

**Kauno technologijos universitetas**

Informatikos fakultetas

Inžinerinio projekto ataskaita

P170B400 Algoritmų sudarymo ir analizės antras laboratorinis darbas

**Autorius**

Gustas Klevinskas

**Akademinė grupė**

IFF-8/7

Kaunas, 2020

Turinys

[Užduotys 3](#_Toc42073253)

[1. Uždavinys 4](#_Toc42073254)

[Programinis kodas 4](#_Toc42073255)

[Teorinio sudėtingumo skaičiavimas 4](#_Toc42073256)

[Rezultatai 5](#_Toc42073257)

[2. Uždavinys 7](#_Toc42073258)

[Rekurentinė lygtis 7](#_Toc42073259)

[Sprendimo iliustravimas 7](#_Toc42073260)

[Kodas ir sudėtingumo skaičiavimas 7](#_Toc42073261)

[Rezultatai 8](#_Toc42073262)

[3. Uždavinys 9](#_Toc42073263)

[Programinis kodas 9](#_Toc42073264)

[Rezultatai 9](#_Toc42073265)

[Išlygiagretinimo koeficientas 10](#_Toc42073266)

[Priedas 11](#_Toc42073267)

[Visas programinis kodas 11](#_Toc42073268)

# Užduotys

1. Duota:

1. Duota simbolių seka . Raskite mažiausią simbolių skaičių, kurį reikia įterpti į simbolių seką, kad ji taptų simetrinė.

# 1. Uždavinys

## Programinis kodas

static int F(int W) {

if (W == 0)

return 0;

int max = 0;

for (int i = 0; i < n; i++)

if (w[i] <= W)

max = Math.Max(max, F(W - w[i]) + v[i]);

return max;

}

static int FDyn(int W) {

if (W == 0)

return 0;

int max = 0;

for (int i = 0; i < n; i++) {

if (w[i] <= W) {

if (FDynValues[W - w[i]] == null)

FDynValues[W - w[i]] = FDyn(W - w[i]);

max = Math.Max(max, (int)FDynValues[W - w[i]] + v[i]);

}

}

return max;

}

## Teorinio sudėtingumo skaičiavimas

|  |  |  |
| --- | --- | --- |
| Kodas | Kaina | Kiekis |
| static int F(int W) {  if (W == 0)  return 0;  int max = 0;  for (int i = 0; i < n; i++)  if (w[i] <= W)  max = Math.Max(max, F(W - w[i]) + v[i]);  return max;  } | c  c  c  c  c  F(W-w[i])  c | 1  1  1  n  n - 1  n – 1  1 |

|  |  |  |
| --- | --- | --- |
| Kodas | Kaina | Kiekis |
| static int FDyn(int W) {  if (W == 0)  return 0;  int max = 0;  for (int i = 0; i < n; i++) {  if (w[i] <= W) {  if (FDynValues[W - w[i]] == null)  FDynValues[W - w[i]] = FDyn(W - w[i]);  max = Math.Max(max, (int)FDynValues[W - w[i]] + v[i]);  }  }  return max;  } | c  c  c  c  c  c  FDyn(W-w[i])  c  c | 1  1  1  n  n - 1  n – 1  n - 1  n - 1  1 |

## Rezultatai

|  |  |  |
| --- | --- | --- |
| n | F laikai, Ticks | FDyn laikai, Ticks |
| 1000 | 53439 | 226 |
| 2000 | 594551 | 1122 |
| 3000 | 2815666 | 591 |
| 4000 | 4812442 | 1004 |
| 5000 | 7067064 | 1156 |
| 6000 | 12730828 | 1638 |
| 7000 | 18674513 | 1446 |
| 8000 | 27590198 | 1925 |
| 9000 | 40388497 | 1725 |
| 10000 | 57010340 | 2494 |
| 11000 | 69227429 | 2528 |
| 12000 | 91781722 | 3815 |
| 13000 | 115999168 | 2872 |
| 14000 | 143490484 | 3049 |
| 15000 | 191600995 | 3066 |

Iš grafikų aiškiai matosi, kad F(W) algoritmo sudėtingumas yra laipsninis, o F(W) dinaminio spendimo sudėtingumas yra tiesinis. Taip pat verta paminėti, kad rekursyvus algoritmas užtrunka žymiai ilgiau nei dinaminis. Laikas matuotas „ticks“, nes milisekundė buvo per didelis laiko tarpas išmatuoti dinaminio algoritmo vykdymo laiką.

# 2. Uždavinys

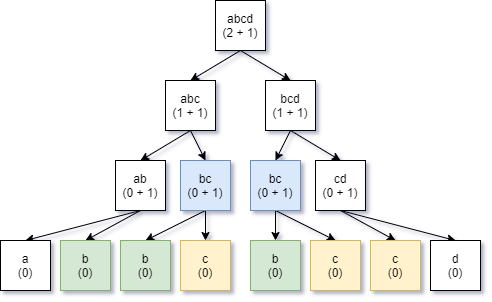
## Rekurentinė lygtis

– funkcija, grąžinanti minimalų kiekį pridėjimų, reikalingų, kad simbolių seka taptų simetrinė.

ir - simbolių sekos indeksai.

– simbolių seka.

## Sprendimo iliustravimas



Kiekvienas kvadratėlis iliustruoja analizuojamą seką. Skliausteliuose parašyta, kokią reikšmę grąžintų funkcija (minimalus pridėjimų kiekis, kad seka taptų simetrine). Einama gilyn, kol pasiekiama viena raidė – tuomet algoritmas grąžina 0. Aukštesniuose kvadratėliuose randamas minimumas dviejų apatinių kvadratėlių ir pridedamas vienetas. Taip rezultatai grąžinami aukštyn, kol randamas pradinės sekos atsakymas.

## Kodas ir sudėtingumo skaičiavimas

|  |  |  |
| --- | --- | --- |
| Kodas | Kaina | Kiekis |
| static int MinInsertsDyn(string s) {  int n = s.Length;  int[,] table = new int[n, n];  for (int len = 1; len < n; len++) {  for (int l = 0, h = len; h < n; l++, h++) {  if (s[l] == s[h]) {  table[l, h] = table[l + 1, h - 1];  continue;  }  table[l, h] = Math.Min(table[l, h - 1], table[l + 1, h]) + 1;  }  }  return table[0, n - 1];  } | c  c  c  c  c  c  c  c  c | 1  1  n - 1  (n – 2)2  (n – 2)2  (n – 2)2  (n – 2)2  (n – 2)2  1 |

Kadangi yra greičiausiai augantis dėmuo, galima teigti, kad sudėtingumas yra .

## Rezultatai

Rezultatų lentelė nepridėta, nes joje yra 600 eilučių.

Matome, kad algoritmas yra laipsninis. Galime teigti, kad laipsnis yra kvadratas, nes sukuriama lentelė, kurios dydis yra ( – simbolių sekos ilgis) ir algoritmas turi užpildyti jos pusę, kad gautų galutinį atsakymą. Taip pat yra du ciklai, kurie iteruoja per sekos elementus.

# 3. Uždavinys

## Programinis kodas

static int FParallel(int W) {

if (W == 0)

return 0;

int max = 0;

Parallel.For(0, n, i => {

if (w[i] <= W)

Interlocked.Exchange(ref max, Math.Max(max, F(W - w[i]) + v[i]));

});

return max;

}

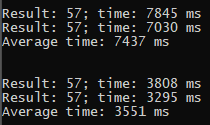
## Rezultatai

|  |  |  |
| --- | --- | --- |
| n | F laikai, Ticks | FParallel laikai, Ticks |
| 1000 | 53439 | 2600370 |
| 2000 | 594551 | 272789 |
| 3000 | 2815666 | 768532 |
| 4000 | 4812442 | 1641206 |
| 5000 | 7067064 | 2875648 |
| 6000 | 12730828 | 5116659 |
| 7000 | 18674513 | 6787845 |
| 8000 | 27590198 | 10958854 |
| 9000 | 40388497 | 15315013 |
| 10000 | 57010340 | 20653577 |
| 11000 | 69227429 | 25794891 |
| 12000 | 91781722 | 33877299 |
| 13000 | 115999168 | 42678590 |
| 14000 | 143490484 | 52797197 |
| 15000 | 191600995 | 65029631 |

Matome, kad paralelizuoto algoritmo vykdymo trukmė yra mažesnė palyginus su rekursyviu algoritmu. Pradžioje paralelizuotas algoritmas tikriausiai veikė truputį lėčiau, nes paralelizuojant atsiranda papildomas „overhead“ ir jis labiausiai pasireiškia lyginant algoritmus su maža duomenų imtimi.

## Išlygiagretinimo koeficientas

Savo programą paleidau su duomenų imtimi . Gauti rezultatai:



Viršuje atspausdintas rekursinio algoritmo vykdymo laikai ir apskaičiuotas dviejų laikų vidurkis, antroje eilutėje – paralelizuoto algoritmo.

Išvada – programą paralelizavus ji pagreitėjo du kartus.

# Priedas

## Visas programinis kodas

using System;

using System.Diagnostics;

using System.Threading;

using System.Threading.Tasks;

namespace L2 {

class Program {

#region

static int nStep;

static int nEnd;

static int repeats;

static int wBound;

static int n;

static int W;

static int[] w;

static int[] v;

static int?[] FDynValues;

static string s;

#endregion

static void Main() {

W = 3;

wBound = 20;

n = 1000;

nStep = 1000;

nEnd = 15000;

repeats = 2;

Utils.Init(nEnd, W, wBound, out w, out v, out FDynValues);

Warmup();

//TestF();

//TestFParallel();

//TestMinInserts();

for (; n <= nEnd; n += nStep) {

Console.WriteLine("n = {0}", n);

TestF();

//TestFDyn();

//TestFParallel();

}

}

static void Warmup() {

int previousN = n;

int previousRepeats = repeats;

n = 20;

repeats = 3;

Console.WriteLine("--- Warmup ---");

Console.WriteLine("Total: {0} ticks\n\n", MeasureTime(FDyn));

n = previousN;

repeats = previousRepeats;

FDynValues = new int?[FDynValues.Length];

}

static void TestF() {

long time = MeasureTime(F);

Utils.SaveResults(Utils.FResultsFile, n, time);

Console.WriteLine("Average time: {0} ms\n\n", time);

}

static void TestFDyn() {

long time = MeasureTime(FDyn);

Utils.SaveResults(Utils.FDynResultsFile, n, time);

Console.WriteLine("Average time: {0} ticks\n\n", time);

}

static void TestFParallel() {

long time = MeasureTime(FParallel);

Utils.SaveResults(Utils.FParallelResultsFile, n, time);

Console.WriteLine("Average time: {0} ms\n\n", time);

}

static void TestMinInserts() {

Random random = new Random();

Stopwatch stopwatch = new Stopwatch();

string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

string line = "";

for (int i = 0; i < 600; i++) {

line += chars[random.Next(chars.Length)];

stopwatch.Start();

Console.WriteLine("Result = {0}", MinInsertsDyn(line));

stopwatch.Stop();

Utils.SaveResults(Utils.MinInsertsFile, i, stopwatch.ElapsedTicks);

}

}

static long MeasureTime(Func<int, int> function) {

Stopwatch stopwatch = new Stopwatch();

long time = 0;

for (int i = 0; i < repeats; i++) {

stopwatch.Reset();

Console.Write("Running...");

stopwatch.Start();

int result = function(W);

stopwatch.Stop();

time += stopwatch.ElapsedMilliseconds;

Console.WriteLine("\rResult: {0}; time: {1} ms", result, stopwatch.ElapsedMilliseconds);

FDynValues = new int?[FDynValues.Length];

}

return time / repeats;

}

static int F(int W) {

if (W == 0)

return 0;

int max = 0;

for (int i = 0; i < n; i++)

if (w[i] <= W)

max = Math.Max(max, F(W - w[i]) + v[i]);

return max;

}

static int FDyn(int W) {

if (W == 0)

return 0;

int max = 0;

for (int i = 0; i < n; i++) {

if (w[i] <= W) {

if (FDynValues[W - w[i]] == null)

FDynValues[W - w[i]] = FDyn(W - w[i]);

max = Math.Max(max, (int)FDynValues[W - w[i]] + v[i]);

}

}

return max;

}

static int FParallel(int W) {

if (W == 0)

return 0;

int max = 0;

Parallel.For(0, n, i => {

if (w[i] <= W)

Interlocked.Exchange(ref max, Math.Max(max, F(W - w[i]) + v[i]));

});

return max;

}

static int MinInsertsDyn(string s) {

int n = s.Length;

int[,] table = new int[n, n];

for (int len = 1; len < n; len++) {

for (int l = 0, h = len; h < n; l++, h++) {

if (s[l] == s[h]) {

table[l, h] = table[l + 1, h - 1];

continue;

}

table[l, h] = Math.Min(table[l, h - 1], table[l + 1, h]) + 1;

}

}

return table[0, n - 1];

}

}

}

using System;

using System.IO;

using System.Text;

namespace L2 {

public static class Utils {

private const string arraysFile = "Arrays.txt";

public const string FResultsFile = "FResults.csv";

public const string FDynResultsFile = "FDynResults.csv";

public const string FParallelResultsFile = "FParallelResults.csv";

public const string MinInsertsFile = "MinInsertResults.csv";

public static void Init(int nEnd, int W, int randomBound, out int[] w, out int[] v, out int?[] FDynValues) {

if (File.Exists(FResultsFile))

File.Delete(FResultsFile);

if (File.Exists(FDynResultsFile))

File.Delete(FDynResultsFile);

if (File.Exists(FParallelResultsFile))

File.Delete(FParallelResultsFile);

if (File.Exists(MinInsertsFile))

File.Delete(MinInsertsFile);

if (IsFileValid(nEnd)) {

Read(W, out w, out v, out FDynValues);

} else {

Generate(nEnd, W, randomBound, out w, out v, out FDynValues);

Write(nEnd, w, v);

}

}

public static void SaveResults(string filename, int n, long time) {

StreamWriter sw = File.AppendText(filename);

sw.WriteLine("{0},{1}", n, time);

sw.Close();

}

private static void Read(int W, out int[] w, out int[] v, out int?[] FDynValues) {

StreamReader reader = new StreamReader(arraysFile);

int n = int.Parse(reader.ReadLine());

w = new int[n];

v = new int[n];

for (int i = 0; i < n; i++)

w[i] = int.Parse(reader.ReadLine());

for (int i = 0; i < n; i++)

v[i] = int.Parse(reader.ReadLine());

FDynValues = InitializeFDynValues(W, w);

reader.Close();

}

private static void Write(int n, int[] w, int[] v) {

StringBuilder sb = new StringBuilder();

sb.AppendLine(n.ToString());

foreach (int i in w)

sb.AppendLine(i.ToString());

foreach (int i in v)

sb.AppendLine(i.ToString());

File.WriteAllText(arraysFile, sb.ToString());

}

private static void Generate(int n, int W, int randomBound, out int[] w, out int[] v, out int?[] FDynValues) {

w = new int[n];

v = new int[n];

Random random = new Random();

for (int i = 0; i < n; i++) {

w[i] = random.Next(1, randomBound);

v[i] = random.Next(1, randomBound);

}

FDynValues = InitializeFDynValues(W, w);

}

private static int?[] InitializeFDynValues(int W, int[] w) {

int minW = int.MaxValue;

bool found = false;

foreach (int i in w) {

if (i <= W && minW > i) {

minW = i;

found = true;

}

}

if (found)

return new int?[W - minW + 1];

else

return null;

}

private static bool IsFileValid(int maxN) {

if (!File.Exists(arraysFile))

return false;

StreamReader sr = new StreamReader(arraysFile);

bool result = sr.ReadLine() == maxN.ToString();

sr.Close();

return result;

}

}

}