Matthias Jauernig
Kurs: 03In1

18.12.03
linux-related.de

11. Aufgabenserie zu den Grundlagen der Informatik

Abgabetermin: Mi, 07.01.04

Zu 31.) asymptotische Laufzeitkomplexität

- **allgemeiner Fall:**

(a) $= O(f(n))+O(g(n))+O(h(n))$
$= O(\max[\max\{f(n),g(n)\},h(n)])$
(b) $= m*O(f(n))$
$= O(m*f(n))$
(c) $= O(\max\{g(n), h(n)\})$
(d) $= n*(O(f(n)+m*g(n))$
$= O(n*\max\{f(n), m*g(n)\})$

- **Speziell: f(n):=log(n); g(n):=n*log(n); h(n):=n²**

(a) $O(\max[\max\{f(n),g(n)\},h(n)]) = O(\max[\max\{\log(n),n*\log(n)\},n^2])$
$= O(\max\{n*\log(n),n^2\}) = O(n^2)$
(b) $O(m*f(n)) = O(m*\log(n))$
(c) $O(\max\{(g(n)), h(n)\}) = O(\max\{n*\log(n), n^2\}) = O(n^2)$
(d) $O(n*\max\{f(n), m*g(n)\}) = O(n*\max\{\log(n), m*n*\log(n)\})$
$= O(m*n^2*\log(n)\})$

Zu 32.) Sieb des Eratosthenes

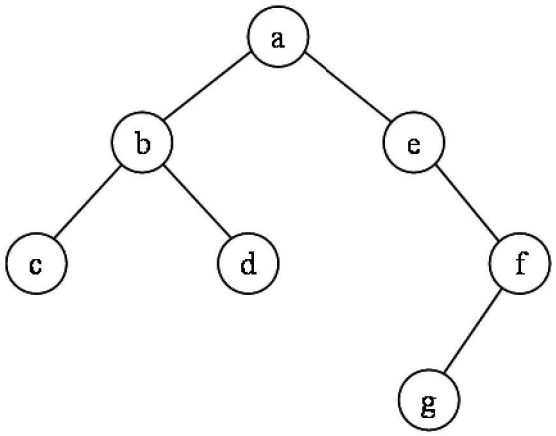
```
/*      era_sieb.c -- Matthias Jauernig, 18.12.03                                */
/*      Programm realisiert "Sieb des Eratosthenes" mithilfe einer verkett. Liste */
/*      mit gcc kompilieren:          "gcc -Wall -lm -o era_sieb era_sieb.c"      */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* ---- Basistruktur für ein Listenelement ------------------------------------ */
typedef struct NODE{
        long long zahl;
        struct NODE *next;
} LISTELEM;     LISTELEM *kopf;


/* ---- anhaengen() -- hängt ein Element an die vorhandene verkettete Liste an ---- */
LISTELEM *anhaengen(long long i, LISTELEM *akt){
        LISTELEM *neu;
        if((neu=(LISTELEM *)malloc(sizeof(LISTELEM)))==NULL){
                perror("!! Speicherreservierung");
                exit(1);
        }
        akt->next=neu;
        neu->zahl=i;
        neu->next=NULL;

        return neu;
}

/* ---- entferne_next() -- entfernt das nächste Element der verketteten Liste ----- */
void entferne_next(LISTELEM *akt){
        LISTELEM *h=akt->next;
```

```
        if(h!=NULL){
                akt->next=akt->next->next;
                free(h);
        }
}

/* ---- ausgabe() -- gibt alle Inhalte der verketteten Liste aus ------------------ */
void ausgabe(void){
        LISTELEM *akt=kopf->next;
        long long i;
        for(i=1; akt!=NULL; i++, akt=akt->next){
                printf("%7lld, ", akt->zahl);
                if(i%8==0) printf("\n");
        }
        printf("\b\b  \n");
}

/* ---- loesche_liste() -- gibt Speicherplatz der verketteten Liste frei ---------- */
void loesche_liste(void){
        LISTELEM *h, *akt=kopf->next;
        kopf->next=NULL;
        while(akt!=NULL){
                h=akt;
                akt=akt->next;
                free(h);
        }
}

/* ---- main() -------------------------------------------------------------- */
int main(void){
        long long n, sn, i, j;
        LISTELEM *akt;
        //################################# Eingabe ###################################
        printf( "\nSieb des Eratosthenes\n"
                "---------------------\n\n");
        do{
                printf("Primzahlen bis zu welcher Zahl n berechnen (n>1)?: ");
                scanf("%lld", &n);
        }while(n<2 && printf("!! n muss größer als 1 sein!\n\n"));
        //############################## Liste erstellen ##############################
        kopf=(LISTELEM *)malloc(sizeof(LISTELEM));
        kopf->zahl=0;
        kopf->next=NULL;
        akt=kopf;
        for(i=2; i<=n; i++)
                akt=anhaengen(i,akt);
        //######################### Primzahlen herausfiltern ##########################
        sn=(long long)sqrt(n);
        for(i=2; i<=sn; i++){
                //setze akt auf das i. Element und lösche von da an Vielfache von i
                for(j=2, akt=kopf; j<=i; j++)
                        akt=akt->next;
                while(akt->next!=NULL){
                        if((akt->next->zahl)%i==0)
                                entferne_next(akt);
                        if(akt->next!=NULL)
                                akt=akt->next;
                }
        }
        //########################### Ausgabe und beenden #############################
        printf("\n=> Primzahlen im Bereich [2...%lld]:\n", n);
        ausgabe();
        loesche_liste();
        free(kopf);

        return 0;
}
```

Zu 33.) binäre Bäume

(a) Binärbaum (a,(b,(c,ε,ε),(d,ε,ε)),(e,ε,(f,(g,ε,ε),ε)))

- Graphische Darstellung des Baumes:



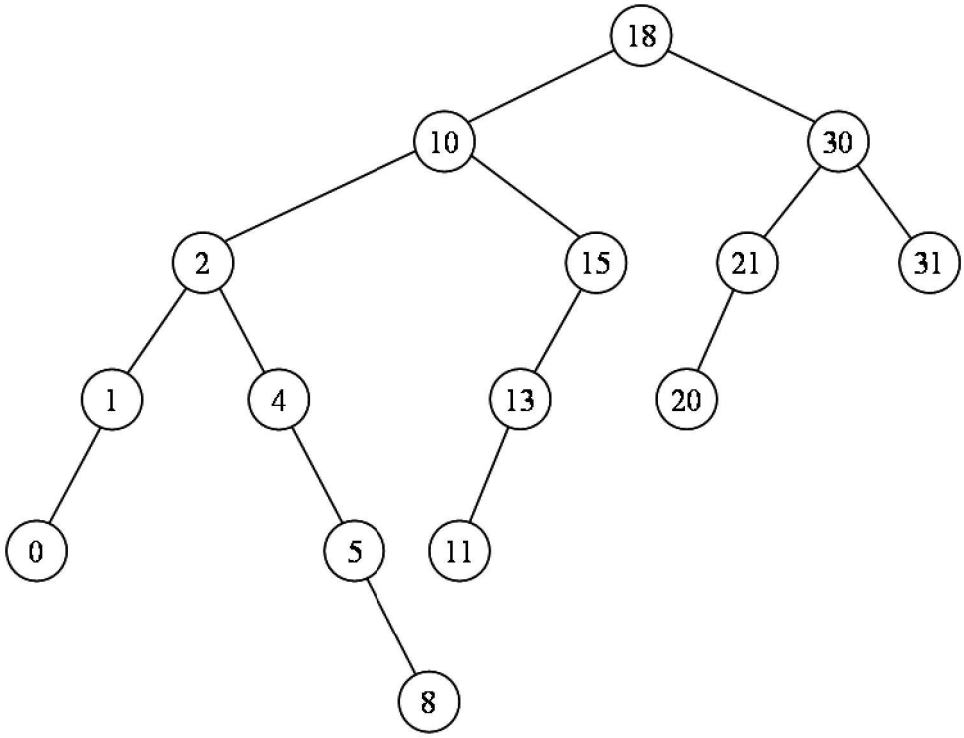
- Darstellung als Tabelle (beispielsweise):

Index	Knoten	Linker Sohn	Rechter Sohn
0	a	1	4
1	b	2	3
2	c	-	-
3	d	-	-
4	e	-	5
5	f	6	-
6	g	-	-

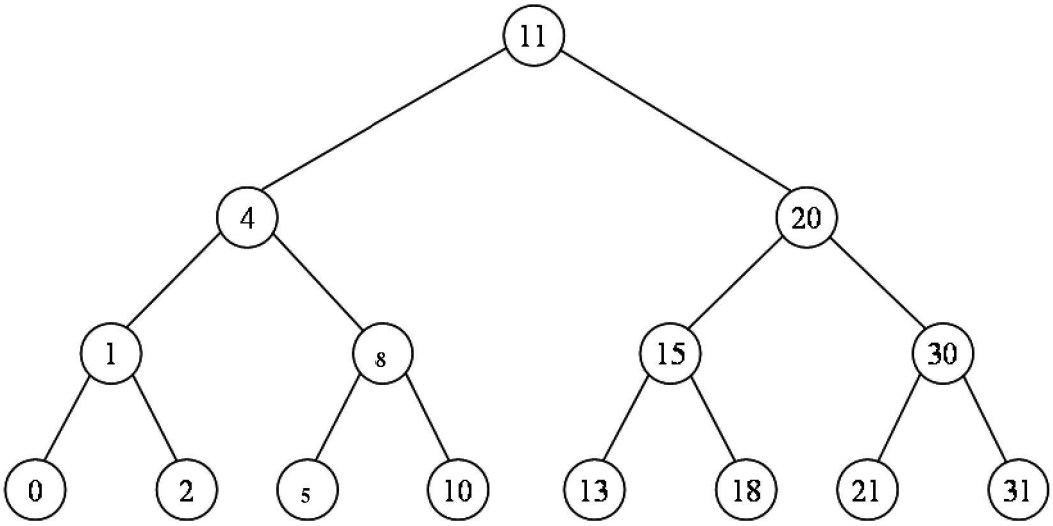
- Inorder-Durchlauf: c b d a e g f
- Postorder-Durchlauf: c d b g f e a
- Preorder-Durchlauf: a b c d e f g

(b) Suchbaum (18,10,15,30,2,21,31,20,1,4,13,5,11,0,8)

- Graphische Darstellung des Baumes:



- Suchbaum, in dem alle Blätter dieselbe Tiefe haben:



d.h. wenn z.B. folgende Reihenfolge der Schlüssel vorliegt:
(11, 4, 20, 1, 8, 15, 30, 0, 2, 5, 10, 13, 18, 21, 31)

Möglich wären hier auch viele andere Kombinationen, z.B. wenn nur 1 Blatt vorhanden ist, wenn jeder Knoten also nur einen linken (oder rechten) Nachfolger hat etc.