Fatih Çağatay Gülmez

213962062

CS 372 – Advanced Algorithms

Barış Arslan

**Assignment 1**

**Part I:**

**Pseudo Code:**

bag\_switch:

input:

A list outcome that contains bag draw outcomes.

An integer first:=0 representing first index of sublist.

output:

An integer representing bag switch trial number.

procedure:

array bag := ['W', 'R', 'O']

int i := 0

if outcomes[first + i] in bag then;

return first + I

else then;

do while i >= 0;

if outcome[first + 2\*\* i] in bag then;

if i is 0 then:

return first + 2\*\*i

end if.

int last := first + 2\*\*i

return bag\_switch(outcome, first + 2\*\*(i-1))

end if.

i++

end while.

end if.

**Complexity Analysis:**

For 2^X = N sized array input, algorithm should do X lookups in worst case, which is equal to log2(N). Therefore the complexity of the algorithm is O(logN).

**Python Function:**

Attached in .zip file as “part1.py”.

**Part II:**

**Pseudo Code:**

partition\_list:

input:

A list *numbers* containing input unordered list.

Integers *start*, *stop* representing sublist starting point, sublist endpoint.

A boolean *forward* to check forward checkind status.

Output:

A list which is partitioned by forward checking or backward checking.

Procedure:

int min := 999999

int max := 0

if forward then:

if numbers[start] < numbers[start+1]:

return partition\_list(numbers, start := start+1, forward=True)

else:

int x := start+1

do while x<len(numbers);

if min>numbers[x] then:

min := numbers[x]

end if.

X++

end while.

do while start>0;

if numbers[start]<=min then:

return numbers[:start-1]

end if.

Start--

end while.

Else:

if numbers[stop] > numbers[stop-1]:

return partition\_list(numbers=numbers, stop := stop-1, forward=False)

else:

x := stop-1

while x>0:

if max<numbers[x]:

max := numbers[x]

end if.

x--

end while.

While stop<len(numbers):

if numbers[stop]>=max then:

max := numbers[x]

end if.

Stop++

end while.

End if.

numbers\_left = partition\_list(numbers=numbers, forward = True)

numbers\_right = partition\_list(numbers=numbers, forward = False)

numbers\_middle := numbers[numbers\_left.lenght : numbers.length – numbers\_right.length]

**Complexity Analysis:**

Algorithm does 3N comparisons in worst case. Therefore complexity is O(N).

**Python Function:**

Attached in .zip file as “part2.py”

**Part III:**

**Pseudo Code:**

number\_of\_paths:

input:

A list *G* contains tuple of edges of a graph, sorted according to starting point of edge.

Strings *start*, *stop* which are starting and finish point of graph search.

Output:

An integer *path\_num* representing number of paths.

Procedure:

deque edges := []

list g := []

int path\_num := 0

do for each tuple i in G;

if i[0] == start then:

edges.append(i)

else:

g.append(i)

end if.

end for.

G := g

do while edges.length > 0;

tuple iter := edges.popleft()

if iter[1] == stop then:

path\_num++

end if.

path+=number\_of\_paths(G, iter[1], stop)

end while.

return path\_num

**Complexity Analysis:**

Algorithm needs a Queue data structure to work, so we need to create a appropriate Queue. The worst case of this algorithm is if the input will be a linear graph, then queueing operation need to do N comparison in first recursion. Each step will have minus one coımparison so total comparison number is N+N-1+N-2+…+1 = N(N+1)/2 and complexity is O(N^2).

**Python Function:**

Attached in .zip file as “part3.py”.