Sarah MacAdam
Algorithms
Written 2
2/6/15
Worked with Divya Bhaskara

1

Problem Description
Create an itinerary that requires the fewest stops between points A and B
Inputs
int d: the maximum number of miles you can drive in a day
list x: contains all possible rest stops and the distance between it and the previous stop
int L: the length of the trip from A to B
Outputs
A list of all the stops on the optimized itinerary
Assumptions
The distance between any two stops is less than or equal to d. You will always be able to reach at least one stop every day.
Strategy Overview
This solution will use a greedy interval approach by traveling the maximum distance each day without going over d miles.
Algorithm Description

```
int t = 0;              //the distance traveled each day
int trip = 0;           //the distance traveled over the course of the trip
trip = dist(x_1);       //the distance to the first rest stop

    while( trip< L){        //while you haven't reached the endpoint

        if(t = d){      //if you traveled d miles after latest move
            s.add(x_n);         //add the current position to the itinerary
            t = 0;      //reset t for a new day
```

```
        }

        move(x_n, x_{n+1});          //move from current position to next stop
        t = t + dist(x_{n+1});       //update the day's distance traveled

        if(t > d){                   //check if you have traveled over d miles
           s.add(x_n);               //add the last position to the itinerary list
           t = dist(x_{n+1});        //create a new day of distance traveled

        }

            //else if you haven't gone d miles for the day, repeat the loop

    }

            //loop terminates, you have covered the distance L
return s;               //return the optimized itinerary
```

2

Proof of optimization for problem 1

Show the greedy algorithm from problem one is the optimal solution by proving that the cardinality of the greedy algorithm itinerary is equal to the cardinality of the optimal itinerary, i.e. they make the same number of stops.
$|G| = |O|$
lemma $g_i \geq o_i$ where g_i and o_i are the distances traveled at step i

At each step i the greedy algorithm has traveled either further than or equal to the optimal's distance

Prove using induction
Base case $L \leq d$
Both algorithms make no stops. This is clearly optimal.
Inductive Hypothesis Assume that $\forall i \leq k-1 \quad g_i \geq o_i$
Show that this must also be true for k

FSC: Assume $g_k < o_k$
From the inductive hypothesis, the greedy algorithm has traveled an equal or greater distance than the optimal after $k-1$ steps. However, this assumption implies that there was a stop closer to or equal to d distance away that, by definition, the greedy algorithm should have taken. So $g_k < o_k$ is false. The greedy algorithm's distance thus far must be equal to or greater than the optimal's at step k

Proof using lemma:
Assume FSC $|G| > |O|$
This implies that the last choice the greedy algorithm chose to make had at least an additional stop x_i while the optimal completed the trip in one final step. From the lemma, we know that the greedy algorithm has always traveled a greater or equal distance to the optimal algorithm at any step i. If the optimal was able to travel all the way to the end point without a stop, then the optimal and the greedy must both be within distance d of the endpoint. In this case, by the definition of the greedy algorithm above, the greedy should have chosen to travel all the way to the end at this step rather than make an additional stop. This is a contradiction and makes the last assumption
$|G| > |O|$
false.
Conclusion Therefore the greedy algorithm produces the optimal solution.

3

Problem Description
Find the fastest route from Charlottesville to Montreal.
Inputs
A graph G of the various paths from Charlottesville to Montreal
A start node s, which is Charlottesville
An end node t, which is Montreal
Outputs
A list of the roads in the optimized itinerary
Assumptions
you cannot travel backwards in time

$\forall_{u,v,t} f(u,v,t) \geq t$
you will never arrive earlier by leaving later
$t_1 > t_2 \rightarrow f(u,v,t_1) \geq f(u,v,t_2)$
The equation $f(u,v,t)$ returns the total time it takes to travel the shortest path from Charlottesville to u, plus the time from u to v where $e = (u,v) \in G$ and t is the time it takes to travel from the start node to u plus the time it takes to travel from u to v.

Strategy Overview

This algorithm will use Dijkstra's shortest path. The weight of each edge will be determined by the length of time it takes to travel each road.

Algorithm Description

1. Set the weight of each road to infinity
2. Begin at s, Charlottesville
3. Update the path to s as 0
4. Use the function $f(u,v,t)$ to find the time for each possible path from the current node to any connected nodes
5. Update all of the paths
6. Take the path on the graph with the shortest weight to an unvisited node
7. If the new node is Montreal, stop and return the series of nodes in that path
8. Otherwise, record the new, current node's shortest path, the series of nodes from Charlottesville to that node and its weight, the total time from Charlottesville to that point
9. Loop back to 3

Runtime

$\theta(n^2)$ where n is the number of nodes

In the worst case scenario, the last node visited is Montreal and all nodes are visited (that's n steps) and every node is connected to every other node (the function is called n times at n nodes) giving n^2 as the run time.

4

Proof of optimization for problem 3

Prove using induction that the greedy will find a path as fast as or faster than the optimal solution

$g \leq o$ the greedy solution will take less than or an equal amount of time as

the optimal
Base case There is an edge from Charlottesville to Montreal that is the shortest edge from Charlottesville to any other node. The greedy selects this edge. This is clearly the optimal solution because all other paths out of Charlottesville have a greater weight and we know there is no negative weight in the graph. This edge is the optimal solution.
Inductive hypothesis $\forall i < k - 1$ $time(n_1)$ is minimal
The greedy algorithm has optimal substructure up to k-1 where it finds the path to n_1 that takes the least amount of time.
Show $time(k)$ **is minimal**
$time(k) = time(i) + cost(e)$
where $i < k$ and $cost(e)$ is the time it takes to travel from i to k
Assume FSC $time(k)$ is not minimal
Therefore, some other shorter path exists
$time(j) + cost(e') + \delta < time(i) + cost(e)$ where best case $\delta = 0$
This alternate path was faster than the one we assumed the greedy algorithm chose, but this contradicts the definition of the greedy algorithm which always selects the fastest path at each stage. Therefore the last assumption, that $time(k)$ is not minimal is false. The greedy algorithm will always choose the fastest path and therefore is optimal.

5

Elite Trees versus MSTs
1. Every MST is an elite tree. True
We have proven that MSTs have optimal substructure.
Assume FSC the bottleneck edge of an MST is not minimal
Using an MSTs optimal substructure, we can divide an MST into two forests

f_1 and f_2 and an edge e where e is the bottleneck edge

$T = total\ cost(T) = cost(f_1) + cost(f_2) + cost(e)$

If the bottleneck edge of the MST is not minimal, then there exists an alternative edge e' that can connect two forests, which we now are minimal because of the optimal substructure of MSTs.

$cost(e') < cost(e)$.

$cost(f_1) + cost(f_2) + cost(e') < cost(f_1) + cost(f_2) + cost(e)$

therefore $cost(T') < cost(T)$

This contradicts that the MST is optimal. If this were true, the MST would have picked this alternative edge e'. Therefore the last assumption, that the bottleneck edge of an MST is not minimal, is false.
We can conclude that all MSTs have the minimal bottleneck edge and therefore are all elite trees.

2. Every elite tree of G is also an MST of G. False
Counterexample Graph G has three nodes and three edges connecting all of the vertices. The edges are weights 10, 10, and 1. A minimum spanning tree for this graph would include the edges 10 and 1. An elite tree could also include the edges 10 and 1. No matter what, the elite tree must have an edge of size 10. Therefore the bottleneck edge for any elite tree of graph G will be 10. By the definition of elite trees, it could also be 10 and 10. This would not change the value of the bottleneck edge, but it clearly is not an MST. We can conclude that not all elite trees are MSTs.