KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

# Algoritmų sudarymas ir analizė

Laboratorinis darbas nr. 1

Atliko:

IFF-8/2 gr. studentas

Nedas Šimoliūnas

2020 m. kovo 30 d.

Priėmė:

Doc. Dalius Makackas

KAUNAS 2020

TURINYS

Algoritmų sudarymas ir analizė 1

1. Merge Sort teorinis įvertinimas literatūroje 3

2. Algoritmo sudėtingumas, remiantis programos išeities tekstu 3

3. Linked List realizacija išorinėje atmintyje 5

4. Grafikai 6

5. Išvados 7

# Merge Sort teorinis įvertinimas literatūroje

Iš knygos „Introduction to Algorithms. Third Edition“:

***Merge sort*** takes time roughly equal to c2n lg n, where lg n stands for log2 n and c2 is another constant that also does not depend on n.

# Algoritmo sudėtingumas, remiantis programos išeities tekstu

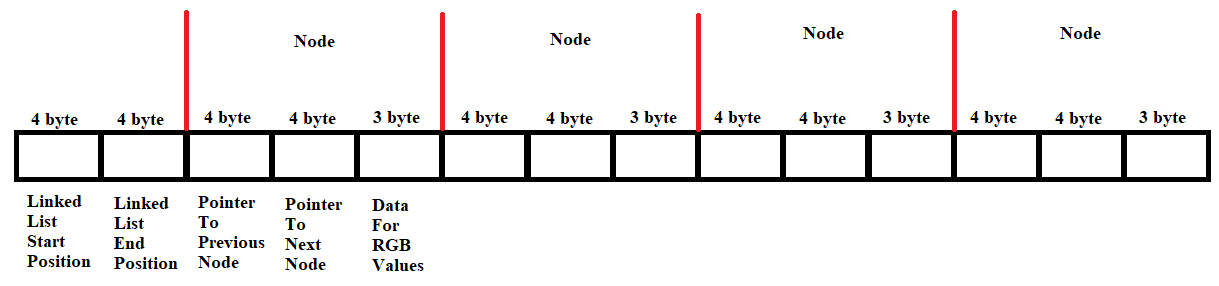
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | Kaina | Kartai |
| public static void MergeSort(DataArray<int> array) | | | | |  | T(array.Length) |  |
| { |  |  |  |  |  |  |  |
| if (array.Size() > 1) | | |  |  |  | c1 | 1 |
| { |  |  |  |  |  |  |  |
| DataArray<int> left = LeftHalf(array); | | | | |  | array.Length | 1 |
| DataArray<int> right = RightHalf(array); | | | | |  | array.Length | 1 |
|  |  |  |  |  |  |  |  |
| MergeSort(left); | | |  |  |  | T(array.Length/2) | 1 |
| MergeSort(right); | | |  |  |  | T(array.Length/2) | 1 |
|  |  |  |  |  |  |  |  |
| Merge(array, left, right); | | | |  |  | array.Length | 1 |
| actions += 5; | | |  |  |  |  |  |
| } |  |  |  |  |  |  |  |
| actions++; | |  |  |  |  |  |  |
| } |  |  |  |  |  |  |  |
|  | n = array.Length | |  |  |  |  |  |
|  | T(n) = 2 \* T(n/2) + 3\*n + c1 | | |  |  |  |  |
|  | **T(n) = ϴ (nlog[2]n)** | | |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | Kaina | Kiekis |
| public static DataArray<int> LeftHalf(DataArray<int> array) | | | | | | T(array.Length) |  |
| { |  |  |  |  |  |  |  |
| int size1 = array.Size() / 2; | | | |  |  | c1 | 1 |
| DataArray<int> left = new DataArray<int>(size1); | | | | | | c2 | 1 |
| for (int i = 0; i < size1; i++) | | | |  |  | c3 | size1 + 1 |
| { |  |  |  |  |  |  |  |
| left.Replace(i, array[i]); | | |  |  |  | c4 | size1 |
| } |  |  |  |  |  |  |  |
| actions += (2 + size1); | | |  |  |  |  |  |
| return left; | |  |  |  |  | c5 |  |
| } |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | n = array.Length | |  |  |  |  |  |
|  | T(n) = c1 + c2 + c3 \* (n/2 + 1) + c4 \* (n/2) + c5 | | | | |  |  |
|  | T(n) = An + B | |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | Kaina | Kiekis |
| public static DataArray<int> RightHalf(DataArray<int> array) | | | | | | T(array.Length) |  |
| { |  |  |  |  |  |  |  |
| int size1 = array.Size() / 2; | | | |  |  | c1 | 1 |
| int size2 = array.Size() - size1; | | | |  |  | c2 | 1 |
| DataArray<int> right = new DataArray<int>(size2); | | | | | | c3 | 1 |
| for (int i = 0; i < size2; i++) | | | |  |  | c4 | size2 + 1 |
| { |  |  |  |  |  |  |  |
| right.Replace(i, array[i + size1]); | | | |  |  | c5 | size2 |
| } |  |  |  |  |  |  |  |
| actions += (3 + size2); | | |  |  |  |  |  |
| return right; | |  |  |  |  | c6 | 1 |
| } |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | n = array.Length | |  |  |  |  |  |
|  | T(n) = c1 + c2 + c3 + c4 \* (size2 + 1) + c5 \* size2 + c6 | | | | |  |  |
|  | T(n) = An + B | |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | Kaina | Kiekis |
| public static void Merge(DataArray<int> result, | | | | |  | T(result.Length) |  |
| DataArray<int> left, DataArray<int> right) | | | | | |  |  |
| { |  |  |  |  |  |  |  |
| int i1 = 0; | |  |  |  |  | c1 | 1 |
| int i2 = 0; | |  |  |  |  | c2 | 1 |
| for (int i = 0; i < result.Size(); i++) | | | |  |  | c3 | result.Length + 1 |
| { |  |  |  |  |  |  |  |
| if (i2 >= right.Size() || | | |  |  |  | c4 | result.Length |
| (i1 < left.Size() && | | |  |  |  |
| left[i1] <= right[i2])) | | | |  |  |
| { |  |  |  |  |  |  |
| result.Replace(i, left[i1++]); | | | |  |  | c5 |
| } |  |  |  |  |  |  |
| else | |  |  |  |  | c6 |
| { |  |  |  |  |  |  |
| result.Replace(i, right[i2++]); | | | |  |  | c7 |
| } |  |  |  |  |  |  |  |
| } |  |  |  |  |  |  |  |
| actions += (2 + 2 \* result.Size()); | | | |  |  |  |  |
| } |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | n = result.Length | |  |  |  |  |  |
|  | T(n) = c1 + c2 + c3 \* (result.Length + 1) + C \* result.Length | | | | | |  |
|  | T(n) = An + B | |  |  |  |  |  |

# Linked List realizacija išorinėje atmintyje



Pirmi 4 baitai – rodyklė į sąrašo pradžią, kiti 4 – į pabaigą. Po to kiekvienas sąrašo mazgas sudarytas iš 11 baitų – 4 rodo į prieš tai esantį mazgą, kiti 4 į sekantį mazgą ir likę 3 yra pikselio RGB reikšmės. Kiekvieno mazgo rodyklės reikšmės prasideda iš eilės nuo 0 ir tikslios mazgų vietos yra paskaičiuojamos metoduose (8 + 11 \* node). Kiekviena nauja reikšmė yra įrašoma į sąrašo galą.

Kelių sąsajos metodų pavyzdžiai:

public int GetPrevious(string file, int node)

{

using (Reader = new BinaryReader(File.Open(file, FileMode.Open, FileAccess.Read)))

{

Reader.BaseStream.Seek(8 + 11 \* node, SeekOrigin.Begin);

int prev = Reader.ReadInt32();

Reader.Close();

return prev;

}

}

public int GetNext(string file, int node)

{

using(Reader = new BinaryReader(File.Open(file, FileMode.Open, FileAccess.Read)))

{

Reader.BaseStream.Seek(8 + 11 \* node + 4, SeekOrigin.Begin);

int next = Reader.ReadInt32();

Reader.Close();

return next;

}

}

public int GetValueByNode(string file, int node)

{

using(Reader = new BinaryReader(File.Open(file, FileMode.Open, FileAccess.Read)))

{

Reader.BaseStream.Seek(8 + 11 \* node + 8, SeekOrigin.Begin);

byte[] values = Reader.ReadBytes(3);

int value = (((values[2] << 8) + values[1]) << 8) + values[0];

Reader.Close();

return value;

}

}

# Grafikai

Parodomos laiko milisekundėmis priklausomybė nuo rikiuojamo pikselių kiekio.

# Išvados

Laboratorinio darbo užduotis buvo realizuoti Merge Sort algoritmą vidinėje ir išorinėje atmintyje. Parašyto kodo sudėtingumas atitiko teorinį, laiko priklausomybė nuo įvesties kiekio panaši į teorinį sudėtingumą. Šiame laboratoriniame darbe išmokau dirbti su nuotraukomis, manipuliuoti jų pikseliais, taip pat, išmokau masyvo ir susietojo sąrašo idėją įgyvendinti išorinėje atmintyje.