Assignment #1

due Wed. Feb. 14, 2024

(sent to teaching assistant: rasagnya53@gmail.com)

1. (15)
   (a) Consider the ER diagram of Figure 3.16 (found on my home page. Click on "figures on Chapter 3), which shows a simplified schema for an airline reservation system. Extract from the ER diagram the requirements and constraints that produced this schema. Try to be as precise as possible in your requirement and constraint specification.
   (b) A database is being constructed to keep track of the teams and games of a sport league. A team has a number of players, not all of whom participate in each game. It is desired to keep track of the players participating in each game for each team, the positions they played in that game, and the result of the game. Try to design an ER schema diagram for this application, stating any assumption you make. Choose your favorite sport (soccer, baseball, football, …).

2. (15) Specify the following queries in SQL (see Fig. 7.5 on my home page).

a. Retrieve the name of all employees in department 5 who work more than 10 hours per week on the ‘Product X’ project.

b. List the names of all employees who have a dependent with the same first name as themselves.

c. Find the names of all employees who are directly supervised by ‘Franklin Wong’.

d. For each project, list the project name and the total hours per week (by all employees) spent on that project.

e. Retrieve the names of all employees who work on every project.

f. Retrieve the names of all employees who do not work on any project

3. (25) Linear Hashing

- collision resolution strategy: chaining
- split rule: when load factor > 0.7
- initially M = 4 (M: size of the primary area)
- hash functions: $h_i(key) = key \bmod 2^i \times M$ (i = 0, 1, 2, …)
- bucket capacity = 4

Trace the insertion process of the following keys into a linear hashing file:

32, 44, 36, 9, 14, 18, 10, 25, 5, 30, 31, 35, 7, 11, 43, 37, 29, 50

4. (25) Apply the following algorithm to the B+-tree shown in Fig. 1 to store it in a data file (each node in a B+-tree is stored as a page in the data file) In the algorithm, a stack is used to explore a B+-tree in the depth-first manner. Each entry *X* in the stack has three data fields:

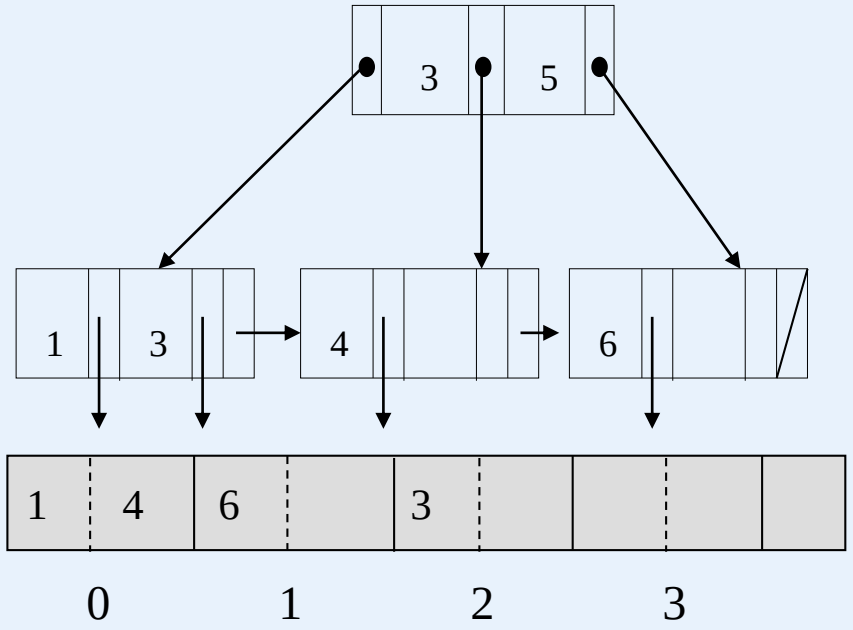
   X.data – to store all the key values in a node,
   X.address-of-parent – to record the address (in the file) of the parent of a node,
   X.position – used to indicate, for a node, what child of its parent the node is (i.e., whether it is the first, second, …, child of its parent.)

Trace the computation process.

Algorithm:

```
push(root, -1, -1);
while (S is not empty) do
{     x := pop( );
      store x.data in file F;
      assume that the address of x in F is ad;
      if x.address-of-parent ≠ -1 then {
            y := x.address-of-parent;
            z := x.position;
            write ad in page y at position z in F;
      }
      let x1, …, xk be the children of x;
      for (i = k to 1) {push(xi, ad, i)};
}
```

Fig. 1:



5. (20) Assume that for an attribute of a table, *m* bit maps (bit vectors) are created and then compressed using the following procedure:

a. Decompose each bit vector into a series of runs such that each contains a set consecutive 0's followed by a 1.
b. compress each run with i 0's as below:
   part 1: i expressed as a binary number, denoted as $b_1(i)$.
   part 2: Assume that b(i) is j bits long. Then, part 2 is a sequence of (j – 1) 1's followed by a 0, denoted a $b_2(i)$.
c. The compressed bit string is set to be $b_2(i)\ b_1(i)$.

Let s_j be the jth compressed bit vector. Putting all the compressed bit vectors together, we get a bit string:

$$s = s_1 s_2 \ldots s_j \ldots s_m$$

Please design a method to uncompress any compressed bit vector efficiently.