ԲՆԱԿԱՆ ԳԻՏՈՒԹՅՈՒՆՆԵՐԻ ՖԱԿՈՒԼՏԵՏ

ԲԱԼԱՍԱՆՅԱՆ ՎԼԱԴԻՄԻՐ ԼԵՎՈՆԻ

ՄԵԼՔՈՒՄՅԱՆ ՎՈԼՈԴՅԱ ԳՐԻԳՈՐԻԻ

ՀԵՏԱԶՈՏԱԿԱՆ ԱՇԽԱՏԱՆՔ

ՄԱՍՍԱՅԱԿԱՆ ՍՊԱՍԱՐԿՄԱՆ ՏԵՍՈՒԹՅՈՒՆ

ԳՈՐԾՈՒՅԹՆԵՐԻ ՀԵՏԱԶՈՏՈՒՄ

ԻՋԵՎԱՆ 2021

ԲՈՎԱՆԴԱԿՈՒԹՅՈՒՆ

ՆԵՐԱԾՈՒԹՅՈՒՆ․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․ԷՋ 3

ՄԱՍՍԱՅԱԿԱՆ ՍՊԱՍԱՐԿՄԱՆ ՏԵՍՈՒԹՅՈՒՆ․․․․․․․․․․․․․․․․․․․․․․․․․․․․ԷՋ 4

ՀԱՎԵԼՎԱԾՆԵՐ․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․․ԷՋ 7

ՆԵՐԱԾՈՒԹՅՈԻՆ

Ֆինանսների, էկոնոմիկայի, արտադրության և այլ բազմատիվ ոլորտներում մեծ դեր են խաղում հատուկ տեսակի համակարգերը,որոնք իրականացնում են մեծ թվով միանման գործողություններ։Այսպիսի համակարգերին անվանում են՝ մասսայական սպասարկման համակարգեր (ՄՍՀ)։Դրա օրինակ են հանդիսանում՝ վարսավիրանոցները, շուկաները, բանկերը, գործարանները և այլն։ Մասսայական սպասարկման տեսությունը մաթեմատիկայի բաժին է, որը ուսումնասիրում է պատահական բնույթ ունեցող պահանջների զանգվածային հոսքն սպասարկող համակարգեր։ Սպասարկման համակարգ է կոչվում ամեն մի ձեռնարկություն, ուր ժամանակի պատահական պահերի ստացվում են որոշակի ծառայություն կատարելու պատվերներ,ընդ որում, պատահական կարող է լինել նաև պատվերի կատարման տևողությունը։

Մասսայական Սպասարկման Տեսություն

Ամեն մի ՄՍՀ իր կառուցվածքի մեջ պարունակում է որոշակի քանակությամբ միավորներ (սարքեր, տեխնիկա) որոնց անվանում են սպասարկման խողովակներ։

ՄՍՀ-ն նախատեսված է որոշակի հարցումների հոսքի սպասարկման համար, որոնք մտնւմ են համակարգ մեծ մասամբ ոչ հաստատուն, այլ ժամանակի պատահական պահերի։ Հարցումների սպասարկումը նույնպես ընդհանուր դեպքում տևում է, ոչ թե նախորոք հայտնի, այլ պատահական ժամանակում։ Հարցումի սպասարկումից հետո սպասարկման խողովակը ազատվում է և պատրաստ է հաջորդ հարցման սպասարկմանը։

Այսպիսով ամեն մի ԶՍՀ-ում կարելի է առանձնացնել հետևյալ էլեմենտները՝

1) Մուտքային հարցումների հոսք

2) Հերթ

3) Սպասարկման խողովակներ

4) Ելքային սպասարկված հարցումների հոսք

Մասսայական սպասարկման համակարգերը բաժանվում են տարբեր տեսակների՝ ըստ մի շարք բնութագրերի․

Ըստ սպասարկման խողովակի քանակի բաժանվում են երկու տեսակի՝ միախողավակային և բազմախողովակային։Այստեղ մենք ինկատի ենք ունենալու, որ ամեն մի խողովակ կարող է սպասարկել միայն մեկ հարցում։

Ըստ սպասարկման կարգի ՄՍՀ-ն բաժանվում է երեք տեսակի։

1․ ՄՍՀ մերժումով, որում հարցումը գալով ՄՍՀ և բոլոր խողովակների զբաղված լինելու դեպքում՝ ստանում է մերժում։

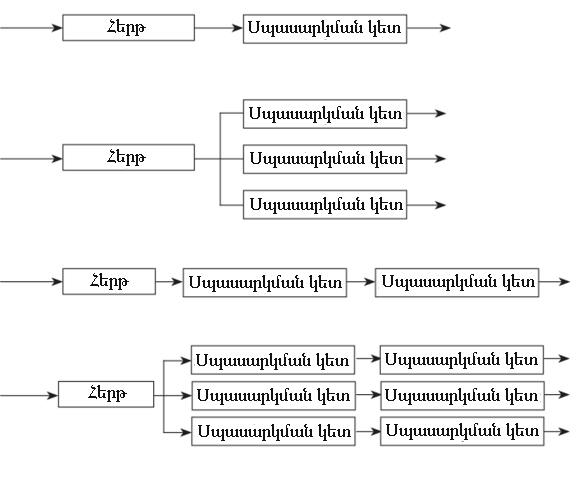
2․ ՄՍՀ անսահման սպասումով կամ հերթի առաջացումով, որի դեպքում հարցումը գալով ՄՍՀ և բոլոր խողովակների զբաղված լինելու դեպքում, կանգնում է հետևում և սպասում խողովակներից մեկի ազատմանը։

3․ ՄՍՀ խառը տիպի, որը նույն հերթի առաջացումով ՄՍՀ-ն է որոշակի սահմանափակումներով։ Այդ սահմանափակումները կարող են լինել օրինակ՝ հերթի երկարությունը, հերթում եղած ժամանակը։ Այս և հերթով ՄՍՀ-ները կարող են լինել կանոնավոր, որի դեպքում հարցումները սպասարկվում են նույն հերթականությամբ որով մուտք են գործել, և անկանոն որի դեպքում հարցումները սպասարկվում են անկանոն ձևով։

Ըստ հարցումների հոսքի սահմանափակման ԶՍՀ-ն լինում է բաց և փակ։

Եթե հարցումների հոսքը սահմանափակված է և եթե համակարգը լքած հարցումները կարող են վերադառնալ համակարգ, ապա համակարգը համարվում է փակ, հակառակ դեպքում՝ բաց։

Ըստ սպասարկման փուլերի լինում են միաֆազային և բազմաֆազային համակարգեր։ Եթե ՄՍՀ-ի խողովակները կատարում են սպասարկման նույն գործողությունները, ապա համակարգը համարվում է միաֆազային, իսկ եթե խողովակները հաջորդում են մեկը միյուսին և կատարում են տարբեր աշխատանքներ համարվում են բազմաֆազային համակարգեր։



Մասսայական սպասարկման խնդիրների լուծման համար օգտագործվում է Պուասսոնի բանաձևը․

Որտեղ -ն սպասարկման հավանականությունն է կախված և m-ից -ն կամայական ժամանակահատված է, որի ընթացքում կաշխատի համակարգը, -ն հարցումների հոսքի հաճախությունն է, m-ը հարցումների քանակն է որը պետք է կատարվի։

ՀԱՎԵԼՎԱԾՆԵՐ

Հետևյալ կոդը կատարում է հաշվարկներ օգտագործելով Պուասսոնի բանաձևը ըստ մուտքում տրված արժեքների և իրականացնում մասսայական սպասարկման համակարգի իմիտացիա։

using QueuingTheory.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace QueuingTheory

{

    /// <summary>

    /// Interaction logic for MainWindow.xaml

    /// </summary>

    public partial class MainWindow : Window

    {

        public MainWindow()

        {

            InitializeComponent();

        }

        private void Calculate(object sender, RoutedEventArgs e)

        {

            List<ChartItem> servedItems = new List<ChartItem>();

            List<ChartItem> refusedItems = new List<ChartItem>();

            List<ChartItem> inQueueItems = new List<ChartItem>();

            double lambda = Convert.ToDouble(this.inputRate.Text);

            double avgTime = Convert.ToDouble(this.avgTime.Text);

            double mu = 60d / avgTime;

            int workersCount = Convert.ToInt32(this.workersCount.Text);

            int maxLength = Convert.ToInt32(this.maxLength.Text);

            double time = Convert.ToDouble(this.time.Text);

            double ro = (lambda / mu) / workersCount;

            float dt = 0.001f;

            List<Worker> workers = new List<Worker>();

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

            {

                workers.Add(new Worker());

            }

            avgTime = avgTime / 60;

            double p = 1 - Math.Pow(Math.E, (-1d) \* lambda\*dt);

            int queue = 0;

            Random randomGenerator = new Random();

            int came = 0;

            for(float t = 0; t < time; t += dt)

            {

                double random = randomGenerator.NextDouble();

                var nt = Math.Round(t);

                var servedItem = new ChartItem

                {

                    Time = nt

                };

                if (random < p)

                {

                    came++;

                    if (queue > maxLength)

                    {

                        if (!refusedItems.Any((i) => i.Time == nt))

                        {

                            refusedItems.Add(new ChartItem

                            {

                                Time = nt,

                                Value = 1d,

                            });

                        }

                        else

                        {

                            var index = refusedItems.FindIndex((i) => i.Time == nt);

                            refusedItems[index].Value += 1d;

                        }

                        queue--;

                    }

                    inQueueItems.Add(new ChartItem

                    {

                        Time = nt,

                        Value = queue,

                    });

                    queue++;

                    for(int i = 0; i < workersCount && queue > 0; i++)

                    {

                        if(workers[i].serves >= avgTime)

                        {

                            workers[i].Served();

                        }

                        if (!workers[i].busy)

                        {

                            queue--;

                            servedItem.Value += 1d;

                            workers[i].busy = true;

                            workers[i].served++;

                        }

                        else

                        {

                            workers[i].serves += dt;

                        }

                    }

                }

                else

                {

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

                    {

                        if (workers[i].serves >= avgTime)

                        {

                            workers[i].Served();

                        }

                        else

                        {

                            workers[i].serves += dt;

                        }

                    }

                }

                if (!servedItems.Any((i) => i.Time == nt))

                {

                    servedItems.Add(servedItem);

                }

                else

                {

                    var index = servedItems.FindIndex((i) => i.Time == nt);

                    servedItems[index].Value += servedItem.Value;

                }

            }

            p = Math.Pow(Math.E, (-1d) \* lambda);

            double L = ro \* ro / (1d - ro);

            string client = "հարցում";

            List<Property> items = new List<Property>

            {

                new Property { Name = "Հարցման սպասարկման հաճախությունը", Value = mu.ToString(), Measurement = "հ/ժ"},

                new Property { Name = "1 ժամում գոնե 1 հարցում գալու հավանականությունը", Value = (1 - p).ToString(), },

                new Property { Name = "1 ժամում ոչ մի հարցում գալու հավանականությունը", Value = p.ToString(), },

                new Property { Name = "Սպասարկողի ծանրաբեռնվածությունը", Value = ro.ToString(), },

                new Property { Name = "Սպասարկողի ազատ մնալու հավանականությունը", Value = (1d - ro).ToString(), },

                new Property { Name = "Սպասարկողի զբաղված լինելու հավանականությունը", Value = ro.ToString(), },

                new Property { Name = "Հերթում եղած հարցումների միջին թիվը", Value = L.ToString(), Measurement = client },

                new Property { Name = "Հերթում սպասման միջին ժամանակը", Value = (L/lambda).ToString(), Measurement = "ժամ" },

                new Property { Name = "Ընդհանուր ժամանած հարցումների քանակը", Value = came.ToString(), Measurement = "հարցում" },

                new Property { Name = "Սպասարկված են", Value = workers.Select((w)=>w.served).Sum().ToString(), Measurement = client },

                new Property { Name = "Մերժված են", Value = refusedItems.Count.ToString(), Measurement = client }

            };

            for (int i = 0; i < workers.Count; i++)

            {

                items.Add(

                    new Property

                    {

                        Name = "Սպասարկող " + i,

                        Value = workers[i].served.ToString(),

                        Measurement = client

                    });

            }

            data.ItemsSource = items;

            served.ItemsSource = servedItems;

            refused.ItemsSource = refusedItems;

            inQueue.ItemsSource = (from customer in inQueueItems

                                   group customer by customer.Time into g

                                   select new ChartItem

                                   {

                                       Time = g.Key,

                                       Value = g.Average(c => c.Value)

                                   });

        }

    }

    public class ChartItem

    {

        public double Time { get; set; }

        public double Value { get; set; }

    }

    public class Property

    {

        public string Name { get; set; }

        public string Value { get; set; }

        public string Measurement { get; set; }

    }

    public class Worker

    {

        public bool busy { get; set; }

        public float serves { get; set; }

        public int served { get; set; }

        public void Served()

        {

            this.busy = false;

            this.serves = 0;

        }

    }

}