Taller 2

Rodrigo Alexander Fagua Arevalo
rafaguaa@unal.edu.co

June 2018

1. Dijkstra

1.1. Ejecute el algoritmo de dijkstra y muestre los pasos ejecutados

- Inicializamos todas las distancias en D con un valor infinito debido a que son desconocidas al principio, la de el nodo start se debe colocar en 0 debido a que la distancia de start a start sería 0.
- Sea a = start (tomamos a como nodo actual).
- Visitamos todos los nodos adyacentes de a, excepto los nodos marcados, llamaremos a estos nodos no marcados vi.
- Para el nodo actual, calculamos la distancia desde dicho nodo a sus vecinos con la siguiente formula: dt(vi) = Da + d(a,vi). Es decir, la distancia del nodo 'vi' es la distancia que actual- mente tiene el nodo en el vector D mas la distancia desde dicho el nodo 'a' (el actual) al nodo vi. Si la distancia es menor que la distancia almacenada en el vector, actualizamos el vector con esta distancia tentativa. Es decir:

  [newDuv = D[u] + G[u][v]
  if newDuv ¡D[v]:
  P[v] = u
  D[v] = newDuv
  updateheap(Q,D[v],v)

- Marcamos como completo el nodo a
- Tomamos como proximo nodo actual el de menor valor en D (lo hacemos almacenando los valores en una cola de prioridad) y volvemos al paso 3 mientras existan nodos no marcados.

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

1.2. ejecute el algoritmo de bellman-ford y muestre los pasos ejecutados

- Inicializamos el grafo. Ponemos distancias a INFINITO menos el nodo origen que tiene dis- tancia 0.
- Tenemos un diccionario de distancias

  nales y un diccionario de padres.
- Visitamos cada arista del grafo tantas veces como numero de nodos -1 haya en el grafo
- comprobamos si hay ciclos negativo.
- La salida es una lista de los vertices en orden de la ruta mas corta

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

1.3. ejecute el algoritmo de Floyd-warshal y muestre los pasos ejecutados

- Formar las matrices iniciales C y D.
- Se toma k=1.
- Se selecciona la

  la y la columna k de la matriz C y entonces, para i y j, con i 6= k, j 6= k e i6=j, hacemos:
- Si (Cik + Ckj) ¡Cij , Dij = Dkj y Cij = Cik + Ckj
- En caso contrario, dejamos las matrices como estan.
- Si k ¡= n, aumentamos k en una unidad y repetimos el paso anterior, en caso contrario paramos las iteraciones.
- La matriz final C contiene los costes optimos para ir de un vertice a otro, mientras que la matriz D contiene los penultimos vertices de los caminos optimos que unen dos vertices, lo cual permite reconstruir cualquier camino optimo para ir de un vertice a otro.

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

2. Resuelva los ejercicios del problem set 6 de udacity

3. Resuelva los puntos del Final Exam del curso Algorithms de Udacity.

4. Considere el problema de cubrir una tira rectangular de longitud n con 2 tipos de fichas de domino con longitud 2 y 3 respectivamente. Cada ficha tiene un costo C2 y C3 respectivamente. El objetivo es cubrir totalmente la tira con un conjunto de fichas que tenga costo minimo. La longitud de la secuencia de fichas puede ser mayor o igual a n, pero en ningun caso puede ser menor.

- Muestre que el problema cumple con la propiedad de subestructura optima Para la resolucion de un problema de longitud n, primero se obtiene la solucion obtiene la solucion para una tira de longitud menor a n, calculando estas soluciones puede dar solucion al problema de longitud n.
- Plantee una ecuacion recursiva para resolver el problema
- Escriba un programa en Python que resuelva el problema de manera e e

  ciente de cubrir(C2, C3, n)

```
def cubrir (C2 , C3 , n , r ) :
        r[0] = 0
        q = float ( 'i n f' )
        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

- Llene la siguiente tabla:

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

screenshots

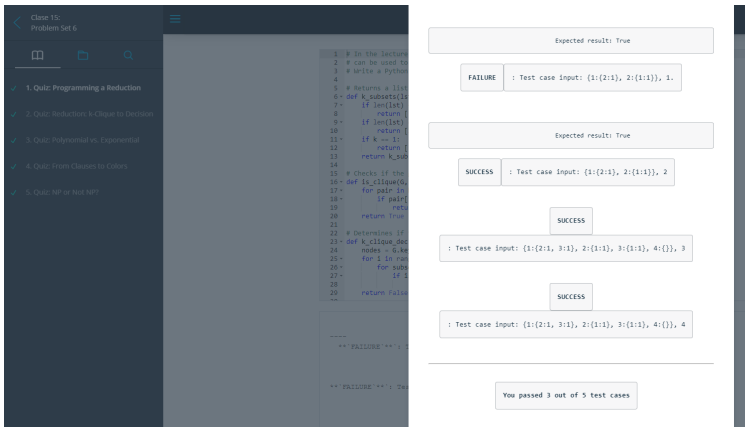

Figura 1: quiz 1

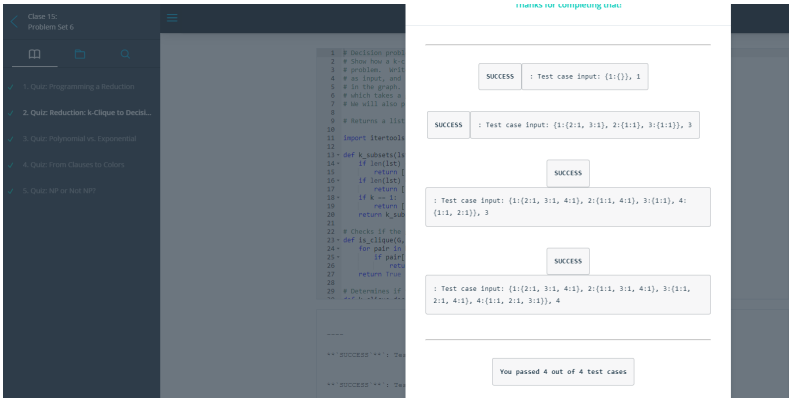

Figura 2: quiz 2

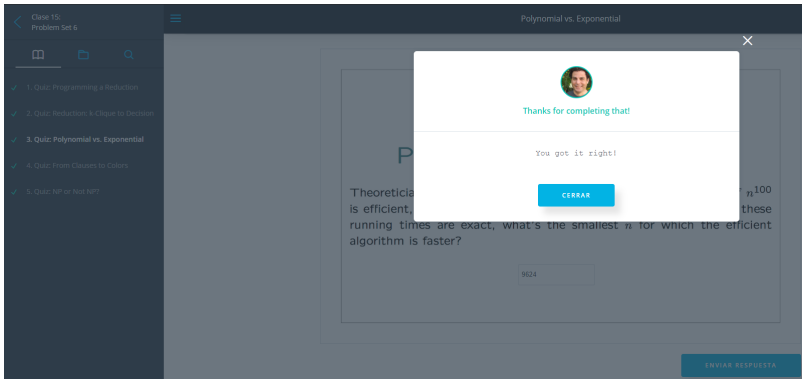

Figura 3: quiz 3

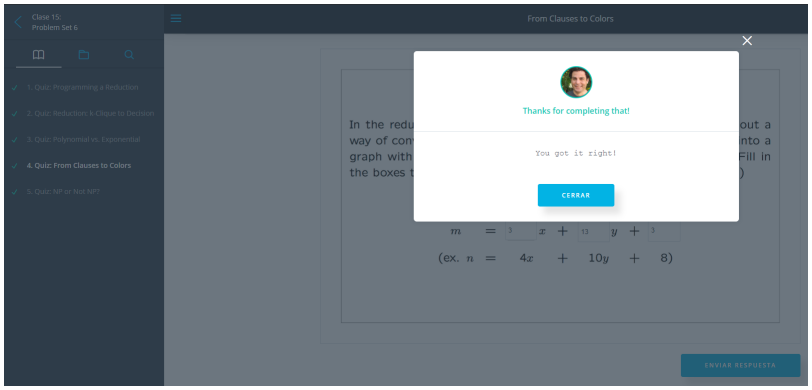

Figura 4: quiz 4

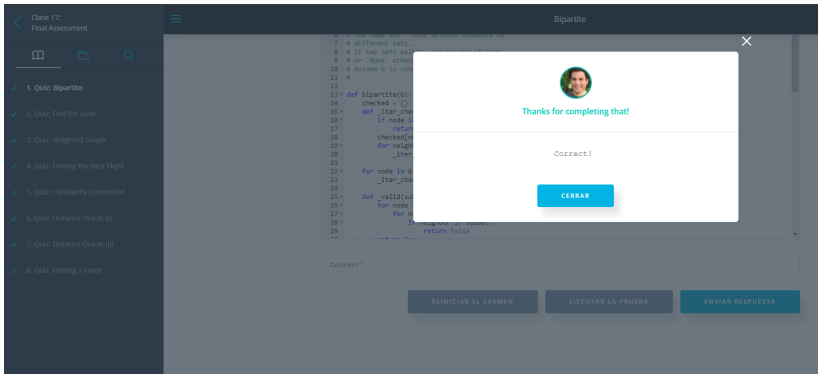

Figura 5: final quiz 1

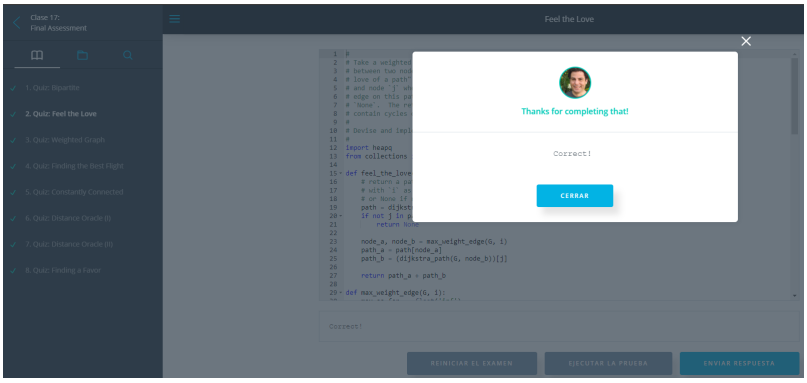

Figura 6: final quiz 2

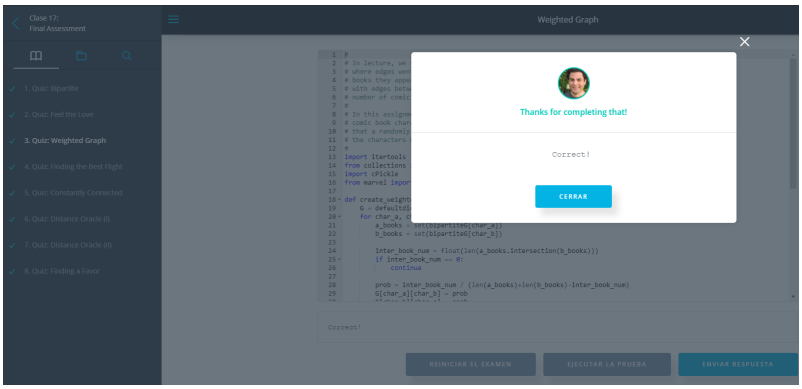

Figura 7: final quiz 3

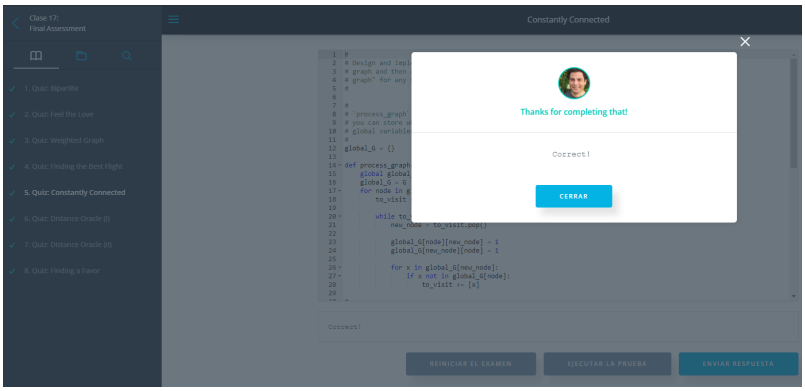

Figura 8: final quiz 5

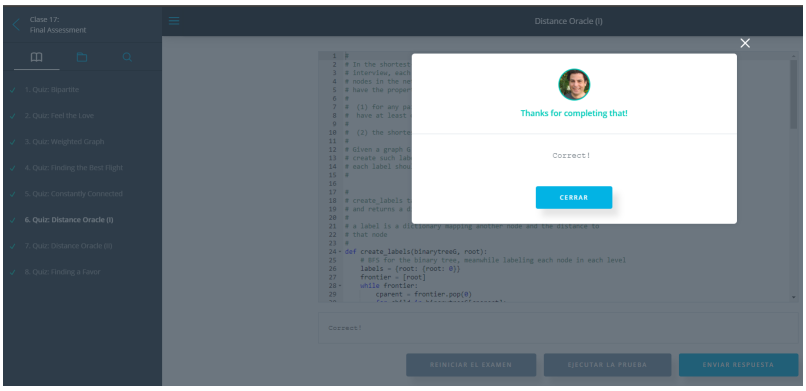

Figura 9: final quiz 6

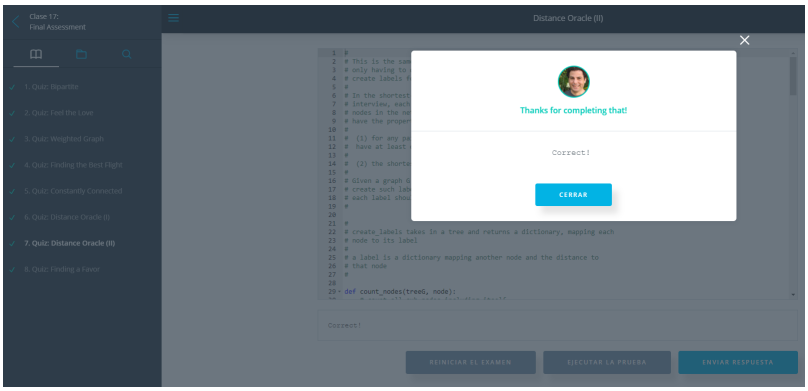

Figura 10: final quiz 7

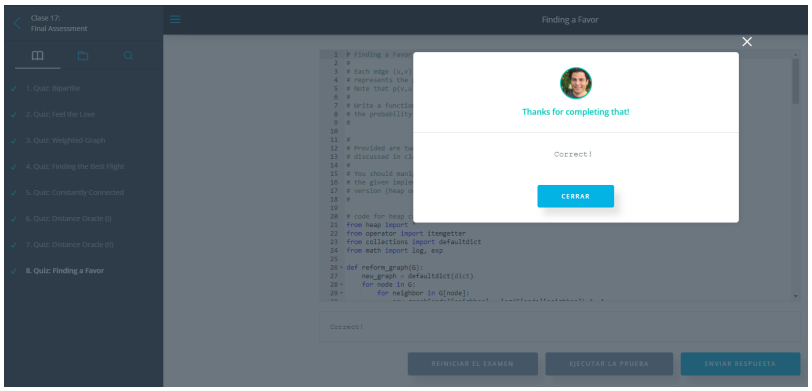

Figura 11: final quiz 8