Elementare Sortierverfahren

Algorithmen und Datenstrukturen I

Prof. Dr. Oliver Braun

Letzte Änderung: 18.11.2019 07:38

Sortieren

- Untersuchungen zeigen seit Jahren, dass mehr als ein Viertel der kommerziell verbrauchten Rechenzeit sortiert wird
- wir haben eine Menge von Datensätzen
- jeder Satz hat einen Schlüssel
- zwischen Schlüsseln ist eine Ordnungsrelation $<$ oder $\leq$ definiert
- außer dem Schlüssel können Datensätze weitere Komponenten enthalten
- wenn nicht anders festgelegt, sind Schlüssel ganzzahlig

Das Sortierproblem

- gegeben eine Folge von Sätzen $s_1, ..., s_N$
- jeder Satz s_i hat einen Schlüssel k_i
- gesucht wird eine Permutation π der Zahlen 1 bis N
- so dass die Umordnung der Sätze gemäß π die Schlüssel in aufsteigende Reihenfolge bringt

$$k_{\pi(1)} \leq k_{\pi(2)} \leq ... \leq k_{\pi(N)}$$

- gegeben `N` Datensätze in einem Feld (Array) `a`
- Ziel: $a[1].key \leq a[2].key \leq ... \leq a[N].key$
- zur Messung der Laufzeit:
  - Anzahl Schlüsselvergleiche (*comparsions*) im besten und schlechtesten Fall und im Mittel:

$$C_{min}(N), C_{max}(N) \text{ und } C_{mit}(N)$$

  - Anzahl Bewegungen (*movements*) im besten und schlechtesten Fall und im Mittel:

$$M_{min}(N), M_{max}(N) \text{ und } M_{mit}(N)$$

Sortieren durch Auswahl

Methode

- bestimme die Position j_1, an der das Element mit dem kleinsten Schlüssel unter $a[1], ..., a[N]$ auftritt und vertausche $a[1]$ mit $a[j_1]$
- dann bestimme j_2, an der das Element mit dem kleinsten Schlüssel unter $a[2], ..., a[N]$ auftritt und vertausche $a[2]$ mit $a[j_2]$
- ...
- bis alle Elemente an ihrem richtigen Platz stehen

C++

```
1 void selectionsort(int *a, int n) {
2     for (int i = 0; i < n - 1; i++) {
3         int min = i;
4         for (int j = i + 1; j < n; j++)
5             if (a[j] < a[min])
6                 min = j;
7         int t = a[min];
8         a[min] = a[i];
9         a[i] = t;
10     }
11 }
```

- Schlüsselvergleiche in Zeile 5
- Bewegungen in Zeilen 7-9

- Anzahl der Schlüsselvergleiche unabhängig von der Ausgangsanordnung jeweils $(N-i)$, d.h.

$$C_{min}(N) = C_{max}(N) = C_{mit}(N) = \sum_{i=1}^{N-1}(N-i)$$

$$= \sum_{i=1}^{N-1} i = \frac{N(N-1)}{2} = \Theta(N^2)$$

- Anzahl der Bewegungen

$$M_{min}(N) = M_{max}(N) = M_{mit}(N) = 3(N-1) = \Theta(N)$$

Satz

Jeder Algorithmus zur Bestimmung des Minimums von N *Schlüsseln, der allein auf Schlüsselvergleichen basiert, muss wenigstens* $N-1$ *Schlüsselvergleiche ausführen.*

Sortieren durch Einfügen

Methode

- die N zu sortierenden Elemente werden nacheinander betrachtet und in die bereits sortierte, anfangs leere Teilfolge an die richtige Stelle eingefügt

C++

```
1  void insertionsort(int *a, int n) {
2      for (int i = 1; i < n; i++) {
3          int j = i;
4          int t = a[i];
5          while (j && a[j - 1] > t) {
6              a[j] = a[j - 1];
7              j--;
8          }
9          a[j] = t;
10     }
11 }
```

- Schlüsselvergleiche in Zeile 5
- Bewegungen in Zeilen 4, 6 und 9

- zum Einfügen des i-ten Elements mindestens 1 und höchstens i Schlüsselvergleiche, d.h.

$$C_{min}(N) = N - 1; \qquad C_{max}(N) = \sum_{i=2}^{N} i = \Theta(N^2)$$

- zum Einfügen des i-ten Elements werden mindestens 2 und höchstens $i + 1$ Bewegungen ausgeführt, d.h.

$$M_{min}(N) = 2(N-1); \qquad M_{max}(N) = \sum_{i=2}^{N} (i+1) = \Theta(N^2)$$

Mittlerer Aufwand

- im Mittel kann man erwarten, dass in jedem Schritt die Hälfte der Elemente im bereits sortierten Anfangsstück größer als das aktuell einzufügende Element ist
- damit ist Sortieren durch Einfügen von der Größenordnung

$$\sum_{i=1}^{N} \frac{i}{2} = \Theta(N^2)$$

Shellsort

Methode

- vorgeschlagen von Donald Lewis Shell
- Sortieren mit abnehmenden Inkrementen
- Elemente sollen in größeren Sprüngen schneller an ihre endgültige Position gebracht werden
- wir benötigen eine mit 1 endende Folge von abnehmenden Inkrementen $h_i, t \geq i \geq 1$
- dann werden der Folge der Inkremente nach die Teilfolgen aller Elemente, die h_i Positionen voneinander entfernt sind, mittels Einfügesort sortiert
    - die gesamte Folge heißt dann jeweils h_i-sortiert

```
void shellsort(int *a, int n, vector<int> incs) {
    for (auto h : incs) {
        for (int i = h; i < n; i++) {
            int j = i;
            int t = a[i];
            bool continue_ = true;
            while (a[j-h] > t && continue_) {
                a[j] = a[j-h];
                j = j - h;
                continue_ = j > h;
            }
            a[j] = t;
        }
    }
}
```

Analyse

- wichtigste Frage: Welche Folge abnehmender Inkremente soll verwendet werden um möglichst effizient zu sein?
- eine Reihe überraschender, aber insgesamt unvollständiger Erkenntnisse:
    - man kann zeigen, dass die Laufzeit $O(Nlog^2N)$ ist, wenn man als Inkremente alle Zahlen der Form 2^p3^q wählt, die kleiner als N sind
    - Herstellen einer h-sortierten Folge aus einer bereits k-sortierten Folge zerstört die k-Sortiertheit nicht

Bubblesort

Methode

- als Bewegungen wird nur das Vertauschen zweier benachbarter Elemente zugelassen
- die Folge wird solange wiederholt durchlaufen, bis im letzten Durchlauf keine Vertauschungen mehr vorgenommen wurden
- größere Elemente haben die Tendenz, wie Luftblasen im Wasser, nach oben aufzusteigen
  - daher der Name Bubblesort

```cpp
void bubblesort(int *a, int n) {
    bool swappedSomething;
    do {
        swappedSomething = false;
        for (int i = 0; i < N - 1; i++) {
            if (a[i] > a[i + 1]) {
                int t = a[i];
                a[i] = a[i + 1];
                a[i + 1] = t;
                swappedSomething = true;
            }
        }
    } while (swappedSomething);
}
```

Analyse

- ist die Folge bereits sortiert, ergibt sich

$$C_{min}(N) = N - 1; \qquad M_{min}(N) = 0$$

- ungünstigester Fall: Schlüssel in Folge absteigend sortiert
  - N Durchläufe mit je $N - 1$ Schlüsselvergleichen, d.h.

$$C_{max}(N) = N(N-1) = \Theta(N^2)$$

  - beim i-ten Durchlauf sind $N - i$ Vertauschungen, also $3(N - i)$ Bewegungen, d.h.

$$M_{max}(N) = \sum_{i=1}^{N-1} 3(N-i) = \Theta(N^2)$$

- man kann auch zeigen: $C_{mit}(N) = M_{mit}(N) = \Theta(N^2)$