Basi di dati II — Progetto 1 — 27 marzo 2018

Mergesort a più vie

SimpleDB, nella versione scaricabile, realizza un mergesort tradizionale a due vie (cioè in cui la fusione è fatta su due run alla volta) e partendo da run iniziali molto piccoli (le porzioni già ordinate del file). Il tutto è realizzato dalle classi seguenti:

- **SortPlan**, implementazione dell'interfaccia Plan, che esegue nel metodo open() la divisione in run e i passi di merge precedenti l'ultimo.
- **SortScan**, implementazione dell'interfaccia Scan, che, con i classici metodi next() e getXXX(), realizza l'ultimo passo di merge nel corso della scansione.

Si propone di:

- Definire, nel package *materialize*, le classi **NWaysSortPlan** e **NWaysSortScan** per implementare il mergesort a più vie.
- Creare delle basi di dati di prova con tabelle di grandi dimensioni per eseguire il testing. Si suggerisce di ispirarsi alla classe **StudentMajorNoServer** per testare l'ordinamento senza avviare il server (e senza modificare parser, planner, etc.).
- Analizzare e fornire report di esecuzione di ordinamenti di tabelle di varie dimensioni. Per documentazione, stampare il numero di blocchi e il numero di record della tabella e il numero di run all'inizio e dopo ciascuna iterazione. Stampare anche il numero di buffer disponibili e il numero di quelli utilizzati (modificare opportunamente il numero di buffer del sistema).
- Una questione preliminare è l'individuazione del numero di buffer da utilizzare, che è opportuno sia una radice (quadrata, cubica, quarta, etc) del numero di blocchi del file e precisamente la più grande radice che sia minore del numero di buffer disponibili: ad esempio, con file di 8000 blocchi e 100 buffer disponibili, se ne utilizzerebbero 90 (arrotondamento della radice quadrata di 8000) mentre con 60 blocchi disponibili se ne utilizzerebbero 20 (radice cubica di 8000, perché la radice quadrata 90 è maggiore di 60). Per individuare tale numero, è disponibile il metodo bestRoot() della classe BufferNeeds.
- Il primo passo dell'algoritmo è la divisione del file in run. Come si può notare dal metodo splitIntoRuns() della classe SortPlan esistente, SimpleDB utilizza una tabella temporanea (molto piccola) per ciascun run. Per realizzare correttamente l'algoritmo, occorre creare tabelle temporanee e ordinate di k blocchi ciascuna. Tale passo richiede artifici tecnici e l'utilizzo di più moduli di SimpleDB. Si fornisce (nel file allegato **`assignment1_metodi_ausiliari.txt`** l'implementazione di alcuni metodi ausiliari:
  - splitIntoRuns(Scan src, int runs) per la classe **SortPlan** che effettua lo split in run ordinati di k blocchi ciascuno e restituisce la lista dei run.
  - compare(...) della classe **RecordComparator** e swap(...) per la classe **SortPlan**. Come si può vedere dall'implementazione, si tratta di metodi di supporto per splitIntoRuns(...).
- I passi di fusione vanno realizzati confrontando k run alla volta e vanno realizzati a partire da quelli in SortPlan (che li gestisce tutti escluso l'ultimo) e SortScan (che gestisce l'ultimo).