Taller 2: Grafos, Complejidad Computacional y Programación Dinámica

Juan Camilo Lozano Mejía

1. Ejecutar algoritmos sobre un grafo:

1.1. Dijkstra:

1.1.1.1.1.

```
G = {'1': {'1':40, '3':8, '4':10},
     '2': {'5':6, '7':10},
     '3': {'4':12, '6':2},
     '4':{'6':1},
     '5':{'3':2,'6':2,'7':4},
     '6':{'8':4,'9':3},
     '7':{'8':20,'10':1},
     '8':{'5':0,'10':20},
     '9':{'4':6,'8':10,'10':2},
     '10':{}}
```

1.1.2. Inicializamos todas las distancias en D con un valor muy grande debido a que son desconocidas al principio, la del nodo start se debe colocar en 0 debido a que la distancia de start a start sería 0.

1.1.3. Visitamos todos los nodos adyacentes de a, excepto los nodos marcados, llamaremos a estos nodos no marcados vi.

1.1.4. Para el nodo actual, calculamos la distancia desde dicho nodo a sus vecinos con la siguiente fórmula: dt(vi) = Da + d(a,vi). Es decir, la distancia del nodo ‘vi’ es la distancia que actualmente tiene el nodo en el vector D más 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]

1.1.4.1.1.1.1. if newDuv < D[v]:

1.1.4.1.1.1.1.1.1. P[v] = u

1.1.4.1.1.1.1.1.2. D[v] = newDuv

1.1.4.1.1.1.1.1.3. updateheap(Q,D[v],v)

1.1.5. Marcamos como completo el nodo a.

1.1.6. Tomamos como próximo 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.

1.1.7. Cuando terminamos de ejecutar el algoritmo tendremos una lista con el valor para cada nodo del grafo, desde que el mismo sea alcanzable.

1.1.8. Todos las listas serian:

```
({'10': 15, '1': 0, '3': 8, '2': 10, '5': 14, '4': 10, '7': 18, '6': 10, '9': 13, '8': 14}, {'10': '
'2': '3', '5': '8', '4': '1', '7': '5', '6': '3', '9': '6', '8': '6'})
({'10': 11, '1': inf, '3': 8, '2': 0, '5': 6, '4': 17, '7': 10, '6': 8, '9': 11, '8': 12}, {'10': '7
'5': '2', '4': '9', '7': '2', '6': '5', '9': '6', '8': '6'})
({'10': 7, '1': inf, '3': 0, '2': 2, '5': 6, '4': 11, '7': 10, '6': 2, '9': 5, '8': 6}, {'10': '9',
: '8', '4': '9', '7': '5', '6': '3', '9': '6', '8': '6'})
({'10': 6, '1': inf, '3': 7, '2': 9, '5': 5, '4': 0, '7': 9, '6': 1, '9': 4, '8': 5}, {'10': '9', '3
'3', '5': '8', '7': '5', '6': '4', '9': '6', '8': '6'})
({'10': 5, '1': inf, '3': 2, '2': 4, '5': 0, '4': 11, '7': 4, '6': 2, '9': 5, '8': 6}, {'10': '7', '
'3', '4': '9', '7': '5', '6': '5', '9': '6', '8': '6'})
({'10': 5, '1': inf, '3': 6, '2': 8, '5': 4, '4': 9, '7': 8, '6': 0, '9': 3, '8': 4}, {'10': '9', '3
'3', '5': '8', '4': '9', '7': '5', '9': '6', '8': '6'})
({'10': 1, '1': inf, '3': 22, '2': 24, '5': 20, '4': 31, '7': 0, '6': 22, '9': 25, '8': 20}, {'10':
, '2': '3', '5': '8', '4': '9', '6': '5', '9': '6', '8': '7'})
({'10': 5, '1': inf, '3': 2, '2': 4, '5': 0, '4': 11, '7': 4, '6': 2, '9': 5, '8': 0}, {'10': '7', '
'3', '5': '8', '4': '9', '7': '5', '6': '5', '9': '6'})
({'10': 2, '1': inf, '3': 12, '2': 14, '5': 10, '4': 6, '7': 14, '6': 7, '9': 0, '8': 10}, {'10': '9
'2': '3', '5': '8', '4': '9', '7': '5', '6': '4', '8': '9'})
({'10': 0, '1': inf, '3': inf, '2': inf, '5': inf, '4': inf, '7': inf, '6': inf, '9': inf, '8': inf}
```

1.1.8.1.1.

1.2. Bellman-Ford:

```
G = {'1': {'1':40, '3':8, '4':10},
     '2': {'5':6, '7':10},
     '3': {'4':12, '6':2},
     '4':{'6':1},
     '5':{'3':2,'6':2,'7':4},
     '6':{'8':4,'9':3},
     '7':{'8':20,'10':1},
     '8':{'5':0,'10':20},
     '9':{'4':6,'8':10,'10':2},
     '10':{}}
```

1.2.1. Inicializamos el grafo. Ponemos las distancias en infinito menos el nodo origen que tiene una distancia de 0.

1.2.2. Tenemos un diccionario de distancias finales y un diccionario de padres.

1.2.3. Visitamos cada arista del grafo tantas veces como número de nodos -1 haya en el grafo

1.2.4. Comprobamos si hay ciclos negativos.

1.2.5. La salida es una lista de los vértices en orden de la ruta más corta.

1.2.6. Este sería el resultado:

```
({'10': 15, '1': 0, '3': 8, '2': 10, '5': 14, '4': 10, '7': 18, '6': 10, '9': 13, '8': 14}, {'10': '9', '3': '1',
'2': '3', '5': '8', '4': '1', '7': '5', '6': '3', '9': '6', '8': '6'})
({'10': 11, '1': inf, '3': 8, '2': 0, '5': 6, '4': 17, '7': 10, '6': 8, '9': 11, '8': 12}, {'10': '7', '3': '5',
'5': '2', '4': '9', '7': '2', '6': '5', '9': '6', '8': '6'})
({'10': 7, '1': inf, '3': 0, '2': 2, '5': 6, '4': 11, '7': 10, '6': 2, '9': 5, '8': 6}, {'10': '9', '2': '3', '5'
: '8', '4': '9', '7': '5', '6': '3', '9': '6', '8': '6'})
({'10': 6, '1': inf, '3': 7, '2': 9, '5': 5, '4': 0, '7': 9, '6': 1, '9': 4, '8': 5}, {'10': '9', '3': '5', '2':
'3', '5': '8', '7': '5', '6': '4', '9': '6', '8': '6'})
({'10': 5, '1': inf, '3': 2, '2': 4, '5': 0, '4': 11, '7': 4, '6': 2, '9': 5, '8': 6}, {'10': '7', '3': '5', '2':
'3', '4': '9', '7': '5', '6': '5', '9': '6', '8': '6'})
({'10': 5, '1': inf, '3': 6, '2': 8, '5': 4, '4': 9, '7': 8, '6': 0, '9': 3, '8': 4}, {'10': '9', '3': '5', '2':
'3', '5': '8', '4': '9', '7': '5', '9': '6', '8': '6'})
({'10': 1, '1': inf, '3': 22, '2': 24, '5': 20, '4': 31, '7': 0, '6': 22, '9': 25, '8': 20}, {'10': '7', '3': '5'
, '2': '3', '5': '8', '4': '9', '6': '5', '9': '6', '8': '7'})
({'10': 5, '1': inf, '3': 2, '2': 4, '5': 0, '4': 11, '7': 4, '6': 2, '9': 5, '8': 0}, {'10': '7', '3': '5', '2':
'3', '5': '8', '4': '9', '7': '5', '6': '5', '9': '6'})
({'10': 2, '1': inf, '3': 12, '2': 14, '5': 10, '4': 6, '7': 14, '6': 7, '9': 0, '8': 10}, {'10': '9', '3': '5',
'2': '3', '5': '8', '4': '9', '7': '5', '6': '4', '8': '9'})
({'10': 0, '1': inf, '3': inf, '2': inf, '5': inf, '4': inf, '7': inf, '6': inf, '9': inf, '8': inf}, {})
```

1.3. Floyd-Warshall:

1.3.1. Formar las matrices iniciales C y D.

1.3.2. Se toma k=1.

1.3.3. Se selecciona la fila y la columna k de la matriz C y entonces, para i y j, con i≠k, j≠k e i≠j, hacemos:

1.3.3.1. Si (Cik + Ckj) < Cij → Dij = Dkj y Cij = Cik + Ckj

1.3.3.2. En caso contrario, dejamos las matrices como están.

1.3.3.3. Si k ≤ n, aumentamos k en una unidad y repetimos el paso anterior, en caso contrario paramos las iteraciones.

1.3.4. La matriz final C contiene los costes óptimos para ir de un vértice a otro, mientras que la matriz D contiene los penúltimos vértices de los caminos óptimos que unen dos vértices, lo cual permite reconstruir cualquier camino óptimo para ir de un vértice a otro.

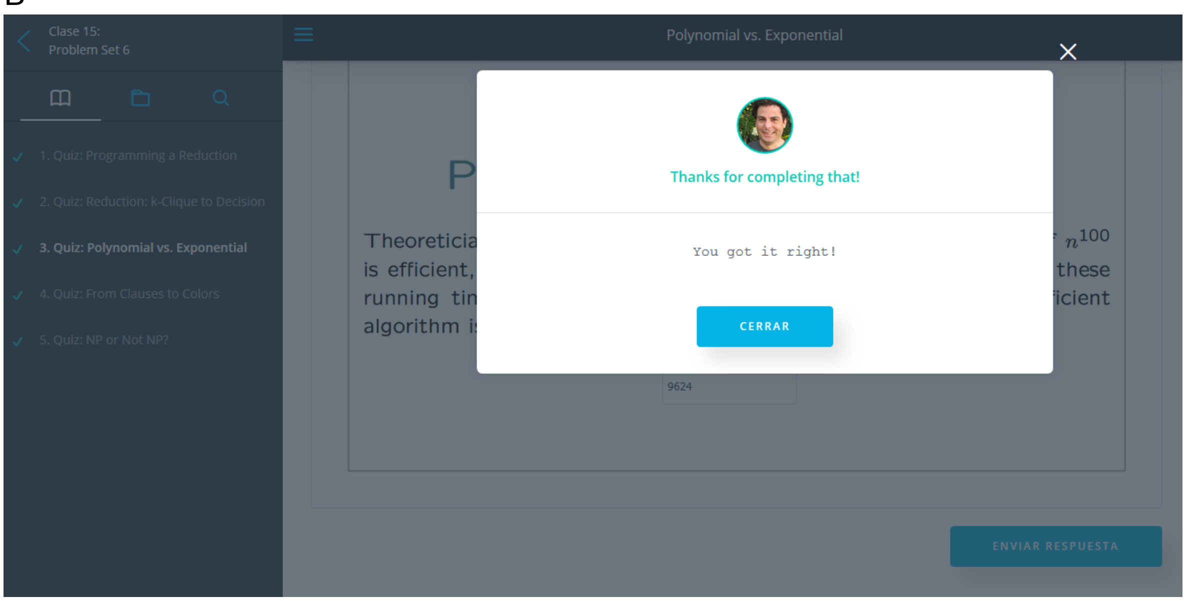
1.3.5. La matriz resultante es:

```
       1    10    3    2    5    4    7    6    9    8
('1', [40, 15, 8, 10, 14, 10, 18, 10, 13, 14])
('10', [inf, 0, inf, inf, inf, inf, inf, inf, inf, inf])
('3', [inf, 7, 0, 2, 6, 11, 10, 2, 5, 6])
('2', [inf, 11, 8, 0, 6, 17, 10, 8, 11, 12])
('5', [inf, 5, 2, 4, 0, 11, 4, 2, 5, 6])
('4', [inf, 6, 7, 9, 5, 0, 9, 1, 4, 5])
('7', [inf, 1, 22, 24, 20, 31, 0, 22, 25, 20])
('6', [inf, 5, 6, 8, 4, 9, 8, 0, 3, 4])
('9', [inf, 2, 12, 14, 10, 6, 14, 7, 0, 10])
('8', [inf, 5, 2, 4, 0, 11, 4, 2, 5, 0])
```

2. Udacity set 6:

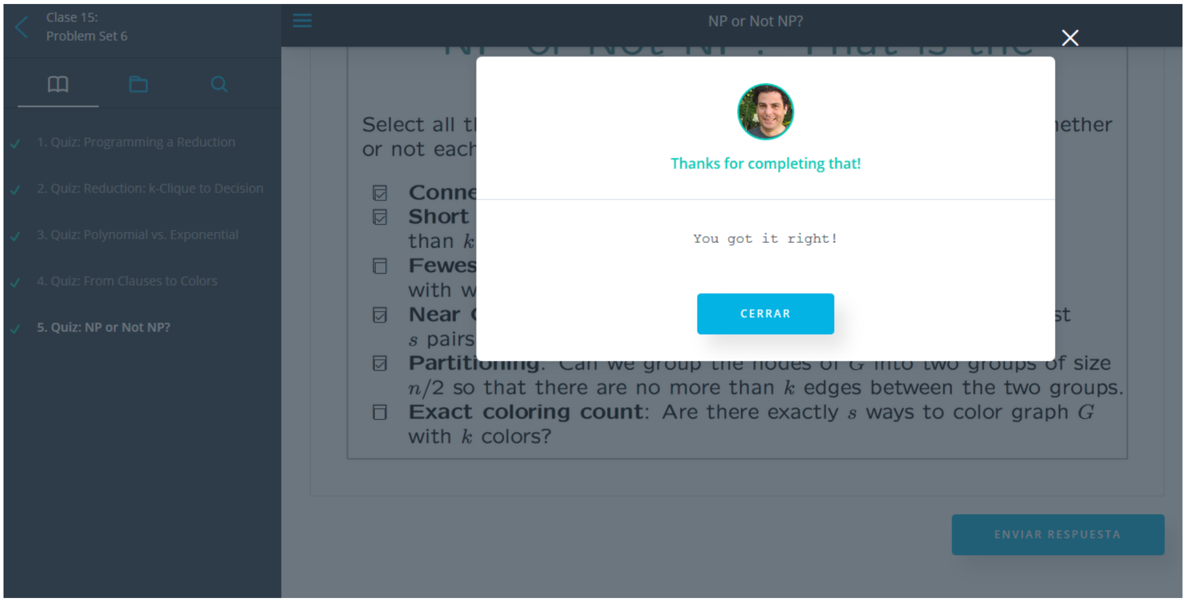
2.1. A

2.2. B

2.3. 

2.4. D

2.5.



Clase 15:
Problem Set 6
1. Quiz: Programming a Reduction
2. Quiz: Reduction: k-Clique to Decision
3. Quiz: Polynomial vs. Exponential
4. Quiz: From Clauses to Colors
5. Quiz: NP or Not NP?
NP or Not NP?
Select all t
or not each
hether
Conne
Short
than k
Fewes
with w
Near
s pairs
st
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?
Thanks for completing that!
You got it right!
CERRAR
ENVIAR RESPUESTA