ADA Assignment 5

Ankur Sharma

April 2018

Partners: Anvit Mangal (2016135) and Ishaan Bassi (2016238).

Problem 1

Given a set $\{x_1, x_2, ..., x_n\}$ of points on the real line, determine the smallest set of unit-length closed intervals (e.g. the interval [x, x+1]) that contains all of the points. Give the most efficient algorithm you can to solve this problem, prove it is correct and analyze the time complexity.

Solution:
Let X be the given set of real numbers $\{x_1, x_2, ..., x_n\}$ with size equal to n and R be the required smallest set of unit-length intervals s.t. it contains all of point in X.
From inspection, we can see that **A Greedy Approach** can be applied here in order to get the desired result. A rough idea is to find the smallest element(x) in X, and put the interval $[x, x+1]$ in R and removing all the numbers that lie inside $[x, x+1]$ from X. We do this repeatedly till X gets empty.
Now, let's to compute the time-complexity of above discussed approach. While X is not empty: First we find the smallest element(x) in X which would take $O(n)$ time (Since X is not sorted) and then, finally removing points from X that lie inside $[x, x+1]$ which would again take $O(n)$ time. Hence making it $O(n^2)$ in the worst case when 'while loop' runs n times.
Can we think of a better way? Yes! If X was a sorted set(arraylist), then overall time-complexity can be reduced. Therefore, we first sort X and then do the same computations. Following is the Pseudo-Code for the same.

Procedure getSetOfIntervals(X):
  # sort X using mergesort algorithm
  S = mergeSort(X) # $S = \{s_1, s_2, ..., s_n\}$
  # S is a sorted set(arraylist) s.t. $S[0]$ is the smallest element in S
  set R = ϕ
  while S is not empty:

s = S[0]
I = [s, s+1]
R.add(I)
remove points from S that lie inside I starting with s
return R

Proof of Correctness:
By way of contradiction, suppose that R is not a correct set returned by above mentioned procedure. Let T be the correct required set. Let $q = [t, t+1]\ \epsilon$ T s.t. $s_1\ \epsilon\ q$. Now, since s_1 is the smallest element in X $\rightarrow$ there are no points on the left of s_1 in X $\rightarrow$ $t = s_1$ and therefore, $q = [s_1, s_1 + 1]$. Moving on **inductively**, we get that R is same as T b/c once we remove points from S that lie in $[s_1, s_1 + 1]$ and we proceed inductively by same argument. Hence, R = T.
Algorithm terminates: We start with merge-sort which is a trivial instruction and then we come to 'while loop'. Since we are removing atleast one element(more precisely, atleast smallest one) from S in 'while loop'. Therefore, 'while loop' runs atmost n times.
Time-Complexity:
First we have merge-sort which takes $O(n * log\ n)$. 'While loop' runs for atmost n times which is the worst-case when we only remove one element from S at each iteration $\rightarrow$ there are n intervals in R. Hence, $O(n)$ here. And therefore, $O(n * log\ n + n)$ gives us $O(n * log\ n)$.

Problem 2

Consider the problem of making change from n cents using the fewest coins when the available coins are quarters, dimes, nickels and pennies. Design a greedy algorithm for this problem and prove its correctness. Also analyze the running time of your algorithm.

Solution:
Available coins: quarters(=25 cents), dimes(=10 cents), nickels(=5 cents) and pennies(= 1 cent). Let n be the given no. of cents. Clearly, if $n < 5$, we are forced to use pennies (b/c for a nickel we need 5 cents) and if $5 <= n < 10$, we will go for nickels etc. So, the idea is to use quarter whenever we can else dimes > nickels > pennies acc. to n. Pseudo-code is as follows:
Procedure minimumCoin(n):
coins = 0 # *total no. of coins*
if $n < 5$:
coins = n # *n pennies*
else if $n < 10$:
coins = 1 + n - 5 # 1 *nickel and $n - 5$ pennies*
else if $n < 25$:
coins = n25(n)

```
else:
    q = Math.floor(n/25)
    coins = q # q quarters and ..
    n -= q*25
    if n < 5 :
        coins += n # n pennies
    else if n < 10 :
        coins += 1 + n - 5 # 1 nickel and n - 5 pennies
    else if n < 25 :
        coins += n25(n)
```

```
Procedure n25(n):
  d = Math.floor(n/10)
    coins = d # d dimes and..
    n -= d*10
    if n < 5 :
        coins += n # n pennies
    else if n < 10 :
        coins += 1 + n - 5 # 1 nickel and n - 5 pennies
  return coins
```

Proof of Correctness:
The idea is to use coins s.t. it has the maximum value(cents). Therefore, we use quarters first if possible, then dimes if possible, then nickel if possible, then pennies.
Claim: Our algorithm works in the same manner as the idea suggests.
If no. of pennies are greater than or equal to 25, function minimumCost(n) enters in the conditional statement where $n >= 25$ i.e. else. Similarly, follows for dimes from the code. Q.E.D.

Time-Complexity: $O(1)$ ($\because$ *there are only* $if - else$ *statements.*)

Problem 3

The HAM-PATH problem is the following: Given an undirected graph G, is there a path in G that visits all vertices exactly once. The HAM-CYCLE problem is the following: Given an undirected graph G, is there a cycle in G that visits all vertices exactly once. Give a sketch of a proof that HAM-PATH is in NP. Now show that HAM-PATH is NP-complete by reducing HAM-CYCLE to HAM-PATH. (Assume the HAM-CYCLE is NP-HARD.)

Solution:
a. To prove: HAM-PATH belongs to NP.
Proof: Let G be a graph. It is enough to prove that if there exists a path s.t.

it visits all vertices of G exactly one. Let no. of vertices in G = n. Take any ordering of vertices (an ordered set) C. Now, verify if given ordering is correct i.e. $\exists$ an edge between consecutive vertices in given ordering(or Certificate). Verification of any given ordering can be done in polynomial time. Note: All possible orderings are tried non-deterministically. Hence, overall problem reduces to polynomial time.
Algorithm:
Choose an ordering C=$\{v_1, v_2, ..., v_n\}$ non-deterministically.
for i in range (1, n-1):
  if $\exists$ an edge between v_i and v_{i+1}:
    # do nothing.
  else:
    return false
return true
Hence, it's done in $O(n)$ time i.e. polynomial therefore, HAM-PATH belongs to NP.

b. To prove: HAM-PATH is NP-complete.
Given: HAM-CYCLE is NP-HARD.
Proof: We know that if some language L is in NP and NP-HARD, then L is NP-COMPLETE. $\because$ in part **a.**, we proved that HAM-PATH is in NP. $\therefore$ it is enough to prove that HAM-PATH is NP-HARD. If we can reduce HAM-CYCLE to HAM-PATH, this implies HAM-PATH is NP-HARD because HAM-CYCLE is NP-HARD (given).
Let G be a graph. We will solve HAM-CYCLE problem using HAM-PATH. Now, construct a graph G' as follows:
1) G' = G
2) Pick any arbitrary vertex v from G and add another vertex v' (a copy of v with all edges of v) to G'.
3) Add two more vertices x and y to G' s.t. $\exists$ an edge between v and x, and $\exists$ an edge between v' and y $\rightarrow$ degree of x = degree of y = 1.
We claim that: HAM-PATH in G' $\rightarrow$ HAM-CYCLE in G.
As $\exists$ HAM-PATH in G', it must visit x and y but deg(x) = deg(y) = 1 $\rightarrow$ HAM-PATH begins x and ends with y. As x is connected to v and y is connected to v', removing x and y from G' would change HAM-PATH. Therefore, now HAM-PATH in new G' begins with v and ends with v'. Let e be an edge in HAM-PATH s.t. e connects v' to some vertex t. Now, remove $v' \rightarrow$ HAM-PATH begins with v and ends with t. But v' is a copy of v, therefore $\exists$ an edge from t to v hence, making a cycle in G. Q.E.D.

Problem 4

Given an integer k, divide a set of n objects into k coherent clusters such that spacing, i.e., Min distance between any pair of points in different clusters, is maximized. Write an efficient algorithm for this problem.

Solution:
K-Means Clustering can be used to achieve the required result.
Let given set of points be X=$\{x_1, x_2, ..., x_n\}$. Since we want to divide this set into k coherent clusters, each cluster will have a centroid c_i s.t. c_i belongs to i^{th} cluster (where $1 <= i <= k$). First, we will randomly place our c_is(or initialize them with random co-ordinates) initially. After that, for every x_j we assign a centroid which is nearest(Euclidian distance) to that (where $1 <= j <= n$). Then, we optimize the co-ordinates of each centroid by taking mean of euclidian distance from c_i to x_js (to which c_i belongs) for all dimensions individually. We repeat this until no x_j gets assigned to a new c_i.
Procedure K-Means Clustering Algorithm(X, k):
C : 2-D arraylist s.t. arraylist c[i] stores a list of x_js assigned to i^{th} centroid.
A : 1-D array s.t. X[j] = centroid assigned to j^{th} point in X.
for i in range(1, k):
    c_i=RandomCoordinate()
while (no convergence) :
    for j in range(1,n):
        A[j]=getNearestCentroid(x_j) # *acc. to euclidian dist.*
        C[A[j]].add(x_j)
    for i in range(1, k):
        for each point x in C[i]:
            mean+= getEuclidianDistance(c_i, x) # $for\ each\ dimension\ seperately.$
        mean/=size(C[i])
        c_i = mean (d-Dimension vector)

Problem 5

Prove the cut property and the cycle property for the MST.

Solution:
To Prove: Cut Property for the MST- For any cut C of the graph, if the weight of an edge e in the cut-set of C is strictly smaller than the weights of all other edges of the cut-set of C, then this edge belongs to all MSTs of the graph. (as stated on wikipedia)
Proof:
Let C be a cut of a graph and edge e belongs to the cut-set C s.t. weight of e is strictly smaller than the other edges in cut-set C.
By way of Contradiction, let e does not belong to some MST M. Now, add e to M thereby, creating a cycle in M (Since M is a tree). Now, we remove an edge e' from generated cycle s.t. it also belongs to the cut e belongs to. Since, e and e' belong to the same cut s.t. e is an edge with strictly smaller weight than the other. Hence, after removing e', we get an MST with smaller weight than M which is a contradiction. Q.E.D.

To Prove: Cycle Property for the MST- For any cycle C in the graph, if the weight of an edge e of C is larger than the individual weights of all other edges of C, then this edge cannot belong to a MST. (as stated on wikipedia)
Proof:
Let C be a cycle in a graph and edge e belongs to the cycle C s.t. weight of e is larger than the other edges in cycle C.
By way of Contradiction, let e belongs to some MST M. Removing e from M would disconnect the M into two sub-trees. Since, e belongs to cycle C, there exists an edge e' in C s.t. adding that edge to M would again connect the sub-tress and thereby making a new MST. But weight of e is larger than e' implying new MST has weight less than M which is a contradiction. Q.E.D.

Problem 6

You have 2 random variables X_1 and X_2. You compute 500 Y values following the equation: $Y = aX_1 + bX_2 + c+$ small Gaussian noise where $a = b = c = 2$. Now consider that you do not know values of a, b and c. Consider a, b and c as unknowns. Given the 500 Y values, try to estimate those using gradient descent such that the plane fits the points best.

Solution:
It is a simple problem that can be solved using gradient descent algorithm which essentially minimize the cost function determined by

$$J(a,b,c) = \frac{1}{2*500}\sum_{i=1}^{500}(f_i(a,b,c) - Y_i)^2 = \frac{1}{1000}\sum_{i=1}^{500}(f_i(a,b,c) - Y_i)^2$$

$$Let\ J_a = \frac{\partial J}{\partial a} = \frac{1}{500}\sum_{i=1}^{500}(f_i(a,b,c) - Y_i) * X_{1i},$$

$$J_b = \frac{\partial J}{\partial b} = \frac{1}{500}\sum_{i=1}^{500}(f_i(a,b,c) - Y_i) * X_{2i},$$

$$J_c = \frac{\partial J}{\partial c} = \frac{1}{500}\sum_{i=1}^{500}(f_i(a,b,c) - Y_i)$$

Procedure gradientDescent:
a = RandomNumber(), b = RandomNumber(), c = RandomNumber();
$\alpha = 0.1$ # *learning rate*
while (no convergence):
  a = a - $(\alpha * J_a)$

```
    b = b - (α * J_b)
    c = c - (α * J_c)
return a, b, c;
```