch(IOException e){
System.out.println(e);
}
}
```

Å andra sidan är Pythons API väldigt mycket sämre dokumenterat än Javas, google är en större vän än organisationen i den officiella dokumentationen. Delvis uppvägs detta av `dir()` och `help()` men inte fullständigt. Exekveringsmiljön, d.v.s. den interaktiva interpretatorn i Python, gör dock att ”trial and error” går betydligt smidigare än i Java vilket också gör API:t mindre livsnödvändigt.

Vi fick känslan av att vi hade kunnat komma på fler varianter på implementationer med Python än med Java, även om det inte var något vi utnyttjade då vi bara översatte koden rakt av. Överhuvudtaget kändes Python mer dynamiskt och inbjudande till lek, inte som Java helt resultatinriktat.