Week8 Notes

Knapsack Problem

- Definition: Given weights and values of n items, put these items in a knapsack with capacity W to get the maximum total value in the knapsack.
- Formulation:

$$x_i = copies\ of\ each\ kind\ of\ item$$
$$v_i = value$$
$$w_i = weight$$
$$W = maximum\ weight\ capacity$$
$$i = items\ numbered\ 1..n$$

$$maximize \sum_{i=1}^{n} v_i\, x_i$$
$$subject\ to \sum_{i=1}^{n} w_i\, x_i \leq W, x_i \in \{0,1\}$$

Source: The Optimization Expert

- Combinatorial Optimization: Discrete (countable) number of feasible solutions.
- Combinatorial optimization problems are usually computational challenging!
- For more details for geeks about computational complexity, wikipedia page is a good start: https://www.wikiwand.com/en/Computational_complexity
- Also you can check the following lecture: https://www.youtube.com/watch?v=mr1FMrwi6Ew&ab_channel=MITOpenCourseWare

Reading from File

- In real life, pulling data from excel or txt is a weak practice. Generally database and json format are used since they are more structured!
- What is database? Check Oracle web-page: https://www.oracle.com/database/what-is-database/
- In this lecture, we will cover raw text file: We will use csv (comma-separated values) but reading from txt works the same way too.
- Buffered Reader - FileReader can be utilized for reading purposes.
- Syntax:

```
import java.io.FileReader;
import java.io.BufferedReader;

OR

import java.io.* >> imports everything in java.io!

try {
    FileReader -filereader_name- = new FileReader(“-data-“);  // You will give path to the data. Full path is not needed if the file is in the same folder with the class file.
    BufferedReader -bufferedreader_name- = new BufferedReader(-filereader_name-);  // Compiler can complain since FileReader not initialized.

    String -line_name- = -bufferedreader_name-.readLine(); >> Read entire line as String
    String[ ] -array_name- =  -line_name-.split(“-split_char-“); >>" split" "given line of string" into elements  according to given “split_char"

} catch (FileNotFoundException e) {
    System.out.println(e); // If your data.csv file can not
}catch (IOException e) {
    System.out.println(e); //

}
```

Structuring the Code

- When you are dealing with complex structures, you should write your code “modular”.
- A Heuristic Solution for the Knapsack Problem:
    - Define value/weight for every item.
    - Sort them in a descending order.
    - Start from the first item (i.e. biggest in terms of value per weight)
    - Include item if there is enough capacity, otherwise check the next item in sorted array.
- What is heuristic approach?
    - Although it can find optimal solution, you can not guarantee that the solution is optimal.
- Next week, we are going to solve same problem with a commercial solver.

Use of HashMap

- Syntax:
- HashMap<-datatype-,-datatype-> -hashmap_name- = new HashMap<-datatype-,-

datatype->();

- Functions:
    - .put(key, value) >> add element to the HashMap
    - .get(key) >> get elements with given key.
    - .keySet() >> returns all keys. Note that hashmap.keySet() can be iterated (i.e. for each loop)
    - .values() >> returns all values. Note that, hashmap.values() can be iterated (i.e. for each loop)
    - .remove(key) >> remove given key and associated value.
    - .clear() >> clear all key-value pairs of HashMap
    - .size() >> returns size of the HashMap