Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

Fachbereich Informatik und Kommunikation
Prof. Dr. Ulrike Griefahn
Dr. Hansjürgen Paul

Prozedurale Programmierung (PPR)

WS 2020/21

Praktikumsaufgabe 9 (Huffman-Kodierung)

In dieser Aufgabe sollen Sie die Huffman-Kodierung vervollständigen, d.h., Ihr Projekt aus Aufgabe 6 um die letzten für die Huffman-Kodierung notwendigen Schritte ergänzen.

Ihr Progamm soll dann in der Lage sein, anhand des Verfahrens der Huffman-Kodierung Dateien zu komprimieren und wieder zu dekomprimieren. Die Algorithmen zum Komprimieren und Dekomprimieren nach dem Huffman-Verfahren haben Sie in der Vorlesung ADS kennen gelernt.

In dieser Aufgabe müssen Sie im Wesentlichen die beiden Funktionen

```
void compress(char *in_filename, char *out_filename);
void decompress(char *in_filename, char *out_filename);
```

die bislang nur den Inhalt der Dateien entweder bit- oder byteweise kopieren, so ändern, dass sie die Eingabedatei komprimieren bzw. dekomprimieren und das Ergebnis in die zweite der angegebenen Dateien ausgeben.

Ablauf des Huffman-Verfahrens:

Die Komprimierung einer Datei verläuft in den folgenden Schritten:

1. Ermitteln der Häufigkeit der in der Eingabedatei vorkommenden Zeichen
2. Aufbau des optimalen Codebaums aus den Häufigkeiten
3. Ableiten der Codetabelle aus dem Codebaum
4. Schreiben der Häufigkeiten aus Schritt 1 in die Ausgabedatei (Diese Information ist notwendig, damit die Datei später wieder dekomprimiert werden kann.)
5. Kodieren der Zeichen und Schreiben der Codes in die Ausgabedatei

Die Dekomprimierung einer komprimierten Datei verläuft in den folgenden Schritten:

1. Einlesen der Häufigkeiten aus der komprimierten Datei
2. Aufbau des optimalen Codebaums aus den Häufigkeiten
3. Dekodieren der weiteren Zeichen der Eingabedatei anhand des Codebaums und Schreiben der Zeichen in die dekomprimierte Ausgabedatei.

Hinweise zu dieser Praktikumsaufgabe

- In Schritt 2 können Sie den optimalen Codebaum mit Hilfe des in Blatt 7 realisierten Heaps und des in Blatt 8 realisierten Binärbaums aufbauen.
- Um die Häufigkeiten der im Text vorkommenden Zeichen als Daten im Binärbaum abzulegen, können Sie das Modul `frequency` aus der Vorlesung verwenden. Da Sie nur Zeichen und keine Wörter ablegen müssen, bietet es sich an, das Modul entsprechend anzupassen.
- Den optimalen Codebaum können Sie mit Hilfe des binären Heaps aus Aufgabe 7 aufbauen. Legen Sie dazu Binärbäume im Heap ab und fassen Sie immer die beiden kleinsten zu einem neuen Codebaum zusammen. Sie benötigen dazu eine Funktion, mit der Sie zwei Binärbäume miteinander vergleichen können. Definieren Sie daher im Modul `huffman` eine Funktion, die Binärbäume entsprechend der in der Wurzel abgelegten Häufigkeit vergleicht und übergeben Sie diese Funktion dem Heap.
- Da Sie bei der Huffman-Komprimierung die Häufigkeiten in die Ausgabedatei schreiben und diese **int**-Werte enthält, bietet es sich an, im Modul `io` zwei weitere Lese- und Schreibfunktionen für **int**-Werte zu definieren. Realisieren Sie daher die Funktionen

  ```
  extern unsigned int read_int(void);
  extern void write_int(unsigned int i);
  ```

  die den nächsten **int**-Wert aus dem Eingabestrom liefern bzw. einen **int**-Wert in den Ausgabestrom schreiben. Sie können beim Lesen davon ausgehen, dass immer ausreichend Zeichen für einen **int**-Wert in der Eingabedatei vorhanden sind, d. h., Sie müssen hier nicht auf `EOF` prüfen. Definieren Sie eine Union, so dass Sie auf die einzelnen Bytes eines **int**-Werts zugreifen können (siehe Übungsaufgabe). Sie können die Funktionen dann unter Verwendung von `read_char` bzw. `write_char` implementieren.
- Da Sie bei der Komprimierung bitweise in die Ausgabedatei speichern, ist nicht gesagt, dass das letzte Byte voll belegt ist. Überlegen Sie, wie Sie damit umgehen.
- Die Kommandozeilen-Schnittstelle von Blatt 6 erlaubt die Angabe eines Levels. Diesen können Sie in dieser Aufgabe ignorieren.
- Behandeln Sie alle Dateien (komprimierte und nicht komprimierte) als Binärdateien und nicht als Textdateien, d.h. verwenden Sie die Modi `"rb"` bzw. `"wb"`, wenn Sie die Dateien zum Lesen bzw. zum Schreiben öffnen.
- Testen Sie am Anfang mit kleinen Dateien, deren Komprimierung Sie detailliert prüfen können. Lassen Sie sich dazu die komprimierte Datei hexadezimal anzeigen, bspw. mit Notepad++ und dem Plugin HEX-Editor).
- Mit dem Befehl `fc` (Windows) bzw. `diff` (Linux) können Sie zwei Dateien miteinander vergleichen, so dass Sie überprüfen können, ob eine kodierte und wieder dekodierte Datei mit dem Original übereinstimmt. Im Moodle-Kurs finden Sie als Zusatzmaterial einige Dateien, mit denen Sie Ihre Implementierung testen können.