CSE21 Winter 2017, Day 19 (B00), Day 13 (A00)

March 1, 2017

http://vlsicad.ucsd.edu/courses/cse21-w17
Announcements

HW #7 assigned
Due: Tuesday 2/7 11:59pm

MT2 handback in Gradescope
By Saturday, latest

HW #6 is in Gradescope
72-hour window for regrade requests
Output size?

Assume $n/k$ is a power of two. Consider $s$ a binary string of length $n$ with $k$ 1s.

**Best case**: 1s toward the beginning of the string. $E(s)$ has
- One bit for each 1 in $s$ to indicate that next bits denote positions in window.
- $\log_2(n/k)$ bits for each 1 in $s$ to specify position of that 1 in a window.
- $k$ such ones.
- No bits representing empty windows because all 0s are either "caught" in windows with 1s or after the last 1.

Total $|E(s)| = k \log_2(n/k) + k$
**Output size?**

Assume \( n/k \) is a power of two. Consider \( s \) a binary string of length \( n \) with \( k \) 1s.

**Worst case**: 1s toward the end of the string. \( E(s) \) has

- At most \( k \) bits for empty windows since at most \( k \) nonoverlapping windows of length \( n/k \) will fit in a string of length \( n \).
- One bit for each 1 in \( s \) to indicate that next bits denote positions in window.
- \( \log_2(n/k) \) bits for each 1 in \( s \) to specify position of that 1 in a window.
- \( k \) such ones.

**Total** \( |E(s)| \leq k \log_2(n/k) + 2k \)
Output size?

Assume $n/k$ is a power of two. Consider $s$ a binary string of length $n$ with $k$ 1s. Given $|E(s)| \leq k \log_2 (n/k) + 2k$, we need at most $k \log_2 (n/k) + 2k$ bits to represent all length $n$ binary strings with $k$ 1s. Hence, there are at most $2^{k \log_2 (n/k) + 2k}$ many such strings.
Output size?

Assume \( n/k \) is a power of two. Consider \( s \) a binary string of length \( n \) with \( k \) 1s. Given \( |E(s)| \leq k \log_2(n/k) + 2k \), we need at most \( k \log_2(n/k) + 2k \) bits to represent all length \( n \) binary strings with \( k \) 1s. Hence, there are at most \( 2^{k \log_2(n/k) + 2k} \) many such strings.
Output size?

Assume \( n/k \) is a power of two. Consider \( s \) a binary string of length \( n \) with \( k \) 1s. Given \(| E(s) | \leq k \log_2(n/k) + 2k\), we need at most \( k \log_2(n/k) + 2k \) bits to represent all length \( n \) binary strings with \( k \) 1s. Hence, there are at most \( 2^{(4n/k)^k} \) many such strings.

\[
2^{(k \log(n/k)+2k)} = 2^{(k \log(n/k))} \cdot 2^{(2k)}
\]
\[
= \left(2^{(\log(n/k))}\right)^k \cdot 2^{(2k)}
\]
\[
= (n/k)^k \cdot 4^k = (4n/k)^k
\]

\[
C(n,k) = \# \text{ Length } n \text{ binary strings with } k \text{ 1s} \leq (4n/k)^k
\]
Bounds for Binomial Coefficients

Using \( \text{windowEncode}() : \binom{n}{k} \leq (4n/k)^k \)

Lower bound?

**Idea:** find a way to count a subset of the fixed density binary strings.

Some fixed density binary strings have one 1 in each of \( k \) chunks of size \( n/k \).

**How many such strings are there?**

A. \( n^n \)  
B. \( k! \)  
C. \( (n/k)^k \)  
D. \( C(n,k)^k \)  
E. None of the above.
Bounds for Binomial Coefficients

Using `windowEncode()`:

\[
{n \choose k} \leq (4n/k)^k
\]

Using evenly spread strings:

\[
(n/k)^k \leq \binom{n}{k}
\]

Counting helps us analyze our compression algorithm.

Compression algorithms help us count.
A **theoretically optimal encoding** for length $n$ binary strings with $k$ 1s would use the ceiling of \( \log_2 \binom{n}{k} \) bits.

**How?**
- List all length $n$ binary strings with $k$ 1s in some order.
- **To encode:** Store the position of a string in the list, rather than the string itself.
- **To decode:** Given a position in list, need to determine string in that position.

**Example:** \( \binom{12}{3} = \) 220
A **theoretically optimal encoding** for length $n$ binary strings with $k$ 1s would use the ceiling of $\log_2 \left( \binom{n}{k} \right)$ bits.

**How?**
- List all length $n$ binary strings with $k$ 1s in some order.
- **To encode**: Store the position of a string in the list, rather than the string itself.
- **To decode**: Given a position in list, need to determine string in that position.

Use lexicographic (dictionary) ordering ...
Lex Order

String $a$ comes before string $b$ if the first time they differ, $a$ is smaller.

I.e.

$$a_1 a_2 ... a_n <_{\text{lex}} b_1 b_2 ... b_n$$

means there exists $j$ such that

$$a_i = b_i \text{ for all } i < j \text{ AND } a_j < b_j$$

Which of these comes last in lex order?

A. 1001   C. 1101   E. 0000
B. 0011   D. 1010

C. 1101
Lex Order

E.g. Length \( n=5 \) binary strings with \( k=3 \) ones, listed in lex order:

<table>
<thead>
<tr>
<th>Original string, ( s )</th>
<th>Encoded string (i.e. position in this list)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00111</td>
<td>0 = 0000</td>
</tr>
<tr>
<td>01011</td>
<td>1 = 0001</td>
</tr>
<tr>
<td>01101</td>
<td>2 = 0010</td>
</tr>
<tr>
<td>01110</td>
<td>3 = 0011</td>
</tr>
<tr>
<td>10011</td>
<td>4 = 0100</td>
</tr>
<tr>
<td>10101</td>
<td>5 = 0101</td>
</tr>
<tr>
<td>10110</td>
<td>6 = 0110</td>
</tr>
<tr>
<td>11001</td>
<td>7 = 0111</td>
</tr>
<tr>
<td>11010</td>
<td>8 = 1000</td>
</tr>
<tr>
<td>11100</td>
<td>9 = 1001</td>
</tr>
</tbody>
</table>

\( \binom{5}{3} = 10 \)
Lex Order

E.g. Length \( n = 5 \) binary strings with \( k = 3 \) ones, listed in lex order:

<table>
<thead>
<tr>
<th>Original string, ( s )</th>
<th>Encoded string (i.e. position in this list)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00111</td>
<td>0 = 0000</td>
</tr>
<tr>
<td>01011</td>
<td>1 = 0001</td>
</tr>
<tr>
<td>01101</td>
<td>2 = 0010</td>
</tr>
<tr>
<td>01110</td>
<td>3 = 0011</td>
</tr>
<tr>
<td>10011</td>
<td>4 = 0100</td>
</tr>
<tr>
<td>10101</td>
<td>5 = 0101</td>
</tr>
<tr>
<td>10110</td>
<td>6 = 0110</td>
</tr>
<tr>
<td>11001</td>
<td>7 = 0111</td>
</tr>
<tr>
<td>11010</td>
<td>8 = 1000</td>
</tr>
<tr>
<td>11100</td>
<td>9 = 1001</td>
</tr>
</tbody>
</table>
Need two algorithms, given specific n and k:

\[ s \rightarrow E(s,n,k) \]

and

\[ p \rightarrow D(p,n,k) \]

**Idea:** Use recursion.

**Key insight:** In lex order, strings that start with 0 come before strings that start with 1.
Lex Order: Algorithm?

For $E(s,n,k)$:

- Any string that starts with 0 must have position before $\binom{n-1}{k}$.
- Any string that starts with 1 must have position at or after $\binom{n-1}{k}$. 

Length $n$-1 binary strings with $k$ 1s

Length $n$-1 binary strings with $k-1$ 1s

$$E(s_1, \ldots, s_n) = \binom{n-1}{k}$$
Example: Encode 010010100

(Note: this is a length 9 binary string with 3 ones.)

\[ E_{9,3}(010010100) \]

\[ = 0 + E_{5,3}(10010100) \]

\[ = 0 + \binom{9}{3} + E_{7,2}(0010100) \]

\[ = 0 + \binom{9}{3} + 0 + E_{6,2}(010100) \]

\[ = 0 + \binom{9}{3} + 35 + 0 + E_{5,2}(10100) \]

\[ + \binom{9}{4} + E_{4,1}(0100) \]

\[ - \binom{9}{5} + E_{3,1}(100) \]
Lex Order: Algorithm?

Example: Encode 010010100

(Note: this is a length 9 binary string with 3 ones.)

Decoding:

43 < \left( \frac{8}{3} \right) \text{ so } 1^{st} \text{ position is } 0

43 - 0 > \left( \frac{7}{3} \right) \text{ so } 2^{nd} \text{ position is } 1

43 - 1 \left( \frac{7}{3} \right)

= 8 < \left( \frac{6}{2} \right) \text{ so } 3^{rd} \text{ position is } 0

8 < \left( \frac{5}{2} \right) \text{ so } 4^{th} \text{ position is } 0

8 > \left( \frac{4}{2} \right) \text{ so } 5^{th} \text{ position is } 1

8 - 4 \left( \frac{4}{2} \right) = 2 < \left( \frac{3}{1} \right)

2 = (2) \quad \text{ so } 6^{th} \text{ position is } 0

0

1

0

1

0

2

3

0

1

0

1

0

7
Lex Order: Algorithm?

For E(s,n,k):

- Any string that starts with 0 must have position before $\binom{n-1}{k}$.
- Any string that starts with 1 must have position at or after $\binom{n-1}{k}$.

procedure lexEncode (b₁b₂...bₙ, n, k)

1. If n = 1, return 0.
2. If s₁ = 0, return lexEncode (b₂...bₙ, n-1, k)
3. Else
4. return C(n-1,k) + lexEncode(b₂...bₙ, n-1, k-1)
Lex Order: Algorithm?

For D(s, n, k):

- Any position before \(\binom{n-1}{k}\) must correspond to string that starts with 0.
- Any position at or after \(\binom{n-1}{k}\) must correspond to string that starts with 1.

procedure lexDecode(p, n, k)

1. If \(n = k\),
2. return 1111..1 // length n string of all 1s
3. If \(p < C(n-1,k)\),
4. return "0" + lexDecode(p, n-1, k)
5. Else
6. return "1" + lexDecode(p-C(n-1,k), n-1, k-1)
**Victory!**

**Using lexEncode, lexDecode,** we can represent any fixed density length \( n \) binary string with \( k \) 1s as one of \( \binom{n}{k} \) positions.

So, it takes \( \log_2(\binom{n}{k}) \) bits to store fixed-density binary strings using lex order.

**Theoretical lower bound:** \( \log_2(\binom{n}{k}) \).

Same! So this encoding algorithm is optimal.
11-avoiding binary strings

Let’s consider the set of all n-bit binary strings with the property that 11 is not a substring.

*example: 1000101000010101*

*How many of these strings are there?*
Let’s consider the set of all n-bit binary strings with the property that 11 is not a substring.

How many of these strings are there?

Let $S(n)$ be the set of all n-bit binary strings of this type. Then split them into two subsets:

- $S_1(n)$ is the subset that starts with 1
- $S_0(n)$ is the subset that starts with 0

Are $S_1(n)$ and $S_0(n)$ disjoint?
11-avoiding binary strings

Let’s consider the set of all n-bit binary strings with the property that 11 is not a substring.

\[ S(n) = S1(n) \cup S0(n) \text{ and } S1(n) \cap S0(n) = \emptyset \]

\[ |S(n)| = |S1(n)| + |S0(n)| \]

All elements of S0(n) are of the form 0[11-avoiding (n-1)-bit binary string]
All elements of S1(n) are of the form 10[11-avoiding (n-2)-bit binary string]

\[ |S0(n)| = |S(n - 1)| \]
\[ |S1(n)| = |S(n - 2)| \]

\[ |S(n)| = |S(n - 1)| + |S(n - 2)| \]
11-avoiding binary strings

$$|S(n)| = |S(n - 1)| + |S(n - 2)|$$

what does this look like?

What is $|S(1)|$? What is $|S(2)|$?

1, 0

0, 1, 1, 10
11-avoiding binary strings

\[ |S(n)| = |S(n - 1)| + |S(n - 2)| \]

what does this look like?

|S(1)| = 2 \quad |S(2)| = 3

\[ |S(n)| = F_{n+2} \text{ (Fibonacci number.)} \]

\[ F_1 = 1, F_2 = 1 \]
The most straightforward way to encode one of these strings is by using the string itself.

This gives us an upper bound on the number of bits needed to encode these types of strings.

n-bits is enough to encode these strings so this means that

$$|S(n)| = F_{n+2} \leq 2^n$$
A **theoretically optimal encoding** for length n 11-avoiding binary strings would use the ceiling of $\log_2(F_{n+2})$ bits.

**How?**
- List all length n 11-avoiding binary strings in lex-order
- **To encode:** Store the position of a string in the list, rather than the string itself.
- **To decode:** Given a position in list, need to determine string in that position.
E.g. the 13 Length n=5 11-avoiding binary strings:

<table>
<thead>
<tr>
<th>Original string, $s$</th>
<th>Encoded string (i.e. position in this list)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>0 = 0000</td>
</tr>
<tr>
<td>0 0 0 0 1</td>
<td>1 = 0001</td>
</tr>
<tr>
<td>0 0 0 1 0</td>
<td>2 = 0010</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
<td>3 = 0011</td>
</tr>
<tr>
<td>0 0 1 0 1</td>
<td>4 = 0100</td>
</tr>
<tr>
<td>0 1 0 0 0</td>
<td>5 = 0101</td>
</tr>
<tr>
<td>0 1 0 0 1</td>
<td>6 = 0110</td>
</tr>
<tr>
<td>0 1 0 1 0</td>
<td>7 = 0111</td>
</tr>
<tr>
<td>1 0 0 0 0</td>
<td>8 = 1000</td>
</tr>
<tr>
<td>1 0 0 0 1</td>
<td>9 = 1001</td>
</tr>
<tr>
<td>1 0 0 1 0</td>
<td>10 = 1010</td>
</tr>
<tr>
<td>1 0 1 0 0</td>
<td>11 = 1011</td>
</tr>
<tr>
<td>1 0 1 0 1</td>
<td>12 = 1100</td>
</tr>
</tbody>
</table>

$S(4)$ + $S(3)$
Lex Order

E.g. the 13 Length $n=5$ 11-avoiding binary strings:

<table>
<thead>
<tr>
<th>Original string, $s$</th>
<th>Encoded string (i.e. position in this list)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>0 = 0000</td>
</tr>
<tr>
<td>00001</td>
<td>1 = 0001</td>
</tr>
<tr>
<td>00010</td>
<td>2 = 0010</td>
</tr>
<tr>
<td>00100</td>
<td>3 = 0011</td>
</tr>
<tr>
<td>00101</td>
<td>4 = 0100</td>
</tr>
<tr>
<td>01000</td>
<td>5 = 0101</td>
</tr>
<tr>
<td>01001</td>
<td>6 = 0110</td>
</tr>
<tr>
<td>01010</td>
<td>7 = 0111</td>
</tr>
<tr>
<td>10000</td>
<td>8 = 1000</td>
</tr>
<tr>
<td>10001</td>
<td>9 = 1001</td>
</tr>
<tr>
<td>10010</td>
<td>10 = 1010</td>
</tr>
<tr>
<td>10100</td>
<td>11 = 1011</td>
</tr>
<tr>
<td>10101</td>
<td>12 = 1100</td>
</tr>
</tbody>
</table>
Lex Order: Algorithm?

Need two algorithms, given specific $n$:

\[ s \rightarrow E(s,n) \]

and

\[ p \rightarrow D(p,n) \]

*Idea*: Use recursion (reduce & conquer).
Lex Order: Algorithm?

For $E(s, n)$:

- Any string that starts with 0 must have position before $|S(n - 1)| = F_{n+1}$
- Any string that starts with 1 must have position at or after $|S(n - 1)| = F_{n+1}$
Lex Order: Algorithm?

Example: Encode 010101001

(Note: this is a length 9 binary string.)

S(0)=1
S(1)=2
S(2)=3
S(3)=5
S(4)=8
S(5)=13
S(6)=21
S(7)=34
S(8)=55
S(9)=89
Lex Order: Algorithm?

Example: Encode 010101001

(Note: this is a length 9 binary string.)

So this string is number 53 in the list and so it will be encoded by the binary expansion of 53. The maximum number of bits needed to store any of these strings is \(\lceil \log(F_{11}) \rceil = 7\). So we will pad the left with 0’s

\[ E(010101001,9) = 53 \]
Lex Order: Algorithm?

Example: Decode 1001011 = 75 into a length 9 binary string.
Lex Order: Algorithm?

Example: Decode $1001011 = 75$ into a length 9 binary string.

75>55=F_10 so the first bit is 1
75-55=20

20<34=F_9 so the next bit is 0
20<21=F_8 so the next bit is 0

20>13=F_7 so the next bit is 1
20-13=7

7<8=F_6 so the next bit is 0

7>5=F_5 so the next bit is 1
7-5=2

2<3=F_4 so the next bit is 0
2=2=F_3 so the next bit is 1
2-2=0

0<1=F_2 so the next bit is 0

D(1001011,9)=100101010
A *theoretically optimal encoding* for length n 11-avoiding binary strings would use the ceiling of $\log_2(F_{n+2})$ bits.

How big is $\log_2(F_{n+2})$?

$$F_{n+2} < (1.6)^{n+2} \approx (2^{0.7})^{n+2} = 2^{0.7n+1.4}$$

So…….

$$[\log_2(F_{n+2})] < [\log_2 2^{0.7n+1.4}] = [0.7n + 1.4]$$
Another application of counting … lower bounds

**Sorting algorithm:** performance was measured in terms of number of comparisons between list elements

*What's the fastest possible worst case* for any sorting algorithm?
Another application of counting … lower bounds

**Sorting algorithm**: performance was measured in terms of number of comparisons between list elements.

*What's the* **fastest possible worst case** *for any sorting algorithm?*

**Tree diagram for a sorting algorithm** represents possible comparisons we might have to do, based on relative sizes of elements.

Sometimes called a decision tree
Another application of counting ... lower bounds

Tree diagram for a sorting algorithm

Rosen p. 761
Another application of counting … lower bounds

**Sorting algorithm:** performance was measured in terms of number of comparisons between list elements

What's the **fastest possible worst case** for any sorting algorithm?

Maximum (worst-case) number of comparisons for a sorting algorithm is the **height** of its tree diagram.
How many leaves will there be in a decision tree that sorts $n$ elements?

A. $2^n$
B. $\log n$
C. $n!$
D. $C(n,2)$
E. None of the above.
Another application of counting … lower bounds

**Sorting algorithm:** performance was measured in terms of number of comparisons between list elements

What's the **fastest possible worst case** for any sorting algorithm?

\[
\text{max # of comparisons} = \text{height of tree diagram}
\]

For any algorithm, what would be **smallest possible height**?

What do we know about the tree?
* Internal nodes correspond to comparisons.
* Leaves correspond to possible input arrangements.
Another application of counting … lower bounds

**Sorting algorithm:** performance was measured in terms of number of comparisons between list elements

What's the **fastest possible worst case** for any sorting algorithm?

max # of comparisons = **height** of tree diagram

For any algorithm, what would be **smallest possible height**?

What do we know about the tree?

* Internal nodes correspond to comparisons.
* Leaves correspond to possible input arrangements.
How does height relate to number of leaves?

**Theorem**: There are at most $2^h$ leaves in a binary tree with height $h$.

Which of the following is true?

A. It's possible to have a binary tree with height 3 and 1 leaf.
B. It's possible to have a binary tree with height 1 and 3 leaves.
C. Every binary tree with height 3 has 1 leaf.
D. Every binary tree with height 1 has 3 leaves.
E. None of the above.
How does height relate to number of leaves?

**Theorem:** There are at most $2^h$ leaves in a binary tree with height $h$.

Which of the following is true?

A. It's possible to have a binary tree with height 3 and 1 leaf.
B. It's possible to have a binary tree with height 1 and 3 leaves.
C. Every binary tree with height 3 has 1 leaf.
D. Every binary tree with height 1 has 3 leaves.
E. None of the above.

Proof by induction on $h$
Another application of counting … lower bounds

What's the **fastest possible worst case** for any sorting algorithm?

max # of comparisons = **height** of tree diagram

\[
\# \text{ leaves} \leq 2^h \\
n! \leq 2^h \\
\log_2 n! \leq h \\
h \geq \log_2 n!
\]

Fastest possible worst case performance of sorting is \( \log_2(n!) \)
Another application of counting … lower bounds

What's the **fastest possible worst case** for any sorting algorithm? $\log_2(n!)$

How big is that?

**Lemma:** For $n>1$,

$$\left(\frac{n}{2}\right)^{\frac{n}{2}} < n! < n^n$$
Another application of counting … lower bounds

What's the fastest possible worst case for any sorting algorithm? \( \log_2(n!) \)

How big is that?

**Lemma:** For \( n>1 \), \( \left( \frac{n}{2} \right)^{\frac{n}{2}} < n! < n^n \)

**Theorem:** \( \log_2(n!) \) is in \( \Theta(n \log n) \)

**Proof:** For \( n>1 \), taking logs of both sides in the lemma gives

\[
\frac{n}{2} \log \left( \frac{n}{2} \right) < \log_2(n!) < n \log n
\]

\[
\frac{1}{2} (n \log n - n \log 2) < \log_2(n!) < n \log n
\]
Another application of counting … lower bounds

What's the fastest possible worst case for any sorting algorithm? \[ \log_2(n!) \]

How big is that? \[ \Theta(n \log n) \]

Therefore, the best sorting algorithms will need \[ \Theta(n \log n) \] comparisons in the worst case.

It's impossible to have a comparison-based algorithm that does better than Merge Sort (in the worst case).
Announcements

HW #7 assigned
Due: Tuesday 2/7 11:59pm

MT2 handback in Gradescope
By Saturday, latest

HW #6 is in Gradescope
72-hour window for regrade requests

OH’s and 1-1 Slots
Use them well !!!