COMP 7720 - Online Algorithms

List Update & Compression

Shahin Kamali

Lecture 6 - Sep. 25, 2017
University of Manitoba
Review & Plan
Today’s objectives

- Announcements
- More list update algorithms
- List update & compression
• A1 is posted and is due October 4th at 11:59 pm
• Proposal is due October 6th at 11:59 pm
• Final exam will be in the exam period
The input is a set of requests to items in a list of length \( L \).
- The cost of accessing an item in index \( i \) is \( i \).
- Update the list to adjust it to the patterns in the input sequence of length \( n \) (\( n \gg L \)).
The input is a set of *requests* to items in a list of length $L$.

- The cost of accessing an item in index $i$ is $i$.

Update the list to adjust it to the patterns in the input sequence of length $n$ ($n \gg L$).

- Free exchanges: Move a requested item closer to the front without any cost.
- Paid exchanges: Swap positions of two consecutive items with a cost 1.

Total cost: access cost + paid-exchange cost
Move-To-Front algorithm

- After each access, move the requested item to the front.
- Competitive ratio of Move-To-Front is 2.
  - Potential function method!
- No deterministic algorithm can be better than 2-competitive.
Define the potential as the number of inversions.

See how potential changes after each action of Alg and Opt.

Bound the amortized cost (actual-cost + Δ potential) for each access in terms of OPT’s cost for that access.
How many inversions are removed by moving $x$ to front?

- Before moving to front, there are $i - 1$ items before $x$ in MTF list.
- At most $j - 1$ of them can also appear before $x$ in $OPT$ list (are non-inversions) $\Rightarrow$ the rest, at least, $i - 1 - (j - 1) = i - j$ are inversions $\Rightarrow$ By moving to front at least $i - j$ inversions are removed.

![Diagram showing list update and inversions](image-url)
How many inversions are added by moving $x$ to front?

- $x$ is in front of MTF list after the move and at position $j$ of $OPT$’s list
- items that appear after $x$ in MTF and before $x$ in $OPT$ are at most $j - 1$
- At most $j - 1$ inversions are added
Review of MTF Analysis

- When moving $x$ to front:
  - Actual cost is $i$, at least $i - j$ inversions are removed, at most $j - 1$ inversions are added.
Review of MTF Analysis

- When moving $x$ to front:
  - Actual cost is $i$, at least $i - j$ inversions are removed, at most $j - 1$ inversions are added
  - Assume $\text{OPT}$ makes $k$ paid exchanges.

\[
\Phi(t + 1) - \Phi(t) = \text{added inversions} - \text{removed inversions} \leq (j + k - 1) - (i - j) = 2j + k - i - 1.
\]

Amortized cost $= \text{actual cost} + \Phi(t + 1) - \Phi(t) \leq i + 2j + k - i - 1 = 2j + k - 1$.

Cost of $\text{OPT}$ is $j + k$ and amortized cost is $2j + k$. 

Lemma: At any time $t$, amortized cost $\leq 2\text{OPT}(t)$. 

COMP 7720 - Online Algorithms  List Update & Compression
Review of MTF Analysis

- When moving \( x \) to front:
  - Actual cost is \( i \), at least \( i - j \) inversions are removed, at most \( j - 1 \) inversions are added

- Assume \( \text{OPT} \) makes \( k \) paid exchanges.

\[
\Phi(t + 1) - \Phi(t) = \text{added inversions} - \text{removed inversions} \leq (j + k - 1) - (i - j) = 2j + k - i - 1.
\]
When moving $x$ to front:

- Actual cost is $i$, at least $i - j$ inversions are removed, at most $j - 1$ inversions are added.

- Assume $\text{OPT}$ makes $k$ paid exchanges.

- $\Phi(t + 1) - \Phi(t) = \text{added inversions} - \text{removed inversions} \leq (j + k - 1) - (i - j) = 2j + k - i - 1$.

- $amortized\_cost = actual\_cost + \Phi(t + 1) - \Phi(t) \leq i + 2j + k - i - 1 = 2j + k - 1$
Review of MTF Analysis

- When moving \( x \) to front:
  - Actual cost is \( i \), at least \( i - j \) inversions are removed, at most \( j - 1 \) inversions are added

- Assume \( \text{OPT} \) makes \( k \) paid exchanges.

- \( \Phi(t + 1) - \Phi(t) = \text{added inversions} - \text{removed inversions} \leq (j + k - 1) - (i - j) = 2j + k - i - 1 \).

- \( \text{amortized cost} = \text{actual cost} + \Phi(t + 1) - \Phi(t) \leq i + 2j + k - i - 1 = 2j + k - 1 \)

- Cost of \( \text{Opt} \) is \( j + k \) and \( \text{amortized cost} \) is \( 2j + k \).

**Lemma**

\[ \text{At any time } t, \text{ amortized cost}(t) \leq 2 \text{OPT}(t). \]
A quick example

- Assume at time $t$:
  - the list of MTF is
    \[ 8 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \]
  - the list of OPT is
    \[ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \]

- Assume $x$ is 3, which means $i = 6$ and $j = 3$.

- The number of removed inversions in this case is at least $i - j - 1 = 2$. In fact, it turns out to be 5 because all 1,2,4,5,6,7,8 form inversions with 3 which will be removed by moving 3 to the front.

- The number of new inversions will be at most $j - 1 = 2$. In fact, it is 0 as no new inversion is added.
Transpose algorithm

- Transpose: Move the accessed item one unit closer to the front.
- What is the competitive ratio?
Transpose algorithm

- **Transpose**: Move the accessed item one unit closer to the front.
- What is the competitive ratio?

\[
a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_{m-1} \rightarrow a_m
\]

sequence: \(a_m\)
**Transpose algorithm**

- **Transpose**: Move the accessed item one unit closer to the front.
- **What is the competitive ratio?**

\[ a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_m \rightarrow a_{m-1} \downarrow \]

sequence: \((a_m \ a_{m-1})\)
List Update Problem - A Review

Transpose algorithm

- Transpose: Move the accessed item one unit closer to the front.
- What is the competitive ratio?

\[ a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_{m-1} \rightarrow a_m \]

sequence: \((a_m \ a_{m-1})^k\)

- The cost of Transpose after \(n\) requests on a list of length \(m\) will be \(n \cdot m\)
- What does \(\text{OPT}\) do?

\[ \text{OPT does not move} \quad a_m \quad \text{and} \quad a_{m-1} \quad \text{to the front using 2} \quad (m-3) \quad \text{paid exchanges, and does not move them after.} \]

The cost for accesses to \(a_{m-1}\) and \(a_m\) are respectively 2 and 1.

The cost of \(\text{OPT}\) will be \(2 \cdot (m-3) + n/2 \cdot 1 + n/2 \cdot 2 \approx 1.5n + 2m\).

The competitive ratio will be at least \(n \cdot m \frac{1}{1.5n+2m} \in O(m)\).
Transpose algorithm

- **Transpose**: Move the accessed item one unit closer to the front.

- **What is the competitive ratio?** sequence: \((a_m \ a_{m-1})^k\)

- The cost of Transpose after \(n\) requests on a list of length \(m\) will be \(n \cdot m\)

- **What does OPT do?**
  - It moves \(a_m\) and \(a_{m-1}\) to the front using \(2m - 3\) paid exchanges, and does not move them after.
  - The cost for accesses to \(a_{m-1}\) and \(a_m\) are respectively 2 and 1.
  - The cost of \(OPT\) will be \(2m - 3 + n/2 \cdot 1 + n/2 \cdot 2 \approx 1.5n + 2m\).
Transpose algorithm

- **Transpose**: Move the accessed item one unit closer to the front.

- What is the competitive ratio? sequence: \((a_m \ a_{m-1})^k\)

- The cost of Transpose after \(n\) requests on a list of length \(m\) will be \(n \cdot m\).

- What does \(\text{OPT}\) do?
  - It moves \(a_m\) and \(a_{m-1}\) to the front using \(2m - 3\) paid exchanges, and does not move them after.
  - The cost for accesses to \(a_{m-1}\) and \(a_m\) are respectively 2 and 1.
  - The cost of \(\text{OPT}\) will be \(2m - 3 + n/2 \cdot 1 + n/2 \cdot 2 \approx 1.5n + 2m\).

- The competitive ratio will be at least \(\frac{n \cdot m}{1.5n + 2m} \in O(m)\).
Other deterministic algorithms

Consider an algorithm that moves a requested item at index $i$ half way to front.
Consider an algorithm that moves a requested item at index $i$ half way to front.

What is the competitive ratio of this algorithm?
Other deterministic algorithms

- Consider an algorithm that moves a requested item at index \( i \) half way to front.

- What is the competitive ratio of this algorithm?

\[
a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_m \rightarrow a_{m+1} \rightarrow a_{m+2} \rightarrow \ldots \rightarrow a_{2m-1} \rightarrow a_{2m}
\]

sequence: \( a_{2m} \)
Consider an algorithm that moves a requested item at index $i$ half way to front.

What is the competitive ratio of this algorithm?

$$a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_m \rightarrow a_{2m} \rightarrow a_{m+1} \rightarrow \ldots \rightarrow a_{2m-2} \rightarrow a_{2m-1}$$

sequence: $a_{2m} \ a_{2m-1}$
Other deterministic algorithms

- Consider an algorithm that moves a requested item at index $i$ half way to front.

- What is the competitive ratio of this algorithm?

\[ a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_m \rightarrow a_{2m-1} \rightarrow a_{2m} \rightarrow \ldots \rightarrow a_{2m-3} \rightarrow a_{2m-2} \]

Sequence: \( a_{2m} \ a_{2m-1} \ a_{2m-2} \)
Other deterministic algorithms

- Consider an algorithm that moves a requested item at index \( i \) half way to front.

- What is the competitive ratio of this algorithm?

\[
a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_m \rightarrow a_{m+2} \rightarrow a_{m+3} \rightarrow \ldots \rightarrow a_{2m} \rightarrow a_{m+1}
\]

sequence: \( a_{2m} \ a_{2m-1} \ a_{2m-2} \ldots \ a_{m+1} \)
Consider an algorithm that moves a requested item at index $i$ half way to front.

What is the competitive ratio of this algorithm?

$$a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_m \rightarrow a_{m+1} \rightarrow a_{m+2} \rightarrow \ldots \rightarrow a_{2m-1} \rightarrow a_{2m}$$

Sequence: \((a_{2m} \ a_{2m-1} \ a_{2m-2} \ldots \ a_{m+1})^k\)
Consider an algorithm that moves a requested item at index $i$ half way to front.

What is the competitive ratio of this algorithm?

$$a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_m \rightarrow a_{m+1} \rightarrow a_{m+2} \rightarrow \ldots \rightarrow a_{2m-1} \rightarrow a_{2m}$$

sequence: $(a_{2m} \ a_{2m-1} \ a_{2m-2} \ldots \ a_{m+1})^k$

→ Alg’s cost after $n$ requests is $n \cdot 2m$
Other deterministic algorithms

Consider an algorithm that moves a requested item at index $i$ half way to front.

What is the competitive ratio of this algorithm?

$$a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_m \rightarrow a_{m+1} \rightarrow a_{m+2} \rightarrow \ldots \rightarrow a_{2m-1} \rightarrow a_{2m}$$

sequence: $(a_{2m} \ a_{2m-1} \ a_{2m-2} \ldots \ a_{m+1})^k$

$\rightarrow$ Alg’s cost after $n$ requests is $n \cdot 2m$

OPT moves the requested $m$ items to the front (using $O(m^2)$ paid exchanges at the beginning) and does not move after that.

$$a_{m+1} \rightarrow a_{m+2} \rightarrow \ldots \rightarrow a_{2m-1} \rightarrow a_{2m} \rightarrow a_1 \rightarrow a_2 \ldots \rightarrow a_m$$
Consider an algorithm that moves a requested item at index \( i \) half way to front.

What is the competitive ratio of this algorithm?

\[
a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_m \rightarrow a_{m+1} \rightarrow a_{m+2} \rightarrow \ldots \rightarrow a_{2m-1} \rightarrow a_{2m}
\]

sequence: \((a_{2m} \ a_{2m-1} \ a_{2m-2} \ldots a_{m+1})^k\)

→ Alg’s cost after \( n \) requests is \( n \cdot 2m \)

\( \text{OPT} \) moves the requested \( m \) items to the front (using \( O(m^2) \) paid exchanges at the beginning) and does not move after that.

\[
a_{m+1} \rightarrow a_{m+2} \rightarrow \ldots \rightarrow a_{2m-1} \rightarrow a_{2m} \rightarrow a_1 \rightarrow a_2 \ldots \rightarrow a_m
\]

Opt’s cost after \( n \) requests is \( n \cdot m/2 + O(m^2) \).
Other deterministic algorithms

- Consider an algorithm that moves a requested item at index $i$ half way to front.

- What is the competitive ratio of this algorithm?

  $a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_m \rightarrow a_{m+1} \rightarrow a_{m+2} \rightarrow \ldots \rightarrow a_{2m-1} \rightarrow \downarrow a_{2m}$

  sequence: $(a_{2m} \ a_{2m-1} \ a_{2m-2} \ldots a_{m+1})^k$

  $\rightarrow$ Alg’s cost after $n$ requests is $n \cdot 2m$

- Opt moves the requested $m$ items to the front (using $O(m^2)$ paid exchanges at the beginning) and does not move after that.

  $a_{m+1} \rightarrow a_{m+2} \rightarrow \ldots \rightarrow a_{2m-1} \rightarrow a_{2m} \rightarrow a_1 \rightarrow a_2 \ldots \rightarrow a_m$

  Opt’s cost after $n$ requests is $n \cdot m/2 + O(m^2)$.

### Theorem

The competitive ratio is at least $\frac{n \cdot 2m}{n \cdot m/2 + O(m^2)} \approx 4$
Consider an algorithm that moves a requested item $x$ to the front of the list on every-other-access to $x$. The competitive ratio of this algorithm is indeed 2.5.
Other deterministic algorithms (cntd.)

- Consider an algorithm that moves a requested item \( x \) to the front of the list on every-other-access to \( x \).
- The competitive ratio of this algorithm is indeed 2.5.

**Theorem**

*The best existing deterministic algorithms are Move-To-Front and Timestamp (and some algorithms which combine them). Other list update algorithms do not achieve competitive ratio of 2.*
Randomized list update algorithms

- What is a good randomized algorithm?
- Algorithm 1: on access to an item, move it to front with probability $1/2$. This turns out to have a competitive ratio of 2.
Randomized list update algorithms

What is a good randomized algorithm?

Algorithm 1: on access to an item, move it to front with probability 1/2.

- This turns out to have a competitive ratio of 2.

Is it good?
Consider an algorithm that maintains a bit for each item:

- Initially all bits are set uniformly at random.

- On an access to $x$, flip the bit of $x$ and move it to front if the bit becomes ‘1’.
Consider an algorithm that maintains a bit for each item

- Initially all bits are set uniformly at random.
- On an access to $x$, flip the bit of $x$ and move it to front if the bit becomes ‘1’.

sequence:

\[ a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \]


sequence: $d$
Consider an algorithm that maintains a bit for each item
- Initially all bits are set uniformly at random.
- On an access to $x$, flip the bit of $x$ and move it to front if the bit becomes ‘1’.

\[
\begin{align*}
  a & \rightarrow b \rightarrow c \rightarrow d \rightarrow e \\
\end{align*}
\]


sequence: $d$
BIT algorithm

Consider an algorithm that maintains a bit for each item
- Initially all bits are set uniformly at random.
- On an access to $x$, flip the bit of $x$ and move it to front if the bit becomes ‘1’.

\[
\rightarrow d \rightarrow a \rightarrow b \rightarrow c \rightarrow e
\]


sequence: $d$
Consider an algorithm that maintains a bit for each item.

- Initially all bits are set uniformly at random.
- On an access to $x$, flip the bit of $x$ and move it to front if the bit becomes ‘1’.

\[
\begin{align*}
d &\to a \to b \to c \to e
\end{align*}
\]

bits: \(a[0] \ b[1] \ c[1] \ d[1] \ e[1]\)

sequence: \(d \ e\)
Consider an algorithm that maintains a bit for each item

- Initially all bits are set uniformly at random.
- On an access to $x$, flip the bit of $x$ and move it to front if the bit becomes ‘1’.

\[
\rightarrow d \rightarrow a \rightarrow b \rightarrow c \rightarrow \downarrow e
\]


sequence: $d \ e$
Theorem

BIT has a competitive ratio of 1.75.
Theorem

**BIT has a competitive ratio of 1.75.**

- The best randomized algorithm is COMB with a competitive ratio of 1.6.
- Apply Bit with probability 0.8 and TimeStamp with probability 0.2.
The best randomized algorithm is COMB with a competitive ratio of 1.6.
- Apply Bit with probability 0.8 and TimeStamp with probability 0.2.
- No randomized algorithm can have a competitive ratio better than 1.5.
- There is a gap between the competitive ratio of the best algorithm (1.6 of COMB) and the best lower bound (1.5).
We use a **projective property** to prove upper bounds for competitive ratios of most algorithms.

An algorithm is projective if the relative order of any two items only depend of accesses to those items.
We use a **projective property** to prove upper bounds for competitive ratios of most algorithms.

An algorithm is projective if the relative order of any two items only depend on accesses to those items.

- Move-To-Front: an item $x$ appears before $y$ if and only if the last access to $x$ is more recent than the last access to $y$ (and request to another item $z$ does not change it).
We use a projective property to prove upper bounds for competitive ratios of most algorithms.

An algorithm is projective if the relative order of any two items only depend on accesses to those items.

- Move-To-Front: an item $x$ appears before $y$ if and only if the last access to $x$ is more recent than the last access to $y$ (and request to another item $z$ does not change it).

In order to show a projective algorithm is $c$-competitive, it suffices to show that it is $c$-competitive for lists of length 2.

- Most existing algorithms for list-update are projective.
How many bits of advice to achieve an optimal solution?

- For a sequence of length $n$, $O(n)$ bits are sufficient and $\Omega(n)$ are required!
  - We see the proof in another lecture (hopefully).
- What about advice of small size?
Consider the following three deterministic algorithms:

- **MTF**: Move-To-Front (competitive ratio is 2)
- **MTFO**: Move-To-Front on every-other-access on *odd* accesses (competitive ratio is 2.5).
- **MTFE**: Move-To-Front on every-other-access on *even* accesses (competitive ratio is 2.5).

For any sequence, at least one of these algorithms has a cost no more than \( \frac{1}{6} \) times the cost of \( \text{Opt} \). This is better than any possible deterministic algorithm.
Consider the following three deterministic algorithms:

- **MTF**: Move-To-Front (competitive ratio is 2)
- **MTFO**: Move-To-Front on every-other-access on odd accesses (competitive ratio is 2.5).
- **MTFE**: Move-To-Front on every-other-access on even accesses (competitive ratio is 2.5).

The better algorithm among the above algorithms has a competitive ratio of 1.66.

- For any sequence, at least one of these algorithms is has a cost no more than 1.66 times the cost of \( \text{OPT} \).
- This is better than any possible deterministic algorithm.

What does it mean in terms of advice?

- With only two bits of advice, the better algorithm can be specified.
For deterministic case, Move-To-Front and TimeStamp have competitive ratio of 2, and it is the best ratio that a deterministic algorithm can have.
For deterministic case, Move-To-Front and TimeStamp have competitive ratio of 2, and it is the best ratio that a deterministic algorithm can have.

For randomized case, COMB is the best existing algorithm with competitive ratio 1.6. No randomized algorithm can be better than 1.5-competitive. There is a gap here!
For deterministic case, Move-To-Front and TimeStamp have competitive ratio of 2, and it is the best ratio that a deterministic algorithm can have.

For randomized case, COMB is the best existing algorithm with competitive ratio 1.6. No randomized algorithm can be better than 1.5-competitive. There is a gap here!

Potential Project topics:
- Continue studying advice complexity of list update algorithms
- What algorithms complement each other in a better way that we saw for MTF, MTFE, MTFO?
List Update Algorithms Conclusions

- For deterministic case, Move-To-Front and TimeStamp have competitive ratio of 2, and it is the best ratio that a deterministic algorithm can have.

- For randomized case, COMB is the best existing algorithm with competitive ratio 1.6. No randomized algorithm can be better than 1.5-competitive. There is a gap here!

Potential Project topics:

- Continue studying advice complexity of list update algorithms
  - What algorithms complement each other in a better way that we saw for MTF, MTFE, MTFO?
- Assume we do not allow free exchanges? What will be the best algorithm? (paid exchange model)
List Update & Compression
One important application of list update is in data compression.

Given a data-sequence (e.g., an English text), we want to compress it.

We should be able to recover the exact text from the compressed one → Lossless compression.
Basics of Compression

How to encode some data (e.g., an English text)?
How to encode some data (e.g., an English text)?

- Solution 1: write the ASCII or Unicode code for each character
  - The code for ‘A’ has the same length as ‘Q’.

- Solution 2: let more common characters have smaller length
  - In Huffman code ‘A’ is encoded shorter than ‘Q’.
  - The ‘context’ is ignored: the code for ‘TH’ is longer than ‘Q’.
How to encode some data (e.g., an English text)?

Solution 1: write the ASCII or Unicode code for each character

- The code for ‘A’ has the same length as ‘Q’.

Solution 2: let more common characters have smaller length

- In Huffman code ‘A’ is encoded shorter than ‘Q’ :)

How to encode some data (e.g., an English text)?

Solution 1: write the ASCII or Unicode code for each character

- The code for ‘A’ has the same length as ‘Q’.

Solution 2: let more common characters have smaller length

- In Huffman code ‘A’ is encoded shorter than ‘Q’ :) 
- The ‘context’ is ignored: the code for ‘TH’ is longer than ‘Q’ :(
MTF-based Compression

- Solutions 3: use MTF index to encode the characters

\[ S = \text{INEFFICIENCIES} \]

\[ C = \]
List Update & Compression

MTF-based Compression

- Solutions 3: use MTF index to encode the characters

\[
S = \text{INEFFICIENCIES}
\]

\[
C = 8
\]
MTF-based Compression

- Solutions 3: use MTF index to encode the characters

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
I A B C D E F G H J K L M N O P Q R S T U V W X Y Z
```

\[
S = \text{INEFFICIENCIES}
\]

\[
C = 8 \ 13
\]
MTF-based Compression

- Solutions 3: use MTF index to encode the characters

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>I</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>

$$S = \text{INEFFICIENCIES}$$

$$C = 8 \ 13 \ 6$$
MTF-based Compression

Solutions 3: use MTF index to encode the characters

\[
S = \text{INEFFICIENCIES}
\]

\[
C = 8\ 13\ 6\ 7
\]
MTF-based Compression

Solutions 3: use MTF index to encode the characters

\[ S = \text{INEFFICIENCIES} \]

\[ C = 8\ 13\ 6\ 7\ 0 \]
MTF-based Compression

- Solutions 3: use MTF index to encode the characters

\[
S = \text{INEFFICIENCIES}
\]

\[
C = 8 \ 13 \ 6 \ 7 \ 0 \ 3
\]
MTF-based Compression

- Solutions 3: use MTF index to encode the characters

\[
S = \text{INEFFICIENCIES}
\]

\[
C = 8\ 13\ 6\ 7\ 0\ 3\ 6
\]
MTF-based Compression

- Solutions 3: use MTF index to encode the characters

```plaintext
S = INEFFICIENCIES

C = 8 13 6 7 0 3 6 1
```
MTF-based Compression

- Solutions 3: use MTF index to encode the characters

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
S E I C N F A B D G H J K L M O P Q R T U V W X Y Z
```

\[
S = \text{INEFFICIENCIES}
\]

\[
C = 8 \ 13 \ 6 \ 7 \ 0 \ 3 \ 6 \ 1 \ 3 \ 4 \ 3 \ 3 \ 3 \ 1 8
\]

- What does a run in \( S \) encode to in \( C \)?

- This results in good compression if we have high locality in the input.
Increase locality using Burrows-Wheeler Transform!
Burrows-Wheeler Transform

- Increase locality using Burrows-Wheeler Transform!

- How it works?
  - Create all rotations of a given sequence.
  - Sort those rotations into lexicographic order.
  - Take as output the last column!

Why it is useful?
- Creates output with high locality!
Burrows-Wheeler Transform

- Increase locality using Burrows-Wheeler Transform!
- How it works?
  - Create all rotations of a given sequence.
  - Sort those rotations into lexicographic order.
  - Take as output the last column!
- Why it is useful?
  - Creates output with high locality!
  - This is reversible

<table>
<thead>
<tr>
<th>Original</th>
<th>1st Rotation</th>
<th>2nd Rotation</th>
<th>3rd Rotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>banana</td>
<td>banana$</td>
<td>anana$b</td>
<td>nana$ba</td>
</tr>
<tr>
<td></td>
<td>anana$ban</td>
<td>ana$ban</td>
<td>na$bana</td>
</tr>
<tr>
<td></td>
<td>a$banan</td>
<td>$banana</td>
<td>na$bana</td>
</tr>
</tbody>
</table>

BWT(banana) = annb$aa
Why Burrows-Wheeler outputs have high locality?

Consider an example of English text; there are many ’the’s such text.
Why Burrows-Wheeler outputs have high locality?

Consider an example of English text; there are many 'the's such text.

- When we sort, rotations starting with 'he' appear together.
- The last column for these rotations has character 't', i.e., we will have a run of t's
B-Zip2 compression scheme

- Assume we want to compress a data sequence $S$.
- Apply BWT on $S$ to increase its locality
  - $baanana$ $\rightarrow$ $annb$ $aa$

You expect to see a lot of 0's and 1's. Use run-length encoding to store these indices.
Assume we want to compress a data sequence $S$.
Apply BWT on $S$ to increase its locality

$baanana\$ \rightarrow annb\$aa$

Apply MTF on BWT output and encode the indices in the list

$a \rightarrow b \rightarrow \ldots \rightarrow n \rightarrow \ldots \rightarrow z \rightarrow \$

$annb\$aa \rightarrow 0\ 13\ 0\ 2\ 27\ 3\ 0$

You expect to see a lot of 0’s and 1’s.
B-Zip2 compression scheme

- Assume we want to compress a data sequence $S$.
- Apply BWT on $S$ to increase its locality
  - $bananana$ $\rightarrow$ $annb$ $aa$
- Apply MTF on BWT output and encode the indices in the list
  
  $$a \rightarrow b \rightarrow \ldots \rightarrow n \rightarrow \ldots \rightarrow z \rightarrow \$

  $$annb$aa \rightarrow 0\ 13\ 0\ 2\ 27\ 3\ 0$$

- You expect to see a lot of 0’s and 1’s.
- Use run-length encoding to store these indices
  - Write down the length of each run!
  - $\langle 1\ 1\ 1\ 1\ 1\ 2\ 2\ 2\ 2\ 1\ 1\ 4\ 4\ 4 \rangle \rightarrow \langle (1\ 5)\ (2\ 4)\ (1\ 2)\ (4\ 3) \rangle$
decompression

- Assume we are given the indices in the compressed file

\[0, 13, 0, 2, 27, 3, 0 = \text{annb} \]
Assume we are given the indices in the compressed file.

Follow the steps of MTF and write down the character of each index:

\[ a \rightarrow b \rightarrow \ldots \rightarrow n \rightarrow \ldots \rightarrow z \rightarrow $ \]

\[ 0 \ 13 \ 0 \ 2 \ 27 \ 3 \ 0 \Rightarrow annb$aa \]
decompression

- Assume we are given the indices in the compressed file
- Follow the steps of MTF and write down the character of each index

\[ a \rightarrow b \rightarrow \ldots \rightarrow n \rightarrow \ldots \rightarrow z \rightarrow $ \]

\[ 0 \ 13 \ 0 \ 2 \ 27 \ 3 \ 0 \ \rightarrow \ annb$aa \]

- Can we replace MTF by another algorithm?
Assume we are given the indices in the compressed file.

Follow the steps of MTF and write down the character of each index:

\[ a \rightarrow b \rightarrow \ldots \rightarrow n \rightarrow \ldots \rightarrow z \rightarrow $ \]

\[ 0 13 0 2 27 3 0 \rightarrow annb$aa \]

Can we replace MTF by another algorithm?

- Yes, any online list update algorithm can be used.
- The quality of compression might change!
Assume we are given the indices in the compressed file.
Follow the steps of MTF and write down the character of each index.

\[ a \rightarrow b \rightarrow \ldots \rightarrow n \rightarrow \ldots \rightarrow z \rightarrow \$ \]

0 13 0 2 27 3 0 \(\Rightarrow\) annb$aa$

Can we replace MTF by another algorithm?
- Yes, any online list update algorithm can be used.
- The quality of compression might change!

What about an algorithm with advice?
Any list update algorithm with advice can be used to replace MTF in bzip2.

The advice bits should be included in the compressed file.

- E.g., we store two bits of advice in the compressed file for text $T$.
- Advice bits indicate whether MTF, MTFE, or MTFO is better and used for compressing $T$. 
List Update & Compression

List Update, Advice, & Compression

- Any list update algorithm with advice can be used to replace MTF in bzip2.

- The advice bits should be included in the compressed file.
  - E.g., we store two bits of advice in the compressed file for text $T$.
  - Advice bits indicate whether MTF, MTFE, or MTFO is better and used for compressing $T$.

- Project topic: implement and compare different list-update algorithms for compression purposes. Explore, how advice can be helpful in achieving more compressed results.