Rudy Ruiz
Austin Pickett

Grocery Bagging

Grocery Bagging is a java application for finding a solution to the given problem. Determine a way to bag groceries that has constraints on what items can be bagged with what, how much you can put in a bag, and the number of bags available.

Installation

Use the included Makefile utility to compile and clean the GroceryBaggin.java solution.

‘Make’ - Compiles projects and generates the solution ‘groceryBaggin.java’

‘Make Clean’ - Delete object files and executables for a ‘clean’ state.

Usage

First give executable permissions to bagit.sh in order to run the GroceryBaggin program

‘Chmod a+rx bagit.sh’

Run bagit script with problem text file as arg1 and [-breadth/depth] flag

‘./bagit.sh problem1.txt -breadth’ for a BFS solution GroceryBaggin

‘./bagit.sh problem1.txt -depth’ for a DFS solution to GroceryBaggin

Rudy Ruiz
Austin Pickett

Design

Figure 1 shows a general view of our design layout.

GroceryBaggin.java - Application base with the main function. Parses the program arguments, creates each item, searches for solution via BFS/DFS

Bag.java - Represents a bag entity. Contains weight of each bag, items currently stored, and constraints what items can fit in BitSet form.

Item.java - Represents an item entity. Contains item index, item weight, and item bagging constraints in BitSet form.

WorldState.java - Contains information about which bags hold what items.

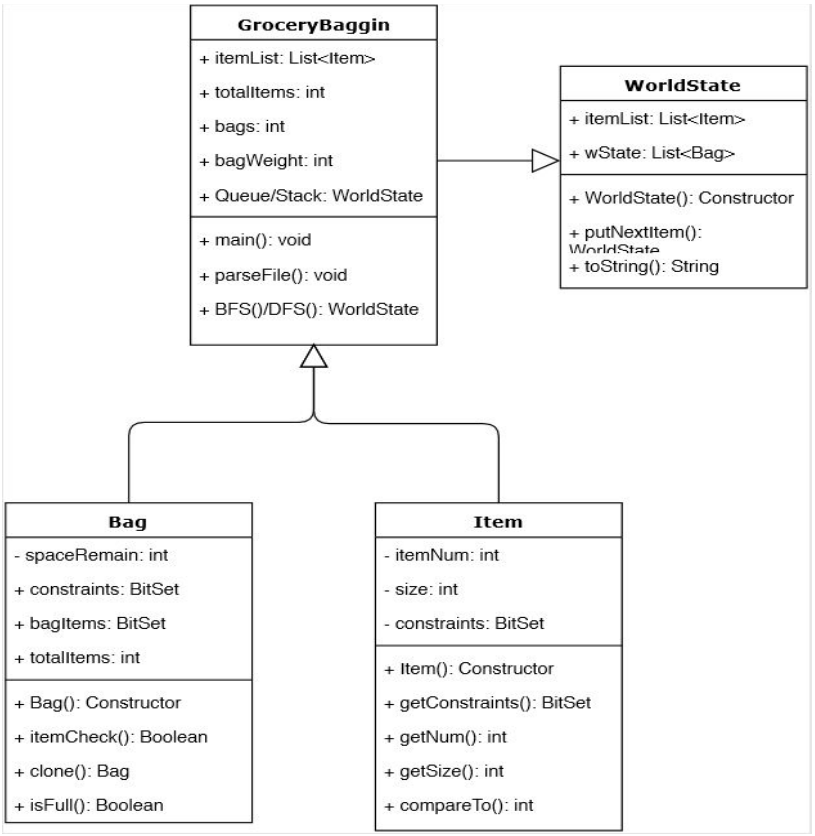

Figure 1

Rudy Ruiz
Austin Pickett

The design for our breadth first search and depth first search algorithm was modeled after the pseudocode on page 82 from our textbook “Artificial Intelligence: A Modern Approach.” This helped us layout the world state and what was needed to build a tree after our model.

Testing

During production of the Grocery Baggin application, we used the simple test case given to us in Figure 2. Once we had the simple test running, we started thinking of corner cases (Total items weight > total bag weight, no bags, no items, etc…) and handled those. We then started running a random problem generator with the provided baggen.tar.gz file. Keeping the sample case small to verify a solution. We believe the small test cases similar to figure 2 passed the test but as for problems with greater scale, we can only trust in our program.

```
3                  //number of bags available
7                  //maximum bag size is 7
bread  3 + rolls   //1st item size
rolls  2 + bread   //1st item size
squash 3 - meat    //2nd item size
meat   5           //3rd item size
lima_beans 1 - meat //5th item size
```

Figure 2

Rudy Ruiz
Austin Pickett

Experimental Results

Constraint checks was one issue in which our team put a lot of thought and effort into. Our groups original approach was that of storing multidimensional array of to act as a look up table for the items constraints. We would first select the item then check if there were any conflicting constraints between the items in the bag and the selected item. We later decided on the idea of using a binary string the length of total items to represent both the constraints and items contained in bags. Using this method along with the java BitSet library made checking conflicting constraints simple and quick.

Contributing

The two contributors to the Grocery Baggin Problem were Austin Pickett and Rudy Ruiz. The project was hosted on GitHub and features were implemented on branches then pushed to the master branch.