École Normale Supérieure de Lyon
Latvijas Universitāte

M1 Internship Repport

Complexity of recognizing Dyck languages of bounded height with quantum query algorithms.

Key words: *Quantum query complexity, Dyck words, Quantum algorithms, regular languages, star free languages, adversary methods.*

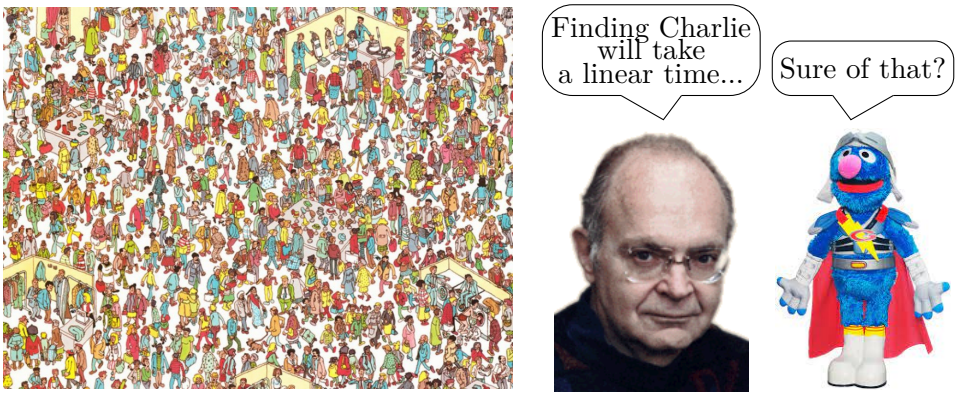

Figure 1: Historical discussion between Knuth and Grover.

Student:
Maxime Cautrès

Supervisor:
Andris Ambainis
Kamil Khadiev

May the 2nd 2022 - July the 22th 2022

Contents

1 Introduction

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 [4] and further by Kamil Khadiev [7].

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 [4] 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 [5]. 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 [6]. 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) [11]. 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 and complexity

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 $|1\rangle$) represents a 1 and an other one with a spin of $-\frac{1}{2}$ (note $|0\rangle$) represents a 0. But the use of particles is motivated by their properties and mainly by a quantum property called superposition. A quantum state is not only $|0\rangle$ or $|1\rangle$, but can be $\lambda_0|0\rangle + \lambda_1|1\rangle$ for every $\lambda_{0,1} \in \mathbb{C}$ such that $\lambda_0 + \lambda_1 = 1$ and a quantum bit is now called a qubit. As before, the computations are done by gates, here quantum gates which transform the quantum state of the qubit into another one. At the end, to get the result of a computation, it is mandatory to measure the state of the quantum system, which break the quantum superposition. A quantum state of n qubits can be represented with a length 1 vector in a 2^n dimensional space and a quantum gates by a linear unitary transformation on a 2^n dimensional space. A transformation is said unitary if is preserved the length.

A quantum circuit is a precise configuration of quantum gates on a finite number of qubits. The following Figure 2 represents the quantum circuit that computes a uniform randomize on $\{0,1\}^n$.

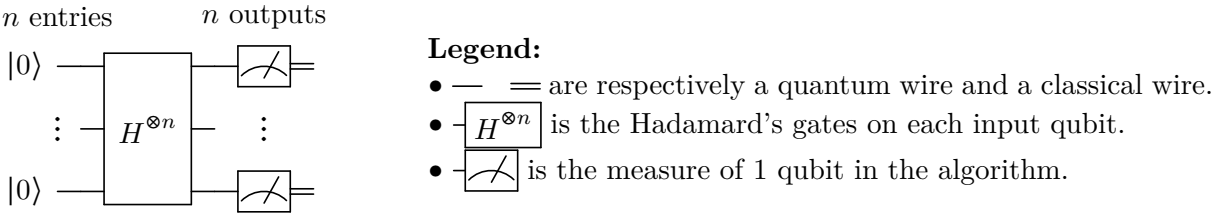

Figure 2: A quantum circuit computing the uniform random on $\{0,1\}^n$.

The quantum query circuit is a quantum circuit used to compute function with an entry space where $x = x_1 \dots x_N$ belongs. The black box model [2] of a quantum query circuit is composed of an input state $|\psi_{start}\rangle$ and a sequence $U_0, Q, U_1, \dots, Q, U_T$ of linear unitary transformations such that $|\psi_{start}\rangle$ and all U_i do not depend on entry x unlike the Q_i which depend on x. The quantum state $|\psi_{start}\rangle$ belong to a d-dimensional space generated by $|1\rangle, |2\rangle, \dots, |d\rangle$. To define Q, the basis vectors first need to be renamed from $|1\rangle, \dots, |d\rangle$ to $|i,j\rangle$ with $i \in [\![0, N]\!]$ and $j \in [\![1, d_i]\!]$ for some d_i such that $d_1 + d_2 + \dots + d_N = d$. Next, Q is define such that

$$Q(|i,j\rangle) := \begin{cases} |0,j\rangle & \text{if } i = 0 \\ |i,j\rangle & \text{if } i > 0 \text{ and } x_i = 0 \\ -|i,j\rangle & \text{if } i > 0 \text{ and } x_i = 1. \end{cases}$$

The gates Q are doing the queries to input x by flipping some of the vectors. Finally, to get the output of the quantum query algorithm it is necessary to measure the output. A quantum query algorithm can the summarized with the following quantum circuit.

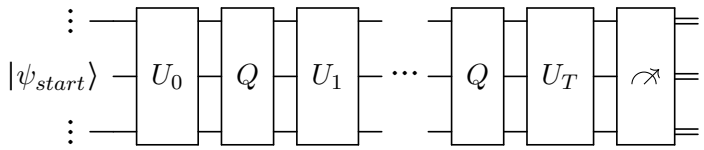

Figure 3: Structure of a quantum query algorithm.

The quantum query complexity of an algorithm corresponds to the number of calls to the Q gate. Often, this number of calls is depending on the size of the entry. The quantum query complexity of a problem corresponds to the higher bound for which it is certain there is no quantum query algorithm with a greater quantum query complexity solving the problem.

1.3 Dyck Languages of height k

First, the Dyck Language corresponds to the set of correct and balanced word of parenthesis. An other convenient definition used increasing $\overrightarrow{(1,1)}$ and decreasing $\overrightarrow{(1,-1)}$ steps to create a path whose starting point is (0, 0), always stays above the abscise axe and end on it. A Dyck word of length 12 is presented Figure 4. The Dyck language is a context free language as it can easily be recognized using a context free grammar. The work done by Andris Ambainis' team [4] focus on a restriction Dyck language with fixed height k. More precisely, a Dyck word has a height of k if, in every of its prefix, the difference between the number of opening and closing parenthesis does not exceeds k. These restricted Dyck languages are called $\text{DYCK}_{k,n}$ and are interesting because they belong to the already well studied class of star free languages (Detail in subsubsection 2.1.2).

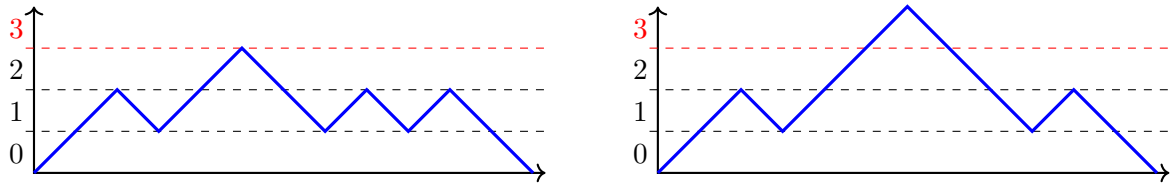

Figure 4: **On the left**, a valid Dyck word of height 3. **On the right**, an invalid Dyck word of height 3.

1.4 State of the art

This state of the art will not be to precise as the understanding of the bibliography has taken almost the first half the internship and will be more detailed in the section 2. First, few years ago Aaronson, Grier and Schaeffer [1, 2019] worked on quantum complexity of recognizing regular languages as they can model a lot of tasks. They concluded that there are 3 different cases depending of the language:

- $O(1)$ if it is sufficient to read constant number of letters at the beginning and the end.
- $\tilde{\Theta}(\sqrt{n})$ if a Grover's search is the best way to recognize the language.
- $\Theta(n)$ if recognizing the language is the same as counting modulo some value that can be computed with $O(n)$.

Further more, it is proved that being in the second classes is equivalent to being a star free language. Andris ambainis' team decided to work on $\text{DYCK}_{k,n}$ as this language is a beautiful example of star free languages. In [4, 2020], the team focus on the power of the logarithm in $\tilde{\Theta}(\sqrt{n})$. They first proved by reduction that the quantum query complexity of $\text{DYCK}_{k,n}$ called $Q(\text{DYCK}_{k,n})$ is in $\Omega\left(c^k\sqrt{n}\right)$ where c is a constant greater than 1 and after gave an algorithm for $\text{DYCK}_{k,n}$ with a quantum query complexity of $O\left(\sqrt{n}(\log(n))^{0.5k}\right)$. Since then, no better lower bound or algorithm have been found.

1.5 Goals of the internship

The problem on the quantum query complexity of $\textsc{Dyck}_{k,n}$ is still open, my internship has for goal to reduce the gap between the lower bound and the best known algorithm. The researches have been organized on two main axes:

- **Increasing the lower bound.** To do this, It is necessary to understand the bibliography on the adversary method in order to try to find a new adversary with better property.
- **Lowering the upper bound.** It is sufficient to find new algorithms with a quantum query complexity better than the previous best known algorithm's one.

Finally, the overall goal would be to made the two bounds match in order to get the exact quantum query complexity of $\textsc{Dyck}_{k,n}$. Finally, the problem could also be reformulated for multi-brackets word as done by Kamil Khadiev in [7].

1.6 Results

TODO

2 Preliminaries

In order to understand from where comes the $\tilde{\Theta}(\sqrt{n})$ quantum query complexity, it is mandatory to start by understanding the Trichotomy theorem [1] and its dependency on Regular languages and star free languages. After that, to understand how to find a lower bounds, it is necessary to explore the different adversary methods [12].

2.1 Quantum query for regular languages.

In the article [1], Aaronson, Grier and Schaeffer presented a really interesting algebraic characterization of regular languages based on the recognition by monoids and the syntactic congruence. This totally new to me definition give me some hard time in order to understand it correctly.

2.1.1 Regular languages

Usually, the set of regular languages $\mathcal{R}$ on the alphabet Σ is defined as the smallest fixed point of the function F (the function that computes concatenations, unions, and Kleene stars) including $\{\varnothing\} \cup \{\varepsilon\} \cup (\cup_{l\in\Sigma}\{l\})$ with F equal

$$\begin{aligned} F(X) =& \{AB \ \forall (A,B) \in X^2\} \\ & \cup \{A \cup B \ \forall (A,B) \in X^2\} \\ & \cup \{A \cap B \ \forall (A,B) \in X^2\} \ \# \texttt{ optional} \\ & \cup \{A^* \ \forall A \in X\}. \end{aligned}$$

But here, the more convenient way to characterize regular language is with their algebraic characterization. Indeed, a regular language is always a pre image of a subset of a monoid under monoid homomorphism. A monoid is a 3-tuple of a set M, an internal associative binary operation and finally the identity element associated to the operation. A monoid homomorphism is a map from a monoid to another that preserve the operation and the identity. Usually, to get the monoid representation of a regular language, the first step is to compute the syntactical monoid. The syntactical monoid is obtained by dividing Σ^* by the following equivalence relation called syntactic congruence

$$x \sim_L y \iff \forall (u,v) \in \left(\Sigma^*\right)^2, (uxv \in L \iff uyv \in L).$$

This equivalence relation is a congruence relation as the equivalence class can be multiplied (i.e. if $x \sim_L y$ and $u \sim_L v$ then $xu \sim_L yv$). Now, it is necessary to get the monoid homomorphism. For that it is sufficient to take the homomorphism that map an element to its congruence class. Inside a congruence class, none or every element of the class is in L. Indeed, if x is in L and $x \sim_L y$ then for all (u, v) in $\left(\Sigma^*\right)^2$, (uxv in $L \Leftrightarrow uyv$ in L) thus x in L implies y in L and finally as x is in L then y is also in L. Now, it is sufficient to prove that the syntactical monoid is of finite size. ma be put the proof if time else say folklore.

Finally, every regular language can be recognized by finite monoid.

2.1.2 Star free languages

The set of star free languages is a really well studied subset of the regular languages. Its definition differs a little from regular languages' one as the Kleene star is replace by the complement operation (note $\overline{L}$). So star free languages are defined as the smallest fixed point of the function F' (the function that computes concatenations, unions and complements) and such that it includes $\{\varnothing\} \cup \{\varepsilon\} \cup (\cup_{l \in \Sigma} \{l\})$. This restriction does not imply that every star free language is finite. Indeed, Σ^* can be written $\overline{\varnothing}$. For example the language on $\Sigma = \{1, 2, 3\}$ described with the regular expression $\Sigma^* 20^* 2\Sigma^*$ can be written as following in the star free way $\overline{\varnothing}2\overline{\overline{\varnothing}\Sigma \setminus \{0\}}\overline{\varnothing}2\overline{\varnothing}$.

As for regular languages, it exists an algebraic characterization for star free languages. Let M be a monoid, M is said to be aperiodic if for every x in m it exits a positive integer n such that $x^n = x^n + 1$. A theorem proved by Schützenberger [10] state that a language is recognized by a finite aperiodic monoid if and only if if is star free.

Good examples of star free languages are the Dyck word languages with bounded heights. It is first easy to have a finite automata that recognize Dyck word of height k by putting one state for each integer from 0 up to k. However, the belonging to the star free regular languages is more delicate to prove. It has been done done by Italian researcher in [13, 1978].

2.1.3 Trichotomy theorem

Theorem 2.1 (Aaronson, Grier and Schaeffer [1])**.** *Every regular language has quantum query complexity* $0, \Theta(1), \tilde{\Theta}(\sqrt{n})$, *or* $\Theta(n)$ *according to the smallest class in the following hierarchy that contains the language.*

- *Degenerate: One of the four languages* $\varnothing, \varepsilon, \Sigma^*$, *or* Σ^+.
- *Trivial: The set of languages which have trivial* [1] *regular expressions.*
- *Star free: The set of languages which have star-free regular expressions.*
- *Regular: The set of languages which have regular expressions.*

This theorem is really important as it give a good idea on the quantum query complexity of many language recognitions. Moreover, the classes are now clearly defined so it is now easier to know where is a problem compared to the classes in the introduction. However, it does not give the exact quantum query complexity because for star free languages the result is given using $\tilde{\Theta}$. The $\tilde{\Theta}(\sqrt{n})$ means that the quantum query complexity of any star free regular language is in $\Theta(\sqrt{n}(\log_2(n))^p)$ for some p a non negative constant. As it is known that $\textsc{Dyck}_k$ languages are star free, it is an interesting problem to find the power of $\log_2(n)$ depending on the value of k.

[1] A language L is said to be trivial if and only if it exists 2 finite size alphabets L_1 and L_2 such that $L = L_1 \Sigma^* L_2$.

2.2 The bounds for Dyck$_{k,n}$ problem

The trichotomy theorem state that DYCK$_{k,n}$ language has a quantum query complexity in $\tilde{\Theta}(\sqrt{n})$. The Θ means that the best possible algorithm is both a $\tilde{O}(\sqrt{n})$ and a $\tilde{\Omega}(\sqrt{n})$. So, a common method to find the power of $\log_2(n)$ depending on k is the squeeze theorem. More precisely, the quantum query complexity of DYCK$_{k,n}$ is trapped between a lower and an upper bound for quantum query complexity. So it is possible to deduce the quantum query complexity from an increasing sequence of lower bound and a decreasing sequence of upper bound such that they share the same limit l with l equals to $Q(\text{DYCK}_{k,n})$. How to find this sequence ?

- **For the lower bound sequence.** Some of the most important tools to compute lower bounds are called **quantum adversary methods** and have been invented by Ambainis [3]. This tools can compute lower bounds more or less tight and some of this adversary methods have really useful property such as being compatible with a certain type of composition of problems. This lead to the **reduction methods** that can compute a lower bounds from different but easier problem's ones.
- **For the upper bound sequence.** The main method is to **find quantum query algorithms** that are more and more efficient. An other way is also **by reduction** to a more difficult problem with a good enough upper bound.
- **For the same limit.** The idea is to do an iterative process where each step improves successively each bound until only one will continue to be improved. Finally, both bounds may end up matching. However, before my internship, both DYCK$_{k,n}$'s lower and upper bounds where stuck to $\Omega\left(c^k\sqrt{n}\right)$ for some constant c greater than 1 and $O\left(\sqrt{n}\left(\log_2(n)\right)^{0.5k}\right)$.

2.2.1 Lower bound on the quantum query complexity of Dyck$_{k,n}$

The quantum adversary methods: This part as for goal to present some quantum adversary methods and their usages.

The adversary method. The adversary method is a classical way to prove lower bound in classical algorithmic. Indeed, the idea behind the classical adversary is to prove that for every algorithm, it exist an entry such that the algorithm cannot decide in less unit of complexity than the lower bounds. In general, during the algorithm execution, the adversary modifies entry values that have not been already used in order to increase the execution time. This classical adversary work great for classical algorithm but is not really useable on quantum algorithms.

The super basic quantum adversary method. In order to recognize a language it is mandatory to be able to distinguish between a valid word v and an invalid word w. So at the output of the quantum query algorithm, the states $|\psi_v^T\rangle$ and $|\psi_w^T\rangle$ should be distinguishable thus $|\langle\psi_v^T|\psi_w^T\rangle| < \frac{2}{3}$. How do they will differs? The quantum query algorithm start with $|\psi_v^0\rangle = |\psi_w^0\rangle = |\psi_{start}\rangle$. Moreover, the inner product of both states isn't affected by U_i gates because there are unitary. However, the Q_i gates affect this inner product as the Q_i's behaviors are depending on the input. Let define the progress measure $\mathcal{P}$ such that $\mathcal{P}(t) := \langle\psi_v^t|\psi_w^t\rangle$ where $|\psi^t\rangle$ represents the state after Q_t. Let suppose it exists d such that for all $0 \leq t \leq T-1$, $|\mathcal{P}(t+1) - \mathcal{P}(t)| \leq d$. It implies the following inequality

$$\mathcal{P}(T) = \underbrace{\mathcal{P}(0)}_{=1} + \sum_{i=0}^{T-1} \underbrace{\mathcal{P}(i+1) - \mathcal{P}(i)}_{\geq -d} \geq \mathcal{P}(0) - T \times d.$$

Moreover, $\mathcal{P}(T)$ should be lower than $\frac{2}{3}$ so it implies that $1 - T \times d$ should also be lower than $\frac{2}{3}$ and finally that T should be a $O(\frac{1}{d})$. This give a general idea of how to determined a lower

bound but it isn't precise enough to get a theorem. In its courses, Ryan O'Donnell [8] explained a simple to understand adversary method called the super basic adversary method. This adversary allow to compute a lower bound by using the following method:

Theorem 2.2. *Let define YES the set equal to $f^{-1}(accepted)$ and NO the set equal to $f^{-1}(rejected)$. If it exist two subset $Y \subseteq YES$ and $Z \subseteq NO$ such that:*

- *For each y in Y, there are at least m strings z in Z with $dist(y,z) = 1$.*
- *For each z in Z, there are at least m' strings y in Y with $dist(y,z) = 1$.*

Then the quantum query complexity $Q(f)$ is in $\Omega(\sqrt{mm'})$.

The proof of the theorem can be found in Appendix A. One important result of the adversary method is the lower bound on the quantum query complexity of $Ex_{2m}^{m|m+1}$ in $\Omega(m)$ where the problem $Ex_{2m}^{m|m+1}$ consists to recognize between $|x| = 2m \wedge |x|_1 = m$, and $|x| = 2m \wedge |x|_1 = m+1$. More generally, the method of an adversary define a process to find the lower d possible. The super-basic adversary method as its name let suggest is simple but does not give a tight lower bounds. For this, the method should be more complicated which results in longer days on the board to find the parameters of the methods.

Let $f : D \to \{0,1\}$ be the function whose quantum query complexity is unknown. Two other adversary methods are described as following:

- **The Basic adversary method by Ambainis [3].**

  Theorem 2.3. *Let define YES the set equal to $f^{-1}(accepted)$ and NO the set equal to $f^{-1}(rejected)$. Moreover, let $Y \subseteq YES$, $Z \subseteq NO$, and let $\mathcal{R} \subseteq Y \times Z$ be a set of "hard to distinguish" pairs, such that:*

  - *For each y in Y, there are at least m strings z in Z with (y,z) in $\mathcal{R}$.*
  - *For each z in Z, there are at least m' strings y in Y with (y,z) in $\mathcal{R}$.*

  Also, let define $\mathcal{R}_i$ as a restriction from $\mathcal{R}$ such that (y,z) is in $\mathcal{R}_i$ if and only if (y,z) is in $\mathcal{R}$ and $y_i \neq z_i$. In addition,

  - *For each y in Y and i, there are at most l strings z in Z with (y,z) in $\mathcal{R}_i$*
  - *For each z in Z and i, there are at most l' strings y in Y with (y,z) in $\mathcal{R}_i$*

  Then the quantum query complexity $Q(f)$ is in $\Omega(\sqrt{\frac{mm'}{ll'}})$.

- **The general adversary method by Reichardt [9].**

  A symmetric matrix Γ is an adversary matrix for f if the rows and cols of Γ can be indexed by input x in D such that $\Gamma_{x,y} = 0$ if $f(x) = f(y)$. $\Gamma^{(i)}$ is defined from Γ and is a similarly sized matrix such that $\Gamma^{(i)}_{x,y} = \begin{cases} \Gamma_{x,y} & \text{if } x_i \neq y_i \\ 0 & \text{otherwise} \end{cases}$. This two objects allow to define the following notion of adversary plus minus

  $$Adv^{\pm}(f) = \max_{\substack{\Gamma \text{ - an adversary} \\ \text{matrix for } f}} \frac{\|\Gamma\|}{\max_i \|\Gamma^{(i)}\|}.$$

  Theorem 2.4. *The adversary plus minus is such that $Q(f) = O(Adv^{\pm}(f))$.*

  In [9], Reichardt gave the proof that the general adversary method is not only giving a lower bound to the quantum query complexity as the method return the directly the quantum query complexity of f.

All this adversary methods are really interesting to compute lower bounds, but sometime it is useful to reuse already computed ones.

The reduction method: For some problems, it looks obvious that some are easier than the others. Computer scientists have developed tools to handle more formally this notion of difficulty comparison. One of the main tool is named reduction. A reduction is the process to solve a first problem using an algorithm for a second one. Because the second problem is able to solve the first one, it is said to be harder, which is written $P_1 \leq P_2$.

Theorem 2.5. *Reduction and* $Adv^{\pm}$

$$P_1 \leq P_2 \implies Adv^{\pm}(P_1) \leq Adv^{\pm}(P_2)$$

Finally, problems can also be composed. Lets take $P_1 : \{0,1\}^n \to \{0,1\}$ and $P_2 : \{0,1\}^m \to \{0,1\}$. Then $P_1 \circ P_2$ is defined as

$$(P_1 \circ P_2)(x_1 \dots x_{nm}) := P_1\left(\underbrace{P_2(x_1 \dots x_m), \dots, P_2(x_{(n-1)m+1} \dots x_{nm})}_{n}\right).$$

This composition has a great behavior with the adversary plus minus.

Theorem 2.6. *Composition and* $Adv^{\pm}$

$$Adv^{\pm}(P_1 \circ P_2) \geq Adv^{\pm}(P_1) \times Adv^{\pm}(P_1)$$

Andris' team found that the $Ex_{2m}^{m|m+1}$ problem can be reduce using the OR and AND problems to $\textsc{Dyck}_k$ problem with the reduction described in [4]. They finally get a lower bound in $\Omega(c^k\sqrt{n})$ for the quantum query complexity of $\textsc{Dyck}_{k,n}$.

2.2.2 Best known algorithm to recognize $\textbf{Dyck}_{k,n}$

Before checking the algorithm for $\textsc{Dyck}_{k,n}$, it is necessary to define some terms useful for later.

Preliminary definition:

- **The height function** h**:** This first utilarity function allow the computation of final height of a string. It is define as following

$$h(x_1 \dots x_n) := \sum_{i=1}^{n} (-1)_i^x$$

- $\pm k$**-strings:** A string $x_1 \dots x_n$ is said to be a $+k$-string (resp. $-k$-string) if

$$\max_{1 \leq i \leq j \leq n} h(x_i \dots x_j) = k \left(\text{resp. } \min_{1 \leq i \leq j \leq n} h(x_i \dots x_j) = -k\right).$$

- **minimal** $\pm k$**-strings:** A $\pm k$-string $x_1 ... x_n$ is said to be minimal if it doesn't exist i, j such that $1 \leq i \leq j \leq n$, $(i,j) \neq (1,n)$ with $x_i \dots x_j$ a $\pm k$-string.

$\textbf{Dyck}_{k,n}$ characterization: In order to recognize $\textsc{Dyck}_k$ language, multiple approach are possible. The most method used in [4] by Ambainis's team is to search for a substring that cannot be seen into a Dyck word of height at most k. A natural way to reject a word w from $\textsc{Dyck}_k$ is to search for a $\pm k+1$-string into $1^k w 0^k$. Lets detail this technic more precisely. First, a Dyck word is always above the abscise axe, so it cannot exist i such that $h(w_1 \dots w_i) = -1$ thus it cannot exist i such that $h((1^k w 0^k)_1 \dots (1^k w 0^k)_i) = -k-1$ and finally it cannot exist a $-(k+1)$-string into $1^k w 0^k$. After that, a Dyck word always end on the abscise axis, which means that it doesn't exist i such that $h(w_i \dots w_n) = 1$ which implies that $1^k w 0^k$ cannot contain any $+(k+1)$-string. Moreover, for a Dyck word of height at most k, it cannot exist i such that $h(w_1 \dots w_i) = k+1$ so the bounded height constraint is already taken into account by the impossibility of having a $+(k+1)$-string into $1^k w 0^k$. Finally, this give us that the belonging of a $\pm(k+1)$-string into $1^k w 0^k$ is sufficient to reject every non Dyck word of height at most k.

$\pm k+1$-strings recognition: In order to recognize DYCK_k, the main point is now to find efficiently a $\pm(k+1)$-string. Let detailed a little bit how it is done.

- **For k=1:** To reject a non-DYCK_1 word w, it is sufficient to find ± 2-strings. However, the only every minimal ± 2)-strings is of size 2, so every $\pm(2)$-strings can be found by searching for 11 or 00 using 2 grover search. This methods implies a quantum query complexity of $O(\sqrt{n})$ from its two calls to Grover's one.
- **For k=2 (naive approach):** To reject, the goal is to find a $\pm(3)$-string. Unfortunately, there are an infinite number of minimal $\pm(3)$-strings as they form the language $1(10)^*11 + 0(01)^*00$. So trying every possible minimal $\pm(3)$-string for an input string of size n require $O(n)$ calls to Grover with a final quantum query complexity of $O(n\sqrt{n})$. This algorithm has its complexity already above the known upper bounds of the trichotomy theorem. In order to stay into the trichotomy theorem, the algorithm should be improved.
- **For any k+1:** In order to have a faster algorithm, Ambainis' team has found an inductive algorithm on the depth. Indeed, a minimal $\pm(k+1)$-string can be share into two smaller $\pm k$-strings as shown in Figure 5. The main ideas of the induction step are:

  1. Chose an upper bound d in $\{2, 4, 8, \ldots, 2^{\lfloor \log_2(n) \rfloor}\}$ for the length of the $\pm(k+1)$-string.
  2. Chose an indices t in $\{1, 2, 3, \ldots, n\}$ that have to be in the $\pm(k+1)$-string.
  3. Try to find two $\pm(k)$-string in an interval of length at most d that include t
     (a) Try to find a $\pm(k)$-string that include t of length at most d-1.
     (b) If it exists, find an other $\pm(k)$-string on the left or on the right, if it fail return `NULL`.
     (c) If it does not exist, try to find $\pm(k)$-string on the left, and another $\pm(k)$-string on the right, if it fail return `NULL`.
     (d) Test if both $\pm(k)$-string are of the same sign and if the string that include both if of length lower than d.
     (e) If test is good, return the $\pm(k+1)$-string.
     (f) Otherwise return `NULL`.

  Let explain the quantum query complexities and the idea behind each step. It has been shown before that searching for every minimal $\pm(k+1)$-string isn't a solution has it is too slow. So the idea is to bound the size of minimal $\pm(k+1)$-string the function is currently searching for. This is interesting as it allows to use a function describes in step 3 named $\text{FINDATLEFTMOST}_{k+1}$, whose main parameters are d and t, and which is able to find a minimal $\pm(k+1)$-string if and only if it include the index t and the size of the minimal $\pm(k+1)$-string is bounded between $\frac{d}{2}$ and d. These constraints imply two things: First, the parameter t implies a call to Grover (as it may be possible to have a $\pm(k+1)$-string not including t), but there is $O(d)$ value of t such that $\text{FINDATLEFTMOST}_{k+1}$ returns a minimal $\pm(k+1)$-string so it is possible to cut early the Grover search to get its quantum query complexity in $O\left(\sqrt{\frac{n}{d}}\right)$. This Grover search corresponds to item 2 named $\text{FINDFIXEDLENGTH}_{k+1}$. After that for d, in order not to miss any minimal $\pm(k+1)$-string, it is necessary to call $\text{FINDFIXEDLENGTH}_{k+1}$ for every d in $\{2, 4, 8, \ldots, 2^{\lfloor \log_2(n) \rfloor}\}$ (item 1) which implies a call to Grover in $O\left(\sqrt{\log(n)}\right)$. This is done in item 1 by the function named FINDANY_{k+1}. Finally, the quantum query complexity of $\text{FINDATLEFTMOST}_{k+1}$ is $O\left(\sqrt{d}(\log_2(d))^{0.5(k-1)}\right)$, it comes from complex calls to many subroutines. In steps 3a and 3b, at most three calls to FINDATLEFTMOST_k are done with a quantum query complexity of $O\left(\sqrt{d}(\log_2(d))^{0.5(k-2)}\right)$. In step 3b and 3c, at most 4 calls to FINDFIRST_k are done. The subroutines FINDFIRST_k find the closer $\pm k$-strings from the end r or the beginning l

of a specified interval. This function finally do a binary search using calls to FindAny$_k$ and FindFixedPos$_k$ in a quantum query complexity of $O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right)$. This last subroutine FindFixedPos$_k$ searches only for $\pm k$-strings which include the index t with a quantum query complexity of $O\left(\sqrt{n}(\log_2(n))^{0.5(k-1)}\right)$.

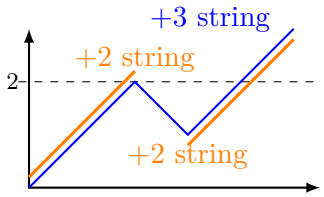

Figure 5: Decomposition of a +3-string into two $\pm(k+1)$-strings.

Finally, this complex algorithm has a quantum query complexity of $O\left(\sqrt{n}(\log(n))^{0.5k}\right)$ which is a $\tilde{\Theta}(\sqrt{n})$. This description of the existing algorithm stay at the surface in order to stay simple, every the pseudo code of every subroutines can be found in Appendix B. No proof of their quantum query complexity is provided as a proof for Theorem C.1, almost identical, is already provided in Appendix C. However, this algorithm can be improved and two personal improvements are described in section 3.

3 A better algorithm for Dyck$_{k,n}$

3.1 A better Complexity Analysis of the original algorithm

In the article [4], 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 $k=1$ ($O(\sqrt{n}\log_2(n))$) is not as good as a Grover's search $O(\sqrt{n})$ which is sufficient (seen in subsubsection 2.2.2). More precisely, the logarithmic search done by FindAny$_{k+1}$ for $k=1$ is useless as there is exactly one minimal ± 2 -string so it is sufficient to add a new initial case to FindAny$_{k+1}$ for $k=1$ in order to use a Grover search for 00 and 11 in $1w0$ instead of FindFixedLength$_2$. This lowers the quantum query complexity for $k=1$ 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 $s \subseteq \{1,-1\}$
  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

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 $\{2^{\lceil \log_2(k) \rceil}, 2^{\lceil \log_2(k)+1 \rceil}, \ldots, 2^{\lceil \log_2(r-l) \rceil}\}$ 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 B) 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 than its older definition.

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

1. $Q($DYCK$_{k,n}) = O\left(\sqrt{n}(\log_2(n))^{0.5(k-1)}\right)$ *for* $k \geq 1$
2. $Q($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. $Q($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. $Q($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. $Q($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. $Q($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 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 [4] 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 C.

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, $\exists c > 1$ such that $Q\left(\text{DYCK}_{k,n}\right) = \Omega\left(\sqrt{n}c^k\right)$, describes by Andris Ambainis' team in [4]. So the searched algorithm must have a quantum query complexity of $O(\sqrt{n})$.

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 $O(n)$ 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 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.

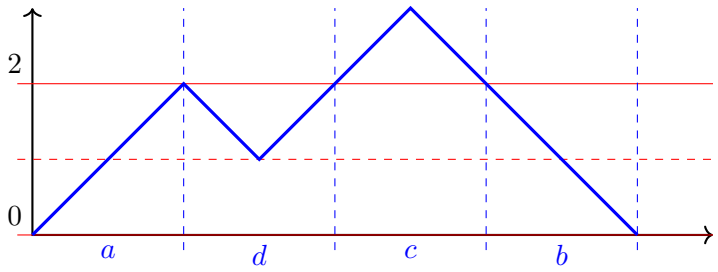

Figure 6: Illustration of the letters of $\mathcal{A}$ using Dyck's representation.

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 7 shows each couple of $\mathcal{X}$.
- 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 8 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 $\mathcal{X}$?

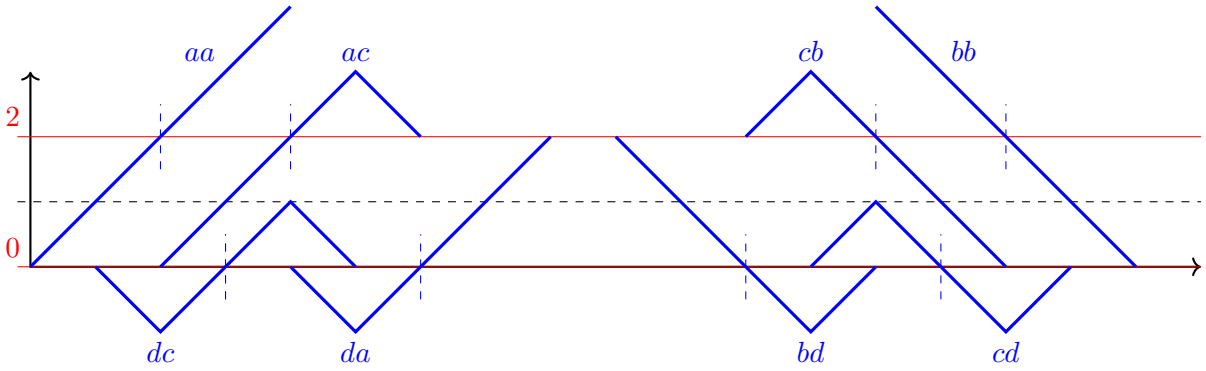

Figure 7: Every 2 letters configuration that implies the word, whom the configuration is a substring, is not a Dyck word of height at most 2.

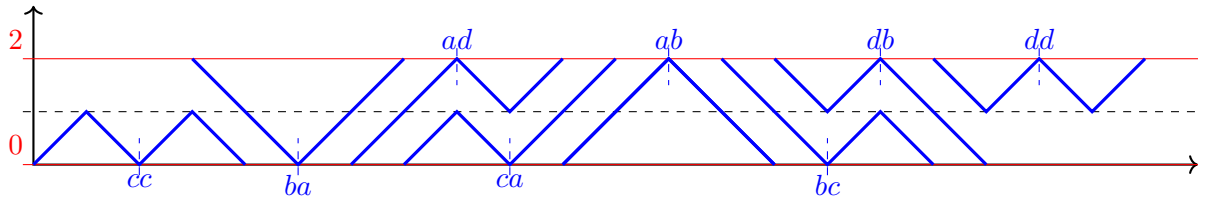

Figure 8: Every 2 letters configuration that can be found in a valid Dyck word of height at most 2.

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 9 and Figure 10 show only the cases for $+3$ strings. In Figure 9, every $+3$ string of size 3 is include in aa or ac so it is sufficient to search for this two couple. In Figure 10 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.

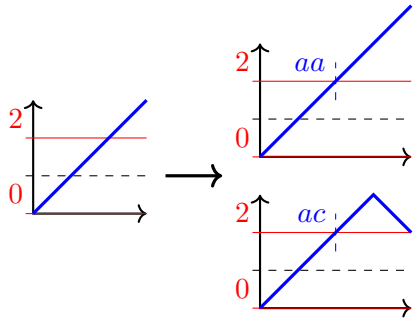

Figure 9: Configuration for a $+3$ strings of size 3.

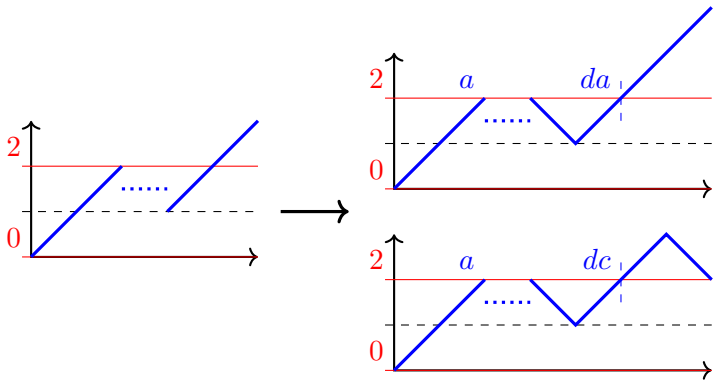

Figure 10: Configurations for a $+3$ string of size greater than 3.

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.

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

Algorithm 3 DyckFast$_{2,n}$

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

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.

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

3.3 A new algorithm for k=3

3.4 A try for a new algorithm for any k.

4 Conclusion

Conclusion here.

References

[1] Scott Aaronson, Daniel Grier, and Luke Schaeffer. A quantum query complexity trichotomy for regular languages, 2018.

[2] Andris Ambainis. Understanding quantum algorithms via query complexity.

[3] Andris Ambainis. Quantum lower bounds by quantum arguments, 2000.

[4] 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] 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.

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

[7] 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.

[8] Ryan O'Donnell. The adversary method: Lecture 20 of quantum computation at cmu. 2018.

[9] Ben W. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, oct 2009.

[10] Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Inf. Control.*, 8:190–194, 1965.

[11] 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.

[12] Robert Spalek and Mario Szegedy. All quantum adversary methods are equivalent. 2004.

[13] Crespi-Reghizzi Stefano, Guida Giovanni, and Mandrioli Dino. Noncounting context-free languages. 1978.

List of Figures

List of Algorithms

5 Appendix

The frame of the intership

A Proof of the super-basic adversary method

As a little reminder here is the theorem.

Theorem A.1. *Let define YES the set equal to $f^{-1}(accepted)$ and NO the set equal to $f^{-1}(rejected)$. If it exist two subset $Y \subseteq YES$ and $Z \subseteq NO$ such that:*

- *For each y in Y, there are at least m strings z in Z with $dist(y,z) = 1$.*
- *For each z in Z, there are at least m' strings y in Y with $dist(y,z) = 1$.*

Then the quantum query complexity $Q(f)$ is in $\Omega(\sqrt{m \times m'})$

Understanding this proof is important as it will give the intuitions to understand the general adversary.

Proof Theorem A.1. First, let define $\mathcal{R} := \{(y,z)$ such that $y \in Y, z \in Z$, and $\text{dist}(y,z) = 1\}$. Now, it is necessary to redefine the progress notion such that $\mathcal{P}(t) := \sum_{(y,z)\in\mathcal{R}} |\langle\psi_y^t|\psi_z^t\rangle|$ where $|\psi_x^t\rangle$ is the state of the algorithm after t query to the entry x. So, $\mathcal{P}(0) = |\mathcal{R}|$ and $\mathcal{P}(T) = |\frac{2}{3}\mathcal{R}|$. In order to prove the theorem, lets try to prove that for all t,

$$\mathcal{P}(t) - \mathcal{P}(t+1) \leq \frac{2}{\sqrt{mm'}}|\mathcal{R}|$$

as it is a correct value of d such that the lower bound is $O(\sqrt{mm'})$. Moreover, there are relations between size of the sets:

$$|\mathcal{R}| \geq m|Y|, |\mathcal{R}| \geq m'|Z| \implies |2\mathcal{R}| \geq m|Y| + m'|Z|.$$

Thus, instead of proving $\mathcal{P}(t) - \mathcal{P}(t+1) \leq \frac{2}{\sqrt{mm'}}|\mathcal{R}|$, lets try to prove

$$\mathcal{P}(t) - \mathcal{P}(t+1) \leq \frac{1}{\sqrt{mm'}}(m|Y| + m'|Z|) = \sqrt{\frac{m}{m'}}|Y| + \sqrt{\frac{m'}{m}}|Z|.$$

Now, lets take any a couple (y,z) in $\mathcal{R}$, it exists a unique i^* such that $y_{i^*} \neq z_{i^*}$. This small variation between y and z can be well translated to the progress after quantum query gate. Lets write $|\psi_y^t\rangle$ and $|\psi_z^t\rangle$:

$$|\psi_y^t\rangle = \sum_{\substack{i\in[\![0,N]\!] \\ j\in[\![1,d_i]\!]}} \alpha_{i,j}|i,j\rangle \qquad |\psi_z^t\rangle = \sum_{\substack{i\in[\![0,N]\!] \\ j\in[\![1,d_i]\!]}} \beta_{i,j}|i,j\rangle.$$

Now by applying Q_t, the states become

$$|\psi_y^t\rangle = \sum_{\substack{i\in[\![0,N]\!] \\ j\in[\![1,d_i]\!]}} (-1)^{(0y)_{i+1}}\alpha_{i,j}|i,j\rangle \qquad |\psi_z^t\rangle = \sum_{\substack{i\in[\![0,N]\!] \\ j\in[\![1,d_i]\!]}} (-1)^{(0z)_{i+1}}\beta_{i,j}|i,j\rangle.$$

However, y and z only differ on the index i^*, so the restricted progress measure to y and z at t and $t+1$ are equal to

$$\mathcal{P}_{y,z}(t) = \sum_{\substack{i\in[\![0,N]\!]\\ j\in[\![1,d_i]\!]}} \overline{\alpha_{i,j}}\beta_{i,j} \qquad \mathcal{P}_{y,z}(t+1) = \sum_{\substack{i\in[\![0,N]\!]\\ j\in[\![1,d_i]\!]}} (-1)^{(0y)_{i+1}+(0z)_{i+1}}\overline{\alpha_{i,j}}\beta_{i,j}$$
$$= \sum_{\substack{i\in[\![0,N]\!]\\ j\in[\![1,d_i]\!]}} \overline{\alpha_{i,j}}\beta_{i,j} - 2\sum_{j\in[\![1,d_{i^*}]\!]} \overline{\alpha_{i^*,j}}\beta_{i^*,j}$$
$$= \mathcal{P}_{y,z}(t) - 2\sum_{j\in[\![1,d_{i^*}]\!]} \overline{\alpha_{i^*,j}}\beta_{i^*,j}.$$

So, for restricted progress measure there is the inequality

$$|\mathcal{P}_{y,z}(t)| - |\mathcal{P}_{y,z}(t+1)| \le |\mathcal{P}_{y,z}(t) - \mathcal{P}_{y,z}(t+1)|$$
$$\le |2\sum_{j\in[\![1,d_{i^*}]\!]} \overline{\alpha_{i^*,j}}\beta_{i^*,j}|$$
$$\le 2\sum_{j\in[\![1,d_{i^*}]\!]} |\alpha_{i^*,j}||\beta_{i^*,j}|.$$

Moreover, the math trick for all p, q real numbers and h non negative real number $2pq \le hp^2 + \frac{1}{h}q^2$ allow to rewrite the previous inequality to

$$|\mathcal{P}_{y,z}(t)| - |\mathcal{P}_{y,z}(t+1)| \le 2\sum_{j\in[\![1,d_{i^*}]\!]} |\alpha_{i^*,j}||\beta_{i^*,j}|$$
$$\le \sum_{j\in[\![1,d_{i^*}]\!]} h|\alpha_{i^*,j}|^2 + \frac{1}{h}|\beta_{i^*,j}|^2.$$

By chosing h equal to $\sqrt{\frac{m}{m'}}$, it becomes

$$|\mathcal{P}_{y,z}(t)| - |\mathcal{P}_{y,z}(t+1)| \le \sum_{j\in[\![1,d_{i^*}]\!]} \sqrt{\frac{m}{m'}}|\alpha_{i^*,j}|^2 + \sqrt{\frac{m'}{m}}|\beta_{i^*,j}|^2.$$

Now, it is time to sum on every couple (y,z) of $\mathcal{R}$. But i^* is depending on y,z as much as $\alpha_{i,j}$s and $\beta_{i,j}$s so lets give them y and z as argument

$$\mathcal{P}(t) - \mathcal{P}(t+1) = \sum_{(y,z)\in\mathcal{R}} \left|\mathcal{P}_{y,z}(t)\right| - \sum_{(y,z)\in\mathcal{R}} \left|\mathcal{P}_{y,z}(t+1)\right|$$
$$\le \sum_{\substack{(y,z)\in\mathcal{R}\\ j\in[\![1,d^*_{i^*_{y,z}}]\!]}} \sqrt{\frac{m}{m'}}|\alpha^y_{i^*_{y,z},j}|^2 + \sqrt{\frac{m'}{m}}|\beta^z_{i^*_{y,z},j}|^2.$$

Now, it is useful to do the first summation in two part in order to find a upped bound to the

sum

$$\mathcal{P}(t) - \mathcal{P}(t+1) \leq \sqrt{\frac{m}{m'}} \sum_{\substack{y \ st \ \exists z \\ (y,z)\in\mathcal{R}}} \sum_{\substack{z \\ (y,z)\in\mathcal{R}}} \sum_{\substack{\text{width} \\ j\in[\![1,d_{i^*_{y,z}}]\!]}} |\alpha^y_{i^*_{y,z},j}|^2 + \sqrt{\frac{m'}{m}} \sum_{\substack{z \ st \ \exists y \\ (y,z)\in\mathcal{R}}} \sum_{\substack{y \\ (y,z)\in\mathcal{R}}} \sum_{\substack{\text{width} \\ j\in[\![1,d_{i^*_{y,z}}]\!]}} |\beta^z_{i^*_{y,z},j}|^2.$$

Now, at y fixed, $z \mapsto i^*_{y,z}$ is an injective function from $\{z, (y,z) \in \mathcal{R}\}$ to $[\![1,N]\!]$ as y and z are at distance 1. Thus $(z,j) \mapsto (i^*_{y,z}, j)$ is also an injective function from $\{(z,j), (y,z) \in \mathcal{R}$ and $j \in [\![1, d_{i^*_{y,z}}]\!]\}$ to $[\![1,N]\!] \times [\![1, d_{i^*_{y,z}}]\!]$. So, it implies that

$$\bigsqcup_{\substack{z \ \text{width} \\ (y,z)\in\mathcal{R}}} \{i^*_{y,z}\} \times [\![1, d_{i^*_{y,z}}]\!] \subseteq \bigsqcup_{n\in[\![0,N]\!]} \{n\} \times [\![1,d_n]\!]$$

and finally that

$$\sum_{\substack{z \ \text{width} \\ (y,z)\in\mathcal{R}}} \sum_{j\in[\![1,d_{i^*_{y,z}}]\!]} |\alpha^y_{i^*_{y,z},j}|^2 \leq \sum_{i\in[\![0,N]\!]} \sum_{j\in[\![1,d_i]\!]} |\alpha^y_{i,j}|^2 \leq \left\langle \psi^t_y | \psi^t_y \right\rangle \leq 1.$$

By symmetry it also works for the second sum, so the difference of progress is now the following

$$\begin{aligned}
\mathcal{P}(t) - \mathcal{P}(t+1) \leq & \sqrt{\frac{m}{m'}} \sum_{\substack{y \ st \ \exists z \\ (y,z)\in\mathcal{R}}} \sum_{\substack{z \\ (y,z)\in\mathcal{R}}} \sum_{\substack{\text{width} \\ j\in[\![1,d_{i^*_{y,z}}]\!]}} |\alpha^y_{i^*_{y,z},j}|^2 \\
& + \sqrt{\frac{m'}{m}} \sum_{\substack{z \ st \ \exists y \\ (y,z)\in\mathcal{R}}} \sum_{\substack{y \\ (y,z)\in\mathcal{R}}} \sum_{\substack{\text{width} \\ j\in[\![1,d_{i^*_{y,z}}]\!]}} |\beta^z_{i^*_{y,z},j}|^2 \\
\leq & \sqrt{\frac{m}{m'}} \sum_{\substack{y \ st \ \exists z \\ (y,z)\in\mathcal{R}}} 1 + \sqrt{\frac{m'}{m}} \sum_{\substack{z \ st \ \exists y \\ (y,z)\in\mathcal{R}}} 1 \\
\leq & \sqrt{\frac{m}{m'}} |Y| + \sqrt{\frac{m'}{m}} |Z| \\
\leq & \frac{2}{\sqrt{mm'}} |\mathcal{R}|.
\end{aligned}$$

To conclude, this give a lower bound $\Omega(\sqrt{mm'})$ for the quantum query complexity of f.

□

B 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: $|x| = n$
  $x \leftarrow 1^k x 0^k$
  $v \leftarrow \text{FindAny}_{k+1}(0, n + 2 * k - 1, \{1, -1\})$
  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
    $v_d \leftarrow \text{FindFixedLength}_k(l, r, d, s)$ 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 $\{l, l + 1, \ldots, r\}$ such that
    $v_t \leftarrow \text{FindAtLeftMost}_k(l, r, t, d, s)$ is **not** Null
  return v_t of Nullif none

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 $v \neq$ Null **then**
    $v' = (i_2, j_2, \sigma_2) \leftarrow \text{FindAtRightMost}_{k-1}(l, r, i_1 - 1, d - 1, \{1, -1\})$
    if $v' =$ Null **then**
      $v' = (i_2, j_2, \sigma_2) \leftarrow \text{FindFirst}_{k-1}(\max(l, j_1 - d + 1), i_1 - 1, \{1, -1\}, left)$
    if $v' \neq$ Null and $\sigma_2 \neq \sigma_1$ **then** $v' \leftarrow$ Null
    if $v' =$ Null **then**
      $v' = (i_2, j_2, \sigma_2) \leftarrow \text{FindAtLeftMost}_{k-1}(l, r, j_1 + 1, d - 1, \{1, -1\})$
    if $v' =$ Null **then**
      $v' = (i_2, j_2, \sigma_2) \leftarrow \text{FindFirst}_{k-1}(j_1 + 1, \max(r, i_1 + d - 1), \{1, -1\}, right)$
    if $v' =$ Null **then return** Null
  else
    $v = (i_1, j_1, \sigma_1) \leftarrow \text{FindFirst}_{k-1}(t, min(t + d - 1, r), \{1, -1\}, right)$
    if v = Null **then return** Null
    $v' = (i_2, j_2, \sigma_2) \leftarrow \text{FindFirst}_{k-1}(max(t - d + 1, l), t, \{1, -1\}, left)$
    if $v' =$ 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 $\textsc{FindFirst}_k(l, r, s, left)$

Require: $0 \le l < r$ and $s \subseteq \{1, -1\}$
  $lBorder \leftarrow l, rBorder \leftarrow r, d \leftarrow 1$
  while $lBorder + 1 < rBorder$ **do**
    $mid \leftarrow \lfloor (lBorder + rBorder)/2 \rfloor$
    $v_l \leftarrow \textsc{FindAny}_k(lBorder, mid, s)$
    if $v_l \neq \textsc{Null}$ **then** $rBorder \leftarrow mid$
    else
      $v_{mid} \leftarrow \textsc{FindFixedPos}_k(lBorder, rBorder, mid, s, left)$
      if $v_{mid} \neq \textsc{Null}$ **then return** v_{mid}
      else $lBorder \leftarrow mid + 1$
    $d \leftarrow d + 1$
  return Null

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

Require: $0 \le l < r$, $l \le t \le r$ 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
    $v_d \leftarrow \textsc{FindAtLeftMost}_k(l, r, t, d, s)$ is **not** Null
  return v_d or Null if none

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

Theorem C.1. $\textbf{Dyck}_{k,n}$'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* $k \ge 1$
2. $Q(\textsc{FindAny}_{k+1}(l, r, s)) = O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right)$ *for* $k \ge 1$
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* $k \ge 2$
4. $Q(\textsc{FindAtLeftMost}_{k+1}(l, r, t, d, s)) = \begin{cases} O\left(\sqrt{d}(\log_2(d))^{0.5(k-2)}\right) & \text{for } k \ge 2 \\ O(1) & \text{for } k = 1 \end{cases}$
5. $Q(\textsc{FindFirst}_k(l, r, s, left)) = O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right)$ *for* $k \ge 2$
6. $Q(\textsc{FindFixedPos}_k(l, r, t, s)) = \begin{cases} O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right) & \text{for } k \ge 3 \\ O(1) & \text{for } k = 2 \end{cases}$

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

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 C.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 $\textsc{FindFirst}_{k+1}$.

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

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

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 C.1 is true for $k \in \mathbb{N}^*$