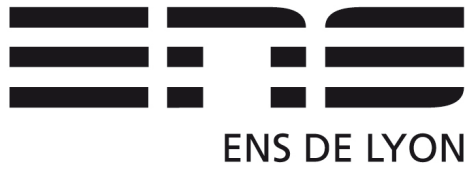


ÉCOLE NORMALE SUPÉRIEURE DE LYON
LATVIJAS UNIVERSITĀTE



UNIVERSITY OF LATVIA
**FACULTY OF
COMPUTING**

M1 INTERNSHIP REPORT

COMPLEXITY OF RECOGNIZING DYCK LANGUAGE
WITH A QUANTUM COMPUTER.

Student :
Maxime CAUTRÈS

Supervisor :
Andris AMBAINIS
Kamil KHADIEV

May the 2nd 2022 - July the 22th 2022

Contents

1	Introduction	2
2	Preliminaries	2
3	A better algorithm for $\text{Dyck}_{k,n}$	2
3.1	A better Complexity Analysis of the original algorithm	2
3.2	A new algorithm for $\text{DYCK}_{2,n}$	3
4	Conclusion	6
5	Appendix	7
A	The algorithm for $\text{Dyck}_{k,n}$	7
B	The proof of the quantum query complexity for $\text{Dyck}_{k,n}$ algorithm's subrou-	
	tines	7

1 Introduction

- presentation of the Internship
- state of the art
- result
- goal to reach

2 Preliminaries

- Quantum Query Complexity
- $DYCK_{k,n}$ problem
- Trichotomy statement
- Adversary methode

3 A better algorithm for $Dyck_{k,n}$

3.1 A better Complexity Analysis of the original algorithm

In the article [1], Andris Ambainis give us a quantum algorithm to recognize the belonging of a n length bit string in $DYCK_{k,n}$ using $O(\sqrt{n}(\log_2(n))^{0.5k})$ quantum queries. But the quantum query complexity for $k = 1$ is not as good as a Grover's search which is sufficient. More precisely, for $k = 1$ the algorithm is searching for a minimal ± 2 string in $1x0$ but we know that every minimal ± 2 string is of size 2. So the logarithmic search of the upper bound on the size of the minimal ± 2 string is no more useful and the algorithm can be summarized to applying a Grover search for 2 consecutive 0 or two consecutive 1. This lower the quantum query complexity of the initial case of the function to $O(\sqrt{n})$ instead of $O(\sqrt{n \log_2(n)})$. This give us this following algorithm for $FINDANY_k$.

Algorithm 1 $FINDANY_k(l,r,s)$

Require: $0 \leq l < r$ and $s \in \{1, -1\}$

if $k > 2$ **then**

Find d in $\{2^{\lceil \log_2(k) \rceil}, 2^{\lceil \log_2(k)+1 \rceil}, \dots, 2^{\lceil \log_2(r-l) \rceil}\}$ such that

$v_d \leftarrow FINDFIXEDLENGTH_k(l, r, d, s)$ is **not** NULL

return v_d or NULL if none

else

Find t in $\{l, l+1, \dots, r\}$ such that

$v_t \leftarrow FINDATLEFTMOST_2(l, r, t, 2, s)$ is **not** NULL

return v_t of NULL if none

The same improvement can be done on $FINDFIXEDPOS_k$ because if $k = 2$ the logarithmic search is useless. So $FINDFIXEDPOS_k$ can be redefined as in ALGORITHM 2. For $k = 2$, the complexity is lowered from $O(\sqrt{\log_2(l-r)})$ to $O(1)$.

This small improvements on the initial cases will improve the global quantum query complexity of each subroutine and finally the quantum query complexity for $DYCK_{k,n}$.

Algorithm 2 FINDFIXEDPOS_k(l, r, t, s)

Require: $0 \leq l < r$, $l \leq t \leq r$ and $s \subseteq \{1, -1\}$

if $k > 2$ **then**

Find d in $\{2^{\lceil \log_2(k) \rceil}, 2^{\lceil \log_2(k)+1 \rceil}, \dots, 2^{\lceil \log_2(r-l) \rceil}\}$ such that

$v_d \leftarrow \text{FINDATLEFTMOST}_k(l, r, t, d, s)$ is **not** NULL

return v_d or NULL if none

else $v \leftarrow \text{FINDATLEFTMOST}_k(l, r, t, 2, s)$ is **not** NULL

return v_d or NULL if none

Theorem 3.1. Dyck_{k,n}'s algorithm correctness *The new definition of FINDANY and FINDFIXEDPOS does not change the behavior the original algorithm.*

Proof Theorem 3.1. The behavior of the DYCK_{k,n} algorithm with the new subroutines is the same than the older one as FINDANY (resp. FINDFIRST) has the same sub-behavior on every entry with its older definition.

Theorem 3.2. Dyck_{k,n}'s Subroutines complexity *The subroutines' quantum query complexity for k are the following.*

1. $Q(\text{DYCK}_{k,n}) = O\left(\sqrt{n}(\log_2(n))^{0.5(k-1)}\right)$ for $k \geq 1$
2. $Q(\text{FINDANY}_{k+1}(l, r, s)) = O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right)$ for $k \geq 1$
3. $Q(\text{FINDFIXEDLENGTH}_{k+1}(l, r, d, s)) = O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right)$ for $k \geq 2$
4. $Q(\text{FINDATLEFTMOST}_{k+1}(l, r, t, d, s)) = \begin{cases} O\left(\sqrt{d}(\log_2(d))^{0.5(k-2)}\right) & \text{for } k \geq 2 \\ O(1) & \text{for } k = 1 \end{cases}$
5. $Q(\text{FINDFIRST}_k(l, r, s, \text{left})) = O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right)$ for $k \geq 2$
6. $Q(\text{FINDFIXEDPOS}_k(l, r, t, s)) = \begin{cases} O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right) & \text{for } k \geq 3 \\ O(1) & \text{for } k = 2 \end{cases}$

Unfortunately, the improvements done on the initial cases of some of the subroutines are not sufficient to get a significant improvement for the quantum query complexity of DYCK_{k,n} algorithm. In order to improve more the query complexity, an other algorithm using a different strategy should be found.

3.2 A new algorithm for Dyck_{2,n}

First, we would like to find an algorithm with a quantum query complexity near to match the lower bound, $\exists c \geq 1$ such that $Q(\text{DYCK}_{k,n}) = \Omega(\sqrt{n}c^k)$, describes by Andris Ambainis team in [1]. This means that we are searching for an algorithm with a quantum query complexity of $O(\sqrt{n})$.

If we come back to the case were $k = 1$, the query complexity comes only from a call to Grover's search because rejecting is easily by finding a 00 or a 11 substrings inside the entry. For $k = 2$ it no more possible as the substrings that reject are of the form 00(10)*0 or of the form 11(01)*1. It implies that the number of calls to Grover's search in the naive approach is in $O(n)$ so the quantum query complexity finally becomes $O(n\sqrt{n})$. In order to keep it in $O(\sqrt{n})$, the algorithm must do a constant number of calls to Grover's search.

For that, we will define a new alphabet that allow to express every even length binary strings and that will have convenient property compatible with Grover's search. Let $\mathcal{A} = \{a, b, c, d\}$ the

alphabet where a corresponds to 00, b to 11, c to 01, and d to 10. So every string of size 2 has its letter in \mathcal{A} thus every even length bit string is expressed in \mathcal{A}^* . This alphabet allow us to prove the following theorem.

Theorem 3.3. *Substrings rejection for Dyck word of height at most 2. A word on the alphabet \mathcal{A} embodies a Dyck word of height at most 2 if and only if it does not contain $aa, ac, bb, bd, cb, cd, da, dc$ as substrings.*

Proof Theorem 3.2. This alphabet \mathcal{A} is important because each of this letter has a height variation in $\{-2, 0, 2\}$. Indeed, a has a 2 height variation, b a -2 , c a 0, and d a zero. This means that after each letter in a word, the current height will be even. Moreover, for a valid Dyck word of height at most 2, after every letter the height will be 0 or 2 which are respectively the lower and upper bound for the height. It means that no letter can cross a border between its two bits.

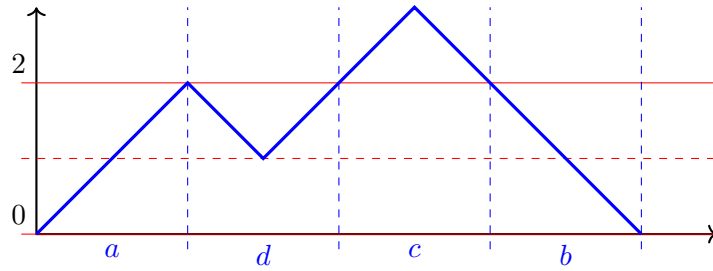


Figure 1: Illustration of the letters of \mathcal{A} using Dyck's representation.

This property is important as it implies that every ± 3 strings uses at least two letters. First, we can get a partition of \mathcal{A}^2 give in Table 1 by checking if each pair as a substring of a word make it not a Dyck word.

Table 1: Partition of \mathcal{A} into \mathcal{X}, \mathcal{V}

\mathcal{X}	$aa \ ac \ bb \ bd \ cb \ cd \ da \ dc$
\mathcal{V}	$ab \ ad \ ba \ bc \ ca \ cc \ db \ dd$

- The set \mathcal{X} . First, every couple of letter such that it contains a ± 3 strings is in \mathcal{X} . This first condition explains the belonging of $aa, ac, dc, da, cb, bb, bd$, and cd . Next, cd and dc belong to \mathcal{X} because of the following property: For any valid Dyck word of height 2, the current height is bounded between 0 and 2, moreover after each letter the current height is even this means that both couple cd and dc start on a boundary and finish on the same one. Because cd and dc are going above and below the height at which they start, both are going outside off the boundary which implies the word can not be a Dyck Word of height at most 2. The Figure 2 shows each couple of \mathcal{X} .
- The set \mathcal{V} . The couple of \mathcal{A} do not implies that the word is not a Dyck word of size 2 if each can fit inside the height boundaries. The Figure 3 shows that every couple not in \mathcal{X} (ie. $ab, ad, ba, bc, ca, cc, db, dd$) fit between height 0 and 2.

So a word, whose letter representation has a substring in \mathcal{X} , cannot be a Dyck word of height two. But does every non Dyck word of height at most 2 have a substring in \mathcal{X} ?

A word is not a dyck word of height 2 if it include a ± 3 strings. But how are represented ± 3 strings using the letters? There are 8 different cases which are 2 by 2 symmetrical so Figure 4 and Figure 5 shows only the cases for $+3$ strings. In Figure 4, every $+3$ string of size 3 is include in aa or ac so it is sufficient to search for this two couple. In Figure 5 every $+3$ strings of length

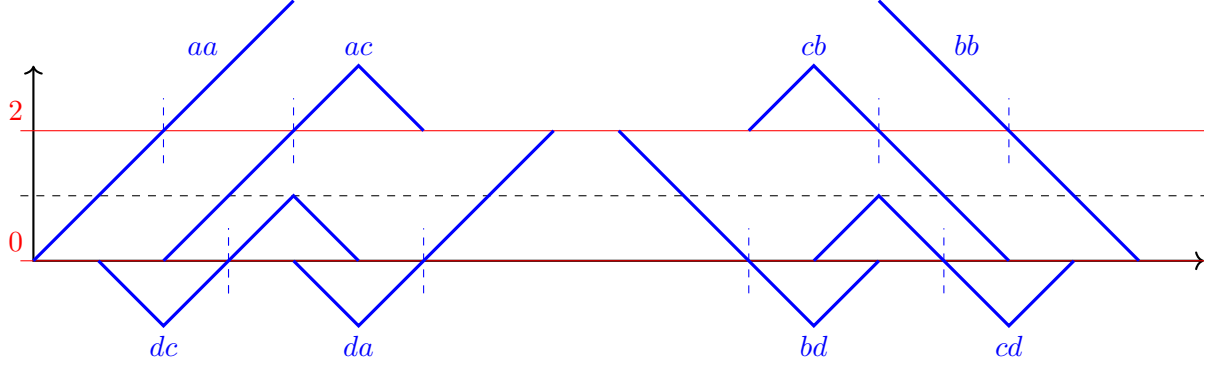


Figure 2: Every 2 letters configuration that implies the word, whom the configuration is a substring, is not a Dyck word of height 2.

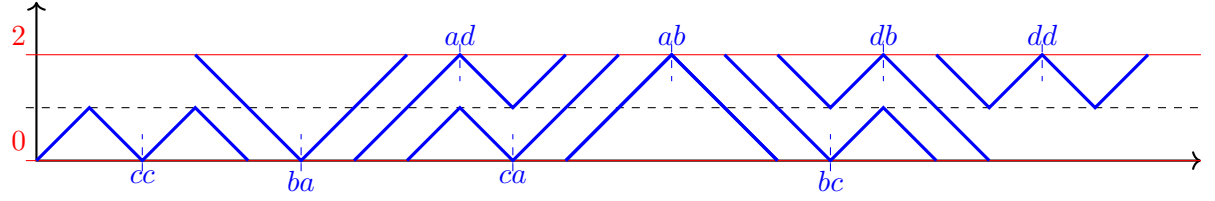


Figure 3: Every 2 letters configuration that can be found in a valid Dyck word of height 2.

greater than 3 are composed of 2 minimal $+2$ strings. This implies that one must be a a while the other must be da or dc . Because da or dc are rejecting substrings, it is sufficient to search for them.

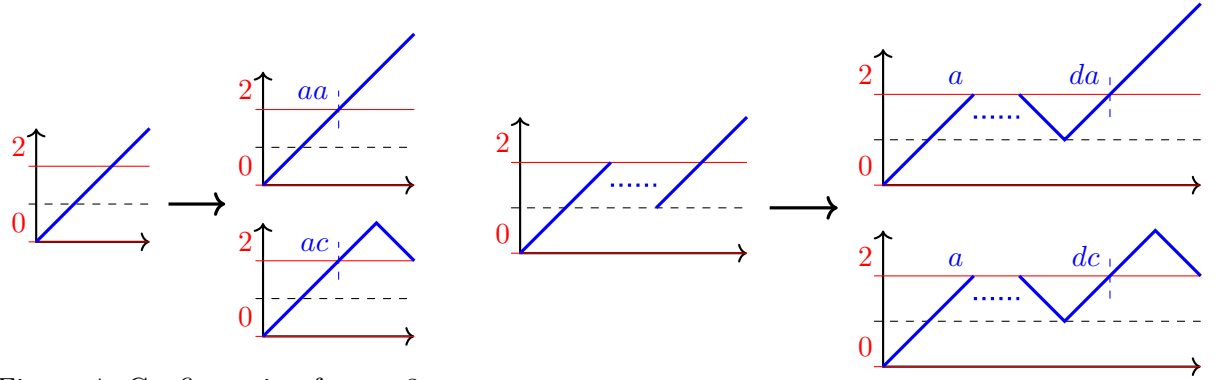


Figure 4: Configuration for a $+3$ strings of size 3.

Figure 5: Configurations for a $+3$ string of size greater than 3.

Finally, A word on the alphabet \mathcal{A} embodies a Dyck word of height at most 2 if and only if it does not contain $aa, ac, bb, bd, cb, cd, da, dc$ as substrings. The following ALGORITHM 3 for $\text{DYCK}_{2,n}$ comes from the direct application of the theorem.

Algorithm 3 DYCKFAST_{2,n}

Require: $n \geq 0$, x such that $|x| = 2n$
 $x \leftarrow 11x00$
 $t \leftarrow \text{NULL}$
for $\text{reject_symbol} \in \{aa, ac, bb, bd, cb, cd, da, dc\}$ **do**
 if $t == \text{NULL}$ **then**
 Find t in $\llbracket 0, n \rrbracket$ such that
 $x[2t, \dots, 2t + 3] = \text{reject_symbol}$
return $t == \text{NULL}$

4 Conclusion

bonsoir paris
 bonnoaisn
 boabdado
 ibabdiubaoiz
 iabdiabdo

References

- [1] Andris Ambainis, Kaspars Balodis, Jānis Iraids, Kamil Khadiev, Vladislavs Kļevickis, Krišjānis Prūsis, Yixin Shen, Juris Smotrovs, and Jevgēnijs Vihrovs. Quantum lower and upper bounds for 2d-grid and dyck language. *Leibniz International Proceedings in Informatics*, 170, 2020.

5 Appendix

The frame of the intership

A The algorithm for Dyck_{k,n}

Algorithm 4 DYCK_{k,n}

;

Require: $n \geq 0$ and $k \geq 1$

Ensure: $|x| = n$

$x \leftarrow 1^k x 0^k$

$v \leftarrow \text{FINDANY}_{k+1}(0, n + 2 * k - 1, \{1, -1\})$

return $v = \text{NULL}$

Algorithm 5 FINDANY_k(l, r, s)

Require: $0 \leq l < r$ and $s \subseteq \{1, -1\}$

Find d in $\{2^{\lceil \log_2(k) \rceil}, 2^{\lceil \log_2(k)+1 \rceil}, \dots, 2^{\lceil \log_2(r-l) \rceil}\}$ such that

$v_d \leftarrow \text{FINDFIXEDLENGTH}_k(l, r, d, s)$ is **not** NULL

return v_d or NULL if none

Algorithm 6 FINDFIXEDLENGTH_k(l, r, d, s)

Require: $0 \leq l < r$, $1 \leq d \leq r - l$ and $s \subseteq \{1, -1\}$

Find t in $\{l, l + 1, \dots, r\}$ such that

$v_t \leftarrow \text{FINDATLEFTMOST}_k(l, r, t, d, s)$ is **not** NULL

return v_t of NULL if none

B The proof of the quantum query complexity for Dyck_{k,n} algorithm's subroutines

Proof Theorem B.1. The proof is done by induction on the height of the Dyck word k .

Initialization: For $k = 1$ and $k = 2$ we have the following initialization.

- For $k = 1$, only FINDATLEFTMOST₂, FINDANY₂, and DYCK_{1,n} are defined. The $O(1)$ quantum query complexity of FINDATLEFTMOST₂ comes directly from the definition of its initial case, as the $O(\sqrt{r-l})$ quantum query complexity of FINDANY₂. Then the $O(\sqrt{n})$ quantum query complexity of DYCK_{1,n} comes from the call to FINDANY₂.
- For $k = 2$, the inductive part of the algorithm start and every subroutines is defined. The $O(1)$ quantum query complexity of FINDFIXEDPOS₂ comes from the call to FINDATLEFTMOST₂. The $O(\sqrt{r-l})$ quantum query complexity of FINDFIRST₂ comes from the dichotomize search using FINDANY₂ and FINDFIXEDPOS₂ because $\sum_{u=1}^{\log_2(r-l)} 2^u \left(O\left(\sqrt{\frac{r-l}{2^{u-1}}}\right) + O(1) \right) = O(\sqrt{r-l})$ (Detailed in the induction). The $O(\sqrt{d})$ quantum query complexity of FINDATLEFTMOST₃ comes from the constant amount of calls to FINDFIRST₂ and FINDATLEFTMOST₂ with entry of size d . The $O(\sqrt{r-l})$ quantum query complexity of FINDFIXEDLENGTH₃ comes

Algorithm 7 FINDATLEFTMOST_k(l, r, d, t, s)

Require: $0 \leq l < r$, $l \leq r \leq r$, $1 \leq d \leq r - l$ and $s \subseteq \{1, -1\}$
 $v = (i_1, j_1, \sigma_1) \leftarrow \text{FINDATLEFTMOST}_{k-1}(l, r, t, d - 1, \{1, -1\})$
if $v \neq \text{NULL}$ **then**
 $v' = (i_2, j_2, \sigma_2) \leftarrow \text{FINDATRIGHTMOST}_{k-1}(l, r, i_1 - 1, d - 1, \{1, -1\})$
 if $v' = \text{NULL}$ **then**
 $v' = (i_2, j_2, \sigma_2) \leftarrow \text{FINDFIRST}_{k-1}(\max(l, j_1 - d + 1), i_1 - 1, \{1, -1\}, \text{left})$
 if $v' \neq \text{NULL}$ and $\sigma_2 \neq \sigma_1$ **then** $v' \leftarrow \text{NULL}$
 if $v' = \text{NULL}$ **then**
 $v' = (i_2, j_2, \sigma_2) \leftarrow \text{FINDATLEFTMOST}_{k-1}(l, r, j_1 + 1, d - 1, \{1, -1\})$
 if $v' = \text{NULL}$ **then**
 $v' = (i_2, j_2, \sigma_2) \leftarrow \text{FINDFIRST}_{k-1}(j_1 + 1, \max(r, i_1 + d - 1), \{1, -1\}, \text{right})$
 if $v' = \text{NULL}$ **then return** NULL
else
 $v = (i_1, j_1, \sigma_1) \leftarrow \text{FINDFIRST}_{k-1}(t, \min(t + d - 1, r), \{1, -1\}, \text{right})$
 if $v = \text{NULL}$ **then return** NULL
 $v' = (i_2, j_2, \sigma_2) \leftarrow \text{FINDFIRST}_{k-1}(\max(t - d + 1, l), t, \{1, -1\}, \text{left})$
 if $v' = \text{NULL}$ **then return** NULL
if $\sigma_1 = \sigma_2$ and $\sigma_1 \in s$ and $\max(j_1, j_2) - \min(i_1, i_2) + 1 \leq d$ **then**
 return $(\min(i_1, i_2), \max(j_1, j_2), \sigma_1)$
else return NULL

Algorithm 8 FINDFIRST_k(l, r, s, left)

Require: $0 \leq l < r$ and $s \subseteq \{1, -1\}$
 $l\text{Border} \leftarrow l, r\text{Border} \leftarrow r, d \leftarrow 1$
while $l\text{Border} + 1 < r\text{Border}$ **do**
 $\text{mid} \leftarrow \lfloor (l\text{Border} + r\text{Border})/2 \rfloor$
 $v_l \leftarrow \text{FINDANY}_k(l\text{Border}, \text{mid}, s)$
 if $v_l \neq \text{NULL}$ **then** $r\text{Border} \leftarrow \text{mid}$
 else
 $v_{\text{mid}} \leftarrow \text{FINDFIXEDPOS}_k(l\text{Border}, r\text{Border}, \text{mid}, s, \text{left})$
 if $v_{\text{mid}} \neq \text{NULL}$ **then return** v_{mid}
 else $l\text{Border} \leftarrow \text{mid} + 1$
 $d \leftarrow d + 1$
return NULL

Algorithm 9 FINDFIXEDPOS_k(l, r, t, s)

Require: $0 \leq l < r$, $l \leq t \leq r$ and $s \subseteq \{1, -1\}$
Find d in $\{2^{\lceil \log_2(k) \rceil}, 2^{\lceil \log_2(k)+1 \rceil}, \dots, 2^{\lceil \log_2(r-l) \rceil}\}$ such that
 $v_d \leftarrow \text{FINDATLEFTMOST}_k(l, r, t, d, s)$ is **not** NULL
return v_d or NULL if none

from the $O\left(\sqrt{\frac{r-l}{d}}\right)$ calls to FINDATLEFTMOST_3 . The $O\left(\sqrt{(r-l)\log_2(r-l)}\right)$ quantum query complexity of FINDANY_3 comes from the $O\left(\sqrt{\log_2(r-l)}\right)$ calls to FINDFIXEDLENGTH_3 . Finally, the $O\left(\sqrt{(r-l)\log_2(r-l)}\right)$ quantum query complexity of DYCK_2 comes from the call to FINDANY_3 .

Induction: Let suppose it exists k such that Theorem 3.2 is true for k . Let prove that it is true for $k+1$.

First, the $O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right)$ quantum query complexity of $\text{FINDFIXEDPOS}_{k+1}$ comes from the $O\left(\sqrt{\log(r-l)}\right)$ calls to $\text{FINDATLEFTMOST}_{k+1}$.

$$\begin{aligned} Q(\text{FINDFIXEDPOS}_{k+1}(l, r, t, s)) &= O(\sqrt{\log(r-l)}) \times O(Q(\text{FINDATLEFTMOST}_{k+1}(l, r, t, d, s))) \\ &\stackrel{IH}{=} O\left(\sqrt{\log(r-l)} \times \sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right) \\ &= O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right) \end{aligned}$$

Thus the $O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right)$ quantum query complexity of FINDFIRST_{k+1} comes from the dichotomize search using calls to FINDANY_{k+1} and $\text{FINDFIXEDPOS}_{k+1}$.

$$\begin{aligned} Q(\text{FINDFIRST}_{k+1}(l, r, t, d, s)) &= \sum_{u=1}^{\log_2(r-l)} 2u \times O\left(Q(\text{FINDANY}_{k+1}(0, \frac{r-l}{2^{u-1}}, s))\right) \\ &\quad + \sum_{u=1}^{\log_2(r-l)} 2u \times O\left(Q(\text{FINDFIXEDPOS}_{k+1}(0, \frac{r-l}{2^{u-1}}, _, s, left))\right) \\ &\stackrel{IH}{=} O\left(\sum_{u=1}^{\log_2(r-l)} 2u \times \sqrt{\frac{r-l}{2^{u-1}}}(\log_2(\frac{r-l}{2^{u-1}}))^{0.5(k-1)}\right) \\ &= O\left(\sum_{u=1}^{\log_2(r-l)} 2u \times \sqrt{\frac{r-l}{2^{u-1}}}(\log_2(r-l))^{0.5(k-1)}\right) \\ &= O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)} \sum_{u=1}^{\log_2(r-l)} u \times \left(\frac{1}{\sqrt{2}}\right)^{u-1}\right) \\ &\stackrel{a}{=} O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)} \frac{\sqrt{2}^2}{(\sqrt{2}-1)^2}\right) \\ &= O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right) \end{aligned}$$

Next, the $O\left(\sqrt{d}(\log_2(d))^{0.5(k-1)}\right)$ quantum query complexity comes of $\text{FINDATLEFTMOST}_{k+2}$ from the constant amount of calls to $\text{FINDATLEFTMOST}_{k+1}$, $\text{FINDATRIGHTMOST}_{k+1}$, and FINDFIRST_{k+1} .

$$\begin{aligned} Q(\text{FINDATLEFTMOST}_{k+2}(l, r, t, d, s)) &= 3 \times O(Q(\text{FINDATLEFTMOST}_{k+1}(l, r, t, d, \{1, -1\}))) \\ &\quad + 4 \times O(Q(\text{FINDFIRST}_{k+1}(l, r, \{1, -1\}, left))) \\ &\stackrel{IH}{=} O\left(\sqrt{d}(\log_2(d))^{0.5(k-1)}\right) \end{aligned}$$

$$^a \sum_{u=1}^{+\infty} \left(\frac{d}{dx}(x^u)\right) \left(\frac{1}{\sqrt{2}}\right) \leq \left(\frac{d}{dx} \left(\sum_{u=1}^{+\infty} x^u\right)\right) \left(\frac{1}{\sqrt{2}}\right) \leq \left(\frac{d}{dx} \left(\frac{x}{1-x}\right)\right) \left(\frac{1}{\sqrt{2}}\right) \leq \left(\frac{1}{(1-x)^2}\right) \left(\frac{1}{\sqrt{2}}\right) \leq \frac{1}{(1-\frac{1}{\sqrt{2}})^2} \leq \frac{\sqrt{2}^2}{(\sqrt{2}-1)^2}$$

After that, the $O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right)$ quantum query complexity of $\text{FINDFIXEDLENGTH}_{k+2}$ comes from the $O\left(\sqrt{\frac{r-l}{d}}\right)$ calls to $\text{FINDATLEFTMOST}_{k+2}$.

$$\begin{aligned} Q(\text{FINDFIXEDLENGTH}_{k+2}(l, r, d, s)) &= O\left(\sqrt{\frac{r-l}{d}}\right) \times O(Q(\text{FINDATLEFTMOST}_{k+2}(l, r, t, d, s))) \\ &= O\left(\sqrt{\frac{r-l}{d}} \times \sqrt{d}(\log_2(d))^{0.5(k-1)}\right) \\ &= O\left(\sqrt{r-l}(\log_2(d))^{0.5(k-1)}\right) \\ &= O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right) \end{aligned}$$

Hence the $O\left(\sqrt{r-l}(\log_2(r-l))^{0.5k}\right)$ quantum query complexity of FINDANY_{k+2} comes from the the $O\left(\sqrt{\log_2(r-l)}\right)$ calls to $\text{FINDFIXEDLENGTH}_{k+2}$

$$\begin{aligned} Q(\text{FINDANY}_{k+2}(l, r, s)) &= O\left(\sqrt{\log(r-l)}\right) \times O(Q(\text{FINDFIXEDLENGTH}_{k+2}(l, r, d, s))) \\ &= O\left(\sqrt{\log(r-l)} \times \sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right) \\ &= O\left(\sqrt{r-l}(\log_2(r-l))^{0.5k}\right) \end{aligned}$$

Finally, the $O\left(\sqrt{n}(\log_2(n))^{0.5k}\right)$ quantum query complexity of $\text{DYCK}_{k+1,n}$ comes from the call to FINDANY_{k+2} .

$$\begin{aligned} Q(\text{DYCK}_{k+1,n}) &= O(Q(\text{FINDANY}_{k+2}(0, n+2k+1, s))) \\ &= O(Q(\text{FINDANY}_{k+2}(0, n, s))) \\ &= O\left(\sqrt{n}(\log_2(n))^{0.5k}\right) \end{aligned}$$

Conclusion: By the induction principle we get that the Theorem 3.2 is true for $k \in \mathbb{N}^*$