Decision analytics - Assignment-1

A. Identify the objects, attributes and predicates for the puzzle and create the decision variables in a CP-SAT model [3 points].

The predicates of the problem are the total number of possible variables required to solve the problem itself in the first task we have:

- Universities they are going to attend
- Courses they are going to study
- The country of origin

All of these operate on the same object domain i.e. names of the people

Each of these domains have specific attributes that a variable can take mentioned below,

Names = { Carol , Elisa , Oliver , Lucas }

Universities = { Cambridge, Oxford, Edinburgh, London }

Country of origin = { USA, SA , Canada , Australia }

Courses = { history , medicine , law , architecture }

B. For each of the seven sentences in the puzzle define the explicit constraints contained in the sentences and add them to the CP-SAT model [7 points]. Identify clearly, which sentence you are referring to.

Prior assumptions that we are making before we begin this problem :

- Every individual is going to a different university, belongs to a different country, studies a unique subject and has unique name
- Every individual is going to study a specific course in a specific university and has a specific country of origin

1. Notice the first statement **“one of them is from London**” is already contained into the above statement of each individual studying in a unique university.

So let’s see how we can implement the two conditions in OR tools

Condition 1 : each of them going to a different entity

```
# every name has a different entity
for i in range(4): #iterate through all the names
    for j in range(i+1,4): #iterate through all the names afterwards
        for k in range(4): #iterate through all the countries , universities , courses
            model.AddBoolOr([
                    name_origin_country[names[i]][origin_country[k]].Not(),
                    name_origin_country[names[j]][origin_country[k]].Not()])
            model.AddBoolOr([
                name_university[names[i]][universities[k]].Not(),
                            name_university[names[j]][universities[k]].Not()])
            model.AddBoolOr([
                name_course[names[i]][courses[k]].Not(),
                name_course[names[j]][courses[k]].Not()])
```

And now let's see how to assign exactly how to assign exactly one entity per name -

Condtion 2 : assign one entity per name

```
for name in names:
    # at least one entity per name
    variables = []
    for country in origin_country:
        variables.append(name_origin_country[name][country])
    model.AddBoolOr(variables)

    variables = []
    for college in universities:
        variables.append(name_university[name][college])
    model.AddBoolOr(variables)

    variables = []
    for course in courses:
        variables.append(name_course[name][course])
    model.AddBoolOr(variables)

    # max one entity per name
    for i in range(4):
        for j in range(i+1,4):
            model.AddBoolOr([
                    name_origin_country[name][origin_country[i]].Not(),
                    name_origin_country[name][origin_country[j]].Not()])
            model.AddBoolOr([
                    name_university[name][universities[i]].Not(),
                    name_university[name][universities[j]].Not()])
            model.AddBoolOr([
                    name_course[name][courses[i]].Not(),
                    name_course[name][courses[j]].Not()])
```

By assessing both the conditions we can safely conclude that one (and exactly one) of them is from London university.

2. **Exactly one boy and one girl chose a university in a city with the same initial of their names.**

The above statement denotes that one of the girl either Carol or Elisa goes to attend university of the same initial and one of the boy either Lucas or Oliver goes to attend the university with their same initial.

What does this mean ?

If the girl is Carol then she goes to Cambridge,

If the girl is Elisa then she goes to Edinburgh,

If the boy is Lucas then he goes to London,

If the boy is Oliver then he goes to Oxford,

Either of the two statements (exactly two) are True and not all of them,

Statement 1: This means Carol can only go to Cambridge if Elisa is not going to Edinburgh, and along side Oliver can go to Oxford if Lucas is not going to London.

Another possible set could be,

Statement 2: Elisa going to Edinburgh if Carol is not going to Cambridge and Oliver not going to Oxford if Lucas is going to London.

Notice statement 2 is actually the converse of the statement 1, where the output is not certain and is very much dependent on the second part of the statement.

```
#Exactly one boy and one girl chose a university in a city with the same initial of their names (2).

#carol is going to cambridge if elisa is not going to edinburgh
model.AddBoolOr([name_university[names[0]][universities[0]]]).OnlyEnforceIf([name_university[names[1]][universities[
#and oliver is going to Oxford if lucas is not going to london
model.AddBoolAnd([name_university[names[2]][universities[1]]]).OnlyEnforceIf([name_university[names[3]][universities
#carol is not going to cambridge if elisa is going to edinburgh
model.AddBoolOr([name_university[names[0]][universities[0]].Not()]).OnlyEnforceIf([name_university[names[1]][univers
# and oliver is not going to oxford if lucas is going to london
model.AddBoolAnd([name_university[names[2]][universities[1]].Not()]).OnlyEnforceIf([name_university[names[3]][univer
```

(Note- Please see list ending to get the exact order of items referred.)

3. A boy is from Australia, the other studies History.

We have two boys Oliver and Lucas,

Statement 1: If Lucas is from Australia then Oliver will study history and if Lucas is not from Australia then Lucas will study history.

Conversely,

Statement 2: If Oliver is from Australia then Lucas will study History and if Oliver is not from Australia then Oliver will study history.

Derived statement using 1 and 2: Elisa and carol cannot be from Australia or study history because one of the boy above is studying history and the other is from Australia. This means both the spots are taken by either of the two boys.

```
#A boy is from Australia, the other studies History (3).

#lucas is from austrailia if  olver studies history
model.AddBoolOr([name_origin_country[names[3]][origin_country[3]]]).OnlyEnforceIf([name_course[names[2]][courses[0
#oliver is from austrailia if lucas studies history
model.AddBoolOr([name_origin_country[names[2]][origin_country[3]]]).OnlyEnforceIf([name_course[names[3]][courses[0
#Elisa can't be from austraila or study history, carol can't be from austrailia or study history
model.AddBoolAnd([name_origin_country[names[1]][origin_country[3]].Not(),name_course[names[0]][courses[0]].Not(),
                  name_origin_country[names[0]][origin_country[3]].Not(),name_course[names[1]][courses[0]].Not()])
```

4. A girl goes to Cambridge, the other studies Medicine.

Statement 1: If Carol is going to Cambridge then Elisa will study medicine and if Carol is not going to Cambridge then Carol will study medicine.

Conversely,

Statement 2: If Elisa is going to Cambridge then Carol will study medicine and if Elisa is not going to Cambridge then Elisa will study medicine.

Derived statement using 1 and 2: Oliver and Lucas cannot be going to Cambridge or study medicine because one of the girl above is studying medicine and the other has to go to Cambridge. This means both the spots are taken by either of the two girls.

```
#A girl goes to Cambridge, the other studies Medicine (4).

#carol goes to cambridge if elisa studies medicine
model.AddBoolOr([name_university[names[0]][universities[0]]]).OnlyEnforceIf([name_course[names[1]][courses[1]]])
#elisa goes to cambridge if carol studies medicine
model.AddBoolOr([name_university[names[1]][universities[0]]]).OnlyEnforceIf([name_course[names[0]][courses[1]]])
#oliver cant go to cambridge and he can't study medicine
#lucas cant go to cambridge and he can't study medicine
model.AddBoolAnd([name_university[names[2]][universities[0]].Not(),name_course[names[2]][courses[1]].Not(),
                  name_university[names[3]][universities[0]].Not(),name_course[names[3]][courses[1]].Not()])
```

5. Oliver studies Law or is from USA; He is not from South Africa.

Statement 1: If Oliver studies law , he can't be from USA

Conversely,

Statement 2: If Oliver is not studying law then he is from USA.

And one more solid fact we have.

Statement 3: Oliver can't be form South Africa.

```
In [51]: #Oliver studies Law or is from USA; He is not from South Africa (5).
         #oliver studies Law if he is not from usa
         model.AddBoolOr([name_course[names[2]][courses[2]]]).OnlyEnforceIf([name_origin_country[names[2]][origin_country[0]]
         #oliver does not study law if he is from US
         model.AddBoolOr([name_course[names[2]][courses[2]].Not()]).OnlyEnforceIf([name_origin_country[names[2]][origin_count
         #oliver is not from south africa for sure
         model.AddBoolAnd([name_origin_country[names[2]][origin_country[1]].Not()])

Out[51]: <ortools.sat.python.cp_model.Constraint at 0x1142456d0>
```

6. The student from Canada is a historian or will go to Oxford

This student can be apparently anyone hence we traverse through all the names that we have and set this possibility,

This student can study history or can go to Oxford if he is from Canada.

```
]: #The student from Canada is a historian or will go to Oxford (6)
   # a student in the list of student
   for name in names:
       #this student can study history or go to oxford if he is from Canada
       model.AddBoolOr([name_course[name][courses[0]],name_university[name][universities[1]]])
       .OnlyEnforceIf([name_origin_country[name][origin_country[2]]])
```

7. The student is from South Africa is going to Edinburgh or will study Law

This student can be apparently anyone hence we traverse through all the names that we have and set this possibility,

This student can go to Edinburgh or can study law if he is from South Africa.

```
#The student from South Africa is going to Edinburgh or will study Law (7).
for name in names:
    model.AddBoolOr([name_course[name][courses[2]],name_university[name][universities[2]]])
    .OnlyEnforceIf([name_origin_country[name][origin_country[1]]])
```

C. The puzzle also contains some implicit constraints. Define and implement these implicit constraints in the CP-SAT model [3 points].

The implicit condition means all the conditions that are already contained in the statements or the prior assumptions made before solving the problem

1. Each of the names attend different course, have a different origin country, and goes to a different university
2. Each individual can select a maximum of one possibility of the selections possible
3. Each individual has to select one of the prior possibility of the selections available implying every candidate has to go to a university select a course and has a unique origin country
4. We base our prior assumptions for the gender names.

D. Solve the CP-SAT model and determine for each student the nationality, the university and the course they chose [3 points].

```
solver = cp_model.CpSolver()
solver.SearchForAllSolutions(model, SolutionPrinter(name_university, name_origin_country, name_course,name_gender))
```

```
solution 1
 - Carol:
origin country :  SA
college going :  Cambridge
course studying:  law
gender of the individual:  girl
 - Elisa:
origin country :  Canada
college going :  Oxford
course studying:  medicine
gender of the individual:  girl
 - Oliver:
origin country :  USA
college going :  Edinburgh
course studying:  history
gender of the individual:  boy
 - Lucas:
origin country :  Australia
college going :  London
course studying:  architecture
gender of the individual:  boy
```

The above screenshot is obtained from the execution of the code that describes each of the individual's identity with respect to the country, course and university they are applying to study.

E. Evaluate for each of the seven sentences in the puzzle if they are redundant. Which sentences can be omitted from the puzzle and why? [7 points]

1. Exactly one of them is going to London : This statement is already contained in the previous statement where we said each of them is going to at most one university and a minimum of one university.

2. Exactly one boy and one girl chose a university in a city with the same initial of their names : This means that the girl could be either Carol or Elisa and on the converse the guy could be Lucas or Oliver.

S1: Carol is going to Cambridge, if Elisa is not going to Edinburgh.
S2: Oliver is going to Oxford if Lucas is not going to London.
S3: Carol is not going to Cambridge if Elisa is going to Edinburgh.
S4: Oliver is not going to Oxford if Lucas is going to London.

S3 and S4 are the converse statements of the S1 and S2 meaning the second has to be True if the First is already True.

(we can individually remove each of the two statements as when we remove both of them we get two solutions)

3. A boy is from Australia, the other studies History:

S1: Lucas is from Australia if Oliver studies history
S2: Oliver is from Australia if Lucas studies history

And implication of S1 and S2

S3: Elisa can't be from Australia or study history, carol can't be from Australia or study history

(None of them is a redundant statement)

4. A girl goes to Cambridge, the other studies Medicine

S1: Carol goes to Cambridge if Elisa studies medicine
S2: Elisa goes to Cambridge if carol studies medicine

And implication of S1 and S2

S3: Lucas cant go to Cambridge and he can't study medicine, Oliver cant go to Cambridge and he can't study medicine

(both S1 and S2 can be removed alternatively without impacting the final solution)

5. Oliver studies Law or is from USA; He is not from South Africa

S1: Oliver studies Law if he is not from USA
S2: Oliver does not study law if he is from US

And S3 has no link to S1 and S2

S3: Oliver is not from south Africa for sure

(S1 and S2 are complement of each other while S3 is an absolute condition)

(S3 is a redundant statement meaning it can be removed without hurting the the total number of solutions similarly S2 can also be removed)

6. The student from Canada is a historian or will go to Oxford

(Not a redundant statement)

7. The student from South Africa is going to Edinburgh or will study Law

(Not a redundant statement)

Task 2 - We are given 4 specific sheets each denoting different information in each,

Projects | Quotes | Dependencies | Value

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
Project A	Job A	Job B	Job C									
Project B			Job G	Job K	Job M							
Project C				Job H	Job E	Job G	Job B	Job E				
Project D		Job D	Job F	Job I	Job H							
Project E								Job J	Job A			
Project F									Job B	Job C	Job K	
Project G					Job L	Job M	Job E					
Project H								Job A	Job B	Job D	Job I	
Project I										Job L	Job F	Job K

This sheet indicates which project requires which Jobs in which Month.

	Job A	Job B	Job C	Job D	Job E	Job F	Job G	Job H	Job I	Job J	Job K	Job L	Job M
Contractor A	230				30				80			200	
Contractor B											140		290
Contractor C							550			190			
Contractor D				450								190	
Contractor E		130			20								
Contractor F			200			140							
Contractor G							505		60				
Contractor H	300			400				120					
Contractor I							600						
Contractor J		150				170				240			
Contractor K			150					140			150		300

The Contractors can each do a different job at a different price

	Project A	Project B	Project C	Project D	Project E	Project F	Project G	Project H	Project I
Project A									
Project B	x								
Project C									
Project D									
Project E			x	x					
Project F							x		
Project G		x							
Project H									
Project I					x				

Each of these projects are dependent upon some other projects and cannot be began if the preliminary projects are not complete

	Value
Project A	500
Project B	300
Project C	400
Project D	1000
Project E	2000
Project F	100
Project G	1500
Project H	1000
Project I	1000

Each of the project has a particular budget assigned to it and has to be done under the budget.

- Load the excel file **Assignment_DA_1_data.xlsx** and extract all relevant information [1 point]. Make sure to use the data from the file in your code, please do not hardcode any values that can be read from the file.

We load the data using the pandas Dataframe and then we select each of the 4 sheets available into a data frame,

```
total_projects=shape_df1[0]
total_months=shape_df1[1]
total_contractors=shape_df2[0]
total_jobs=shape_df2[1]

all_projects = range(total_projects)
all_months = range(1,total_months)
all_contractors = range(total_contractors)
all_jobs = range(1,total_jobs)
```

We assign particular names to particular variables such as the projects range from 0 to 8.

We assign months that range from 1 to 12.

We assign contractors that range from 0 to 11

We assign jobs that range from 0 to 11.

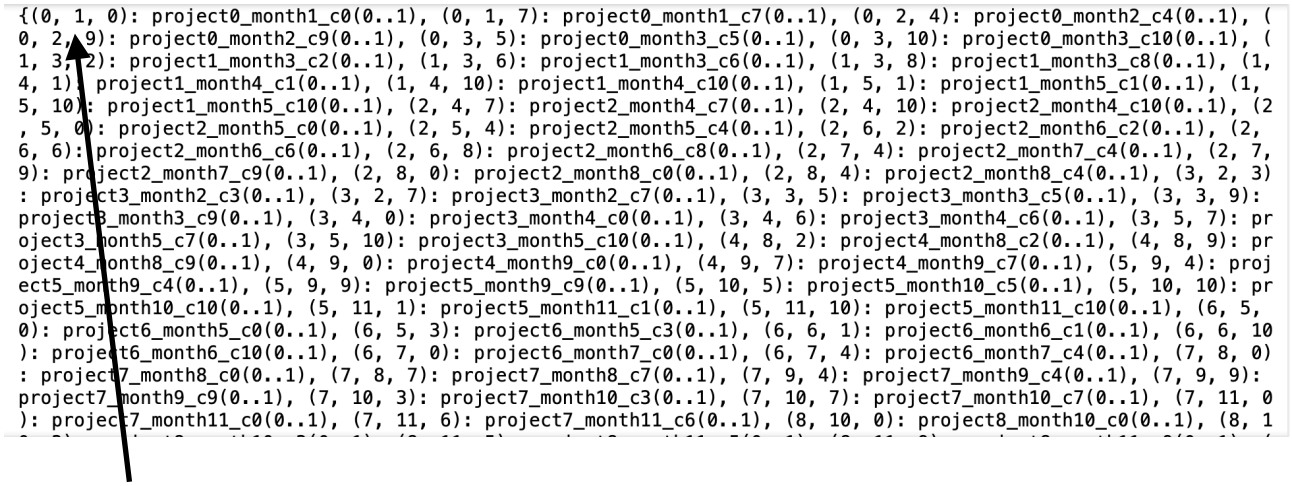
Now we initialise the model and assign all the variables these variables denote all the possible case that can happen (without any constraints) this means for every project we will see for every month, for those months that have jobs already assigned we will see all the possible contractors that can take up that job.

Notice two contractors can still work on the same job and at the same time we will later filter these conditions.

To do so we will use a get all contractors function given a job type an individual will get all the possible contractors ready to work before hand as given in the second sheet or df2

```
model = cp_model.CpModel()

def get_all_contractors(job_name):
    """takes the name of the job and returns all the possible contractors available"""
    contractors=list(df2[job_name])
    avl_contractors=[]
    for i,element in enumerate(contractors):
        if(element !=0):
            avl_contractors.append(i)

    return avl_contractors

avl_contractors=get_all_contractors("Job A")


#Assign sheet 1
projects_months_contractors = {} #contractors available
for p in all_projects:
    for m in all_months:
            #select possible job blocks
            if((p,m) in job_names ):
                #assign only those jobs that are possible
                get_job=job_names[(p,m)] #this is a text
                j=jobs_to_num[get_job] #this is a number
                #get contractors list possible for this job
                allpossible_contractors=get_all_contractors(get_job)
                for c in allpossible_contractors:
                    projects_months_contractors[(p,m,c)]= model.NewBoolVar('project%i_month%i_c%i' % (p, m,c))

# equals 1 if job j is assigned to contractor c on month m, and 0 if the task is not assigned
print(projects_months_contractors)
```

Let's see how our variable look like



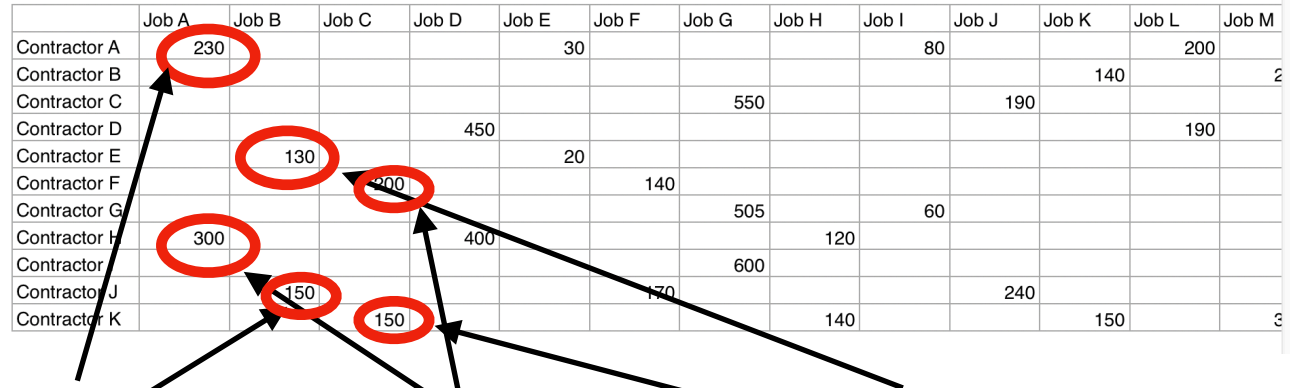
This denotes the index of the variable meaning the project 0 .ie Project A is done in month 1 i.e M1 is done by contractor 0 i.e Contractor A

(Note-The month indexing starts from 1 and not 0)

Let's verify this from the table, and see for other values if they exist.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
Project A	Job A	Job B	Job C									
Project B				Job K	Job M							
Project C				Job H	Job E	Job G	Job B	Job E				
Project D		Job D	Job F	Job I	Job H							
Project E								Job J	Job A			
Project F									Job B	Job C	Job K	
Project G					Job L	Job M	Job E					
Project H								Job A	Job B	Job D	Job I	
Project I										Job L	Job F	Job K

	Job A	Job B	Job C	Job D	Job E	Job F	Job G	Job H	Job I	Job J	Job K	Job L	Job M
Contractor A	230				30				80			200	
Contractor B											140		2
Contractor C							550			190			
Contractor D				450								190	
Contractor E		130			20								
Contractor F			200			140							
Contractor G							505		60				
Contractor H	300			400				120					
Contractor							600						
Contractor J		150				170				240			
Contractor K			150					140			150		3



{(0, 1, 0): project0_month1_c0(0..1), (0, 1, 7): project0_month1_c7(0..1), (0, 2, 4): project0_month2_c4(0..1), (
0, 2, 9): project0_month2_c9(0..1), (0, 3, 5): project0_month3_c5(0..1), (0, 3, 10): project0_month3_c10(0..1), (
1, 3, 2): project1_month3_c2(0..1), (1, 3, 6): project1_month3_c6(0..1), (1, 3, 8): project1_month3_c8(0..1), (1,
4, 1): project1_month4_c1(0..1), (1, 4, 10): project1_month4_c10(0..1), (1, 5, 1): project1_month5_c1(0..1), (1,
5, 10): project1_month5_c10(0..1), (2, 4, 7): project2_month4_c7(0..1), (2, 4, 10): project2_month4_c10(0..1), (2
, 5, 0): project2_month5_c0(0..1), (2, 5, 4): project2_month5_c4(0..1), (2, 6, 2): project2_month6_c2(0..1), (2,
6, 6): project2_month6_c6(0..1), (2, 6, 8): project2_month6_c8(0..1), (2, 7, 4): project2_month7_c4(0..1), (2, 7,
9): project2_month7_c9(0..1), (2, 8, 0): project2_month8_c0(0..1), (2, 8, 4): project2_month8_c4(0..1), (3, 2, 3)
: project3_month2_c3(0..1), (3, 2, 7): project3_month2_c7(0..1), (3, 3, 5): project3_month3_c5(0..1), (3, 3, 9):
project3_month3_c9(0..1), (3, 4, 0): project3_month4_c0(0..1), (3, 4, 6): project3_month4_c6(0..1), (3, 5, 7): pr
oject3_month5_c7(0..1), (3, 5, 10): project3_month5_c10(0..1), (4, 8, 2): project4_month8_c2(0..1), (4, 8, 9): pr
oject4_month8_c9(0..1), (4, 9, 0): project4_month9_c0(0..1), (4, 9, 7): project4_month9_c7(0..1), (5, 9, 4): proj
ect5_month9_c4(0..1), (5, 9, 9): project5_month9_c9(0..1), (5, 10, 5): project5_month10_c5(0..1), (5, 10, 10): pr
oject5_month10_c10(0..1), (5, 11, 1): project5_month11_c1(0..1), (5, 11, 10): project5_month11_c10(0..1), (6, 5,
0): project6_month5_c0(0..1), (6, 5, 3): project6_month5_c3(0..1), (6, 6, 1): project6_month6_c1(0..1), (6, 6, 10
): project6_month6_c10(0..1), (6, 7, 0): project6_month7_c0(0..1), (6, 7, 4): project6_month7_c4(0..1), (7, 8, 0)
: project7_month8_c0(0..1), (7, 8, 7): project7_month8_c7(0..1), (7, 9, 4): project7_month9_c4(0..1), (7, 9, 9):
project7_month9_c9(0..1), (7, 10, 3): project7_month10_c3(0..1), (7, 10, 7): project7_month10_c7(0..1), (7, 11, 0
): project7_month11_c0(0..1), (7, 11, 6): project7_month11_c6(0..1), (8, 10, 0): project8_month10_c0(0..1), (8, 1

As we can see that we are getting our variables formed and can take a boolean value assignment that is this can happen or not depending upon the constraints that we will look later on.

- Define and implement the constraint that a contractor cannot work on two projects simultaneously [3 points].

Now we try to set up our first constraint, that is, **no contractor can work on two projects simultaneously**, let's see how we will set up our constraints.

```
for p in all_projects: #traverse through all projects
    for m in all_months: #traverse through all months
        temp_list=[] #make an empty list which will be later used to assign conditions
        for element in projects_months_contractors: #check if the variable already exist in the (p,m,c) variables
            if (element[0] == p and element[1] == m): #check for the first two elements
                #not all the values will be selected as the index wont match for the blank spots
                temp_list.append(projects_months_contractors[element]) #if the two match with the same value then
                #we can add this value to the constraint
        model.Add(sum(temp_list)<=1)
```

This means that we can have first two elements same but not the third that defines the contractor So we can't have (0,1,0) and (0,1,7) at the same time because as we said no contractor can together work but we can have (0,1,0) and (1,0,7) this means two contractors are working on different project at the same time.

- Define and implement the constraint that if a project is accepted to be delivered then exactly one contractor per job of the project needs to work on it [4 points].

We then set up our second constraint that is we try to set job limit that It could be accepted or denied depending upon the availability of contractor and price(which we will later look at)

```
#each job has to be done by a contractor which can select or not select
for p in all_projects:
    for m in all_months:
        if((p,m) in job_names): #if there exists a job
            get_job=job_names[(p,m)] #this is a text
            j=jobs_to_num[get_job] #this is a number
            allpossible_contractors=get_all_contractors(get_job)
            model.Add(sum(projects_months[(p,m,c)] for c in allpossible_contractors ) <= 1)
```

As we can see in that for every possible combination applicable the occurrence can be either a yes or a no condition meaning the max size could be 1 implying if the job is selected and 0 if it is not selected.

Each project is associated with a certain cost meaning that we need to stay under the budget,

```
#project cost with budget

for p in all_projects:
    total_price=[]
    for m in all_months:
        if((p,m) in job_names): #if there exists a job
            get_job=job_names[(p,m)] #this is a text
            j=jobs_to_num[get_job] #this is a number
            allpossible_contractors=get_all_contractors(get_job)
            #get the price for all the possible contractors
            current=[]
            for c in allpossible_contractors:
                price=df2.iloc[c,j+1]
                current.append((price,c))
            #select the minimum of the possible prices for
            #print(current)
            selected=min(current)
            total_price.append(selected)
    fprice=0
    contractor_names=[]
    for element in total_price:
        fprice+=element[0]
        contractor_names.append(element[1])
    print(fprice,contractor_names)
    if(fprice<df4.iloc[p,1]):
        print("valid solution with savings: {}".format(df4.iloc[p,1]-fprice))
    else:
        print("not valid solution at all")
```

As we can see in the screenshot below we see some of the solutions are valid while others are not

```
510.0 [0, 4, 10]
not valid solution at all
935.0 [6, 1, 1]
not valid solution at all
795.0 [7, 4, 6, 4, 4]
not valid solution at all
720.0 [7, 5, 6, 7]
valid solution with savings: 280.0
420.0 [2, 0]
valid solution with savings: 1580.0
420.0 [4, 10, 1]
not valid solution at all
500.0 [3, 1, 4]
valid solution with savings: 1000.0
820.0 [0, 4, 7, 6]
valid solution with savings: 180.0
470.0 [3, 5, 1]
valid solution with savings: 530.0
```