Year 10 Algorithmic thinking

The Happy Camping Challenge

By Kelvin

The question goes as follow:

At Trinity, we are organizing a one week camping excursion for year 10 students. The camping site has tents with capacity for 5 persons, so it is time to decide who is going to share a tent with who. Previous years, when deciding who was going to share a tent with who we had some students feeling left out. To prevent conflicts between students, this year we decided to hand in a form to each student where they had to rate from 1 to 10 how happy they would be sharing a tent with each of their class mates.

Now we have all the students preferences and we want the students as happy as possible with their tentmates. However, it is way too much information, the teachers are going crazy trying to make the perfect matches, so they decided to delegate the problem to the students of algorithmic thinking.

You have to design an algorithm that generates groups of 5 people among the students. This algorithm should:

- Assign every student to a tent
- Not put a student in more than one tent
- Maximize the happiness of the students with their tentmates
- Be as efficient as possible

To solve the problem, you will be given a csv file with a table similar to the following:

Students	Jonathan Spence	Shannon Rasmussen	Nicholas Shaw	Michael Lyons
Jonathan Spence	0	5	8	10
Shannon Rasmussen	8	0	6	2
Nicholas Shaw	10	6	0	7
Michael Lyons	10	6	9	0

Where, in the first row and the first column you have the name of all the students participating in the camping. On each row you have the information about the rating a specific student has given to its classmates. Thus, for example Jonathan Spence and Michael Lyons both have rated each other with a 10 and thus would be ideal for them to be in the same tent.

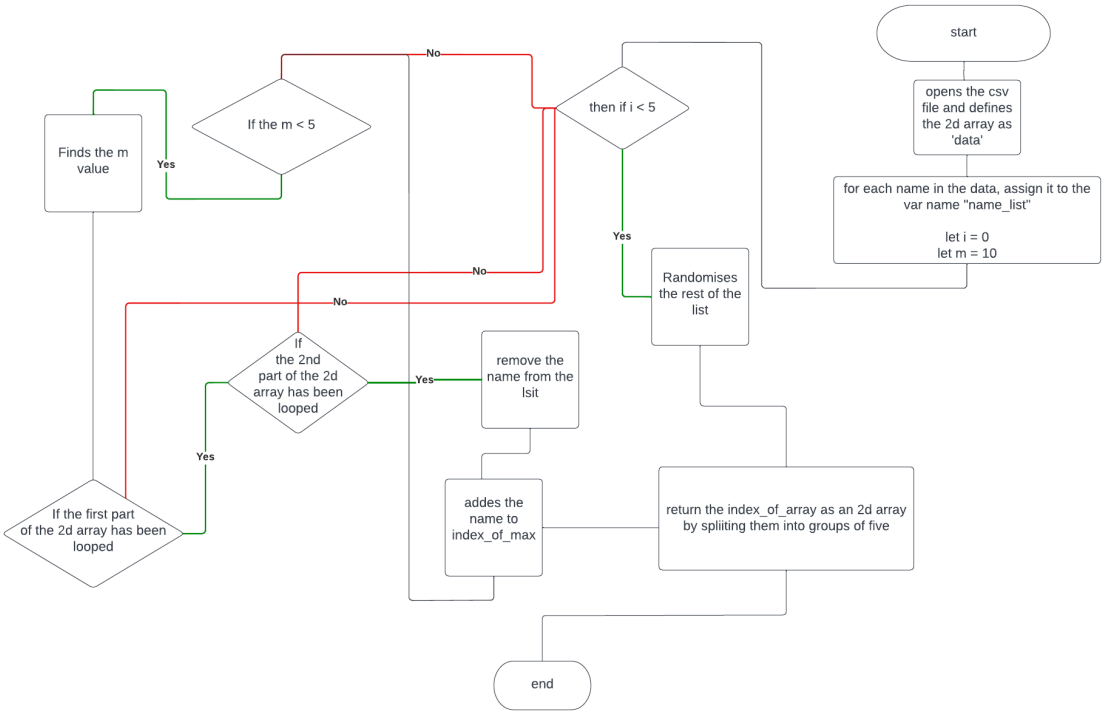
My solution:

My solution is based on ranking people from the highest to the lowest score. Then, I start with the highest score and try to find the best match for him. If I find a match, I remove the two people from the list and continue with the next person. If I don't find a match, I remove the person from the list and continue with the next person. I do this until I have no more people in the list.

The algorithm is efficient because it only goes through the list once and it only removes people from the list when it finds a match or when it can't find a match.

After putting all the people up to score 5, it randomise the rest of the list because there is no point putting them in the list because they are all low ranks and it would be a waste of time to try to find a match for them.

My flow chart of the code is below:



Due to the limited time duration of this assignment, there was not enough time to think of a way to make the algorithm more efficient. However, I believe that there is a way to make it more efficient by using a different data structure. Prehaps a dictionary would be more efficient than a list. Indeed, it might be possible to use a dictionary to store the people and their scores. Then, it would be possible to use the score as the key and the people as the value. This would make it easier to find the best match for a person. But this could reveal a more complex solution.

My first initial suggestion would be using pearson correlation in a 3d space. Where the x and y axis is the name of the list and the z axis is the average rating of the 2 people. Then using pearosn corrlation to find the best match. However, this would be a very complex solution

and would require a lot of time to implement. Conceivably, the program would be less effcient, as there is a lot of data.

Another suggesting would be the idea of nodes. Where the nodes are the people and the edges are the rating. Then, it would be possible to use a graph to find the best match. However, this would be a very complex solution and would require a lot of time to implement. Ultimately, the program would be less effcient, as there is a lot of data.

In this idea, I radomised the list of people. This is because the people with a low score are not going to be happy with anyone. So, it is a waste of time to try to find a match for them.

My pseudocode is also shown below:

```
Algorithem(file):
    CONVERT file to array

    DEFINE name_list as a 2d array
    DEFINE data_of_max as a 2d array
    DEFINE out as a array
    DEFINE pos as 0
    DEFINE n as 0
    DEFINE list_ as 0
    DEFINE out as 2d array

    GET the name of the array and put it into "name_list"
    DELETES the name off the data
    FOR i in the range of the data DO:
        gets the people who rated a 10 and put in "data_of_max"


    WHILE i < 2 DO:
        DEFINE index_of_max as array
            FOR x in data_of_max DO:
                FOR y in x DO:
                    IF y = n DO:
                        IF name_list is empty DO:
                          n = 3
                          BREAK LOOP

                        IF lengh of name list <= 5 DO:
                            add "name list" to "index_of_max"
                            DEFINE name_list as a empty list

                            ELSE DO:
                                WHILE lengh of "index_of_max" <= 3 DO:
                                    SET "list_" to index of data_of_max
relative to position "x"
                                    ADD "name_list[list_]" to
"index_of_max"
                                    REMOVE "name_list[list_]" to
"name_list"
```

```
        IF name_list is empty DO:
            ADD "index_of_max" to "pos" position of array
            BREAK LOOP
        ELSE DO:
            ADD "name_list[n]" to "index_of_max"
            REMOVE "name_list[n]" to "name_list"
            ADD "index_of_max" to "pos" position of array

    RETUN out

END Algorithem
```

My code is shown below:

In [ ]:

```
def create_groups(file):
    import csv


    with open(file, 'r') as f:
        reader = csv.reader(f)
        #create list of lists
        data = list(reader)
        #remove header
        data.pop(0)

    #function is called data

    name_list = []
    for i in range(len(data)):
        name_list.append(data[i][0])

    for i in range(len(data)):
        del data[i][0]


    remove_data_ = [list( map(int,i) ) for i in data] #converts string to int of th
    #print(remove_data_ )

    data_of_max = []

    for i in range(len(remove_data_)):
        #print(i)
        inlist = remove_data_[i]
        max_value = int(max(inlist)) #had error where max valye was only 1 less tha
        index_value = [index for index in range(len(inlist)) if inlist[index] == ma
        data_of_max.insert(i, index_value)

    #print(data_of_max)



    #data_of_max[0].index(3)

    #print(name_list[0])
    #print(data_of_max)
```

```
out = []
pos = 0
#per_tent = int(len(name_list)/5)

n = 0 #people who ranked the first name


while n < 2:

    index_of_max = [] # 5 per tent
    #print(name_list)
    #print(len(name_list))
    #print("")
    #print(data_of_max)


    for x in data_of_max:

        for y in x:
            #print(x, y)
            if y == n:

                if name_list == []:
                    n = 3
                    break

                if len(name_list) <= 5:

                    #print("yes", index_of_max)
                    index_of_max.append(name_list)

                    index_of_max = name_list
                    #print("it is", name_list )

                    name_list = []
                    n = 3
                    break


                else:

                    while len(index_of_max) <= 3:

                        list_ = data_of_max.index(x) #finds the index of the li
                        index_of_max.append(name_list[list_]) #adds the name to
                        name_list.remove(name_list[list_])

            #n = n + 1

    if name_list == []:
        #print(index_of_max)
        out.insert(pos, index_of_max)
        break

    else:
        index_of_max.append(name_list[n])
```

```
            name_list.remove(name_list[n])
            out.insert(pos, index_of_max)

        #print(index_of_max)

    print(len(out))
    print(out)

    return out
```

Code analysis

In the first few lines, the code is importing a library called csv, this library helps me open the csv file. The next few lines (i.e., lines 9-14), opens the csv file and puts in into an array caled "data". Then after, it defines a variable name called name_list as an aray. After this, it is looping thorugh the data, ajd getting the value of the data (i.e., the names) in the ith position of the 0th value which shuold be the name. Then it appends (i.e, adds) the names onto the variables. After adding the names, it deletes the names off the origiinal vairabte i.e., data. Then the code defines "remove_data" as as a list comprehention tyhat converts the string to an intiger of the data. After this, it defines data_of_max as an empty array. In the nex for loop)line 34) it is finding the max value (ie.e, 10) of the data and adding it to the vairabe data_of_max, the index not the value. i.e., the indexing of the position of the max value - which is the number 10.

The main part of my code: firstly, it defines out as an empy array - this is what I would be ouputting. Then is defines pos as 0 and n as 0. In the while loop, it is looping while n is less than 2. In this loop, it defines iondex_of_max as an empy array, this value of the array keeps on resetting everytime after this looop. In the next two for loops, it is looping through the array of the array (i.e., data of max) and the value inside the array of array is 'y' and comparing 'y' to 'n'. Then, if the name list is empty, it breaks the loop. If the lengh of the name list is less than or equal to 5. It addes index of max to name list and defines index of max to name list and sets name list as a emptpy array, then finlly breaks the array. If none of the conditions meet - i.e., else - it puts the values in another while loop, to ensure that there s only 5 people per tent. It defines list_ as the data of the max and index x relative to y. Then it adds the name list position of list_ to index of max and finally, removes the value of name list posotion of list_ from the oringla array i.e., name lsit. Then finally, it adds the names into out and the person whos checking it with and returns the value.

A disadvantage of this method it ath it does not consider it both way - i.e., if the person does not like them back, thus, not giving that well of an happioness score. In contrast, an advantage is taht it is fast. On my good computer (personal computer that does not use the bad school one) it runs 50 names in 0.33 seconds and on the school laptop it runs double the time, both in non optinal areas due to having a lot of apps open and not restarting my laotop prior. I beleive this method could get a sub 0.1 second effiecney if it ran on a supercomputer, but currently, due to the limitation of mankinds computers, it is impossible to fathem such possiblities, thus, my comptuer and laptop is a limitation of this program. A

further limitatino is it is pretty slow, but coulve be solved by the method above - a better computer.

Algorithm efficiency analysis

Time complexity is $O(n^2)$ Space is $O(n)$

Program output and conclusions

Ulimtaely, this program tries to compute an output by putting people who ranked eachother 10 together. My output for preferences.csv was:

```
[['Justin Myers', 'Patrick Jackson', 'Carlos White', 'Ronald Cooper',
'Corey Leach']

['Robert Brooks', 'Omar Collier', 'Marvin Walters', 'John Owens', 'Andrew
Wagner']

['Michael Morgan', 'Antonio King', 'Austin Gutierrez', 'Ryan Pollard',
'Jesse Flores']

['Joseph Campbell', 'Jesus Garcia', 'Matthew Johnson', 'Phillip Chapman',
'John Cervantes']

['Erik Lowe', 'Michael Scott', 'Dean Perry', 'Anthony Robertson', 'Stephen
Mendez']

['James Lopez', 'Troy Sanchez', 'Michael Harris', 'Phillip Harper', 'Jason
Rodgers']

['Jonathan Palmer', 'Michael Molina', 'Phillip Meyer', 'Kevin Raymond',
'Calvin Porter']

['Steven Smith', 'Bradley Griffin', 'David Clark', 'Jason Ayers', 'Aaron
Perkins']

['Luis Brown', 'Gregory Rice', 'Christopher Reyes', 'Derrick Rangel',
'Tyler Cook']

['Randall Gregory', 'Thomas Anderson', 'Bruce Pena', 'Adrian Martin',
'Charles Harvey']]
```

I thought by putting all the people who ranked 10 together, it would try to show the max happiness, but the program does not check both ways, i.e., it only rank the people who ranked the other person a 10 together.

Ultimately, this solution is efficient, put the happiness is not that good. A better solution would be maybe check for happiness in both directions. Sorry for the bad grammar and spelling mistakes in the report!

A report by Kelvin Liang