Quantum Walk Algorithm for Element Distinctness

Jan Luca de Riese

July 2024

Abstract

Ambainis algorithm for element distinctness from his 2007 paper is an influential result many other papers have built upon. Although it comes with significant drawbacks regarding space, it serves as an excellent example to analyze as a quantum algorithm. It runs at the proven lower bound of $\mathcal{O}(N^{2/3})$ and uses clever applications of quantum random graph walks to achieve this speedup.

Contents

1 Introduction

Quantum computers are computers that operate on a fundamentally different paradigm. In contrast to the binary system classical computers use, in which the machines work on zeroes and ones, a quantum computer can take on any number of states in between. Among others, this means that such a computer can have bits which are in a one-state and a zero-state simultaneously, a so-called superposition. By leveraging these superpositions, quantum computers are able to perform calculations on many possible inputs simultaneously, delivering significant speedup. Additionally, one may entangle states, making actions on certain bits influence other bits without direct intervention. However, things are not that simple. Despite these seemingly magical advantages, such computers also bring serious disadvantages. Any state that the quantum computer is in, which is not simply a classical state, cannot simply be looked at and measured. A state is non-classical if it contains some form of e.g. superposition or entanglement. Such actions collapse these states and instead of gaining answers about the calculation result, one gets a random result. This, by itself, would mean quantum computers do not actually speed up calculations at all. By employing clever tricks and letting the quantum bits interact and interfere with each other, one can manipulate the probabilities for which result state the collapse leads to, making significant speedups possible.

The following report is about one such algorithm. This algorithm, published in a 2007 paper [3] by Ambainis entitled "Quantum walk algorithm for element distinctness", solves the problem of element distinctness with significant speedup compared to classical approaches. It uses quantum search on graphs as well as quantum walks as its main components in solving element distinctness. The paper was hugely influential, being cited well over a thousand times. It led to multiple other authors building on top of his algorithm's ideas.

The runtime of Ambainis' algorithm was proven as a lower bound in a paper by Aaronson and Shi[2].

x_1	x_2	x_3	x_4	x_5	$\dots$	x_N
8	5	42	2	42	$\dots$	7

Figure 1: Example for an element distinctness problem.

1.1 Element distinctness

Element distinctness is a basic problem in computer science that involves determining whether a list has any duplicate elements. It is relevant in different applications of database management. For example, a bank could need to check if all transactions in a certain account are unique in order to determine fraud. The basic formulation of the problem involves a list of N items no greater than a number M, with each item bearing the index i and being labeled x_i. For an example, see fig. 1. Such a list of N items is not distinct if there are two equal items with different indices.

A formal definition is given by Definition 1.
Definition 1 (Element Distinctness)**.** *Given numbers $x_1, \dots, x_N \in [M]$, are there two indices $i, j \in [N]$ with $i \neq j$ such that $x_i = x_j$.*

This problem can also be generalized to more than two items. k-element-distinctness is the problem where we decide if a given list (defined similarly to the previous problem) contains k equal items. In the element distinctness problem, this k was set to 2. As this version is closely related to element distinctness, it can be solved by cleverly choosing a subset of the given list and running an algorithm designed for element distinctness on it. A formal definiton for k-element distinctness is as follows:
Definition 2 (k-Element Distinctness)**.** *Given numbers $x_1, \dots, x_N \in [M]$, are there $k > 2$ distinct indices $i_1, \dots, i_k \in [N]$ such that $x_i = \dots = x_k$.*

1.2 Runtime in a quantum world

When talking about runtime in a quantum context, one model that is often used is that of the quantum query model. In this model, runtime is measured by asking how often we query an oracle, meaning how often we need to ask for new variables pertinent to our problem. In the element distinctness case, these would be the actual entries of the list. This quantum query model is used because it allows us to draw sensible conclusions as to the difference that this algorithm has when compared to a classical one. We are then better able

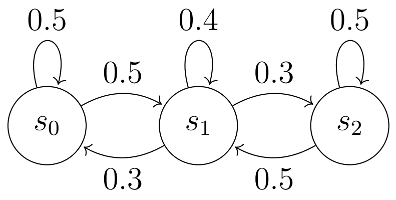

Figure 2: A small example for a markov chain process

to estimate the so called quantum advantage. We define our general problem of element distinctness as a function $f(x_1, \ldots, x_N)$, which is 1 if and only if there are two equal items. To evaluate this function, we need to query the oracle and gain our x_i which we feed into f. Except for these queries, no other unitary operations incur a cost as far as runtime goes.

Without this background, the first approach to element distinctness that comes to mind is simply pairwise comparison (in $\mathcal{O}(n^2)$). With a bit more thought, it quickly becomes clear that sorting first and then comparing neighbors leads to a good speedup already. When measured in the quantum query model, like Ambaini uses, we need to shift our mode of thinking. Sorting and then comparing, uses $\Omega(N)$ queries. This is because we need to query every single element in the list in order to sort it. Ambainis' algorithm, however, runs in $\mathcal{O}(N^{k/(k+1)})$, which is significantly faster. Note that the k here is the k in Element k-Distinctness. For the standard element distinctness problem with $k = 2$, this leads to a runtime of $N^{2/(2+1)} = N^{2/3}$.

2 Quantum Background

In this next section we will introduce the concepts necessary to understand Ambainis' algorithm. Since the algorithms works with quantum walks as well as graph walks, these two concepts will be explained. First, a general explanation of random walks, and then the quantum setting is given.

A random walk is a Markov chain process. A Markov chain is a probabilistic process in which each steps probabilities only depend on the current state, not on the previous steps taken. In Figure 2 one can see an example of a Markov chain. A very simple example of a random walk is called walk on the line. During a walk on the line, we walk over the number line. We start off at number 0 and walk left with probability 1/2 and right with probability

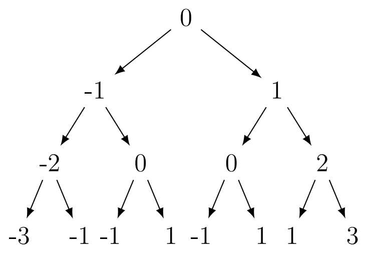

Figure 3: How our state develops during a walk on the line.

1/2. We keep track with a state n, adding or subtracting 1 to and from the state respectively (depending on the outcome of the coinflip). In each step these probabilities stay equal. In Figure 3 we can see the first 3 steps of such a walk. Intuitively it is clear that we most likely stay in, or close to the center. In order to drift to the left or the right swiftly, we would have to choose either of them repeatedly, which is only one out of all the possible paths available and therefore not as probable.

Next, we build the analogous case for a quantum context. The state n which keeps track of our walk's position now becomes a quantum state $|n\rangle$. In a similar transition to the classical case, we can initialy map this state to $|n\rangle \to a\,|n-1\rangle + b\,|n\rangle + c\,|n+1\rangle$. (This example includes the option to move in neither direction but is otherwise identical.) If we take a more detailed look at this, we quickly notice that this is not a valid unitary. It only works if one of our factors (a, b, c) is 1 and the other two are 0.

To solve the problem with our unitary, we add an additional state. We call this the coinflip state C. Its current position is simply appended to our state, making the new statespace $|n, 0\rangle$ and $|n, 1\rangle$ with $n \in \mathbb{Z}$. Now the one step that previously was our transition gets split in two: First we flip the coin and then we shift that result onto n. The coin flip changes the value in the second position of our state with probability 1/2, and leaves it the same with equal probability. In more formal terms, it maps $C\,|n, 0\rangle = \frac{1}{\sqrt{2}}\,|n, 0\rangle + \frac{1}{\sqrt{2}}\,|n, 1\rangle$ and $C\,|n, 1\rangle = \frac{1}{\sqrt{2}}\,|n, 0\rangle - \frac{1}{\sqrt{2}}\,|n, 1\rangle$. After having flipped the coin, we just shift its value onto n: $S\,|n, 0\rangle = |n-1, 0\rangle$ and $S\,|n, 1\rangle = |n+1, 0\rangle$. Now we have all the components for our random walk. A step of walk looks like this: SC. Regarding the probabilites for the flip, an astute observer might have noticed that the probabilities in C are the entries of the Hadamard-matrix. This would therefore be an intuitive choice for the coin flip matrix. Upon

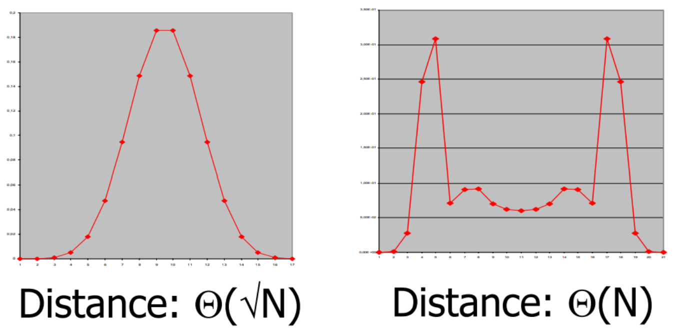

Figure 4: A graph from [1] comparing quantum (r.) and classical (l.) walks. Plotted by repeatedly running a walk for N steps, then graphing the achieved distances.

further inspection however, it is revealed that the different sign on the last factor makes the transition asymmetric. In longer runs, the random walks tend to the left side of the number line [4]. A more balanced matrix than Hadamard would for example be P.

$$P = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

If we are operating on more general graphs, this matrix can be replaced with Grover's diffusion operator ([7]).

If we plot the progression on a number line after X steps, we get a distribution that shows how the classical walk progresses. In the quantum context, it works similarly. Since we cannot measure after every step, we have to run the experiment to completion for every result that we want to measure. Surprisingly, the Distribution of quantum results does not take similar shape to the classical one. Whereas the classical one is concentrated around the middle, the quantum distribution shows two visible 'horns' to the left and right, while sporting a depression in the middle. This means, that the quantum walk spreads far quicker than a classical walk would. In fact, it spreads quadratically quicker.

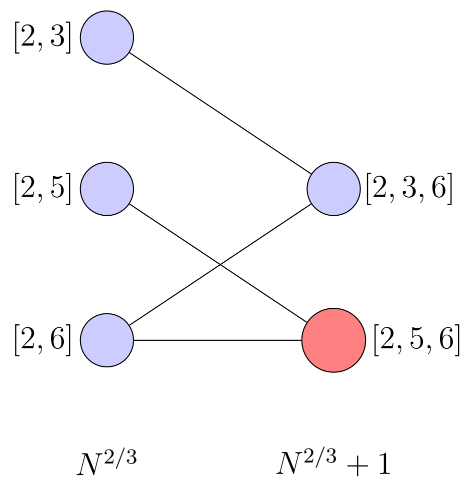

Figure 5: The graph that Ambainis constructs to reduce element distinctness to

3 The Algorithm

In the next section, we will use the aforementioned concepts to construct a graph and describe how we work on it. We will construct the examples for $k = 2$, however they analogously work for other k. Because the algorithm is a random walk algorithm, it takes graphs as input. An example for this graph is shown in figure 5. We first construct the graph that our problem reduces to. This is the graph in figure 5. It is bipartite, and every vertex is labelled. We label all the vertices of the graph with subsets of the list. We keep track of which elements are in these sub-lists, by writing the *indices* of the relevant items into the label. Each subset on the left side of our graph contains $N^{2/3}$ indices, and each subset on the right side $N^{2/3}+1$. Since this number will come up more often for the rest of this report, let r be equal to $N^{2/3}$. Our graph vertices are connected by edges if and only if the labels differ in exactly one element, i.e. if the larger label only adds one element and changes nothing else. Finally, some of our vertices are marked. A vertex is marked if the indices contained in its label include the indices of a valid solution for our element distinctness problem. Our challenge now consists of finding a marked vertex in as little steps as possible.

This graph has $\binom{N}{r} + \binom{N}{r+1}$ vertices, as the binomial factor can be used to compute the sizes of subsets. A formal definition is given by Definition 3.

1. We start with the superposition

$$\frac{1}{\sqrt{\binom{N}{r}(N-r)}} \sum_{|S|=r, y \notin S} |S\rangle |y\rangle .$$

2. For all $i \in S$ for our starting set, query the corresponding x_i. Our state is now:

$$\frac{1}{\sqrt{\binom{N}{r}(N-r)}} \sum_{|S|=r, y \notin S} |S\rangle |y\rangle \bigotimes_{i \in S} |x_i\rangle$$

3. $\mathcal{O}((N/r)^{k/2}) = \mathcal{O}(N^{1/3})$ times:
   a) Apply the unitary to phase flip all $|S\rangle |y\rangle |x\rangle \rightarrow - |S\rangle |y\rangle |x\rangle$ iff S contains i, j where $x_i = x_j$.
   b) Perform $\mathcal{O}(N^{1/3})$ steps of quantum walk.

4. Measure the final state, answer if S contains a doubling of elements or not

Figure 6: The Element Distinctness main algo [3].

Definition 3 (Distinctness Graph)**.** *Given $x_1, \ldots, x_N$, we construct graph $G = (E, V)$ with $V = \{v_S \mid S \subseteq [N], |S| = r\} \cup \{v_T \mid T \subseteq [N], |T| = r + 1\}$ and $E = \{\{v_S, v_T\} \mid T = S \cup \{i\}$ for some $i \in [N]\}$. A vertex V_S is marked if S contains i, j, $i \neq j$, with $x_i = x_j$.*

In the following part, we will define and illustrate the algorithm. It consists of a main loop, and a quantum walk subroutine. We will give definitions for each and then explain them in more detail after.

Similar to basically all quantum algorithms out there, we start off in a superposition. In our case, this is the superposition of all possible state combinations of S and y where S is of size r and y a possible element to add to S. Now in step 2, we query all the x_i corresponding to the is in our starting set S. This makes our superposition contain that information too. Step 3 is the main loop that we go through as often as needed for our algorithm to find a correct solution. Since the algorithms runs off of amplitude amplification, this number of loops is analytically calculated later in the paper. During a single loop, we apply a phase flip to all correct parts of our vector, and then

1. Apply the transformation that maps $|y\rangle$ to $|S\rangle\left(\left(-1+\frac{2}{N-r}\right)|y\rangle+\frac{2}{N-r}\sum_{y'\notin S,y'\neq y}|y'\rangle\right)$.
2. Next we map from S to T, adding y to S and extending x by a zero at location y.
3. We query x_y and insert it into x at y
4. Map $|y\rangle$ to $\left(\left(-1+\frac{2}{r+1}\right)|y\rangle+\frac{2}{r+1}\sum_{y'\in S,y'\neq y}|y'\rangle\right)$.
5. We query for x_y using the x corresponding to the new y. This also erases it.
6. We then map back from T to S by removing the y component from x and y from S.

Figure 7: The Quantum Walk Subroutine [3].

perform $\mathcal{O}(N^{1/3})$ steps of quantum walk. In a final step, we then measure the result and answer whether our problem was distinct or not.

This works similar to Grover's algorithm. Step 3a flips everything that is a good solution and leaves everything orthogonal to it alone. Step 3b we will now explain in more detail.

The quantum walk subroutine in Figure 7 consists of 6 steps. In steps 1-3 we move from the left side of the graph to the right, while in steps 4-6 we move back from the right to the left. The transformations in steps 1 and 3 act as diffusion operators similar to Grover's algorithm. In the context of random walks, they can be conceptualized as coinflips. We therefore start off with a coinflip in step 1. Afterwards, we need to extend our set S by one entry, since we are moving to the vertex group that has set sizes of $r+1$. We then query the corresponding x_i belonging to the i we added. In the second half of the quantum walk, we coinflip once again, and query for x_y using x. This erases x. We then finish up by mapping back from T to S, removing the y-part from x and y itself from S.

In this way we walk over the graph, deciding first what 'direction' to take, and then walking, oscillating back between vertices of label-size r and $r+1$. This walk subroutine itself does not actually take a look at correct solutions. Correct solutions are achieved by amplifying the corresponding amplitude in the previous step. Since the only direction we can walk is along one of the vertices, y represents the index i that we plan on adding or removing from our set S. This corresponds to the set $T = S \cup \{i\}$ that we are moving in the direction of.

4 Analysis

For our second to last section, we describe some of the theoretical results that Ambainis achieves, and the lemmas that help him get there.

We define the state space that the algorithm resides in by defining two Hilbert spaces, $\mathcal{H}$ and $\mathcal{H}'$. These correspond to the vertices with size r and $r+1$ respectively. For $\mathcal{H}$ this means our basis states are $|S, x, y\rangle$, with $S \subseteq [N], |S| = r, x \in [M]^r, y \in [N] \setminus S$. This goes to say that for each subset which could be a label, we define a corresponding basis state. The x contains all the x_i which correspond to the $i \in S$. Finally the state contains the y, representing the state we next want to add to our set S.

For $\mathcal{H}'$ we have basis states $|S, x, y\rangle$ with $S \subseteq [N], |S| = r+1, x \in [M]^{r+1}, y \in S$. Therefore, same definiton as $\mathcal{H}$, but with a set S that is one element larger $(r+1)$, as well as y now being the element that is going to be removed from S next, instead of added to.

We will now present the central theorem that Ambainis proves in order to determine the runtime of his algorithm.

Theorem 1 (Theorem 5 [3]). *Let $x_1, \ldots, x_N$ be such that there exists one set I containing $i_1, \ldots, i_k$ such that $x_{i_1} = \cdots = x_{i_k}$. With a constant probability, measuring after the algorithm gives S such that $I \subseteq S$, meaning we found one such I.*

We start off by dividing our state space in a more sensible manner. Instead of introducing a basis state for all the possible sensible configurations our space can take on, we reduce all states that share a similar behavior under the algorithm. Meaning say they share an equal amplitude when transformed. For k-element distinctness, there exist $2k+1$ such types of states. For the classical problem of element distinctness, this means $k = 2$ and therefore $2k+1 = 5$ different types. These types classify our new basis states. Any state $|S, y\rangle$, $|S| = r, y \notin S$, $|S, y\rangle \in \mathcal{H}$ is in class $(j, 0)$ if $y \notin I$ and in class $(j, 1)$ if $y \in I$. j is the number of $i \in (S \cap I)$. In simpler words, the second (binary) index siginifies whether the current y we are looking at is part of the correct solution, while the first number j denotes how many correct indices we have identified. Identified just means how many of them are in our set S.

We now only analyze the 5-dimensional subspace spanned by the superposition of their respective basis states (for $k = 2$ these are
$|\psi_{(0,0)}\rangle, |\psi_{(0,1)}\rangle, |\psi_{(1,0)}\rangle, |\psi_{(1,1)}\rangle, |\psi_{(2,0)}\rangle$ and we call the space $\tilde{\mathcal{H}}$.) There is no $|\psi_{(2,1)}\rangle$, since $y \notin S \implies y \notin I$. (Analogously for $\tilde{\mathcal{H}}'$.)

Similarly to how we illustrated in our algorithm description for our quantum walk subroutine, where we mapped from S to T, here we map from $\tilde{\mathcal{H}}$ to $\tilde{\mathcal{H}}'$ and back

We take a look at $|\psi_{start}\rangle$ and $|\psi_{good}\rangle$, our starting state and the goal state. One can imagine these similarly to how Grover's algorithm is often illustrated, with $|\psi_{good}\rangle$ being in the place of the y-axis and $|\psi_{start}\rangle$ somewhere almost orthogonal to that. Our algorithm, starting at $|\psi_{start}\rangle$ now aims to move close enough to $|\psi_{good}\rangle$ such that measuring collapses to the correct answer with reasonable probability.

For the next step we define two unitary matrices. U_1 flips the phase on any part of our system that is $|\psi_{good}\rangle$ (3a in the algorithm). While U_2 describes $N1/3$ steps of the quantum walk (3b in the algorithm). This reminds us of the flip and diffuse technique utilized in Grover. Here, U_1 is the flipping operator, while U_2 diffuses. The flip is simple enough, and the diffusion corresponds to the quantum walk, because in walking across the graph in superposition, we "diffuse" our state across possible solutions. Then, describing our algorithm as a transformation $(U_2U_1)^t|\psi_{start}\rangle$, there exists a t such that the inner product of the result of $(U_2U_1)^t|\psi_{start}\rangle$ with $|\psi_{good}\rangle$ is constant. Calculating the t required for constant success leaves us with $t_1 = \mathcal{O}((N/r)^{k/2})$

For this analysis so far, we have taken a look at the query oracle model. In a different model called the comparison query model, we measure how often we compare two values. Converting it to the comparison query model leaves us with $\mathcal{O}(N^{k/(k+1)} \log N)$ queries. If we implement the algorithm and analyze its entire runtime, not just the queries, we need $\mathcal{O}(N^{k/(k+1)} \log^c(N + M))$ steps. This also poses one other central criticism of the paper. Ambainis states, that to implement this algorithm the standard circuit model is augmented with gates for access to a quantum RAM. This quantum RAM is central in order to uphold the algorithm's performance. Without this caveat, we need $\Omega(r)$ instead of $\mathcal{O}(\log N)$ time for a simple data structure operation.

This was also criticized by Grover himself [6], as it substantially slows down the algorithm. Without this supposed access to a QRAM, Dalzell et al. [5] for example outright say that the algorithm is not practical at all.

5 Conclusion

In our final section, we talk about some future developments, some future work that Ambainis' talked about, and summarize the report.

After this paper was published, Magniez et al. used this algorithm to give an $\mathcal{O}(n^{1.3})$ query algorithm for finding triangles in a graph. Ambainis himself used the ideas for a faster 2-dimensional grid search algorithm. Another author, Szegedy, found that for a class of Markov chains, quantum walks are quadratically faster than classical.

Ambainis states that the algorithm uses space to store $\mathcal{O}(N^{2/3})$ items. He postulates about time-space tradeoffs and where there might be gained speed-ups by using more space. Moreover, the k-distinctness algorithm does not yet have proven optimality. For element distinctness the lower bound is proven. Therefore the worst case for k-distinctness would be repeating a normal distinctness $k-1$ times. This would give you a runtime of $\Omega(N^{2/3})$. Finally, quantum walks on other graphs can be explored with similar frameworks.

In conclusion, this algorithm uses the structure inherent to the problem to optimize search in this case. It is a structured search algorithm. We get a general query count of $\mathcal{O}(N^{k/(k+1)})$. Interestingly, by changing the way we mark the goal vertices, it is possible to solve all kinds of 2-subset problems, and not just Element Distinctness.

References

[1] May 9, 2006. URL: `https://qipconference.org/2004/presentations/ambainis.pdf` (visited on 07/16/2024).

[2] Scott Aaronson and Yaoyun Shi. "Quantum lower bounds for the collision and the element distinctness problems". In: *Journal of the ACM (JACM)* 51.4 (2004), pp. 595–605.

[3] Andris Ambainis. "Quantum walk algorithm for element distinctness". In: *SIAM Journal on Computing* 37.1 (2007), pp. 210–239.

[4] Andris Ambainis. “Quantum walks and their algorithmic applications”. In: *International Journal of Quantum Information* 1.04 (2003), pp. 507–518.

[5] Alexander M Dalzell et al. “Quantum algorithms: A survey of applications and end-to-end complexities”. In: *arXiv preprint arXiv:2310.03011* (2023).

[6] Lov Grover and Terry Rudolph. “How significant are the known collision and element distinctness quantum algorithms?” In: *arXiv preprint quant-ph/0309123* (2003).

[7] Lov K. Grover. “A fast quantum mechanical algorithm for database search”. In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC ’96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866. URL: https://doi.org/10.1145/237814.237866.

6 Appendix

1. We define $T_1 = [N]$ and $j = 1$
2. Repeat while $|T_j| > max(r, \sqrt{N})$:
    (a) Run algorithm 2 on $x_i, i \in T_j$. Measure the final state, and check if it includes a k-collision
    (b) Choose q_i as an even power of a prime for which $|T_j| \leq q_j \leq (1 + \frac{1}{2k^2})|T_j|$ holds. Select a random permutation π_j on $[q_j]$
    (c) Define $T_{j+1} = \left\{\pi_1^{-1}\pi_2^{-1} \dots \pi_j^{-1}(i), i \in \left[\lceil \frac{2k}{2k+1} q_j \rceil\right]\right\}$
    (d) And j = j+1
3. If $|T_j| \leq r$, query all $x_i, i \in T_j$ classically. Check if there is a k-collision.
4. If $|T_j| \leq \sqrt{N}$, run Grover search on the set of at most $N^{k/2}$ k-tuples of pairwise distinct i, searching for a tuple of k i's s.t. the corresponding x are equal. Answer if you find the tuple.

Figure 8: Algorithm for k-Element Distinctness. [3]