Map, Jumble, DLLMap, and SkipMap

Victor Milenkovic

Department of Computer Science
University of Miami

CSC220 Programming II – Spring 2018

Map

Map

- A *Map* is what Java calls a PhoneDirectory.

Map

- A *Map* is what Java calls a PhoneDirectory.
- Let's have a look at the Map interface.

Map

- A *Map* is what Java calls a PhoneDirectory.
- Let's have a look at the Map interface.
  - **V put(K key, V value)** is like **addOrChangeEntry**.

Map

- A *Map* is what Java calls a PhoneDirectory.
- Let's have a look at the Map interface.
  - **V put(K key, V value)** is like **addOrChangeEntry**.
  - **V get(K key)** is like **lookupEntry**.

Map

- A *Map* is what Java calls a PhoneDirectory.
- Let's have a look at the Map interface.
  - **V put(K key, V value)** is like **addOrChangeEntry**.
  - **V get(K key)** is like **lookupEntry**.
  - **V remove(Object Key)** is like **removeEntry**.

Map

- A *Map* is what Java calls a PhoneDirectory.
- Let's have a look at the Map interface.
  - **V put(K key, V value)** is like **addOrChangeEntry**.
  - **V get(K key)** is like **lookupEntry**.
  - **V remove(Object Key)** is like **removeEntry**.
- We have already implemented a PhoneDirectory using a doubly linked list.

Map

- A *Map* is what Java calls a PhoneDirectory.
- Let's have a look at the Map interface.
  - **V put(K key, V value)** is like **addOrChangeEntry**.
  - **V get(K key)** is like **lookupEntry**.
  - **V remove(Object Key)** is like **removeEntry**.
- We have already implemented a PhoneDirectory using a doubly linked list.
- We will reimplement it as a DLLMap.

Map

- A *Map* is what Java calls a PhoneDirectory.
- Let's have a look at the Map interface.
  - **V put(K key, V value)** is like **addOrChangeEntry**.
  - **V get(K key)** is like **lookupEntry**.
  - **V remove(Object Key)** is like **removeEntry**.
- We have already implemented a PhoneDirectory using a doubly linked list.
- We will reimplement it as a DLLMap.
  - Internal Node class instead of external DLLEntry class.

Map

- A *Map* is what Java calls a PhoneDirectory.
- Let's have a look at the Map interface.
    - **V put(K key, V value)** is like **addOrChangeEntry**.
    - **V get(K key)** is like **lookupEntry**.
    - **V remove(Object Key)** is like **removeEntry**.
- We have already implemented a PhoneDirectory using a doubly linked list.
- We will reimplement it as a DLLMap.
    - Internal Node class instead of external DLLEntry class.
    - If key isn't there, find will return the *previous* Node instead of the next Node.

Map

- A *Map* is what Java calls a PhoneDirectory.
- Let's have a look at the Map interface.
  - **V put(K key, V value)** is like **addOrChangeEntry**.
  - **V get(K key)** is like **lookupEntry**.
  - **V remove(Object Key)** is like **removeEntry**.
- We have already implemented a PhoneDirectory using a doubly linked list.
- We will reimplement it as a DLLMap.
  - Internal Node class instead of external DLLEntry class.
  - If key isn't there, find will return the *previous* Node instead of the next Node.
  - Or null if there is no previous Node. (Inserting Aaron.)

Map

- A *Map* is what Java calls a PhoneDirectory.
- Let's have a look at the Map interface.
  - **V put(K key, V value)** is like **addOrChangeEntry**.
  - **V get(K key)** is like **lookupEntry**.
  - **V remove(Object Key)** is like **removeEntry**.
- We have already implemented a PhoneDirectory using a doubly linked list.
- We will reimplement it as a DLLMap.
  - Internal Node class instead of external DLLEntry class.
  - If key isn't there, find will return the *previous* Node instead of the next Node.
  - Or null if there is no previous Node. (Inserting Aaron.)
  - You can also tell find where to start looking.

Map

- A *Map* is what Java calls a PhoneDirectory.
- Let's have a look at the Map interface.
  - **V put(K key, V value)** is like **addOrChangeEntry**.
  - **V get(K key)** is like **lookupEntry**.
  - **V remove(Object Key)** is like **removeEntry**.
- We have already implemented a PhoneDirectory using a doubly linked list.
- We will reimplement it as a DLLMap.
  - Internal Node class instead of external DLLEntry class.
  - If key isn't there, find will return the *previous* Node instead of the next Node.
  - Or null if there is no previous Node. (Inserting Aaron.)
  - You can also tell find where to start looking.
  - Partial implementation is here.

This week's application

This week's application

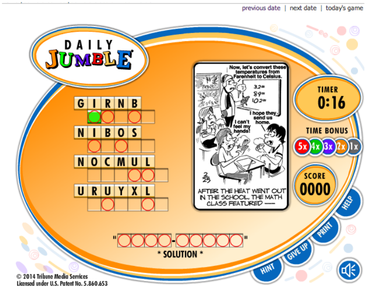
- We need a nice application for our Map.

This week's application

- We need a nice application for our Map.
  - Yet another game!

This week's application

- We need a nice application for our Map.
  - Yet another game!

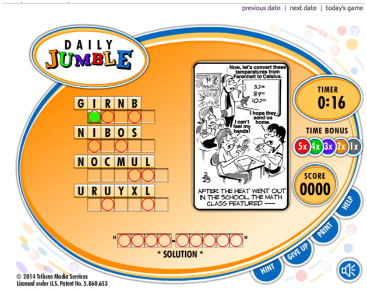


previous date | next date | today's game
DAILY JUMBLE
G I R N B
N I B O S
N O C M U L
U R U Y X L
Now, let's convert these temperatures from Farenheit to Celsius.
32=
89=
102=
I hope they send us home.
I can't feel my hands!
AFTER THE HEAT WENT OUT IN THE SCHOOL, THE MATH CLASS FEATURED ——
* SOLUTION *
TIMER
0:16
TIME BONUS
5x 4x 3x 2x 1x
SCORE
0000
HELP
PRINT
GIVE UP
HINT
© 2014 Tribune Media Services
Licensed under U.S. Patent No. 5,860,653

- Daily Jumble

This week's application

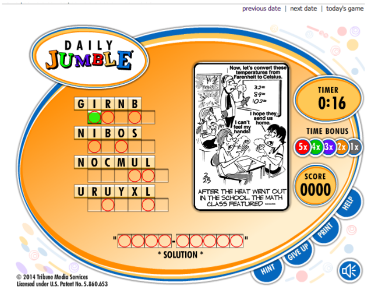
- We need a nice application for our Map.
  - Yet another game!


previous date | next date | today's game
DAILY JUMBLE
GIRNB
NIBOS
NOCMUL
URUYXL
Now, let's convert these temperatures from Farenheit to Celsius.
32= 89= 102=
I hope they send us home.
I can't feel my hands!
AFTER THE HEAT WENT OUT IN THE SCHOOL, THE MATH CLASS FEATURED ——
TIMER
0:16
TIME BONUS
5x 4x 3x 2x 1x
SCORE
0000
HELP
PRINT
GIVE UP
HINT
* SOLUTION *
© 2014 Tribune Media Services
Licensed under U.S. Patent No. 5,860,653

  - Daily Jumble
- Need to unscramble words.

This week's application

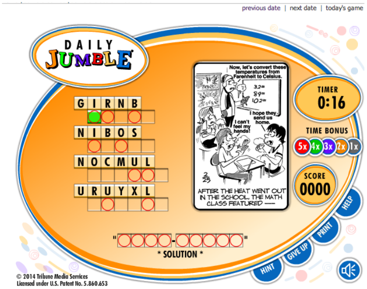
- We need a nice application for our Map.
  - Yet another game!


previous date | next date | today's game
DAILY JUMBLE
GIRNB
NIBOS
NOCMUL
URUYXL
Now, let's convert these temperatures from Farenheit to Celsius.
32=
89=
102=
I hope they send us home.
I can't feel my hands!
AFTER THE HEAT WENT OUT IN THE SCHOOL, THE MATH CLASS FEATURED ——
TIMER
0:16
TIME BONUS
5x 4x 3x 2x 1x
SCORE
0000
* SOLUTION *
HINT
GIVE UP
PRINT
HELP
© 2014 Tribune Media Services
Licensed under U.S. Patent No. 5,860,653

  - Daily Jumble
- Need to unscramble words.
  - Puzzle has "rtpocmue"?

This week’s application

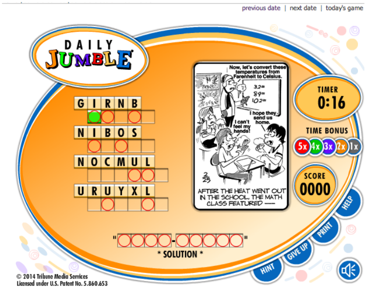
- We need a nice application for our Map.
  - Yet another game!


previous date | next date | today's game
DAILY JUMBLE
G I R N B
N I B O S
N O C M U L
U R U Y X L
Now, let's convert these temperatures from Farenheit to Celsius.
32= 89= 102=
I hope they send us home.
I can't feel my hands!
AFTER THE HEAT WENT OUT IN THE SCHOOL, THE MATH CLASS FEATURED ——
TIMER 0:16
TIME BONUS
5x 4x 3x 2x 1x
SCORE 0000
HELP
PRINT
GIVE UP
HINT
* SOLUTION *
© 2014 Tribune Media Services
Licensed under U.S. Patent No. 5,860,653

  - Daily Jumble
- Need to unscramble words.
  - Puzzle has “rtpocmue”?
  - Unscrambled is “computer”.

This week's application

- We need a nice application for our Map.
  - Yet another game!


previous date | next date | today's game
DAILY JUMBLE
G I R N B
N I B O S
N O C M U L
U R U Y X L
Now, let's convert these temperatures from Farenheit to Celsius.
32=
89=
102=
I hope they send us home.
I can't feel my hands!
AFTER THE HEAT WENT OUT IN THE SCHOOL, THE MATH CLASS FEATURED ——
TIMER
0:16
TIME BONUS
5x 4x 3x 2x 1x
SCORE
0000
HELP
PRINT
GIVE UP
HINT
* SOLUTION *
© 2014 Tribune Media Services
Licensed under U.S. Patent No. 5,860,653

  - Daily Jumble
- Need to unscramble words.
  - Puzzle has “rtpocmue”?
  - Unscrambled is “computer”.
  - How can a Map help us to do that?

Slow Way

Slow Way

- We have a dictionary file.

Slow Way

- We have a dictionary file.
    - Read it in.

Slow Way

- We have a dictionary file.
  - Read it in.
  - Try every possible ordering of “rtpocmue”.

Slow Way

- We have a dictionary file.
  - Read it in.
  - Try every possible ordering of “rtpocmue”.
  - Look up each one in the dictionary.

Slow Way

- We have a dictionary file.
    - Read it in.
    - Try every possible ordering of “rtpocmue”.
    - Look up each one in the dictionary.
- What is the running time?

Slow Way

- We have a dictionary file.
  - Read it in.
  - Try every possible ordering of “rtpocmue”.
  - Look up each one in the dictionary.
- What is the running time?
  - Lookup is O(log n) time, good.

Slow Way

- We have a dictionary file.
  - Read it in.
  - Try every possible ordering of “rtpocmue”.
  - Look up each one in the dictionary.
- What is the running time?
  - Lookup is O(log n) time, good.
  - But the number of orderings is 8! = 40,320, bad!.

Using a Map

Using a Map

- Let’s use a Map.

Using a Map

- Let’s use a Map.
  - The value will be "computer".

Using a Map

- Let’s use a Map.
  - The value will be "computer".
  - What will be the key?

Using a Map

- Let’s use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?

Using a Map

- Let's use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".

Using a Map

- Let's use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:

Using a Map

- Let's use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:
  - Read each word from the dictionary file,

Using a Map

- Let's use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:
  - Read each word from the dictionary file,
  - Put it into the Map.

Using a Map

- Let's use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:
  - Read each word from the dictionary file,
  - Put it into the Map.
  - The key will be the letters of the word in alphabetical order.

Using a Map

- Let’s use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:
  - Read each word from the dictionary file,
  - Put it into the Map.
  - The key will be the letters of the word in alphabetical order.
  - The value will be the word.

Using a Map

- Let's use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:
  - Read each word from the dictionary file,
  - Put it into the Map.
  - The key will be the letters of the word in alphabetical order.
  - The value will be the word.
- To solve a scramble "rtpmceuo":

Using a Map

- Let’s use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:
  - Read each word from the dictionary file,
  - Put it into the Map.
  - The key will be the letters of the word in alphabetical order.
  - The value will be the word.
- To solve a scramble “rtpmceuo”:
  - Alphabetize it to “cemoprtu”.

Using a Map

- Let’s use a Map.
    - The value will be "computer".
    - What will be the key?
    - How about the letters in alphabetical order?
    - That is "cemoprtu".
- To get ready:
    - Read each word from the dictionary file,
    - Put it into the Map.
    - The key will be the letters of the word in alphabetical order.
    - The value will be the word.
- To solve a scramble “rtpmceuo”:
    - Alphabetize it to “cemoprtu”.
    - Look it up in the map: “computer”.

Using a Map

- Let’s use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:
  - Read each word from the dictionary file,
  - Put it into the Map.
  - The key will be the letters of the word in alphabetical order.
  - The value will be the word.
- To solve a scramble “rtpmceuo”:
  - Alphabetize it to “cemoprtu”.
  - Look it up in the map: “computer”.
- Does anyone see a problem?

Using a Map

- Let’s use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:
  - Read each word from the dictionary file,
  - Put it into the Map.
  - The key will be the letters of the word in alphabetical order.
  - The value will be the word.
- To solve a scramble “rtpmceuo”:
  - Alphabetize it to “cemoprtu”.
  - Look it up in the map: “computer”.
- Does anyone see a problem?
  - The words “dare”, “dear”, and “read”

Using a Map

- Let's use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:
  - Read each word from the dictionary file,
  - Put it into the Map.
  - The key will be the letters of the word in alphabetical order.
  - The value will be the word.
- To solve a scramble "rtpmceuo":
  - Alphabetize it to "cemoprtu".
  - Look it up in the map: "computer".
- Does anyone see a problem?
  - The words "dare", "dear", and "read"
  - will all be stored under the key "ader".

Using a Map

- Let’s use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:
  - Read each word from the dictionary file,
  - Put it into the Map.
  - The key will be the letters of the word in alphabetical order.
  - The value will be the word.
- To solve a scramble “rtpmceuo”:
  - Alphabetize it to “cemoprtu”.
  - Look it up in the map: “computer”.
- Does anyone see a problem?
  - The words “dare”, “dear”, and “read”
  - will all be stored under the key “ader”.
  - So the value will be “read” because it is last.

Using a Map

- Let's use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:
  - Read each word from the dictionary file,
  - Put it into the Map.
  - The key will be the letters of the word in alphabetical order.
  - The value will be the word.
- To solve a scramble "rtpmceuo":
  - Alphabetize it to "cemoprtu".
  - Look it up in the map: "computer".
- Does anyone see a problem?
  - The words "dare", "dear", and "read"
  - will all be stored under the key "ader".
  - So the value will be "read" because it is last.
  - Solution is to use **List<String>** as the value type.

Using a Map

- Let’s use a Map.
  - The value will be "computer".
  - What will be the key?
  - How about the letters in alphabetical order?
  - That is "cemoprtu".
- To get ready:
  - Read each word from the dictionary file,
  - Put it into the Map.
  - The key will be the letters of the word in alphabetical order.
  - The value will be the word.
- To solve a scramble “rtpmceuo”:
  - Alphabetize it to “cemoprtu”.
  - Look it up in the map: “computer”.
- Does anyone see a problem?
  - The words “dare”, “dear”, and “read”
  - will all be stored under the key “ader”.
  - So the value will be “read” because it is last.
  - Solution is to use **List<String>** as the value type.
  - But we won’t do that this time.

Implementation using DLLMap

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.
  - The lookup is actually $\mathrm{O}(n)$, as you know, but that's not the problem.

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.
    - The lookup is actually $\mathrm{O}(n)$, as you know, but that's not the problem.
    - The problem is adding all the words in the dictionary.

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.
  - The lookup is actually $O(n)$, as you know, but that's not the problem.
  - The problem is adding all the words in the dictionary.
  - We never got add faster than $O(n)$

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.
  - The lookup is actually $\mathrm{O}(n)$, as you know, but that's not the problem.
  - The problem is adding all the words in the dictionary.
  - We never got add faster than $\mathrm{O}(n)$
  - If we add n words, that's $\mathrm{O}(n^2)$,

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.
  - The lookup is actually $\mathrm{O}(n)$, as you know, but that's not the problem.
  - The problem is adding all the words in the dictionary.
  - We never got add faster than $\mathrm{O}(n)$
  - If we add n words, that's $\mathrm{O}(n^2)$,
  - which is pretty slow.

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.
    - The lookup is actually $O(n)$, as you know, but that's not the problem.
    - The problem is adding all the words in the dictionary.
    - We never got add faster than $O(n)$
    - If we add n words, that's $O(n^2)$,
    - which is pretty slow.
- It's even slower for a bigger dictionary, as I will show you.

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.
  - The lookup is actually $\mathrm{O}(n)$, as you know, but that's not the problem.
  - The problem is adding all the words in the dictionary.
  - We never got add faster than $\mathrm{O}(n)$
  - If we add n words, that's $\mathrm{O}(n^2)$,
  - which is pretty slow.
- It's even slower for a bigger dictionary, as I will show you.
  - To get to entry *i* in the list takes *i* steps.

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.
  - The lookup is actually $\mathrm{O}(n)$, as you know, but that's not the problem.
  - The problem is adding all the words in the dictionary.
  - We never got add faster than $\mathrm{O}(n)$
  - If we add n words, that's $\mathrm{O}(n^2)$,
  - which is pretty slow.
- It's even slower for a bigger dictionary, as I will show you.
  - To get to entry *i* in the list takes *i* steps.
  - An array can do it in one step (**theArray[i]**)

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.
  - The lookup is actually $\mathrm{O}(n)$, as you know, but that's not the problem.
  - The problem is adding all the words in the dictionary.
  - We never got add faster than $\mathrm{O}(n)$
  - If we add n words, that's $\mathrm{O}(n^2)$,
  - which is pretty slow.
- It's even slower for a bigger dictionary, as I will show you.
  - To get to entry *i* in the list takes *i* steps.
  - An array can do it in one step (**theArray[i]**)
  - but then adding at the head will cost O(n)

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.
  - The lookup is actually $O(n)$, as you know, but that's not the problem.
  - The problem is adding all the words in the dictionary.
  - We never got add faster than $O(n)$
  - If we add n words, that's $O(n^2)$,
  - which is pretty slow.
- It's even slower for a bigger dictionary, as I will show you.
  - To get to entry *i* in the list takes *i* steps.
  - An array can do it in one step (**theArray[i]**)
  - but then adding at the head will cost O(n)
  - The linked list lets us add quickly once we get there,

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.
    - The lookup is actually $O(n)$, as you know, but that's not the problem.
    - The problem is adding all the words in the dictionary.
    - We never got add faster than $O(n)$
    - If we add n words, that's $O(n^2)$,
    - which is pretty slow.
- It's even slower for a bigger dictionary, as I will show you.
    - To get to entry *i* in the list takes *i* steps.
    - An array can do it in one step (**theArray[i]**)
    - but then adding at the head will cost O(n)
    - The linked list lets us add quickly once we get there,
    - but it takes a while to get there.

Implementation using DLLMap

- I will run the Jumble solver for you using the DLLMap.
  - The lookup is actually $O(n)$, as you know, but that's not the problem.
  - The problem is adding all the words in the dictionary.
  - We never got add faster than $O(n)$
  - If we add n words, that's $O(n^2)$,
  - which is pretty slow.
- It's even slower for a bigger dictionary, as I will show you.
  - To get to entry *i* in the list takes *i* steps.
  - An array can do it in one step (**theArray[i]**)
  - but then adding at the head will cost O(n)
  - The linked list lets us add quickly once we get there,
  - but it takes a while to get there.
- We need a faster way.

SkipMap Idea

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.
  - $n/4$ steps instead of $n/2$.

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.
  - $n/4$ steps instead of $n/2$.
  - Plus at most one more step in the original list.

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.
  - $n/4$ steps instead of $n/2$.
  - Plus at most one more step in the original list.
  - So $n/4 + 1$ instead of $n/2$.

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.
  - $n/4$ steps instead of $n/2$.
  - Plus at most one more step in the original list.
  - So $n/4 + 1$ instead of $n/2$.
  - So what? It's still $\mathrm{O}(n)$.

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.
  - $n/4$ steps instead of $n/2$.
  - Plus at most one more step in the original list.
  - So $n/4 + 1$ instead of $n/2$.
  - So what? It's still $\mathrm{O}(n)$.
- Suppose we create a linked list which stores the location of every other element of the second list?

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.
  - $n/4$ steps instead of $n/2$.
  - Plus at most one more step in the original list.
  - So $n/4 + 1$ instead of $n/2$.
  - So what? It's still $\mathrm{O}(n)$.
- Suppose we create a linked list which stores the location of every other element of the second list?
  - Get to the middle of the third list in $n/8$.

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.
  - $n/4$ steps instead of $n/2$.
  - Plus at most one more step in the original list.
  - So $n/4 + 1$ instead of $n/2$.
  - So what? It's still $\mathrm{O}(n)$.
- Suppose we create a linked list which stores the location of every other element of the second list?
  - Get to the middle of the third list in $n/8$.
  - To the middle of the second list in $n/8 + 1$.

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.
  - $n/4$ steps instead of $n/2$.
  - Plus at most one more step in the original list.
  - So $n/4 + 1$ instead of $n/2$.
  - So what? It's still $\mathrm{O}(n)$.
- Suppose we create a linked list which stores the location of every other element of the second list?
  - Get to the middle of the third list in $n/8$.
  - To the middle of the second list in $n/8 + 1$.
  - To the middle of the first list in $n/8 + 2$.

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.
  - $n/4$ steps instead of $n/2$.
  - Plus at most one more step in the original list.
  - So $n/4 + 1$ instead of $n/2$.
  - So what? It's still $\mathrm{O}(n)$.
- Suppose we create a linked list which stores the location of every other element of the second list?
  - Get to the middle of the third list in $n/8$.
  - To the middle of the second list in $n/8 + 1$.
  - To the middle of the first list in $n/8 + 2$.
- Keep creating lists!

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.
  - $n/4$ steps instead of $n/2$.
  - Plus at most one more step in the original list.
  - So $n/4 + 1$ instead of $n/2$.
  - So what? It's still $\mathrm{O}(n)$.
- Suppose we create a linked list which stores the location of every other element of the second list?
  - Get to the middle of the third list in $n/8$.
  - To the middle of the second list in $n/8 + 1$.
  - To the middle of the first list in $n/8 + 2$.
- Keep creating lists!
  - $n/16 + 3$, $n/32 + 4$,...,$n/2^k + k - 1$.

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.
  - $n/4$ steps instead of $n/2$.
  - Plus at most one more step in the original list.
  - So $n/4 + 1$ instead of $n/2$.
  - So what? It's still $\mathrm{O}(n)$.
- Suppose we create a linked list which stores the location of every other element of the second list?
  - Get to the middle of the third list in $n/8$.
  - To the middle of the second list in $n/8 + 1$.
  - To the middle of the first list in $n/8 + 2$.
- Keep creating lists!
  - $n/16 + 3$, $n/32 + 4$,...,$n/2^k + k - 1$.
  - Use $k = \log_2 n$ lists.

SkipMap Idea

- Suppose we create a linked list which stores the location of every other element of the first list?
  - We can get to the middle of the second list twice as fast.
  - $n/4$ steps instead of $n/2$.
  - Plus at most one more step in the original list.
  - So $n/4 + 1$ instead of $n/2$.
  - So what? It's still $\mathrm{O}(n)$.
- Suppose we create a linked list which stores the location of every other element of the second list?
  - Get to the middle of the third list in $n/8$.
  - To the middle of the second list in $n/8 + 1$.
  - To the middle of the first list in $n/8 + 2$.
- Keep creating lists!
  - $n/16 + 3$, $n/32 + 4$,...,$n/2^k + k - 1$.
  - Use $k = \log_2 n$ lists.
  - Number of steps is $n/2^{\log_2 n} + \log_2 n - 1 = 1 + \log_2 n + 1 = \log_2 n$

See an example lookup here.

Add and Remove

Add and Remove

- Great idea, but...

Add and Remove

- Great idea, but...
    - How can we add or remove quickly?

Add and Remove

- Great idea, but...
  - How can we add or remove quickly?
  - Changes who is “every other”.

Add and Remove

- Great idea, but...
  - How can we add or remove quickly?
  - Changes who is “every other”.
  - Takes $\mathrm{O}(n)$ time to redo.

Add and Remove

- Great idea, but...
  - How can we add or remove quickly?
  - Changes who is “every other”.
  - Takes $\mathrm{O}(n)$ time to redo.
- One more idea:

Add and Remove

- Great idea, but...
  - How can we add or remove quickly?
  - Changes who is “every other”.
  - Takes $O(n)$ time to redo.
- One more idea:
  - Use a coin flip to determine “every other”.

Add and Remove

- Great idea, but...
  - How can we add or remove quickly?
  - Changes who is “every other”.
  - Takes $O(n)$ time to redo.
- One more idea:
  - Use a coin flip to determine “every other”.
  - If you flip heads in the original list.

Add and Remove

- Great idea, but...
  - How can we add or remove quickly?
  - Changes who is “every other”.
  - Takes $O(n)$ time to redo.
- One more idea:
  - Use a coin flip to determine “every other”.
  - If you flip heads in the original list.
  - You get into the first skip list.

Add and Remove

- Great idea, but...
  - How can we add or remove quickly?
  - Changes who is “every other”.
  - Takes $O(n)$ time to redo.
- One more idea:
  - Use a coin flip to determine “every other”.
  - If you flip heads in the original list.
  - You get into the first skip list.
  - If you flip heads in the first skip list.

Add and Remove

- Great idea, but...
  - How can we add or remove quickly?
  - Changes who is “every other”.
  - Takes $O(n)$ time to redo.
- One more idea:
  - Use a coin flip to determine “every other”.
  - If you flip heads in the original list.
  - You get into the first skip list.
  - If you flip heads in the first skip list.
  - You get into the second skip list.

Add and Remove

- Great idea, but...
  - How can we add or remove quickly?
  - Changes who is “every other”.
  - Takes $O(n)$ time to redo.
- One more idea:
  - Use a coin flip to determine “every other”.
  - If you flip heads in the original list.
  - You get into the first skip list.
  - If you flip heads in the first skip list.
  - You get into the second skip list.
  - And so on.

Add and Remove

- Great idea, but...
  - How can we add or remove quickly?
  - Changes who is “every other”.
  - Takes $O(n)$ time to redo.
- One more idea:
  - Use a coin flip to determine “every other”.
  - If you flip heads in the original list.
  - You get into the first skip list.
  - If you flip heads in the first skip list.
  - You get into the second skip list.
  - And so on.
- Don’t re-flip if someone else is added or removed.

Add and Remove

- Great idea, but...
  - How can we add or remove quickly?
  - Changes who is “every other”.
  - Takes $O(n)$ time to redo.
- One more idea:
  - Use a coin flip to determine “every other”.
  - If you flip heads in the original list.
  - You get into the first skip list.
  - If you flip heads in the first skip list.
  - You get into the second skip list.
  - And so on.
- Don’t re-flip if someone else is added or removed.
  - Current layout only depends on the items that are there.

Add and Remove

- Great idea, but...
  - How can we add or remove quickly?
  - Changes who is “every other”.
  - Takes $O(n)$ time to redo.
- One more idea:
  - Use a coin flip to determine “every other”.
  - If you flip heads in the original list.
  - You get into the first skip list.
  - If you flip heads in the first skip list.
  - You get into the second skip list.
  - And so on.
- Don’t re-flip if someone else is added or removed.
  - Current layout only depends on the items that are there.
  - Examples of add, find, and remove,

Analysis: Space

Analysis: Space

- How many times do you expect to flip heads before you flip tails?

Analysis: Space

- How many times do you expect to flip heads before you flip tails?
  - Let's all *n* of you try it.

Analysis: Space

- How many times do you expect to flip heads before you flip tails?
  - Let's all n of you try it.
  - $n/2$ will flip heads the first time.

Analysis: Space

- How many times do you expect to flip heads before you flip tails?
  - Let's all n of you try it.
  - $n/2$ will flip heads the first time.
  - $n/4$ will flip heads the second time.

Analysis: Space

- How many times do you expect to flip heads before you flip tails?
  - Let's all n of you try it.
  - $n/2$ will flip heads the first time.
  - $n/4$ will flip heads the second time.
  - $n/8$ will flip heads the third time

Analysis: Space

- How many times do you expect to flip heads before you flip tails?
  - Let's all n of you try it.
  - $n/2$ will flip heads the first time.
  - $n/4$ will flip heads the second time.
  - $n/8$ will flip heads the third time
- How many heads in all?

Analysis: Space

- How many times do you expect to flip heads before you flip tails?
  - Let's all n of you try it.
  - $n/2$ will flip heads the first time.
  - $n/4$ will flip heads the second time.
  - $n/8$ will flip heads the third time
- How many heads in all?
  - $n/2 + n/4 + n/8 + ... =$ what?

Analysis: Space

- How many times do you expect to flip heads before you flip tails?
  - Let's all n of you try it.
  - $n/2$ will flip heads the first time.
  - $n/4$ will flip heads the second time.
  - $n/8$ will flip heads the third time
- How many heads in all?
  - $n/2 + n/4 + n/8 + ... =$ what?
  - Answer: n.

Analysis: Space

- How many times do you expect to flip heads before you flip tails?
  - Let's all n of you try it.
  - $n/2$ will flip heads the first time.
  - $n/4$ will flip heads the second time.
  - $n/8$ will flip heads the third time
- How many heads in all?
  - $n/2 + n/4 + n/8 + ... =$ what?
  - Answer: n.
  - So on average you each flip $n/n = 1$ heads.

Analysis: Space

- How many times do you expect to flip heads before you flip tails?
  - Let's all n of you try it.
  - $n/2$ will flip heads the first time.
  - $n/4$ will flip heads the second time.
  - $n/8$ will flip heads the third time
- How many heads in all?
  - $n/2 + n/4 + n/8 + ... =$ what?
  - Answer: n.
  - So on average you each flip $n/n = 1$ heads.
- So the Skip List uses only *twice* as much space as a regular list.

Analysis: Space

- How many times do you expect to flip heads before you flip tails?
  - Let's all n of you try it.
  - $n/2$ will flip heads the first time.
  - $n/4$ will flip heads the second time.
  - $n/8$ will flip heads the third time
- How many heads in all?
  - $n/2 + n/4 + n/8 + ... =$ what?
  - Answer: n.
  - So on average you each flip $n/n = 1$ heads.
- So the Skip List uses only *twice* as much space as a regular list.
  - Each of you is in the original list.

Analysis: Space

- How many times do you expect to flip heads before you flip tails?
  - Let's all n of you try it.
  - $n/2$ will flip heads the first time.
  - $n/4$ will flip heads the second time.
  - $n/8$ will flip heads the third time
- How many heads in all?
  - $n/2 + n/4 + n/8 + ... =$ what?
  - Answer: n.
  - So on average you each flip $n/n = 1$ heads.
- So the Skip List uses only *twice* as much space as a regular list.
  - Each of you is in the original list.
  - On average, each of you appears in one skip list.

Analysis: time

Analysis: time

- How much time do find, add, and remove take?

Analysis: time

- How much time do find, add, and remove take?
    - They all have find the item or where it should go.

Analysis: time

- How much time do find, add, and remove take?
    - They all have find the item or where it should go.
    - The actual add or remove occurs on only two lists on average.

Analysis: time

- How much time do find, add, and remove take?
  - They all have find the item or where it should go.
  - The actual add or remove occurs on only two lists on average.
  - There are $\mathrm{O}(\log n)$ lists.

Analysis: time

- How much time do find, add, and remove take?
  - They all have find the item or where it should go.
  - The actual add or remove occurs on only two lists on average.
  - There are $O(\log n)$ lists.
- So the question is: how many steps forward per list?

Analysis: time

- How much time do find, add, and remove take?
  - They all have find the item or where it should go.
  - The actual add or remove occurs on only two lists on average.
  - There are $\mathrm{O}(\log n)$ lists.
- So the question is: how many steps forward per list?
  - We do not step forward to an item that appeared in a previous list.

Analysis: time

- How much time do find, add, and remove take?
  - They all have find the item or where it should go.
  - The actual add or remove occurs on only two lists on average.
  - There are $O(\log n)$ lists.
- So the question is: how many steps forward per list?
  - We do not step forward to an item that appeared in a previous list.
  - Because we would have stepped forward to it then.

Analysis: time

- How much time do find, add, and remove take?
  - They all have find the item or where it should go.
  - The actual add or remove occurs on only two lists on average.
  - There are $O(\log n)$ lists.
- So the question is: how many steps forward per list?
  - We do not step forward to an item that appeared in a previous list.
  - Because we would have stepped forward to it then.
  - So we only step forward to items that flipped tails in this list.

Analysis: time

- How much time do find, add, and remove take?
  - They all have find the item or where it should go.
  - The actual add or remove occurs on only two lists on average.
  - There are $O(\log n)$ lists.
- So the question is: how many steps forward per list?
  - We do not step forward to an item that appeared in a previous list.
  - Because we would have stepped forward to it then.
  - So we only step forward to items that flipped tails in this list.
  - How many times do we expect to flip tails in a row?

Analysis: time

- How much time do find, add, and remove take?
  - They all have find the item or where it should go.
  - The actual add or remove occurs on only two lists on average.
  - There are $O(\log n)$ lists.
- So the question is: how many steps forward per list?
  - We do not step forward to an item that appeared in a previous list.
  - Because we would have stepped forward to it then.
  - So we only step forward to items that flipped tails in this list.
  - How many times do we expect to flip tails in a row?
  - Answer: 1.

Analysis: time

- How much time do find, add, and remove take?
  - They all have find the item or where it should go.
  - The actual add or remove occurs on only two lists on average.
  - There are $O(\log n)$ lists.
- So the question is: how many steps forward per list?
  - We do not step forward to an item that appeared in a previous list.
  - Because we would have stepped forward to it then.
  - So we only step forward to items that flipped tails in this list.
  - How many times do we expect to flip tails in a row?
  - Answer: 1.
  - So on average, we only step forward once per list.

Analysis: time

- How much time do find, add, and remove take?
  - They all have find the item or where it should go.
  - The actual add or remove occurs on only two lists on average.
  - There are $O(\log n)$ lists.
- So the question is: how many steps forward per list?
  - We do not step forward to an item that appeared in a previous list.
  - Because we would have stepped forward to it then.
  - So we only step forward to items that flipped tails in this list.
  - How many times do we expect to flip tails in a row?
  - Answer: 1.
  - So on average, we only step forward once per list.
- $\log_2 n$ steps on average, so $O(\log n)$.

Summary

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.

Summary

- "Formal Phone Directory" Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list DLLMap.

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list DLLMap.
  - If key isn’t there, find returns previous node (or null). You can tell it where to start looking.

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list DLLMap.
  - If key isn’t there, find returns previous node (or null). You can tell it where to start looking.
- Map helps solve Daily Jumble.

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list DLLMap.
  - If key isn’t there, find returns previous node (or null). You can tell it where to start looking.
- Map helps solve Daily Jumble.
  - Store each word under its sorted key.

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list DLLMap.
  - If key isn’t there, find returns previous node (or null). You can tell it where to start looking.
- Map helps solve Daily Jumble.
  - Store each word under its sorted key.
  - Example: key is “cemoprtu”, value is “computer”.

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list DLLMap.
  - If key isn’t there, find returns previous node (or null). You can tell it where to start looking.
- Map helps solve Daily Jumble.
  - Store each word under its sorted key.
  - Example: key is “cemoprtu”, value is “computer”.
  - Unscramble “rtpmceuo” by sorting to “cemoprtu” and looking up its value.

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list DLLMap.
  - If key isn’t there, find returns previous node (or null). You can tell it where to start looking.
- Map helps solve Daily Jumble.
  - Store each word under its sorted key.
  - Example: key is “cemoprtu”, value is “computer”.
  - Unscramble “rtpmceuo” by sorting to “cemoprtu” and looking up its value.
- DLLMap requires $O(n^2)$ to read in dictionary.

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
    - Can implement as sorted doubly linked list DLLMap.
    - If key isn’t there, find returns previous node (or null). You can tell it where to start looking.
- Map helps solve Daily Jumble.
    - Store each word under its sorted key.
    - Example: key is “cemoprtu”, value is “computer”.
    - Unscramble “rtpmceuo” by sorting to “cemoprtu” and looking up its value.
- DLLMap requires $\mathrm{O}(n^2)$ to read in dictionary.
    - Use “skip lists” which skip “every other” node.

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list DLLMap.
  - If key isn’t there, find returns previous node (or null). You can tell it where to start looking.
- Map helps solve Daily Jumble.
  - Store each word under its sorted key.
  - Example: key is “cemoprtu”, value is “computer”.
  - Unscramble “rtpmceuo” by sorting to “cemoprtu” and looking up its value.
- DLLMap requires $\mathrm{O}(n^2)$ to read in dictionary.
  - Use “skip lists” which skip “every other” node.
  - Use coin flips to define “every other”.

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
    - Can implement as sorted doubly linked list DLLMap.
    - If key isn’t there, find returns previous node (or null). You can tell it where to start looking.
- Map helps solve Daily Jumble.
    - Store each word under its sorted key.
    - Example: key is “cemoprtu”, value is “computer”.
    - Unscramble “rtpmceuo” by sorting to “cemoprtu” and looking up its value.
- DLLMap requires $O(n^2)$ to read in dictionary.
    - Use “skip lists” which skip “every other” node.
    - Use coin flips to define “every other”.
- Analysis of SkipMap

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list DLLMap.
  - If key isn’t there, find returns previous node (or null). You can tell it where to start looking.
- Map helps solve Daily Jumble.
  - Store each word under its sorted key.
  - Example: key is “cemoprtu”, value is “computer”.
  - Unscramble “rtpmceuo” by sorting to “cemoprtu” and looking up its value.
- DLLMap requires $O(n^2)$ to read in dictionary.
  - Use “skip lists” which skip “every other” node.
  - Use coin flips to define “every other”.
- Analysis of SkipMap
  - Only twice a much space as DLLMap.

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list DLLMap.
  - If key isn’t there, find returns previous node (or null). You can tell it where to start looking.
- Map helps solve Daily Jumble.
  - Store each word under its sorted key.
  - Example: key is “cemoprtu”, value is “computer”.
  - Unscramble “rtpmceuo” by sorting to “cemoprtu” and looking up its value.
- DLLMap requires $O(n^2)$ to read in dictionary.
  - Use “skip lists” which skip “every other” node.
  - Use coin flips to define “every other”.
- Analysis of SkipMap
  - Only twice a much space as DLLMap.
  - get, put, and remove are all $O(\log n)$ on average.

Summary

- "Formal Phone Directory" Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list DLLMap.
  - If key isn't there, find returns previous node (or null). You can tell it where to start looking.
- Map helps solve Daily Jumble.
  - Store each word under its sorted key.
  - Example: key is "cemoprtu", value is "computer".
  - Unscramble "rtpmceuo" by sorting to "cemoprtu" and looking up its value.
- DLLMap requires $\mathrm{O}(n^2)$ to read in dictionary.
  - Use "skip lists" which skip "every other" node.
  - Use coin flips to define "every other".
- Analysis of SkipMap
  - Only twice a much space as DLLMap.
  - get, put, and remove are all $\mathrm{O}(\log n)$ on average.
  - So total time to read in the dictionary is

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list DLLMap.
  - If key isn’t there, find returns previous node (or null). You can tell it where to start looking.
- Map helps solve Daily Jumble.
  - Store each word under its sorted key.
  - Example: key is “cemoprtu”, value is “computer”.
  - Unscramble “rtpmceuo” by sorting to “cemoprtu” and looking up its value.
- DLLMap requires $O(n^2)$ to read in dictionary.
  - Use “skip lists” which skip “every other” node.
  - Use coin flips to define “every other”.
- Analysis of SkipMap
  - Only twice a much space as DLLMap.
  - get, put, and remove are all $O(\log n)$ on average.
  - So total time to read in the dictionary is
  - $O(n \log n)$.