ASSIGNMENT 4 – COMP 252

Alexandre St-Aubin and Jonathan Campana

February 26, 2024

1. BROWSING THE SMALL ELEMENTS IN A RED-BLACK TREE.

2. GREEDY ALGORITHM. On a flat table, we have placed n disks of radii $r_1, \ldots, r_n$, numbered from left to right. We push them together without creating overlap, as in the figure below. Give an $O(n)$ time algorithm to compute the size of the smallest axis-aligned rectangle that can hold the disks.

   Solution:
   See the algorithm on the next page. We show that its complexity is $O(n)$. To begin, the outermost loop at line 8 iterates a total of n times. The maximum number of elements pushed onto the stack is also n (since at most one is pushed at each iteration), hence an equal number is removed, maintaining the overall $O(n)$ complexity. Finally, as we traverse through each circle once more at the end of the algorithm, the complexity remains $O(n)$.

Algorithm 1: Greedy circle packing

Input: An ordered list $\Omega := \{1, 2, 3, \ldots, n\}$ of n circles.
Output: The minimum width of a rectangle that can hold the disks.

```
// radius of the circle a
Function Radius(circle a):
    return radius of a;

// distance between the centres of a and b if they are pushed together.
Function centre_dist(circle a, circle b):
    return sqrt((Radius(a) + Radius(b))^2 − (Radius(a) − Radius(b))^2);

// largest subarray of Ω ending with k, decreasing in radii.
MAKENULL(possible_adjacent_stack);

// distance from the left side of the rectangle to circle at index
left_to_circle[] ← new array;

left_to_circle[0] ← 0;
for i ← 1 to n do
    if i = 1 then
        left_to_circle[i − 1] ← Radius(i);
        PUSH(i, possible_adjacent_stack)
    else
        // initialize max distance between left side of rectangle and circle
           i to Radius (i) to account for the case where the circle would
           touch the side.
        max_d ← Radius(i);
        // POP each circle on the stack that's smaller than i.
        while Radius(TOP(possible_adjacent_stack)) ≤ Radius(i) do
            max_d ← max{max_d, left_to_circle(POP(possible_adjacent_stack)) +
             centre_dist(i, possible_adjacent_stack(j))};

        // check first circle that's larger, but keep it on the stack
        max_d ← max{max_d, left_to_circle(TOP(possible_adjacent_stack)) +
         centre_dist(i, possible_adjacent_stack(j))};
        // push i to stack, keeping the decreasing order
        PUSH(i, possible_adjacent_stack);
        // max_d is the distance from the left side of the rectangle to the
           center of circle i when it is pushed as much as possible without
           overlapping
        left_to_circle[i − 1] ← max_d;

// find the width
Width ← 0;
for i ← n − 1 to 0 do
    if left_to_circle[i] + Radius(i) > Width then
        Width ← left_to_circle[i] + Radius(i);

return Width;
```