Miscellaneous Results

1 Catalan Objects and Dyck Paths

Dyck paths are one interpretation of the Catalan numbers. Here, we will instead consider a more general form of Dyck Paths, which correspond to numbers in the *Catalan Trapezoid.*

A Dyck path can be constructed as a $2n$ step one-dimensional random walk (Figure 1). Each step in the walk moves one unit along the positive x-axis and one unit up or down the positive y-axis. Given these restrictions, we would obtain a 1D random walk pinned to zero on both sides. A Dyck path also has the additional restriction that the y-coordinate of any point on the random walk is ≥ 0. i.e. the walk is always north of the origin.

The number of possible Dyck paths is the n^{th} Catalan number –

$$C_n = \frac{1}{n+1} \cdot \binom{2n}{n}$$

We will attempt to support queries to a uniformly random instance of a Dyck path. Specifically, we will want to answer queries of the form $\textsc{Height}(i)$, which returns the position of the path after i steps.

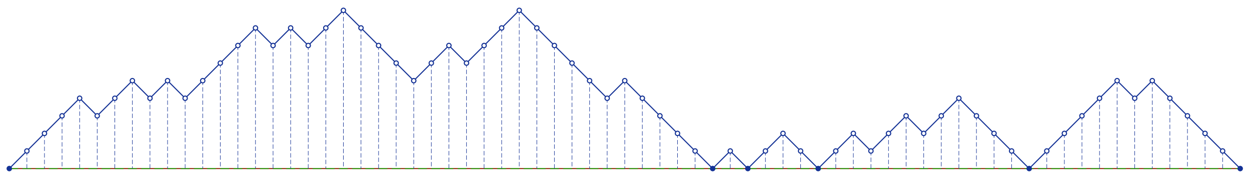


Figure 1: Simple Dyck path with $n = 35$.

1.1 Catalan Trapezoids and Generalized Dyck Paths

First, we define Catalan trapezoids as presented in [Reu14]. Let $C_k(n, m)$ be the $(n, m)^{th}$ entry of the Catalan trapezoid of order k, where $C_1(n, m)$ corresponds to the Catalan triangle.

The interpretation is as follows. Consider a sequence of n up-steps and m down-steps, such that the sum of any initial sub-string is not less than $1 - k$. This means that we start our Dyck path at a height of $k-1$, and we are never allowed to cross below zero (Figure 2). The total number of such paths is exactly $C_k(n,m)$. For $k=1$, we obtain the definition of the simple Dyck path (Figure 1).

Now, we state a result from [Reu14] without proof

$$C_k(n,m) = \begin{cases} \binom{n+m}{m} & 0 \le m < k \\ \binom{n+m}{m} - \binom{n+m}{m-k} & k \le m \le n+k-1 \\ 0 & m > n+k-1 \end{cases}$$

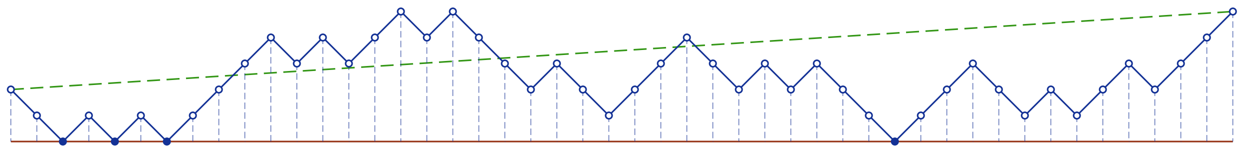


Figure 2: Complex Dyck path with $n=25$, $m=22$ and $k=3$. Notice that the boundary is shifted.

1.2 Generating Dyck Paths

Our general recursive step is as follows. We consider a sequence of length $2S$ comprising of $2U$ up moves $(+1)$ and $2D$ down moves (-1). Additionally, the sum of any initial sequence prefix? canon be less than $k-1$. Without loss of generality, let's assume that $2D \le S$. If this were not the case, we could simply flip the sequence and negate the elements. This essentially means that the overall Dyck path is non-decreasing.

Lemma 1. $S - 2D = \mathcal{O}(\log n\sqrt{S}) \implies U - D = \mathcal{O}(\log n\sqrt{S})$

We want to sample the height of this path after S steps. This is the same as sampling the number of $(+1)$s that get assigned to the first half of the elements in the sequence. We define p_d as the probability that exactly $D-d$ (-1)s get assigned to the first half. This means that exactly $U+d$ $(+1)$s get assigned to the first half. Consequently, the second half will contain exactly $D+d$ (-1)s and $U-d$ $(+1)$s.

What is d is negative?

Let us first compute this probability.

$$p_d = \frac{D_{left} \cdot D_{right}}{D_{tot}}$$

Here, D_{left} denotes the number of valid starting sequences (first half) and D_{right} denotes the number of valid ending sequences. Here, *valid* means that each half sequence gets the appropriate number of ups and downs and the initial sums never drop below $1-k$. For, D_{right}, we will start the Dyck path from the end of the $2S$ sequence. In this case the invalidation threshold will be a different k'. This k' is the final height of the $2S$ sequence. So, $k' = k + 2U - 2D = k + 4S - 2D$. We will use this fact extensively moving forward.

Also, D_{tot} is the total number of possible sequences of length $2S$, given the initial conditions. Note that in this case the threshold remains at k.

1.3 The Simple Case

The problem of sampling reduces to the binomial sampling case when $k > \mathcal{O}(\log n)\sqrt{S}$ for some constant c. This is because with high probability, will never dip below the threshold. In this case, the we can simply approximate the probability as

$$\frac{\binom{S}{D-d} \cdot \binom{S}{D+d}}{\binom{2S}{2D}}$$

This is because unconstrained random walks will not dip below the $1-k$ threshold with high probability. This problem was solved in [GGN10] using $\mathcal{O}(poly(\log n))$ resources.

1.4 Path Segments Close to Zero

The problem arises when we $k < \mathcal{O}(\log n)\sqrt{S}$. In this case we need to compute the actual probability, Using the formula from [Reu14], we find that.

$$D_{left} = \binom{S}{D-d} - \binom{S}{D-d-k} \qquad D_{right} = \binom{S}{U-d} - \binom{S}{U-d-k'} \qquad D_{tot} = \binom{2S}{2D} - \binom{2S}{2D-k} \tag{1}$$

Here, $k' = k + 2U - 2D$, and so $k' = \mathcal{O}(\log n)\sqrt{S}$ (using Lemma 1).

The final distribution we wish to sample from is given by $\{p_d\}_d$ where $p_d = \frac{D_{left} \cdot D_{right}}{D_{tot}}$. To acieve this, we will use the following Lemma from [GGN10].

Lemma 2. *Let $\{p_i\}$ and $\{q_i\}$ be distributions satisfying the following conditions*

1. *There is a poly-time algorithm to approximate p_i and q_i up to $\pm n^{-2}$*
2. *Generating an index i according to q_i is closely implementable.*
3. *There exists a $poly(logn)$-time recognizable set S such that*
   - $1 - \sum_{i \in S} p_i$ *is negligible*
   - *There exists a constant c such that for every i, it holds that $p_i \leq \log^{\mathcal{O}(1)} n \cdot q_i$*

Then, generating an index i according to the distribution $\{p_i\}$ is closely-implementable.

In this process, we will first disregard all values of d where $|d| > \Theta(\log n\sqrt{S})$. The probability mass associated with these values can be shown to be negligible .

bound variance of path

Next, we will construct an appropriate $\{q_i\}$ and show that $p_d < \log^{\mathcal{O}(1)} n \cdot q_d$ for all $|d| < \Theta(\sqrt{S})$ and some constant c. We will use the following distribution

$$q_d = \frac{\binom{S}{D-d} \cdot \binom{S}{D+d}}{\binom{2S}{2D}} = \frac{\binom{S}{D-d} \cdot \binom{S}{U-d}}{\binom{2S}{2D}}$$

It is shown in [GGN10] that this distribution is closely implementable.

First, we consider the case where $k \cdot k' \leq 2U+1$. In this case, we use loose bounds for $D_{left} < \binom{S}{D-d}$ and $D_{right} < \binom{S}{U-d}$. We also use the following lemma (proven in Section A).

Lemma 3. *When $kk' > 2U+1$, $D_{tot} > \frac{1}{2} \cdot \binom{2S}{2D}$.*

Combining the three bounds we obtain $p_d < \frac{1}{2} q_d$. Intuitively, in this case the dyck boundary is far away, and therefore the number of possible paths is only a constant factor away from the number of unconstrained paths (no boundary).

The case where the boundaries are closer (i.e. $k \cdot k' \leq 2U+1$) is trickier, since the individual counts need not be close to the corresponding binomial counts. However, in this case we can still ensure that the sampling probability is within poly-logarithmic factors of the binomial sampling probability. We use the following lemmas (proven in Section A).

Lemma 4. *$D_{left} \leq c_1 \frac{k \cdot \log n}{\sqrt{S}} \cdot \binom{S}{D-d}$ for some constant c_1.*

Lemma 5. *$D_{right} < c_2 \frac{k' \cdot logn}{\sqrt{S}} \cdot \binom{S}{U-d}$ for some constant c_2.*

Lemma 6. *When $kk' \leq 2U+1$, $D_{tot} < c_3 \frac{k \cdot k'}{S} \cdot \binom{2S}{2D}$ for some constant c_3.*

We can now put these lemmas together to show that $p_d/q_d \leq \Theta(\log^2 n)$. Now, we can apply Lemma 2 to sample the value of d, which gives us the height of the Dyck path at the midpoint of the two given points.

Theorem 1. *There is an algorithm that given two points at distance a and b (with $a < b$) along a Dyck path of length $2n$, with the guarantee that no position between a and b has been sampled yet, returns the height of the path halfway between a and b. Moreover, this algorithm only uses $\mathcal{O}(poly(\log n))$ resources.*

Proof. If $b-a$ is even, we can set $S = (b-a)/2$. Otherwise, we first sample a single step from a to $a+1$, and then set $S = (b-a-1)/2$. Since there are only two possibilities for a single step, we can explicitly compute an approximation of the probabilities, and then sample accordingly. Now, if $S > \Theta(\log^2 n)$ we can simply use the rejection sampling procedure described above to obtain a $\mathcal{O}(poly(\log n))$ algorithm. Otherwise, we sample each step induividually. Since there are only $2S = \Theta(\log^2 n)$ steps, the sampling is still efficient. □

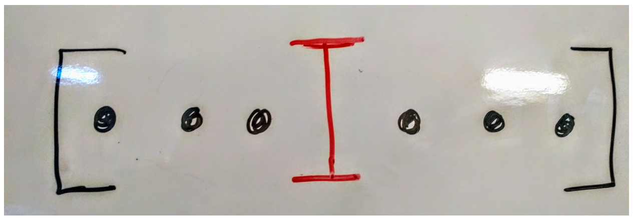
Theorem 2. *There is an algorithm that provides sample access to a Dyck path of length $2n$, by answering queries of the form* HEIGHT(x) *with the correctly sampled height of the Dyck path at position x using only $\mathcal{O}(poly(\log n))$ resources per query.*

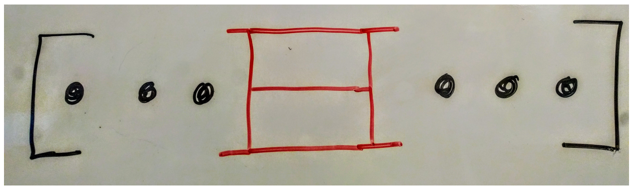
Proof. The algorithm maintains a successor-predecessor data structure (e.g. Van Emde Boas tree) to store all positions x that have already been queried. Each newly queried position is added to this structure. Given a query HEIGHT(x), the algorithm first finds the successor and predecessor (say a and b) of x among the alredy queried positions. This provides us the quarantee required to apply Theorem 1, which allows us to query the height at the midpoint of a and b. We then binary search by updating either the successor or predecessor of x. Once the interval length becomes less than $\Theta(\log^2 n)$, we perform the full sampling (as in Theorem 1) which provides us the height at position x. □

2 Domino Tiling the $2 \times n$ grid

We will consider the problem of tiling a square grid with dominos. This problem has a long history and various importtant applications in statistical physics. Specifically, we will focus on the local generation of domino tilings of a $2 \times n$ grid from the uniform distribution. The queries will be as an index, and the generator should report the orientation of the domino at the i^{th} position in the grid.

It is a well known result that the number of tilings of a $2 \times n$ grid is exactly F_n. To aid with generalization, we will instead allow the generator to respond with the splitting boundary of the current tiling instead. For example, in Figure 3a, the boundary is a vertical line at the specified position. In Figure 3b, the boundary is horizontal, indicating that there are two horizontal dominos at that location. Note that Figure 3c is impossible for a $2 \times n$ grid. It should be clear that this query model is equivalent. cite


(a)


(b)

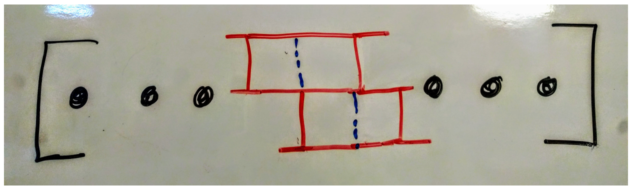

(c)

Figure 3: Caption for this figure with two images

Now, consider a query to the location i, such that all positions between $i - a$ and $i + b$ have not been queriesd so far. So, there is a blank $2 \times (a + b)$ size sub-grid that we have to sample from. Let us consider the number of possible tilings resulting from each possible splitting boundary.

1. Vertical Boundary – This indicates that we divide the region into two sub-grids with sizes $2 \times a$ and $2 \times b$. So, the total number of possible tilings is exactly $F_a \cdot F_b$.

2. Horizontal Boundary – This indicates that we divide the region into two sub-grids with sizes $2 \times (a-1)$ and $2 \times (b-1)$. So, the total number of possible tilings is exactly $F_{a-1} \cdot F_{b-1}$.

So the probabilities are computed as $\frac{F_a \cdot F_b}{F_a \cdot F_b + F_{a-1} \cdot F_{b-1}}$ and $\frac{F_{a-1} \cdot F_{b-1}}{F_a \cdot F_b + F_{a-1} \cdot F_{b-1}}$. Now, we face the issue of approximating these fractions. If either of the values a or b are less than $\Theta(\sqrt{n})$, then we can compute the exact value of the corresponding F_a or F_b. Otherwise, we use Lemma **??** to approximate $F_a = \phi \cdot F_{a-1}$ and $F_b = \phi \cdot F_{b-1}$. So, the probability of the vertical boundary becomes

$$\frac{F_a \cdot F_b}{F_a \cdot F_b + F_{a-1} \cdot F_{b-1}} = \frac{\phi^2}{\phi^2 + 1}$$

Similarly, the probability of a horizontal split with a top and bottom domino becomes $1/(\phi^2 + 1)$. Note that this also determines the two adjacent boundaries.

The only information we needed to make this query was the extent of the un-queried interval $[i - a, i + b]$. We can use any standard data-structure that allows insertion in positions $\{1, 2 \cdots, n+1\}$, and provides successor and predecessor queries.

Here's we can be fancy and use Van-Emde-Boas trees to get a $\mathcal{O}(\log \log n)$ query time. However, in some cases, the exact value of a Fibonacci number still needs to be computed, and this takes $\mathcal{O}(\log n)$ time. The faster queries only work when the new query is "far enough" ($\mathcal{O}(\log n)$ distance) away from all previous queries.

2.1 Permutations

In addition to a table (dictionary) containing all assigned values $\pi(i)$, we maintain the following binary trees, whose nodes are generated on-the-fly in response to queries.

T_1: Each node of T_1 corresponds to a specific range of indices, where the root represents the entire range $\{1, \ldots, N\}$, and its two children represents each (approximately) half of the parent's range. Each node counts the indices i in the range, such that $\pi(i) = \perp$. Initially T_1 only contains the root node, and the number of unassigned indices are N. The children are only generated when we need to traverse down from the root; as these nodes are generated, all of the indices in their ranges are unassigned. Once an index becomes assigned, we simply update the information along the path in $\mathcal{O}(\log n)$ time.

T_2: T_2 is similar to T_1 but instead of maintaining the number of indices in the range that are still unassigned, it maintains the number of values in the range that are still unused (have not been assigned to an index). Similarly, we may sample an unused value or mark it as used within $\mathcal{O}(\log n)$ time.

To compute $\pi(i)$, first we check the table for $\pi(i)$ and return its value if $\pi(i) \neq \perp$. Otherwise, sample an unused value j from T_2 and mark that value as used. Add $\pi(i) = j$ to the table, and mark index i as used on T_1.

If we wish to support $\pi^{-1}(i)$, then also store the table of $\pi^{-1}(i)$. To assign $\pi^{-1}(i)$, sample an unassigned index j from T_1 then mark it as assigned, add $\pi(j) = i$ and $\pi^{-1}(i) = j$ to the table, and mark the value i as used on T_2.

2.2 Generating Permutations with given Cyclic Structure

We will use the technique from [NR02] to locally generate a random permutation with a given cyclic structure. This algorithm uses two permutations π and σ, where π is a uniformly random permutation and σ is a fixed permutation with the given structure. The resulting random permutation is formed by the composition $\pi^{-1}(\sigma(\pi(\cdot)))$. We will generate the permutation π as described in the previous section.;

We receive as input a list of cycle sizes $\{c_1, c_2, \cdots, c_k\}$ with the restriction that $\sum c_i = n$. Now we need to locally generate the permutation σ with the prescribed structure. Define the indices $C_j = \sum_{i=1}^{j} c_i$ with $C_0 = 0$. We will construct the cycle corresponding to c_i as all the elements in the interval $\{c_{i-1} + 1, c_{i-1} + 2, \cdots, c_i\}$.

Of course we will not be computing σ explicitly. Instead, we will pre-compute the C_j indices, and when given a query $\sigma(x)$, we binary search amongst C_j to find the cycle that x belongs to. Then we can report the value of $\sigma(x)$ accordingly.

So, we can now compose the generator oracles for π, σ, and π^{-1} to get the full generator.

References

[GGN10] Oded Goldreich, Shafi Goldwasser, and Asaf Nussboim. On the implementation of huge random objects. *SIAM Journal on Computing*, 39(7):2761–2822, 2010.

[NR02] Moni Naor and Omer Reingold. Constructing pseudo-random permutations with a prescribed structure. *Journal of Cryptology*, 15(2):97–102, 2002.

[Reu14] Shlomi Reuveni. Catalan's trapezoids. *Probability in the Engineering and Informational Sciences*, 28(03):353–361, 2014.

A Dyck Path Generator

A.1 Assumptions

fix

- $d < c \cdot \sqrt{S} \log n$
- $k < c \cdot \sqrt{S} \log n \implies U - D < c \cdot \sqrt{S} \log n$
- $k' < c \cdot \sqrt{S} \log n$
- $S > \log^2 n \implies \sqrt{S} \log n < S$

Lemma 7. *For $x < 1$ and $k \geq 1$,*

$$1 - kx < (1-x)^k < 1 - kx + \frac{k(k-1)}{2}x^2.$$

Lemma 8. $D_{left} \leq c_1 \frac{k \cdot \log n}{\sqrt{S}} \cdot \binom{S}{D-d}$ *for some constant c_1.*

Proof. This involves some simple manipulations.

$$\begin{aligned}
D_{left} &= \binom{S}{D-d} - \binom{S}{D-d-k} && (2)\\
&= \binom{S}{D-d} \cdot \left[1 - \frac{(D-d)(D-d-1)\cdots(D-d-k+1)}{(S-D-d+k)(S-D-d+k-1)\cdots(S-D-d+1)}\right] && (3)\\
&\leq \binom{S}{D-d} \cdot \left[1 - \left(\frac{D-d-k+1}{S-D+d+k}\right)^k\right] && (4)\\
&\leq \binom{S}{D-d} \cdot \left[1 - \left(\frac{U+d+k-(U-D+d+k-1)}{U+d+k}\right)^k\right] && (5)\\
&\leq \binom{S}{D-d} \cdot \left[1 - \left(\frac{U+d+k-\mathcal{O}(\log n\sqrt{S})}{U+d+k}\right)^k\right] && (6)\\
&\leq \Theta\left(\frac{k \log n}{\sqrt{S}}\right) \cdot \binom{S}{D-d} && (7)
\end{aligned}$$

□

Lemma 9. $D_{right} < c_2 \frac{k' \cdot logn}{\sqrt{S}} \cdot \binom{S}{U-d}$ *for some constant c_2.*

Proof.

$$D_{right} = \binom{S}{U-d} - \binom{S}{U-d-k'} \tag{8}$$
$$= \binom{S}{U-d} \cdot \left[1 - \frac{(U-d)(U-d-1)\cdots(U-d-k'+1)}{(S-U-d+k')(S-U-d+k'-1)\cdots(S-U-d+1)}\right] \tag{9}$$
$$\leq \binom{S}{U-d} \cdot \left[1 - \left(\frac{U-d-k'+1}{S-U+d+k'}\right)^{k'}\right] \tag{10}$$
$$\leq \binom{S}{U-d} \cdot \left[1 - \left(\frac{2D-U-d-k+1}{2U-D+k+d}\right)^{k'}\right] \tag{11}$$
$$\leq \binom{S}{U-d} \cdot \left[1 - \left(\frac{U+k+d-(2U-2D+2d+2k-1)}{U+k+d}\right)^{k'}\right] \tag{12}$$
$$\leq \binom{S}{U-d} \cdot \left[1 - \left(\frac{U+k+d-\mathcal{O}(\log n\sqrt{S})}{U+k+d}\right)^{k'}\right] \tag{13}$$
$$\leq \Theta\left(\frac{k'\log n}{\sqrt{S}}\right) \cdot \binom{S}{U-d} \tag{14}$$

□

Lemma 10. ~~$D_{tot} \geq \binom{2S}{2D} \cdot \left[1 - \left(1 - \frac{k'}{2U+1}\right)^k\right]$.~~

change statement

Proof.

$$D_{tot} = \binom{2S}{2D} - \binom{2S}{2D-k} \tag{15}$$
$$= \binom{2S}{2D} \cdot \left[1 - \frac{(2D)(2D-1)\cdots(2D-k+1)}{(2S-2D+k)(2S-2D+k-1)\cdots(2S-2D+1)}\right] \tag{16}$$
$$\geq \binom{2S}{2D} \cdot \left[1 - \left(\frac{2D-k+1}{2S-2D+1}\right)^{k}\right] \tag{17}$$
$$\geq \binom{2S}{2D} \cdot \left[1 - \left(\frac{2U-(2U-2D+k-1)}{2U+1}\right)^{k}\right] \tag{18}$$
$$\geq \binom{2S}{2D} \cdot \left[1 - \left(\frac{(2U+1)-k'}{2U+1}\right)^{k}\right] \tag{19}$$
$$\geq \binom{2S}{2D} \cdot \left[1 - \left(1 - \frac{k'}{2U+1}\right)^{k}\right] \tag{20}$$

(21)

□

Reference previous lemma

Lemma 11. *When $kk' > 2U+1$, $D_{tot} > \frac{1}{2} \cdot \binom{2S}{2D}$.*

Proof. When $kk' > 2U+1 \implies k > \frac{2U+1}{k'}$, we will show that the above expression is greater than $\frac{1}{2}\binom{2S}{2D}$. Defining $\nu = \frac{2U+1}{k'} > 1$, we see that $(1-\frac{1}{\nu})^k \leq (1-\frac{1}{\nu})^\nu$. Since this is an increasing function of ν and since the limit of this function is $\frac{1}{e}$, we conclude that

$$1 - \left(1 - \frac{k'}{2U+1}\right)^k > \frac{1}{2}$$

□

Lemma 12. *When $kk' \leq 2U+1$, $D_{tot} < c_3 \frac{k \cdot k'}{S} \cdot \binom{2S}{2D}$ for some constant c_3.*

Proof. Now we bound the term $1 - \left(1 - \frac{k'}{2U+1}\right)^k$, given that $kk' \leq 2U+1 \implies \frac{kk'}{2U+1} \leq 1$. Using Taylor expansion, we see that

$$1 - \left(1 - \frac{k'}{2U+1}\right)^k \tag{22}$$

$$\leq \frac{kk'}{2U+1} - \frac{k(k-1)}{2} \cdot \frac{k'^2}{(2U+1)^2} \tag{23}$$

$$\leq \frac{kk'}{2U+1} - \frac{k^2k'^2}{2(2U+1)^2} \tag{24}$$

$$\leq \frac{kk'}{2U+1}\left(1 - \frac{kk'}{2(2U+1)}\right) \tag{25}$$

$$\leq \frac{kk'}{2(2U+1)} \leq \frac{kk'}{\Theta(S)} \tag{26}$$

$$\tag{27}$$

□