Appendix II

/* The following routine uses a pair of epp nodes to represent each of the interval
transactions in the input interval transaction database $S_0$; except for the field isdeleted,
all the other fields of the epp nodes are set; pointers to these epp nodes are inserted
in the array $epp[]$. */

create_eppnodes(S): void

Parameter:

S: interval transaction database

{
    i: integer
    i = 0

    for every interval $[a, b]$ in $S$
        i = i + 2
        ep[i]->e = (b,']')
        ep[i]->pe = ep[i-1]
        ep[i]->freq = 1
        ep[i-1]->e = (a,'[')
        ep[i-1]->pe = ep[i]
        ep[i-1]->freq = 1
    end for

}
/* The following routine uses a single pair of epp nodes to represent all the interval transactions in the input interval transaction database $S_0$ that have the same interval. The routine is thus essentially an elimination of duplicates in the sorted ep[1...2N] array. However the freq and pe fields of *(ep[i]) need to be adjusted appropriately. If the interval [a, b] appears in m out of N interval transactions, then in the sorted ep[1..2N] array, the endpoint (a, '[') will be ep[k1]->e for $k_1$ in the range $i+1 ... i+m$ and the endpoint (b, ']') will be ep[k2]->e for $k_2$ in the range $j+1 ... j+m$ where $j+1 > i + m$. Obviously then, ep[k1]->pe will be ep[k2] for some $k_2$ in the range $j+1 ... j+m$. Similarly, ep[k2]->pe will be ep[k1] for some $k_1$ in the range $i+1 ... i+m$. The routine hence obtains $I, J$ with $I < J$ such that ep[I]->freq = m, ep[I]->e = (a, '['), ep[I]->pe = ep[J], ep[J]->freq = m, ep[J]->e = (b, ']') and ep[J]->pe = ep[I]. $I$ gets determined first when the processing of the sub-array ep[i+1 ... i+m] starts. When the sub-array ep[j+1 ... j+m] is processed later, $J$ gets determined. The routine returns the number of distinct intervals in $S_0$. */

compress(T): integer

Parameter:
T: integer

{ 
  id, is1, is2: integer

  id = is1 = is2 = 1

  while (is1 ≤ 2T) 
  
    if (ep[is1]->e is a left endpoint) 
      ep[id]->e = ep[is1]->e

  
  179
while (ep[isl]→e is equal to ep[is2]→e and
    ep[isl]→pe→e is equal to ep[is2]→pe→e)
    ep[is2]→pe→pe = ep[ld]
    is2 = is2 + 1
end while

ep[ld]→freq = is2 - isl
isl = is2

else /* (ep[isl]→e is a right endpoint */
    ep[ld]→e = ep[isl]→e
    ep[ld]→pe = ep[isl]→pe
    ep[ld]→pe→pe = ep[ld]
    ep[ld]→freq = ep[ld]→pe→freq
    isl = isl + ep[ld]→freq
    is2 = is1
endif

id = id + 1
end while

return (id - 1)/2
}

while (ep[isl]→e is equal to ep[is2]→e and
    ep[isl]→pe→e is equal to ep[is2]→pe→e)
    ep[is2]→pe→pe = ep[ld]
    is2 = is2 + 1
end while

ep[ld]→freq = is2 - isl
isl = is2

else /* (ep[isl]→e is a right endpoint */
    ep[ld]→e = ep[isl]→e
    ep[ld]→pe = ep[isl]→pe
    ep[ld]→pe→pe = ep[ld]
    ep[ld]→freq = ep[ld]→pe→freq
    isl = is1 + ep[ld]→freq
    is2 = is1
endif

id = id + 1
end while

return (id - 1)/2
}