Tasks 12, 13 and 15

# Task 12 – Batch Node Delete

This program takes the input received in the form batchnodedelete NODEFILENAME GRAPHDBNAME.

The program which performs the subsequent tasks takes one input at a time. Each input is of type String and contains nodelabel. A new temporary node is created for each nodelabel. The setLabel() method is invoked for this node and the parameter passed in the setLabel() method is the nodelabel. This creates a temporary node which has the nodelabel string as its label. To delete a node from the database which has the nodelabel given in the file, the NID of the node is required. This is provided by initiating a nodeHeapFile scan using the openScan() method of the NodeHeapFile class. The NodeHeapFile object used is the one initiated by graphDB and passed to the program. The scan is run until a node with the Label same as the Label of temporary node is not found.

Once the required node is found, a EdgeHeapFile scan is started. This is done to find the edges which has the obtained node as its source or destination node. All the edges which have the given node as its source or destination node as removed by finding their EID(Edge ID). Once the edges are removed, edgeHeapFile Scan is closed and the node found earlier is deleted from the database. Once this is finished, the nodeHeapFile Scan is closed using closescan function of the NScan class.

Next String is read from the NODEFILENAME file and all the steps described above are performed. This is performed until all the Strings in NODEFILENAME are not read.

# Task 13 – Batch Edge Delete

Implemented by first reading the file and obtaining one line of the EDGEFILENAME File’s content. Each line is split into 3 strings which correspond to EdgeLabel, SourceNode NID, Destination Node NID. An EScan is run on the Edge heap file to search for the corresponding which needs to be deleted. The scan first just checks for the edge Label. The Label obtained from EDGEFILENAME is not used directly as during formation of an edge the formatting of the String Label is changed by adding 0’s to obtained constant number of bytes for each Label. Hence a temporary Edge is created for which the setLabel() is used with the input obtained from the EDGEFILENAME as the parameter. The edge obtained is used to search for the edge with Label consistent with input. Once found, the Source Node NID and Destination Node NID is cross checked with the one obtained in the input File. Once they are cross checked, it proves that this is the edge which needs to be deleted. The edge obtained is deleted by passing its EID to deleteEdge function. The Escan is closed.

The above steps are performed for each line of input in the EDGEFILENAME file given in the command line.

# Task 15 – Edge Query

## For value of Index in command line: 1

## QTYPE 0:

For printing the edges. An IndexScan is run on BTree Index File on edge Labels, in case the file doesn’t exist, an index file is created on the edge Label of edges. The Scan is run until the edge obtained in the scan using getNextEdge function is not null. The Edge obtained in each call of getNextEdge function is printed.

## QTYPE 1:

An IndexScan is run on BTree Index File on edge Labels, in case the file doesn’t exist, an index file is created on the edge Label of edges. The scan is run until the edge obtained in the scan is not null. Each subsequent edge is obtained using getNextEdge function of the IndexScan object. For each edge a Source NID is obtained, this NID is used to obtain the Source Node. This Node is passed to an Array of Nodes. Once the entire Array has been created, the Labels of the Array are sorted in alphanumerical order. The Array is printed subsequently.

## QTYPE 2:

An IndexScan is run on BTree Index File on edge Labels, in case the file doesn’t exist, an index file is created on the edge Label of edges. The scan is run until the edge obtained in the scan is not null. Each subsequent edge is obtained using getNextEdge function of the IndexScan object. For each edge a Destination NID is obtained, this NID is used to obtain the Destination Node. This Node is passed to an Array of Nodes. Once the entire Array has been created, the Labels of the Array are sorted in alphanumerical order. The Array is printed subsequently.

## QTYPE 3:

An IndexScan is run on BTree Index File on edge Labels, in case the file doesn’t exist, an index file is created on the edge Label of edges. The scan is run until the edge obtained in the scan is not null. Each subsequent edge is obtained using getNextEdge function of the IndexScan object. The edge obtained in each iteration is passed to an Array containing edges. These edges are passed to sortEdges function with parameter which sorts the edgeLabels of edges in increasing alphanumerical order. The array containing the edges is printed subsequently.

## QTYPE 4:

An IndexScan is run on BTree Index File on edge Labels, in case the file doesn’t exist, an index file is created on the edge Label of edges. The scan is run until the edge obtained in the scan is not null. Each subsequent edge is obtained using getNextEdge function of the IndexScan object. The edge obtained in each iteration are printed.

## QTYPE 5:

An IndexScan is run on BTree Index File on edge weights, in case the file doesn’t exist, an index file is created on the edge Label of edges. The scan is run until the edge obtained in the scan is not null. Each subsequent edge is obtained using getNextEdge function of the IndexScan object. A check is performed on the edge weights and if they satisfy the condition of being in the lowerbound and upperbound provided in the command line input, then the edge is printed.

## QTYPE 6:

An IndexScan is run on BTree Index File on edge weights, in case the file doesn’t exist, an index file is created on the edge Label of edges. The scan is run until the edge obtained in the scan is not null. Each subsequent edge is obtained using getNextEdge function of the IndexScan object. The source NID and destination NID obtained in each edge is stored in a 2-Dimensional array of NIDs. Looping is performed on this array to obtain the edges which have a same vertex. In case the condition is satisfied, the edge labels of both the edges are printed.

# For value of index in command: 0

## QTYPE 1:

An EScan is run on the object of edgeHeapFile which is initiated by the graphDB ‘s constructor. The scan is run until the edge obtained in the scan is not null. Each subsequent edge is obtained using getNext function of the EScan object. For each edge a Source NID is obtained, this NID is used to obtain the Source Node. This Node is passed to an Array of Nodes. Once the entire Array has been created, the Labels of the Array are sorted in alphanumerical order. The Array is printed subsequently.

## QTYPE 2:

An EScan is run on the object of edgeHeapFile which is initiated by the graphDB ‘s constructor. The scan is run until the edge obtained in the scan is not null. Each subsequent edge is obtained using getNext function of the EScan object. For each edge a Destination NID is obtained, this NID is used to obtain the Destination Node. This Node is passed to an Array of Nodes. Once the entire Array has been created, the Labels of the Array are sorted in alphanumerical order. The Array is printed subsequently.

## QTYPE 3:

An EScan is run on the object of edgeHeapFile which is initiated by the graphDB ‘s constructor. The scan is run until the edge obtained in the scan is not null. Each subsequent edge is obtained using getNext function of the EScan object. The edge obtained in each iteration is passed to an Array containing edges. These edges are passed to sortEdges function with parameter which sorts the edgeLabels of edges in increasing alphanumerical order. The array containing the edges is printed subsequently.

## QTYPE 4:

An EScan is run on the object of edgeHeapFile which is initiated by the graphDB ‘s constructor. The scan is run until the edge obtained in the scan is not null. Each subsequent edge is obtained using getNext function of the EScan object. The edge obtained in each iteration is passed to an Array containing edges. These edges are passed to sortWeights function. The function sorts the Array on weights of the edges. The sorted Array is subsequently printed.

## QTYPE 5:

An EScan is run on the object of edgeHeapFile which is initiated by the graphDB ‘s constructor. The scan is run until the edge obtained in the scan is not null. Each subsequent edge is obtained using getNext function of the EScan object. A check is performed on the edge weights and if they satisfy the condition of being in the lowerbound and upperbound provided in the command line input, then it is printed.

## QTYPE 6:

An EScan is run on the object of edgeHeapFile which is initiated by the graphDB ‘s constructor. The scan is run until the edge obtained in the scan is not null. Each subsequent edge is obtained using getNext function of the EScan object. The source NID and destination NID obtained in each edge is stored in a 2-D array of NIDs. Looping is performed on this array to obtain the edges which have a same vertex. In case the condition is satisfied, the edge labels of both the edges are printed.