Technische Universität München
Institut für Informatik
Dr. rer. nat. Tobias Weinzierl
Kaveh Rahnema

März 2013

Kompaktkurs Einführung in die Programmierung

Übungsblatt 9: Verkettete Listen

Lernziele

- Erweiterter Umgang mit Zeigern und Referenzen.
- Verwendung dynamischer Datenstrukturen (Listen).

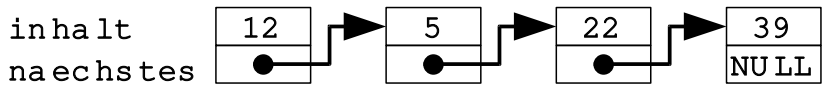
1) Liste und Zeiger (T)

Aus der Vorlesung sind Ihnen verkettete Listen und deren Verwendung bekannt. Notieren Sie sich die Ausgabe des folgenden Programms ohne Verwendung eines Compilers. Anschließend könnten Sie Ihren Versuch verifizieren.

```
struct SX {
    int x;
    struct SX* p;
};

int main() {
    struct SX s1 = {3, NULL};
    struct SX *p1 = &s1, *p2 = new struct SX;

    p2->x = 2;
    p2->p = &s1;
    std::cout <<    s1.p            << std::endl;
    std::cout <<    p1->p           << std::endl;
    std::cout <<    (*p1).x         << std::endl;
    std::cout <<    p2->x           << std::endl;
    std::cout <<    p2->p->x        << std::endl;
    std::cout <<    (*(p2->p)).x    << std::endl;

    p1->p = p2;
    std::cout <<    p1->p->x        << std::endl;

    p1->p = (*p2).p;
    std::cout <<    p1->p->p->p->x  << std::endl;

    p1 = p2;
    s1.p = p1;
    std::cout <<    (*(*((*p1).p)).p).x  << std::endl;

    delete p2;
}
```

2) Einfach verkettete Liste (P)

In dieser Aufgabe wird eine Datenstruktur für (einfach) verkettete Listen implementiert. Da es hier nur auf die Organisation der Liste ankommt, wird der Einfachheit halber wieder in jedem Listenelement (von dem zu definierenden Strukturtyp `struct element`) als Inhalt nur eine Ganzzahl gespeichert, dazu kommt noch ein Zeiger auf das nächste Element der Liste. Die Liste `(12, 5, 22, 39)` wird dann wie gewohnt gespeichert:



Es gilt:

- Eine Liste wird durch einen Zeiger auf das erste Element repräsentiert bzw. – im Fall einer leeren Liste – durch den Zeiger `NULL`.
- Das Listenende ist dadurch markiert, dass statt eines Zeigers auf ein nächstes Element `NULL` gespeichert wird.

Nun soll folgendes getan werden:

i) Definieren Sie dazu einen Strukturtyp `struct element`, der eine Zahl `int inhalt` und einen Zeiger `struct element *naechstes` auf den nächsten Listeneintrag enthält.

   Um etwas in die Liste hineinzubekommen, schreiben Sie eine Funktion

   ```
   struct element *anfuegen(int x, struct element *liste)
   ```

   die an eine gegebene Liste `liste` **hinten** ein neues Element mit `x` als Inhalt anfügt (vorne anfügen kennen wir ja schon) und die verlängerte Liste als Ergebnis zurückliefert. Dazu ist Speicherplatz für eine neue Struktur vom Typ `element` anzufordern und passend zu belegen. Anschließend ist der Zeiger des bisher letzten Listenelements passend „umzubiegen“. Vorsicht mit der leeren Liste (`liste == NULL`), die eine gewisse Sonderrolle spielt!

   Dadurch, dass die neue Liste zurückgeliefert wird, kann man durch einen verschachtelten Aufruf in einem Ausdruck die ganze Liste aus dem Beispiel aufbauen:

   ```
   anfuegen(39, anfuegen(22, anfuegen(5, anfuegen(12, NULL))))
   ```

ii) Schreiben Sie eine Funktion `liste_drucken(struct element* liste)` zum Ausdrucken von angelegten Listen (damit Sie Ihre Listen auch sehen können).

iii) Nun geht es an eine Funktion

   ```
   struct element *umkehren(struct element *liste)
   ```

   die die Reihenfolge der Elemente von `liste` umkehrt – also aus `(12, 5, 22, 39)` die Liste `(39, 22, 5, 12)` erzeugt. Analog zu `anfuegen` soll die Umordnung nur durch Verändern der Zeiger bewerkstelligt werden, es sind hier also gar keine neuen Elemente zu erzeugen; das Ergebnis der Funktion soll die neue Liste sein.

   Hinweise:

   - Alternativ können Sie auch versuchen vollständig auf der übergebenen Liste zu operieren und keine neue liste zurückgeben.

- Wie immer ist es ratsam, Ihr Programm erstmal mit Anweisungen zu versehen und die Informationen über den Zustand der verwendeten Größen während des Programmlaufs z.B. über Ihre Funktion `liste_drucken` ausgeben zu lassen.
- Denken Sie auch an den Sonderfall `liste == NULL`.

iv) Schreiben Sie ein geeignetes Hauptprogramm, mit dem Sie Ihre geschriebenen Funktionen ausprobieren können.

v) ★ Achten Sie wiederum auf korrekte Freigabe aller Daten des Freispeichers.

3) Verfahren von Hockney/Golub (★)

Das Verfahren von Hockney/Golub kann zum Lösen von tridiagonalen linearen Gleichungssystemen $Ax = d$ mit

$$A = \begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \mathbf{0} \\ & a_3 & b_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & \mathbf{0} & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix} \in \mathbb{R}^{n\times n}, \quad d = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix} \in \mathbb{R}^n, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

verwendet werden. Mit $N := \lceil \log_2 n \rceil$ und den Definitionen

$$\begin{aligned} a_i^{(0)} &:= a_i\,, \quad i = 2,\dots,n\,, & a_1^{(0)} &:= 0 \\ b_i^{(0)} &:= b_i\,, \quad i = 1,\dots,n\,, & & \\ c_i^{(0)} &:= c_i\,, \quad i = 1,\dots,n-1\,, & c_n^{(0)} &:= 0 \\ d_i^{(0)} &:= d_i\,, \quad i = 1,\dots,n\,, & & \end{aligned}$$

$$\begin{aligned} a_i^{(k)} := c_i^{(k)} := d_i^{(k)} &:= 0\,, \quad b_i^{(k)} := 1 \\ \text{für } i \in \mathbb{Z}\backslash\{1,\dots,n\}\,, &\quad k = 0,\dots,N \end{aligned}$$

werden die Werte $a_i^{(k)}$, $b_i^{(k)}$, $c_i^{(k)}$, $d_i^{(k)}$ rekursiv berechnet über:

$$\begin{aligned} a_i^{(k)} &:= \alpha_i^{(k-1)}\, a_{i-2^{k-1}}^{(k-1)} \\ c_i^{(k)} &:= \gamma_i^{(k-1)}\, c_{i+2^{k-1}}^{(k-1)} \\ b_i^{(k)} &:= \alpha_i^{(k-1)}\, c_{i-2^{k-1}}^{(k-1)} + b_i^{(k-1)} + \gamma_i^{(k-1)}\, a_{i+2^{k-1}}^{(k-1)} \\ d_i^{(k)} &:= \alpha_i^{(k-1)}\, d_{i-2^{k-1}}^{(k-1)} + d_i^{(k-1)} + \gamma_i^{(k-1)}\, d_{i+2^{k-1}}^{(k-1)} \end{aligned}$$

mit

$$\begin{aligned} \alpha_i^{(k-1)} &:= -a_i^{(k-1)} / b_{i-2^{k-1}}^{(k-1)} \\ \gamma_i^{(k-1)} &:= -c_i^{(k-1)} / b_{i+2^{k-1}}^{(k-1)}. \end{aligned}$$

Damit ist die folgende Gleichung gültig für $i \in \mathbb{Z}$:

$$a_i^{(k)}\, x_{i-2^k} + b_i^{(k)}\, x_i + c_i^{(k)}\, x_{i+2^k} = d_i^{(k)}. \tag{1}$$

Das Verfahren von Hockney/Golub benutzt (1) um $Ax = d$ zu lösen. Es besteht somit aus den Schritten:

1. Berechne die Werte $a_i^{(k)}$, $b_i^{(k)}$, $c_i^{(k)}$, $d_i^{(k)}$ für $k = 1,\dots,N$, $i = 1,\dots,n$.

2. Berechne $x_i = d_i^{(N)}/b_i^{(N)}$ für $i = 1, \ldots, n$.

i) Versuchen Sie obigen Algorithmus von Hockney/Golub zu verstehen.

ii) Implementieren Sie nun das Verfahren anhand der Punkte 1. und 2. .

iii) Was könnten die Vorteile dieses Verfahrens sein im Hinblick auf Parallelisierung.

iv) Ist das Verfahren für beliebige tridiagonale Systeme anwendbar?