Advanced Algorithms and Data Structures
Assignment 6

Isabella Odorico	Caroline Kierkegaard	Mikkel Willén	Fie Hammer
jkd427	qlj556	bmq419	gsr530

January 15, 2024

NP-completeness

34.2-10

34.3-6

34.4-3

van Emde Boas Tree

20.3-1

One way of doing this would be to add tuples into the process. If the *Min* and *Max* fields contained a tuple with two elements, one element saving the *Min / Max* element and the other saving the number of instances this *Min / Max* element occurs. This should help the vEB-Tree-Insert from the previous exercise, allow the duplicate element.

Instead of storing just a bit in each of the entries of the array, we could store integers representing how many elements of that value the `vEB` contains, which would make us able to support dublicate keys, for each $u = 2$ `vEB`

20.3-2

We will need to store the satellite data with the `min` value, for any key which is a minimum on some `vEB`, since the key does not appear in the subtree. The rest of the satellite data can be stored alongside the keys of the `vEB` trees of size 2. So each non-summary `vEB` tree, we store a pointer in addition to `min`. If `min` is `NIL`, the pointer also points to `NIL`. If not, the pointer points to the satellite data ssociated with that minimum. In size 2 `vEB` trees, we also need two pointers, for pointing to the minimum's and maximum's satellite data, or `NIL` if they do not exist. If `min` = `max`, the pointers point to the same data.

20.3-3

```
VEB-TREE-CREATE-EMPTY(V, u)
If u = 2
    V <- CreateNewNode()
    V.min <- NIL
    V.max <- NIL
Else
    V ← CreateNewNode()
    V.min <- NIL
    V.max <- NIL
    V.cluster <- CreateArrayWithSizeSqrtU(u)
    V.summary <- NIL
    For each: subtree in V.cluster
        subtree <- VEB-TREE-CREATE-EMPTY(sqrt(u))
Return V
```

20.3-4

`vEB-Tree-Insert:`
Nothing should happen in this case. This is due to an already-existing element will cause recursion and the case won't be matched as there are no subclusters. And so, no modification would happen with this. This procedure ensures that there won't be duplicate elements in the tree.

`vEB-Tree-Delete:`
This will have two different outcomes, depending on whether or not the vEV Tree contains 1 or more elements. The first and simplest case is that the tree only contains 1 element. The element would be deleted should the called element not be present in the tree.
If the tree, however, contains more than just 1 element, the call will, like in Insert, go through recursion and eventually reach the "V.u == 2" line, where it will end up deleting the largest element.

To best use these operations, one could try to introduce an auxiliary structure with the size of 'u'. With this addition, it could be checked whether or not the tree contains the element. This struckture, an array or bit array, should indicate whether or not the element is present, and assessing it would be a constant time operation.

20.3-5

We analysis

$$T(u) \leq T\left(u^{1-\frac{1}{k}}\right) + T\left(u^{\frac{1}{k}}\right) + O(1)$$

We let $T(2^m) = S(m)$ and get

$$S(m) \leq S\left(m\left(1-\frac{1}{k}\right)\right) + S\left(\frac{m}{k}\right) + O(1)$$

then if $k > 2$ the first term will dominate, so by master theorem, we have that

$$S(m) = O(\lg m)$$

which means

$$T = O(\lg(\lg u))$$

as in the original case.

20.3-6

We start by setting

$$n = \frac{u}{\lg(\lg u)}$$

Performing n operations will take

$$c(u + n \lg(\lg u))$$

time for some constant c. We then divide by n using the aggregate amortized analysis, which gives us the amortized cost of each operation to be

$$c(\lg(\lg u) + \lg(\lg u)) = O(\lg(\lg u))$$

we then have that we need

$$n \geq \frac{u}{\lg(\lg u)}$$

Polygon triangulation

part a

part b

3.4

Theorem 3.2, the Art Gallery Theorem, states that: *for a simple polygon with n vertices, [n/3] cameras are occasionally necessary and always sufficient to have every point in the polygon visible from at least one of the cameras.*

But, if we consider a polygon with vertices that has a set of diagonals inside, which divide it into convex quadrilaterals, this might require a different number of cameras. In this scenario, when determining the number of cameras, we can subtract the number of convex quadrilaterals from the total number of vertices to get the number of cameras sufficient to guard p. This can be written as:

$$C = n - k$$

The reason this does not contradict the Art Gallery Theorem, is because this theorem sets an upper limit for cameras needed for a simple polygon, and having quadrilaterals can in cases reduce the number of cameras, since one camera can cover two adjacent vertices. Here the formula above takes the reduction of cameras by the diagonal partition into account.

3.7

When a polygon with n vertices is partitioned into monotone pieces, each piece can either be a monotone increasing or monotone decreasing sequence of verticies. We can denote those as V_{inc} and V_{dec} respectively. The sum of these two are equal to the number of vertices in the polygon, $V_{inc} + V_{dec} = n$.

In order to prove that the sum of the number of vertices of the pieces is O(n), we write $V_{inc} + V_{dec} = O(n)$.

Looking at the monotone increasing and decreasing pieces, we see that for the increasing sequence of vertices, each vertex has a higher y-coodinate than the previous, and for the decreasing, each vertex has a lower y-coordinate than the previous. For simple polygons, they cannot intersect itself, and therefore, both each monotone increasing and monotone decreasing pieces, can add maximum two vertivcies to the overall count, one in the beginning and one in the end.

This means that each monotone increasing piece contributes max 2 x monotone increasing pieces (lets call it P_{inc}) and each monotone decreasing pieces contributes max 2 x monotone decreasing pieces (lets call it P_{dec}). To make this easier for ourselves, we can write it as:

$V_{inc} + V_{dec} \leq 2xP_{inc} + 2xP_{dec}$

Since the number of total monotone pieces is constant for a partition, the sum of the vertices of the pieces is O(n).