EXERCISES

4.1 Write out the code for the earlier `sum` function.

4.2 Write a recursive function to count the number of items in a list.

4.3 Find the maximum number in a list.

4.4 Remember binary search from chapter 1? It's a divide-and-conquer algorithm, too. Can you come up with the base case and recursive case for binary search?

(4.1)

```
int sum(int arr[],int size)
{
if(size==0){
   return 0;
}
else{
   return arr[0]+sum
(arr+1,size-1);
}
}
```

(4.2)

```
int count (int arr[],int size){
if(size==0){
   return 0;
}
else{
   return 1 +count
(arr+1,size-1);
}
}
```

(4.3)

```
int FindMax (int arr
[],int size){
   int max=arr[0];
   int max_index=0;
for(int i=1;i<size;++i){
   if(arr[i]>max){
      max=arr[i];
      max_index=i;
   }
}
   return max;
```

EXERCISES

How long would each of these operations take in Big O notation?

4.5 Printing the value of each element in an array. o(n)

4.6 Doubling the value of each element in an array. o(n)

4.7 Doubling the value of just the first element in an array. o(1)

4.8 Creating a multiplication table with all the elements in the array. So if your array is [2, 3, 7, 8, 10], you first multiply every element by 2, then multiply every element by 3, then by 7, and so on. o(n2)

EXERCISES

It's important for hash functions to consistently return the same output for the same input. If they don't, you won't be able to find your item after you put it in the hash table!

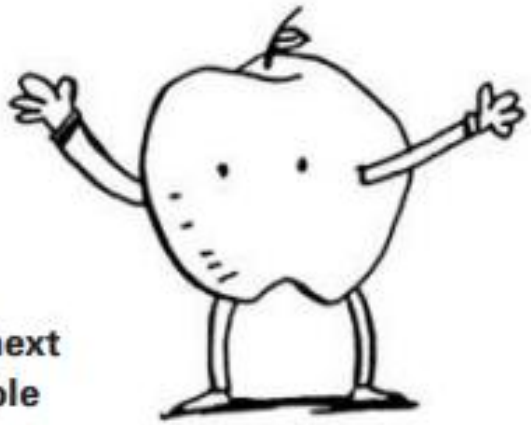
Which of these hash functions are consistent?

5.1 `f(x) = 1` ← Returns "1" for all input

5.2 `f(x) = rand()` ← Returns a random number every time

5.3 `f(x) = next_empty_slot()` ← Returns the index of the next empty slot in the hash table

5.4 `f(x) = len(x)` ← Uses the length of the string as the index



EXERCISES

It's important for hash functions to have a good distribution. They should map items as broadly as possible. The worst case is a hash function that maps all items to the same slot in the hash table.

Suppose you have these four hash functions that work with strings:

A. Return "1" for all input.

B. Use the length of the string as the index.

C. Use the first character of the string as the index. So, all strings starting with *a* are hashed together, and so on.

D. Map every letter to a prime number: a = 2, b = 3, c = 5, d = 7, e = 11, and so on. For a string, the hash function is the sum of all the characters modulo the size of the hash. For example, if your hash size is 10, and the string is "bag", the index is 3 + 2 + 17 % 10 = 22 % 10 = 2.

For each of these examples, which hash functions would provide a good distribution? Assume a hash table size of 10 slots.

5.5 A phonebook where the keys are names and values are phone numbers. The names are as follows: Esther, Ben, Bob, and Dan. C-D

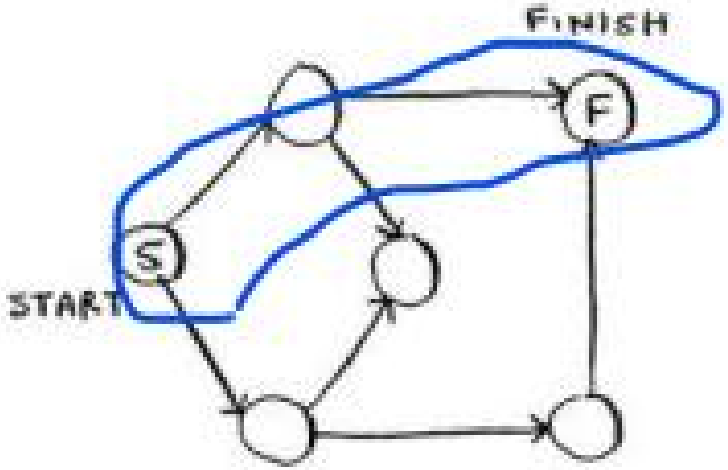
5.6 A mapping from battery size to power. The sizes are A, AA, AAA, and AAAA. B-D

5.7 A mapping from book titles to authors. The titles are *Maus*, *Fun Home*, and *Watchmen*. B-C-D

EXERCISES

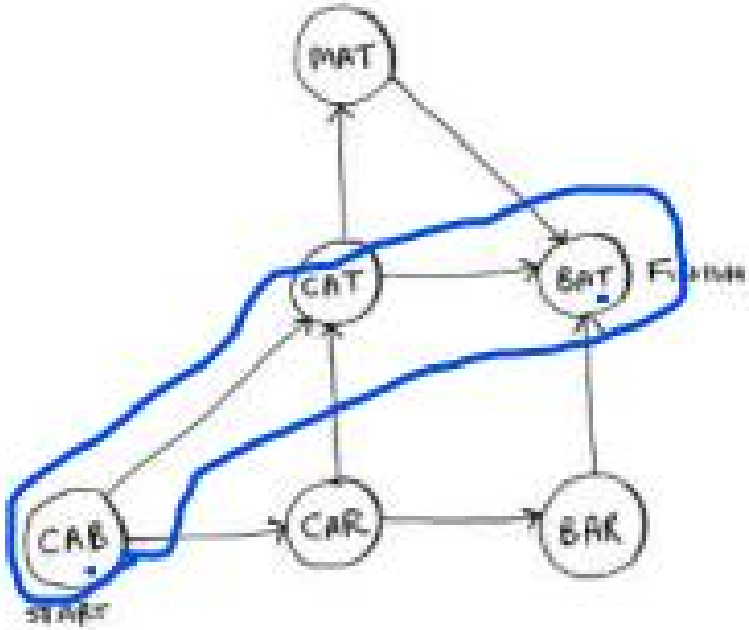
Run the breadth-first search algorithm on each of these graphs to find the solution.

6.1 Find the length of the shortest path from start to finish.



2-path

6.2 Find the length of the shortest path from "cab" to "bat".



2-path

6.3 For these three lists, mark whether each one is valid or invalid.

A. ✗

1. WAKE UP
2. SHOWER
3. EAT BREAKFAST
4. BRUSH TEETH

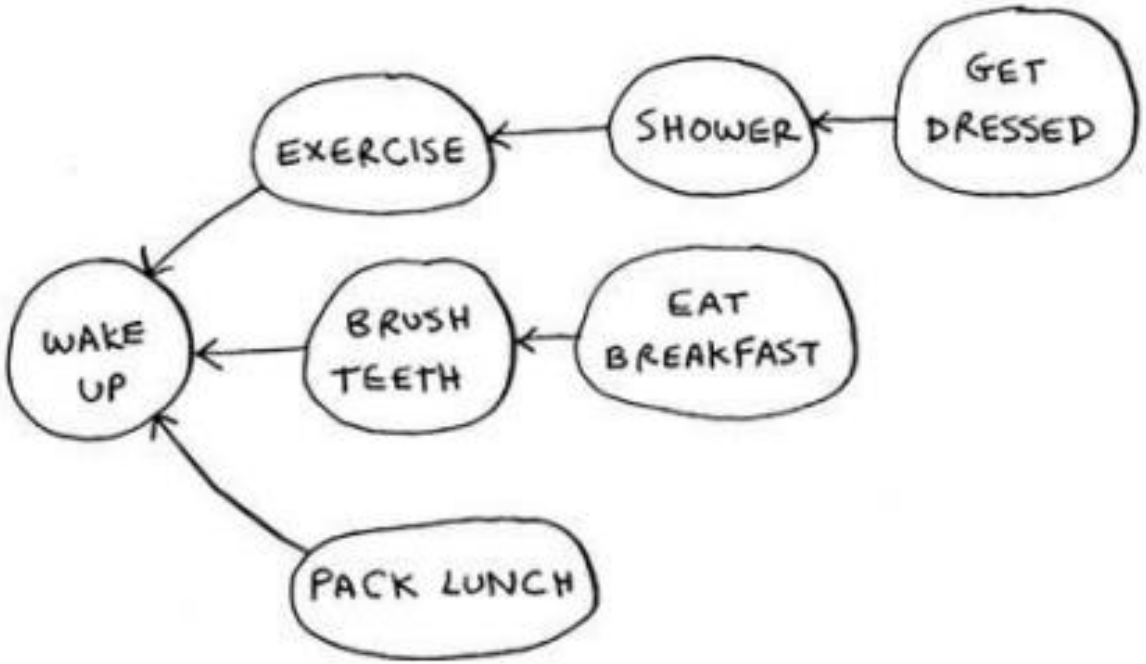
B. ✓

1. WAKE UP
2. BRUSH TEETH
3. EAT BREAKFAST
4. SHOWER

C. ✗

1. SHOWER
2. WAKE UP
3. BRUSH TEETH
4. EAT BREAKFAST

6.4 Here's a larger graph. Make a valid list for this graph.

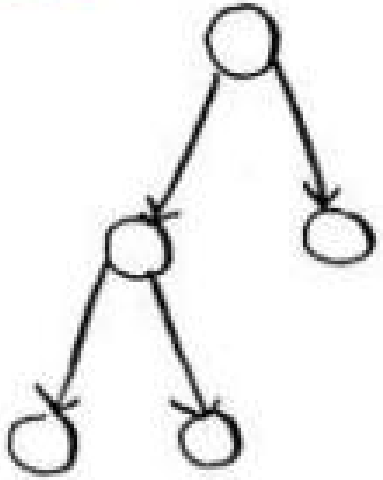


1-wake up
2-Exercise
3-Brush teeth
4-Shower
5-Get dressed
6-Eat breakfast
7-Pack lunch

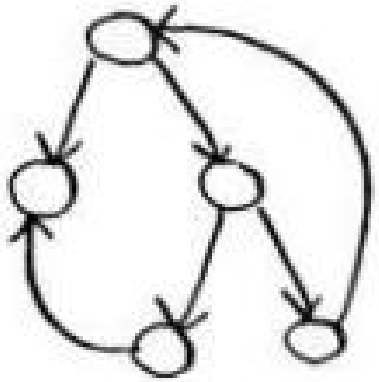
This is called a *tree*. A tree is a special type of graph, where no edges ever point back.

6.5 Which of the following graphs are also trees?

A.



B.



C.