Tharun Saravanan

Professor Simha

Algorithms

28 November 2022

Assignment 3

1. Pen and Paper Exercises:

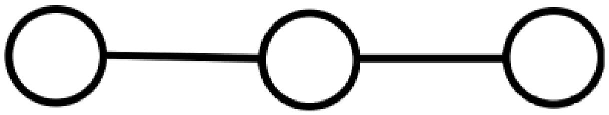
A.

i.

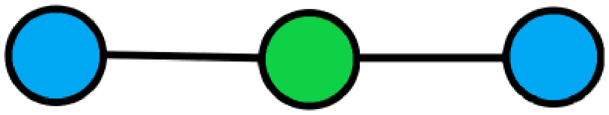
```
boolean isProper (adjList, blueVertices, greenVertices)
    //O(V) time to put all vertices with colors in array
    String[] arr = new String[adjList.length];
    //Assume each node must have an index field, the
    //adjacency list will be invalid without mapping indices
    for(node n: BlueVertices)
        arr[n.index] = "Blue"
    for(node n: GreenVertices)
        arr[n.index] = "Green"

    //Now parse through all of the edges (O(E) time)
    for(int i = 0; i < adjList.size; i++)
        string currColor = arr[i]
        //Parse through all nodes each LinkedList in adjList
        for(node n: adjList[i])
            //If any edge in the neighbors same color
            //Then the graph is invalid
             if(arr[n.index].equals(currColor))
                 return false //Return false
    //If no invalid edges have been found, valid graph
    return true
    //Thus total efficiency O(V + E)
```

ii.

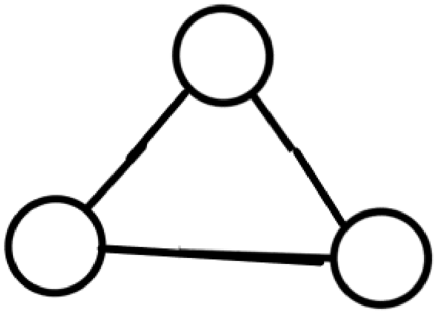
Graph that can be properly colored:



Example Coloring:



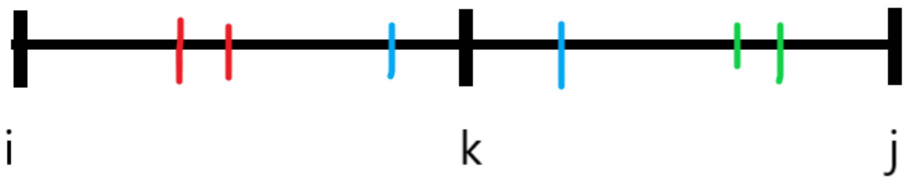
Graph that cannot be properly colored:



iii. Such an algorithm simply needs to check whether there exists a cycle of odd degree. We can employ a DFS to mark colors in an adjacency matrix styled representation, where we mark nodes as opposing colors of the parent node as we traverse through the graph. Reaching a previously visited node as part of the DFS that has been colored the same color as the current node being traversed means the graph cannot be colored properly. If no such odd cycle is found, then the graph can be properly colored. Such a traversal will have the same efficiency as a normal DFS, or O(V + E).

B.

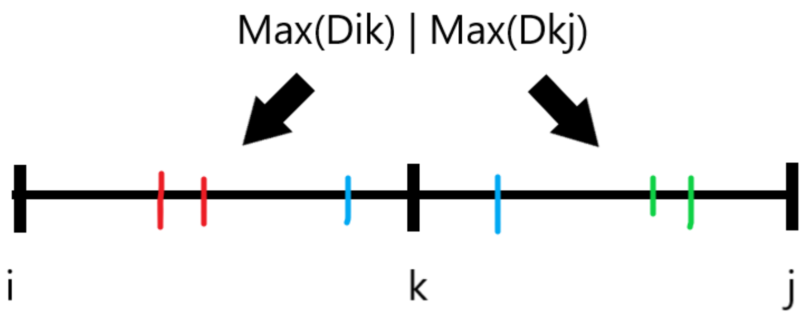
1. This problem cannot be solved using dynamic programming. Regardless of the order in which we parse through, all n elements in the range must be checked and compared to a max tracker to find the maximum value, with no way to further break the problem down as the data is unsorted.
2. This problem can be solved using dynamic programming. If we divide the set of data into indexes i, k, and j which represent the leftmost value, the middle value, and the rightmost value respectively, then there are essentially 3 cases the problem can be "broken down" into.



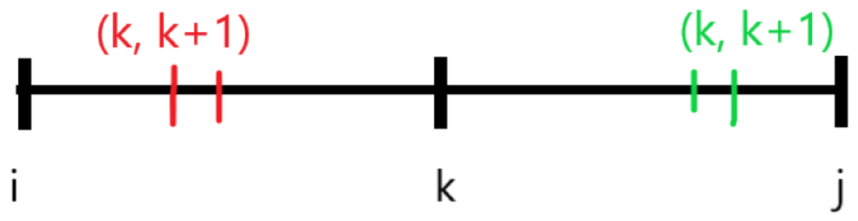
- Case 1: Both maximum values in (i, k)
- Case 2: Both maximum values in (k, j)
- Case 3: One maximum value in (i, k) and one maximum value in (k, j)

We will then have a case k where Dij can then be expressed as:

Dij = Max (Dik, Dkj, A[m] + A[n]) - Where A[m] is the largest value in (i, k) and A[n] is the largest value in (k, j)



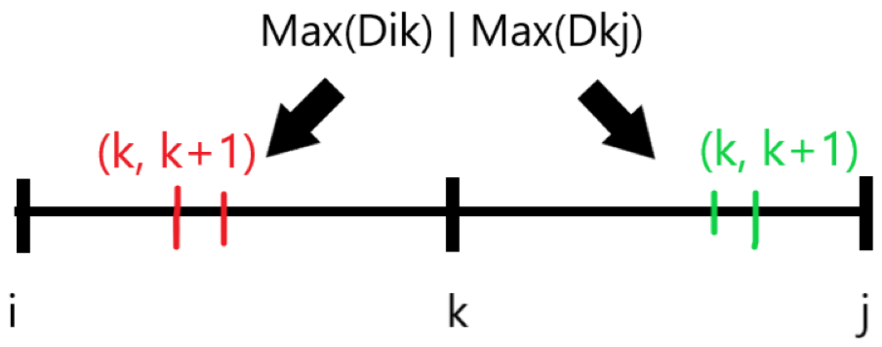
3. This problem can be solved using dynamic programming. The summation across pairwise numbers can be calculated as sub problems similar to the last problem. If we divide the set of data into indexes i, k, and j which represent the leftmost value, the middle value, and the rightmost value respectively, then there are essentially 2 cases the problem can be "broken down" into.



- Case 1: Max k, k+1 values in (i, k)
- Case 2: Max k, k+1 values in (k, j)

We will then have a case k where Dij can then be expressed as:

Dij = Max (Dik, Dkj, A[m] + A[n]) - Where A[m] is the largest consecutive sum in (i, k) and A[n] is the consecutive sum in (k, j)



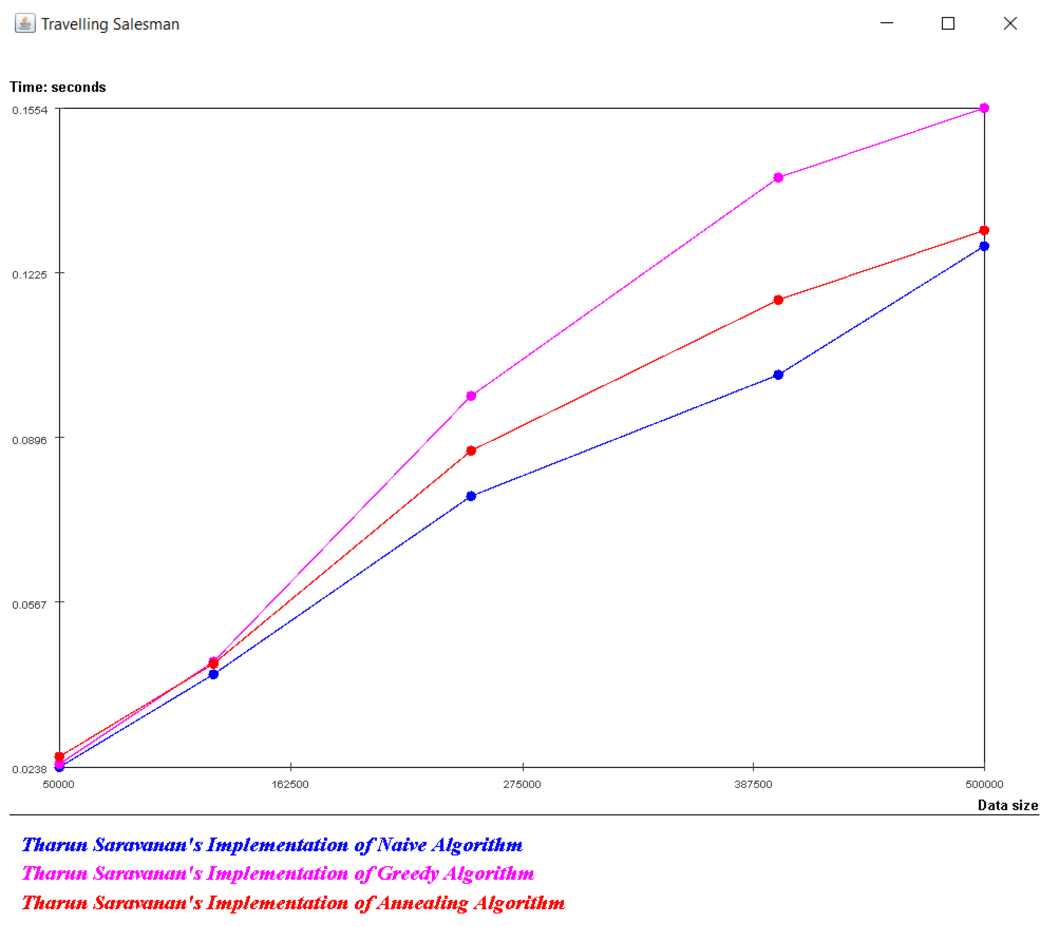
C. *Optimization Version*: Given n items with weights w1, w2, …, wn and K people able to carry these items, find the minimum maximum weight any person will hold
Decision Version: Given n items with weights w1, w2, …, wn, K people able to carry these items, and a weight Y, is there a way to distribute the weights such that each person is carrying no more than Y weight.

The decision version of the bin packing problem (Given n items with sizes s1, s2 ..., sn, a bin size B and a number K, is there an assignment of items to bins that uses at most K bins?) can be morphed into the decision version by replacing the sizes with the unit weight, the number K into K people, and size B into maximum size Y. Thus producing:

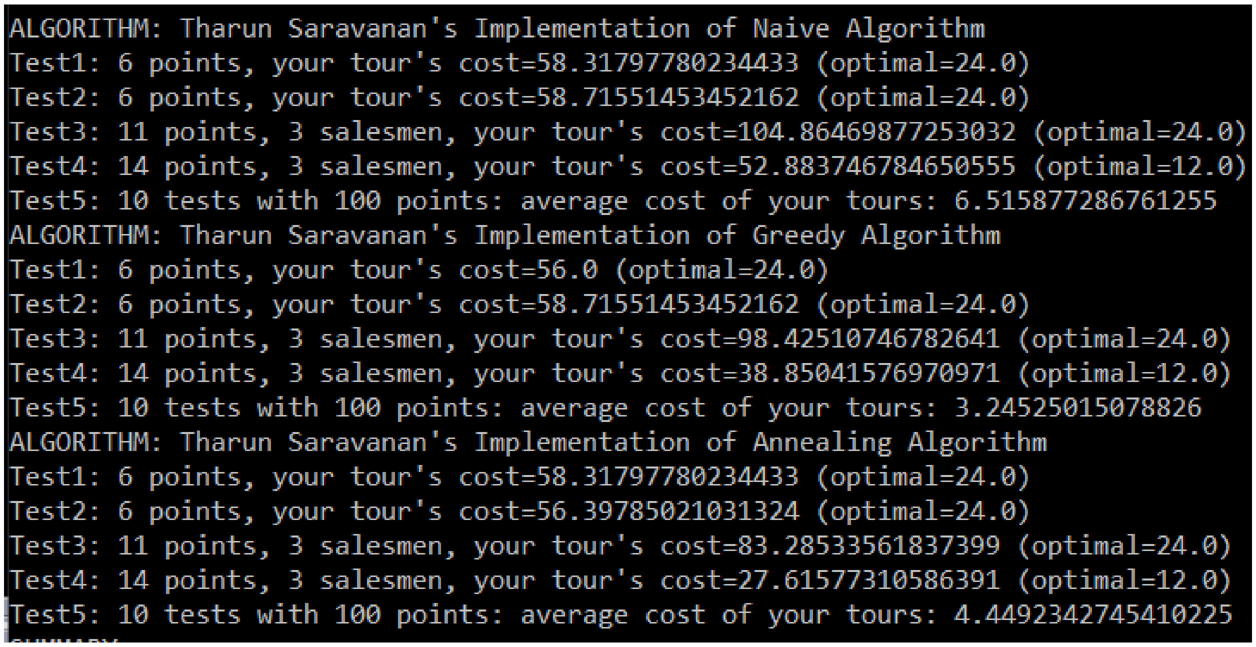
Given n items with weights w1, w2 ..., wn, a maximum weight Y per person and a number K people, is there an assignment of items to people that uses at most K people?

2. Implementation Questions:

a. Compare the quality of solution produced and time taken to produce the solutions for these algorithms:



- As shown by this analysis using large amounts of data, the Naive algorithm is overall the fastest solution, as it simply splits the points evenly across the number of salesmen. The annealing algorithm is the second best time wise, as it makes use of the naive solution to produce initial tours, then proceeds with simulated annealing. The greedy algorithm's need to parse the list for the closest node at each point proves to be costly as shown here, making it the slowest algorithm.



```
ALGORITHM: Tharun Saravanan's Implementation of Naive Algorithm
Test1: 6 points, your tour's cost=58.31797780234433 (optimal=24.0)
Test2: 6 points, your tour's cost=58.71551453452162 (optimal=24.0)
Test3: 11 points, 3 salesmen, your tour's cost=104.86469877253032 (optimal=24.0)
Test4: 14 points, 3 salesmen, your tour's cost=52.883746784650555 (optimal=12.0)
Test5: 10 tests with 100 points: average cost of your tours: 6.515877286761255
ALGORITHM: Tharun Saravanan's Implementation of Greedy Algorithm
Test1: 6 points, your tour's cost=56.0 (optimal=24.0)
Test2: 6 points, your tour's cost=58.71551453452162 (optimal=24.0)
Test3: 11 points, 3 salesmen, your tour's cost=98.42510746782641 (optimal=24.0)
Test4: 14 points, 3 salesmen, your tour's cost=38.85041576970971 (optimal=12.0)
Test5: 10 tests with 100 points: average cost of your tours: 3.24525015078826
ALGORITHM: Tharun Saravanan's Implementation of Annealing Algorithm
Test1: 6 points, your tour's cost=58.31797780234433 (optimal=24.0)
Test2: 6 points, your tour's cost=56.39785021031324 (optimal=24.0)
Test3: 11 points, 3 salesmen, your tour's cost=83.28533561837399 (optimal=24.0)
Test4: 14 points, 3 salesmen, your tour's cost=27.61577310586391 (optimal=12.0)
Test5: 10 tests with 100 points: average cost of your tours: 4.4492342745410225
```

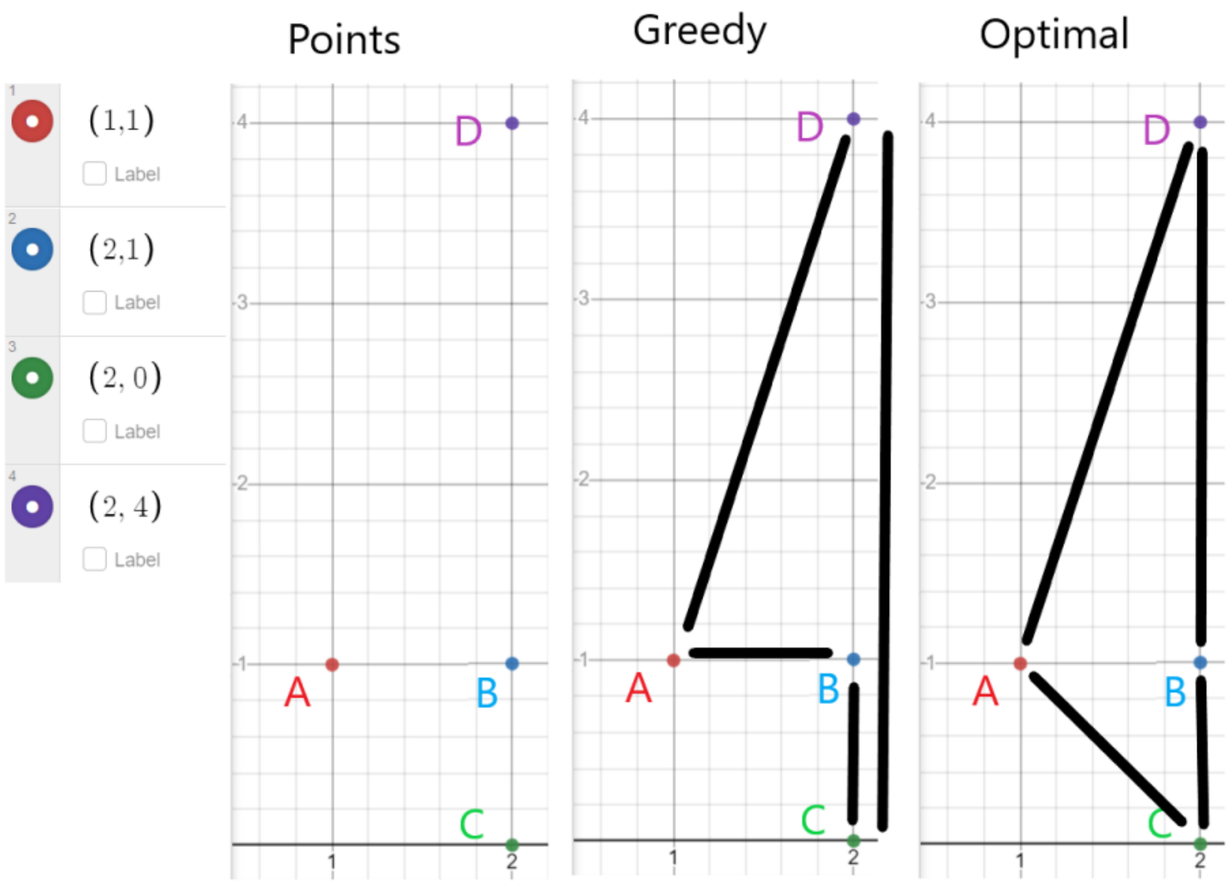
- Comparing the quality of the tours however produces the inverse result. The naive algorithm overall produces the worst tours across all of the tests with an average cost in the last test of 6.51. The greedy algorithm overall produces the best tours among the following tests, with the average cost in the last test of 3.25. The Annealing algorithm also produces "good" tours however with an average cost in the last test of 4.45, however it produced a better tour than the greedy algorithm in the test with 14 points, showing that the best algorithm among these two is not a black and white decision.

b. Submit an example (on paper) to show that your greedy algorithm does not always produce the optimal solution.

- Given the set of points {(1, 1), (2, 1), (2, 0), (2, 4)} and 1 salesman

(1, 1)	(2, 1)	(2, 0)	(2, 4)
A	B	C	D

- The greedy algorithm will produce the tour A - B - C - D - A
    - The algorithm will start from A as it is the first node passed in.
    - B is the closest node to A of length 1 away
    - C is the closest node to B of length 1 away
    - D is the last remaining node to choose from of length 4 away
    - Completing the tour from D to A has length 3.16
    - Total length of 9.16
- The optimal tour however is A - D - B - C - A of total length 8.57



c. Analyze the running time (in order-notation, on paper) of your greedy algorithm.

   - This algorithm parses through every node in the set of points and assigns them to a tour based on the greedy approach of finding the closest node. At first, the algorithm will add all nodes to a LinkedList to track which nodes are still to be added to a tour, taking O(n) time. An array holding the m tours is also created. From here a counter is then assigned to the first salesman, and the first nodes available in the LinkedList are assigned to the first location in each tour. From here, the algorithm cycles between the tours, and finds the closest node to the node last added to each tour and adds it to the current tour for all points available. This takes a total of $O(n^2)$ time as there are n nodes to parse, and finding the closest node for each node also takes n time.

d. Using an example (on paper), explain the neighborhood function used by your implementation of simulated annealing.

   - The neighborhood function of the annealing algorithm at any particular step is capable of producing any tours that can result by switching two of the nodes. We thus have a total neighborhood of n Choose 2, where n is the number of nodes in the current tour we are annealing. Given the following set of points for example:

(1, 1)	(2, 1)	(2, 0)	(2, 4)
1	2	3	4

The neighborhood function could choose any two points, choosing 1 and 2 however will produce the tour 2 - 1 - 3 - 4 - 2

(1, 1)	(2, 1)	(2, 0)	(2, 4)
1	2	3	4