COP5615
Operating System Principles
Summer 2004

Project 4

Date Assigned: July 9th, 2004
Date Due: Midnight, July 30th, 2004

Introduction

The purpose of this project is for you to implement a distributed algorithm and to learn Java MulticastSocket.
In class we mentioned several leader election algorithms. One of them is to form a Minimum Spanning Tree (MST) and every member (node of the tree) agrees on the root as the leader. In this project, you will implement Sollin's MST algorithm. You can obtain information regarding the algorithm from Internet. For your convenience, we describe the algorithm as follows: (**Note: we will apply it in a distributed system where every node of the graph is an agent (a multi-threaded process). The forming of the tree happens asynchronously.**)

We can view Sollin's algorithm as a hybrid version of Kruskal's and Prim's algorithms for MST (see data structure text books).Sollin's algorithm maintains a collection of trees spanning the nodes N_1, N_2, …, and adds arcs to this collection. However, at every iteration (**Note: there is no iteration concept in our distributed setting.**) , it adds minimum cost arc emanating from these trees, an idea borrowed from Prim's algorithm. Sollin's algorithm repeatedly performs the following two basic operations:

> *nearest-neighbor* (N_k, i_k, j_k). This operation takes as an input a tree spanning the nodes N_k and determines an arc (i_k, j_k) with the minimum cost among all arcs emanating from N_k. To perform this operation we need to scan all the arcs in the adjacency lists of nodes in N_k and find a minimum cost arc among those arcs that have one endpoint not belonging to N_k.
> *merge* (i_k, j_k). This operation takes as an input two nodes i_k and j_k, and if the two nodes belong to two different trees, then merges these two trees into a single tree.

Using these two basic operations, we state Sollin's algorithm as

> **procedure** Sollin
> **begin**
>   **for** each $i \in N$ **do** $N_i := \{i\}$;
> $T^* := \emptyset$;
>   **while** $|T^*| < (n - 1)$ **do**
>   **begin**
>   **for** each tree N_k **do** *nearest-neighbor* (N_k, i_k, j_k) ;
>   **for** each tree N_k **do**
>   **if** nodes i_k and j_k belong to different trees **then**
>   *merge* (i_k, j_k) and update $T^* := T^* \cup \{(i_k, j_k)\}$;
>   **end**;

end;

We illustrate Sollin's algorithm on a numerical example. As shown in the Fig.1(b), Sollin's algorithm starts with a forest containing five trees. Each tree is a singleton node. This figure also shows that the least cost arc emanating from each tree. We next perform mergings, reducing the number of trees to only two (see Fig.1(c)). The least cost arc emanating from these two trees is (3,4), and when we add this arc, we obtain the spanning tree shown in Fig.1(d). The algorithm now terminates.

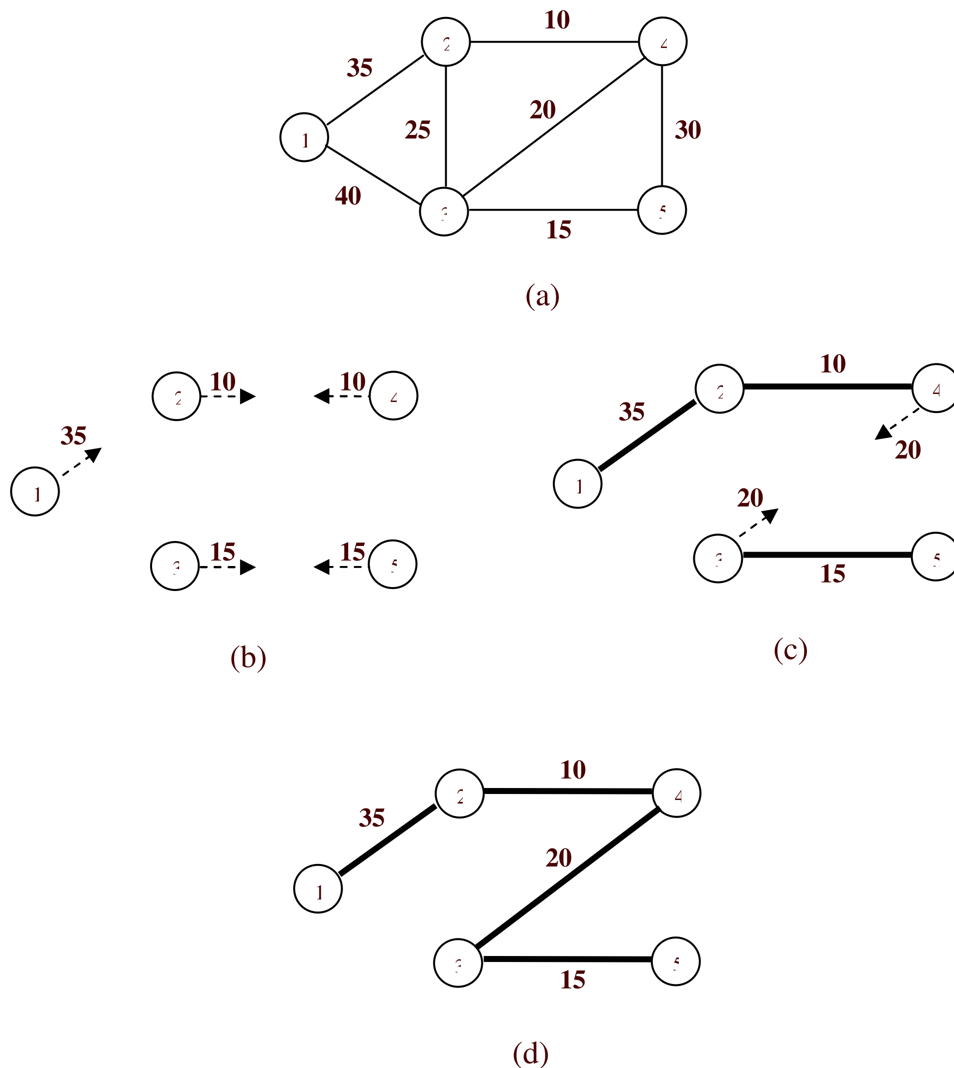

Figure-1 Illustration of Sollin's algorithm on an example.

System

All system information is specified in a provided file, system.properties, as follows:

mst.numberOfAgents = 4

mst.host0 = coralsnake.cise.ufl.edu
mst.port0 = 12310
mst.edges0 = 1,10;2,20;3,5

mst.host1 = coralsnake.cise.ufl.edu
mst.port1 = 12320
mst.edges1 = 0,10;2,30;3,8

mst.host2 = coralsnake.cise.ufl.edu
mst.port2 = 12330
mst.edges2 = 0,20;1,30;3,40

mst.host3 = coralsnake.cise.ufl.edu
mst.port3 = 12340
mst.edges3 = 0,5;1,8;2,40

mst.sleepMaxLength = 10000
mst.socketTimeout = 5000

mst.multicastPort = 4444
mst.multicastGroupId = 230.0.0.1

In the above example, we have a graph of 4 nodes (0, 1, 2, 3) consisting 6 edges (0-1, 0-2, 0-3, 1-2, 1-3, 2-3) with weights (10, 20, 5, 30, 8, 40). Utilities are provided here for you to retrieve the information of the properties. They are provided simply to facilitate your implementation. It is not required to use them. You will need to write a Java class named MSTAgent.java representing a node that takes one integer argument as the identifier of the node. You also need to write another Class named Start.java that triggers the algorithm by making a multicast using Java multicastSocket and then exits. Here is how the system is started: for the above example, you will need to open five Unix shells of the specified hosts. One runs “Java MSTAgent 0”, one runs “Java MSTAgent 1”, one runs “Java MSTAgent 2”, one runs “Java MSTAgent 3”, finally runs “Java Start” in the 4th shell. The first thing that an agent does is to receive a multicast dummy message from the program “Start”. After an agent receives the broadcast, it first sleeps for random mini seconds limited to the property “mst.sleepMaxLength” before it starts to run the algorithm. Agents in the system have peer-to-peer relationship communicating via TCP/IP (Java Socket). They must be multi-threaded so that they can play both client and server roles.

Protocol as to decide the root node while two trees are merging is up to your design.

Race Condition and Dead Lock

For any two trees, at any moment, only one edge across the two trees is allowed to form. Also for a tree, at any moment, only one edge of the tree is allowed to expand. Special cares need to be done to avoid the race condition and possible dead lock during the forming of the MST.

Output

In a Unix shell where agent 0 runs, we expect to see outputs like the following.((After the algorithm terminates, every process exits. The root of the resulting MST dumps the tree information as follows before it exits, and which agent will be the root depends on your implementation)

In the Unix shell where agent 0 runs, we expect to see outputs like the following:

Agent 0 is making a request to merge edge 0-3
Agent 0: The request is rejected due to confliction
Agent 0 is making a request to merge edge 0-1
Agent 0: The request is rejected due to confliction
Agent 0 cannot find any other edge that can be merged
The MST is formed. System exits

In the Unix shell where agent 1 runs, we expect to see outputs like the following:

Agent 1 is making a request to merge edge 1-3
Agent 1: The request is rejected due to confliction
Agent 1 is making a request to merge edge 1-3
Agent 1: Request is done, now the new root is 1
Agent 1 cannot find any other edge that can be merged
The construction of the tree has been completed, send an exit message to all members
--OUTPUT-----------------------------
The MST is formed. The tree is the following:
Agent 1(root), Children: Agent 3
Agent 3, Children: Agent 0
Agent 0, Children: Agent 2
Agent 2, Children: No Children
System exits

In the Unix shell where agent 2 runs, we expect to see outputs like the following:

Agent 2 is making a request to merge edge 2-0
Agent 2: The request is rejected due to confliction
Agent 2 is making a request to merge edge 2-0
Agent 2:Request is done, now the new root is 2
Agent 2 cannot find any other edge that can be merged
The MST is formed. System exits

In the Unix shell where agent 3 runs, we expect to see outputs like the following:

Agent 3 is making a request to merge edge 3-0
Agent 3:Request is done, now the new root is 3
Agent 3 is making a request to merge edge 3-1
Agent 3: The request is rejected due to confliction
Agent 3 cannot find any other edge that can be merged
The MST is formed. System exits

Assumptions

You can assume the weights of all edges in the graph are different so that the resulting MST is unique (the root may be different among different runs). The complexity of your implementation will not be concerned.

Requirements

- System configuration should be read from `system.properties` file. Parameters in this file should not be hardcoded into the source code.
- All programs and threads should terminate gracefully after completing their tasks. No run-away process is allowed to exist. Please visit the project FAQ page for downloading the small script used by us for cleaning run-away processes. Please modify it to suit your needs or write your own scripts. Thanks.
- The whole system should be started with programs named `Start.java` and MSTAgent.java
- You should not submit anything other than makefile, report and java files.
- The requirements in Project Overview and Requirements document should be followed strictly.

Submission

Please follow the submission guidelines in Project Overview and Requirements document .

Resources

You may find tutorials and sample code on general Java programming, TCP/IP programming in Java and Threads by following the link http://java.sun.com and searching for specific information you need. You may also find many other relevant resources by searching the web.