Enumerating m-Length Walks in Directed Graphs with Constant Delay

Duncan Adamson, Paweł Gawrychowski, Florin Manea

March 17, 2024

Motivation

- There are many settings in which we wish to enumerate every object within a class of combinatorial objects.
- As such classes are often of exceptional size relative to the description of the class.
- One such example is **regular languages**, where a deterministic, finite automaton can describe a class of possibly unbounded size.
    - We are motivated particularly by the class of *prefix closed langagues* avoiding a set of forbidden substrings, via a problem originating in chemistry.

Our Results

Theorem 1. There is an algorithm which, given a number m, enumerates, without repetitions, all walks of length m in a (directed) graph $G = (V, E)$ with a worst-case delay of $O(1)$, after preprocessing taking $O(|E|)$ time.

Overview

Given the number of walks in a graph (up to n^n), there are two main objectives with the output of our algorihtm:

- **Reconstructability**: The output must allow the most recently enumerated walk to be reconstructed "efficiently".
  - Lower Bound: $O(m)$ (for an m-length walk).
  - Upper Bound $O(n^n)$.
- **Efficient**: The algorithm needs to minimise the delay between outputs.
  - Lower Bound $O(1)$.
  - Upper Bound $O(n^n)$.

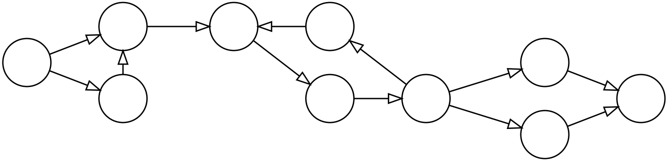
We match the lower bound for both conditions by introducing the data structures of **default edges**, **default paths**, and the **default graph**.

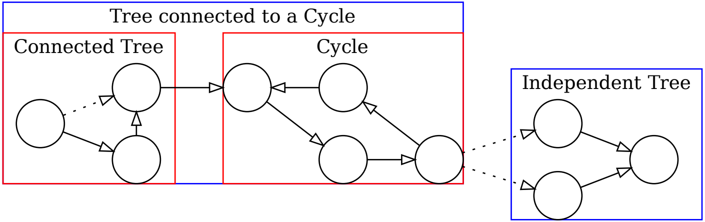
Default Edges, Paths, and Graphs

Definition

Given a graph $G = (V, E)$ and vertex $v \in V$, the **Default Edge** from v is the edge from v in the longest walk starting at v in $\mathcal{G}$. The **Default Path** from v is the path formed by following default edges from each state starting at v. The **Default Graph** of $\mathcal{G}$, $D(\mathcal{G})$, is the graph induced by the default edges.

Default Graph Example





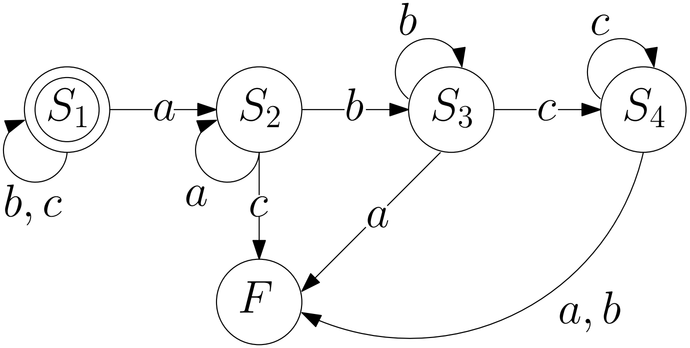
Using the Defualt Graph to represent walks

Key Idea. We represent each walk as a list of vertex-length pairs, each representing a sub walk. The tuple (v, ℓ) represents the walk on the delault path from v for ℓ steps.

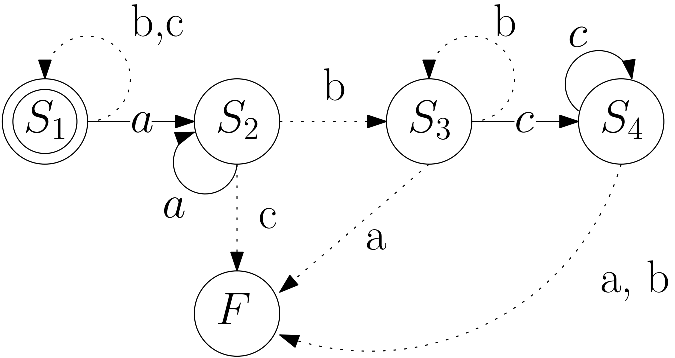
For **strings** accepted by a given automaton $\mathcal{A}$, each walk represents a string, and the tuples in the represetnation represent *substrings*

.

Representation example



Representation example

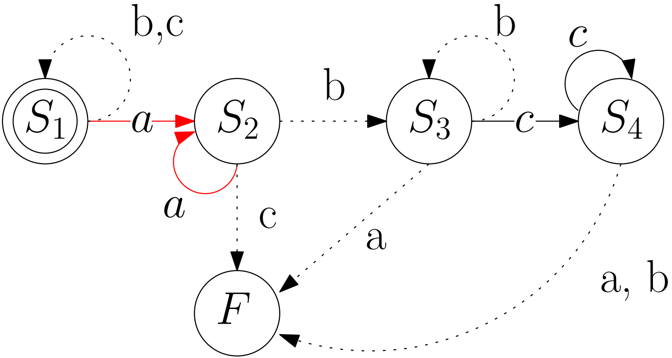


String	default path representation
aaaabbbbcccc	$[(S_1, 4), (S_3, 0), (S_3, 0), (S_3, 0), (S_3, 0), (S_3, 4)]$
aaaaaabccccc	$[(S_1, 6), (S_3, 5)]$
abcccccccccc	$[(S_1, 1)(S_3, 10)]$

Output

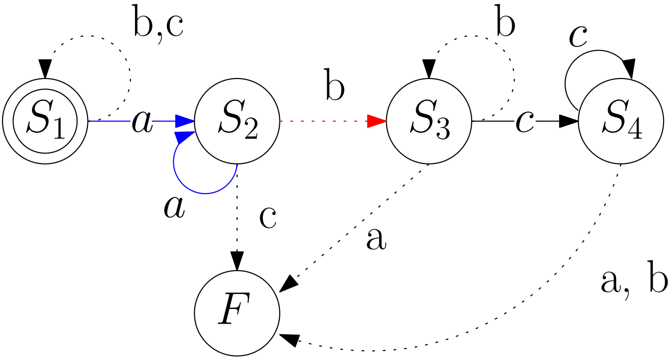
- At each step, we assume that we have a stack of state-length pairs, allowing the full word to be reconstructed in $O(n)$ time.
- Our goal now is to **edit** the stack with a constant number of operations in order to represent the next string.
- At each step, the output is a representation of the edit operations, allowing the strings to be explicitly output on demand.

Representation example



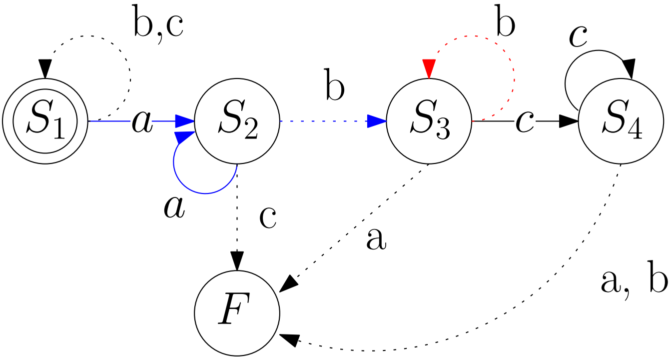
$(S_1, 4) \mapsto aaaa$

Representation example



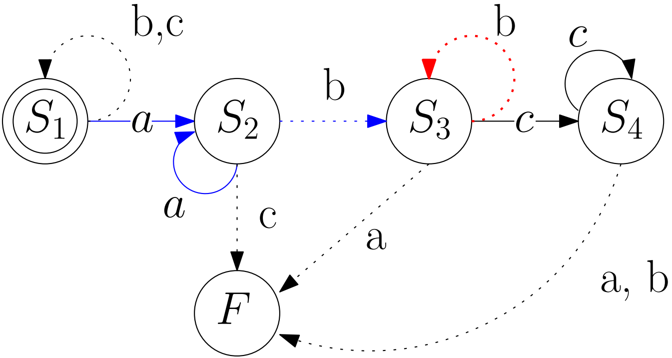
$(S_1, 4)(S_3, 0) \mapsto aaaab$

Representation example



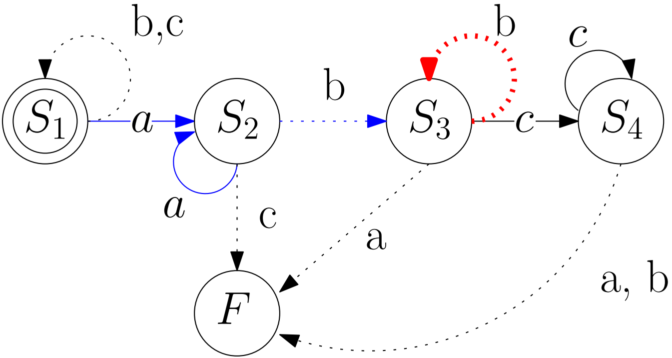
$(S_1, 4)(S_3, 0)(S_3, 0) \mapsto aaaabb$

Representation example



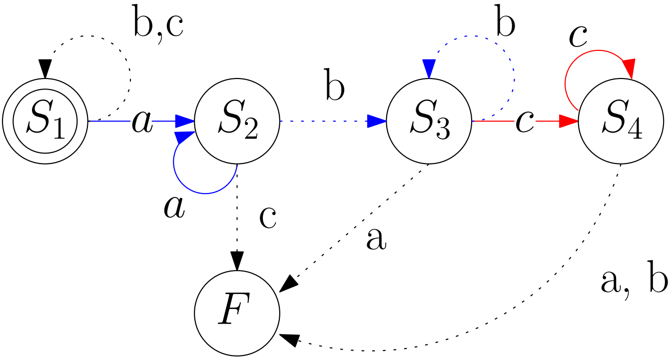
$(S_1, 4)(S_3, 0)(S_3, 0)(S_3, 0) \mapsto aaaabbb$

Representation example



$(S_1, 4)(S_3, 0)(S_3, 0)(S_3, 0)(S_3, 0) \mapsto aaaabbbb$

Representation example



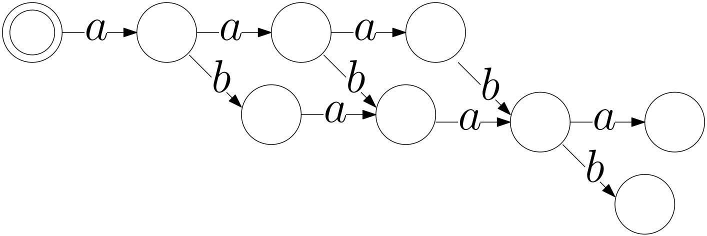
$(S_1, 4)(S_3, 0)(S_3, 0)(S_3, 0)(S_3, 0)(S_3, 4) \mapsto aaaabbbbcccc$

High Level Idea

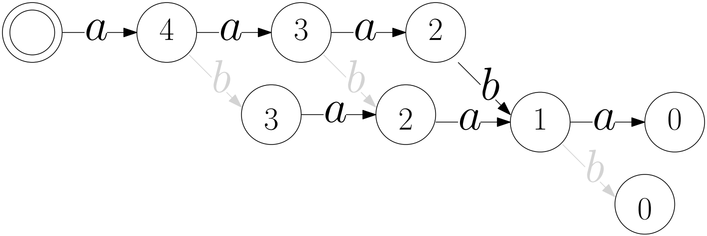
Our algorithm operates as follows:

- Starting with a walk W (which we assume has been output) we find the the possible ways of *branching* from this walk.
- Each node on the path is weighted by the depth of the node in the default path, and the length of the **second** longest walk from the node.
- Using these weights, we do a **range maximum query** on the graph, and choose the best state on the walk to branch from.

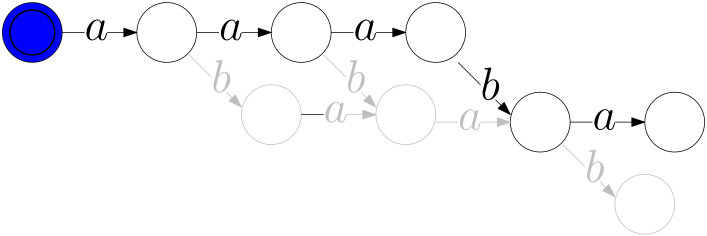
Example



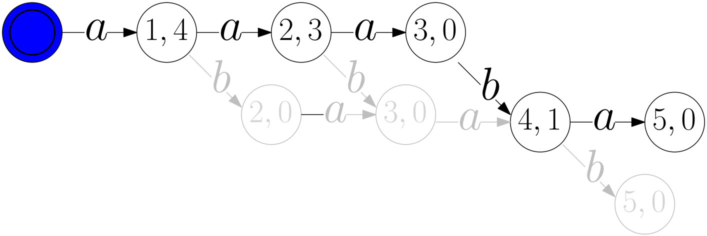
Example



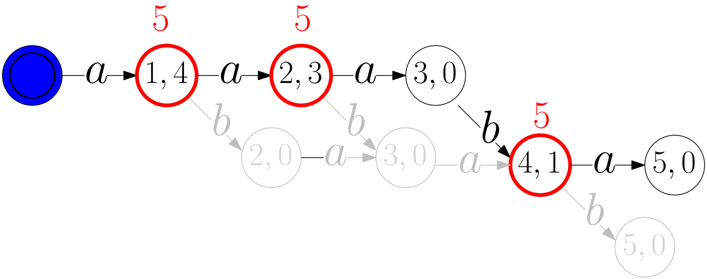
Example



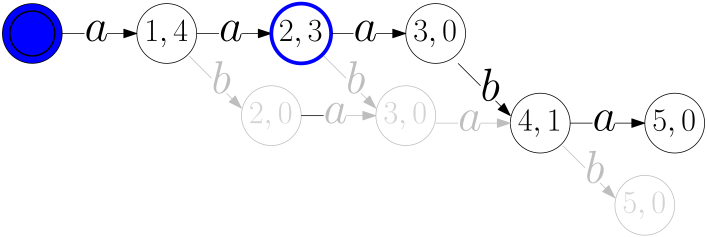
Example



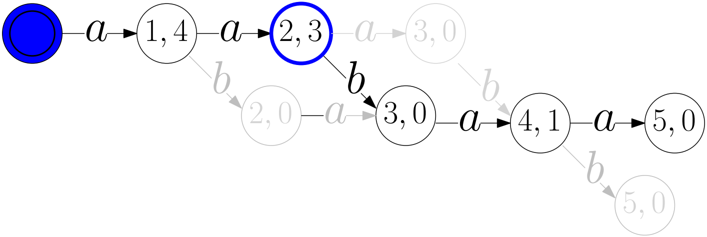
Example



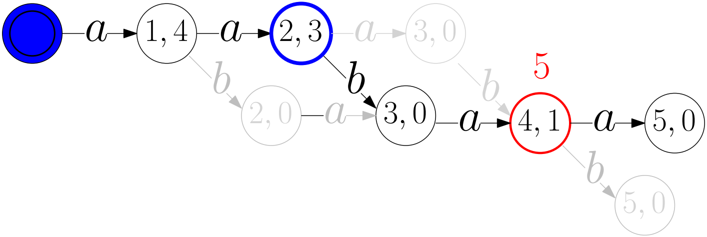
Example



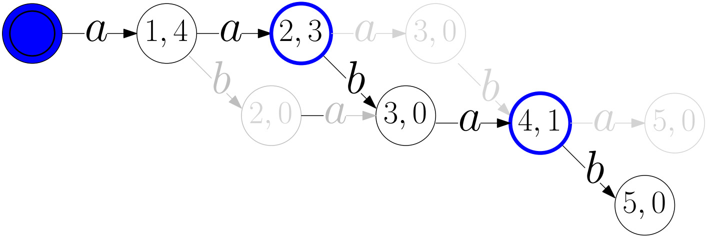
Example



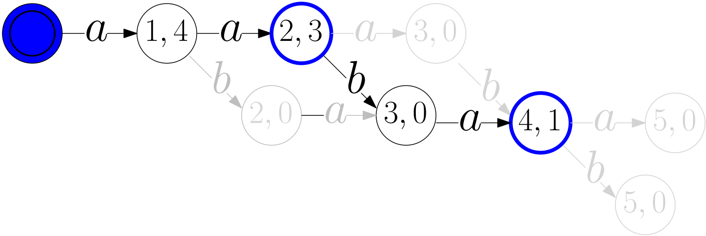
Example



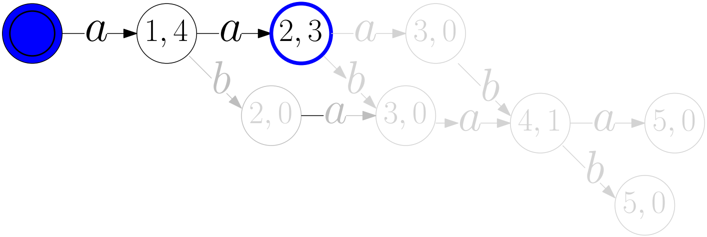
Example



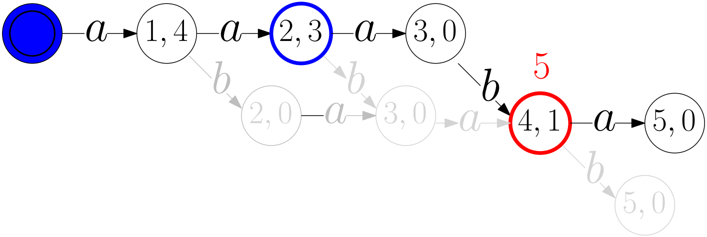
Example



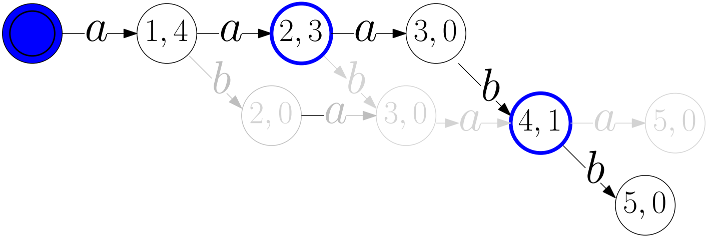
Example



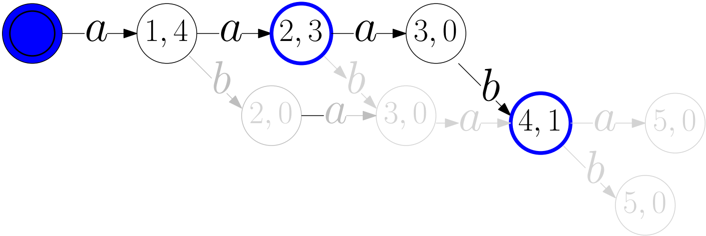
Example



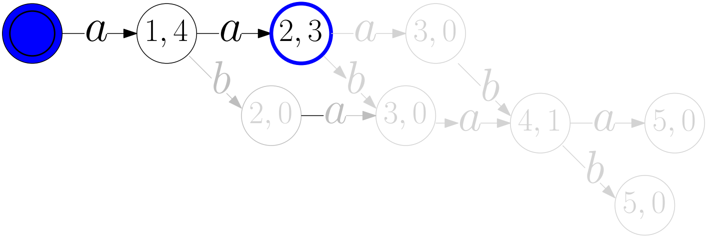
Example



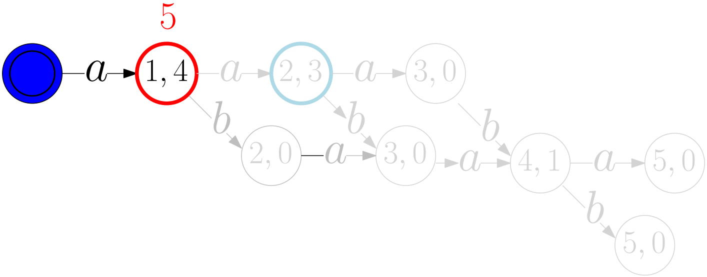
Example



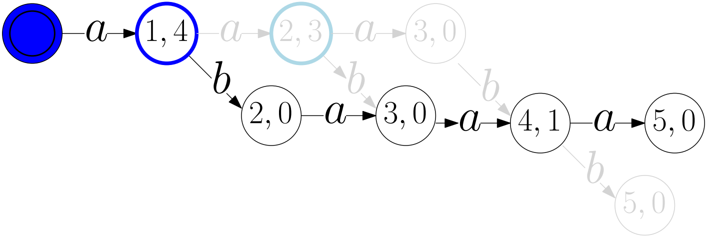
Example



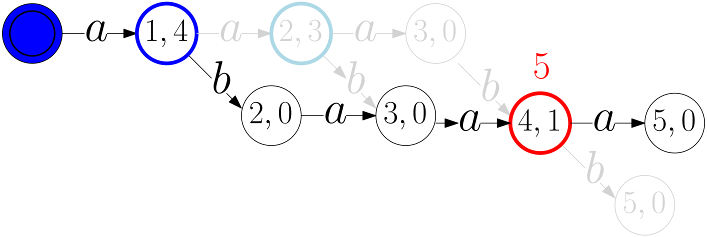
Example



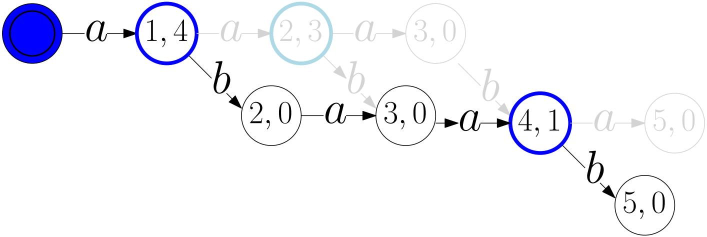
Example



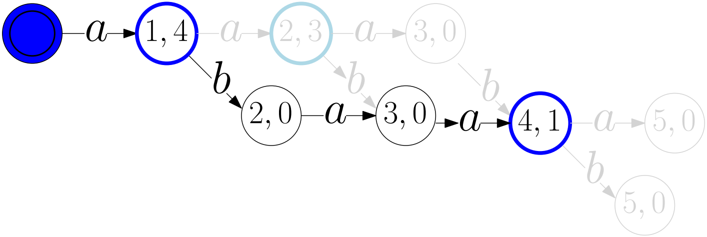
Example



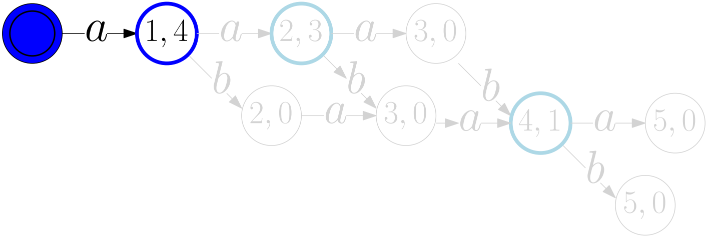
Example



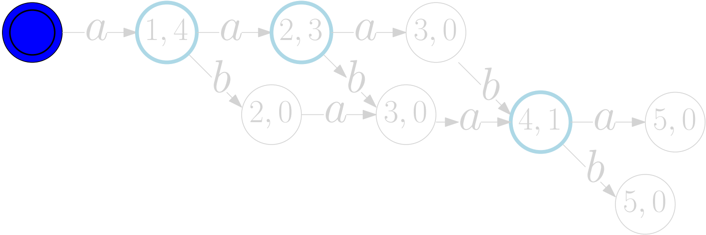
Example



Example



Example



Avoiding unnecessary backtracking

- To avoid unnecessary backtracking in the execution, we use **tail recursion**, to ensure that we can skip any backtracking to a node that is **exhausted**.
- A node is **exhausted** if every walk from the node has already been enumerated (from the current prefix).

Additional Results

- Along with the enumeration results, we give auxiliary results for **ranking** and **unranking** walks in the order output by the enumeration of the algorithm.
- The **rank** of a walk is the integer corresponding to the number of walks output before it.
- The **unranking** problem takes an integer i and asks for the walk with a rank of i.

Ranking and Unranking results (informal)

The **rank** of an m-length walk in the enumeration output order of the graph $G = (V, E)$ with m vertices can be computed in $O(|E| + mn^{\omega} + m^2\Delta)$ time, where ω is the exponent for matrix multiplication and Δ is the largest degree of any vertex in G.

Given an integer i, the i^{th} m-lengthwalk in the enumeration output order of the graph G with n vertices can be computed in $O(|E| + mn^{\omega} + m^2\Delta)$ time, where ω is the exponent for matrix multiplication and Δ is the largest degree of any vertex in G.

Summary

Results:

- An enumeration algorithm with constant delay for walks of length m in the graph G
- Polynomial time algorithms for **ranking** and **unranking** in enumeration order.

Open Problems:

- Can we enumerate **regular languages** with constant delay?
- Can we rank and unrank in linear (Or $O(\Delta n^{\omega})$) time?