Fachbereich Mathematik
Prof. Dr. Martin
Dipl.-Math. Andrea Peter
Dipl.-Math. Christine Schönberger

TECHNISCHE
UNIVERSITÄT
DARMSTADT

SS 2008
14./15.05.08

3. Tutoriumsblatt zur „Algorithmischen Diskreten Mathematik“

Übung am Computer

Heute wollen wir uns mit Arrays und Funktionen beschäftigen. Außerdem werden wir unseren ersten “Graphen-Algorithmus” implementieren.

Tipp: Testen Sie mal den Editor “nedit” und darin die Funktionalitäten “Show Line Numbers” und “Language Mode”, beides unter “Preferences”.

Im Gegensatz zu einer Variablen entspricht ein **Array** einer bestimmten Anzahl von Variablen vom gleichen Typ. Auch ein Array wird mit Typ und Namen deklariert. Hinter dem Namen muss, in eckigen Klammern eingeschlossen, angegeben werden, aus wie vielen Komponenten der Array bestehen soll.
Beispiel: `int bit[6];` Die Komponenten können dann mit `bit[0], bit[1], ... , bit[5]` angesprochen werden. Die Indices der Komponenten beginnen immer bei 0!

Aufgabe C16 (Binärdarstellung)
Erinnern Sie sich an Aufgabe C6 des 1.Tutoriums und schreiben Sie ein Programm, das eine ganze Zahl zwischen 0 und 255 vom Bildschirm einliest, und ihre Binärdarstellung berechnet. Geben Sie die Binärdarstellung am Bildschirm aus. Testen Sie ihr Programm an einigen Beispielen. Speichern Sie hierzu die einzelnen Bits in einem Array.

Aufgabe C17 (Rekursion)
Unter (direkter) **Rekursion** versteht man, dass sich eine **Funktion** selbst aufruft. Betrachten Sie als Beispiel

```
#include <stdio.h>

/* DEKLARATION der Unterfuntion
 * Hier wird mitgeteilt, wie die Funktion heisst,
 * mit welchen Parametern sie aufgerufen wird
 * und welchen Typ von Funktionswert sie zurueckgibt.
*/
int multiply(int a, int b);

int main() {
```

```
  int zahl1, zahl2;

  printf("Zwei ganze Zahlen eingeben: ");
  scanf("%d %d", &zahl1, &zahl2);
  printf("Das Produkt der Zahlen ist: %d\n", multiply(zahl1, zahl2));

  return 0;
}

/* DEFINITION der Unterfuntion
 * Hier werden die Operationen festgelegt,
 * welche beim Aufruf der Funktion auszufuehren sind.
*/
int multiply(int a, int b){
   int produkt;
   if ( b == 0 )
    produkt = 0;
   else
    produkt = a + multiply(a, b-1);

   return (produkt);
}
```

(a) Schreiben Sie ein Programm, das eine der beiden folgenden Aufgabenstellungen rekursiv bearbeitet.

i. (Leichter) Berechnung der Falkultät

ii. (Etwas schwieriger) Berechnung der Fibonacci-Zahlen

Bei Interesse, vergleichen Sie die Laufzeit der rekursiven Fibonacci-Formulierung mit der Formulierung als Schleife, die Sie schon in C5 gesehen haben.

(b) Betrachten wir das Problem die (ganzzahlige, nicht-negative) Potenz n einer reellen Zahl x zu bestimmen.

- Die naheliegende, nicht-rekursive Formulierung ist

$$x^n = \prod_{i=1}^{n} x.$$

  Diese kann z.B. mit einer einfachen for-Schleife realisiert werden.

- Man kann aber auch einen rekursiven Algorithmus formulieren:

$$x^n = \begin{cases} 1 & n = 0 \\ (x^{n/2})^2 & n > 0, \text{ gerade} \\ x \cdot (x^{(n-1)/2})^2 & n > 0, \text{ ungerade} \end{cases}$$

Überzeugen Sie sich davon, dass die zweite Formulierung das erwartete Ergebnis produziert.
Wie viele Multiplikation sind zur Auswertung der nicht-rekursiven Formel, wie viele zur Auswertung der rekursiven notwendig? Welche Formulierung sollte also aus diesem Aspekt zur kürzeren Laufzeit führen.
Bemerkung: Natürlich kann x^n auch ähnlich zur Fakultät rekursiv berechnet werden. Wieviele Multiplikationen sind dann notwendig?

Aufgabe C18 (Austausch von Ziffern)

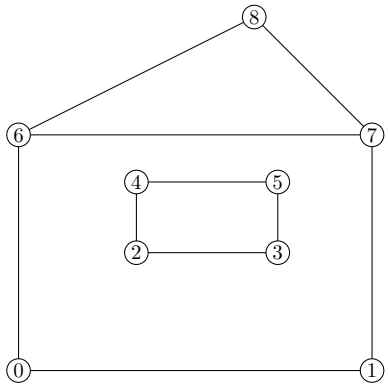
Betrachten Sie das folgende Code-Fragment. Welche Ausgaben erwarten Sie? Lassen Sie das Programm nun laufen. Überrascht Sie die Ausgabe? Was müssen Sie ändern, damit das Programm Ihre Erwartungen erfüllt? Sie können den Code auch von unserer Webseite unter Tutorium ("exchange-wrong.c") herunterladen.

```
#include <stdio.h>

void exchange(int a, int b);

void main()
{ /* WRONG CODE */
    int a, b;

    a = 5;
    b = 7;
    printf("From main: a = %d, b = %d\n", a, b);

    exchange(a, b);
    printf("Back in main: ");
    printf("a = %d, b = %d\n", a, b);
}

void exchange(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
    printf("From function exchange: ");
    printf("a = %d, b = %d\n", a, b);
}
```

Aufgabe C19 (Depth-First-Search)

Ziel dieser Aufgabe ist es den DFS-Algortihmus aus der Vorlesung zu implementieren.

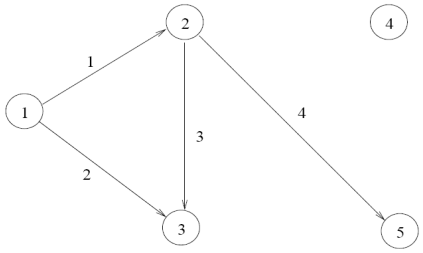
(a) Betrachten Sie den folgenden Graphen. Stellen Sie seine Adjazenzmatrix auf.



(b) *Für den Einstieg*: Nutzen die Vorlage "dfs.c", die Sie unter Tutorium auf unserer Webseite finden und implementieren Sie den DFS-Algorithmus, der in der Vorlesung vorgestellt wurde, mit Hilfe der gegebenen Adjazenzmatrix.

(c) *Für Fortgeschrittene*: Betrachten Sie folgende Methode um (gerichtete) Graphen effizient in einer Art Adjazenzliste zu speichern. Sie erlaubt die Speicherung in der Größe $\mathcal{O}(n+m)$ ohne dass verkettete Listen oder Pointer benutzt werden müssen.

Betrachten Sie als Beispiel folgenden Graphen. Dabei sind die Zahlen an den Kanten hier die Indices der Kanten und keine Gewichte.



Wir speichern den Graphen nun in folgender Form

Knoten	1	2	3	4	5
fstt	1	3	-1	-1	-1
fsth	-1	1	2	-1	4

Bögen	1	2	3	4
nxth	-1	3	-1	-1
nxtt	2	-1	4	-1

Dabei steht `fst` für *first*, `nxt` für *next*, `t` für *tail* und `h` für *head*. Beispielsweise bezeichnet also `fstt[v]` den ersten Bogen, dessen *tail* an dem Knoten v anliegt, dass heißt den ersten Bogen der von Knoten v abgeht. `nxtt[e]` bezeichnet dann den nächsten Bogen, der von demselben Knoten abgeht wie Bogen e. Im Inneren der Tabellen stehen also immer Indices von Kanten.
Wir gehen davon aus, dass wir für jede Kante ihren "head" und "tail" kennen. Ungerichtete Graphen betrachten wir als gerichtete, indem wir jede Kante durch zwei entgegengesetzt gerichtete beschreiben. Für obiges Beipiel haben wir also

Kanten	1	2	3	4
tail	1	1	2	2
head	2	3	3	5

Von der Webseite können Sie den Code "init-adjlisten.c" herunterladen. Diese Unterroutine initialisiert die Datenstrukturen `fstt, fsth, nxtt,` und `nxth` in $\mathcal{O}(m)$, gegeben die Arrays `tail` und `head`. Implementieren Sie einen DFS-Algorithmus, der diese Speicherstruktur nutzt.