IOL & DISCOGA research seminar

Greedy Strategies for Exhaustive Generation

Arturo Merino

The Greedy Gray Code Algorithm

Aaron Williams

Department of Mathematics and Statistics, McGill University
haron@uvic.ca

Abstract. We reinterpret classic Gray codes for binary strings, permutations, combinations, binary trees, and set partitions using a simple greedy algorithm. The algorithm begins with an initial object and an ordered list of operations, and then repeatedly creates a new object by applying the first possible operation to the most recently created object.

1 Introduction

Let $\mathbb{B}(n)$ be the set of n-bit binary strings. The *binary reflected Gray code* **Gray**(n) orders $\mathbb{B}(n)$ so that successive strings have Hamming distance one (i.e., they differ in one bit). For example, the order for $n = 3$ appears below.

The Greedy Gray Code Algorithm

Aaron Williams

Department of Mathematics and Statistics, McGill University
haron@uvic.ca

Abstract. We reinterpret classi
tations, combinations, binary tr
greedy algorithm. The algorithm
ordered list of operations, and t
applying the first possible operat

1 Introduction

Let $\mathbb{B}(n)$ be the set of n-bit binar
Gray(n) orders $\mathbb{B}(n)$ so that succe
(i.e., they differ in one bit). For exa

COMBINATORIAL GENERATION VIA PERMUTATION LANGUAGES. I. FUNDAMENTALS

ELIZABETH HARTUNG, HUNG P. HOANG, TORSTEN MÜTZE, AND AARON WILLIAMS

ABSTRACT. In this work we present a general and versatile algorithmic framework for exhaustively generating a large variety of different combinatorial objects, based on encoding them as permutations. This approach provides a unified view on many known results and allows us to prove many new ones. In particular, we obtain the following four classical Gray codes as special cases: the Steinhaus-Johnson-Trotter algorithm to generate all permutations of an n-element set by adjacent transpositions; the binary reflected Gray code to generate all n-bit strings by flipping a single bit in each step; the Gray code for generating all n-vertex binary trees by rotations due to Lucas, van Baronaigien, and Ruskey; the Gray code for generating all partitions of an n-element ground set by element exchanges due to Kaye.

We present two distinct applications for our new framework: The first main application is the generation of pattern-avoiding permutations, yielding new Gray codes for different families of permutations that are characterized by the avoidance of certain classical patterns, (bi)vincular patterns, barred patterns, boxed patterns, Bruhat-restricted patterns, mesh patterns, monotone and geometric grid classes, and many others. We also obtain new Gray codes for all the combinatorial objects that are in bijection to these permutations, in particular for five different types of geometric rectangulations, also known as floorplans, which are divisions of a square into n rectangles subject to certain restrictions.

The second main application of our framework are lattice congruences of the weak order on the symmetric group S_n. Recently, Pilaud and Santos realized all those lattice congruences as $(n-1)$-dimensional polytopes, called quotientopes, which generalize hypercubes, associahedra, permutahedra etc. Our algorithm generates the equivalence classes of each of those lattice congruences, by producing a Hamilton path on the skeleton of the corresponding quotientope,

Introduction

Exhaustive generation problems

- **Given:** Some combinatorial objects of interest.

Exhaustive generation problems

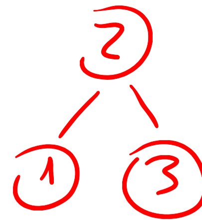
- **Given:** Some combinatorial objects of interest.
  - n-bitstrings. 0000 0110

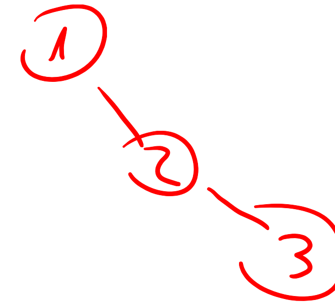
Exhaustive generation problems

- **Given:** Some combinatorial objects of interest.
  - n-bitstrings.
  - permutations of $[n]$ 1234 1324

Exhaustive generation problems

- **Given:** Some combinatorial objects of interest.
  - n-bitstrings.
  - permutations of $[n]$
  - binary trees on n nodes





Exhaustive generation problems

- **Given:** Some combinatorial objects of interest.
  - n-bitstrings.
  - permutations of $[n]$
  - binary trees on n nodes
  - spanning trees of a given graph.

Exhaustive generation problems

- **Given:** Some combinatorial objects of interest.
  - n-bitstrings.
  - permutations of $[n]$
  - binary trees on n nodes
  - spanning trees of a given graph.
- **Objective:** Output each object exactly once.

Exhaustive generation problems

- **Given:** Some combinatorial objects of interest.
  - n-bitstrings.
  - permutations of $[n]$
  - binary trees on n nodes
  - spanning trees of a given graph.
- **Objective:** Output each object exactly once.
  - fundamental algorithmic task.

Exhaustive generation problems

- **Given:** Some combinatorial objects of interest.
  - n-bitstrings.
  - permutations of $[n]$
  - binary trees on n nodes
  - spanning trees of a given graph.
- **Objective:** Output each object exactly once.
  - fundamental algorithmic task.
  - key component in many algorithms.

Exhaustive generation problems

- **Given:** Some combinatorial objects of interest.
  - n-bitstrings.
  - permutations of $[n]$
  - binary trees on n nodes
  - spanning trees of a given graph.
- **Objective:** Output each object exactly once.
  - fundamental algorithmic task.
  - key component in many algorithms.
- **Dream:** $\mathcal{O}(1)$ time between generated objects.

Exhaustive generation problems

- **Given:** Some combinatorial objects of interest.
  - n-bitstrings.
  - permutations of $[n]$
  - binary trees on n nodes
  - spanning trees of a given graph.
- **Objective:** Output each object exactly once.
  - fundamental algorithmic task.
  - key component in many algorithms.
- **Dream:** $\mathcal{O}(1)$ time between generated objects.

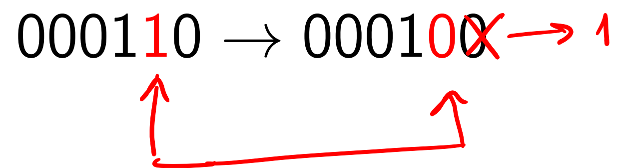
delay

Gray codes

- Listing of the objects such that consecutives ones differ in **local change**.

Gray codes

- Listing of the objects such that consecutives ones differ in **local change**.
    - n-bitstrings by bitflips [Gray 53]

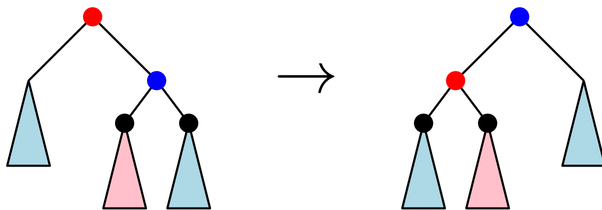


Gray codes

- Listing of the objects such that consecutives ones differ in **local change**.
  - n-bitstrings by bitflips [Gray 53]

    $000110 \rightarrow 000100$

  - binary trees on n nodes by rotation [Lucas, Roelants van Baronaigien, Ruskey 93]

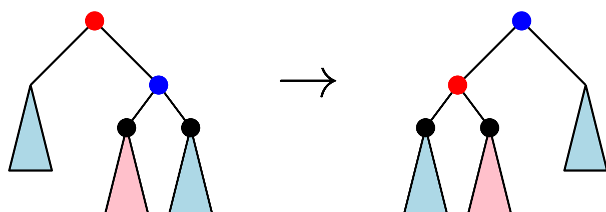


Gray codes

- Listing of the objects such that consecutives ones differ in **local change**.
  - n-bitstrings by bitflips [Gray 53]

    $000110 \rightarrow 000100$

  - binary trees on n nodes by rotation [Lucas, Roelants van Baronaigien, Ruskey 93]

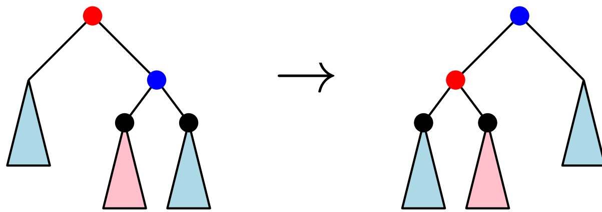
  - permutations of $[n]$ by adjacent transpositions [Steinhaus, Johnson, Trotter 63]

    $32415 \rightarrow 32145$ 23

Gray codes

- Listing of the objects such that consecutives ones differ in **local change**.
  - n-bitstrings by bitflips [Gray 53]

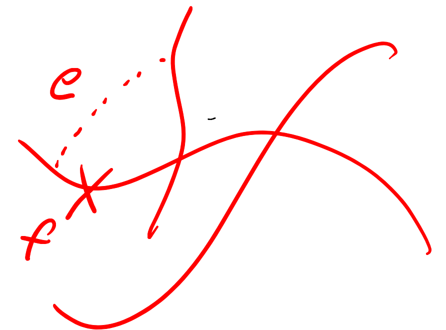
    $000110 \rightarrow 000100$
  - binary trees on n nodes by rotation [Lucas, Roelants van Baronaigien, Ruskey 93]



  - permutations of $[n]$ by adjacent transpositions [Steinhaus, Johnson, Trotter 63]

    $32415 \rightarrow 32145$
  - spanning trees by edge exchanges [Holzmann, Harary 72]

    $T \rightarrow T + e - f$

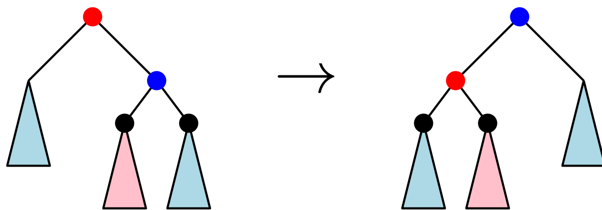


Gray codes

- Listing of the objects such that consecutives ones differ in **local change**.
  - n-bitstrings by bitflips [Gray 53]

    $000110 \to 000100$

  - binary trees on n nodes by rotation [Lucas, Roelants van Baronaigien, Ruskey 93]

  - permutations of $[n]$ by adjacent transpositions [Steinhaus, Johnson, Trotter 63]

    $32415 \to 32145$

  - spanning trees by edge exchanges [Holzmann, Harary 72]

    $T \to T + e - f$

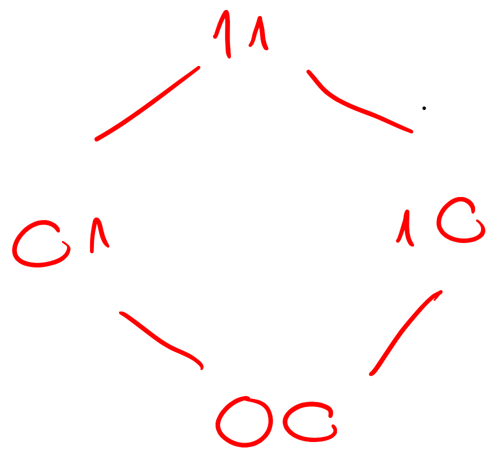
- A first step towards efficient generation.

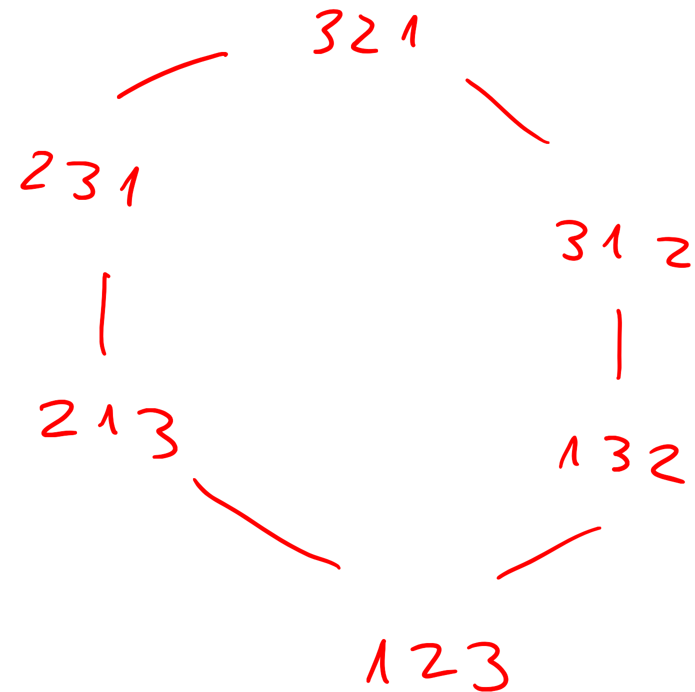
Flip graphs

- Graph where V =combinatorial objects, E =local change.

Flip graphs

- Graph where V =combinatorial objects, E =local change.
- **Examples:**





Flip graphs

- Graph where V =combinatorial objects, E =local change.
- **Examples:**

Flip graph has Hamilton ~~cycle~~ path $\iff$ Existance of Gray code

Lack of common strategies

- Other fundamental tasks on combinatorial objects have very powerfull and well understood techniques:

Lack of common strategies

- Other fundamental tasks on combinatorial objects have very powerfull and well understood techniques:
  - Counting $\leftarrow$ generating functions

Lack of common strategies

- Other fundamental tasks on combinatorial objects have very powerfull and well understood techniques:
  - Counting $\leftarrow$ generating functions
  - Random sampling $\leftarrow$ markov chains

Lack of common strategies

- Other fundamental tasks on combinatorial objects have very powerfull and well understood techniques:
  - Counting $\leftarrow$ generating functions
  - Random sampling $\leftarrow$ markov chains
  - Optimization $\leftarrow$ linear programming

Lack of common strategies

- Other fundamental tasks on combinatorial objects have very powerfull and well understood techniques:
  - Counting $\leftarrow$ generating functions
  - Random sampling $\leftarrow$ markov chains
  - Optimization $\leftarrow$ linear programming
  - Exhaustive generation $\leftarrow$???

Lack of common strategies

- Other fundamental tasks on combinatorial objects have very powerfull and well understood techniques:
  - Counting $\leftarrow$ generating functions
  - Random sampling $\leftarrow$ markov chains
  - Optimization $\leftarrow$ linear programming
  - Exhaustive generation $\leftarrow$???
- Many tailormade algorithms, few general approaches.

Lack of common strategies

- Other fundamental tasks on combinatorial objects have very powerfull and well understood techniques:
  - Counting $\leftarrow$ generating functions
  - Random sampling $\leftarrow$ markov chains
  - Optimization $\leftarrow$ linear programming
  - Exhaustive generation $\leftarrow$???
- Many tailormade algorithms, few general approaches.
  - Reverse-search **[Avis, Fukuda 96]**

Lack of common strategies

- Other fundamental tasks on combinatorial objects have very powerfull and well understood techniques:
  - Counting $\leftarrow$ generating functions
  - Random sampling $\leftarrow$ markov chains
  - Optimization $\leftarrow$ linear programming
  - Exhaustive generation $\leftarrow$???
- Many tailormade algorithms, few general approaches.
  - Reverse-search [Avis, Fukuda 96]
  - ECO framework [Barcucci, del Lungo, Pergola, Pinzani 99]

Lack of common strategies

- Other fundamental tasks on combinatorial objects have very powerfull and well understood techniques:
  - Counting $\leftarrow$ generating functions
  - Random sampling $\leftarrow$ markov chains
  - Optimization $\leftarrow$ linear programming
  - Exhaustive generation $\leftarrow$???
- Many tailormade algorithms, few general approaches.
  - Reverse-search [Avis, Fukuda 96]
  - ECO framework [Barcucci, del Lungo, Pergola, Pinzani 99]
  - Reflectable-languages [Li, Sawada 09]

Lack of common strategies

- Other fundamental tasks on combinatorial objects have very powerfull and well understood techniques:
  - Counting $\leftarrow$ generating functions
  - Random sampling $\leftarrow$ markov chains
  - Optimization $\leftarrow$ linear programming
  - Exhaustive generation $\leftarrow$???
- Many tailormade algorithms, few general approaches.
  - Reverse-search [Avis, Fukuda 96]
  - ECO framework [Barcucci, del Lungo, Pergola, Pinzani 99]
  - Reflectable-languages [Li, Sawada 09]
  - Bubble-languages [Ruskey, Sawada, Williams 12]

Lack of common strategies

- Other fundamental tasks on combinatorial objects have very powerfull and well understood techniques:
  - Counting $\leftarrow$ generating functions
  - Random sampling $\leftarrow$ markov chains
  - Optimization $\leftarrow$ linear programming
  - Exhaustive generation $\leftarrow$???
- Many tailormade algorithms, few general approaches.
  - Reverse-search [Avis, Fukuda 96]
  - ECO framework [Barcucci, del Lungo, Pergola, Pinzani 99]
  - Reflectable-languages [Li, Sawada 09]
  - Bubble-languages [Ruskey, Sawada, Williams 12]
- **This talk:** the Greedy approach.

Greedy Gray codes

What is a greedy Gray code?

- **Greedy approach:** Perform *locally optimal* move. Don't worry too much about the future.

What is a greedy Gray code?

- **Greedy approach:** Perform *locally optimal* move. Don't worry too much about the future.
- **Gray codes:** [Williams 13]

What is a greedy Gray code?

- **Greedy approach:** Perform *locally optimal* move. Don't worry too much about the future.
- **Gray codes:** [Williams 13]
  - Rank the flips.

What is a greedy Gray code?

- **Greedy approach:** Perform *locally optimal* move. Don't worry too much about the future.
- **Gray codes:** [Williams 13]
  - Rank the flips.
  - **Repeat:** Perform the highest ranked flip such that a new object is generated.

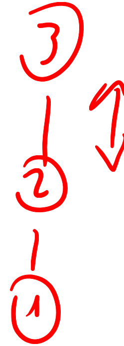
Examples

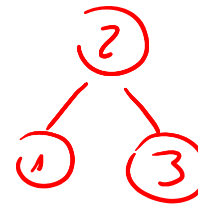
- n-bitstrings by bitflips.

0 0 0
0 0 1
0 1 1
0 1 0
1 1 0
1 1 1
1 0 1
1 0 0

Examples

- n-bitstrings by bitflips.
- binary trees on n nodes by rotation.





Examples

- n-bitstrings by bitflips.
- binary trees on n nodes by rotation.
- permutations of $[n]$ by transpositions.

1 2 3 4 5
2 1 3 4 5
3 1 2 4 5
1 3 2 4 5

Examples

- n-bitstrings by bitflips.
- binary trees on n nodes by rotation.
- permutations of $[n]$ by transpositions.
- permutations of $[n]$ by adjacent transpositions.

1 2 3 4 5

1 2 3 5 4

1 2 5 3 4

1 5 2 3 4

5 1 2 3 4

5 1 2 4 3

Examples

- n-bitstrings by bitflips.
- binary trees on n nodes by rotation.
- permutations of $[n]$ by transpositions.
- permutations of $[n]$ by adjacent transpositions.
- permutations of $[n]$ by prefix-reversal.

Examples

- n-bitstrings by bitflips.
- binary trees on n nodes by rotation.
- permutations of $[n]$ by transpositions.
- permutations of $[n]$ by adjacent transpositions.
- permutations of $[n]$ by prefix-reversal.
- n pancakes with a burnt side by flips.

Examples

- n-bitstrings by bitflips. BRGC
- binary trees on n nodes by rotation.
- permutations of $[n]$ by transpositions.
- permutations of $[n]$ by adjacent transpositions.
- permutations of $[n]$ by prefix-reversal.
- n pancakes with a burnt side by flips.

Not the traditional viewpoint.

A bit of discussion

- Easy algorithms conceptually. ✓

A bit of discussion

- Easy algorithms conceptually. ✓
- Easy algorithms to implement. ✓

A bit of discussion

- Easy algorithms conceptually. ✓
- Easy algorithms to implement. ✓
- **By default:** Very computationally inefficient ✗

A bit of discussion

- Easy algorithms conceptually. ✓
- Easy algorithms to implement. ✓
- **By default:** Very computationally inefficient ✗
- Why do they work?

A bit of discussion

- Easy algorithms conceptually. ✓
- Easy algorithms to implement. ✓
- **By default:** Very computationally inefficient ✗
- Why do they work?
  - cf. the theory of matroids in optimization.

Zig-zag framework

Introduction

- Zig-zag framework **[Hartung, Hoang, Mütze, Williams 20]**

Introduction

- Zig-zag framework [Hartung, Hoang, Mütze, Williams 20]
  - Pattern avoiding permutations.

Introduction

- Zig-zag framework [Hartung, Hoang, Mütze, Williams 20]
  - Pattern avoiding permutations.
  - Quotientopes [Hoang, Mütze 20]

Introduction

- Zig-zag framework [Hartung, Hoang, Mütze, Williams 20]
  - Pattern avoiding permutations.
  - Quotientopes [Hoang, Mütze 20]
  - Pattern-avoiding rectangulations [M, Mütze 20]

Introduction

- Zig-zag framework [Hartung, Hoang, Mütze, Williams 20]
  - Pattern avoiding permutations.
  - Quotientopes [Hoang, Mütze 20]
  - Pattern-avoiding rectangulations [M, Mütze 20]
  - Graph associahedra [Cardinal, M, Mütze 21+]

Introduction

- Zig-zag framework [Hartung, Hoang, Mütze, Williams 20]
  - Pattern avoiding permutations.
  - Quotientopes [Hoang, Mütze 20]
  - Pattern-avoiding rectangulations [M, Mütze 20]
  - Graph associahedra [Cardinal, M, Mütze 21+]

- C++ implementation available on the combinatorial object server: `www.combos.org`

Introduction

- Zig-zag framework [Hartung, Hoang, Mütze, Williams 20]
  - Pattern avoiding permutations.
  - Quotientopes [Hoang, Mütze 20]
  - Pattern-avoiding rectangulations [M, Mütze 20]
  - Graph associahedra [Cardinal, M, Mütze 21+]
- C++ implementation available on the combinatorial object server: `www.combos.org`
- **Main idea:** Encode combinatorial objects as a set $F_n \subseteq S_n$ of permutations of length n.

Introduction

- Zig-zag framework [Hartung, Hoang, Mütze, Williams 20]
  - Pattern avoiding permutations.
  - Quotientopes [Hoang, Mütze 20]
  - Pattern-avoiding rectangulations [M, Mütze 20]
  - Graph associahedra [Cardinal, M, Mütze 21+]
- C++ implementation available on the combinatorial object server: `www.combos.org`
- **Main idea:** Encode combinatorial objects as a set $F_n \subseteq S_n$ of permutations of length n.
- **Beyond permutations:** Just be inspired by them.

A simple algorithm

- **Start** with the identity object.

A simple algorithm

- **Start** with the identity object.
- **Repeat:** perform the flip such that:

A simple algorithm

- **Start** with the identity object.
- **Repeat:** perform the flip such that:
  - it generates a new object from the class, and

A simple algorithm

- **Start** with the identity object.
- **Repeat:** perform the flip such that:
  - it generates a new object from the class, and
  - it flips the elements with highest *value*

A simple algorithm

- **Start** with the identity object.
- **Repeat:** perform the flip such that:
  - it generates a new object from the class, and
  - it flips the elements with highest *value*
- What is the difference with the generic greedy?

A simple algorithm

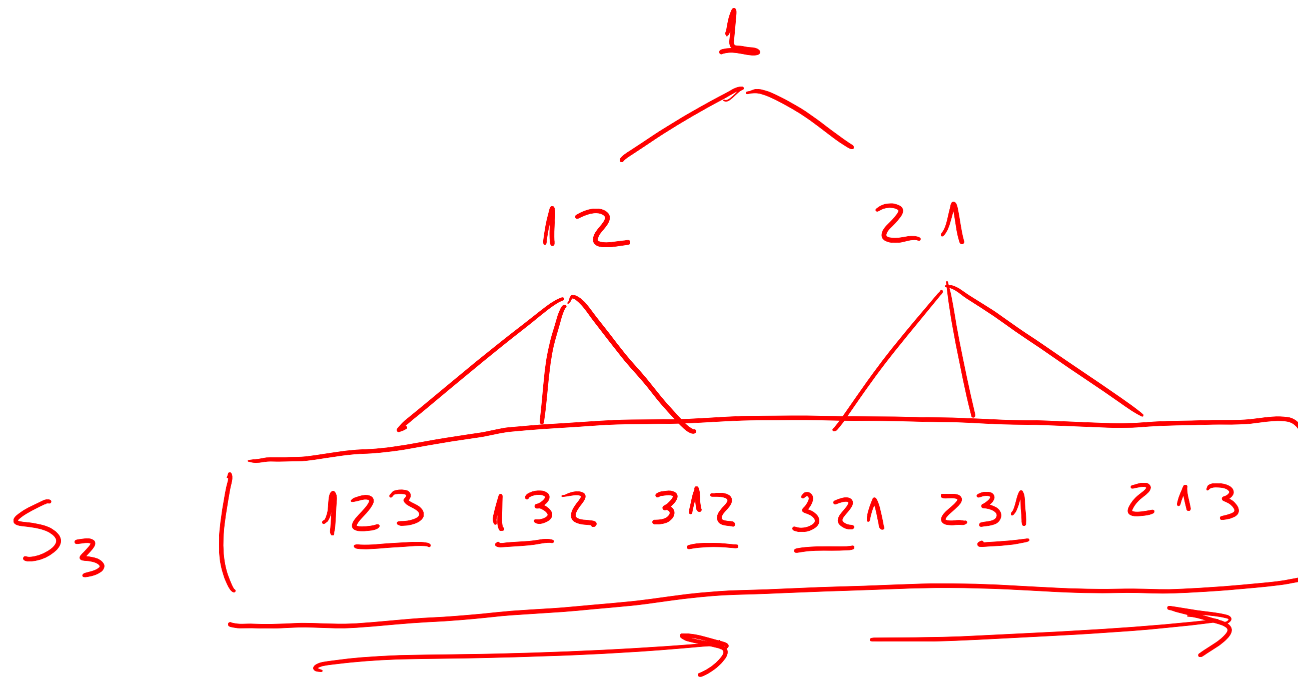
- **Start** with the identity object.
- **Repeat:** perform the flip such that:
    - it generates a new object from the class, and
    - it flips the elements with highest *value*

- What is the difference with the generic greedy?
- **Reminder:** SJT

1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3

Tree of permutations for SJT

Idea: Build a recursive generation tree



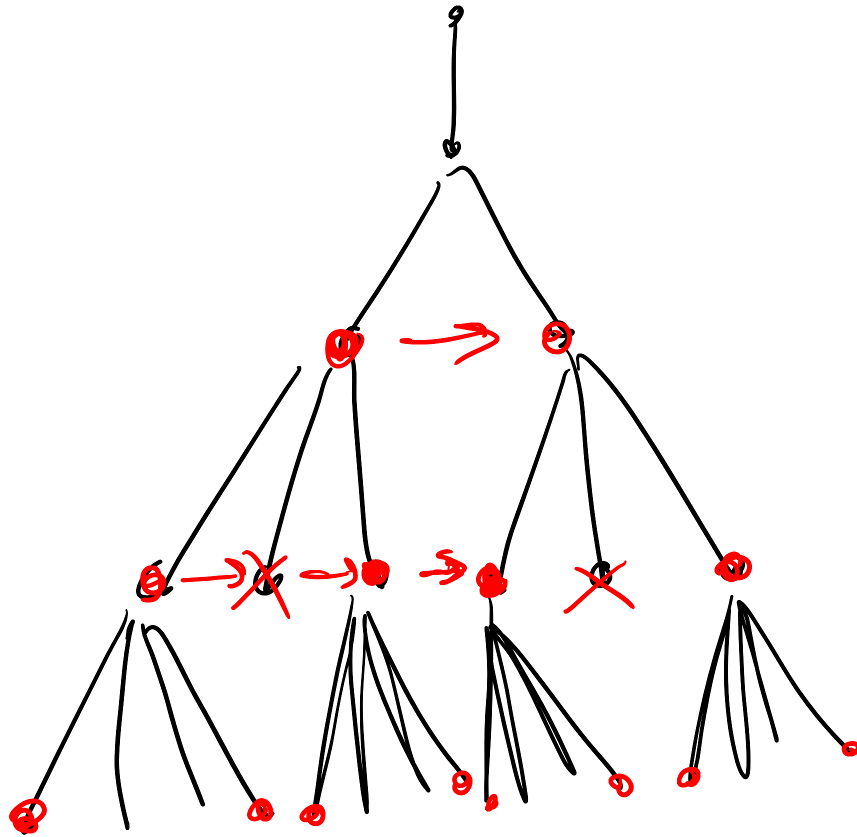
Tree of permutations for SJT

Key insight: The tree of permutations and the zig-zag algorithm produce the same listings.

Proof: Double induction

Depth + left to right.

Pruning



Zig-zag languages

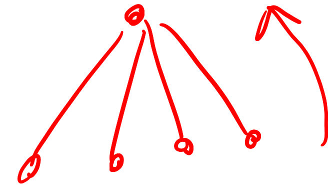
- When does this work?

Zig-zag languages

- When does this work?
- A: Zig-zag conditions

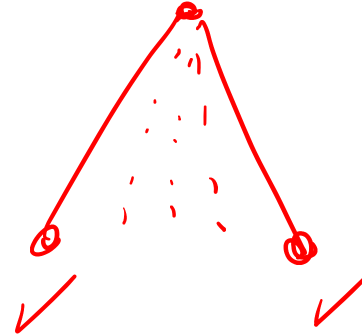
Zig-zag languages

- When does this work?
- A: Zig-zag conditions
  - closed under deletion of n



Zig-zag languages

- When does this work?
- A: Zig-zag conditions
  - closed under deletion of n
  - If $\pi \in \mathcal{F}$ then $n\pi, \pi n \in \mathcal{F}$



Zig-zag languages

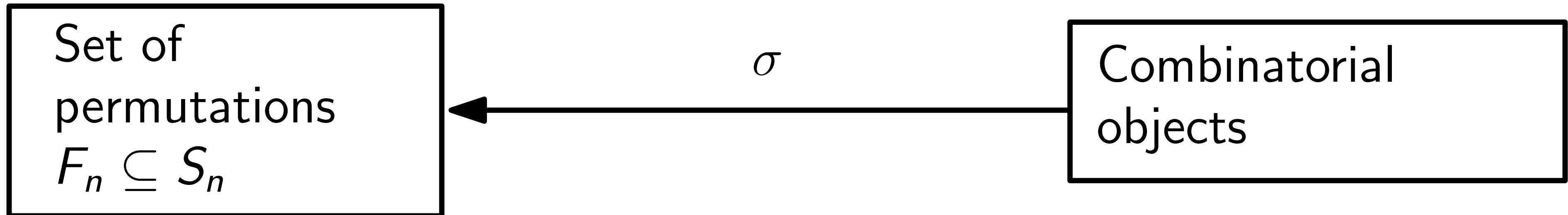
- When does this work?
- A: Zig-zag conditions
  - closed under deletion of n
  - If $\pi \in \mathcal{F}$ then $n\pi, \pi n \in \mathcal{F}$

Very compatible with pattern-avoidance

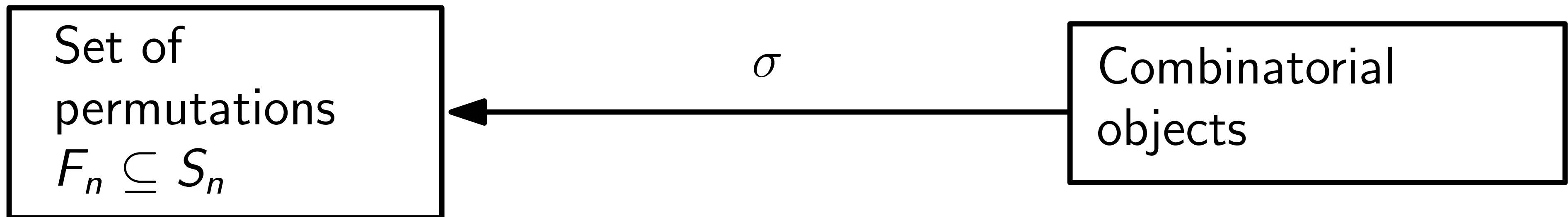
General approach

Combinatorial objects

General approach



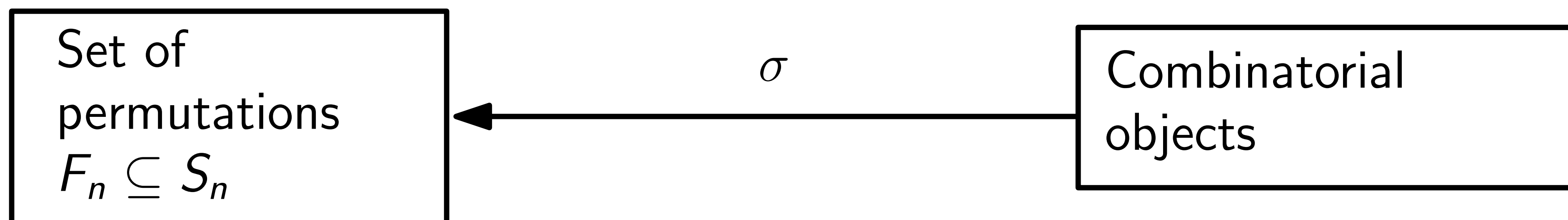
General approach



- run zig-zag algorithm

List $=$ **ZZ**(F_n) $\longrightarrow$ σ^{-1}(**List**)

General approach



- run zig-zag algorithm

$$\textbf{List} = \textbf{ZZ}(F_n) \longrightarrow \sigma^{-1}(\textbf{List})$$

- interpret zig-zag algorithm under the bijection

$$\textbf{ZZ} \longrightarrow \sigma^{-1}(\textbf{ZZ})$$

Towards efficiency: Memoryless

- **Repeat:** Perform a flip such that:
  - a new permutation from the family is generated, and
  - it flips the largest value.

Towards efficiency: Memoryless

- **Repeat:** Perform a flip such that:
  - a new permutation from the family is generated, and
  - it flips the largest value.

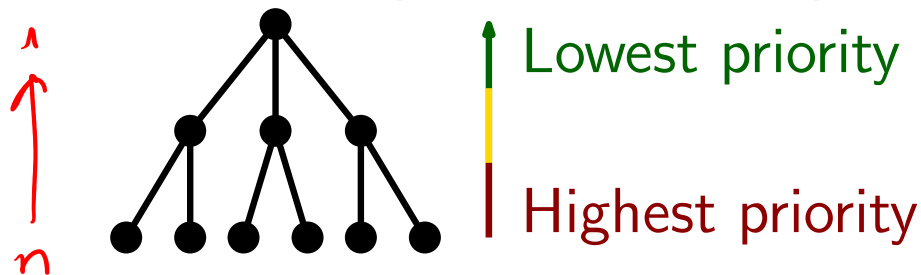
Towards efficiency: Memoryless

- **Repeat:** Perform a flip such that:
  - a new permutation from the family is generated, and
  - it flips the largest value.
- Naive implementation is very inefficient.

Towards efficiency: Memoryless

- **Repeat:** Perform a flip such that:
    - a new permutation from the family is generated, and
    - it flips the largest value.

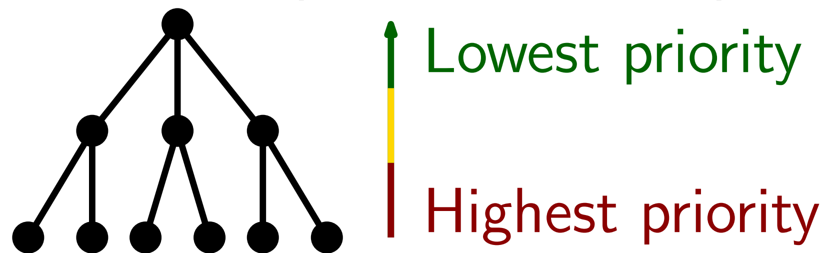
- Naive implementation is very inefficient.
- Permutations are processed in a "priority-queue" fashion.



Towards efficiency: Memoryless

- **Repeat:** Perform a flip such that:
  - a new permutation from the family is generated, and
  - it flips the largest value.

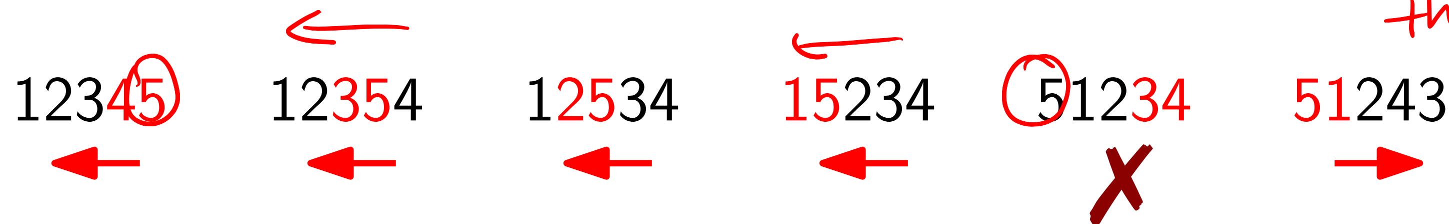
- Naive implementation is very inefficient.
- Permutations are processed in a "priority-queue" fashion.



who is moving?

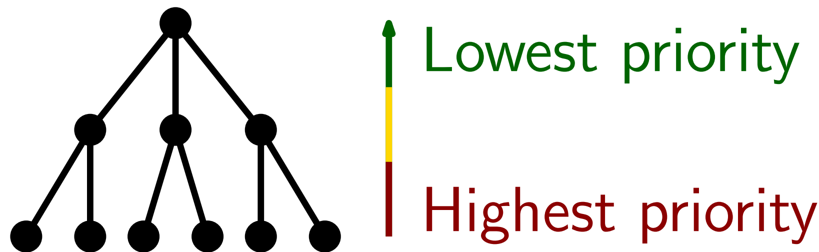
- Elements change direction when they cannot be moved.

where are they moving?

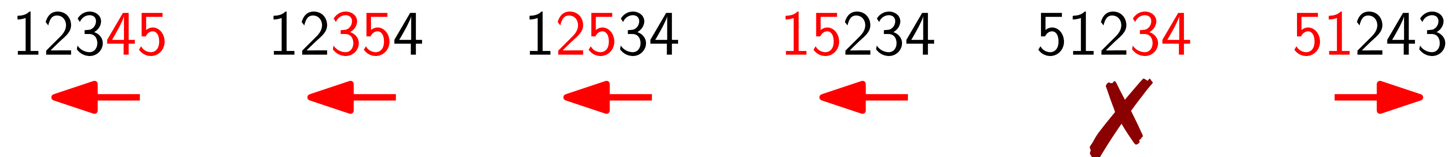


Towards efficiency: Memoryless

- **Repeat:** Perform a flip such that:
  - a new permutation from the family is generated, and
  - it flips the largest value.
- Naive implementation is very inefficient.
- Permutations are processed in a "priority-queue" fashion.



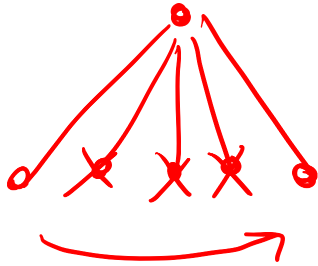
- Elements change direction when they cannot be moved.



- Memoryless version of the algorithm!

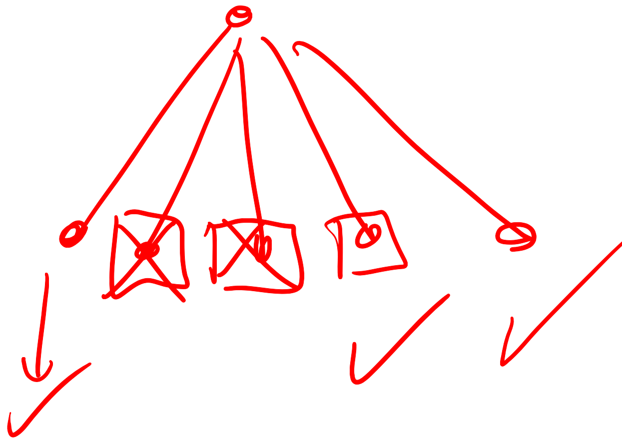
Towards efficiency: membership testing

- We only need to detect pruned branches.



Towards efficiency: membership testing

- We only need to detect pruned branches.
- *Difference* membership testing.



Without permutations: another view-point

- Try to *directly* construct a tree of the objects.

Without permutations: another view-point

- Try to *directly* construct a tree of the objects.

- *Spiritual* zig-zag condition.

Without permutations: another view-point

- Try to *directly* construct a tree of the objects.

- *Spiritual* zig-zag condition.
  - closed under deletion of the largest element.

Without permutations: another view-point

- Try to *directly* construct a tree of the objects.

- *Spiritual* zig-zag condition.
    - closed under deletion of the largest element.
    - closed under insertion at the beginning and at the end.

Examples: BRGC and Binary trees

BRGC:

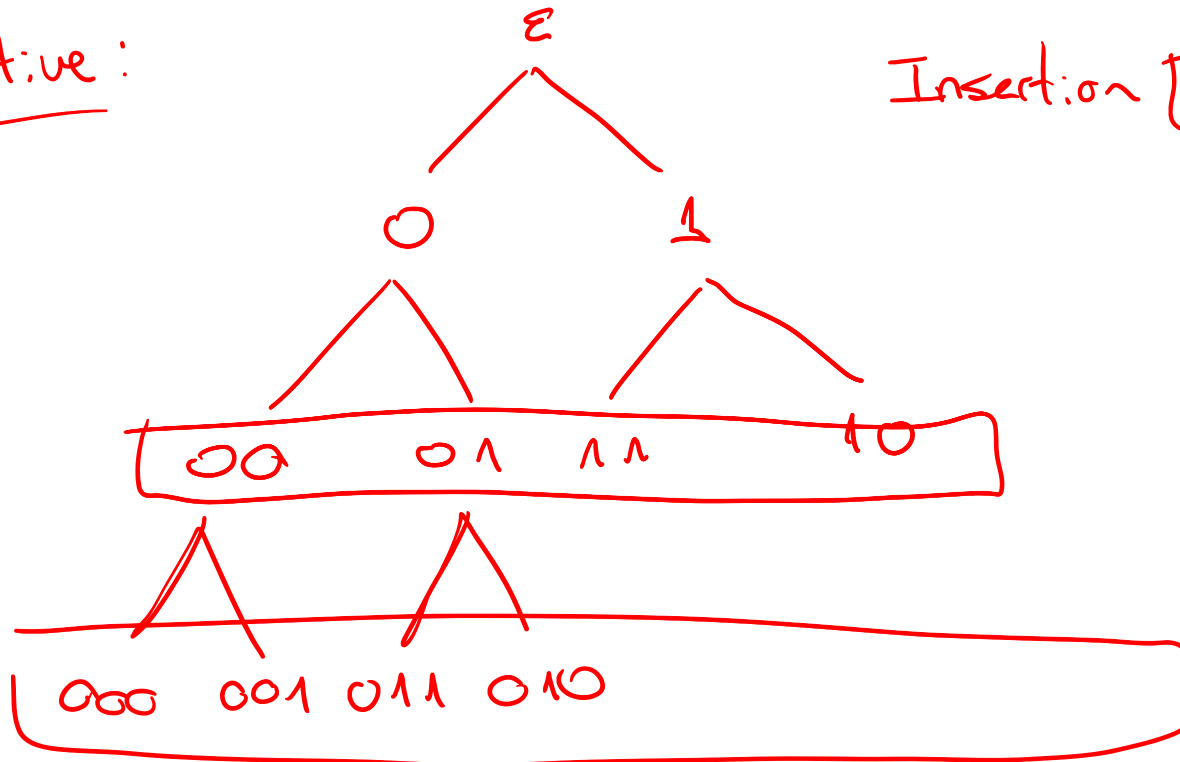
Main approach:

$\{0,1\}^n \hookrightarrow$ Peakless permutations

Deletion 0000 ✗

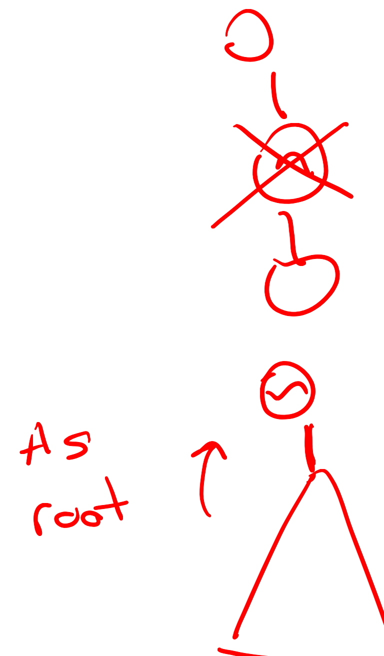
Insertion 0000 0 1

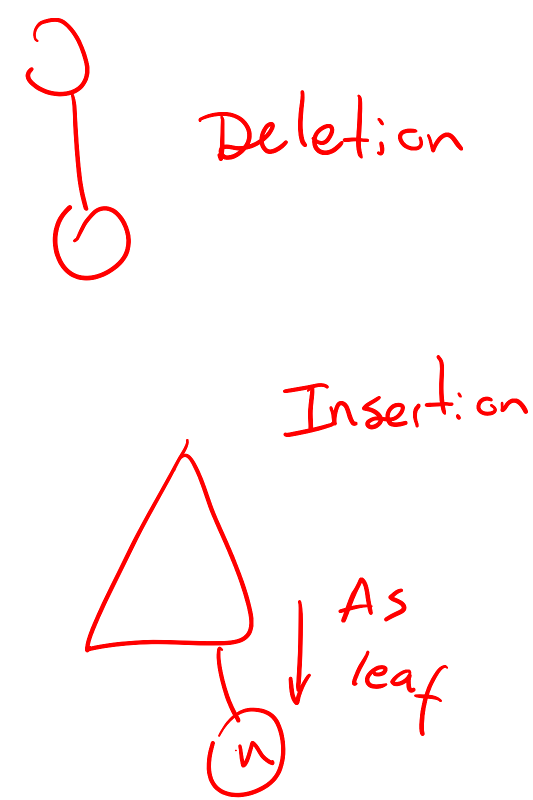
Alternative:



Examples: BRGC and Binary trees

Binary trees:





Non-examples (?)

non-necessarily adjacent.

- permutations of $[n]$ by transpositions.

Non-examples (?)

- permutations of $[n]$ by transpositions.
- permutations of $[n]$ by prefix-reversal.

1 2 3 4 5

3 2 1 4 5

Non-examples (?)

- permutations of $[n]$ by transpositions.
- permutations of $[n]$ by prefix-reversal.
- n pancakes with a burnt side by flips.

Non-examples (?)

- permutations of $[n]$ by transpositions.
- permutations of $[n]$ by prefix-reversal.
- n pancakes with a burnt side by flips.
- *index-based* Gray codes.

Prefix-based Gray codes

???

A simple algorithm

- **Start** with any object.

A simple algorithm

- **Start** with any object.
- **Repeat:** perform the flip such that:

A simple algorithm

- **Start** with any object.
- **Repeat:** perform the flip such that:
  - it generates a new object from the class, and

A simple algorithm

- **Start** with any object.
- **Repeat:** perform the flip such that:
  - it generates a new object from the class, and
  - it flips the elements within the *shortest prefix*

A simple algorithm

- **Start** with any object.
- **Repeat:** perform the flip such that:
  - it generates a new object from the class, and
  - it flips the elements within the *shortest prefix*
- What is the difference with the generic greedy? Zig-zag?

Examples

- permutations of $[n]$ by transpositions.
- permutations of $[n]$ by prefix-reversal.
- n pancakes with a burnt side by flips.

Examples

- permutations of $[n]$ by transpositions.
- permutations of $[n]$ by prefix-reversal.
- n pancakes with a burnt side by flips.
- n-bitstrings by prefix-complementation

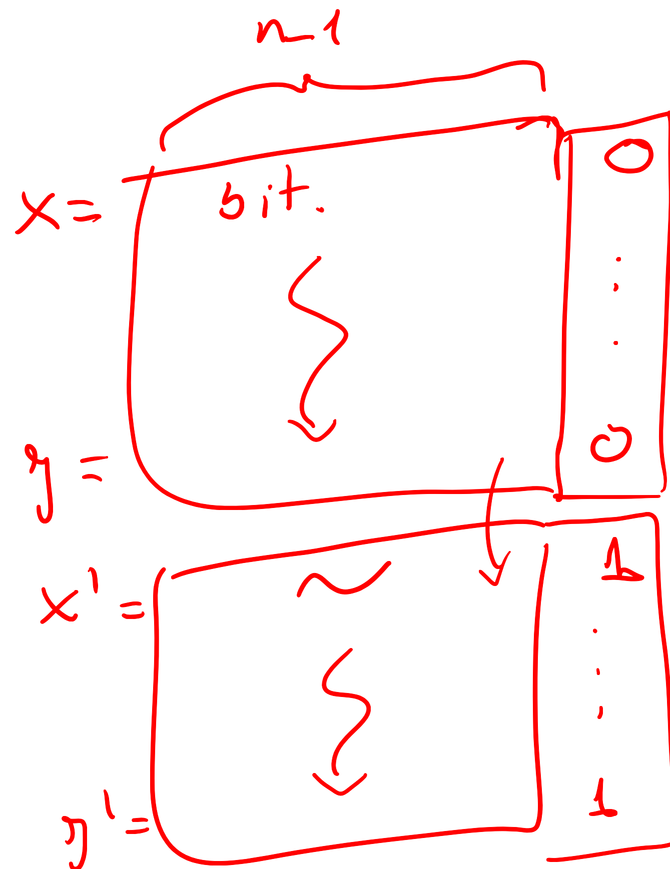
000
100
010
110
001

Proof: prefix-complementation.

By induction:

Hip: This alg. always works on n-bitstrings when starting w/ any bitstring.

induction gives everything on $n-1$-bitstrings

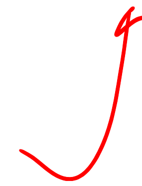


induction gives everything on $n-1$-bitstrings

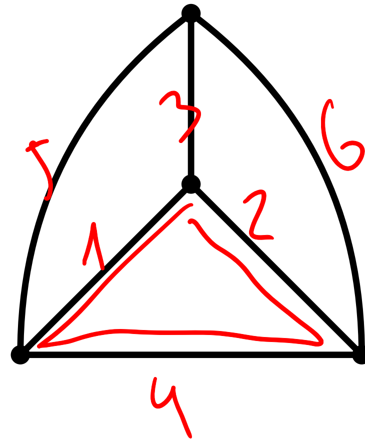
New: a greedy Gray code for spanning trees

- **Label** the edges in any order.
- **Start** with any spanning tree.
- **Repeat:** perform the flip $T \to T + i - j$ such that:
    - it generates a new object from the class, and
    - $\max\{i, j\}$ is maximized and break ties by minimizing $\min\{i, j\}$.

Doesn't really matter



Example: Spanning trees



1 2 3 $\longleftrightarrow$ 1 1 1 0 0 0

0 1 1 1 0 0

1 0 1 1 0 0

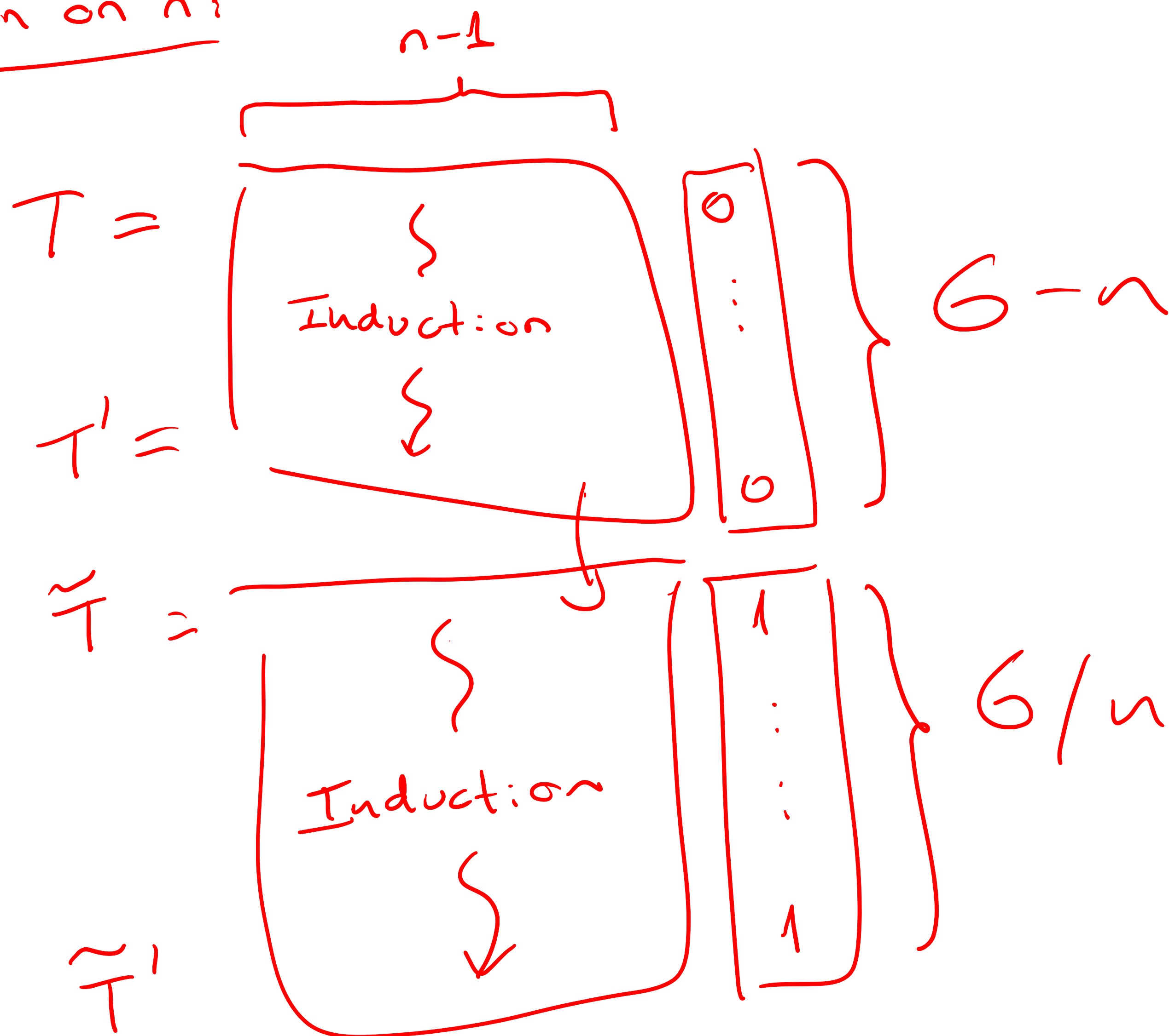
etc...

Perform shortest

Prefix edge-exchange

Proof: Spanning trees

Induction on n:



Some surprises/remarks about the spanning tree algorithm

- A lot of things don't matter.

Some surprises/remarks about the spanning tree algorithm

- A lot of things don't matter.
  - Labelling of the edges.

Some surprises/remarks about the spanning tree algorithm

- A lot of things don't matter.
  - Labelling of the edges.
  - Starting spanning tree.

Some surprises/remarks about the spanning tree algorithm

- A lot of things don't matter.
  - Labelling of the edges.
  - Starting spanning tree.
  - tie-breaking rule.

Some surprises/remarks about the spanning tree algorithm

- A lot of things don't matter.
  - Labelling of the edges.
  - Starting spanning tree.
  - tie-breaking rule.
- Not so clear how to implement efficiently.

Some surprises/remarks about the spanning tree algorithm

- A lot of things don't matter.
  - Labelling of the edges.
  - Starting spanning tree.
  - tie-breaking rule.
- Not so clear how to implement efficiently.
- Works for matroids!

Horizons

- All correctness proofs look very very similar.

Horizons

- All correctness proofs look very very similar.
    - Is there some general principle at play?

Horizons

- All correctness proofs look very very similar.
  - Is there some general principle at play?
  - Something like the zig-zag conditions?

Horizons

- All correctness proofs look very very similar.
  - Is there some general principle at play?
  - Something like the zig-zag conditions?
  - $\{0,1\}^n$ case seems easier than the general case.

Horizons

- All correctness proofs look very very similar.
  - Is there some general principle at play?
  - Something like the zig-zag conditions?
  - $\{0, 1\}^n$ case seems easier than the general case.

- Memoryless?

Final remarks

A link between worlds (?)

- Despite their differences:

A link between worlds (?)

- Despite their differences:
  - Any start v/s identity start.

A link between worlds (?)

- Despite their differences:
  - Any start v/s identity start.
  - Efficient v/s not yet efficient.

A link between worlds (?)

- Despite their differences:
  - Any start v/s identity start.
  - Efficient v/s not yet efficient.
- **Duality**: values v/s indices.

A link between worlds (?)

- Despite their differences:
  - Any start v/s identity start.
  - Efficient v/s not yet efficient.
- **Duality**: values v/s indices.
- **Link (?):** inversion table.

Summary

- Greedy gray codes: powerful tool for generation.

Summary

- Greedy gray codes: powerful tool for generation.

- Two approaches zig-zag and prefix-based.

Summary

- Greedy gray codes: powerful tool for generation.
- Two approaches zig-zag and prefix-based.
- New matroid result.

Open questions

- Too many.

Open questions

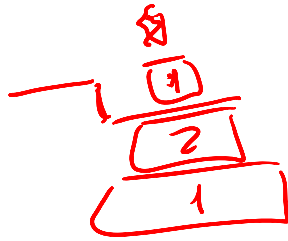
- Too many.

- Better understanding of prefix-based greedy Gray codes.

Open questions

- Too many.
- Better understanding of prefix-based greedy Gray codes.
- Potential link between dual approaches?

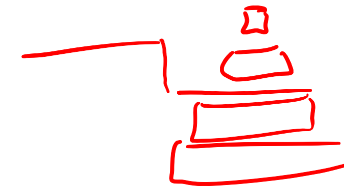
Open questions



Pancakes

- Too many.
- Better understanding of prefix-based greedy Gray codes.

- Potential link between dual approaches?



- **Concrete:** $\mathcal{O}(1)$ delay for spanning trees.
  - **Known:** Amortized $\mathcal{O}(1)$ delay.

Thanks!