Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Software Technologies
Assignment 4 (Hadoop)

Minimum Spanning Tree and Min Cuts in Large Graphs

Due Date: 19/03/2014

In this assignment, you are required to develop a mapreduce implementation on Hadoop for computing MST (Minimum Spanning Forest) and Min Cut in the input graph using ideas from "Filtering: A Method for Solving Graph Problems in MapReduce" by Lattanzi et al (SPAA 2011).

Input: The input is a weighted undirected graph $G = (V, E)$ given as new line separated list of weighted edges where each edge is represented by the source vertex, destination vertex and the edge weight all separated by tabspaces. Vertices can be assumed to be numbered as $1, \ldots, n$ where n is the number of vertices.

Implementation ideas and details:

MST:

1. The algorithm works in a distributed fashion using the observation that the heaviest edge in any cycle of the graph can be removed from the input edge set without altering the MST cost. (Prove this.). Applying this filtering procedure in a distributed fashion, the graph is pruned to a smaller size. The procedure is repeated until the resulting pruned edge set fits into a single machine and then the final MST is computed in a single machine.

2. The input program should take as command line arguments: the name of the graph file, name of the output file and η as given in Section 2.4 of the paper.

3. If $|E| \leq \eta$ then the MST for that graph is computed in a single machine. Otherwise the MST is computed in a distributed manner as follows.

4. If $|E| > \eta$ then the initial mapper use a universal hash function to map each edge (u, v) to one of l buckets, where $l = |E|/\eta$. This ensures that each bucket contains roughly η edges. Each $i \in \{1, \ldots, l\}$ denotes one reducer.

5. Edges mapped to a bucket $i \in \{1, \ldots, l\}$ goes to the same reducer.

6. Each reducer compute the MST forest for the subgraph defined by the edges mapped to it.

7. Observe that all edges that are removed by any reducer (while computing the MST for its input graph) happen to be the heaviest edge in some cycle in the original graph as well. Hence this is a valid pruning (edge removal) for the original graph also.

8. Now create a pruned set of edges by combining the MST forest edges in each of the reducers.

9. Go to step 3 and repeat the above steps on this pruned set of edges.

10. Observe that for a undirected graph, the each tree in the final MST forest is also a maximal connected component of the input graph G.

Min Cut:

1. We consider undirected unweighted graph. A partition of the vertex set into two parts give rise to a subset of edges called the *cut set*, where each edge in the cut set has one vertex in one part and the other vertex in the second part. A min cut is the cut set with smallest cardinality. You need to write MapReduce tasks to compute min cut of a graph in a distributed fashion.

2. Read about Karger's randomized mincut algorithm using random edge contraction (See the book 'Randomized Algorithms' by Motwani and Raghavan). The algorithm does a sequence of edge contractions (where loops arising after edge contraction is removed and parallel edges are retained), where the edge that is contracted is chosen uniformly at random. After each edge contraction, the number of vertices in the graph decreases by one. Finally when the graph has only two vertices, the set of edges going across the two vertices is reported as the cut set. Observe that min cut value do not decrease because of edge contraction. Also, observe that there is a small success probability that this reported cut set is the true min cut set. Now amplify this success probability by repeating the algorithm number of times maintaining the minimum cardinality cut set that has been reported so far in each repeated trial.

3. We will turn this into a distributed implementation using MapReduce as given in the above paper (Section 4). Identify multiple edges to be contracted in one step in a random fashion. Now contract all these edges simultaneously in a distributed manner. The resulting graph has lesser number of edges. If the resulting graph is small enough then run any standard mincut algorithm on this and report the min cut. Other wise repeat the edge contraction phase. Now repeat the whole algorithm multiple times to amplify the success probability.

4. The input program should take five command line arguments: the name of the graph file, name of the output file, a value t between zero and one, value of η (which is described below) and amplification value α which is a positive integer. You may ignore the weight information associated with edge edge in the input graph for Min Cut.

5. Let $G = (V, E)$ be the input graph.

6. If the number of edges $|E| \leq \eta$ then compute the min cut value in a single machine (non distributed) using any standard min cut library. Report the min cut set and the min cut size.

7. Write a mapreduce task that assigns to each edge an independent random number between 0 and 1 and aggregate those edges whose assigned random number is below t. Call this subset of edges as E'. We will now contract all edges of E' simultaneously as given below. The goal now is to compute the contracted graph $G_c = (V_c, E_c)$ obtained by contracting all edges in E' and then removing all self loops.

8. Consider the graph $G' = (V, E')$. Run the previous MapReduce based Minimum Spanning Forest algorithm on G'. Recall that each tree in the spanning forest also correspond to a connected component. Let $C_1, \cdots, C_k$ denote the connected components of G'. Each connected component C_i becomes a single vertex in G_c. Let the edge set V_c of G_c be denoted by $\{1, \cdots, k\}$.

9. Recall that each tree in the MST forest correspond to a connected component in G'. Observe that if we contract all edges in E' then each edge of $E - E'$ that belong to the same connected component will become a self loop in the contracted graph. Hence the only edges of importance in the contracted graph are the edges in $E - E'$ that lie between two different connected components in G'.

10. For each vertex $u \in V$, store its connected component id, denoted by $CC(u)$, which has value i if it belongs to connected component C_i. You may write a MapReduce task to do this.

11. Write a MapReduce task that maps each edge $(u, v) \in E$ to a mapped edge $(CC(u), CC(v))$. To do this, define a universal hash function $h(u)$ which maps a vertex $u \in V$ to $\{0, \cdots, r-1\}$, where r is the number of mapreduce instances you wish to create. Now the mapper maps an edge (u, v) to the reducer $h(u) + h(v)$. In the reducer, if $CC(u) = CC(v)$ then the reducer drops this edge (it is a self loop). The remaining edges correspond to the edge set E_c of the graph G_c.

12. Go to step 5 where the new input graph is $G_c = (V_c, E_c)$ and repeat the same.

13. To construct the min cut set of the original graph, we have to remember the original edge (u, v) for each mapped edge $(CC(u), CC(v)$.

14. Run α instances of the above MapReduce based Min Cut all in parallel. Each instance would report its own solution (cut set and cut value).

15. Find and report the cut set having the minimum cut value (minimum cardinality) among the α solutions produced in the α parallel executions.