EECE 230C Introduction to Computation and Programming, Sections 1 and 2
Final Exam

Dec 10, 2018

- The duration of this exam is 3 hours. Keep in mind that you need around 10 minutes at the end of the duration of the exam to submit your answers. It is your responsibility to make sure your files are correctly submitted.
- The exam consists of 5 problems for 210 points
- You can use all the material in the following compressed folders on moodle: `lectures.rar` (lecture slides and source code), `assignments.rar` (programming assignments, solving sessions, and solutions), and `finalMatrial.rar` (`testPrograms.py` and `graph.py`). As soon as you download the compressed folders, moodle will be disconnected.
- At the end of the exam, moodle will reopen for exam submission. If you would like to submit your work before the end of the exam, please talk to the proctors for instructions.
- You are asked to submit a single compressed file containing your `Python` files (ending with `.py` extension). Failure to do so may lead to a failing grade on the exam. It is your responsibility to make sure your files are correctly submitted.
- You are **NOT** allowed to use the **web**. You are not allowed to use **USB's** or files previously stored on your machine.
- If you get caught violating the above rules or if you communicate with a person other than the exam proctors during the exam, you will immediately get zero and you will be referred to the appropriate disciplinary committee.
- Cell phones and any other unauthorized electronic devices are absolutely not allowed in the exam rooms. They should be turned off and put away.
- The problems are of varying difficulty. Below is a rough ordering estimate of the problems in order of increasing difficulty.
  - Level 1 (130 points): Problems 1 (nonefficient), 2, 3, 4 (nonefficient)
  - Level 2 (40 points): Problems 1 (efficient), Problem 4 (efficient)
  - Level 3 (40 points): Problem 5
- Detailed comments are worth partial credit.
- Plan your time wisely. Do not spend too much time on any one problem. Read through all of them first and attack them in the order that allows you to make the most progress.
- Good luck!

Problem 1 (40 points). Find all numbers in a list whose negatives are also in the list

Write a function `find(A)`, which given a list `A` of numbers, returns a list `B` consisting of all elements x of `A` such that $-x$ is also in `A`.

Assume that all the elements in `A` are distinct. The order of the elements in the output list `B` does not matter.

For instance, the elements x of `A =[`**`3,-10,-3`**`,5,6,`**`2`**`,8,`**`10,-2`**`]` such that $-x$ is also in `A` are highlighted in bold.

Faster algorithms are worth more point. Any correct solution is worth 30 point. To get full grade, do it in $O(n \log n)$ time.

Test program/output:

```
print(find([3,-10,-3,5,6,2,8,10,-2]))
print(find([3,-10,-3,5,6,8,2,10,-2,-5,-6,-8]))
print(find([3,-10,5,6,8,2]))
```

```
[3, -10, -3, 2, 10, -2]
[3, -10, -3, 5, 6, 8, 2, 10, -2, -5, -6, -8]
[]
```

Submit your solution in a file called Prob1.py including your name and ID number.

Problem 2 (40 points). Hybrid Merge-Insertion Sort

In this problem, you need the code of the functions `insertionSort` and `mergeSort` we did in class. For convenience, they are included in `testPrograms.py`.

a) **Insertion Sort on slices (20 points).** Given a list `L` of numbers, the function `insertionSort(L)` rearranges `L` so that `L` is sorted (in nondecreasing order). Modify `insertionSort` so that it only sorts `L[low...high]`, where `low` and `high` are given indices. That is, write a function `insertionSortModified (L,low,high)`, which given a list `L` of numbers and two indices `low` and `high`, rearranges `L` by sorting `L[low...high]`. You function is not supposed to modify `L[0...low−1]` or `L[high+1...len(L)−1]`. Assume that `low` and `high` are within range. Note that you are not asked to use the slicing operator.

   Test program/Output:

```
A = [5,6,1,3,2,1,7,9,5,100,15, 2,17,56]
print(A)
insertionSortModified(A, 6, 11)
print(A)
```

```
[5, 6, 1, 3, 2, 1, 7, 9, 5, 100, 15, 2, 17, 56]

[5, 6, 1, 3, 2, 1, 2, 5, 7, 9, 15, 100, 17, 56]
```

b) **Hybrid Merge-Insertion Sort (20 points).** Modify the base case of Merge Sort by performing Insertion Sort when the sublist size becomes less than or equal to 10. Call your function `hybridMergeInsertionSort(A,low,high)`. Use the function `insertionSortModified` in Part (a).

   Test program:

```
A = []
hybridMergeInsertionSort(A,0,len(A)-1)
print(A)
A = [5,6,1,3,2,1,7,9,5,15,100, 2,17,56]
hybridMergeInsertionSort(A,0,len(A)-1)
print(A)
A = A+A
hybridMergeInsertionSort(A,0,len(A)-1)
print(A)
```

   Output:

```
[]
[1, 1, 2, 2, 3, 5, 5, 6, 7, 9, 15, 17, 56, 100]
[1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 5, 5, 5, 5, 6, 6, 7, 7, 9, 9, 15, 15, 17, 17, 56, 56, 100, 100]
```

Submit your solution in a file called Prob2.py including your name and ID number.

Problem 3 (40 points). Circle class

In this problem, you will design an abstract data types for circles based on planar `Point` class from Programming Assignment 10.

The class `Point` is included in `testPrograms.py` for convenience.

A circle C is represented by a `Point center` and a nonnegative number `radius`. Using the class `Point`, design the class `Circle` which defines a circle as an Abstract Data Type (ADT). Include the data attributes `center` and `radius` and the method attributes:

- `__init__`, which takes `center` and `radius` as input arguments, with default values `center=Point(0,0)` and `radius=1`. This method should return the exception "Bad Input!" `center` is not of type `Point`, or `radius` is not of type `int` or `float`, or `radius` is not nonnegative.
- `__str__`, which casts the point into a string of the form `"((center.x,center.y),radius)"`, as shown in the below test program.
- `diameter`, which returns the diameter (i.e., $2 \times$ `radius`)
- `perimeter`, which returns the perimeter (i.e., $2\pi \times$ `radius`)
- `area`, which returns the area (i.e., $\pi \times$ `radius`^{2})
- `contains`, which checks if a given point `pt` is inside the circle. This method should return the exception "Bad Input!" if the type of `pt` is not `Point`
- `intersect`, which checks if the disk associated with the circle (i.e., the circle and its interior) has a nonempty intersection with the disk associated with another given circle `other`. This method should return the exception "Bad Input!" if the type of `other` is not `Circle`.

Test program:

```
C1 = Circle()
C2 = Circle(Point(1,0.5),0.75)
C3 = Circle(Point(10,5),2)
print(C1)
print(C2)
print(C3)
print(C1.diameter())
print(C2.perimeter())
print(C3.area())
print(C1.contains(Point(0.5,0.5)))
print(C1.contains(Point(5,5)))
print(C1.intersect(C2))
print(C1.intersect(C3))
```

Output:

```
((0,0),1)
((1,0.5),0.75)
((10,5),2)
2
4.71238898038469
12.566370614359172
True
False
True
False
```

Submit your solution in a file called Prob3.py including your name and ID number.

Problem 4 (50 points). Rotated sorted list

a) **Find index of min (40 points).** We are given a rotated sorted list `L` of distinct integers, i.e., the elements of L are sorted in increasing order but possibly shifted by i_0 positions modulo n, where $n = len(A)$. That is, there exists an index i_0, where $0 \le i_0 \le n-1$, such that $L[i_0] < L[i_0+1] < \ldots < L[n-1] < L[0] < L[1] < \ldots < L[i_0-1]$ if $i_0 \neq 0$, or $L[0] < L[1] < \ldots < L[n-1]$ if $i_0 = 0$.

Given `L`, we would like to find i_0, i.e., the index of the minimum element of `L`.

Write a function `findIndexOfMin(L)`, which given a rotated sorted list `L` of distinct integers, returns the the index of the minimum element of `L`.

You are not asked to check if `L` is a rotated sorted list; assume that this is the case.

Faster algorithms are worth more point. Any correct solution is worth 20 point. To get full grade, do it in $O(\log n)$ time.

Test program/Output:

`print(findIndexOfMin([40,50, 55, 2, 3, 16]))`	3
`print(findIndexOfMin([50, 55, 2, 3, 16, 40]))`	2
`print(findIndexOfMin([55, 2, 3, 16, 40,50]))`	1
`print(findIndexOfMin([2, 3, 16, 40,50,55]))`	0
`print(findIndexOfMin([16,40,50, 55, 2, 3]))`	4
`print(findIndexOfMin([3,16,40,50, 55, 2]))`	5

b) **Search (10 points).** Write a function `searchRotated(L,x)`, which given a rotated sorted list `L` of distinct integers and an integer `x`, searches for `x` in `L`. If `x` is in `L`, `searchRotated(L,x)` should return the index of `x` in `L`, i.e., `i` such that `L[i]=x`. Otherwise, `searchRotated(L,x)` should return -1.

As in Part (a), you are not asked to check if `L` is a rotated sorted list; assume that this is the case.

Faster algorithms are worth more point. To get a nonzero grade, do it in $O(\log n)$ time.

Test program:

```
for A in ([2, 3, 16, 40,50,55], [40,50, 55, 2, 3, 16], [55, 2, 3, 16, 40,50]):
    print("A=",A)
    for x in (2,4,16,100):
        print("searchRotated(A,"+str(x)+"):",searchRotated(A,x))
print(A)
```

Output:

```
A= [2, 3, 16, 40, 50, 55]
searchRotated(A,2): 0
searchRotated(A,4): -1
searchRotated(A,16): 2
searchRotated(A,100): -1
A= [40, 50, 55, 2, 3, 16]
searchRotated(A,2): 3
searchRotated(A,4): -1
searchRotated(A,16): 5
searchRotated(A,100): -1
A= [55, 2, 3, 16, 40, 50]
searchRotated(A,2): 1
searchRotated(A,4): -1
searchRotated(A,16): 3
```

Submit your solution in a file called Prob4.py including your name and ID number.

Problem 5 (40 points). Hypercube graph

In this problem, you need the file `graph.py` from Programming Assignment 11. In particular, you need the class `UndirectedGraph` from `graph.py`. For convenience, a copy of `graph.py` is included in `finalMatrial.rar`.

Given a integer $n \geq 1$, the n-Hypercube G_n is an undirected graph defined as follows. The nodes of G_n are all length-n binary strings. Two binary strings in G_n are connected by an edge if they differ by exactly one position.

See the below examples.

Write a function `buildHypercube(n)`, which given an integer `n` ≥ 1, returns G_n. Note that G_n has 2^n nodes, each of which is `string`.

Needed modules:

```
from graph import UndirectedGraph
import matplotlib.pyplot as plt
```

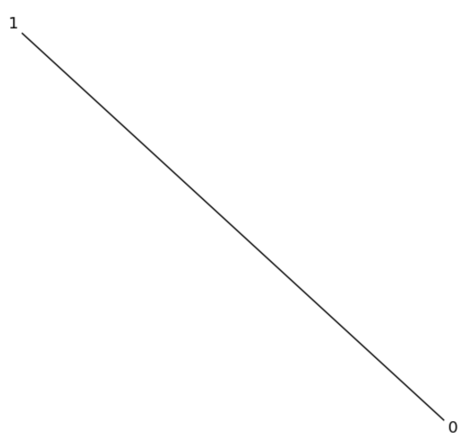
Faster algorithms are worth more point. Any correct solution is worth 25 point. To get full grade, do it in $O(2^n)$ (expected) time.

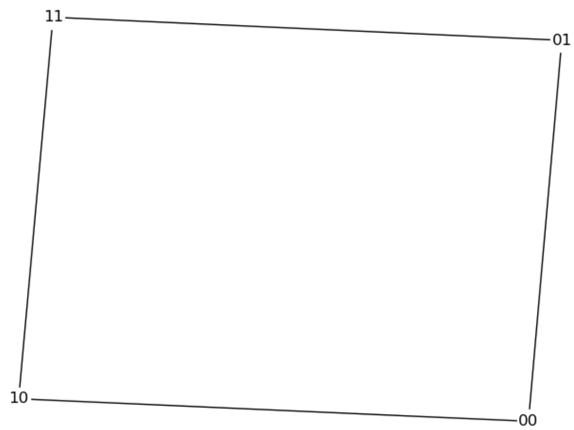
(Hint: Use recursion not only to find all binary strings as we did in class but also to construct the graph G_n from the graph G_{n-1}. See the below examples. How is G_2 as a graph related to G_1? how is G_3 related to G_2?).

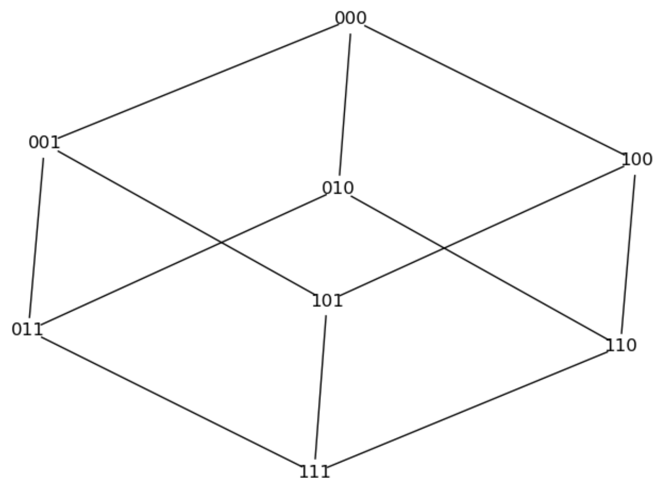
Test program:

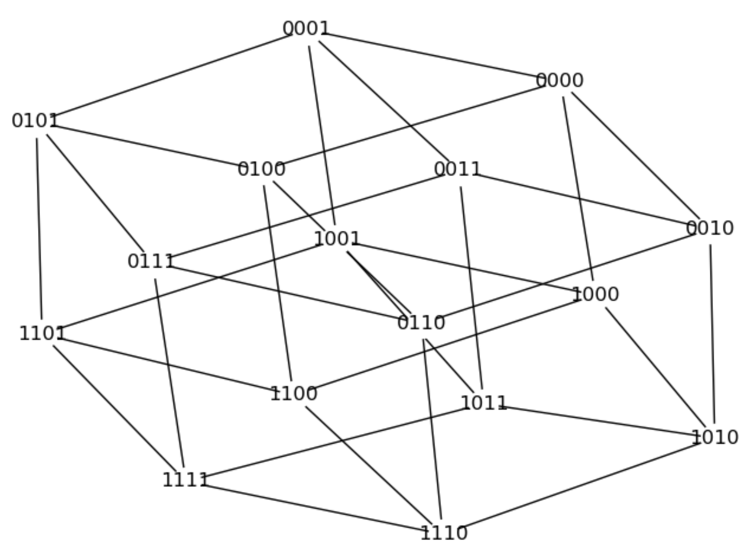
```
for i in range(1,5):
    G = buildHypercube(i)
    plt.figure(i)
    plt.clf()
    G.draw()
```

Output:









Submit your solution in a file called Prob5.py including your name and ID number.