HW 9 Problem 1

Jin Yucheng (yucheng9) Fan Wenyan (wenyanf2) Lu Hanxiao (hl4)

1.

(a) We can solve this problem by using greedy algorithm. Each time we try to stop at the next station that is most far relative to the current station but under below 100 miles. Otherwise it is impossible to reach destination. D is an array which shows distance of each station from starting point.

```
Count_min_station(D):
    Count = 1
    Total_station_num= D.size() -1
    Current_station_num = 0
    While(Current_station_num < Total_station_num ):
        Next_station_num = Current_station_num
        While (D[Next_station_num] < = D[Current_station_num] + 100
         && Next_station_num != Total_station_num):
            Next_station_num + =1;
        If(Next_station_num == Current_station_num):
            Return inf
        Else:
            Count + =1
            Current_station_num = Next_station_num
    Return Count

Main:
    Count_min_station(D);
```

Proof of the greedy algorithm:

Suppose there exists a solution S that does not use greedy solution. There is at least one stop i such that S[i] + 100 > = S[i+1+k] for some positive integer k. If we use greedy algorithm only on this stop i by deleting S[i+1]….S[i+k] in the original solution, we can show that Count(S') < Count(S) because k is a positive integer, where S' is our modified solution. By strong induction, we can easily conclude that we will have the fewest number of stops in solution if we do greedy algorithm on every stop.

(b) We can solve this problem by using dijastra algorithm. Since this is a undirected weighted graph, we can modify this graph to directed weighted graph and then use dijastra algorithm. For every edge uv in original graph, we delete it and add directed edges u -> v and v ->u if both of them are less than 100 miles. In the final step we apply dijastra algorithm. V is vertex list and E is edge list of original graph. S is starting point and D is final point.

```
Tramsform_edge(V,E):
    For every uv in E:
        E.delete(uv)
        If(uv <= 100):
            E.add(u - > v , v -> u)

INITSSSP(S):
    Dist(S) <- 0
    Pred(S) <- null
    For all vertices v != S
        Dist(v) <- inf
        Pred(v) <- null
```

```
Dijastra(V,E, S, D):
    Transform_edge(V,E)
    INITSSSP(S)
    Insert(S,0)
    While priority queue is not empty:
        u <- Extract_min()
        For all edges u -> v:
            If u->v is tense:
                Relax(u -> v)
                If v is in priority queue
                    Decrease_key(v,dist(v))
                Else
                    Insert(v,dist(v))

Main:
    Dijastra(V,E,S,D)
    Count = 0;
    For(u = D; u ! = null ; u = pred(u)):
        If u has a station:
            Count +=1
```

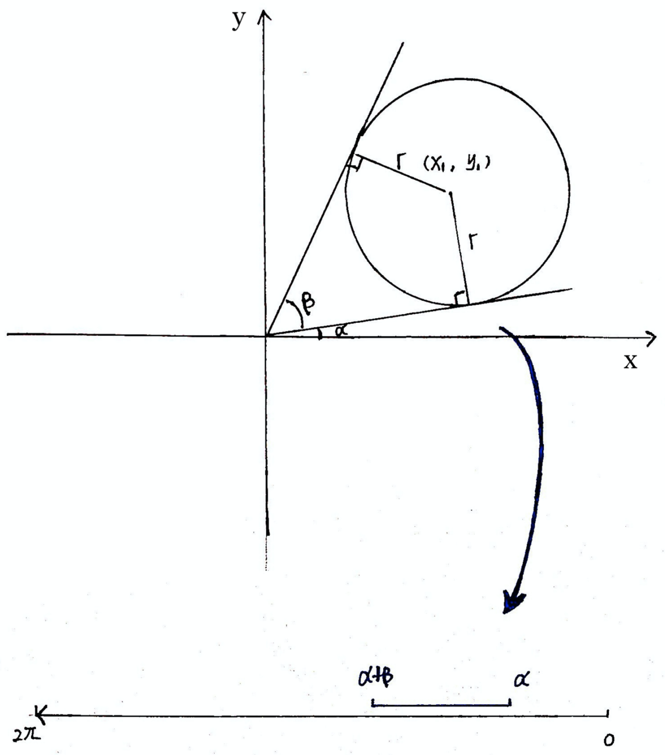
CS/ECE 374 B Fall 2019
Homework 9 Problem 2

Jin Yucheng (yucheng9)
Fan Wenyan (wenyanf2)
Lu Hanxiao (hl4)

Solution:

(c) I want to explain the general case first.

To begin with, implement a subrountine ConvertToAngle(C, r), which accepts the coordinates of the center of a circle, C ($C = (x, y)$), radius r, and returns an interval, $[a, b]$. The meaning of this interval is explained by the following figure.



We draw two lines from the origin that intersect the circle, each has an angle with respect to the positive x-axis (in the above figure, one is α, the other is $\alpha + \beta$, so $[a, b]$ is $[\alpha, \alpha + \beta]$ in this example). We can view that the smaller angle is obtained by rotating the positive x-axis counterclockwisely, therefore it is in the range of $[0, 360]$. A laser beam can intersect a ballon if it is launched at an angle between a and b, but there are some edge cases, where the larger angle, $b = \alpha + \beta$ is greater than 360, in these cases, a laser beam can intersect a ballon if it is launched at an angle between a and 360, or between 0 and $b - 360$.

We convert this problem to the problem of *Covering Segments by Points*, which is:

Given a set of n segments, $\{[a(1), b(1)], [a(2), b(2)], \ldots, [a(n), b(n)]\}$, a set of n labels that indicates the label of a segment (which circle a segment belongs), $\{L(1), L(2), \ldots, L(n)\}$, find the minimum number of points, m, such that each label contains at least one point.

The greedy algorithm works as follows,

```
def ConvertAllCircles(C[1...n], R[1...n]):
    1 : Segments ← [ ]
    2 : Labels ← [ ]
    3 : Index ← 0
    4 : for $i$ ← 1 to $n$:
    5 :     Index ← Index + 1
    6 :     $[a, b]$ ← ConvertToAngle($C[i]$, $R[i]$)
    7 :     if $b > 360$:
    8 :         Segments[Index] ← $[a, 360]$
    9 :         Labels[Index] ← $i$
    10 :        Index ← Index + 1
    11 :        Segments[Index] ← $[0, b - 360]$
    12 :        Labels[Index] ← $i$
    13 :    else:
    14 :        Segments[Index] ← $[a, b]$
    15 :        Labels[Index] ← $i$
    16 : return Segments, Labels
```

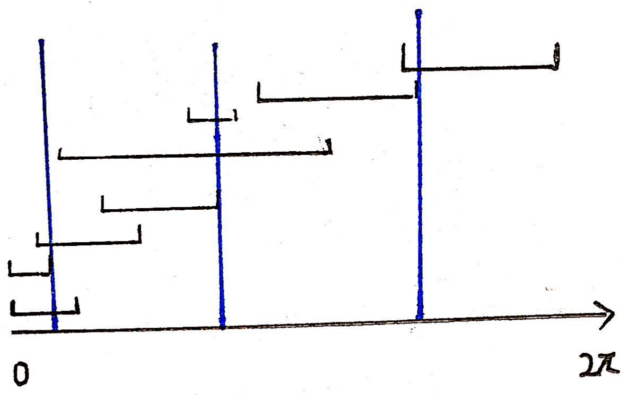
```
def GreedyBallonZapping(C[1...n], R[1...n]):
    1 : Segments, Labels ← ConvertAllCircles(C[1...n], R[1...n])
    # Segments is an array with $m$ elements of intervals of a start angle and an end angle,
    # Segments = $\{[a_1, b_1], [a_2, b_2], ..., [a_m, b_m]\}$,
    # $m$ is at most $2n$, that all $n$ circles are edge cases.
    2 : Sort Segments by end angles $(b_1, ..., b_m)$ and permute start angles $(a_1, ..., a_m)$ to match
    # End angles are sorted in increasing order.
    3 : Count ← 0
    4 : Angle ← []
    5 : for $i$ ← 1 to $m$:
    7 :     if Segments is empty: break
    8 :     Count ← Count + 1
    9 :     Angle[Count] ← $b_1$
    # $b_1$ is the smallest end angle
    10 :    for every interval $[a_j, b_j]$ with start angle $a_j > b_1$:
    11 :        CurrentLabel ← Label[$j$]
    12 :        Remove interval(s) with CuurentLabel from Segments
```

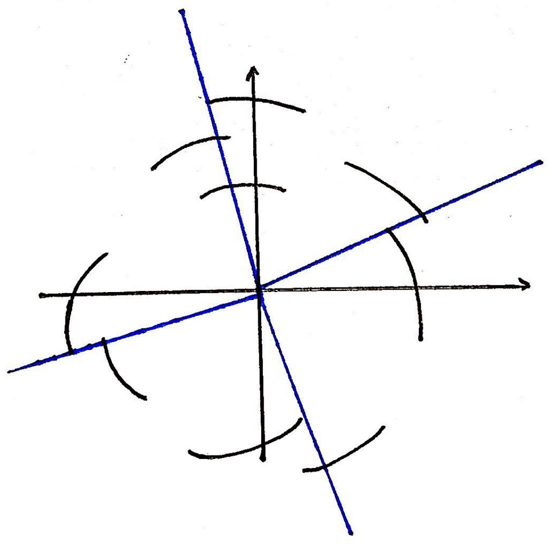
If we counter edge cases, we should remove two intervals with the same label.

13 : return Count, Angle

The above algorithm zaps all ballons and returns the smallest number of laser beams and the array of every lase beam angle.

The following figure explains the algorithm. Every time we pick the current smallest end angle as the angle of the laser beam, remove the ballons that are zapped by the laser beam (intervals with start angles larger than the smallest end angle), and repeat until all ballons are zapped.





To obtain intervals of angles takes $O(n)$ time. To sort the intervals by end angles in increasing order takes $O(nlogn)$ time, the main for-loop will execute at most m times, which is bounded by $O(2n)$, every time to remove intervals with start angles larger than the smallest end angle takes at most $O(m)$ time, therefore, it takes $O(n + nlogn + m^2)$ time, which is bounded by $O(n^2)$.

Proof of Correctness:

Lemma 1 (Basis Step): *At least one optimal solution with minimal number of laser beams includes the angle that is equal to the smallest end angle.*

Proof: Suppose there is only 1 ballon, then we need 1 laser beam, its angle can be the end angle. Suppose there are 2 ballons, if they are disjoint, then we need 2 laser beams, the first angle can be the smaller end angle; if they are not disjoint, which means that the smaller end angle is larger than the larger start angle, then we need 1 laser beam, its angle can be the smaller end angle. Suppose there are more than 2 ballons, since we must zap the ballon with the smallest end angle, the first laser beam (the laser beam with the smallest angle) must be launched at an angle between the interval of the ballon with the smallest end angle, then the greedy algorithm gives an optimal solution since not only that ballon is zapped, but also all ballons are not disjoint with that ballon are zapped.

Lemma 2 (Inductive Step): *The greedy algorithm gives an optimal solution.*

Proof: Let f be the ballon with the smallest end angle, and let A be the ballons whose start angles are larger than f's end angle. The previous lemma implies that some optimal solution contains a laser beam with angle that is equal to f's end angle, so the solution with the laser beam launched at f's end angle is an optimal solution. The optimal solution that zaps f must contain an optimal solution for ballons that are not zapped together with f, that is, an optimal solution for A. The greedy algorithm chooses the smallest end angle and then, by the inductive hypothesis, computes an optimal solution for A.

(a) Then let me solve some special cases (Hi, TA, if you are confused please see my (c) first since I first give a general solution).

First, call ConvertAllCircles(C[1...n], R[1...n]) to obtain an array Segment that contains the interval between the start and end angles of every ballon, an array Labels that contains the label (which ballon an interval belongs) of each interval, this takes $O(n)$ time.

Second, we choose a reference angle and set this angle as 0, if a laser beam is launched at this angle, no ballons will be zapped. Then convert the intervals of angles obtained above with respect to the reference angle, this only needs some additions and subtractions, it takes $O(n)$ time.

Third, sort Segments by end angles in increasing order, this takes $O(nlogn)$ time. Every time, the greedy algorithm picks the angle which is equal to the smallest end angle of all ballons that are not zapped yet. Then we remove the ballons zapped by the laser, which are ballons whose start angles are smaller than the chosen angle. Repeat until all ballons are zapped. The above procedure takes $O(n)$ time.

Overall, the algorithm takes $O(nlogn)$ time. The proof is as follows,

Proof of Correctness:

Lemma 1 (Basis Step): *At least one optimal solution with minimal number of laser beams includes the angle that is equal to the smallest end angle.*

Proof: Suppose there is only 1 ballon, then we need 1 laser beam, its angle can be the end angle. Suppose there are 2 ballons, if they are disjoint, then we need 2 laser beams, the first angle can be the smaller end angle; if they are not disjoint, which means that the smaller end angle is larger than the larger start angle, then we need 1 laser beam, its angle can be the smaller end angle. Suppose there are more than 2 ballons, since we must zap the ballon with the smallest end angle, the first laser beam (the laser beam with the smallest angle) must be launched at an angle between the interval of the ballon with the smallest end angle, then the greedy algorithm gives an optimal solution since not only that ballon is zapped, but also all ballons are not disjoint with that ballon are zapped.

Lemma 2 (Inductive Step): *The greedy algorithm gives an optimal solution.*

Proof: Let f be the ballon with the smallest end angle, and let A be the ballons whose start angles are larger than f's end angle. The previous lemma implies that some optimal solution contains a laser beam with angle that is equal to f's end angle, so the solution with the laser beam launched at f's end angle is an optimal solution. The optimal solution that zaps f must contain an optimal solution for ballons that are not zapped together with f, that is, an optimal solution for A. The greedy algorithm chooses the smallest end angle and then, by the inductive hypothesis, computes an optimal solution for A.

(b) First, shoot at an angle that can zap at least 1 ballon. If after this shoot there is no ballon left, then we are done with only 1 shot. Else, there are some ballons left, and there is a direction in which we can shoot a laser beam that does not intersect any ballons. It is exactly the situation in (a)! We can simply implement the algorithm in (a). As we've analyzed in (a), it takes $O(nlogn)$ time.

Proof of Correctness:

Denote the ballons the first laser beam zaps as A, the remaining ballons as B. Suppose it takes m shots for our greedy algorithm to zap all ballons in B, as we've proved in (a), m is the optimal solution for B. Besides, it takes 1 shot to zap ballons in A.

Now consider $A+B$. If we add some ballons that can be zapped by 1 laser beam to B, there are two cases to consider:

• The optimal solution stays the same, it is still m. Since we take $m+1$ shots, the greedy algorithm is within 1 of optimal.

• The optimal solution increases by 1 (it can not increase by more than 1 since there is a solution of $m+1$), it becomes $m+1$. Since we take $m+1$ shots, the greedy algorithm gives the optimal.

In both cases, the greedy algorithm successfully returns the solution within 1 of optimal. ■

Homework 9 Problem 3

Jin, Yucheng (yucheng9) Fan, Wenyan (wenyanf2) Lu, Hanxiao (hl4)

December 3, 2019

Solution

(a)

thick_count is a thick counter.
shelf_count is a shelf counter.
S[i] stores the index of the shelf where the book is stored.

We will solve this problem by the greedy algorithm.

```
Store(T[1:n]):
    thick_count ← 0
    shelf_count ← 1
    for i ← 1 to n
        # if there is room to place the next book on the shelf?
        if thick_count + T[i] > L
            thick_count ← T[i]
            shelf_count ← shelf_count + 1
            S[i] ← shelf_count
        else
            thick_count ← thick_count + T[i]
            S[i] ← shelf_count
    return S[1:n]
```

Why is it correct? Consider a set of books. Clearly, the first book must be placed on the first shelf. The greedy algorithm then places books on the first shelf as many as possible. This now fills the first shelf, and so we then must place books on the second shelf as many as possible. And so on. And, we can not do better. Instead of this layout, we can place as few books as possible on some first shelves, and so on, and some last books on some last shelves. In this case, the first shelves have only a few books, and the last shelves have too many books, at the high risk of needing more shelves, which is not better than what the greedy algorithm produced. We can prove more strictly as follows. Suppose there is an optimal algorithm that starts differently from our greedy algorithm for the b-th book. In the greedy algorithm, the b-th book is placed on the s-th shelf. In the optimal algorithm, the b-th book is placed on the (s+1)-th shelf, because it lacks space before and it has to be different from the greedy algorithm. The optimal algorithm must be at least as optimal as the greedy algorithm after the b-th book. Both algorithms are optimal for books prior to the b-th book. By the inductive hypothesis, the greedy algorithm is not worse than the optimal algorithm globally, so it is a correct algorithm.

The algorithm is a simple linear-time loop, so the entire algorithm runs in O(n) time.

(b)
Consider the situation below.
H[1:3] = [1, 3, 2], T[1:3] = [5, 5, 5], L = 10.
Our greedy algorithm solution is S[1:3] = [1, 1, 2], the sum of the heights of the shelves is 5.
The best solution is S[1:3] = [1, 2, 2], the sum of the heights of the shelves is 4.
Our greedy algorithm from part (a) does not always give the best solution to this problem.

(c)
thick_count is a thick counter.
shelf_count is a shelf counter.
S[i] stores the index of the shelf where the book is stored.
cost[i] stores the best possible cost of placing book i through n.
nextshelf[i] stores the index of the book that will start the next shelf.
begin is the index of the book that will start the shelf, end is the index of the end.

We will solve this problem first by going backwards. We will loop from n down to 1 and determine the cheapest way of placing books i through n if we start a new shelf with book i. We store the best cost in cost[i].

```
MATCH(H[1:n], T[1:n]):
    thick_count ← 0
    cost[1:n] ← H[1:n]
    nextshelf[1:n] ← [∞, ∞, ∞, ..., ∞]
    for i ← (n - 1) to 1
        thick_count ← T[i]
        for j ← (i + 1) to n
            thick_count ← thick_count + T[j]
            # if there is room to place the next book on the shelf?
            # if it will be cheaper to place the current book on the shelf we're on?
            if thick_count ⩽ L and max{H[i], ..., H[j]} + cost[j + 1] ⩽ cost[i]
```

```
                    cost[i] ← max{H[i], ..., H[j]} + cost[j + 1]
                    nextshelf[i] ← j + 1
    begin ← 1
    end ← nextshelf[1] - 1
    shelf_count ← 1
    while end ≤ n
        for i ← begin to end
            S[i] ← shelf_count
        shelf_count ← shelf_count + 1
        begin ← end + 1
        end ← nextshelf[end]
    return S[1:n]
```

The outer loop goes through the different subproblems in reverse order. For each subproblem, the inner loop iterates through the books to be placed on the shelf.

The execution of the inside of the innermost for loop is bounded by a constant amount. The number of times the innermost loop is executed is bounded by n, and the outer loop is executed n - 1 times, so the performance will be $O(n^2)$. The execution of the reconstruction is a simple linear-time loop, so the performance will be $O(n)$. Thus, the entire algorithm runs in $O(n^2)$ time.