Home Assignment 2 - Project Groups 88 - Souptik Paul, Venkata Say Dinesh Uddagiri

Self-stabilizing maximum matching (due: September 22, 2022)
Let p0,...,pn−1 be n processors on a directed ring that Dijkstra considered in his self- stabilizing token circulation algorithm (in which processors are semi-uniform, i.e., thereis one distinguished processor, p0, that runs a different program than all other processorsbut processors have no unique identifiers). Recall that in Dijkstra's directed ring, eachprocessor pi can only read from pi−1 mod n's shared variables and each processor can use shared variables of constant size. Design a deterministic self-stabilizing matching algorithm that always provides an optimal solution. Please give a clearly written pseudo-code, define the set of legal executions, provide a correctness proof with all the arguments needed to convince the reader that the algorithm is correct and self-stabilizing. Show that your algorithm is always optimal after convergence. What is the number of states (in the finite state machine that represents each processor) that your algorithm needs? Prove your claims.

Assumption:-

1. We assume that for each processor P(i), there exist a register r(i), which is a pointer register, that can store(or point) the address of another Processor.
2. In any given graph or diagram we have, an edge is considered a part of the maximal matching set, if a processor uses the edge to point towards a subsequent edge and the subsequent edge also points back as the initial edge(this is explained in the diagrams below, its is showcased as two parallel arrows in different directions).

Pseudo Code

```
1 do forever
2 for i = 0
3   if (r(n-1) == P(i))) Then
4       r(i) = Null
5    Else
6        r(i) = P(i+1) mod n
7 for i ≠ 0
8   if(r(i-1)==P(i)) Then
9       r(i) = P(i-1)
10    else
11       r(i) = P(i+1) mod n
```

Legal Executions:-

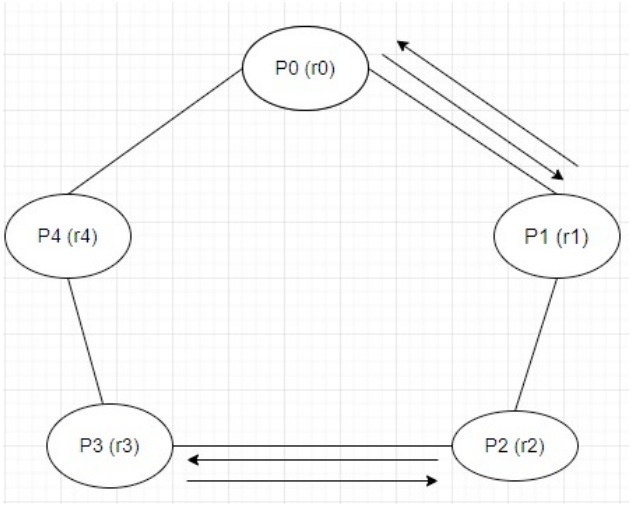
As the algorithm works on Dijkstra's directed ring, let LE(MM) be the set of all legal executions in which the processors can access the register of its previous processors infinitely often.

The task MM can be defined as :-
1. Exactly one processor can change the value of its registers(state), in any one configuration.
2. Every processor can change the value of its registers(state) in infinitely many configurations in every sequence in ME.

3. After a certain set of configurations, the safe state should be achieved, where we can get the the maximal matching for the graph.

So we can say that the set of legal executions is the respective portion of the pseudo running during each configuration. For example, let c be the configuration when P(0) is checks the condition that is r(n-1) == P(0) and accordingly updates the value of its own register to either NULL or the address of its subsequent register. The subsequent configuration can be c1, where P(1) executes its respective condition and updates the value of its own register to be the address of its previous or subsequent register. This process continues infinitely and all the processors change their value infinitely often.

Proof



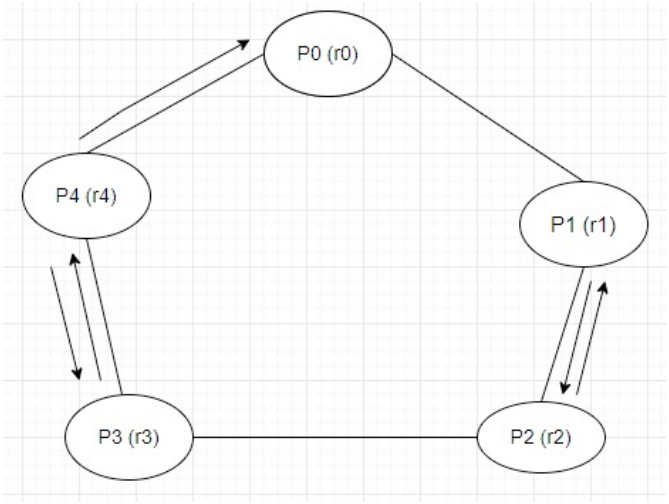
To prove the functioning of the algorithm, let us define the safe state of this Maximal Matching algorithm. For n processors, c(n) states, there should be a set of edges, which is a matching M, such that adding an edge to M, which is in graph but not in M, would make M not a matching. Also M should be the set of maximum possible edges.

To demonstrate the proof, of this algorithm, let us consider the function of this algorithm in few configurations

Case 1: As shown in the diagram, the execution starts at a configuration c, where the condition r(n-1) == P(i) for i=0, is not true. Then in c, the step or action will be that the register of P(0), will point towards P(1), by the condition r(i) = P(i+1) mod n.
Let c1 be the configuration, after the register of P(0) points towards P(1). In this configuration c1, the step or action will be that P(1), will check the condition r(i-1)==P(i), for i=1, which is true. Then update the value of its register r(1) to point back to P(0).
Let the configuration after the updation of register r(1), be c2. In the configuration c2, the step or action will be that P(2), will check condition r(i-1)==P(i), which is false. Then update the value of its register r(2), using r(i) = P(i+1)mod n, to point towards, P(3).
Let the configuration after the updation of the register r(2) be c3. In the configuration c3, the step will be the same as c1, and register r(3) will point back to P(2).
Let the configuration after the updation of r(3) be c4. In the configuration c4, the step will be the same as c2, and the register r4 will be point towards P(0).
Let the configuration after the updation of register r(4) be c5.

Let us consider the state of the graph at configuration c5. The maximal matching can be obtained as M(max) = {P0P1,P2P3}, also showcased in the diagram(we are only considering edges, joining processors, whose register point to each other).The maximal matching was obtain after the configuration c5 was reached, hence we can say that he number of configurations required for obtaining the maximal matching is n. Hence we can say that c5 is a safe configuration.

Case 2: Subsequently let us consider another case, where the configuration begins after c5. To handle this case separately, we are considering the configuration to be an arbitrary c', where c'= c5(for reader understanding).



In the configuration c', the condition r(n-1) == P(i) for i=0, is true, hence the register r(0) of processor P(0), will be updated as NULL(it does not point to any register) using the statement r(i) =NULL.
Let the configuration after the updation of the register r(0) be c1'. In the configuration c1', the step or action will be that P(1), will check condition r(i-1)==P(i), which is false. Then update the value of its register r(1), using r(i) = P(i+1)mod n, to point towards, P(2).
Let the configuration after the updation of the register r(1) be c2'. In this configuration c2', the step or action will be that P(2), will check the condition r(i-1)==P(i), for i=1, which is true. Then update the value of its register r(2) to point back to P(1).
Let the configuration after the updation of the register r(2) be c3'. In the configuration c3', the step or action will be that P(3), will check condition r(i-1)==P(i), which is false. Then update the value of its register r(3), using r(i) = P(i+1)mod n, to point towards, P(4).
Let the configuration after the updation of the register r(3) be c4'. In this configuration c4', the step or action will be that P(4), will check the condition r(i-1)==P(i), for i=1, which is true. Then update the value of its register r(4) to point back to P(3).
Let the configuration after the updation of r(4) be c5'.

Let us consider the state of the graph at configuration c5'. The maximal matching can be obtained as M(max) = {P1P2,P3P4}, also showcased in the diagram(we are only considering edges, joining processors, whose register point to each other). The maximal matching was obtain after the configuration c5' was reached, hence we can say that he number of configurations required for obtaining the maximal matching is n. Hence c5' is a safe configuration.

If we observe the above two cases we can say that, the configuration c5' from case 2, is equal to the configuration c for case 1. Hence the same execution will take place infinitely.
So from the two cases we can summarise that, after n^2 configurations, the system will return to the safe configurations c or c5', with the same Maximum matching sets {P0P1,P2P3} or {P1P2,P3P4} . Also the configuration after n configuration or steps, the system will retain the safe configuration. Hence from the above cases and the definition of self stabilisation we can conclude that the algorithm is self stabilising.