Samenvatting Algoritmen I

26 mei 2018

Inhoudsopgave

Deel I

Rangschikken

Hoofdstuk 1

Inleiding

Rangschikken is het sorteren van gegevens van klein naar groot.
De performantie van een algoritme hangt af van

- het aantal gegevens (= n)
- de aard van die gegevens
- de mate waarin de gegevens reeds geordend zijn
- de plaats van de gegevens (*inwendig* en *uitwendig*)

Naast de performantie is ook belangrijk:

- Het geheugengebruik.
- De stabiliteit.

Hoofdstuk 2

Eenvoudige methodes

2.1 Rangschikken door tussenvoegen

2.1.1 Insertion Sort

```
void insertion_sort(vector<T>& v){
    for(int i = 1; i < v.size(); i++){
        T h = move(v[i]);
        int j = i - 1;
        while(j >= 0 && h < v[j]){
            v[j + 1] = move(v[j]);
            j--;
        }
        v[j + 1] = move(h);
    }
}
```

Stappen

1. Overloop heel de tabel startend vanaf $i = 1$.
2. Sla de waarde van het element op index i tijdelijk op in h.
3. Controleer of h kleiner is dan de waarde op index $j = i - 1$.
4. Indien ja, verplaats de waarde van index j naar de index $j + 1$.
5. Blijf dit doen totdat $j = 0$ of tot dat er een element gevonden is dat kleiner is als h.
6. De waarde h moet geplaatst worden op index $j + 1$.

Analyse

Insertion Sort werkt door elementen één per één op te schuiven. Per schuifoperatie is er dus hoogsten één sleutelvergelijking. Helemaal op het einde als de tabel gesorteerd is, wordt er nog één keer gecontroleerd of alles op zijn plaats staat, dit is een extra term n. Een paar elementen dat niet in de juiste volgorde staan zijn een inversie.
Voorbeeld met n = 4.

|4|3|2|1|

- 4 vormt een inversie met 3, 2 en 1
- 3 vormt een inversie met 2 en 1
- 2 vormt een inversie met 1

Het aantal inversies is $\frac{4 \cdot 3}{2} = 6$. Er zullen 10 sleutelvergelijkingen en 6 schuifoperaties zijn.

- *slechtste* geval — $\Theta(n^2)$: Aangezien er $n(n-1)/2$ inversies zijn , zijn er $n(n-1)/2$ schuifoperaties en $n(n-1)/2 + n$ sleutelvergelijkingen.
- *gemiddelde* geval — $\Theta(n^2)$: Elke permutatie van de tabelelementen zijn even waarschijnlijk. Gemiddeld is het aantal inversies gelijk aan $\frac{n(n-1)}{4} (= \frac{n(n-1)}{2} \cdot \frac{1}{2})$
- *beste* geval — $O(n)$: De tabel is al gesorteerd en de while lus zal dus nooit uitgevoerd worden. Er is ook een speciaal geval als de tabel *bijna* in juiste volgorde staat zolang het aantal inversies $O(n)$ is.

Eigenschappen

- Sorteert ter plaatse
- Sorteert niet stabiel. Insertion Sort kan stabiel sorteren indien de lus achterstevoren uitgevoerd wordt.

2.1.2 Shell Sort

```
void shell_sort(vector<T>& v){
    int k = ... // initieel increment
    while(k >= 1){
        for(int i = k; i < v.size(); i++){
            T h = move(v[i]);
            int j = i - k;
            while(j >=0 && h < v[j]){
                v[j + k] = move(v[j]);
                j -= k;
            }
            v[j + k] = move(h);
        }
        k = ... // volgend increment
    }
}
```

Stappen

1. Stel k gelijk aan de eerste waarde van een bepaalde reeks (zie analyse voor meer info over de reeks)
2. Overloop de tabel startend vanaf $i = k$.
3. Voer Insertion Sort uit met maar vervang de sprongen door k.
4. kies de volgende waarde van de reeks en herhaal zolang $k >= 0$.

Analyse

Insertion Sort heeft in het slechtste geval een kwadratische uitvoeringstijd. Dit komt omdat elementen één per één opgeschoven worden, of het wegnemen van één inversie per schuifoperatie. Shell Sort lost dit op door grotere sprongen te nemen en dus meerdere inversies per sprong weg te nemen. Shell Sort gaat er voor zorgen dat elementen die k posities van elkaar liggen gesorteerd worden. De reeksen die gebruikt worden bestaan meestal uit $\log n$ elementen. Belangrijke reeksen zijn:

- Reeks van Shell(1959):
$$\frac{n}{2}, \frac{n}{4}, \frac{n}{8}, ..., 1$$
- Reeks van Sedgewick(1986):
$$\sum_{k=0}^{\log n} \begin{cases} 9(2^k - 2^{k/2}) + 1 & \text{voor } k \text{ even} \\ 8 \cdot 2^k - 6 \cdot 2^{(k+1)/2}) + 1 & \text{voor } k \text{ oneven} \end{cases} (= 1, 5, 19, 41, 109, ...)$$
- Reeks van Tokuda(1992):
$$\sum_{k=0}^{\log n} \frac{1}{5}\left(9 \cdot \left(\frac{9}{4}\right)^{k-1} - 4\right) (= 1, 4, 9, 20, 46, 103, ...)$$
- Reeks van Ciura(2001): Experimenteel gevonden $1, 4, 10, 23, 57, 132, 301, 701, ...$

Belangrijk is dat de reeks van groot naar klein moet gaan, anders sorteer je eerst de tabel met Insertion Sort ($k = 1$) en dan hebben de grotere sprongen geen zin meer want de tabel is dan helemaal gesorteerd. De reeksen van Sedgewick,Tokuda en Ciura zijn gedefinieerd van klein naar groot maar moeten dus omgedraaid worden.
De performantie hangt af van de reeks. Voor de reeks van Shell kan het slechtste geval nog altijd $\Theta(n^2)$ zijn. Voor de reeks van Sedgewick kan aangetoond worden (via complexe analyse) dat de gemiddelde uitvoeringstijd slechts $O(n^{7/6})$.

Eigenschappen

- Sorteert ter plaatse
- Sorteert niet stabiel aangezien verschillende deelreeksen de volgorde van gelijke elementen kan wijzigen.

2.2 Rangschikken door eenvoudig selecteren

2.2.1 Selection Sort

```
void selection_sort(vector<T>& v){
    for(int i = v.size() - 1; i > 0; i--){
        int imax = i;
        for(int j = 0; j < i; j++){
            if(v[j] > v[imax]){
                imax = j;
            }
        }
        swap(v[i], v[imax]);
    }
}
```

Stappen

- Overloop de tabel in dalende volgorde startend vanaf $i = n - 1$. De index i is de index waar we k-de grootste getal willen zetten ($k = 1, 2, ..., n$).
- De variabele $imax$ zal bijhouden waar het k-de grootste getal zit.
- Overloop de tabel nu startend vanaf $j = 0$ en zoek het grootste element tot en met $j = i - 1$.
- Aangezien $imax$ de index bevat van het getal met de grootste waarde die op index i moet komen, voeren we de swap uit.

Analyse

Het principe hier is: zoek het grootste element en zet het op de laatste index, zoek het tweede grootste element en zet het op de voorlaatste index, enz. Het aantal sleutelvergelijkingen is hier onafhankelijk van de oorspronkelijke volgorde (elk element moet altijd met elk ander element vergeleken worden). Het aantal sleutelvergelijkingen is dus

$$\frac{n(n-1)}{2}$$

Het aantal verwisselingen is wel slechts $n - 1$. Dit algoritme wordt gedomineerd door de binnenste for lus en is dus $\Theta(n^2)$ voor zowel het slechtste, beste als gemiddelde geval (aangezien die altijd uitgevoerd wordt onafhankelijk van de oorspronkelijke staat van de tabel).

Eigenschappen

- Sorteert ter plaatse
- Sorteert niet stabiel

Hoofdstuk 3

Efficiënte methodes

3.1 Rangschikken door efficiënt selecteren

3.1.1 Heaps

Een heap is een complete binaire boom en heeft dus volgende eigenschappen:

- Elk niveau is volledig opgevuld. Indien een niveau niet volledig opgevuld is liggen alle knopen zoveel mogelijk links. Er kan dus hoogstens één knoop zijn met één kind.
- Elke knoop kan hoogstens 2 kinderen hebben. Een heap wordt vaak voorgesteld door een één dimensionale tabel. De wortel komt op index 0, zijn kinderen komen op index 1 en 2, de kinderen van 1 op plaatsen 3 en 4, enz. Dit kan vertaald worden naar eenvoudige formules met i de index van een willekeurige knoop.
  1. Het linkerkind van een knoop staat op positie $2i + 1$.
  2. Het rechterkind van een knoop staat op positie $2i + 2$.
  3. De ouder van een knoop staat op positie $(i - 1)/2$.
- De hoogte h van de heap is gelijk aan $\lfloor \log_2 n \rfloor$.
- De heap moet voldoen aan de *heapvoorwaarde*.
  - Bij een stijgende heap (maxheap) moet de sleutel van de ouder groter zijn dan de sleutels van zijn kinderen.
  - Bij een dalende heap (minheap) moet de sleutel van de ouder kleiner zijn dan de sleutels van zijn kinderen.

3.1.2 Bewerkingen op heaps

3.1.3 Constructie van een heap

3.1.4 Heap sort

Hoofdstuk 4

Speciale methodes

Hoofdstuk 5

De selectie-operatie

Hoofdstuk 6

Uitwendig rangschikken