ÉCOLE NORMALE SUPÉRIEURE DE LYON

LATVIJAS UNIVERSITĀTE

M1 INTERNSHIP REPPORT

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

- Contexte du stage
- Histoire de l'informatique quantique
- présentation du model des quantum query
- présentation des mot des dyck
- présentation de ce qui a été fait sur le sujet
- présentation de l'ojectif du stage
- présentation des résultat

Context of the internship

As part of the first year of Master at the École Normale Supérieure de Lyon, I was able to do a 12 weeks research internship in a laboratory.

My research for an internship in Quantum Algorithmic have brought me to the Faculty of Computing at the University of Latvia and my supervisor Andris Ambainis. My research also brings me to discuss with Kamil Khadiev from Kazan Federal University who has become my co-supervisor. We have discussed by email to find an interesting subject of research on which I have liked to work on. I thank them for their help, their supervision and the time they have given to me during this 12 weeks.

During the internship, I have been integrated to the life of the Center for Quantum Computing Science. I thanks members of the team for the great discussions we had after the seminar.

I also want to thank Omar Fawzi for having introduced me to quantum computing and its fascinating possibilities.

The team's research area is quantum algorithms and complexity theory. More precisely, the team works on establishing new quantum algorithm with better complexity and proving new lower bound to the quantum complexity for many different type of problem belonging from graph theory to cryptography passing by language recognition theory. My work on the recognition of restricted Dyck words integrate itself great in the team work as it has already been studied by the team few years ago [1] and further by Kamil Khadiev [4].

My internship, named "Complexity of recognizing Dyck language with a quantum computer", as for goal to reduce the gap between the lower and the upper bound for a quantum query algorithm that recognizes Dyck words of bounded height. The best lower and upper bound are describe in [1] by Andris Ambainis team.

In the end of the introduction the field of research will be presented more precisely. After that, technical preliminaries, which are useful to understand the current and the new results, ill be detailed. Finally, the last section presents the new results on the problem and the failures.

1.1 History of quantum computing

The history of quantum computing has started in 1980 when Paul Benioff, an american physicist, proposed a quantum mechanical model of the Turing machine [2]. This machine use some properties of the matter that has been discovered by quantum physicist. After that, some computer scientists suggested that the quantum model of turing machine may be more expressive that the classical model. Few years after, the first bricks of the quantum circuit have been introduced by Richard Feymann [3]. The first quantum computers have started to arrived middle of 1990s. During the last 20 years, the founds given to the creation of the first quantum computer have skyrocketed, as the number of start-up and company dedicated to it. This emulation has made from the quantum computer field one of the most active field of research today. On the algorithmic side, the first astonishing result is the algorithm designed by Peter Shor (1994) [5]. The algorithm improves a lot the complexity of factorizing integers, enough to break our cryptographic protocols when quantum computer will be powerful enough. Since 1994, the quantum algorithm area has evolved almost independently from the quantum computers and has developed many beautiful theories and interesting results. But how does a quantum circuit works ?

1.2 The quantum circuit and quantum query model

In classical computer science, the piece of information is represented by using 0 and 1. This two states can be easily obtained using electricity with 0 equal to 0V and 1 equal 5V. It is easy to propagate electricity through wires and to stock its level into capacitor. Moreover a little piece of hardware, named transitor, has allow to do some computations using logical gates which when include in a complex machine create our so-called "computers".

In quantum computer, the story isn't so different. First, the 0 and 1 are now represented using particles like electrons or photons. For example, an electron with a spin of $+\frac{1}{2}$ (note $|1>$) represents a 1 and an other one with a spin of $-\frac{1}{2}$ (note $|0>$) represents a 0.

2 Preliminaries

- language sans étoile
- Trichotomy theorem
- LOwer bound
- Upper bound

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 $\textsc{Dyck}_{k,n}$ 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 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 $\textsc{FindAny}_k$.

The same improvement can be done on $\textsc{FindFixedPos}_k$ because if $k = 2$ the logarithmic search is useless. So $\textsc{FindFixedPos}_k$ can be redefined as in $\textsc{Algorithm}$ 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 $\textsc{Dyck}_{k,n}$.

Algorithm 1 FindAny$_k$(l,r,s)

Require: $0 \le l < r$ and $s \subseteq \{1, -1\}$
  if $k > 2$ **then**
    Find d in $\{2^{\lceil \log_2(k) \rceil}, 2^{\lceil \log_2(k)+1 \rceil}, \ldots, 2^{\lceil \log_2(r-l) \rceil}\}$ 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 $\{l, l+1, \ldots, r\}$ such that
      $v_t \leftarrow$ FindAtLeftMost$_2(l, r, t, 2, s)$ is **not** Null
    return v_t of Null if none

Algorithm 2 FindFixedPos$_k(l, r, t, s)$

Require: $0 \le l < r$, $l \le t \le r$ and $s \subseteq \{1, -1\}$
  if $k > 2$ **then**
    Find d in $\{2^{\lceil \log_2(k) \rceil}, 2^{\lceil \log_2(k)+1 \rceil}, \ldots, 2^{\lceil \log_2(r-l) \rceil}\}$ such that
      $v_d \leftarrow$ FindAtLeftMost$_k(l, r, t, d, s)$ is **not** Null
    return v_d or Null if none
  else $v \leftarrow$ 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 as other subroutines (Appendix A) stay unchanged.*

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* $k \ge 1$
2. $Q(\text{FindAny}_{k+1}(l, r, s)) = O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right)$ *for* $k \ge 1$
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* $k \ge 2$
4. $Q(\text{FindAtLeftMost}_{k+1}(l, r, t, d, s)) = \begin{cases} O\left(\sqrt{d}(\log_2(d))^{0.5(k-2)}\right) & \text{for } k \ge 2 \\ O(1) & \text{for } k = 1 \end{cases}$
5. $Q(\text{FindFirst}_k(l, r, s, left)) = O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right)$ *for* $k \ge 2$
6. $Q(\text{FindFixedPos}_k(l, r, t, s)) = \begin{cases} O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right) & \text{for } k \ge 3 \\ O(1) & \text{for } k = 2 \end{cases}$

Proof Theorem 3.2. The idea is that only the $O(\sqrt{n})$ comes from the initial cases for $k = 1$ and for each of the $k-1$ level of the recursion, the quantum query complexity is increased by a $O\left(\sqrt{\log_2(n)}\right)$ factor. The $O\left(\sqrt{\log_2(n)}\right)$ factor is proven by Andris Ambainis' team in [1] while the $O(\sqrt{n})$ for $k = 1$ comes from the new version of FindAny$_k$ (Algorithm 2). The complete proof for the theorem is given in Appendix B.

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 $\textsc{Dyck}_{k,n}$ 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 $\mathrm{Dyck}_{2,n}$

First, we would like to find an algorithm with a quantum query complexity near to match the lower bound, $\exists c > 1$ such that $Q\left(\textsc{Dyck}_{k,n}\right) = \Omega\left(\sqrt{n}c^k\right)$, describes by Andris Ambainis' team in [1]. So the searched algorithm must have a quantum query complexity of $O\left(\sqrt{n}\right)$.

For $k = 1$, 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 $O\left(n\right)$ so the quantum query complexity finally becomes $O\left(n\sqrt{n}\right)$. In order to keep it in $O\left(\sqrt{n}\right)$, the algorithm must do a constant number of calls to Grover's search.

For that, we define a new alphabet that can express every even length binary strings and that have convenient property for a 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 $aa, ac, bb, bd, cb, cd, da,$ or dc as substrings.*

Proof Theorem 3.3. First, this alphabet $\mathcal{A}$ is important because each 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 0. 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 after its first bit.

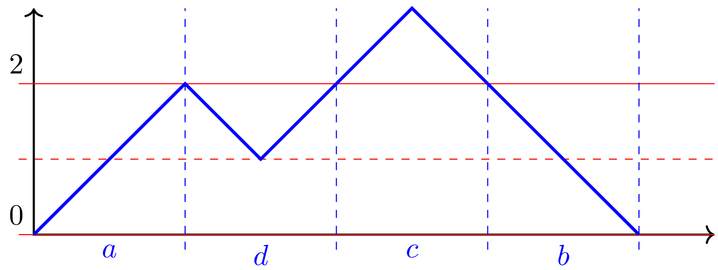

Figure 1: Illustration of the letters of $\mathcal{A}$ using Dyck's representation.

This property is important as it implies that every ± 3 strings uses at least two letters. So by checking if a pair of letter as a substring of a word make it not a Dyck word, $\mathcal{A}^2$ can be split into two sets described in Table 1.

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 which 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 at most 2, the current height is bounded between 0 and 2, moreover after each letter the current height is even so both couple cd and dc start and finish on the same bound. Futhermore, cd and dc are going above and below the height at which they start so both are going outside off the bounds, thus a word which contains cd or dc can not be a Dyck Word of height a most 2. The Figure 2 shows each couple of $\mathcal{X}$.

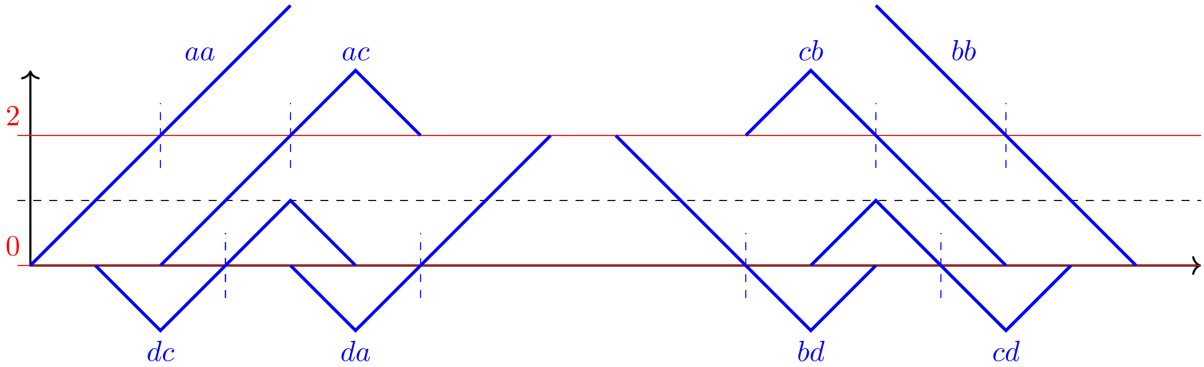

Figure 2: Every 2 letters configuration that implies the word, whom the configuration is a substring, is not a Dyck word of height at most 2.

- The set $\mathcal{V}$. The couples of $\mathcal{A}$ do not imply that the word is not a Dyck word of height at most 2 because each couple can fit inside the height bounds. 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.

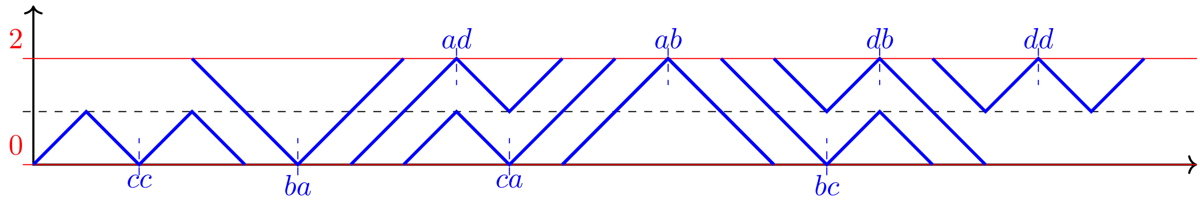

Figure 3: Every 2 letters configuration that can be found in a valid Dyck word of height at most 2.

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

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.

Theorem 3.4. ***The quantum query complexity of* DyckFast$_{2,n}$.** *The* $\text{DYCKFAST}_{2,n}$ *algorithm has a quantum query complexity of* $O(\sqrt{n})$.

Proof Theorem 3.4. The algorithm is doing at most 8 Grover's search on the modified input string $11x00$. So the total quantum query complexity is the folling.

$$Q(\text{DYCKFAST}_{2,n}) = 8 \times O\left(\sqrt{n+4}\right) = O\left(\sqrt{n}\right)$$

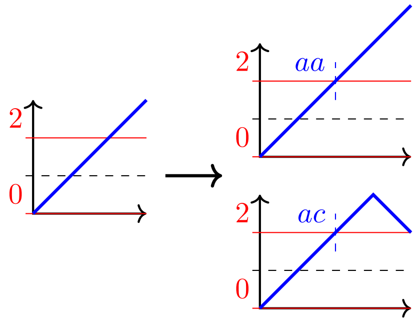

Figure 4: Configuration for a +3 strings of size 3.

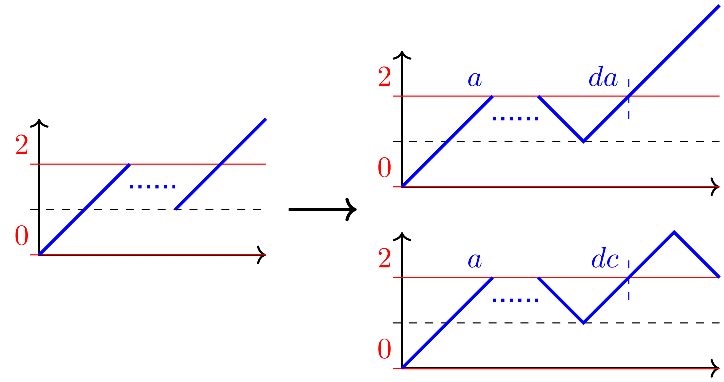

Figure 5: Configurations for a +3 string of size greater than 3.

Algorithm 3 $\textsc{DyckFast}_{2,n}$

Require: $n \geq 0$, x such that $|x| = 2n$
  $x \leftarrow 11x00$
  $\texttt{t} \leftarrow \textsc{Null}$
  for `reject_symbol` $\in \{aa, ac, bb, bd, cb, cd, da, dc\}$ **do**
    if $t == \textsc{Null}$ **then**
      Find `t` in $[\![0, n]\!]$ such that
      $x[2\texttt{t}, \ldots, 2\texttt{t} + 3] = \texttt{reject\_symbol}$
  return $\texttt{t} == \textsc{Null}$

3.3 Search for an algorithm for all k

4 Conclusion

Conclusion here.

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.

[2] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of Statistical Physics*, (22(5):653-591), 1980.

[3] Richard Feymann. Quantum mechanical computers. *Foundamentale Physics*, (16, 507-531), 1986.

[4] Kamil Khadiev and Dmitry Kravchenko. Quantum algorithm for dyck language with multiple types of brackets. In Irina Kostitsyna and Pekka Orponen, editors, *Unconventional Computation and Natural Computation - 19th International Conference, UCNC 2021, Espoo, Finland, October 18-22, 2021, Proceedings*, volume 12984 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 2021.

[5] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

5 Appendix

The frame of the intership

A The algorithm for Dyck$_{k,n}$

All the subroutines' pseudo code can be found from ALGORITHM 4 to ALGORITHM 9.

Algorithm 4 DYCK$_{k,n}$

Require: $n \geq 0$ and $k \geq 1$
Ensure: $|x| = n$
  $x \leftarrow 1^k x 0^k$
  $v \leftarrow \textsc{FindAny}_{k+1}(0, n + 2 * k - 1, \{1, -1\})$
  return v = NULL

Algorithm 5 FINDANY$_k(l, r, s)$

Require: $0 \leq l < r$ and $s \subseteq \{1, -1\}$
  Find d in $\{2^{\lceil \log_2(k) \rceil}, 2^{\lceil \log_2(k)+1 \rceil}, \ldots, 2^{\lceil \log_2(r-l) \rceil}\}$ such that
    $v_d \leftarrow \textsc{FindFixedLength}_k(l, r, d, s)$ 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 $\{l, l + 1, \ldots, r\}$ such that
    $v_t \leftarrow \textsc{FindAtLeftMost}_k(l, r, t, d, s)$ is **not** NULL
  return v_t of NULLif none

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

Theorem B.1. Dyck$_{k,n}$*'s Subroutines complexity* *The subroutines' quantum query complexity for k are the following.*

1. $Q(\textsc{Dyck}_{k,n}) = O\left(\sqrt{n}(\log_2(n))^{0.5(k-1)}\right)$ *for* $k \geq 1$

2. $Q(\textsc{FindAny}_{k+1}(l, r, s)) = O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right)$ *for* $k \geq 1$

3. $Q(\textsc{FindFixedLength}_{k+1}(l, r, d, s)) = O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right)$ *for* $k \geq 2$

4. $Q(\textsc{FindAtLeftMost}_{k+1}(l, r, t, d, s)) = \begin{cases} O\left(\sqrt{d}(\log_2(d))^{0.5(k-2)}\right) & \text{for } k \geq 2 \\ O(1) & \text{for } k = 1 \end{cases}$

5. $Q(\textsc{FindFirst}_k(l, r, s, left)) = O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right)$ *for* $k \geq 2$

6. $Q(\textsc{FindFixedPos}_k(l, r, t, s)) = \begin{cases} O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right) & \text{for } k \geq 3 \\ O(1) & \text{for } k = 2 \end{cases}$

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

Algorithm 7 $\textsc{FindAtLeftMost}_k(l, r, d, t, s)$

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, σ_1) ← FindAtLeftMost_{k−1}(l, r, t, d − 1, {1, −1})
if v ≠ Null then
    v' = (i_2, j_2, σ_2) ← FindAtRightMost_{k−1}(l, r, i_1 − 1, d − 1, {1, −1})
    if v' = Null then
        v' = (i_2, j_2, σ_2) ← FindFirst_{k−1}(max(l, j_1 − d + 1), i_1 − 1, {1, −1}, left)
    if v' ≠ Null and σ_2 ≠ σ_1 then v' ← Null
    if v' = Null then
        v' = (i_2, j_2, σ_2) ← FindAtLeftMost_{k−1}(l, r, j_1 + 1, d − 1, {1, −1})
    if v' = Null then
        v' = (i_2, j_2, σ_2) ← FindFirst_{k−1}(j_1 + 1, max(r, i_1 + d − 1), {1, −1}, right)
    if v' = Null then return Null
else
    v = (i_1, j_1, σ_1) ← FindFirst_{k−1}(t, min(t + d − 1, r), {1, −1}, right)
    if v = Null then return Null
    v' = (i_2, j_2, σ_2) ← FindFirst_{k−1}(max(t − d + 1, l), t, {1, −1}, left)
    if v' = Null then return Null
if σ_1 = σ_2 and σ_1 ∈ s and max(j_1, j_2) − min(i_1, i_2) + 1 ≤ d then
    return (min(i_1, i_2), max(j_1, j_2), σ_1)
else return Null
```

Algorithm 8 $\textsc{FindFirst}_k(l, r, s, left)$

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

```
lBorder ← l, rBorder ← r, d ← 1
while lBorder + 1 < rBorder do
    mid ← ⌊(lBorder + rBorder)/2⌋
    v_l ← FindAny_k(lBorder, mid, s)
    if v_l ≠ Null then rBorder ← mid
    else
        v_mid ← FindFixedPos_k(lBorder, rBorder, mid, s, left)
        if v_mid ≠ Null then return v_mid
        else lBorder ← mid + 1
    d ← d + 1
return Null
```

Algorithm 9 $\textsc{FindFixedPos}_k(l, r, t, s)$

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

```
Find d in {2^⌈log_2(k)⌉, 2^⌈log_2(k)+1⌉, ..., 2^⌈log_2(r−l)⌉} such that
        v_d ← FindAtLeftMost_k(l, r, t, d, s) is not Null
return v_d or Null if none
```

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

- For $k = 1$, only $\textsc{FindAtLeftMost}_2$, $\textsc{FindAny}_2$,and $\textsc{Dyck}_{1,n}$ are defined. The $O(1)$ quantum query complexity of $\textsc{FindAtLeftMost}_2$ comes directly from the definition of its initial case, as the $O(\sqrt{r-l})$ quantum query complexity of $\textsc{FindAny}_2$. Then the $O(\sqrt{n})$ quantum query complexity of $\textsc{Dyck}_{1,n}$ comes from the call to $\textsc{FindAny}_2$.
- For $k = 2$, the inductive part of the algorithm start and every subroutines is defined. The $O(1)$ quantum query complexity of $\textsc{FindFixedPos}_2$ comes from the call to $\textsc{FindAtLeftMost}_2$. The $O\left(\sqrt{r-l}\right)$ quantum query complexity of $\textsc{FindFirst}_2$ comes from the dichotomize search using $\textsc{FindAny}_2$ and $\textsc{FindFixedPos}_2$ because $\sum_{u=1}^{log_2(r-l)} 2u\left(O\left(\sqrt{\frac{r-l}{2^{u-1}}}\right) + O(1)\right) = O(\sqrt{r-l})$ (Detailed in the induction). The $O(\sqrt{d})$ quantum query complexity of $\textsc{FindAtLeftMost}_3$ comes from the constant amount of calls to $\textsc{FindFirst}_2$ and $\textsc{FindAtLeftMost}_2$ with entry of size d. The $O(\sqrt{r-l})$ quantum query complexity of $\textsc{FindFixedLength}_3$ comes from the $O\left(\sqrt{\frac{r-l}{d}}\right)$ calls to $\textsc{FindAtLeftMost}_3$. The $O\left(\sqrt{(r-l)\log_2(r-l)}\right)$ quantum query complexity of $\textsc{FindAny}_3$ comes from the $O\left(\sqrt{\log_2(r-l)}\right)$ calls to $\textsc{FindFixedLength}_3$. Finally, the $O\left(\sqrt{(r-l)\log_2(r-l)}\right)$ quantum query complexity of $\textsc{Dyck}_2$ comes from the call to $\textsc{FindAny}_3$.

Induction: Let suppose it exists k such that Theorem B.1 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 $\textsc{FindFixedPos}_{k+1}$ comes from the $O\left(\sqrt{\log(r-l)}\right)$ calls to $\textsc{FindAtLeftMost}_{k+1}$.

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

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

$$\begin{aligned}
Q(\textsc{FindFirst}_{k+1}(l,r,t,d,s)) = \; & \begin{array}{l} \sum_{u=1}^{\log_2(r-l)} 2u \times O\left(Q(\textsc{FindAny}_{k+1}(0, \frac{r-l}{2^{u-1}}, s))\right) \\ + \sum_{u=1}^{\log_2(r-l)} 2u \times O\left(Q(\textsc{FindFixedPos}_{k+1}(0, \frac{r-l}{2^{u-1}}, _, s, left))\right) \end{array} \\
&\overset{IH}{=} O\left(\sum_{u=1}^{\log_2(r-l)} 2u \times \sqrt{\frac{r-l}{2^{u-1}}}(\log_2(\frac{r-l}{2^{u-1}}))^{0.5(k-1)}\right) \\
&= O\left(\sum_{u=1}^{\log_2(r-l)} 2u \times \sqrt{\frac{r-l}{2^{u-1}}}(\log_2(r-l))^{0.5(k-1)}\right) \\
&= O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)} \sum_{u=1}^{\log_2(r-l)} u \times (\frac{1}{\sqrt{2}})^{u-1}\right) \\
&=^a O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)} \frac{\sqrt{2}^2}{(\sqrt{2}-1)^2}\right) \\
&= O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right)
\end{aligned}$$

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

$$\begin{aligned} Q(\textsc{FindAtLeftMost}_{k+2}(l,r,t,d,s)) = \;& 3 \times O\left(Q(\textsc{FindAtLeftMost}_{k+1}(l,r,t,d,\{1,-1\}))\right) \\ & +4 \times O\left(Q(\textsc{FindFirst}_{k+1}(l,r,\{1,-1\},left))\right) \\ \overset{IH}{=}\;& O\left(\sqrt{d}(\log_2(d))^{0.5(k-1)}\right) \end{aligned}$$

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

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

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

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

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

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

Conclusion: By the induction principle we get that the Theorem B.1 is true for $k \in \mathbb{N}^*$

---

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