AAD Aflevering 6

Andreas Hammer (qgk998), Anders Munkvad (tqs326), Johan Topp (fpg662)

January 2024

Approximation Algorithms

20.3-1

This can be done by increasing the size of the vEB tree.
Since the vEB tree represents integer keys, then rather than having a bit showing if the entry exists or not, then an integer can be stored representing the number of each keys stored.
This would increase the size of the vEB tree with the representation of the int, which would e.g. for a 64 bit representation would increase the space by a constant factor, compared to the bit representation.

20.3-2

Assuming that `inserts` then take *(key,data)* as input, then rather than having a single bit representing if a key exists, then a pointer to the satellite data should be stored.
Since the information regarding keys are stored in the min/max of each level of the vEB tree, as well as in the clusters of size 2, since the original vEB tree does not contain pointers to subsequent clusters, but only min/max of that single bit.
So when inserting a key *k.key* with satellite data pointer *k.data*, then when reaching the point where *k.key* is min or max of a cluster, the *k.data* pointer should be stored as well. So a vEB tree structure would contain: *u, min, min_data, max, max_data, summary, cluster[]*

20.3-3

Assumption that the key size = k so even if the amount of keys are not a power of 2, then u will be a power of 2.

```
u = 2^k
min = $\emptyset$
max = $\emptyset$
summary = $\emptyset$
cluster[] = [None] * $\lceil \frac{k}{2} \rceil$ // python syntax for generating an array
```

20.3-4

For this exercise we are using the pseudocode presented on slide 17 in the lecture 'van Emde Boas Trees'. If we insert an element x that is already in S, then S cannot be empty so line 3 can never be executed. If S.min = S.max then we must have found x since it has to be the only element in the set, and in this case x is neither smaller or larger than itself, so we return nothing in line 7.
x is contained within S so line 8 and 9 will not hold true. Until we reach the subset where the last part of x is contained, the condition on line 11 cannot be true, so we essentially call insert on the low part of x continuously until we reach the subset where the last part of x is contained. We will describe the remainder of x at this point as x_w, here we have the possibilities

that $x_w = S.min = S.max$, in which case nothing happens. $x_w = S.min$ or $S.max$, but not both, then the algorithm will attempt to split x_w into hi and lo, and insert it into the cluster and summary, because none of the previous conditions are met before line 10. If we keep inserting x, the algorithm would keep trying to split x even if it has found it, meaning we could have a single bit, but still trying to split it, resulting in an error.

For delete we use the pseudocode in the book. If we reach a node where condition in line 1 is met, then we delete another element which is not x. Alternatively if we reach the condition on line 4 and it holds true, then line 5 cannot be true since then x would be in the set, which we assumed it wasn't, this means we delete V.max instead which shouldn't have been deleted. All other parts of the code are recursive calls which will in some way or another lead us to one of the base-cases in the end.

To check if something is a member we can keep an array separately of length U, and every time we add or delete an element we can lookup it's position in constant time to check if it already there or not in constant time.

20.3-5

To analyse the running time of the functions in this modification we need to account for the recursion depth both in the summary and in the clusters. Since we have $u^{\frac{1}{k}}$ clusters then the summary must be of that size since each element in the summary corresponds to a cluster. The amount of work in each node of the tree can be done in constant time so we can write a recursion fomula as such,

$$T(u) = T(u^{\frac{1}{k}}) + T(u^{1-\frac{1}{k}}) + 1$$

We want to get this formula on the form of the master theorem to be able to use it. Since we know $u = 2^w$ aka. is always a power of two, we can define another recursion formula which instead takes w as the input. We then get,

$$T'(w) = T(\frac{w}{k})) + T(w(1 - \frac{1}{k})) + 1$$

Since $k > 2$ since we are taking more than the square root we can see that asymptotically the second term will grow much faster than the first so we can focus on that term for the runtime. We can see that this turns into case two of the master theorem where $f(n) = \theta(1)$, and then $T'(w) = O(\log w)$, and since we assumed w to be $\log u = w$, then $T(u) = O(\log\log u)$, which is the same as before.

20.3-6

Creating the tree only happens once, and since it is $O(u)$, then there exists a constant c_1 so $c_1 \cdot u$ is greater than the actual time it takes to create the the vEB tree.
Each operation in the vEB tree is bounded by $O(\lg \lg u$ so for this, there exist a constant c_2 so $c_2 \cdot \lg \lg u$. Setting $c = max\{c_1, c_2\}$ then this c will be a bound for both.
The total time for n operations will therefore be bounded by $c(u + n\lg \lg u)$ where only the second term is depending on n. In order to ignore the first term u, the second term needs to be dominant for the run-time, which will happen when $n\lg \lg u = u$, by dividing on both sides by $\lg \lg u$ we get that $n = \frac{u}{\lg \lg u}$, so for $n \geq \frac{u}{\lg \lg u}$ the run-time is bounded by the amortized time of each operation rather than the creation of the vEB-tree

34.2-10

It follows from CLRS (p. 1064) that $P \subseteq NP \cap co - NP$. Now if we assume that $P = NP$ while $NP \neq co - NP$. Under this assumption, every problem in P is also in NP, and since we have that P is closed under complement (CLRS), it implies that the complements of the problems in P are in NP as well. However, we arrive at a contradiction. Since we assumed that $P = NP$, while $NP \neq co-NP$, NP should be equal to $co-NP$ (since all of the complements of the problems are in NP under this assumption). Hence, if $NP \neq co-NP$, then $P \neq NP$.

34.3-6

We now say that $\emptyset$ and $\{0,1\}^*$ are $\in P$. For the sake of contradiction let's assume that $\emptyset$ is complete in P. Then there must exist a reduction from $\emptyset$ to $\{0,1\}^*$. But since $\emptyset$ consists of nothing and $\{0,1\}^*$ contains everything, there can be no reduction from one to another meaning neither can be complete for P.

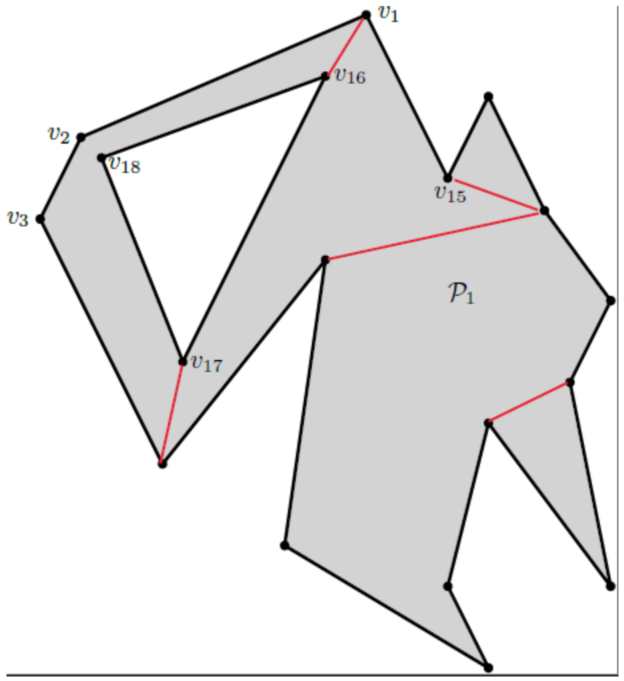
34.4-3

The size of the truth table used in the proof of Theorem 34.10 with n variables is 2^n (CLRS). Since a truth table lists all possible combinations of truth values (namely true or false) for the variables, there will be 2^n rows in the truth table. Deriving the 3-DNF formula equivalent to θ would, in the worst case, need to check all of the rows, hence being 2^n. Hence, by using this method, we would need to create an exponential number of entries in the truth table to perform the reduction, hence why the reduction cannot be performed in polynomial time.

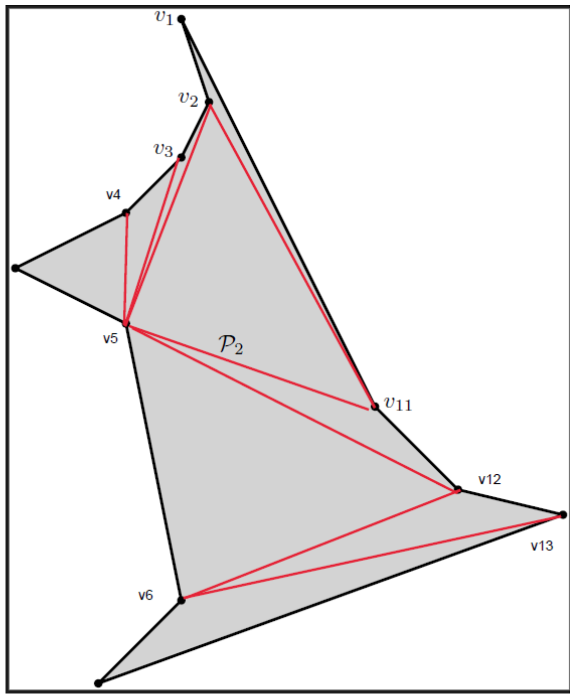
Polygon triangulation

a

For this algorithm we use the pseudocode described in the book page 53 `MakeMonotone(P)`.
We start at v_1 and add it as the helper vertex, when we reach v_16 we need to run `HandleSplitVertex`, we find we insert a diagonal to v_1 since it is the helper of the edge, and add v_16 as helper.
We then reach an unlabelled start vertex and add it as a helper. v_2 is added as a helper but since v_2, v_18, amd v_3 are regular no vertexes are drawn here.
v_15 is a merge vertex so we run `HandleMergeVertex`. There are no previous merge vertexes so we don't draw diagonals, but we insert v_15 as the helper for the left edge. When we reach the regular vertex below it recognizes the helper as a merge vertex and draws a diagonal to it, and becomes the new helper. When we reach the split vertex to the left we draw a diagonal to this edge. For the merge vertex v_17 we add it as the helper to the edge so once we reach the end vertex below we run `HandleEndVertex`, and we see that the end helper to the edge is a merge vertex and so we draw the diagonal.
Finally we have the last bottom split edge which we reach before the previous end vertex, and here the helper vertex to the long edge to the left, is the regular edge to right, so we draw a diagonal. From here there are no more splits or merges, so we finish.



b



When reaching $v5$, then the previous vertices on the left side which can create diagonals will be popped, resulting in diagonals from $\{((v5 - v4), (v5, v3), (v5, v2)\}$. $v2$ and $v5$ will be pushed back.
When reaching $v11$ then $v5 and v2$ will be popped, creating diagonals $(v11, v5)$ and $(v11, v2)$ will be created, and $v5$ will be pushed back.
when reaching $v12$ then $v5$ will be pushed and popped resulting in diagonal $(v12, v5)$.
When $v6$ is reached then $v13$ and $v12$ will be popped, resulting in diagonals $\{((v6, v13), (v6, v12)\}$
The algorithm states that the resulting diagonals from each step are inserted into the algorithm,

but not in which order, so looking at the last step where $\{(v6, v13), (v6, v12)\}$ are inserted, this is seen as simultaneous, but since each vertex is popped, $(v6, v13)$ might be assumed to get inserted before $(v6, v12)$.

CG 3.4

In the case of the art gallery problem, there are no restriction on the placement of the vertices, so it might be concave or convex. This is not the case of the simple polygons which are convex, so any camera placed inside one of the quadrilaterals will be able to observer at least 4 vertices.
By doing the same coloring as in the art gallery problem, it will be possible to select the color that was not represented in the edge between the two vertices, when coloring the next triangle. For the quadrilateral it will be possible to use the two colors not used for the edge, to create a new quadrilateral.
As in the case of the art gallery problem, it is possible to view all parts of the polygon by placing a guard in either of the colors, this is also the case for the quadrilateral, since it is convex. If it had been concave this might not have been the case.
Since each vertex will only have a single color, then $n = n_{c1} + n_{c2} + n_{c3} + n_{c4}$, so the min node, can at most be $\lfloor \frac{n}{4} \rfloor$.
Which does not contradict the Art Gallery Theorem since the triangles cannot always be transformed into convex quadrilateral.

CG 3.7

This can be seen from slide 25 of the lecture "TriangulationMA" where every new diagonal adds 2 new vertices, and there are at most $n-3$ diagonals, resulting in a total of at most $n+2(n-3) = 3n-6$ vertices and $3n-6 = O(n)$, since $c = 3$ would be an upper bound when $n <= 3$ which is the smallest number in order to create triangles.