C200 Programming Assignment №6

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

Indiana University, Bloomington, IN, USA

October 28, 2023

Instructions

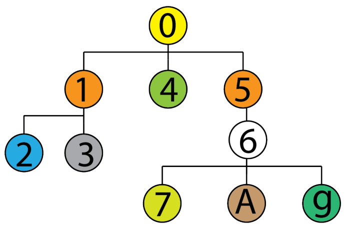
The submission deadline is **Saturday, November 4, 2023 at 11:00 PM**. Please, please begin this homework immediately.

We are comparing the code submitted by different groups. If the submissions made by different groups are similar, we will give a zero in the first instance. In the second instance, we will likely take more drastic actions.

You must both submit to the Autograder (`c200.luddy.indiana.edu`) and also push to GitHub before the deadline.

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.

Parent	Child
0	1
0	4
0	5
1	2
1	3
5	6
6	7
6	A
6	g



Problem 1: Comprehension & File I/O

In this problem, you'll implement functions as comprehensions. The data is in a file called family.txt, it is shown in the table, and also given as a visualization. It is a family tree. The names are simply characters, so no type conversion is necessary.

You will implement a series of queries based on the data that you read from the file. The queries are formed as Python functions. A few are done here:

```python

# children of name
def get_child(name):
    return [child for parent,child in data if parent == name]

#does name have any children
def has_children(name):
    return bool(get_child(name))

#get all values in data (cannot help duplicates)
def get_all():
    return [k for k in [i for i,j in data] + [j for i,j in data]]

print(has_children('0')) #true
print(has_children('7')) #false
print(get_child('6'))
print(get_all())
```

has outputs:

```
True
False
['7', 'A', 'g']
['0', '0', '0', '1', '1', '5', '6', '6', '6', '1', '4', '5', '2', '3', '6'↩
, '7', 'A', 'g']
```

Programming Problem 1: Comprehension & File I/O

- Complete the code to read the file. Do not change any string–these are the names
- Complete each of the functions only using comprehension or bool on a comprehension

Problem 2: Tiling

In this problem, you're given a collection of tiles as numbers, for example, [1,2,3]. You're given a space to fit the tiles (linearly), for example 6. Your function, using recursion, gives all the possible patterns where the tiles add exactly to the fit. For example,

```
  n = 6
  v = [1,2,3]
  print(tiles(n,v,[[i] for i in v]))
  for i in tiles(n,v,[[i] for i in v]):
        print(sum(i), end="")
```

produces:

```
[[3, 3], [3, 2, 1], [2, 3, 1], [3, 1, 2], [2, 2, 2], [1, 3, 2], [2, 1, 3],
 [1, 2, 3], [3, 1, 1, 1], [2, 2, 1, 1], [1, 3, 1, 1], [2, 1, 2, 1],
 [1, 2, 2, 1], [1, 1, 3, 1], [2, 1, 1, 2], [1, 2, 1, 2], [1, 1, 2, 2],
 [1, 1, 1, 3], [2, 1, 1, 1, 1], [1, 2, 1, 1, 1], [1, 1, 2, 1, 1],
 [1, 1, 1, 2, 1], [1, 1, 1, 1, 2], [1, 1, 1, 1, 1, 1]]
666666666666666666666666
```

All the possible configurations and all equal 6. For numbers [1,2] and size = 4, we have:

```
[[2, 2], [2, 1, 1], [1, 2, 1], [1, 1, 2], [1, 1, 1, 1]]
44444
```

The function tiles takes n (the total size), v (the different numbers), and the starter tiles. When approaching this problem, you should inspect the output–it is helpful with reference to what the recursion does.

Programming Problem 2: Tiling

- Complete the function

Problem 3: Max Adjacent

In this problem, you'll implement a function that takes a list of numbers and returns a pair: a sum and a Boolean vector of the numbers that are summed. The condition is that the sum is maximum and that no adjacent numbers can be used. For example,

```
    data = [[5,1,4,1,5],[5,6,2,4],[4,5,1,1],[1,5,10,4,1],[1,1,1,1,1]]
    for d in data:
        print(max_adjacent(d))
```

produces:

```
[14, [1, 0, 1, 0, 1]]
[10, [0, 1, 0, 1]]
[6, [0, 1, 0, 1]]
[12, [1, 0, 1, 0, 1]]
[3, [1, 0, 1, 0, 1]]
```

In the first list, [5,1,4,1,5] we see that 5+4+5 produces the maximal value; so the Boolean list is [1,0,1,0,1]. Observe no numbers are adjacent. For the second list [5,6,2,4], we could have possibly: [7, [1,0,1,0]]; [9,[1,0,0,1]], [10,[0,1,0,1]]. The output is 10, [0,1,0,1]. You are free to implement this as either recursion or looping, but we advise that the recursion is easier if you take advantage of the Boolean list (as you recursively build it) and the base case is a list of length three (no smaller list is used).

Programming Problem 3: Max Adjacent

- Complete the function.
- You may use either recursion or looping (or both I suppose).

Problem 4: Models: Least Squares

A model is a characterization of something. A mathematical model is a formula that describes a relationship among values. The simplest mathematical formula is a linear equation in 2D:

$$0 = Ax + By + C$$

We usually rewrite the equation above as:

$$y = mx + b$$

This means we can determine any value of y by using x. The relationship is **linear** because we're only using $mx + b$, not for example, x^2. Least squares, as we've discussed in lecture is several hundred years old, but remains the most popular modeling for simple relationships. We are given a set of pairs:

$$data = \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$$

and want to build the "best" function $f(x) = \hat{m}x + \hat{b}$. where the $\hat{}$ (called hat) means we're approximating a value (we don't know the actual m, b). Without going through the math, there is now a simple procotol to build $\hat{m}, \hat{b}$:

$$\begin{aligned}
data &= \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\} \\
xy_p &= \sum_{i=0}^{n} x_i y_i \\
x_s &= \sum_{i=0}^{n} x_i \\
y_s &= \sum_{i=0}^{n} y_i \\
x_{sq} &= \sum_{i=0}^{n} x_i^2 \\
S_{xy} &= xy_p - \frac{x_s y_s}{n} \\
S_{xx} &= x_{sq} - \frac{x_s^2}{n} \\
\hat{m} &= \frac{S_{xy}}{S_{xx}} \\
\hat{b} &= \frac{y_s - \hat{m}x_s}{n} \\
f(x) &= \hat{m}x + \hat{b}
\end{aligned}$$

To find the quality of the model, we calculate the so-called $0 \leq R^2 \leq 1$ value. Towards unity, the model is a better predictor–at one, it has no error. Toward zero, it is a worse predictor. We need two values: the total sum of squares SST and the sum of square error SSE. We'll write this as

$M Payroll	Wins
209	89
139	74
101	86
74	74
67	68
49	67
119	97
98	92

Table 1: Payroll/Wins for eight years.

typically seen in Statistics texts:

$$\text{SST} = \sum y^2 - \frac{(\sum y)^2}{n} \quad (1)$$

$$\text{SSE} = \sum y^2 - b\sum y - m\sum xy \quad (2)$$

$$R^2 = \frac{\text{SST} - \text{SSE}}{\text{SST}} \quad (3)$$

While this seems like a lot of loops, using Python comprehension and lambda, we can make quick work of this. Observe each summation is the same–the only thing changing is f. We could write a function that takes the data and function returning the value–here are a couple for y_s and x_{sq}.

```
...
def r(data,f):
        return sum([f(x,y) for x,y in data])
...
y_s = r(data,lambda x,y:y)
x_sq = r(data,lambda x,y:x**2)
...
```

Assume you're working in sports analytics and have the payroll for the team for the last eight years with the number of wins. Table 1 shows the data. Here's a run on some data:

```
m_hat, b_hat, R_sq  = std_linear_regression(data6)
print(m_hat,b_hat, R_sq)
plt.plot([x for x,_ in data6],[y for _,y in data6],'ro')
plt.plot([x for x,_ in data6],[m_hat*x + b_hat for x,_ in data6],'b')
plt.xlabel("$M Payroll")
plt.ylabel("Season Wins")
plt.title(f"Least Squares: m = {m_hat}, b = {b_hat}, R^2 = {R_sq} ")
plt.ylabel("Y")
plt.show()
```

with output:

```
0.125 67.5 0.298
```

and graphic:

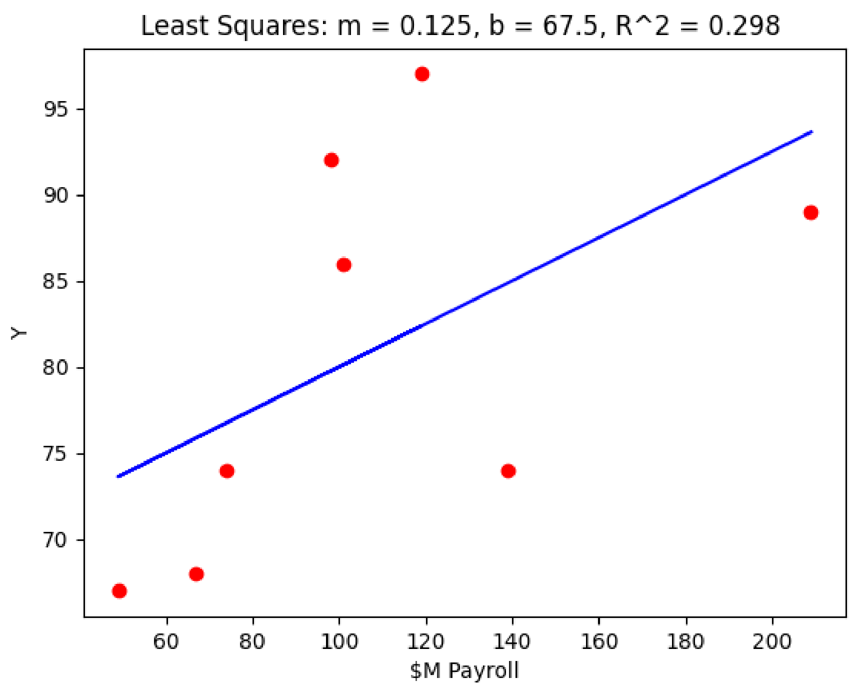


Figure 1: Least squares function (blue) and data (red). The least-squares regression models the linear relationship between the data points. Observe the R^2 value.

Note: For this problem, we have already given the code under (__name__ == __main__) to plot the graph as shown above. After completing this problem, you can un-comment the code to draw the graph. Remember to comment it back before submitting to the Autograder. The Autograder can not draw graphical plots in the web bowser, and therefore your submission attempt may fail with errors even though the code is correct, so please comment the code related to the plot before making final submission.

Deliverables Programming Problem 4

- Complete the function–you are encouraged to use comprehension and lambda, but are free to implement this as you wish
- You cannot use any existing tools for regression–we'll be using those later
- Round the final values of $\hat{m}$, $\hat{b}$, and R^2 to three decimal places
- Food for thought: How would you interpret your findings? What would be your discussion with the coaches and owners?

Problem 5: Polynomial Regression using numpy

In this problem, you'll be modeling rock bass otolith data: the size as a function of years. The

Age (year)	Length (inches)
1	4.8
2	8.8
2	8.0
3	7.9
4	11.9
5	14.4
6	14.1
6	15.8
7	15.6
8	17.8
9	18.2
9	17.1
10	18.8
10	19.5
11	18.9
12	21.7
12	21.9
13	23.8
14	26.9
14	25.1

data is in a csv file called fish_data.txt. When plotted, the data appears polynomial–size three (see Fig. 2).

For this we'll use numpy's polyfit function found here:
`https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html`
You're responsible for reading this document to understand how to use it. To be specific, we have data D and a model

$$f(x) \quad = \quad a_0 + a_1 x + a_2 x^2 + a_3 x^3 \tag{4}$$

From D, we compute the "best" values of a_0, a_1, a_2, a_3. We saw in the previous problem we can use bounded loops to find these values, but there are a myriad of packages and modules available for us to use. Numpy has a polyfit function that takes the data $D = [(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)]$ as a list of x values and y values and returns an array of a_0, a_1, a_2, a_3. We can use these with numpy to build a function called poly1d( ). In this problem you'll need to:

1. Read the data from the file
2. Read about the package from the url provided and build the graph

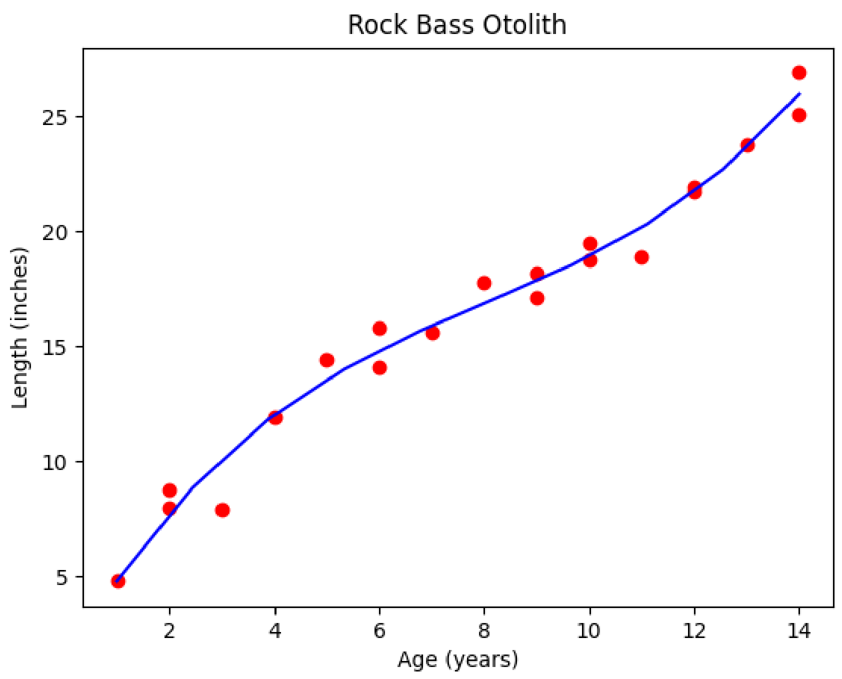


Figure 2: Plots of rock bass data (red) and model (blue). We use numpy's polyfit to do a regression on a polynomial of degree three.

```
def get_fish_data(path,name):
   pass

#INPUT two lists
#RETURN a polynomial function degree 3
def make_function(X,Y, degree):
    pass


path,name = "","fish_data.txt"
X,Y = get_fish_data(path,name)
data4 = [[i,j] for i,j in zip(X,Y)]
print(data4)

plt.plot(X,Y,'ro')                 #plots data D in red

xp = np.linspace(1,14,10) #makes input for model
p3 = make_function(X,Y)  #builds model from data
plt.plot(xp,p3(xp),'b')  #plots  model in bule

plt.xlabel("Age (years)")  #visualization elements
plt.ylabel("Length (inches)")
plt.title("Rock Bass Otolith")
plt.show()
```

gives output:

```
[[1, 4.8], [2, 8.8], [2, 8.0], [3, 7.9], [4, 11.9], [5, 14.4], [6, 14.1], ↩
    [6, 15.8], [7, 15.6], [8, 17.8], [9,
18.2], [9, 17.1], [10, 18.8], [10, 19.5], [11, 18.9], [12, 21.7], [12, ↩
    21.9], [13, 23.8], [14, 26.9], [14, 25.1]]
```

and the plot.

Programming Problem 5: Polynomial Regression with Numpy

- Please read the documentation–please do the small examples shown.
- Complete the functions.
- You should plot the data first (red dots) without plotting the function (blue line) to see if the data is correct.
- Keep in mind that after you have tested your code, you must comment out the 'import matplotlib' and the plotting code given under the __main__ before submitting to the Autograder. Autograder can not draw graphical plots on the web browser so it will return an error if you do not comment out the plotting code/import matplotlib.
- Unlike the previous file, this has a so-called *header*. Headers are useful, but in the simplest settings can be discarded. Please refer to: `https://docs.python.org/3/library/csv.html`

Problem 6: Same, but Different

This problem will ask you to break the problem into smaller, function parts. When solving a complex computing problem, we never write one monolithic program. We break the problem into small, manageable parts and integrate them. For this problem, you are encouraged to look at previous homeworks too. The task is paradoxically simple. Given a string $s_0 s_1 \cdots s_m$ (length is $m+1$) and a number n, return a list of the longest string that has, at most, n distinct symbols. Of course, there might be more than one string satisfying this. For example, the code:

```python
 data = ["aaaba", "abcba", "abbcde","aaabbbaaaaaac","abcdeffg"]
    for d in data:
        for i in range(1,7):
            print(f"{d} with {i} max is\n {max_n(d,i)}")
```

returns:

```
aaaba with 1 max is
 ['aaa']
```

```
aaaba with 2 max is
 ['aaaba']
aaaba with 3 max is
 ['']
aaaba with 4 max is
 ['']
aaaba with 5 max is
 ['']
aaaba with 6 max is
 ['']
abcba with 1 max is
 ['a', 'b', 'c']
abcba with 2 max is
 ['bcb']
abcba with 3 max is
 ['abcba']
abcba with 4 max is
 ['']
abcba with 5 max is
 ['']
abcba with 6 max is
 ['']
abbcde with 1 max is
 ['bb']
abbcde with 2 max is
 ['abb', 'bbc']
abbcde with 3 max is
 ['abbc', 'bbcd']
abbcde with 4 max is
 ['abbcd', 'bbcde']
abbcde with 5 max is
 ['abbcde']
abbcde with 6 max is
 ['']
aaabbbaaaaaac with 1 max is
 ['aaaaaa']
aaabbbaaaaaac with 2 max is
 ['aaabbbaaaaaa']
aaabbbaaaaaac with 3 max is
 ['aaabbbaaaaaac']
aaabbbaaaaaac with 4 max is
 ['']
aaabbbaaaaaac with 5 max is
 ['']
aaabbbaaaaaac with 6 max is
 ['']
abcdeffg with 1 max is
```

```
 ['ff']
abcdeffg with 2 max is
 ['eff', 'ffg']
abcdeffg with 3 max is
 ['deff', 'effg']
abcdeffg with 4 max is
 ['cdeff', 'deffg']
abcdeffg with 5 max is
 ['bcdeff', 'cdeffg']
abcdeffg with 6 max is
 ['abcdeff', 'bcdeffg']
```

For example, let's find the longest strings with at most 2 different symbols:

```
aaaba with 2 max is
 ['aaaba']
abbcde with 2 max is
 ['abb', 'bbc']
aaabbbaaaaaac with 2 max is
 ['aaabbbaaaaaa']
abcdeffg with 2 max is
 ['eff', 'ffg']
```

The first string returns the entire string–there are only two symbols entirely. The second string returns the first three 'abbcde'[0:3] and 'abbcde'[1:4]. Including any other symbols would violate $n = 2$. The second to last string includes every symbol except the last.

Programming Problem 6: Same, but Different

- Complete the function.
- You are strongly encouraged to write several smaller (local) functions to solve this problem. You cannot write global functions, because the autograder can't determine what you need.
- Look to other, older homework for some ideas.

Programming Problem 7: Just One More Throw

In this problem, you'll be simulating a simple gambling game. Our interest is whether the theoretical result matches a simluation. In this game, you flip a coin to reach a sum of money m. You begin with an initial amount of money $b = \$0, 1, 2, \ldots$. If you get heads, you add \$1 to your money and are nearer to m. If you get tails, you lose \$1 from your money and get closer to zero money. The coin is measured at the start and your probability of winning is declared. The

Trial	Outcomes
(1)	2, 3, 2, 3, 2, 1, 0
(2)	2, 3, 2, 3, 2, 3, 2, 1, 2, 3, 4
(3)	2, 1, 2, 3, 4
(4)	2, 3, 4
(5)	2, 3, 2, 1, 0
(6)	2, 1, 2, 3, 2, 3, 4
(7)	2, 3, 4
(8)	2, 3, 2, 3, 2, 3, 2, 1, 2, 1, 0
(9)	2, 1, 2, 3, 4
(10)	2, 1, 2, 3, 2, 3, 2, 1, 0

game continues until you reach your goal or have no money left. In this simulation, you want to run k trials (games) and keep track of the successes. For example, say you simluate $k = 10$ trials, you start with $b = \$2, p = .6$ (the coin gives heads 60% of the time), and you stop when you reach $m = \$4$. On trial 1 (as shown in the table above), you won, lost, won, lost, lost, lost. This resulted in zero money. On the other hand, on trial 4, you won, won. The outcome was success. There are 6 wins out of 10 total attempts so the percentage is 60% or .6. Six times you reached $4 dollars. The theoretical model (we can go over how to derive this later) is:

$$win(b, p, m) \quad = \quad \frac{1-(\frac{1-p}{p})^b}{1-(\frac{1-p}{p})^m} \tag{5}$$

For our simuation,

$$win(2, .6, 4) \quad = \quad \frac{0.555}{0.802} \approx 0.69 \tag{6}$$

So the theory says we should win almost 70% of the time. When this code is run:

```python
model_parameters = (2,.6,4) #starting amount, probablity of win, goal
print(simulation(model_parameters,100000))
```

my output is (when rounded to two decimal places):

```
0.69
```

Your task is to implement this model. The function will take the model parameters (b, p, m) and number of trials and return the percentage of wins as a fraction. To help us, we'll use numpy random variable method:
`https://numpy.org/doc/stable/reference/random/generated/numpy.random.binomial.html`
As you can see, it says *binomial*, but you can use it to simulate a Bernoulli distribution–please visit

https://en.wikipedia.org/wiki/Bernoulli_distribution. A Bernoulli is a stochastic even that has only two outcomes: 1 with probabily p and 0 with probably $1-p$. We can use this (as you will in your simulation) to determine a single outcome.

```
>>> import numpy as np
>>> np.random.binomial(1,.6, 1)
array([1])
>>> for _ in range(10):
...     np.random.binomial(1,.6, 1)[0]
...
1
1
0
0
1
1
0
1
1
1
>>>
```

As you can see, you get an array of values. Since we only wanted one trial, it's a single number, either 1 (for success p) or 0 (for $1-p$). Creating a list of millions of values isn't a good approach, since we can do it piecemeal.

Programming Problem 7: Just One More Throw

- Complete the function.
- Use numpy's random.binomial to simulate the Bernoulli.
- Return the percent success rounded to two numbers.
- Run your simulation on at least 1,000,000 trials.

Programming partners

agrevel@iu.edu, sasayini@iu.edu
ridbhan@iu.edu, mmunaf@iu.edu
brownset@iu.edu, ijvelmur@iu.edu
adwadash@iu.edu, keasandl@iu.edu
fdonfrio@iu.edu, scotbray@iu.edu
mkames@iu.edu, samrile@iu.edu
lawmat@iu.edu, emisimps@iu.edu

nolakim@iu.edu, gepearcy@iu.edu
mohiambu@iu.edu, emgward@iu.edu
bjdahl@iu.edu, dernguye@iu.edu
marganey@iu.edu, jarlmint@iu.edu
dce@iu.edu, maklsmit@iu.edu
quecox@iu.edu, emluplet@iu.edu
mwclawso@iu.edu, mzagotta@iu.edu
nfelici@iu.edu, joshroc@iu.edu
dkkosim@iu.edu, patedev@iu.edu
zguising@iu.edu, vrradia@iu.edu
saganna@iu.edu, dukthang@iu.edu
jdc6@iu.edu, impofujr@iu.edu
earuland@iu.edu, wtatoole@iu.edu
hk120@iu.edu, crmoll@iu.edu
evacoll@iu.edu, aidnschi@iu.edu
ryanbren@iu.edu, ap79@iu.edu
phjhess@iu.edu, ir1@iu.edu
mdiazrey@iu.edu, pateishi@iu.edu
bencalex@iu.edu, mnimmala@iu.edu
spdamani@iu.edu, rpoludas@iu.edu
mbeigie@iu.edu, rorymurp@iu.edu
cpkerns@iu.edu, skp2@iu.edu
oeichenb@iu.edu, jwu6@iu.edu
daxbills@iu.edu, sahimann@iu.edu
brhint@iu.edu, nmr1@iu.edu
mbrockey@iu.edu, anajmal@iu.edu
lpfritsc@iu.edu, lpelaez@iu.edu
egoldsto@iu.edu, jwmullis@iu.edu
ajgrego@iu.edu, benprohm@iu.edu
ethickma@iu.edu, ruska@iu.edu
hamac@iu.edu, blswing@iu.edu
abellah@iu.edu, erschaef@iu.edu
deombeas@iu.edu, pw18@iu.edu
oakinsey@iu.edu, vyeruba@iu.edu
joehawl@iu.edu, drsnid@iu.edu
davgourl@iu.edu, nsatti@iu.edu
jdgonzal@iu.edu, lmadiraj@iu.edu
escolber@iu.edu, justyou@iu.edu
jakchap@iu.edu, ao9@iu.edu
bkante@iu.edu, asteini@iu.edu
kraus@iu.edu, kamaharj@iu.edu

efritch@iu.edu, aditpate@iu.edu
howardbw@iu.edu, surapapp@iu.edu
colrkram@iu.edu, ntorpoco@iu.edu
rl29@iu.edu, ryarram@iu.edu
alscarr@iu.edu, vmungara@iu.edu
tfreson@iu.edu, adsize@iu.edu
nbernot@iu.edu, tpandey@iu.edu
clearle@iu.edu, clscheum@iu.edu
skunduru@iu.edu, ajtse@iu.edu
grafe@iu.edu, tolatinw@iu.edu
simadams@iu.edu, etprince@iu.edu
apchavis@iu.edu, snyderjk@iu.edu
swconley@iu.edu, cnyarko@iu.edu
nfarhat@iu.edu, annaum@iu.edu
nharkins@iu.edu, leolin@iu.edu
gcopus@iu.edu, wwtang@iu.edu
kekchoe@iu.edu, gmpierce@iu.edu
laburkle@iu.edu, evataylo@iu.edu
amkhatri@iu.edu, gsilingh@iu.edu
ethbrock@iu.edu, rafir@iu.edu
kaneai@iu.edu, pravulap@iu.edu
lcoveney@iu.edu, bdyiga@iu.edu
sakalwa@iu.edu, gavsteve@iu.edu
milhavil@iu.edu, rt11@iu.edu
cuizek@iu.edu, sahaan@iu.edu
spgreenf@iu.edu, audtravi@iu.edu
ohostet@iu.edu, rwan@iu.edu
tcconnol@iu.edu, hasiddiq@iu.edu
bellcol@iu.edu, liwitte@iu.edu
ayoajayi@iu.edu, btasa@iu.edu
dblackme@iu.edu, dyashwar@iu.edu
masharre@iu.edu, zshamo@iu.edu
sgaladim@iu.edu, aptheria@iu.edu
loggreen@iu.edu, aveluru@iu.edu
cannan@iu.edu, avraya@iu.edu
leegain@iu.edu, reddyrr@iu.edu
aroraarn@iu.edu, abiparri@iu.edu
tchapell@iu.edu, vpolu@iu.edu
zacbutle@iu.edu, coenthom@iu.edu
wgurley@iu.edu, patekek@iu.edu
gmhowell@iu.edu, masmatth@iu.edu

jabbarke@iu.edu, bcmarret@iu.edu
diebarro@iu.edu, mz24@iu.edu
hh35@iu.edu, mjroelle@iu.edu
ruqchen@iu.edu, jsadiq@iu.edu
rcaswel@iu.edu, megapaul@iu.edu
nmcastan@iu.edu, myeralli@iu.edu
micahand@iu.edu, nlippman@iu.edu
aberkun@iu.edu, pp31@iu.edu
lflenoy@iu.edu, deturne@iu.edu
aakindel@iu.edu, mveltri@iu.edu
keswar@iu.edu, amanocha@iu.edu
coopelki@iu.edu, kvpriede@iu.edu
twfine@iu.edu, rvinzant@iu.edu
alchatz@iu.edu, cltran@iu.edu
sfuneno@iu.edu, pricemo@iu.edu
kjj6@iu.edu, tarturnm@iu.edu
mrfehr@iu.edu, jpochyly@iu.edu
matgarey@iu.edu, aledminc@iu.edu
jacobben@iu.edu, arirowe@iu.edu
gandhira@iu.edu, wilsori@iu.edu
nokebark@iu.edu, cialugo@iu.edu
stkimani@iu.edu, perezand@iu.edu
arnadutt@iu.edu, asaokho@iu.edu
mrcoons@iu.edu, wlyzun@iu.edu
aaamoako@iu.edu, ragmahaj@iu.edu
adiyer@iu.edu, fshamrin@iu.edu
althart@iu.edu, rnschroe@iu.edu
flynncj@iu.edu, apavlako@iu.edu
krisgupt@iu.edu, ltmckinn@iu.edu
dja1@iu.edu, gszopin@iu.edu
mdonato@iu.edu, muyusuf@iu.edu
alelefeb@iu.edu, anemlunc@iu.edu
sydecook@iu.edu, antando@iu.edu
aaragga@iu.edu, sezinnkr@iu.edu
sg40@iu.edu, asultano@iu.edu
saecohen@iu.edu, nniranj@iu.edu
huhasan@iu.edu, jcn1@iu.edu
ceub@iu.edu, aranjit@iu.edu
ckdiallo@iu.edu, jtsuter@iu.edu
fkanmogn@iu.edu, chrimanu@iu.edu
leokurtz@iu.edu, fmahamat@iu.edu

aketcha@iu.edu, asidda@iu.edu
jwdrew@iu.edu, bmpool@iu.edu
apbabu@iu.edu, lqadan@iu.edu
migriswo@iu.edu, orrostew@iu.edu
ek37@iu.edu, rtrammel@iu.edu
ameydesh@iu.edu, nrizvi@iu.edu
ovadeley@iu.edu, wtubbs@iu.edu
hermbrar@iu.edu, ansakrah@iu.edu
ag69@iu.edu, wardjohn@iu.edu
jhar@iu.edu, reedkier@iu.edu
ahavlin@iu.edu, savebhat@iu.edu
skatiyar@iu.edu, jdw14@iu.edu
ejhaas@iu.edu, jnzheng@iu.edu
coopjose@iu.edu, cjwaller@iu.edu
caegrah@iu.edu, smremmer@iu.edu
jacklapp@iu.edu, thomps16@iu.edu
kdembla@iu.edu, awsaunde@iu.edu
allencla@iu.edu, ysanghi@iu.edu
kaihara@iu.edu, cmarcuka@iu.edu
jtbland@iu.edu, jomeaghe@iu.edu
kapgupta@iu.edu, dwo@iu.edu
anlego@iu.edu, rosenbbj@iu.edu
cgkabedi@iu.edu, gavilleg@iu.edu
greenpat@iu.edu, tzuyyen@iu.edu
liansia@iu.edu, mmarotti@iu.edu
hawkjod@iu.edu, clmcevil@iu.edu
seangarc@iu.edu, aamathew@iu.edu
garcied@iu.edu, sahishah@iu.edu
johnguen@iu.edu, cstancom@iu.edu
arklonow@iu.edu, aselki@iu.edu
adhuria@iu.edu, linjaso@iu.edu
jhhudgin@iu.edu, brayrump@iu.edu
maudomin@iu.edu, woodsky@iu.edu
agawrys@iu.edu, gs29@iu.edu
schinitz@iu.edu, ism1@iu.edu
jc168@iu.edu, majtorm@iu.edu
spgerst@iu.edu, cmvanhov@iu.edu
vkommar@iu.edu, nichojop@iu.edu
anrkram@iu.edu, ntuhl@iu.edu
jwcase@iu.edu, ksadiq@iu.edu
bencho@iu.edu, jneblett@iu.edu

delkumar@iu.edu, thnewm@iu.edu
achordi@iu.edu, jlzhao@iu.edu
mkleinke@iu.edu, lvansyck@iu.edu
austdeck@iu.edu, apathma@iu.edu
daminteh@iu.edu, jactrayl@iu.edu
eakanle@iu.edu, voram@iu.edu
maladwa@iu.edu, isaramir@iu.edu
edfran@iu.edu, utwade@iu.edu
laharden@iu.edu, krbpatel@iu.edu
jdemirci@iu.edu, lmeldgin@iu.edu
fu7@iu.edu, jaslnu@iu.edu
nihanas@iu.edu, epautsch@iu.edu
wanjiang@iu.edu, wtrucker@iu.edu
ajeeju@iu.edu, mehtriya@iu.edu
khannni@iu.edu, ammulc@iu.edu
josespos@iu.edu, samyuan@iu.edu
bcdutka@iu.edu, jwember@iu.edu
fkeele@iu.edu, patelsak@iu.edu
makinap@iu.edu, patel89@iu.edu
ddrotts@iu.edu, giomayo@iu.edu
blacount@iu.edu, jwetherb@iu.edu
avulas@iu.edu, schwajaw@iu.edu
tychid@iu.edu, qshamsid@iu.edu
cfampo@iu.edu, mszczas@iu.edu
wilcusic@iu.edu, dsummit@iu.edu
jkielcz@iu.edu, owysmit@iu.edu
mdoxsee@iu.edu, tangtom@iu.edu
howamatt@iu.edu, iperine@iu.edu
maxklei@iu.edu, madymcsh@iu.edu