CS 4820, Spring 2014 **Homework 3, Problem 2**
Name: Piyush Maheshwari
NetID: pm489
Collaborators: Eeshan Wagh

(2) *(10 points)* You're consulting for the state highway authority on a project to determine where they should place speed limit signs on a new highway. Assume that the highway has a length of L miles, and that points on the highway are identified by non-negative numbers d representing their distance from the west endpoint. The highway authority wants to enforce two types of speed constraints: a maximum speed of $s_{\max}$ miles per hour, which applies to the entire length of the highway, and local speed limits which can be expressed by ordered pairs of integers (s_i, d_i) expressing the constraint that the cars should be moving at a speed *no faster than* s_i miles per hour when passing through point d_i on the highway. There are n such constraints, numbered (s_1, d_1) through (s_n, d_n) with $d_1 < d_2 < \cdots < d_n$.

A speed limit sign is designated by a pair of integers (s, d), meaning that at point d on the highway there is a sign telling drivers that their speed must be at most s when traveling between d and d', where $d' > d$ is the position of the next speed limit sign on the highway, or $d' = L$ if there are no speed limit signs after d. For reasons that make sense only to bureaucrats and CS 4820 professors, the highway authority has decided that they can only use k speed limit signs, where k is a positive number less than n. A collection of speed limit signs is *valid* if:

1. There are at most k signs.
2. One of the signs is at the highway's starting point, i.e. at 0.
3. A driver who obeys the posted speed limits cannot possibly violate the maximum speed constraint $s_{\max}$, nor any of the local constraints $(s_1, d_1), \ldots, (s_n, d_n)$.
4. None of the signs is located at a point in the set $\{d_1, \ldots, d_n\}$.

Subject to these conditions, the highway authority wants to allow drivers to get from 0 to L as rapidly as possible while obeying the speed limits. Design an efficient algorithm that computes the minimum time T such that there is a valid collection of speed limit signs allowing drivers to get from 0 to L in T hours.

Solution

Building the algorithm

Lemma 1. *If the allowed limit for any driver is s between any two points, then the driver should drive at the max speed possible.*

Proof. The proof directly follows from the fact that we want to minimize the time it takes to cross the road. The driver would only follow the speed signs which decides the maximum speed possible. □

Lemma 2. *Any optimal solution will place speed limit signs at $d_i \pm 1$*

Proof. If suppose we place any speed limit sign (s,d) such that d lies between some d_i, d_{i+1} and $d \neq \{d_i + 1, d_{i+1} - 1\}$. Suppose a driver was moving at speed s' when it entered this interval which means that the previous speed limit sign was s'. The driver will be driving at maximum driving speed possible by lemma 1. Now there can be two cases :

1. s' < s : In this case if speed increases upon hitting the speed limit sign within the interval. So having it at $d_{i+1} - 1$ will minimize the travel time without violating the constraint at d_{i+1}.

2. s' > s : In this case if speed decreases upon hitting the speed limit sign within the interval. So having it at $d_i + 1$ will minimize the travel time without violating the constraint at d_i.

So in both the cases we get a lower travel time by having the speed limit sign at $d_i \pm 1$. Hence the lemma holds. □

Now we know that the only possible places where we can place the speed limit signs are $d_i \pm 1$ and 0. Let P = $\{p_1, p_2, ...p_m\}$be the set of all possible such values such that each element is less than equal to L. Let M be the size of set P. If we put a speed limit sign at any p_i, we will get a sub problem with a smaller value of k. If we want to know the time it takes to reach some p_i using at most k speed signs, all we need to know is the time it took to reach the speed sign before it at some p_j using k-1 speed limit sign and we can calculate the speed limit we should put between speed limit signs k-1 and k by calculating the maximum speed that the constraints allow. This will make sure that we don't break any constraints. Let d(a,b) be the minimum time it takes to reach p_a using atmost b speed limit signs. We have the following recurrence -

d(a,b) = min{ d(c,b-1) + $(p_a - p_c)/s_{ac}$} $\forall c < a$

Here s_{ij} represents the time the maximum speed a driver can have between p_i and p_j without breaking any speed constraints. Since this s_{ij} will be used for every d(a,b) we can calculate as as part of pre computation step. The way we do that is first we initialize all s(i,j) to s_{max}. Then for every local constraint s_j we find the pair p_i, p_j between s_j lies. Since we can have the array P sorted, we can use binary search to do that. Now we have all possible s(i,i+1) initialized for all local constraints. Now we find s(i,j) for bigger values of j-i by using the smaller intervals which have already been computed. This is shown in lines 11-18.

After we calculate all values of d(a,b) the minimum time it takes to cross the road would be -

min{ dp(i,k) + (L - p_i)/ S_{iL}} $\forall i$ from 1 to M.

We can calculate s_{iL} which the maximum speed allowed from p_i to L following all constraints as part of precomputation as well.

Once we find d(i,j) for all values of i,j ,then it minimum time to reach the end of the road is simply d(i,k-1) for any p_i plus the time it takes to go from p_i to L. This fact is basically saying that the last speed sign we place is on p_i and hence the formula follows.

mintime = min d(i,k-1) + S(i,L) $\forall i$ from 1 to M

Algorithm

Algorithm 1

```
1: procedure MinRoad_Cross($T[], E[], X[]$)
2:     $P[1...2N+1] \leftarrow 0 \forall i$        ▷ P is the array of set of valid speed sign positions
3:     $P[0] \leftarrow 0$        ▷ Add 0 to the list of valid speed signs
4:     $C[i,j] \leftarrow (s_i, d_i)\ \forall n$        ▷ Store the constraints in an sorted order such that $d_1 < .. < d_n$

5:     for all $i = 1 - N$ do
6:         if $d_i - 1 \leq L$ then
7:             $P[] \leftarrow d_i - 1$        ▷ Add element $d_i - 1$to the list
8:         if $d_i + 1 \leq L$ then
9:             $P[] \leftarrow d_i + 1$        ▷ Add element $d_i + 1$ to the list

10:    $s[1...m, 1..m] \leftarrow s_{max} \forall i, j$        ▷ Initialize the max speed matrix to $s_{max}$

11:    for all $s_i, d_i$ in $C[]$ do        ▷ This loop sets the base cases in max speed matrix
12:        $j \leftarrow binary - search(P[], d_i)$        ▷ Find largest $p_i < d_i$ using BS since P[] is sorted
13:        $s(i, i+1) \leftarrow min(s(i, i+1), d_i)$
14:    M ← size of $P[]$
15:    for all l = 2 to M do
16:        for all i = 1 to M - l +1 do
17:            $j \leftarrow i + l - 1$
18:            $s(i,j) \leftarrow min(s(i, i+1), s(i+1, j))$

19:    $d[1..m, 1..k] \leftarrow 0$        ▷ d(i,j) represents the minimum time it takes to each $p_i$ with upto j
       speed signs

20:    dp(1,i) ← 0 $\forall i$ from 1 to k  ▷ Since $p_1$ is 0, hence the time taken from beginning to that
       point using any number of speed signs is 0

21:    dp(i,1) ← $p_i/s(1,i)$ $\forall i$ from 1 to M ▷ Since the first speed sign is always at 0, any d(i,1)
       would simply be the distance $p_i$ divided by maximum allowed speed from $p_1$ to $p_i$

22:    for all a = 2 to M do
23:        for all b = 2 to k do
24:            d(a,b) = INF
25:            for all c = 1 to a-1 do
26:                if d(a,b) > (d(c,b-1) + ($p_a - p_c$)/ $s(a,c)$) then
27:                    d(a,b) = (d(c,b-1) + ($p_a - p_c$)/ $s(a,c)$

28:    S(i,L) ← $s_{max}$        ▷ S(i,L) represent the maximum speed between $p_i$ and L
29:    for all i = M-1 to 1 do
30:        S(i,L) ← $s(i, M)$        ▷ Except the final value i.e $p_m$, this is equal to $s(i, M)$

31:    mintime ← INF
32:    for all i = 1 to M do
33:        if mintime > (d(i,k-1) + S(i,L) ) then
34:            mintime = d(i,k-1) + S(i,L)
35:    return mintime
```

Proof of correctness

Proof. We will prove this by induction. We claim that d(i,j) is the minimum time it takes to reach p_i by using at most j speed signs.

Base case - Base cases are defined on line 20 and 21 of the algorithm. dp(i,1) = $p_i/s(1,i)\forall i$. This follows from the fact that the first speed sign has to be places on location 0. Hence the its speed has to the maximum speed allowed between the constraints places between 1 and p_i. Also d(1,i) = 0 $\forall i$. This is because p_i is 0 and the time taken to each 0 is 0 for any number of speed signs.

Inductive Step - Consider d(i,j). From the inductive hypothesis we know that d(m,n) is equal to opt(m,n) $\forall m < i, n < j$. Now from the discussion above we know that

opt(a,b) = min{ opt(c,b-1) + $(p_a - p_c)/s_{ac}\}$ $\forall c < a$

Using the inductive hypothesis we have

opt(a,b) = min{ d(c,b-1) + $(p_a - p_c)/s_{ac}\}$ $\forall c < a$

opt(a,b) = d(a,b) □

Complexity Analysis

The maximum value of M can be 2N+1. Hence $O(M) = O(n)$

1. Line 5 - 9 takes $O()$
2. Line 12 takes $O(logn)$. Hence lines 11-13 take $O(nlogn)$
3. Line 15-18 takes $O(M^2) = O(n^2)$
4. Line 22-27 takes $O(M^2k) = O(n^2k)$
5. Line 28-20 takes $O(M) = O(n)$
6. Line 31-34 takes $O(M) = O(n)$

Total time complexity = $O(n)$ + $O(nlogn)$ + $O(n^2)$ + $O(n^2k)$ + $O(n)$ + $O(n)$ = $O(n^2k)$

Total Space Complexity = $O(M^2) + O(Mk)$. Since $k < n$ and $O(M) = O(n)$
Total Space Complexity = $O(n^2)$