Plantilla de Programación Competitiva - Python

Universidad de Bogotá Jorge Tadeo Lozano

Facultad de Ciencias Naturales e Ingeniería

Semillero de Programación Competitiva

Maratón Nacional ACIS REDIS 2025

Elaborado por: Nicolás Leitón

Bogotá D.C., 17 de octubre de 2025

Índice

1. Leer Input

1.1. n inicial y luego casos

```
import sys
inputs = iter(sys.stdin.readlines())
TC = int (next(inputs))
for _ in range(TC):
  print(sum(map(int, next(inputs).split())))
```

```
3
1 2
5 7
6 3
--- out
3
12
9
```

1.2. Terminar cuando es 0 0

```
import sys
for line in sys.stdin.readlines():
  if line=='0 0\n': break
  print(sum(map(int, line.split())))
```

```
1 2
5 7
6 3
0 0
1 1 <- no se procesa
--- out
3
12
9
```

1.3. Enumerar casos hasta EOF

```

import  sys
for c, line in enumerate(sys.stdin.readlines(), 1):
    print("Case %s: %s\n" % (c, sum(map(int, line.split()))))
```

```
1 2
5 7
6 3
--- out
Case 1: 3

Case 2: 12

Case 3: 9
```

1.4. Input matriz nxm

```
1
2 import sys
3 lines = sys.stdin.readlines()
4 i= 0
5 matrices = []
6
7 while i<len(lines):
8   # Leer dimensiones
9   n, m = map(int, lines[i].split())
10  i += 1
11
12  #Leer la matriz de tamano n
13  matrix = [list(map(int, lines[i+j].split())) for j in
        range(n)]
14  matrices.append(matrix)
15
16  i +=n
```

2. Output Format

```
1 #Decimals output
2 text = "The price is {:.2f} dollars"
3 print(txt.format(45))
4 # The price is 45.00 dollar
5
6 #Use ',' as thousand separator:
7 txt = "the universe is {:,} years old "
8 print(txt.format(13800000000))
9 # The universe is 13,800,000,000 years old
10
11 # Convert to percentage format:
```

```
txt ="you scored {:.1%}"
print(txt.format(0.255)''
# you scored 25.5%

txt ="you scored {:.0%}"
print(txt.format(0.255)''
# you scored 26%


#covert number to binary

txt = "The binary version of {0} is {0:b}"
print(txt.format(5))
#The binary version of 5 is 101
```

3. Macros

```
def crear_matriz(n, m, valor =0):
  return [[valor for _ in range(m)] for _ in range(n)]

def imprimir_matriz(mat):
  for fila in mat:
     print(*fila)

n, m = 3, 4
matriz= crear_matriz(n, m)
imprimir_matriz(matriz)

"""
0 0 0 0
0 0 0 0
0 0 0 0
"""
```

4. Estructuras de Datos Built-In

Esta sección contiene las estructuras de datos más útiles de Python para programación competitiva.

4.1. Collections - Deque

```
from collections import deque

```

```
dq = deque([1, 2, 3])
dq.append(4)        # [1, 2, 3, 4]
dq.appendleft(0)    # [0, 1, 2, 3, 4]
dq.popleft()        # -> 0
```

4.2. Heapq - Priority Queue

```
import heapq

h = []
heapq.heappush(h, 3)
heapq.heappush(h, 1)
heapq.heappush(h, 5)

print(heapq.heappop(h))  # 1 (menor elemento)

# Max heap:
h = []
heapq.heappush(h, -3)
heapq.heappush(h, -1)
heapq.heappush(h, -5)
print(-heapq.heappop(h)) # 5 (mayor)
```

4.3. DefaultDict - Grafos

```
from collections import defaultdict
#Representar grafos con listas de adyacencia.
g = defaultdict(list)
n, m = 5, 6 #Nodos, aristas
edges = [(1,2), (1,3), (2,4), (3,4), (4,5), (5,1)]

for u, v in edges:
    g[u].append(v)
    g[v].append(u)  # si es no dirigido
```

4.4. Counter - Frecuencias

```
from collections import Counter
#Cuenta frecuencias mas rapido
cnt = Counter([1,2,2,3,3,3])
print(cnt[2])       # 2
print(cnt.most_common(1))  # [(3, 3)]
```

4.5. Bisect - Búsqueda Binaria

```python
import bisect
#Insercion en lista ordenada
arr = [1, 3, 3, 5, 8]

print(bisect.bisect_left(arr, 3))   # -> 1
print(bisect.bisect_right(arr, 3))  # -> 3

#Cuantos elemento hay entre L y R
arr = [1, 2, 4, 4, 5, 7, 9]
L, R = 4, 7
left = bisect.bisect_left(arr, L)
right = bisect.bisect_right(arr, R)
print(right - left)  # 4 elementos en el rango [4,7]
```

5. Algoritmos

5.1. Two Sum

```python
#La funcion intenta determinar si en el arreglo a (de tamano n)
#existen dos numeros cuya suma sea igual a k.
def isPairSum(a, n, k):
    global x, y
    i, j = 0, n - 1
    while i < j:
        if a[i] + a[j] == k:
            x, y = i, j
            return 1
        elif a[i] + a[j] > k:
            j = j - 1
        else:
            i = i + 1
    return 0
```

5.2. Sliding Window - Suma Mínima

```python
# Halla la suma minima de un subarreglo de k elementos en un
   array,
#  y devuelve los indices donde empieza y termina dicho
   subarreglo.
for _ in range(int(input())):
    n, k = map(int, input().split())
    arr = list(map(int, input().split()))
```

```
    cursum = res_ind = 0
    # cursum = sum(x for x in arr[:k])
    for i in range(k):
        cursum += arr[i]
    minsum = cursum
    for i in range(k, n):
        cursum = cursum + arr[i] - arr[i - k] #Deslizar ventana
        if cursum < minsum:
            minsum = cursum
            res_ind = i - k + 1
    print(res_ind, res_ind + k - 1)

'''
#----INPUT----

1
7 3
3 7 90 20 10 50 40

#----OUTPUT----

3 5
'''
```

6. Grafos: Dijkstra

6.1. Dijkstra - Camino más corto

Complejidad: $O((N+M)\log N)$ con heapq.

```
from heapq import heappush, heappop
import math

def dijkstra(graph, start):
    dist = {u: math.inf for u in graph}
    dist[start] = 0
    pq = [(0, start)]
    while pq:
        d, u = heappop(pq)
        if d > dist[u]: continue
        for v, w in graph[u]:
            if dist[v] > d + w:
                dist[v] = d + w
                heappush(pq, (dist[v], v))
    return dist
```

```
16
17 # Ejemplo de uso
18 graph = {
19     1: [(2, 4), (3, 1)],
20     2: [(3, 2), (4, 5)],
21     3: [(4, 8)],
22     4: []
23 }
24 #Distancias minimas desde start a cada nodo
25 distancias = dijkstra(graph, 1)
26 print(distancias) #{1: 0, 2: 3, 3: 1, 4: 8}
```

Trucos:

- Usar heapq para min-heap eficiente.
- Para max-heap, negar los valores.
- Mantener diccionario de padres para reconstruir camino.

7. Template Completo - Template.py

Este es el template completo que incluye utilidades comunes para programación competitiva:

```
1
2 # PRINT ARRAY WITHOUT [,,,]
3 #print(*arr)
4
5
6 #La funcion intenta determinar si en el arreglo a (de tamano n)
7 #existen dos numeros cuya suma sea igual a k.
8 def isPairSum(a, n, k):
9     global x, y
10     i, j = 0, n - 1
11     while i < j:
12         if a[i] + a[j] == k:
13             x, y = i, j
14             return 1
15         elif a[i] + a[j] > k:
16             j = j - 1
17         else:
18             i = i + 1
19     return 0
20
21
22 # Halla la suma minima de un subarreglo de k elementos en un
    array,
```

```
#  y devuelve los indices donde empieza y termina dicho subarreglo.
for _ in range(int(input())):
    n, k = map(int, input().split())
    arr = list(map(int, input().split()))
    cursum = res_ind = 0
    # cursum = sum(x for x in arr[:k])
    for i in range(k):
        cursum += arr[i]
    minsum = cursum
    for i in range(k, n):
        cursum = cursum + arr[i] - arr[i - k] #Deslizar ventana
        if cursum < minsum:
            minsum = cursum
            res_ind = i - k + 1
    print(res_ind, res_ind + k - 1)

'''
#----INPUT----

1
7 3
3 7 90 20 10 50 40

#----OUTPUT----

3 5
'''




'''
COLLECTIONS PYTHON
'''

from collections import deque

dq = deque([1, 2, 3])
dq.append(4)       # [1, 2, 3, 4]
dq.appendleft(0)   # [0, 1, 2, 3, 4]
dq.popleft()       # -> 0
```

```python
import heapq

h = []
heapq.heappush(h, 3)
heapq.heappush(h, 1)
heapq.heappush(h, 5)

print(heapq.heappop(h))  # 1 (menor elemento)

# Max heap:
h = []
heapq.heappush(h, -3)
heapq.heappush(h, -1)
heapq.heappush(h, -5)
print(-heapq.heappop(h)) # 5 (mayor)




from collections import defaultdict
#Representar grafos con listas de adyacencia.
g = defaultdict(list)
n, m = 5, 6 #Nodos, aristas
edges = [(1,2), (1,3), (2,4), (3,4), (4,5), (5,1)]

for u, v in edges:
    g[u].append(v)
    g[v].append(u)  # si es no dirigido


from collections import Counter
#Cuenta frecuencias mas rapido
cnt = Counter([1,2,2,3,3,3])
print(cnt[2])       # 2
print(cnt.most_common(1))  # [(3, 3)]



import bisect
#Insercion en lista ordenada
arr = [1, 3, 3, 5, 8]

print(bisect.bisect_left(arr, 3))   # -> 1
print(bisect.bisect_right(arr, 3))  # -> 3

```

```python
#Cuantos elemento hay entre L y R
arr = [1, 2, 4, 4, 5, 7, 9]
L, R = 4, 7
left = bisect.bisect_left(arr, L)
right = bisect.bisect_right(arr, R)
print(right - left)  # 4 elementos en el rango [4,7]



'''Djistra'''
from heapq import heappush, heappop
import math

def dijkstra(graph, start):
    dist = {u: math.inf for u in graph}
    dist[start] = 0
    pq = [(0, start)]
    while pq:
        d, u = heappop(pq)
        if d > dist[u]: continue
        for v, w in graph[u]:
            if dist[v] > d + w:
                dist[v] = d + w
                heappush(pq, (dist[v], v))
    return dist
graph = {
    1: [(2, 4), (3, 1)],
    2: [(3, 2), (4, 5)],
    3: [(4, 8)],
    4: []
}
#Distancias minimas desde start a cada nodo
distancias = dijkstra(graph, 1)
print(distancias) #{1: 0, 2: 3, 3: 1, 4: 8}
```

8. Consejos Finales

- Usar `sys.stdin.readlines()` para lectura rápida.
- Preferir list comprehensions para mejor rendimiento.
- Usar `collections` para estructuras de datos eficientes.
- Familiarizarse con `heapq`, `bisect` y `itertools`.
- Para problemas de grafos, considerar usar diccionarios de adyacencia.