Algorithms Assignment Five - Dynamic Programming (graphs) and Greedy Algorithms (spice)

charles.schmitz2@marist.edu

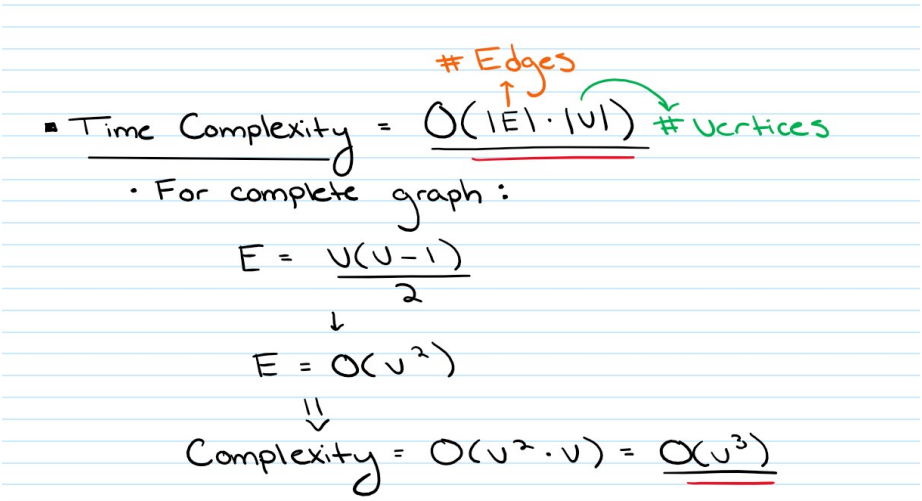
December 2020

---

https://github.com/charlesschmitz2/CMPT435—Algorithms/tree/Assignment-Five—Dynamic-Programming-(graphs)-and-Greedy-Algorithms-(spice)

1 Bellman Ford

The time complexity of this algorithm is dependent on the number of edges and the number of vertices. This time around we are working with weighted and directed graphs but this does not affect the asymptotic running time of these graphs all that much. Since it is dependent on the number of edges/vertices, the time complexity is $O(|V||E|)$ where V is the number of vertices and E is the number of Edges.
Furthermore, if the graph happens to be a complete graph then rather than using $|E|$ within the time complexity calculations, $|V|^2$ is used. So this would make the complexity of a complete graph must incorporate $O(|V|^2)$. All together this would become $O(|V|^3)$.
Below is a diagram that displays this complexity :



2 Knapsack Problem

There are two general approach's to solving the Knapsack problem (as well as many more but two popular ones). A recursive approach where you consider all subsets of items and calculate the total weight of all subsets. Then consider the only subsets whose total weight is smaller than the capacity where you pick the maximum value subset. In this method the item is either included in the solution subset or not. This method does have flaws though due to using a recursive tree. Mainly that the same subsets could be evaluated more than once. This adds to the time complexity. Moreover, this makes the time complexity of this recursive method is $O(2^n)$.
The next approach (which is more relevant to this assignment) is the Dynamic approach. This approach and is typically implemented using a matrix where all possible weights from 1 to the capacity are considered. This matrix is then used to compare and determine the maximum of two possibilities in order to fill the current capacity. This makes the complexity of this approach $O(N * W)$ where W is the capacity and N is the number of weight elements.
There are many other approaches to this problem but the dynamic solution provides a optimal time complexity as it is linear while recursion is exponential. In my solution I chose to implement my own unique approach which honestly may not be as efficient but still maintains a relatively low run time. It's main downfall is the amount of loops and iterations through these loops needed to reach the end goal. While this is relatively quick sample sizes. This complexity can grow very quickly if a very large sample size is used. Lastly, the dynamic solution requires solving $O(nS)$ sub-problems and the solution to one depends on the other sub-problems as well so the solution can be found as fast as $O(1)$ in some cases.

3 Main Class

```
import java.io.*;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.*;

public class Assignment5_DynamicProgramming_GreedyAlgorithms {

    public static int vertexCount = 0;
    public static int startVertex = 0;
    public static ArrayList<Integer> vertexListALL = new ArrayList<
    Integer>();
    public static int edgeCount = 0;
    public static ArrayList<Integer> edgeListALL = new ArrayList<
    Integer>();
    public static Graph graph = new Graph(0, 0);

    public static void main(String[] args) {

    /*----BellmanFord Single Source Shortest Path Algorithm----*/
        readAndProcess("graphs2.txt");

        /*
        //-----FOR SOME REASON WORKS WHEN HARDCODING IN BUT CANT
    SEEM TO GET TO WORK WHEN LOOPING EVEN THOUGH MY BELLMAN FORD
    FUNCTION SEEMS CORRECT
                                    int V = 5; // Number of
    vertices in graph
                                    int E = 8; // Number of edges
    in graph
                                    graph = new Graph(V, E);
                                    // add edge 0-1
                                    graph.edgeArray[0].source = 0;
                                    graph.edgeArray[0].destination
    = 1;
                                    graph.edgeArray[0].weight = -1;
                                    // add edge 0-2
                                    graph.edgeArray[1].source = 0;
                                    graph.edgeArray[1].destination
    = 2;
                                    graph.edgeArray[1].weight = 4;
                                    // add edge 1-2
                                    graph.edgeArray[2].source = 1;
                                    graph.edgeArray[2].destination
    = 2;
                                    graph.edgeArray[2].weight = 3;
                                    // add edge 1-3
                                    graph.edgeArray[3].source = 1;
                                    graph.edgeArray[3].destination
    = 3;
                                    graph.edgeArray[3].weight = 2;
                                    // add edge 1-4
                                    graph.edgeArray[4].source = 1;
```

```
                                    graph.edgeArray[4].destination
    = 4;
                                    graph.edgeArray[4].weight = 2;
                                    // add edge 3-2
                                    graph.edgeArray[5].source = 3;
                                    graph.edgeArray[5].destination
    = 2;
                                    graph.edgeArray[5].weight = 5;
                                    // add edge 3-1
                                    graph.edgeArray[6].source = 3;
                                    graph.edgeArray[6].destination
    = 1;
                                    graph.edgeArray[6].weight = 1;
                                    // add edge 4-3
                                    graph.edgeArray[7].source = 4;
                                    graph.edgeArray[7].destination
    = 3;
                                    graph.edgeArray[7].weight = -3;
                                    graph.bellmanFord(graph, 0);
                                    int Ver = 4;
                                    int edg = 8;
                                    graph = new Graph(Ver, edg);
                                    int j = 0;
                                    while(j < edg){
                                        System.out.println("--"+j);
                                            int randomNum =
    ThreadLocalRandom.current().nextInt(0, 50 + 1);
                                        graph.edgeArray[j].source =
     randomNum;
                                        System.out.println(
    randomNum);
                                            int randomNum1 =
    ThreadLocalRandom.current().nextInt(0, 50 + 1);
                                        graph.edgeArray[j].
    destination = randomNum1;
                                        System.out.println(
    randomNum1);
                                            int randomNum2 =
    ThreadLocalRandom.current().nextInt(0, 50 + 1);
                                        graph.edgeArray[j].weight =
     randomNum2;
                                        System.out.println(
    randomNum2);
                                        j++;
                                    }
                                    System.out.print("DONE");
                                    graph.bellmanFord(graph, 0);
         */


        /*----Knapsack Problem----*/
        //spice name = red;    total_price =  4.0;  qty = 4;
        //spice name = green;  total_price = 12.0;  qty = 6;
        //spice name = blue;   total_price = 40.0;  qty = 8;
        //spice name = orange; total_price = 18.0;  qty = 2;
        /*List<KnapsackItem> items = new ArrayList<>(); //
    Initialize Variables
```

```
        Knapsack knapsack = new Knapsack(items,0);
            knapsack.addItem("red", 4.0, 4, 4.0/4);//Add the items to the knapsack
            knapsack.addItem("green", 12.0, 6, 12.0/6);
            knapsack.addItem("blue", 40.0, 8, 40.0/8);
            knapsack.addItem("orange", 18.0, 2, 18.0/2);
            knapsack.sort(); //sort the items
                                //calculate the solution based on the capacity provided
        knapsack.print();
        Knapsack knapsackSolution = knapsack.findWorth(21);
        knapsackSolution.print();

         */

        parseAndSoveSpiceProblem("spice.txt");









    }//main

    /*----BellmanFord Single Source Shortest Path Algorithm Functions----*/

        /*----This function is responsible for the bellman ford functions and processing the input of the graph2.txt file----*/
        private static void readAndProcess(String s) {
            try {
                // Open the file
                FileInputStream fstream = new FileInputStream(s);
                BufferedReader br = new BufferedReader(new InputStreamReader(fstream));

                String strLine;
                String graphName = " ";

                System.out.println("PROCESSING FILE " + "'"+s+"'" + "[");
                //Read File Line By Line
                while ((strLine = br.readLine()) != null) {
                    // Print the content on the console
                    //System.out.println(strLine);
                    String[] words = strLine.split(" ");
                    ArrayList<Integer> edgeList = new ArrayList<Integer>();
                    ArrayList<Integer> vertexList = new ArrayList<Integer>();
                    for (int i = 0; i < words.length; i++){
                        if(isInteger(words[i]) && strLine.contains("add")){
```

```
                            System.out.println("\t\t\u2022 '" +
    strLine + "'" + " --> Processing Number -  " + words[i]);
                            if (strLine.contains("vertex")) {
                             //System.out.println("--Adding Vertex"
     + " WORD " + words[i]);
                             vertexList.add(Integer.parseInt(words[
    i]));
                                vertexListALL.add(Integer.parseInt(
    words[i]));
                             vertexCount++;
                            }//if
                            else if (strLine.contains("edge")){
                                //System.out.println("--Adding Edge
    " + " " + words[i] + " ----- " + i);
                                edgeList.add(Integer.parseInt(words
    [i]));
                                    edgeListALL.add(Integer.
    parseInt(words[i]));
                                edgeCount++;

                            }//else if

                        }
                        else if (strLine.contains("--")){
                            //these are comments so do nothing
                            if (strLine.contains("directed") ||
    strLine.contains("CLRS")){
                                graphName = strLine;
                            }//if
                        }//else if
                        else if (strLine.contains("new") && i == 0)
    {
                            System.out.println("\n\n\t" + "
    GENERATING NEW GRAPH " + graphName);
                            vertexCount = 0;
                            edgeCount = 0;


                        }//else if
                        else if (strLine.trim().isEmpty()){
                            //System.out.println("BlankLine");
                            int vertices = vertexCount; // Number
    of vertices in graph
                            int edges = edgeCount/3; // Number of
    edges in graph divided by 3 because each time count is
    incremented it counts the weight value so this takes care of
    that.
                            //System.out.println(vertices);
                            //System.out.println(edges);
                            System.out.println("List of all
    Vertices : " + vertexListALL + "   VertexCount = " +
    vertexCount);
                            System.out.println("List of all Edges (
    w/ weights): " + edgeListALL + "   EdgeCount = " + edgeCount/3)
    ;
                            Graph graph = new Graph(vertices, edges
    );
```

```
172                             /* for (int y = startVertex; y <
    edgeListALL.size();y++){
173                                 //int edge1 = edgeListALL.get(0);
174                                 System.out.println(edgeListALL.get
    (0));
175                                     edgeListALL.remove(0);
176                                 //graph.edgeArray[y].source = edge1
    ;
177                                 //int edge2 = edgeListALL.get(0);
178                                 System.out.println(edgeListALL.get
    (0));
179                                     edgeListALL.remove(0);
180                                 //graph.edgeArray[y].destination =
    edge2;
181                                 //int weight = edgeListALL.get(0);
182                                 System.out.println(edgeListALL.get
    (0));
183                                     edgeListALL.remove(0);
184                                 //graph.edgeArray[y].weight =
    weight;
185                             }
186                             */
187                             /*
188                             int y = startVertex;
189                             while(!vertexListALL.isEmpty()){
190                                 int edge1 = edgeListALL.get(0);
191                                 graph.edgeArray[y].source = edge1;
192                                 edgeListALL.remove(0);
193                                 int edge2 = edgeListALL.get(0);
194                                 graph.edgeArray[y].destination =
    edge2;
195                                 edgeListALL.remove(0);
196                                 int weight = edgeListALL.get(0);
197                                 graph.edgeArray[y].weight = weight;
198                                 edgeListALL.remove(0);
199                                 y++;
200                                 System.out.print(y);
201                             }//while
202                              */
203                             for(int k = 0; k < (edgeCount/3); k++){
204                                 if(!edgeListALL.isEmpty()) {
205                                     int edge1 = edgeListALL.get(0);
206                                     graph.edgeArray[k].source =
    edge1;
207                                     //System.out.println(edge1);
208                                     edgeListALL.remove(0);
209
210                                     int edge2 = edgeListALL.get(0);
211                                     graph.edgeArray[k].destination
    = edge2;
212                                     //System.out.println(edge2);
213                                     edgeListALL.remove(0);
214
215                                     int weight = edgeListALL.get(0)
    ;
216                                     graph.edgeArray[k].weight =
    weight;
```

```
                                    //System.out.println(weight);
                                    edgeListALL.remove(0);
                                }//if
                            }
                            //graph.bellmanFord(graph, startVertex);
                            edgeListALL.clear();
                            vertexListALL.clear();
                        }


                    }//for

                    if(!strLine.contains("--") && !strLine.isEmpty() && !strLine.contains("new")){
                        System.out.println("\t\t\tVertex's Being Added: " + vertexList );
                        if(!vertexList.isEmpty()){
                            //add the vertex
                            //Testing - System.out.println("---- Vertex Count " + vertexCount);

                        }//if
                        System.out.println("\t\t\tEdge's Being Added: " + edgeList);
                        if(!edgeList.isEmpty()){
                            //add the edge, the first and second items in the list are the edge connection and the third is the weight

                        }//if
                    }//if

                }//while

                //Close the input stream
                fstream.close();
                        //need to perform one more graph since my method adds and performs functions on the graphs at each white space and since there is no extra space after the last item this is what I did
                        //a much smarter thing to do would be to simply hit return one more time after the last graph so they all follow the same pattern but for now this will be in here and I feel as though
                        //that's fine because of how the text file is set up and it doesn't make sense to rethink how I read the graphs in just for something so minor
                        int vertices = vertexCount; // Number of vertices in graph
                        int edges = edgeCount/3; // Number of edges in graph divided by 3 because each time count is incremented it counts the weight value so this takes care of that.
                        //System.out.println(vertices);
                        //System.out.println(edges);
                        System.out.println("List of all Vertices : " + vertexListALL + "    VertexCount = " + vertexCount);
```

```
                        System.out.println("List of all Edges (w/
    weights): " + edgeListALL  + "   EdgeCount = " + edgeCount/3);
                        Graph graph = new Graph(vertices, edges);
                        graph.bellmanFord(graph, startVertex);
                System.out.println("\n\n] PROCESSING FILE COMPLETE"
    );

            }//try
            catch (IOException e) {
                e.printStackTrace();
            }//catch

        }//readAndProcess

        public static boolean isInteger(String s) {
            boolean isValidInteger = false;
            try
            {
                Integer.parseInt(s);

                // s is a valid integer

                isValidInteger = true;
            }
            catch (NumberFormatException ex)
            {
                // s is not an integer
            }

            return isValidInteger;
        }//isInteger

    /*----Knapsack Problem----*/

    private static void parseAndSoveSpiceProblem(String fileName) {
        List<KnapsackItem> items = new ArrayList<>(); // Initialize
     Variables
        Knapsack knapsack = new Knapsack(items,0);

        try {
            Path path = Paths.get(fileName);
            Scanner scanner = new Scanner(path);

            System.out.println("PROCESSING FILE " + "'"+fileName+"'
    " + "[");
            //read line by line
            while(scanner.hasNextLine()){
                //process each line
                String line = scanner.nextLine();
                if (!line.isEmpty() && !line.contains("--")){
                    String spiceName = "";
                    double totalPrice = 0;
                    int quantity = 0;
                    int capacity = 0;
                    //System.out.println("\t\u2022 " + line);
                        String words[] = line.split(";");
                        for (int i = 0; i < words.length; i++){
```

```
                                String searchValue = words[i].substring(words[i].lastIndexOf(" ")+1);

                                if (words[i].contains("spice name")){
                                    spiceName = searchValue;
                                    //System.out.println(spiceName);
                                }
                                else if (words[i].contains("total_price")){
                                    totalPrice = Double.parseDouble(searchValue);
                                    //System.out.println(totalPrice);
                                }
                                else if (words[i].contains("qty")){
                                    quantity = Integer.parseInt(searchValue);
                                    //System.out.println(quantity);
                                }
                                else if (words[i].contains("capacity")){
                                    capacity = Integer.parseInt(searchValue);
                                    //System.out.println(capacity);
                                }//else if
                            }//for
                        if (capacity == 0) {
                            knapsack.addItem(spiceName,totalPrice,quantity,totalPrice/quantity);
                            System.out.println("\t\tItem Added to Knapsack : " + "KnapsackItem{" +
                    " spiceName = '" + spiceName + '\'' +
                    ", totalPrice = " + totalPrice +
                    ", quantity = " + quantity +
                    ", unitPrice = " + totalPrice/quantity +
                     '}');
                          }//if
                          else {
                            System.out.println("\n\n----Running with Capacity = " + capacity + "----");
                            knapsack.sort();
                            System.out.println("KNAPSACK CONTENTS BEFORE SOLUTION : ");
                            knapsack.print();
                            //knapsack.print();
                            System.out.println("\nSOLVING : ");
                            Knapsack knapsackSolution = knapsack.findWorth(capacity);
                            System.out.println("SOLUTION : ");
                            knapsackSolution.print();
```

```
                        System.out.println("\nTOTAL ~SPICE~ YOU
    STOLE = " + knapsackSolution.totalWorth());
                    }
                }//if

            }
            scanner.close();

            System.out.println("\n] PROCESSING FILE COMPLETE");
        }//try
        catch (Exception e) {
            e.printStackTrace();
        }


    }



    }//Assignment5_DynamicProgramming_GreedyAlgorithms
```

4 Graph Class

```
public class Graph {
    int vertices;
    int edges;
    Edge edgeArray[];

    Graph (int verticesNum, int edgesNum){
        vertices = verticesNum;
        edges = edgesNum;
        edgeArray = new Edge[edgesNum];

        //initialize the values in the edgeArray to the new edge
        for (int i = 0; i < edgesNum; i++){
            edgeArray[i] = new Edge();
        }//for
    }//Graph constructor

    /*----BellmanFord Algorithm----*/
    public void bellmanFord(Graph graph, int source){
        int vertices = graph.vertices;
        int edges = graph.edges;
        int distance[] = new int[vertices];

        //1.
        for (int i = 0; i < vertices; i++){
            distance[i] = Integer.MAX_VALUE;
        }//for
        distance[source] = 0;

        //2.
        for (int i = 1; i < vertices; i++){
            for (int j = 0; j < edges; j++){
                int src = graph.edgeArray[j].source;
                int dest = graph.edgeArray[j].destination;
                int weight = graph.edgeArray[j].weight;

                if ((distance[src] != Integer.MAX_VALUE) && (
    distance[src]+weight < distance[dest])){
                    distance[dest] = distance[src] + weight;
                }//if
            }//for
        }//for

        //3.
        for (int j = 0; j < edges; j++){
            int src = graph.edgeArray[j].source;
            int dest = graph.edgeArray[j].destination;
            int weight = graph.edgeArray[j].weight;

            if (distance[src] != Integer.MAX_VALUE && distance[src
    ]+weight < distance[dest]){
                System.out.println("There is a negative weight
    cycle within the graph");
                return;
```

```
            }//if
        }//for

        printSolution(distance, vertices);
    }//BellmanFord

    public void printSolution(int[] distance, int vertices) {
        System.out.println("\nVertex Distance from Source w/ Cost
    Analysis: ");
        for (int i = 0; i < vertices; i++){
            System.out.println("\t\t" + i + "\t\t" + distance[i] +
    "\t\t | 0 --> " + i + " - Cost is " + distance[i]);
        }//for

    }//printSolution


}//graph
```

5 Edge Class

```java
public class Edge {
    int source;
    int destination;
    int weight;

    Edge(){
        source = 0;
        destination = 0;
        weight = 0;
    }//edge constructor

}
```

6 Knapsack Class

```
1  import java.util.ArrayList;
2  import java.util.Collections;
3  import java.util.Comparator;
4  import java.util.List;
5
6  public class Knapsack {
7
8      private List<KnapsackItem> knapsackItems;
9      private int knapsackCapacity;
10
11     public Knapsack(List<KnapsackItem> items, int capacity) {
12         knapsackItems = items;
13         knapsackCapacity = capacity;
14     }//knapsack
15
16     public Knapsack findWorth(int capacity) {
17
18         ArrayList<KnapsackItem> solution = new ArrayList<>(); //
       Initialize Variables
19         Knapsack knapsackSolution = new Knapsack(solution,0);
20
21         if (capacity <= 0) {
22             System.out.println("\tUnable to compute worth, '
       Capacity' was listed as a negative value or none was given");
23             return knapsackSolution;
24         }//if
25         if (capacity > this.totalCapacity()){
26             System.out.println("\tUnable to compute worth, '
       Capacity was listed as a value larger than the size of the
       Knapsack");
27             return knapsackSolution;
28         }
29
30             this.setKnapsackCapacity(capacity);
31
32             boolean capacityRemains = true;
33
34             int counter = 0;
35             //int tempQuantity = knapsackItems.get(counter).
       getQuantity();
36             while (capacityRemains) {
37                 if (capacity == 0) {
38                     capacityRemains = false;
39                     //System.out.println("Setting Capacity to false
       ");
40                 }//if
41                 else {
42                     int tempQuantity = knapsackItems.get(counter).
       getQuantity();
43                     while (tempQuantity > 0 && capacity > 0) {
44                         //System.out.println("in tempQuantity > 0 \
       nBefore : ");
45                         //System.out.println(tempQuantity);
```

```
                        //System.out.println(capacity);
                        if (ifExistsInSolution(knapsackItems.get(counter).getSpiceName(), solution)){
                            int temp1 = solution.get(counter).getQuantity();
                            temp1++;
                            solution.get(counter).setQuantity(temp1);
                            System.out.println("\t\u2022 Adding another scoop of " + knapsackItems.get(counter).getSpiceName());
                            //System.out.println("in tempQuantity > 0 if statement");
                        }//if
                        else {
                            //System.out.println("in tempQuantity > 0 else statement");
                            System.out.println("\t\u2022 Adding to Solution Knapsack the first scoop of " + knapsackItems.get(counter).getSpiceName());
                            knapsackSolution.addItem(knapsackItems.get(counter).getSpiceName(), knapsackItems.get(counter).getTotalPrice(), 1, knapsackItems.get(counter).getUnitPrice());
                        }//else\
                        tempQuantity--;
                        capacity--;
                        //System.out.println("After: ");
                        //System.out.println(tempQuantity);
                        //System.out.println(capacity);
                    }//while
                    counter++;
                }//else
                //System.out.println("in Capacity Remains");
            }//while

            return knapsackSolution;


    }//findWorth

    public void addItem(String name, double price, int quanity, double unitPrice){
        KnapsackItem item = new KnapsackItem(name, price, quanity, unitPrice);
        knapsackItems.add(item);
    }//addItem

    public void sort(){
        Collections.sort(knapsackItems, Comparator.comparing(KnapsackItem::getUnitPrice));
        Collections.reverse(knapsackItems);
    }


    public List<KnapsackItem> getKnapsackItems() {
        return knapsackItems;
    }
```

```
89
90      public int getKnapsackCapacity() {
91          return knapsackCapacity;
92      }
93
94      public void setKnapsackCapacity(int knapsackCapacity) {
95          this.knapsackCapacity = knapsackCapacity;
96      }
97
98      public boolean ifExistsInSolution(String name, ArrayList<
        KnapsackItem> solutions){
99          for (int i = 0; i < solutions.size(); i++){
100             if(solutions.get(i).getSpiceName().trim().
        compareToIgnoreCase(name.trim()) == 0) {
101                 return true;
102             }//if
103         }//for
104         return false;
105     }//ifExists
106
107     public int totalCapacity(){
108         int totalCapacity = 0;
109         for (int i = 0; i < knapsackItems.size(); i++){
110             totalCapacity += knapsackItems.get(i).getQuantity();
111         }//for
112         return totalCapacity;
113     }//totalCapacity
114
115     public double totalWorth(){
116         double totalWorth = 0.0;
117         for(int i = 0; i < knapsackItems.size(); i++){
118             totalWorth += knapsackItems.get(i).getUnitPrice()*
        knapsackItems.get(i).getQuantity();
119         }
120         return totalWorth;
121     }//totalWorth
122
123     public void print(){
124         if (!knapsackItems.isEmpty()) {
125             System.out.println("\t-- Knapsack Contents --  ");
126             System.out.println("\t\tCapacity Left : " +
        knapsackCapacity);
127             System.out.println("\t\tItems : ");
128
129             for (int j = 0; j < knapsackItems.size(); j++) {
130                 System.out.println("\t\t\t\u2022" + knapsackItems.
        get(j));
131             }//for
132         }//if
133     }//print
134
135
136 }//Knapsack
```

7 KnapsackItem Class

```
1  public class KnapsackItem {
2
3      //Our spices will have the attributes of
4          //spiceName -- Here being a color name
5          //totalPrice -- Represented in "quatloos"
6          //quantity -- How many "scoops"
7
8      private String spiceName;
9      private double totalPrice;
10     private int quantity;
11     private final double unitPrice;
12
13     public KnapsackItem(String name, double price, int qty, double
       up){
14         spiceName = name;
15         totalPrice = price;
16         quantity = qty;
17
18         unitPrice = up;
19     }//knapsackItem Constructor
20
21     public int getQuantity() {
22         return quantity;
23     }//getQuantity
24
25     public void setQuantity(int quantity) {
26         this.quantity = quantity;
27     }//setQuantity
28
29     public double getTotalPrice() {
30         return totalPrice;
31     }//getTotalPrice
32
33     public void setTotalPrice(int totalPrice) {
34         this.totalPrice = totalPrice;
35     }//setTotalPrice
36
37     public String getSpiceName() {
38         return spiceName;
39     }//getSpiceName
40
41     public void setSpiceName(String spiceName) {
42         this.spiceName = spiceName;
43     }//setSpiceName
44
45     public double getUnitPrice() {
46         return unitPrice;
47     }
48
49
50     @Override
51     public String toString() {
52         return "KnapsackItem{" +
```

```
53                  " spiceName = '" + spiceName + '\'' +
54                  ", totalPrice = " + totalPrice +
55                  ", quantity = " + quantity +
56                  ", unitPrice = " + unitPrice +
57                  '}';
58      }//toString
59  }
```