Project 2 Report

By: Seth Beckman, Travis McCormick
12/13/2022

```java
void setValues(Subject sub[], int size) {
    for(int i = 0; i <= size - 1; i++){
        sub[i].SetSubjectValues();
    }
    makeWeight(sub, size);
    for (int i = 0; i <= size - 1; i++) {
        //the value is determined by the weight divided by cost to make sur you get the most bang for your buck
        //the weight is casted as a double since it is instantiated as an int
        sub[i].value = ((double) sub[i].weight * sub[i].cost);
        sub[i].name = i;
    }
}
void makeWeight(Subject sub[], int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            //this is to make sure it does not add weight because of itself
            if (i == j) {
                continue;
            }
            //the sqrt equation is the distance equation for a circle, if it is less that 4 that means the circles overlap
            if (sqrt(pow(sub[i].cord_x - sub[j].cord_x, 2) + pow(sub[i].cord_y - sub[j].cord_y, 2)) < 4) {
                sub[i].weight++;
            }
        }
    }
}
```

In these two functions we create and give a weight to each 100 vales. The weight is determined by how many phone are in its radius (all start at the weight 1) and then we take that weight and multiply it by the cost and that gives you the “valuableness”.

```java
double totalValue(Subject sub[], int size) {
    double x = 0;
```

```java
void quickSort(Subject arr[], int start, int end) {

    // base case
    if (start >= end)
        return;

    // partitioning the array
    int p = partition(arr, start, end);
```

This then takes those values and sorts them from least to greatest using the quicksort method.

these functions are getting the total values and costs of all the phones. The level of coverage is the total area that all of the phones cover

```java
void MostValuable(Subject sub[], Subject data[], double budget, int size) {
    quickSort(sub, 0, size - 1);
    int x = 0;
    for (int i = size - 1; i >= 0; i--) {
        if (budget <= 0) {
            break;
        }
        //the sqrt equation is the distance equation for a circle, if it is less than 4 that means the circles overlap
        //this is to insure that two subjects are not taken if they are to close together
        if (sqrt(pow(sub[i].cord_x - data[x].cord_x, 2) + pow(sub[i].cord_y - data[x].cord_y, 2)) < 4)) {
            continue;
        }
        if(budget - sub[i].cost >= 0){
            data[x] = sub[i];
            budget -= sub[i].cost;
            x++;
        }
        else{
            continue;
        }
    }
}
```

```
148 int Greedy_partition(Subject arr[], int start, int end) {
149
150     int pivot = arr[start].cost;
151
152     int count = 0;
153     for (int i = start + 1; i <= end; i++) {
154         if (arr[i].cost <= pivot)
155             count++;
156     }
157
158     // Giving pivot element its correct position
159     int pivotIndex = start + count;
160     swap(arr[pivotIndex], arr[start]);
161
162     // Sorting left and right parts of the pivot element
```

This function uses the sort function to list them form least to greatest. Then it takes the most valuable and saves it in a separate array. Making sure that none of the phones are to close or that it goes over budget.

this function takes the cost of all the phones and sorts them based off of cost.

```
183 void Greedy_quickSort(Subject arr[], int start, int end) {
184
185     // base case
186     if (start >= end)
187         return;
188
189     // partitioning the array
190     int p = Greedy_partition(arr, start, end);
191
192     // Sorting the left part
193     Greedy_quickSort(arr, start, p - 1);
194
195     // Sorting the right part
196     Greedy_quickSort(arr, p + 1, end);
197 }
```

This is the same quicksort method but using cost instead of the “value” value.

UPDATED IN GITHUB

```
200  void Greedy(Subject sub[], Subject data[], double budget, int size) {
201      int x = 0;
202      Greedy_quickSort(sub, 0, size - 1);
203      for (int i = 0; i <= size - 1; i++) {
204          if (budget - sub[i].cost > 0) {
205              data[x] = sub[i];
206              budget -= sub[i].cost;
207              x++;
208          }
209      }
210  }
```

This take the sort for the Greed_quickSort and grabs the cheapest costing phones and puts them into a separate array. This does not care about distance because it only cares about grabbing the cheapest options. This is inefficient, as shown by running the code. Not only is it less than our algorithm but the level of coverage is greater than what the grid provides because Greedy algorithms only care about grabbing the least costing phones. This means that the information that it collects would be redundant and therefore useless

```
215  void Random(Subject sub[], Subject arr[], double budget, int size) {
216      int i = 0;
217      while (true){
218          int x = rand() % size;
219          if (!sub[x].HasBeen && budget - sub[x].cost >= 0) {
220              arr[i] = sub[x];
221              i++;
222              budget -= sub[x].cost;
223              sub[x].HasBeen = true;
224          }
225          else if (budget - sub[x].cost < 0){
226              break;
227          }
228      }
229  }
```

This randomly grabs different phones and puts it into a separate array. This does not care about distance either, it only makes sure that you don’t surpass the budget. This is inefficient, as shown by running the code. Not only is it less valuable than our algorithm, and sometimes less valuable than the greedy sort but the level of coverage is greater than what the grid provides because the random algorithms only randomly grab phones. It makes sure the budget is not exceeded but other than that if two phones are too close it doesn’t matter

```
17  void Subject::SetSubjectValues() {
18      //randomly sets coordinates
19      cord_x = (rand() % 100) + 1;
20      cord_y = (rand() % 100) + 1;
21
22      if (cord_x < 50 && cord_y < 50) {
23          cost = 0.5; //Super Poor District
24      } else if (cord_x < 50 && cord_y >= 50) {
25          cost = 1; //Not-so poor District
26      } else if (cord_x >= 50 && cord_y < 50) {
27          cost = 1.5; // Middle District
28      } else if (cord_x >= 50 && cord_y >= 50) {
29          cost = 2; //Rich District
30      } else {
31          cout << "ERROR" << endl;
32      }
33  }
```

this is inside our classes instead of having it in our constructor because inside the constructor it was setting value to everything when we just wanted certain things to have values and never everything have a value