CSE 562: Project #1 (due 05/06/16)

Description

You are to implement Top-K queries over a relation $Table$ of arity $N + 1$ (as defined in class).

Assumptions

- All the columns of $Table$ have non-negative integer values.
- The relation $Table$ is stored in the CSV format with the comma serving as the field delimiter and the end-of-line as the row delimiter. The relation has $N + 1$ attributes $Id, A_1, \ldots, A_N$. The attribute Id is the primary key and represents tuple identifiers.
- $F_V(A_1, \ldots, A_N)$ is a monotone scoring function defined by a linear arithmetic expression with positive integer coefficients over the attributes $A_1, \ldots, A_N$. The coefficients are specified as a vector $V = (v_1, \ldots, v_N)$. The vector is provided as the arguments of the `run1` and `run2` commands below.
- The scoring function

  $$F_V(A_1, \ldots, A_N) = \sum_{i=1}^{N} v_i * A_i.$$

  For example, the vector $V_0 = (0, 6, 2)$ specifies the function

  $$F_{V_0}(A_1, A_2, A_3) = 6 * A_2 + 2 * A_3.$$

Programs

- Your program is supposed to compute the set of rows of $Table$, defined by the following query:

  ```
  SELECT *
  FROM Table
  ORDER BY F_V(A_1,...,A_N) DESC
  LIMIT K
  ```

- Your program will compute the results in two different ways. It should be invoked as:

  ```
  %java topk K N
  topk>init tfile
  topk>run1 v1 v2 ... vN
  ```

  or

  ```
  %java topk K N
  topk>init tfile
  topk>run2 v1 v2 ... vN
  ```

where `tfile` is the name of the CSV file containing the contents of $Table$.

- Program functions:
  - `init` performs:
    1. construction of the relation $Table$.
    2. construction of a dense primary B-tree index on the attribute Id.
    3. construction of a dense secondary B-tree index on each of the remaining attributes $A_1, A_2, \ldots, A_N$.
    4. any other necessary initialization.
  - `run1` implements the Threshhold Algorithm of Fagin et al.[1] (as discussed in class), that computes the top K rows of $Table$ using the above indexes.
  - `run2` implements a naive priority queue algorithm that computes the top K rows of $Table$ in a single pass.
- You should test both algorithms on the given data.

Instructions:

- This is a team project, with 2 or 3 students per team.
- You are not supposed to implement B-tree insertion/deletion but only B-tree index creation and lookup.
- You are allowed to use B-tree code available on the web but you must reveal the source. Your team is not allowed to share your code with any other team. If you decide to write your own code, you will receive extra credit (see below).
- You are not supposed to implement a parser.
- Use main memory for all your data and operations.
- Detailed submission instructions and test datasets are forthcoming.

Extra credit

(1) Up to 5% of the final grade if you implement your own B-tree code.
(2) Implement Top-K join queries of the form

`SELECT` *
`FROM` $Table_1, \ldots, Table_m$
`WHERE` *Join condition*
`ORDER BY` $F_V(A_1, \ldots, A_N)$ `DESC`
`LIMIT` K

The score for this problem will depend on the sophistication of the algorithm [1]:

- A naive algorithm performing the join first and the Top-K computation second can earn up to 5% of the final grade.

- A sophisticated algorithm intermixing join and Top-K computation (like Rank-Join or J*) can earn up to 10% of the final grade.
- Before starting to implement the selected algorithm, discuss it with the instructor. You can only get credit for one of the above algorithms.

References

[1] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.