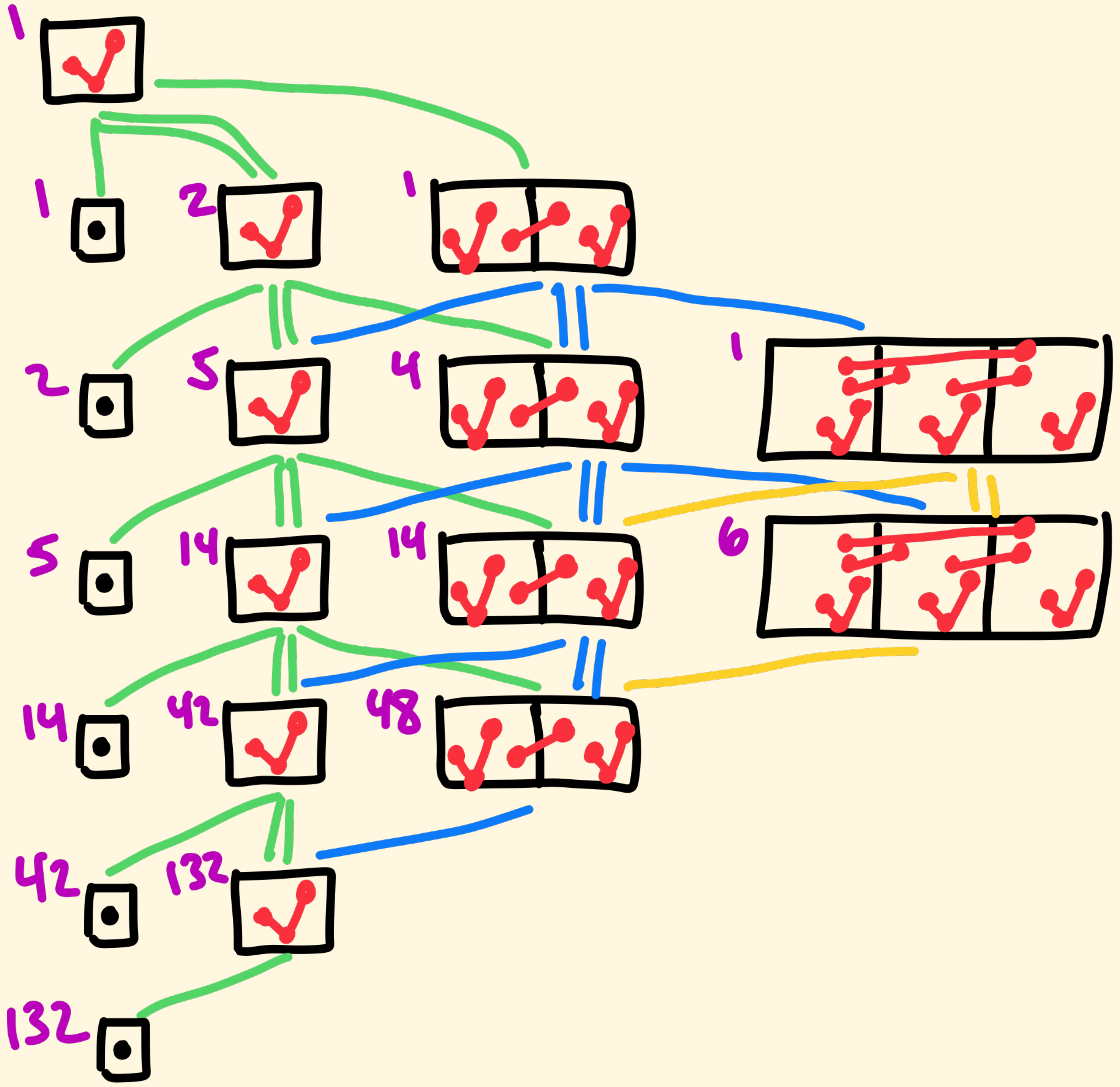
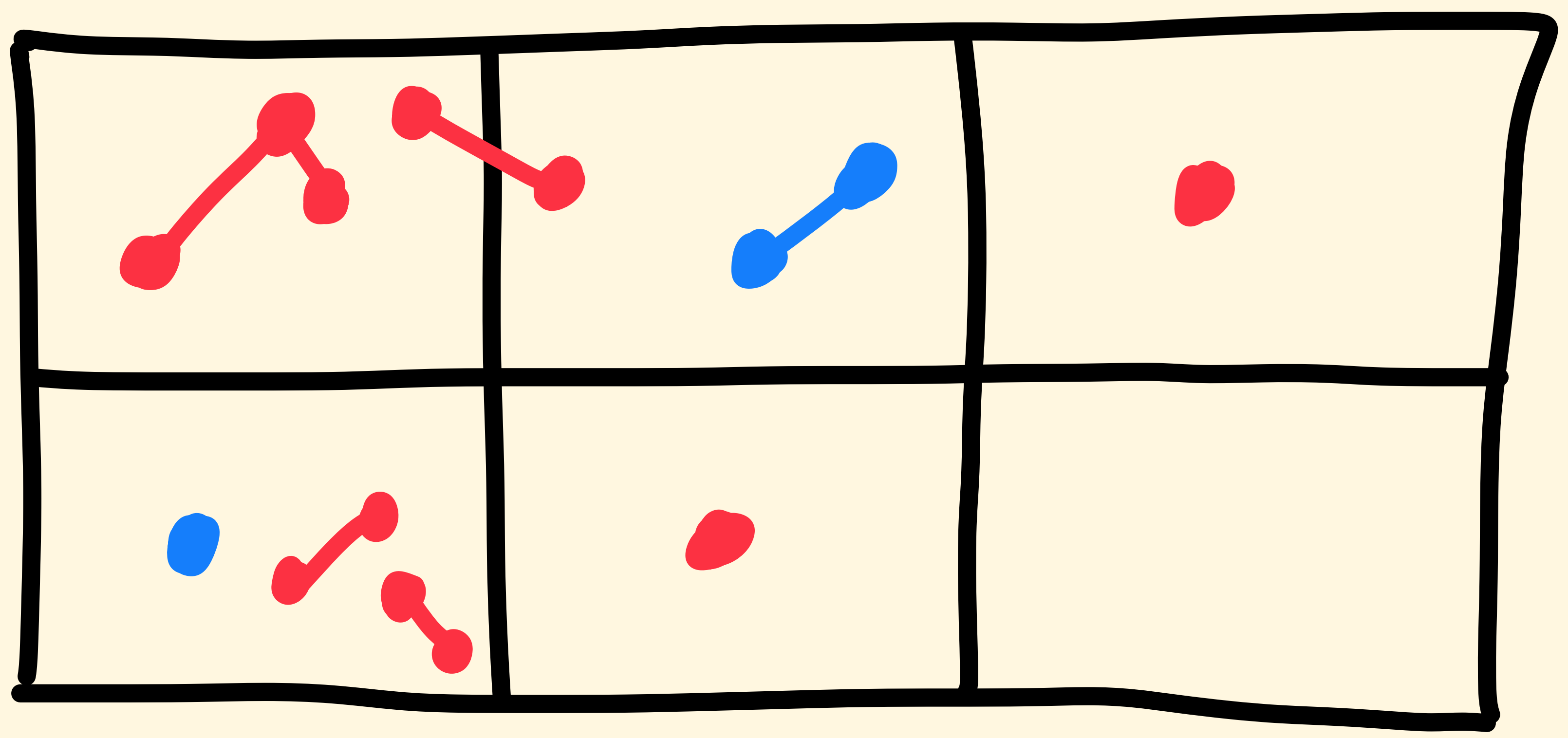
Counting Permutation Classes Quickly Using Tilings



Jay Pantone
Marquette University

joint work with Christian Bean



Permutation Patterns 2025
July 10, 2025

Inspiration

Contents lists available at ScienceDirect

Advances in Applied Mathematics

www.elsevier.com/locate/yaama

1324-avoiding permutations revisited

Andrew R. Conway, Anthony J. Guttmann, Paul Zinn-Justin *

School of Mathematics and Statistics, The University of Melbourne, Victoria 3010, Australia

A R T I C L E I N F O

Article history:
Received 28 October 2017
Accepted 9 January 2018
Available online 7 February 2018

MSC:
05A05
05A15
05A16

A B S T R A C T

We give an improved algorithm for counting the number of 1324-avoiding permutations, resulting in 14 further terms of the generating function, which is now known for all lengths ≤ 50. We re-analyse the generating function and find additional evidence for our earlier conclusion that unlike other classical length-4 pattern-avoiding permutations, the generating function does not have a simple power-law singularity, but rather, the number of 1324-avoiding permutations of length n behaves as

$$B \cdot \mu^n \cdot \mu_1^{\sqrt{n}} \cdot n^g.$$

We estimate $\mu = 11.600 \pm 0.003$, $\mu_1 = 0.0400 \pm 0.0005$, $g = -1.1 \pm 0.1$ while the estimate of B depends sensitively on the precise value of μ, μ_1 and g. This reanalysis provides substantially more compelling arguments for the presence of the stretched exponential term $\mu_1^{\sqrt{n}}$.

- computed the number of 1324-avoiding permutations up to length 50
- asymptotic analysis

Inspiration

ed the number of 1324-
g permutations up to length 50

otic analysis

ELSI

1324

Andr

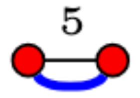
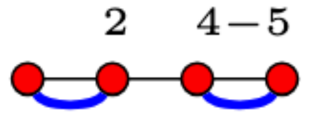
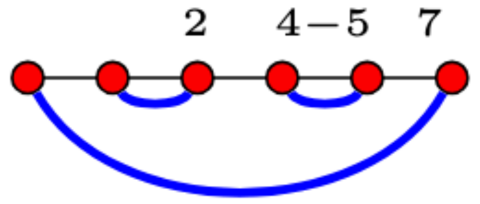
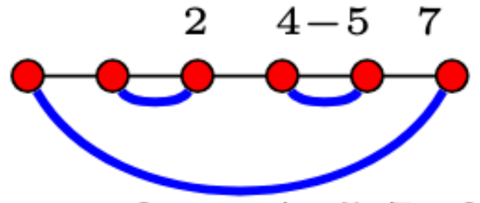
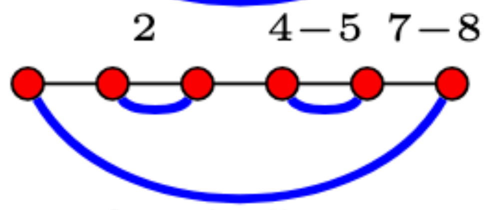
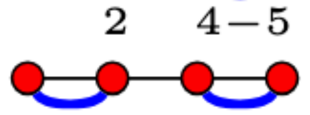
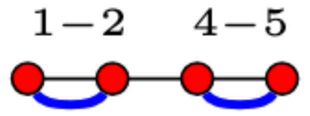
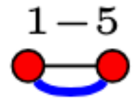
School
Austra

A R T

Article
Receiv
Accept
Availab

MSC:
05A05
05A15
05A16

Table 1
An example of the state as the permutation 5 4 2 7 10 8 9 1 3 6 is built up. The numbers above the link diagrams indicate the actual numbers that the link end represents. The numbers in parentheses correspond to the four types of insertion given in the text. Conversely, consider the similar permutation 5 4 2 7 10 6 9 1 3 8 which is not 1324 avoiding. The first five elements are the same; the sixth element is not allowed, as it would have to go inside the loop ending at 7.

Element	Notes	Result
	Start state	∅
5	Not consecutive with anything; future elements could go either side. (1)	
4	Consecutive with 5 and so merged into it. (3)	
2	New link as not consecutive with anything. (1)	
7	Larger than a previous link, makes constraint that no new elements between 2 and 7 may be added until every element greater than 7 has been added. (1)	
10	Removed from consideration as largest element. (3)	
8	Merged with 7. (2)	
9	Merges the 7 − 8 link with the largest element; said link removed from consideration. (4)	
1	Merges with 2 link. (3)	
3	Merges the 1 − 2 and 3 − 5 links. (4)	
6	Merges the 1 − 5 link with the largest element. (4)	∅

Inspiration

ELS

1324

Andr

School
Austra

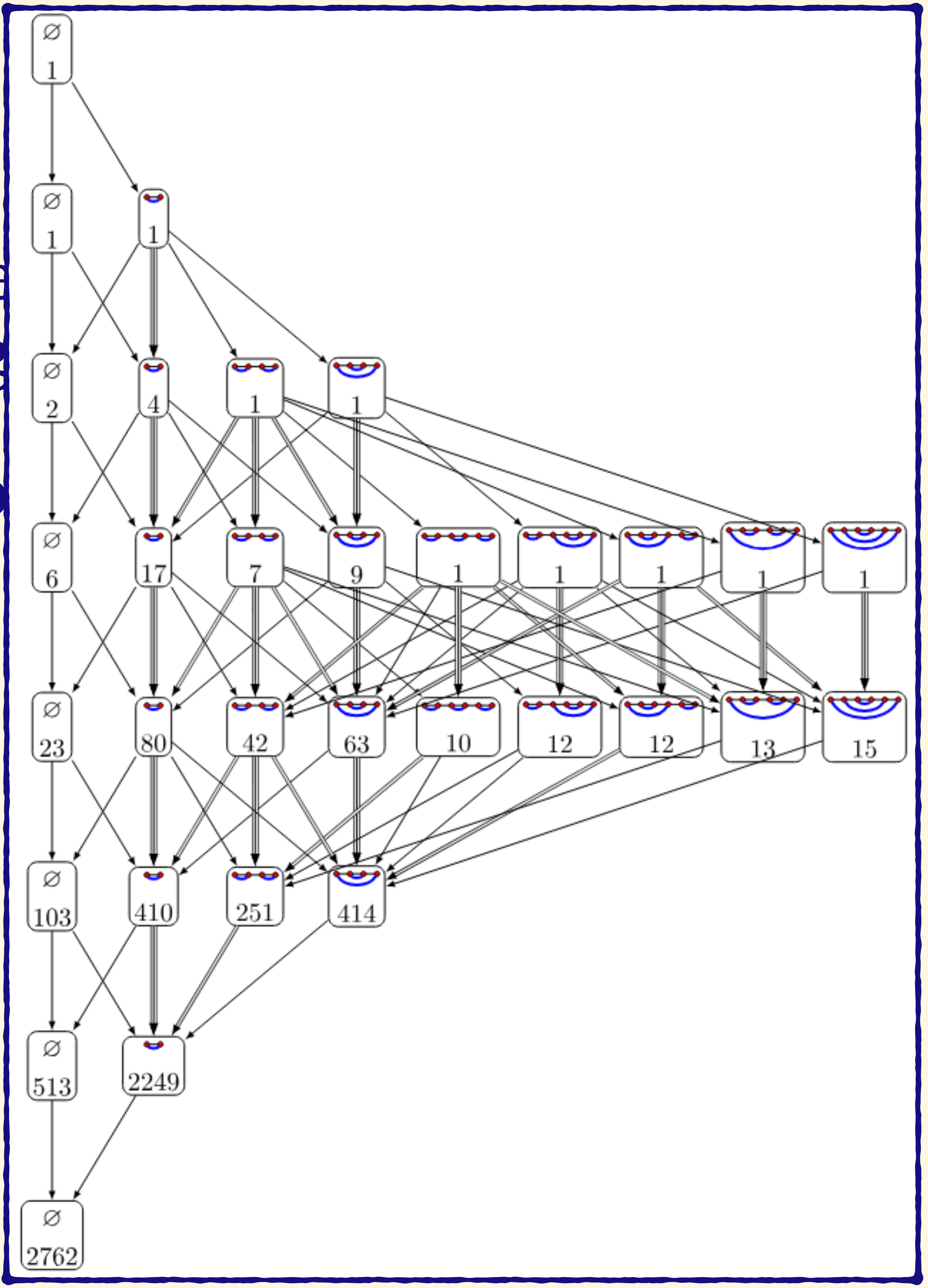
A R T

Article
Receiv
Accept
Availab

MSC:
05A05
05A15
05A16

Table 1
An example of the state as the permutation 5 4 2 7 10 8 9 1 3 6 is built up. The numbers above the link diagrams indicate the actual numbers that the link end represents. The numbers in parentheses correspond to the four types of insertion given in the text. Conversely, consider the similar permutation 5 4 2 7 10 6 9 1 3 8 which is not 1324 avoiding. The first five elements are the same; the sixth element is not allowed, as it would have to go inside the loop ending at 7.

Element	Notes	Result
	Start state	∅
5	Not consecutive with anything; future elements could go either side. (1)	5
4	Consecutive with 5 and so merged into it. (3)	4 – 5
2	New link as not consecutive with anything. (1)	2 4 – 5
7	Larger than a previous link, makes constraint that no new elements between 2 and 7 may be added until every element greater than 7 has been added. (1)	2 4 – 5 7
10	Removed from consideration as largest element. (3)	2 4 – 5 7
8	Merged with 7. (2)	2 4 – 5 7 – 8
9	Merges the 7 – 8 link with the largest element; said link removed from consideration. (4)	2 4 – 5
1	Merges with 2 link. (3)	1 – 2 4 – 5
3	Merges the 1 – 2 and 3 – 5 links. (4)	1 – 5
6	Merges the 1 – 5 link with the largest element. (4)	∅



Inspiration

Based on the first 50 terms, they predict the exponential growth rate for the number of 1324-avoiding permutations is ~11.600.

Inspiration

Based on the first 50 terms, they predict the exponential growth rate for the number of 1324-avoiding permutations is ~11.600.

The number of link patterns of size n is the nth Catalan number, so their exponential growth rate is 4.

Inspiration

Based on the first 50 terms, they predict the exponential growth rate for the number of 1324-avoiding permutations is ~11.600.

The number of link patterns of size n is the nth Catalan number, so their exponential growth rate is 4.

BUT: each operation adds at most one link, and takes away at most one link, and the counts we care about are the ones with zero links.

Inspiration

Based on the first 50 terms, they predict the exponential growth rate for the number of 1324-avoiding permutations is ~11.600.

The number of link patterns of size n is the nth Catalan number, so their exponential growth rate is 4.

BUT: each operation adds at most one link, and takes away at most one link, and the counts we care about are the ones with zero links.

So, they compute 1324-avoiding permutations to length n, we only need link patterns with at most $n/2$ links.

Inspiration

Based on the first 50 terms, they predict the exponential growth rate for the number of 1324-avoiding permutations is ~11.600.

The number of link patterns of size n is the nth Catalan number, so their exponential growth rate is 4.

BUT: each operation adds at most one link, and takes away at most one link, and the counts we care about are the ones with zero links.

So, they compute 1324-avoiding permutations to length n, we only need link patterns with at most $n/2$ links.

That makes this a $o((4+\epsilon)^{n/2}) = o((2+\epsilon)^n)$ algorithm!

Inspiration

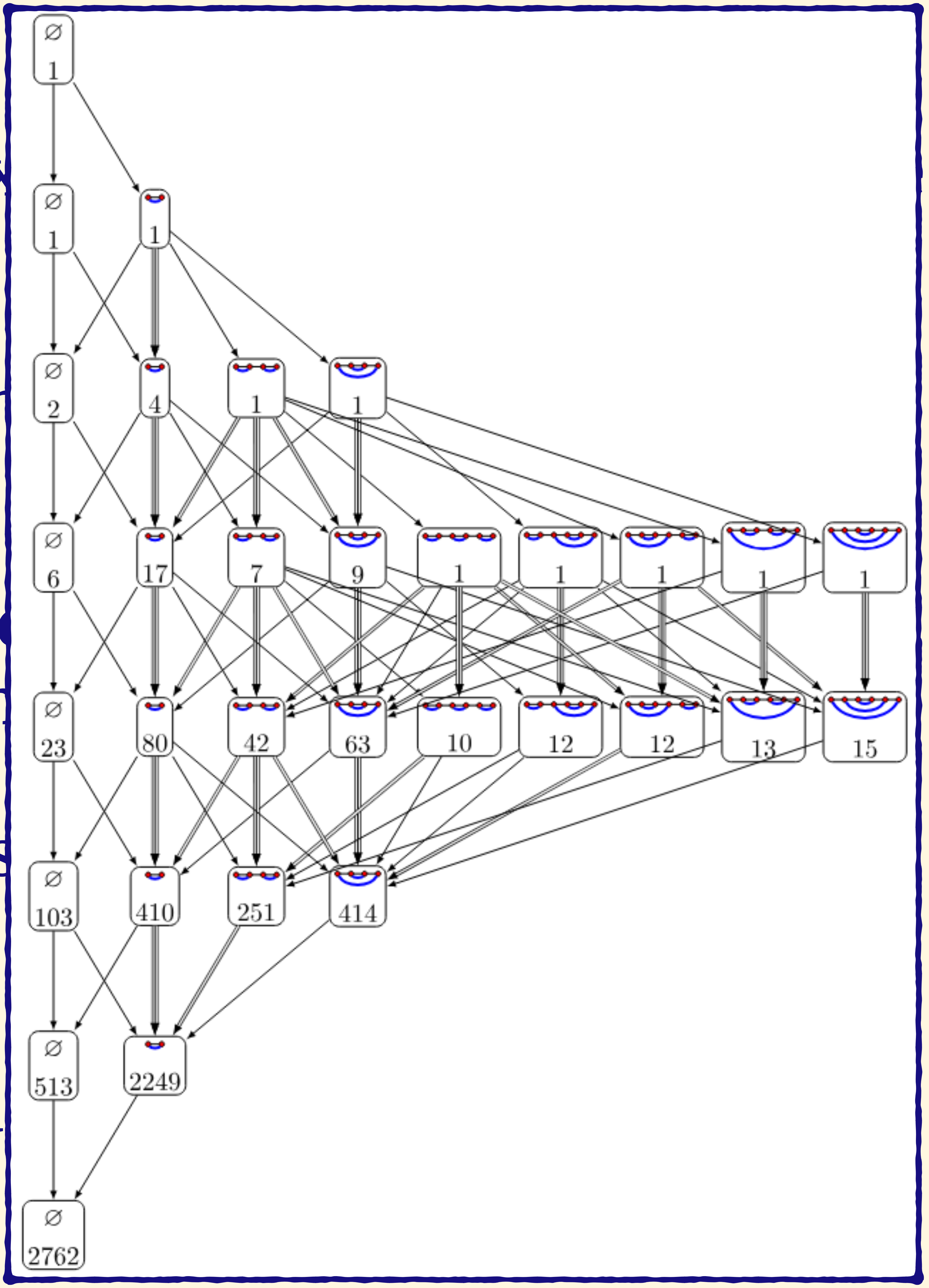
Based on the first 50 terms, they predict the ex
number of 1324-avoiding permutations is ~11.

The number of link patterns of size n is the nth
exponential growth rate is 4.

BUT: each operation adds at most one link, an
and the counts we care about are the ones with

So, they compute 1324-avoiding permutations
link patterns with at most $n/2$ links.

That makes this a $o((4+\epsilon)^{n/2}) = o((2+\epsilon)^n)$ a



Inspiration

It always bugged me that I didn't see the "big picture" of the paper.

Inspiration

It always bugged me that I didn't see the "big picture" of the paper.

I downloaded the paper onto my iPad to re-read on the flight home from Permutation Patterns 2023 in Dijon.

Inspiration

It always bugged me that I didn't see the "big picture" of the paper.

I downloaded the paper onto my iPad to re-read on the flight home from Permutation Patterns 2023 in Dijon.

By the time I landed, I understood the big picture, which gave me the idea for this project:

Inspiration

It always bugged me that I didn't see the "big picture" of the paper.

I downloaded the paper onto my iPad to re-read on the flight home from Permutation Patterns 2023 in Dijon.

By the time I landed, I understood the big picture, which gave me the idea for this project:

> Counting permutations avoiding *<u>any</u>* set of patterns by automatically discovering the "link patterns" for that set.

Insertion Encoding

(The Insertion Encoding of Permutations, Albert, Linton, and Ruškuc, 2005)

Encodes how a permutation can be built from bottom to top.

689274153

Insertion Encoding

(The Insertion Encoding of Permutations, Albert, Linton, and Ruškuc, 2005)

Encodes how a permutation can be built from bottom to top.

<u>689274153</u> $\diamond$

Insertion Encoding

(The Insertion Encoding of Permutations, Albert, Linton, and Ruškuc, 2005)

Encodes how a permutation can be built from bottom to top.

689274153

$$\overbrace{\diamond\ 1\ \diamond}^{\diamond}$$

middle placement

Insertion Encoding

(The Insertion Encoding of Permutations, Albert, Linton, and Ruškuc, 2005)

Encodes how a permutation can be built from bottom to top.

689274153

◇

◇1◇ middle placement

◇2◇1◇ middle placement

Insertion Encoding

(The Insertion Encoding of Permutations, Albert, Linton, and Ruškuc, 2005)

Encodes how a permutation can be built from bottom to top.

<u>689274153</u>

◇

middle placement

◇1◇

middle placement

◇2◇1◇

right placement

◇2◇1◇3

Insertion Encoding

(The Insertion Encoding of Permutations, Albert, Linton, and Ruškuc, 2005)

Encodes how a permutation can be built from bottom to top.

689274153

◇

middle placement

◇1◇

middle placement

◇2◇1◇

right placement

◇2◇1◇3

right placement

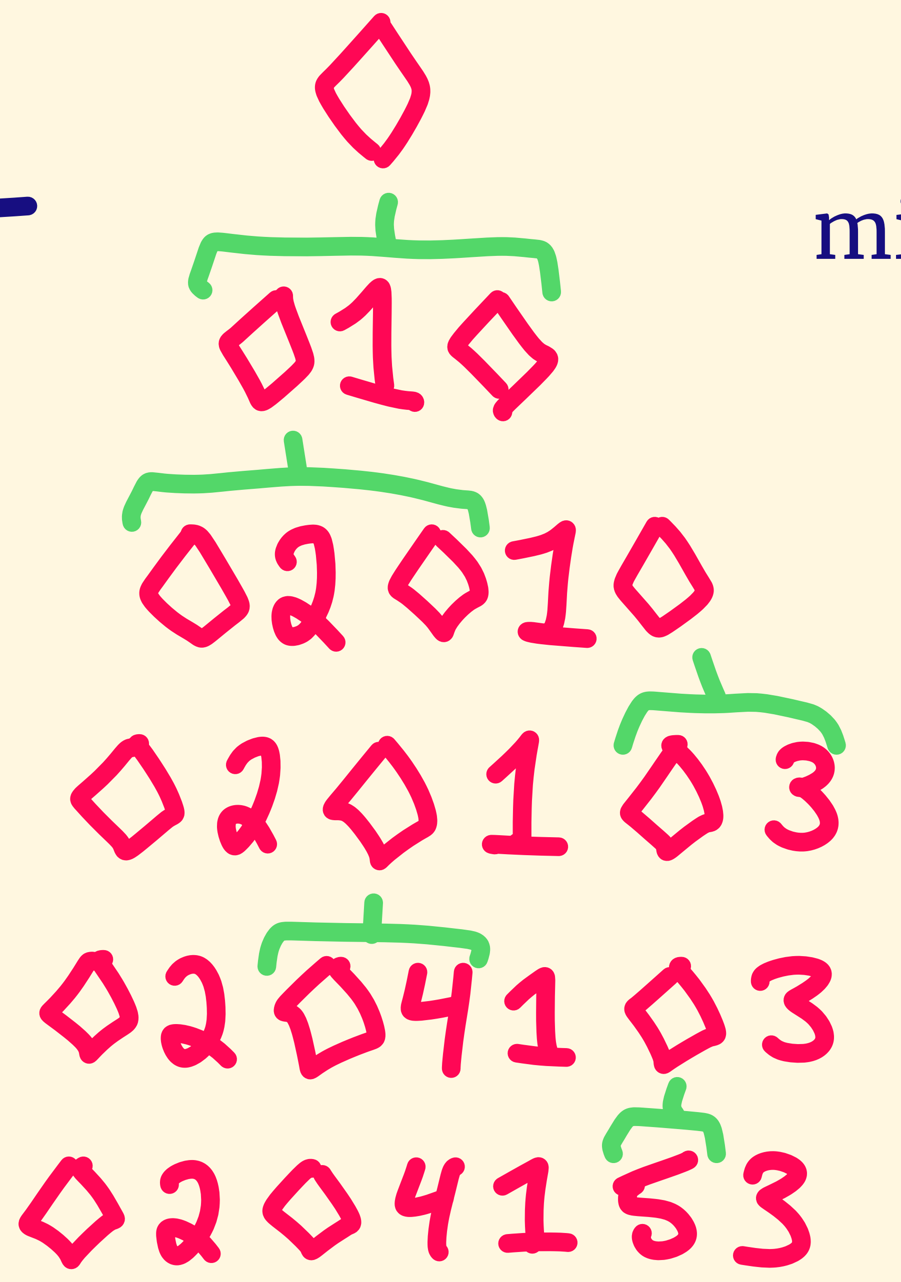
◇2◇41◇3

Insertion Encoding

(The Insertion Encoding of Permutations, Albert, Linton, and Ruškuc, 2005)

Encodes how a permutation can be built from bottom to top.

<u>689274153</u>



middle placement

middle placement

right placement

right placement

fill

Insertion Encoding

(The Insertion Encoding of Permutations, Albert, Linton, and Ruškuc, 2005)

Encodes how a permutation can be built from bottom to top.

689274153

◇

middle placement

◇1◇

middle placement

◇2◇1◇

right placement

◇2◇1◇3

right placement

◇2◇41◇3

fill

◇2◇4153

left placement

6◇2◇4153

Insertion Encoding

(The Insertion Encoding of Permutations, Albert, Linton, and Ruškuc, 2005)

Encodes how a permutation can be built from bottom to top.

689274153

◇

middle placement

◇1◇

middle placement

◇2◇1◇

right placement

◇2◇1◇3

right placement

◇2◇41◇3

fill

◇2◇4153

left placement

6◇2◇4153

fill

6◇274153

Insertion Encoding

(The Insertion Encoding of Permutations, Albert, Linton, and Ruškuc, 2005)

Encodes how a permutation can be built from bottom to top.

689274153

◇

middle placement

◇1◇

middle placement

◇2◇1◇

right placement

◇2◇1◇3

right placement

◇2◇41◇3

fill

◇2◇4153

left placement

6◇2◇4153

fill

6◇274153

left placement

68◇274153

Insertion Encoding

(The Insertion Encoding of Permutations, Albert, Linton, and Ruškuc, 2005)

Encodes how a permutation can be built from bottom to top.

689274153

◇

middle placement

◇1◇

middle placement

◇2◇1◇

right placement

◇2◇1◇3

right placement

◇2◇41◇3

fill

◇2◇4153

left placement

6◇2◇4153

fill

6◇274153

left placement

68◇274153

fill

689274153

Insertion Encoding

Some permutation classes have a “finite insertion encoding” — if you write down the stages of the insertion encodings of every permutation in the class, and simplify them in certain ways, you end up with a finite set.

(Finding Regular Insertion Encodings for Permutation Classes, Vatter, 2012)

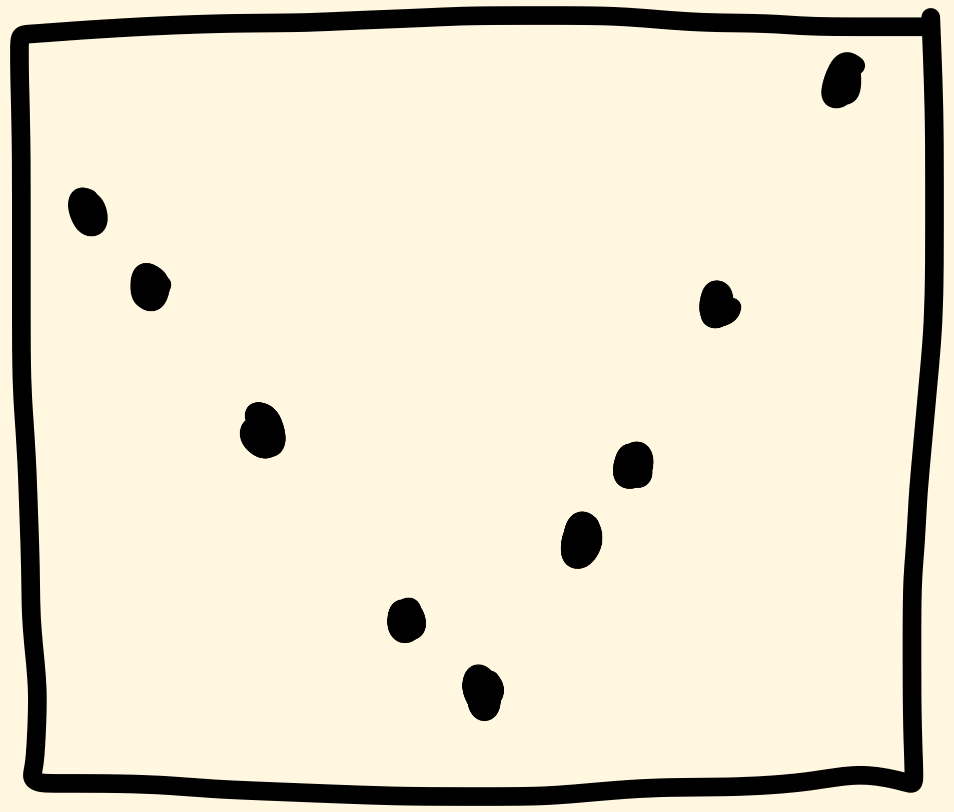
Insertion Encoding

Some permutation classes have a "finite insertion encoding" — if you write down the stages of the insertion encodings of every permutation in the class, and simplify them in certain ways, you end up with a finite set.

(Finding Regular Insertion Encodings for Permutation Classes, Vatter, 2012)

Example: Av(132, 231)

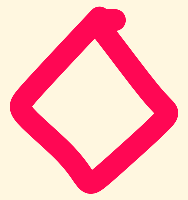
This is the set of permutations made of up a decreasing sequence followed by an increasing sequence



875213469

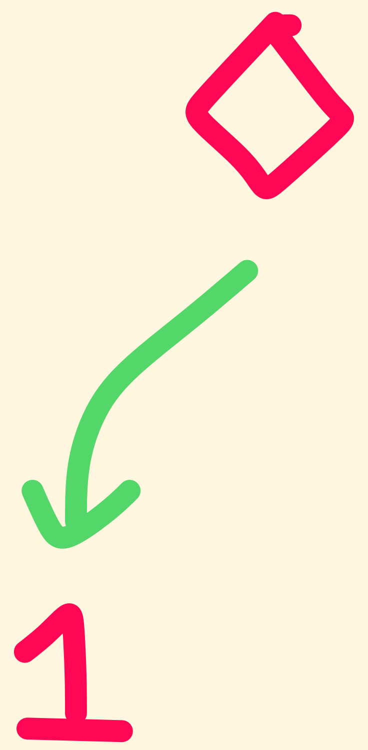
Insertion Encoding

Av(132, 231)



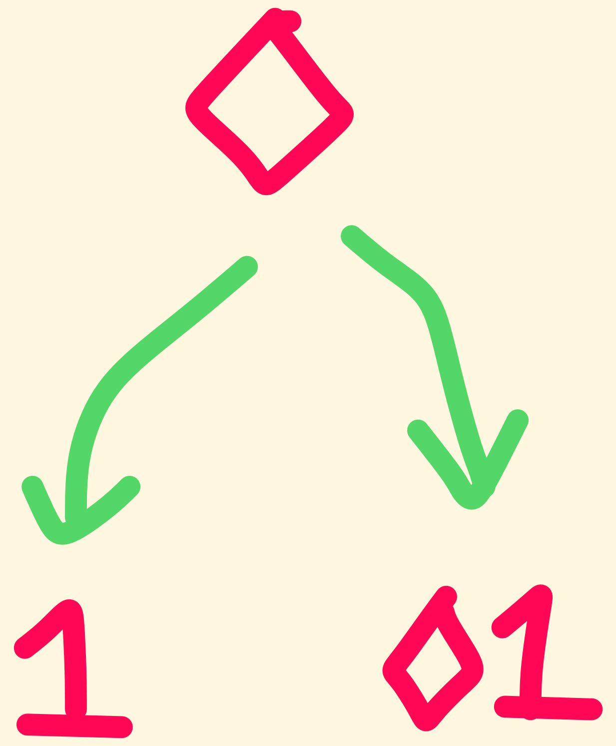
Insertion Encoding

Av(132, 231)



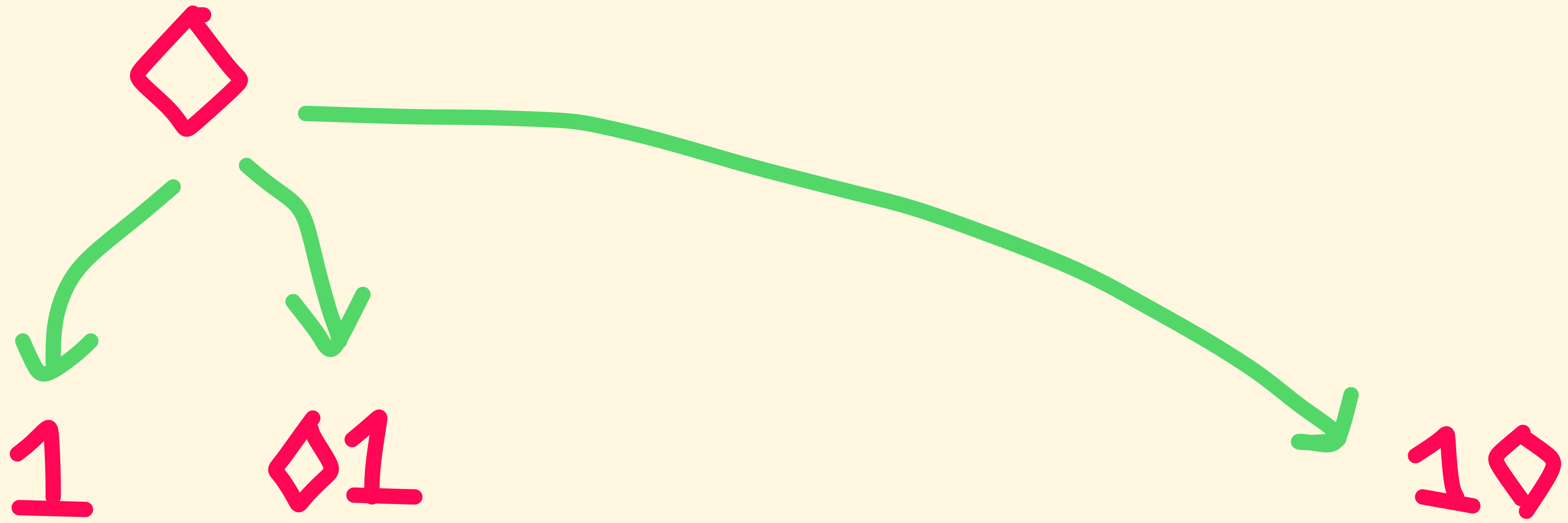
Insertion Encoding

Av(132, 231)



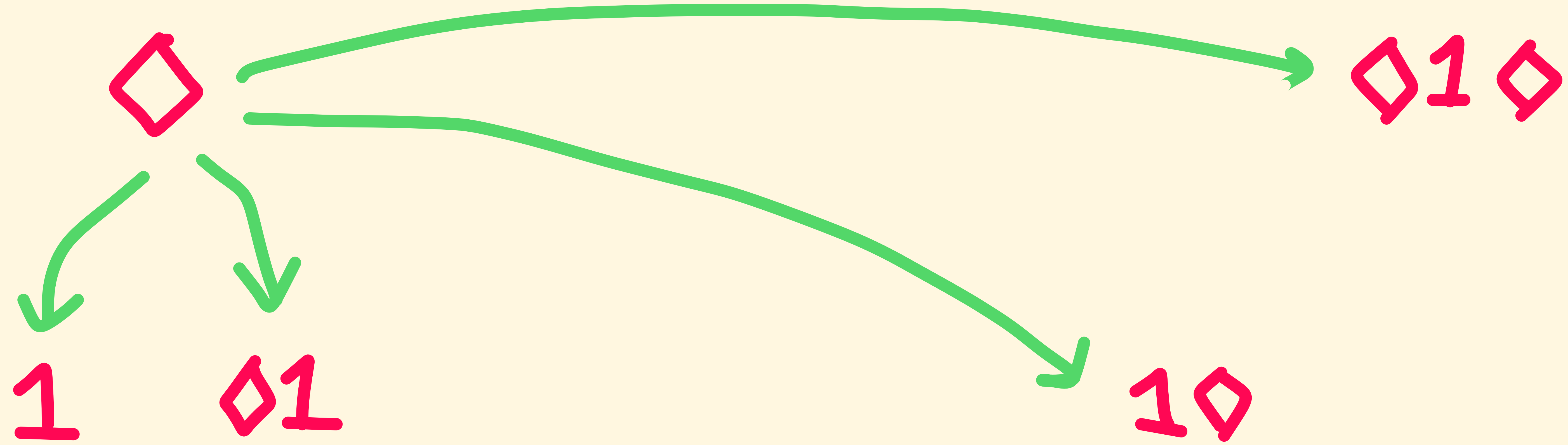
Insertion Encoding

Av(132, 231)



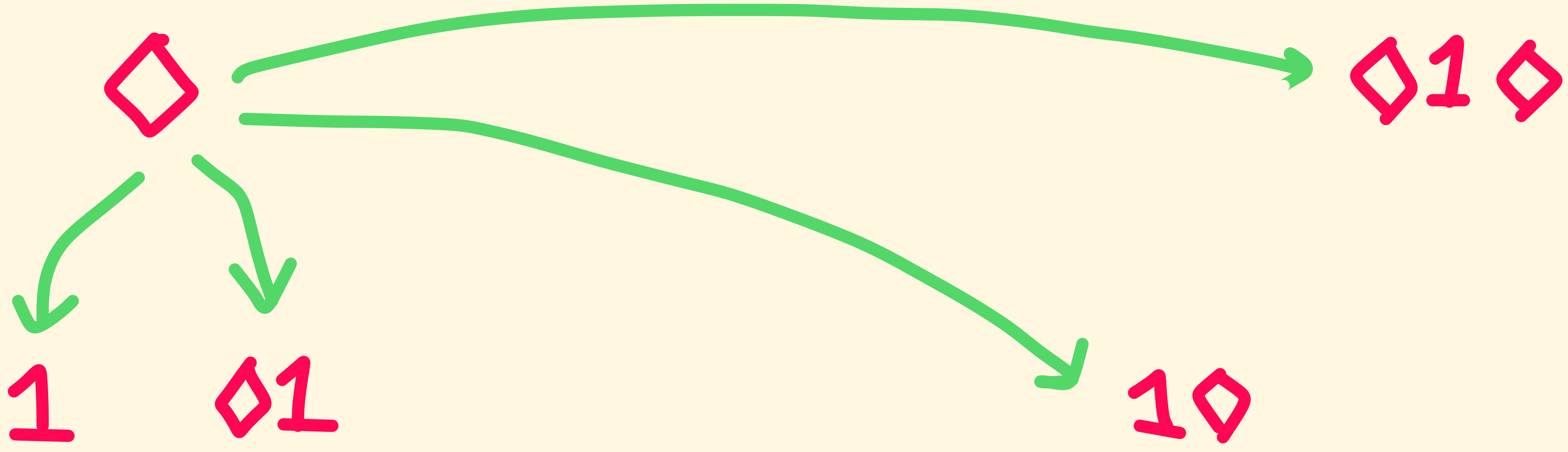
Insertion Encoding

Av(132, 231)



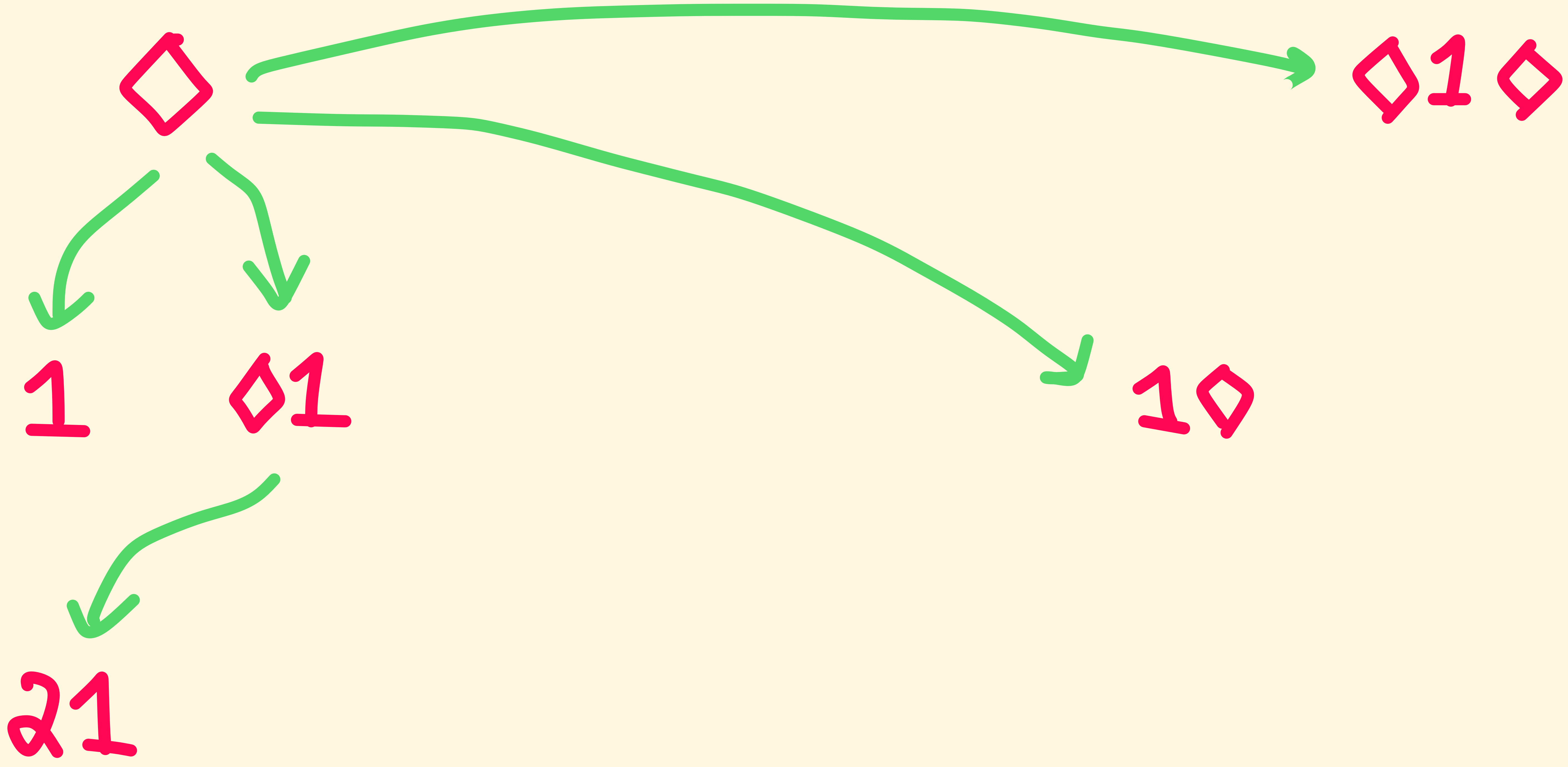
Insertion Encoding

Av(132, 231)



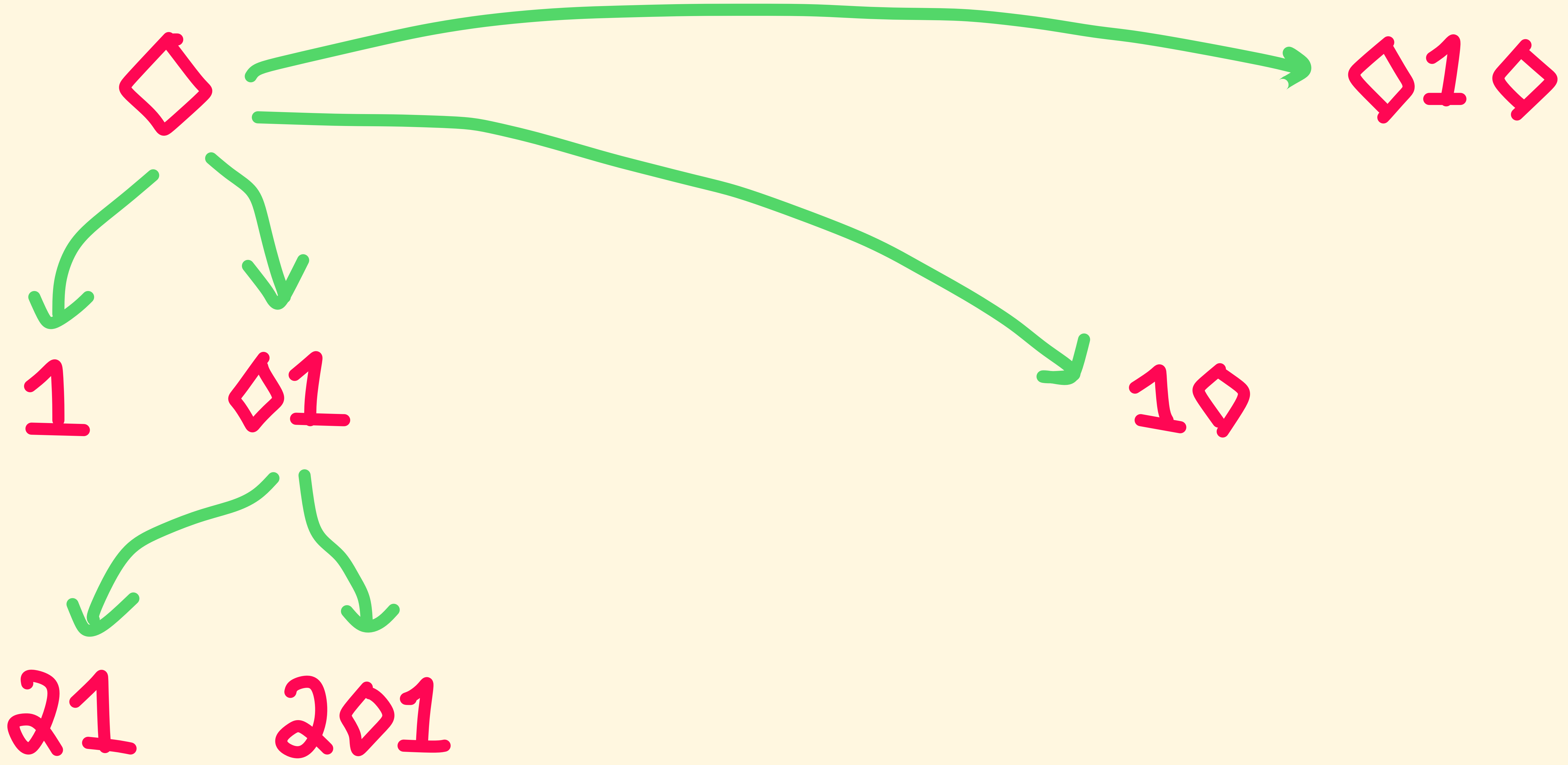
Insertion Encoding

Av(132, 231)



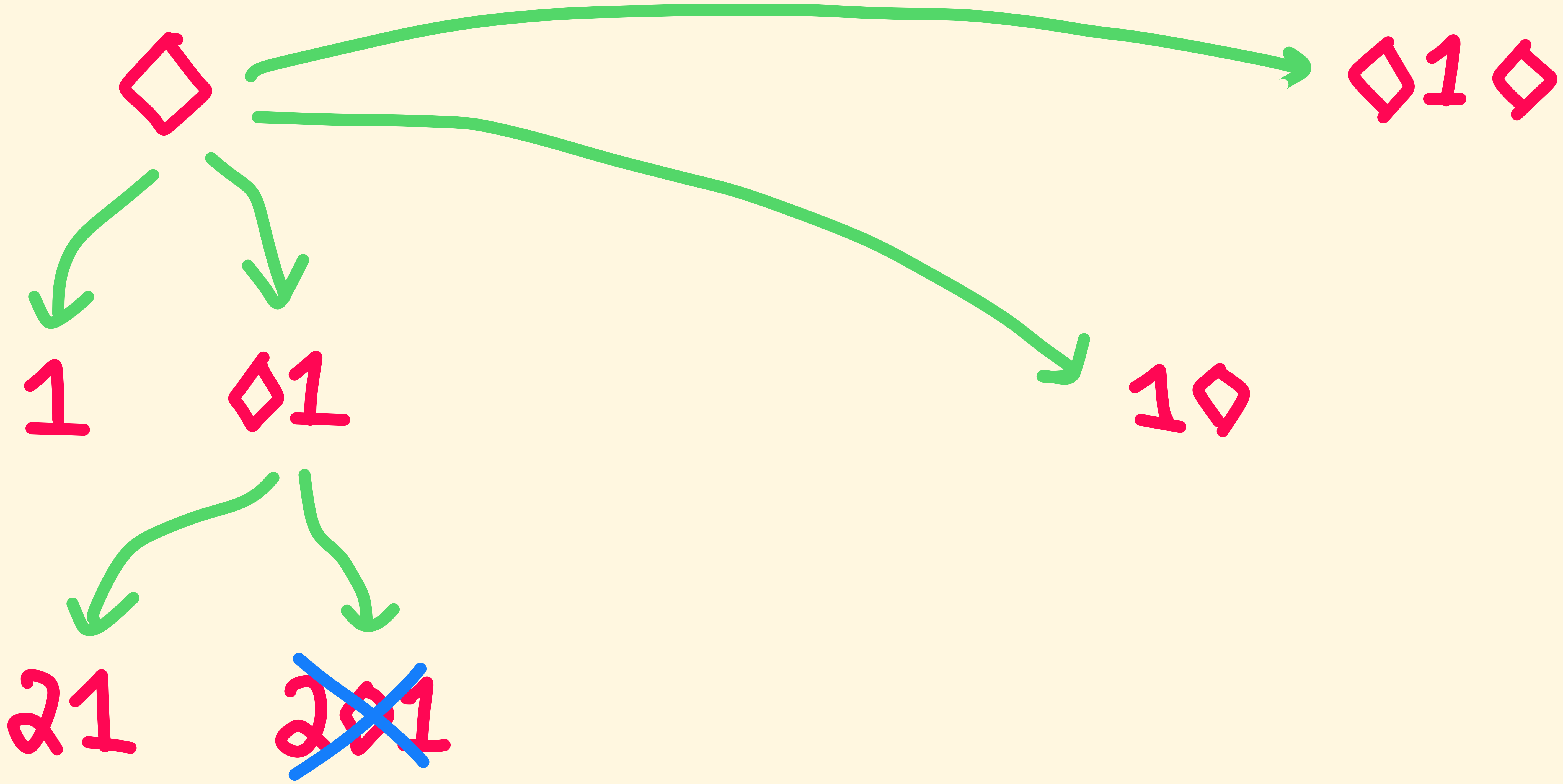
Insertion Encoding

Av(132, 231)



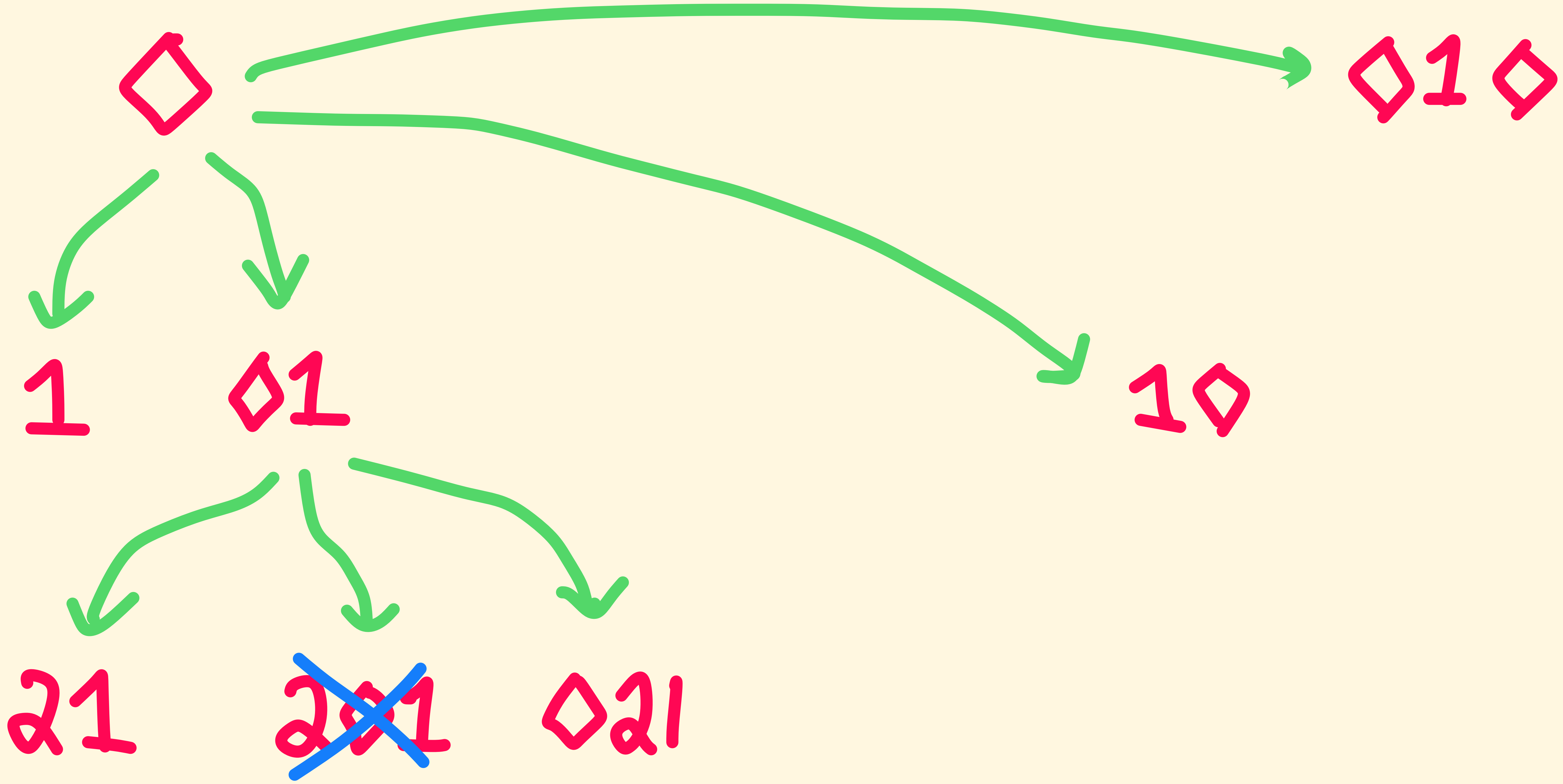
Insertion Encoding

Av(132, 231)



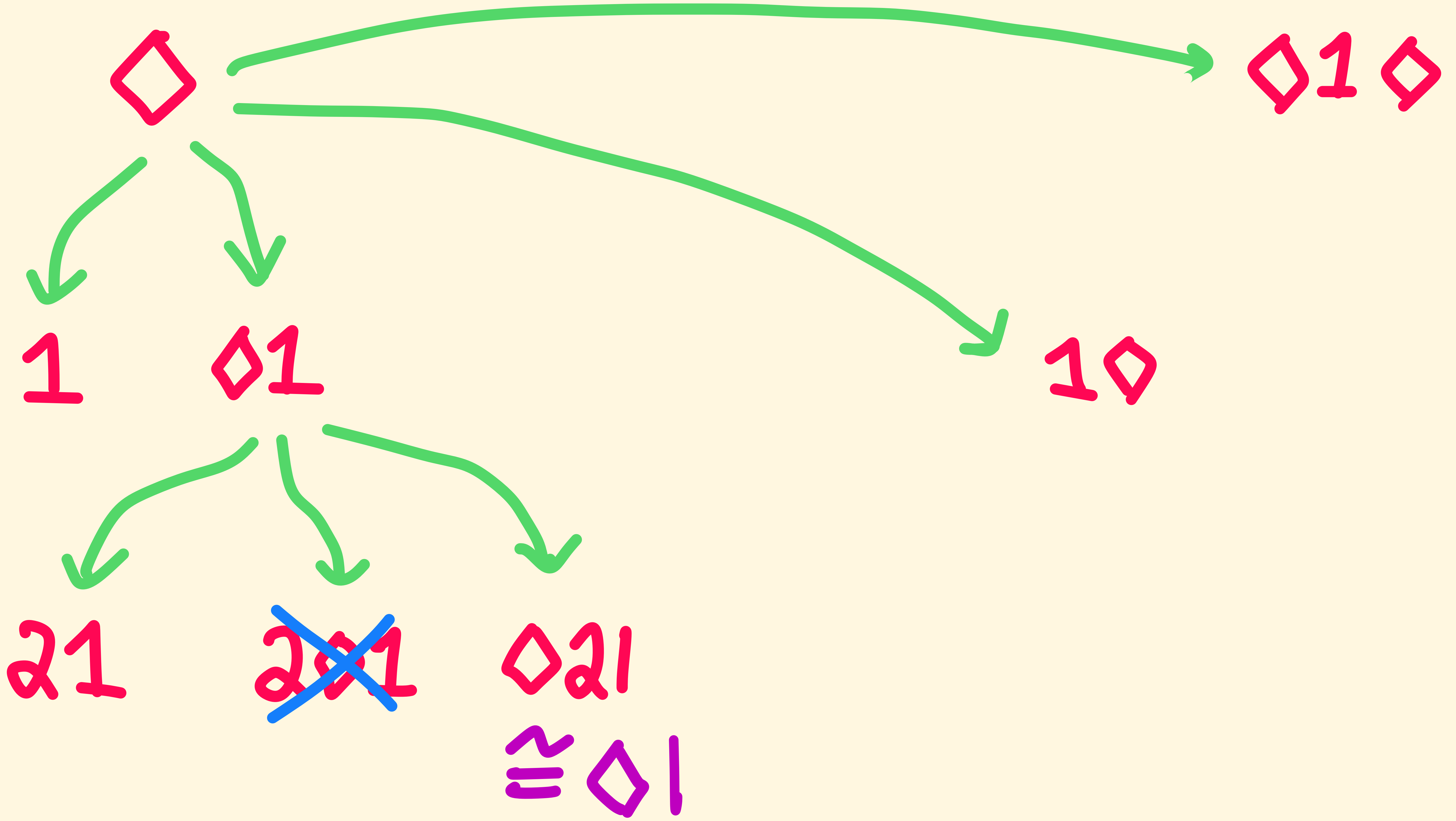
Insertion Encoding

Av(132, 231)



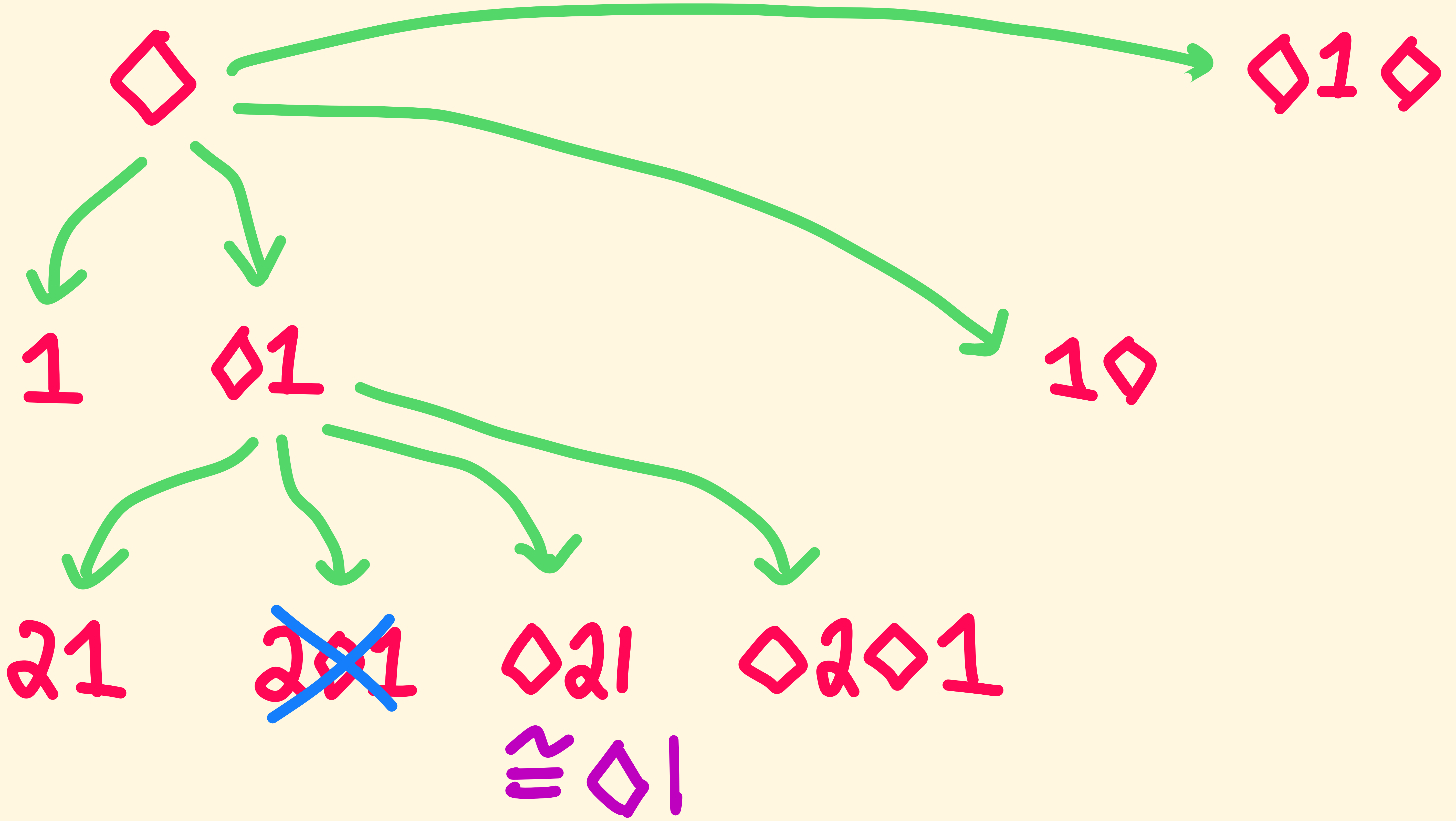
Insertion Encoding

Av(132, 231)



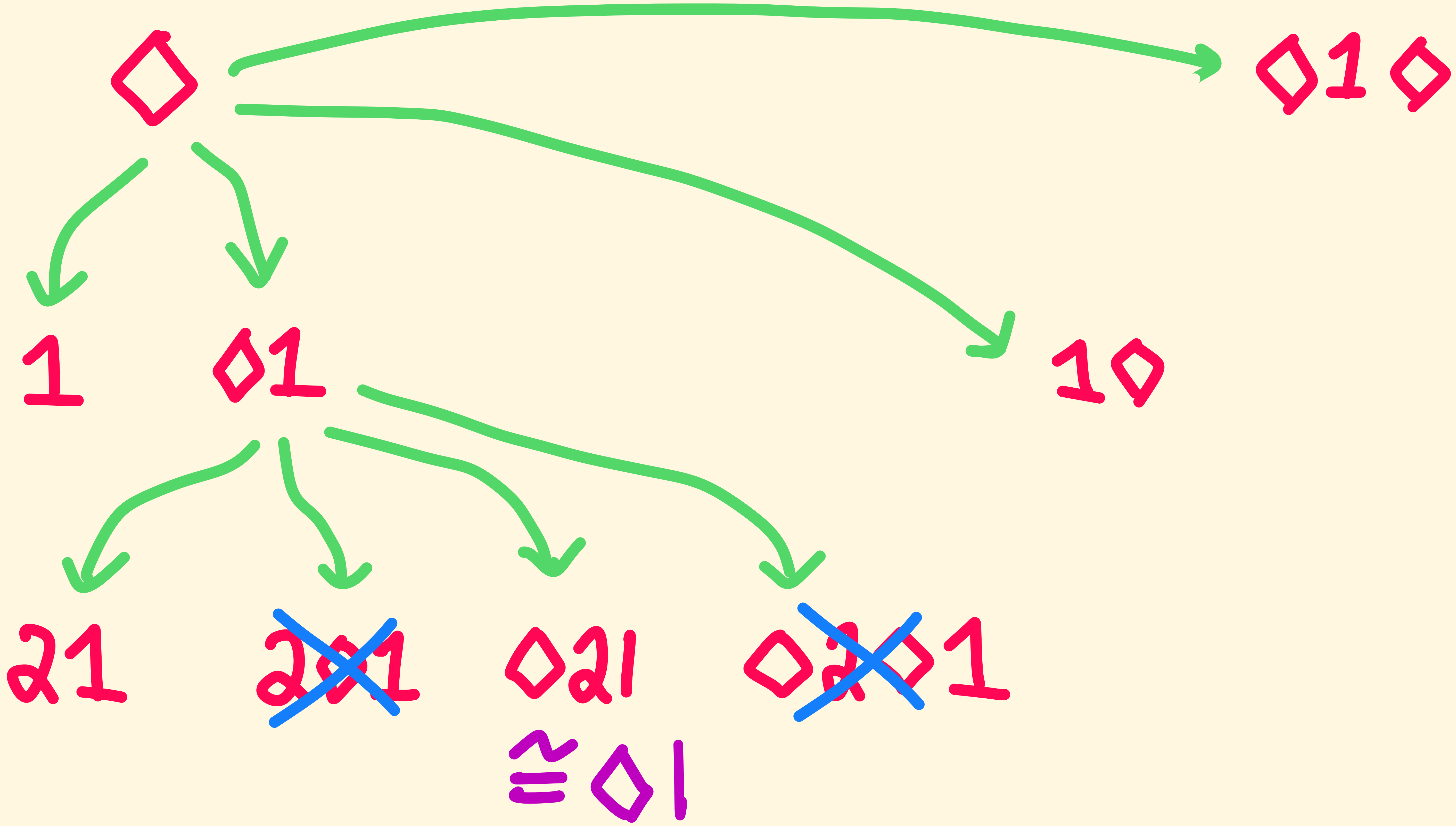
Insertion Encoding

Av(132, 231)



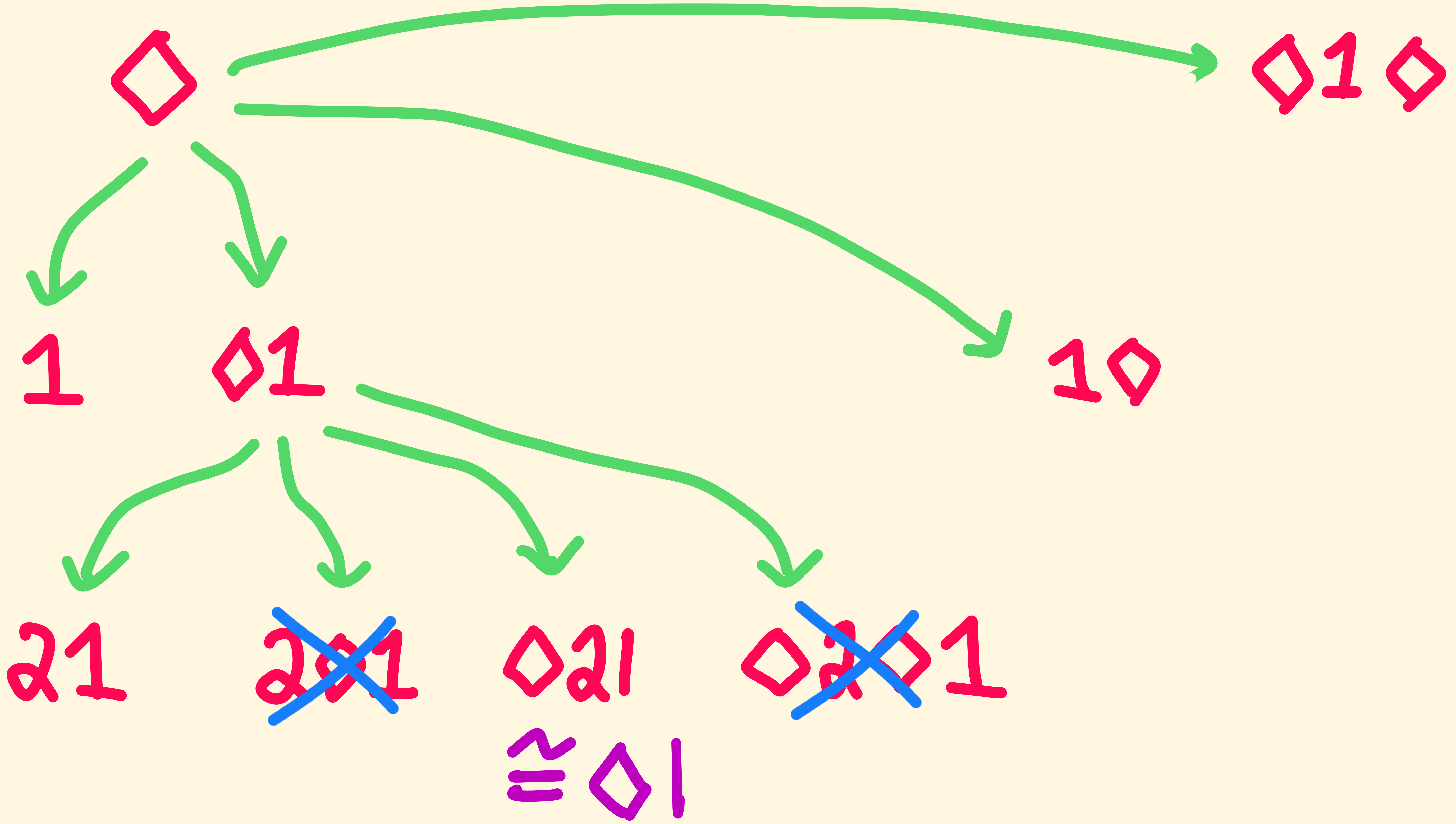
Insertion Encoding

Av(132, 231)



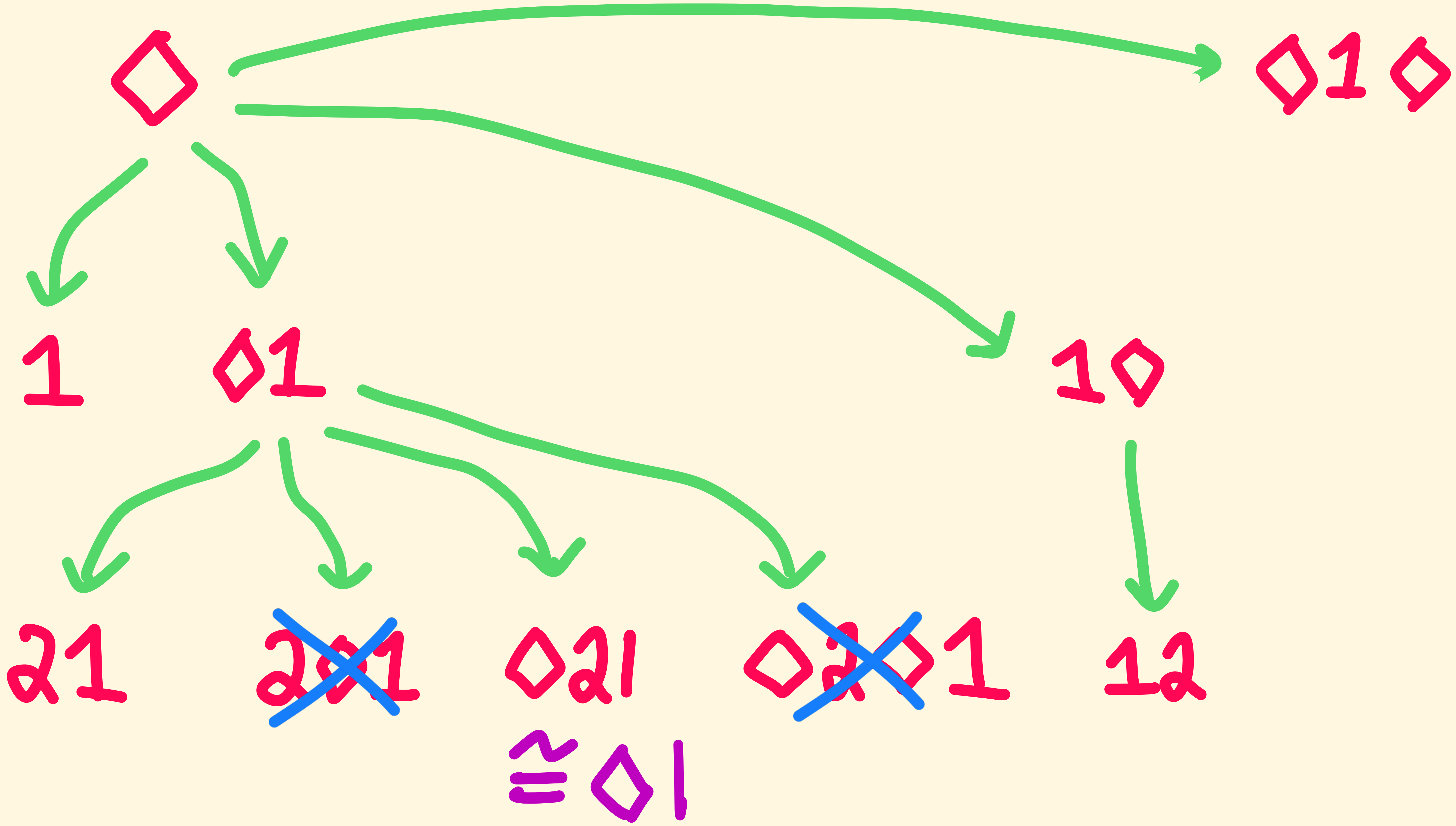
Insertion Encoding

Av(132, 231)



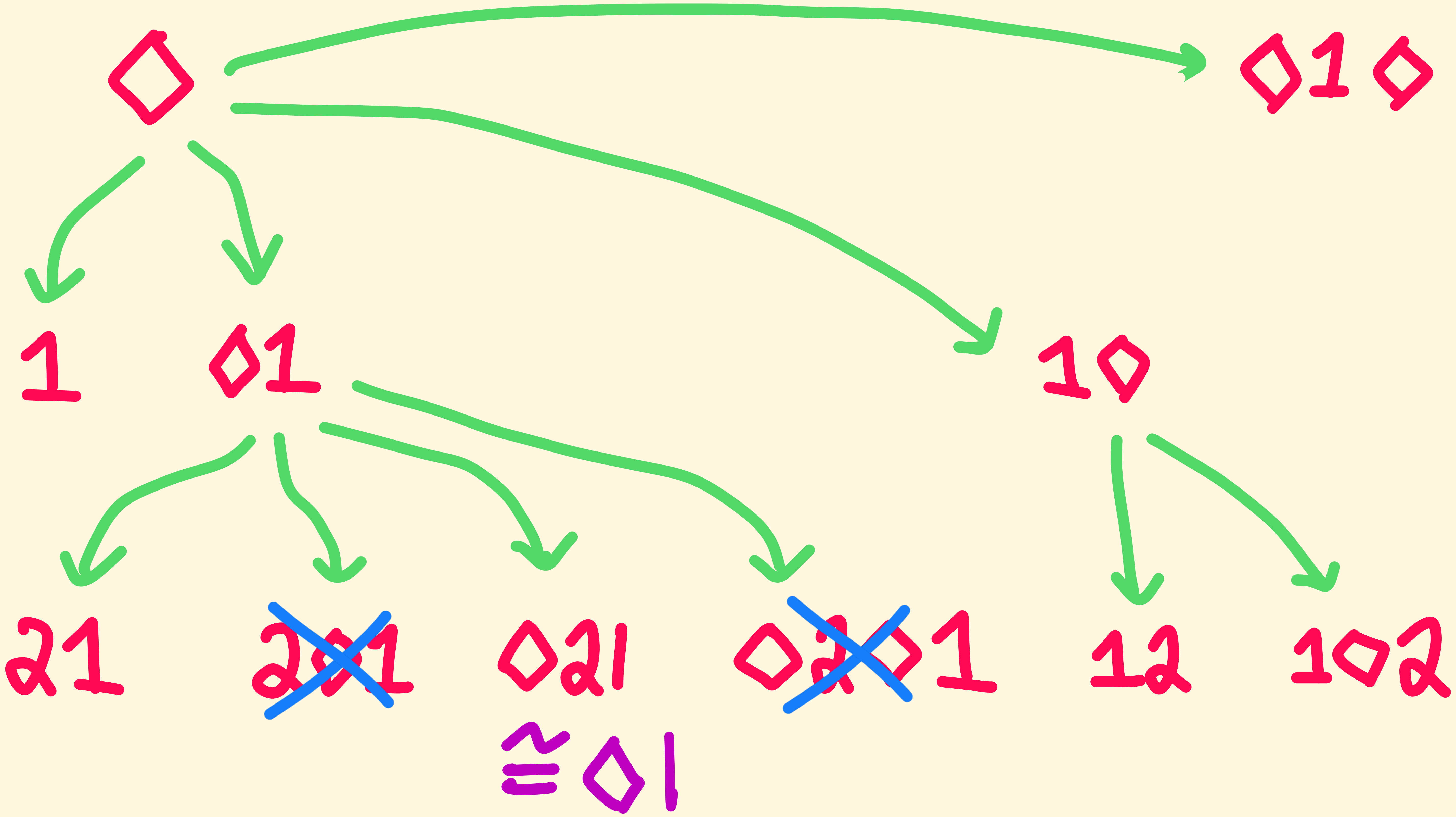
Insertion Encoding

Av(132, 231)



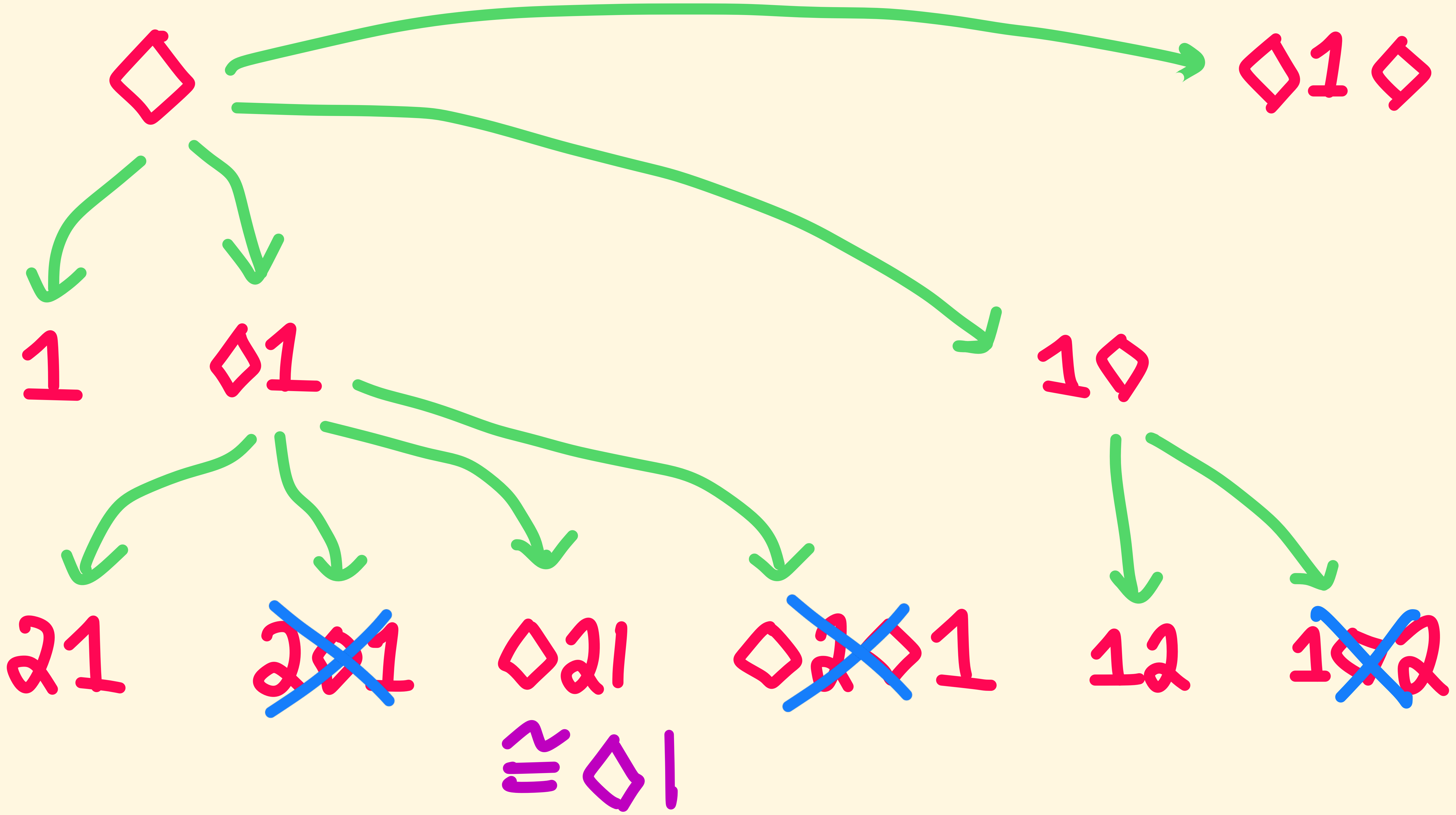
Insertion Encoding

Av(132, 231)



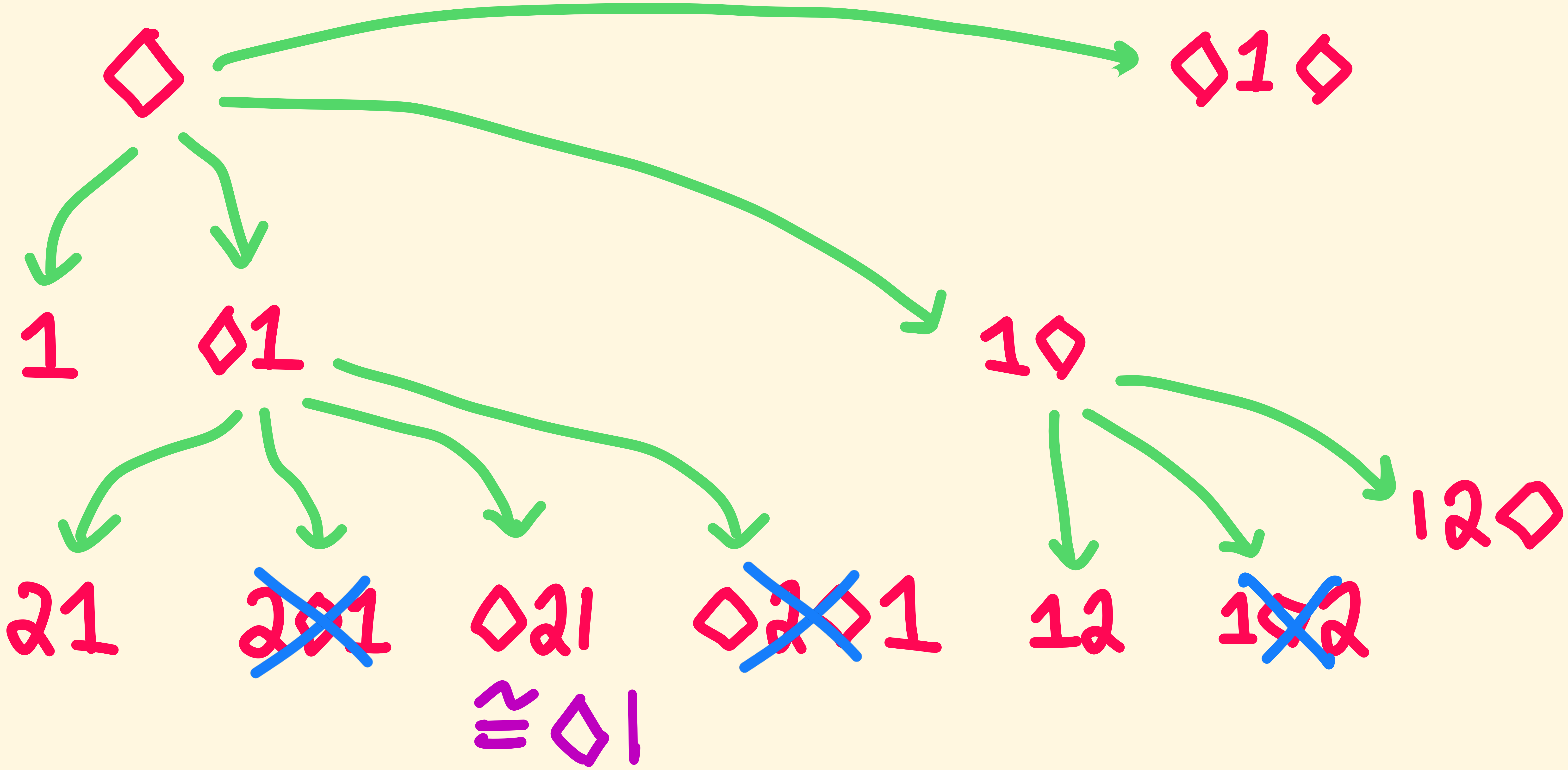
Insertion Encoding

Av(132, 231)



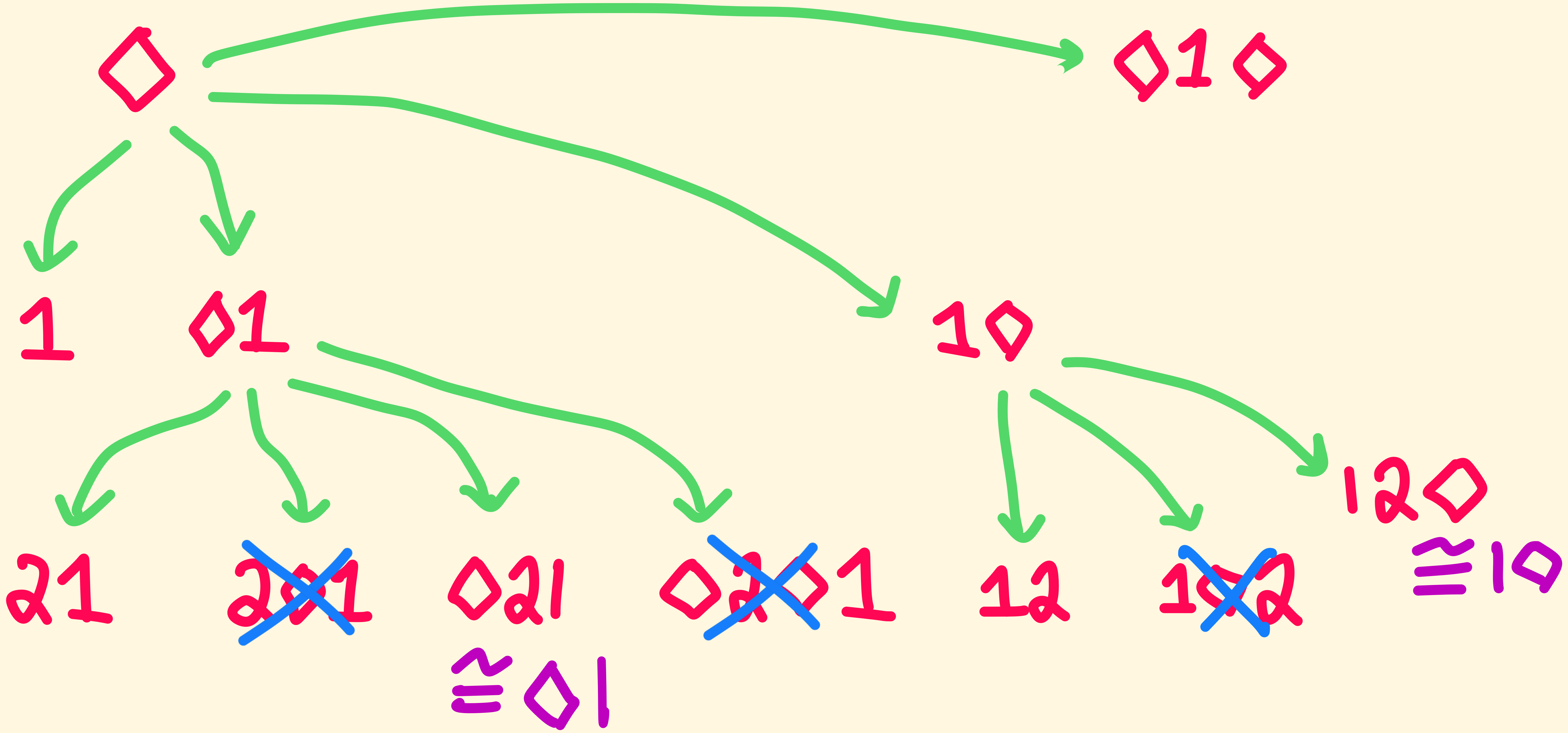
Insertion Encoding

Av(132, 231)



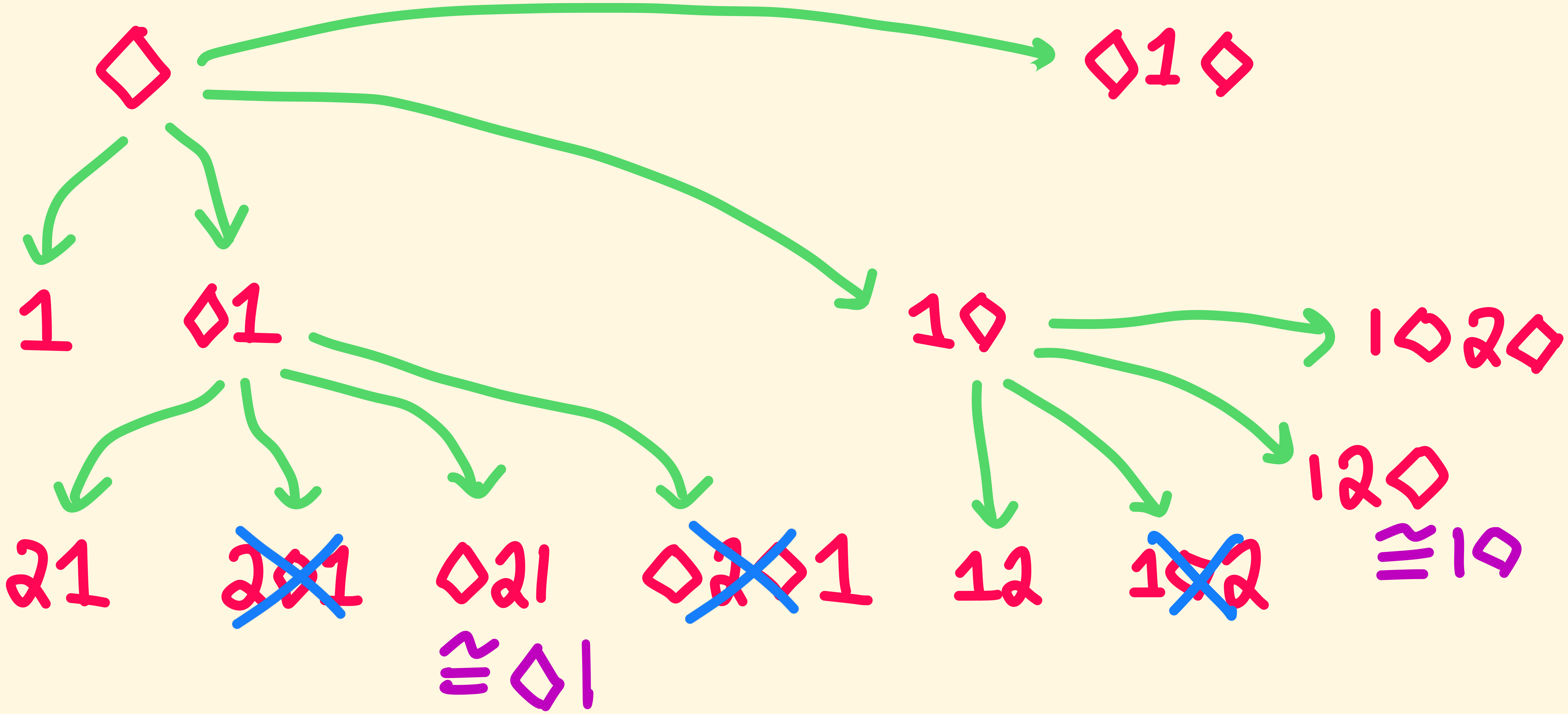
Insertion Encoding

Av(132, 231)



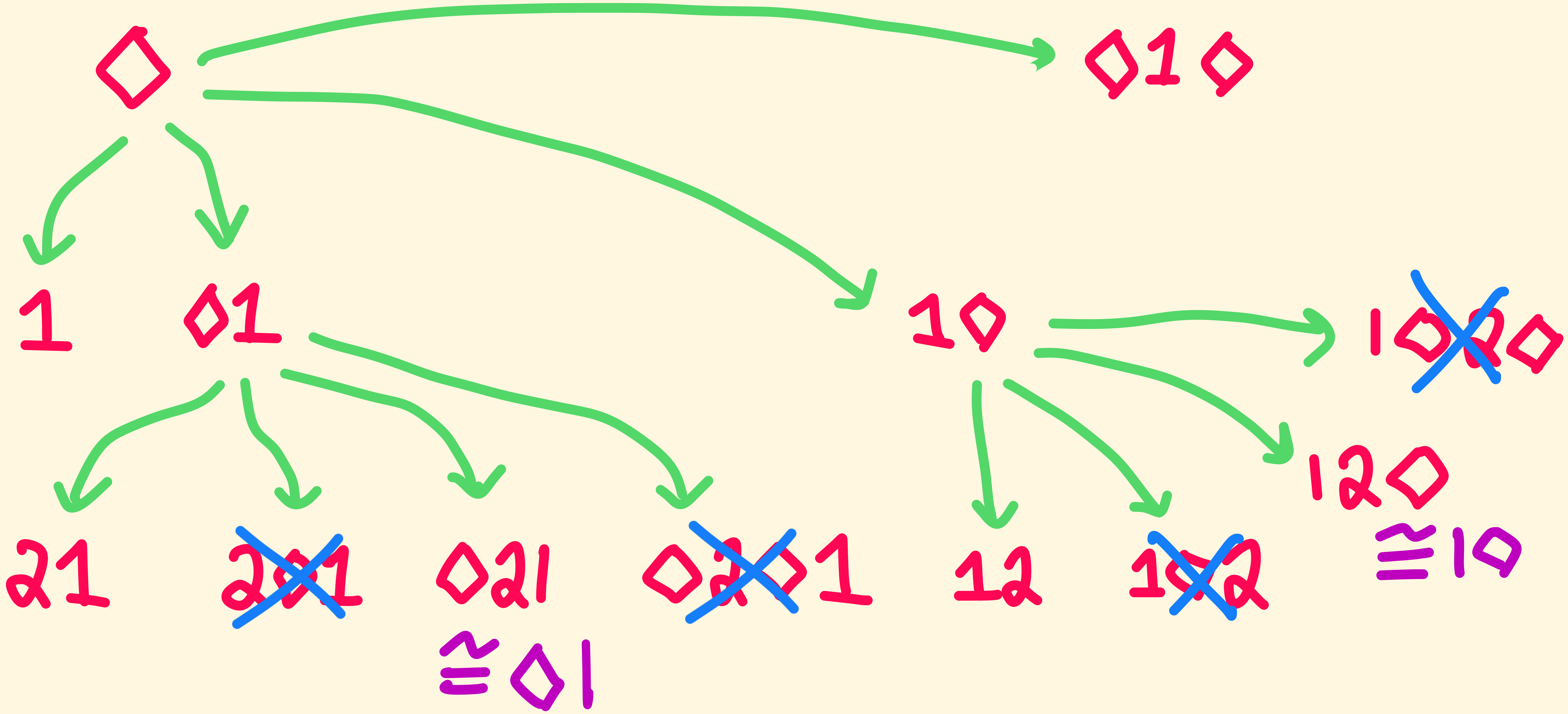
Insertion Encoding

Av(132, 231)



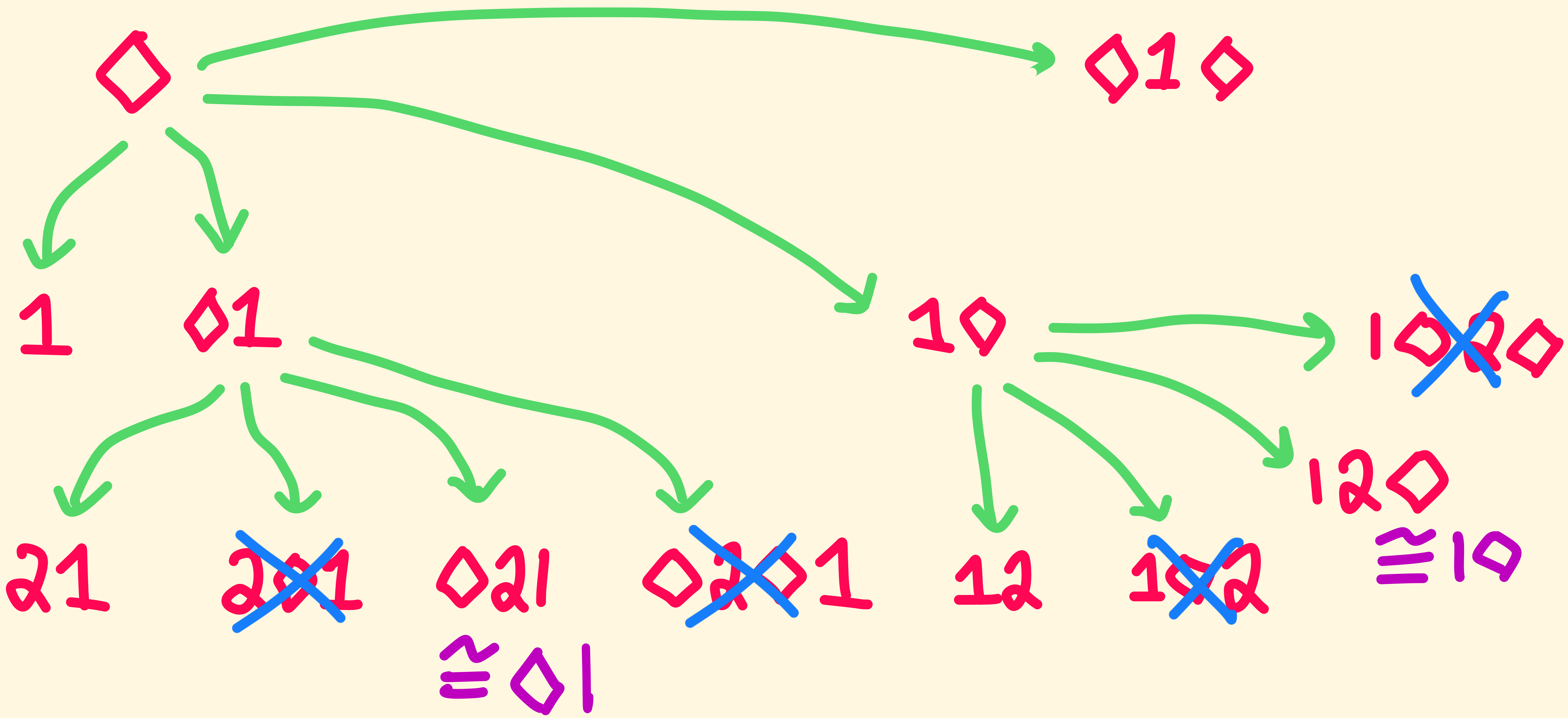
Insertion Encoding

Av(132, 231)



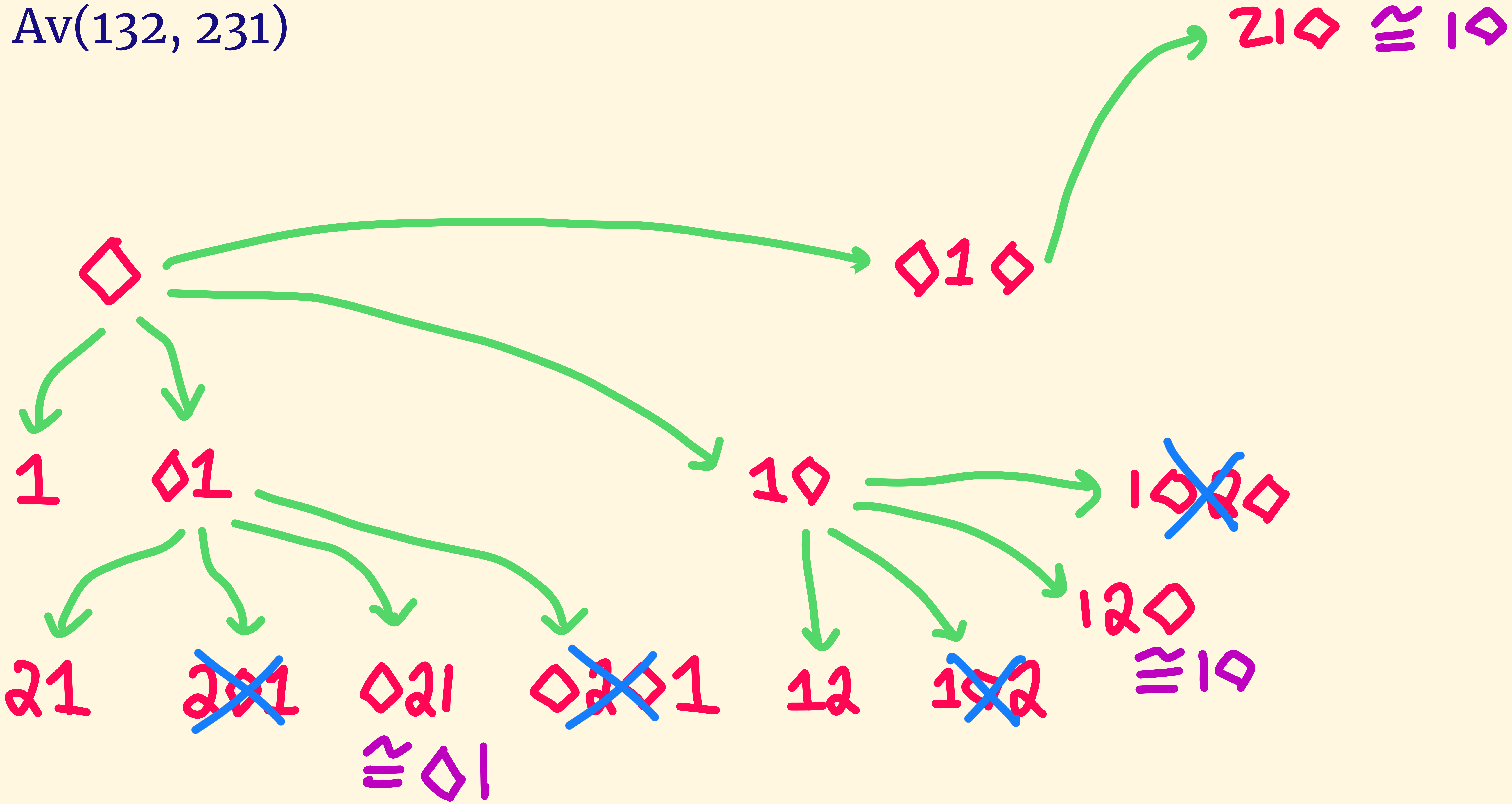
Insertion Encoding

Av(132, 231)



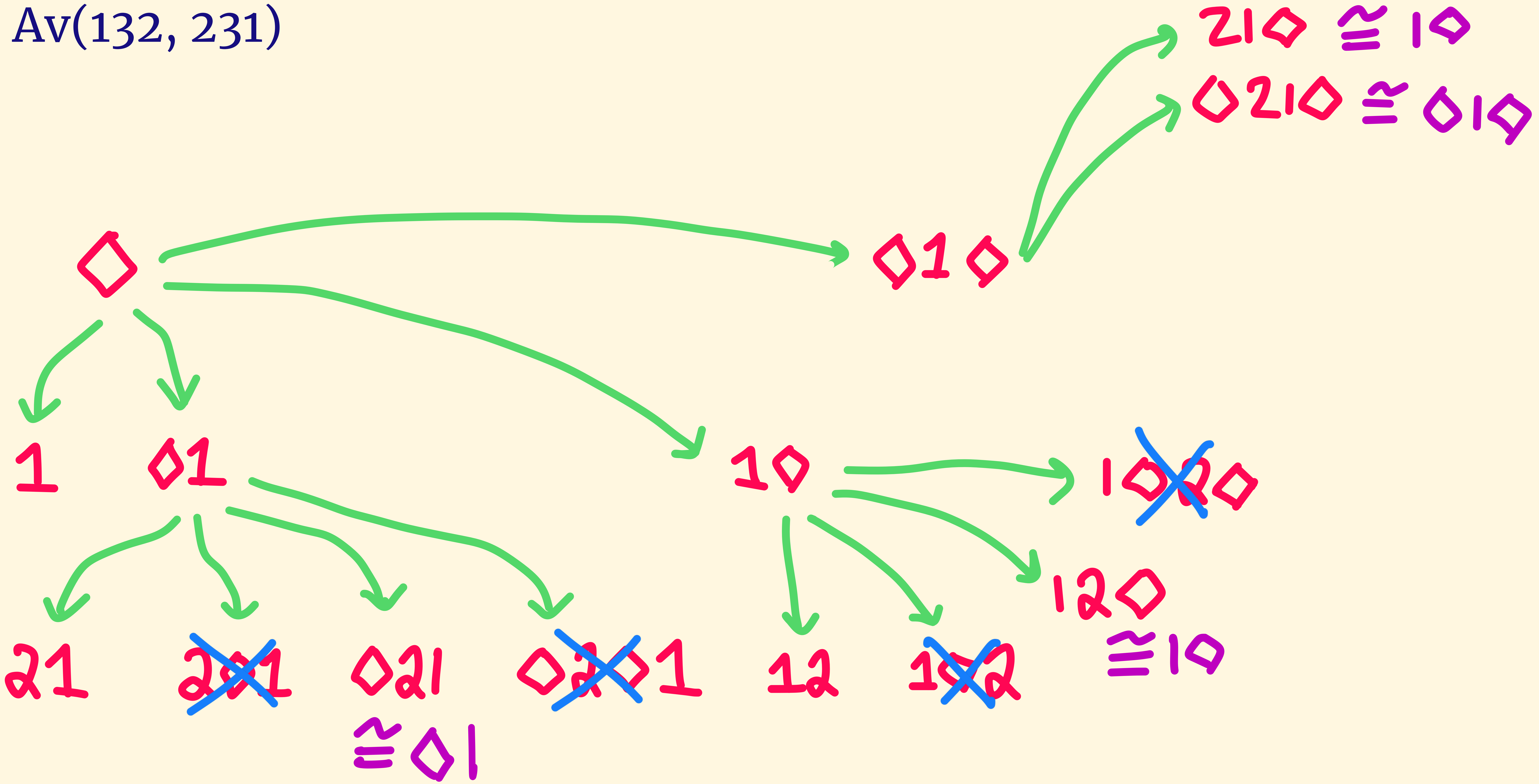
Insertion Encoding

Av(132, 231)



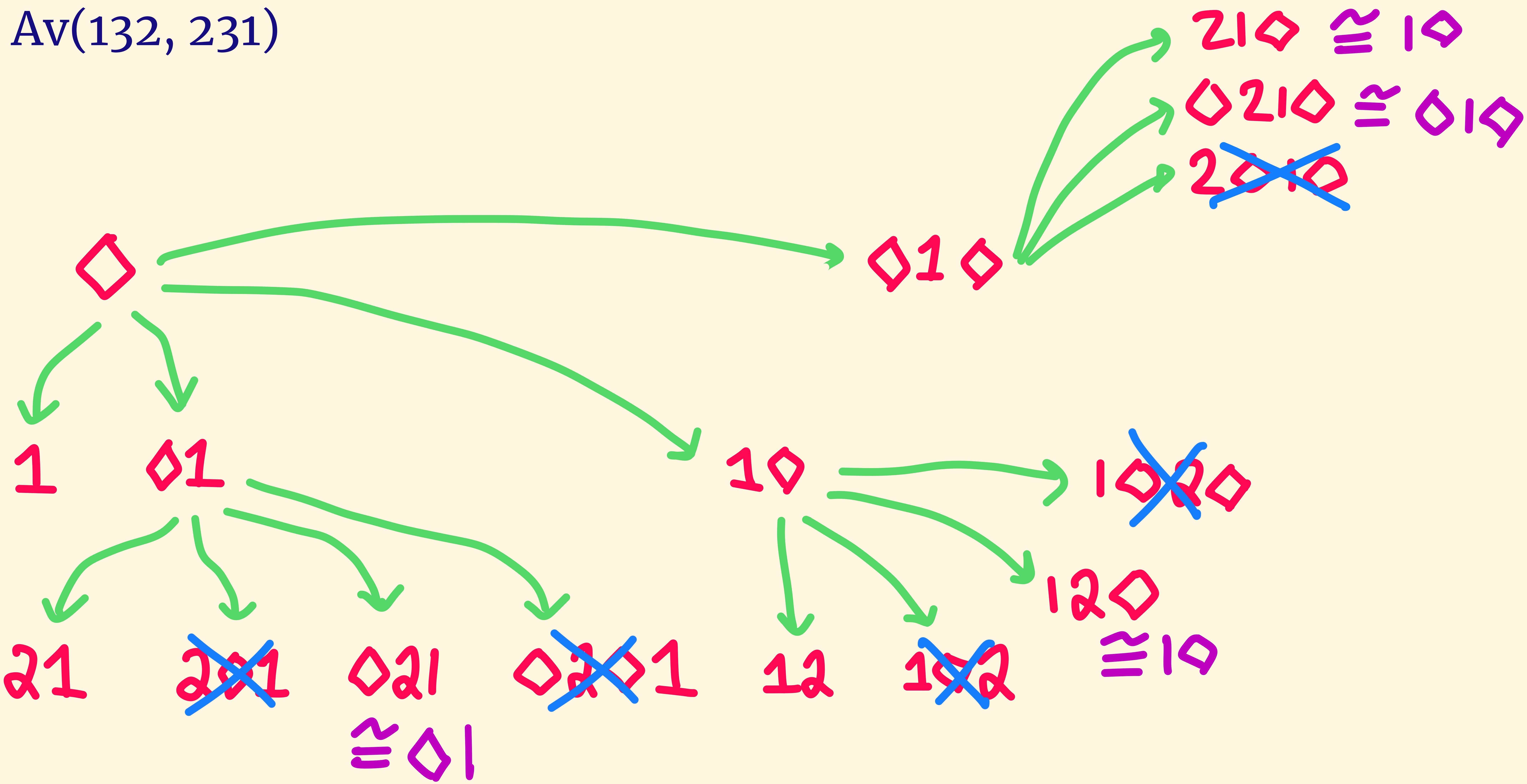
Insertion Encoding

Av(132, 231)



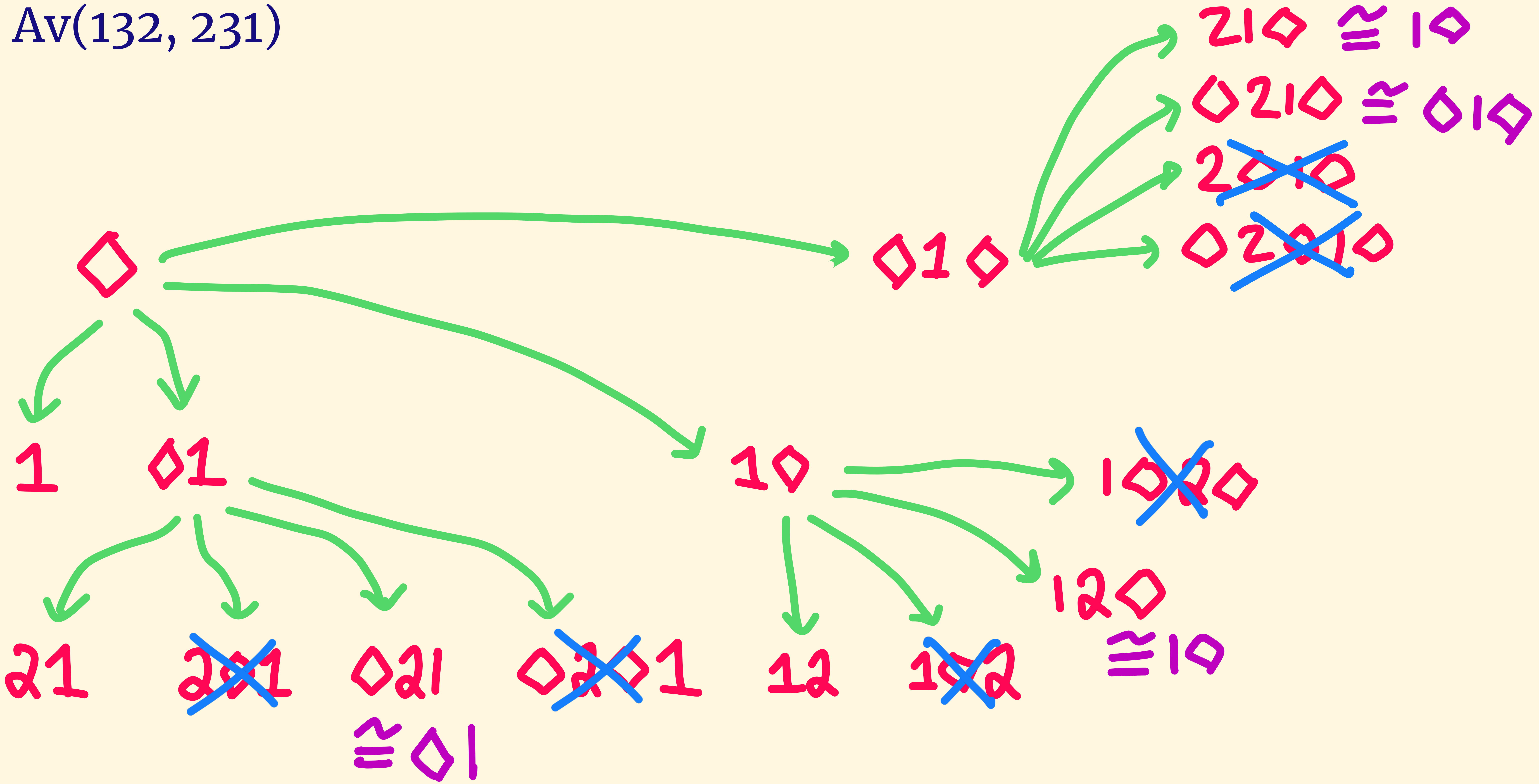
Insertion Encoding

Av(132, 231)



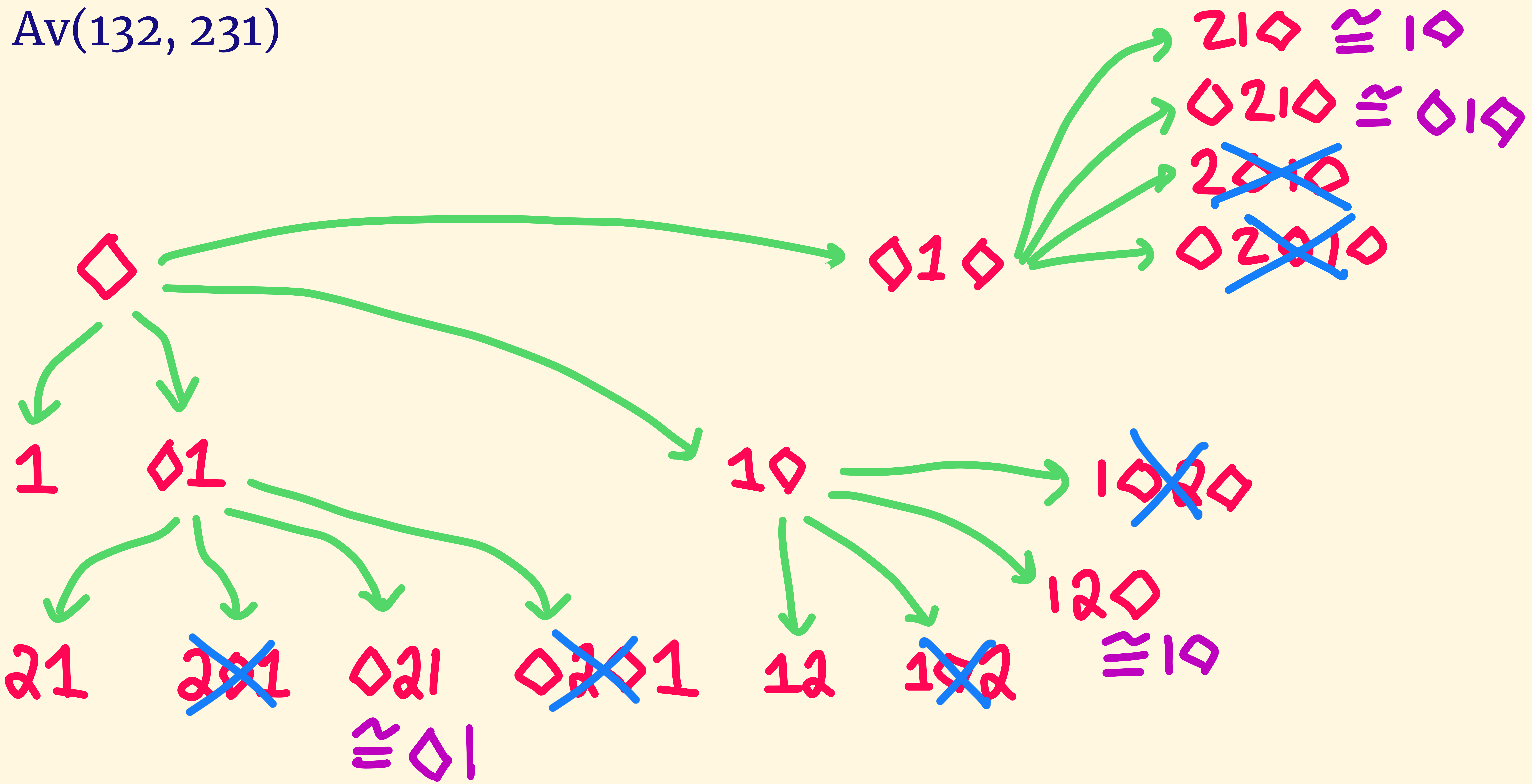
Insertion Encoding

Av(132, 231)



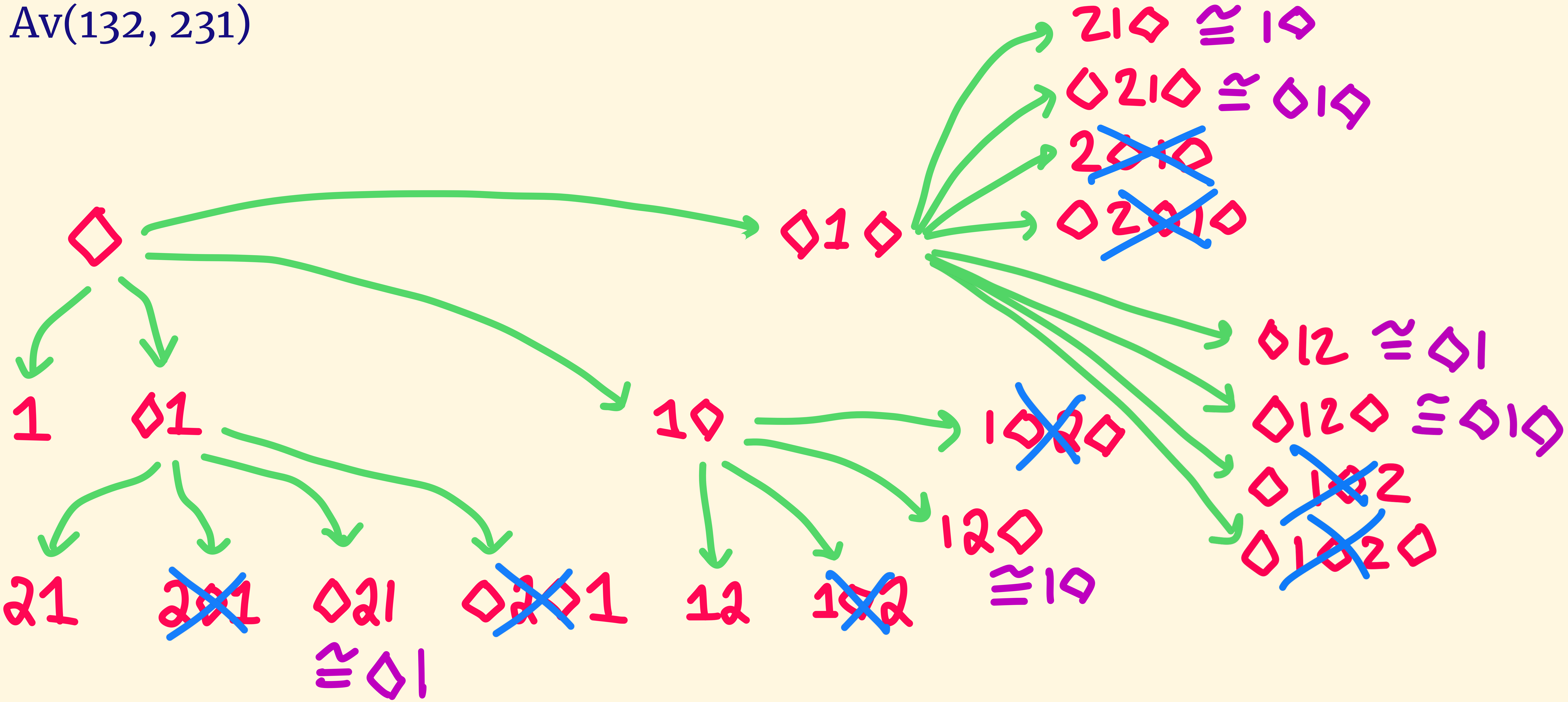
Insertion Encoding

Av(132, 231)



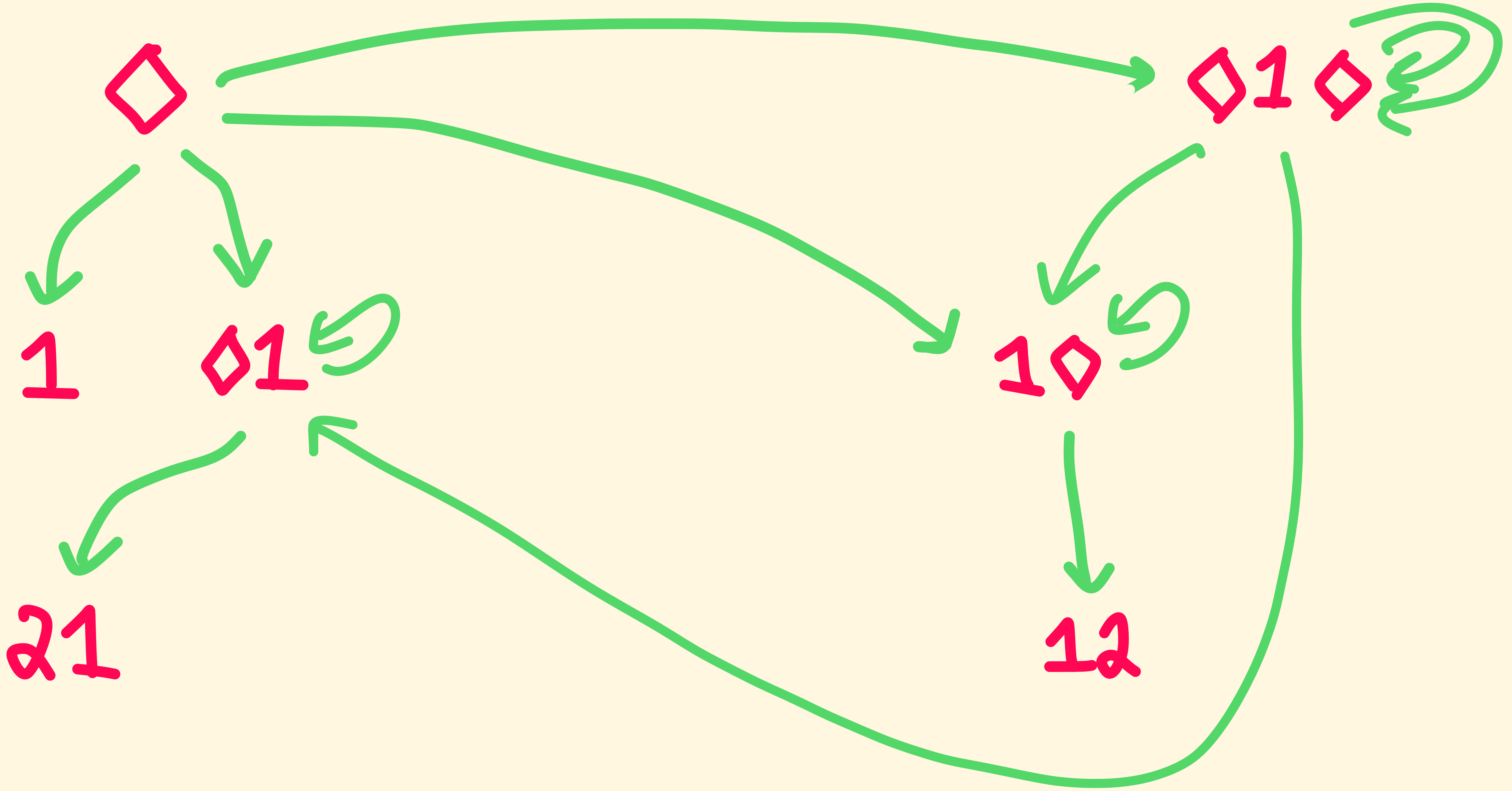
Insertion Encoding

Av(132, 231)



Insertion Encoding

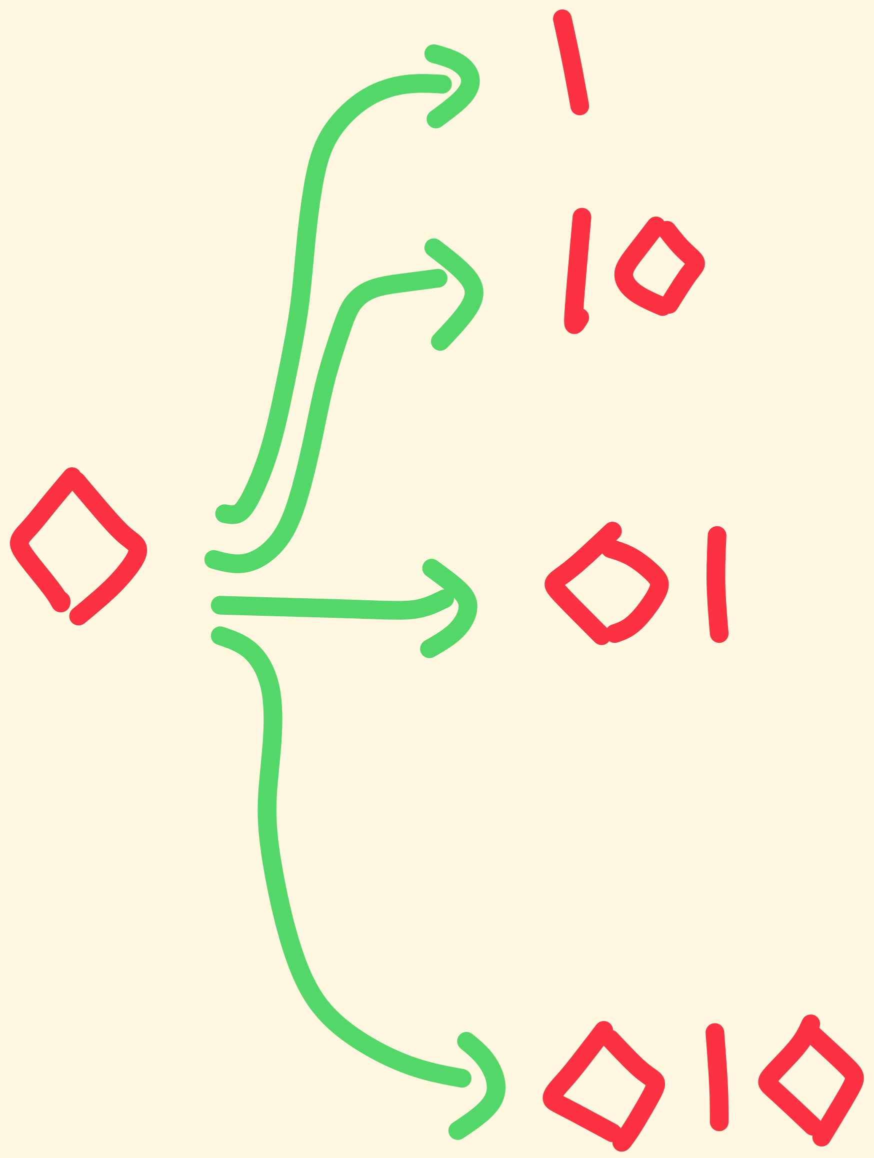
Av(132, 231)



Insertion Encoding

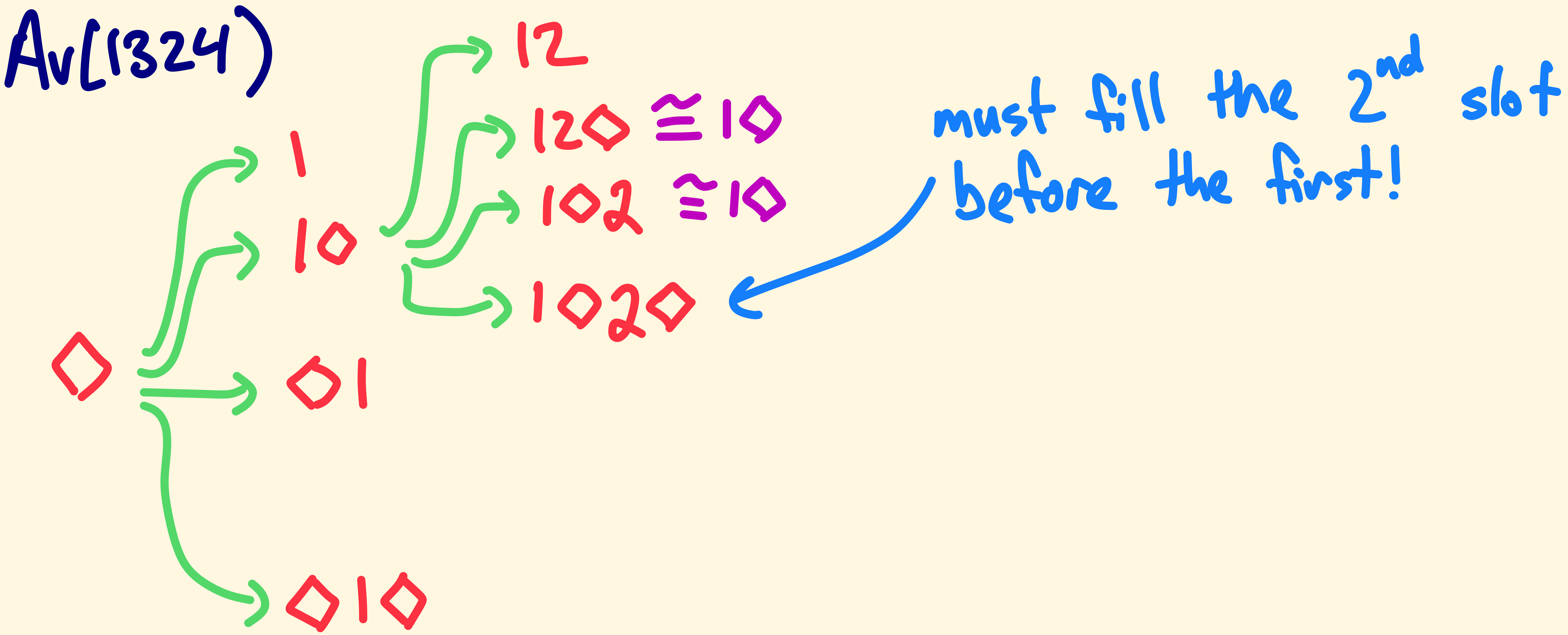
Even if a class has an infinite insertion encoding, you can still use it to count the number of permutations in a class up to some point.

Av(1324)



Insertion Encoding

Even if a class has an infinite insertion encoding, you can still use it to count the number of permutations in a class up to some point.



Insertion Encoding

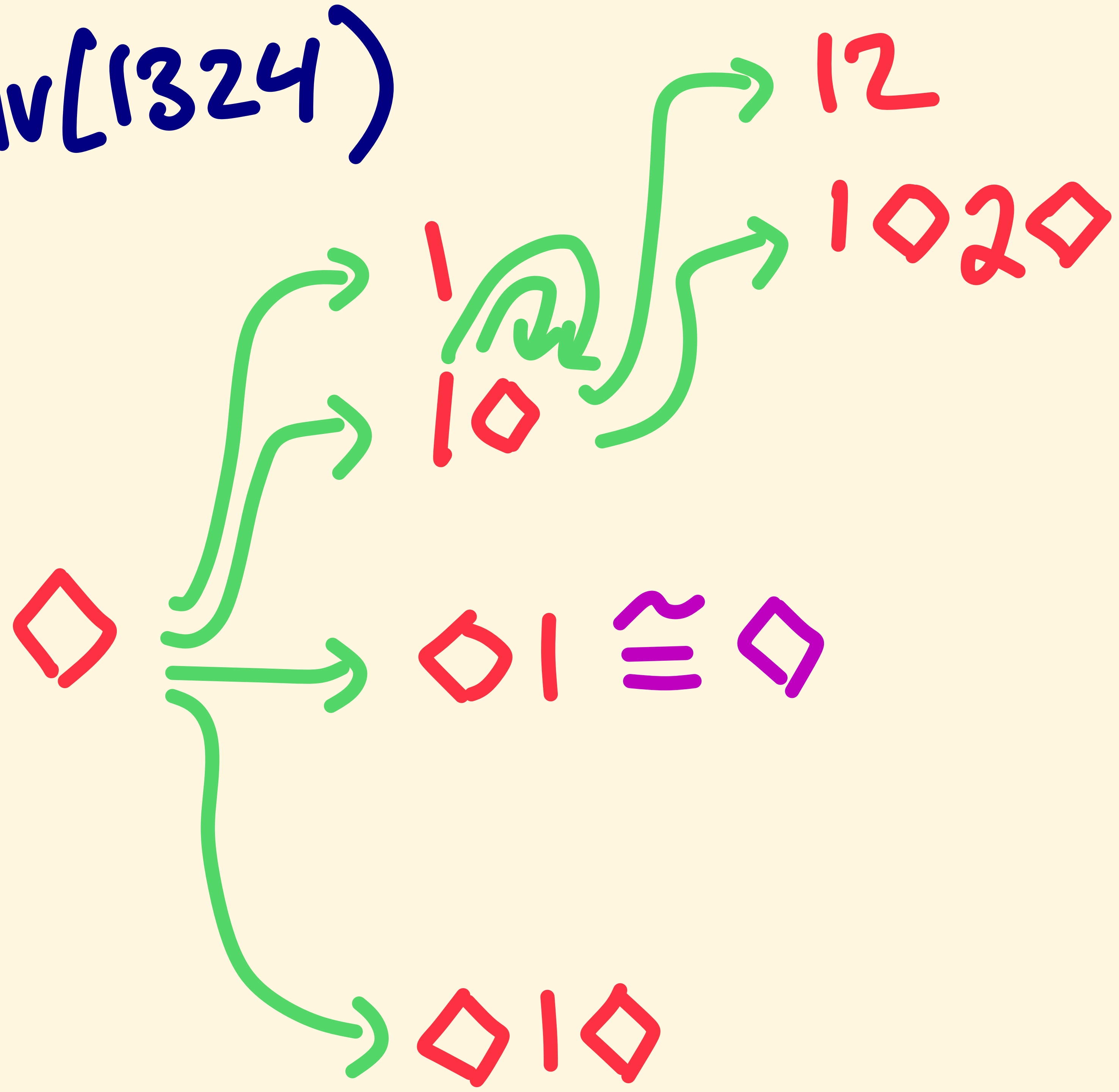
Even if a class has an infinite insertion encoding, you can still use it to count the number of permutations in a class up to some point.

Av(1324)

Insertion Encoding

Even if a class has an infinite insertion encoding, you can still use it to count the number of permutations in a class up to some point.

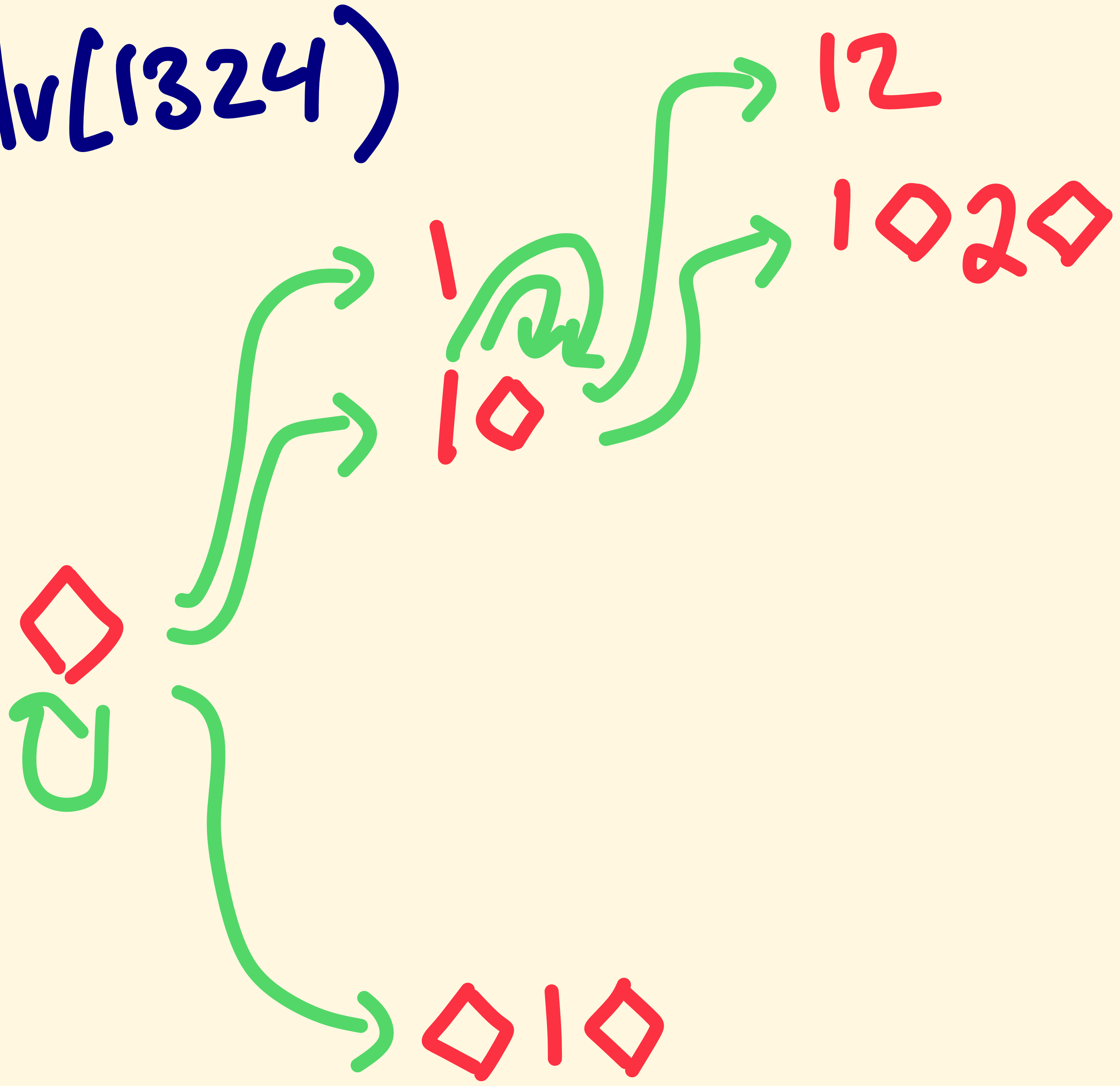
Av(1324)



Insertion Encoding

Even if a class has an infinite insertion encoding, you can still use it to count the number of permutations in a class up to some point.

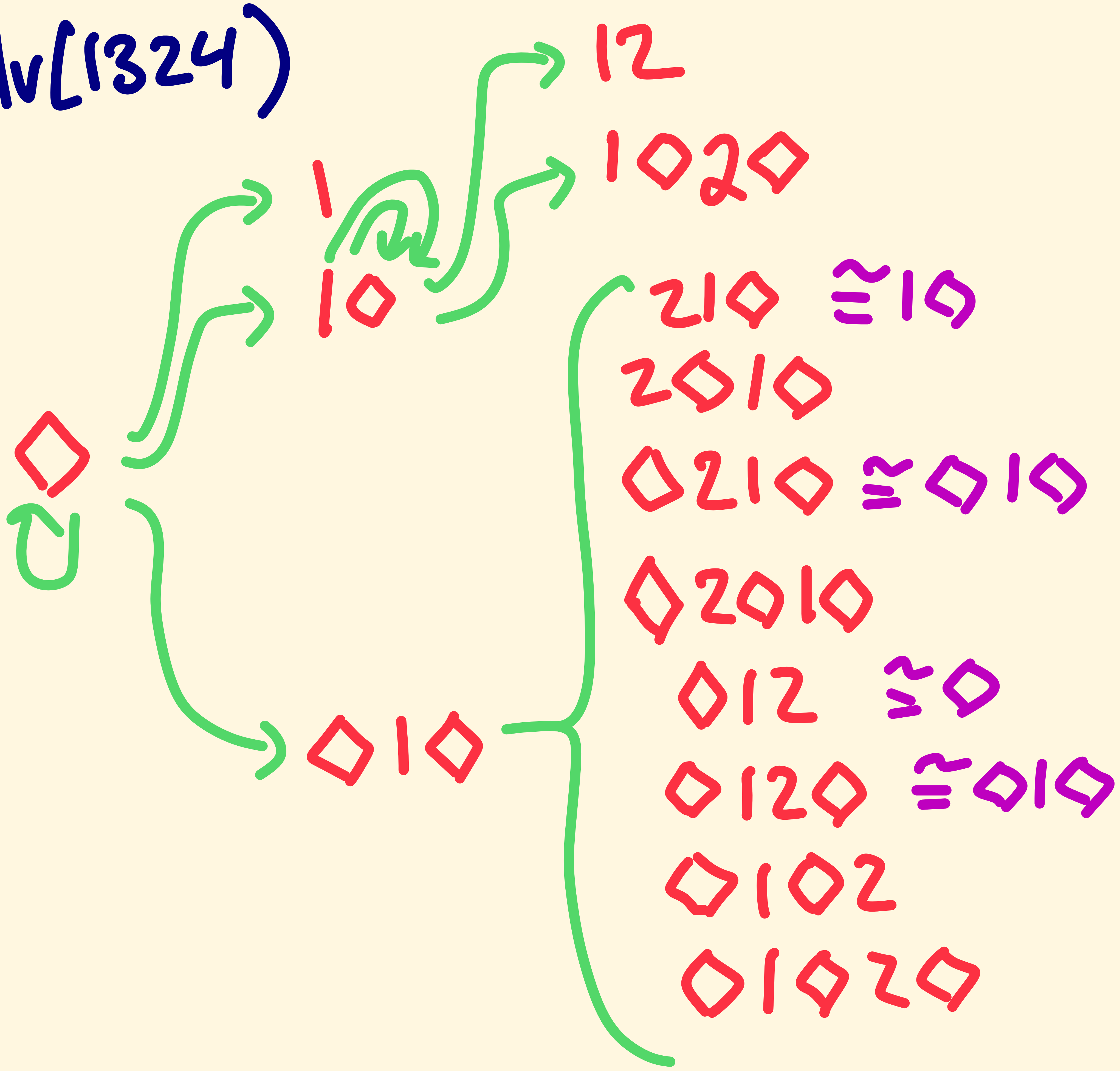
Av(1324)



Insertion Encoding

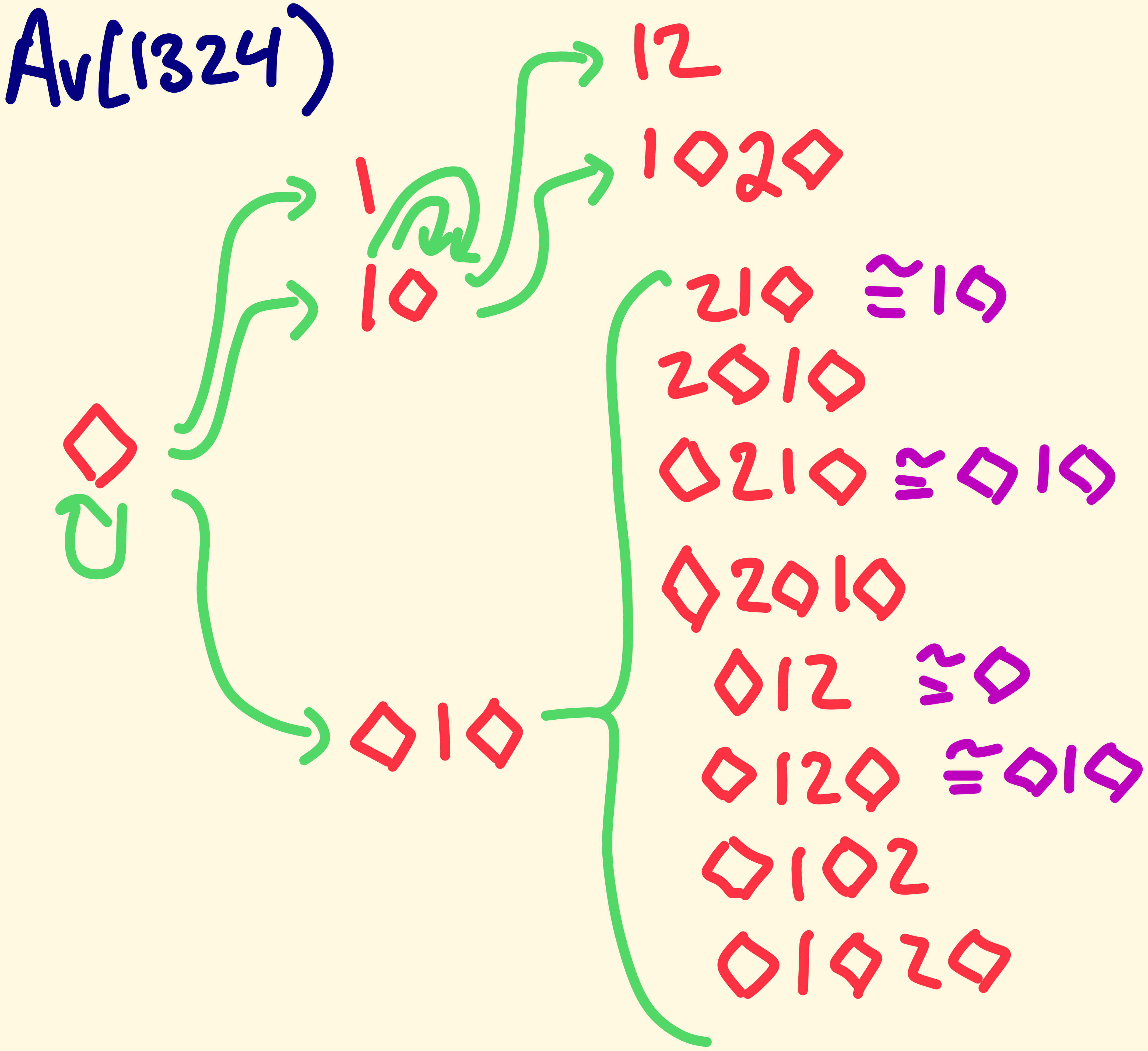
Even if a class has an infinite insertion encoding, you can still use it to count the number of permutations in a class up to some point.

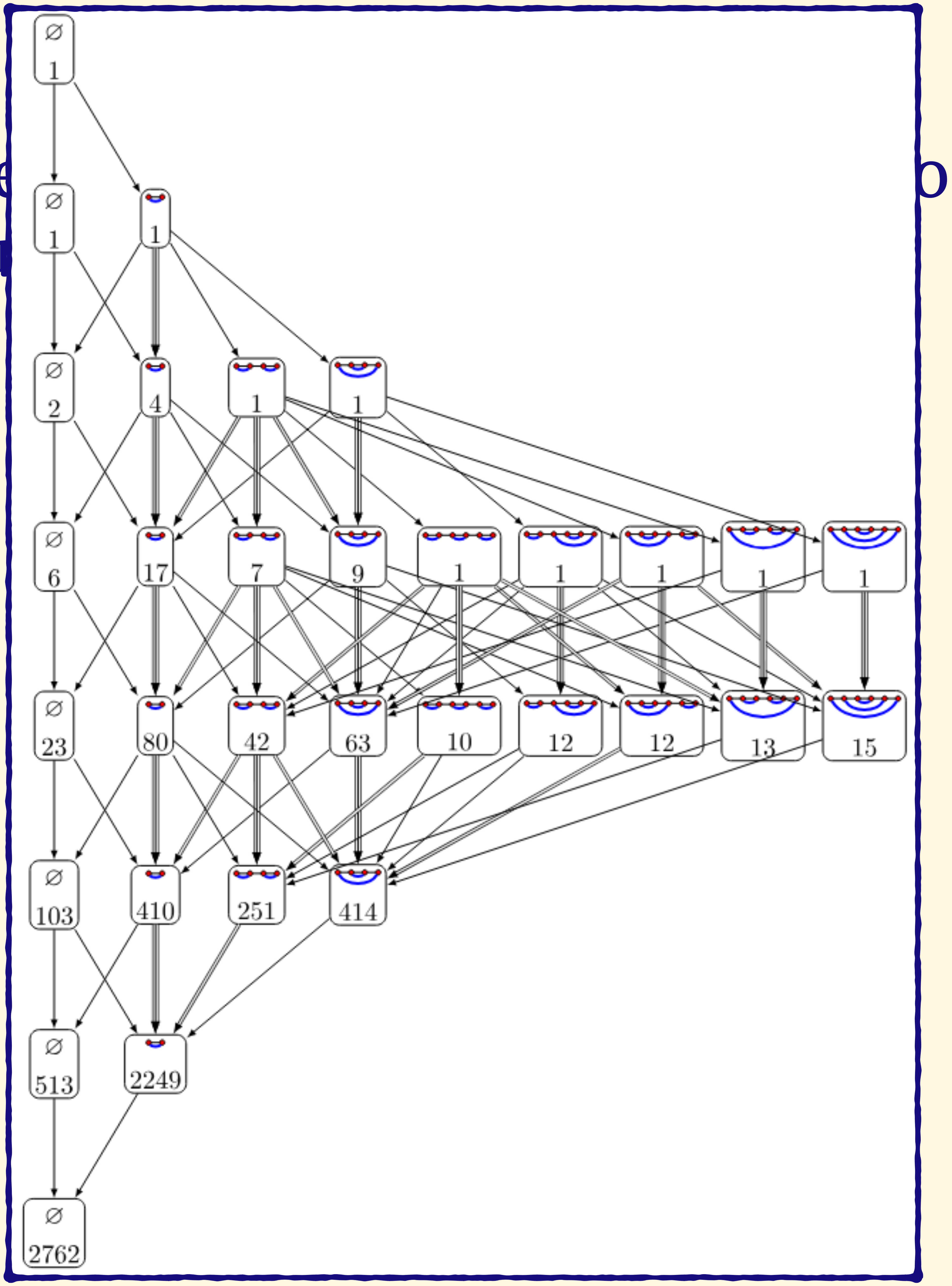
Av(1324)



Insertion Encoding

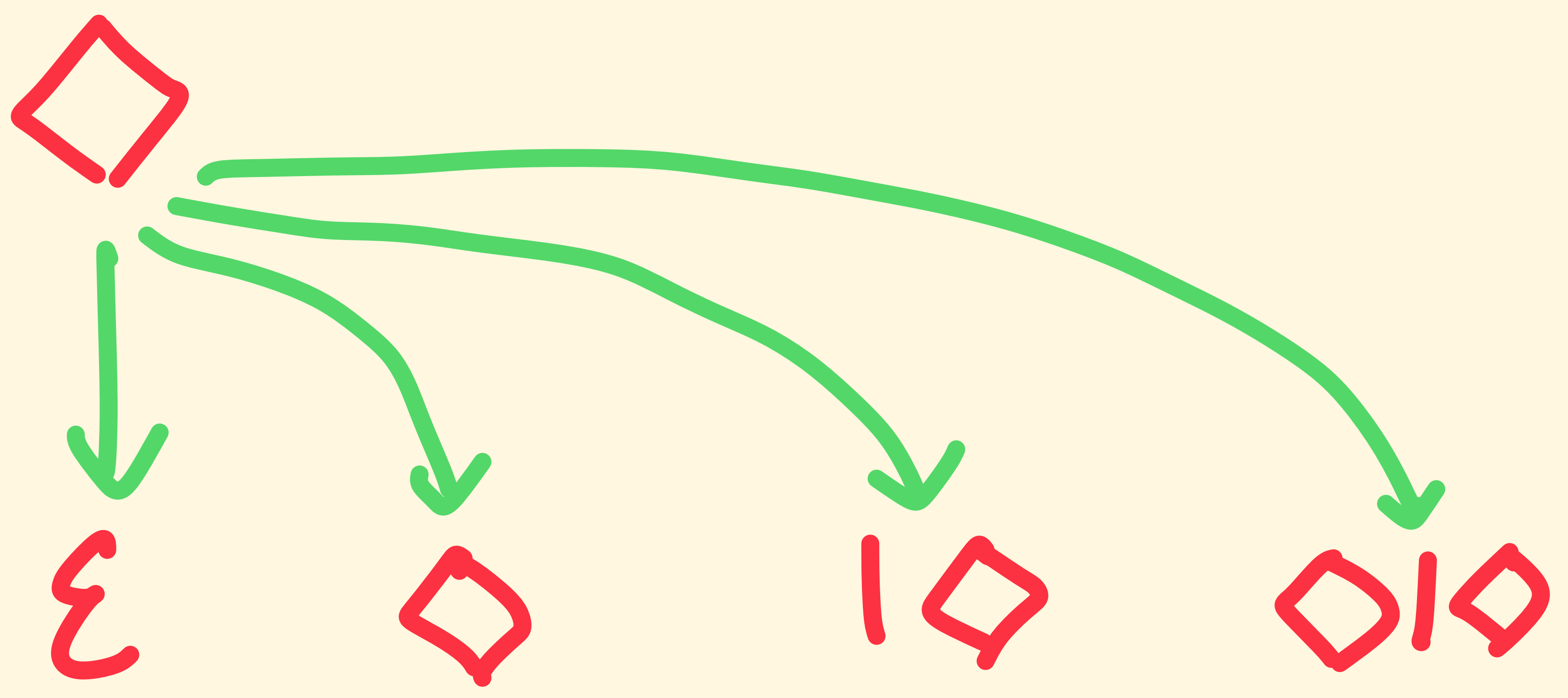
Even if a class has an infinite insertion e … o count the number of permutations in a class u …





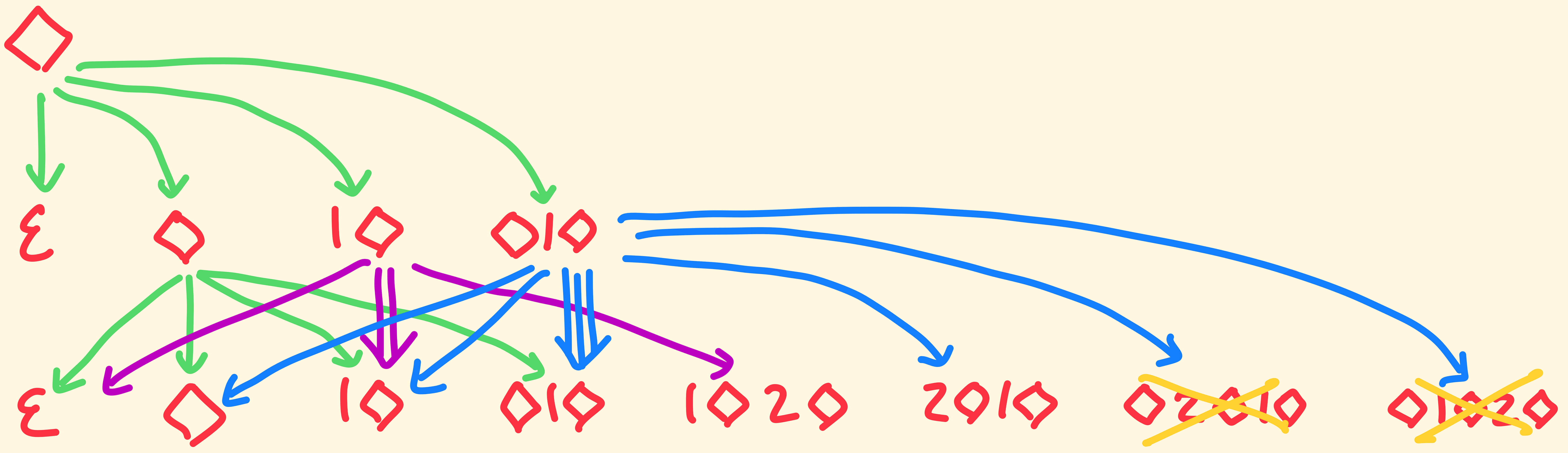
Insertion Encoding

Av(1324)



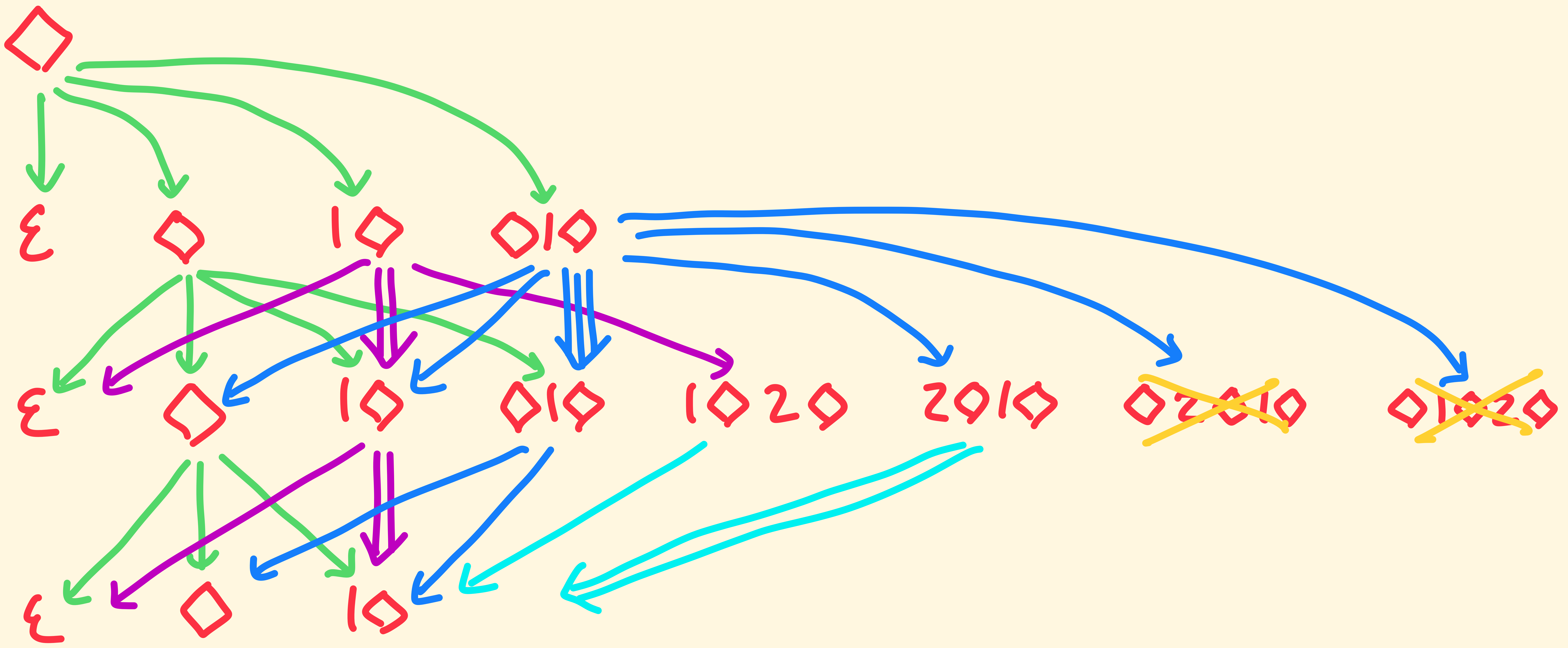
Insertion Encoding

Av(1324)



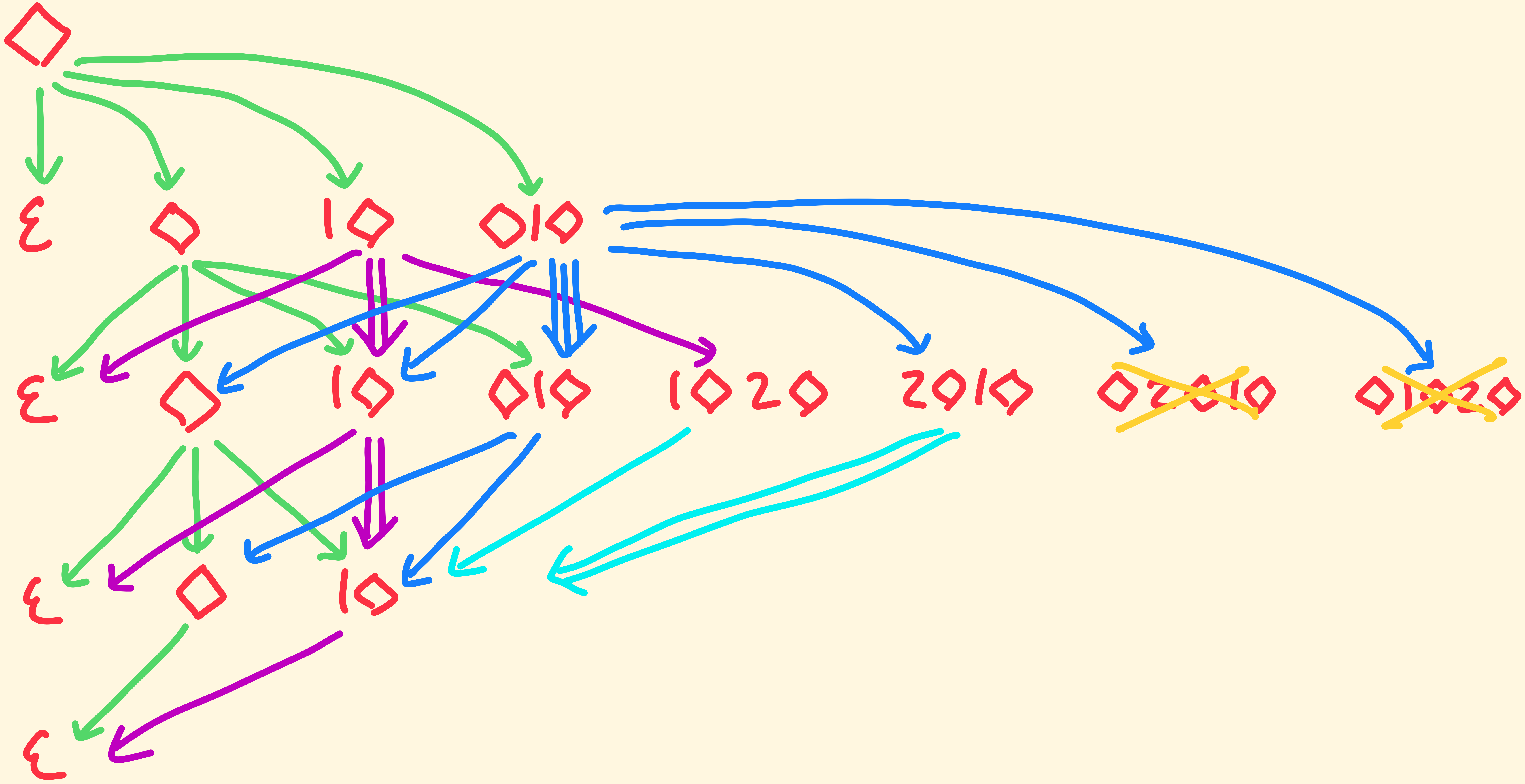
Insertion Encoding

Av(1324)



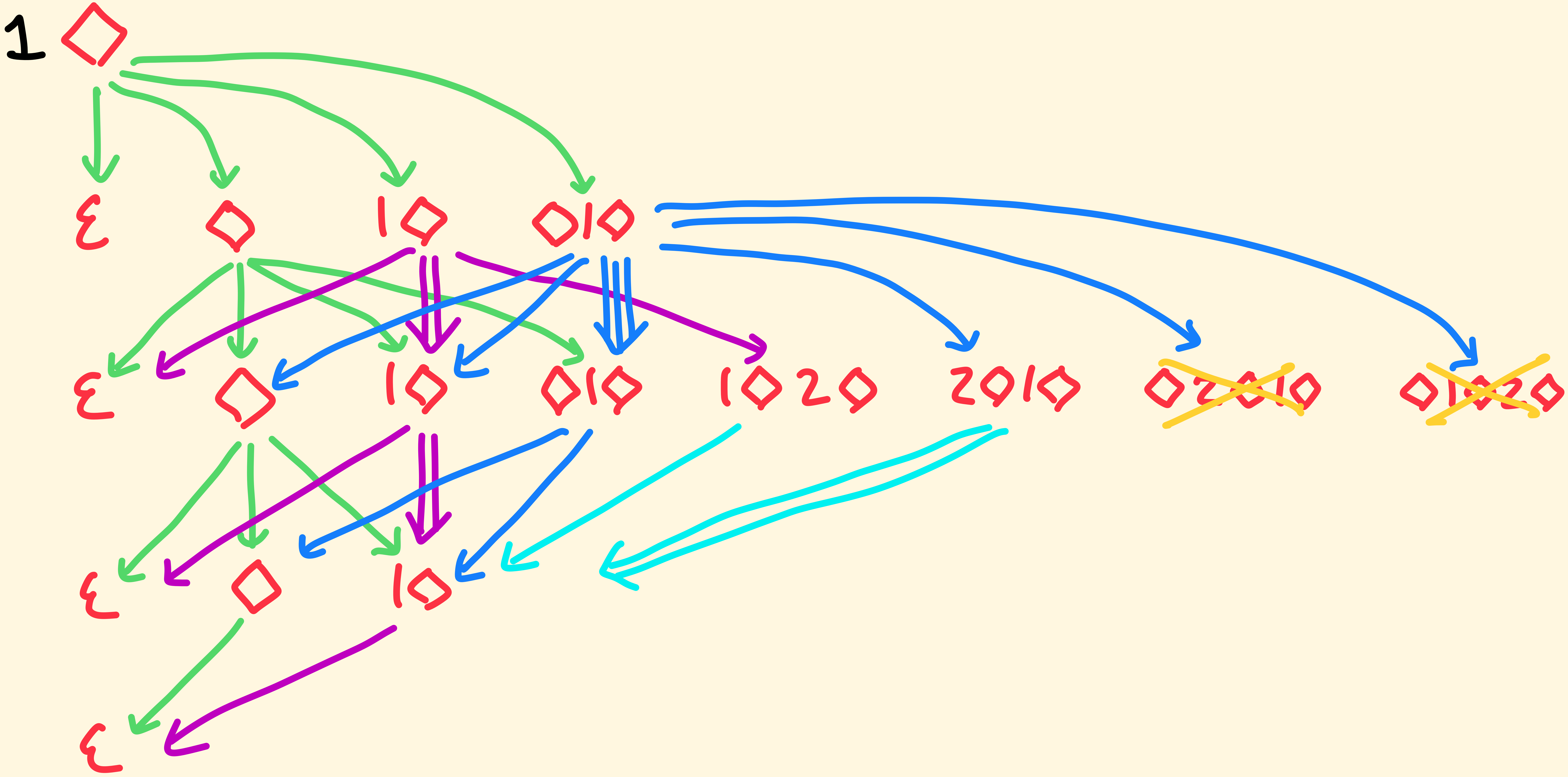
Insertion Encoding

Av(1324)



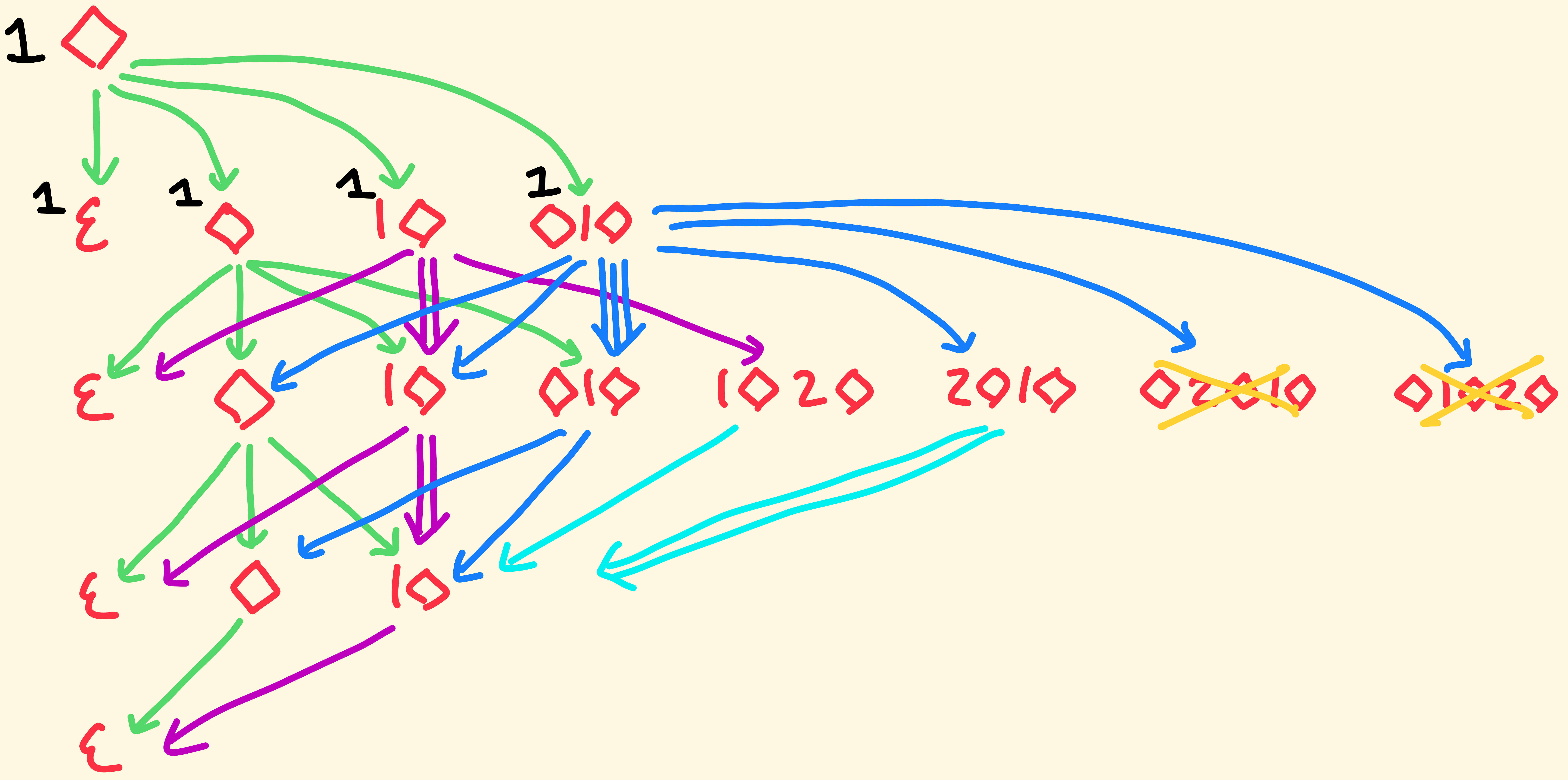
Insertion Encoding

Av(1324)



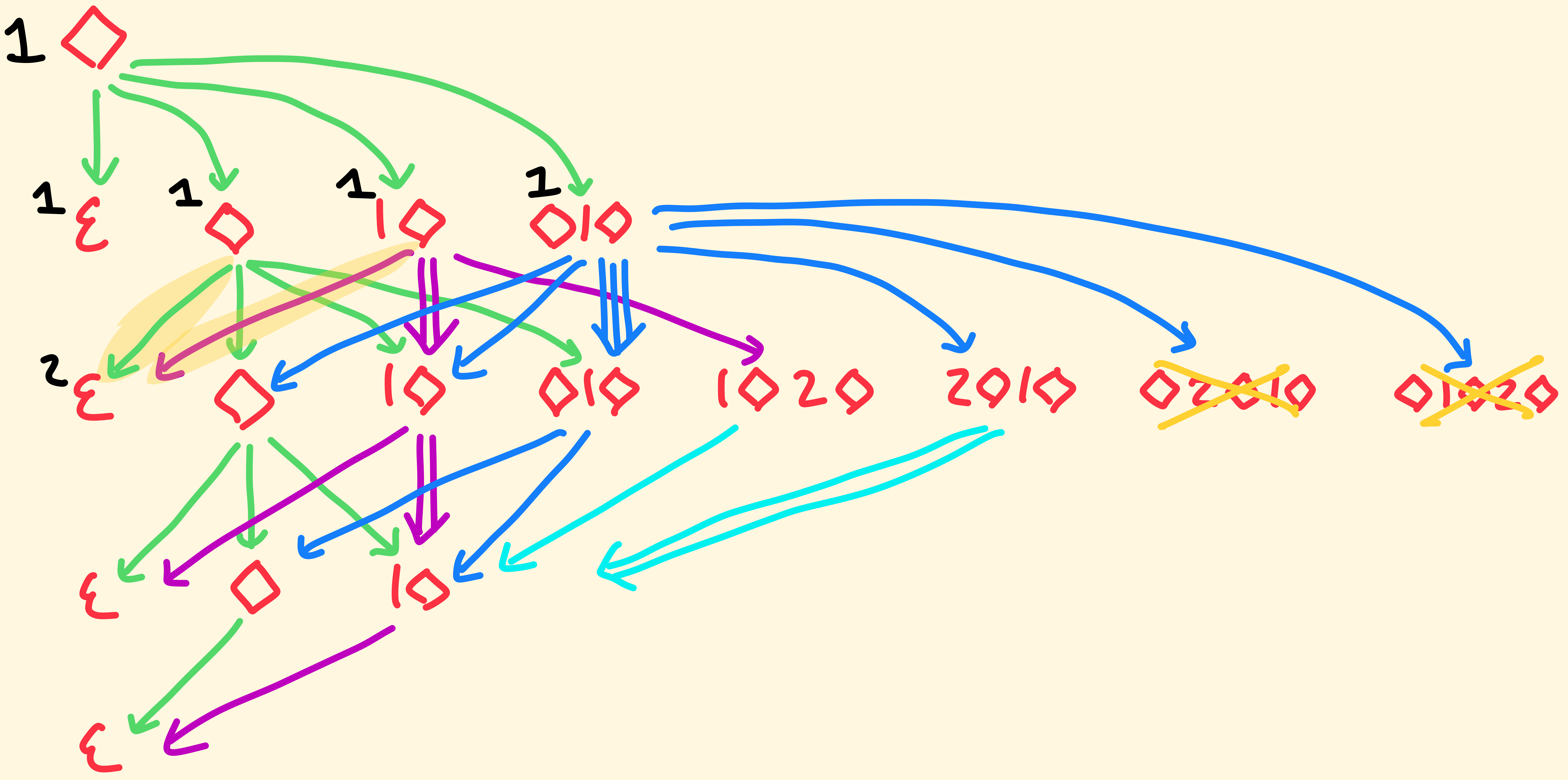
Insertion Encoding

Av(1324)



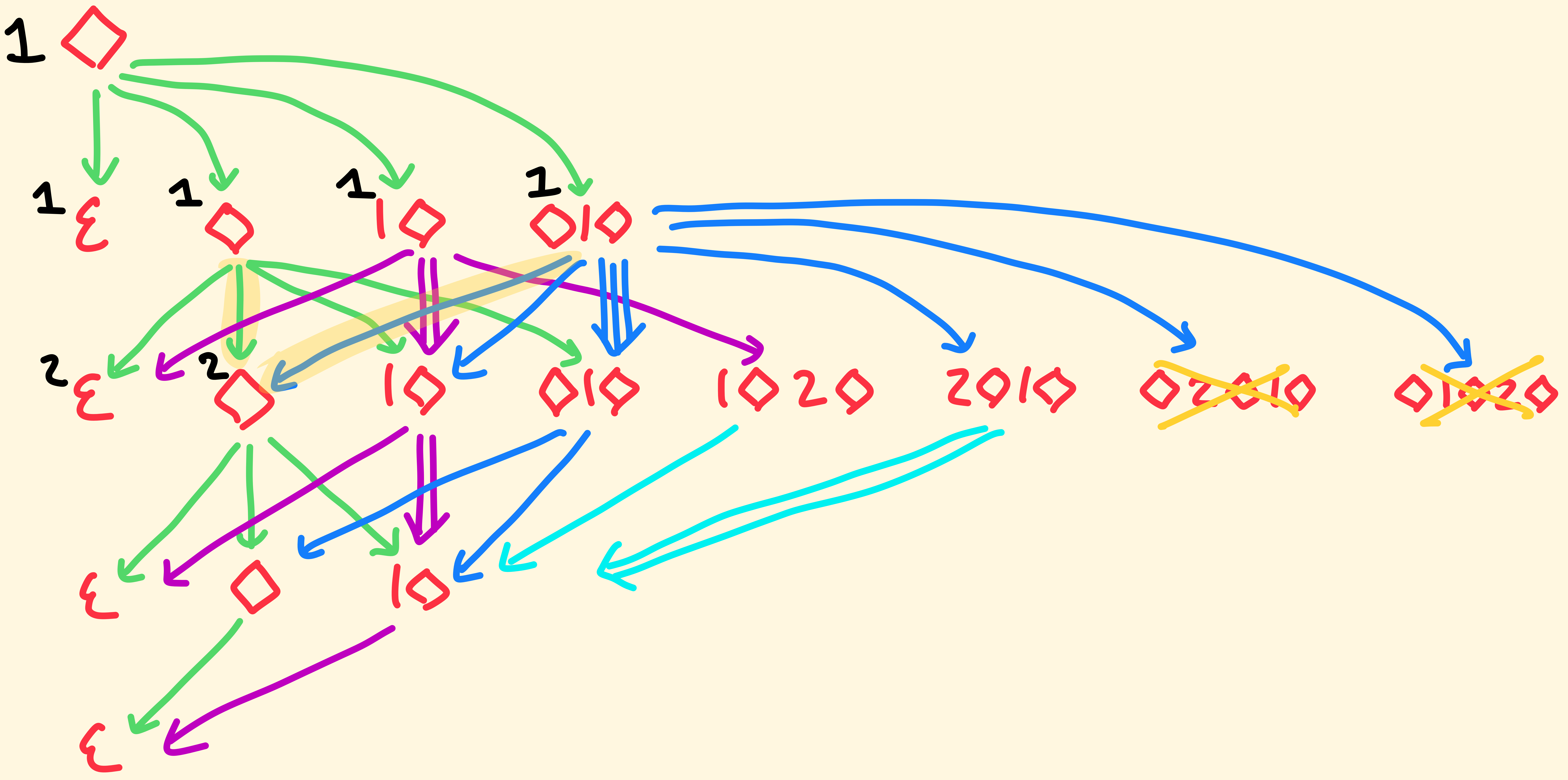
Insertion Encoding

Av(1324)



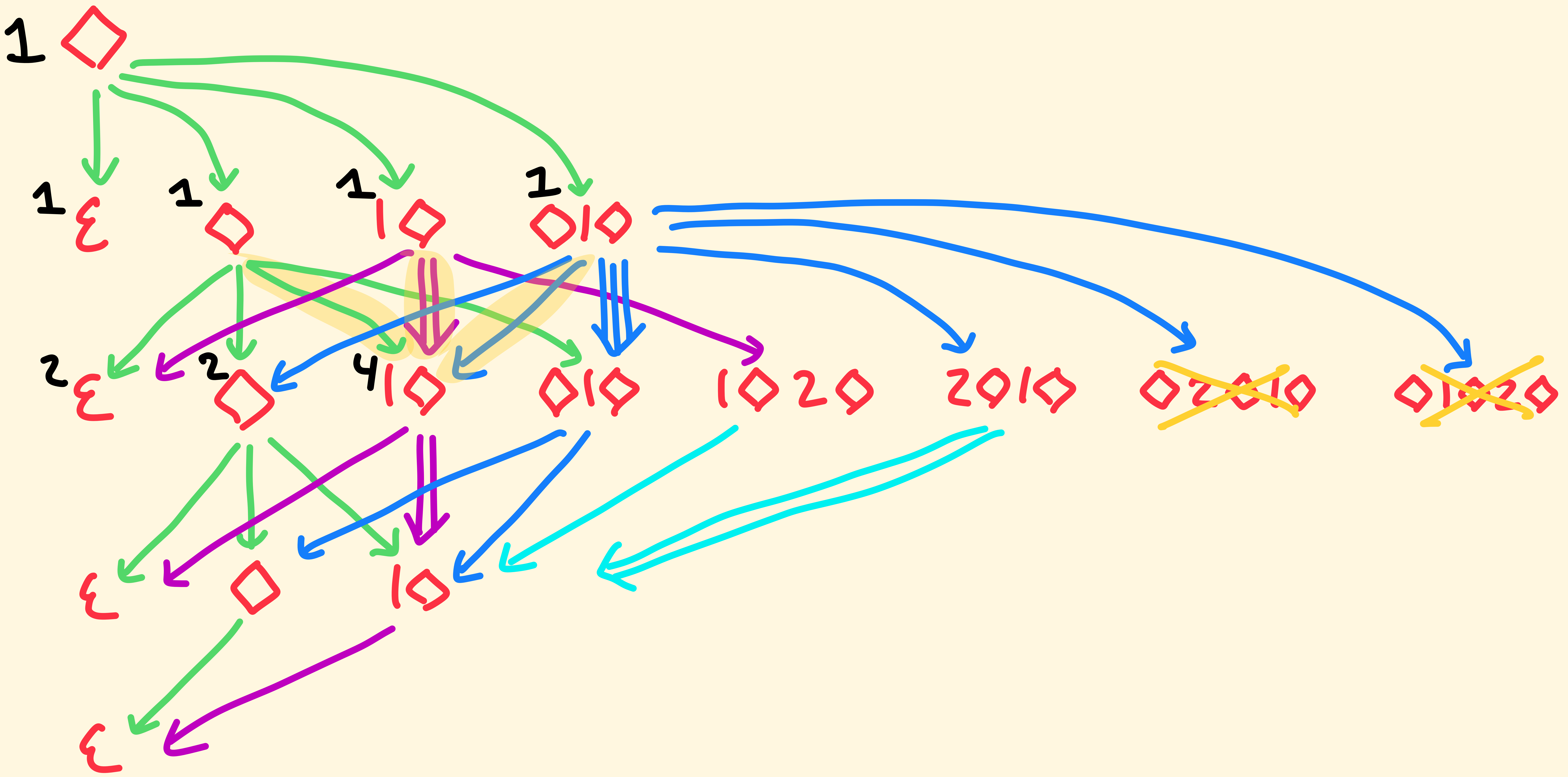
Insertion Encoding

Av(1324)



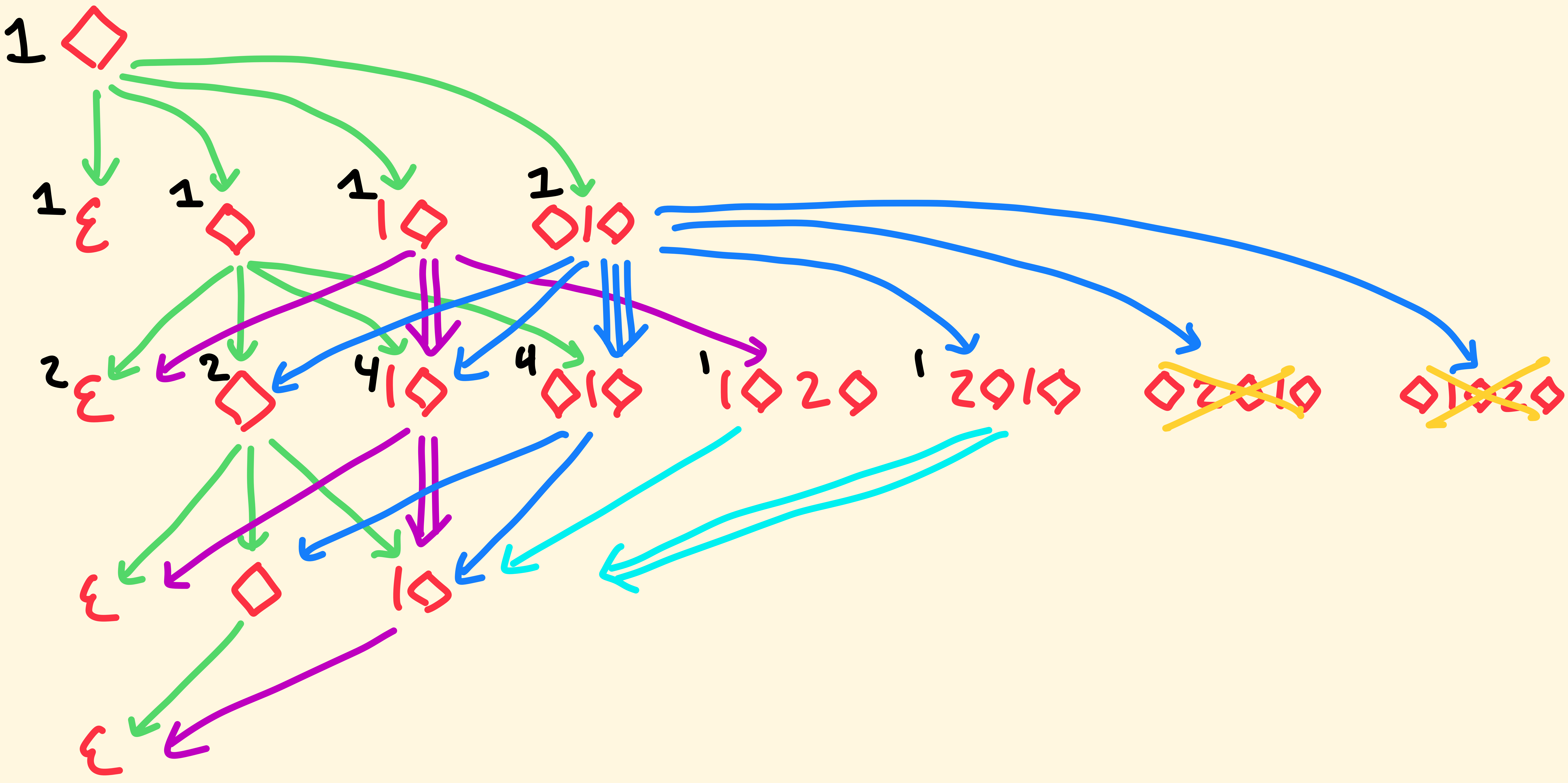
Insertion Encoding

Av(1324)



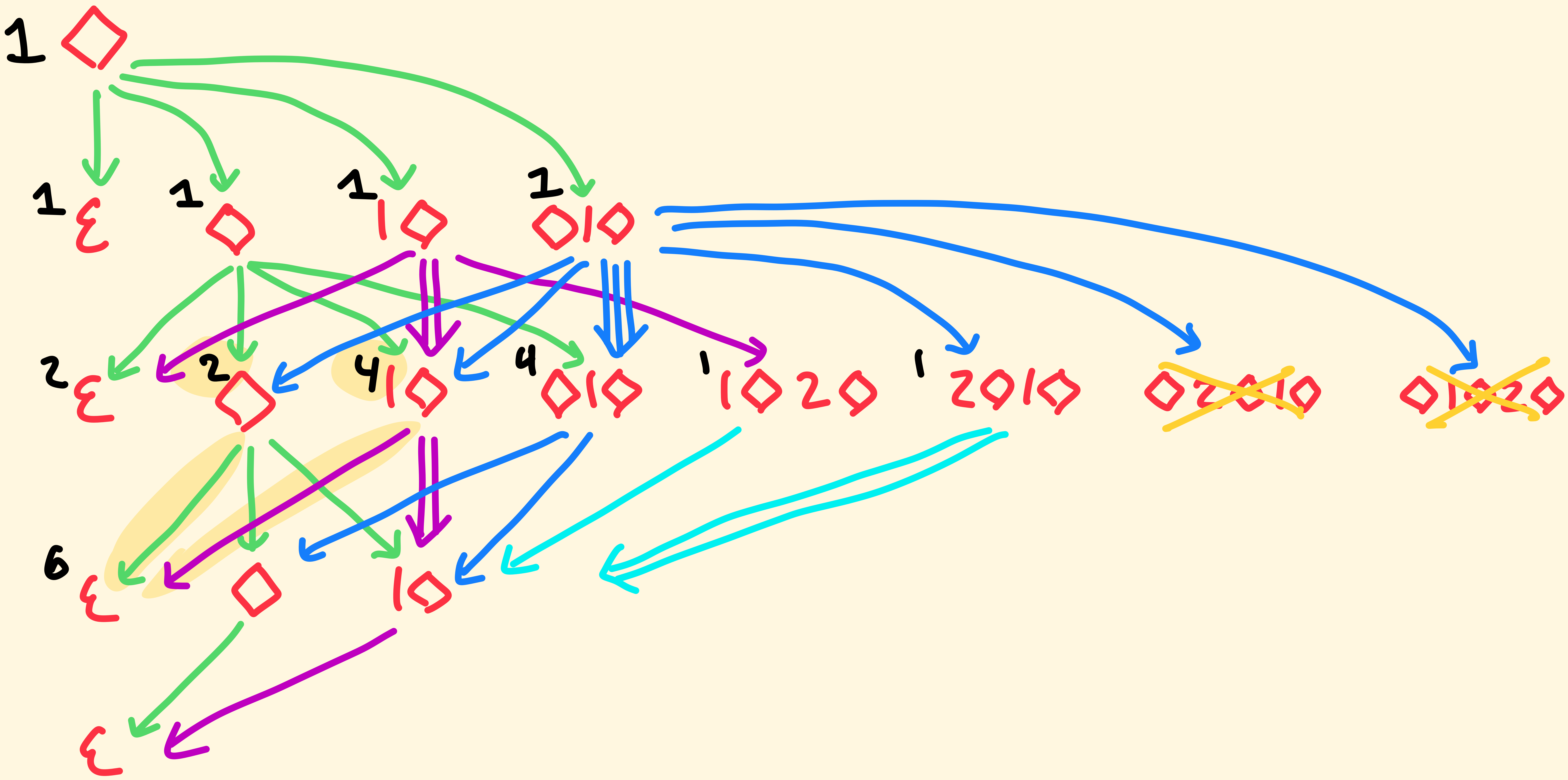
Insertion Encoding

Av(1324)



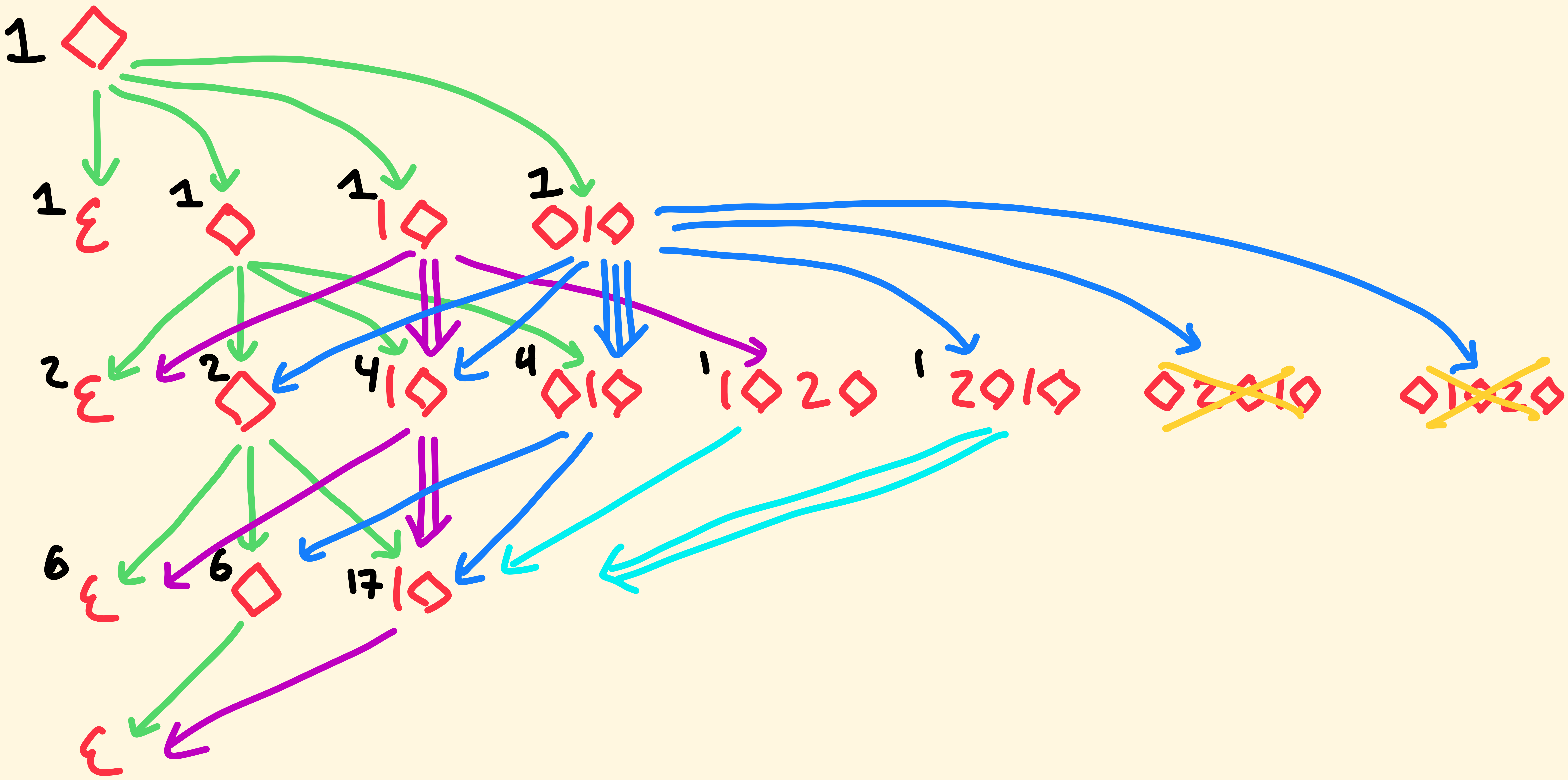
Insertion Encoding

Av(1324)



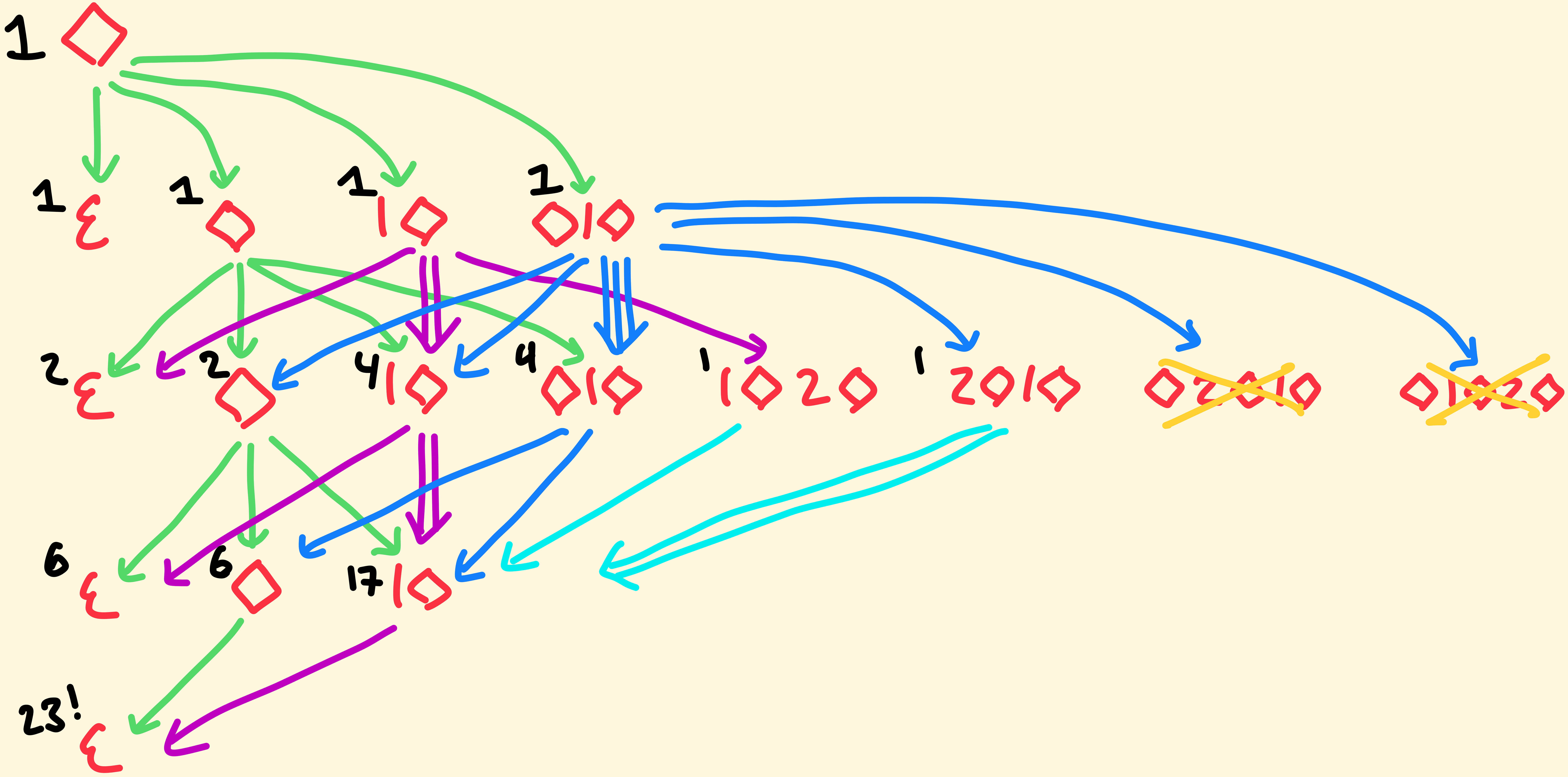
Insertion Encoding

Av(1324)



Insertion Encoding

Av(1324)



Insertion Encoding

The “link diagrams” in the 1324 paper are precisely encoding the relationships between slots — often you cannot fill slot A until after slot B has been closed.

Insertion Encoding

The “link diagrams” in the 1324 paper are precisely encoding the relationships between slots — often you cannot fill slot A until after slot B has been closed.

That means they can follow the link diagrams to know exactly what the transitions between simplified slot configurations are.

Insertion Encoding

The “link diagrams” in the 1324 paper are precisely encoding the relationships between slots — often you cannot fill slot A until after slot B has been closed.

That means they can follow the link diagrams to know exactly what the transitions between simplified slot configurations are.

Huge computational savings because the simplification is an expensive operation in the original insertion encoding.

Insertion Encoding

So the "big picture", translated into the insertion encoding, is that the paper uses a very efficient construction to generate the insertion encoding finite state machine for all states with up to 25 slots.

Insertion Encoding

So the “big picture”, translated into the insertion encoding, is that the paper uses a very efficient construction to generate the insertion encoding finite state machine for all states with up to 25 slots.

It also uses some extremely clever theoretical and optimization tricks to reach length 50!

Table 2
The first 50 terms of the $Av(1324)$ series.

1
2
6
23
103
513
2762
15793
94776
591950
3824112
25431452
173453058
1209639642
8604450011
62300851632
458374397312
3421888118907
25887131596018
198244731603623
1535346218316422
12015325816028313
94944352095728825
757046484552152932
6087537591051072864
49339914891701589053
402890652358573525928
3313004165660965754922
27424185239545986820514
228437994561962363104048
1914189093351633702834757
16130725510342551986540152
136664757387536091240503406
1163812341034817216384582333
9959364766841851088593974979
85626551244475524038311935717
739479176041581588794042743521
6413612398452364144369673970347
55855094052029166019855630997080
488354507551082299792086219184434
4286013140398612535730177106798038
37753338738386034300928290519149333
333720028221302436110132711265898937
2959914488410727889919188039470296624
26338690757116988316771828238926079326
235113956679181729949424482617740434207
2105162587512716675745868833684827184388
18904804517351837590874336467009693522354
170253750251391700942449152528030601519757
1537516984674177479234766336099763469212469

Generalizing to Any Permutation Class

In the rest of this talk, I’ll explain how to generalize this to any permutation class.

Generalizing to Any Permutation Class

In the rest of this talk, I'll explain how to generalize this to any permutation class.

Big idea: We use a structure that automatically discovers and tracks the relationships between slots.

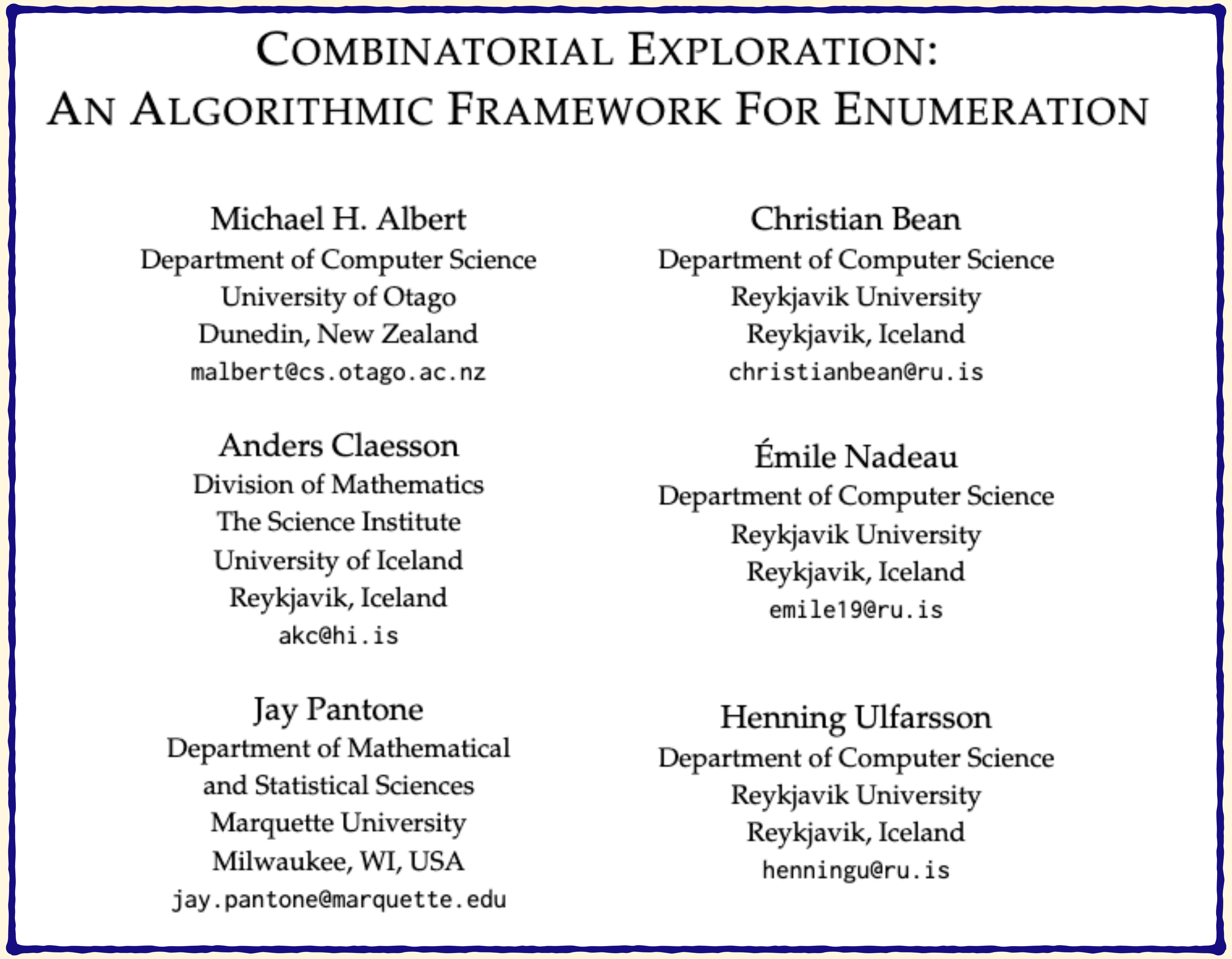
Generalizing to Any Permutation Class

In the rest of this talk, I'll explain how to generalize this to any permutation class.

Big idea: We use a structure that automatically discovers and tracks the relationships between slots.

It simultaneously derives the right "link pattern" analogues and uses them to count.

Tilings



COMBINATORIAL EXPLORATION:
AN ALGORITHMIC FRAMEWORK FOR ENUMERATION

Michael H. Albert
Department of Computer Science
University of Otago
Dunedin, New Zealand
malbert@cs.otago.ac.nz

Christian Bean
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
christianbean@ru.is

Anders Claesson
Division of Mathematics
The Science Institute
University of Iceland
Reykjavik, Iceland
akc@hi.is

Émile Nadeau
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
emile19@ru.is

Jay Pantone
Department of Mathematical
and Statistical Sciences
Marquette University
Milwaukee, WI, USA
jay.pantone@marquette.edu

Henning Ulfarsson
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
henningu@ru.is

Automatic enumeration of permutation classes (and other objects)

Discovers (rigorously) combinatorial specifications, which can be turned into generating functions and polynomial-time counting algorithms.

Tili

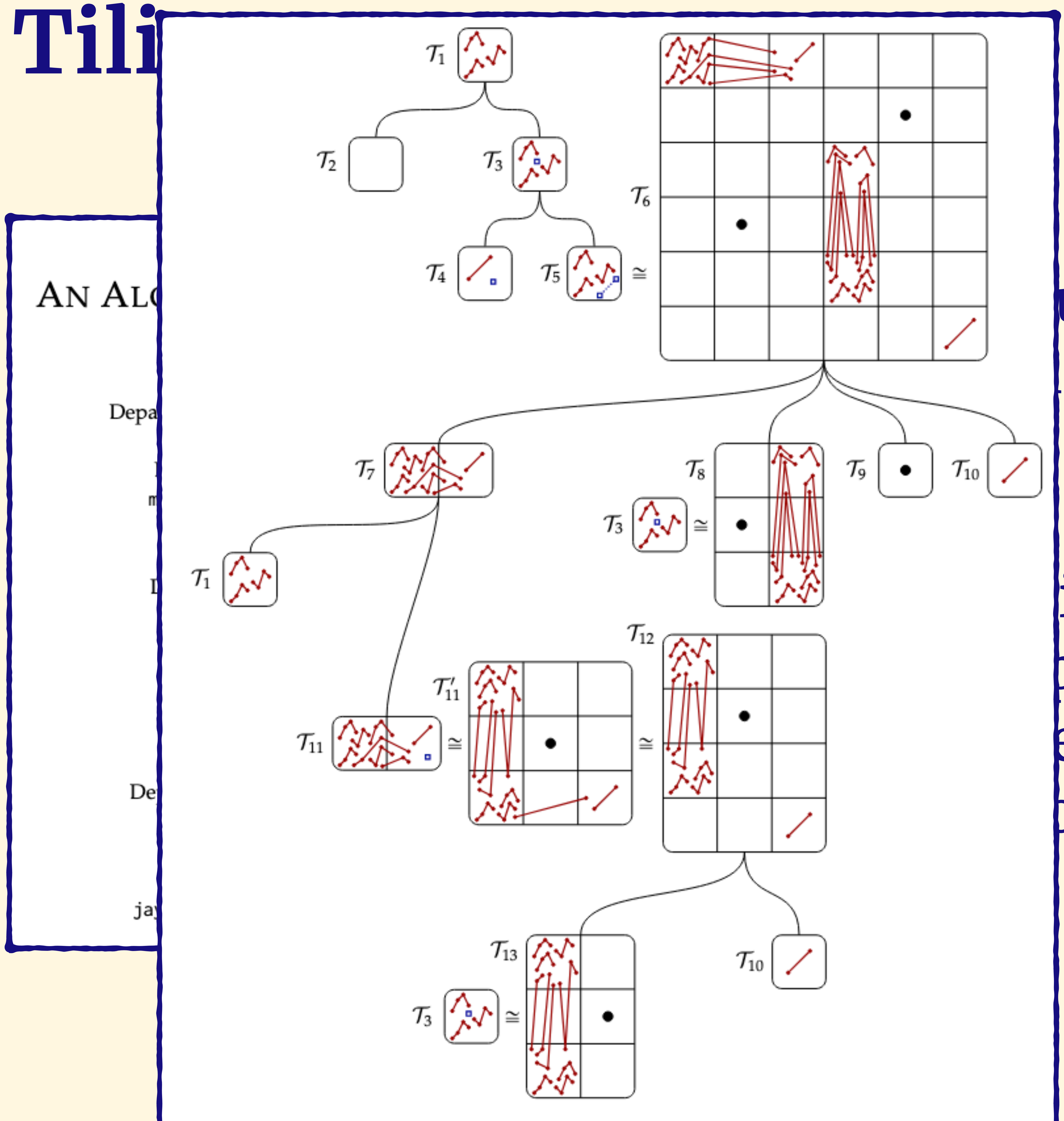
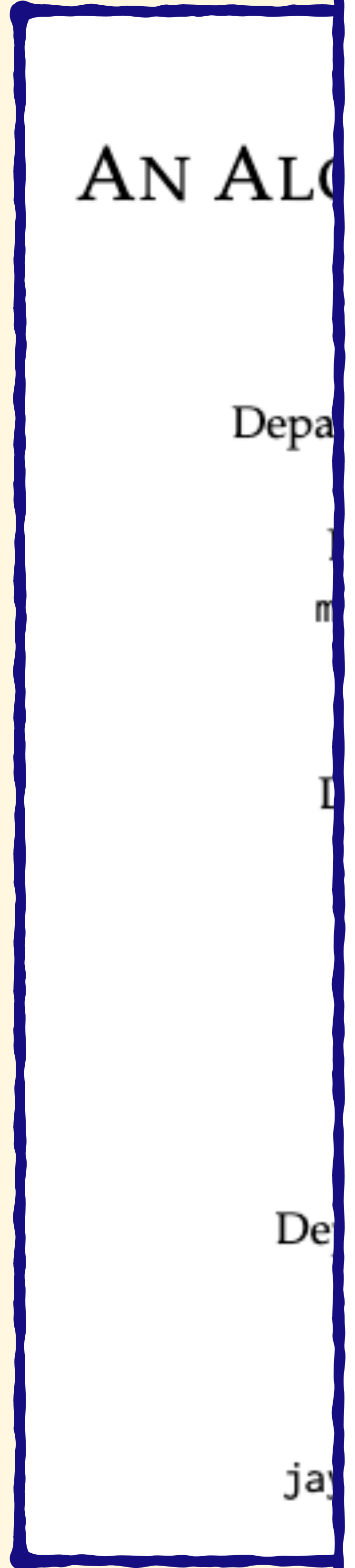


Figure 24: A pictorial representation of the combinatorial specification found by Combinatorial Exploration for Av(1243, 1342, 2143).

utomatic enumeration of permutation
lasses (and other objects)

iscovers (rigorously) combinatorial
pecifications, which can be turned into
enerating functions and polynomial-time
ounting algorithms.

Tili



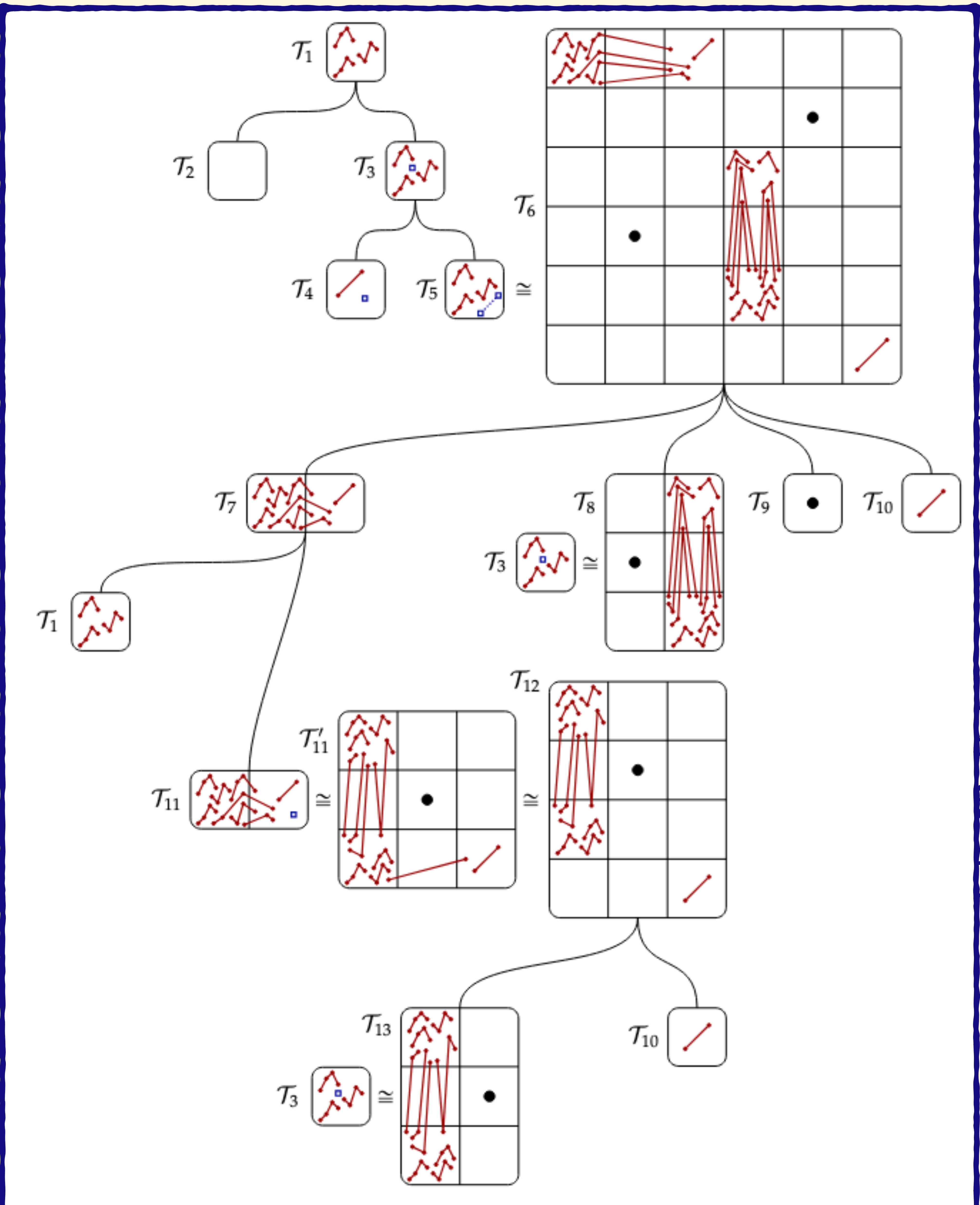


Figure 24: A pictorial representation of the combinatorial specification found by Combinatorial Exploration for Av(1243, 1342, 2143).

$$
\begin{aligned}
T_1(x) &= T_2(x) + E_3(x)\\
T_2(x) &= 1\\
E_3(x) &= T_4(x) + E_5(x)\\
T_4(x) &= x/(1-x)\\
E_5(x) &= T_7(x)\cdot E_3(x)\cdot T_9(x)\cdot T_{10}(x)\\
T_7(x) &= T_1(x) + E_{11}(x)\\
T_9(x) &= x\\
T_{10}(x) &= 1/(1-x)\\
E_{11}(x) &= E_3(x)\cdot T_{10}(x)
\end{aligned}
$$

$$T_1(x) = \frac{1 + x - \sqrt{1 - 6x + 5x^2}}{2x(2-x)}.$$

u … ion of permutation
a … jects)

iscovers (rigorously) combinatorial pecifications, which can be turned into enerating functions and polynomial-time ounting algorithms.

Tili

AN ALG

Depa

Dep

jay

T_1

Fig

Exp

PermPAL Home Examples Search Random

The Permutation Pattern Avoidance Library (PermPAL)

PermPAL is a database of algorithmically-derived theorems about permutation classes.

The Combinatorial Exploration framework produces rigorously verified combinatorial specifications for families of combinatorial objects. These specifications then lead to generating functions, counting sequence, polynomial-time counting algorithms, random sampling procedures, and more.

This database contains 24,454 permutation classes for which specifications have been automatically found. This includes many classes that have been previously enumerated by other means and many classes that have not been previously enumerated.

Some Notables Successes:

- 6 out of 7 of the principal classes of length 4
- all 56 symmetry classes avoiding two patterns of length 4
- all 317 symmetry classes avoiding three patterns of length 4
- the "domino set" used by Bevan, Brignall, Elvey Price, and Pantone to investigate Av(1324)
- the class Av(3412, 52341, 635241) of Alland and Richmond corresponding a type of Schubert variety
- the class Av(2341, 3421, 4231, 52143) equal to the (Av(12), Av(21))-staircase (see Albert, Pantone, and Vatter), which appears to be non-D-finite
- all of the permutation classes counted by the Schröder numbers conjectured by Eric Egge
- the class Av(34251, 35241, 45231), equal to the preimage of Av(321) under the West-stack-sorting operation (see Defant)

Section 2.4 of the article Combinatorial Exploration: An Algorithmic Framework for Enumeration gives a more comprehensive list of notable results.

The comb_spec_searcher github repository contains the open-source python

$(x) \cdot T_{10}(x)$

$6x + 5x^2$

$x)$

ion of permutation

jects)

rously) combinatorial

which can be turned into

ctions and polynomial-time

ithms.

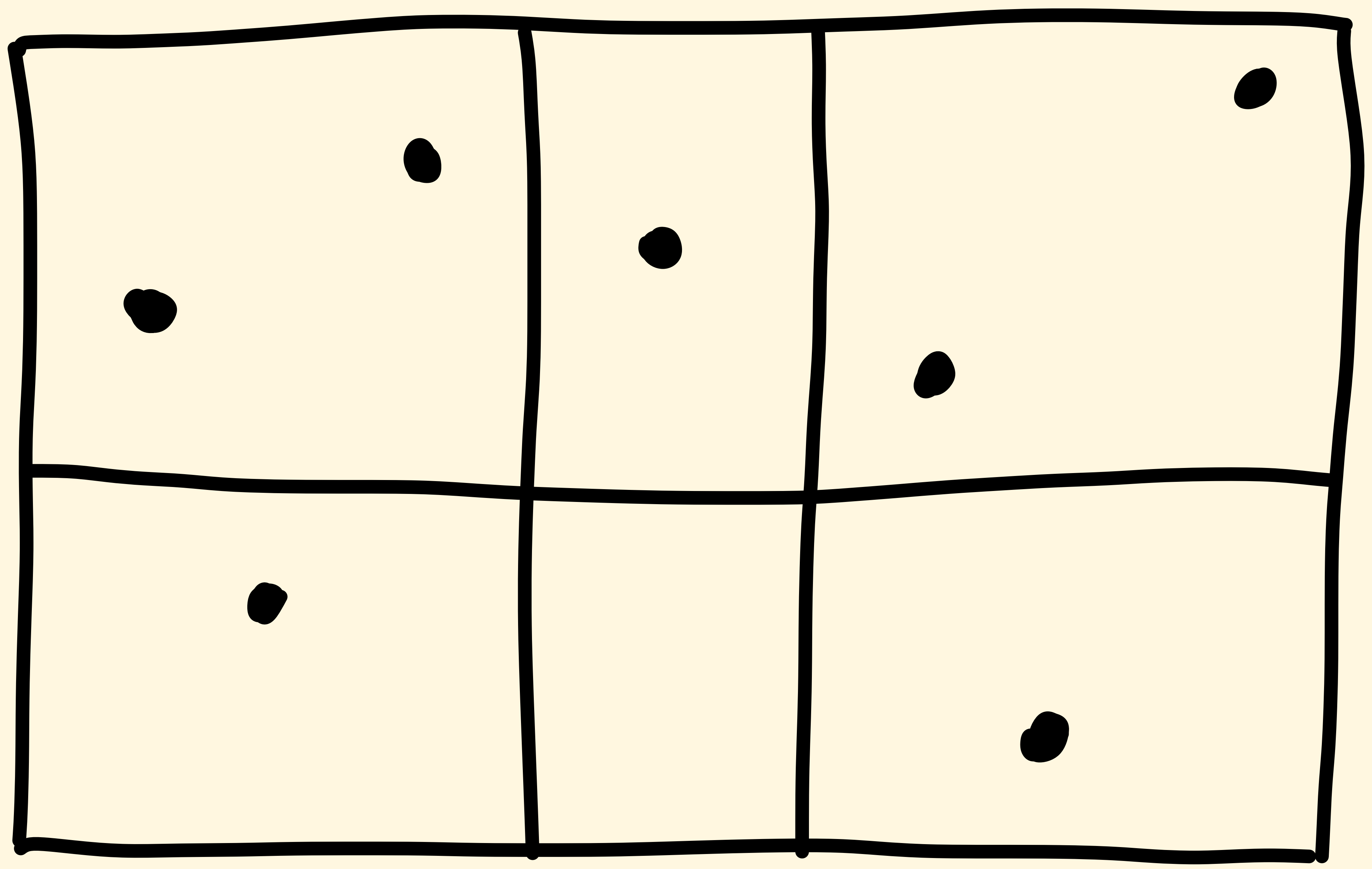
Tilings

One of the fundamental tools for Combinatorial Exploration is the *tiling*. It's essentially a data structure that represents a set of (gridded) permutations.

Tilings

One of the fundamental tools for Combinatorial Exploration is the *tiling*. It's essentially a data structure that represents a set of (gridded) permutations.

Gridded permutation = a permutation with grid lines draw so that entries are split into cells of a grid

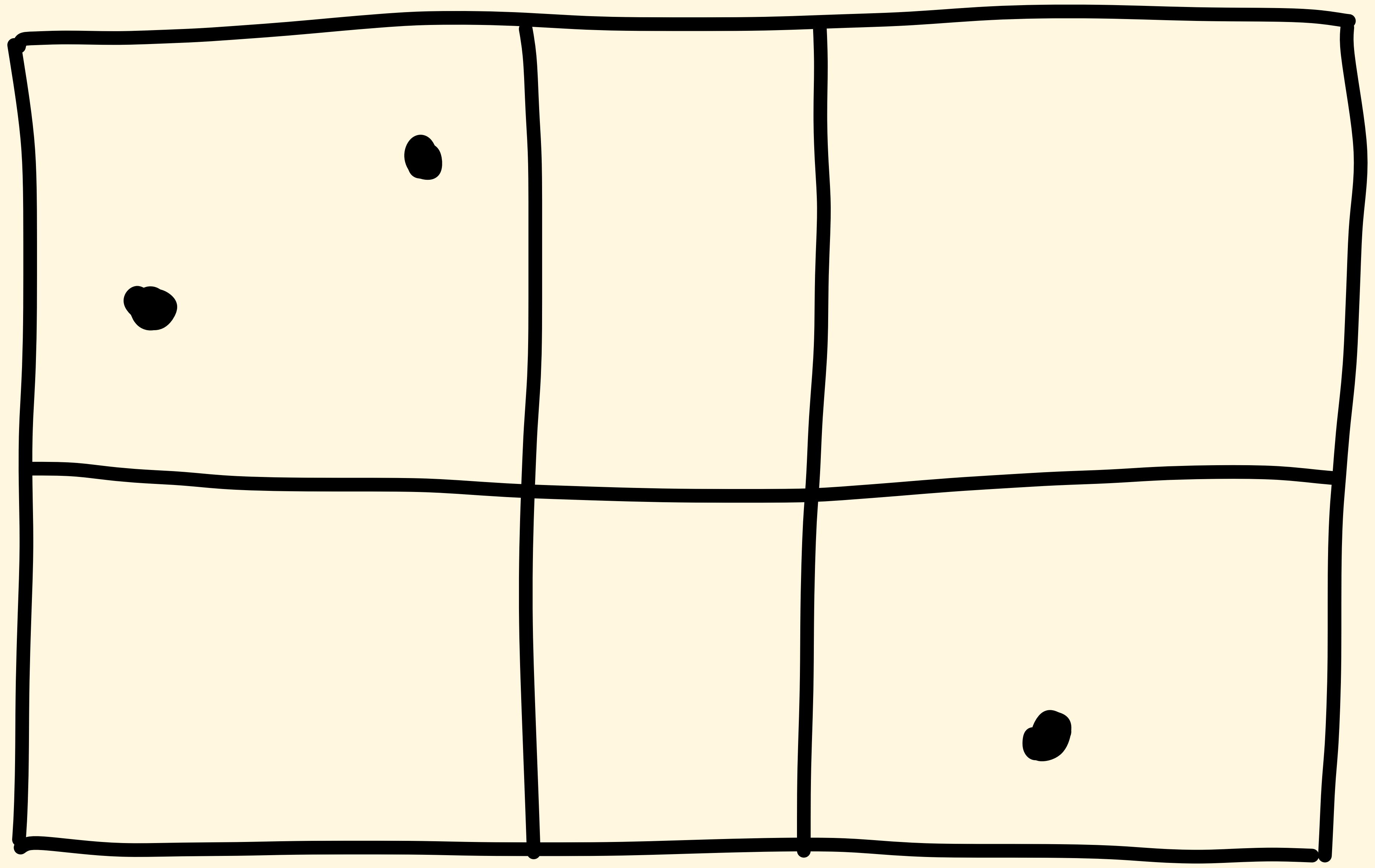


underlying permutation:

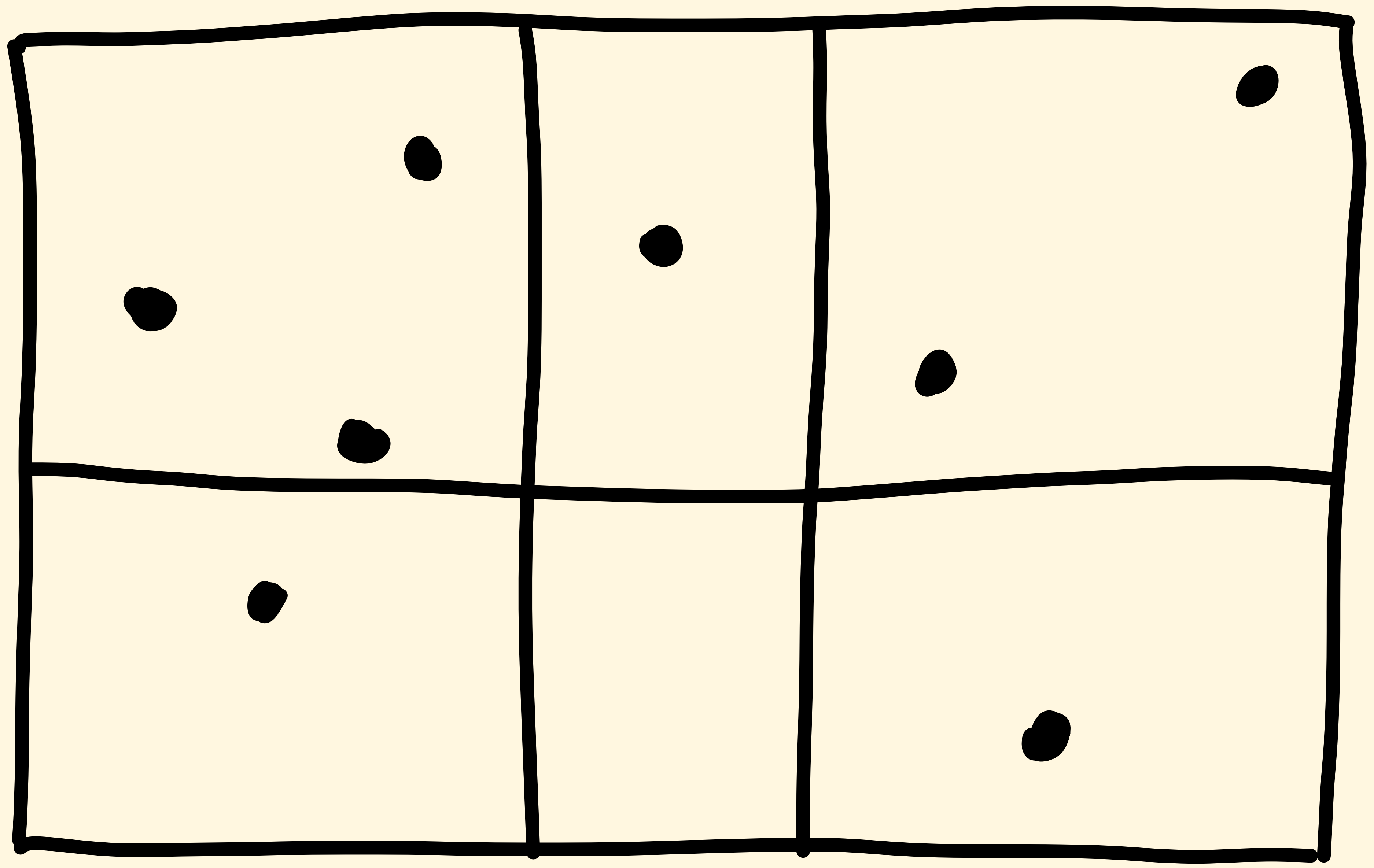
4265317

Tilings

A gridded permutation p contains a gridded permutation q as a pattern if there is a subsequence of entries of p that are order-isomorphic to q and <u>in the same cells.</u>

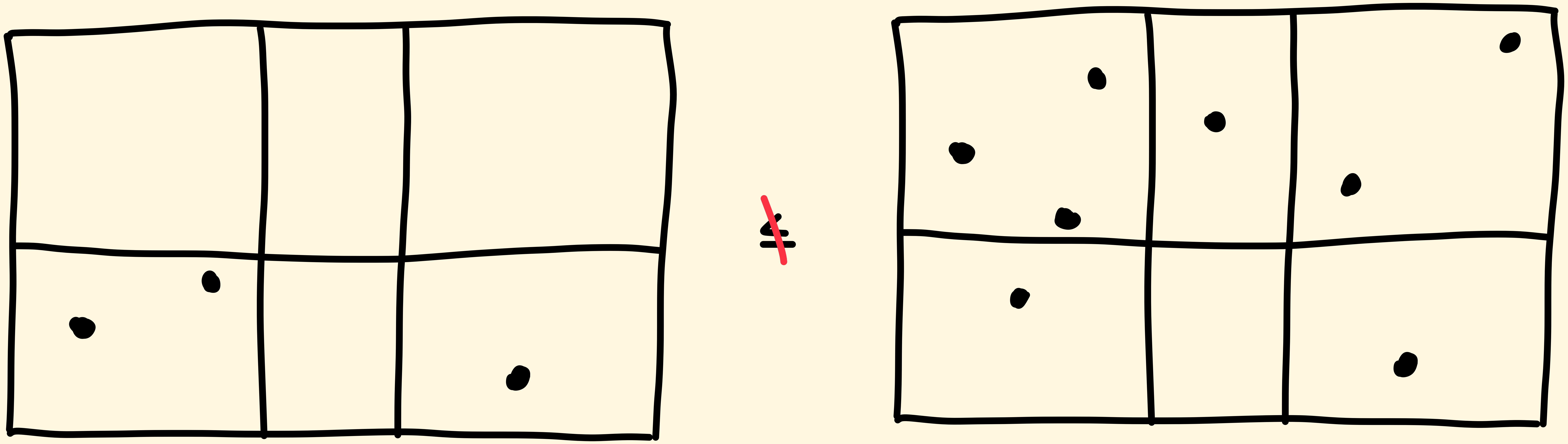


$\leq$



Tilings

A gridded permutation p contains a gridded permutation q as a pattern if there is a subsequence of entries of p that are order-isomorphic to q and in the same cells.

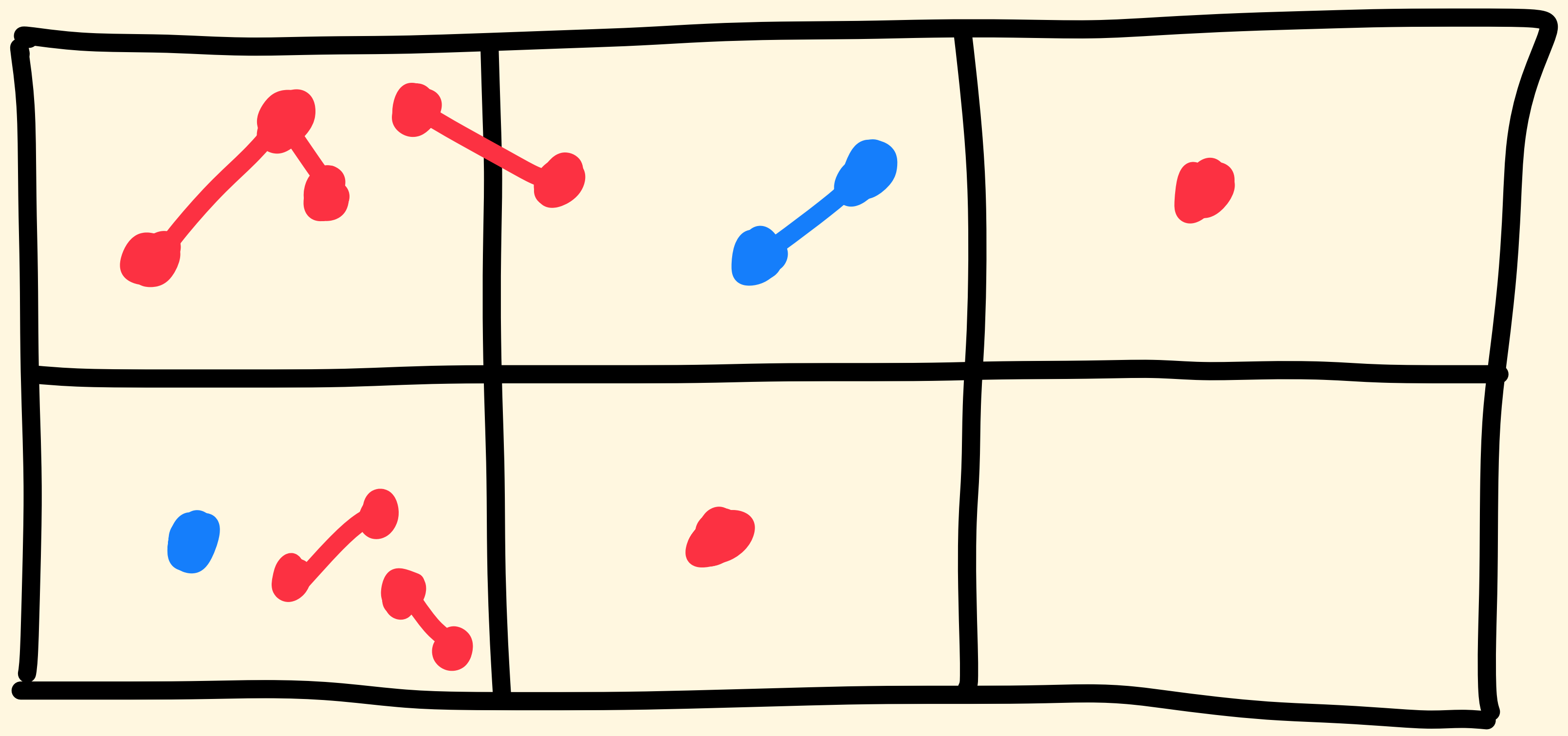


Tilings

A *tiling* is a grid with
obstructions: gridded permutations that must be avoided
requirements: gridded permutations that must be contained

A tiling represents the set of all gridded permutations that can be drawn on that grid that avoid all of the obstructions and contain all of the requirements.

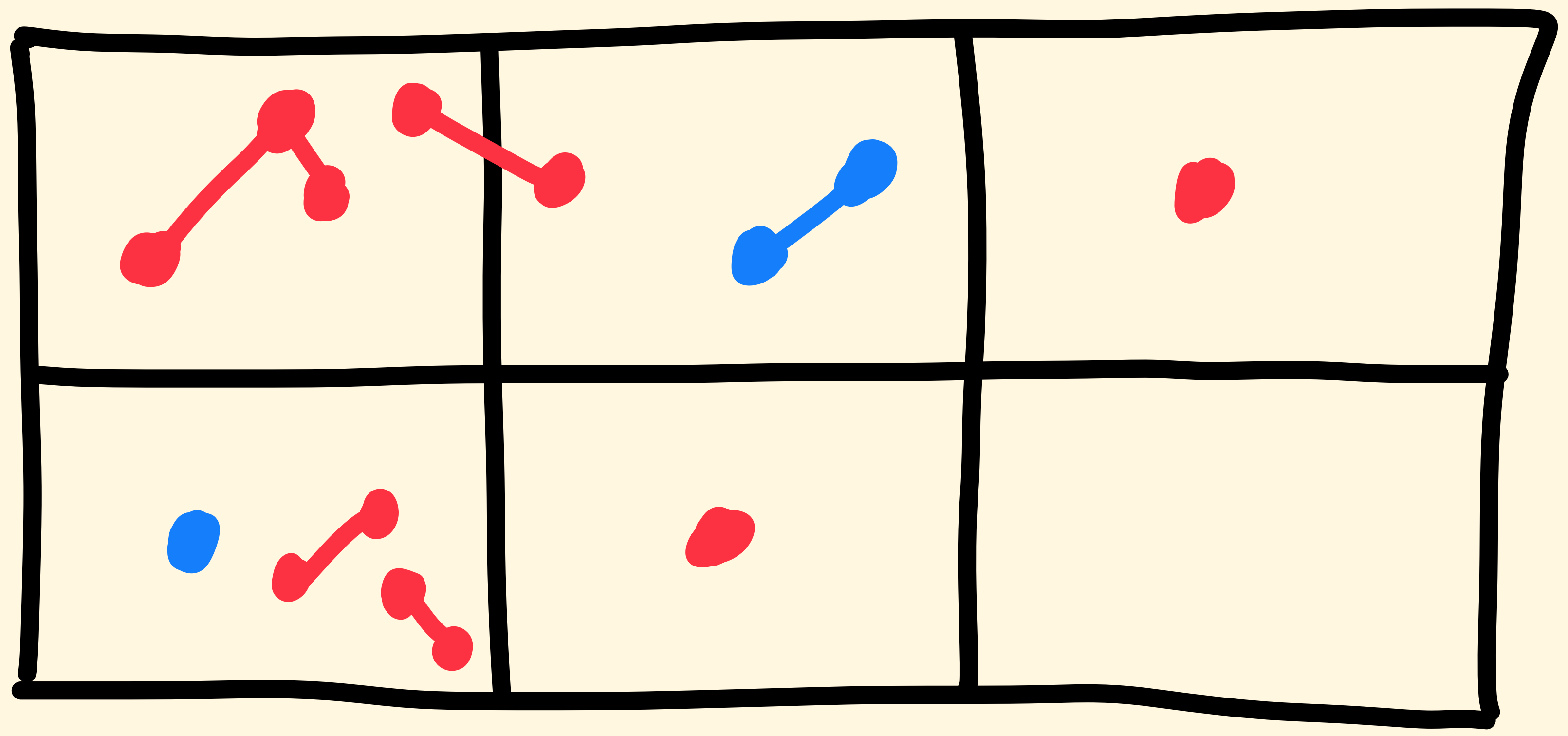
Tilings

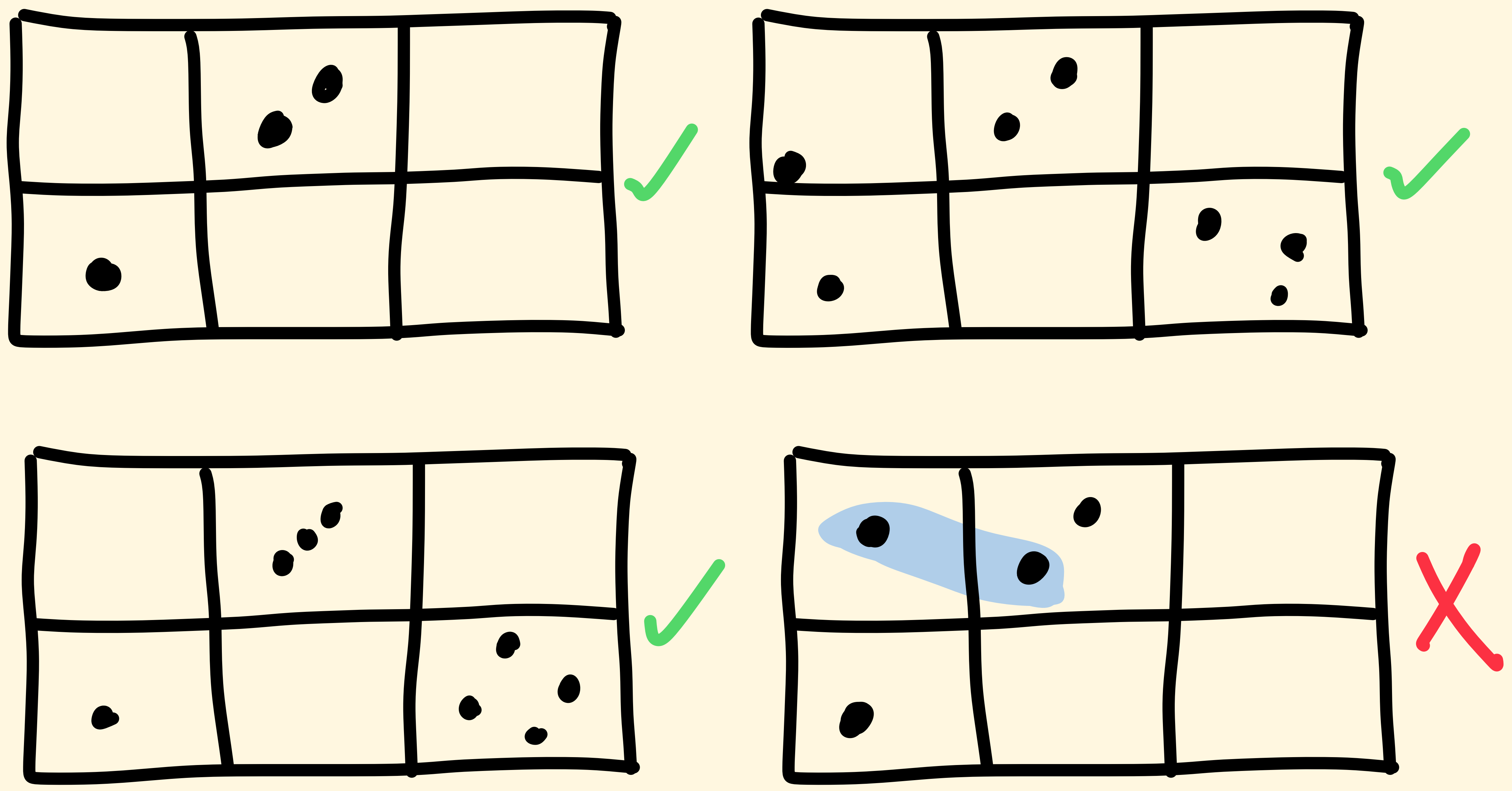


The tiling represents all gridded permutations on a 2x3 grid with:

- exactly one point in the bottom-left cell
- no points in the bottom-middle or top-right cells
- no 132 pattern in the top left cell
- no crossing 21 pattern between the top-left and top-middle cells
- contains a 12 pattern in the top-middle cell

Tilings

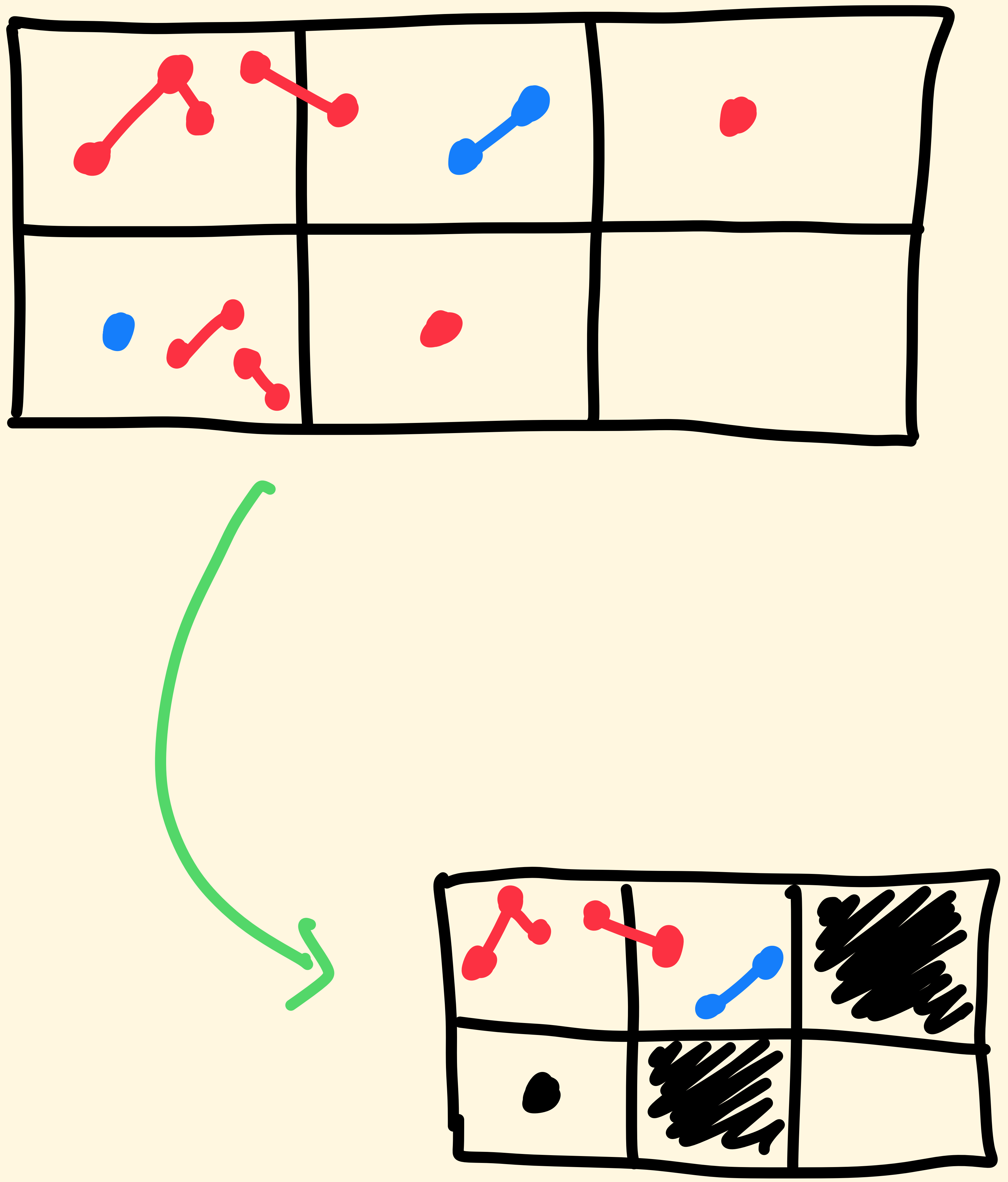


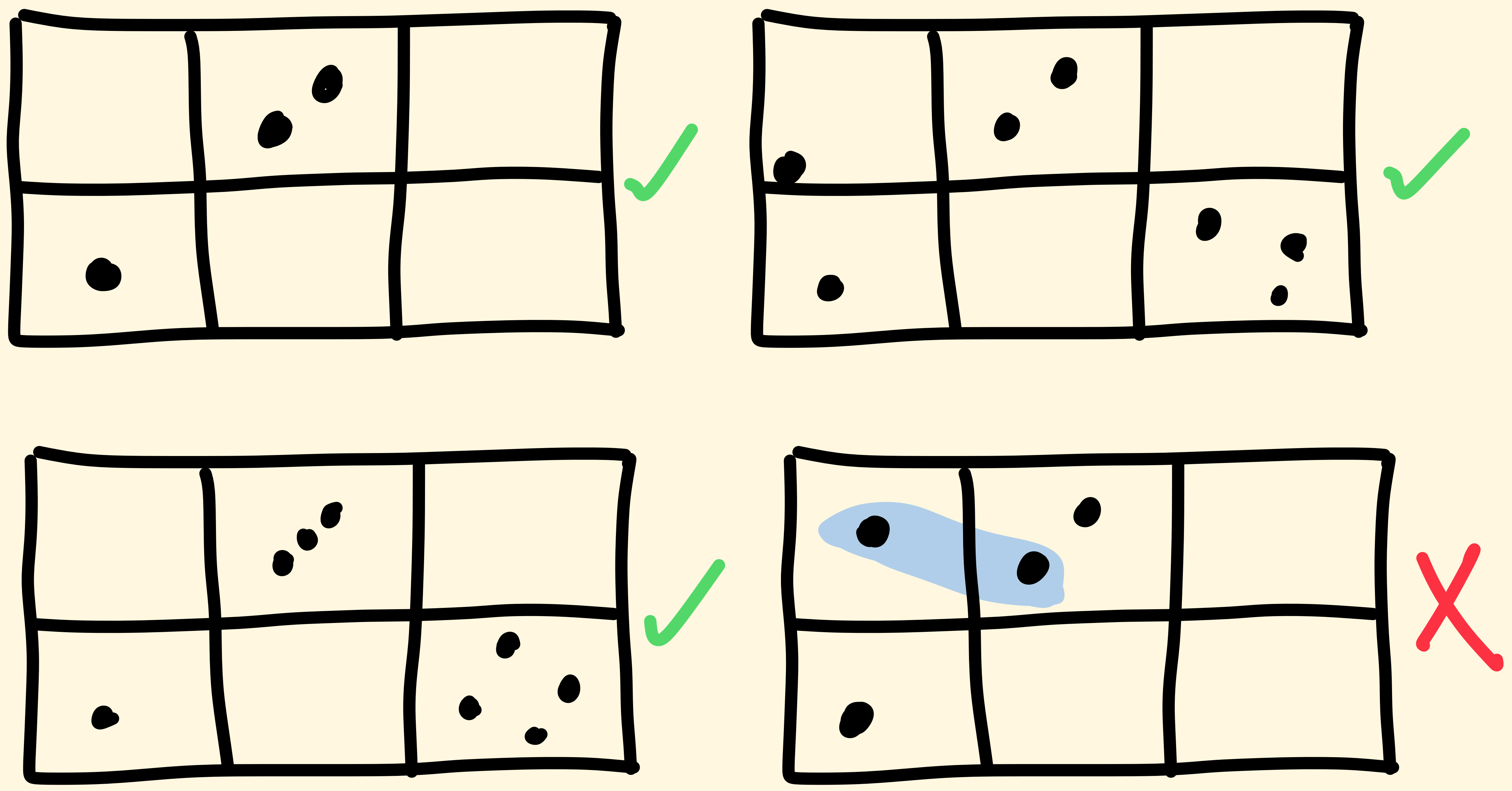


The tiling represents all gridded permutations on a 2x3 grid with:

- exactly one point in the bottom-left cell
- no points in the bottom-middle or top-right cells
- no 132 pattern in the top left cell
- no crossing 21 pattern between the top-left and top-middle cells
- contains a 12 pattern in the top-middle cell

Tilings





Tilings

We can generate the insertion encoding graph using tilings instead of slot configurations!

Tilings

We can generate the insertion encoding graph using tilings instead of slot configurations!

Each tiling represents a set of permutations just like each insertion encoding configurations represents the set of permutations that can be generated from that configuration.

Tilings

We can generate the insertion encoding graph using tilings instead of slot configurations!

Each tiling represents a set of permutations just like each insertion encoding configurations represents the set of permutations that can be generated from that configuration.

It is a fast operation to “place an entry into a slot” on a tiling and simplify the obstructions.

Tilings

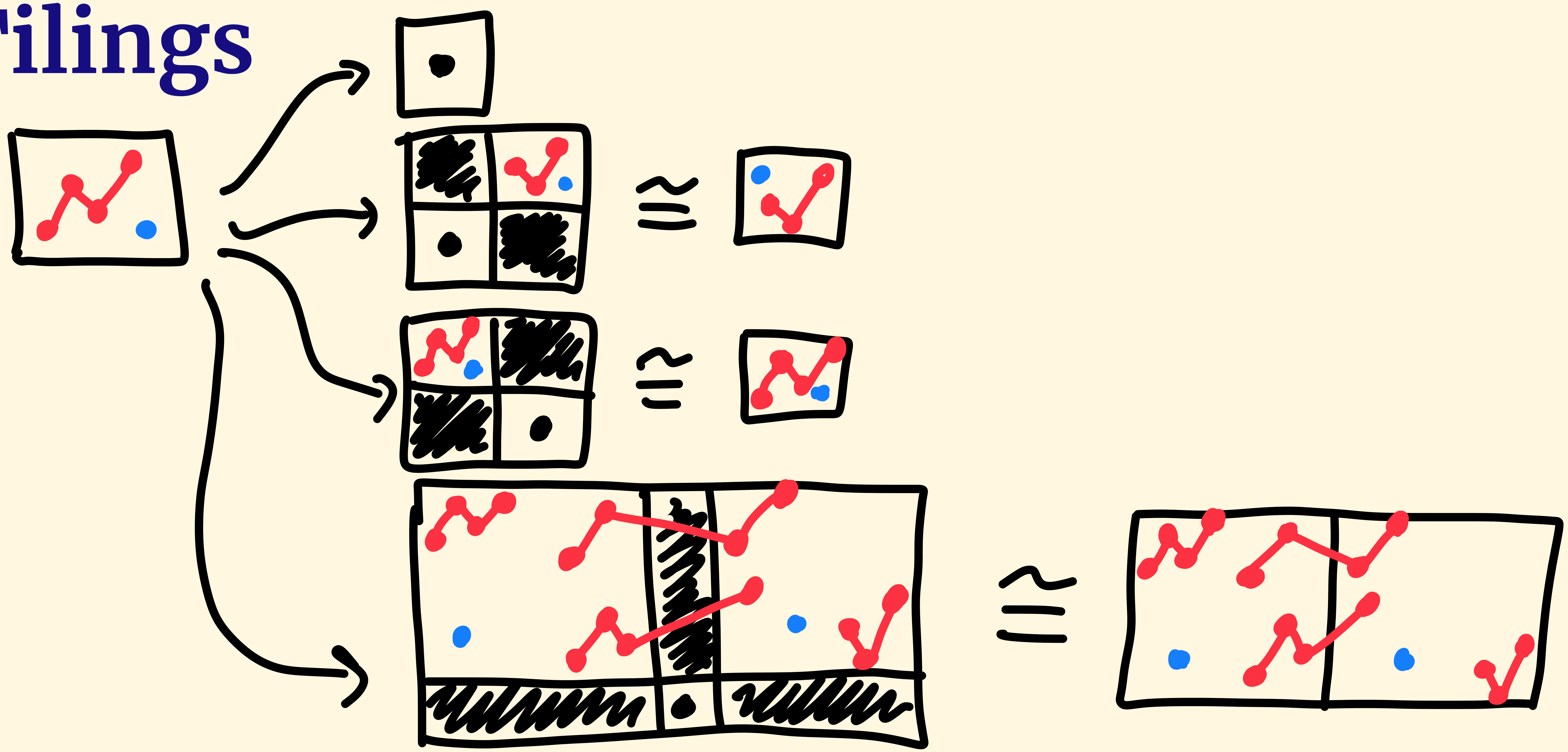
We can generate the insertion encoding graph using tilings instead of slot configurations!

Each tiling represents a set of permutations just like each insertion encoding configurations represents the set of permutations that can be generated from that configuration.

It is a fast operation to "place an entry into a slot" on a tiling and simplify the obstructions.

No expensive checks, just like the link patterns in 1324, but we didn't need to first describe and prove any structure by hand.

Tilings



We can remove the points and only use the top row because the obstructions already keep track of where the bad patterns can show up.

Two states are isomorphic when they are simply the same tiling. For the original insertion encoding this was a <u>very</u> expensive check.

Tilings

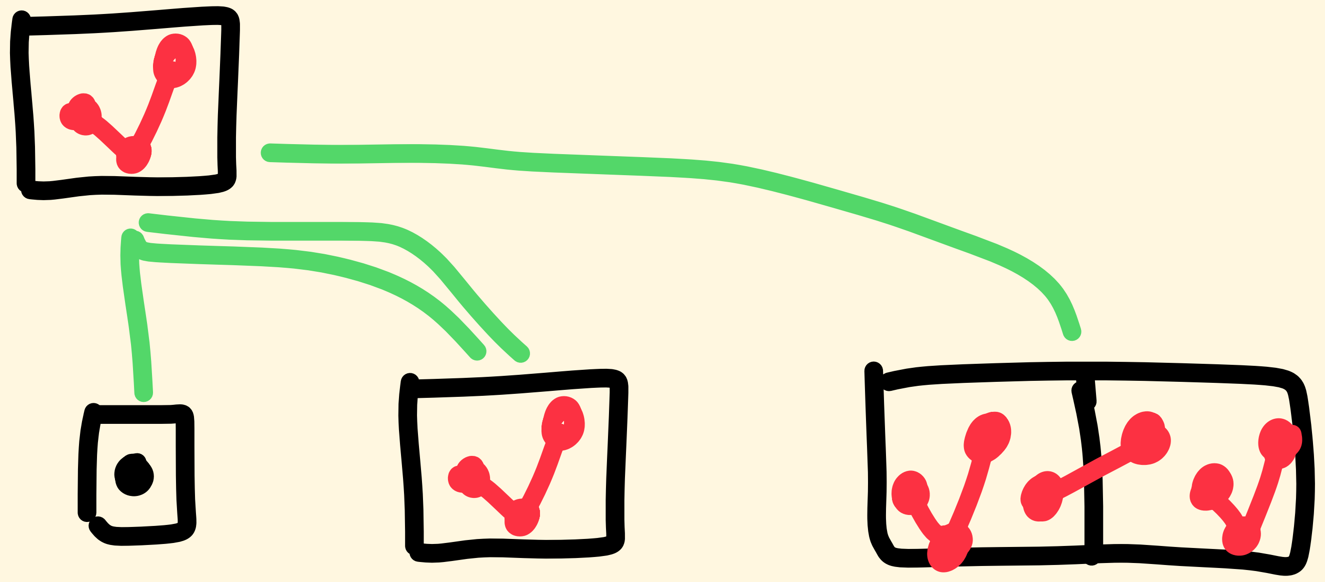
None of this is specific to 1324, and we can do it with any set of forbidden patterns.

Tilings

Av(213)

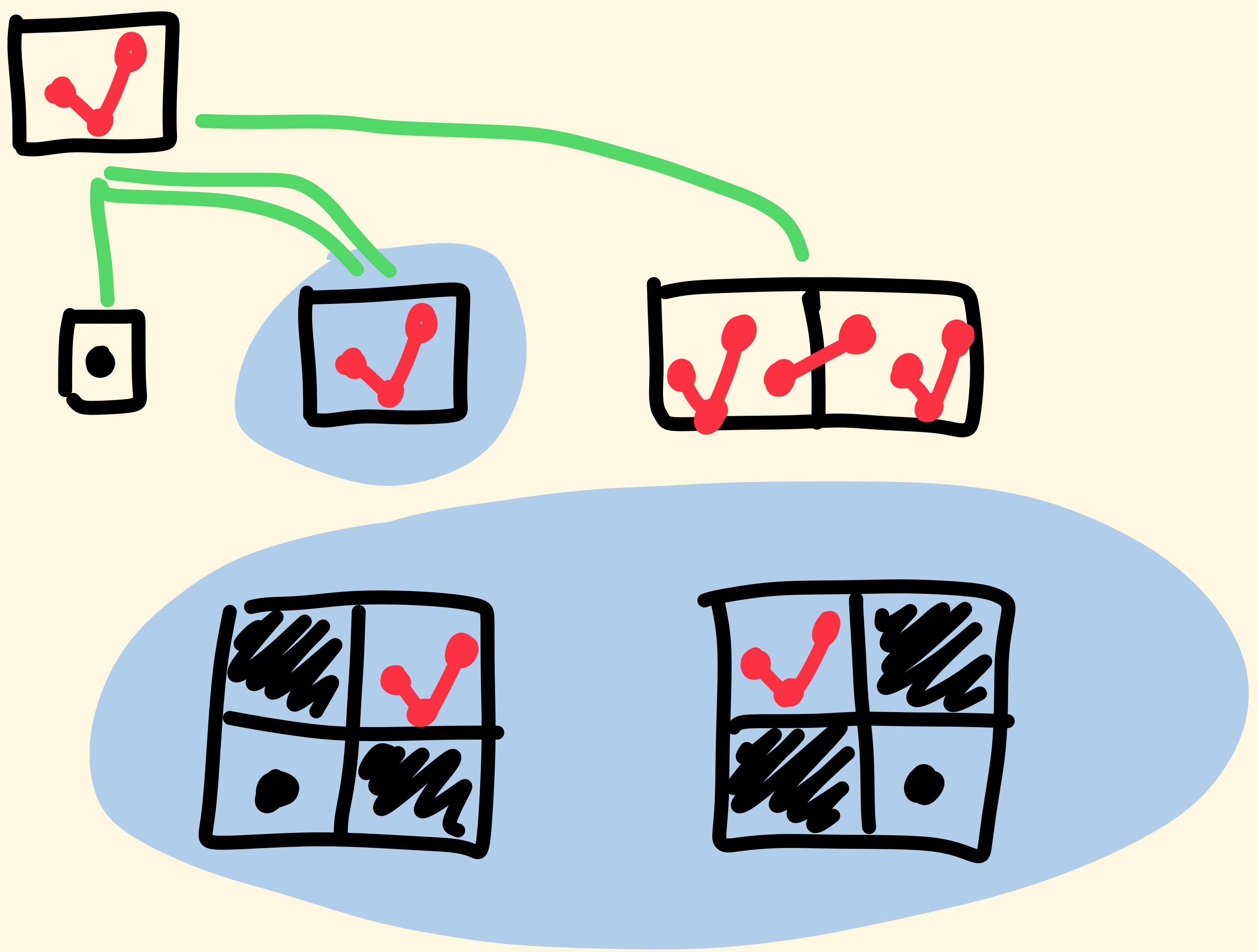
Tilings

Av(213)



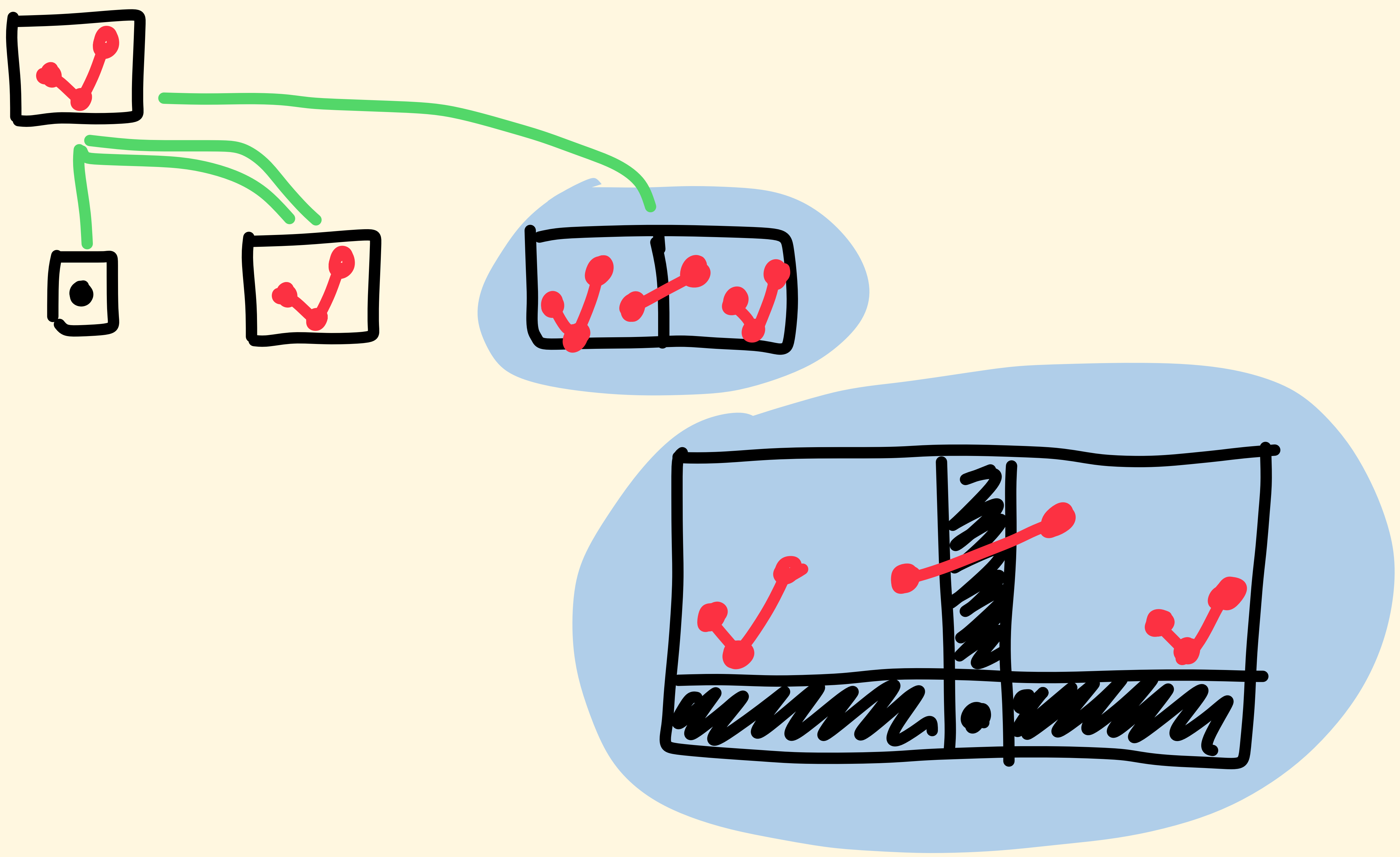
Tilings

Av(213)



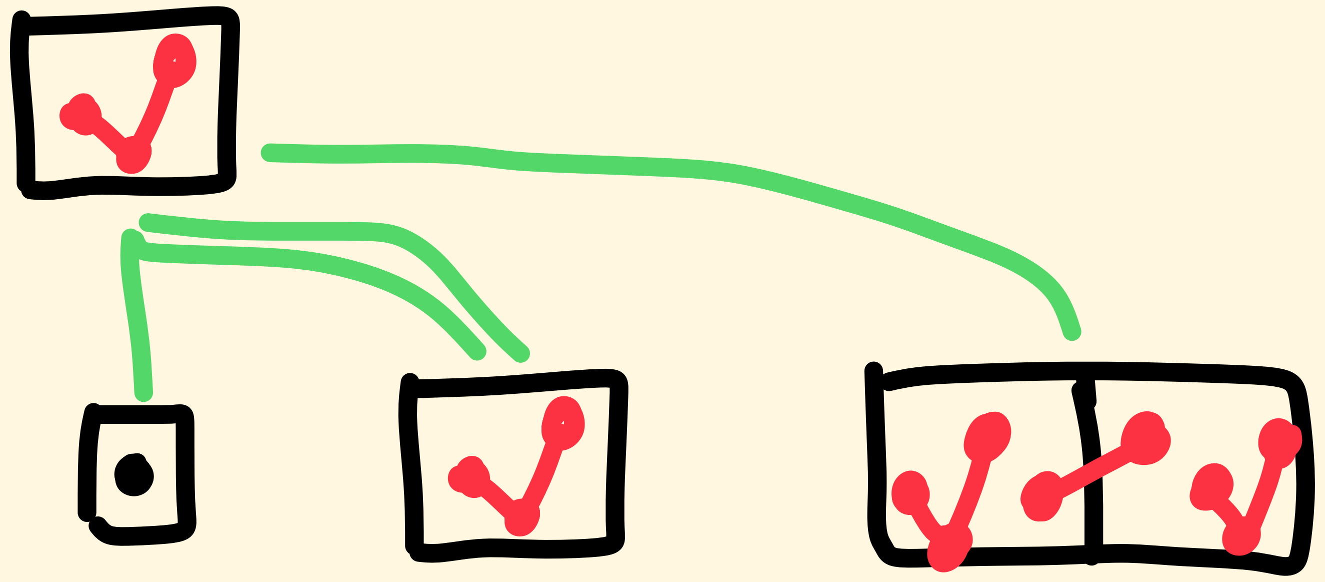
Tilings

Av(213)



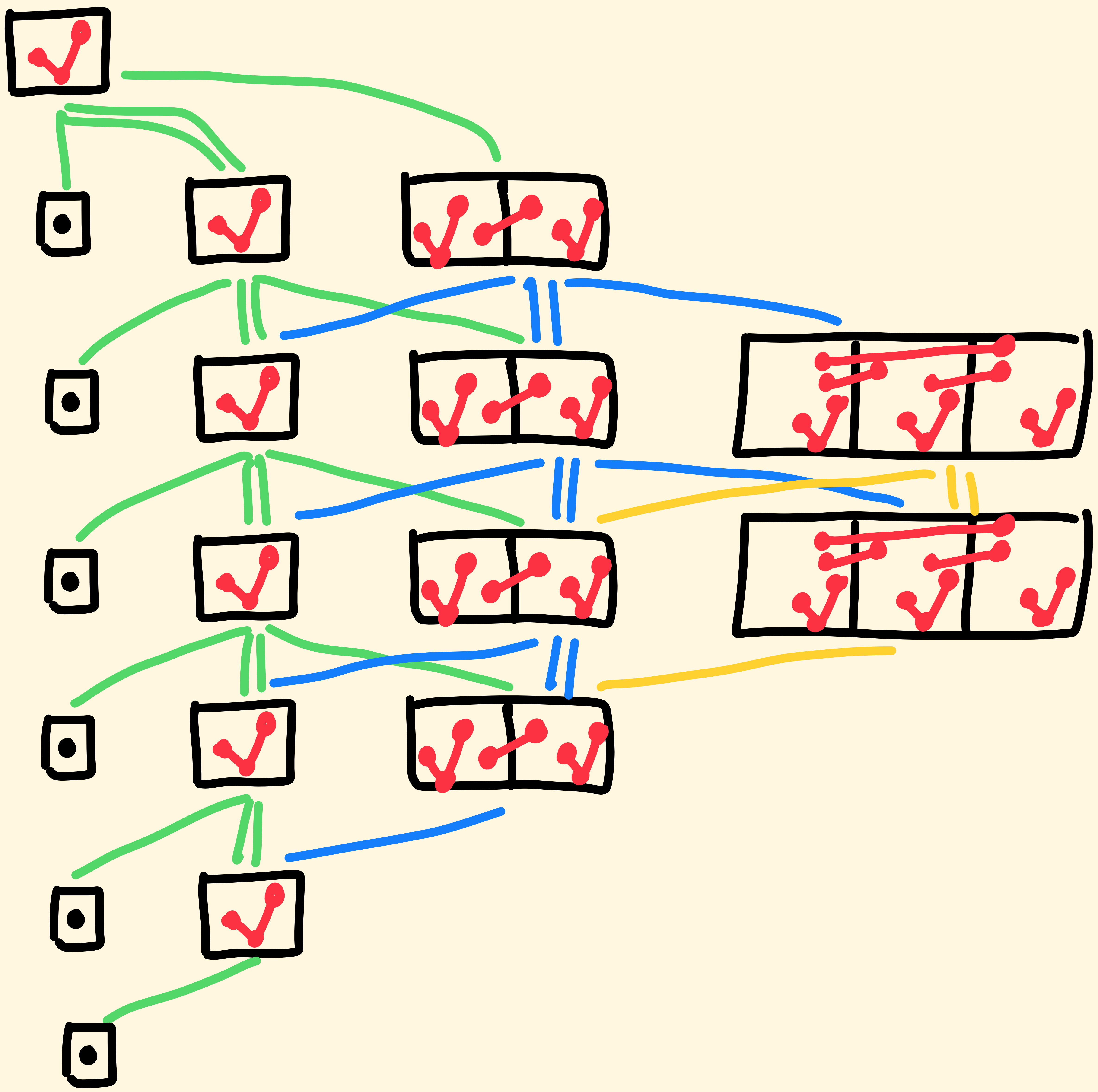
Tilings

Av(213)



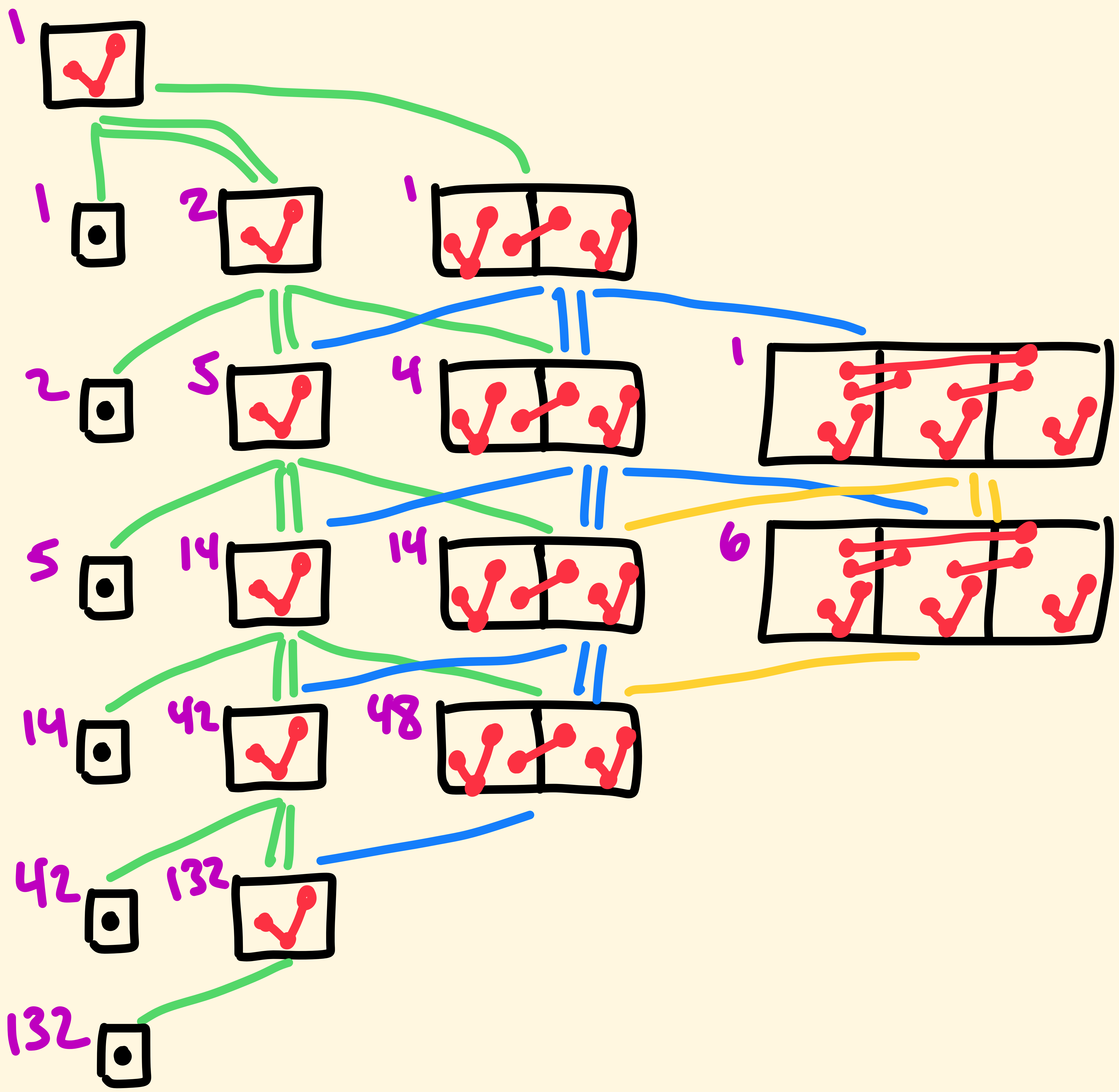
Tilings

Av(213)



Tilings

Av(213)



Results

Very preliminary — still improving the parallel implementation

Definitely does not beat 50 terms of Av(1324)!

> Since this is general purpose, it doesn't "know" a structural theorem like the link patterns ahead of time.

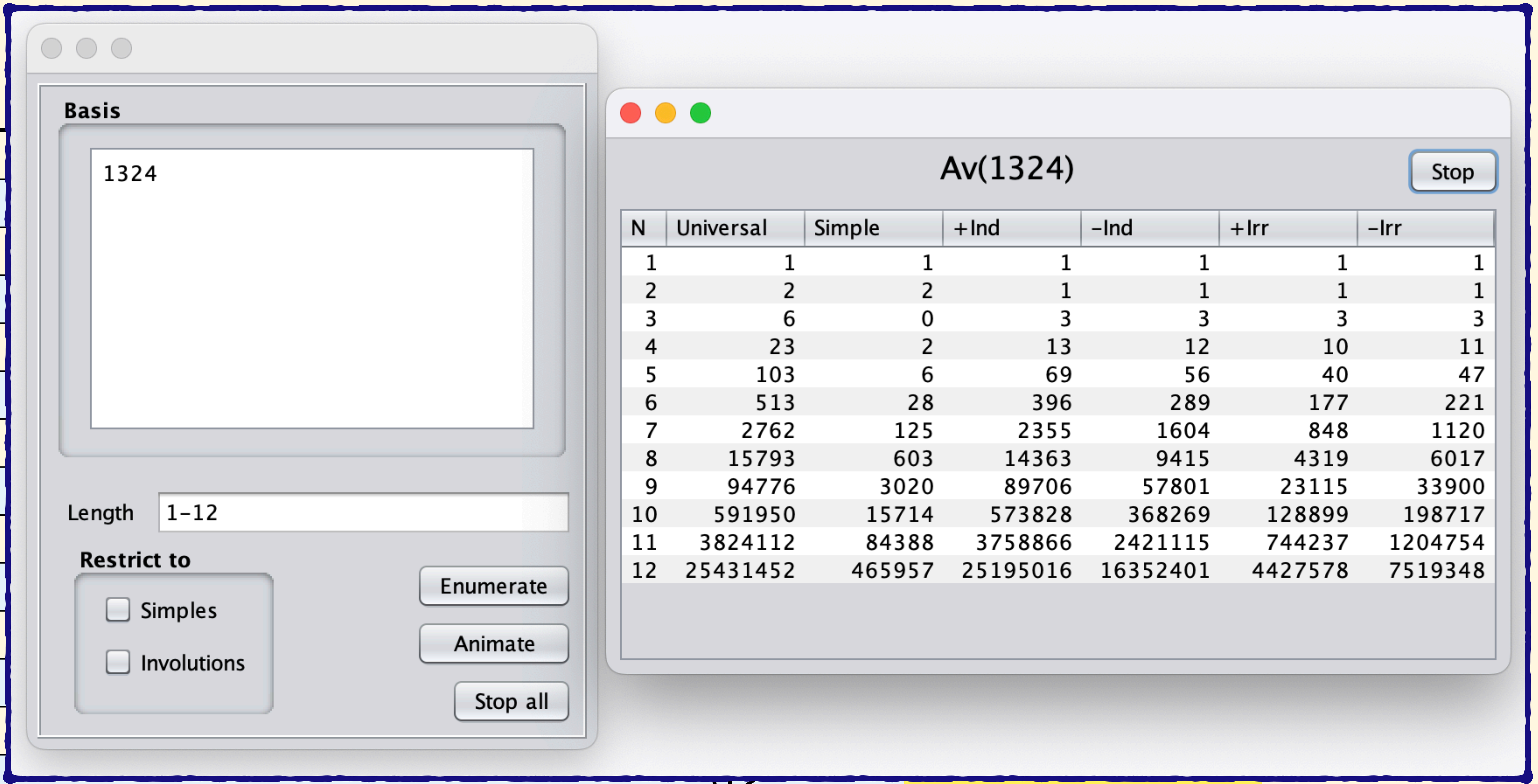
But, I can get to 30s on my laptop and into the 40s on a larger machine.

Time to compute Av(1324) in seconds

Length	Permlab	Permuta	Kuszmaul	Inoue	this work
10	0.53	1.20	0.01	0.02	0.28
11	1.13	8.23	0.05	0.02	0.41
12	5.69	58.40	0.21	0.03	0.63
13	36.5		1.32	0.09	0.91
14	270		8.80	0.20	1.44
15			61.6	0.41	2.11
16			438.5	0.95	3.39
17				2.54	5.43
18				6.72	10.3
19				16.6	17.6
20				40.2	34.9
21				93.8	61.6
22				218	124
23				512 ~26GB memory	255
24				938 ~50GB memory	526 ~1GB memory
25				2539 ~75GB memory	941 ~2GB memory
26					1857 ~3GB memory

Time to compute Av(1324) in seconds

Length	Permlab		
10	0.53		
11	1.13		
12	5.69		
13	36.5		
14	270		
15			
16			
17			
18			
19			
20			
21			
22			
23		512 ~26GB memory	255
24		938 ~50GB memory	526 ~1GB memory
25		2539 ~75GB memory	941 ~2GB memory
26			1857 ~3GB memory



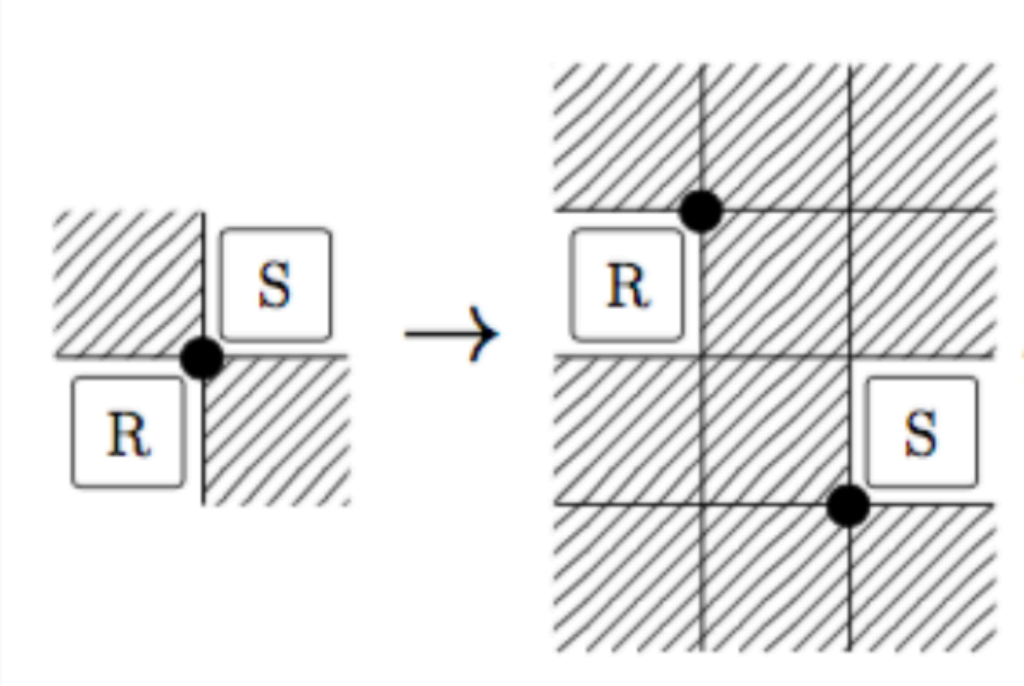
N	Universal	Simple	+Ind	-Ind	+Irr	-Irr
1	1	1	1	1	1	1
2	2	2	1	1	1	1
3	6	0	3	3	3	3
4	23	2	13	12	10	11
5	103	6	69	56	40	47
6	513	28	396	289	177	221
7	2762	125	2355	1604	848	1120
8	15793	603	14363	9415	4319	6017
9	94776	3020	89706	57801	23115	33900
10	591950	15714	573828	368269	128899	198717
11	3824112	84388	3758866	2421115	744237	1204754
12	25431452	465957	25195016	16352401	4427578	7519348

Time to compute Av(1324) in seconds

Length	Permlab	Permuta
10	0.53	1.20
11	1.13	8.23
12	5.69	58.40
13	36.5	
14	270	
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		

Permuta Triangle   About   Blog   People   Papers   Programs   Research Projects   Talks

The study of permutation patterns is a very active area of research and has connections to many other fields of mathematics as well as to computer science and physics. One of the main questions in the field is the enumeration problem: Given a particular set of permutations, how many permutations does the set have of each length? The main goal of this research group is to develop a novel algorithm which will aid researchers in finding structures in sets of permutations and use those structures to find generating functions to enumerate the set. Our research interests lead also into various topics in discrete mathematics and computer science.

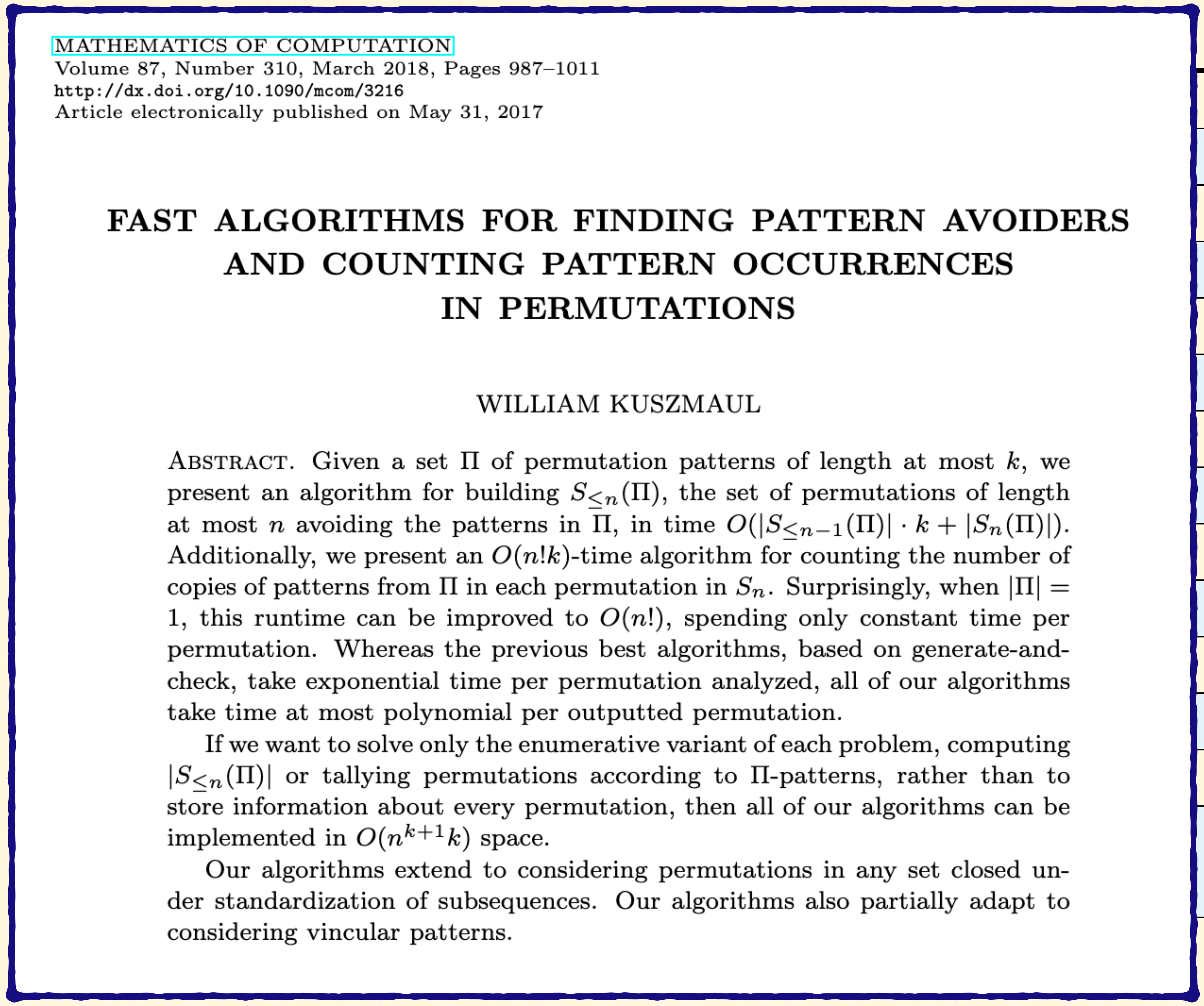


Members

- Michael Albert, Professor, Otago University
- Christian Bean, Lecturer, Keele University
- Anders Claesson, Professor, University of Iceland
- Jay Pantone, Assistant Professor, Marquette University
- Henning Ulfarsson, Assistant Professor, Reykjavik University

1857
~3GB memory

Time to compute Av(1324) in seconds



MATHEMATICS OF COMPUTATION
Volume 87, Number 310, March 2018, Pages 987–1011
http://dx.doi.org/10.1090/mcom/3216
Article electronically published on May 31, 2017

FAST ALGORITHMS FOR FINDING PATTERN AVOIDERS AND COUNTING PATTERN OCCURRENCES IN PERMUTATIONS

WILLIAM KUSZMAUL

ABSTRACT. Given a set Π of permutation patterns of length at most k, we present an algorithm for building $S_{\leq n}(\Pi)$, the set of permutations of length at most n avoiding the patterns in Π, in time $O(|S_{\leq n-1}(\Pi)| \cdot k + |S_n(\Pi)|)$. Additionally, we present an $O(n!k)$-time algorithm for counting the number of copies of patterns from Π in each permutation in S_n. Surprisingly, when $|\Pi| = 1$, this runtime can be improved to $O(n!)$, spending only constant time per permutation. Whereas the previous best algorithms, based on generate-and-check, take exponential time per permutation analyzed, all of our algorithms take time at most polynomial per outputted permutation.

If we want to solve only the enumerative variant of each problem, computing $|S_{\leq n}(\Pi)|$ or tallying permutations according to Π-patterns, rather than store information about every permutation, then all of our algorithms can be implemented in $O(n^{k+1}k)$ space.

Our algorithms extend to considering permutations in any set closed under standardization of subsequences. Our algorithms also partially adapt to considering vincular patterns.

	Kuszmaul	Inoue	this work
	0.01	0.02	0.28
	0.05	0.02	0.41
	0.21	0.03	0.63
	1.32	0.09	0.91
	8.80	0.20	1.44
	61.6	0.41	2.11
	438.5	0.95	3.39
		2.54	5.43
		6.72	10.3
		16.6	17.6
		40.2	34.9
		93.8	61.6
		218	124
		512 ~26GB memory	255
		938 ~50GB memory	526 ~1GB memory
25		2539 ~75GB memory	941 ~2GB memory
26			1857 ~3GB memory

Time to compute Av(1324) in seconds

Implicit Generation of Pattern-Avoiding Permutations Based on πDDs

Yuma Inoue
Division of Computer Science
Graduate School of Info. Sci. and Tech.
Hokkaido University
Sapporo 060-0814, Japan

Takahisa Toda
JST ERATO Minato Project
Graduate School of Info. Sci. and Tech.
Hokkaido University
Sapporo 060-0814, Japan

Shin-ichi Minato*
Division of Computer Science
Graduate School of Info. Sci. and Tech.
Hokkaido University
Sapporo 060-0814, Japan

September 21, 2013

Abstract

Pattern-avoiding permutations are permutations where none of the subsequences match the relative order of a given pattern. Pattern-avoiding permutations are related to practical and abstract mathematical problems and can provide simple representations for such problems. For example, some *floorplans*, which are used for optimizing very-large-scale integration(VLSI) circuit design, can be encoded into pattern-avoiding permutations. The generation of pattern-avoiding permutations is an important topic in efficient VLSI design and mathematical analysis of patten-avoiding permutations. In this paper, we present an algorithm for generating pattern-avoiding permutations, and extend this algorithm beyond classical patterns to generalized patterns with more restrictions. Our approach is based on the data structure πDDs, which can represent a permutation set compactly and has useful set operations. We demonstrate the efficiency of our algorithm by computational experiments.

Inoue	this work
0.02	0.28
0.02	0.41
0.03	0.63
0.09	0.91
0.20	1.44
0.41	2.11
0.95	3.39
2.54	5.43
6.72	10.3
16.6	17.6
40.2	34.9
93.8	61.6
218	124
512 ~26GB memory	255
938 ~50GB memory	526 ~1GB memory
2539 ~75GB memory	941 ~2GB memory
	1857 ~3GB memory

Time to compute Av(1324) in seconds

Length	Permlab	Permuta	Kuszmaul	Inoue	this work
10	0.53	1.20	0.01	0.02	0.28
11	1.13	8.23	0.05	0.02	0.41
12	5.69	58.40	0.21	0.03	0.63
13	36.5		1.32	0.09	0.91
14	270		8.80	0.20	1.44
15			61.6	0.41	2.11
16			438.5	0.95	3.39
17				2.54	5.43
18				6.72	10.3
19				16.6	17.6
20				40.2	34.9
21				93.8	61.6
22				218	124
23				512 ~26GB memory	255
24				938 ~50GB memory	526 ~1GB memory
25				2539 ~75GB memory	941 ~2GB memory
26					1857 ~3GB memory

Results

Classical Length-5 Pattern-Avoiding Permutations

Nathan Clisby
Department of Mathematics,
Swinburne University of Technology,
Hawthorn, Vic. 3122, Australia
nclisby@swin.edu.au

Andrew R. Conway
Fairfield, Vic. 3078, Australia
andrewpermutations5@greatcactus.org

Anthony J. Guttmann
School of Mathematics and Statistics
The University of Melbourne
Vic. 3010, Australia
guttmann@unimelb.edu.au

Yuma Inoue
Google Japan, SHIBUYA STREAM,
3-21-3 Shibuya, Shibuya-ku,
Tokyo 150-0002, Japan
yumai@google.com

Submitted: Oct 19, 2021; Accepted: Mar 30, 2022; Published: Jul 15, 2022

Abstract

We have made a systematic numerical study of the 16 Wilf classes of length-5 classical pattern-avoiding permutations from their generating function coefficients. We have extended the number of known coefficients in fourteen of the sixteen classes. Careful analysis, including sequence extension, has allowed us to estimate the growth constant of all classes, and in some cases to estimate the sub-dominant power-law term associated with the exponential growth.

There are 120 classes of the form Av(β) where $|\beta| = 5$. They split into 16 different groups based on their counting sequence.

One is already solved, one independently counted up to length 38, and this paper computed the other 14 up to lengths between 23 and 27.

Our method looks like to get most of the 14 up to length 30, some up to 35 or 40.

Efficiency varies a lot between classes. The number of different tilings computed could be exponential, polynomial, even linear.

Bounds on the Growth Rate

In addition to the counting sequences, you can also turn these truncated insertion encoding trees into rigorous lower bounds for the growth rate of the class. (maybe upper bounds too?)

Av(12453):
growth rate is known to be $9 + 4\sqrt{2} \approx 14.6568$
we get a lower bound of 13.3748 by counting up to length 30

Av(41235):
Guttmann estimates the growth rate is ≈ 13.703 using 27 terms
we get a lower bound of 12.1619 by counting up to length 27

Other Avenues

We have adapted this to count pattern-avoiding involutions, and applied it to the patterns 1324 and 4231. Forthcoming paper with Christian Bean and Tony Guttmann.

Christian and I have also adapted it to count pattern-avoiding inversion sequences. You can really do this for any combinatorial object that you can make a tiling-like object for.

Thank you!