ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Τμήμα Πληροφορικής και Τηλεπικοινωνιών • Εισαγωγή στον Προγραμματισμό

Περιεχόμενα

Κεφάλαιο 8: Λίστες και δυαδικά δέντρα

Διαχείριση συνδεδεμένων λιστών

```c
/* File: listmanagement.c */
#include <stdio.h>
#include <stdlib.h>

typedef struct listnode *Listptr;

struct listnode {
  int value;
  Listptr next;
};

int empty(Listptr);
int in(Listptr, int);
int n_th(Listptr, int, int *);
void insert_at_start(Listptr *, int);
void insert_at_end(Listptr *, int);
int delete(Listptr *, int);
void print(Listptr);
```

```c
int main(void)
{ Listptr alist;
  int v;
  alist = NULL;                                    /* List is NULL */
                                                /* Check if list is empty */
  printf("List is%s empty\n", empty(alist) ? "" : " not");
  insert_at_start(&alist, 44);                  /* List is 44--> NULL */
  printf("List is "); print(alist);
  insert_at_end(&alist, 55);              /* List is 44--> 55--> NULL */
  printf("List is "); print(alist);
  insert_at_start(&alist, 33);      /* List is 33--> 44-> 55--> NULL */
  printf("List is "); print(alist);
  insert_at_end(&alist, 66); /* List is 33--> 44-> 55--> 66--> NULL */
  printf("List is "); print(alist);
                                                /* Check if list is empty */
  printf("List is%s empty\n", empty(alist) ? "" : " not");
                                                   /* Check membership */
  printf("55 is%s in list\n", in(alist, 55) ? "" : " not");
  printf("77 is%s in list\n", in(alist, 77) ? "" : " not");
  if (n_th(alist, 2, &v))                       /* Return 2nd element */
    printf("Item no 2 is %d\n", v);
  else
    printf("Item no 2 does not exist\n");
  if (n_th(alist, 6, &v))                       /* Return 6th element */
    printf("Item no 6 is %d\n", v);
  else
    printf("Item no 6 does not exist\n");
  printf("Deleting 55. %s\n", delete(&alist, 55) ? "OK!" : "Failed!");
                                  /* List is 33--> 44--> 66--> NULL */
  printf("List is "); print(alist);
  printf("Deleting 22. %s\n", delete(&alist, 22) ? "OK!" : "Failed!");
                                  /* List is 33--> 44--> 66--> NULL */
  printf("List is "); print(alist);
  printf("Deleting 33. %s\n", delete(&alist, 33) ? "OK!" : "Failed!");
                                             /* List is 44--> 66--> NULL */
  printf("List is "); print(alist);
  return 0;
}

int empty(Listptr list)                      /* Check if list is empty */
{ if (list == NULL)                             /* Is it really empty? */
    return 1;                                            /* Yes, it is */
  else
```

```
    return 0;                                        /* No, it isn't */
}

int in(Listptr list, int v)         /* Check if v is member of list */
{ while (list != NULL)          /* Visit list elements up to the end */
    if (list->value == v)   /* Did we find what we are looking for? */
      return 1;                                      /* Yes, we did */
    else
      list = list->next;                  /* No, go to next element */
  return 0;                             /* Finally, v is not in list */
}

int n_th(Listptr list, int n, int *vaddr)
           /* Return n-th element of list, if it exists, into vaddr */
{ while (list != NULL)    /* Maybe search up to the end of the list */
    if (n-- == 1) {              /* Did we reach the right element? */
      *vaddr = list->value;                       /* Yes, return it */
      return 1;                                      /* We found it */
    }
    else
      list = list->next;                  /* No, go to next element */
  return 0;                              /* Sorry, list is too short */
}

void insert_at_start(Listptr *ptraddr, int v)
                      /* Insert v as first element of list *ptraddr */
{ Listptr templist;
  templist = *ptraddr;                /* Save current start of list */
  *ptraddr = malloc(sizeof(struct listnode)); /* Space for new node */
  (*ptraddr)->value = v;                                /* Put value */
  (*ptraddr)->next = templist;      /* Next element is former first */
}

void insert_at_end(Listptr *ptraddr, int v)
                       /* Insert v as last element of list *ptraddr */
{ while (*ptraddr != NULL)                     /* Go to end of list */
    ptraddr = &((*ptraddr)->next);/* Prepare what we need to change */
  *ptraddr = malloc(sizeof(struct listnode)); /* Space for new node */
  (*ptraddr)->value = v;                                /* Put value */
  (*ptraddr)->next = NULL;              /* There is no next element */
}

int delete(Listptr *ptraddr, int v)
```

```
                /* Delete element v from list *ptraddr, if it exists */
{ Listptr templist;
  while ((*ptraddr) != NULL)   /* Visit list elements up to the end */
    if ((*ptraddr)->value == v) {    /* Did we find what to delete? */
      templist = *ptraddr;         /* Yes, save address of its node */
      *ptraddr = (*ptraddr)->next;        /* Bypass deleted element */
      free(templist);     /* Free memory for the corresponding node */
      return 1;                           /* We deleted the element */
    }
    else
      ptraddr = &((*ptraddr)->next);/* Prepare what we might change */
  return 0;        /* We did't find the element we were looking for */
}

void print(Listptr list)                  /* Print elements of list */
{ while (list != NULL) {       /* Visit list elements up to the end */
    printf("%d--> ", list->value);         /* Print current element */
    list = list->next;                        /* Go to next element */
  }
  printf("NULL\n");                            /* Print end of list */
}
```

```
% gcc -o listmanagement listmanagement.c
% ./listmanagement
List is empty
List is 44--> NULL
List is 44--> 55--> NULL
List is 33--> 44--> 55--> NULL
List is 33--> 44--> 55--> 66--> NULL
List is not empty
55 is in list
77 is not in list
Item no 2 is 44
Item no 6 does not exist
Deleting 55. OK!
List is 33--> 44--> 66--> NULL
Deleting 22. Failed!
List is 33--> 44--> 66--> NULL
Deleting 33. OK!
List is 44--> 66--> NULL
%
```

Διαχείριση δυαδικών δέντρων

```c
/* File: treemanagement.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct tnode *Treeptr;

typedef struct tnode {
  char *word;
  Treeptr left;
  Treeptr right;
} Treenode;

Treeptr addtree(Treeptr, char *);
void treeprint(Treeptr, int);
void nodesprint(Treeptr);
int treedepth(Treeptr);
int treesearch(Treeptr, char *);

int main(int argc, char *argv[])
{ Treeptr p;
  char buf[80];
  p = NULL;                                /* Initialize binary tree */
  while (scanf("%s", buf) != EOF)          /* Read words from input */
    p = addtree(p, buf);                /* and insert them into the tree */
  printf("Tree is:\n"),
  treeprint(p, 0);                   /* Kind of tree pretty printing */
  printf("\nNodes are:\n");
  nodesprint(p);          /* Print tree nodes in alphabetical order */
  printf("\n\nTree depth is %d\n", treedepth(p));
                                /* Compute and print depth of tree */
  printf("\n");
  while (--argc) {                              /* For each argument */
    argv++;       /* check whether it coincides with any tree node */
    printf("%s found %s\n",
           (treesearch(p, *argv)) ? "   " : "not", *argv);
  }
  return 0;
}

Treeptr addtree(Treeptr p, char *w)   /* Insert word w into tree p */
{ int cond;
```

```
  if (p == NULL) {                              /* If tree is empty */
    p = malloc(sizeof(Treenode));    /* Allocate space for new node */
                                     /* Allocate space to copy word */
    p->word = malloc((strlen(w)+1) * sizeof(char));
    strcpy(p->word, w);                 /* Copy word w to tree node */
    p->left = NULL;            /* Left subtree of new node is empty */
    p->right = NULL;          /* Right subtree of new node is empty */
  }
  else if ((cond = strcmp(w, p->word)) < 0)
                       /* Does word w precede word of current node? */
    p->left = addtree(p->left, w);
                             /* If yes, insert it into left subtree */
  else if (cond > 0)        /* Does it follow word of current node? */
    p->right = addtree(p->right, w);
                            /* If yes, insert it into right subtree */
  /* If it is the same with word of current node, do not insert it */
  return p;                                             /* Return tree */
}

void treeprint(Treeptr p, int indent)          /* Pretty print tree */
{ int i;
  if (p != NULL) {                          /* If tree is not empty */
    treeprint(p->right, indent+4);
                /* Print right subtree 4 places right of root node */
    for (i=0 ; i < indent ; i++)
      printf(" ");                       /* Take care for indentation */
    printf("%s\n", p->word);                      /* Print root node */
    treeprint(p->left, indent+4);
                 /* Print left subtree 4 places right of root node */
  }
}

void nodesprint(Treeptr p)                      /* Print tree nodes */
{ if (p != NULL) {                            /* If tree is not empty */
    nodesprint(p->left);                        /* Print left subtree */
    printf("%s ", p->word);                        /* Print root node */
    nodesprint(p->right);                     /* Print right subtree */
  }
}

int treedepth(Treeptr p)                  /* Compute depth of tree p */
{ int n1, n2;
  if (p == NULL)                          /* Depth of empty tree is 0 */
```

```
    return 0;
  n1 = treedepth(p->left);         /* Compute depth of left subtree */
  n2 = treedepth(p->right);       /* Compute depth of right subtree */
  return (n1 > n2) ? n1+1 : n2+1;
     /* Return maximun of depths of left and right subtrees plus 1 */
}

int treesearch(Treeptr p, char *w)
                               /* Check whether word w is in tree p */
{ int cond;
  if (p == NULL)                                  /* If tree is empty */
    return 0;                                   /* We didn't find word */
  if ((cond = strcmp(w, p->word)) == 0)
                    /* Word w is the same with word of current node */
    return 1;
  else if (cond < 0)          /* If w precedes word of current node */
    return treesearch(p->left, w);           /* Search left suftree */
  else                                                /* Otherwise */
    return treesearch(p->right, w);         /* search right subtree */
}
```

```
% gcc -o treemanagement treemanagement.c
% cat words.txt
some
words
to
insert
into
a
binary
tree
% ./treemanagement into found words < words.txt
Tree is:
    words
            tree
        to
some
        into
    insert
            binary
        a

Nodes are:
a binary insert into some to tree words
```

```
Tree depth is 4

    found into
not found found
    found words
%
```