Clint Wyatt

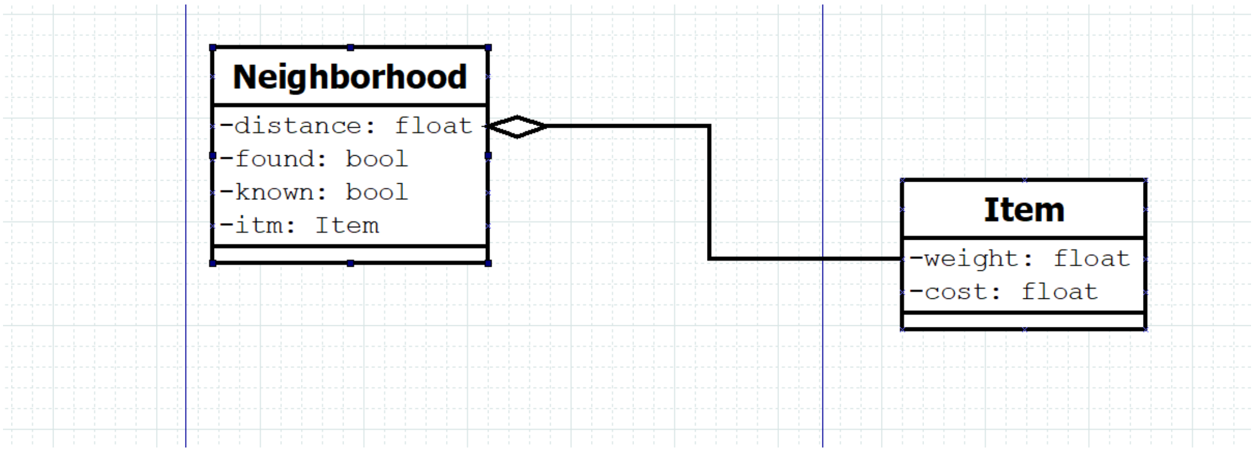
CSCE 4110

Greedy Van optimization

This project was about finding the max profit that a delivery van can generate delivering items to neighborhoods. Each neighborhood is connected to all other neighborhoods, which is a complete graph. Each item has a weight and a cost, and the van has a weight limit on how many items it can carry. Every mile the van travels costs $1.00 and the profit the van generates is 10% the cost of delivering the item. This problem involved finding the shortest path to each neighborhood and taking the path that gives the maximum sale of an item.

Data Structures used

This problem required a couple of different data structures. The first data structure used was objects, which represented the neighborhoods and items. Each neighborhood has an item object that holds the item's information such as weight and cost. The neighborhood object has a distance variable, Boolean variables named found and known and an item object. Below is a UML diagram that shows the relationship between the items and neighborhoods:



Next, there is dynamic memory allocation for 2 two-dimensional arrays for the adjacency matrix. This is done because there is always a risk of running out of stack space, and the heap is considerably larger than the stack since the heap does not have a strict limit on space like the stack does. Below is the code that implements the two-dimensional arrays on the heap:

```
matrix = new neighborhood*[x];
matrix2 = new neighborhood*[x];
for (int i = 0; i < x; i++)
{
	matrix[i] = new neighborhood[x];
	matrix2[i] = new neighborhood[x];
}
```

Algorithm Used

The first challenge in developing the algorithm for this problem was to find the cost of delivering an item to a neighborhood. This was solved by the following equation:

Total Cost = item price – neighborhood distance

Every time the van attempts to deliver to another neighborhood, the driver of the van needs to know the shortest distance to all other neighborhoods he is delivering to. The van loses $1.00 every mile it drives, so in order to minimize the loss the van must find the neighborhood that has a very valuable item that is close by.

The next challenge was deciding how to maximize the profit every n trips. The options were to use a minimum spanning tree or use a shortest path algorithm. I decided to use a shortest path algorithm, modified Dijkstra's, and use it n amount of times until all the houses had a delivery attempt or the weight of delivered items equals the vans weight capacity. Below are the steps to the algorithm:

1. Have the user input the maximum weight that the van can carry, the maximum distance between 2 houses, the maximum price of an item, the maximum weight of an item, and the number of neighborhoods near the distribution center.
2. Dynamically allocate memory for 2 adjacency matrixes of neighborhood objects based on the number of neighborhoods entered by the user.
3. Randomize all the neighborhood distances, item weights, item prices, in the 2d adjacency matrix.
4. Find max sale for the first neighborhood to be delivered to from the distribution center.
5. Use Dijkstra's algorithm to find the shortest path to all the other neighborhoods starting at the previous neighborhood that was delivered to.
6. After Dijkstra's algorithm is finished, loop through all the neighborhoods and find the maximum profit by subtracting the houses distance from the neighborhoods item's cost. Use a temporary variable to keep track of the index in the adjacency matrix where the max price was found.
7. Mark the neighborhood with the max sale in the adjacency matrix as found in all its locations in the second adjacency matrix. This neighborhood can be used with Dijkstra's later but cannot be delivered to again.
8. Copy the second adjacency matrix to the first adjacency matrix.
9. Repeat steps 5-8 until the weight of all the items equals the van's weight capacity or all neighborhoods have had a delivery attempt.
10. Print out the total multiplied by 10%.

Pros and Cons to the algorithm

The first upside for this algorithm is it guarantees the max profit because it uses Dijkstra's algorithm to find the minimum distance, and the cost is determined by the price of the item minus the distance. This algorithm works particularly well for urban areas (houses that are somewhat far from each other, around 50 miles) since Dijkstra's algorithm will likely update shorter paths more often since the paths from different neighborhoods will have more changes for the shortest path. The minimum spanning tree approach would not work well since it would not guarantee the shortest path to all other neighborhoods.

The second upside is the algorithm will rarely crash because it uses the heap more than the stack. If the device has enough ram, this program could simulate well over a thousand neighborhoods (would take a while). If we were to use the stack, the space the objects would take in the stack would likely cause a stack overflow error and crash the program.

The downside is the runtime. Since the runtime is cubic, inputs over a thousand take a very long time to complete. For an area with over a thousand homes, this algorithm would take too long to complete. This algorithm can work on inputs under a thousand for the neighborhoods near the distribution center, but not big cities.

The second downside is a minimum spanning tree would work better for many neighborhoods that are very close to each other due to time constraints. A greedy algorithm taking the best possible sale at a current moment would be quadradic and would work well for a city with very close neighborhoods (would minimize the loss the van has when traveling to other houses), but not necessarily guarantee the max sale. The minimum spanning tree would be similar to the traveling salesman problem, and therefore an approximation algorithm.

Conclusion

This assignment was very challenging and rewarding when I found the solution. I enjoyed the algorithms class and will us the information I learned in this class on future problems.