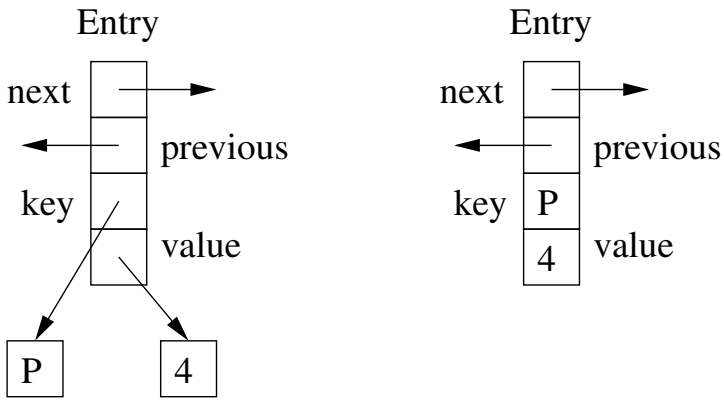
Skip Lists

Victor Milenkovic

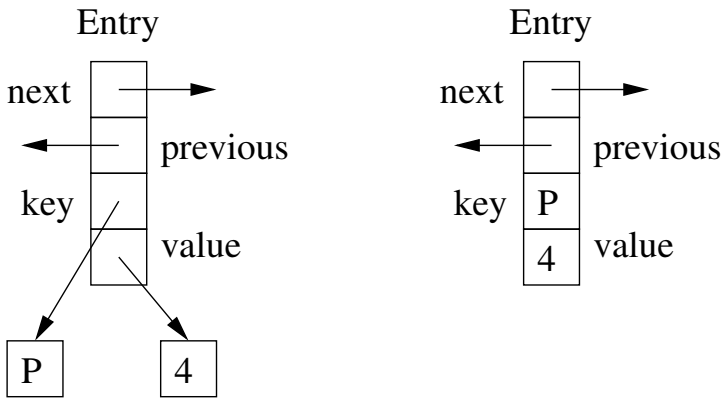
Department of Computer Science
University of Miami

CSC220 Programming II – Spring 2022

Linked Lists

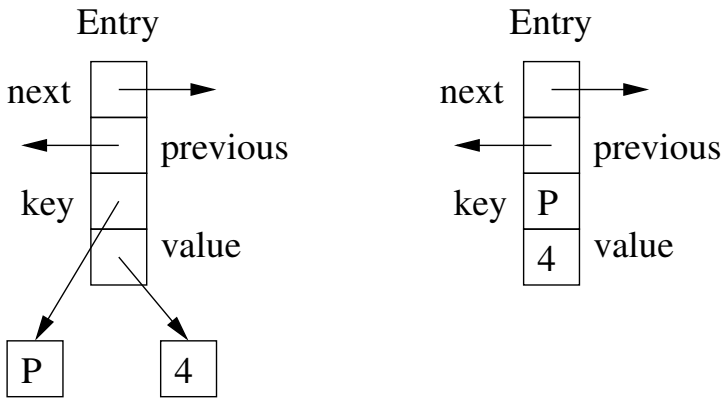


Linked Lists



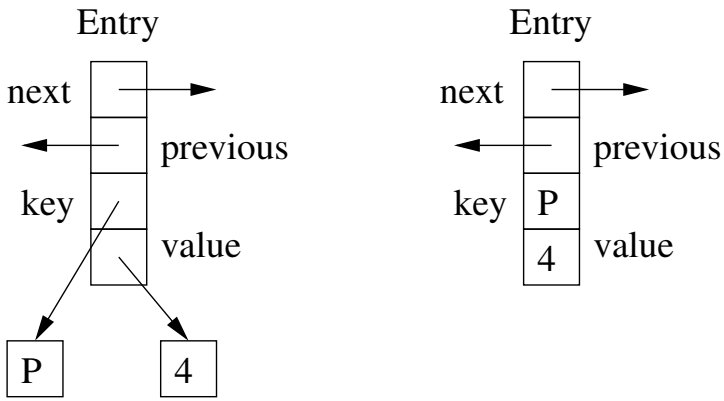
- A linked list uses an Entry class

Linked Lists



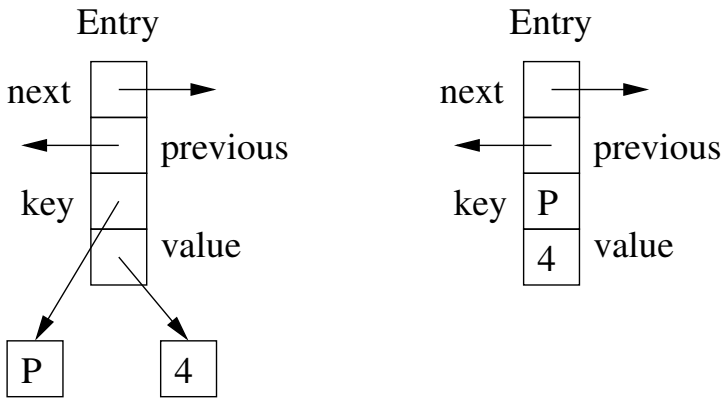
- A linked list uses an Entry class
- with next, previous, key, and value.

Linked Lists



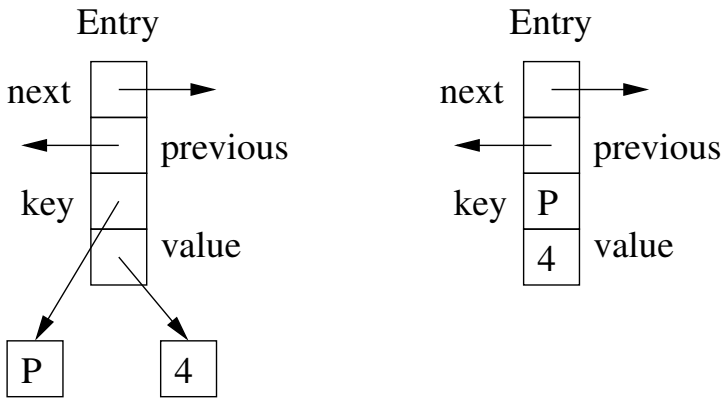
- A linked list uses an Entry class
- with next, previous, key, and value.
- They are all pointers (arrows) to other objects,

Linked Lists



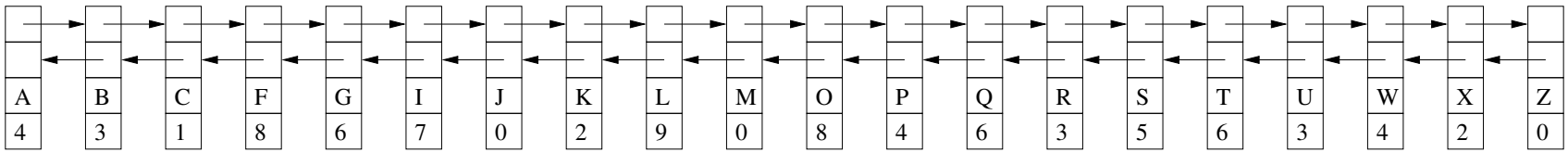
- A linked list uses an Entry class
- with next, previous, key, and value.
- They are all pointers (arrows) to other objects,
- but we usually just write the key and value in the boxes.

Linked Lists

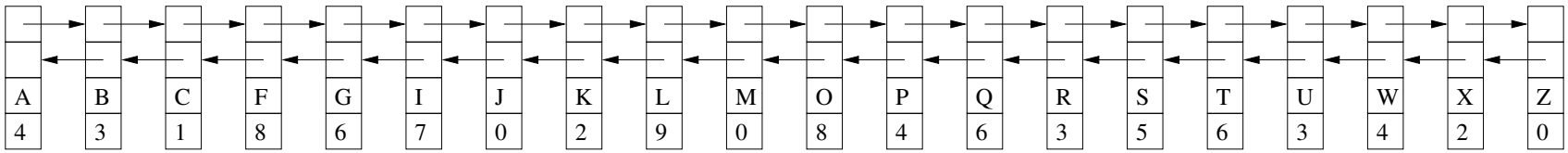


- A linked list uses an Entry class
- with next, previous, key, and value.
- They are all pointers (arrows) to other objects,
- but we usually just write the key and value in the boxes.
- For convenience of drawing the diagrams, I have put next and previous before key and value.

Map using Sorted Linked List

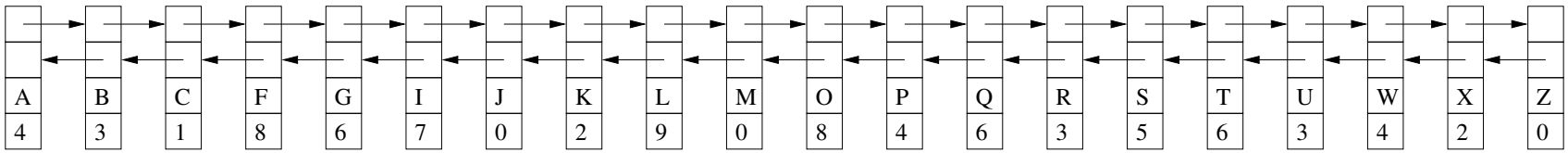


Map using Sorted Linked List



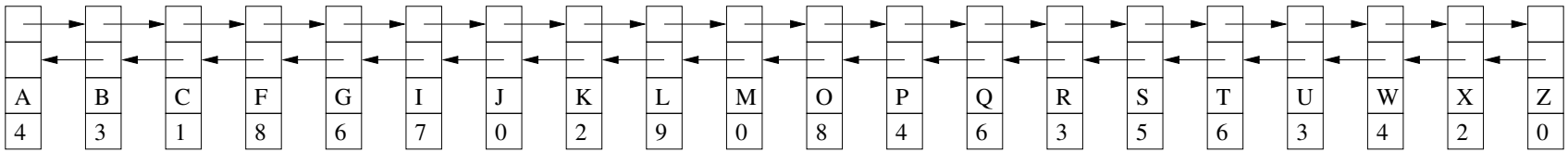
- Linked lists allow us to add or remove an Entry in $O(1)$ time,

Map using Sorted Linked List



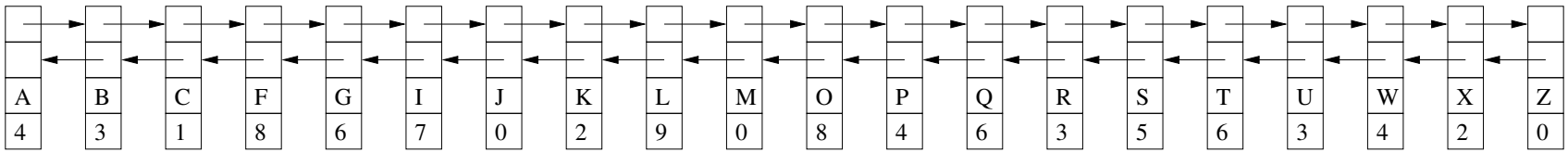
- Linked lists allow us to add or remove an Entry in $O(1)$ time,
  - unlike an array which requires $O(n)$ time

Map using Sorted Linked List



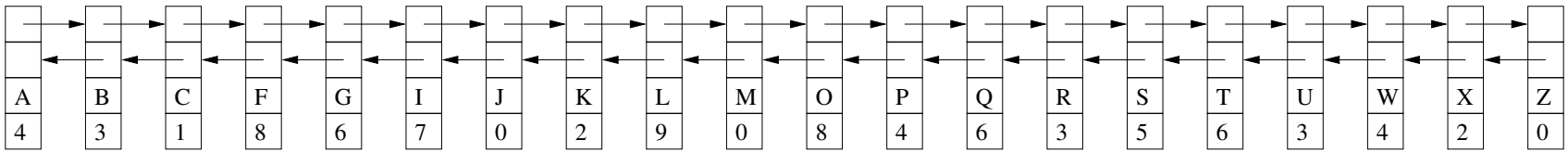
- Linked lists allow us to add or remove an Entry in $O(1)$ time,
  - unlike an array which requires $O(n)$ time
  - because it may have to move many of the elements forward or back.

Map using Sorted Linked List



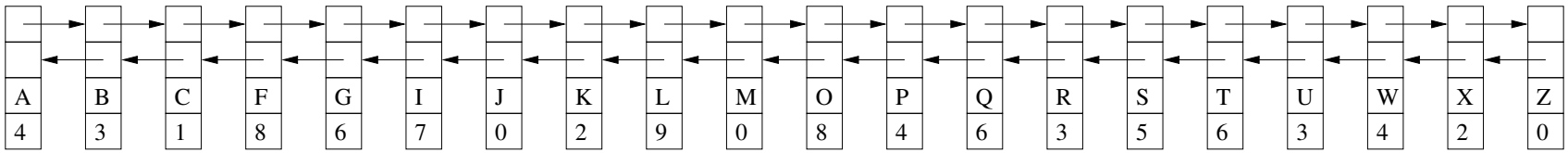
- Linked lists allow us to add or remove an Entry in $O(1)$ time,
  - unlike an array which requires $O(n)$ time
  - because it may have to move many of the elements forward or back.
- Unfortunately, it takes $O(n)$ time to get to an Entry in a linked list.

Map using Sorted Linked List



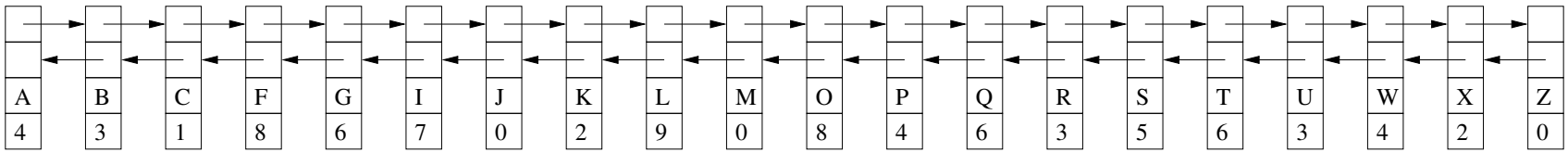
- Linked lists allow us to add or remove an Entry in $O(1)$ time,
  - unlike an array which requires $O(n)$ time
  - because it may have to move many of the elements forward or back.
- Unfortunately, it takes $O(n)$ time to get to an Entry in a linked list.
  - Binary search would actually be $O(n \log n)$ time on a linked list.

Map using Sorted Linked List



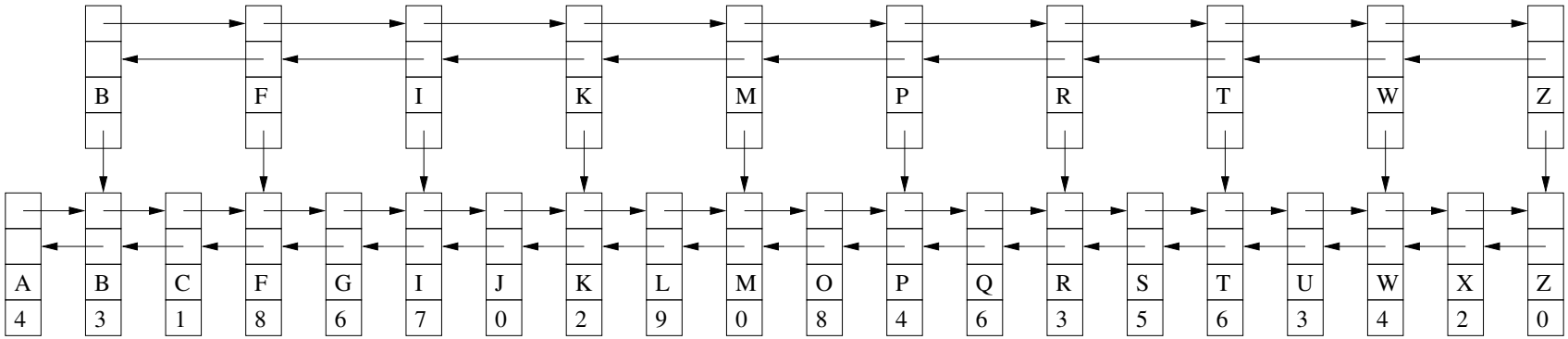
- Linked lists allow us to add or remove an Entry in $O(1)$ time,
  - unlike an array which requires $O(n)$ time
  - because it may have to move many of the elements forward or back.
- Unfortunately, it takes $O(n)$ time to get to an Entry in a linked list.
  - Binary search would actually be $O(n \log n)$ time on a linked list.
  - It would do $O(\log n)$ gets, each in $O(n)$ time.

Map using Sorted Linked List

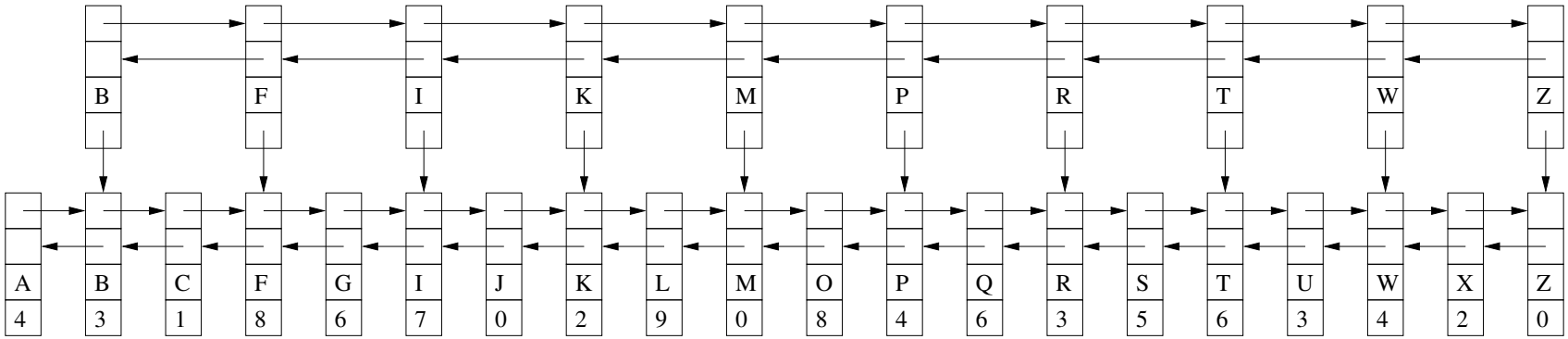


- Linked lists allow us to add or remove an Entry in $O(1)$ time,
  - unlike an array which requires $O(n)$ time
  - because it may have to move many of the elements forward or back.
- Unfortunately, it takes $O(n)$ time to get to an Entry in a linked list.
  - Binary search would actually be $O(n \log n)$ time on a linked list.
  - It would do $O(\log n)$ gets, each in $O(n)$ time.
- So how can we apply the “gold coin” idea?

Skipping Keys

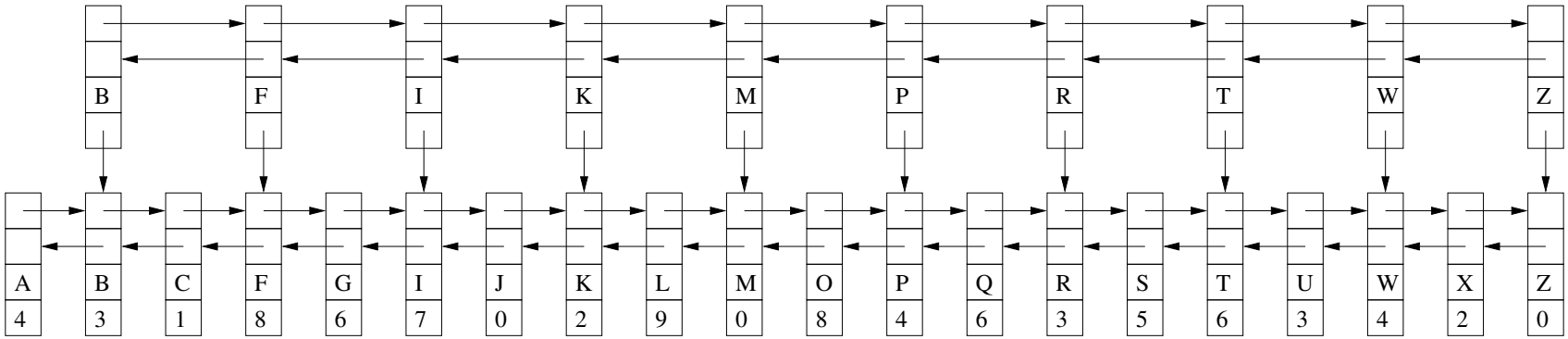


Skipping Keys



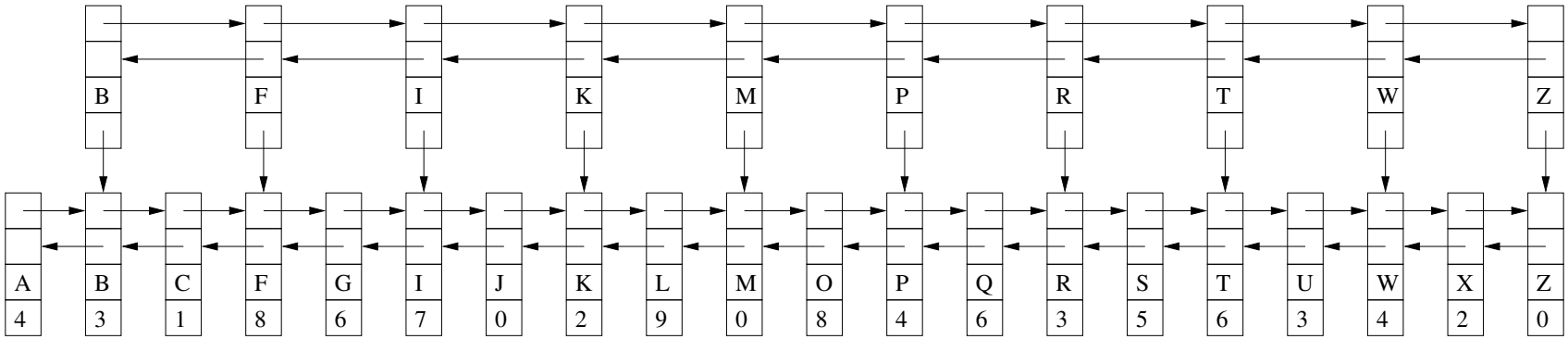
- What if we create a *second* linked list that skips every other key?

Skipping Keys



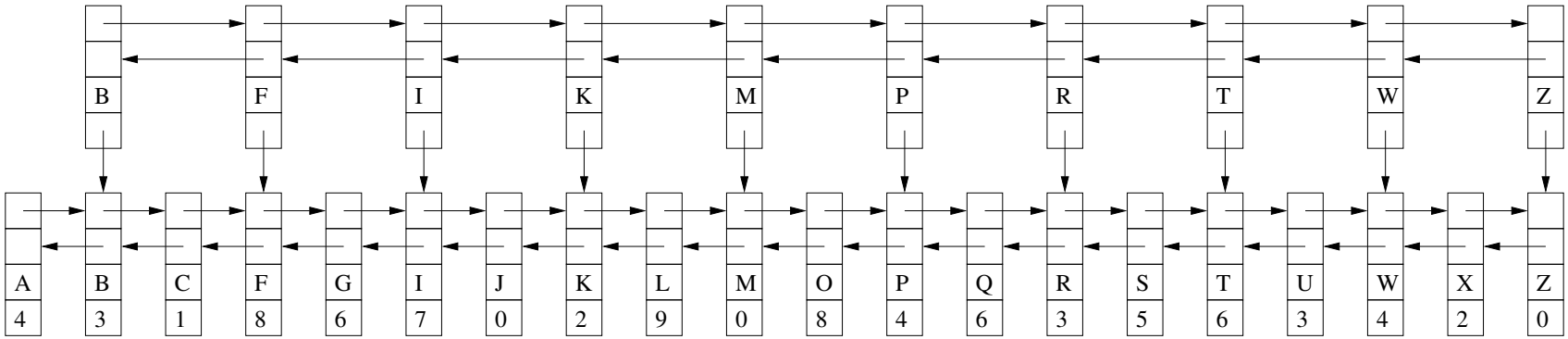
- What if we create a *second* linked list that skips every other key?
- The value would be a pointer to the Entry with that key in the first list.

Skipping Keys



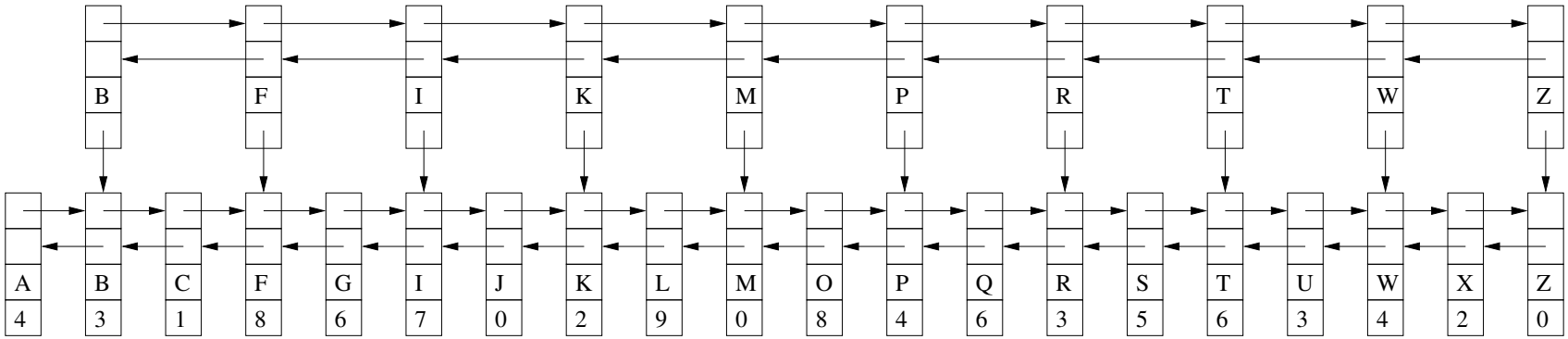
- What if we create a *second* linked list that skips every other key?
- The value would be a pointer to the Entry with that key in the first list.
- We could get as close as possible to a key in the second list,

Skipping Keys



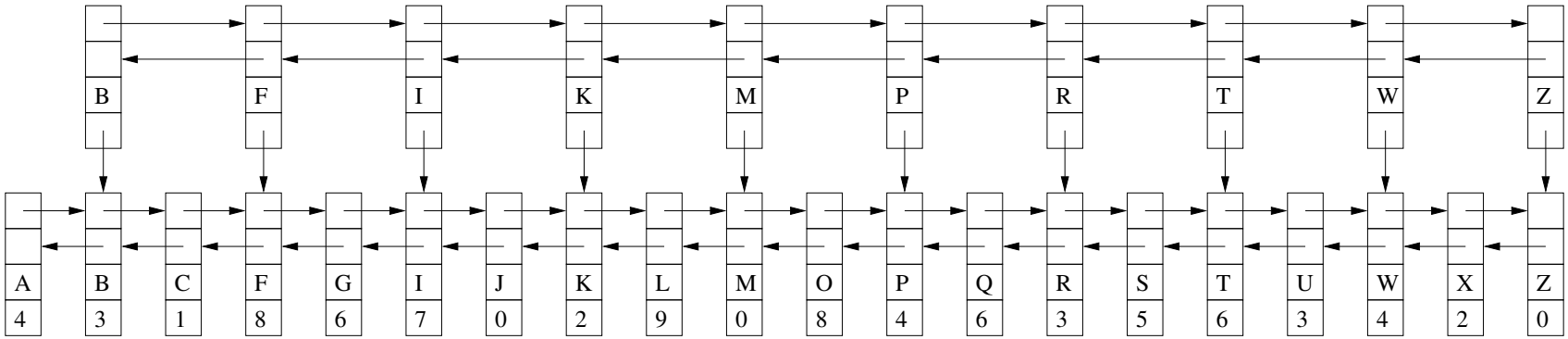
- What if we create a *second* linked list that skips every other key?
- The value would be a pointer to the Entry with that key in the first list.
- We could get as close as possible to a key in the second list,
- and then go at most *one* step in the first list to get to the key.

Skipping Keys



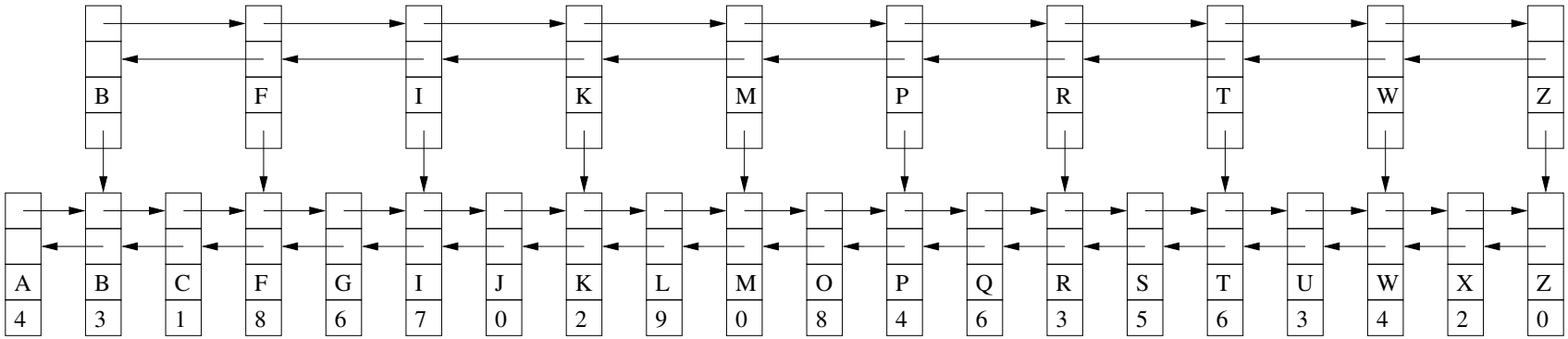
- What if we create a *second* linked list that skips every other key?
- The value would be a pointer to the Entry with that key in the first list.
- We could get as close as possible to a key in the second list,
- and then go at most *one* step in the first list to get to the key.
- That reduces the number of steps from n to $n/2 + 1$.

Skipping Keys



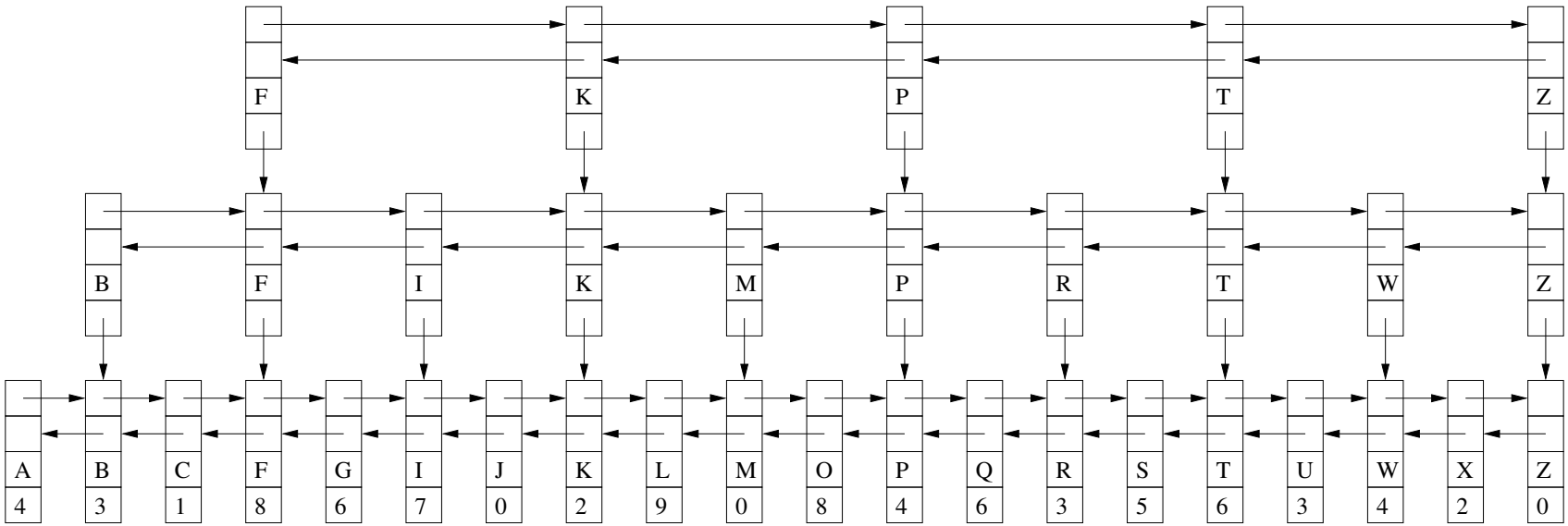
- What if we create a *second* linked list that skips every other key?
- The value would be a pointer to the Entry with that key in the first list.
- We could get as close as possible to a key in the second list,
- and then go at most *one* step in the first list to get to the key.
- That reduces the number of steps from n to $n/2 + 1$.
- But that is still $\mathrm{O}(n)$.

Skipping Keys

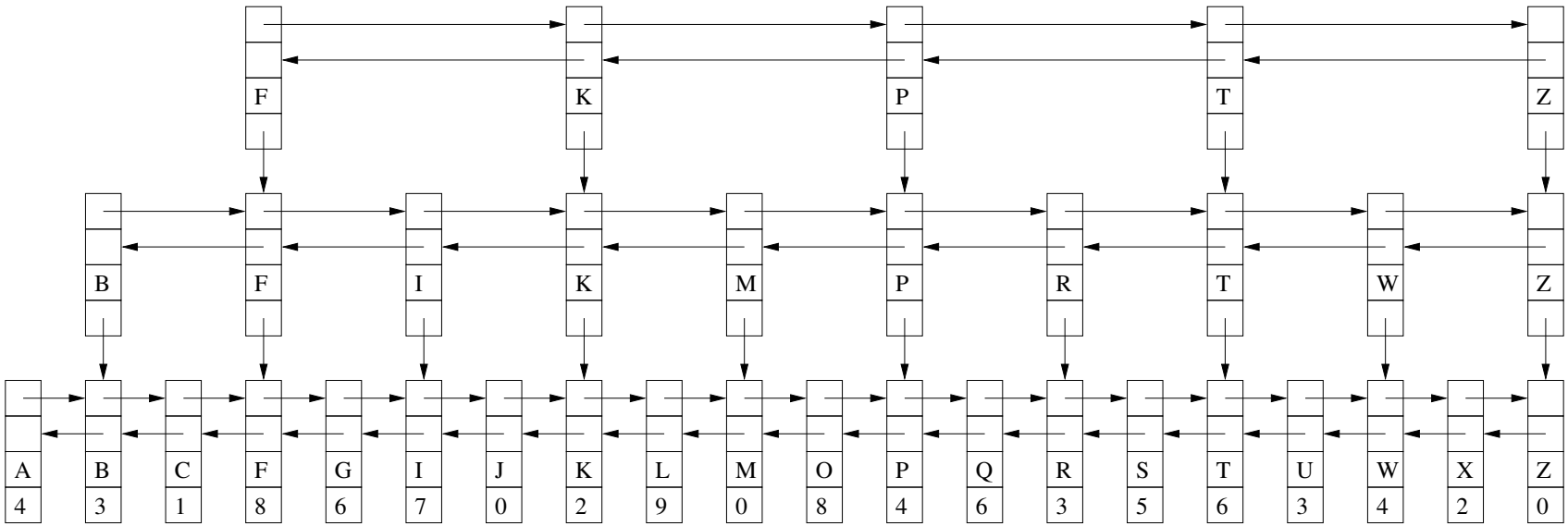


- What if we create a *second* linked list that skips every other key?
- The value would be a pointer to the Entry with that key in the first list.
- We could get as close as possible to a key in the second list,
- and then go at most *one* step in the first list to get to the key.
- That reduces the number of steps from n to $n/2 + 1$.
- But that is still $\mathrm{O}(n)$.
- Too bad. What should we do?

Do it again!

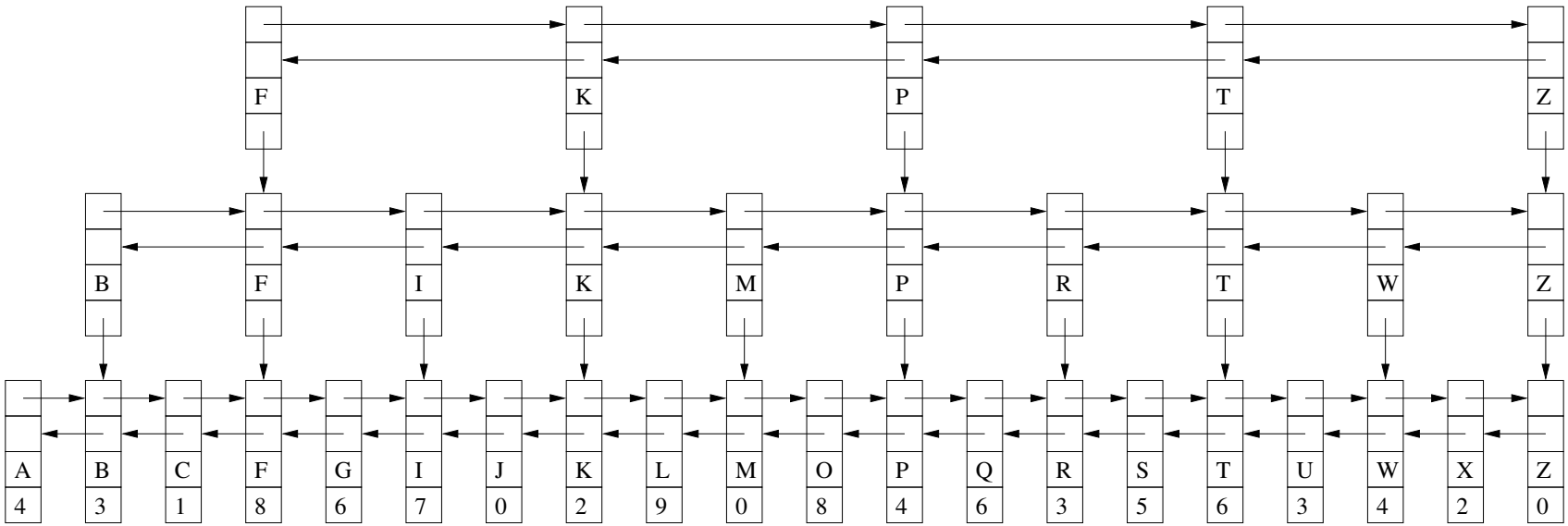

F K P T Z
B F I K M P R T W Z
A B C F G I J K L M O P Q R S T U W X Z
4 3 1 8 6 7 0 2 9 0 8 4 6 3 5 6 3 4 2 0

Do it again!



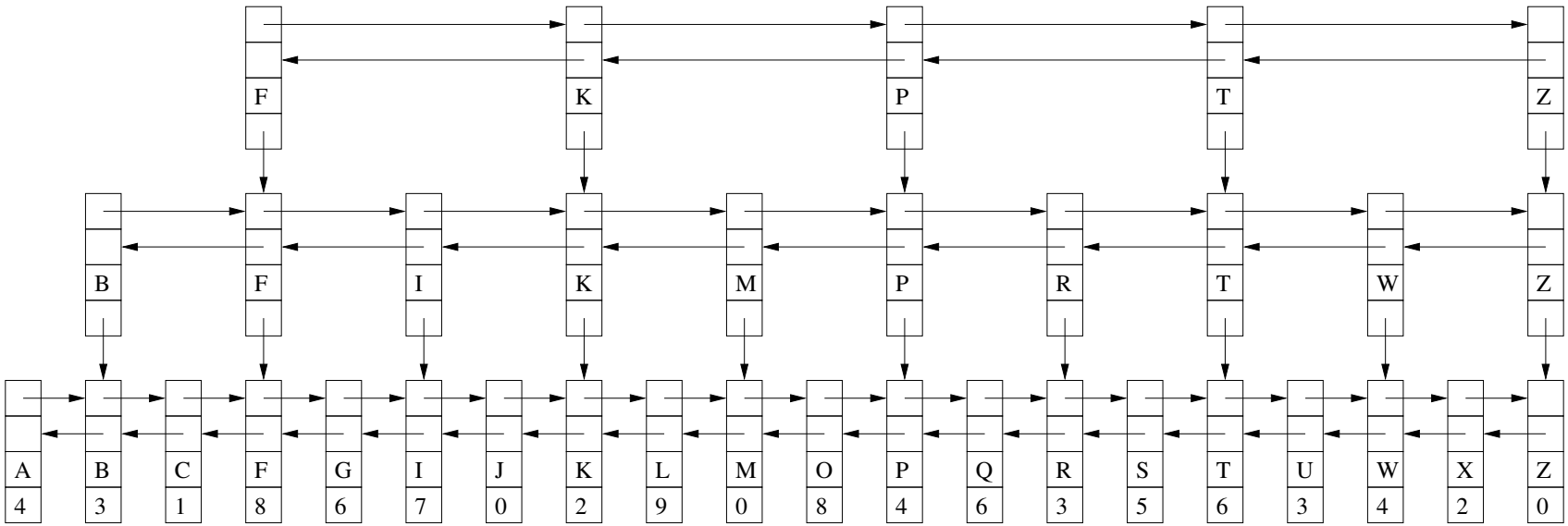
- OK, add a *third* list that skips every other key in the second list.

Do it again!



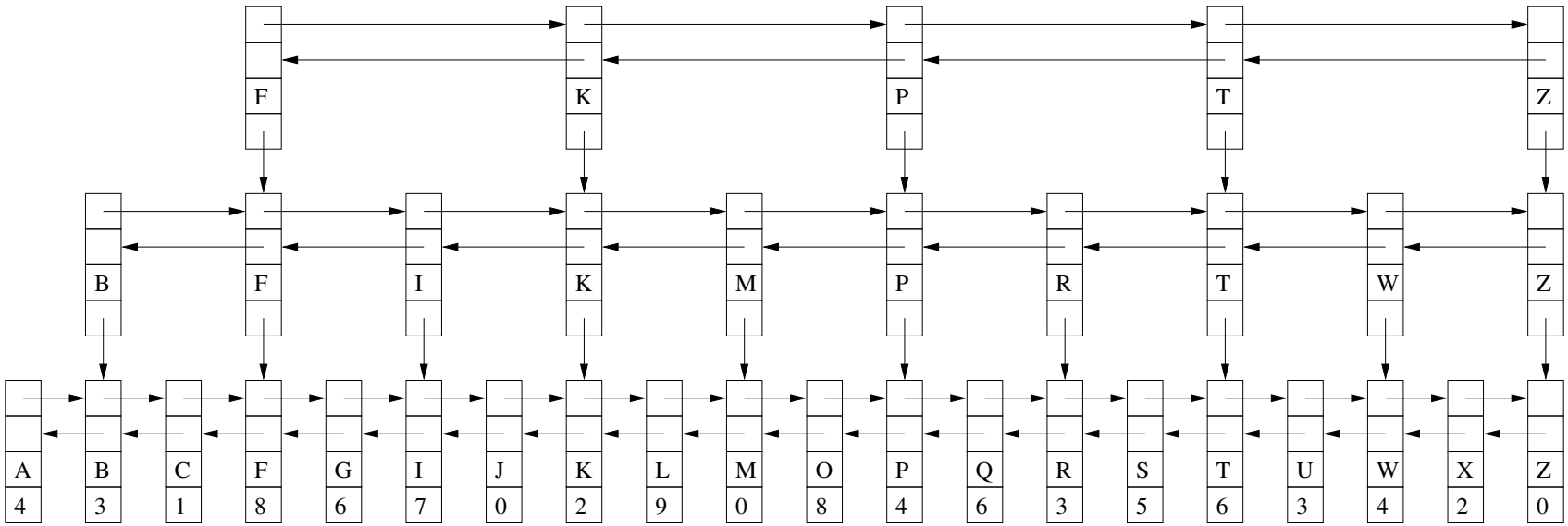
- OK, add a *third* list that skips every other key in the second list.
- So we will only have to go at most *one* step in the second list.

Do it again!



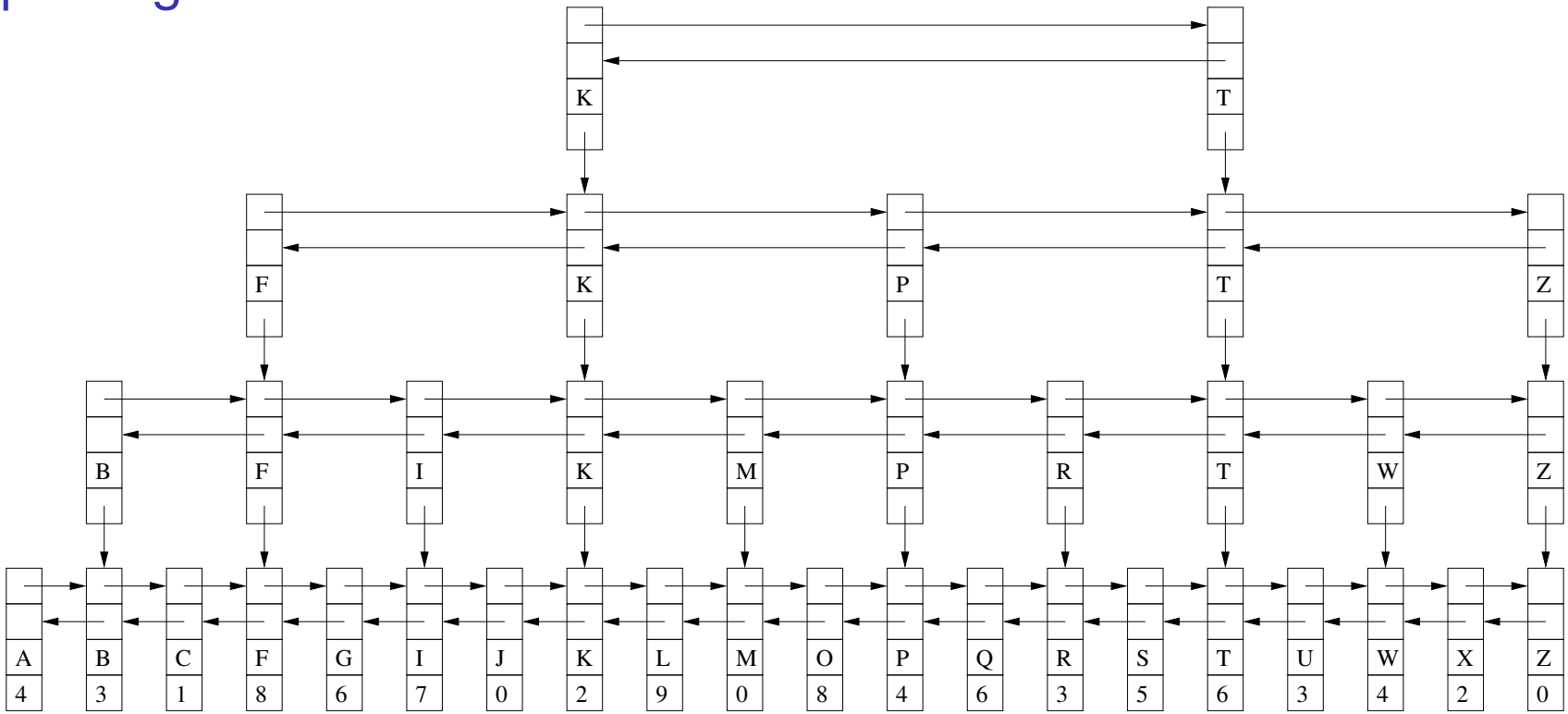
- OK, add a *third* list that skips every other key in the second list.
- So we will only have to go at most *one* step in the second list.
- And as before at most one step in the first list.

Do it again!

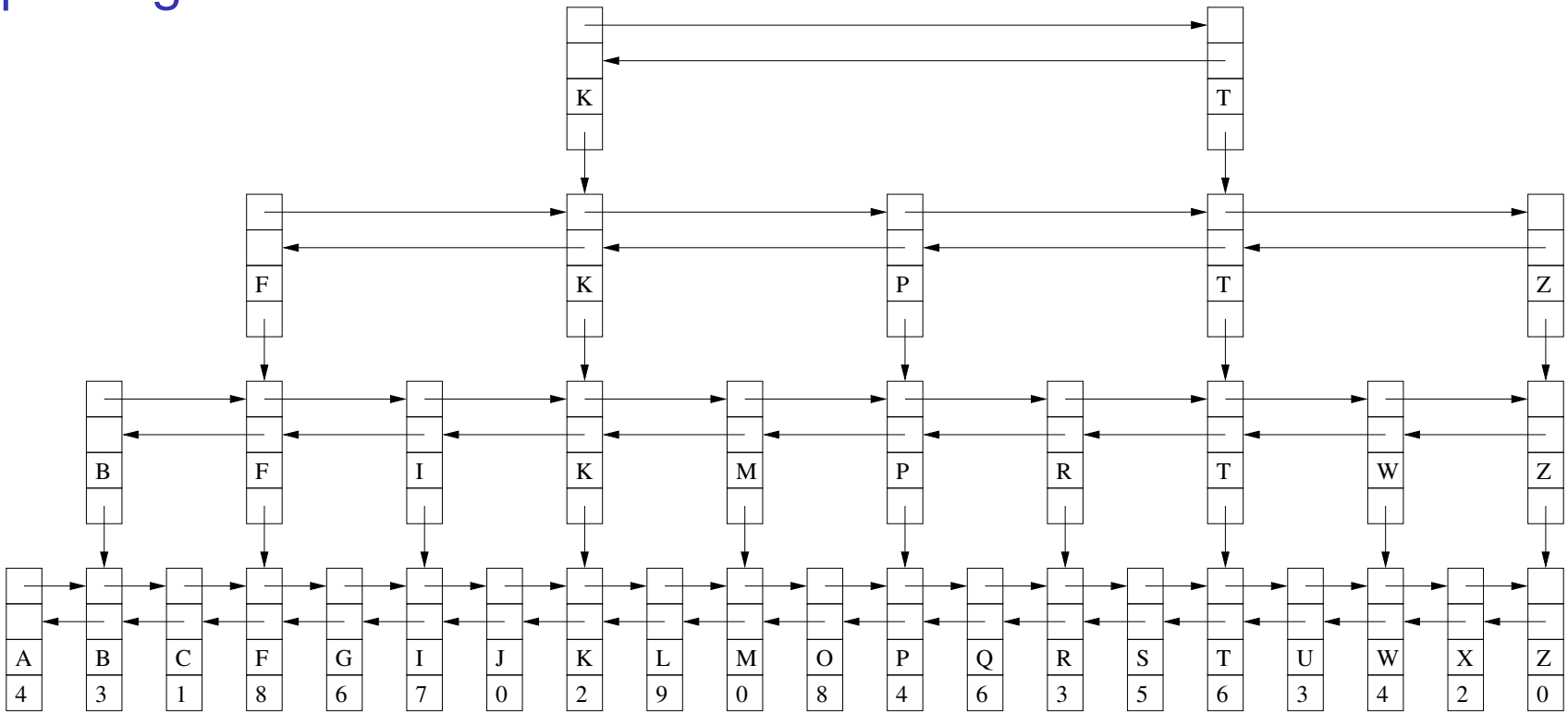


- OK, add a *third* list that skips every other key in the second list.
- So we will only have to go at most *one* step in the second list.
- And as before at most one step in the first list.
- $n/4 + 2$ is still $\mathrm{O}(n)$.

Keep doing it until...

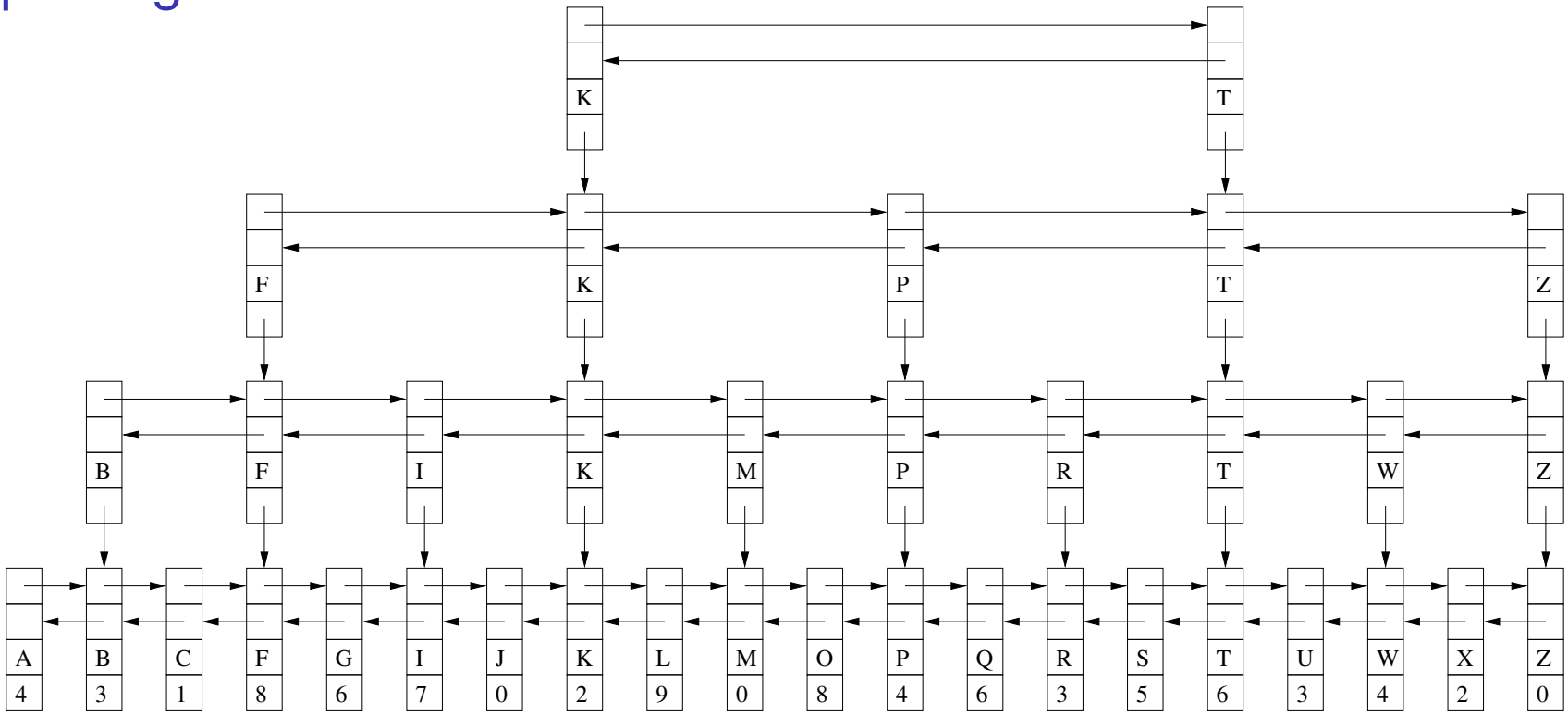


Keep doing it until...



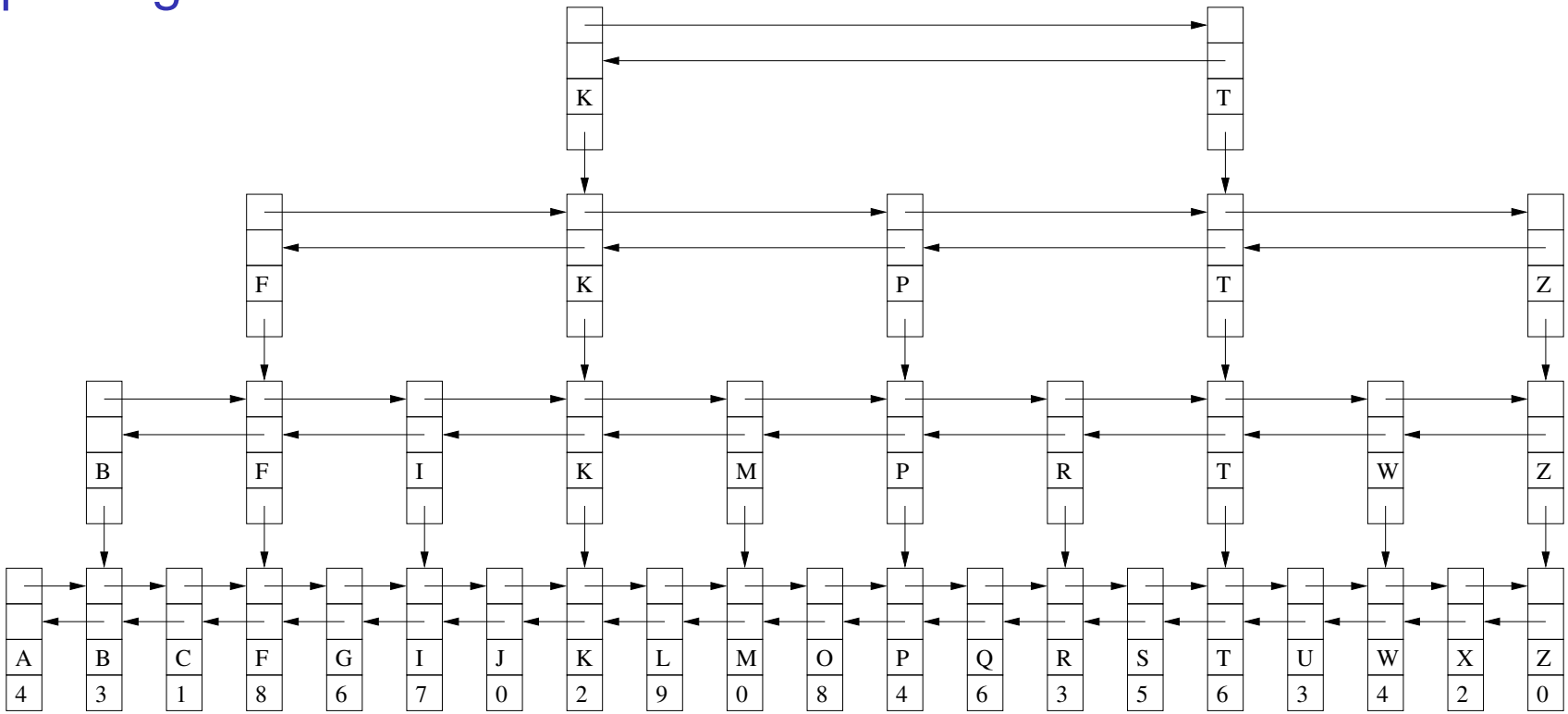
- But what if we keep doing this until there is only a *constant* number in the topmost list?

Keep doing it until...



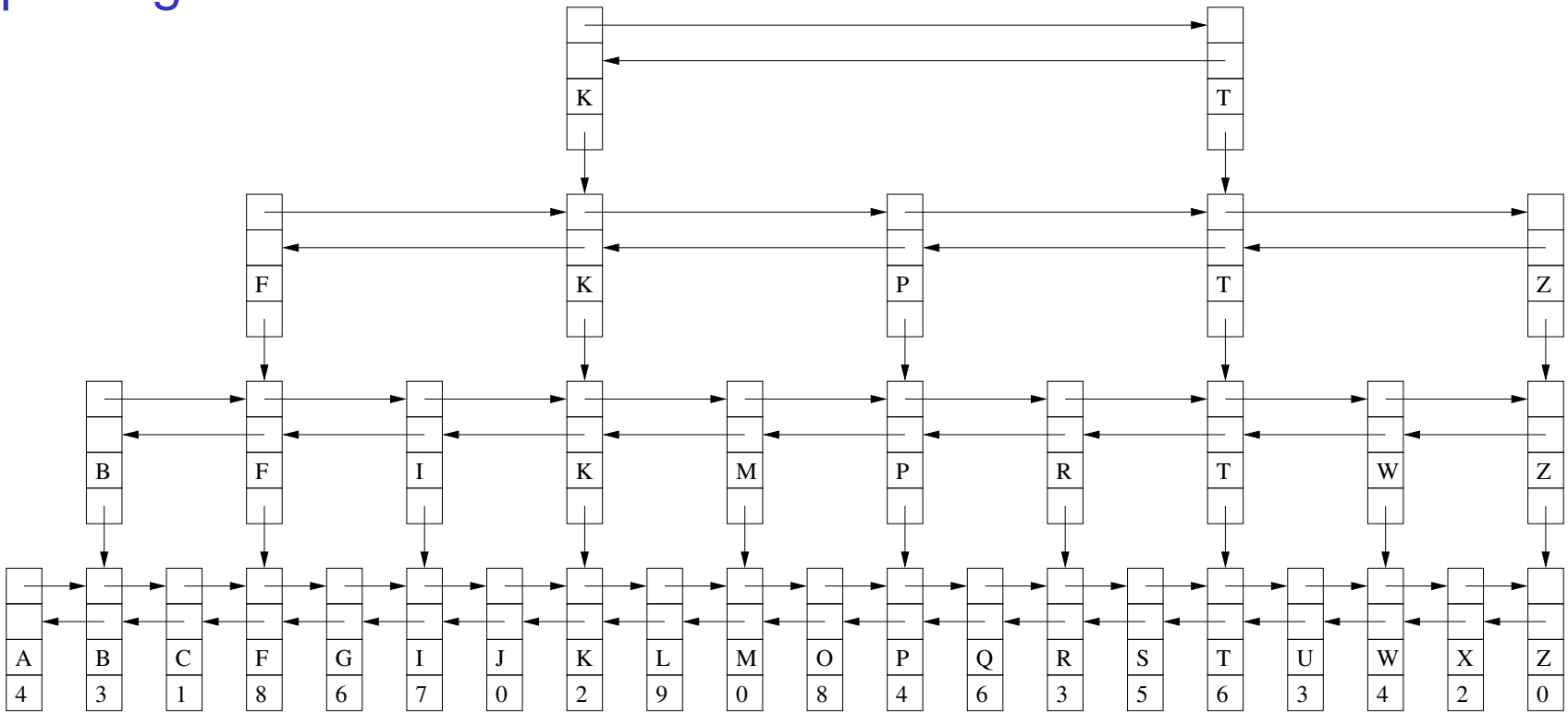
- But what if we keep doing this until there is only a *constant* number in the topmost list?
- So we will go at most one step in *every* list.

Keep doing it until...



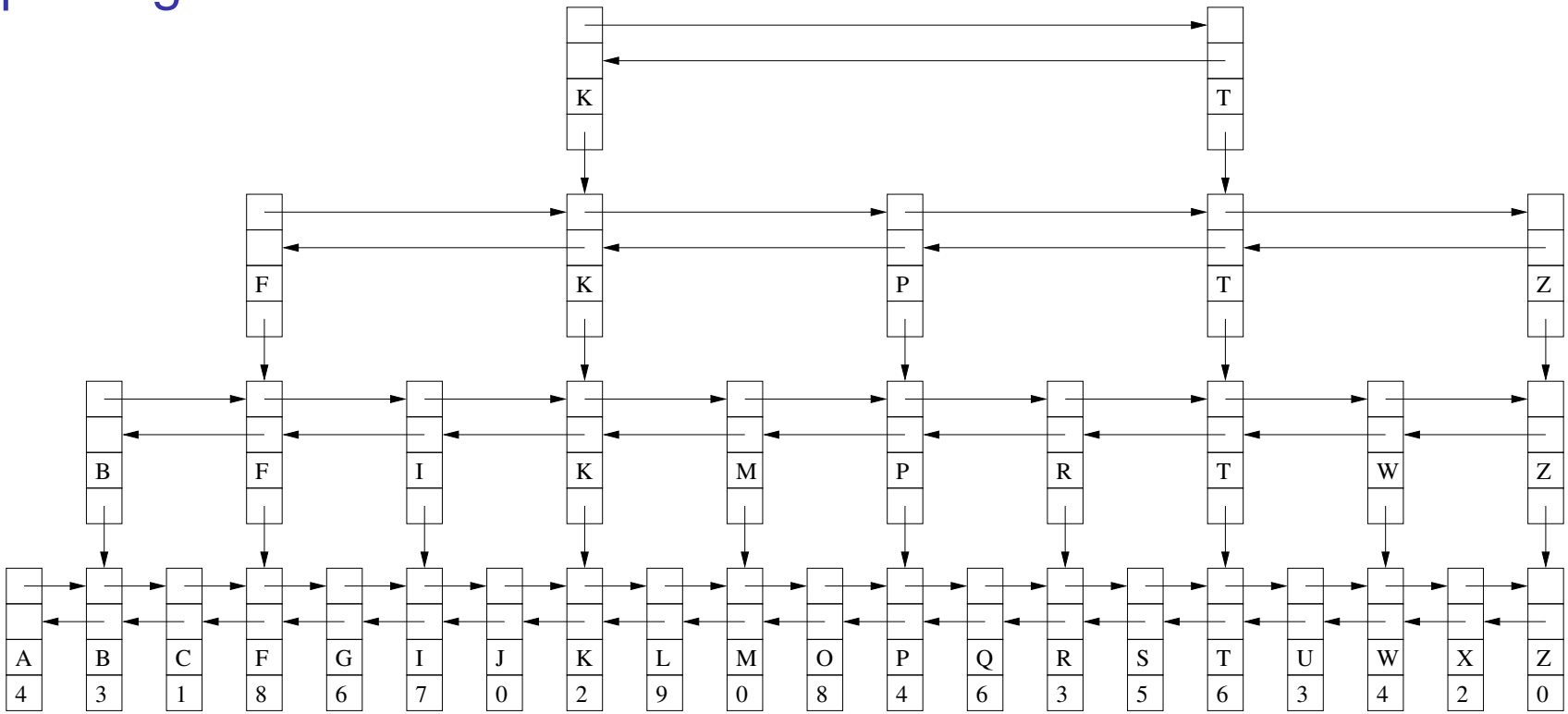
- But what if we keep doing this until there is only a *constant* number in the topmost list?
- So we will go at most one step in *every* list.
- How many lists will there be?

Keep doing it until...



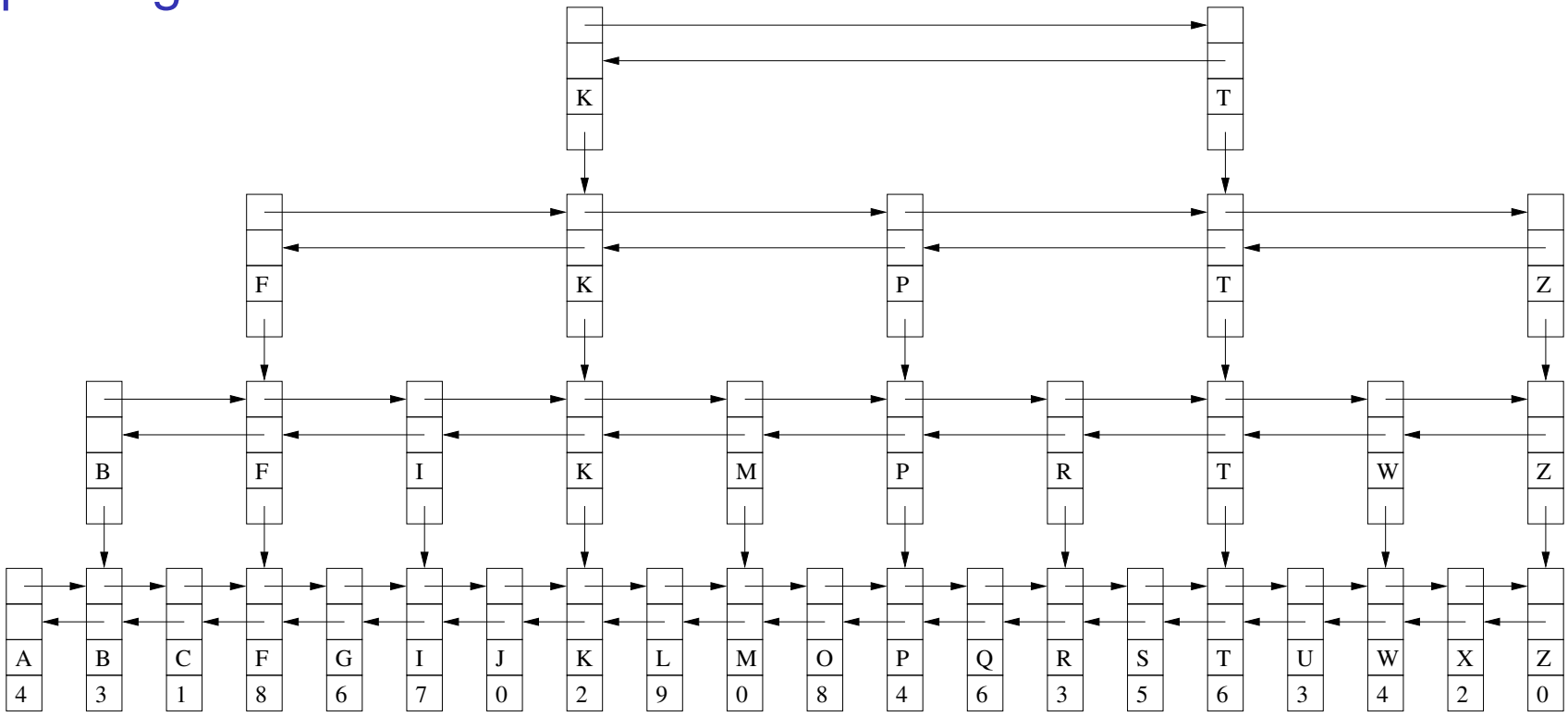
- But what if we keep doing this until there is only a *constant* number in the topmost list?
- So we will go at most one step in *every* list.
- How many lists will there be?
- Hint: each list has half as many elements as the one below it.

Keep doing it until...



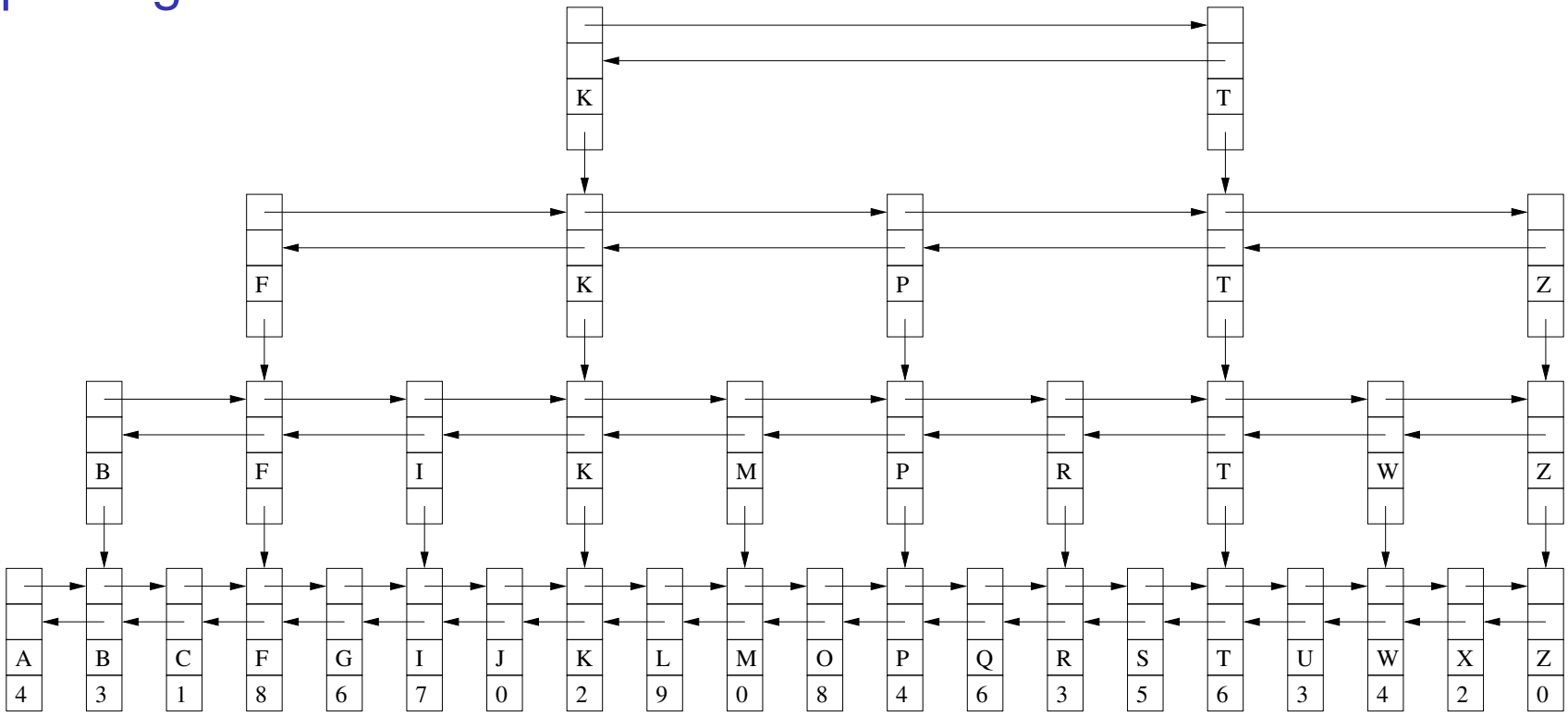
- But what if we keep doing this until there is only a *constant* number in the topmost list?
- So we will go at most one step in *every* list.
- How many lists will there be?
- Hint: each list has half as many elements as the one below it.
- I hope you all *immediately* thought $\log_2 n$.

Keep doing it until...



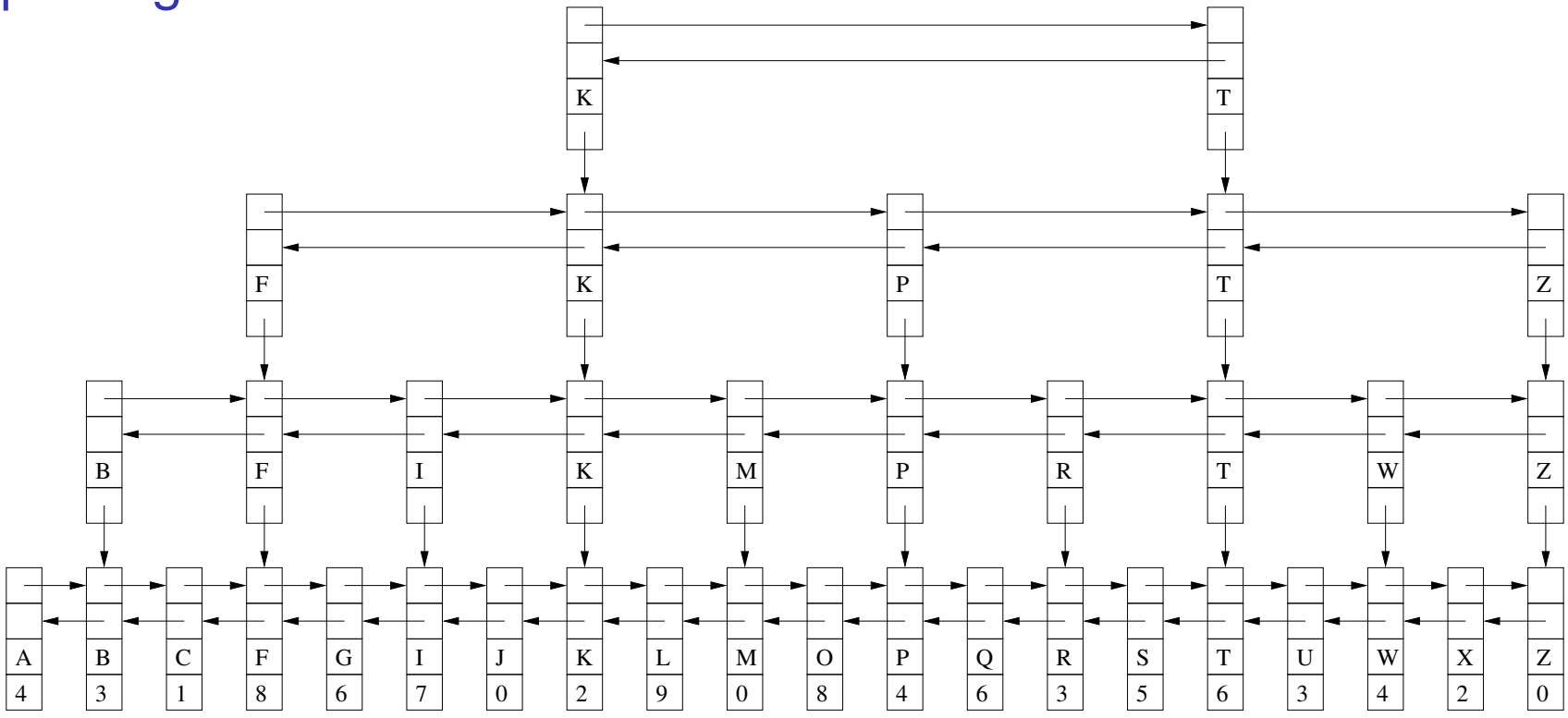
- But what if we keep doing this until there is only a *constant* number in the topmost list?
- So we will go at most one step in *every* list.
- How many lists will there be?
- Hint: each list has half as many elements as the one below it.
- I hope you all *immediately* thought $\log_2 n$.
- So $\log_2 n$ lists and we go at most one step in each one?

Keep doing it until...



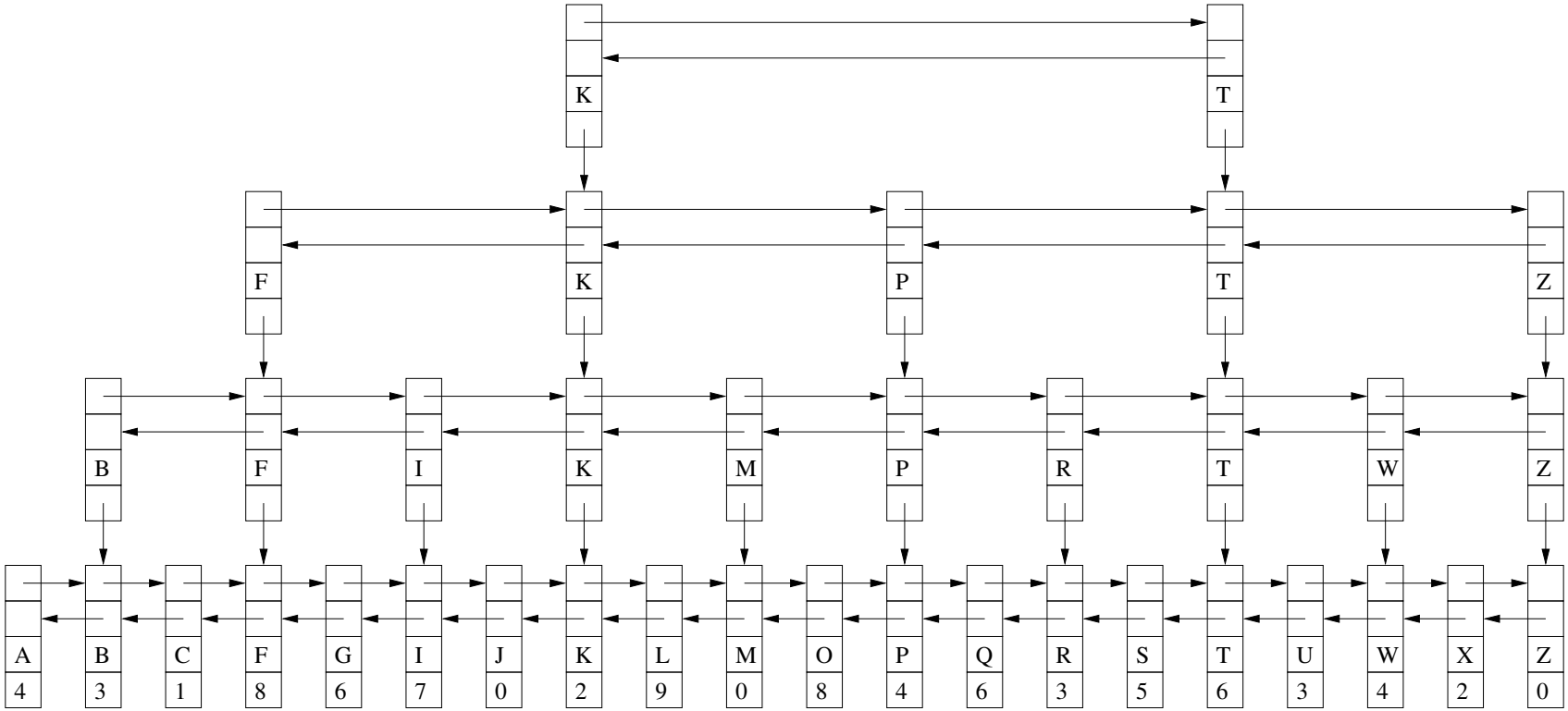
- But what if we keep doing this until there is only a *constant* number in the topmost list?
- So we will go at most one step in *every* list.
- How many lists will there be?
- Hint: each list has half as many elements as the one below it.
- I hope you all *immediately* thought $\log_2 n$.
- So $\log_2 n$ lists and we go at most one step in each one?
- So getting the (original) Entry for a key takes $O(\log n)$ time.

Keep doing it until...

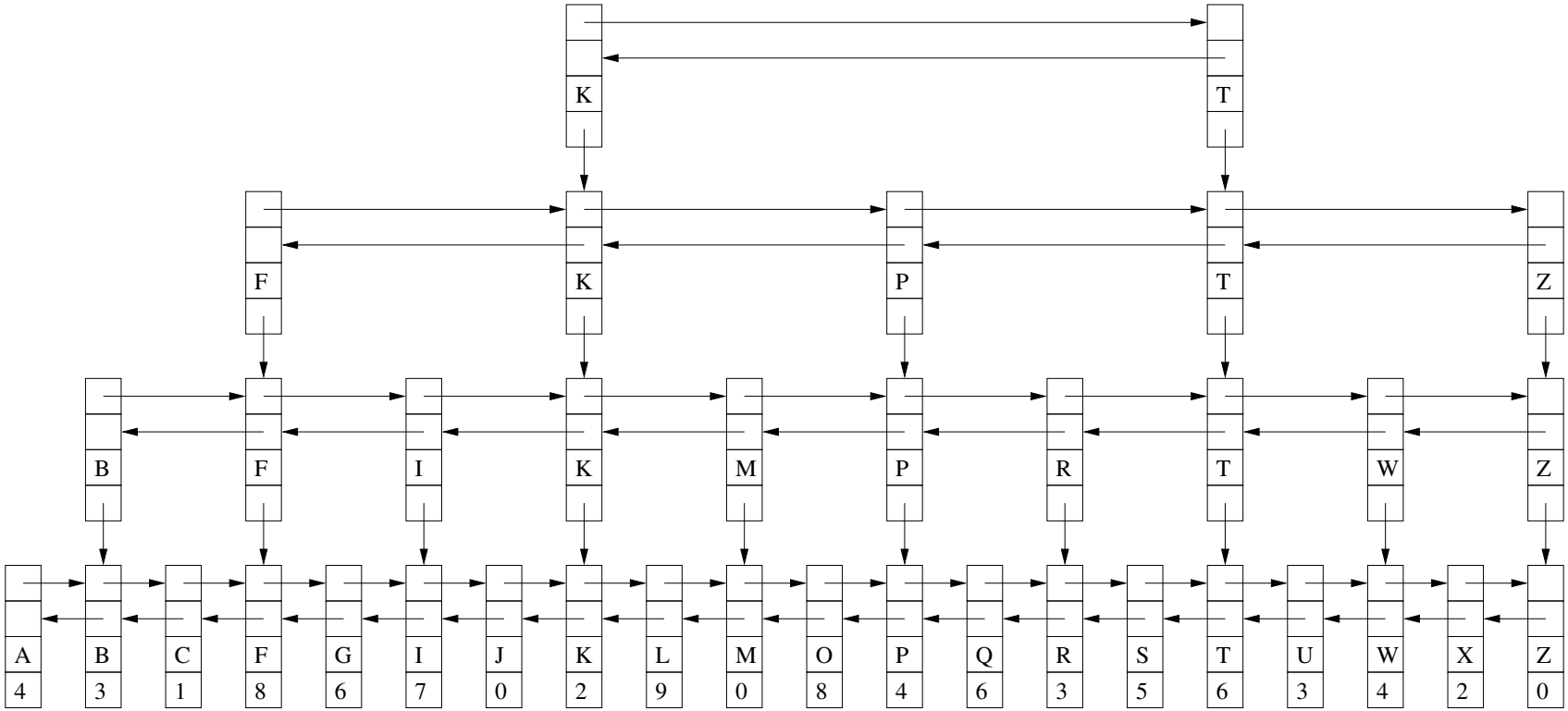


- But what if we keep doing this until there is only a *constant* number in the topmost list?
- So we will go at most one step in *every* list.
- How many lists will there be?
- Hint: each list has half as many elements as the one below it.
- I hope you all *immediately* thought $\log_2 n$.
- So $\log_2 n$ lists and we go at most one step in each one?
- So getting the (original) Entry for a key takes $O(\log n)$ time.
- This is a SKIP LIST.

Skip List

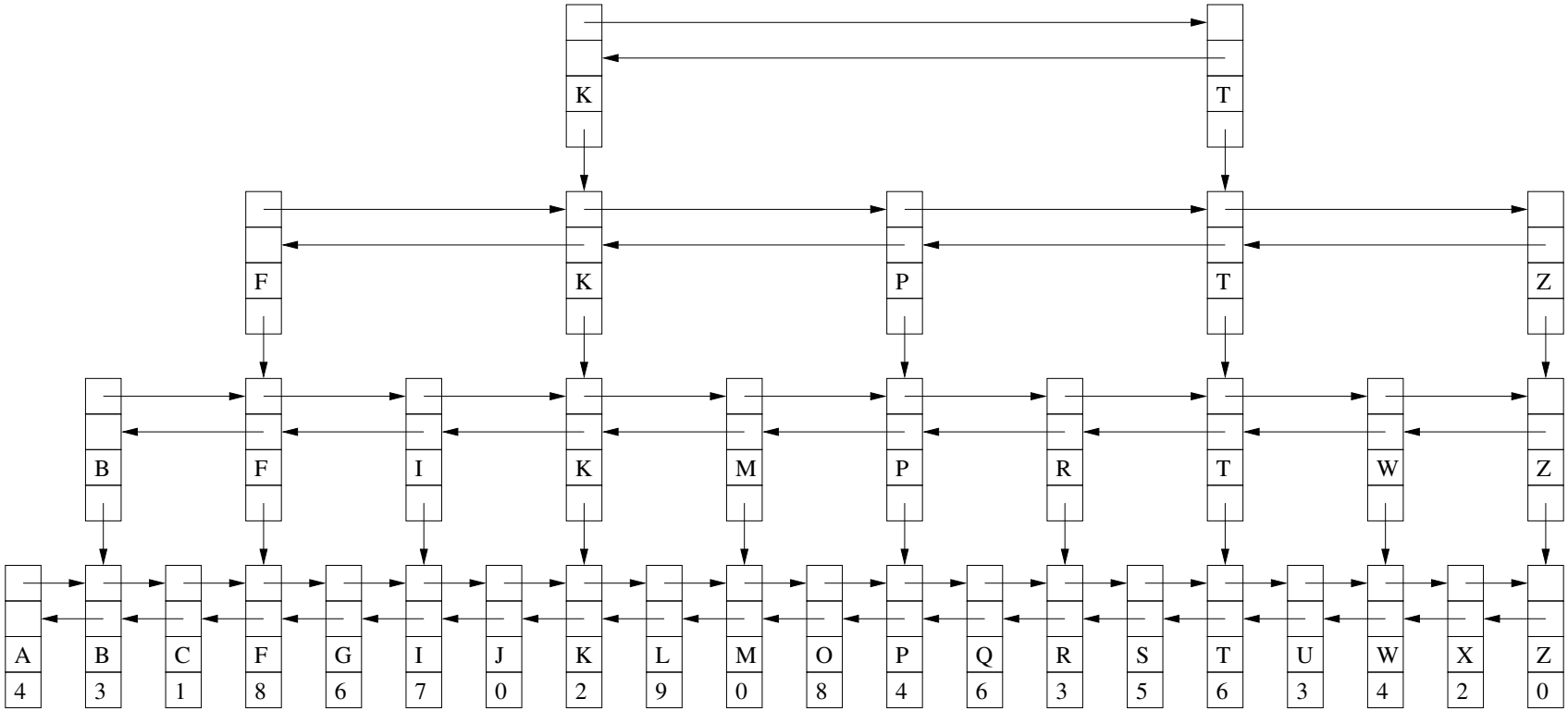

K
T
F
K
P
T
Z
B
F
I
K
M
P
R
T
W
Z
A 4
B 3
C 1
F 8
G 6
I 7
J 0
K 2
L 9
M 0
O 8
P 4
Q 6
R 3
S 5
T 6
U 3
W 4
X 2
Z 0

Skip List



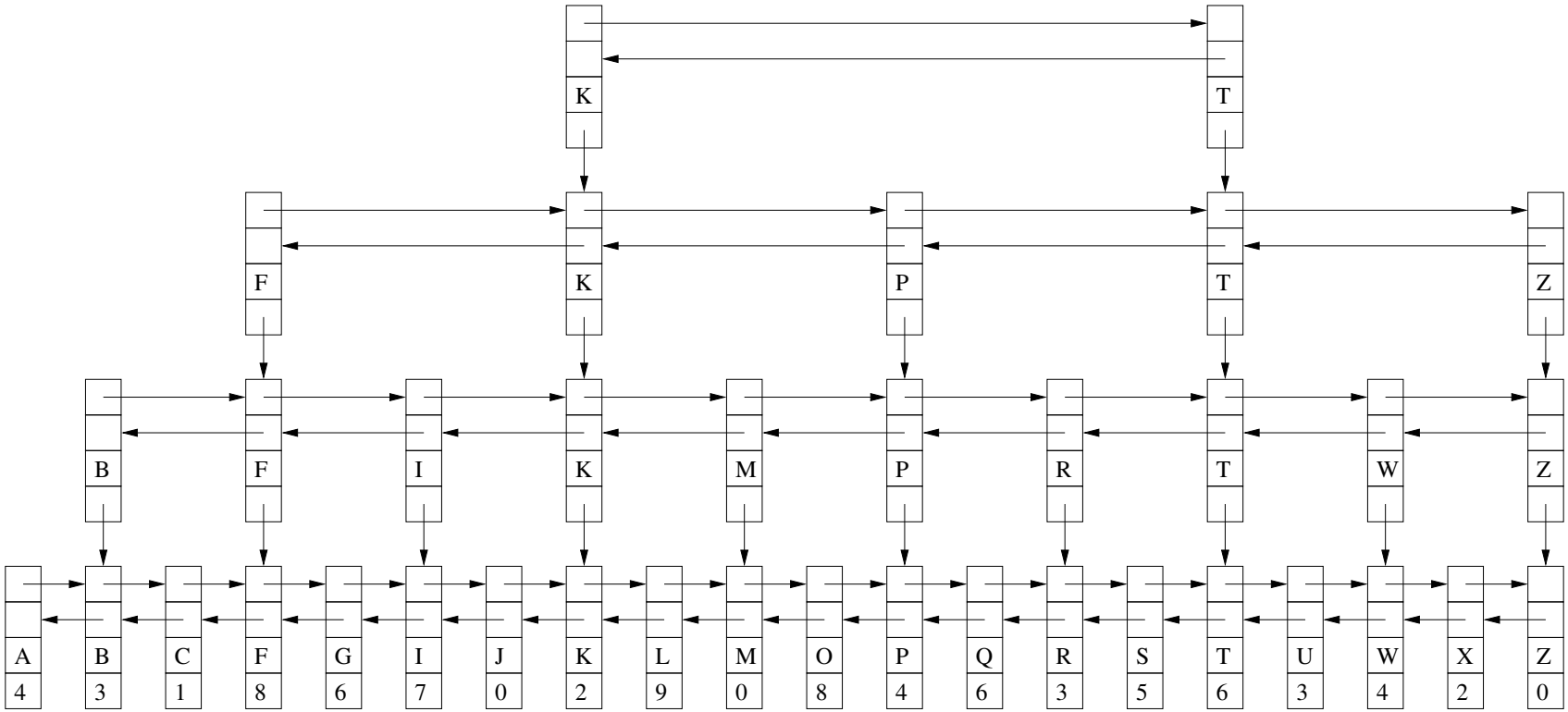
- By the way, how much extra *space* does a skip list use?

Skip List

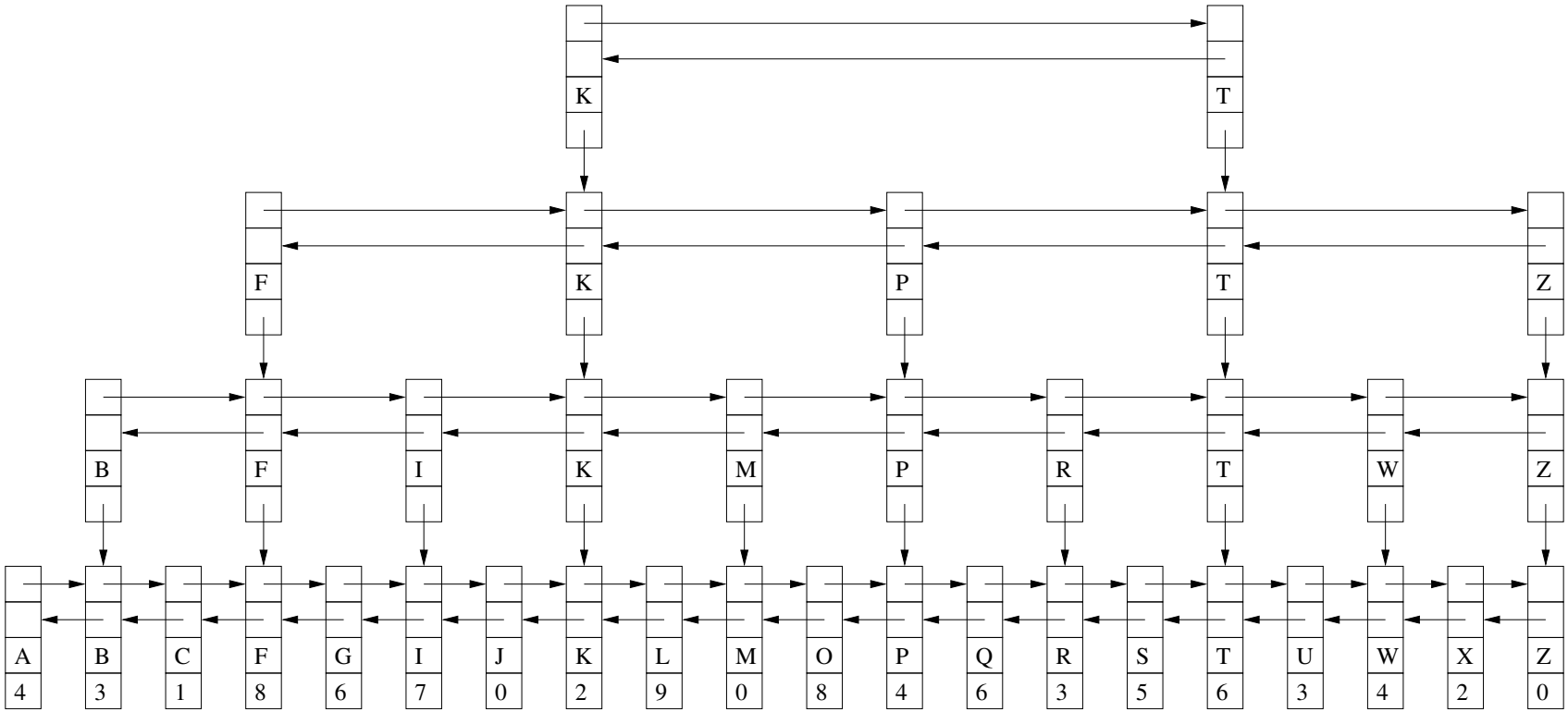


- By the way, how much extra *space* does a skip list use?
- $n/2 + n/4 + n/8 + \cdots = ???$

Skip List


K
T
F
K
P
T
Z
B
F
I
K
M
P
R
T
W
Z
A 4
B 3
C 1
F 8
G 6
I 7
J 0
K 2
L 9
M 0
O 8
P 4
Q 6
R 3
S 5
T 6
U 3
W 4
X 2
Z 0

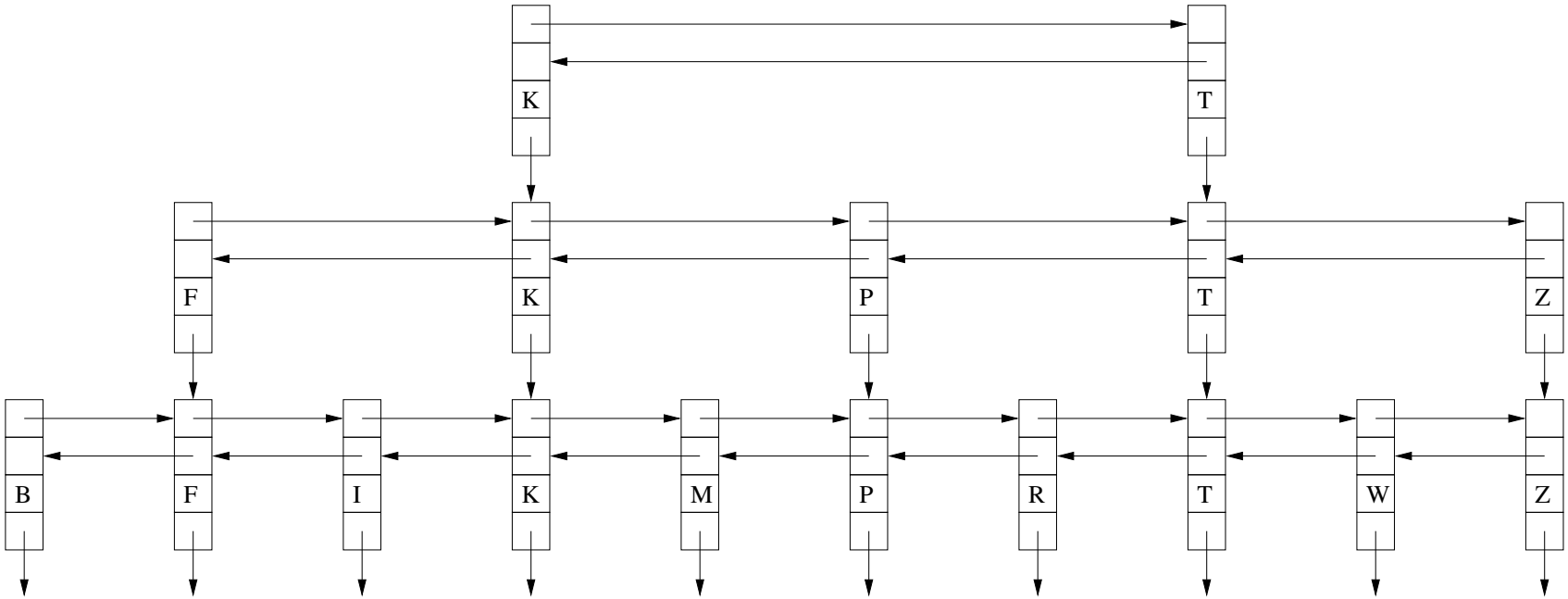
Skip List



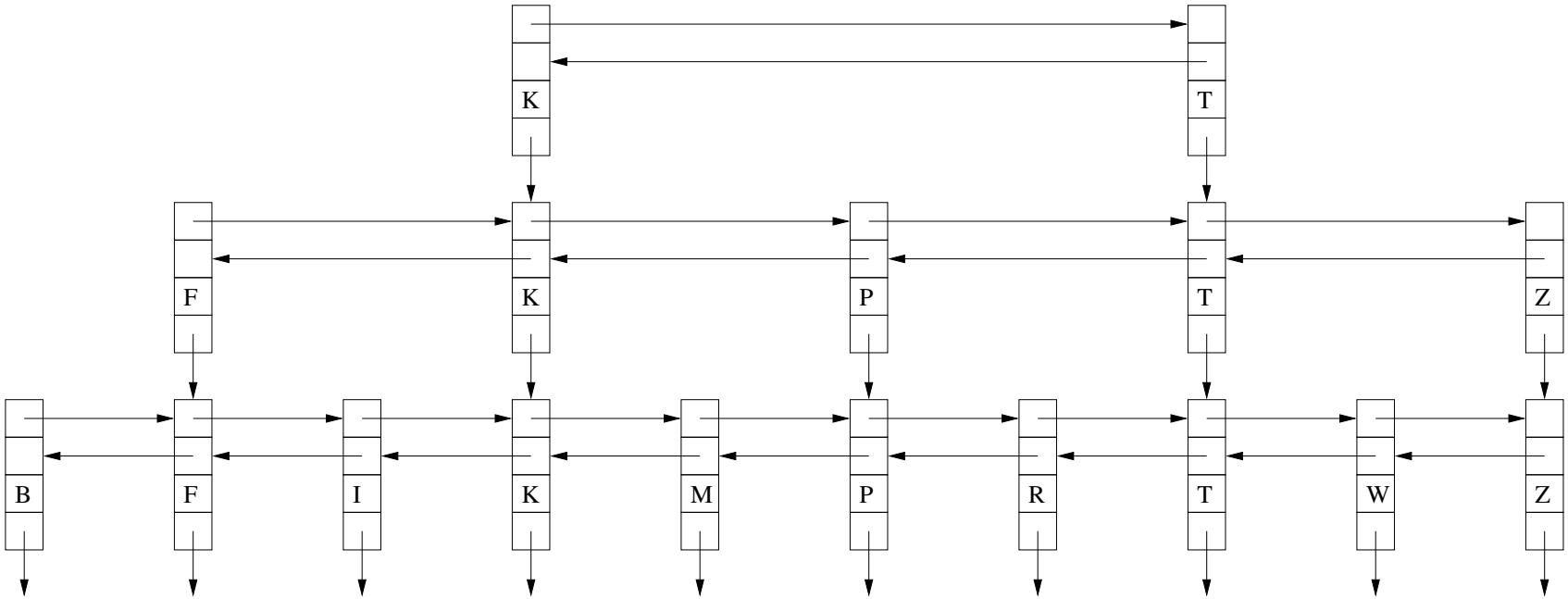
- So if this is a

  SkipList<String , Integer>

Skip List



Skip List



- Isn't this a

  `SkipList<String, Entry>`

  ?

Skip List Implementation

So a

```
SkipList<String, Integer>
```

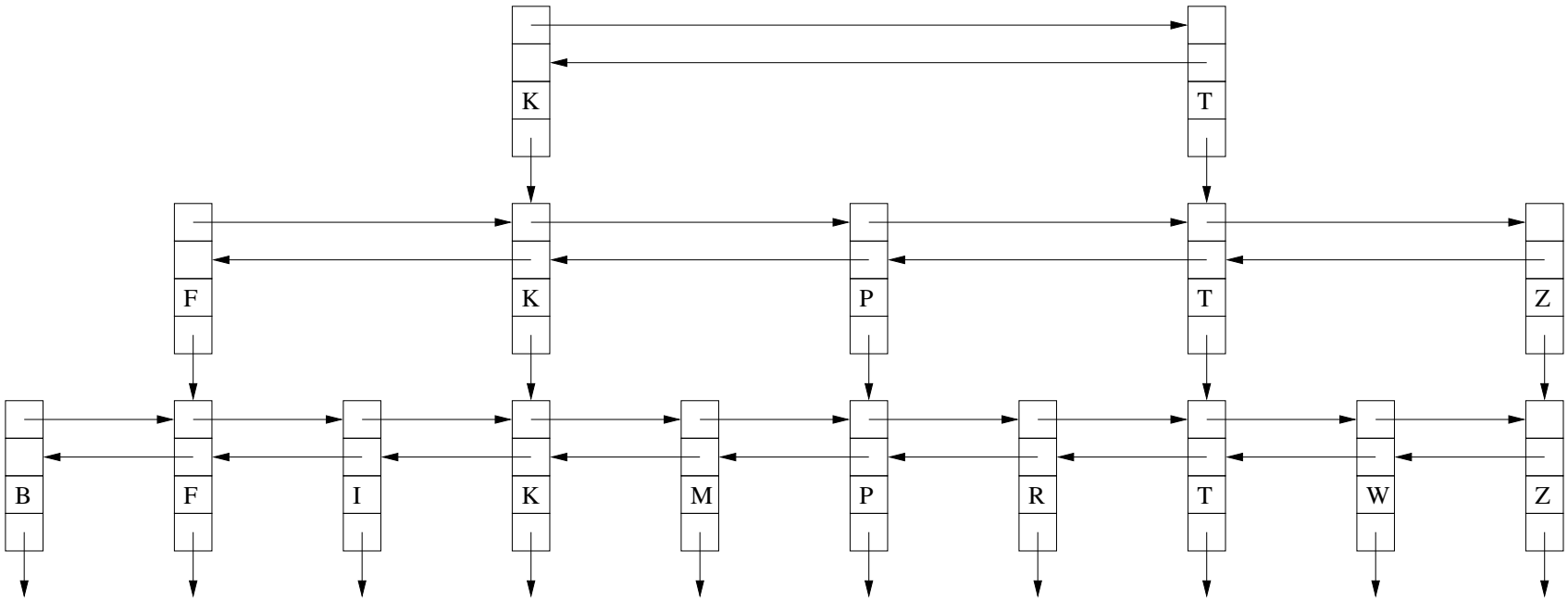
Skip List Implementation

So a

```
SkipList<String, Integer>
```

is a

```
SkipList<String, Entry>
```



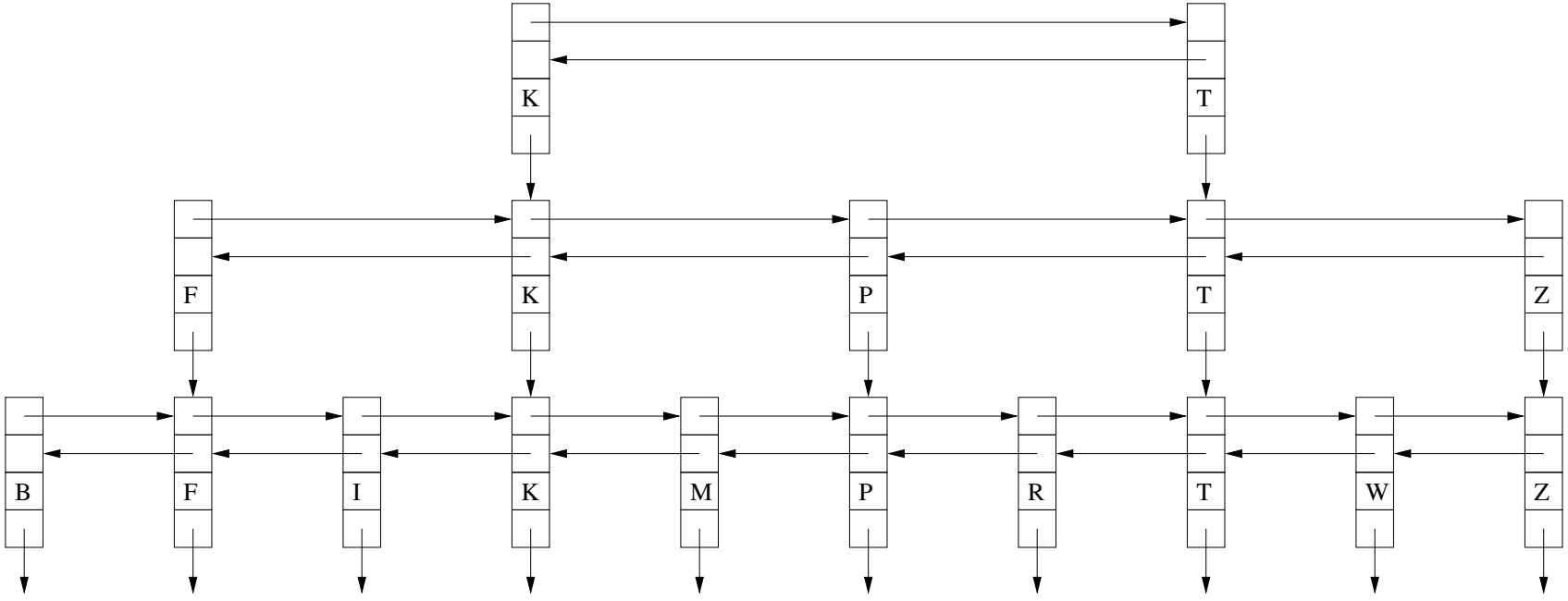
Skip List Implementation

So a

SkipList<String , Integer>

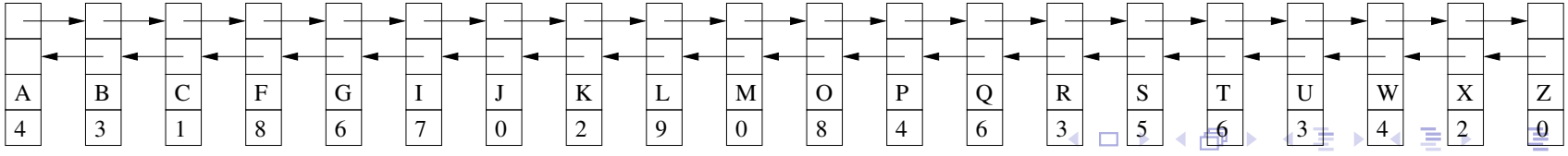
is a

SkipList<String , Entry>



plus a

LinkedMap<String , Integer>



Switching to ASCII

							K							T					
			F				K				P			T				Z	
	B		F		I		K		M		P		R		T		W		Z
A	B	C	F	G	I	J	K	L	M	O	P	Q	R	S	T	U	W	X	Z
4	3	1	8	6	7	0	2	9	0	8	4	6	3	5	6	3	4	2	0

Switching to ASCII

```
                                K                               T
                F               K               P               T               Z
    B           F       I       K       M       P       R       T       W       Z
A   B   C   F   G   I   J   K   L   M   O   P   Q   R   S   T   U   W   X   Z
4   3   1   8   6   7   0   2   9   0   8   4   6   3   5   6   3   4   2   0
```

- The figures take a lot of time to draw.

Switching to ASCII

```
                                K                               T
                F               K               P               T               Z
    B           F       I       K       M       P       R       T       W       Z
A   B   C   F   G   I   J   K   L   M   O   P   Q   R   S   T   U   W   X   Z
4   3   1   8   6   7   0   2   9   0   8   4   6   3   5   6   3   4   2   0
```

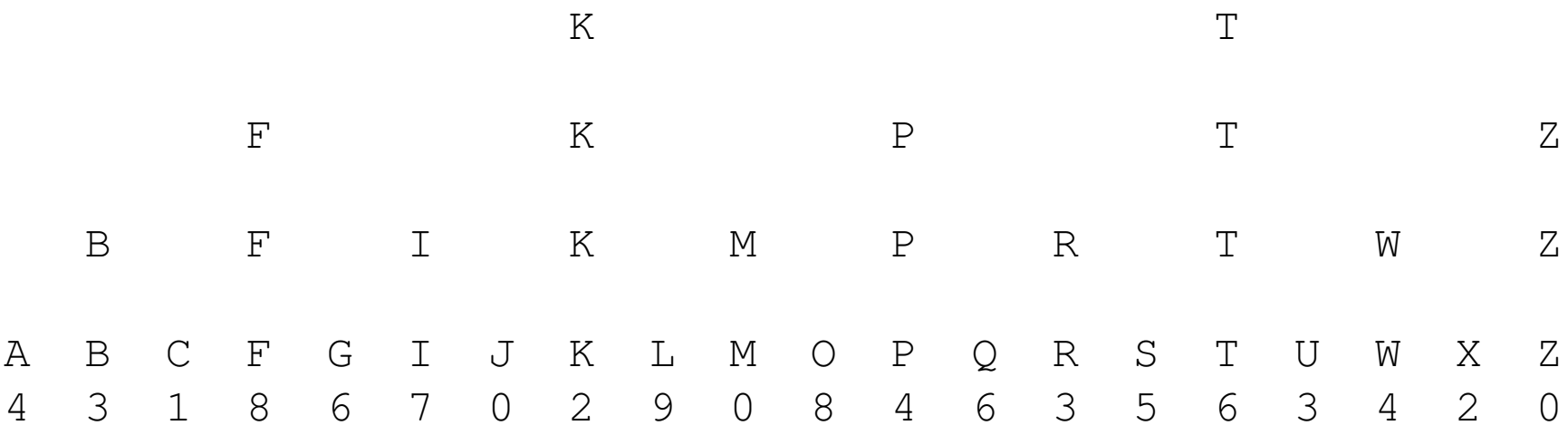
- The figures take a lot of time to draw.
- Here is an ASCII version.

Switching to ASCII

```
                                K                               T
                F               K               P               T               Z
    B           F       I       K       M       P       R       T       W       Z
A   B   C   F   G   I   J   K   L   M   O   P   Q   R   S   T   U   W   X   Z
4   3   1   8   6   7   0   2   9   0   8   4   6   3   5   6   3   4   2   0
```

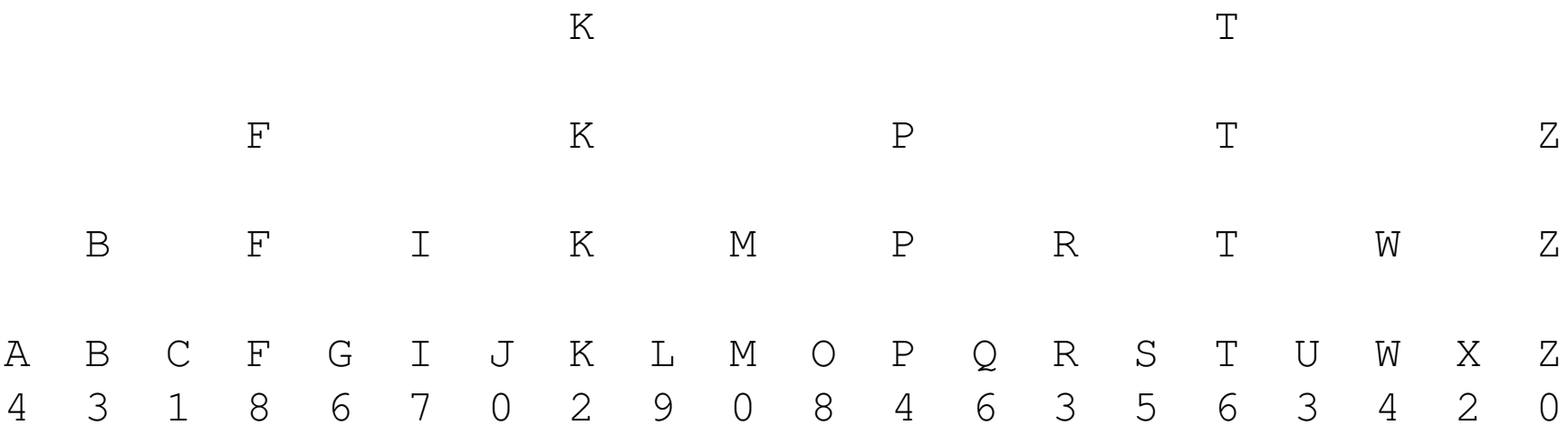
- The figures take a lot of time to draw.
- Here is an ASCII version.
- You have to imagine the boxes.

Switching to ASCII



- The figures take a lot of time to draw.
- Here is an ASCII version.
- You have to imagine the boxes.
- And the next and previous arrows.

Switching to ASCII



- The figures take a lot of time to draw.
- Here is an ASCII version.
- You have to imagine the boxes.
- And the next and previous arrows.
- And the downward arrow when the value is an Entry in a lower list.

Every Other?

Every Other?

- What if someone *removes* every other key: B, F, I, K, M, P, R, T, W, Z?

Every Other?

- What if someone *removes* every other key: B, F, I, K, M, P, R, T, W, Z?
- Then we are left with just one sorted linked list.

Every Other?

- What if someone *removes* every other key: B, F, I, K, M, P, R, T, W, Z?
- Then we are left with just one sorted linked list.
- And get takes $\mathrm{O}(n)$ again. ($n/2$ is still $\mathrm{O}(n)$.)

Every Other?

- What if someone *removes* every other key: B, F, I, K, M, P, R, T, W, Z?
- Then we are left with just one sorted linked list.
- And get takes $\mathrm{O}(n)$ again. ($n/2$ is still $\mathrm{O}(n)$.)
- How can we select "every other" key in each list that won't be spoiled by removals?

Every Other?

- What if someone *removes* every other key: B, F, I, K, M, P, R, T, W, Z?
- Then we are left with just one sorted linked list.
- And get takes $\mathrm{O}(n)$ again. ($n/2$ is still $\mathrm{O}(n)$.)
- How can we select "every other" key in each list that won't be spoiled by removals?


- Answer: flip a coin!

Every Other?

- What if someone *removes* every other key: B, F, I, K, M, P, R, T, W, Z?
- Then we are left with just one sorted linked list.
- And get takes $\mathrm{O}(n)$ again. ($n/2$ is still $\mathrm{O}(n)$.)
- How can we select "every other" key in each list that won't be spoiled by removals?


- Answer: flip a coin!
- For each key in a list, I flip a coin. If it is heads, it goes into the next higher list.

Every Other?

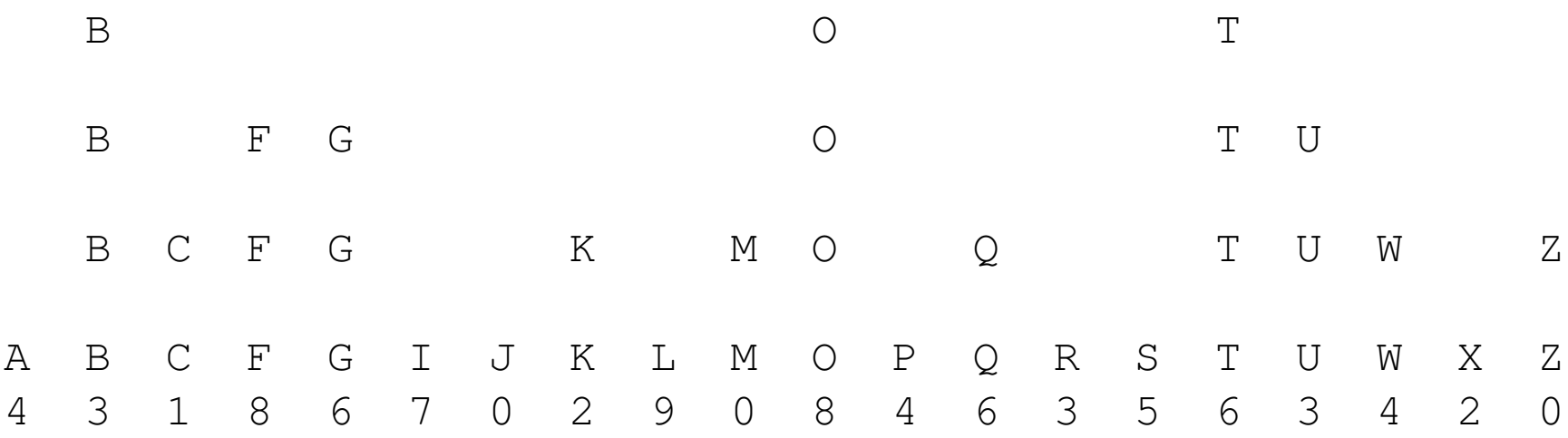
- What if someone *removes* every other key: B, F, I, K, M, P, R, T, W, Z?
- Then we are left with just one sorted linked list.
- And get takes $O(n)$ again. ($n/2$ is still $O(n)$.)
- How can we select "every other" key in each list that won't be spoiled by removals?


- Answer: flip a coin!
- For each key in a list, I flip a coin. If it is heads, it goes into the next higher list.
- If you can predict which of my keys will flip heads, you don't need to be studying Computer Science!

Randomized Skip List

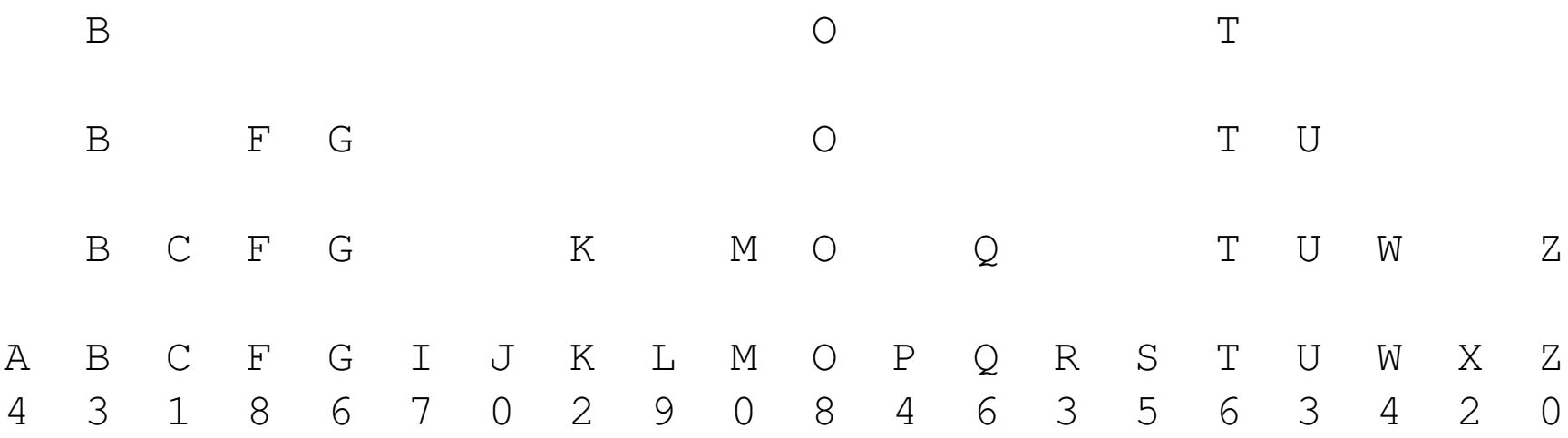
	B								O					T					
	B		F	G					O					T	U				
	B	C	F	G			K		M	O		Q		T	U	W		Z	
A	B	C	F	G	I	J	K	L	M	O	P	Q	R	S	T	U	W	X	Z
4	3	1	8	6	7	0	2	9	0	8	4	6	3	5	6	3	4	2	0

Randomized Skip List



- On *average* you take one step on each level.

Randomized Skip List



- On *average* you take one step on each level.
- Why? Suppose all n of you flipped a coin.

Randomized Skip List

```
    B                       O               T
    B       F   G           O               T   U
    B   C   F   G       K   M   O       Q   T   U   W       Z
A   B   C   F   G   I   J   K   L   M   O   P   Q   R   S   T   U   W   X   Z
4   3   1   8   6   7   0   2   9   0   8   4   6   3   5   6   3   4   2   0
```

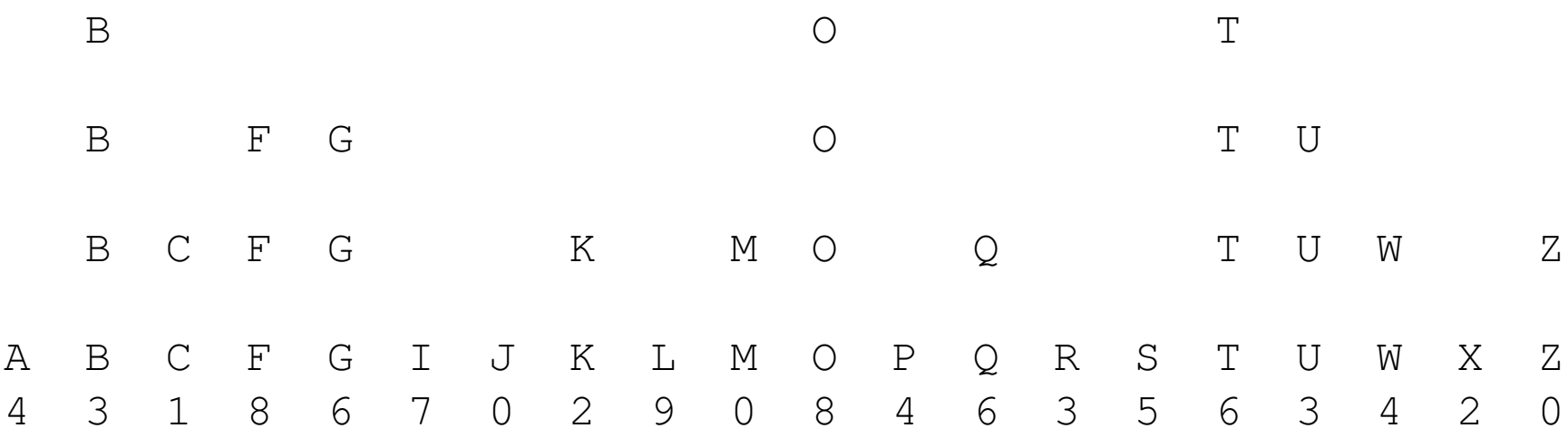
- On *average* you take one step on each level.
- Why? Suppose all n of you flipped a coin.
- About $n/2$ of you get tails and are allowed to flip again.

Randomized Skip List

```
    B                                   O                   T
    B       F   G                       O                   T   U
    B   C   F   G           K       M   O       Q           T   U   W       Z
A   B   C   F   G   I   J   K   L   M   O   P   Q   R   S   T   U   W   X   Z
4   3   1   8   6   7   0   2   9   0   8   4   6   3   5   6   3   4   2   0
```

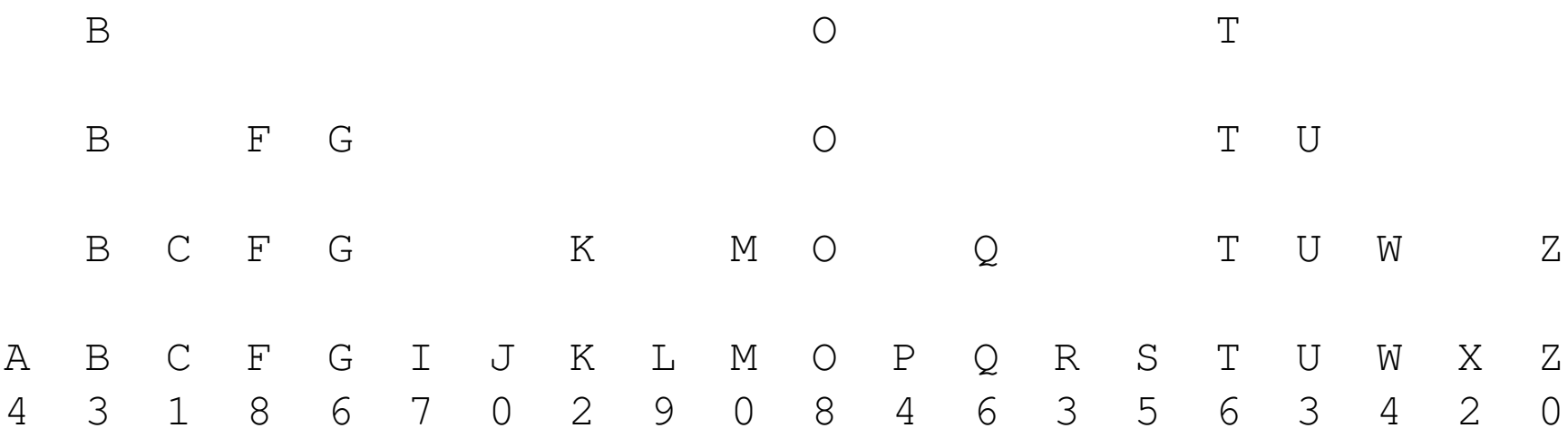
- On *average* you take one step on each level.
- Why? Suppose all n of you flipped a coin.
- About $n/2$ of you get tails and are allowed to flip again.
- About $n/4$ of you get tails and are allowed to flip again.

Randomized Skip List



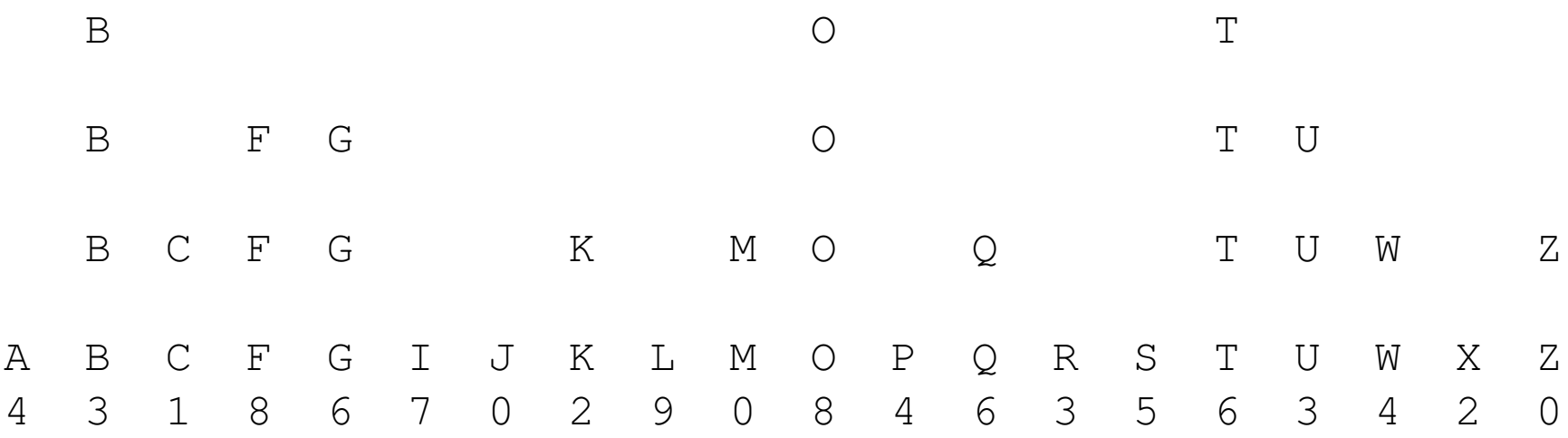
- On *average* you take one step on each level.
- Why? Suppose all n of you flipped a coin.
- About $n/2$ of you get tails and are allowed to flip again.
- About $n/4$ of you get tails and are allowed to flip again.
- About $n/8$ of you get tails and are allowed to flip again.

Randomized Skip List



- On *average* you take one step on each level.
- Why? Suppose all n of you flipped a coin.
- About $n/2$ of you get tails and are allowed to flip again.
- About $n/4$ of you get tails and are allowed to flip again.
- About $n/8$ of you get tails and are allowed to flip again.
- Etc. Total number of tails flipped is $n/2 + n/4 + n/8 + \cdots = n$

Randomized Skip List



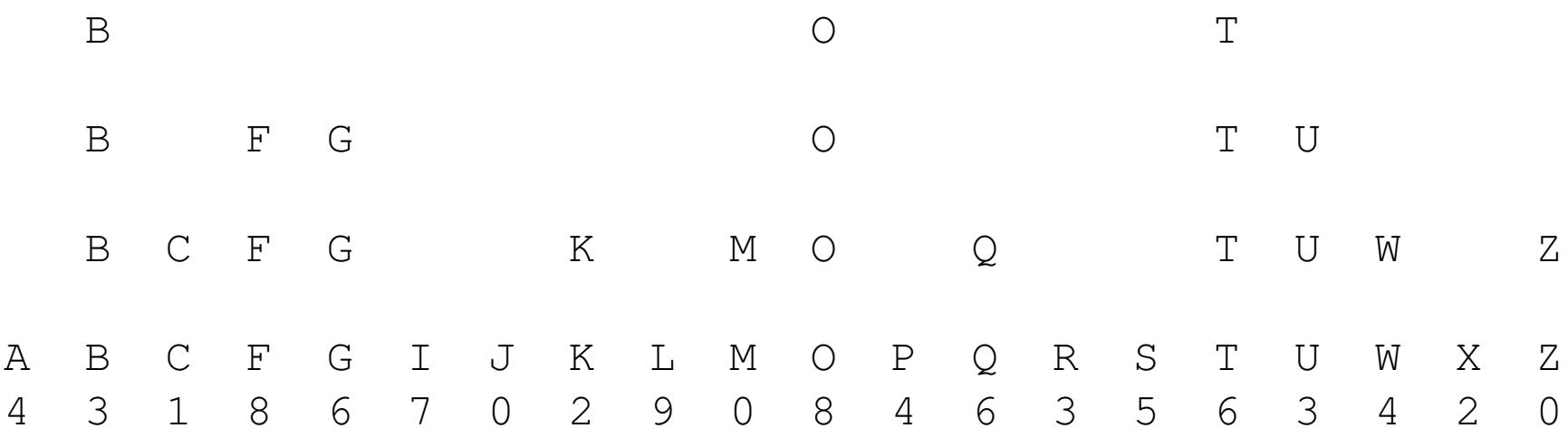
- On *average* you take one step on each level.
- Why? Suppose all n of you flipped a coin.
- About $n/2$ of you get tails and are allowed to flip again.
- About $n/4$ of you get tails and are allowed to flip again.
- About $n/8$ of you get tails and are allowed to flip again.
- Etc. Total number of tails flipped is $n/2 + n/4 + n/8 + \cdots = n$
- So on average everyone flips one tail before getting heads.

Randomized Skip List

```
    B                                   O                   T
    B       F   G                       O                   T   U
    B   C   F   G           K       M   O       Q           T   U   W       Z
A   B   C   F   G   I   J   K   L   M   O   P   Q   R   S   T   U   W   X   Z
4   3   1   8   6   7   0   2   9   0   8   4   6   3   5   6   3   4   2   0
```

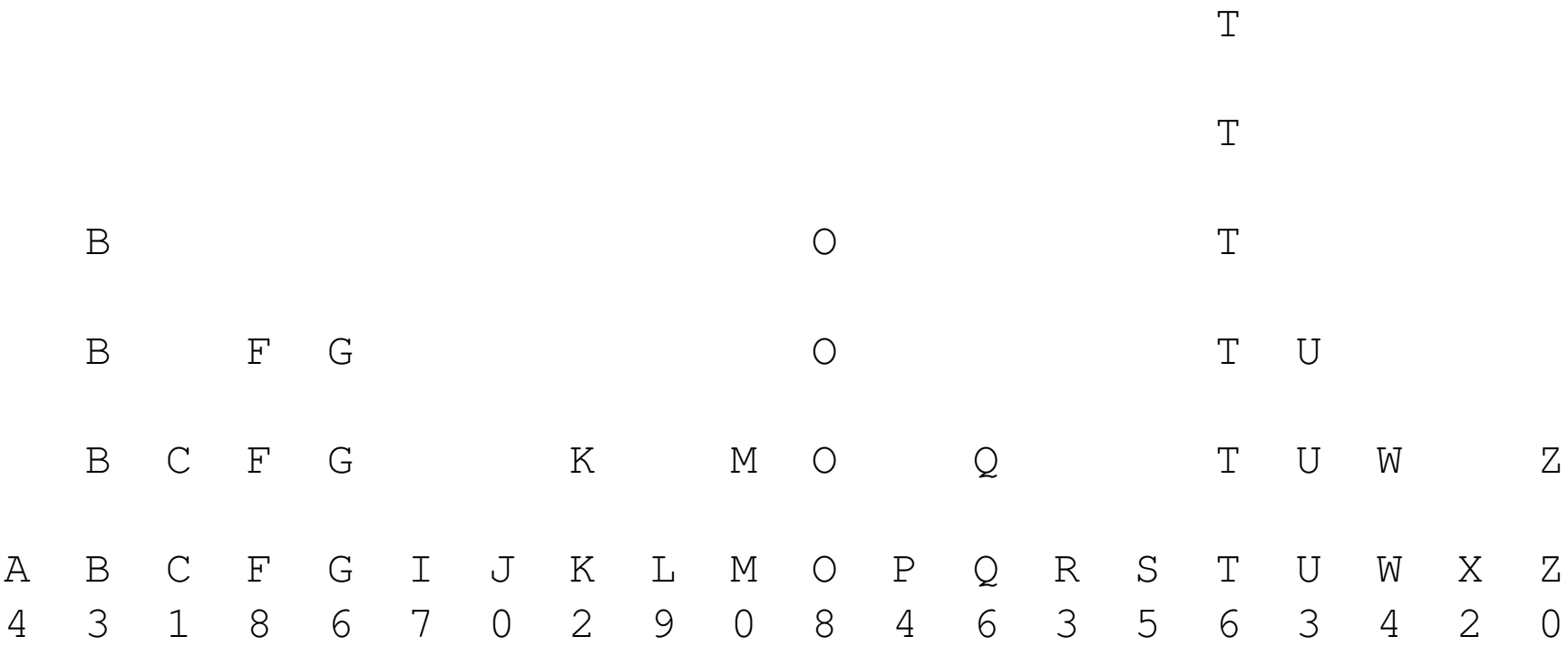
- On *average* you take one step on each level.
- Why? Suppose all n of you flipped a coin.
- About $n/2$ of you get tails and are allowed to flip again.
- About $n/4$ of you get tails and are allowed to flip again.
- About $n/8$ of you get tails and are allowed to flip again.
- Etc. Total number of tails flipped is $n/2 + n/4 + n/8 + \cdots = n$
- So on average everyone flips one tail before getting heads.
- So on average you skip one key before promoting the next one to a higher level.

Randomized Skip List

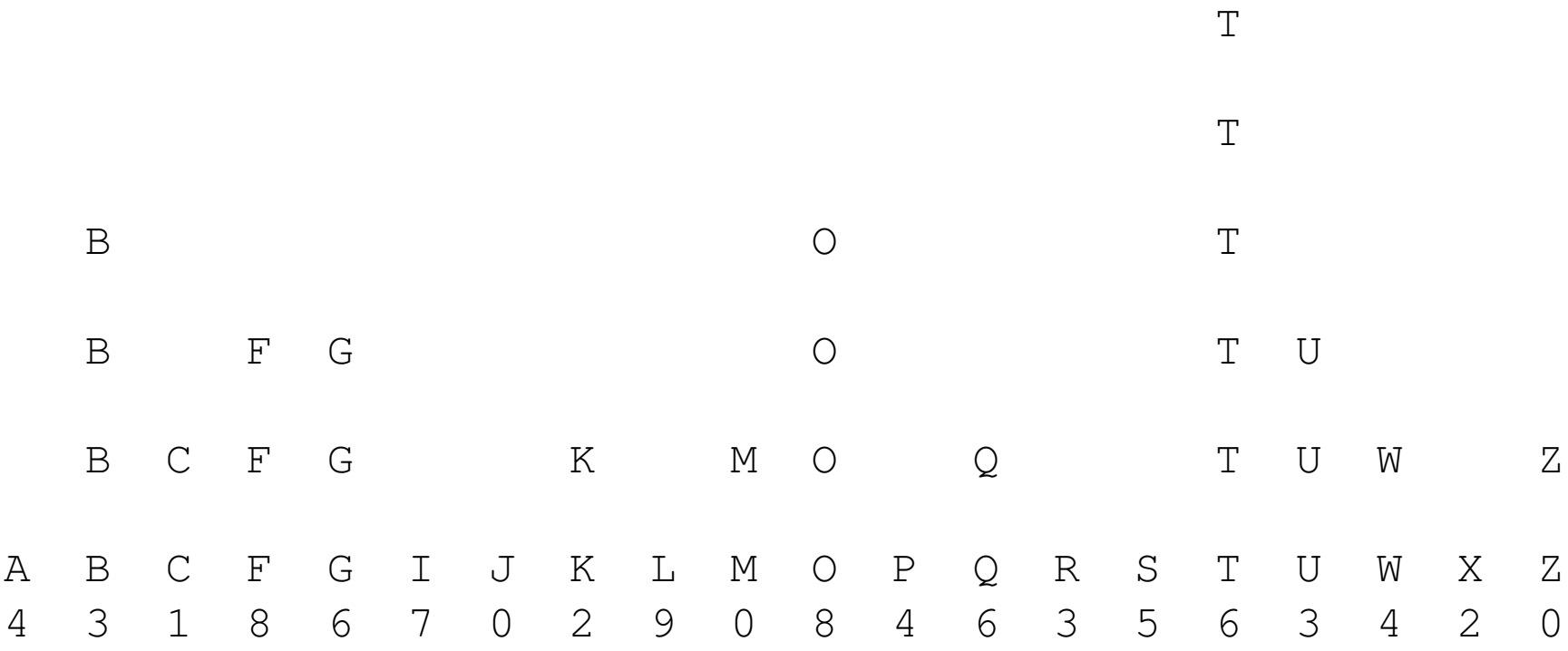


- On *average* you take one step on each level.
- Why? Suppose all n of you flipped a coin.
- About $n/2$ of you get tails and are allowed to flip again.
- About $n/4$ of you get tails and are allowed to flip again.
- About $n/8$ of you get tails and are allowed to flip again.
- Etc. Total number of tails flipped is $n/2 + n/4 + n/8 + \cdots = n$
- So on average everyone flips one tail before getting heads.
- So on average you skip one key before promoting the next one to a higher level.
- You will learn this more formally in MTH224.

Randomized Skip List

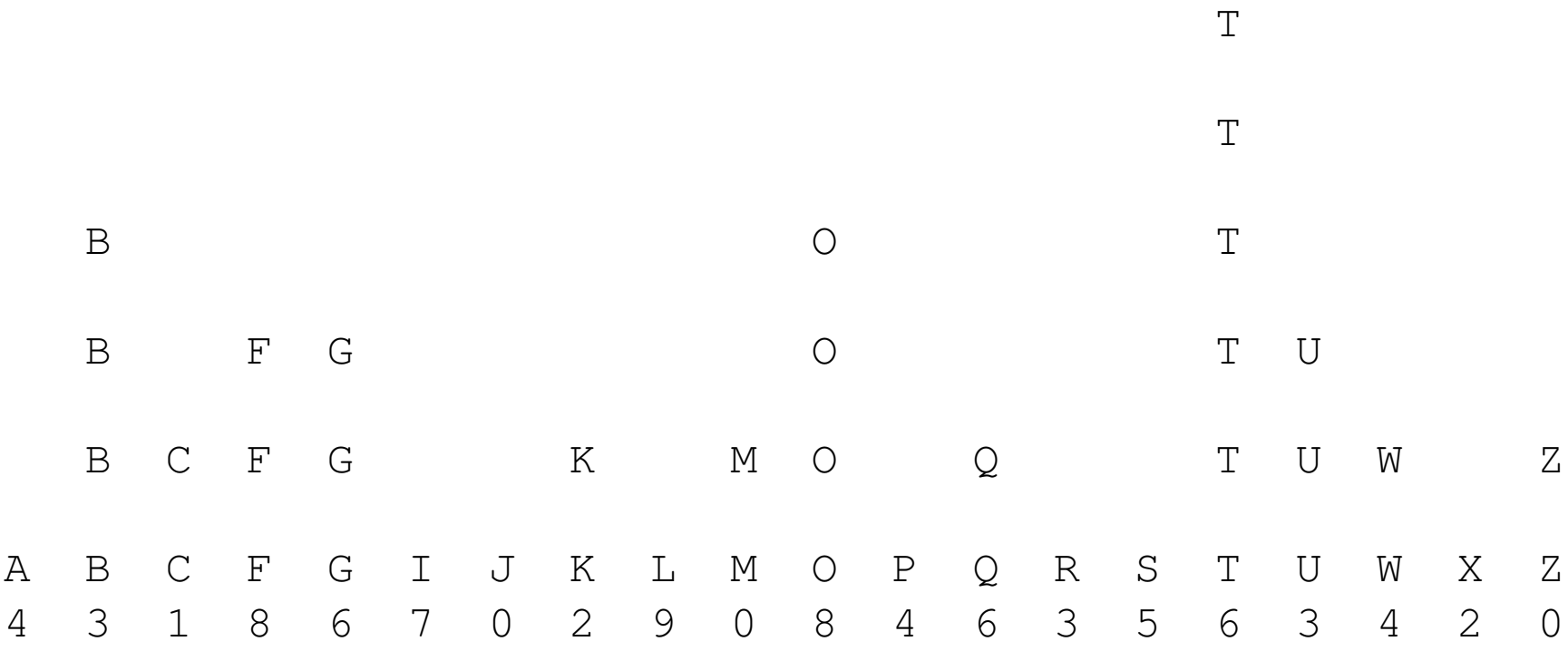


Randomized Skip List



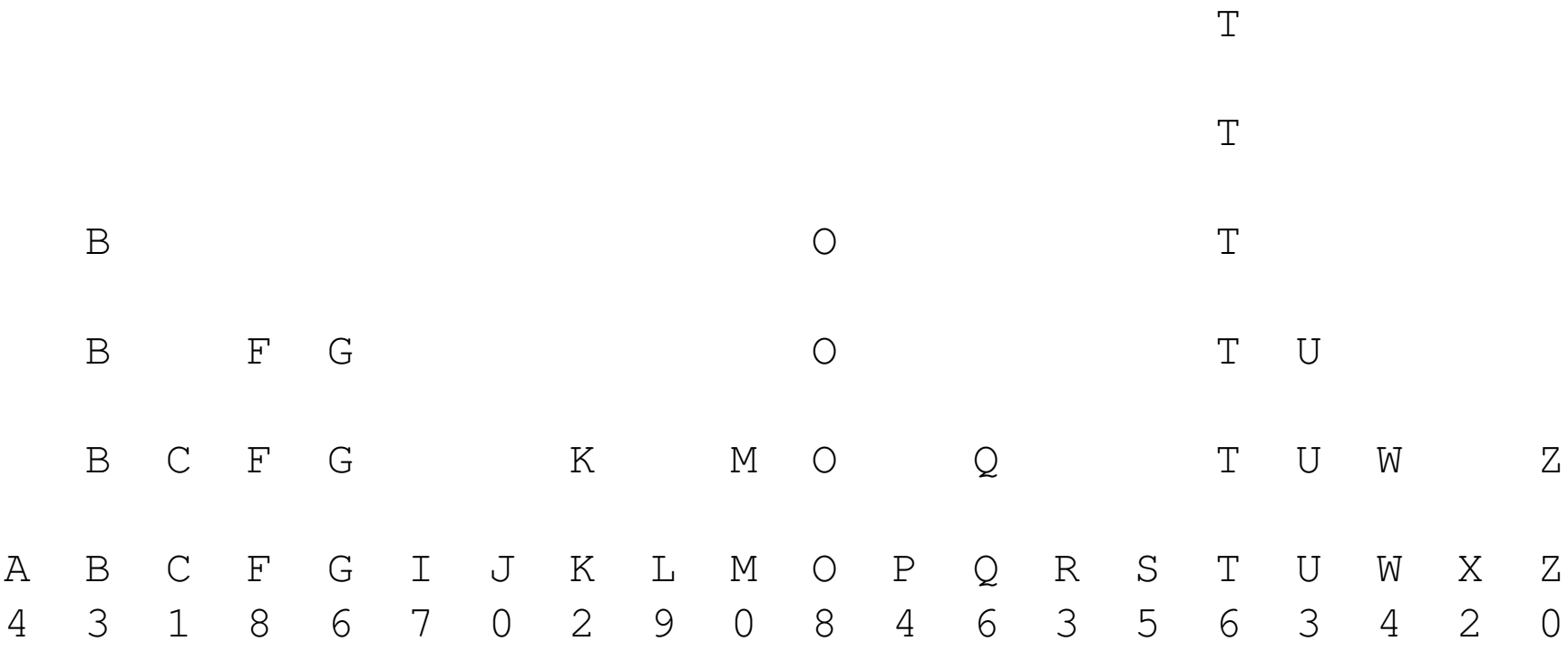
- Actually, it usually looks more like this.

Randomized Skip List



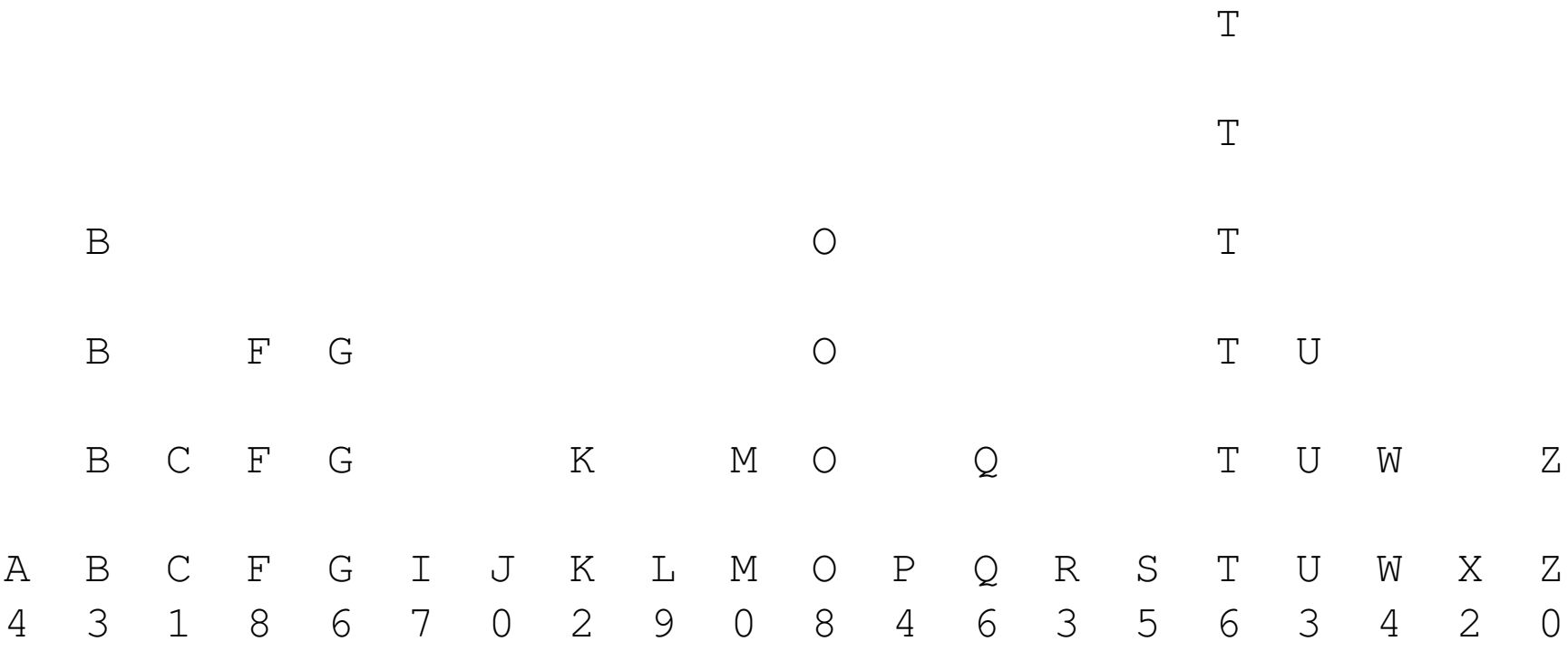
- Actually, it usually looks more like this.
- There always seems to be one key that is really lucky.

Randomized Skip List



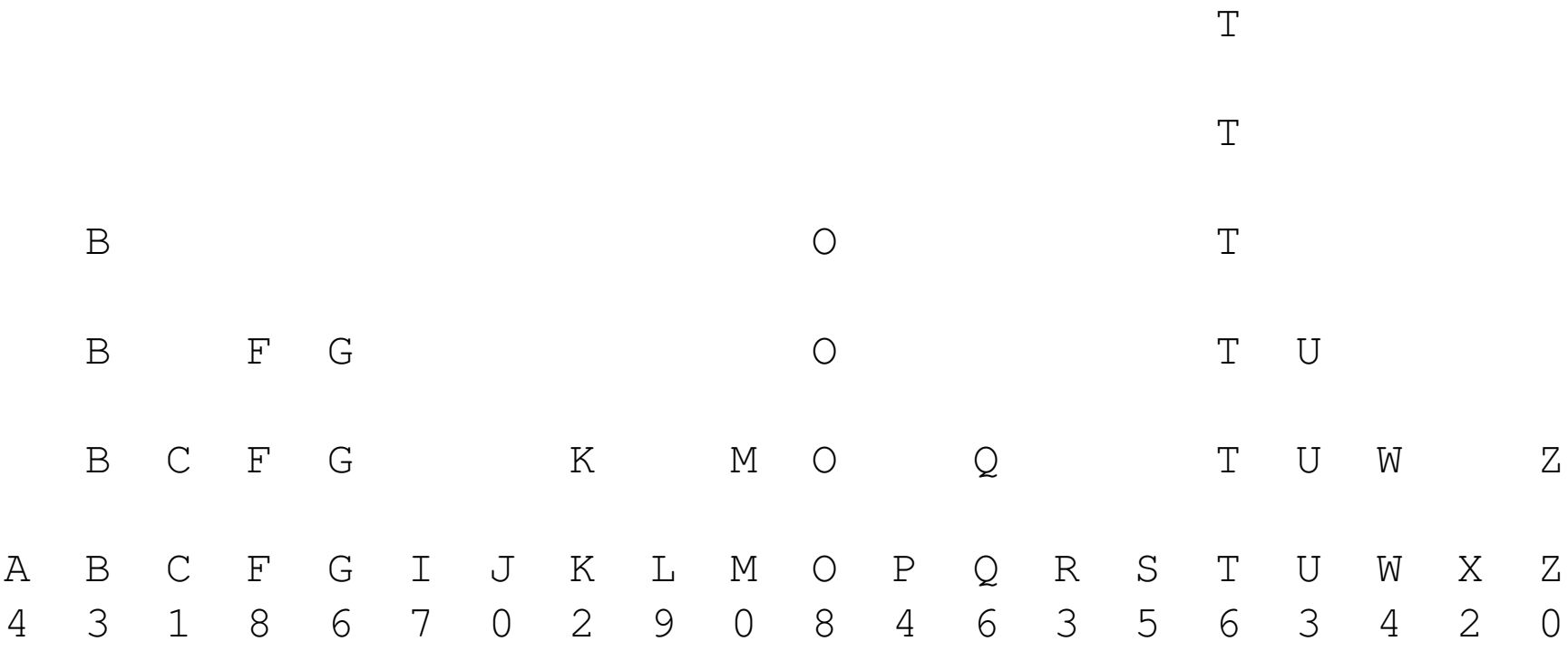
- Actually, it usually looks more like this.
- There always seems to be one key that is really lucky.
- As long as it keeps flipping heads,

Randomized Skip List



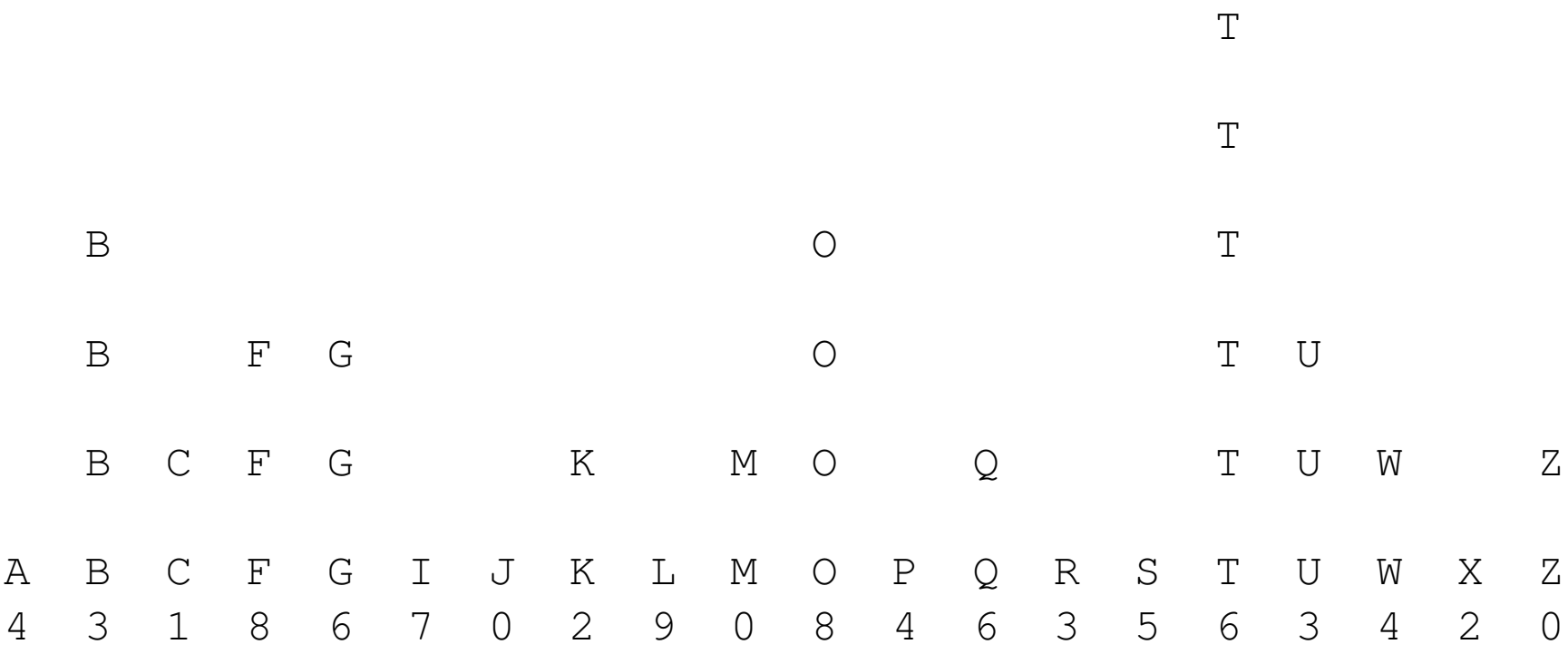
- Actually, it usually looks more like this.
- There always seems to be one key that is really lucky.
- As long as it keeps flipping heads,
- it goes into the next higher list.

Randomized Skip List



- Actually, it usually looks more like this.
- There always seems to be one key that is really lucky.
- As long as it keeps flipping heads,
- it goes into the next higher list.
- Even if that means it is by itself.

Randomized Skip List



- Actually, it usually looks more like this.
- There always seems to be one key that is really lucky.
- As long as it keeps flipping heads,
- it goes into the next higher list.
- Even if that means it is by itself.
- If the list doesn't exist already, you have to create it.

Lab and Homework

Lab and Homework

- For lab yesterday, you implemented LinkedMap

Lab and Homework

- For lab yesterday, you implemented LinkedMap
- For the homework, you will implement SkipMap: a Skip List based Map.

Lab and Homework

- For lab yesterday, you implemented LinkedMap
- For the homework, you will implement SkipMap: a Skip List based Map.
- A SkipMap<K,V> is a (extends) LinkedMap<K,V>

Lab and Homework

- For lab yesterday, you implemented LinkedMap
- For the homework, you will implement SkipMap: a Skip List based Map.
- A SkipMap<K,V> is a (extends) LinkedMap<K,V>
- plus a SkipMap<K,Entry>

Lab and Homework

- For lab yesterday, you implemented LinkedMap
- For the homework, you will implement SkipMap: a Skip List based Map.
- A SkipMap<K,V> is a (extends) LinkedMap<K,V>
- plus a SkipMap<K,Entry>
- containing the half of the keys

Lab and Homework

- For lab yesterday, you implemented LinkedMap
- For the homework, you will implement SkipMap: a Skip List based Map.
- A SkipMap<K,V> is a (extends) LinkedMap<K,V>
- plus a SkipMap<K,Entry>
- containing the half of the keys
- that throw “heads” when we add them to the LinkedMap.

SkipMap put

SkipMap put

- If put(key, value) creates a new Entry,

SkipMap put

- If put(key, value) creates a new Entry,
- it flips a coin.

SkipMap put

- If put(key, value) creates a new Entry,
- it flips a coin.
- If it comes up heads,

SkipMap put

- If put(key, value) creates a new Entry,
- it flips a coin.
- If it comes up heads,
- it puts that key and Entry into the (internal) SkipMap.

SkipMap put

- If put(key, value) creates a new Entry,
- it flips a coin.
- If it comes up heads,
- it puts that key and Entry into the (internal) SkipMap.
- It has to allocate the SkipMap the first time this happens.

SkipMap get

SkipMap get

- If find(key) needs to search the list

SkipMap get

- If find(key) needs to search the list
- it first calls find(key) on the (internal) SkipMap to get a starting Entry.

SkipMap get

- If find(key) needs to search the list
- it first calls find(key) on the (internal) SkipMap to get a starting Entry.
- From there is only has to go one Entry forward (on average).

SkipMap remove

SkipMap remove

- If remove(key) removes an Entry,

SkipMap remove

- If remove(key) removes an Entry,
- it also removes that key from the (internal) SkipMap.

SkipMap remove

- If remove(key) removes an Entry,
- it also removes that key from the (internal) SkipMap.
- If the SkipMap ends up empty, it sets it to null.

Summary

Summary

- A Skip List consists of $O(\log n)$ linked lists.

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.
  - In each list, a key goes into the next higher list if it flips heads.

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.
  - In each list, a key goes into the next higher list if it flips heads.
- Find starts at the first Entry in the highest list,

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.
  - In each list, a key goes into the next higher list if it flips heads.
- Find starts at the first Entry in the highest list,
  - moves forward to get as close as possible without going past,

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.
  - In each list, a key goes into the next higher list if it flips heads.
- Find starts at the first Entry in the highest list,
  - moves forward to get as close as possible without going past,
  - then goes down to the next lower list.

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.
  - In each list, a key goes into the next higher list if it flips heads.
- Find starts at the first Entry in the highest list,
  - moves forward to get as close as possible without going past,
  - then goes down to the next lower list.
- Add adds a new key to the lowest list,

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.
  - In each list, a key goes into the next higher list if it flips heads.
- Find starts at the first Entry in the highest list,
  - moves forward to get as close as possible without going past,
  - then goes down to the next lower list.
- Add adds a new key to the lowest list,
  - and a higher list for each time it flips heads.

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.
  - In each list, a key goes into the next higher list if it flips heads.
- Find starts at the first Entry in the highest list,
  - moves forward to get as close as possible without going past,
  - then goes down to the next lower list.
- Add adds a new key to the lowest list,
  - and a higher list for each time it flips heads.
  - Put has to keep creating new lists on top,

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.
  - In each list, a key goes into the next higher list if it flips heads.
- Find starts at the first Entry in the highest list,
  - moves forward to get as close as possible without going past,
  - then goes down to the next lower list.
- Add adds a new key to the lowest list,
  - and a higher list for each time it flips heads.
  - Put has to keep creating new lists on top,
  - as long as the key keeps flipping heads.

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.
  - In each list, a key goes into the next higher list if it flips heads.
- Find starts at the first Entry in the highest list,
  - moves forward to get as close as possible without going past,
  - then goes down to the next lower list.
- Add adds a new key to the lowest list,
  - and a higher list for each time it flips heads.
  - Put has to keep creating new lists on top,
  - as long as the key keeps flipping heads.
- Remove just removes a key from every list it appears in.

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.
  - In each list, a key goes into the next higher list if it flips heads.
- Find starts at the first Entry in the highest list,
  - moves forward to get as close as possible without going past,
  - then goes down to the next lower list.
- Add adds a new key to the lowest list,
  - and a higher list for each time it flips heads.
  - Put has to keep creating new lists on top,
  - as long as the key keeps flipping heads.
- Remove just removes a key from every list it appears in.
- Thanks to recursion, you only have to implement the bottom list plus a SkipList containing half the elements.

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.
  - In each list, a key goes into the next higher list if it flips heads.
- Find starts at the first Entry in the highest list,
  - moves forward to get as close as possible without going past,
  - then goes down to the next lower list.
- Add adds a new key to the lowest list,
  - and a higher list for each time it flips heads.
  - Put has to keep creating new lists on top,
  - as long as the key keeps flipping heads.
- Remove just removes a key from every list it appears in.
- Thanks to recursion, you only have to implement the bottom list plus a SkipList containing half the elements.
- Get, put, and remove are all $O(\log n)$ expected time.

Summary

- A Skip List consists of $O(\log n)$ linked lists.
  - The bottom list has all the keys and their actual values.
  - In each other list, the value is the Entry with the same key in the next lower list.
  - In each list, a key goes into the next higher list if it flips heads.
- Find starts at the first Entry in the highest list,
  - moves forward to get as close as possible without going past,
  - then goes down to the next lower list.
- Add adds a new key to the lowest list,
  - and a higher list for each time it flips heads.
  - Put has to keep creating new lists on top,
  - as long as the key keeps flipping heads.
- Remove just removes a key from every list it appears in.
- Thanks to recursion, you only have to implement the bottom list plus a SkipList containing half the elements.
- Get, put, and remove are all $O(\log n)$ expected time.
- The amount of space $2n$ (expected), still $O(n)$.

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 LinkedMap.

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list LinkedMap.
  - If key isn’t there, find returns the one before (or null).

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list LinkedMap.
  - If key isn’t there, find returns the one before (or null).
- Map helps solve Daily Jumble.

Summary

- “Formal Phone Directory” Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list LinkedMap.
  - If key isn’t there, find returns the one before (or null).
- 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 LinkedMap.
  - If key isn’t there, find returns the one before (or null).
- 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 LinkedMap.
  - If key isn’t there, find returns the one before (or null).
- 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 LinkedMap.
  - If key isn't there, find returns the one before (or null).
- 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.
- LinkedMap 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 LinkedMap.
  - If key isn't there, find returns the one before (or null).
- 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.
- LinkedMap requires $O(n^2)$ to read in dictionary.
- SkipMap extends LinkedMap.

Summary

- "Formal Phone Directory" Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list LinkedMap.
  - If key isn't there, find returns the one before (or null).
- 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.
- LinkedMap requires $O(n^2)$ to read in dictionary.
- SkipMap extends LinkedMap.
  - Adds a SkipMap containing "every other" key.

Summary

- "Formal Phone Directory" Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list LinkedMap.
  - If key isn't there, find returns the one before (or null).
- 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.
- LinkedMap requires $O(n^2)$ to read in dictionary.
- SkipMap extends LinkedMap.
  - Adds a SkipMap containing "every other" key.
  - Uses coin flips to define "every other".

Summary

- "Formal Phone Directory" Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list LinkedMap.
  - If key isn't there, find returns the one before (or null).
- 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.
- LinkedMap requires $O(n^2)$ to read in dictionary.
- SkipMap extends LinkedMap.
  - Adds a SkipMap containing "every other" key.
  - Uses 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 LinkedMap.
  - If key isn't there, find returns the one before (or null).
- 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.
- LinkedMap requires $O(n^2)$ to read in dictionary.
- SkipMap extends LinkedMap.
  - Adds a SkipMap containing "every other" key.
  - Uses coin flips to define "every other".
- Analysis of SkipMap
  - Only twice as much space as LinkedMap.

Summary

- "Formal Phone Directory" Map interface has put, get, and remove.
  - Can implement as sorted doubly linked list LinkedMap.
  - If key isn't there, find returns the one before (or null).
- 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.
- LinkedMap requires $O(n^2)$ to read in dictionary.
- SkipMap extends LinkedMap.
  - Adds a SkipMap containing "every other" key.
  - Uses coin flips to define "every other".
- Analysis of SkipMap
  - Only twice as much space as LinkedMap.
  - 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 LinkedMap.
  - If key isn't there, find returns the one before (or null).
- 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.
- LinkedMap requires $O(n^2)$ to read in dictionary.
- SkipMap extends LinkedMap.
  - Adds a SkipMap containing "every other" key.
  - Uses coin flips to define "every other".
- Analysis of SkipMap
  - Only twice as much space as LinkedMap.
  - get, put, and remove are all $O(\log n)$ 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 LinkedMap.
  - If key isn't there, find returns the one before (or null).
- 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.
- LinkedMap requires $O(n^2)$ to read in dictionary.
- SkipMap extends LinkedMap.
  - Adds a SkipMap containing "every other" key.
  - Uses coin flips to define "every other".
- Analysis of SkipMap
  - Only twice as much space as LinkedMap.
  - get, put, and remove are all $O(\log n)$ on average.
  - So total time to read in the dictionary is
  - $O(n \log n)$.