Trabalho 1 – Valor de um vértice

Objetivo: Calcular a porcentagem ou valor/importância daquele vértice em um determinado grafo.

Código e implementação.

```
#coding: utf-8

import networkx as nx  #Importa networkx X responsavel por trabalhar com grafos

import numpy as np   # Numpy, trabalha com vetores

import matplotlib.pyplot as plt

import collections #biblioteca utilizada para ordenar um dicionario

G = nx.read_gml('C:/Users/Claudio/Desktop/trabalhografo/dolphins.gml') # Importando o grafo enviado pelo professor


prob=np.empty([]) #Criando vetor vazio chamado prob com numpy e sua propriedade empty


Narestas2=(G.number_of_edges())*2 #Multiplica o numero de aresta por 2, necessario para encontrar a probabilidade desejada


graus=G.degree() # jogo na variavel graus, o dicionario com o nome do vertice e seus graus.


#print(graus)


print("----------------------------------------")

print(" ")

print('Vertices e seus respectivos graus')

print(" ")

print(collections.OrderedDict(sorted(graus.items())))
```

```
def distribuicao(G): # Função que executa o calculo da probabilidade

    graus.update((x, y / (Narestas2)*100) for x, y in graus.items()) #Utilizo a função update, que seve para editar dicionarios, onde X é o nome do vertice e Y o grau do mesmo,

                                        #o for logo em seguida serve para iterar ou seja fazer a conta em todos os itens do dicionario(.item())

    prob=graus #atribuo o dicionario acima calculado ao vetor vazio prob


    return prob # retorne o vetor


result=distribuicao(G) #variavel resultado recebe a função com sua propriedade G

#print(result) #Escreva a variavel result.

print("----------------------------------------")

print(" ")

print('Vertices e sua porcentagem')

print(" ")

print(collections.OrderedDict(sorted(result.items())))#impressão do dicionario result ordenado alfabeticamente

print("----------------------------------------")

print(" ")

print('Soma da porcentagem de todos os vertices')

print(sum(graus.values()))#soma todos os valores para conferir se chegam a 100 %


print("----------------------------------------")

print(" ")

print('Vertice com a maior porcentagem')

print(max(result, key=result.get))#Retira do dicionario o maior valor
```

 # Parte da plotagem do grafico.

y_axis =graus.values()#define valores para Y do grafico, os valores de cada vertice

x_axis = range(len(y_axis))#tamanho do eixo X , range dos valores

width_n=0.5 # tamanho das barras

bar_color='yellow'#cor das barras

legenda=graus.keys() # legenda de cada barra, sua correspondente chave

ax=plt.axes() #ax para eixos, facilitar programçaão

grafico = plt.bar(x_axis,y_axis, width=width_n,color=bar_color) #define o grafico(eixox,eixoy,tamanho,cor)

ax.set_xticks(x_axis)#define que para cada barra havera uma legenda, define o espaçamento delas.

ax.set_xticklabels(legenda,rotation='vertical') #atribui a legenda os nomes dos vertices rotacionando para vertical

plt.xlabel('Vertices') #label x = vertice

plt.ylabel('Valores') # label y = valores

plt.title('Relacao de vertice de um grafo e seu valor dentro da estrutura') #titulo

plt.grid(True) #coloque grid = sim

plt.show() #mostre o grafico.

Resultados.

Vértices e seus respectivos graus

```
Vertices e seus respectivos graus

OrderedDict([('Beak', 6), ('Beescratch', 8), ('Bumper', 4), ('CCL', 3), ('Cross'
, 1), ('DN16', 4), ('DN21', 6), ('DN63', 5), ('Double', 6), ('Feather', 7), ('Fi
sh', 5), ('Five', 1), ('Fork', 1), ('Gallatin', 8), ('Grin', 12), ('Haecksel', 7
), ('Hook', 6), ('Jet', 9), ('Jonah', 7), ('Knit', 4), ('Kringel', 9), ('MN105',
 6), ('MN23', 1), ('MN60', 3), ('MN83', 6), ('Mus', 3), ('Notch', 3), ('Number1'
, 5), ('Oscar', 5), ('PL', 5), ('Patchback', 9), ('Quasi', 1), ('Ripplefluke', 3
), ('SMN5', 1), ('SN100', 7), ('SN4', 11), ('SN63', 8), ('SN89', 2), ('SN9', 8),
 ('SN90', 5), ('SN96', 6), ('Scabs', 10), ('Shmuddel', 5), ('Stripes', 7), ('TR1
20', 2), ('TR77', 6), ('TR82', 1), ('TR88', 2), ('TR99', 7), ('TSN103', 4), ('TS
N83', 2), ('Thumper', 4), ('Topless', 11), ('Trigger', 10), ('Upbang', 7), ('Vau
', 2), ('Wave', 2), ('Web', 9), ('Whitetip', 1), ('Zap', 5), ('Zig', 1), ('Zipfe
l', 3)])
----------------------------------------
```

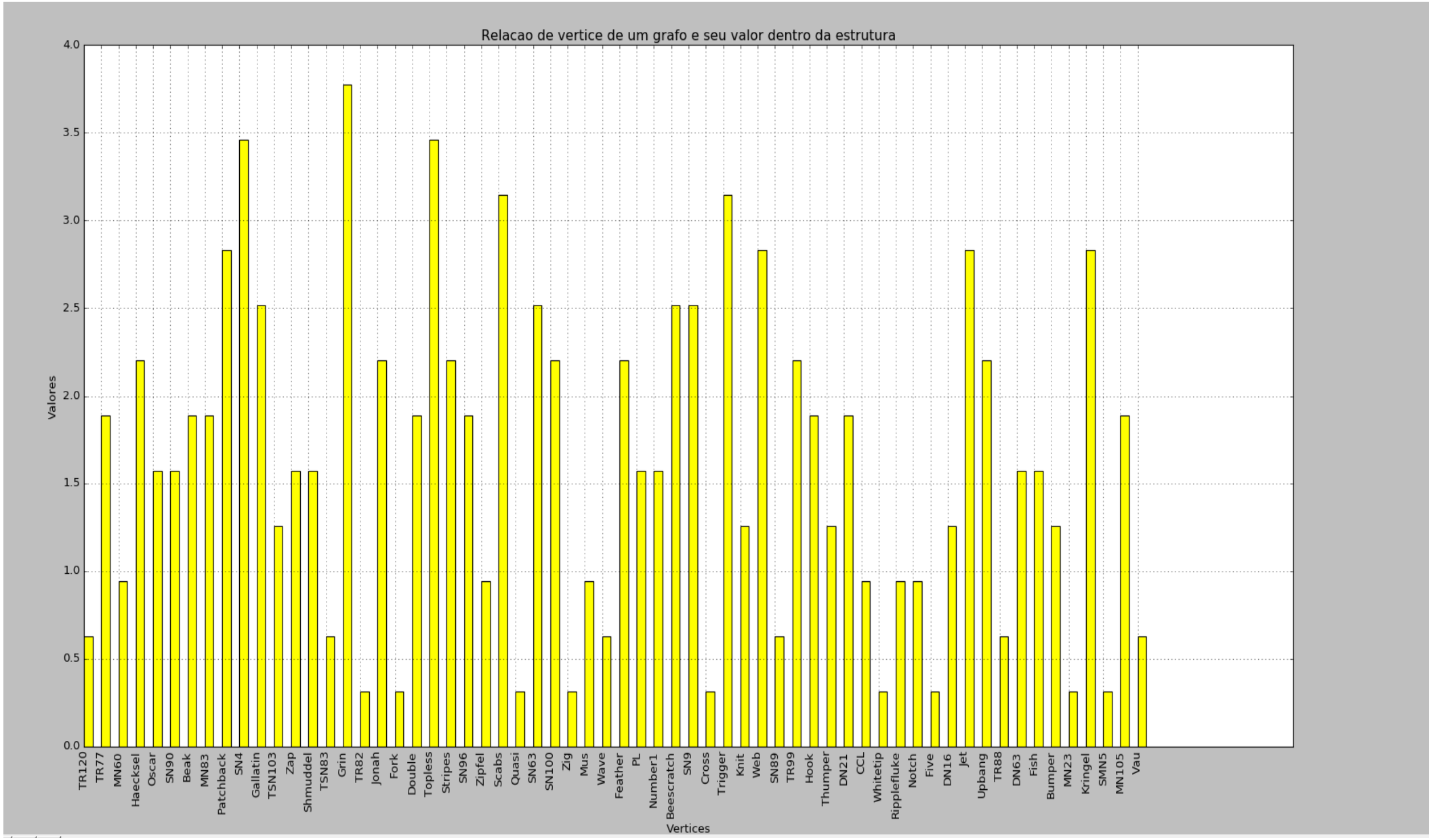
Calculo da porcentagem de cada vértice

```
Vertices e sua porcentagem

OrderedDict([('Beak', 1.8867924528301887), ('Beescratch', 2.515723270440252), ('
Bumper', 1.257861635220126), ('CCL', 0.9433962264150944), ('Cross', 0.3144654088
050315), ('DN16', 1.257861635220126), ('DN21', 1.8867924528301887), ('DN63', 1.5
723270440251573), ('Double', 1.8867924528301887), ('Feather', 2.20125786163522),
 ('Fish', 1.5723270440251573), ('Five', 0.3144654088050315), ('Fork', 0.31446540
88050315), ('Gallatin', 2.515723270440252), ('Grin', 3.7735849056603774), ('Haec
ksel', 2.20125786163522), ('Hook', 1.8867924528301887), ('Jet', 2.830188679245283
3), ('Jonah', 2.20125786163522), ('Knit', 1.257861635220126), ('Kringel', 2.830
1886792452833), ('MN105', 1.8867924528301887), ('MN23', 0.3144654088050315), ('M
N60', 0.9433962264150944), ('MN83', 1.8867924528301887), ('Mus', 0.9433962264150
944), ('Notch', 0.9433962264150944), ('Number1', 1.5723270440251573), ('Oscar',
1.5723270440251573), ('PL', 1.5723270440251573), ('Patchback', 2.830188679245283
3), ('Quasi', 0.3144654088050315), ('Ripplefluke', 0.9433962264150944), ('SMN5',
 0.3144654088050315), ('SN100', 2.20125786163522), ('SN4', 3.459119496855346), (
'SN63', 2.515723270440252), ('SN89', 0.628930817610063), ('SN9', 2.515723270440252
52), ('SN90', 1.5723270440251573), ('SN96', 1.8867924528301887), ('Scabs', 3.144
6540880503147), ('Shmuddel', 1.5723270440251573), ('Stripes', 2.20125786163522),
 ('TR120', 0.628930817610063), ('TR77', 1.8867924528301887), ('TR82', 0.31446540
88050315), ('TR88', 0.628930817610063), ('TR99', 2.20125786163522), ('TSN103', 1
.257861635220126), ('TSN83', 0.628930817610063), ('Thumper', 1.257861635220126),
 ('Topless', 3.459119496855346), ('Trigger', 3.1446540880503147), ('Upbang', 2.2
0125786163522), ('Vau', 0.628930817610063), ('Wave', 0.628930817610063), ('Web',
 2.8301886792452833), ('Whitetip', 0.3144654088050315), ('Zap', 1.572327044025157
3), ('Zig', 0.3144654088050315), ('Zipfel', 0.9433962264150944)])
----------------------------------------
```

Exibição do vértice com maior porcentagem

```
Vertice com a maior porcentagem
Grin
```

Gráfico com a relação de valor em porcentagem por vértice.



Conclusão.

Concluímos que os vértices que possuem maior grau, ou seja, maior ligação com outros vértices, possuem maior influencia dentro de um grafo, no caso o vértice chamado ‘Grin’ que possui grau 12 detém 3,77 % de “valor” dentro desta estrutura.

Trabalho 2 – Andar de bêbado

Objetivo: Executar o andar de bêbado dentro do grafo sugerido e verificar através da relação de vértice e seus vizinhos a influencia deste ponto na caminhada executada.

Código e implementação.

```
#coding: utf-8

import networkx as nx #Biblioteca para se trabalhar com grafos

import numpy as np #Trabalhar com vetores

import matplotlib.pyplot as plt

import pylab as p

import random #Biblioteca para gerar numeros aleatorios

import collections #biblioteca utilizada para ordenar um dicionario



G = nx.read_gml('C:/Users/Claudio/Desktop/trabalhografo/dolphins.gml') #Importando o grafo



passos=100 #Definindo o numero de passos a ser dado (Prof passou este valor)

i=0 #contador para fazer o for

grafoV=G.nodes() #Atribuo a variavel grafoV todos os vertices

visitas={x:0 for x in G.nodes()} #Atribuindo a todos estes vertices o valor 0 para serem o dicionario de visitas

inicio=(random.choice(grafoV)) #Professor deixou decidirmos qual seria o inicio, porem preferi definir aleatoriamente.


#vizinhos=(G.neighbors(inicio))#Função devolve os vizinhos do grafo inicial / fiz apenas para teste
```

#contador=0 #Contador para saber se foram feitas todas as iterações,

for i in range(passos): #laçõ de repetição que varree de i=0 até o tamanho dos passos, função range faz o i ser um vetor de 0 a 100 posições

visitas[inicio]+=1 #Incremento o vertice inicial com uma visita,

vizinhos=(G.neighbors(inicio)) #Verifico os vizinhos deste vertice

inicio=(random.choice(vizinhos)) #inicio recebe um destes vizinhos, e repete o laço

print("----------")

print(" ")

print('Numero de visitas de cada vertice')

print(" ")

print(collections.OrderedDict(sorted(visitas.items())))

print(" ")

*visitas.update((x, y /(passos)*100) for x, y in visitas.items()) #utilizo a função .update novamente no dicionario visitas, onde tenho o vertice o numero de vezes que passamos por ele*

#X é o vertice e Y o numero de visitas, novamente faço a conta iterando o valor Y/passos e multiplico por 100 para

*#encontrar a % que estas visitas equivalem na caminhada de bebado, como são 100 passos fica similar a nao * 100.*

print("----------")

print(" ")

print('Porcentagem equivalente a quantidade de visitas')

print(" ")

```
print(collections.OrderedDict(sorted(visitas.items()))) #mostra dicionario ja calculado com a % de cada vistias

print(" ")

print("----------------------------------------")

print(" ")

print('Soma da porcentagem de todas as visitas')

print(" ")

print(sum(visitas.values())) #soma todos os valores para conferir se chegam a 100 %

print(" ")


print("----------------------------------------")

print(" ")

print('Maior numero de visitas')

print(" ")

print(max(visitas, key=visitas.get))#Retira do dicionario o maior valor



#Plotagem do grafico



y_axis =visitas.values()#define valores para Y do grafico, os valores de cada vertice

x_axis = range(len(y_axis))#tamanho do eixo X , range dos valores

width_n=0.5 # tamanho das barras

bar_color='red'#cor das barras


legenda=visitas.keys() # legenda de cada barra, sua correspondente chave


ax=plt.axes() #ax para eixos, facilitar programçaão
```

grafico = plt.bar(x_axis,y_axis, width=width_n,color=bar_color) #define o grafico(eixox,eixoy,tamanho,cor)

ax.set_xticks(x_axis)#define que para cada barra havera uma legenda, define o espaçamento delas.

ax.set_xticklabels(legenda,rotation='vertical') #atribui a legenda os nomes dos vertices rotacionando para vertical

plt.xlabel('Vertices') #label x = vertice

plt.ylabel('Numero de visitas') # label y = valores

plt.title('Relacao de vertice de um grafo e seu numero de visitas no andar de bebado') #titulo

plt.grid(True) #coloque grid = sim

plt.show() #mostre o grafico.

Resultados.

Exibição do numero de visitas de cada vértice.

```
Numero de visitas de cada vertice

OrderedDict([('Beak', 2), ('Beescratch', 3), ('Bumper', 0), ('CCL', 0), ('Cross'
, 0), ('DN16', 6), ('DN21', 3), ('DN63', 1), ('Double', 2), ('Feather', 10), ('F
ish', 0), ('Five', 0), ('Fork', 0), ('Gallatin', 11), ('Grin', 2), ('Haecksel',
2), ('Hook', 0), ('Jet', 6), ('Jonah', 4), ('Knit', 1), ('Kringel', 1), ('MN105'
, 3), ('MN23', 1), ('MN60', 2), ('MN83', 2), ('Mus', 3), ('Notch', 2), ('Number1
', 3), ('Oscar', 2), ('PL', 0), ('Patchback', 1), ('Quasi', 0), ('Ripplefluke',
3), ('SMN5', 0), ('SN100', 1), ('SN4', 1), ('SN63', 0), ('SN89', 1), ('SN9', 0),
 ('SN90', 3), ('SN96', 1), ('Scabs', 0), ('Shmuddel', 3), ('Stripes', 0), ('TR12
0', 0), ('TR77', 0), ('TR82', 0), ('TR88', 0), ('TR99', 0), ('TSN103', 0), ('TSN
83', 0), ('Thumper', 2), ('Topless', 2), ('Trigger', 5), ('Upbang', 0), ('Uau',
0), ('Wave', 1), ('Web', 4), ('Whitetip', 0), ('Zap', 0), ('Zig', 0), ('Zipfel',
 0)])
```

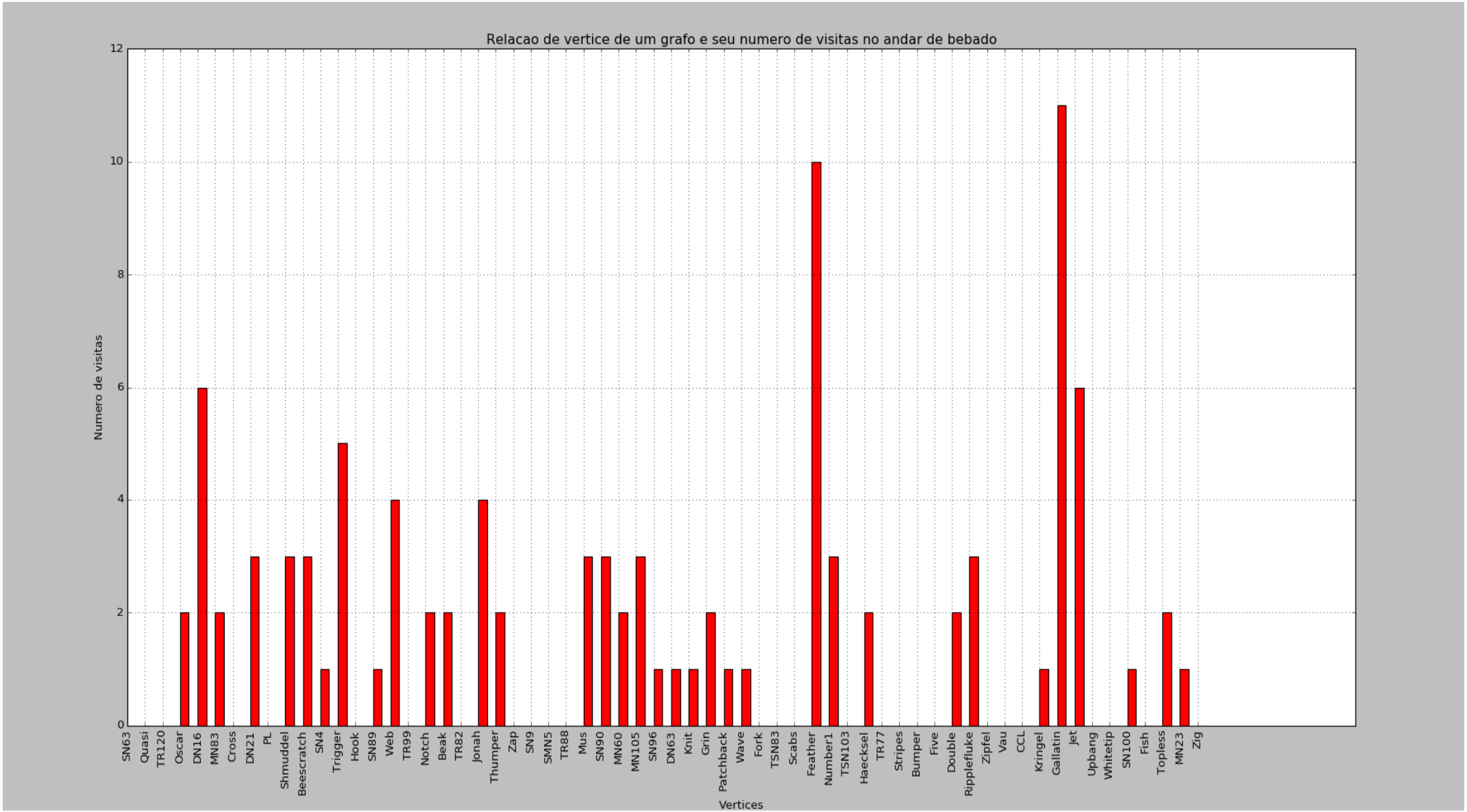
Porcentagem equivalente ao numero de visitas

```
Porcentagem equivalente a quantidade de visitas

OrderedDict([('Beak', 2.0), ('Beescratch', 3.0), ('Bumper', 0.0), ('CCL', 0.0),
('Cross', 0.0), ('DN16', 6.0), ('DN21', 3.0), ('DN63', 1.0), ('Double', 2.0), ('
Feather', 10.0), ('Fish', 0.0), ('Five', 0.0), ('Fork', 0.0), ('Gallatin', 11.0)
, ('Grin', 2.0), ('Haecksel', 2.0), ('Hook', 0.0), ('Jet', 6.0), ('Jonah', 4.0),
 ('Knit', 1.0), ('Kringel', 1.0), ('MN105', 3.0), ('MN23', 1.0), ('MN60', 2.0),
('MN83', 2.0), ('Mus', 3.0), ('Notch', 2.0), ('Number1', 3.0), ('Oscar', 2.0), (
'PL', 0.0), ('Patchback', 1.0), ('Quasi', 0.0), ('Ripplefluke', 3.0), ('SMN5', 0
.0), ('SN100', 1.0), ('SN4', 1.0), ('SN63', 0.0), ('SN89', 1.0), ('SN9', 0.0), (
'SN90', 3.0), ('SN96', 1.0), ('Scabs', 0.0), ('Shmuddel', 3.0), ('Stripes', 0.0)
, ('TR120', 0.0), ('TR77', 0.0), ('TR82', 0.0), ('TR88', 0.0), ('TR99', 0.0), ('
TSN103', 0.0), ('TSN83', 0.0), ('Thumper', 2.0), ('Topless', 2.0), ('Trigger', 5
.0), ('Upbang', 0.0), ('Uau', 0.0), ('Wave', 1.0), ('Web', 4.0), ('Whitetip', 0.
0), ('Zap', 0.0), ('Zig', 0.0), ('Zipfel', 0.0)])

----------------------------------------
```

Soma para verificação e vértice com maior numero de visitas

```
----------------------------------------

Soma da porcentagem de todas as visitas

100.0

----------------------------------------

Maior numero de visitas

Gallatin
```

Gráfico com relação entre numero de visitas por vértice.



Conclusão.

Como o vetor inicial para a caminha foi definido aleatoriamente, a cada execução do código, um novo passeio será realizado, e percebemos em todos que o vértice que possui maior numero de visitas é consequentemente o mais importante dentro do passeio em questão.