
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 |
| 1.1. n inicial y luego casos | 1 |
| 1.2. Terminar cuando es 0 0 | 1 |
| 1.3. Enumerar casos hasta EOF | 1 |
| 1.4. Input matriz nxm | 2 |
| 2. Output Format | 2 |
| 3. Macros | 3 |
| 4. Estructuras de Datos Built-In | 3 |
| 4.1. Collections - Deque | 3 |
| 4.2. Heapq - Priority Queue | 4 |
| 4.3. DefaultDict - Grafos | 4 |
| 4.4. Counter - Frecuencias | 4 |
| 4.5. Bisect - Búsqueda Binaria | 5 |
| 5. Algoritmos | 5 |
| 5.1. Two Sum | 5 |
| 5.2. Sliding Window - Suma Mínima | 5 |
| 6. Grafos: Dijkstra | 6 |
| 6.1. Dijkstra - Camino más corto | 6 |
| 7. Template Completo - Template.py | 7 |
| 8. Consejos Finales | 10 |

1. Leer Input

1.1. n inicial y luego casos

```
1 import sys
2 inputs = iter(sys.stdin.readlines())
3 TC = int (next(inputs))
4 for _ in range(TC):
5     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

```
1 import sys
2 for line in sys.stdin.readlines():
3     if line=='0 0\n': break
4     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

```
1
2 import sys
3 for c, line in enumerate(sys.stdin.readlines(), 1):
4     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 tamaño n
13    matrix = [list(map(int, lines[i+j].split())) for j in
14               range(n)]
15    matrices.append(matrix)
16
17    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:
```

```

12 txt = "you scored {:.1%}"
13 print(txt.format(0.255))
14 # you scored 25.5%
15
16 txt = "you scored {:.0%}"
17 print(txt.format(0.255))
18 # you scored 26%
19
20
21 #convert number to binary
22
23 txt = "The binary version of {0} is {0:b}"
24 print(txt.format(5))
25 #The binary version of 5 is 101

```

3. Macros

```

1 def crear_matriz(n, m, valor =0):
2     return [[valor for _ in range(m)] for _ in range(n)]
3
4 def imprimir_matriz(mat):
5     for fila in mat:
6         print(*fila)
7
8 n, m = 3, 4
9 matriz= crear_matriz(n, m)
10 imprimir_matriz(matriz)
11
12 """
13 0 0 0 0
14 0 0 0 0
15 0 0 0 0
16 """

```

4. Estructuras de Datos Built-In

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

4.1. Collections - Deque

```

1 from collections import deque
2

```

```

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

```

4.2. Heapq - Priority Queue

```

1 import heapq
2
3 h = []
4 heapq.heappush(h, 3)
5 heapq.heappush(h, 1)
6 heapq.heappush(h, 5)
7
8 print(heapq.heappop(h)) # 1 (menor elemento)
9
10 # Max heap:
11 h = []
12 heapq.heappush(h, -3)
13 heapq.heappush(h, -1)
14 heapq.heappush(h, -5)
15 print(-heapq.heappop(h)) # 5 (mayor)

```

4.3. DefaultDict - Grafos

```

1 from collections import defaultdict
2 #Representar grafos con listas de adyacencia.
3 g = defaultdict(list)
4 n, m = 5, 6 #Nodos, aristas
5 edges = [(1,2), (1,3), (2,4), (3,4), (4,5), (5,1)]
6
7 for u, v in edges:
8     g[u].append(v)
9     g[v].append(u) # si es no dirigido

```

4.4. Counter - Frecuencias

```

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

```

4.5. Bisect - Búsqueda Binaria

```
1 import bisect
2 #Insercion en lista ordenada
3 arr = [1, 3, 3, 5, 8]
4
5 print(bisect.bisect_left(arr, 3))    # -> 1
6 print(bisect.bisect_right(arr, 3))   # -> 3
7
8 #Cuantos elemento hay entre L y R
9 arr = [1, 2, 4, 4, 5, 7, 9]
10 L, R = 4, 7
11 left = bisect.bisect_left(arr, L)
12 right = bisect.bisect_right(arr, R)
13 print(right - left)    # 4 elementos en el rango [4,7]
```

5. Algoritmos

5.1. Two Sum

```
1 #La funcion intenta determinar si en el arreglo a (de tamaño n)
2 #existen dos numeros cuya suma sea igual a k.
3 def isPairSum(a, n, k):
4     global x, y
5     i, j = 0, n - 1
6     while i < j:
7         if a[i] + a[j] == k:
8             x, y = i, j
9             return 1
10        elif a[i] + a[j] > k:
11            j = j - 1
12        else:
13            i = i + 1
14    return 0
```

5.2. Sliding Window - Suma Mínima

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

```

6     cursum = res_ind = 0
7     # cursum = sum(x for x in arr[:k])
8     for i in range(k):
9         cursum += arr[i]
10    minsum = cursum
11    for i in range(k, n):
12        cursum = cursum + arr[i] - arr[i - k] #Deslizar ventana
13        if cursum < minsum:
14            minsum = cursum
15            res_ind = i - k + 1
16    print(res_ind, res_ind + k - 1)
17
18    '''
19    #----INPUT----
20
21    1
22    7 3
23    3 7 90 20 10 50 40
24
25    #----OUTPUT----
26
27    3 5
28    '''

```

6. Grafos: Dijkstra

6.1. Dijkstra - Camino más corto

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

```

1 from heapq import heappush, heappop
2 import math
3
4 def dijkstra(graph, start):
5     dist = {u: math.inf for u in graph}
6     dist[start] = 0
7     pq = [(0, start)]
8     while pq:
9         d, u = heappop(pq)
10        if d > dist[u]: continue
11        for v, w in graph[u]:
12            if dist[v] > d + w:
13                dist[v] = d + w
14                heappush(pq, (dist[v], v))
15    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 tamaño 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,

```

```

23 # y devuelve los indices donde empieza y termina dicho
    subarreglo.
24 for _ in range(int(input())):
25     n, k = map(int, input().split())
26     arr = list(map(int, input().split()))
27     cursum = res_ind = 0
28     # cursum = sum(x for x in arr[:k])
29     for i in range(k):
30         cursum += arr[i]
31     minsum = cursum
32     for i in range(k, n):
33         cursum = cursum + arr[i] - arr[i - k] #Deslizar ventana
34         if cursum < minsum:
35             minsum = cursum
36             res_ind = i - k + 1
37     print(res_ind, res_ind + k - 1)
38
39 '''
40 #----INPUT----
41
42 1
43 7 3
44 3 7 90 20 10 50 40
45
46 #----OUTPUT----
47
48 3 5
49 '''
50
51
52
53
54 '''
55 COLLECTIONS PYTHON
56 '''
57
58 from collections import deque
59
60 dq = deque([1, 2, 3])
61 dq.append(4)          # [1, 2, 3, 4]
62 dq.appendleft(0)      # [0, 1, 2, 3, 4]
63 dq.popleft()          # -> 0
64
65
66

```

```

67 import heapq
68
69 h = []
70 heapq.heappush(h, 3)
71 heapq.heappush(h, 1)
72 heapq.heappush(h, 5)
73
74 print(heapq.heappop(h)) # 1 (menor elemento)
75
76 # Max heap:
77 h = []
78 heapq.heappush(h, -3)
79 heapq.heappush(h, -1)
80 heapq.heappush(h, -5)
81 print(-heapq.heappop(h)) # 5 (mayor)
82
83
84
85
86 from collections import defaultdict
87 #Representar grafos con listas de adyacencia.
88 g = defaultdict(list)
89 n, m = 5, 6 #Nodos, aristas
90 edges = [(1,2), (1,3), (2,4), (3,4), (4,5), (5,1)]
91
92 for u, v in edges:
93     g[u].append(v)
94     g[v].append(u) # si es no dirigido
95
96
97 from collections import Counter
98 #Cuenta frecuencias mas rapido
99 cnt = Counter([1,2,2,3,3,3])
100 print(cnt[2]) # 2
101 print(cnt.most_common(1)) # [(3, 3)]
102
103
104
105 import bisect
106 #Insercion en lista ordenada
107 arr = [1, 3, 3, 5, 8]
108
109 print(bisect.bisect_left(arr, 3)) # -> 1
110 print(bisect.bisect_right(arr, 3)) # -> 3
111

```

```

112 #Cuantos elemento hay entre L y R
113 arr = [1, 2, 4, 4, 5, 7, 9]
114 L, R = 4, 7
115 left = bisect.bisect_left(arr, L)
116 right = bisect.bisect_right(arr, R)
117 print(right - left) # 4 elementos en el rango [4,7]
118
119
120
121 '''Dijkstra'''
122 from heapq import heappush, heappop
123 import math
124
125 def dijkstra(graph, start):
126     dist = {u: math.inf for u in graph}
127     dist[start] = 0
128     pq = [(0, start)]
129     while pq:
130         d, u = heappop(pq)
131         if d > dist[u]: continue
132         for v, w in graph[u]:
133             if dist[v] > d + w:
134                 dist[v] = d + w
135                 heappush(pq, (dist[v], v))
136     return dist
137 graph = {
138     1: [(2, 4), (3, 1)],
139     2: [(3, 2), (4, 5)],
140     3: [(4, 8)],
141     4: []
142 }
143 #Distancias minimas desde start a cada nodo
144 distancias = dijkstra(graph, 1)
145 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.