ch.14_(ch. stuff gone)

December 3, 2020

0.1 Ch. 14: Algorithms

0.1.1 Exercsises

1. The section in this chapter called Alice in Wonderland, again! started with the observation that the merge algorithm uses a pattern that can be reused in other situations. Adapt the merge algorithm to write each of these functions, as was suggested there:

(a) Return only those items that are present in both lists.
(b) Return only those items that are present in the first list, but not in the second.
(c) Return only those items that are present in the second list, but not in the first.
(d) Return items that are present in either the first or the second list.
(e) Return items from the first list that are not eliminated by a matching element in the second list. In this case, an item in the second list "knocks out" just one matching item in the first list. This operation is sometimes called bagdiff. For example bagdiff([5,7,11,11,11,12,13], [7,8,11]) would return [5,11,11,12,13]

(a) Return only those items that are present in both lists.

[24]:
```
def return_both_present(xs, ys):
    """ merge sorted lists xs and ys. Return a sorted result """
    result = []
    xi = 0
    yi = 0

    while True:
        if xi >= len(xs):
            return result

        if yi >= len(ys):
            return result

        if xs[xi] < ys[yi]:
            xi += 1
        elif xs[xi] > ys[yi]:
            yi += 1
        else:
            result.append(xs[xi])
```

```
            xi += 1
            yi += 1
```

[25]:
```
print(return_both_present([1, 1, 3, 5, 7, 8, 9, 10], [1, 2, 3, 4, 5, 6, 7, 10]))
```

```
[1, 3, 5, 7, 10]
```

(b) Return only those items that are present in the first list, but not in the second.

[26]:
```
def return_first_list_present_only(xs, ys):
    """ merge sorted lists xs and ys. Return a sorted result """
    result = []
    xi = 0
    yi = 0

    while True:
        if xi >= len(xs):
            return result

        if yi >= len(ys):
            result.append(xs[yi:])
            return result

        if xs[xi] < ys[yi]:
            result.append(xs[xi])
            xi += 1
        elif xs[xi] == ys[yi]:
            xi += 1
        else:
            yi += 1
```

[27]:
```
print(return_first_list_present_only([0, 1, 1, 2, 3, 4, 5, 7], [1, 3, 5, 6, 7,␣
↪10]))
```

```
[0, 2, 4]
```

(c) Return only those items that are present in the second list, but not in the first.

[28]:
```
def return_second_list_present_only(xs, ys):
    """ merge sorted lists xs and ys. Return a sorted result """
    result = []
    xi = 0
    yi = 0

    while True:
        if xi >= len(xs):
            result.extend(ys[yi:])
            return result
```

```
        if yi >= len(ys):

            return result

        if xs[xi] < ys[yi]:
            xi += 1
        elif xs[xi] == ys[yi]:
            yi += 1
        else:
            result.append(ys[yi])
            yi += 1
```

[29]:
```
print(return_second_list_present_only([0, 1, 1, 2, 3, 4, 5, 7], [-1, 1, 3, 5,␣
↪6, 7, 8, 10]))
```

```
[-1, 6, 8, 10]
```

(d) Return items that are present in either the first or the second list.

[30]:
```
def return_unique_items_in_both_lists(xs, ys):
    """ merge sorted lists xs and ys. Return a sorted result """
    result = []
    xi = 0
    yi = 0

    while True:
        if xi >= len(xs):
            result.extend(ys[yi:])
            return result

        if yi >= len(ys):
            result.extend(xs[xi:])
            return result

        if xs[xi] < ys[yi]:
            result.append(xs[xi])
            xi += 1
        elif xs[xi] > ys[yi]:
            result.append(ys[yi])
            yi += 1
        else:
            xi += 1
            yi += 1
```

[31]:
```
print(return_unique_items_in_both_lists([0, 1, 2.5, 3, 4, 4.5, 5, 7], [-1, 1,␣
↪2, 3, 5, 6, 7, 8, 10]))
```

```
[-1, 0, 2, 2.5, 4, 4.5, 6, 8, 10]
```

(e) Return items from the first list that are not eliminated by a matching element in the second list. In this case, an item in the second list "knocks out" just one matching item in the first list. This operation is sometimes called bagdiff. For example bagdiff([5,7,11,11,11,12,13], [7,8,11]) would return [5,11,11,12,13]

[32]:
```
def bagdiff(xs, ys):
    """ merge sorted lists xs and ys. Return a sorted result """
    result = []
    xi = 0
    yi = 0

    while True:
        if xi >= len(xs):
            result.extend(ys[yi:])
            return result

        if yi >= len(ys):
            result.extend(xs[xi:])
            return result

        if xs[xi] < ys[yi]:
            result.append(xs[xi])
            xi += 1
        elif xs[xi] > ys[yi]:
            yi += 1
        else:
            xi += 1
            yi += 1
```

[33]:
```
print(bagdiff([5, 7, 11, 11, 11, 12, 13], [7, 8, 11]))
from unit_tester import test

test(bagdiff([5,7,11,11,11,12,13], [7,8,11]) == [5,11,11,12,13])
```

```
[5, 11, 11, 12, 13]
Test at line 4 ok.
```

#Ex. 2, 3, 4
#skipped

5. Every week a computer scientist buys four lotto tickets. She always chooses the same prime numbers, with the hope that if she ever hits the jackpot, she will be able to go onto TV and Facebook and tell everyone her secret. This will suddenly create widespread public interest in prime numbers, and will be the trigger event that ushers in a new age of enlightenment. She represents her weekly tickets in Python as a list of lists:

```
my_tickets = [ [ 7, 17, 37, 19, 23, 43],
[ 7, 2, 13, 41, 31, 43],
[ 2, 5, 7, 11, 13, 17],
[13, 17, 37, 19, 23, 43] ]
```

Complete these exercises.

(a) Each lotto draw takes six random balls, numbered from 1 to 49. Write a function to return a lotto draw.

(b) Write a function that compares a single ticket and a draw, and returns the number of correct picks on that ticket:
`test(lotto_match([42,4,7,11,1,13], [2,5,7,11,13,17]) == 3)`

(c) Write a function that takes a list of tickets and a draw, and returns a list telling how many picks were correct on each ticket:
`test(lotto_matches([42,4,7,11,1,13], my_tickets) == [1,2,3,1])`

(d) Write a function that takes a list of integers, and returns the number of primes in the list:
`test(primes_in([42, 4, 7, 11, 1, 13]) == 3)`

(e) Write a function to discover whether the computer scientist has missed any prime numbers in her selection of the four tickets. Return a list of all primes that she has missed:

[34]:
```
import random
from unit_tester import test

my_tickets = [[7, 17, 37, 19, 23, 43], [7, 2, 13, 41, 31, 43], [2, 5, 7, 11,␣
↪13, 17],
              [13, 17, 37, 19, 23, 43]]


def lotto_draw():
    lotto_generator = random.Random()
    result = []
    for i in range(6):
        result.append(lotto_generator.uniform(1, 50))
    return result


def lotto_match(lotto1, lotto2):
    count = 0
    for item in lotto1:
        if item in lotto2:
            count += 1
    return count
```

```
def lotto_matches(lotto, mytick):
    result = []
    for i in range(4):
        result.append(lotto_match(lotto, my_tickets[i]))
    return result


def PriNumGenerator(upperlimit):
    Plist = [2]
    for num in range(2, upperlimit + 1):
        isprime = True
        for i in Plist:
            if num % i == 0:
                isprime = False
                break
        if isprime == True:
            Plist.append(num)
    return Plist


def primes_in(l):
    prime_list = PriNumGenerator(50)
    count = 0
    for item in l:
        if item in prime_list:
            count += 1
    return count


def prime_misses(l):
    prime_list = PriNumGenerator(50)
    result = []
    new = []
    for i in range(len(l)):
        new += l[i]
    for item in prime_list:
        if item in new:
            continue
        else:
            result.append(item)
    return result
```

[23]:
```
test(lotto_match([42, 4, 7, 11, 1, 13], [2, 5, 7, 11, 13, 17]) == 3)
test(lotto_matches([42, 4, 7, 11, 1, 13], my_tickets) == [1, 2, 3, 1])
test(primes_in([42, 4, 7, 11, 1, 13]) == 3)
```

```
test(prime_misses(my_tickets) == [3, 29, 47])
```

```
Test at line 1 ok.
Test at line 2 ok.
Test at line 3 ok.
Test at line 4 ok.
```

[ ]: