CX2101 Algorithm Design and Analysis

Tutorial 6
Introduction to NP

(Week 13)

Q1: Is this problem in the class of P or NP? Justify your answers.

Given a network of cities G and a positive integer k. Are the shortest paths between all pairs of cities not longer than k?

Dijkstra's algorithm is able to compute the shortest path from a single vertex to all other vertices in $O(n^2)$ time.

Running Dijkstra's algorithm from every vertex will find the shortest paths between all pairs of vertices in $O(n^3)$ time.

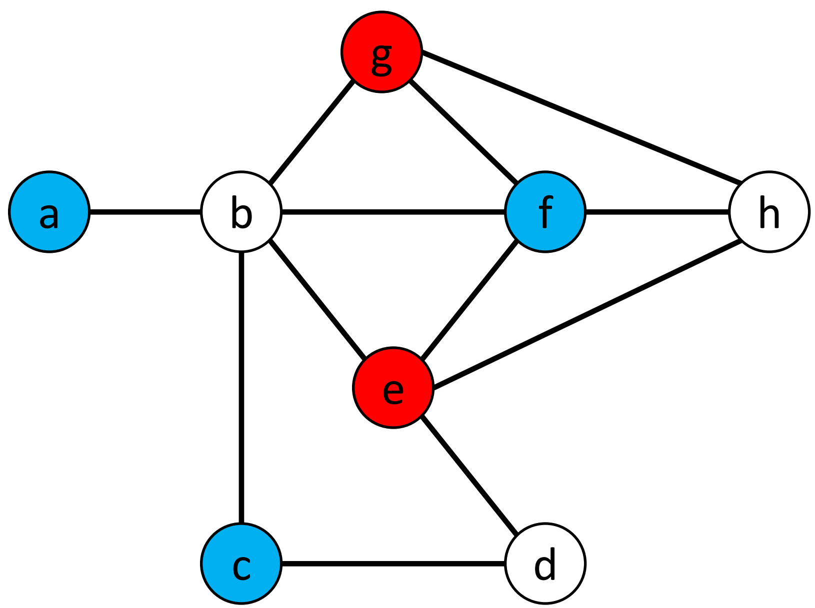
So checking all the shortest paths can be done in $O(n^3)$ time.

Therefore this is a P problem.

Note the **Floyd-Warshall Algorithm** will compute the all-pairs shortest paths more elegantly in $O(n^3)$ time.

Q2: Show that the K-colouring problem is in NP.

Given a graph G = (V, E), where V is the set of vertices and E is the set of edges, and a positive integer k. Is there a way to colour the vertices of the graph using k colours or less such that adjacent vertices have different colours?



Q2: Show that the K-colouring problem is in NP.

Given a graph G = (V, E), where V is the set of vertices and E is the set of edges, and a positive integer k. Is there a way to colour the vertices of the graph using k colours or less such that adjacent vertices have different colours?

Prove that K-colouring is in NP:

- Given a plan to colour the n vertices, the verifier accepts the solution if
- The number of colours is $<= k$ --- $O(n)$
- Each vertex has a colour --- $O(n)$
- For each vertex, the colour of its neighbours is different from its own. --- $O(n^2)$
- Thus the solution can be verified in n^2 time.

Q3: Why do we say NP-Complete problems are the hardest problems in NP?

1. A problem D is **NP-complete** if it is in NP and every problem Q in NP is reducible to D in polynomial time.

2. This means that if an NP-complete problem D can be solved in a certain amount of time, e.g. O(f(n)), every NP problem can be solved in O(f(n)) time. So no NP problem is harder than an NP-complete problem D.

Q4: Implementing shortestLinkTSP()

Implement the shortestLinkTSP() algorithm below (slide 29 of lecture notes) to find a TSP tour in graph G. You may consider using a minimizing heap, a union-find data structure and other data structures in your implementation of the algorithm.

1. A minimizing heap pg is used to store the edges of G, the keys of the nodes are the edge weights.
2. A union-find data structure is used to store connected vertices (vertices already in some fragments of TSP tours)
3. An adjacency matrix/list representation of a graph C is used to store the edges chosen for the TSP tour.
4. An array edgeCount to store the number of edges incident on each vertex v in C

shortestLinkTSP(V, E, W)

{ **pq** = minimizing heap of the edges of G;

id = array in the union-find structure, each vertex v in its own component, i.e. id[v] = v;

initialise all elements in **edgeCount** to 0;

C = empty; // C is a graph with no edges

```
   while (no. of edges in C < n - 1)  {
          vw = getMin(pq);
          deleteMin(pq);
          if (not connected(vw.from, vw.to) and
             edgeCount[vw.from] < 2 and edgeCount[vw.to] < 2)  {
                 C[vw.from][vw.to] = 1;
                 no. of edges in C ++;
                 union(vw.from, vw.to);
                 edgeCount[vw.from] ++;
                 edgeCount[vw.to] ++;        }
    }
   add edge connecting the end points whose edgeCounts are 1
to C;
   return C;                  }
```

Q5: Heuristic for the chain matrix multiplication

Greedy heuristic algorithms are often used to solve problems because of its simplicity. Design a greedy heuristic method to solve the chain matrix multiplication problem where array d is used to store the dimensions of *n* matrices.

A Greedy Method

i) use an array to record the dimensions.

E.g $A_1 x\ A_2 x\ A_3 x A_4$

	0	1	2	3	4
d	30	1	40	10	25

ii) choose the multiplication of two matrices whose cost is the minimum at each step:

$A_2 x\ A_3$ first:

0	1	2	3	4
30	1	~~40~~	10	25

First, min(30x1x40, 1x40x10, 40x10x25) $\Rightarrow$ 1x40x10 is the minimum

Second, min(30x1x10, 1x10x25) $\Rightarrow$ 1x10x25 is the minimum

Last: 30x1x25

Total : 1.40.10 + 1.10.25 + 30.1.25 = 1400

iii) works in most cases except some sequences of 3 matrices (i.e. 2 matrix multiplications) e.g.

$A_1 x\ A_2 x\ A_3$: 10x1x10x15 $\Rightarrow$ 10x1x10 + 10x10x15

- Typically, dynamic programming algorithms are more expensive than greedy algorithms
- so DP is used only when no greedy strategy can be found to deliver the optimal solution