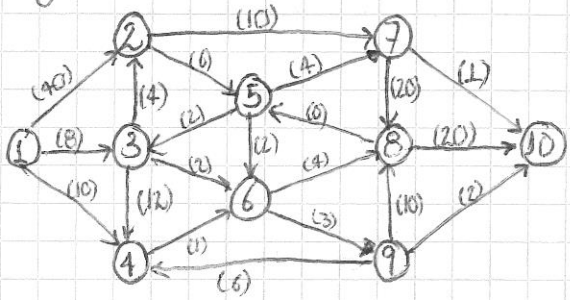
Taller 2 - Algoritmos

Jonathan Campo Rangel

1)



a) Dijkstra:

1) Iniciar distancias con valor ∞, excepto la del nodo inicial (=0)
2) a = nodo inicial
3) Visitar todos los vecinos de a y los marcamos al visitados (V_i)
4) Calcular distancia del nodo actual a sus vecinos. $dT(V_i) = Da + d(a, V_i)$. (Distancia total igual a distancia acumulada + distancia del nodo actual al vecino)
5) Si la distancia calculada es menor que la almacenada, se actualiza como nueva distancia tentativa.
6) Se marca como completado el nodo a.
7) Se toma un nuevo nodo inicial (el de menor distancia es D) y se repite desde el paso 3 mientras existan nodos no marcados.

Al terminar el algoritmo D tendrá las distancias entre los nodos.

b) Bellman - Ford

1) Inicear distancias con valor ∞, excepto el nodo inicial.
2) Crear diccionario de Dist_Finales y de Padres
3) Visitamos cada arista del grafo n-1 veces (n = # de nodos)
4) Comprobar que no hayan ciclos negativos.
5) Se obtiene una lista de los vertices ordenados por la ruta mas corta.

c) Floyd - Warshal: Comparar todos los posibles caminos a través de un grafo entre cada par de vértices

1) Definir matrices de vértices C, D
2) Inicializar constante K=1
3) Seleccionar fila y columna K de C y para $i \neq j \neq K$:
4) Si $(C_{iK} + C_{Kj}) < C_{ij} \Rightarrow D_{ij} = D_{Kj} \wedge C_{ij} = C_{iK} + C_{Kj}$
5) Si no $\Rightarrow$ No hay cambios
6) Si $K \leq n \Rightarrow K++$; else $\Rightarrow$ Stop.
7) C, al final, contiene los costos optimos para ir de un vértice a otro y D tiene los penúltimos vertices de los caminos optimos entre dos nodos.

DIJKSTRA

```
Costos desde 1:
({'1': 0, '2': 12, '3': 8, '4': 10, '5': 14, '6': 10, '7': 18, '8': 14, '9': 13, '10': 15}, {'2': '3', '3': '1', '4': '1',
'6': '3', '8': '6', '9': '6', '5': '8', '7': '5', '10': '9'})
Costos desde 2:
({'1': inf, '2': 0, '3': 8, '4': 17, '5': 6, '6': 8, '7': 10, '8': 12, '9': 11, '10': 11}, {'5': '2', '7': '2', '3': '5', '
6': '5', '4': '9', '8': '6', '9': '6', '10': '7'})
Costos desde 3:
({'1': inf, '2': 4, '3': 0, '4': 11, '5': 6, '6': 2, '7': 10, '8': 6, '9': 5, '10': 7}, {'2': '3', '4': '9', '6': '3', '8':
'6', '9': '6', '5': '8', '7': '5', '10': '9'})
Costos desde 4:
({'1': inf, '2': 11, '3': 7, '4': 0, '5': 5, '6': 1, '7': 9, '8': 5, '9': 4, '10': 6}, {'6': '4', '8': '6', '9': '6', '10':
'9', '5': '8', '3': '5', '7': '5', '2': '3'})
Costos desde 5:
({'1': inf, '2': 6, '3': 2, '4': 11, '5': 0, '6': 2, '7': 4, '8': 6, '9': 5, '10': 5}, {'3': '5', '6': '5', '7': '5', '2':
'3', '4': '9', '8': '6', '9': '6', '10': '7'})
Costos desde 6:
({'1': inf, '2': 10, '3': 6, '4': 9, '5': 4, '6': 0, '7': 8, '8': 4, '9': 3, '10': 5}, {'8': '6', '9': '6', '4': '9', '10':
'9', '5': '8', '3': '5', '7': '5', '2': '3'})
Costos desde 7:
({'1': inf, '2': 26, '3': 22, '4': 31, '5': 20, '6': 22, '7': 0, '8': 20, '9': 25, '10': 1}, {'8': '7', '10': '7', '5': '8'
, '3': '5', '6': '5', '2': '3', '4': '9', '9': '6'})
Costos desde 8:
({'1': inf, '2': 6, '3': 2, '4': 11, '5': 0, '6': 2, '7': 4, '8': 0, '9': 5, '10': 5}, {'5': '8', '10': '7', '3': '5', '6':
'5', '7': '5', '2': '3', '4': '9', '9': '6'})
Costos desde 9:
({'1': inf, '2': 16, '3': 12, '4': 6, '5': 10, '6': 7, '7': 14, '8': 10, '9': 0, '10': 2}, {'4': '9', '8': '9', '10': '9',
'6': '4', '5': '8', '3': '5', '7': '5', '2': '3'})
Costos desde 10:
({'1': inf, '2': inf, '3': inf, '4': inf, '5': inf, '6': inf, '7': inf, '8': inf, '9': inf, '10': 0}, {})
```

BELLMAN-FORD

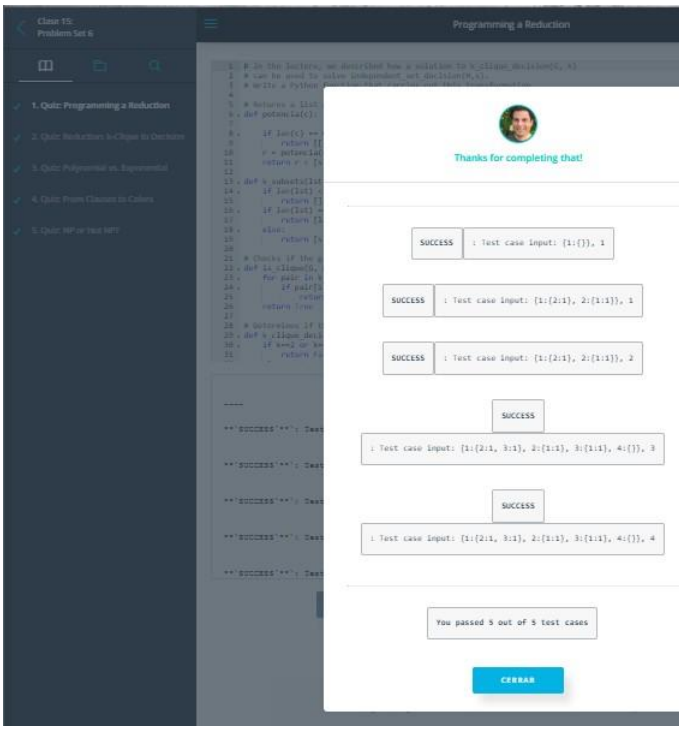
```
Costos y precedencias para 1:
({'1': 0, '2': 12, '3': 8, '4': 10, '5': 14, '6': 10, '7': 18, '8': 14, '9': 13, '10': 15}, {'2': '3', '3': '1', '4': '1',
'5': '8', '7': '5', '6': '3', '8': '6', '9': '6', '10': '9'})
Costos y precedencias para 2:
({'1': inf, '2': 0, '3': 8, '4': 17, '5': 6, '6': 8, '7': 10, '8': 12, '9': 11, '10': 11}, {'5': '2', '7': '2', '3': '5', '
6': '5', '8': '6', '9': '6', '10': '7', '4': '9'})
Costos y precedencias para 3:
({'1': inf, '2': 4, '3': 0, '4': 11, '5': 6, '6': 2, '7': 10, '8': 6, '9': 5, '10': 7}, {'2': '3', '4': '9', '6': '3', '8':
'6', '9': '6', '5': '8', '10': '9', '7': '5'})
Costos y precedencias para 4:
({'1': inf, '2': 11, '3': 7, '4': 0, '5': 5, '6': 1, '7': 9, '8': 5, '9': 4, '10': 6}, {'6': '4', '8': '6', '9': '6', '5':
'8', '10': '9', '3': '5', '7': '5', '2': '3'})
Costos y precedencias para 5:
({'1': inf, '2': 6, '3': 2, '4': 11, '5': 0, '6': 2, '7': 4, '8': 6, '9': 5, '10': 5}, {'3': '5', '6': '5', '7': '5', '8':
'6', '9': '6', '10': '7', '4': '9', '2': '3'})
Costos y precedencias para 6:
({'1': inf, '2': 10, '3': 6, '4': 9, '5': 4, '6': 0, '7': 8, '8': 4, '9': 3, '10': 5}, {'8': '6', '9': '6', '5': '8', '10':
'9', '4': '9', '3': '5', '7': '5', '2': '3'})
Costos y precedencias para 7:
({'1': inf, '2': 26, '3': 22, '4': 31, '5': 20, '6': 22, '7': 0, '8': 20, '9': 25, '10': 1}, {'8': '7', '10': '7', '5': '8'
, '3': '5', '6': '5', '9': '6', '4': '9', '2': '3'})
Costos y precedencias para 8:
({'1': inf, '2': 6, '3': 2, '4': 11, '5': 0, '6': 2, '7': 4, '8': 0, '9': 5, '10': 5}, {'5': '8', '10': '7', '3': '5', '6':
'5', '7': '5', '9': '6', '4': '9', '2': '3'})
Costos y precedencias para 9:
({'1': inf, '2': 16, '3': 12, '4': 6, '5': 10, '6': 7, '7': 14, '8': 10, '9': 0, '10': 2}, {'4': '9', '8': '9', '10': '9',
'6': '4', '5': '8', '3': '5', '7': '5', '2': '3'})
Costos y precedencias para 10:
({'1': inf, '2': inf, '3': inf, '4': inf, '5': inf, '6': inf, '7': inf, '8': inf, '9': inf, '10': 0}, {})
```

FLOYD-WARSHAL

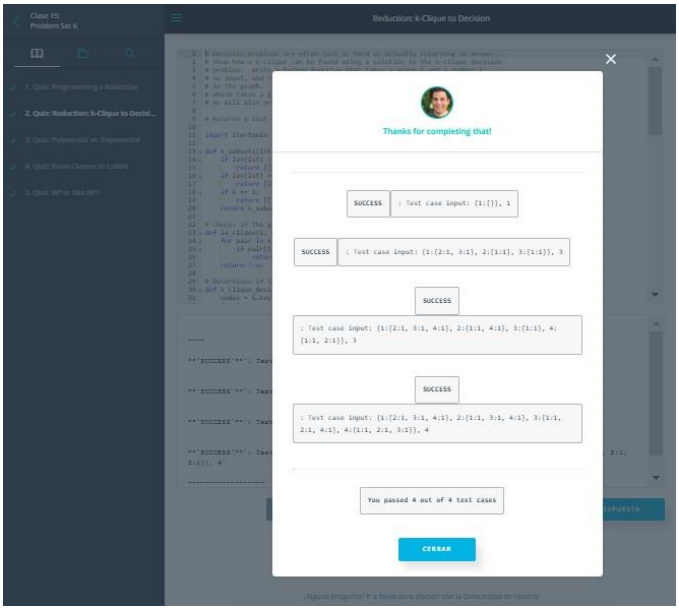
```
    1    2    3    4    5    6    7    8    9   10
 1 [0, 12, 8, 10, 14, 10, 18, 14, 13, 15]
 2 [inf, 0, 8, 17, 6, 8, 10, 12, 11, 11]
 3 [inf, 4, 0, 11, 6, 2, 10, 6, 5, 7]
 4 [inf, 11, 7, 0, 5, 1, 9, 5, 4, 6]
 5 [inf, 6, 2, 11, 0, 2, 4, 6, 5, 5]
 6 [inf, 10, 6, 9, 4, 0, 8, 4, 3, 5]
 7 [inf, 26, 22, 31, 20, 22, 0, 20, 25, 1]
 8 [inf, 6, 2, 11, 0, 2, 4, 0, 5, 5]
 9 [inf, 16, 12, 6, 10, 7, 14, 10, 0, 2]
 10 [inf, inf, inf, inf, inf, inf, inf, inf, inf, 0]
```

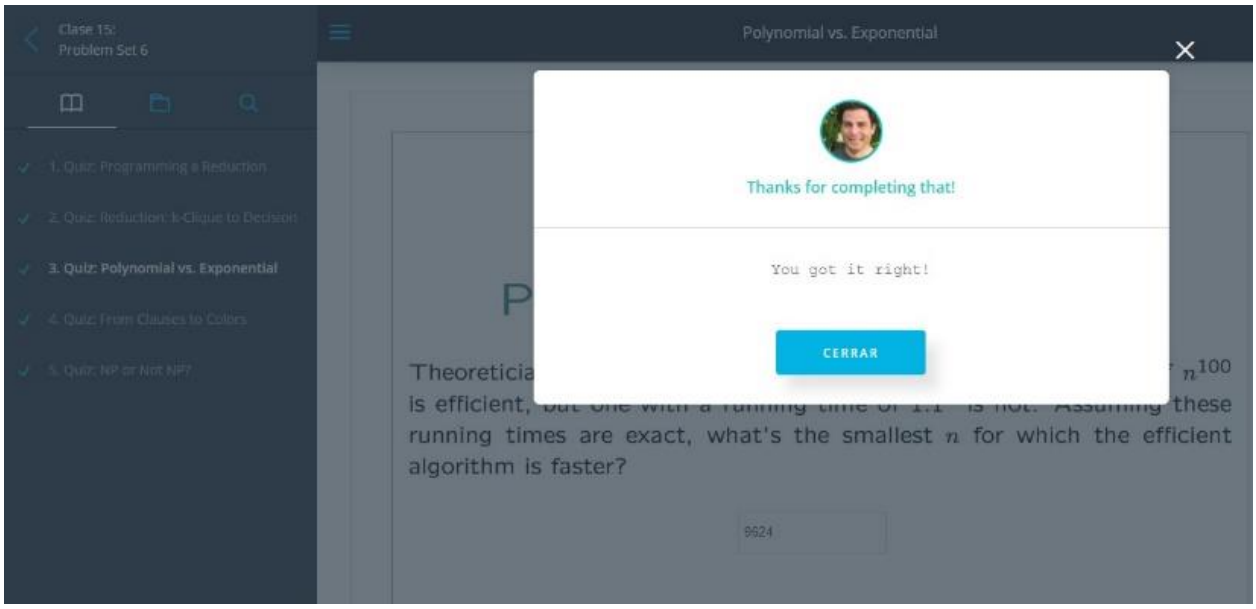
2) Problem set 6

- **PROGRAMMING A REDUCTION**



- **REDUCTION: K-CLIQUE TO DECISION**





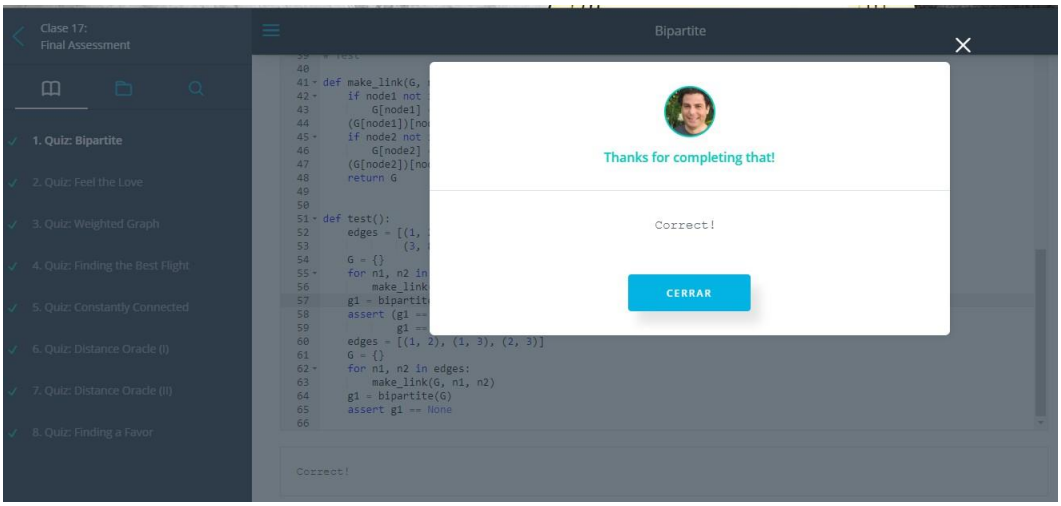
NP or Not NP? That is the Question

Select all the problems below that are in NP. Hint: Think about whether or not each one has a short accepting certificate.

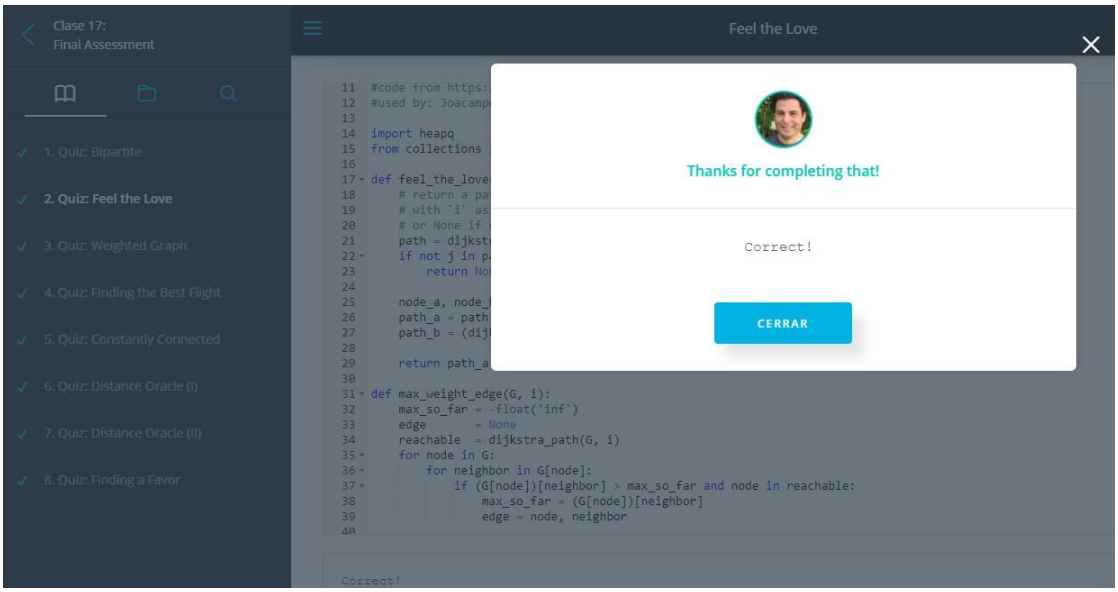
- [x] **Connectivity**: Is there a path from x to y in G?
- [x] **Short path**: Is there a path from x to y in G that is no more than k steps long?
- [ ] **Fewest colors**: Is k the absolute minimum number of colors with which G can be colored?
- [x] **Near Clique**: Is there a group of k nodes in G that has at least s pairs that are connected?
- [x] **Partitioning**: Can we group the nodes of G into two groups of size $n/2$ so that there are no more than k edges between the two groups.
- [ ] **Exact coloring count**: Are there exactly s ways to color graph G with k colors?

3) FINAL TEST

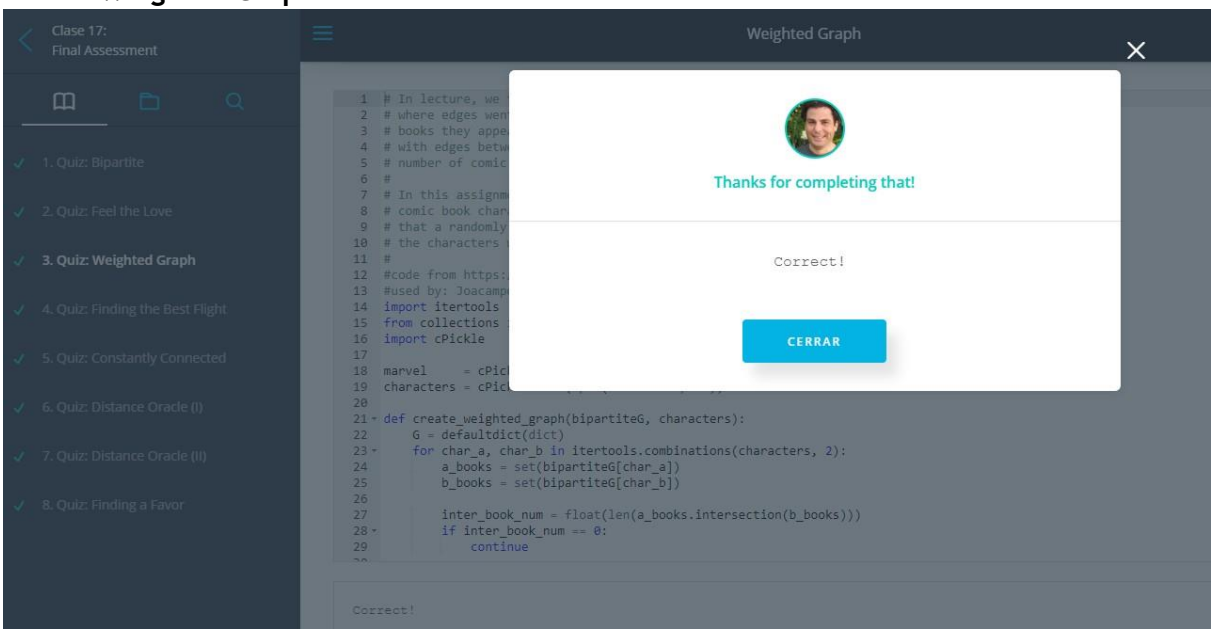
- **BIPARTITE**



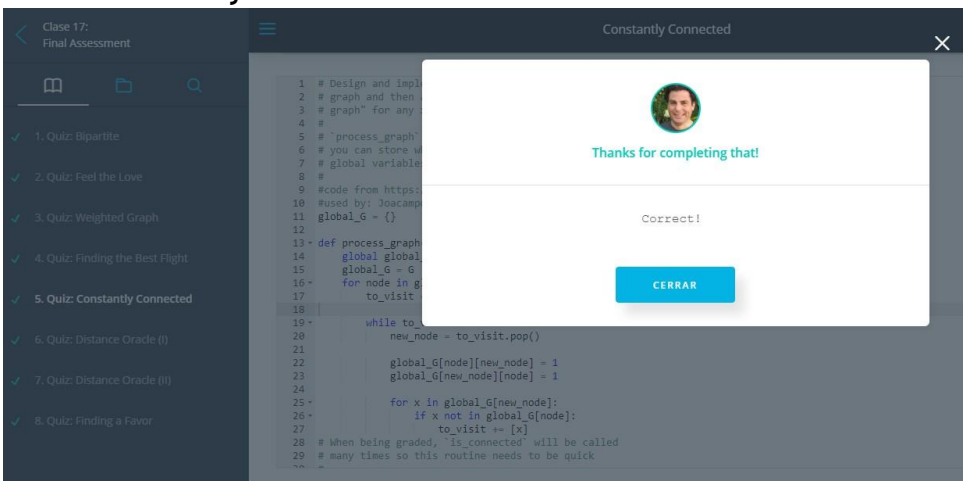
- **Feel the love**



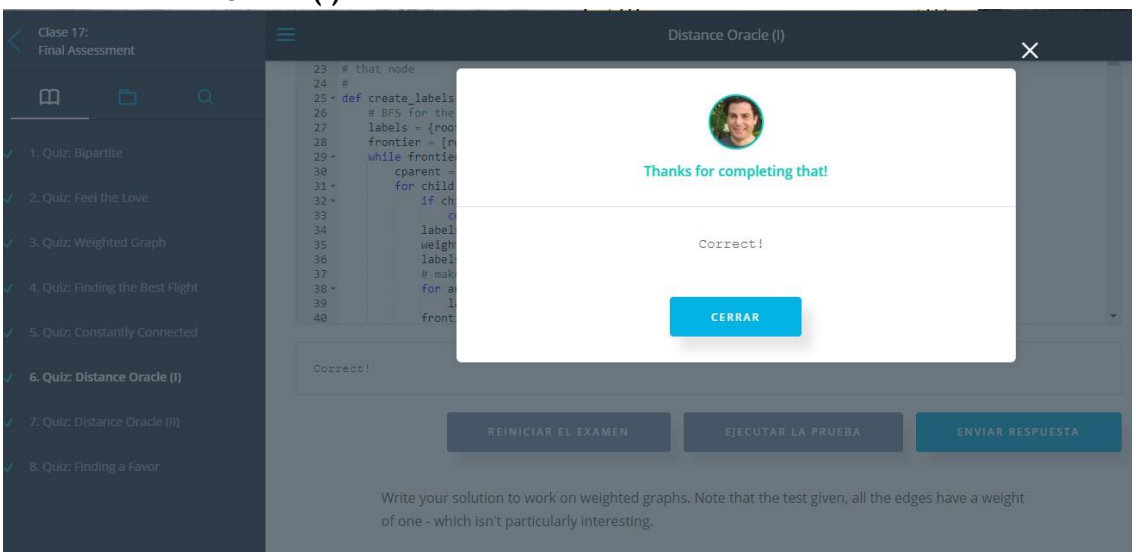
- **Weighted Graph**



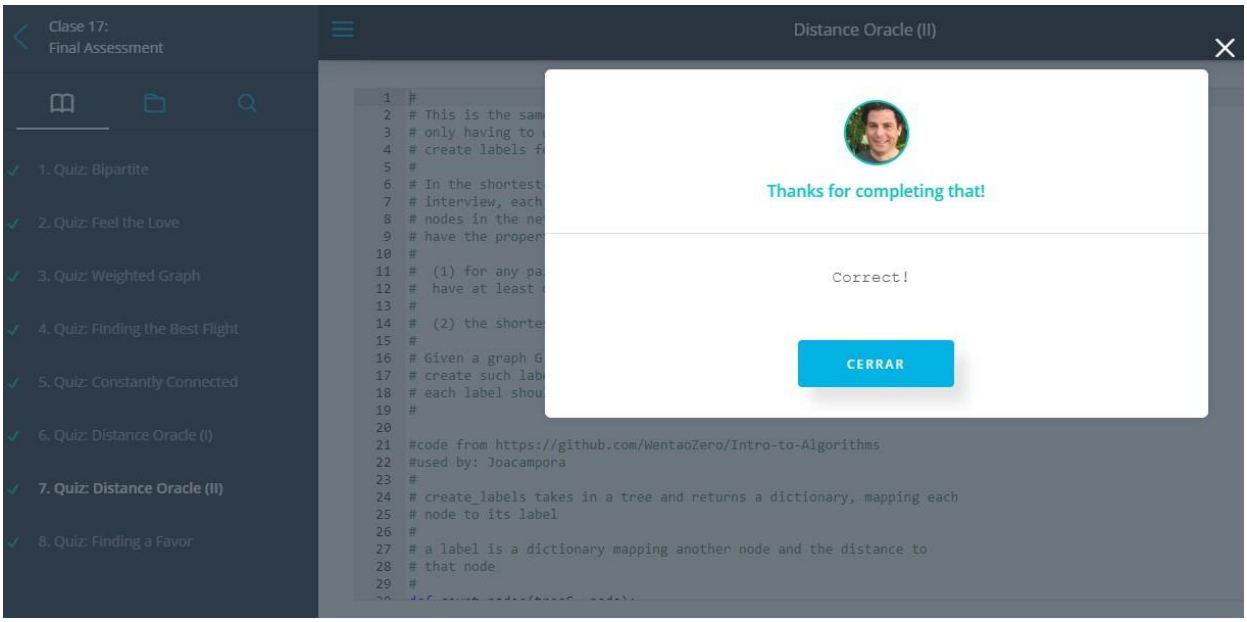
- **Constantly Connected**



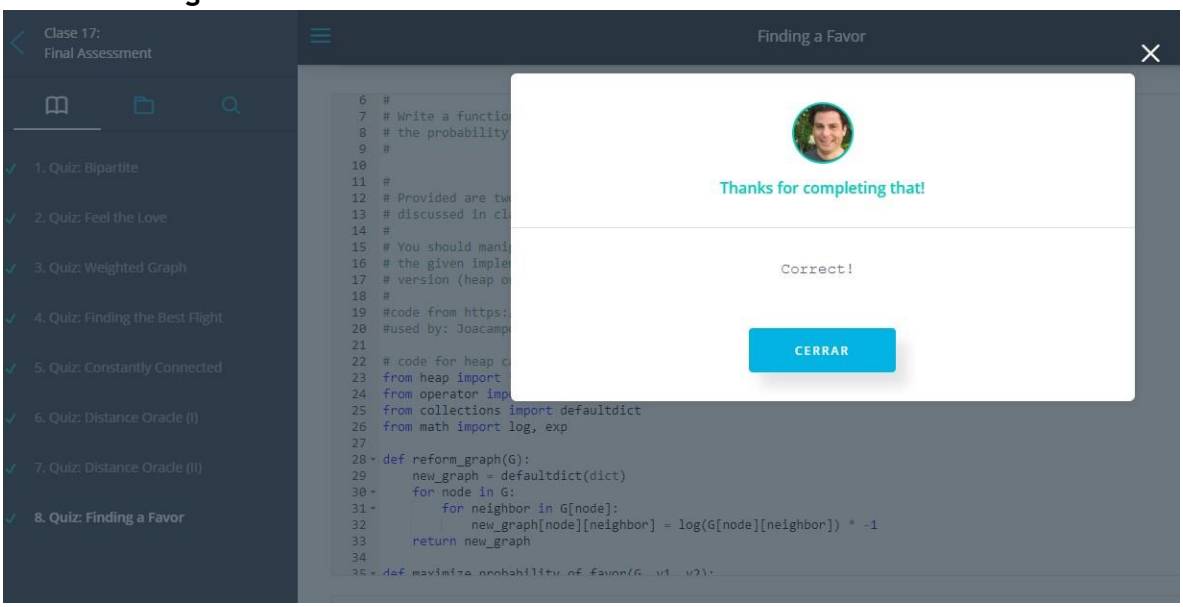
- **Distance Oracle(I)**



Distance Oracle(II)



- **Finding a Favor**



4) Tira de longitud n cubierta con fichas C_1 y C_2

a) Subestructura Optima: El problema se puede solucionar solucionando el mismo problema con un n menor.

b) Ecuación Recursiva.

$$P_n = \begin{cases} \min(P_2, P_3); & \text{si } n \leq 2; \\ \min(2P_2, P_3); & \text{si } n = 3 \\ \min(P_i + P_{n-1}); & 1 \leq i \leq n-1; \text{ si } n > 3 \end{cases}$$

c) Programa en Python: *

d) Tabla para $C_2 = 5, C_3 = 7, n = 10$.

n	1	2	3	4	5	6	7	8	9	10
Cubrir(5, 7, n)	5	5	7	10	12	14	17	19	21	24

5) Tablero 3Xn cubierto con fichas

C) Recurrencias:

$$A_n = D_{(N-1)} + C_{(N-1)}$$

$$C_n = A_{N-1}$$

$$D_n = D_{N-2} + 2 \cdot C_{n-1}$$

d) $B_n = E_n = 0$ (No es posible que resulte una forma del tablero que representan)

e) Python *

f) D_n para $n = 10; 50; 100$

$n = 10$
$D_n = 203$

$n = 50$
D_n

$n = 100$
D_n

$n = 20$
$D_n = 38651$

*Punto 4-c: By https://es.scribd.com/document/358405261/Grafos-Complejidad-Computacional-Programacion-Dinamica

```
def cubrir(C2, C3, n, r):
  r[0] = 0
  if n == 1 or n == 2:
    q = min(C2, C3)
  elif n == 3:
    q = min(2 * C2, C3)
  if i in r and (n - i) in r:
    q = min(q, r[i] + r[n - i])
  else:
    q = min(q, cubrir(C2, C3, i, r) + cubrir(C2, C3, n - i, r)
    )
  r[n] = q
  return q
```

*Punto 5-e: By https://es.scribd.com/document/358405261/Grafos-Complejidad-Computacional-Programacion-Dinamica

```
def A(N):
  if N == 0:
    return 0
  if N <= 1:
    return 1
  print "Haciendo A"
  return D(N - 2) + C(N -1)
def C(N):
  if N == 0: return 0
  if N <= 2: return 1
  return A(N - 1)
def D(N):
  if N == 0: return 0
  if N <= 2: return 3
  print "haciendo D"
  return D(N - 2) + 2*A(N-1)

```