Principles of Computer System Design
Exam

Tudor Dragan (xlq880)

January 16, 2015

Question 1: Data Processing

1. Sort-based external memory algorithm

```
//TODO: Insert pseudocode here
```

Figure 1: Sort-based external memory algorithm

For this part of the exam we must implement an algorithm that first applies an aggregated function (in our case the *count* function) on the table *friends*, and then combine the results to write up the table that consists of (uid, networh, nrOfFriends). In order to implement this, we use a modified multi-way external sorting algorithm based on merge-sort.

I made the following assumptions:

1. We don't have any indices on the tables so the entries are not in sorted order in any of the two tables.
2. The friends table has only uni-directional relations, in the sense that Person1 can be friends with Person2 but it's not requested that Person2 has to be friends with Person1. (Person1 has 1 friend and Person2 has 0 friends.)
3. The table with the biggest number of records has N records.
4. We know that the main memory can hold $\sqrt{N}$ records. If we read B records at a time in memory, the number of runs will be N/B.
5. Number of passes in Phase 2 is P then: $B(B-1)^P = N$

First I build the table with the aggregate count function for the friends table by altering the way that we merge the pages on a pass. I will explain how it is done in the following paragraphs on a concrete example:

1. Every entry in the friends table has this form : ($uid1$, $uid2$). The input is split up in multiple blocks. Let us assume that we have a block that has these following ($uid1$, $uid2$) relations: (3,7) (1,4) (2,3) (2,5) (1,3) (1,2) (2,4) (3,1)

2. In order to calculate the number of friends for each user, I will sort records with the form ($uid1$, $friendCount$). because every record in the *friends* table essentially means 1 friend added, when reading in the first phase, the $friendCount$ is 1. So we will have initially tuples with the form (*uid*, 1) that need to be sorted by *uid*. We assume that the buffer page number is 4. So after the first phase we will have [(3,1) (1,1) (2,1) (2,1)] [(1,1) (1,1) (2,1) (3,1)].

3. The next pass will be [(1,1) (2,2) (3,1)] [(1,2) (2,1) (3,1)], Then we combine the values by adding up the number of friends by comparing the heads of the lists (the heads contain the smallest *uid*) and add up or write to disk the smallest one *uid* with the friend count.

4. Finally we will have [(1,3) (2,3) (3,2)].

5. This will continue until all the pages buffers are empty and have no more data to fill them up with.

2. Hash-based external memory algorithm

3. I/O costs