Universidad Nacional Autonoma de México Facultad de Ingeniería

Inteligencia Artificial

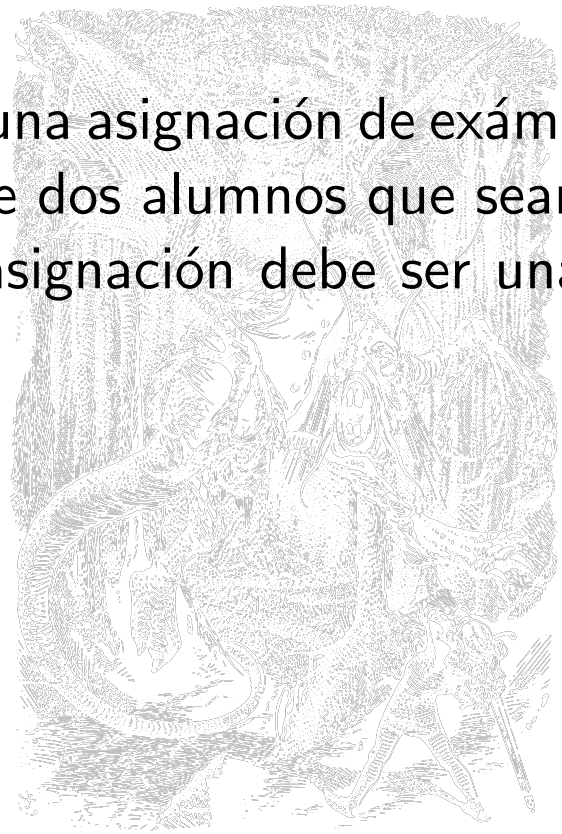
Proyecto 1

Saldaña Hernandez Norman Brandon
Guerrero Ferrusca Francisco
González Lorenzo Daniela Michel
Tenorio Veloz Alisson Dafne

September 28, 2021

Planteamiento del problema

Diseñar un agente inteligente que encuentre una asignación de exámenes dentro de una red social de alumnos, tal que dos alumnos que sean amigos no tengan un mismo examen y la asignación debe ser una de menor número de exámenes.



Solución basada en busqueda

La solución que se plantea es una con las siguientes características

- Estados (Nodos): Asignaciones de examen a cada alumno (cumplan o no las restricciones).
- Estado Inicial: Asignación vacía ['Juan':0,'Jorge':0,'Diana':0]
- Función sucesor: Asignar un examen al alumno siguiente.
- Prueba meta: Asignar todos los examenes y probar que la asignación cumpla las restricciones.
- Costo: Una unidad por examen distinto.
- Problema de entrada: El problema será definido con un diccionario, donde la llave es el nombre del alumno y a cada alumno se le asocia una lista con los nombres de sus amigos.
- Solucion de salida: la asignación con menor coste

Búsqueda en profundidad

La forma más intuitiva desde nuestro planteamiento fue búsqueda en profundidad

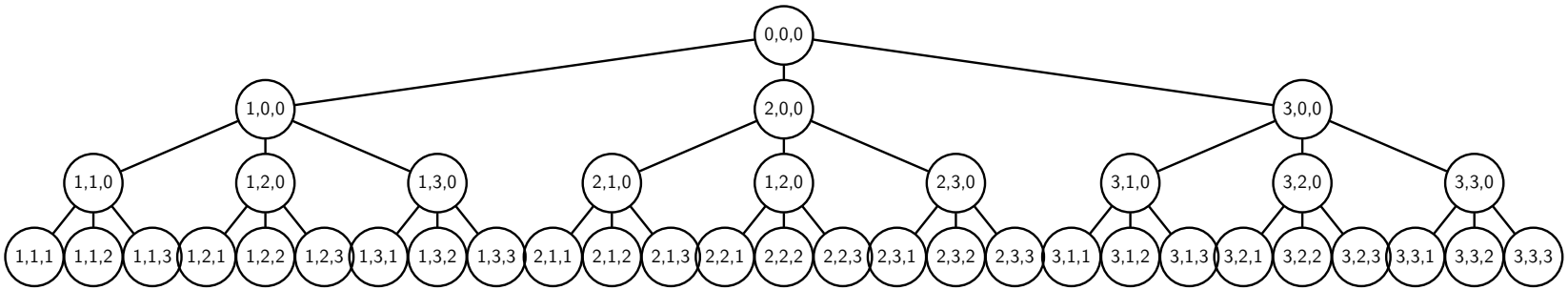
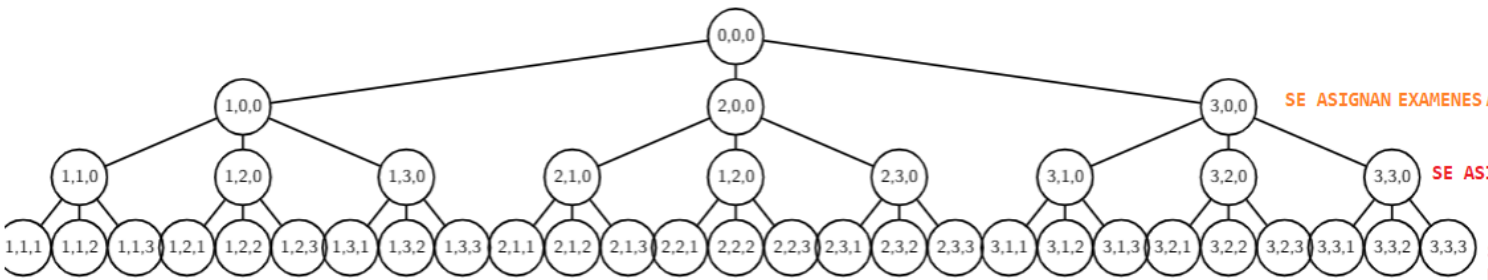


Figure: Espacio de estados

el número de nodos hoja es N^N donde N es el número de alumnos, para 3 alumnos es $3^3 = 27$

Considerando únicamente 3 alumnos: Juan Jorge y Diana



Obtención de nodos hoja para el problema de 3 alumnos

El pseudocódigo que planteamos fue el siguiente, nótese que recorre todo el árbol de estados.

```
#Obtencion de todas las asignaciones completas para el problema de 3 alumnos
for examen_alumno1 en lista_examenes
    Asignar examen_alumno1  a alumno1
    for examen_alumno2 en lista_examenes
        Asignar examen_alumno2  a alumno2
        for examen_alumno3 en lista_examenes
            Asignar examen_alumno3  a alumno3
```

Codificación para 3 alumnos

Antes de implementar el algoritmo hay que definir variables

```
#Declaracion de variables y del problema
mejorAsig ={}
examenes=[] # [1,2,3,4,5,.., N]
Asignacion = {}
Problema ={
    'Juan':['Jorge','Diana'],
    'Jorge':['Juan'],
    'Diana':['Juan'],
}


Alumnos = list(Problema.keys())#Hacemos una lista con los alumnos
i=0
for alumno in Problema.keys(): #inicializamos el diccionario de asignaciones con los alumnos
    Asignacion[alumno] = 0
    i = i +1
    examenes.append(i) #llenamos la lista de examenes

costmin =len(examenes)

GetAsig(0)
```

Codificación para 3 alumnos, ciclos for anidados

Imprime solo los nodos hoja. O asignaciones completas

```
for i in examenes: #iteramos 3 veces; i itera en la lista [1 2 3]
    Asignacion['Juan'] = i #asignamos el valor de i a Juan en el diccionario de asignacion
    for j in examenes: #iteramos 3 veces; j itera en la lista [1 2 3]
        Asignacion['Jorge'] = j #asignamos el valor de j a Juan en el diccionario de asignacion
        for k in examenes: # iteramos 3 veces; k itera en la lista [1 2 3]
            Asignacion['Diana'] = k # asignamos el valor de k a Juan en el diccionario de asignacion
            print(Asignacion) # imprime la asignación Final
        print()
```

Output del código para 3 alumnos

```
{'Juan': 0, 'Jorge': 0, 'Diana': 0}
{'Juan': 1, 'Jorge': 1, 'Diana': 1}
{'Juan': 1, 'Jorge': 1, 'Diana': 2}
{'Juan': 1, 'Jorge': 1, 'Diana': 3}

{'Juan': 1, 'Jorge': 2, 'Diana': 1}
{'Juan': 1, 'Jorge': 2, 'Diana': 2}
{'Juan': 1, 'Jorge': 2, 'Diana': 3}

{'Juan': 1, 'Jorge': 3, 'Diana': 1}
{'Juan': 1, 'Jorge': 3, 'Diana': 2}
{'Juan': 1, 'Jorge': 3, 'Diana': 3}

{'Juan': 2, 'Jorge': 1, 'Diana': 1}
{'Juan': 2, 'Jorge': 1, 'Diana': 2}
{'Juan': 2, 'Jorge': 1, 'Diana': 3}

{'Juan': 2, 'Jorge': 2, 'Diana': 1}
{'Juan': 2, 'Jorge': 2, 'Diana': 2}
{'Juan': 2, 'Jorge': 2, 'Diana': 3}

{'Juan': 2, 'Jorge': 3, 'Diana': 1}
{'Juan': 2, 'Jorge': 3, 'Diana': 2}
{'Juan': 2, 'Jorge': 3, 'Diana': 3}

{'Juan': 3, 'Jorge': 1, 'Diana': 1}
{'Juan': 3, 'Jorge': 1, 'Diana': 2}
{'Juan': 3, 'Jorge': 1, 'Diana': 3}

{'Juan': 3, 'Jorge': 2, 'Diana': 1}
{'Juan': 3, 'Jorge': 2, 'Diana': 2}
{'Juan': 3, 'Jorge': 2, 'Diana': 3}

{'Juan': 3, 'Jorge': 3, 'Diana': 1}
{'Juan': 3, 'Jorge': 3, 'Diana': 2}
{'Juan': 3, 'Jorge': 3, 'Diana': 3}
```

Generalizando a N alumnos

Una funcion recursiva que llame a otros ciclos for parece buena idea

Hace lo mismo que el pseudocódigo anterior pero para N alumnos, se empieza con profundidad 0

```
Funcion_Recursiva(profundidad)
    if profundidad < numero_de_alumnos:
        for examen en lista_de_examenes
            Asignacion(alumno en profundidad actual) <- examen
            Funcion_Recursiva(profundidad + 1)
    else
        Imprimir asignacion
```

Codificación

El output sería el mismo que el anterior para los mismos 3 alumnos

Esta función se llama con : getAsig(0)

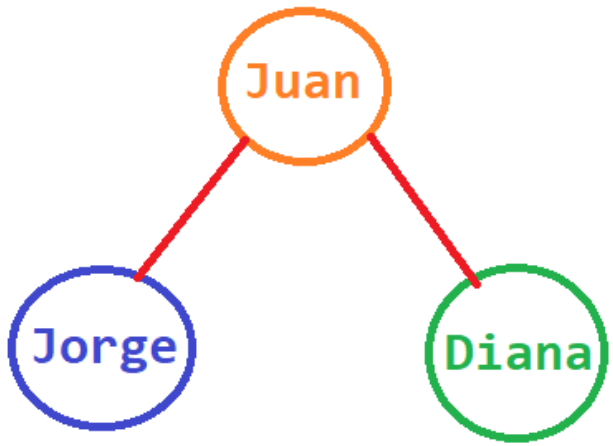
```
def GetAsig(prof):# Funcion recursiva búsqueda en profundidad
    if prof < len(Alumnos):# si la profundidad del árbol de búsqueda aún no es la máxima (no hemos lleg
        for i in examenes: # iterando en los valores de los posibles exámenes [1 2 3]
            Asignacion[Alumnos[prof]] = i #se le asigna el valor de el examen i al alumno de profundida
            GetAsig(prof + 1)  # Se llama a esta misma funcion para bajar en profundidad, sumando profu
    else:                   # si la profundidad es máxima tenemos una asignación completa (se ha asigna
        print(Asignacion) # Imprimimos la asignación completa
```

Evaluación de Restricciones

No todas las asignaciones son válidas

En la definición del problema, ademas de definirse el número de alumnos y sus nombres, tambien se definen las restricciones entre ellos

```
Problema ={
    'Juan':['Jorge','Diana'],
    'Jorge':['Juan'],
    'Diana':['Juan'],
}
```



Evaluación de Restricciones

En python se vería así

Devuelve 1 si es válida, 0 si no lo es

```
def IsValid(Asig):
    for alumno in Alumnos:
        for friend in Problema.get(alumno):
            if Asig.get(friend) == Asig.get(alumno):
                return 0
    return 1
```

Filtrando las asignaciones completas

Combinamos la funcion IsValid() con getAsig()

El programa completo debe tener las variables declaras e inicializadas, ademas de llamar a la funcion recursiva : getAsig(0)

```
#Esta funcion combina el programa anterior agregando una prueba de validez de asignación
# es lo mismo que IA3.py pero para n alumnos
def IsValid(Asig):
    for alumno in Alumnos:
        for friend in Problema.get(alumno):
            if Asig.get(friend) == Asig.get(alumno):
                return 0
    return 1

def GetAsig(prof):
    if prof < len(Alumnos):
        for i in examenes:
            Asignacion[Alumnos[prof]] = i
            GetAsig(prof +1)
    else:
        if IsValid(Asignacion):# Si la asignación es válida la imprime
            print(Asignacion)
```

Output del programa anterior

Nos da los nodos hoja filtrados, solo imprime los válidos

```
{'Juan': 1, 'Jorge': 2, 'Diana': 2}
{'Juan': 1, 'Jorge': 2, 'Diana': 3}

{'Juan': 1, 'Jorge': 3, 'Diana': 2}
{'Juan': 1, 'Jorge': 3, 'Diana': 3}

{'Juan': 2, 'Jorge': 1, 'Diana': 1}
{'Juan': 2, 'Jorge': 1, 'Diana': 3}


{'Juan': 2, 'Jorge': 3, 'Diana': 1}
{'Juan': 2, 'Jorge': 3, 'Diana': 3}

{'Juan': 3, 'Jorge': 1, 'Diana': 1}
{'Juan': 3, 'Jorge': 1, 'Diana': 2}

{'Juan': 3, 'Jorge': 2, 'Diana': 1}
{'Juan': 3, 'Jorge': 2, 'Diana': 2}
```

Función de coste

Para seleccionar la mejor asigación(la menos costosa), hay que definir una función de coste

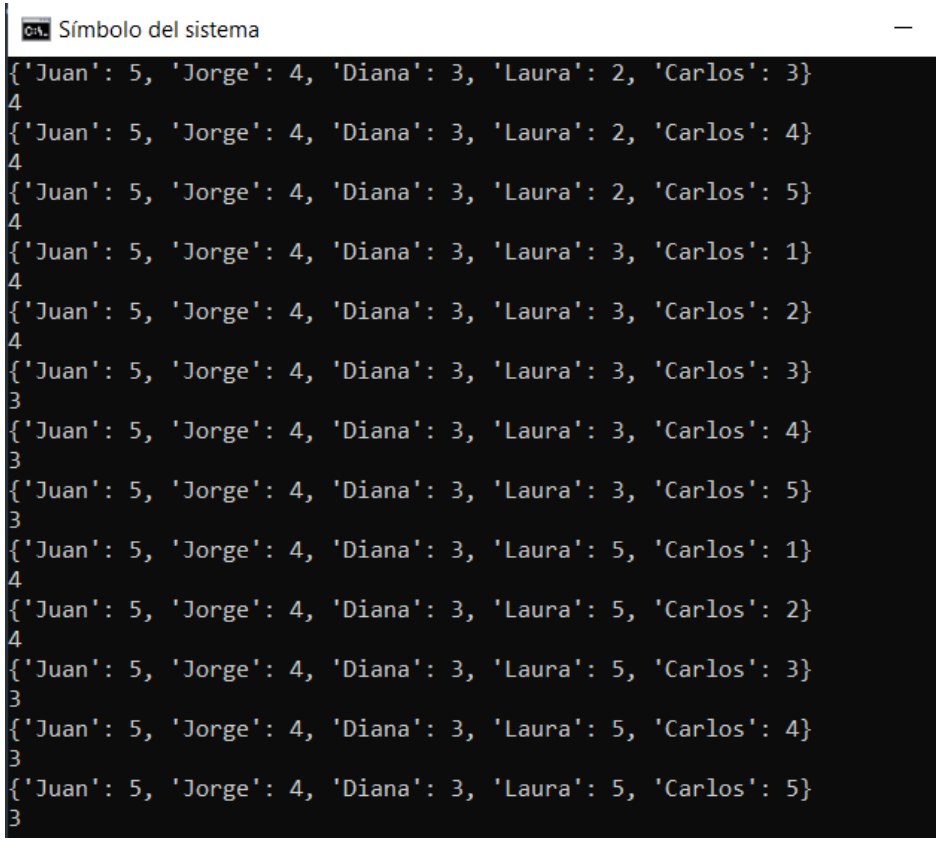
Devuelve el coste de la función, por ejemplo para {'Juan':1,'Jorge':1,'Diana':2} tiene coste 2

```
def cost(Asig):
    exdif =[] #lista vacia llamada examenes distintos abreviado
    for alumno in Alumnos:
        if Asig.get(alumno) not in exdif: #si el examen asignado al alumno N no esta en la lista, agr
            exdif.append(Asig.get(alumno))

    costo = len(exdif)#exdif contendra solo examenes distintos, no repetidos, por lo tanto su longitu
    return costo
```

Imprimiendo nodos hoja para 5 alumnos

Este es el output imprimiendo el coste de las asignaciones validas



Solo nos queda buscar la mejor de las mejores: Variables auxiliares

Para seleccionar la de menos coste entre todas las asignaciones válidas

costmin, debe inicializarse con un valor muy alto, aunque el valor mas alto que puede tener el coste es igual a N

```
def getCostMin():
    global costmin
    return costmin
def setCostmin(value):
    global costmin
    costmin = value
def getBestAsig():
    global mejorAsig
    return mejorAsig
def setBestAsig(value):
    global mejorAsig
    mejorAsig = value
```

Combinación de códigos, a unir el rompecabezas

si el costo de la asignacion es menor al ultimo menor coste, esta es la nueva ganadora y asi sucesivamente

Al terminar la funcion recursiva tendremos la de menor coste

```
def GetAsig(prof):
    if prof < len(Alumnos):
        for i in examenes:
            Asignacion[Alumnos[prof]] = i
            GetAsig(prof +1)
    else:
        if IsValid(Asignacion):
            if  cost(Asignacion) < getCostMin():
                setCostmin(cost(Asignacion))
                setBestAsig(Asignacion)
                print("Una asignación optima es: "+str(getBestAsig()))
                print("Su costo : " + str(cost(getBestAsig())))
```

El código completo, parte 1:

"Ya dame la fórmula"

```
1  def cost(Asig):
2      numexa =[]
3      for alumno in Alumnos:
4          if Asig.get(alumno) not in numexa:
5              numexa.append(Asig.get(alumno))
6      costo = len(numexa)
7      return costo
8
9  def IsValid(Asig):
10     for alumno in Alumnos:
11         for friend in Problema.get(alumno):
12             if Asig.get(friend) == Asig.get(alumno):
13                 return 0
14     return 1
15
16 def getCostMin():
17     global costmin
18     return costmin
19 def setCostmin(value):
20     global costmin
21     costmin = value
22 def getBestAsig():
23     global mejorAsig
24     return mejorAsig
25 def setBestAsig(value):
26     global mejorAsig
27     mejorAsig = value
28
29 def GetAsig(prof):
30     if prof < len(Alumnos):
31         for i in examenes:
32             Asignacion[Alumnos[prof]] = i
33             GetAsig(prof +1)
34     else:
35         if IsValid(Asignacion):
36             if  cost(Asignacion) < getCostMin():
```

El código completo, parte 2:

"Ya dame la fórmula"

```
31          for i in examenes:
32              Asignacion[Alumnos[prof]] = i
33              GetAsig(prof +1)
34      else:
35          if IsValid(Asignacion):
36              if  cost(Asignacion) < getCostMin():
37                  setCostmin(cost(Asignacion))
38                  setBestAsig(Asignacion)
39                  print("Una asignación optima es: "+str(getBestAsig()))
40                  print("Su costo : " + str(cost(getBestAsig())))
41
42  import time
43  inicio = time.time()
44  #Declaracion de variables y del problema
45  mejorAsig ={}
46  examenes=[] # [1,2,3,4,5,.., N]
47  Asignacion = {}
48  Problema ={
49      'Juan':['Jorge','Diana'],
50      'Jorge':['Juan'],
51      'Diana':['Juan'],
52  }
53
54  Alumnos = list(Problema.keys())
55  i=0
56  for alumno in Problema.keys():
57      Asignacion[alumno] = 0
58      i = i +1
59      examenes.append(i)
60
61  costmin =len(examenes)
62
63  GetAsig(0)
64  fin = time.time()
65  print("tiempo de ejecución: "+str(fin-inicio))
```

Output

"Si funciona, aunque ..."

Esto es lo que imprime, pero debemos ver cuanto tarda para mayor número de alumnos

```
Una asignación optima es: {'Juan': 1, 'Jorge': 2, 'Diana': 2}
Su costo : 2
tiempo de ejecución: 0.0001485347747802734 4
```

Experimentos

Experimento de baja dificultad

```
#Declaracion de variables y del problema
mejorAsig ={}
examenes=[] # [1,2,3,4,5,.., N]
Asignacion = {}
Problema ={
    'Juan':['Jorge','Diana'],
    'Jorge':['Juan'],
    'Diana':['Juan'],
}


Alumnos = list(Problema.keys())#Hacemos una lista con los alumnos
i=0
for alumno in Problema.keys(): #inicializamos el diccionario de asignaciones con los alumnos
    Asignacion[alumno] = 0
    i = i +1
    examenes.append(i) #llenamos la lista de examenes

costmin =len(examenes)

GetAsig(0)
```

Experimentos

Experimento de media dificultad

```
40  Problema ={
41          'Laura':['Jorge','Juan','Nancy'],
42          'Jose':['Diana','Fernanda','Carlos','Diana','Axel'],
43          'Liliana':['Axel','Fernanda'],
44          'Juan':['Carlos','Laura','Jorge'],
45          'Jorge':['Laura','Juan','Axel'],
46          'Diana':['Jose','Nancy','Jose'],
47          'Carlos':['Juan','Jose','Fernanda'],
48          'Nancy':['Diana','Laura'],
49          'Axel':['Liliana','Jorge','Jose'],
50          'Fernanda':['Jose','Liliana','Carlos']
51  }
```

Experimentos

Experimento de alta dificultad

```
Problema ={
        'Laura':['Jorge','Ana'],
        'Jose':['Diana','Gabriela'],
        'Liliana':['Axel','David','Gabriela'],
        'Juan':['Carlos','David','Gabriela'],
        'Jorge':['Laura','Diana'],
        'Diana':['Jose','Jorge','Nancy'],
        'Carlos':['Juan','Gabriela'],
        'Nancy':['Arturo','Diana','Angel'],
        'Axel':['Liliana','Gabriela'],
        'Fernanda':['Diego','Gabriela'],
        'Oscar':['David','Maria','Gabriela'],
        'Maria':['Gabriela','Oscar','Christian'],
        'Angel':['Christian','Nancy','Arturo'],
        'Arturo':['Nancy','Angel','Diego'],
        'Ana':['Karen','Laura','Gabriela'],
        'Diego':['Fernanda','Arturo','Karen'],
        'David':['Oscar','Liliana','Juan'],
        'Gabriela':['Maria','Jose','Carlos','Axel','Fernanda','Ana','Oscar','Liliana','Juan'],
        'Christian':['Angel','Maria'],
        'Karen':['Ana','Diego']
}
```

Bibliografía

 Sloman,Leila. (2021) *Mathematician Answers Chess Problem About Attacking Queens*. Quanta Magazine.