How to Count Words Cleverly

Lara Pudwell

Rutgers University

Graduate Student Combinatorics Conference, 2007

Outline

Outline

Reduction

- Given a string of letters $p = p_1 \ldots p_n$, the reduction of p is the string obtained by replacing the i^{th} smallest letter of p with i.

Reduction

- Given a string of letters $p = p_1 ... p_n$, the reduction of p is the string obtained by replacing the i^{th} smallest letter of p with i.
- For example, the reduction of 26745 is

Reduction

- Given a string of letters $p = p_1 \ldots p_n$, the reduction of p is the string obtained by replacing the i^{th} smallest letter of p with i.
- For example, the reduction of 26745 is 1• • ••.

Reduction

- Given a string of letters $p = p_1...p_n$, the reduction of p is the string obtained by replacing the i^{th} smallest letter of p with i.
- For example, the reduction of `26745` is `1●●2●`.

Reduction

- Given a string of letters $p = p_1 ... p_n$, the reduction of p is the string obtained by replacing the i^{th} smallest letter of p with i.
- For example, the reduction of 26745 is 1●●23.

Reduction

- Given a string of letters $p = p_1 ... p_n$, the reduction of p is the string obtained by replacing the i^{th} smallest letter of p with i.
- For example, the reduction of 26745 is 14•23.

Reduction

- Given a string of letters $p = p_1 ... p_n$, the reduction of p is the string obtained by replacing the i^{th} smallest letter of p with i.
- For example, the reduction of 26745 is 14523.

Pattern Avoidance in Permutations

- Given $p \in S_n$ and $q \in S_k$, we say p contains q if there are $1 \leq i_1 < \cdots < i_k \leq n$ such that $p_{i_1} ... p_{i_k}$ reduces to q. Otherwise, p avoids q.

Pattern Avoidance in Permutations

- Given $p \in S_n$ and $q \in S_k$, we say p contains q if there are $1 \leq i_1 < \cdots < i_k \leq n$ such that $p_{i_1} \ldots p_{i_k}$ reduces to q. Otherwise, p avoids q.
- $p = 21354$ *contains* 132.
  (since **2**1**354** reduces to 132.)

Pattern Avoidance in Permutations

- Given $p \in S_n$ and $q \in S_k$, we say p contains q if there are $1 \leq i_1 < \cdots < i_k \leq n$ such that $p_{i_1} \ldots p_{i_k}$ reduces to q. Otherwise, p avoids q.
- $p = 21354$ *contains* 132.
  (since **2**13**54** reduces to 132.)
- $p = 21354$ *avoids* 321.
  (since p has no decreasing subsequence of length 3.)

Pattern Avoidance in Words

- A word in $[k]^n$ is a string $w = w_1 \dots w_n$ where $1 \leq w_i \leq k$ for each i.

Pattern Avoidance in Words

- A word in $[k]^n$ is a string $w = w_1 \ldots w_n$ where $1 \leq w_i \leq k$ for each i.
- Pattern avoidance in words is defined analogously for words.

Pattern Avoidance in Words

- A word in $[k]^n$ is a string $w = w_1 \ldots w_n$ where $1 \leq w_i \leq k$ for each i.
- Pattern avoidance in words is defined analogously for words.
  - $w = 13531246 \in [6]^8$ *contains* 123 and 122.
    (13531**246** reduces to 123.
    13531246 reduces to 122.)

Pattern Avoidance in Words

- A word in $[k]^n$ is a string $w = w_1 \ldots w_n$ where $1 \leq w_i \leq k$ for each i.
- Pattern avoidance in words is defined analogously for words.
  - $w = 13531246 \in [6]^8$ *contains* 123 and 122.
    (13531**246** reduces to 123.
    13531246 reduces to 122.)
  - $w = 13531246$ *avoids* 111 and 12345.
    (There is no letter repeated 3 times, and
    there is no increasing subsequence of length 5.)

Key Question

- For Permutations...
  - Let $S_n(q) = |\{p \in S_n \mid p \text{ avoids } q\}|$.

Key Question

- For Permutations...
  - Let $S_n(q) = |\{p \in S_n \mid p \text{ avoids } q\}|$.
  - Find a way to count $S_n(q)$.

Key Question

- For Permutations...
  - Let $S_n(q) = |\{p \in S_n \mid p \text{ avoids } q\}|$.
  - Find a way to count $S_n(q)$.
- For Words...
  - Let $A_{[a_1,\ldots,a_k]}(q) = |\{w \in [k]^{a_1+\cdots+a_k} \mid w \text{ avoids } q, \text{ and } w \text{ has } a_i \; i\text{'s }\}|$

Key Question

- For Permutations...
  - Let $S_n(q) = |\{p \in S_n \mid p \text{ avoids } q\}|$.
  - Find a way to count $S_n(q)$.
- For Words...
  - Let $A_{[a_1,\ldots,a_k]}(q) = |\{w \in [k]^{a_1+\cdots+a_k} \mid w \text{ avoids } q\text{, and } w \text{ has } a_i\ i\text{'s}\}|$
  - Given q, find a way to count $A_{[a_1,\ldots,a_k]}(q)$.

Key Question

- For Permutations...
  - Let $S_n(q) = |\{p \in S_n \mid p \text{ avoids } q\}|$.
  - Find a way to count $S_n(q)$.
- For Words...
  - Let $A_{[a_1,\ldots,a_k]}(q) = |\{w \in [k]^{a_1+\cdots+a_k} \mid w \text{ avoids } q\text{, and } w \text{ has } a_i\ i\text{'s}\}|$
  - Given q, find a way to count $A_{[a_1,\ldots,a_k]}(q)$.
  - Note: $A_{[1,\ldots,1]}(q) = S_n(q)$.

Outline

For Permutations

- Most techniques studying $S_n(q)$ finds formulas for a *specific* q.

For Permutations

- Most techniques studying $S_n(q)$ finds formulas for a *specific* q.
- 1998: Zeilberger's *prefix enumeration schemes*, i.e. a system of recurrences to count $S_n(q)$.

For Permutations

- Most techniques studying $S_n(q)$ finds formulas for a *specific* q.
- 1998: Zeilberger's *prefix enumeration schemes*, i.e. a system of recurrences to count $S_n(q)$.
- 2005: Vatter's modified schemes automate the enumeration of $S_n(q)$ for even more patterns q.

For Words

- 1998: Burstein's thesis counts many pattern-avoiding words using generatingfunctionology.

For Words

- 1998: Burstein's thesis counts many pattern-avoiding words using generatingfunctionology.
- Vatter and Zeilberger's schemes can be modified to count pattern-avoiding words.

For Words

- 1998: Burstein's thesis counts many pattern-avoiding words using generatingfunctionology.
- Vatter and Zeilberger's schemes can be modified to count pattern-avoiding words.
  - Advantage: One plan of attack for many different patterns.

Outline

1. Introduction
   - Pattern Avoidance
   - Previous Work
2. Counting Pattern-Avoiding Words
   - Counting Concepts
   - Examples

Refinement

Suppose that we want to count a set $A(n)$.

- Ideally, find a recurrence $A(n) = \sum_{i \in I} c(i) * A(n-i)$.

Refinement

Suppose that we want to count a set $A(n)$.

- Ideally, find a recurrence $A(n) = \sum_{i \in I} c(i) * A(n-i)$.
- If not...

Refinement

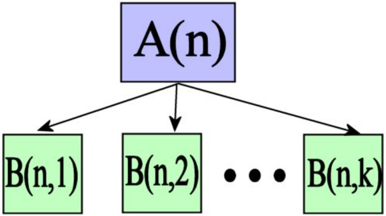
Suppose that we want to count a set $A(n)$.

- Ideally, find a recurrence $A(n) = \sum_{i \in I} c(i) * A(n-i)$.
- If not...
  1. Break $A(n)$ into disjoint union $\bigcup_{i \in I} B(n, i)$. (refinement according to parameter i)

Refinement

Suppose that we want to count a set $A(n)$.

- Ideally, find a recurrence $A(n) = \sum_{i \in I} c(i) * A(n-i)$.
- If not...
  1. Break $A(n)$ into disjoint union $\bigcup_{i \in I} B(n,i)$. (refinement according to parameter i)
  2. Look for a recurrence for each $B(n,i)$.



Refinement for Words

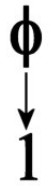
- For pattern-avoiding words, refine according to prefixes.

Refinement for Words

- For pattern-avoiding words, refine according to prefixes.
  - to count all words avoiding q, count all words starting with a 1 pattern.

Refinement for Words

- For pattern-avoiding words, refine according to prefixes.
    - to count all words avoiding q, count all words starting with a 1 pattern.



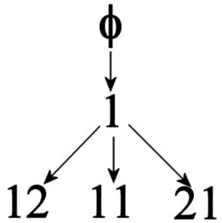
Refinement for Words

- For pattern-avoiding words, refine according to prefixes.
  - to count all words avoiding q, count all words starting with a 1 pattern.
  - to count all words starting with a 1, count all words starting with a 12, a 11, or a 21 pattern.

$$\phi \downarrow 1$$

Refinement for Words

- For pattern-avoiding words, refine according to prefixes.
  - to count all words avoiding q, count all words starting with a 1 pattern.
  - to count all words starting with a 1, count all words starting with a 12, a 11, or a 21 pattern.



Reversibly Deletable

- Given a prefix $p = p_1...p_t$, position r is reversibly deletable if every possible bad pattern involving p_r implies another bad pattern without p_r.

Reversibly Deletable

- Given a prefix $p = p_1 ... p_t$, position r is reversibly deletable if every possible bad pattern involving p_r implies another bad pattern without p_r.
- For example, avoid $q = 123$, and let $p = 21...$

21...a...b

Reversibly Deletable

- Given a prefix $p = p_1 ... p_t$, position r is reversibly deletable if every possible bad pattern involving p_r implies another bad pattern without p_r.
- For example, avoid $q = 123$, and let $p = 21...$

21...a...b
21...a...b

Reversibly Deletable

- Given a prefix $p = p_1 ... p_t$, position r is reversibly deletable if every possible bad pattern involving p_r implies another bad pattern without p_r.
- For example, avoid $q = 123$, and let $p = 21...$

21...a...b

21...a...b

- If p_r is reversibly deletable, and the role of p_r is played by letter j, then $A_{[a_1,...a_j,...a_n]}(q) = A_{[a_1,...a_j-1,...a_n]}(q)$.

Gap Vectors

Consider words that avoid $q = 123$ and begin with prefix $p = 12$

sorted prefix: 1 2
letters involved in prefix: i j
vector: <a, b, c>

Gap Vectors

Consider words that avoid $q = 123$ and begin with prefix $p = 12$

sorted prefix: 1 2
letters involved in prefix: i j
vector: <a, b, c>

sorted word: $\underbrace{\cdots}_{\geq a} i \underbrace{\cdots}_{\geq b-1} j \underbrace{\cdots}_{\geq c}$

Gap Vectors

Consider words that avoid $q = 123$ and begin with prefix $p = 12$

sorted prefix: 1 2
letters involved in prefix: i j
vector: <a, b, c>

sorted word: $\underbrace{\cdots}_{\geq a} i \underbrace{\cdots}_{\geq b-1} j \underbrace{\cdots}_{\geq c}$

v is a gap vector for p if there are no words avoiding q with prefix p and spacing v.

Gap Vectors

Consider words that avoid $q = 123$ and begin with prefix $p = 12$

sorted prefix: 1 2
letters involved in prefix: i j
vector: <a, b, c>

sorted word: $\underbrace{\cdots}_{\geq a} i \underbrace{\cdots}_{\geq b-1} j \underbrace{\cdots}_{\geq c}$

v is a gap vector for p if there are no words avoiding q with prefix p and spacing v.

e.g. $v =< 0, 1, 1 >$ is a gap vector for $q = 123$, $p = 12$.

Enumeration Scheme

Algorithm for finding an enumeration scheme:

1. Start with the empty prefix.

Enumeration Scheme

Algorithm for finding an enumeration scheme:

1. Start with the empty prefix.
2. Find refinements of all prefixes in scheme.

Enumeration Scheme

Algorithm for finding an enumeration scheme:

1. Start with the empty prefix.
2. Find refinements of all prefixes in scheme.
3. Find gap vectors for all such refinements.

Enumeration Scheme

Algorithm for finding an enumeration scheme:

1. Start with the empty prefix.
2. Find refinements of all prefixes in scheme.
3. Find gap vectors for all such refinements.
4. Find reversibly deletable positions with respect to gap vectors.

Enumeration Scheme

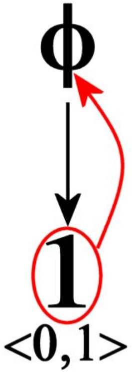
Algorithm for finding an enumeration scheme:

1. Start with the empty prefix.
2. Find refinements of all prefixes in scheme.
3. Find gap vectors for all such refinements.
4. Find reversibly deletable positions with respect to gap vectors.
5. Repeat steps 2-4 until all unrefined prefixes have reversibly deletable elements.

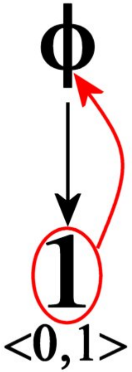
Outline

1. Introduction
   - Pattern Avoidance
   - Previous Work
2. Counting Pattern-Avoiding Words
   - Counting Concepts
   - Examples

Avoid(12)

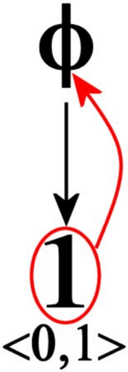


Avoid(12)



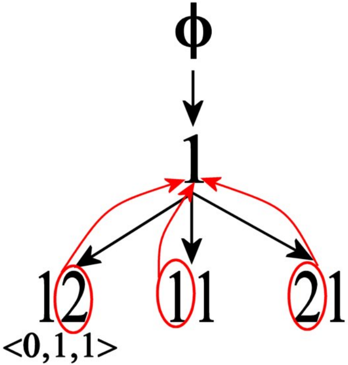
- Gap Vector: If a word avoids 12, the first letter must be the biggest letter.

Avoid(12)

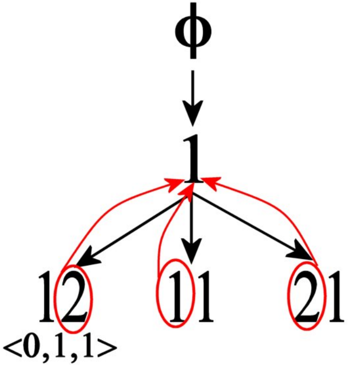


- Gap Vector: If a word avoids 12, the first letter must be the biggest letter.
- Reversibly Deletable: The first (i.e. biggest) letter cannot be in a 12 pattern, so it can be deleted.

Avoid(123)

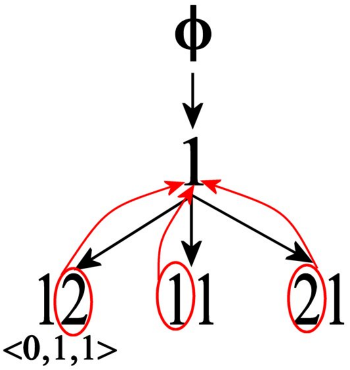


Avoid(123)



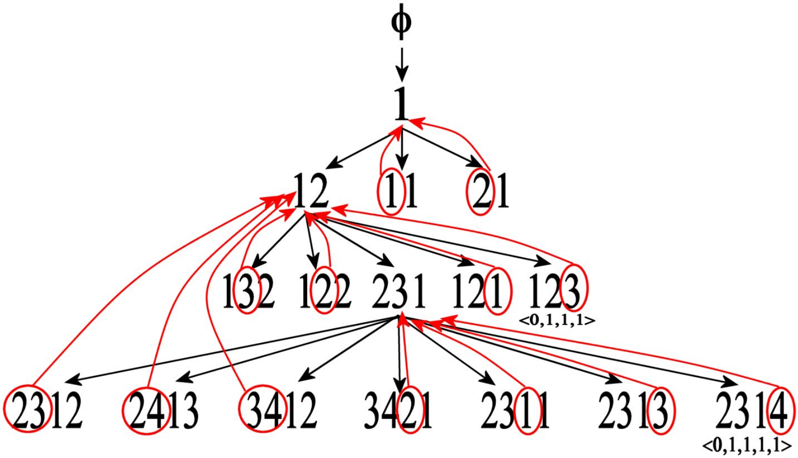
- Gap Vector: If a word avoids 123, and has prefix 12, the role 2 must be played by the largest letter in the word.

Avoid(123)



- Gap Vector: If a word avoids 123, and has prefix 12, the role 2 must be played by the largest letter in the word.
- Reversibly Deletable:
  - The 2 in a 12 prefix cannot be part of a bad pattern if it is the largest letter.
  - If the 1st 1 in a 11 prefix is in a bad pattern, the other 1 is as well.
  - If the 2 in a 21 prefix is part of a bad pattern, the 1 is as well.

Avoid(1234)


ϕ
1
12
11
21
132
122
231
121
123
<0,1,1,1>
2312
2413
3412
3421
2311
2313
2314
<0,1,1,1,1>

Summary

- There are few techniques to count large classes of pattern-avoiding words.

Summary

- There are few techniques to count large classes of pattern-avoiding words.
- Extending Zeilberger's and Vatter's schemes gives a good success rate for words avoiding permutations.

Summary

- There are few techniques to count large classes of pattern-avoiding words.
- Extending Zeilberger's and Vatter's schemes gives a good success rate for words avoiding permutations.

- Future work
  - Find closed formulas from recurrence relations given by schemes.

Summary

- There are few techniques to count large classes of pattern-avoiding words.
- Extending Zeilberger's and Vatter's schemes gives a good success rate for words avoiding permutations.

- Future work
  - Find closed formulas from recurrence relations given by schemes.
  - Modify schemes to count words avoiding other words.

Summary

- There are few techniques to count large classes of pattern-avoiding words.
- Extending Zeilberger's and Vatter's schemes gives a good success rate for words avoiding permutations.

- Future work
  - Find closed formulas from recurrence relations given by schemes.
  - Modify schemes to count words avoiding other words.