Central European Olympiad in Informatics

28 July – 4 August 2005 Sárospatak, Hungary
http://ceoi.inf.elte.hu

# Multi-key Sorting

This is actually not a difficult task. The main difficulty is in understanding the problem and "seeing" an algorithm. Programming it is quite easy.
A shortest equivalent sequence can be constructed as follows:

For each column index $i$, retain only the rightmost instance of $i$

For example, $Sort(1);Sort(2);Sort(1)$ is equivalent to $Sort(2);Sort(1)$.
To prove this we need to show that the resulting sequence is indeed equivalent to the original sequence and that no shorter sequence is equivalent.
The second part is easy: there is no shorter equivalent sequence. The original sequence involves sort operations on, say, $K$ of the columns. The proposed algorithm constructs a sequence of $K$ sort operations. Any shorter sequence will "miss" at least one of the columns. Consider a table consisting of two rows that originally are out of order in the missed column, and that are constant in all other columns. The shorter sequence yields the same table, but the original sequence would swap the two rows.
Now for the first part: the constructed sequence is equivalent.
For row $a$, we write $a[i]$ for the entry in column $i$. Consider a sequence of operations $Sort(i);X;Sort(i)$, where $X$ is another sequence of sort operations. We will show that this sequence is equivalent to $X;Sort(i)$. To show equivalence, we must show that for every table, both sequences yield the same result. This can be shown by considering every pair of rows $a$ and $b$ in an arbitrary table, and showing that both sequences put them in the same order. (Actually, we rely on the following lemma. Two permutations $p$ and $q$ are equal if and only if for every pair $a$ and $b$, we have that a precedes $b$ in $p$ if and only if $a$ precedes $b$ in $q$.)
Consider a pair of rows $a$ and $b$.
If $a[i] < b[i]$, then the final $Sort(i)$ in the sequence (present as final sort operation in both sequences) will order them on column $i$, putting a before $b$, regardless of the effect of the preceding operations. A similar argument holds for $a[i] > b[i]$.
If $a[i] = b[i]$, then the final $Sort(i)$ has no effect, and leaves the rows in the order produced by the initial part of the operation sequence. But in that case, the initial $Sort(i)$ in the first sequence will not have changed the order of $a$ and $b$ (though it may have affected the order of other rows). Hence, the initial sequences preceding the final $Sort(i)$ will have the same effect on the order of $a$ and $b$.
Once we have that $Sort(i);X;Sort(i)$ is equivalent to $X;Sort(i)$, it follows that every sequence can be reduced to a sequence in which every column occurs at most once, by eliminating duplicates from the left.

**Programs**

There are various ways to program this algorithm. There is no need to store the entire input sequence. In fact, the entire input sequence can not be stored because of the memory limit.
Maintain the shortest equivalent sequence $S$ for that part of the input sequence read so far. When reading another sort operation, say on column $i$, the sequence $S$ can be updated by appending $i$ to the end of $S$, and deleting any earlier occurrence of $i$ in $S$. Thus, it is sufficient to do the following operations on $S$ efficiently:

**append an element at the end**

**determine if a given element occurs, and if so, where**

**delete an arbitrary element**

Where we may assume that $S$ contains no duplicates ($S$ is a permutation).
This can be accomplished in constant time: For each index that occurs, maintain its predecessor and successor. Include sentinels at begin and end to simplify things a bit.

**Complexity** (optimal):
**Memory**: $O(C)$
**Time**: $O(length\,of\,input\,sequence)$

Other approach

For each column index that has occurred, maintain the rank of its rightmost occurrence. At the end, sort the list on rank.




**Implementation**

```
program Keys;
Const
  MaxN=3000000; {max # sort operations}
  MaxC=1000000; {max # columns}
Var
  inFile,outFile:Text;
  C,            {# colunms}
  n,            {# sort operations}
  m:longint;    {length of the shortest equvilent sequence}
  Next, Prev:array[0..MaxC+1] of longint; {double linked list}
  C1,x,i:longint;
begin
  assign(inFile, 'keys.in'); reset(inFile);
  readln(inFile, C, n);
  C1:=C+1;
  for i:=1 to C do            {indicate that column i has not occured in sort operations}
   Next[i]:=-1;
  Next[0]:=C1;                {create an empty list by linking head sentinel 0}
  Prev[C1]:=0;                {and tail sentinel C+1}
  m:=0;
  for i:=1 to n do begin
    read(inFile,x);
    if Next[x]<>-1 then begin {x already occured}
      Next[Prev[x]]:=Next[x]; {delete x from the list}
      Prev[Next[x]]:=Prev[x];
    end else                  {new column}
      inc(m);
    Prev[x]:=Prev[C1];        {append x at the end of the list}
    Next[Prev[C1]]:=x;
    Prev[C1]:=x;
    Next[x]:=C1;
  end{for i};

  assign(outFile, 'keys.out'); rewrite(outFile);
  writeln(outFile, m);
  i:=Next[0];
  while Next[i]<>C1 do begin  {write out the shortest sequence}
    write(outFile, i,' ');
    i:=Next[i];
  end;
  writeln(outFile, i);
  close(inFile);
  close(outFile);
end.
```