Notebook um pouco dijkstrado

Notebook da equipe Dijkstrados

OPI
Olimpiada Peruana de Informática

OPI
OPI.ESPORT

OPI
FARMA

opi
Connecting the business products world

OPI
OPEN INNOVATION COMPETENCIES

ACTFLOPI
ORAL PROFICIENCY INTERVIEW

OPI
ORDINE PROFESSIONI INFERMIERISTICHE
MILANO - LODI - MONZA E BRIANZA

OPI
OFFICE DE PARTENARIAT INTERNATIONAL

OPI
Oficina do Peso Ideal

OPI
OCEANS PAST INITIATIVE

opi analytics

Referências

Pilhas e filas

```
from collections import deque
deque_vazio = deque()
deque_lista = deque([5,2,3,4])

#pegar da esquerda e da direita
deque_primeiro = fila.popleft()
deque_ultimo = fila.pop()

#adicionar à esquerda e a direita
deque_prioridade.appendleft(valor)
deque_fila.append(valor)
```

Conjuntos

```
set_vazio = set()
set_lista = set([5,2,3,2,2]) #resulta em [5,2,3]
conjunto.add(valor)
conjunto.remove(valor)
conjunto.discard(valor) # remove() sem dar erro se não tiver

s1,s2 = {1,2},{2,3}
união = s1|s2 #resulta em {1,2,3}
interseção = s1&s2 #resulta em {2}
diferença = s1-s2 #resulta em {1}
diferença = s1^s2 #resulta em {1,3}
```

Fila prioritária

```
from queue import PriorityQueue
fila = PriorityQueue()
fila.put(numero)
primeiro = fila.get()
```

Referências

Dicionários

```
mapa_vazio = []
dicionario[nova] = valor_novo
valor_adicionado = dicionario[mesma]
dicionario.pop(valor)

#formas de iterar
chaves = dict.keys()
fechaduras = dict.values()
for chave,fechadura in dict.items():
     continue
```

Entrada e Saída

```
#entradas
import sys
sys.setrecursionlimit(10**6)
a,b = map(int, input().split())
lista_entrada = list(map(int, input().split()))
#prints
print(a,b,c) == print(f”{a} {b}”)
print(*lista, sep = ‘ ‘) #printa todos os valores da lista
print(f”{valor_float:.3f}”)
print(f”{zeros_na_frente:0<2}”)
```

Hashing

```
#apagar os tabs antes de testar
#windows
> certUtil -hashfile questao.txt sha256
#linux
$ sha256 questao.txt
a b
c
```

sha256sum: 721a7916ccfa56849995f47004497d7d8cefa18b0033b017026036cbe016e171

Algoritmos

Soma Acumulada

```
sacumulada=[l[0]]
for i in range(1,len(l)):
    sacumulada.append(sacumulada[i-1]+l[i])
```

sha256sum: ee7fb5f6f7469c296082e49336fc5cf4f71d11843349aa790d468f3a4184aab1

Busca binária

```
def bsearch(i,l):
    p1=0
    p2=len(l)
    while p1!=p2:
        m=(p1+p2)//2
        if i>l[m]:p1=m+1
        else:p2=m
    return p1
```

sha256sum: 4f1bda4e7fe897662529c2b9a2826edb3bc83742f6d644cfd959e49f079d685b

Máximo Divisor Comum

```
def gcd(a,b):
    while a%b !=0:
    aux = b
    b = a%b
    a = aux
    return b
```

sha256sum: 88c1d7d2877ca1b31b7f18c26e75580c476e3c6648e412125f2977916efb5e9c

Mínimo Multiplicador Comum

```
def lcm(a,b):
    return (a/gcd(a,b))*b
```

sha256sum: 1fae73dcba5be425a044356dc3e5a5984570c8752f2359dd6de7042723a39997

Algoritmos

Divisores

```
import math
def divisores(n):
     divisores=[]
     i=1
     while i<=math.sqrt(n):
     if (n % i == 0):
          if(n/i==i):
               divisores.append(i)
          else:
               divisores.append(i)
               divisores.append(int(n/i))
     i=i+1
```

sha256sum: 2c6830bbcd7af64eec738fb86f55fb6e6ad3cda5414e5c0cc3c50bee3d0ed48a

Crivo De aristóteles

```
def crivo(n):
     prime = [True for i in range(n+1)]
     p = 2
     while (p**2<=n):
     if (prime[p] == True):
          for i in range(p**2, n+1, p):
               prime[i] = False
     p += 1
     return prime
```

sha256sum: 0de6f4ae8050662b07e5d8f5c74e16488da35ce9148b11dcccf1198542d8282f

Operações Modulares Simples

```
((a%m) + (b%m))%m = (a+b)%m
((a%m) - (b%m))%m = (a-b)%m
((a%m) * (b%m))%m = (a*b)%m
```

Algoritmos

Inverso multiplicativo

conceito: $b^{-1} = (b^{m-2})\%m$ #quando m é primo

$(a/b)\%m = (a * b^{-1})\%m$

```
#retorna o inverso multiplicativo
def pow_mod(x,n = m-2,m):
     y=1
     while n>0:
     if n%2:
          y=(y*x)%m
     n=n//2
     x=(x * x)%m
     return y
```

sha256sum: e16cb32d854cb62c9fcbcdb2652a254853f14e8971441594631607baf81fec7c

Exponenciação Binária

```
def power(x,y,p):
    res=1
    while (y>0):
        if ((y & 1)!=0):
            res=res*x
        y = y>>1
        x = x**2
    return res%p
```

sha256sum: 8918bdf57b5c79aa718ed6b36cb3c81225f67c89506eeb6ed21aea55961b5b7e

Algoritmos

Depth First Search

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited

graph = {'0': set(['1', '2']),
         '1': set(['0', '3', '4']),
         '2': set(['0']),
         '3': set(['1']),
         '4': set(['2', '3'])}
dfs(graph, '0')
```

sha256sum: 5c6cfdafdf4d6fc9cf3f8d26d5571765f3093bd0945df10478100d28edd2b960

Bredth First Search

```
import collections
def bfs(graph, root):
    visited, queue = set(), collections.deque([root])
    visited.add(root)
    while queue:
        vertex = queue.popleft()
        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)
if __name__ == '__main__':
    graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}
    bfs(graph, 0)
```

sha256sum: 55811e845a83edb57d2ef93e7d272d57069e27e5b7142d7c02c90d4af835a86d

Algoritmos

dijkstra

```
from queue import PriorityQueue
class Graph:
    def __init__(self, num_of_vertices):
        self.v = num_of_vertices
        self.edges = [[-1 for i in range(num_of_vertices)]
for j in range(num_of_vertices)]
        self.visited = []
     def add_edge(self, u, v, weight):
        self.edges[u][v] = weight
        self.edges[v][u] = weight
def dijkstra(graph, start_vertex):
    D = {v:float('inf') for v in range(graph.v)}
    D[start_vertex] = 0
    pq = PriorityQueue()
    pq.put((0, start_vertex))
    while not pq.empty():
        (dist, current_vertex) = pq.get()
        graph.visited.append(current_vertex)
        for neighbor in range(graph.v):
            if graph.edges[current_vertex][neighbor] != -1:
                distance =
graph.edges[current_vertex][neighbor]
                if neighbor not in graph.visited:
                    old_cost = D[neighbor]
                    new_cost = D[current_vertex] + distance
                    if new_cost < old_cost:
                        pq.put((new_cost, neighbor))
                        D[neighbor] = new_cost
    return D
```

sha256sum: 5dd7d05e25c2883a64a9395d7f4f3a72962d706442497030faccd6f88164eb30

Algoritmos

dijkstra Teste

```
g = Graph(9)
g.add_edge(0, 1, 4)
g.add_edge(0, 6, 7)
g.add_edge(1, 6, 11)
g.add_edge(1, 7, 20)
g.add_edge(1, 2, 9)
g.add_edge(2, 3, 6)
g.add_edge(2, 4, 2)
g.add_edge(3, 4, 10)
g.add_edge(3, 5, 5)
g.add_edge(4, 5, 15)
g.add_edge(4, 7, 1)
g.add_edge(4, 8, 5)
g.add_edge(5, 8, 12)
g.add_edge(6, 7, 1)
g.add_edge(7, 8, 3)

D = dijkstra(g, 0)
print(D)
```

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões

Questões