Universidad Politécnica Salesiana

Nombre: Lucy Garay

Evaluación práctica

1. A traves de la misma api generar una semilla diferente.
2. Encontrar el numero de iteraciones hasta que se repita uno de sus datos.
3. Generar 100 simulaciones con diferentes semillas.
4. Generar un histograma con el resultado obtenidos por cada método.
5. Agregar sus conclusiones, opiniones y recomendaciones

Algoritmo de Cuadros Medios

In [18]:

```
import random
import numpy as np
import pandas as pd
import math
import psutil

xn=[]
multiplicacion=[]
log=[]
ui_sem = []
rn=[]

numero = (dict(psutil.virtual_memory()._asdict())).get('total')

digito=int(input())
print(f"digito:, {digito}")

iteraciones = int(input())
print(f"iteraciones:, {iteraciones}")

def centros(mul):
    cortarI=int(digito/2)
    cortarD=digito-cortarI
    mitad=math.floor(len(mul)/2)
    unir=''
    for i in range(mitad-cortarI, mitad+cortarD, 1):
        unir=unir+mul[i]
    ui_sem.append(unir)
    return unir

def cuadrado(num):
    multi=(num*num)
    m=str(multi)
    lon=len(m)
    if(len(m)%2!=0):
        if (lon < len(m)+1):
            m=str(m).zfill(len(m)+1)
    multiplicacion.append(m)
    return m

def dividido(n):
    ceros=[int(str(num).ljust(digito+1, "0")) for num in [1]]
    res=n/ceros[0]
    rn.append(res)
    return res


for i in range(iteraciones):
    m=str(cuadrado(int(numero)))
    if(len(m)-1>digito and int(numero)>0):
        xn.append(numero)
        dividido(int(centros(m))) #RN
```

```
        numero=ui_sem[-1]
    else:
        print('ingrese datos correctos')
        break

#print(xn_sem)
#print(multiplicacion)
print(ui_sem)
#print(rn)

#df=pd.DataFrame({"Xn":xn, "Xn*Xn":multiplicacion ,"UI ":ui, "RN":rn})
#df
```

```
5
digito:, 5
100
iteraciones:, 100
['22199', '27956', '15379', '65136', '26984', '81362', '97750', '50625', '28906', '55568', '78026'
, '80566', '08803', '49280', '85184', '63138', '64070', '49649', '50232', '32538', '87214', '62817
', '59754', '05405', '21402', '80456', '31679', '35590', '66481', '97233', '42562', '15238', '2196
6', '25051', '75526', '41766', '43987', '48561', '81707', '60338', '06742', '45456', '62479', '362
54', '43525', '44256', '85935', '48242', '72905', '51390', '09321', '88104', '23148', '58299', '87
734', '72547', '30672', '07715', '52122', '67028', '27527', '77357', '41054', '54309', '94674', '3
1662', '24822', '61316', '96518', '57243', '67610', '11121', '36766', '17387', '23077', '25479', '
91794', '61384', '79954', '26421', '80692', '11988', '37121', '79686', '98585', '90022', '39604',
'84768', '56138', '14750', '75625', '91406', '50568', '71226', '31430', '78449', '42456', '25119',
'09641', '94888']
```

In [33]:

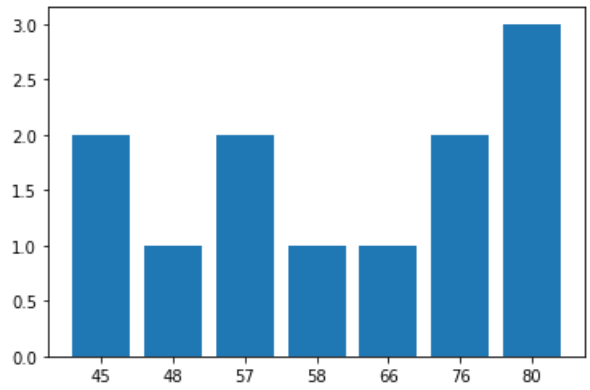
```
import random
import numpy as np
import pandas as pd
import math

xn=[]
multiplicacion=[]
log=[]
ui = []
rn=[]
g={}
rep=0


digito=5

iteraciones = 400

def cuadrado(num):
    multi=(num*num)
    m=str(multi)
    lon=len(m)
    if(len(m)%2!=0):
        if (lon < len(m)+1):
            m=str(m).zfill(len(m)+1)
    multiplicacion.append(m)
    return m

def dividido(n):
    ceros=[int(str(num).ljust(digito+1, "0")) for num in [1]]
    res=n/ceros[0]
    rn.append(res)
    return res

for j in ui_sem:
    numero=j
    for i in range(iteraciones):
        m=str(cuadrado(int(numero)))
        cortarI=int(digito/2)
        cortarD=digito-cortarI
        mitad=math.floor(len(m)/2)
        #print(mitad)
        unir=''
        for i in range(mitad-cortarI, mitad+cortarD, 1):
            unir=unir+m[i]
```

```
            if unir in ui:
                g[rep] = g[rep] + 1 if rep in g else  1
                break
        ui.append(unir)

        xn.append(numero)
        dividido(int(unir)) #RN
        numero=ui[-1]
    rep+=1


#print(xn)
#print(multiplicacion)
#print(ui)
#print(rn)
print(g)

#df=pd.DataFrame({"Xn":xn, "Xn*Xn":multiplicacion ,"UI ":ui, "RN":rn})
#df
```

```
{45: 2, 48: 1, 57: 2, 58: 1, 66: 1, 76: 2, 80: 3}
```

In [34]:

```
import matplotlib.pyplot as plt

grafica= g
plt.bar(range(len(grafica)), list(grafica.values()), align='center')
plt.xticks(range(len(grafica)), list(grafica.keys()))
plt.show()
```



Congruencia lineal

In [37]:

```
import random
import numpy as np
import pandas as pd
import math
import psutil


xn_semillas=[]
un=[]

iteraciones = 100
semilla = int(psutil.cpu_percent())
a=25214903917
c=11
m=pow(2, 48)-1

def formula(xo, A, C, M):
    form=((xo*A)+C)%M
    xn_semillas.append(form)
    return form
```

```
def dividido(n):
    d=n/m
    un.append(d)
    return d

xn_semillas.append(semilla)
un.append(' ')
for i in range(iteraciones):
    primero=semilla
    semilla=formula(primero, a, c, m)
    dividido(semilla)

print(xn_semillas)
#print(un)

#df=pd.DataFrame({"Xn":xn, "Un":un})
#df
```

```
[14, 353008654849, 66015582027714, 139978396537319, 217336184046034, 109737199590414,
233978104010879, 24388994536519, 240744420658704, 11596678581884, 275503305480904,
196373767556124, 54977795761214, 203794064797264, 40949418164064, 263489241471614, 89084888179249,
81091911300159, 108062548728089, 142673698398844, 59108233946724, 173405876053169,
258140717013589, 38783747166504, 239036269320629, 114740490903889, 101605575592749,
23128050736949, 33979792765699, 248544003849264, 92587423856519, 235076709395959, 23290061813799,
145444422382109, 261948094022479, 238309778386929, 109077172188674, 188632888863724,
47236790077614, 184481814994544, 57272291820979, 209139177284979, 132872887462634,
202075582468099, 154238047005459, 246065002205054, 257454427076074, 277613827003944,
80099081680034, 147277962664774, 139717945976109, 12221767604204, 141215932935949, 85492404053634,
208871075462459, 154245188223739, 206744185991214, 91514084902454, 122782502666779,
160784207749869, 84618950781509, 109747607985229, 203833468700544, 103667366917454,
138889612835299, 69587140619574, 229048204959074, 75064486898104, 148830704659599,
235567243577789, 138184559309479, 231847209104214, 173215322927609, 202937091177034,
280025297424009, 154678735411904, 63715613256439, 191500502810649, 25250810285474,
100425241559164, 203347417277694, 137863963907429, 4494321187129, 173730118541319,
158491039668014, 88498933827154, 205972434799359, 281412758425229, 278347481873059,
55240272516279, 169256771080079, 81940830544549, 225690865230429, 139867655922299,
209560294123954, 197230368191469, 242735958032894, 237187677748144, 62081068810989,
192478002263159, 23683478680369]
```

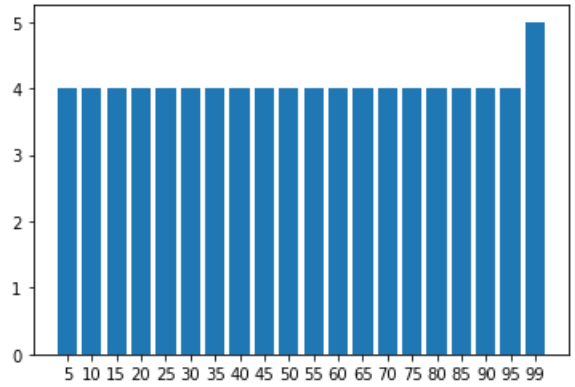
In [38]:

```
import random
import numpy as np
import pandas as pd
import math

xn=[]
un=[]
guardar={}

iteraciones = 100
a=25214903917
c=11
m=pow(2, 48)-1

repeticiones=0

def dividido(n):
    d=n/m
    un.append(d)
    return d

xn.append(semilla)
un.append(' ')
for k in xn_semillas:
    semilla=k
    for i in range(iteraciones):
        semilla=((semilla*a)+c)%m
        if semilla in xn:
            guardar[repeticiones] = guardar[repeticiones] + 1 if repeticiones in guardar else  1
            break
        xn.append(semilla)
        dividido(semilla)
        repeticiones+=1

    #print(guardar)
```

```
    #print(xn)
#print(un)

#df=pd.DataFrame({"Xn":xn, "Un":un})
#df
```

In [39]:

```
import matplotlib.pyplot as plt

grafica= guardar
plt.bar(range(len(grafica)), list(grafica.values()), align='center')
plt.xticks(range(len(grafica)), list(grafica.keys()))
plt.show()
```



Conclusiones, Opiniones y Recomendaciones

Los números aleatorios son la base esencial de la simulación. Usualmente, toda la aleatoriedad involucrada en el modelo se obtiene a partir de un generador de números aleatorios que produce una sucesi´on de valores. En general, la validez de los métodos de transformación dependen fuertemente de la hipótesis de que los valores de partida son realizaciones de variables aleatorias.

Metodo de cuadrados de medios: Los números generados pueden repetirse cíclicamente despúes de una secuencia corta, como se puede observar en el histograma que muestra posicion y las veces en la que se repiten, segun vayan ejecutandose las simulaciones.

Metodo de congruencia lineal: Los principales generadores de n´umeros pseudo-aleatorios utilizados hoy en día son los llamados generadores congruenciales lineales. Tambien se puede observar en el histograma la posicion y las veces que se repite al ejecutar las simulaciones.

Las semillas en los dos metodos se obtiene del sistema del computador en le primer caso es accediendo a la memoria RAM, en el segundo caso es porcentaje de uso del CPU, en el segundo caso se considera los parámentros tomando en cuenta los generadores de uso común para que exista una mejor aletoriedad.

Considero que el metodo con mejores resultados en mi prueba es del cuadrados medios ya que tiene menos repeticiones lo cual garantiza una mejor ejecución.

*Tomar en cuenta la semilla inicial ya que de ella depende el proceso de los metodos.

*Verificar que la generacion para las nuevas semillas sea corrrecto ya que puede ocasionar problemas para las siguienes simulaciones.

*Tomar las recomendaciones del docente.

CUENCA, 22 DE J
O DEL 2020