CS1010E Practical Exam II

Grading Guidelines:

- No mark will be given *if your code cannot run*, namely any syntax error or crashes.
  - Your tutors will just grade what you have submitted and they will **not** fix your code.
  - Comment out any part that you do not want.
- Workable code that can produce correct answers will only give you *partial* marks. Only good and efficient code will give you full marks. Marks will be deducted if your code is unnecessarily long, hard-coded, in poor programming style, including irrelevant code or test code.
- You must use the same function names as those in the skeleton given.
- You ***cannot import*** any additional packages or functions other than those imported in the skeleton code attached in this pdf file.
- Your code should be efficient and able to complete *each* example function call in this paper within 10 seconds.
- In all parts, you should return values instead of printing your output.
- You should remove all your test cases before submitting to Examplify. If you submit more code than required, it may have a possibility of penalty because you code is "unnecessarily long".
- You should save an **exact** copy of your submission in order to submit in Coursemology again after the PE. Any difference between exemplify and coursemology submissions will be severely penalized.

Part 1 Super Fibonacci Sequence (30 marks)

A *super-Fibonacci sequence* (**SFS**) is a list of integers with the property that, from the third term onwards, every term is the sum of all the previous terms. For example,

1, 2, 3, 6, 12, …

The number 6 is the sum of all terms 1, 2 and 3, and 12 is the sum of all the terms 1, 2, 3 and 6.

We assume the first term is always 1. However, we have the freedom to change the second term. By setting different second term, we got different sequence

1, **5**, 6, 12, 24, 48, 96, 192, 384, 768,…

1, **6**, 7, 14, 28, 56, 112, 224, 448, 896,…

Task 1 Recursive SFS (10 marks)

Write a function `superFibonacciSeqR(t2,n)` to return the list of the first `n` terms of an SFS with its second term as `t2` with ***proper*** recursions for n > 0. No iteration is allowed in this part. Sample run:

```
>>> superFibonacciSeqR(10,10)
[1, 10, 11, 22, 44, 88, 176, 352, 704, 1408]
>>> superFibonacciSeqR(20,7)
[1, 20, 21, 42, 84, 168, 336]
>>> sfs1 =  superFibonacciSeqR(11,994)
>>> sfs1[-1]
251134829809281403347287120873437924350329252743484439244628997274301027607406903709343370034928716748655001465051518787153237176334136103968388536906997846967216432222442913720806436056149323637764551144212026757427701748454658614667942436236181162060262417445778332054541324179358384066497007845376
>>> sfs1[-1]%10000
5376
 >>> sfs1[-2]%10000
2688
```

The last example of the list `sfs1` shows that:

1. You do not have to worry about the recursion depth that is greater than 994.
2. Usually the number and sequence are long, you can and you should verify by the last few digits of the last few numbers only. *If you print out the whole list or whole number, your Python IDLE console may be frozen*, or even your computer may.

Task 2 Iterative SFS (10 marks)

You are going to generate an SFS like Task 1-1, but in a different way. **The second input argument of the function is not the number of terms of the sequence anymore**. Write an iterative function `superFibonacciSeqI(t2, upperbound)` to return a list of all SFS numbers that are smaller than or equal to **`upperbound`** with `t2` as the second term for `upperbound >= 1`. No recursion is allowed in this task and you must use *proper* for- or while-loop.

```
>>> superFibonacciSeqI(4,100)
[1, 4, 5, 10, 20, 40, 80]
>>> superFibonacciSeqI(4,160)
[1, 4, 5, 10, 20, 40, 80, 160]
```

The following sample outputs show that you may not want to print out everything to verify the correctness of your output when the sequences and numbers are long.

```
>>> longSFS = superFibonacciSeqI(20,10**4321)
>>> len(longSFS)
14352
>>> from math import log
>>> int(log(longSFS[-1])/log(10))
4320
>>> longSFS[-1] % (10**10)
4087703552
```

Your function should be efficient and should complete each of the above function calls of `superFibonacciSeqI()` in less than 5 seconds. You can use `log` for your own testing but not for submission.

Task 3 Search for the Smallest Second Term in SFS (10 marks)

With different second term, the SFS will contain different set of numbers. Given a number n, we want to find the smallest second term that will generate an SFS that contains n. The above example in Task 1 showed that if the second term is 10, the number n = 44 will be in the SFS. However, the number 44 is ***NOT*** in the SFS with 20 as the second term.

Write a function `smallestSecondTermSFScontains(n)` to search for the smallest second term that will generate an SFS that contains the number `n` for `n>1`. Example:

```
>>> smallestSecondTermSFScontains(2016)
62
>>> smallestSecondTermSFScontains(9876)
2468
>>> smallestSecondTermSFScontains(23592960)
44
>>> smallestSecondTermSFScontains(2651336998912)
9876
```

To verify the above answers:

```
>>> superFibonacciSeqI(2468,10**7)
[1, 2468, 2469, 4938, 9876, 19752, 39504, 79008, 158016, 316032, 632064,
1264128, 2528256, 5056512]
```

Part 2 Pizza Delivery 2.0 (45 marks)

Yes, you did not read it wrongly, this task is similar to Assignment 6 and you can fully reuse your code there. However, you will build some new extra features on this part and we will only reward you for the new features excluding what you have done for the assignment. If you are not familiar with Assignment 6, we have attached that assignment at the end.

For the original assignment, pizza shops only have coordinates. We would like to enrich the pizza shop property here to include the following attributes:

- `pos`: the position/coordinates as in Assignment 6
- `name`: The name of the pizza shop in a string
- `radius`: The drone can only deliver pizza for a distance less than or equal to that radius
- `starthour, endhour`: The opening and closing hour of the pizza shop. To simply the question, we only record hours in 24 hour format. E.g. if `starthour` = 10 and `endhour` = 20, it means that the pizza shop will open at 10:00 am in the morning, and close at 8:00pm every day. To be precise, it will close at 8:00:00pm. (The last delivery drone flies out at 7:59:59pm.) The two parameters will be 0 and 24 if a pizza shop opens 24 hours.

Task 4 Class PizzaShop (10 marks)

You are given the code of the basic definition of class `PizzaShop` with initial attributes. Add a new method `distanceSquareTo(i,j)` such that it will return the **square** of the distance between itself and the coordinate `(i,j)` as in Assignment 6 in integer.

Task 5 Pizza Delivery Map 2.0 (35 marks)

Write a function `PDMap(r,c,allPS,currentHour)` to compute the pizza shop delivery map similar to Assignment 6 with the same return value format, namely, a 2D array of symbols, with the following parameters:

- `r,c` : Same as Assignment 6, the number of row and column of the map
- `allPS`: a list of instances of class `PizzaShop`
- `currentHour`: The current time in hour defined in the same way as the `starthour` and `endhour` of class `PizzaShop`.

And some modifications from the original `PDMap` from Assignment 6:

- A pizza shop will NOT deliver to any house that has a distance more than the `radius`. More preciously, it WILL still deliver if the distance is EXACTLY the same as the radius.
- A pizza shop will only deliver when it is open. If the `currentHour` is not within its opening hours, the map should NOT contain that pizza shop.
- On the map, the nearest pizza shop is shown by the first character of its name, instead of numbers. Same as Assignment 6, if a house has more than two nearest pizza shops, then an '`X`' is shown instead. We can assume that no two pizza shops have the same first character in their names.
- And for any house that is not reachable by any pizza shop, just put a period '`.`' in the map.

For example, if we initialize the list of pizza shop as follows:

```
allPSsmall=[PizzaShop([3,3],'Ace Pizza',3,8,14),PizzaShop([6,6],'Bizza',4,12,22)]
```

With `r` = 10 and `c` = 12, the three maps for three different values of `currentHour` as 10, 12 and 16 will be:

The red color of the 'X' is only to make it easier for you to read. You do not have to produce the color. More maps of larger sizes are available at the end of this pdf.

Part 3 Fun with Digits Puzzle (25 marks)

This question is similar to one of the training questions to solve a puzzle. To start, the following equation is wrong.

```
123456789 = 100
```

However, if we are allowed to put any numbers of '+' or '−' signs between the digits on the left side of the equation, you may have some correct equations. In fact, there are altogether 11 ways of setting the above equation right:

```
1+2+3-4+5+6+78+9=100
1+2+34-5+67-8+9 =100
1+23-4+5+6+78-9 =100
1+23-4+56+7+8+9 =100
12+3+4+5-6-7+89 =100
12+3-4+5+67+8+9 =100
12-3-4+5-6+7+89 =100
123+4-5+67-89   =100
123+45-67+8-9   =100
123-4-5-6-7+8-9 =100
123-45-67+89    =100
```

The above equation is just one example, and its left side can change from '`123456789`' to any set of digits, and its right side can be any number. E.g. if the left side digits are '`111111`' and the right number is 121, we have:

```
11+111-1=121
11-1+111=121
111+11-1=121
111-1+11=121
```

Moreover, you can also extend the set of operators from the two signs '+' and '−' to more variety of operators, e.g. '`+-*%`' with the initial equation '`12345=30`', we have two correct equations:

```
1+2*3*4+5=30
1%2+34-5 =30
```

Task 6 (25 marks)

Write a function `sumTo(leftdigits,ops,n)` to return a **list** of all correct equations in **strings**, with

- `leftdigits`: A string containing the digits on the left. We will not include the digit zero.
- `ops`: A string of all possible operators. We will limit the possible operators to `+`, `-`, `*` and `%` only.
- `n`: The number on the right of the equation represented as an integer

On the left side, you can put at most one operator (or none) between any pair of digits. The minus sign will be treated as subtraction, not the unary negative operator.

Your output should be in ascending lexicographic order, and you are allowed to use the built-in sorting functions.

Here is some sample output:

```
>>> sumTo('199','+',100)
['1+99=100']
>>> pprint(sumTo('123456789','+-',100))
['1+2+3-4+5+6+78+9=100',
 '1+2+34-5+67-8+9=100',
 '1+23-4+5+6+78-9=100',
 '1+23-4+56+7+8+9=100',
 '12+3+4+5-6-7+89=100',
 '12+3-4+5+67+8+9=100',
 '12-3-4+5-6+7+89=100',
 '123+4-5+67-89=100',
 '123+45-67+8-9=100',
 '123-4-5-6-7+8-9=100',
 '123-45-67+89=100']
>>> pprint(sumTo('111111','+-*',100))
['1*111-11=100', '111-1*11=100', '111-11*1=100', '111*1-11=100']
```

Hints

This problem seems to be difficult, but here are some hints that maybe useful to you.

First, the built-in function `eval()` should be of great help. The simple usage of this function is to take in a string and evaluate the value for you. For example:

```
>>> x = 2
>>> eval('x*3')
6
```

Second, you may write the following functions to help the task.

`product(str1,n)`: To produce a list of all *Cartesian* products of `str1` with length `n`.

```
>>> product('ab',3)
['aaa', 'aab', 'aba', 'abb', 'baa', 'bab', 'bba', 'bbb']
>>> product('xyz',2)
['xx', 'xy', 'xz', 'yx', 'yy', 'yz', 'zx', 'zy', 'zz']
```

More hints: The answer for `product(str1,n)` is to add each of the character in `str1` to `product(str1,n-1)`. For example, `product('ab',3)` is adding `'a'` to all items in `product('ab',2)` and adding `'b'` to all items in `product('ab',2')`.

There is a more complicated version of this function `product()` in the package `itertools`, but you are not allowed to import that (or other functions).

`interleave(str1,str2)`: To generate a string that is interleaving the two. E.g.

```
>>> interlace('abc','123')
'a1b2c3'
```

Template and Test Cases (Copy and paste all the followings into ONE .py file.)

```
# You should remove all the test cases before submission to examplify
#################################################################
#Part 1
def superFibonacciSeqR(t2, n):
    return []

print(superFibonacciSeqR(4,10))
#ans: [1, 4, 5, 10, 20, 40, 80, 160, 320, 640]

sfs1 = superFibonacciSeqR(11,20)
print(sfs1[-3::])
#ans: [393216, 786432, 1572864]

def superFibonacciSeqI(t2, upperbound):
    return []

print(superFibonacciSeqI(4,10))
#ans: [1, 4, 5, 10]

#longone = superFibonacciSeqI(20,10**9876)
#print(longone[-1]%10000000)
#ans: 6179584
#len(longone) should be 32805

def smallestSecondTermSFScontains(n):
    return 1
print(smallestSecondTermSFScontains(2016))
print(smallestSecondTermSFScontains(9876))
print(smallestSecondTermSFScontains(2651336998912))
print(smallestSecondTermSFScontains(23592960))
#ans: 62, 2468, 9876, 44

#################################################################
#Part 2
class PizzaShop:
    def __init__(self,pos,name,radius,starthour,endhour):
        self.pos = pos
        self.name = name
        self.radius = radius
        self.starthour = starthour
        self.endhour = endhour
def createZeroMatrix(n,m):
    return [[0 for i in range(m)] for j in range(n)]
def mTightPrint(m):
    for i in range(len(m)):
        line = ''
        for j in range(len(m[0])):
            line += str(m[i][j])
        print(line)

def PDMap(r,c,allPS,currentHour):
    return [[]]
```

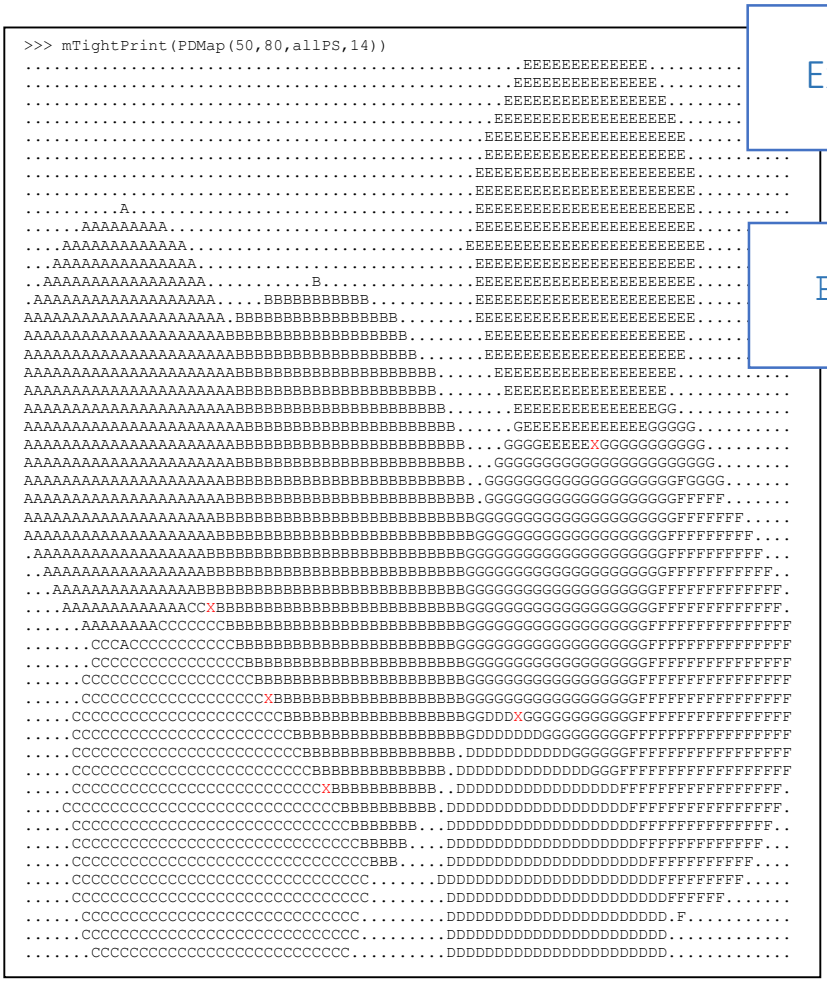
```
allPS2 = []
allPS2.append(PizzaShop([20,10],'Amazing Pizza',10,8,22))
allPS2.append(PizzaShop([29,30],'Beloved Pizza',10,8,22))
#mTightPrint(PDMap(50,60,allPS2,10))
allPS3 = []
allPS3.append(PizzaShop([20,30],'Amazing Pizza',10,8,22))
allPS3.append(PizzaShop([38,30],'Beloved Pizza',10,8,22))
#mTightPrint(PDMap(50,60,allPS3,10))
allPS = []
allPS.append(PizzaShop([20,10],'Amazing Pizza',12,8,22))
allPS.append(PizzaShop([29,30],'Beloved Pizza',17,8,22))
allPS.append(PizzaShop([41,20],'Cute Pizza',16,14,22))
allPS.append(PizzaShop([45,55],'Delicious Pizza',12,8,22))
allPS.append(PizzaShop([10,58],'Elegant Pizza',12,8,22))
allPS.append(PizzaShop([35,68],'Fancinating Pizza',12,14,22))
allPS.append(PizzaShop([32,60],'Good Pizza',15,8,22))
allPS.append(PizzaShop([30,46],'Ideal Pizza',9,8,14))
#mTightPrint(PDMap(50,80,allPS,10))
#mTightPrint(PDMap(50,80,allPS,14))
allPSsmall = []
allPSsmall.append(PizzaShop([3,3],'Amazing Pizza',3,8,14))
allPSsmall.append(PizzaShop([6,6],'Bear Pizza',4,12,22))
#mTightPrint(PDMap(10,12,allPSsmall,10))
#mTightPrint(PDMap(10,12,allPSsmall,16))
#mTightPrint(PDMap(10,12,allPSsmall,12))

#####################################################################
# Part 3
def sumTo(str1,ops,n):
    return[]

from pprint import *
pprint(sumTo('199','+',100))
#Answer above should have 1 equation
pprint(sumTo('123456789','+-',100))
#Answer above should have 11 equations
#pprint(sumTo('111111','+-*',100))
#Answer above should have 4 equations
#pprint(sumTo('12345','+-*%',30))
#Answer above should have 2 equations
#pprint(sumTo('11111','+-',100))
#Answer above should have 1 equation
```

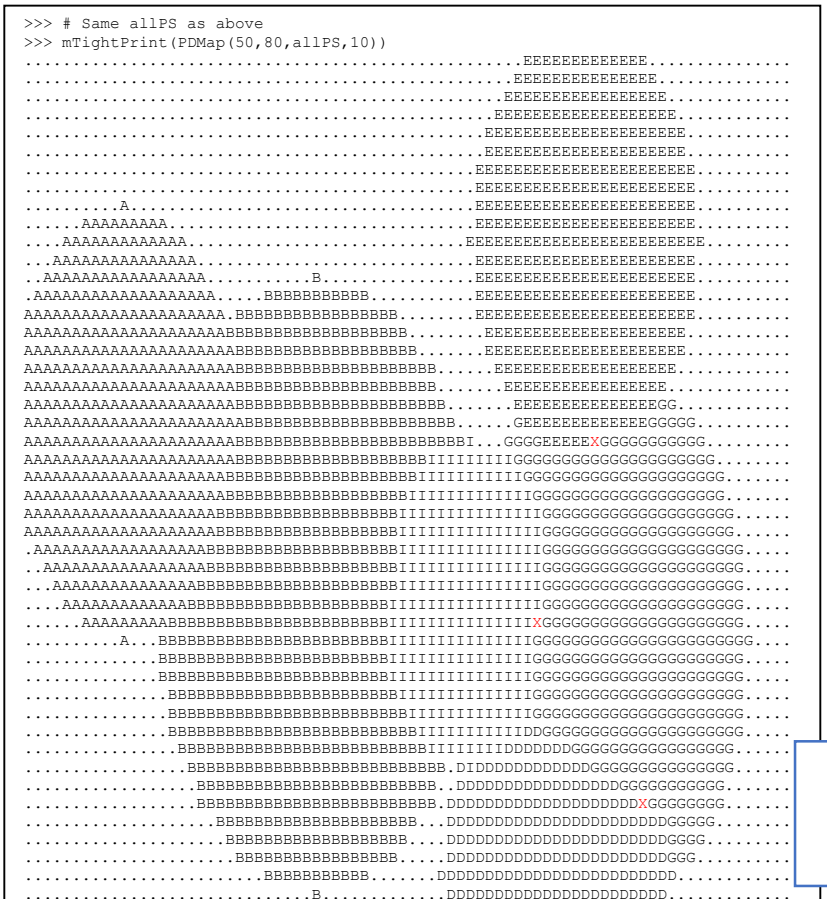
Example cases for Pizza Delivery

```
>>> mTightPrint(PDMap(50,80,allPS,14))
```



PDMap(50,80,allPS,14)

```
>>> # Same allPS as above
>>> mTightPrint(PDMap(50,80,allPS,10))
```



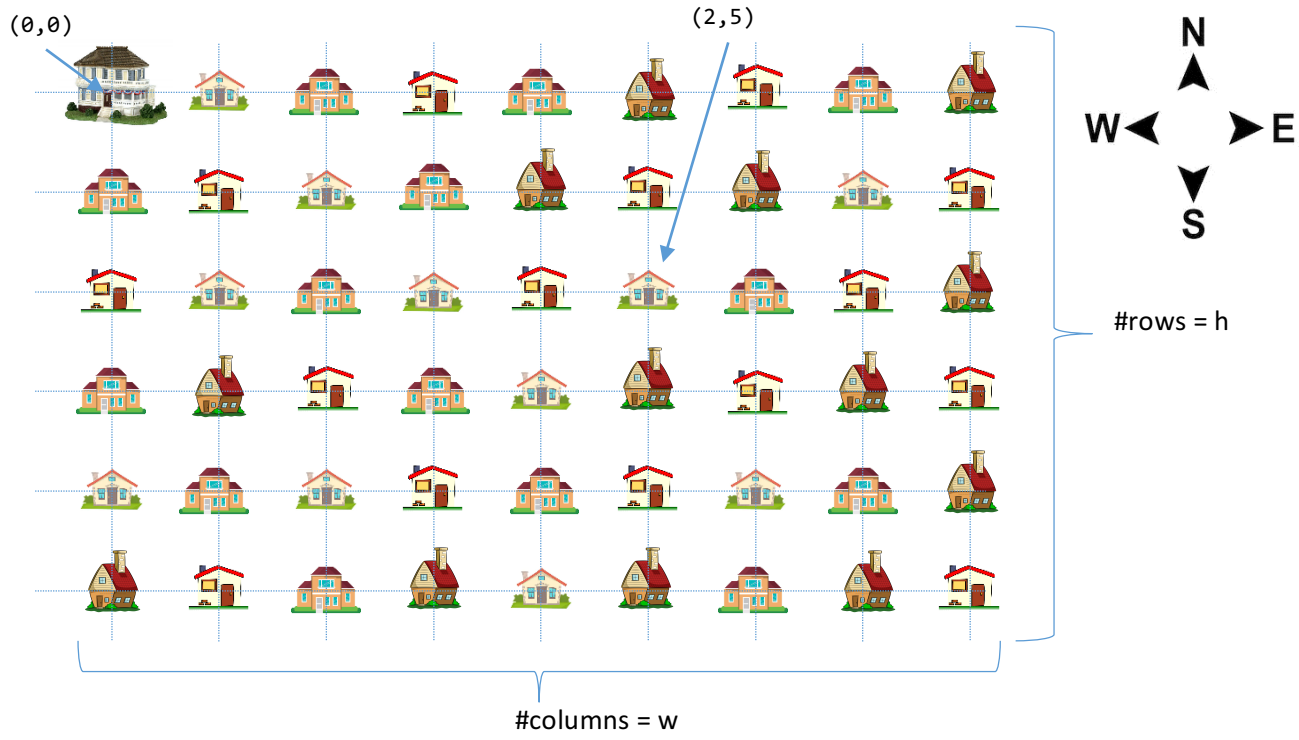
PDMap(50,80,allPS,10)

Appendices (Past Assignment 6 for the reference of Part 2)

This is for your reference only.

Assignment 6: Lightning Pizza Delivery

In a town called ***Regulaville***, and all the people living there have very strange habits. All the buildings have centers on a well-structured rectangular grid. The mayor of the town is at the most north-west corner of the town, and his house coordinates are `(0,0)`. And each building in town has coordinates `(i,j)` that indicate that his house is `i` km *south* and `j` km *east* from the mayor's house for i and j are integers such that $0 \leq i < h$ and $0 \leq j < w$ for some integers h and w.



A new brand of pizza came to town! They set up **n** stores in some of the buildings in town for some **n** ≤ 10. We store the coordinates of the pizza stores in a list. For example, a list of

```
[[10,20],[30,20],[40,50]]
```

represents three stores of pizza with Store 0 at `(10,20)`, Store 1 at `(30,20)` and Store 2 at `(40,50)`*. (Somehow the big boss of the pizza stores knows programming and he starts counting by 0 also.) <u>There is no two pizza stores at the same location</u>. All the people in Regulaville only eat pizza at their own home and call for delivery. And all the stores will deliver pizza by flying drones that will fly directly from the stores to the destination.

In order to minimize the time and power used by the drones, the mayor ordered that every home must only order pizzas from the <u>nearest</u> store, unless there are more than one store with equal minimal distance. For example, for the three pizza stores mentioned about, the house at coordinates `(40,20)` will be closest to Store 1 with the nearest distance 10 km. Compared to Stores 0 and 2 with the same distance $\sqrt{30} > 10$ km.

*The coordinates are a bit confusing because for a location `[10,20]` in ***<u>our</u>*** assignment, it means going south (vertical direction) for 10km and 20km east (horizontal direction), in which, it's the opposite way we think in real life such that `(x,y)` in which `x` is the horizontal direction and `y` is the vertical one.

Task

Write a function PDMap(h,w,pizzaLoc) to compute a map for ALL houses in Regulaville to show the closest pizza store number to each house. As mentioned above, each house has the coordinates (i,j) such that 0 ≤ i < h and 0 ≤ j < w and the list pizzaLoc is a list of pizza store coordinates. You can assume the number of pizza stores is less than or equal to 10 and their numbers are ranging from 0 to 9. Here is a sample usage of the function PDMap().

```
>>> pizzaMap = PDMap(7,8,[[1,3],[4,7],[7,2]])
>>> pprint(pizzaMap)
[[0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 1, 1],
 [0, 0, 0, 0, 0, 1, 1, 1],
 [2, 2, 2, 0, 1, 1, 1, 1],
 [2, 2, 2, 2, 2, 1, 1, 1],
 [2, 2, 2, 2, 2, 1, 1, 1]]
```

```
>>> mTightPrint(pizzaMap)
00000001
00000001
00000011
00000011
22201111
22222111
22222111
```

For example, it shows that the home at (3,6) is closest to the pizza Store 1.

Sometime, there is a chance that some of the house has more than one pizza store that are closest to it with the same minimal distance. If it happens, mark that house with an 'X'. Since we are all using integer arithmetic, the distance computation must be exact. (Wait, isn't that a "sqrt()" in the distance computation?)

```
>>> mTightPrint(PDMap(10,10,[[2,3],[4,9],[7,2]]))
0000000X11
0000000111
0000000111
000000X111
X000001111
22222X1111
2222221111
2222222111
2222222111
2222222X11
```

In this part, you should submit at least the function PDMap(), or together with any functions you created to support your function.