Lektion 3 – Sortering

Magnus Nielsen, Filip Strömbäck

8 oktober 2019

Övningsuppgifter

1. Givet följande array: [9, 4, 5, 1, 2, 3, 7].
   Efter några iterationer av en sortering har vi följande array: [4, 5, 9, 1, 2, 3, 7].

   Vilken av algoritmerna har använts?
   a) Quick sort med elementet längst till höger som pivot.
   b) Bubble sort.
   c) Insertion sort.
   d) Selection sort.

2. Givet följande array: [9, 3, 8, 7, 1, 5, 6].
   Efter några iterationer av en sortering har vi följande array: [3, 1, 5, 6, 7, 8, 9].

   Vilken av algoritmerna har använts?
   a) Quick sort med elementet längst till höger som pivot.
   b) Bubble sort.
   c) Insertion sort.
   d) Selection sort.

3. Givet följande array: [3, 1, 6, 9, 5, 7, 2].
   Efter några iterationer av en sortering har vi följande array: [1, 2, 3, 5, 6, 9, 7].

   Vilken av algoritmerna har använts?
   a) Quick sort med elementet längst till höger som pivot.
   b) Bubble sort.
   c) Insertion sort.
   d) Selection sort.

4. Givet följande array: [7, 1, 8, 6, 5, 3, 9].
   Efter några iterationer av en sortering har vi följande array: [1, 3, 5, 6, 8, 7, 9].

   Vilken av algoritmerna har använts?
   a) Quick sort med elementet längst till höger som pivot.
   b) Bubble sort.
   c) Insertion sort.
   d) Selection sort.

5. Givet följande array: [3, 0, 9, 5, 8, 1, 6].
   Efter några iterationer av en sortering har vi följande array: [3, 0, 1, 5, 6, 8, 9].

   Vilken av algoritmerna har använts?
   a) Quick sort med elementet längst till höger som pivot.
   b) Bubble sort.
   c) Insertion sort.
   d) Selection sort.

6. Försök optimera följande klass:

```
#include <vector>

class SortedList {
public:
    void insert(int i) {
        contents.push_back(i);
        sort();
    }
    int getAndRemoveFirst() {
        int result = contents.first();
        contents.erase(contents.begin());
        return result;
    }
    int getAndRemoveLast() {
        int result = contents.last();
        contents.erase(result.end() - 1);
        return result;
    }
private:
    std::vector<int> contents;

    void sort() {
        for (int i = 0; i < contents.size(); i++) {
            bool swapped = false;
            for (int j = contents.size()-1; j > i; j--) {
                if (contents[j]<contents[j-1]) {
                    std::swap(contents[j], contents[j-1]);
                    swapped = true;
                }
            }
            if (!swapped) {
                break;
            }
        }
    }
}
```

7. Ni har startat en onlinebutik. I er webbserver har ni en (osorterad) länkad lista med alla era 10 miljoner artiklar. På ett styrelsemöte kommer ni fram till att ni vill se över era 1000 dyraste artiklar.

   Vi har två val:

   (a) Sök igenom listan 1000 gånger (tidskomplexitet $\mathcal{O}(n)$).

   (b) Konvertera den länkade listan till en array (tidskomplexitet $\mathcal{O}(n)$), sortera arrayen (tidskomplexitet $\mathcal{O}(n \log n)$) och sedan hämta ut de 1000 dyraste artiklarna?

   Vilket val gör ni? Vi kan avfärda tiden det tar att plocka fram ett data ur arrayen.

8. Efter att ha undersökt prestandan i förra uppgiften ytterligare lite har du insett att det som är dyrast är att traversera den länkade listan. Du vill alltså se till att bara göra det en gång. Du har också insett att minnet ibland tar slut när du skapar en array av alla element, så du vill helst undvika även det. Vi har alltså inte plats att lagra 10 miljoner element i en extra datastruktur, men det är möjligt att lagra en mindre mängd data (≈ 1000 element) i extra datastrukturer.

   Hur kan du hitta de $m = 1000$ största elementen från den länkade listan utan att traversera den mer än en gång?

   Hur kan du göra det så att det blir mer effektivt än den bästa lösningen i den förra uppgiften?

9. Du har blivit bjuden på restaurang av din arbetsgivare. Alla anställda har fått en budget på x kronor att beställa mat för. För att passa inom den utsatta tiden ska alla välja en huvudrätt och en efterrätt. Allt på menyn ser gott ut, så självklart vill du välja två rätter som har ett sammanlagt pris så nära under x som möjligt.

   För att hitta den bästa lösningen har du två listor av rätter: en med huvudrätter, och en med efterrätter. Varje element i listorna innehåller ett par med rättens namn och rättens pris. Antag att det finns n huvudrätter och n efterrätter.

   Beskriv hur du effektivt kan lösa problemet! Restaurangen har ett stort antal rätter att välja på, så din lösning måste vara effektiv för att programmet ska hinna hitta en lösning innan deadline för att beställa!

   (Vad har din lösning för tidskomplexitet om det finns h huvudrätter och e efterrätter?)

10. Du har en mängd av intervall, (a, b), $a < b$. Du kan tänka dig att varje intervall är ett linjesegment i en dimension som går mellan punkten a och b på tallinjen. Givet att du har n sådana intervall, beskriv hur du effektivt kan hitta om det finns intervall som överlappar varandra.

    Det vill säga: hitta om det finns åtminstone ett par av intervall, (a_1, b_1) och (a_2, b_2) där $a_1 \leq b_2$ och $a_2 \leq b_1$.

    (En svårare variant av problemet är att hitta alla intervall som överlappar varandra)