A LaTeX Article

2009-11-23

1 Running Time Analysis

1.1 NearestNeighbor

	NearestNeighbor(P)	
1	$ls \leftarrow$ **LexicographicSmallest**(P)	$O(n)$
2	$p \leftarrow P[ls]$	$O(1)$
3	**for** $i \leftarrow 0$ **to** $\|P\| - 2$ **do**	$O(n^2)$
4	$P \leftarrow P \backslash \{p\}$	$O(1)$
5	$d \leftarrow \infty$	$O(1)$
6	$q \leftarrow p$	$O(1)$
7	**for** $j \leftarrow 0$ **to** $\|P\| - 1$ **do**	$O(n)$
8	**if distance**$(p, P[j]) < d$ **then**	$O(1)$
9	$d \leftarrow$ **distance**$(p, P[j])$	$O(1)$
10	$q \leftarrow P[j]$	$O(1)$
11	**fi**	
12	**od**	
13	$S \leftarrow S \cup \{p\}$	$O(1)$
14	$p \leftarrow q$	$O(1)$
15	**od**	
	Total	$O(n^2)$

The proof for the running time is of the NearestNeighbor is pretty straight forward. There are only three lines that are not constant time; line 1, line 3 and line 7.
Line 1 uses a function that searches for the lexicographic smallest for its given input set. We store sets in an array, so this can be done in $O(n)$ time.
Line 7 starts a for loop that looks through all remaining elements in the set P looking for the smallest distance between point j and given point p. This comparison is done in constant time and does not affect the running time of the for loop. In the first run, this for loop needs to go through the most number of elements, namely $n - 1$ elements. The running time of this loop is $O(n)$.
Line 3 starts the for loop that runs through all elements in the set, minus the last element. Thus, this takes $O(n - 1)$. We need to take into account the nested for loop and finding the lexicographic smallest point, this makes the total running time:
$O(n-1) * O(n) + O(n) =$
$O(n^2 - n) + O(n) =$
$O(n^2)$
Thus **NearestNeighbor** runs in $O(n^2)$.

1.1.1 DirectedNearestNeighbor

	DirectedNearestNeighbor(P)	
01	$ls, p \leftarrow$ **LexicograhicSmallest**$(P), P[ls]$	$O(n)$
02	**for** $i \leftarrow 0$ **to** $\|P\| - 2$ **do**	$O(n^2)$
03	$P, skip \leftarrow P \backslash \{p\}, false$	$O(2)$
04	**if** $\|A\| \geq 2$ **then**	$O(1)$
05	$skip \leftarrow true$	$O(1)$
06	$r \leftarrow \alpha * d(A[\|A\| - 1], A.last)$	$O(1)$
07	$B \leftarrow$ **FindPointsInRange**(P, p, r)	$O(n)$
08	**if** $\|B\| = 0$ **then**	$O(1)$
09	$skip \leftarrow false$	$O(1)$
10	**elseif** $\|B\| = 1$ **then**	$O(1)$
11	$q \leftarrow B[0]$	$O(1)$
	else	
12	$B \leftarrow \{A.last\} \cap \{A[\|A\| - 1]\} \cup B$	$O(1)$
13	$q \leftarrow$ **GiveBestPoint**(P)	$O(n)$
	fi	
	fi	
14	**if not** $skip$ **then**	$O(1)$
15	$r, q \leftarrow \infty, p$	$O(2)$
16	**for** $j \leftarrow 0$ **to** $\|P\| - 1$ **do**	$O(n-1)$
17	$c \leftarrow d(p, P[J])$	$O(1)$
18	**if** $c < r$ **then**	$O(1)$
19	$r, q \leftarrow c, P[j]$	$O(2)$
	fi	
	od	
	fi	
20	$A, p \leftarrow A \cup q, q$	$O(2)$
	od	
	Total	$O(n^2)$

The overall running time of DirectedNearestNeighbor is determined by three subfunctions and two for loops. We will first explain the factors of consequence, then what situations there are that define the running time. DirectedNearestNeighbor uses two

for loops, the first is on line 02 and goes through $n-1$ input element. The second for loop is on line 16 and takes $O(n-1)$ time, but is not run for each iteration, only for the first and last element.
There are three sub functions that have each have a running time of $O(n)$, LexicographicSmallest (see **??**), FindPointsInRange (see **??**) and GiveBestPoints (see **??**). Only FindPointsInRange and GiveBestPoints are nested in the main for loop and are thus of consequence.
There are to main scenario's when running DirectedNearestNeighbour:

- We are busy with either the first or the input last element.
- We are processing any other input element.

The first situation is the same as running our first NearestNeighbor and is done in $O(n^2)$, it's the second case that has been changed.
FindPointsInRange is always run in this situation, taking $O(n)$ time. It is followed by an if statement, that call the function GiveBestPoint that also runs in $O(n)$. The total running time of is:

$O(n) + (O(n-1) * (3O(n)))$
$O(n) + (O(n^2 - 3n))$
$O(n^2 - 2n)$
$O(n^2)$

Thus, the running time is $O(n^2)$, though in practice, it will be slightly slower than our first algorithm.

1.1.2 FindPointsInRange

This function consists of a single for loop that goes through all of its inputs elements. In the for loop is a single if statement, that checks the distance between the given point and the current point. If the current point is within the given range, it is added to the solution.
This takes $O(n)$ time.

1.1.3 GiveBestPoint

This function consists of a single loop that goes through all of its inputs elements, and several if statements. GiveBestPoints first calculates the angle of it's first line segment than goes into the for loop. The for loop goes through all other points, makes the line segment, calculates the angle and compares it with the current best (and replaces if so). After the for loop it returns the solution.
Because only the for loop takes longer than constant time, the running time of GiveBestPoint is $O(n)$

1.2 ImprovedNearestNeighbor

	ImprovedNearestNeighbor(P)	
01	$ls, p \leftarrow$ **LexicograhicSmallest**$(P), P[ls]$	
02	$start, loopcount \leftarrow p, 0$	
03	**repeat**	$O(n^2)$
04	$\quad loopcount \leftarrow loopcount + 1$	
05	$\quad P \leftarrow P \backslash \{p\}$	
06	$\quad$ **If** $loopcount = 5$ **then**	
07	$\quad\quad P \leftarrow P \cup \{start\}$	
08	$\quad q, d \leftarrow p, \infty$	
09	$\quad$ **for** $j \leftarrow 0$ **to** $\lvert P \rvert - 1$	$O(n)$
10	$\quad\quad c \leftarrow d(p, j)$	
11	$\quad\quad$ **if** $c < d$ **then**	
12	$\quad\quad\quad d, q \leftarrow c, P[j]$	
13	$\quad$ **if** $q \neq start$ **then**	
14	$\quad\quad Result \leftarrow Result \cup \{q\}$	
15	$\quad p \leftarrow q$	
16	**until** $p = start$	$O(n)$
17	**if** $\lvert P \rvert > 0$ **then**	
18	$\quad$ **InsertLostPoints**$(P, Result)$	$O(n)$
	Total: $O(n^2)$	

The overall running time of *ImprovedNearestNeighbor* is $O(n^2)$

2 Up to Five

```
     UpToFiveSort(P)
01   i ← 1
02   while |P| > 0 do
03     Result[i − 1] ← FindCurve(P)
04     for j ← 0 to P[i − 1].count − 1 do
05       p ← Result[i − 1].[j]
06       P ← P\{p}
07     i ← i + 1
08   while |Result| > 5 do
09     smallest, smallestdist ←
       0, Result[0].count
10     for i ← 1 to |Result| − 1 do
11       if Result[i].count <
       smallestnum then
12         smallest ← i
13         smallestnum ←
       Result[i].count
14     p, closest ←
       Result[smallest].[0], smallest
15     closestdist ← ∞
16     for i ← 0 to |Result| − 1 do
17       if i ≠ smallest then
18         d, q ← ∞, p
19         for j ← 0 to Result[i].count
       do
20           c ← d(Result[i], Result[j])
21           if c < d then
22             d, q ← c, Result[i].[j]
23         if d < closestdist then
24           closest, closestdist ← i, d
25     Result[closest] ← InsertLost-
       Points(Result[smallest], Result[closest])
26     for i ← smallest to |Result| − 2 do
27       Result[i] ← Result[i + 1]
```

Total: $O(n^2)$