Autorius: **Mantas Damijonaitis IFF-5/4.** Priemė: **dėst. Vytautas Pilkauskas**

Individualus **Algoritmų sudarymo ir analizės** laboratorinis darbas

1 užduotis. Panaudojus pirmame inžineriniame projekte sudarytą paieškos (operatyvinėje atmintyje) algoritmą, realizuoti n elementų paiešką panaudojant lygiagretų programavimą. Eksperimentiškai palyginti **n** elementų paieškos vykdymo laikus, kai nenaudojamas lygiagretus programavimas ir naudojamas lygiagretus programavimas.
Mano atveju užduotis – paieška tiesinėje maišos lentelėje su tiesioginiu adresavimu.

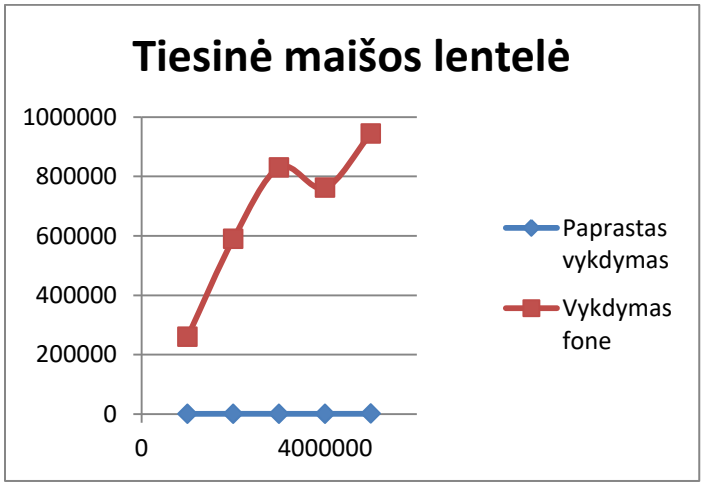
Paieška tiesiogiai:

```
0 references
public void SequentialSearch()
{
    LinearHashTable<int, string> hashTable = new LinearHashTable<int, string>();
    for (int i = 0; i < DataAmount; i++)
    {
        string randomString = Guid.NewGuid().ToString("n").Substring(0, DataLength);
        hashTable.Add(i, randomString);
    }
    int count = 0;
    Stopwatch stopWatch = new Stopwatch();
    stopWatch.Start();
    for (int i = 0; i < DataLength; i++)
    {
        string value = string.Empty;
        if (hashTable[i] != null)
        {
            count++;
        }
    }
    stopWatch.Stop();
}
```

Paieška išlygiagretinus:

```
1 reference
public void ParallelSearch()
{
    LinearHashTable<int, string> hashTable = new LinearHashTable<int, string>();
    for (int i = 0; i < DataAmount; i++)
    {
        string randomString = Guid.NewGuid().ToString("n").Substring(0, DataLength);
        hashTable.Add(i, randomString);
    }
    int cpuCount = 8;
    Task<int>[] tasks = new Task<int>[cpuCount];
    Stopwatch stopWatch = new Stopwatch();
    stopWatch.Start();
    for (int j = 0; j < cpuCount; j++)
    {
        tasks[j] = Task<int>.Factory.StartNew(
            (object p) =>
            {
                int count = 0;
                for (int i = (int)p; i < DataAmount; i += cpuCount)
                {
                    string val = string.Empty;
                    if (hashTable[i] != null)
                    {
                        count++;
                    }
                }
                return count;
            }, j);
    }
    int total = 0;
    for (int i = 0; i < cpuCount; i++)
    {
        total += tasks[i].Result;
    }
    stopWatch.Stop();
}
```

Bandymų metu gauti rezultatai:



Išvada: išlygiagretinus programą, ji pradėjo veikti lėčiau todėl, nes tiesinė maišos realizaciją reikšmes gražina itin greitai, ir perkeliant paiešką į atskiras gijas naudojamas nemažas papildomas laikas.

2. Užduotis: Panaudojus antrame inžineriniame projekte duotą rekurentinę formulę realizuoti jai algoritmą tiesiogiai panaudojant rekursiją bei lygiagretų programavimą. Eksperimentiškai palyginti vykdymo laikus, kai nenaudojamas lygiagretus programavimas ir naudojamas lygiagretus programavimas.

Mano atveju rekurentinė formulė yra: $F_w = max_{i:w_i \leq W}(F(W - w_i) + v_i), F(0) = 0$

Lygties sprendimas panaudojant rekursiją:

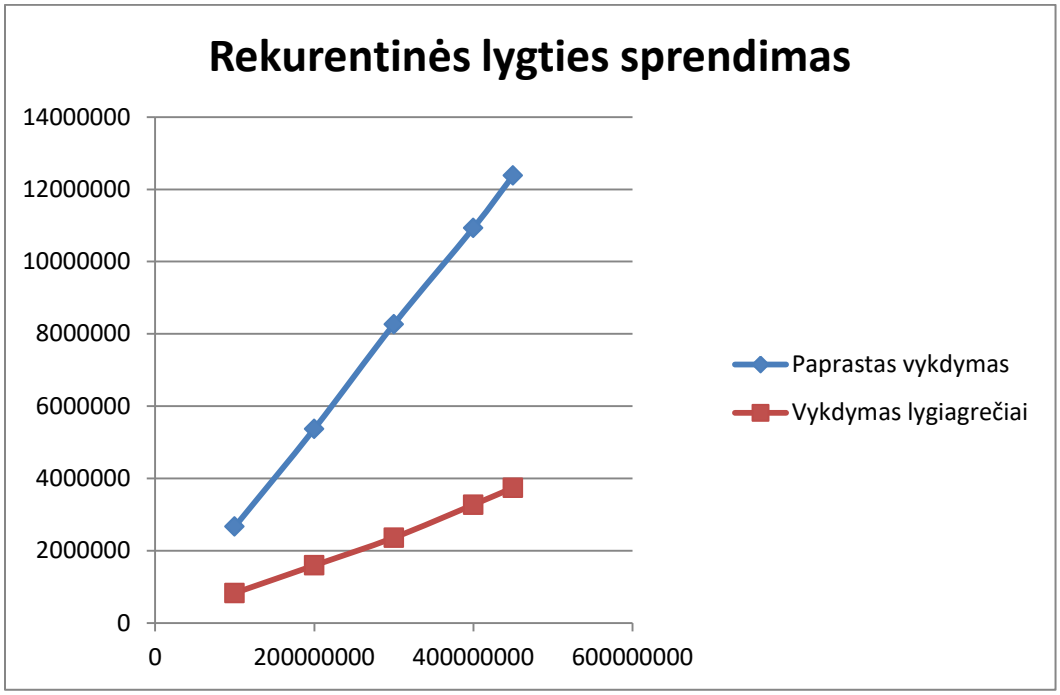
```
2 references
public int F(int W, int[] w, int[] v, int max = 0)
{
    if (W <= 0)
    {
        return 0;
    }
    for (int i = 0; i < n; i++)
    {
        int tarpinis = F(W - w[i], w, v, max);
        if (w[i] <= W && tarpinis + v[i] > max)
        {
            max = tarpinis + v[i];
        }
    }
    return max;
}
```

Lygties sprendimas išlygiagretinant:

```
public int ParallelF(int W, int max = 0)
space AlgorithmsThird
    int result = 0;
    if (W > 0)
    {
        int countCPU = 4;
        Task[] tasks = new Task[countCPU];
        for (int j = 0; j < countCPU; j++)
        {
            tasks[j] = Task.Factory.StartNew(
                (Object p) =>
                {
                    var data = p as CustomData;
                    if (data == null)
                    {
                        return;
                    }
                    data.TResult = FPar(W, data.TNum, countCPU);
                }, new CustomData() { TNum = j });
        }
        Task.WaitAll(tasks);
        for (int i = 0; i < countCPU; i++)
        {
            if ((tasks[i].AsyncState as CustomData).TResult > result)
            {
                result = (tasks[i].AsyncState as CustomData).TResult;
            }
        }
        return result;
    } else
    {
        return 0;
    }
}
```

```
2 references
public int FPar(int W, int start, int countCPU, int max = 0)
{
    if (W <= 0)
    {
        return 0;
    }
    for (int i = start; i < n; i += countCPU)
    {
        int tarpinis = FPar(W - w[i], start, countCPU);
        if (w[i] <= W && tarpinis + v[i] >= max)
        {
            max = tarpinis + v[i];
        }
    }
    return max;
}
```

Bandymų metu gauti rezultatai:



Išvada: laboratorinio darbo metu išmokau išlygiagretinimo pagrindų, bei dar labiau išsianalizavau rekurentinių lygčių sprendimo metodiką.