Answer Set 7

OUT: NOVEMBER, NOT DISTRIB., 1994

```
#include <stdlib.h>
#include <strings.h>
#include <stdio.h>

/* An implementation of 2-4 trees, using preemptive
   splitting of 4 nodes on the way down searching.

   Burt Rosenberg (c) 1994
   Univ. of Miami
   Math 220, Fall 1994
*/

#define FOUR 4

typedef struct da_ {
     int count ;
     char * s ;
     } DA_NODE ;

typedef struct tf__ {
     int n ;
     struct tf_ {
        struct tf__ * ch ;
        DA_NODE * da ; }
     K[FOUR] ;
     } TF_NODE ;

/* TF_NODE invariants:

   2<=n<=4.
   K[i].da points to valid datum for 0<=i<n.
   K[0].ch=NULL iff node is a leaf node.
   else K[i].ch points to valid tf_node for 0<=i<n.
   K[i].da points to largest datum in subtree
      rooted at K[i].ch, unless K[i].ch==NULL.

   DA_NODE invariants:

   s points to a null terminated string.
   count has the number of times string s is to be counted.
```

```
   Two-Four tree invariants:

   The minimum datum is "", the maximum is "~max", and these
   two datum are always present in the tree.
*/

DA_NODE * create_da(char * s) {
   DA_NODE * da ;
   da = (DA_NODE *) malloc(sizeof(DA_NODE)) ;
   da->count = 1 ;
   da->s = (char *) calloc(strlen(s)+1,sizeof(char)) ;
   strcpy(da->s,s) ;
   return da ;
}

#define GetStr(A,B) (A)->K[(B)].da->s
#define IsLeaf(A) ((A)->K[0].ch==NULL)
#define IncrDa(A) (++(A)->count)

TF_NODE * create_tf(void) {
   TF_NODE * t ;
   t = (TF_NODE *) malloc( sizeof(TF_NODE) ) ;
   t->K[0].ch = NULL ;
   t->K[0].da = create_da("") ;
   t->K[1].ch = NULL ;
   t->K[1].da = create_da("~max") ;
   t->n = 2 ;
   return t ;
}


void add_child( TF_NODE * t, TF_NODE * new, DA_NODE * da ) {
   int i = t->n ;
   struct tf_ temp ;

   if (i==4) {
      printf("add_child: asked to add a child to a full node.\n") ;
      return ;
   }
   (t->n)++ ;
   t->K[i].ch = new ;
   t->K[i].da = da ;
   while( i>0 ) {
      /* bubble sort */
```

```
      if( strcmp( GetStr(t,i),GetStr(t,i-1) ) >= 0 ) break ;
      temp = t->K[i-1] ;
      t->K[i-1] = t->K[i] ;
      t->K[i] = temp ;
      i-- ;
   }
   return ;
}

TF_NODE * split_four_node( TF_NODE * t ) {
   TF_NODE * l ;

   l = (TF_NODE *) malloc( sizeof(TF_NODE) ) ;
   l->n = t->n = 2 ;
   l->K[0] = t->K[2] ;
   l->K[1] = t->K[3] ;
   return l ;

}

TF_NODE * split_root( TF_NODE * t ) {
  /* ASSERT t is a four node */
  TF_NODE * l, * r ;
  l = split_four_node( t ) ;
  r = (TF_NODE *) malloc( sizeof(TF_NODE) ) ;
  r->K[0].ch = t ;
  r->K[0].da = t->K[1].da ;
  r->K[1].ch = l ;
  r->K[1].da = l->K[1].da ;
  r->n = 2 ;
  return r ;
}

TF_NODE * split_four_child( TF_NODE * t, int i )
{
   /* split the i-th child. It is a 4-node, and t is a 2 or 3 node. */
   TF_NODE * l ;

   l = split_four_node( t->K[i].ch ) ;
   t->K[i].da = t->K[i].ch->K[1].da ;
   add_child( t, l , l->K[1].da ) ;
   return t ;
}
```

```
int find_child( TF_NODE * t, char * s )
{
   int i ;
   /* return min i st s <= GetStr(t,i) */
   for ( i=0; i<(t->n); i++)
      if (strcmp(GetStr(t,i),s)>=0) return i ;
   printf("find_child: cannot locate proper child\n") ;
   return 0 ;
}

DA_NODE * find_tf( TF_NODE * t, char * s )
{
   int i ;
   DA_NODE * dret = NULL ;

   while (t!=NULL) {
      i = find_child(t,s) ;
      if (strcmp(GetStr(t,i),s)==0) {
         dret = t->K[i].da ;
         break ;
      }
      t = t->K[i].ch ;
   }
   return dret ;
}

TF_NODE * insert_tf( TF_NODE * t, char * s )
{
   int i ;
   DA_NODE * da ;
   TF_NODE * tret ;

   da = create_da(s) ;
   if (t->n==4) t = split_root(t) ;
   tret = t ;

   while (!IsLeaf(t)) {
      i = find_child(t,da->s) ;
      if (t->K[i].ch->n==4) {
         t = split_four_child(t,i) ;
         i = find_child(t,da->s) ;
      }
      t = t->K[i].ch ;
   }
```

```
   add_child(t,NULL,da) ;

   return tret ;
}

void print_da( DA_NODE * da ) {
   if ( strlen(da->s)==0 ) printf("%2d:_min\n",da->count) ;
   else printf("%2d:%s\n",da->count,da->s) ;
}

void pp_tf( TF_NODE * t, char * prefix ) {
   int i ;
   if (IsLeaf(t)) {
      print_da(t->K[0].da) ;
      for (i=1;i<t->n;i++) {
         printf("%s",prefix) ;
         print_da(t->K[i].da) ;
      }
      return ;
   }
   /* else, not a leaf */
   for ( i=0; i<t->n; i++ ) {
      if (i==(t->n-1)) strcat(prefix,"    ") ;
      else strcat(prefix,"|   ") ;
      printf("+---") ;
      pp_tf(t->K[i].ch,prefix) ;
      prefix[strlen(prefix)-4] = '\0' ;
      if (i!=(t->n-1)) printf("%s",prefix) ;
   }
}

void pretty_print( TF_NODE * t ) {
   char buffer[200] ;
   strcpy( buffer, "    ") ;
   printf("\n[]--") ;
   pp_tf( t, buffer ) ;
}

int main() {
   TF_NODE * tree ;
   DA_NODE * datum ;
   int i, n = 35 ;
   char * words[35] ;
   words[0] = "but" ;
```

```
   words[1] = "soft" ;
   words[2] = "what" ;
   words[3] = "light" ;
   words[4] = "through" ;
   words[5] = "yonder" ;
   words[6] = "window" ;
   words[7] = "breaks" ;
   words[8] = "it" ;
   words[9] = "is" ;
   words[10] = "the" ;
   words[11] = "east" ;
   words[12] = "and" ;
   words[13] = "juliet" ;
   words[14] = "is" ;
   words[15] = "the" ;
   words[16] = "sun" ;
   words[17] = "arise" ;
   words[18] = "fair" ;
   words[19] = "sun" ;
   words[20] = "and" ;
   words[21] = "kill" ;
   words[22] = "the" ;
   words[23] = "envious" ;
   words[24] = "moon" ;
   words[25] = "that" ;
   words[26] = "thou" ;
   words[27] = "her" ;
   words[28] = "maid" ;
   words[29] = "art" ;
   words[30] = "far" ;
   words[31] = "more" ;
   words[32] = "fair" ;
   words[33] = "than" ;
   words[34] = "she" ;
   tree = create_tf() ;
   for (i=0;i<n;i++) {
      datum = find_tf( tree, words[i] ) ;
      if (datum!=NULL) IncrDa(datum) ;
      else tree = insert_tf( tree, words[i] ) ;
      pretty_print(tree) ;
      printf("\n") ;
      sleep(2) ;
   }
}
```