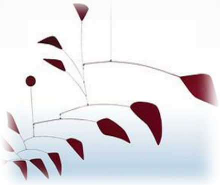


EECS 4101-5101
Advanced Data Structures

Shahin Kamali

Topic 1d - Self Adjusting Linked Lists & Data Compression

York University

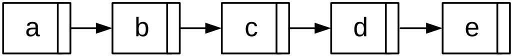
Picture is from the cover of the textbook CLRS.

List Update Problem

List Accessing Problem

- The input is a set of *requests* to items in a list.
- The cost of accessing an item in index i is i.

$$< \mathbf{d\ b\ b\ d\ c\ a\ c} >$$

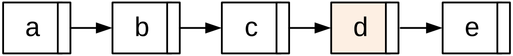


List Accessing Problem

- The input is a set of *requests* to items in a list.
- The cost of accessing an item in index i is i.

< d b b d c a c >

cost: 4

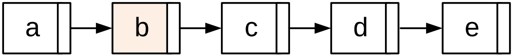


List Accessing Problem

- The input is a set of *requests* to items in a list.
- The cost of accessing an item in index i is i.

$$< \mathbf{d}\ \mathbf{b}\ \mathbf{b}\ \mathbf{d}\ \mathbf{c}\ \mathbf{a}\ \mathbf{c} >$$

cost: $4 + 2$

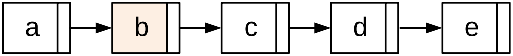


List Accessing Problem

- The input is a set of *requests* to items in a list.
- The cost of accessing an item in index i is i.

< d b b d c a c >

cost: 4+2+2

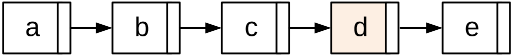


List Accessing Problem

- The input is a set of *requests* to items in a list.
- The cost of accessing an item in index i is i.

$$< \mathbf{d}\ \mathbf{b}\ \mathbf{b}\ \mathbf{d}\ \mathbf{c}\ \mathbf{a}\ \mathbf{c} >$$

cost: $4 + 2 + 2 + 4$

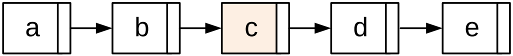


List Accessing Problem

- The input is a set of *requests* to items in a list.
- The cost of accessing an item in index i is i.

$$< \mathbf{d}\ \mathbf{b}\ \mathbf{b}\ \mathbf{d}\ \mathbf{c}\ \mathbf{a}\ \mathbf{c} >$$

cost: $4+2+2+4+3$

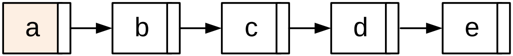


List Accessing Problem

- The input is a set of *requests* to items in a list.
- The cost of accessing an item in index i is i.

< d b b d c a c >

cost: 4+2+2+4+3+1

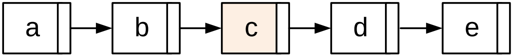


List Accessing Problem

- The input is a set of *requests* to items in a list.
- The cost of accessing an item in index i is i.

$$< \mathbf{d}\ \mathbf{b}\ \mathbf{b}\ \mathbf{d}\ \mathbf{c}\ \mathbf{a}\ \mathbf{c} >$$

cost: $4+2+2+4+3+1+3 = 19$



Introduction to List Update

- An instance of **self-adjusting data structures**.
- The structure adjusts itself based on the input queries.

Introduction to List Update

- An instance of **self-adjusting data structures**.
- The structure adjusts itself based on the input queries.
- List update was formulated in 1984 by Sleator and Tarjan
    - This result of Sleator and Tarjan made online algorithms popular in the following two decades
    - There are applications in data-compression!

Self-Adjusting Lists

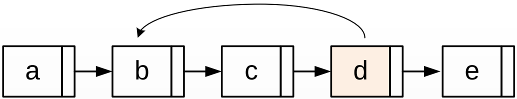
- Update a list of length k to adjust it to the patterns in the input sequence of length n ($n \gg k$).

Self-Adjusting Lists

- Update a list of length k to adjust it to the patterns in the input sequence of length n ($n \gg k$).
  - Free exchanges: Move a requested item closer to the front without any cost.

< d b b d c a c >

cost: 4

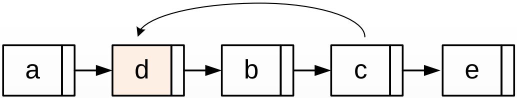


Self-Adjusting Lists

- Update a list of length k to adjust it to the patterns in the input sequence of length n ($n \gg k$).
    - Free exchanges: Move a requested item closer to the front without any cost.

$< \textcolor{red}{d}\ b\ b\ d\ c\ a\ c >$

cost: 4

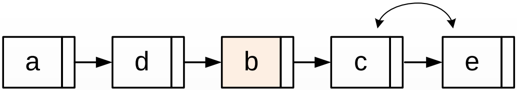


Self-Adjusting Lists

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

< d b b d c a c >

cost: 4

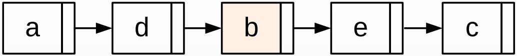


Self-Adjusting Lists

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

$< d\ b\ b\ d\ c\ a\ c >$

cost: 4



List Update Problem

- In the offline version of the problem, you have access to the whole set at the beginning.
    - The problem is NP-hard.

List Update Problem

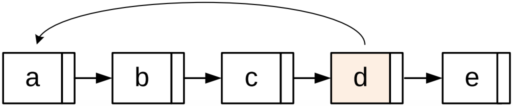
- In the offline version of the problem, you have access to the whole set at the beginning.
  - The problem is NP-hard.
- In the online setting, the requests appear in an online, sequential manner.
  - An online algorithm should reorder the list without looking at the future requests.

Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

< d b b d c a c >

cost: 4



Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

$< d\ b\ b\ d\ c\ a\ c >$

cost: 4

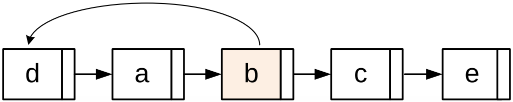
d → a → b → c → e

Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

< d b b d c a c >

cost: 4+3

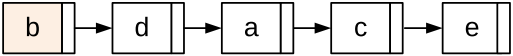


Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

$$< \mathrm{d}\ \mathrm{b}\ \mathrm{b}\ \mathrm{d}\ \mathrm{c}\ \mathrm{a}\ \mathrm{c} >$$

cost: 4+3

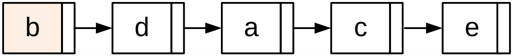


Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

$$< d\ b\ b\ d\ c\ a\ c >$$

cost: 4+3+1

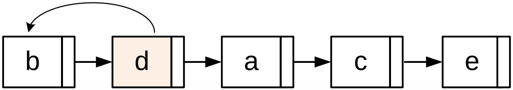


Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

$< d\ b\ b\ d\ c\ a\ c >$

cost: 4+3+1+2

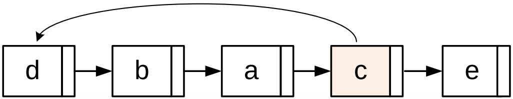


Move-To-Front (MTF)

- After each access, move the requested item to the front.
    - It only uses free exchanges.

< d b b d c a c >

cost: 4+3+1+2+4

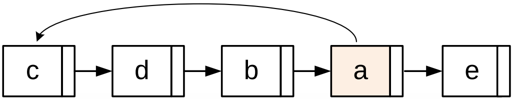


Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

$$< \mathbf{d\ b\ b\ d\ c\ a\ c} >$$

cost: 4+3+1+2+4+4

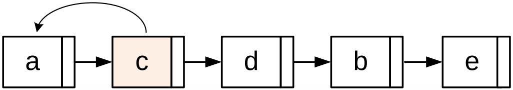


Move-To-Front (MTF)

- After each access, move the requested item to the front.
    - It only uses free exchanges.

< d b b d c a c >

cost: 4+3+1+2+4+4+2

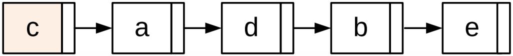


Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

$$< \text{d b b d c a } \textcolor{red}{\text{c}} >$$

cost: 4+3+1+2+4+4+2

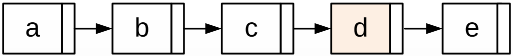


TIMESTAMP

- After an access to x, move x to the front of the first item y which has been requested at most once since the last access to x.
  - Do nothing on the first access to x or if such an item y does not exist.

< d b b d c a c >

cost: 4

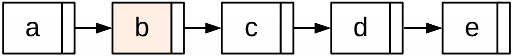


TIMESTAMP

- After an access to x, move x to the front of the first item y which has been requested at most once since the last access to x.
  - Do nothing on the first access to x or if such an item y does not exist.

$$< \mathbf{d}\ \mathbf{b}\ \mathbf{b}\ \mathbf{d}\ \mathbf{c}\ \mathbf{a}\ \mathbf{c} >$$

cost: 4+2

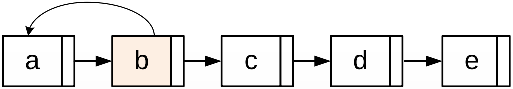


TIMESTAMP

- After an access to x, move x to the front of the first item y which has been requested at most once since the last access to x.
    - Do nothing on the first access to x or if such an item y does not exist.

< d b b d c a c >

cost: 4+2+2

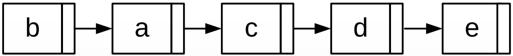


TIMESTAMP

- After an access to x, move x to the front of the first item y which has been requested at most once since the last access to x.
  - Do nothing on the first access to x or if such an item y does not exist.

$< \mathbf{d\ b\ b\ d\ c\ a\ c} >$

cost: 4+2+2

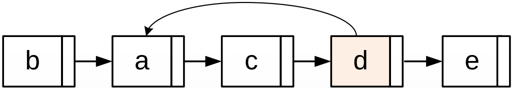


TIMESTAMP

- After an access to x, move x to the front of the first item y which has been requested at most once since the last access to x.
  - Do nothing on the first access to x or if such an item y does not exist.

< d b b d c a c >

cost: 4+2+2+4

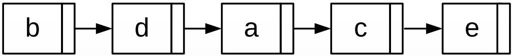


TIMESTAMP

- After an access to x, move x to the front of the first item y which has been requested at most once since the last access to x.
  - Do nothing on the first access to x or if such an item y does not exist.

$$< \mathbf{d\ b\ b\ d\ c\ a\ c} >$$

cost: 4+2+2+4

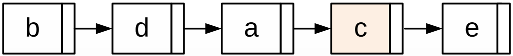


TIMESTAMP

- After an access to x, move x to the front of the first item y which has been requested at most once since the last access to x.
    - Do nothing on the first access to x or if such an item y does not exist.

< d b b d c a c >

cost: 4+2+2+4+4

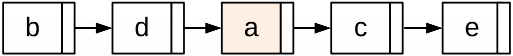


TIMESTAMP

- After an access to x, move x to the front of the first item y which has been requested at most once since the last access to x.
  - Do nothing on the first access to x or if such an item y does not exist.

$$< d\ b\ b\ d\ c\ a\ c >$$

cost: 4+2+2+4+4+3

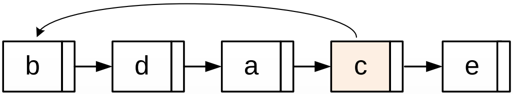


TIMESTAMP

- After an access to x, move x to the front of the first item y which has been requested at most once since the last access to x.
    - Do nothing on the first access to x or if such an item y does not exist.

< **d b b d c a c** >

cost: 4+2+2+4+4+3+4

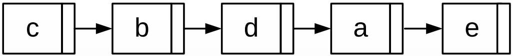


TIMESTAMP

- After an access to x, move x to the front of the first item y which has been requested at most once since the last access to x.
  - Do nothing on the first access to x or if such an item y does not exist.

< d b b d c a **c** >

cost: 4+2+2+4+4+3+4



Optimal Static Algorithm

- Look at the sequence of requests, sort items by the frequency of their accesses.
    - The most accessed item will be at the beginning of the list.

Optimal Static Algorithm

- Look at the sequence of requests, sort items by the frequency of their accesses.
  - The most accessed item will be at the beginning of the list.
- The cost of the algorithm would be at most $nk/2$.

Lower Bound for Competitive Ratio

- Consider a **cruel** sequence in which the adversary always asks for the last item in the list!
- What will be the cost of the algorithm?

Lower Bound for Competitive Ratio

- Consider a **cruel** sequence in which the adversary always asks for the last item in the list!
- What will be the cost of the algorithm?
    - It will be nk.

Lower Bound for Competitive Ratio

- Consider a **cruel** sequence in which the adversary always asks for the last item in the list!
- What will be the cost of the algorithm?
  - It will be nk.
- What is the cost of Opt?
  - We know the optimal static algorithm has a cost of $n(k+1)/2$.
  - So the cost of Opt is no more than $n(k+1)/2$.

Lower Bound for Competitive Ratio

- Consider a **cruel** sequence in which the adversary always asks for the last item in the list!
- What will be the cost of the algorithm?
  - It will be nk.
- What is the cost of Opt?
  - We know the optimal static algorithm has a cost of $n(k+1)/2$.
  - So the cost of Opt is no more than $n(k+1)/2$.
- The competitive ratio of any online list update algorithm is at least $\frac{nk}{nk/2} = 2$.

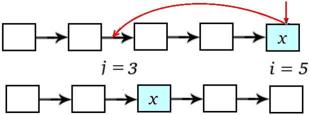
Competitiveness of MTF

On the Nature of Opt

- There is an optimal algorithm that only uses paid exchanges!

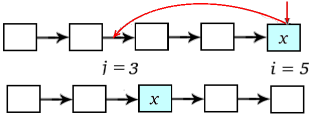
On the Nature of Opt

- There is an optimal algorithm that only uses paid exchanges!
- Assume Opt uses a free exchange after accessing item x at position i to move it closer to the front to position j
  - The cost will be i.

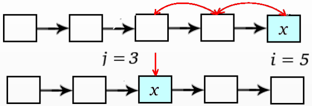


On the Nature of Opt

- There is an optimal algorithm that only uses paid exchanges!
- Assume Opt uses a free exchange after accessing item x at position i to move it closer to the front to position j
  - The cost will be i.



- In a new scheme, **before the access** apply $i - j$ paid exchanges to move i to position j.
  - The new cost will be $i - j$ for paid exchanges and j for the access, which sums to i



Competitiveness of MTF

Theorem

Move-To-Front has competitive ratio of 2.

- We prove it through **potential function method**
  - And it takes a few slides :'-)

Potential Function (Review)

- Assume you face an online problem where the input is a sequence of requests that require you to change the **state** of a problem.
  - These states should be finite and independent of the input length!

Potential Function (Review)

- Assume you face an online problem where the input is a sequence of requests that require you to change the **state** of a problem.
  - These states should be finite and independent of the input length!
- Define a 'potential' as a function of the state of the algorithm and that of Opt (e.g. no. inversions).
  - This is the critical part :-)

Potential Function (Review)

- Assume you face an online problem where the input is a sequence of requests that require you to change the **state** of a problem.
  - These states should be finite and independent of the input length!
- Define a 'potential' as a function of the state of the algorithm and that of Opt (e.g. no. inversions).
  - This is the critical part :-)
- Define the amortized cost at a given time t as the actual cost algorithm plus the difference in potential after the request is served (same for all problems).

Potential Function (Review)

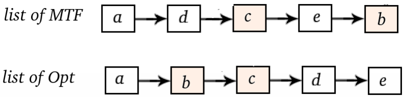
- Assume you face an online problem where the input is a sequence of requests that require you to change the **state** of a problem.
  - These states should be finite and independent of the input length!
- Define a 'potential' as a function of the state of the algorithm and that of Opt (e.g. no. inversions).
  - This is the critical part :-)
- Define the amortized cost at a given time t as the actual cost algorithm plus the difference in potential after the request is served (same for all problems).
- The potential should be defined in a way so that you can show $amortized\_cost(t) \leq c\,\mathrm{Opt}(t)$.

Potential Function (Review)

- Assume you face an online problem where the input is a sequence of requests that require you to change the **state** of a problem.
  - These states should be finite and independent of the input length!
- Define a 'potential' as a function of the state of the algorithm and that of Opt (e.g. no. inversions).
  - This is the critical part :-)
- Define the amortized cost at a given time t as the actual cost algorithm plus the difference in potential after the request is served (same for all problems).
- The potential should be defined in a way so that you can show $amortized\_cost(t) \leq c\,\text{Opt}(t)$.
- Using a telescopic sum, the competitive ratio will be at most c (same for all problems).

Inversions

- At a given time, two items x and y form an **inversion** if their relative order is different in the lists of MTF and Opt
- **Question:** what is the maximum number of inversions for a list of length k?
  - (a) $k/2$ (c) $k(k-1)/2$ (c) k (d) $k^2 - k/2$

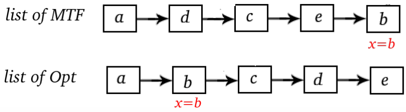


Potential Function

- Assume MTF and Opt are running the same input in parallel
- Assume we are at the t'th request, and there is a request to an item x.
- Define the **potential** at time t to be the total number of inversions before accessing x.

Potential Function

- Assume MTF and Opt are running the same input in parallel
- Assume we are at the t'th request, and there is a request to an item x.
- Define the **potential** at time t to be the total number of inversions before accessing x.



- Inversions are $(b, c), (b, d), (b, e), (c, d)$
- So, $\Phi(t) = 4$.

Amortized Cost

- Intuitively, if we have a high potential, we are in bad state.

Amortized Cost

- Intuitively, if we have a high potential, we are in bad state.
- Define the **amortized cost** at time t (when answering the tth request) as:

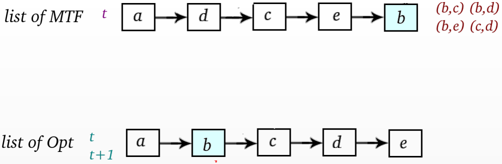
$$amortized\_cost(t) = actual\_cost(t) + \Phi(t+1) - \Phi(t)$$

Amortized Cost

- Intuitively, if we have a high potential, we are in bad state.
- Define the **amortized cost** at time t (when answering the tth request) as:

$$amortized\_cost(t) = actual\_cost(t) + \Phi(t+1) - \Phi(t)$$

- Example: assume at time t, there is a request to b, and Opt does not rearrange the list for accessing t.

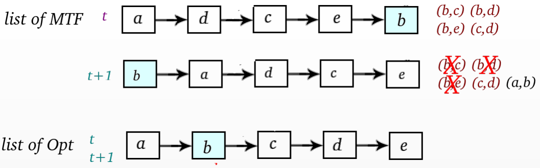


Amortized Cost

- Intuitively, if we have a high potential, we are in bad state.
- Define the **amortized cost** at time t (when answering the tth request) as:

$$amortized\_cost(t) = actual\_cost(t) + \Phi(t+1) - \Phi(t)$$

- Example: assume at time t, there is a request to b, and Opt does not rearrange the list for accessing t.

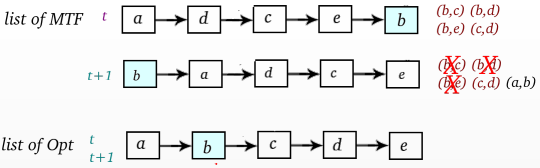


Amortized Cost

- Intuitively, if we have a high potential, we are in bad state.
- Define the **amortized cost** at time t (when answering the tth request) as:

$$amortized\_cost(t) = actual\_cost(t) + \Phi(t+1) - \Phi(t)$$

- Example: assume at time t, there is a request to b, and Opt does not rearrange the list for accessing t.
  - For MTF, we have $actual\_cost(t) = 5$, -$\Phi(t) = 4$ and $\Phi(t+1) = 2$.
  - $amortized\_cost$ is $5 + 2 - 4 = 3$.



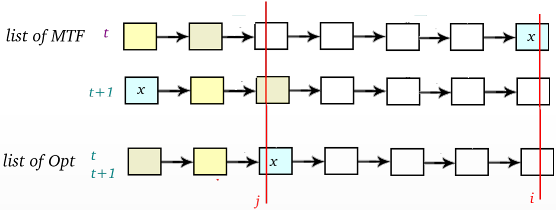
Potential Function Method (cntd.)

Lemma

At any time t, amortized_cost$(t) \leq 2\,\mathrm{Opt}(t)$, i.e., the amortized cost of MTF for the t'th request is at most twice that of Opt.

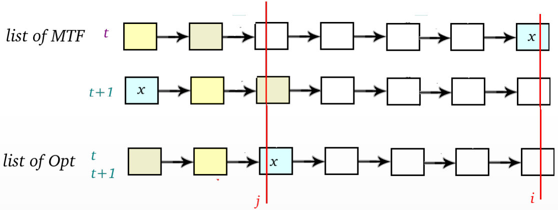
Potential Function Method (cntd.)

- How many inversions are removed by moving x to front?



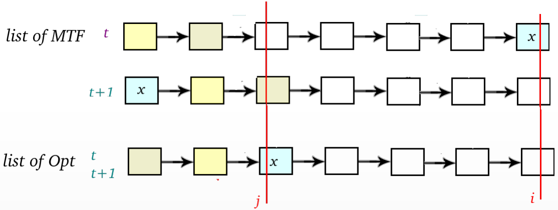
Potential Function Method (cntd.)

- 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.



Potential Function Method (cntd.)

- 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



Potential Function Method (cntd.)

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

Potential Function Method (cntd.)

- When moving x to front:
  - Actual cost is i, at least $i - j$ inversions are removed, at most $j - 1$ inversions are added
- Assume Opt makes k' paid exchanges.
  - Recall that it does no free exchange.
  - The cost of Opt will be $j + k'$.
  - Each paid exchange increases potential by $1 \rightarrow$ potential increases by at most k'.

Potential Function Method (cntd.)

- When moving x to front:
  - Actual cost is i, at least $i - j$ inversions are removed, at most $j - 1$ inversions are added
- Assume Opt makes k' paid exchanges.
  - Recall that it does no free exchange.
  - The cost of Opt will be $j + k'$.
  - Each paid exchange increases potential by 1 $\rightarrow$ potential increases by at most k'.
- $\Phi(t+1) - \Phi(t) = \mathit{added\_inversions} - \mathit{removed\_inversions} \leq (j + k' - 1) - (i - j) = 2j + k' - i - 1.$

Potential Function Method (cntd.)

- When moving x to front:
  - Actual cost is i, at least $i - j$ inversions are removed, at most $j - 1$ inversions are added
- Assume Opt makes k' paid exchanges.
  - Recall that it does no free exchange.
  - The cost of Opt will be $j + k'$.
  - Each paid exchange increases potential by 1 $\rightarrow$ potential increases by at most k'.
- $\Phi(t+1) - \Phi(t) = added\_inversions - removed\_inversions \leq (j + k' - 1) - (i - j) = 2j + k' - i - 1.$
- $amortized\_cost = actual\_cost + \Phi(t+1) - \Phi(t) \leq i + 2j + k' - i - 1 = 2j + k' - 1$

Potential Function Method (cntd.)

- When moving x to front:
  - Actual cost is i, at least $i - j$ inversions are removed, at most $j - 1$ inversions are added
- Assume Opt makes k' paid exchanges.
  - Recall that it does no free exchange.
  - The cost of Opt will be $j + k'$.
  - Each paid exchange increases potential by 1 $\rightarrow$ potential increases by at most k'.
- $\Phi(t+1) - \Phi(t) = \mathit{added\_inversions} - \mathit{removed\_inversions} \leq (j + k' - 1) - (i - j) = 2j + k' - i - 1.$
- $\mathit{amortized\_cost} = \mathit{actual\_cost} + \Phi(t+1) - \Phi(t) \leq i + 2j + k' - i - 1 = 2j + k' - 1$
- Cost of Opt is $j + k'$ and $\mathit{amortized\_cost}$ is less than $2j + k'$.

Lemma

At any time t, $\mathit{amortized\_cost}(t) < 2\,\mathrm{Opt}(t)$.

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** $i - j = 3$. In fact, it turns out to be 5 because all 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.

Competitiveness of MTF

- For the cost of MTF, we have

$$
\begin{aligned}
MTF =& actual\_cost(1) + actual\_cost(2) + \ldots + actual\_cost(n) \\
=& (actual\_cost(1) + \Phi(2) - \Phi(1)) \\
& + (actual\_cost(2) + \Phi(3) - \Phi(2)) \\
& + \ldots \\
& + (actual\_cost(n) + \Phi(n+1) - \Phi(n)) - (\Phi(n+1) - \Phi(1)) \\
& = amortized\_cost(1) + \ldots + amortized\_cost(n) - (\Phi(n+1) - \Phi(1)) \\
& < 2Opt(1) + \ldots + 2\,\mathrm{Opt}(n) - O(k^2) \approx 2Cost\_Opt(n) \\
& \{\text{recall that } \; n \gg k\}
\end{aligned}
$$

- Note that in the second line, we just added and removed values (i.e., we added $\Phi(1) - \Phi(1) + \Phi(2) - \Phi(2) + \ldots + \Phi(n+1) - \Phi(n+1) = 0$).

Competitiveness of MTF

Theorem

Competitive ratio of Mtf *is at most 2*

- No deterministic algorithm can have a competitive ratio better than 2.
  - MTF is an optimal list-update algorithm.
  - Timestamp is another optimal deterministic algorithm.
- There are randomized algorithms that achieve better competitive ratios.

Competitiveness of MTF

Theorem

Competitive ratio of Mtf *is at most 2*

- No deterministic algorithm can have a competitive ratio better than 2.
  - MTF is an optimal list-update algorithm.
  - Timestamp is another optimal deterministic algorithm.
- There are randomized algorithms that achieve better competitive ratios.
- Potential function method is a general framework for analysis of many online algorithms!

Transpose Algorithm

- Transpose: Move the accessed item one unit closer to the front.
- **Question:** What is the competitive ratio of Transpose for a list of length m?

  (a) 1.5 (b) 2 (c) $\Theta(m)$ (d) $\Theta(m^2)$

Transpose Algorithm

- Transpose: Move the accessed item one unit closer to the front.
- **Question:** What is the competitive ratio of Transpose for a list of length m?

(a) 1.5 (b) 2 (c) $\Theta(m)$ (d) $\Theta(m^2)$

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

sequence: a_m

Transpose Algorithm

- Transpose: Move the accessed item one unit closer to the front.
- **Question:** What is the competitive ratio of Transpose for a list of length m?

  (a) 1.5 (b) 2 (c) $\Theta(m)$ (d) $\Theta(m^2)$

$$a_1 \to a_2 \to \ldots \to a_m \to \overset{\downarrow}{a_{m-1}}$$

sequence: $(a_m \; a_{m-1})$

Transpose Algorithm

- Transpose: Move the accessed item one unit closer to the front.
- **Question:** What is the competitive ratio of Transpose for a list of length m?

  (a) 1.5 (b) 2 (c) $\Theta(m)$ (d) $\Theta(m^2)$

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

  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 Opt do?

Transpose Algorithm

- Transpose: Move the accessed item one unit closer to the front.
- **Question:** What is the competitive ratio of Transpose for a list of length m?

  (a) 1.5 (b) 2 (c) $\Theta(m)$ (d) $\Theta(m^2)$

  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.
- **Question:** What is the competitive ratio of Transpose for a list of length m?

  (a) 1.5 (b) 2 (c) $\Theta(m)$ (d) $\Theta(m^2)$

  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 $2m - 3 + n/2 \cdot 1 + n/2 \cdot 2 \approx 1.5n + 2m$.
- The competitive ratio will be at least $\frac{n \cdot m}{1.5n+2m} \in \Theta(m)$.

Other Deterministic Algorithms

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

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?

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?

$$a_1 \to a_2 \to \ldots \to a_m \to a_{m+1} \to a_{m+2} \to \ldots \to a_{2m-1} \to \overset{\downarrow}{\color{red}{a_{2m}}}$$

sequence: a_{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?

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

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?

$$a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_m \rightarrow a_{2m-1} \rightarrow a_{2m} \rightarrow \ldots \rightarrow a_{2m-3} \rightarrow \overset{\downarrow}{a_{2m-2}}$$

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?

$$a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_m \rightarrow a_{m+2} \rightarrow a_{m+3} \rightarrow \ldots \rightarrow a_{2m} \rightarrow \overset{\downarrow}{a_{m+1}}$$

sequence: $a_{2m}\ a_{2m-1}\ a_{2m-2} \ldots a_{m+1}$

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?

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

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

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?

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

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

$$a_1 \to a_2 \to \ldots \to a_m \to a_{m+1} \to a_{m+2} \to \ldots \to a_{2m-1} \to \overset{\downarrow}{\color{red}{a_{2m}}}$$

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

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

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?

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

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 $n \cdot m/2 + O(m^2)$.

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?

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

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

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

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

> **Theorem**
>
> *The competitive ratio is at least* $\frac{n \cdot 2m}{n \cdot m/2 + O(m^2)} \approx 4$

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.

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.

List Update & Compression

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 $\rightarrow$ **Lossless compression**

Basics of Compression

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

Basics of Compression

- 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'.

Basics of Compression

- 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’ :)

Basics of Compression

- 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 Encoding

- 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
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

$$S = \texttt{INEFFICIENCIES}$$

$$C =$$

MTF Encoding

- 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
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

$$S = \texttt{INEFFICIENCIES}$$

$$C = 8$$

MTF Encoding

- 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

$$S = \texttt{INEFFICIENCIES}$$

$$C = \ 8\ 13$$

MTF Encoding

- 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
N	I	A	B	C	D	E	F	G	H	J	K	L	M	O	P	Q	R	S	T	U	V	W	X	Y	Z

$$S = \texttt{INEFFICIENCIES}$$

$$C = 8\ 13\ 6$$

MTF Encoding

- 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
E	N	I	A	B	C	D	F	G	H	J	K	L	M	O	P	Q	R	S	T	U	V	W	X	Y	Z

$$S = \texttt{INEFFICIENCIES}$$

$$C = 8\ 13\ 6\ 7$$

MTF Encoding

- 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
F	E	N	I	A	B	C	D	G	H	J	K	L	M	O	P	Q	R	S	T	U	V	W	X	Y	Z

$$S = \texttt{INEFFICIENCIES}$$

$$C = \;8\;13\;6\;7\;0$$

MTF Encoding

- 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
F	E	N	I	A	B	C	D	G	H	J	K	L	M	O	P	Q	R	S	T	U	V	W	X	Y	Z

$$S = \texttt{INEFFICIENCIES}$$

$$C = 8\ 13\ 6\ 7\ 0\ 3$$

MTF Encoding

- 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	F	E	N	A	B	C	D	G	H	J	K	L	M	O	P	Q	R	S	T	U	V	W	X	Y	Z

$$S = \texttt{INEFFICIENCIES}$$

$$C = 8\ 13\ 6\ 7\ 0\ 3\ 6$$

MTF Encoding

- 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
C	I	F	E	N	A	B	D	G	H	J	K	L	M	O	P	Q	R	S	T	U	V	W	X	Y	Z

$$S = \texttt{INEFFICIENCIES}$$

$$C = 8\ 13\ 6\ 7\ 0\ 3\ 6\ 1$$

MTF Encoding

- 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

$$S = \texttt{INEFFICIENCIES}$$

$$C = \ 8\ 13\ 6\ 7\ 0\ 3\ 6\ 1\ 3\ 4\ 3\ 3\ 3\ 18$$

- What does a run in S encode to in C?
- This results in good compression if we have high **locality** in the input.

Burrows-Wheeler Transform

- 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!

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

banana

banana$		$banana
anana$b		a$banan
nana$ba	sort →	ana$ban
ana$ban		anana$b
na$bana		banana$
a$banan		nana$ba
$banana		na$bana

BWT(banana) = annb$aa

Burrows-Wheeler Transform (cntd.)

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

Burrows-Wheeler Transform (cntd.)

- 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

... the dog ...
...
... *the cat* ...
...
... the duck ...
...
... the man ...

sort $\longrightarrow$

...
he cat ... t
he dog ... t
he duck ... t
he man ... t
...

BWT Decoding

Idea: Given C, We can generate the first column of the array by sorting.
This tells us which character comes after each character in S.

Decoding Algorithm:
View the coded text C as an array of characters.

1. Make array of A of tuples $(C[i], i)$
2. Sort A by the characters, record integers in array N
(Note: $C[N[i]]$ follows $C[i]$ in S, for all $0 \leq i < n$)
3. Set j to index of \$ in C and S to empty string
4. Set $j \leftarrow N[j]$ and append $C[j]$ to S
5. Repeat Step 4 until $C[j] = \$$

BWT Decoding Example

$C =$ `ard$rcaaaabb`
$S =$

$$\begin{array}{c} A \\ \begin{bmatrix} \mathtt{a}, & 0 \\ \mathtt{r}, & 1 \\ \mathtt{d}, & 2 \\ \mathtt{\$}, & 3 \\ \mathtt{r}, & 4 \\ \mathtt{c}, & 5 \\ \mathtt{a}, & 6 \\ \mathtt{a}, & 7 \\ \mathtt{a}, & 8 \\ \mathtt{a}, & 9 \\ \mathtt{b}, & 10 \\ \mathtt{b}, & 11 \end{bmatrix} \end{array}$$

BWT Decoding Example

$C =$ `ard$rcaaaabb`

$S =$

$$A \qquad\qquad\qquad sort(A)$$

$$\begin{bmatrix} \texttt{a}, & 0 \\ \texttt{r}, & 1 \\ \texttt{d}, & 2 \\ \texttt{\$}, & 3 \\ \texttt{r}, & 4 \\ \texttt{c}, & 5 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \end{bmatrix} \longrightarrow \begin{bmatrix} \texttt{\$}, & 3 \\ \texttt{a}, & 0 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \\ \texttt{c}, & 5 \\ \texttt{d}, & 2 \\ \texttt{r}, & 1 \\ \texttt{r}, & 4 \end{bmatrix}$$

BWT Decoding Example

$C =$ `ard$rcaaaabb`
$S =$

$$A = \begin{bmatrix} \texttt{a}, & 0 \\ \texttt{r}, & 1 \\ \texttt{d}, & 2 \\ \texttt{\$}, & 3 \\ \texttt{r}, & 4 \\ \texttt{c}, & 5 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \end{bmatrix} \longrightarrow sort(A) = \begin{bmatrix} \texttt{\$}, & 3 \\ \texttt{a}, & 0 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \\ \texttt{c}, & 5 \\ \texttt{d}, & 2 \\ \texttt{r}, & 1 \\ \texttt{r}, & 4 \end{bmatrix}$$

BWT Decoding Example

$C =$ ard\$rcaaaabb

$S =$

$$A = \begin{bmatrix} \texttt{a}, & 0 \\ \texttt{r}, & 1 \\ \texttt{d}, & 2 \\ \texttt{\$}, & 3 \\ \texttt{r}, & 4 \\ \texttt{c}, & 5 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \end{bmatrix} \longrightarrow sort(A) = \begin{bmatrix} \texttt{\$}, & 3 \\ \texttt{a}, & 0 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \\ \texttt{c}, & 5 \\ \texttt{d}, & 2 \\ \texttt{r}, & 1 \\ \texttt{r}, & 4 \end{bmatrix}$$

$j = 3$

BWT Decoding Example

$C = \texttt{ard\$rcaaaabb}$
$S = \texttt{a}$

$$A = \begin{bmatrix} \texttt{a}, & 0 \\ \texttt{r}, & 1 \\ \texttt{d}, & 2 \\ \texttt{\$}, & 3 \\ \texttt{r}, & 4 \\ \texttt{c}, & 5 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \end{bmatrix} \longrightarrow sort(A) = \begin{bmatrix} \texttt{\$}, & 3 \\ \texttt{a}, & 0 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \\ \texttt{c}, & 5 \\ \texttt{d}, & 2 \\ \texttt{r}, & 1 \\ \texttt{r}, & 4 \end{bmatrix}$$

$j = 7$

BWT Decoding Example

$C = \texttt{ard\$rcaaaabb}$

$S = \texttt{ab}$

$$\begin{array}{c} A \\ \begin{bmatrix} \texttt{a}, & 0 \\ \texttt{r}, & 1 \\ \texttt{d}, & 2 \\ \texttt{\$}, & 3 \\ \texttt{r}, & 4 \\ \texttt{c}, & 5 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \end{bmatrix} \end{array} \longrightarrow \begin{array}{c} sort(A) \\ \begin{bmatrix} \texttt{\$}, & 3 \\ \texttt{a}, & 0 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \\ \texttt{c}, & 5 \\ \texttt{d}, & 2 \\ \texttt{r}, & 1 \\ \texttt{r}, & 4 \end{bmatrix} \end{array}$$

$j = 11$

BWT Decoding Example

$C =$ `ard$rcaaaabb`

$S =$ `abr`

$$A = \begin{bmatrix} \texttt{a}, & 0 \\ \texttt{r}, & 1 \\ \texttt{d}, & 2 \\ \texttt{\$}, & 3 \\ \texttt{r}, & 4 \\ \texttt{c}, & 5 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \end{bmatrix} \longrightarrow sort(A) = \begin{bmatrix} \texttt{\$}, & 3 \\ \texttt{a}, & 0 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \\ \texttt{c}, & 5 \\ \texttt{d}, & 2 \\ \texttt{r}, & 1 \\ \texttt{r}, & 4 \end{bmatrix}$$

$j = 4$

BWT Decoding Example

$C = $ `ard$rcaaaabb`

$S = $ `abracadabra$`

$$\begin{array}{ccc} A & & sort(A) \\ \begin{bmatrix} \texttt{a}, & 0 \\ \texttt{r}, & 1 \\ \texttt{d}, & 2 \\ \texttt{\$}, & 3 \\ \texttt{r}, & 4 \\ \texttt{c}, & 5 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \end{bmatrix} & \longrightarrow & \begin{bmatrix} \texttt{\$}, & 3 \\ \texttt{a}, & 0 \\ \texttt{a}, & 6 \\ \texttt{a}, & 7 \\ \texttt{a}, & 8 \\ \texttt{a}, & 9 \\ \texttt{b}, & 10 \\ \texttt{b}, & 11 \\ \texttt{c}, & 5 \\ \texttt{d}, & 2 \\ \texttt{r}, & 1 \\ \texttt{r}, & 4 \end{bmatrix} \end{array}$$

B-Zip2 compression scheme

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

B-Zip2 compression scheme

- Assume we want to compress a data sequence S.
- Apply BWT on S to increase its locality
  - $baanana\$ \Longrightarrow annb\$aa$
- Apply MTF on BWT output and encode the indices in the list

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

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

- 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
  - $baanana\$ \Longrightarrow annb\$aa$
- Apply MTF on BWT output and encode the indices in the list

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

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

- 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

Decompression

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

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

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

Decompression

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

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

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

- Can we replace MTF by another algorithm?

Decompression

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

$$a \to b \to \ldots \to n \to \ldots \to z \to \$$$

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

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

Decompression

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

$$a \to b \to \ldots \to n \to \ldots \to z \to \$$$

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

- 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?