Distributed Systems Assignment

s1140740

1 Prove $a \to b \iff V(a) \leq V(b)$.

1.1 Proof of $a \to b \implies V(a) \leq V(b)$ by contrapositive

Assume $a \to b \land \neg(V(a) \leq V(b))$. By definition, this means that $\exists j.\, V(a)[j] > V(b)[j]$. For this to be true there had to be two successive events u and v between a and b such that $V(u)[j] > V(v)[j]$. There are two cases:

1. Events u and v were successive events in same process. However, by definition after every local event of a process i, $V_i[i] = V_i[i] + 1$.
2. Event u was the "send" event of message M from process i and event v was the "receive" event of message M at process j. By definition, $V(u)$ was attached to M. Furthermore, after process j receives the message as event v it performs the following steps:
   1. $V_j[k] = max(V_j[k], V_i[k])$, for $k = 1, 2, ..., n$
   2. $V_j[j] = V_j[j] + 1$

Both cases imply $V(u) < V(v)$ and thus we arrived at a contradiction, completing the proof. ■

1.2 Proof of $V(a) \leq V(b) \implies a \to b$ by contrapositive

Assume $V(a) \leq V(b) \land \neg(a \to b)$. This is equivalent to

$$(b \to a \land V(a) \leq V(b)) \lor (b \parallel a \land V(a) \leq V(b))$$

In section 1.1 we have shown that $b \to a \implies V(b) \leq V(a)$, thus:

$$(V(b) \leq V(a) \land V(a) \leq V(b)) \lor (b \parallel a \land V(a) \leq V(b))$$

$$V(a) = V(b) \lor (b \parallel a \land V(a) \leq V(b))$$

However, $V(a) = V(b) \implies a = b$ since we change the vector clock at every event. Thus,

$$(b \parallel a) \land V(a) \leq V(b)$$

By definition, $b \parallel a \iff \neg(b \to a) \land \neg(a \to b)$, therefore:

$$\neg(b \to a) \land \neg(a \to b) \land V(a) \leq V(b)$$

From section 1.1, we have

$$\neg(V(b) \leq V(a)) \land \neg(V(a) \leq V(b)) \land V(a) \leq V(b)$$

and thus we arrived at a contradiction, completing the proof. ■

2 Inductive proof on the position of the request

2.1 Base case

Request is at position 1 in the queue and thus the process can access the resource and satisfy the request.

2.2 Induction hypothesis

The request eventually gets satisfied at position k.

2.3 Inductive step

Request is at position $k+1$. Let the request at first position belong to process i and let us call the request R_i. Since R_i is at the first position in the queue, all the previous processes accessing the resource are finished with it, otherwise we would have their requests in our queue before R_i (we add a request to the queue whenever we get a `REQUEST` message and remove it only once we receive a `RELEASE` message for it). Thus, there are three cases:

1. Process i is currently accessing the resource. Since we assume processes do not fail, this means that it will eventually finish accessing it and when it does it will send us a `RELEASE` message and R_i will be removed from our queue and our request will be in position k.
2. Process i has already finished accessing the resource. This implies we have not yet received the `RELEASE` message. Channels do not fail so we will eventually receive the message and remove R_i from our queue and our request will be in position k.
3. Process i has not started accessing the resource. This implies R_i is not yet at the top of the queue of process i (otherwise it would just access the resource). However, since we have shown that no process with a request before R_i can be accessing the resource this means that process i just has not received the `RELEASE` message from the last process accessing the resource. Once it receives this message it will pop that process's request of its queue and start accessing the resource. Logic in case 1 can then be followed to show that our request will advance to position k.

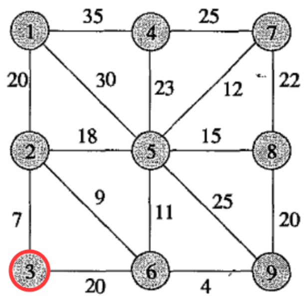
Using induction hypothesis we can show the request will eventually get satisfied in all three cases.

3 Weighted diameter

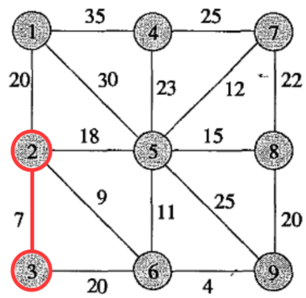
The weighted diameter of this graph is 7. The path realising this diameter is $A \to C \to E \to G \to H$. If the graph was unweighted, the diameter would be 4 and the corresponding path would be $A \to C \to F \to G \to I$.

4 Prim's algorithm

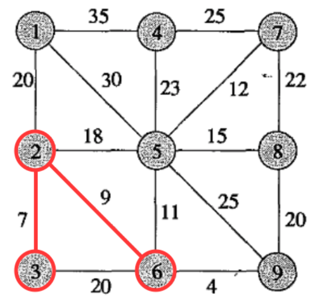
See figure 1 below for Prim's algorithm.



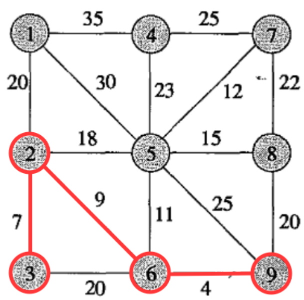
(a) Step 1



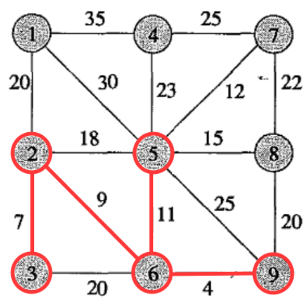
(b) Step 2



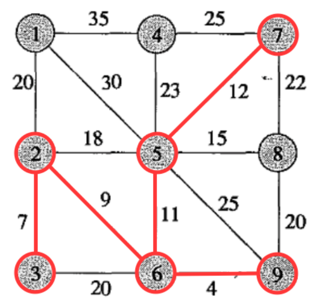
(c) Step 3



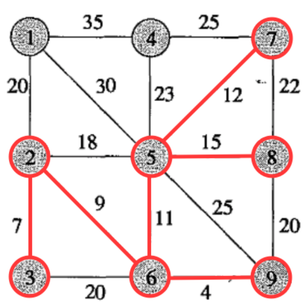
(d) Step 4



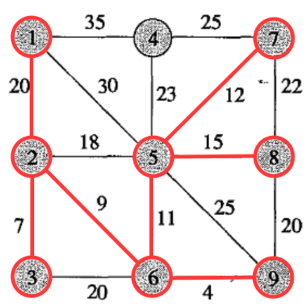
(e) Step 5



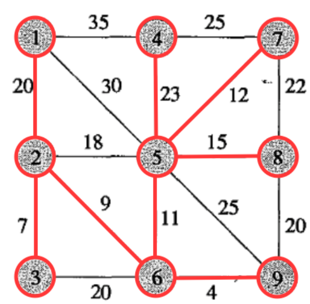
(f) Step 6



(g) Step 7



(h) Step 8



(i) Step 9

Figure 1: Prim's algorithm