C200 Programming Assignment №8
Classes, Files

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

Indiana University, Bloomington, IN, USA

November 5, 2022

Introduction

This is a relatively short HW (2 problems), but involve some amount of reading. Since there are just 2 problems, the HW is due on **Thursday, November, 10, 2022 at 10:59 PM EST**. Please submit to the Autograder (`c200.luddy.indiana.edu`) and push your work before the deadline. It is important that you **try your best to submit to the Autograder and merely pushing to Github is not an alternative for not submittig to the Autograder**. Submitting to the Autograder enable you to find which functions are failing the test cases, and you can actively work on them. Whereas, simply pushing to GitHub won't let you improve the code.

Student pairs: The list of student pairs is given at the end of HW document. Contact your programming partner ahead of time.

Note:

- 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).
- Please ensure that you **debug** the code well (syntax, logical and implementation errors), before submitting to the Autograder.
- 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.
- Given that you already tried above points (bullet 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.

- For Problem-1, use the instructions given in the starter code to run and test it. It may not directly run in the VSC.
- Once you are done testing your code, comment out the tests i.e. the code under the '__name__' == "__main__" section.

Problem 1: Central Dogma

The central dogma in biology is that DNA $\rightarrow$ RNA $\rightarrow$ protein. If you want, you can read more about it at `https://en.wikipedia.org/wiki/Central_dogma_of_molecular_biology`. In this problem you will read in a two files: The first file (amino_acids.txt) contains mapping of codons (a triplet of three DNA bases is called as **codon**) , by mapping we mean a rule that tells which specific codons will convert into a specific amino acid, and the second file (DNA.txt) contains a DNA sequence. The first file will help you to understand the mappings, and then by using the mappings, we will convert the sequence from the second file (DNA.txt) into amino acids. In essence, we will write a program to convert the DNA into protein as living organisms do! (a protein is a sequence of amino acids)

To start with, the contents of amino_acids.txt are shown below in the table.

Name	Abr.	Codons
Isoleucine	I	ATT, ATC, ATA
Leucine	L	CTT, CTC, CTA, CTG, TTA, TTG
Valine	V	GTT, GTC, GTA, GTG
Phenylalanine	F	TTT, TTC
Methionine	M	ATG
Cysteine	C	TGT, TGC
Alanine	A	GCT, GCC, GCA, GCG
Glycine	G	GGT, GGC, GGA, GGG
Proline	P	CCT, CCC, CCA, CCG
Threonine	T	ACT, ACC, ACA, ACG
Serine	S	TCT, TCC, TCA, TCG, AGT, AGC
Tyrosine	Y	TAT, TAC
Tryptophan	W	TGG
Glutamine	Q	CAA, CAG
Asparagine	N	AAT, AAC
Histidine	H	CAT, CAC
Glutamic acid	E	GAA, GAG
Aspartic acid	D	GAT, GAC
Lysine	K	AAA, AAG
Arginine	R	CGT, CGC, CGA, CGG, AGA, AGG
Stop codons	-	TAA, TAG, TGA

The first column is the full name of the amino acid. The second column is the abbreviation as

one letter initial (for the same amino acid). For the Stop_codons, we use a dash. The rightmost column are what three letters of DNA are used to make the amino acid. The amino acid Arginine has an abbreviation R. There are six codon (three bases of DNA) that code for Arginine: CGT, CGC, CGA, CGG, AGA, AGG, as can be seen in the table.

About DNA.txt: It's basically a FASTA file. Please don't be confused by the name-it's just a text file that folllows certain rules, hence the special name. It has two parts: a header (information about the DNA sequence that it contains) and the DNA sequence itself. Here's the one you'll be using (don't copy them from here, we have already pushed both files to your repositories).

```
>HSGLTH1 Human theta 1-globin gene
CCACTGCACTCACCGCACCCGGCCAATTTTTGTGTT
TTTAGTAGAGACTAAATACCATATAGTGAACACCTA
AGACGGGGGGCCTTGGATCCAGGGCGATTCAGAGG
GCCCCGGTCGGAGCTGTCGGAGATTGAGCGCGCGC
GGTCCCGGGATCTCCGACGAGGCCCTGGACCCCCG
GGCGGCGAAGCTGCGGCGCGGCGCCCCCTGGAGGC
CGCGGGACCCCTGGCCGGTCCGCGCAGGCGCAGCG
GGGTCGCAGGGCGCGGCGGGTTCCAGCGCGGGGAT
GGCGCTGTCCGCGGAGGACCGGGCGCTGGTGCGCG
CCCTGTGGAAGAAGCTGGGCAGCAACGTCGGCGTCT
ACACGACAGAGGCCCTGGAAAGGTGCGGCAGGCTG
GGCGCCCCCGCCCCCAGGGGCCCTCCCTCCCCAAG
CCCCCCGGACGCGCCTCACCCACGTTCCTCTCGCAG
GACCTTCCTGGCTTTCCCCGCCACGAAGACCTACTT
CTCCCACCTGGACCTGAGCCCCGGCTCCTCACAAGT
CAGAGCCCACGGCCAGAAGGTGGCGGACGCGCTGA
GCCTCGCCGTGGAGCGCCTGGACGACCTACCCCAC
GCGCTGTCCGCGCTGAGCCACCTGCACGCGTGCCA
GCTGCGAGTGGACCCGGCCAGCTTCCAGGTGAGCG
GCTGCCGTGCTGGGCCCCTGTCCCCGGGAGGGCCC
CGGCGGGGTGGGTGCGGGGGGCGTGCGGGGCGGG
TGCAGGCGAGTGAGCCTTGAGCGCTCGCCGCAGCT
CCTGGGCCACTGCCTGCTGGTAACCCTCGCCCGGCA
CTACCCCGGAGACTTCAGCCCCGCGCTGCAGGCGTC
GCTGGACAAGTTCCTGAGCCACGTTATCTCGGCGCT
GGTTTCCGAGTACCGCTGAACTGTGGGTGGGTGGCC
GCGGGATCCCCAGGCGACCTTCCCCGTGTTTGAGTA
AAGCCTCTCCCAGGAGCAGCCTTCTTGCCGTGCTCT
CTCGAGGTCAGGACGCGAGAGGAAGGCGC
```

Though not necessary but if you want, you can read about this gene here: `https://pubmed.ncbi.nlm.nih.gov/3422341/`. In the file, the first line describes the sequence

providing the name and other attributes. The remaining lines is the DNA sequence (ignore all whitespace), all lines after the header are part of the same sequence so there are no line breaks (ignore whitespaces) in the sequence.

Example: how to translate DNA into a protein

To convert from DNA to protein, we use a sequence of codons. We'll bold the protein when its translated.

Let's look at the first twelve bases: CCACTGCACTCA. Every three bases uniquely determine an amino acid.

1. Start with the first codon CCA, CCACTGCACTCA.
2. Looking at the first file we see: Proline, P, CCT, CCC, CCA, CCG. This means we can rewrite CCA as P.
3. Looking at the next codon CTG, **P**CTGCACTCA
4. We find it matches Leucine, L, CTT, CTC, CTA, CTG, TTA, TTG. So our protein is PL.
5. The next three are CAC **PL**CACTCA.
6. The table has Histidine, H, CAT, CAC.
7. The final three are TCA. **PLH**TCA
8. This matches Serine, S, TCT, TCC, TCA, TCG, AGT, AGC.
9. The protein is **PLHS**.

If you are at the end and only have two bases, you cannot match a protein, so you ignore them. Suppose we had CCAC. We know CCA is P. Then we only have C left. We ignore it.

How to approach this problem

Our main task is to take the DNA (by reading DNA.txt) and produce a string of single letters (as shown in above example, points 1-9) that reflect the encoding. Here is one way to solve this.

1. First you'll read in the table from amino_acids.txt, and create a dictionary whose entries are:

$$aa\_d \quad = \quad \{(c_0, c_1, \ldots, c_n) : [name, letter], \ldots\}$$

where c_i is a three letter codon, $name$ is the full name of the amino acid, and $letter$ is the single letter for the amino acid. Make sure you follow the format of keys and values exactly as shown here. The function get_amino_acid() takes the file name and returns a dictionary. The dictionary is also shown below in the code listing.

2. Read in the DNA sequence, the function get_DNA() takes a file name and returns a faste data structure [header, DNA] (FASTA data structure) where header is the first line of the file DNA.txt and DNA is the DNA sequence (the sequence of A,T,G,C after the first line) (ignoring any whitespace). The read-in FASTA sequence is also shown below in the code listing.

3. The function translate() takes a FASTA data structure (that you created in step-2) and returns a string that is the translation using the dictionary (that you created in step-1). We can do a simple print to see whether our translation is the same as actual. An exmaple of translate() function is also shown below in the code listing (you are encouraged to cross-check via pen and paper).

This code creates the dictionary and FASTA file (as a list), translates, and validates:

```
print("Dictionary")
print(aa_d)
print("FASTA file")
print(DNA_d)
print("Translations match:", str(protein == actual))

#should return "PLHS"
print(translate(["nothing", "CCACTGCACTCA"]))

#should return "D-"
print(translate(["nothing", "GACTAA"]))
```

has output:

```
Dictionary
{('ATT', 'ATC', 'ATA'): ['Isoleucine', 'I'], ('CTT', 'CTC', 'CTA', 'CTG', 'TTA', 'TTG'): ['Leucine', 'L'], ('GTT', 'GTC', 'GTA', 'GTG'): ['Valine', 'V'], ('TTT', 'TTC'): ['Phenylalanine', 'F'], ('ATG',): ['Methionine', 'M'], ('TGT', 'TGC'): ['CYSteine', 'C'], ('GCT', 'GCC', 'GCA', 'GCG'): ['Alanine', 'A'], ('GGT', 'GGC', 'GGA', 'GGG'): ['Glycine', 'G'], ('CCT', 'CCC', 'CCA', 'CCG'): ['Proline', 'P'], ('ACT', 'ACC', 'ACA', 'ACG'): ['Threonine', 'T'], ('TCT', 'TCC', 'TCA', 'TCG', 'AGT', 'AGC'): ['Serine', 'S'], ('TAT', 'TAC'): ['Tyrosine', 'Y'], ('TGG',): ['Tryptophan', 'W'], ('CAA', 'CAG'): ['Glutamine', 'Q'], ('AAT', 'AAC'): ['Asparagine', 'N'], ('CAT', 'CAC'): ['Histidine', 'H'], ('GAA', 'GAG'): ['Glutamic_acid', 'E'], ('GAT', 'GAC'): ['AsparTic acid', 'D'], ('AAA', 'AAG'): ['Lysine', 'K'], ('CGT', 'CGC', 'CGA', 'CGG', 'AGA', 'AGG'): ['Arginine', 'R'], ('TAA', 'TAG', 'TGA'): ['Stop_codons', '-']}
FASTA file
['>HSGLTH1 Human theta 1-globin gene', 'CCACTGCACTCACCGCACCCGGCCAATTTTTGTGTTTTTAGT
AGAGACTAAATACCATATAGTGAACACCTAAGACGGGGGGC
```

```
CTTGGATCCAGGGCGATTCAGAGGGCCCCGGTCGGAGCTGT
CGGAGATTGAGCGCGCGCGGTCCCGGGATCTCCGACGAGGC
CCTGGACCCCCGGGCGGCGAAGCTGCGGCGCGGCGCCCCCT
GGAGGCCGCGGGACCCCTGGCCGGTCCGCGCAGGCGCAGCG
GGGTCGCAGGGCGCGGCGGGTTCCAGCGCGGGGATGGCGCT
GTCCGCGGAGGACCGGGCGCTGGTGCGCGCCCTGTGGAAGA
AGCTGGGCAGCAACGTCGGCGTCTACACGACAGAGGCCCTG
GAAAGGTGCGGCAGGCTGGGCGCCCCCGCCCCCAGGGGCCC
TCCCTCCCCAAGCCCCCCGGACGCGCCTCACCCACGTTCCTC
TCGCAGGACCTTCCTGGCTTTCCCCGCCACGAAGACCTACTT
CTCCCACCTGGACCTGAGCCCCGGCTCCTCACAAGTCAGAGC
CCACGGCCAGAAGGTGGCGGACGCGCTGAGCCTCGCCGTGG
AGCGCCTGGACGACCTACCCCACGCGCTGTCCGCGCTGAGC
CACCTGCACGCGTGCCAGCTGCGAGTGGACCCGGCCAGCTT
CCAGGTGAGCGGCTGCCGTGCTGGGCCCCTGTCCCCGGGAG
GGCCCCGGCGGGGTGGGTGCGGGGGGCGTGCGGGGCGGGT
GCAGGCGAGTGAGCCTTGAGCGCTCGCCGCAGCTCCTGGGC
CACTGCCTGCTGGTAACCCTCGCCCGGCACTACCCCGGAGAC
TTCAGCCCCGCGCTGCAGGCGTCGCTGGACAAGTTCCTGAGC
CACGTTATCTCGGCGCTGGTTTCCGAGTACCGCTGAACTGTG
GGTGGGTGGCCGCGGGATCCCCAGGCGACCTTCCCCGTGTTTG
AGTAAAGCCTCTCCCAGGAGCAGCCTTCTTGCCGTGCTCTCTC
GAGGTCAGGACGCGAGAGGAAGGCGC']
Translations match: True
PLHS
D-
```

Deliverables Problem 1

- Carefully read the instructions on how to run the code for Problem-1 in the starter file a8.py. You may not be able to run it directly in VSC, so follow the instructions in the starter code.
- Complete the functions get_DNA(), get_amino_acids() and translate() as per the specifications.
- You are allowed to use replace for space and "{". Other uses of it will actually be more difficult.

Problem 2: Improving Quicksort

In class we described the traditional quick sort:

```
def qs(lst):
    if lst:
```

```
        pivot,left,right = lst[0],[],[]
        for i in lst[1:]:
            if i <= pivot:
                left.append(i)
            else:
                right.append(i)
        return qs(left) + [pivot] + qs(right)
    else:
        return []
```

One way to improve it is to observe that this method is faster when len(left) = len(right) (please think about why this is true). Although different aproaches exist, a straightforward way is to calculate the mean from a small collection of numbers. For any list of length larger than or equal to 50, we'll take an random sample with replacement of $x\%$ (see example below in code listing), find the mean of that value and us it as the pivot. For example, if lst = [1,2,3,4,5,6] and want 20%, this would be (we'll use ceiling function):

$$pivot \quad = \quad \lceil (\frac{20}{100})6 \rceil = 2 \tag{1}$$

Then we can use the random module (we have already imported it for you in the starter code):

```
>>> import random as rn
>>> lst = [1,2,3,4,5,6]
>>> pivot = sum([lst[rn.randint(0,5)],lst[rn.randint(0,5)]])/6
>>> pivot
1.5
>>> [i for i in lst if i <= pivot],[i for i in lst if i > pivot]
([1], [2, 3, 4, 5, 6])
```

Of course, each time the mean will be calculated randomly. We'll use the traditional sort, but now have this algorithm:

```
#INPUT lst of numbers, percent
#RETURN sorted list
qso(lst,percent)
    if len(lst) >= 50:
        pivot is mean determined by random sample with replacement of percent
        return qso(left) + qso(right)
    otherwise qs(lst)
```

This only a sketch–there are a few other details you'll have to complete. Here's a run of the program with 1000000 numbers:

```
from time import perf_counter, time

```

```
3  def time_it(f,n):
4      start = perf_counter()
5      f(n)
6      return (perf_counter() - start)
7
8  #your code
9
10 lst = [rn.randint(0,100000) for _ in range(1000000)]
11 print("Data size 1000000")
12 print(f" qs      {time_it(qs,lst):.2} sec")
13 for percent in [1,5,10,20]:
14     print(f" qso {percent:<2}% {time_it(qs,lst):.2} sec")
```

with output:

```
1  Data size 1000000
2   qs      3.7 sec
3   qso 1 % 3.3 sec
4   qso 5 % 3.3 sec
5   qso 10% 3.3 sec
6   qso 20% 3.3 sec
```

This isn't enough runs, but we do see there might be some improvement over traditional quick sort. Your computer will likely have different run-times.

Deliverables Problem 2

- Complete the function qso().
- The program will not be tested on larger sizes, but see what happens when there are 10000000 numbers. Here's my run:

```
1  Data size 10000000
2   qs      8.3e+01 sec
3   qso 1 % 8.3e+01 sec
4   qso 5 % 8.4e+01 sec
5   qso 10% 8.7e+01 sec
6   qso 20% 8.4e+01 sec
```

- You must implement quick sort qs() (the traditional quick sort) using either the code here or your own.

Pairs

Here are your partners for homework 5.

C200 student pairs

adamshm@iu.edu, amurli@iu.edu
dadeyeye@iu.edu, adolata@iu.edu
aaher@iu.edu, ryou@iu.edu
omakinfi@iu.edu, jchobbs@iu.edu
shakolia@iu.edu, rilmhart@iu.edu
abalbert@iu.edu, kreddiva@iu.edu
megalbin@iu.edu, dboecler@iu.edu
waasali@iu.edu, sousingh@iu.edu
ahmalman@iu.edu, stebutz@iu.edu
anders14@iu.edu, skrasher@iu.edu
nsantoin@iu.edu, alehami@iu.edu
jaybaity@iu.edu, msisodiy@iu.edu
ianbaker@iu.edu, jarabino@iu.edu
nbalacha@iu.edu, scclotea@iu.edu
aiballou@iu.edu, nysach@iu.edu
jabarbu@iu.edu, ertrice@iu.edu
cmbeaven@iu.edu, lyncsara@iu.edu
olibelch@iu.edu, cwcrotty@iu.edu
sberck@iu.edu, nakoon@iu.edu
evberg@iu.edu, sjvaleo@iu.edu
sbi@iu.edu, tdonoho@iu.edu
obianco@iu.edu, agvore@iu.edu
jbilbre@iu.edu, eg8@iu.edu
aibitner@iu.edu, kyrhod@iu.edu
jetblack@iu.edu, istorine@iu.edu
ablashe@iu.edu, davingo@iu.edu
pblasio@iu.edu, ceifling@iu.edu
obowcott@iu.edu, giancost@iu.edu
sabowe@iu.edu, azaporo@iu.edu
abrandtb@iu.edu, rdkempf@iu.edu
owebrook@iu.edu, cgoeglei@iu.edu
browpr@iu.edu, blakruss@iu.edu
ttbrowne@iu.edu, emcgough@iu.edu
ecaggian@iu.edu, samstuar@iu.edu
petcarmi@iu.edu, rpogany@iu.edu
jlcarrie@iu.edu, jpascov@iu.edu

carcast@iu.edu, ptstorm@iu.edu
chenjunx@iu.edu, cmw26@iu.edu
tchigudu@iu.edu, darisch@iu.edu
adhichin@iu.edu, jscrogha@iu.edu
jconcial@iu.edu, aubhighb@iu.edu
lizcoro@iu.edu, brkapla@iu.edu
lcosens@iu.edu, tcpatel@iu.edu
bcrick@iu.edu, milahusk@iu.edu
mattcrum@iu.edu, mijherr@iu.edu
pcullum@iu.edu, limingy@iu.edu
tdearbor@iu.edu, kereidy@iu.edu
edeporte@iu.edu, jrosebr@iu.edu
jacdick@iu.edu, kyeosen@iu.edu
dixonjh@iu.edu, brohelms@iu.edu
maldowde@iu.edu, stefschr@iu.edu
wjduncan@iu.edu, krgrohe@iu.edu
aareads@iu.edu, alenmurp@iu.edu
ebya@iu.edu, namcbrid@iu.edu
seckardt@iu.edu, davthorn@iu.edu
augeike@iu.edu, harpebr@iu.edu
jpenrigh@iu.edu, bzurbuch@iu.edu
jjepps@iu.edu, mr86@iu.edu
jfahrnow@iu.edu, as145@iu.edu
nfarhat@iu.edu, avincelj@iu.edu
chafiel@iu.edu, svuppunu@iu.edu
riflemin@iu.edu, samsteim@iu.edu
foxjust@iu.edu, kegupta@iu.edu
magacek@iu.edu, elyryba@iu.edu
landgarr@iu.edu, wemurray@iu.edu
rghafoor@iu.edu, dazamora@iu.edu
dgodby@iu.edu, nmwaltz@iu.edu
goel@iu.edu, nkyryk@iu.edu
gonzavim@iu.edu, sehpark@iu.edu
wgranju@iu.edu, howelcar@iu.edu
mahgree@iu.edu, joelna@iu.edu
jgruys@iu.edu, jtohland@iu.edu
kusgupta@iu.edu, hernaga@iu.edu
dgusich@iu.edu, joluca@iu.edu
jhaile@iu.edu, zaschaff@iu.edu
halejd@iu.edu, marebey@iu.edu
chaleas@iu.edu, aokhiria@iu.edu

hallzj@iu.edu, aketcha@iu.edu
thamed@iu.edu, thuhtoo@iu.edu
hardenja@iu.edu, egshim@iu.edu
chataway@iu.edu, sarmayo@iu.edu
lohernan@iu.edu, eweidne@iu.edu
jonhick@iu.edu, anuttle@iu.edu
bhmung@iu.edu, csmalarz@iu.edu
phoen@iu.edu, jurzheng@iu.edu
nichhoff@iu.edu, xujack@iu.edu
tahoss@iu.edu, shawwan@iu.edu
mihough@iu.edu, cjvanpop@iu.edu
gmhowell@iu.edu, msmelley@iu.edu
leghuang@iu.edu, nasodols@iu.edu
ellhuds@iu.edu, sampopek@iu.edu
bizzo@iu.edu, sunreza@iu.edu
mahajenk@iu.edu, daknecht@iu.edu
bj13@iu.edu, jnjeri@iu.edu
rjjorge@iu.edu, comojica@iu.edu
rkabra@iu.edu, dmetodie@iu.edu
oakagzi@iu.edu, wodmaxim@iu.edu
skalivas@iu.edu, arkirt@iu.edu
nekern@iu.edu, owasmith@iu.edu
vkethine@iu.edu, evmtaylo@iu.edu
keysa@iu.edu, tavalla@iu.edu
drewkimb@iu.edu, ayuraiti@iu.edu
mkirolos@iu.edu, mmpettig@iu.edu
nolknies@iu.edu, dl61@iu.edu
abvekoes@iu.edu, derthach@iu.edu
fdkussow@iu.edu, nrs5@iu.edu
aalesh@iu.edu, branwade@iu.edu
lewiserj@iu.edu, spletz@iu.edu
jolindse@iu.edu, ereno@iu.edu
jonllam@iu.edu, mysoladi@iu.edu
isclubia@iu.edu, zhaofan@iu.edu
vimadhav@iu.edu, patelkus@iu.edu
martiro@iu.edu, sharpky@iu.edu
nmccarry@iu.edu, abramjee@iu.edu
mccoyry@iu.edu, parksdr@iu.edu
kmcinto@iu.edu, jbromers@iu.edu
skmcmaho@iu.edu, gokeefe@iu.edu
gmeinerd@iu.edu, ntatro@iu.edu

luilmill@iu.edu, vsivabad@iu.edu
benrmitc@iu.edu, jwa14@iu.edu
kpmorse@iu.edu, mwroark@iu.edu
jnmroch@iu.edu, lanounch@iu.edu
jamundy@iu.edu, rraguram@iu.edu
shnaka@iu.edu, ssetti@iu.edu
chrinayl@iu.edu, btao@iu.edu
sndashi@iu.edu, patevig@iu.edu
laynicho@iu.edu, ashmvaug@iu.edu
kninnema@iu.edu, jthach@iu.edu
coolds@iu.edu, jomayode@iu.edu
patel88@iu.edu, samsieg@iu.edu
albperez@iu.edu, kviele@iu.edu
perkcaan@iu.edu, bwinckle@iu.edu
petersgm@iu.edu, tanaud@iu.edu
phiprice@iu.edu, esisay@iu.edu
etprince@iu.edu, jeffsung@iu.edu
csradtke@iu.edu, joeywill@iu.edu
antreye@iu.edu, wilsdane@iu.edu
uzrivera@iu.edu, cadwinin@iu.edu
zekerobe@iu.edu, jschlaef@iu.edu
rogerju@iu.edu, jensprin@iu.edu
oschwar@iu.edu, zwoolley@iu.edu
burshell@iu.edu, rorshiel@iu.edu
askrilof@iu.edu, ttsegai@iu.edu
notsolo@iu.edu, lufayshi@iu.edu
johsong@iu.edu, lvanjelg@iu.edu
rsstarli@iu.edu, ianwhit@iu.edu
snsung@iu.edu, zhangjoe@iu.edu
kt10@iu.edu, envu@iu.edu
conthom@iu.edu, eawidema@iu.edu
jttrinkl@iu.edu, rvu@iu.edu
rtrujill@iu.edu, matzhang@iu.edu
mvanworm@iu.edu, mew17@iu.edu
pyahne@iu.edu, bdzhou@iu.edu

H200 student pairs

ethcarmo@iu.edu, avreddy@iu.edu
ligonza@iu.edu, nzaerhei@iu.edu
nagopi@iu.edu, alindval@iu.edu

zfhassan@iu.edu, arangwan@iu.edu
tkefalov@iu.edu, joshprat@iu.edu
venguyen@iu.edu, huntang@iu.edu
marafoth@iu.edu, swa5@iu.edu
jarenner@iu.edu, lorivera@iu.edu
snresch@iu.edu, aktumm@iu.edu
sturaga@iu.edu, bbcolon@iu.edu