CSCC73 A4

Saad Makrod

October 2022

Q1 D&C to Search Column and Row Sorted 2D Array

Before writing the algorithm I will define the following terms:

- A is the array we are searching in
- x is the element we are searching for
- UR is the upper right quarter of A
- UL is the upper left quarter of A
- BR is the bottom right quarter of A
- BL is the bottom left quarter of A

Consider the algorithm below:

searchArrayDAndC(A,x**)**

y = middle element of A

if $y == x$

return true

if y is the only element in A and $y \neq x$

return false

if $y > x$

return searchArrayDAndC(UR, x) or searchArrayDAndC(UL, x) or searchArrayDAndC(BL, x)

if $y < x$

return searchArrayDAndC(UR, x) or searchArrayDAndC(BL, x) or searchArrayDAndC(BR, x)

Algorithm Complexity

Note that the algorithm can be complexity can be captured by the following recurrence: $T(n) = 3T(\frac{n}{2}) + c$

This is because for every iteration of the algorithm it can generate up to 3 recursive calls (i.e. 3 sub-problems) of size $\frac{n}{2}$, since we are expressing the running time as a function of the number of rows and columns of A. This leads to $3T(\frac{n}{2})$. Furthermore, clearly it is constant work required to check if we have found the element since we only check with if statements. As a result there is c additional work required.

Thus in this case we have $a = 3$, $b = 2$, and $d = 0$. $a > b^d$ since $3 > 2^0 = 1$. This is the third case of the Master Theorem, and it follows that the algorithm's running time complexity is $\Theta(n^{log_b(a)}) = \Theta(n^{log_2(3)})$. Note that $n^{log_2(3)}$ is roughly $n^{1.585}$ which is clearly more expensive than a $nlog(n)$ algorithm. So if we compare the D&C algorithm above with the Binary Search algorithm proposed earlier, the Binary Search algorithm performs better since it runs in $\Theta(nlog(n))$ time.

Therefore this algorithm will perform in $\Theta(n^{log_2(3)})$ time which is worse than the Binary Search algorithm that performs in $\Theta(nlog(n))$.

Algorithm Justification

The idea behind the algorithm is to discard parts of the array that we do not need to search. Since the array is sorted both row and column wise, we can use this to our advantage. Due to this property, if the middle element is m and $x < m$ we do not need to search the bottom right. Similarly, if $x > m$ we do not need to search the top left. This process can allow us to always discard $1/4$ of A, which improves the complexity from the original n^2. Furthermore, since we only discard areas where we are certain x cannot be eventually we will reach the case of x being the only element in A (assuming it is in A). Once this happens the algorithm returns true as wanted. If x is not found, we know it is not in A because we only discarded areas where it is guaranteed x will not be there. A formal proof can be seen below.

Before proving the algorithm, I will prove two lemmas.

Lemma 1 is: *Let m be $A[\frac{n}{2}][\frac{n}{2}]$. If $x < m$ then x cannot be in BR.*

Lemma 1 Proof

The proof will be done by contradiction,

Assume that this is not the case, x is in BR and $x < m$.

Note that since A is sorted by both column and row so $m \leq A[i][\frac{n}{2}]$ where $\frac{n}{2} \leq i \leq n$. Similarly $m \leq A[\frac{n}{2}][j]$ where $\frac{n}{2} \leq j \leq n$. Since the sorted property is true for all columns and rows we can generalize this further to be $m \leq A[i][j]$ where $\frac{n}{2} \leq i, j \leq n$. This is because all elements to the right or below of an element must be greater than or equal to it.

By transitivity we have $x < m \leq A[i][j]$ where $\frac{n}{2} \leq i, j \leq n$. Thus $x < A[i][j]$. Note that BR is precisely $A[i][j]$ where $\frac{n}{2} \leq i, j \leq n$ since it is defined to be the bottom right quarter of A. Clearly x cannot be in BR. However, we assumed x is in BR so there is a contradiction.

Therefore if $x < m$ then x cannot be in BR.

QED Lemma 1 Proof

Lemma 2 is: *Let m be $A[\frac{n}{2}][\frac{n}{2}]$. If $x > m$ then x cannot be in UL.*

Lemma 2 Proof

The proof will be done by contradiction,

Assume that this is not the case, x is in UL and $x > m$.

Note that since A is sorted by both column and row so $m \geq A[i][\frac{n}{2}]$ where $1 \leq i \leq \frac{n}{2}$. Similarly $m \geq A[\frac{n}{2}][j]$ where $1 \leq j \leq \frac{n}{2}$. Since the sorted property is true for all columns and rows we can generalize this further to be $m \leq A[i][j]$ where $1 \leq i, j \leq \frac{n}{2}$. This is because all elements to the left or above of an element must be less than or equal to it.

By transitivity we have $x > m \geq A[i][j]$ where $1 \leq i, j \leq \frac{n}{2}$. Thus $x > A[i][j]$. Note that UL is precisely $A[i][j]$ where $1 \leq i, j \leq \frac{n}{2}$ since it is defined to be the upper left quarter of A. Clearly x cannot be in UL. However, we assumed x is in UL so there is a contradiction.

Therefore if $x > m$ then x cannot be in UL.

QED Lemma 2 Proof

Algorithm Proof

The proof will be dove by induction on n, note it is assumed that n is always a power of 2.

The basis $n = 1$ is trivial. By the algorithm y will be defined as the middle element of A. Since A has only one element, y is equal to it. If y is the element we are searching for the algorithm returns true, if it is not then the algorithm returns false since it is the only element in A. Thus the algorithm returns the correct result.

Assume that the algorithm performs correctly for $\frac{n}{2}$, we want to show that it performs correctly for n. Note that since n is a power of 2 we consider $\frac{n}{2}$ and n instead of the usual $n - 1$ and n which is common in induction.

If A is nxn first y is defined to be the middle element. If y is the element we are looking for true is returned as desired. If it is not then there are two cases, let x be the element we are looking for, we have $y < x$ and $y > x$.

In the case where $y < x$, the algorithm will perform a search in UR, BL, and BR. By Lemma 2 this is the correct move. Furthermore, note that UR, BL, and BR are all of size $\frac{n}{2}$x$\frac{n}{2}$. So by the induction hypothesis they correctly return true or false depending on whether they contain x. Since the condition returned is an or condition this means that as long as x is found in at least one of these three then the algorithm will return true as wanted. If x is not found in any of these three then false is returned as wanted.

In the case where $y > x$, the algorithm will perform a search in UL, BR, and BL. By Lemma 1 this is the correct move. Furthermore, note that UL, UR, and BL are all of size $\frac{n}{2}$x$\frac{n}{2}$. So by the induction hypothesis they correctly return true or false depending on whether they contain x. Since the condition returned is an or condition this means that as long as x is found in at least one of these three then the algorithm will return true as wanted. If x is not found in any of these three then false is returned as wanted.

Thus all cases perform as expected. This completes the induction step.

QED Algorithm Correctness

Q2 D&C to Search Column and Row Sorted 2D Array

Consider the following algorithm, note that D-SELECT comes straight from the lecture:

D-SELECT(A, k)

if $|A| < 5$ then use brute force and return k smallest element in A

else

partition A into $\lfloor \frac{n}{5} \rfloor$ groups of 5 elements each

find the median of each group

M = list of medians

return D-SELECT(M, $\lceil \frac{|M|}{2} \rceil$)

closePair(S)

if $|S| == 2$ then return the two elements

else

p = D-SELECT(S, $\lceil \frac{|S|}{2} \rceil$)

$S^- = s \in S$ st $s \leq p$

$S^+ = s \in S$ st $s \geq p$

if $ads(S^+) < ads(S^-)$

return closePair(S^+)

else

return closePair(S^-)

Algorithm Complexity

The algorithm running time can be captured by the following recurrence: $T(n) = T(\lfloor \frac{3n}{4} \rfloor) + cn$.

This is because as we proved in lecture D-SELECT will return a p such that $|S^+| < \frac{3n}{4}$ and $|S^-| < \frac{3n}{4}$. Thus each recursive call will result in a problem of size $\frac{3n}{4}$ worst case. As a result we have $T(\lfloor \frac{3n}{4} \rfloor)$. Furthermore, consider the list of additional operations below:

- returning 2 elements is in $O(1)$
- D-SELECT is in $\Theta(n)$ as proved in lecture
- partition S into S^+ and S^- is in $\Theta(n)$
- performing the *ads* operation is $O(n)$ (assuming we cannot compute it in constant time)

As a result we have cn.

We have $a = 1$, $b = \frac{4}{3}$, and $d = 1$. So we have $a < b^d$ since $1 < \frac{4}{3}$. Thus we are in the first case of the Master Theorem and the algorithm is complexity $n^d = n$. Therefore this is a $\Theta(n)$ algorithm.

Algorithm Justification

The idea behind the algorithm is to pick a partition p and then search one of the partitions for a close pair. We only need to search one because of the following lemma: Let p be any element of S, $S_0 = \{x \in S : x \leq p\}$ and $S_1 = \{x \in S : x \geq p\}$. (Note that p belongs to both S_0 and S_1, and so $|S_0| + |S_1| = |S| + 1$.) Let $a_i = ads(S_i)$, for $i \in \{0, 1\}$, and let $m \in \{0, 1\}$ be such that $a_m = min(a_0, a_1)$. Then every close pair in S_m is a close pair in S. Due to this we choose the set with the smaller *ads* value and search there since we know a close pair must exist there. We repeat this until $|S|$ is 2, when this happens the two values in S form the close pair so they can be returned. A formal proof is seen below.

Before proving the algorithm I will prove the following lemma:

Let p be any element of S, $S_0 = \{x \in S : x \leq p\}$ and $S_1 = \{x \in S : x \geq p\}$. (Note that p belongs to both S_0 and S_1, and so $|S_0| + |S_1| = |S| + 1$.) Let $a_i = ads(S_i)$, for $i \in \{0, 1\}$, and let $m \in \{0, 1\}$ be such that $a_m = min(a_0, a_1)$. Then every close pair in S_m is a close pair in S.

Lemma Proof

The proof will be done by contradiction.

Assume that this is not the case and there exists a close pair in S_m that is not a close pair in S. This implies that $a_m > ads(S)$, since otherwise this would be impossible. Since $a_m > ads(S)$ I know that $a_0 > ads(S)$ and $a_1 > ads(S)$.

Note that since the pairs in S_0 and S_1 are mutually exclusive subsets of the pairs in S, I can take the weighted average of S_0 and S_1 to obtain the average in S. That is, the following expression must hold true: $ads(S_0)\frac{|S_0|-1}{|S|-1} + ads(S_1)\frac{|S_1|-1}{|S|-1} = ads(S)$

Note that by definition $a_m = min(ads(S_0), ads(S_1))$. Thus I know that $a_m \frac{|S_0|-1}{|S|-1} + a_m \frac{|S_1|-1}{|S|-1} \leq ads(S_0)\frac{|S_0|-1}{|S|-1} + ads(S_1)\frac{|S_1|-1}{|S|-1}$. By transitivity the following must also hold, $a_m \frac{|S_0|-1}{|S|-1} + a_m \frac{|S_1|-1}{|S|-1} \leq ads(S)$. However, I can evaluate $a_m \frac{|S_0|-1}{|S|-1} + a_m \frac{|S_1|-1}{|S|-1}$ as follows:

$a_m \frac{|S_0|-1}{|S|-1} + a_m \frac{|S_1|-1}{|S|-1} = a_m(\frac{|S_0|-1}{|S|-1} + \frac{|S_1|-1}{|S|-1}) = a_m(\frac{|S_0|-1+|S_1|-1}{|S|-1}) = a_m(\frac{|S|-1}{|S|-1}) = a_m(1) = a_m$

Note that $|S_0| + |S_1| - 1 = |S|$.

Thus I have $a_m \leq ads(S)$. However, this is a contradiction since one of the implications of the supposition is that $a_m > ads(S)$. If $a_m \leq ads(S)$ the it is impossible for there to exist a close pair in S_m that is not a close pair in S.

Thus there is a contradiction. Therefore it is that case that every close pair in S_m is a close pair in S.

QED Lemma Proof

Algorithm Proof

The proof will be done with complete induction on $|S|$, note that for the proof we will let $n = |S|$.

The basis $n = 2$ is straight-forward. Since there are only 2 elements in S they are a close pair in S. The algorithm will return these two elements as wanted.

For the induction step, suppose that the algorithm performs correctly for 1 to $n - 1$. I want to show it performs correctly on n. Since n is greater than 2 the algorithm will use D-SELECT to get p and partition S. From lecture we know that p will be such a value that when S is partitioned, $|S^+| < \frac{3n}{4}$ and $|S^-| < \frac{3n}{4}$.

Note that the recursive call is called on the minimum of $ads(S^+)$ and $ads(S^-)$. By the Lemma this is the correct move since any close pair in that set will also be a close pair in S. Furthermore, we know a close pair must exist in the set because if all points we not close then that means the *ads* value is incorrect, which is not the case. Therefore the algorithm will do the recursive call on the correct subset by the Lemma.

Note that in worst case the size of that set is $\frac{3n}{4}$. By the induction hypothesis the algorithm will correctly return a close pair. This completes the induction step.

QED Algorithm Correctness

It is important to mention that we do not need to consider the case when $|S| < 2$, this is because whenever we partition S, p is always included in the partition. Thus each partition will always have at least 2 elements.