

# Programming Project I, Second Report

Illya Starikov, Claire Trebing, & Timothy Ott

Due Date: March 16, 2016

## 1 Abstract

As proposed in the first report, we would like to compare the efficiency of different memory management methods via a **0-1 Knapsack**. We discussed the current memory management methods, the different approaches we would like to use (Brute Force, Dynamic Programming, Greedy Solution) and our plan of implementations.

After implementing the pseudocode provided in Report #1, rigorous testing and extensive running of sample of data, we have completed conclusive evidence to our proposed problem.

In this report we would like to discuss our findings.

## 2 Implementation

As we discussed in our first report we decided to go with a more object-oriented approach to this problem. We created a class called `App` to represent each application running on our platform. Each instance of the `App` class contains (as properties<sup>1</sup>) an identifying number, its memory usage, and its cost to recover from storage. These properties are largely randomized in the construction of each instance. For example, every app has a random memory usage in the range of 32 to 1028; the cost is a derived value, taking anywhere from 20 to 50 percent of the memory usage of the `App` in question; we also have a property called `ratio` which consists of the cost property divided by the memory property and is calculated at creation of the instance by the constructor function, this will be used in our Greedy Algorithm; and lastly each `App` is given an identifying number based on the index of the cell in the array in which they are initially placed. This ensures that each `App` will have a unique identifier to help differentiate them whilst checking our solutions against one another.

Initially we considered creating a separate class for our “Smartphone” or the platform on which our `Apps` would be running but later decided against this as this functionality could easily be represented simply by a list (or array) of all the `Apps`, cutting down on the amount of code we would need.

---

<sup>1</sup>Or member variables, whatever your preferred notation.

We implemented and tested our three solutions into one overarching test program that produces arrays of apps in ever increasing input sizes and feeding those arrays to our algorithms one at a time, testing the average runtime over 10 iterations. The general flow of our program is as follows:

First we specify a list of input sizes that our program will act on, we will discuss this in more detail further along but for now it should be noted that our first input size is 5. Our program then creates an array of newly created Apps of the specified input size<sup>2</sup>. These newly created apps have each of their properties randomly assigned by its constructor. Next, because our Greedy Solution requires the inputs be sorted based on its ratio variable and the other solutions do not, we sort the list using Python's built in sort function.

Now we simply feed the newly sorted list to each of the solution functions in turn, timing the run time of each by taking a timestamp before running the algorithm, again after, then subtracting the two times. We run each program ten times per input and take the average of these for more accuracy. Once each solution algorithm has run ten times the program outputs the data (which includes the solution set, the amount of memory freed by the solution, the combined cost of the solution and the average time it took) to two text files in the directory of the program.

It is at this point that our program returns to the top of the loop for the next input. To ensure a adequate spread of input sizes we defined a list of base inputs which we initialized as 5, 10, 15, and 25. Once the program finishes iterating over this list, testing each solution as mentioned above, the program returns to the beginning of the input list and multiplies each by 10 to the power of  $n$ , where  $n$  is incremented with each new round of the loop back to the beginning of the base inputs. This entire process is placed in a while loop that is set to infinitely repeat so that we are able to get as many data sets as the program can output in the time we have allotted for testing.

---

<sup>2</sup>Again, in our first case this is 5.

## 3 Experiment

### 3.1 Results Data Set

Input	Memory	Cost	Freed	Time	
5	Brute	609	206.960013686	769	0.0000652
	Dynamic	609	413.920027372	1538	0.008588
	Greedy	609	206.960013686	769	0.00572586666667
10	Brute	1254.0	306.007083914	1307	0.0023169
	Dynamic	1254.0	913.397426561	3126	0.033041
	Greedy	1254.0	418.629747	1961	0.0220281
15	Brute	1896.2	462.268333337	1980	0.0933534
	Dynamic	1896.2	706.822856172	3129	0.10722175
	Greedy	1896.2	506.866912451	2383	0.0714820333333
25	Brute	2469.0	619.523226819	2479	113.6533167
	Dynamic	2469.0	1204.68191427	4035	56.92951935
	Greedy	2469.0	620.454984919	2562	37.9530137333
50	Dynamic	5421.2	1664.89861527	7120	0.870269
	Greedy	5421.2	1177.03320938	5519	0.43513695
100	Dynamic	9783.2	2809.24740865	11268	3.8071061
	Greedy	9783.2	2392.11409139	9849	1.90355725
150	Dynamic	14995.8	4027.50565053	16594	9.4551353
	Greedy	14995.8	3696.37456362	15462	4.7275798
250	Dynamic	26793.4	6591.81933106	28200	32.8375575
	Greedy	26793.4	6272.26215744	27018	16.41878765
500	Dynamic	52354.6	12547.9234913	53429	192.5988691
	Greedy	52354.6	12365.0250774	52842	96.2994775
1000	Dynamic	107148.2	25671.8344823	109000	1334.8144697
	Greedy	107148.2	25256.6939401	107895	667.4074229

## 4 Roles

- Illya Starikov
  - Project Manager
  - Brute Force Development
  - Abstract, Experiment Writement
- Timothy Ott
  - Greedy Algorithm Development
  - Development Writeup

- Testing Documents
- Claire Trebing
  - Dynamic Programming Development
  - Experiment Writing Up
  - Quality Assurance

## **5 Conclusion**