Hector Hernandez
Id:A20391413
CS 430-01.18F: Introduction to Algorithms
Due Date: November 27, 2018

Project

Introduction

For this project both local and greedy algorithms were implemented. However, for the preparation of the report only one algorithm is needed. This report will only refer to the greedy solution.
This solution is composed of the following procedures:

a) getData: This procedure open the instance file and populate the centers and points arrays.

b) addBin: adds element to an sorted linked-list while keeping the order.

c) rmvBin: remove element from an sorted linked-list while keeping the order.

d) dist: returns the distance between a center and a point.

e) greedy: This is the main function in which the greedy solution is implemented.

This algorithm is not always global optimal

An example in which this algorithm would not reach a global optimal is the following.

$$C = \{[1,2,3,4,5],[3,4,5,7],[1,2,6,8],[3,4,6,7],[2,4,8]\}$$

In this case the first center covers 6 points, the second and third cover 4 point each, and the last two centers cover three points each. The minimum number of centers that can be found to cover all points are by inspection centers {2,3}. However, the greedy algorithm would pick centers {1,3,2} in that order of selection. Since the center with more points is the first one, the algorithm will pick that one first. By the second iteration, the algorithm would only check the uncovered values {[],[7],[6,8],[6,7],[8]}. In this case the first best option is center number 3. By the third iteration, the uncovered values are: {[],[7],[],[7],[]}. The first best option for the greedy correspond to the set 2.
The greedy solution presented in the pseudocode below would pick the first largest center that it sees and delete all the points that are covered by it as shown in the previous example.

Algorithm 1 Pseudo-Code: greedy

```
procedure GREEDY(INST)
    procedure ADDGREEDYCENTER(C)
        for p in P2C[c] do
            rmvBin(p,M)
        Q[c] ← 1
        addBin(c,S)
        rmvBin(c,R)
    set P2C,M,R,S,C,P to empty list
    getData(inst,C,P)
    for c = 1 to length(C) do
        R[c] ← c
        maxCover ← [1,0]
        set points to empty list
        cover ← 0
        for p = 1 to length(P) do
            d ← dist(C[c],P[p])
            if d <= 1.0 then
                points[cover] ← p
                cover ← cover+1
        P2C[c] ← points
        if cover > maxCover[2] then
            maxCover[1] ← c
            maxCover[2] ← cover
    for p = 1 to length(P) do
        M[p] ← p
    addGreedyCenter(maxCover[1])
    done ← False
    while not done do
        maxCover ← [1,0]
        for c = 1 to length(R) do
            cover ← 0
            for p = 1 to length(M) do
                d ← dist(C[R[c]],P[M[p]])
                if d <= 1.0 then
                    cover ← cover+1
            if cover > maxCover[2] then
                maxCover[1] ← R[c]
                maxCover[2] ← cover
        if maxCover[2] == 0 then
            done ← True
        if not done then
            addGreedyCenter(maxCover[1])
    set answer to empty list
    for s = 1 to length(S) do
        answer[s] ← S[s]
    return answer
```

Time Complexity

Considering that the input (n) is made of the number of centers (c) and a number of points (p), $n = p + c$. So,

$$O(n) = O(p) \ \ ; \ \ O(n) = O(c) \ \ ; \ \ O(n) = O(p + c)$$

i In lines {2-7} of the greedy solution, the implementation is at most linear since it iterates through all points in one particular center and at most a center can contain all points. Line 6 and 7 perform binary search addition and deletion of items. If adding and removing an item given an index is performed in constant time, then $O(\{2 - 7\}) = O(p + lg(c)) = O(n)$. In the case in which the addition or deletion of a given item is not constant time (simple linked-list implementation), the time complexity of these lines in the worst case would be:

$$O(\{2 - 7\}) = O(p + clg(c)) = O(nlg(n))$$

ii In lines {8,27,43,46} of the greedy solution, are all constant time operations.

$$O(\{8, 27, 43, 46\}) = O(1)$$

iii Line 9 of the greedy solution, makes a call to the `getData` procedure. This procedure is linear iterating to all n elements of the instance file. Therefore, line 9 of the greedy solution is

$$O(\{9\}) = O(c + p) = O(n)$$

iv In lines {10-14,21-23} of the greedy solution, there are constant time operations iterating only through centers.

$$O(\{10 - 14, 21 - 23\}) = O(c) = O(n)$$

v In lines {15-19} of the greedy solution, iterate only through the points but the iteration is performed for each center. Line 16 has a call to the `dist` procedure, but this function is clearly a constant time operation (given that the implementation of the squared root used by the system is linear).

$$O(\{15 - 19\}) = O(c) * O(p) = O(n)^2$$

vi In lines {24-25} of the greedy solution, there are constant time operations iterating only through points.

$$O(\{24-25\}) = O(p) = O(n)$$

vii In lines {28-29,39-42} of the greedy solution, the number of iterations is proportional to the size of output. For a greedy solution as shown in class, this size would be $|OPT|(1+ln(n))$

$$O(\{28-29,39-42\}) = O(|OPT|(1+ln(n))) = O(ln(n))$$

viii In lines {28-29,39-42} of the greedy solution, the number of iterations is proportional to the size of output. For a greedy solution as shown in class, this size would be $|OPT|(1+ln(n))$. For the time complexity it is important to recognize that there is a call to procedure `addGreedyCenter` in line 42. In the worst case scenario, this procedure takes $O(nlg(n))$, therefore:

$$O(\{28-29,39-42\}) = O(|OPT|(1+ln(n)) * nlg(n)) = O(nlg(n)^2)$$

ix Lines {30-31,36-38} of the greedy solution iterate through the subset of centers that still not chosen (c in the worst case). Since this is performed during the while loop started in line 28, it will iterate $bln(n) * c$ the time complexity of this operation would be:

$$O(\{30-31,36-38\}) = O(|OPT|(1+ln(n))) * O(c) = O(nln(n))$$

x Finally, Lines {30-35} of the greedy solution iterate through the subset of points that remained not covered (p in the worst case). Since this is performed during the while loop started in line 28, and for each center (line 30) these lines time complexity would be:

$$O(\{30-35\}) = O(|OPT|(1+ln(n))) * O(c) * O(p) = O(n^2 ln(n))$$

In the time complexity analysis performed above, the bottle neck is definitely between lines {30-35}. Therefore, the overall time complexity of this solution is:

$$O(n^2 ln(n))$$

Apendix: Pseudo code for other procedures

Algorithm 2 addBin

```
procedure addBin(val,E)
    l ← 1
    r ← lenght(E)
    if r == 0 then
        E[1] ← val
        return
    if val < E[l] then
        insert(idx=l,val,E)
        return
    if val > E[r] then
        E[r+1] ← val
        return
    while l<r do
        m ← floor((l+r)/2)
        if E[m] > val then
            r ← m
        if l == r-1 then
            if E[l]< val then
                insert(idx=l+1,val,E)
                return
        else
            if E[m]< val then
                l ← m
                if l == r-1 then
                    if E[r]> val then
                        insert(idx=r,val,E)
                        return
        if E[m]==val or E[l]==val or E[r]==val then
            return
```

Algorithm 3 rmvBin

```
procedure rmvBin(val,E)
    l ← 1
    r ← lenght(E)
    if r == 0 then
        return
    if val < E[l] then
        return
    if val > E[r] then
        return
    while l<=r do
        m ← floor((l+r)/2)
        if E[m] == val then
            delete(E[m])
            return
        if E[l]==val then
            delete(E[l])
            return
        if E[r]==val then
            delete(E[r])
            return
        if E[m] > val then
            r ← m
            if l==r-1 then
                if E[l] < val then
                    return
        else
            if E[m] < val then
                l ← m
                if l==r-1 then
                    if E[r] > val then
                        return
```

Algorithm 4 getData

```
procedure GETDATA(INST,C,P)
    file ← open(concatenate("./instances/instance",inst,".txt"))
    nCenters ← asNumber(file.readline())
    for x = 1 to nCenters do
        C[x] ← asDouble(split(file.readline(),separator=" "))
    nPoints ← asNumber(file.readline())
    for x = 1 to nPoints do
        P[x] ← asDouble(split(file.readline(),separator=" "))
```

Algorithm 5 dist

```
procedure DIST(C,P)
    d ← $sqrt((c[1] - p[1]) * (c[1] - p[1]) + (c[2] - p[2]) * (c[2] - p[2]))$
    return d
```