BERGISCHE UNIVERSITÄT
GESAMTHOCHSCHULE WUPPERTAL

GAUSS-STRASSE 20
42097 WUPPERTAL
(Korrespondenzanschrift)
42119 WUPPERTAL
(Lieferanschrift)
TELEX 8 592 262 bughw
TELEFAX (0202) 439-2901
TELEFON (0202) 439-1

FACHBEREICH 7
MATHEMATIK

Prof. Dr. Andreas Frommer

„Dummy"-Klausur

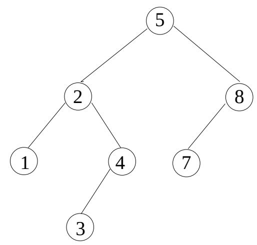
zur Vorlesung

Algorithmen und Datenstrukturen

Inhalt:

Aufgabe 1 (20 Punkte)

Gegeben ist folgender AVL-Baum



a) Geben Sie in jedem Knoten den Balance-Index an.

b) Skizzieren Sie den AVL-Baum jeweils nach
   - Einfügen eines Knotens mit Schlüssel 0,
   - anschließendem Entfernen des Knotens mit Schlüssel 5 und
   - anschließendem Entfernen des Knotens mit Schlüssel 0.

Mit d bezeichnen wir die kleinste Tiefe eines Blattes im AVL-Baum. (Im oben skizzierten AVL-Baum ist also $d = 2$, die zugehörigen Blätter haben die Schlüssel 1 und 7.)

c) Skizzieren Sie für $h = 1, 2$ und 3 jeweils einen AVL-Baum der Höhe h mit $d = \lceil \frac{h}{2} \rceil$.

d) Zeigen Sie: Zu jedem $h \in \mathbb{N}$ gibt es einen AVL-Baum der Höhe h mit $d = \lceil \frac{h}{2} \rceil$.
Hinweis: Kombinieren Sie zwei geeignete AVL-Bäume kleinerer Höhe.

Aufgabe 2 (20 Punkte)

a) Sortieren Sie die Schlüsselfolge 12, 20, 63, 3, 42 mit Heapsort. Skizzieren Sie dazu jede neue Belegung nach Einfügen/Löschen eines Schlüssels und nach Wiederherstellung der Heapeigenschaft an.

Ein 3-Heap ist ein Ternärbaum (jeder Knoten hat maximal 3 Söhne) mit folgenden Eigenschaften:

- In jedem Knoten v gilt: Die Schlüssel der Söhne von v sind höchstens so groß wie der Schlüssel von v.
- Der Baum ist dicht.

b) Skizzieren Sie einen 3-Heap, der die Schlüssel aus a) enthält.

c) Skizzieren Sie, wie ein 3-Heap mit einem Feld implementiert werden kann.
Ist der Knoten k an der Position i des Feldes gespeichert, an welchen Positionen findet man
   - den linken Sohn von k,
   - den mittleren Sohn von k,
   - den rechten Sohn von k,
   - den Vater von k?

d) Ändern Sie das beigefügte C-Programm `Heapsort` so um, dass ein 3-Heap verwendet wird. Geben Sie dazu an, welche Zeilen wie geändert werden müssen.

Aufgabe 3 (20 Punkte)

Gegeben ist eine Folge von ganzen Zahlen $a_1, a_2, \ldots, a_n$, die in einem Feld abgespeichert sind. Für $i, j \in \{1, \ldots, n\}$, $i \leq j$, sei s_{ij} der Folgenabschnitt

$$s_{ij} := \sum_{k=i}^{j} a_k \, .$$

a) Entwerfen Sie einen Algorithmus, der Indizes $i, j \in \{1, \ldots, n\}$, $i \leq j$ ermittelt, so dass der oben definierte Ausdruck s_{ij} *minimal* wird. Die Laufzeit Ihres Verfahren soll $\mathcal{O}(n)$ sein. Begründen Sie, dass dies für Ihren Algorithmus tatsächlich der Fall ist!

b) Zu welchem Algorithmen-Grundtyp zählt Ihr Verfahren?

c) Veranschaulichen Sie schematisch (unter Angabe geeigneter Zwischenergebnisse) den Ablauf des Algorithmus für das Feld

a_1	a_2	a_3	a_4	a_5	a_6	a_7
3	-2	9	-1	-1	-1	2

Aufgabe 4 (20 Punkte)

Ergänzen Sie im beiliegenden Programm die Funktion

```
void bubblesort(Liste *evl)
```

(Zeile 187), die eine einfach verkettete Liste mit Hilfe des Bubblesort-Verfahrens aufsteigend sortiert.

Hinweise:

- Es gibt hierfür zwei Lösungmöglichkeiten: Sie können zwei Elemente vertauschen, indem Sie die Verzeigerung ändern. Die langsamere, aber einfachere und daher hier empfohlene Lösung besteht darin, die Verzeigerung nicht zu ändern und stattdessen den Inhalt der einzelnen Listenknoten zu tauschen.
- Es ist erlaubt (wenn auch nicht notwendig), die Listenelemente vor dem eigentlichen Sortieren zu zählen.
- Das abgedruckte Programm entspricht dem aus der Vorlesung, bis auf die neue Funktion `groesser` (Zeilen 33 – 38) und das geänderte Hauptprogramm.

Aufgabe 5 (20 Punkte)

Gegeben ist ein gerichteter Graph $G = (V, E)$. Zu $v \in V$ bezeichne $d_{in}(v)$ den Eingangsgrad, sowie $d_{out}(v)$ den Ausgangsgrad des Knotens v.

a) Schreiben Sie zu dem beigefügten Datentyp `DCEL` eine `C`-Funktion

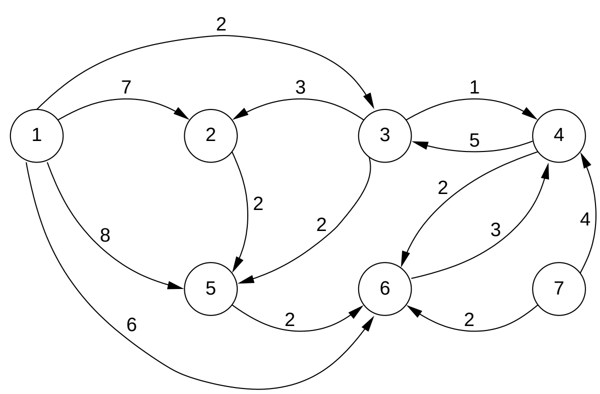
```
int inDeg(int index, DCEL *d),
```

die mit einem Aufwand von $\theta(d_{in}(v))$ zu einem Knoten $v \in V$ (repräsentiert durch eine Ganzzahl `index`) den Eingangsgrad als Funktionsergebnis liefert.
Ein Test ob `index` in den gültigen Grenzen liegt kann vernachlässigt werden. Beachten Sie auch die beigefügte schematische Darstellung des Datentypes zu Teil c).

b) Begründen Sie, warum der Aufwand zum Löschen einer Kante $e = e(v, w)$ eines gerichteten Graphen in einer Datenstruktur `DCEL` maximal $\mathcal{O}(d_{out}(v))$ ist. Wie groß ist der Aufwand bei einer Adjazenzmatrix?

c) Gegeben ist die in der beigefügten Skizze dargestellte Datenstruktur vom Typ `DCEL`. Skizzieren Sie den dazugehörigen gerichteten Graphen.

d) Auf den unten skizzierten Graphen soll der Algorithmus von Dijkstra (kürzeste Wege in nichtnegativ gewichteten Graphen) mit Startknoten **1** angewendet werden. Geben Sie dazu für jeden Schritt in Form einer Tabelle jeweils die bisher berechneten Entfernungen an. Markieren Sie außerdem in jedem Schritt die berechneten Entfernungen, welche bereits als minimal bekannt sind.



Anlage zu Aufgabe 2

```c
/****************************************************************/
/* Programm zur Vorlesung Algorithmen und Datenstrukturen SS00   */
/* Andreas Frommer 8.6.00                                        */
/*                                                               */
/* Heap-Funktionen (einfuegen und max entfernen),                */
/* sowie heapsort                                                */
/****************************************************************/

#include <stdio.h>
#define HP_SIZE 50

enum {false=0, true=1};

typedef int schluesseltyp;
typedef int rest_typ;

typedef struct datensatz
{
  schluesseltyp schluessel;
  rest_typ uebrige_daten;
}
datensatz;

typedef struct heap
{
  int pegel;
  datensatz daten[HP_SIZE];
}
heap;

/****************************************************************/

int kleiner(schluesseltyp s1, schluesseltyp s2)
{
  return s1 < s2;
}

/****************************************************************/

int kleinergleich(schluesseltyp s1, schluesseltyp s2)
{
  return s1 <= s2;
}

/****************************************************************/

void hp_einfuegen( datensatz d, heap *hp)
{

      /* fuegt neuen Datensatz d in den heap ein*/


  int  knoten;       /* zeigt auf den gerade betrachteten Knoten */
  int  vater;                          /* zeigt auf dessen Vater */
  int aufsteigen;               /* geht das "Aufsteigen" weiter ? */

  if ( hp->pegel == (HP_SIZE-1))        /* Heap ist bereits voll */
     printf("Fehler : Heap laeuft ueber\n");
  else
  {
    hp->pegel++;
                 /* den neuen Datensatz vorlaeufig am Heapende
                    einordnen und von dort "aufsteigen" lassen */
    knoten = hp->pegel;
    aufsteigen = true;
    while (aufsteigen)
    {
      vater = (knoten-1) /  2;
```

```
      aufsteigen = ( knoten > 0 );/* an der Wurzel angelangt ? */
      if (aufsteigen)
      {
         if ( kleinergleich( d.schluessel,
                             hp->daten[vater].schluessel ) )
                                      /* Aufsteigen beendet */
            aufsteigen = false;
         else           /* um eine Position aufsteigen lassen */
         {
            hp->daten[knoten] = hp->daten[vater];
            knoten = vater;
         }
      }
    }
                           /* neuen Knoten endgueltig ablegen */
    hp->daten[knoten] = d;
 }
}

/**************************************************************/

datensatz max_entfernen( heap *hp)
{

      /* liefert das oberste Element des heaps zurueck,
         entfernt es aus dem heap und stellt den heap
         wieder her                                         */



  int knoten;      /* zeigt auf den gerade betrachteten Knoten */
  int linker, rechter;
                   /* zeigt auf dessen linken / rechten Sohn */
  int maxsohn;       /* der mit dem groesseren Schluessel      */
  int absinken;     /* geht das "Absinken" weiter?            */
  datensatz max;    /* oberstes Element                       */
  schluesseltyp s;

  if (hp->pegel < 0)                /* heap ist leer, Fehler  */
      printf("Fehler: der heap ist bereits leer\n");
  else
  {                     /* oberstes Element zwischenspeichern */
    max = hp->daten[0];
                         /* Schluessel des sinkenden Elements */
    s = hp->daten[hp->pegel].schluessel;
    hp->pegel--;         /* das sinkende Element bleibt zunaechst
                            an Stelle hp->pegel +1 gespeichert */

    /* absinken lassen  */

    knoten = 0;
    absinken=true;

    do
    {
      linker = 2*knoten + 1;
      rechter = 2*knoten + 2;

      if (linker <= hp->pegel)    /* wenigstens ein Sohn vorhanden  */
      {
        /* bestimme den groesseren */

        maxsohn = linker;
        if ( (rechter <= hp->pegel) &&
             (kleiner(hp->daten[linker].schluessel,
                          hp->daten[rechter].schluessel) ) )
                /* falls kein rechter Sohn vorhanden ergibt sich false */
          maxsohn = rechter;

        /* sinken, falls erforderlich */

```

```
        if ( kleinergleich(s,hp->daten[maxsohn].schluessel) )
        {
           hp->daten[knoten] = hp->daten[maxsohn];
           knoten = maxsohn;
        }
        else
          absinken = false;
      }
      else
                                    /* Knoten ist Blatt */
          absinken = false;
    }
    while (absinken);
                        /* gesunkenen Knoten endgueltig ablegen */
    hp->daten[knoten] = hp->daten[hp->pegel+1];
  }
  return max;
 }

/*************************************************************/

void hp_ausgeben(heap hp)
{
  int i;

  for (i=0; i<=hp.pegel; i++)
     printf("%d ",hp.daten[i].schluessel);
  printf("\n");
}

/*************************************************************/

int main()
{
  int n;
  heap hp;
  datensatz d;
  int i;

  hp.pegel = -1;    /* leeren Heap initialisieren */

  /* Datensaetze einlesen, nur Schluessel  */

  printf("Anzahl der Datensaetze\n");
  scanf("%d",&n);

  for (i=0;i < n; i++)
  {
    printf("Schluessel von Datensatz Nr. %d \n",i);
    scanf("%d",&d.schluessel);
    hp_einfuegen(d,&hp);
  }

  /* sortiert ausgeben */

  printf("Sortiert: \n");

  for (i=0;i < n; i++)
  {
    d = max_entfernen(&hp);
    printf("%d  ",d.schluessel);
  }

  printf("\n");

  return 0;
}
```

Anlage zu Aufgabe 4

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

typedef int Eintrag_typ;                     /* Typ der Eintraege */

typedef struct List_el
/* Element einer Liste von Eintraegen vom Typ Eintrag_typ */
{
  Eintrag_typ Eintrag;
  struct List_el *next;           /* Zeiger auf naechstes Element */
} List_el;

typedef struct Liste
/* besteht aus Anfangszeiger, Zeiger auf aktuelles Element und
   dessen Vorgaenger */
{
  List_el *anf;
  List_el *z;
  List_el *z_vorg;
} Liste;

/*******************************************************************/

int equal(Eintrag_typ eintrag1, Eintrag_typ eintrag2)
{
  /* testet zwei Eintreage auf Gleichheit */

  return (eintrag1 == eintrag2);
}

/*******************************************************************/

int groesser(Eintrag_typ eintrag1, Eintrag_typ eintrag2)
{
  /* testet, ob eintrag 1 > eintrag 2 */

  return (eintrag1 > eintrag2);
}

/*******************************************************************/

void create(Liste *evl)
{
  /* erzeugt eine leere List evl */

  evl->anf=NULL ;
  evl->z=NULL ;
  evl->z_vorg=NULL;
}

/*******************************************************************/

int empty(Liste evl)
{
  /* testet, ob Liste evl leer ist */
  return (evl.anf == NULL);
}

/*******************************************************************/

int out_of_list(Liste evl)
{
  /* zeigt der aktuelle Zeiger ueber das Listenende hinaus? */
  return (evl.z == NULL);
}

/*******************************************************************/

void advance(Liste *evl)
```

```
{
  if (evl->z == NULL)
    {
      printf("bereits am Ende der Liste\n");
      exit(EXIT_FAILURE);
    }
  else
    {
      evl->z_vorg = evl->z;
      evl->z = evl->z->next;
    }
}

/*******************************************************************/

void reset(Liste *evl)
{
  evl->z = evl->anf;
  evl->z_vorg = NULL;
}

/*******************************************************************/

void write_list(Liste evl)
{
  /* Gibt die komplette Liste aus */

  reset(&evl);
  while (!out_of_list(evl))
    {
      printf("%i ", evl.z->Eintrag);
      advance(&evl);
    }
  printf("Listenende\n");
}

/*******************************************************************/

void insert(Liste *evl, Eintrag_typ eintrag)
{
  /* Fuegt ein neues Element mit Eintrag eintrag an der aktuellen
     Position ein                                                  */

  List_el *neu;

  if ((neu = (List_el*)malloc(sizeof(List_el))) == NULL)
    {
      printf("Kein Speicher mehr frei!\n");
      exit(EXIT_FAILURE);
    }

  neu->Eintrag = eintrag; /*Eintrag in neues Li.-element einfuegen*/
  /* Rest der Liste anhaengen */
  if (evl->z==NULL)      /* Ende markieren beim Einfuegen am Ende */
    neu->next = NULL;
  else
    neu->next = evl->z;               /* Rest der Liste anhaengen */

  if ( (evl->anf == evl->z) )             /* Einfuegen am Anfang  */
    {
      evl->anf = neu;
      evl->anf->next = evl->z;
    }
  else
    {
      evl->z_vorg->next = neu;
    }
}

/*******************************************************************/

```

```
void search(Liste *evl, Eintrag_typ eintrag)
{
  /* Sucht den Eintrag eintrag der Liste;
     wird der Eintrag gefunden, so zeigt der Zeiger z auf
     das entsprechende Listenelement                      */

  reset(evl);
  while (!out_of_list(*evl) && !equal(evl->z->Eintrag,eintrag))
    advance(evl);
}

/*****************************************************************/

void delete(Liste *evl)
{
  /* loescht das Listenelement an der aktuellen Zeigerposition */

  if (evl->z == NULL)
    {
      printf("aktueller Zeiger zeigt nicht auf Listenelement");
      exit(EXIT_FAILURE);
    }
  else
    {
      if (evl->z_vorg == NULL)           /* am Anfang loeschen */
        evl->anf = evl->z->next;
      else
        evl->z_vorg->next = evl->z->next;
      free(evl->z);
    }
}

/*****************************************************************/

void delete_list(Liste *evl)
{

  while (!empty(*evl))
    {
      reset(evl);
      delete(evl);
    }
}

/*****************************************************************/

void bubblesort(Liste *evl)
{
  /* HIER ERGAENZEN */
}

/*****************************************************************/

int main(void)
{
  Liste evl;
  Eintrag_typ eintrag;
  int i, n;

  create(&evl);

  printf("Anzahl der Elemente: ");
  scanf("%i", &n);
  for (i = 0; i < n; i++)
    {
      printf("%i-tes Element: ", i);
      scanf("%i", &eintrag);
      reset(&evl);
      insert(&evl, eintrag);
    }

```

```
  printf("unsortiert: ");
  write_list(evl);

  bubblesort(&evl);

  printf("sortiert:   ");
  write_list(evl);

  return 0;
}
```

Anlage zu Aufgabe 5

Teil a: Die Datenstruktur `DCEL`

```
#define NMAX 100

typedef struct kante
{
  int anf, end;
  struct kante *nachf1, *vorg1;
  struct kante *nachf2, *vorg2;
} kante;

typedef struct
{
  kante *anfknoten[NMAX];
  kante *endknoten[NMAX];
  int n;
} DCEL;
```

Teil c: Eine belegte Datenstruktur vom Typ `DCEL` (`n=5`)