École Normale Supérieure de Lyon

Latvijas Universitāte

ENS DE LYON

UNIVERSITY OF LATVIA
FACULTY OF COMPUTING

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

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

2 Preliminaries

- Quantum Query Complexity
- $\textsc{Dyck}_{k,n}$ problem
- Trichotomy statement
- Adversary methode

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

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 we know that every minimal ± 2 string is of size 2. So the logarithmic search of the upper bound on the size of the minimal ± 2 string is no more useful and the algorithm can be summarized to applying a Grover search for 2 consecutive 0 or two consecutive 1. This lower the quantum query complexity of the initial case of the function to $O(\sqrt{n})$ instead of $O(\sqrt{n \log_2(n)})$. This give us this following algorithm for $\textsc{FindAny}_k$.

Algorithm 1 $\textsc{FindAny}_k$(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
      $v_d \leftarrow \textsc{FindFixedLength}_k(l, r, d, s)$ is **not** $\textsc{Null}$
    return v_d or $\textsc{Null}$ if none
  else
    Find t in $\{l, l+1, \ldots, r\}$ such that
      $v_t \leftarrow \textsc{FindAtLeftMost}_2(l, r, t, 2, s)$ is **not** $\textsc{Null}$
    return v_t of $\textsc{Null}$if none

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 2 $\textsc{FindFixedPos}_k(l, r, t, s)$

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
      $v_d \leftarrow \textsc{FindAtLeftMost}_k(l, r, t, d, s)$ is **not** NULL
    return v_d or NULL if none
  else $v \leftarrow \textsc{FindAtLeftMost}_k(l, r, t, 2, s)$ is **not** NULL
    return v_d or NULL if none

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

Proof Theorem 3.1. The behavior of the $\textsc{Dyck}_{k,n}$ 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. $Q(\textsc{Dyck}_{k,n}) = O\left(\sqrt{n}(\log_2(n))^{0.5(k-1)}\right)$ *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. $Q(\textsc{FindFixedPos}_k(l, r, t, s)) = \begin{cases} O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-2)}\right) & \text{for } k \ge 3 \\ O(1) & \text{for } k = 2 \end{cases}$

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 Dyck$_{2,n}$

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

If we come back to the case were $k = 1$, the query complexity comes only from a call to Grover's search because rejecting is easily by finding a 00 or a 11 substrings inside the entry. For $k = 2$ it no more possible as the substrings that reject are of the form 00(10)*0 or of the form 11(01)*1. It implies that the number of calls to Grover's search in the naive approach is in $O(n)$ so the quantum query complexity finally becomes $O\left(n\sqrt{n}\right)$. 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 will define a new alphabet that allow to express every even length binary strings and that will have convenient property compatible with Grover's search. Let $\mathcal{A} = \{a, b, c, d\}$ the alphabet where a corresponds to 00, b to 11, c to 01, and d to 10. So every string of size 2 has its letter in $\mathcal{A}$ thus every even length bit string is expressed in $\mathcal{A}^*$.

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

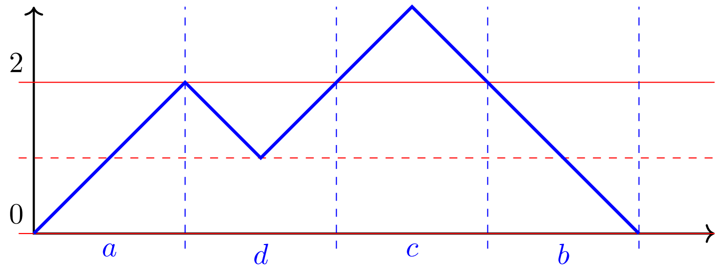

Figure 1: Illustration of the letters of $\mathcal{A}$ using Dyck's representation.

This property is important as it implies that every ± 3 strings uses at least two letter.

4 Conclusion

bonsoir paris

bonnoaisn

boabdado

ibabdiubaoiz

iabdiabdo

References

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

5 Appendix

The frame of the intership

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

Algorithm 3 $\textsc{Dyck}_{k,n}$

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 = $\textsc{Null}$

Algorithm 4 $\textsc{FindAny}_k(l, r, s)$

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** $\textsc{Null}$
  return v_d or $\textsc{Null}$ if none

Algorithm 5 $\textsc{FindFixedLength}_k(l, r, d, s)$

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** $\textsc{Null}$
  return v_t of $\textsc{Null}$if none

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

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

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

- For $k = 1$, only $\textsc{FindAtLeftMost}_2$, $\textsc{FindAny}_2$,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

Algorithm 6 FINDATLEFTMOST$_k(l, r, d, t, s)$

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

Algorithm 7 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**
    $mid \leftarrow \lfloor (lBorder + rBorder)/2 \rfloor$
    $v_l \leftarrow$ FINDANY$_k(lBorder, mid, s)$
    if $v_l \ne$ NULL **then** $rBorder \leftarrow mid$
    else
      $v_{mid} \leftarrow$ FINDFIXEDPOS$_k(lBorder, rBorder, mid, s, left)$
      if $v_{mid} \ne$ NULL **then return** v_{mid}
      else $lBorder \leftarrow mid + 1$
    $d \leftarrow d + 1$
  return NULL

Algorithm 8 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
      $v_d \leftarrow$ FINDATLEFTMOST$_k(l, r, t, d, s)$ is **not** NULL
  return v_d or NULL if none

from the $O\left(\sqrt{\frac{r-l}{d}}\right)$ calls to FindAtLeftMost$_3$. The $O\left(\sqrt{(r-l)\log_2(r-l)}\right)$ quantum query complexity of FindAny$_3$ comes from the $O\left(\sqrt{\log_2(r-l)}\right)$ calls to FindFixedLength$_3$. Finally, the $O\left(\sqrt{(r-l)\log_2(r-l)}\right)$ quantum query complexity of Dyck$_2$ comes from the call to FindAny$_3$.

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

First, the $O\left(\sqrt{r-l}(\log_2(r-l))^{0.5(k-1)}\right)$ quantum query complexity of FindFixedPos$_{k+1}$ comes from the $O\left(\sqrt{\log(r-l)}\right)$ calls to 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 FindFirst$_{k+1}$ comes from the dichotomize search using calls to FindAny$_{k+1}$ and 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 FindAtLeftMost$_{k+2}$from the constant amount of calls to FindAtLeftMost$_{k+1}$, FindAtRightMost$_{k+1}$,and FindFirst$_{k+1}$.

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

---

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

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 3.2 is true for $k \in \mathbb{N}^*$