Kauno technologijos universitetas

Informatikos fakultetas, programų inžinerijos katedra

P170B400 Algoritmų sudarymas ir analizė

Bucket sort algoritmas ir jo analizė

Arnas Švenčionis

Projekto autorius

IFF-8/11

Akademinė grupė

Table of Contents

Algoritmo įvertinimas literatūroje

Bucket sort algoritmo teoriniai įvertinimai, kai naudota duomenų struktūra yra masyvas. Šaltinis – Vikipedija.

Worst-case performance	$O(n^2)$
Average performance	$O(n + \frac{n^2}{k} + k)$, where k is the number of buckets. $O(n)$, when $k \approx n$.

Algoritmo sudėtingumas, remiantis programos išeities tekstu.

```
private static void BucketSortArray(DataArray x)
{
    int n = x.Length;
    int numOfBuckets = 10;

    List<Objektas>[] buckets = new List<Objektas>[numOfBuckets];
    for (int i = 0; i < numOfBuckets; i++)
    {
        buckets[i] = new List<Objektas>();
    }

    for (int i = 0; i < n; i++)
    {
        int bucket = (int)(x[i].flo * numOfBuckets);
        buckets[bucket].Add(x[i]);
    }


    int a = 0;
    for (int i = 0; i < numOfBuckets; i++)
    {
        BubbleSort(buckets[i]);
        for (int j = 0; j < buckets[i].Count; j++)
        {
```

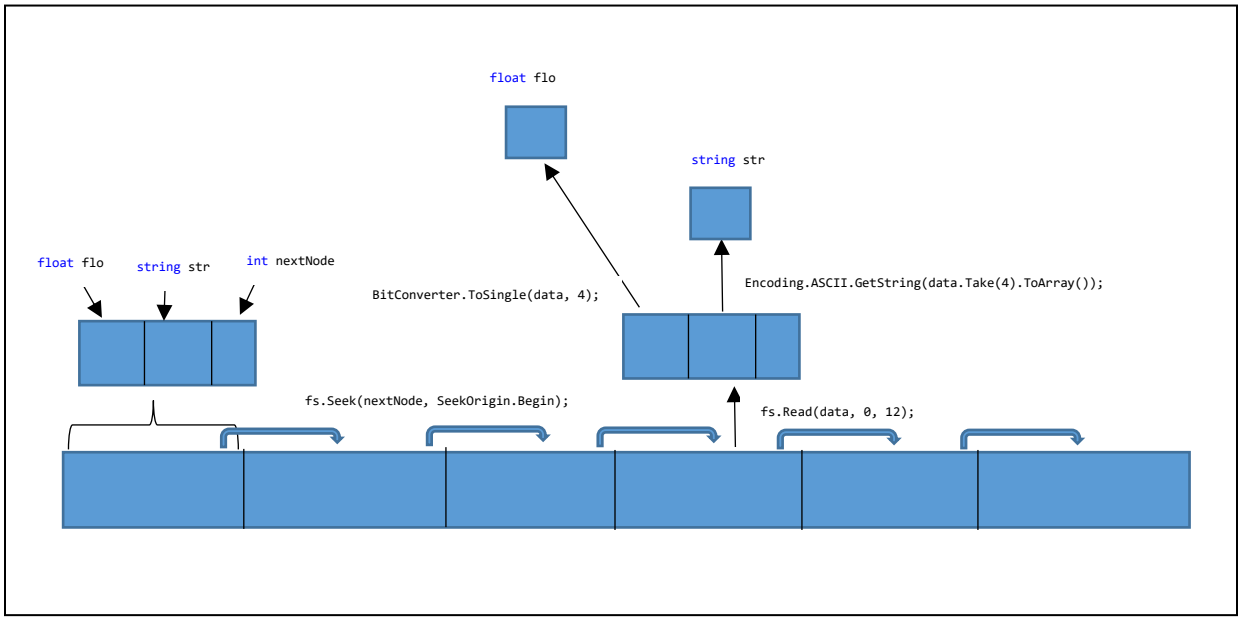
kaina	kiekis
C1	1
C1	1
C1	1
C2	numOfBuckets+1
C3	numOfBuckets
C4	n+1
C5	n
C5	n
C1	1
C4	n+1
C5	n
C5	n
C5	n
C5	n

```
            x.Change(a, buckets[i][j]);
            a++;
        }
    }
}
```

Į įvertinimą neįtrauksiu metodo BubbleSort sudėtingumo, nes BucketSort algoritmui galima naudoti betkokį kitą rikiavimo algoritmą kibirams surykiuoti. O su kiekvienu algoritmu gali skirtis ir BucketSort algoritmo sudėtingumas. Analizei sakysime, kad kibirų skaičius yra 10. x.Length = n(duomenų kiekis). Kadangi visų buckets masyvo kiekių suma yra lygi n, tai paskutinių dviejų ciklų ilgis iš tikrųjų yra lygus n.

T(n) = 3*C1 + (10+1)*C2 + 10*C3 +2*(n+1)*C4 + 6*n*C5 = 2*n*C4 + 2C4 + 6*n*C5 + 3*C1 + 11*C2 + 10*C3 = n(2*C4) + n(6*C5) + 2*C4 + 3*C1 + 11*C2 + 10*C3 = (konstantas prie duomenų kiekio galime sutraukti, o laisvom konstantom - atsikratyti) = C1*n + C2*n = n(C1 + C2) = (konstantas sutraukiame) = C*n = Θ(n);

Panaudotos duomenų struktūros realizacijos išorinėje atmintyje struktūrinę diagrama



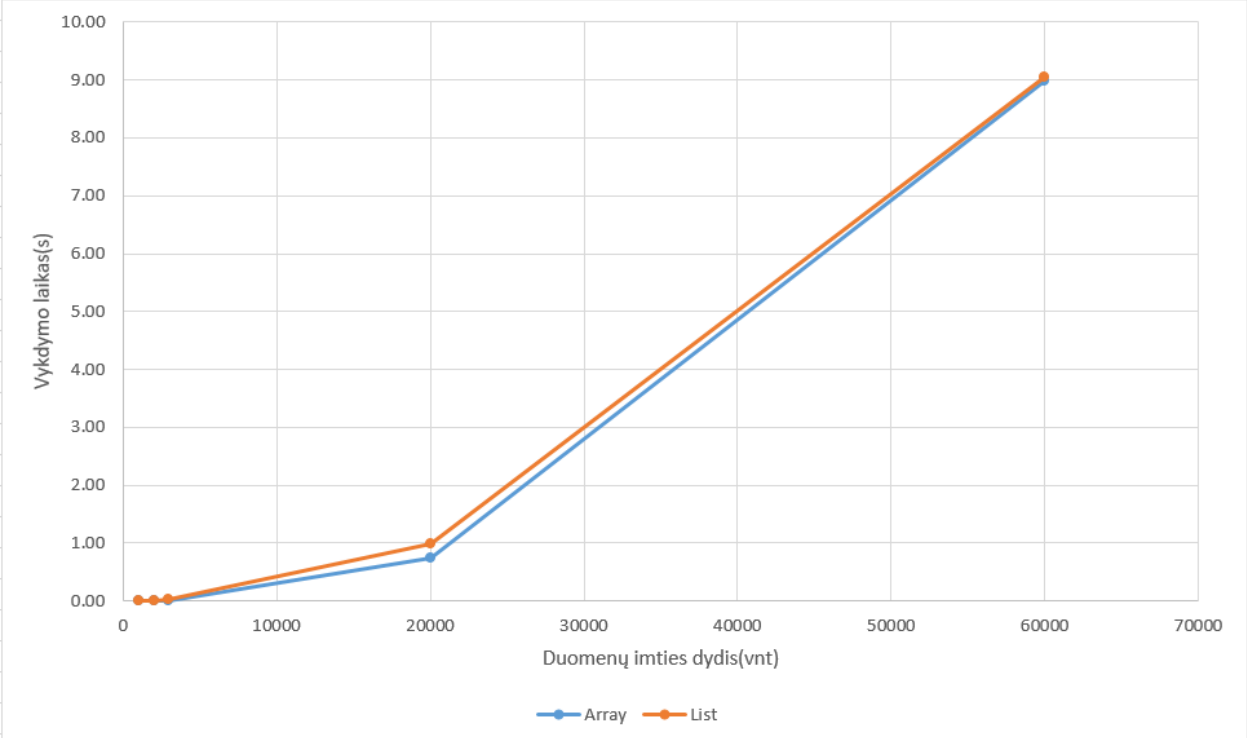

```
public override Objektas Next()
 {
     Byte[] data = new Byte[12];
     fs.Seek(nextNode, SeekOrigin.Begin);
     fs.Read(data, 0, 12);
     prevNode = currentNode;
     currentNode = nextNode;
```

```
            string str = Encoding.ASCII.GetString(data.Take(4).ToArray());
            float flo = BitConverter.ToSingle(data, 4);
            nextNode = BitConverter.ToInt32(data, 8);
            return new Objektas(str, flo);
        }
```

Atlikti eksperimentai, greitaveika

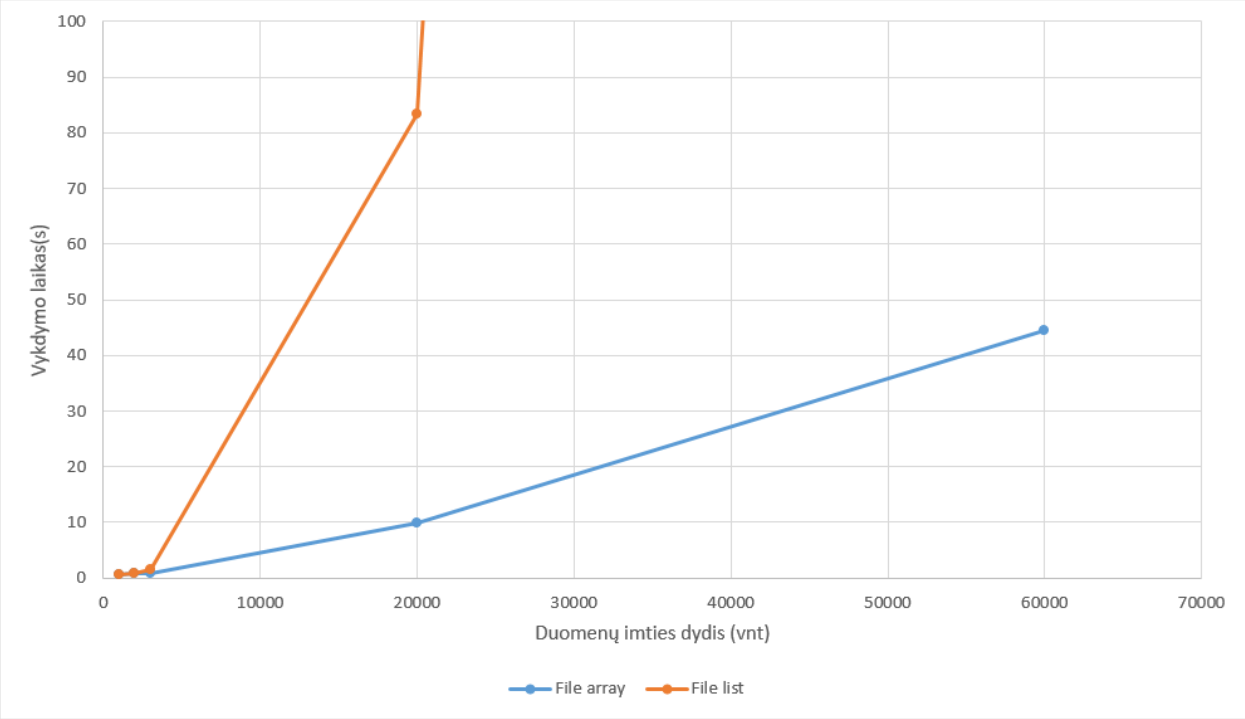
Operatyviosios atminties algoritmo analizė:

```
Array
1000 - 00:00:00.0026678
2000 - 00:00:00.0072528
3000 - 00:00:00.0164186
20000 - 00:00:00.7373715
60000 - 00:00:08.9720422
LinkedList
1000 - 00:00:00.0034246
2000 - 00:00:00.0095418
3000 - 00:00:00.0214299
20000 - 00:00:00.9801791
60000 - 00:00:09.0454277
Press any key to continue . . .
```



Išorinės atminties algoritmo analizė:

```
Array
100 - 00:00:00.4967801
200 - 00:00:00.7359592
300 - 00:00:00.9254607
2000 - 00:00:09.8553651
6000 - 00:00:44.4681253
LinkedList
100 - 00:00:00.5209103
200 - 00:00:00.8165526
300 - 00:00:01.5694866
2000 - 00:01:23.4318119
6000 - 00:31:59.8491978
Press any key to continue . . .
```



Išvados

BucketSort algoritmas nėra sudėtingas, veikia gana greitai. Tačiau nenaudojant kito rušiavimo algoritmo sudarytiems kibirams rūšiuoti, jis tik dalinai išrūšiuoja duomenis. Nuo kito rūšiavimo algoritmo priklauso ir galutinio algoritmo sudėtingumas. Algoritmas veikia gana greitai, nes duomenys prieš galutinį rūšiavimą yra arti vienas kito ir beveik išrūšiuoti. Operatyviojoje atmintyje algoritmas su masyvais dirba greičiau, nei su linkedlist. Masyvas greičiau veikia ir išorinėje atmintyje.

Laboratorinio darbo eiga buvo kėbli. Dirant su operatyviąja atmintimi sunkumų nekilo, tačiau prie antros darbo dalies užtrukau labai ilgai, laboratirinį darbą teko daryti kelis kartus iš naujo.