C200 Programming Assignment №5

Professor M.M. Dalkilic
Computer Science
School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

October 20, 2023

Introduction

Due Date: 10:59 PM, Friday, October 27, 2023 Please note that both Autograder submission `https://c200.luddy.indiana.edu/` and pushing to GitHub **must be done before the deadline. We do not accept late submissions**. As mentioned in the Inscribe post, we will be checking for cheating. If this results in a grade change, we are compelled by the University to send a formal notification to the Dean.

Instructions

1. Make sure that you are **following the instructions** in the PDF, especially the format of output returned by the functions. For example, if a function is expected to return a numerical value, then make sure that a numerical value is returned (not a list or a dictionary). Similarly, if a list is expected to be returned then return a list (not a tuple, set or dictionary).

2. Test **debug** the code well (syntax, logical and implementation errors), before submitting to the Autograder. These errors can be easily fixed by running the code in VSC and watching for unexpected behavior such as, program failing with syntax error or not returning correct output.

3. Make sure that the **code does not have infinite loop (that never exits) or an endless recursion (that never completes)** before submitting to the Autograder. You can easily check for this by running in VSC and watching for program output, if it terminates timely or not.

4. Given that you already tried points 1-3, if you see that Autograder does not do anything (after you press 'submit') and waited for a while (30 seconds to 50 seconds), try refreshing the page or using a different browser.

5. Once you are done testing your code, comment out the tests i.e. the code under the __name__ == "__main__" section.

Problem 1: Recursion on Numbers

In this problem, you'll implement the following recursive functions. Look back at Lab 6 to review similar recursion problems.

$$p(0) = 10000 \tag{1}$$
$$p(n) = p(n-1) + 0.02p(n-1) \tag{2}$$

$$c(1) = 9 \tag{3}$$
$$c(n) = 9c(n-1) + 10^{n-1} - c(n-1) \tag{4}$$

$$d(0) = 1 \tag{5}$$
$$d(n) = 3d(n-1) + 1 \tag{6}$$

$$f(0) = 1 \tag{7}$$
$$f(1) = 1 \tag{8}$$
$$f(n) = f(n-1) + f(n-2) \tag{9}$$

$$e(1) = 12 \tag{10}$$
$$e(n) = -5e(n-1) \tag{11}$$

$$M(0,i) = 1 \; when \; c = 0 \tag{12}$$
$$M(c,i) = \begin{cases} 0 & c < 0 \; \vee \; f(i) > 10 \\ M(c - f(c), i+1) + M(c, i+1) & otherwise \end{cases}$$
$$\binom{n}{k} = \begin{cases} 1 & k = 0 \; \vee \; n = k \\ \binom{n-1}{k-1} + \binom{n-1}{k} & otherwise \end{cases} \tag{13}$$

Programming Problem 1: Recursion on Numbers

- Complete each of the functions.
- Observe that M requires f
- Note that $\binom{n}{k}$ indicates choosing k from n and in the starter code it is denoted by the function c_2(n, k).

Problem 2: Maximal Value

In this problem, you're given a list of numbers. The function msi takes a list of numbers and returns a list of three values (in this same order) namely, the interval start, interval stop and the value of the greatest sum. For example if the list is: x = [7, -9, 5, 10, -9, 6, 9, 3, 3, 9] then the greatest interval is from x[2:10] giving 36:

$$sum(x[2,10]) \quad = \quad 5+10-9+6+9+3+3+9 = 36 \tag{14}$$

The function should return a list of values i.e., [2, 10, 36]

```
1 x =  [7, -9, 5, 10, -9, 6, 9, 3, 3, 9]
2 print(msi(x))
```

produces

```
1 [2,10,36]
```

For this problem you are allowed to use sum(lst) which returns the sum of a non-empty list of numbers.

Programming Problem 2: Maximal Value

- Complete the function msi() function
- You are allowed to use the sum function
- You might find the following from the lecture useful:

```
1 >>> x = [1,2,3,-2]
2 >>> sum(x[1:3])
3 5
```

Problem 3: Move the Cheese

In this problem, you must read the constraints carefully. You are given a list of two types of cheese, one called zero (denoted by 0) and the other one (denoted by 1). The actual names are so long and complicated, it'd make the problem even more difficult. You must move all the zero cheeses next to each other in the front of the list (because zero is more popular than one) leaving all the one cheeses next to each other. I will supply the loop (you need only one) and you'll be given two variables. You

- cannot build any extra lists

- cannot use any list operations other than single subscripting
- cannot use any more loops

Here is the starting code:

```
data = [[1,0],[0,1,0,1,0,1,0],[1,1,1,1,0,0,0,0]]
def move(x):
    lo,hi = 0,len(x)-1
    while lo < hi:
     pass  #you can only add code here -- see the constraints above
     return x

for d in data:
     print(f"{d} => {move(d)}")
```

has output:

```
[1, 0] => [0, 1]
[0, 1, 0, 1, 0, 1, 0] => [0, 0, 0, 0, 1, 1, 1]
[1, 1, 1, 1, 0, 0, 0, 0] => [0, 0, 0, 0, 1, 1, 1, 1]
```

Programming Problem 3: Move the Cheese

- Complete the move() function.
- Follow the constraints.
- Think about small problems first–don't over think it :).

Problem 4: Entropy

In this problem, we will be calculating what's called *Entropy*. Entropy is used in ML, AI, vision, finance, *etc.* Entropy is a single number that describes how random the data is. If it's 0, then the data isn't random. If there are n objects and each one is equally probable, then the entropy is maximal: $\log(n)$. How might this be useful? Decisions are "better" the less entropy there is.

For this problem, we are only working with finite probabilities. We can think of probabilities as a list of numbers $p_0, p_1, \ldots, p_n$ such that

$$\begin{aligned} p_i &\geq 0, \quad i = 0, 1, \ldots, n && (15)\\ 1 &= p_0 + p_1 + p_2 + \cdots + p_n && (16)\\ &= \sum_{i=0}^{n} p_i && (17) \end{aligned}$$

We can make a list (we'll assume the items are of the same type and immutable) into a probability. For example, consider a list `x = ["a","b","a","c","c","a"]`.

1. gather the items uniquely (*hint: dictionary*)
2. count each time the item occurs
3. create a new list of the `count/len(dictionary)`

The list of probabilities would be y = [3/6,1/6,2/6] (a's probability is 3/6, b's probability is 1/6 and c's probability is 2/6). Observe that if you add up these numbers, they will sum to one.

We still need to show you *how* to calculate entropy:

$$\begin{aligned} entropy &= -(p_0 \log_2(p_0) + p_1 \log_2(p_1) + \cdots + p_n \log_2(p_n)) && (18)\\ entropy(y) &= -(\frac{3}{6}\log_2(3/6) + \frac{1}{6}\log(1/6) + \frac{2}{6}\log_2(2/6)) && (19)\\ &= -(.5(-1.0) + 0.17(-2.58) + .33(-1.58)) && (20)\\ &= 1.46 && (21) \end{aligned}$$

Because of continuity arguments we treat $\log_2(0) = 0$. Python's math module will correctly state this math error, so you'll **have to simply add 0 if the probability is 0**.

Deliverables Problem 4

- Take a non-empty list of objects and calculate the entropy.
- You **cannot** use the `count` function.
- Complete the functions for this problem.
- Hint: The makeProbability function is based on points (1-3). Use the makeProbability function in your entropy() function.
- Round the output of entropy to two decimal places.

Problem 5: Run of 1s

For a financial instrument like stock, we want to buy and sell at the lowest and highest values, respectively. By observing trends, better decisions about buying and selling can be made. A "trend" is when when a sequence of values is **monotonic**. A sequence of values $v_0, v_1, v_2, \ldots, v_n$ is increasingly monotonic if

$$v_0 \leq v_1 \leq \cdots \leq v_n$$

The increasing monotonicity allows for selling at a higher price. Conversely, there is decreasing monotonicity. In this problem we'll be writing code to look for increasing monotonicity, but simplifying the problem by using only zero or one. In this problem, you'll take a list of 0s and 1s as an input. Your program will output the *longest* sequential run of 1s (no 0s encountered).

Output function.py

```
L([1, 0, 1, 0, 1, 0])
1

L([0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0])
2

L([1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1])
4
```

Deliverables Problem 5

- Complete the function L().

Problem 6: Nine of Cattails

It's known that if you add the digits of any number, and that it eventually equals nine, that the original number is divisible by nine. Here is an example:

$$39789 \quad \rightarrow \quad 3+9+7+8+9 = 36 \rightarrow 3+6 = 9$$

It doesn't matter the order, *viz.,*

$$3+((9+7)+8)+9$$
$$3+((16)+8)+9$$
$$3+(7+8)+9$$
$$3+15+9$$
$$18+9$$
$$27 \rightarrow 9$$

We are left with only one digit which is 9 and hence the number 39789 is divisible by 9 (Infact, 9 * 4421 = 39789).

So we can see that the number 39789 is indeed divisible by 9 by following our method. Write a program that determines if a number is divisible by nine by using this method. You *cannot* simply divide by 9 and return True! You must continually add the numbers until a single digit is left; that digit will either be 9 or something else.

Caution: Remember that 'sum' is actually a built-in function in Python so we suggest that you store the results of addition in a variable named other than 'sum' to prevent errors or failures during testing.

sum = a + b + c (this is not a good practise because 'sum' is a reserved keyword in Python), rather
my_sum = a + b + c (this is safe to do)

Deliverables Problem 6

- Complete the functions **following** the algorithm described above.
- You can neither use the % sign in the program nor divide directly by 9.

Problem 7: Base 17

In this problem, you'll write a function that takes a number in base 17 (as a string) and return the decimal value. We'll simply extend base 16 by adding another digit G. Since F represents 15, G will represent 16. Interestingly, Python can interpret base 17 as we've imagined:

```
>>> int("G", 17)
16
>>> int("E2",17)
240
>>> int("10",17)
17
```

Programming Problem 7: Base 17

- Complete the function sec_dec(d) where d is a string of base 17 symbols.
- You **cannot** use the integer function int(x,y) to convert to decimal. You must build the decimal value from powers of 17. You can still use int(x). Revising the lectures slides on conversion between number system can help to understand and solve this problem.

8: Recursion finding Factors

Here is a recursion for d for $x = 0, 1, 2, \ldots$ and $y = 1, 2, 3, \ldots$

$$d(1, y) = 1 \tag{22}$$

$$d(0, y) = y \tag{23}$$

$$d(x, y) = d(b \,\%\, a, a) \tag{24}$$

where $a = \min(x, y), b = \max(x, y)$ Here is a recursion e that relies on d:

$$e(x, y) = \begin{cases} xy & d(x, y) = 1 \\ ze(x//z, y//z) & e(x, y) = z \end{cases} \tag{25}$$

where run

```
data = [[15,25],[6,7],[1,1],[1,2],[0,4],[210,2310]]

for i in data:
    print(e(*i))
```

produces

```
75
42
1
2
0
2310
```

Deliverables Programming Problem 8

- Complete the functions d() and e() using recursion
- We are using Python's integer division // and modulus %

9: Pyramids of Egypt

This recursion is an old mathematical way to build a pyramid of numbers.

$$m([x]) = [\,] \tag{26}$$
$$m([x,y]) = [[x+y]] \tag{27}$$
$$m([x_0,x_1,x_2,\ldots,x_n] = m([x_0+x_1,x_1+x_2,+\cdots,x_{n-1}+x_n]) + \tag{28}$$
$$[[x_0+x_1,x_1+x_2,\cdots,x_{n-1}+x_n]] \tag{29}$$

The code:

```
x = [[1,2,3,4,5],[1],[3,4],[5,10,22],[1,2,3,4,5,6]]
for i in x:
    print(m(i))
```

produces

```
[[48], [20, 28], [8, 12, 16], [3, 5, 7, 9]]
[]
[[7]]
[[47], [15, 32]]
[[112], [48, 64], [20, 28, 36], [8, 12, 16, 20], [3, 5, 7, 9, 11]]
```

Deliverables Programming Problem 9

- Complete the function m

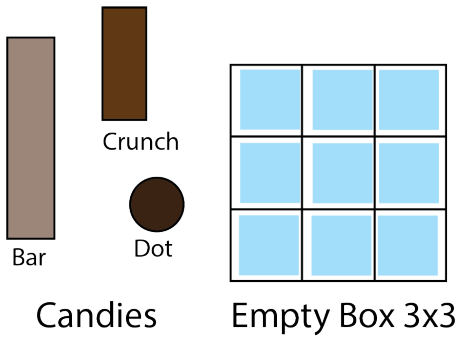

Figure 1: MMMCholocate candy and box.

10: Optimizal Packaging: Minimizing wasted space

You've been contracted by a candy company MMMChocolate. This company has three candies: the bar, the crunch, and the dot. The box that MMMChocolate uses is 3 units. The bar is 3, the crunch is 2×1, and the dot is 1×1. Fig. 1 shows the three candies and the box that is used. The blue highlights the empty spaces. Your task is write a function called package(candies, space=9) that takes a list of candies and an initially empty box and places them for shipping. When a box can not accomodate a candy, then a new box is used. A assembly line of candies has assortments of the three. The robot fills the box. If the next piece of candy can't fit, then the robot uses another box.

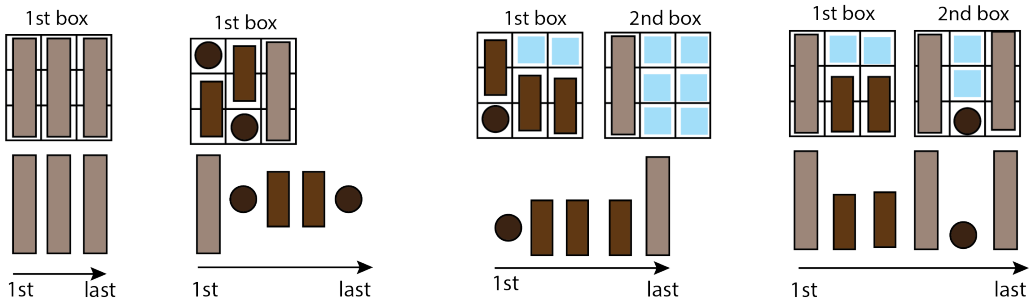

Figure 2: Various lines of candies and the boxes used.

In Fig. 2 (first sub-figure on the left). The Bars come and fit completely into the box. Observe there does not exist any space left. Assume a new assembly line has a Crunch, dot, bar, bar, dot (we'll denote these by the height as [3,1,2,2,1]. Then the box is completely filled. Asssume a new assembly line is started and the candies are [1,2,2,2,3]. Then the first box will have two spaces unfilled. Since the Crunch takes three, a new box is retrieved and the Crunch is placed. Observe that total empty space is 8 (one for each 1×1). The last example, shows that there are a total of four spaces left. The function you write takes a list of candies and an initially empty box (denoted by space=9) and returns the total empty space. The code:

```
1 candies = [[3,3,3],[3,1,2,2,1],[1,2,2,2,3],[3,2,2,3,1,3]]
2 for c in candies:
3     print(package(c))
```

produces

```
0
0
8
4
```

The company is interested in how much space is wasted, since it's a flat-fee to send a box.

Deliverables Programming Problem 10

- Complete the function package().
- You can choose to implement this using any kind of loop, but the recursive implementation, from my perspective, is the easiest.

10: Caeser Cipher

The Caeser Cipher (CC) is a means of encoding a message. Please visit
`https://en.wikipedia.org/wiki/Caesar_cipher`
to read more about it (though it's not as complete as one would hope). We have decided to encode messages for this class using the CC, but with a slight modification. We'll only use the lower case alphabet, and we'll include space as an actual symbol. If you look at the character after 'z' in ASCII, you'll see the left brace "{".

```
>>> for i in range(ord('x'), ord('x') + 4):
...     f"{i} is {chr(i)} in ASCII"
...
'120 is x in ASCII'
'121 is y in ASCII'
'122 is z in ASCII'
'123 is { in ASCII'
>>>
```

We are including "{" in our alphabet after z as the symbol for space. You'll write two functions, encode(msg, shift) and decode(msg, shift) that encode a message and decode the message using shift as discussed in the wikipedia page, except our alphabet is one symbol larger. The following code:

```
data = ["abc xyz","the cat", "i love ctwohundred"]
for i,j in enumerate(data,start=2):
    print(f"original msg {j}")
    print(f"encoded  msg {encode(j,i)}")
    print(f"decoded  msg {decode(encode(j,i),i)}")
```

```
6
7 secret_msg = encode("the quick brown fox jumps over the lazy dog", 24)
8 print(secret_msg)
9 print(decode(secret_msg,24))
```

produces:

```
1  original msg abc xyz
2  encoded  msg cdebz{a
3  decoded  msg abc xyz
4  original msg the cat
5  encoded  msg wkhcfdw
6  decoded  msg the cat
7  original msg i love ctwohundred
8  encoded  msg mdpszidgx{slyrhvih
9  decoded  msg i love ctwohundred
10 original the quick brown fox jumps over the lazy dog
11 encoded qebxnrf{hxzoltkxcluxgrjmpxlsboxqebxiywvxald
12 decoded the quick brown fox jumps over the lazy dog
```

Our output is obviously different from the wikipedia example, since we're expanding our alphabet to consider space a letter and including that in the shift.

Deliverables Programming Problem 10

- Complete the encode() and decode() functions.
- You can use any normal string function that is covered in the class or labs.

Programming partners

bkante@iu.edu, reddyrr@iu.edu
saecohen@iu.edu, orrostew@iu.edu
vkommar@iu.edu, emisimps@iu.edu
phjhess@iu.edu, lmeldgin@iu.edu
jwdrew@iu.edu, jwember@iu.edu
colrkram@iu.edu, clmcevil@iu.edu
ethbrock@iu.edu, rt11@iu.edu
spgreenf@iu.edu, aidnschi@iu.edu, sezinnkr@iu.edu
bencho@iu.edu, erschaef@iu.edu
loggreen@iu.edu, bmpool@iu.edu
zacbutle@iu.edu, patedev@iu.edu
ridbhan@iu.edu, mzagotta@iu.edu
achordi@iu.edu, kvpriede@iu.edu

nharkins@iu.edu, dyashwar@iu.edu
bellcol@iu.edu, anajmal@iu.edu
hermbrar@iu.edu, asaokho@iu.edu
efritch@iu.edu, apavlako@iu.edu
saganna@iu.edu, annaum@iu.edu
cannan@iu.edu, emgward@iu.edu
flynncj@iu.edu, wtatoole@iu.edu
hh35@iu.edu, leolin@iu.edu
micahand@iu.edu, voram@iu.edu
aberkun@iu.edu, ragmahaj@iu.edu
aaragga@iu.edu, gmpierce@iu.edu
greenpat@iu.edu, woodsky@iu.edu
ahavlin@iu.edu, epautsch@iu.edu
wgurley@iu.edu, wwtang@iu.edu
nmcastan@iu.edu, rvinzant@iu.edu
dkkosim@iu.edu, antando@iu.edu
jakchap@iu.edu, coenthom@iu.edu
deombeas@iu.edu, gavilleg@iu.edu
diebarro@iu.edu, amanocha@iu.edu
lpfritsc@iu.edu, tolatinw@iu.edu
simadams@iu.edu, cltran@iu.edu
alchatz@iu.edu, megapaul@iu.edu
earuland@iu.edu, ysanghi@iu.edu
johnguen@iu.edu, rafir@iu.edu
cpkerns@iu.edu, thnewm@iu.edu
oakinsey@iu.edu, jactrayl@iu.edu
ddrotts@iu.edu, vpolu@iu.edu
twfine@iu.edu, wlyzun@iu.edu
evacoll@iu.edu, apathma@iu.edu
davgourl@iu.edu, samrile@iu.edu
leokurtz@iu.edu, lqadan@iu.edu
jdgonzal@iu.edu, scotbray@iu.edu
marganey@iu.edu, rorymurp@iu.edu
agrevel@iu.edu, liwitte@iu.edu
krisgupt@iu.edu, lukastef@iu.edu
althart@iu.edu, muyusuf@iu.edu
sydecook@iu.edu, gavsteve@iu.edu
keswar@iu.edu, majtorm@iu.edu
mkames@iu.edu, ansakrah@iu.edu
sg40@iu.edu, ism1@iu.edu
josespos@iu.edu, jcn1@iu.edu

ethickma@iu.edu, shrrao@iu.edu
mohiambu@iu.edu, ir1@iu.edu
nfelici@iu.edu, owysmit@iu.edu
maudomin@iu.edu, wtrucker@iu.edu
kapgupta@iu.edu, wtubbs@iu.edu
avulas@iu.edu, gepearcy@iu.edu
skunduru@iu.edu, lpelaez@iu.edu
mdiazrey@iu.edu, keasandl@iu.edu
makinap@iu.edu, sasayini@iu.edu
ameydesh@iu.edu, drsnid@iu.edu
sgaladim@iu.edu, ntorpoco@iu.edu
mrcoons@iu.edu, tpandey@iu.edu
mbrockey@iu.edu, reedkier@iu.edu
aaamoako@iu.edu, utwade@iu.edu
kaihara@iu.edu, dernguye@iu.edu
adhuria@iu.edu, ao9@iu.edu
eakanle@iu.edu, jwu6@iu.edu
fkeele@iu.edu, anemlunc@iu.edu
ckdiallo@iu.edu, dsummit@iu.edu
seangarc@iu.edu, ajtse@iu.edu
oeichenb@iu.edu, liansia@iu.edu
zguising@iu.edu, rnschroe@iu.edu
skatiyar@iu.edu, bdyiga@iu.edu
jtbland@iu.edu, jsadiq@iu.edu
bcdutka@iu.edu, mehtriya@iu.edu
ag69@iu.edu, dwo@iu.edu
ruqchen@iu.edu, tzuyyen@iu.edu
escolber@iu.edu, skp2@iu.edu
amkhatri@iu.edu, gszopin@iu.edu
jc168@iu.edu, ijvelmur@iu.edu
jdemirci@iu.edu, pw18@iu.edu
kdembla@iu.edu, savebhat@iu.edu
apchavis@iu.edu, isaramir@iu.edu
mdonato@iu.edu, joshroc@iu.edu
aakindel@iu.edu, aveluru@iu.edu
ayoajayi@iu.edu, aranjit@iu.edu
cfampo@iu.edu, ammulc@iu.edu
apbabu@iu.edu, asidda@iu.edu
jacklapp@iu.edu, ap79@iu.edu
brownset@iu.edu, fshamrin@iu.edu
fkanmogn@iu.edu, mveltri@iu.edu

aketcha@iu.edu, jlzhao@iu.edu
jacobben@iu.edu, pricemo@iu.edu
lcoveney@iu.edu, ryarram@iu.edu
leegain@iu.edu, zshamo@iu.edu
kaneai@iu.edu, jomeaghe@iu.edu
hamac@iu.edu, rpoludas@iu.edu
ajeeju@iu.edu, aselki@iu.edu
fu7@iu.edu, lmadiraj@iu.edu
gandhira@iu.edu, vrradia@iu.edu
anlego@iu.edu, jaslnu@iu.edu
hk120@iu.edu, blswing@iu.edu
nfarhat@iu.edu, nniranj@iu.edu
jwcase@iu.edu, nrizvi@iu.edu
swconley@iu.edu, ntuhl@iu.edu
dblackme@iu.edu, sahaan@iu.edu
bencalex@iu.edu, abiparri@iu.edu
fdonfrio@iu.edu, adsize@iu.edu
lflenoy@iu.edu, perezand@iu.edu
braybrya@iu.edu, dukthang@iu.edu
mkleinke@iu.edu, hasiddiq@iu.edu
joehawl@iu.edu, pravulap@iu.edu
spdamani@iu.edu, cmvanhov@iu.edu
mdoxsee@iu.edu, ruska@iu.edu
alscarr@iu.edu, evataylo@iu.edu
egoldsto@iu.edu, pateishi@iu.edu
harmonnc@iu.edu, justyou@iu.edu
kraus@iu.edu, chrimanu@iu.edu
abellah@iu.edu, rwan@iu.edu
nbernot@iu.edu, jwetherb@iu.edu
cgkabedi@iu.edu, audtravi@iu.edu
rcaswel@iu.edu, nsatti@iu.edu
coopelki@iu.edu, avraya@iu.edu
hawkjod@iu.edu, rtrammel@iu.edu
stkimani@iu.edu, cmarcuka@iu.edu
daxbills@iu.edu, asultano@iu.edu
austdeck@iu.edu, fmahamat@iu.edu
khannni@iu.edu, rosenbbj@iu.edu
bjdahl@iu.edu, maklsmit@iu.edu
nihanas@iu.edu, btasa@iu.edu
sakalwa@iu.edu, mnimmala@iu.edu
elybrewe@iu.edu, snyderjk@iu.edu

brhint@iu.edu, thomps16@iu.edu
tfreson@iu.edu, ltmckinn@iu.edu
adwadash@iu.edu, benprohm@iu.edu
jkielcz@iu.edu, sahishah@iu.edu
arklonow@iu.edu, mmarotti@iu.edu
rl29@iu.edu, tangtom@iu.edu
masharre@iu.edu, ksadiq@iu.edu
tychid@iu.edu, wilsori@iu.edu
mbeigie@iu.edu, mszczas@iu.edu
jabbarke@iu.edu, arirowe@iu.edu
gmhowell@iu.edu, masmatth@iu.edu
delkumar@iu.edu, impofujr@iu.edu
kekchoe@iu.edu, mjroelle@iu.edu
mwclawso@iu.edu, aamathew@iu.edu
jhhudgin@iu.edu, gsilingh@iu.edu
caegrah@iu.edu, patelsak@iu.edu
ohostet@iu.edu, aditpate@iu.edu
dce@iu.edu, vyeruba@iu.edu
allencla@iu.edu, patel89@iu.edu
ceub@iu.edu, mz24@iu.edu
coopjose@iu.edu, clscheum@iu.edu
spgerst@iu.edu, mmunaf@iu.edu
clearle@iu.edu, giomayo@iu.edu
kjj6@iu.edu, bcmarret@iu.edu
maxklei@iu.edu, etprince@iu.edu
tcconnol@iu.edu, nmr1@iu.edu
anrkram@iu.edu, jarlmint@iu.edu
mrfehr@iu.edu, jpochyly@iu.edu
alelefeb@iu.edu, vmungara@iu.edu
ovadeley@iu.edu, pp31@iu.edu
wanjiang@iu.edu, sahimann@iu.edu
wilcusic@iu.edu, jdw14@iu.edu
ajgrego@iu.edu, surapapp@iu.edu
jhar@iu.edu, jwmullis@iu.edu
nokebark@iu.edu, madymcsh@iu.edu
jdc6@iu.edu, lvansyck@iu.edu
daminteh@iu.edu, wardjohn@iu.edu
gcopus@iu.edu, cnyarko@iu.edu
milhavil@iu.edu, krbpatel@iu.edu
aroraarn@iu.edu, iperine@iu.edu
nolakim@iu.edu, schwajaw@iu.edu

howardbw@iu.edu, aptheria@iu.edu
grafe@iu.edu, emluplet@iu.edu
schinitz@iu.edu, crmoll@iu.edu
adiyer@iu.edu, cjwaller@iu.edu
sfuneno@iu.edu, qshamsid@iu.edu
agawrys@iu.edu, gs29@iu.edu
ryanbren@iu.edu, nlippman@iu.edu
huhasan@iu.edu, deturne@iu.edu
edfran@iu.edu, jtsuter@iu.edu
blacount@iu.edu, asteini@iu.edu
garcied@iu.edu, nichojop@iu.edu
arnadutt@iu.edu, linjaso@iu.edu
lawmat@iu.edu, jnzheng@iu.edu
laharden@iu.edu, patekek@iu.edu
howamatt@iu.edu, cstancom@iu.edu
ek37@iu.edu, tarturnm@iu.edu
maladwa@iu.edu, brayrump@iu.edu
migriswo@iu.edu, aledminc@iu.edu
quecox@iu.edu, kamaharj@iu.edu
dja1@iu.edu, myeralli@iu.edu
ejhaas@iu.edu, samyuan@iu.edu
laburkle@iu.edu, cialugo@iu.edu
cuizek@iu.edu, smremmer@iu.edu
tchapell@iu.edu, jneblett@iu.edu
matgarey@iu.edu, awsaunde@iu.edu