Project 2 Report

Dustin Ferguson, Nathan Johnson

November 2025

This program uses two java classes representing items and max binary heaps to solve three different iterations of the knapsack problem, where items are placed into a max binary heap and selected based on a priority factor to be put into a knapsack that must remain under a certain weight. This program's Main() sets up the problems and prints the results.

1 Item.java

This class represents an item to be selected for the knapsack problem. Contains 4 variables, a constructor, and a toString() method override.

1.1 Constructor

Contains takes in an int for Item ID and a double for each weight, value and priority factor of the item

1.2 toString()

Prints a detailed description of the item within square brackets.

2 MaxBinHeap.java

This class represents a max binary heap to be used for item selection for each knapsack problem. It contains a constructor, four helper methods, and a toString() method override.

2.1 Constructor

Constructor takes an arrayList of Items and immediately puts it into createMaxHeap(), storing the result in the variable maxHeap.

2.2 createMaxHeap(itemList)

This method takes in a list of Items, and adds each one iteratively using the addToHeap() method, It returns the max heap formed as an arrayList.

2.3 addToHeap(newItem, maxheap)

addToHeap() takes in the item to be added and the heap (an arrayList) to be added to. It returns the resulting max binary heap.

It firstly adds the new item to the end of the arrayList (the max heap) and then uses bottom-up heapification starting from the newly added node. It uses (x-1)/2 (where x = the node's current index) to find the index of the parent node, then compares the parent's priority factor with its own. If the parent node's priority factor is less than the child, the nodes swap indices in the arrayList. The method continues to compare and swap parent and child nodes in a while loop until the parent's priority factor is higher than the added node and its correct place in the max binary heap is found.

2.4 deleteMax()

deleteMax() takes no arguments, and returns the max Item from the heap that was deleted.

It first stores the maximum node to later return (index = 0), then swaps that node with the last node in the arrayList and removes the (last) node. It then performs top-down heapification. We define the left and right child as null, and if the heap's size is big enough, define the left and right children as items. It then continuously (in a while loop) compares its own priority factor with that of each of it's children and swaps with the larger of the two until it either reaches the bottom of the heap (both children = null)or all of it's children's priority factor is less than it's own. It then returns the deleted (max priority factor) node.

2.5 size()

Returns the size of the heap – the number of items contained.

2.6 toString()

Displays the array representation of the heap containing only item ID's, followed by a detailed list of those items in the same order.

3 Main.java

Contains static methods Main(), 3 methods to handle problem numbers 3,4, and 5 in the assignment, and a printResults() method to nicely print the answers in output.

3.1 main()

For each problem given (3,4 and 5), main prints out the problem label, stores the max binary heap created using the appropriate problem's helper function (described below), then prints the results of the associated knapsack problem using printResults().

3.2 makeProblem3Heap()

Constructs the items with the priority factors specified by problem three, adds them to an arrayList, and creates a max binary heap using MaxBinHeap's constructor. It returns the max binary heap created.

3.3 makeProblem4Heap()

Constructs the items with the priority factors specified by problem four, adds them to an arrayList, and creates a max binary heap using MaxBinHeap's constructor. It returns the max binary heap created.

3.4 makeProblem5Heap()

Constructs the items with the priority factors specified by problem five, adds them to an arrayList, and creates a max binary heap using MaxBinHeap's constructor. It returns the max binary heap created.

3.5 printResults(heap)

This method takes in a max heap created by one of the previous three methods, and then solves and prints the result of the associated knapsack problem.

It first prints the heap that is produced and then makes an empty arrayList called knapsack. It then continually adds the node at the top/front of the heap (the node with highest priority factor) to the knapsack if doing so will keep the knapsack under the given weight limit of 67 lbs. It deletes this max node from the heap as it checks using MaxBinHeap.deleteMax(). It will delete and check the max node until all items have been checked and removed from the max binary heap. It then prints the contents of the knapsack represented by an array of Item ID's.