

Problem 7.21. Let G represent an undirected graph. Also let

$$SPATH = \{\langle G, a, b, k \rangle \mid G \text{ contains a simple path of length at most } k \text{ from } a \text{ to } b\},$$

and

$$LPATH = \{\langle G, a, b, k \rangle \mid G \text{ contains a simple path of length at least } k \text{ from } a \text{ to } b\}.$$

Part a. Show that $SPATH \in P$.

Proof. We show that $SPATH \in NP$ by presenting a polynomial time algorithm that decides $SPATH$. A polynomial time algorithm M for $SPATH$ operates as follows.

$M =$ “On input $\langle G, a, b, k \rangle$:

1. Unmark all nodes.
2. Assign each node a value of ∞ .
3. Mark a , and set its value 0.
4. Let integer $d = 0$.
5. Repeat until no additional nodes are marked:
 6. Let $d = d + 1$.
 7. Scan all edges of G . If an edge (u, v) exists between a marked node u and an unmarked node v , then mark v and set value of v to d .
8. If b is marked and value of b is at most k , then *Accept*. Otherwise *reject*.”

Now we analyze this algorithm to show that it runs in polynomial time. Obviously, stages 1 to 4 and stage 8 are executed only once. Stage 7 runs at most m times because each time except the last it marks an additional node in G , where m is the number of nodes in G . Stage 6 also runs at most m times. Thus, the total number of stages used is at most $1 + 1 + m + m$, giving a polynomial in the size of G .

Stages 1 to 4 along with stages 6 and 8 are easily implemented in polynomial time on any reasonable deterministic model. Stage 7 involves a scan of the input and a test of whether certain nodes are marked and an update of node values, which also is easily implemented in polynomial time. Hence M is a polynomial time algorithm for $SPATH$. \square

Part b. Show that *LPATH* is NP-complete.

Proof. First, we need to show that $LPATH \in NP$, which is easy as certificate is the path. Next, to show that all problems in NP are polynomial time reducible to *LPATH*, we show $3SAT \leq_p LPATH$. We show how to construct an integer k and an undirected graph G with two nodes, s and t , where a simple path of length at least k exists between s and t , iff ϕ is satisfiable.

Let ϕ be any Boolean formula in 3CNF containing m clauses:

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_m \vee b_m \vee c_m).$$

where each a , b and c is a literal x_i or \bar{x}_i , and x_1, x_2, \dots, x_n are the n variables of ϕ . Now we show how to convert ϕ to G . The graph contains nodes for literals, and each node's label is the same as the literal that it represents.

1. Repeat for each literal $l = a_1, b_1, c_1$ of the first clause C_1 in ϕ .
2. Add a node for literal l .
3. Repeat for each subsequent clause C_2, C_3, \dots, C_m :
4. Add nodes for the three literal a_i, b_i and c_i in clause C_i .
5. If $i = 2$, then add an edge between node l and any non-conflicting nodes added in step 3.
6. If $i > 2$, then add an edge between a_{i-1} and any node added in step 3 that does not conflict with a_{i-1} and all the nodes reachable from a_{i-1} . Repeat this step for nodes b_{i-1} and c_{i-1} .
7. Add node s . Add edges between node s and nodes a_1, b_1 and c_1 .
8. Add node t . Add edges between node t and all nodes for literals a_m, b_m and c_m .
9. $k = m + 1$.

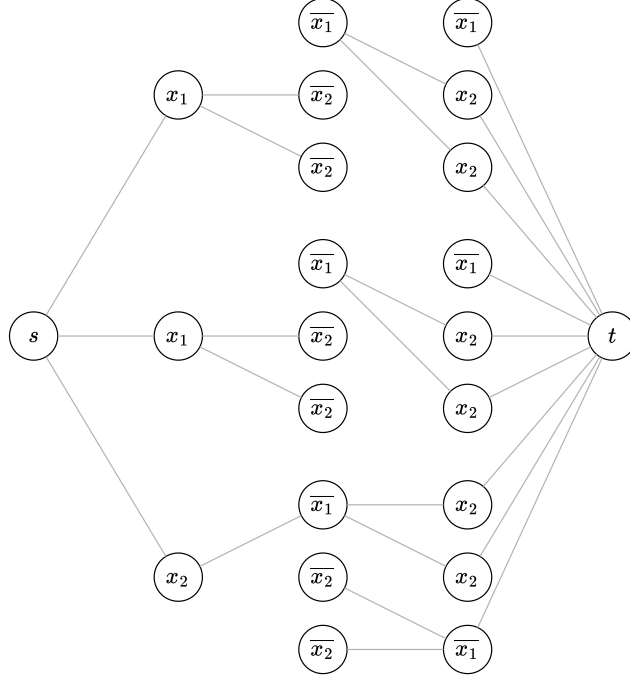
Now we analyze this algorithm to show that it runs in polynomial time. Obviously, stages 7, 8 and 9 are executed only once. Stage 2 runs 3 times. Stages 4, 5 and 6 execute at most m times. Thus, the total number of stages used is at most $1 + 1 + 1 + 3 + m + m + m$, giving a polynomial in the size of ϕ . All stages are easily implemented in polynomial time on any reasonable deterministic model.

Now we demonstrate why this construction works. We show that graph G has a simple path of length at least k from s to t , iff ϕ is satisfiable.

Suppose that ϕ has a satisfying assignment. In that satisfying assignment, at least one literal is true in every clause. In graph G , there is an edge between node s and nodes of literals a_1, b_1 and c_1 of first clause. One of these literal is true, say c_1 is true. Node c_1 would have an edge with at least one of the nodes a_2, b_2 and c_2 of second clause as one of these literals must be true, which means that they cannot all be \bar{c}_1 . Therefore, there is at least one literal among a_2, b_2 and c_2 that does not conflict with c_1 , say b_2 , and node c_1 has an edge to node b_2 . Similarly, node b_2

would have an edge to at least one of the nodes for literals of next clause and so on. Nodes for literals a_m , b_m and c_m have an edge to node t , so there is a path of length at least $m+1$ in G from s to t .

Suppose that G has a path of length at least k from s to t . No two nodes on this path conflict each other. Therefore, intermediate nodes n_1, n_2, \dots, n_{k-1} on the path $(s, n_1, n_2, \dots, n_{k-1}, t)$ give a satisfying assignment for ϕ .



Graph G with $k = 4$ for
 $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2).$

□