Inf Progtech 1, Praktikums-Beispiel

Radix Sort

Schreib ein Programm, das eine *verkettete Liste von Datenelementen* mittels Radix Sort *sortiert*.

Der Radix Sort ist in mehrfacher Hinsicht ein ungewöhnliches Sortier-Verfahren:

- Seine Laufzeit wächst nur *linear* (O(n)) mit der *Anzahl* der zu sortierenden Werte (und nicht mit mindestens O(n * log(n)) wie bei den anderen Sortierverfahren), aber dafür auch *linear* mit der *Sortierschlüssel-Länge*.
- Möglich ist das, weil er *nie* ganze Elemente bzw. Sortierschlüssel *vergleicht*.

  Genauer: Der Radix-Sort vergleicht überhaupt nichts.
  Die einzige Operation auf dem Sortierschlüssel besteht darin,
  den Wert eines *Teils des Schlüssels als Index* zu verwenden.
- Der Radix Sort arbeitet intern mit *Listen*, nicht mit Arrays:

  Sowohl die Eingabe als auch die Ausgabe sind eine *einfach verkettete Liste*
  von Elementen. Braucht man ein Array als Ergebnis,
  muss man die Ergebnisliste ein Mal durchlaufen und umkopieren.

Die Voraussetzung für die Anwendbarkeit des Radix Sort ist, dass der Sortierschlüssel

- bei allen Elementen *gleich lang* (und zwar *möglichst kurz*) ist,
- aus *mehreren einzelnen Zeichen* besteht (oder zum Beispiel
  aus einem langen Integer, der sich in Teile zu je 8 Bit aufspalten lässt),
- die Sortierreihenfolge des Gesamtschlüssels gleich der *Sortierreihenfolge der einzelnen Zeichen* oder Bitgruppen lexikographisch von links nach rechts entspricht,
- und die *Anzahl der Möglichkeiten für ein einzelnes Zeichen fix* und nicht zu groß ist (256 Wertmöglichkeiten pro Zeichen sind sehr praktisch, alles darüber ist zunehmend ungünstig).

Sozialversicherungs- oder Matrikelnummern, Kfz-Kennzeichen etc. erfüllen beispielsweise diese Kriterien, normale Namen oder Adressen nicht, weil sie zu lang und vor allem nicht gleich lang sind.

Die *Idee* ist grundsätzlich folgende:

- Man braucht *so viele Sortier-Durchgänge*, wie der Sortierschlüssel Zeichen hat.
- *In jedem Durchgang sortiert man nach einem Zeichen, und zwar im ersten Durchgang nach dem hintersten Zeichen und im letzten nach dem vordersten (!!!).*
- Dabei achtet man darauf, dass man bei Daten mit gleichem Zeichen
  die *schon bestehende Sortierung* nach den Zeichen dahinter *erhält*.

Ein *einzelner Sortier-Durchgang* funktioniert wie folgt:

- Man hat *pro möglichem Wert des Zeichens eine Liste*
  (also 256 Listen bei 256 möglichen Werten für ein Zeichen).
- Am Anfang setzt man alle diese Listen auf "leer".
- Man geht die Liste mit den zu sortierenden Daten von vorne Element für Element durch und *verteilt die Elemente* (je nachdem, welchen Wert das Zeichen an der Position hat, nach der gerade sortiert wird) *auf die Listen* (*hinten anhängen*, nicht vorne, damit die schon bestehende Sortierung erhalten bleibt!).
- Wenn man alle Elemente verteilt hat, *hängt man die einzelnen Listen in aufsteigender Reihenfolge* wieder zu einer einzigen Liste *zusammen*.

Als Datenelemente und zugleich Sortierschlüssel verwenden wir Strings mit fixer, kurzer Länge (z.B. 8 Zeichen plus `'\0'`, das würde den deutschen Kfz-Kennzeichen entsprechen).
Der Radixsort soll in einer *eigenen Funktion* implementiert werden.
Für die Länge des Sortierschlüssels und die Anzahl der Möglichkeiten pro Zeichen sollen Konstanten eingeführt werden.

Dein Hauptprogramm soll diese Sortierfunktion mit *zufällig erzeugten Daten* testen:

- Das Programm wird mit der *Anzahl* der zu sortierenden Datenelemente aufgerufen.
- Es soll eine *verkettete Liste* mit der angegebenen Anzahl von Elementen erzeugen und das Textfeld jedes Elementes mit *zufälligen Zeichen füllen*
  (nur anzeigbare Zeichen, ev. nur Buchstaben, keine Steuerzeichen!).
- Danach wird die *Sortierfunktion aufgerufen*
  und schließlich das Ergebnis zeilenweise ausgegeben.

Hinweise:

- Lege die Listenelemente *dynamisch* an, erkenne auch, wenn die Speicherallokation fehlschlägt!