Оглавление

[1. Очереди, стеки, дженерики 2](#_Toc32065954)

[2. yield return 6](#_Toc32065955)

[3. Листы и словари 8](#_Toc32065956)

[4. Делегаты 11](#_Toc32065957)

[5. Элементы функционального программирования 13](#_Toc32065958)

[6. LINQ 16](#_Toc32065959)

[7. Графы и обходы 20](#_Toc32065960)

[8. Жадные алгоритмы 23](#_Toc32065961)

[9. Динамическое программирование 31](#_Toc32065962)

[10. Структуры данных 33](#_Toc32065963)

[11. События 35](#_Toc32065964)

[13. Многопоточное программирование 36](#_Toc32065965)

[14. Рефлексия типов 37](#_Toc32065966)

**CTRL + клик** по теме, чтобы перейти к странице

Все практики есть пока что только до Графов включительно

Вопросы и мелкие задачи есть для всего

# 1. Очереди, стеки, дженерики

**Стеки и очереди**:

* 0
* 2
* Deque …
* пуста + 1 элемент

**Максимум в массиве:**

static T Max<T>(T[] source)

{

if(source.Length == 0)

return default(T);

return source.Max();

}

**Практика «Limited Size Stack»**

namespace TodoApplication

{

public class LimitedSizeStack<T>

{

LinkedList<T> num = new LinkedList<T>();

private int Limit;

public LimitedSizeStack(int limit)

{

Limit = limit;

}

public void Push(T item)

{

if (num.Count == Limit) num.RemoveFirst();

num.AddLast(item);

}

public T Pop()

{

var result = num.Last.Value;

num.RemoveLast();

return result;

}

public int Count

{

get

{

var result = Limit;

return result;

}

}

}

}

**Практика «Отмена»**

namespace TodoApplication

{

public class Operation<TItem>

{

public TItem Value;

public bool operation;

}

public class ListModel<TItem>

{

public List<TItem> Items { get; }

public int Limit;

private LimitedSizeStack<Operation<TItem>> stack;

public ListModel(int limit)

{

Items = new List<TItem>();

Limit = limit;

stack = new LimitedSizeStack<Operation<TItem>>(limit);

}

public void AddItem(TItem item)

{

var e = new Operation<TItem> { Value = item, operation = true };

stack.Push(e);

Items.Add(item);

}

public void RemoveItem(int index)

{

var e = new Operation<TItem> { Value = Items[index], operation = false };

stack.Push(e);

Items.RemoveAt(index);

}

public bool CanUndo()

{

return stack.Count!=0;

}

public void Undo()

{

var e = stack.Pop();

if (e.operation)

Items.Remove(e.Value);

else

Items.Add(e.Value);

}

}

}

**Практика «CVS»**

namespace Clones

{

public class Clone

{

public Stack AddProgramm;

public Stack RemoveProgramm;

public Clone()

{

AddProgramm = new Stack();

RemoveProgramm = new Stack();

}

public Clone(Clone baseClone)

{

AddProgramm = new Stack(baseClone.AddProgramm);

RemoveProgramm = new Stack(baseClone.RemoveProgramm);

}

}

public class StackItem

{

public string Value;

public StackItem First;

public StackItem(string value, StackItem first)

{

Value = value;

First = first;

}

}

public class Stack

{

public StackItem Last;

public int Count;

public Stack()

{

}

public Stack(Stack baseStack)

{

Last = baseStack.Last;

}

public void Push(string item)

{

Last = new StackItem(item, Last);

Count++;

}

public string Pop()

{

var result = Last.Value;

Last = Last.First;

Count--;

return result;

}

}

public class CloneVersionSystem : ICloneVersionSystem

{

private List<Clone> ListClone;

public CloneVersionSystem()

{

ListClone = new List<Clone>();

ListClone.Add(new Clone());

}

public string Execute(string query)

{

var str = query.Split(' ');

var num = int.Parse(str[1])-1;

return Commands(str, num);

}

public string Commands(string[] str, int num)

{

var clone = ListClone[num];

switch (str[0])

{

case "learn":

clone.AddProgramm.Push(str[2]);

clone.RemoveProgramm.Last = null; break;

case "rollback":

if (clone.AddProgramm.Last == null) break;

var progRoll = clone.AddProgramm.Pop();

clone.RemoveProgramm.Push(progRoll); break;

case "relearn":

if (clone.RemoveProgramm.Last == null) break;

var progRelearn = clone.RemoveProgramm.Pop();

clone.AddProgramm.Push(progRelearn); break;

case "clone":

ListClone.Add(new Clone(ListClone[num])); break;

case "check":

if (clone.AddProgramm.Last == null) return "basic";

return clone.AddProgramm.Last.Value;

default: return null;

}

return null;

}

}

}

# 2. yield return

**foreach, IEnumerable и IEnumerator:**

* У IEnumerable есть метод, возвращающий IEnumerator;
* Чтобы можно было иметь несколько работающих параллельно итераторов по одной и той же коллекции
* Реализовать в MyClass интерфейс IEnumerable
* Если MoveNext возвращает false, значит следующего элемента нет; С помощью IEnumerator можно получать элементы коллекции

**ZipSum**

private static IEnumerable<int> ZipSum(IEnumerable<int> first, IEnumerable<int> second)

{

return first.Zip(second, (firstEl, secondEl) => firstEl+secondEl);

}

**Особые случаи yield return:**

* Нельзя в одном методе использовать и обычный return и yield return
* yield break в конце метода не нужен

**Практика «Экспоненциальное сглаживание»**

namespace yield

{

public static class ExpSmoothingTask

{

public static IEnumerable<DataPoint> SmoothExponentialy

(this IEnumerable<DataPoint> data, double alpha)

{

var flag = true;

double prewElement = 0.0;

foreach(var item in data)

{

DataPoint result = new DataPoint

{

X =item.X, OriginalY = item.OriginalY,

MaxY = item.MaxY, AvgSmoothedY = item.AvgSmoothedY

};

if (flag)

{

flag = false;

prewElement = item.OriginalY;

result.ExpSmoothedY = prewElement;

yield return result;

}

else

{

prewElement = prewElement + alpha \* (item.OriginalY - prewElement);

result.ExpSmoothedY = prewElement;

yield return result;

}

}

}

}

}

**Практика «Скользящее среднее»**

namespace yield

{

public static class MovingAverageTask

{

public static IEnumerable<DataPoint> MovingAverage(this IEnumerable<DataPoint> data, int windowWidth)

{

var queue = new Queue<double>();

var count = 0;

var sum = 0.0;

foreach(var item in data)

{

DataPoint result = new DataPoint

{

X =item.X, OriginalY = item.OriginalY,

MaxY = item.MaxY, AvgSmoothedY = item.AvgSmoothedY

};

if (count == windowWidth)

{

sum -= queue.Dequeue();

count--;

}

queue.Enqueue(item.OriginalY);

count++;

sum += item.OriginalY;

result.AvgSmoothedY = sum / count;

yield return result;

}

}

}

}

**Практика «Скользящий максимум»**

namespace yield

{

public static class MovingMaxTask

{

public static IEnumerable<DataPoint> MovingMax(this IEnumerable<DataPoint> data, int windowWidth)

{

var queue = new Queue<double>();

var window = new LinkedList<double>();

var num = 0.0;

foreach (var item in data)

{

queue.Enqueue(item.OriginalY);

if(queue.Count > windowWidth)

{

num = queue.Dequeue();

if(num == window.First.Value)

window.RemoveFirst();

}

while(window.Count > 0 && item.OriginalY > window.Last.Value)

window.RemoveLast();

window.AddLast(item.OriginalY);

item.MaxY = window.First.Value;

yield return item;

}

}

}

}

# 3. Листы и словари

**Сложность операций:**

* Сложность операции доступа к элементу по индексу Θ(1)
* Сложность операции Contains Θ(n)

**Тонкости операторов:**

* -
* -
* +
* +
* +
* public static A operator ==(A a, A b); public static bool operator ==(A a, int b)
* Не стоит перегружать оператор, если это сделает код более загадочным; Не стоит перегружать оператор, если это может подтолкнуть читателя к неверной интерпретации кода

**Dictionary, Equals и GetHashCode:**

* Словарь позволяет эффективно проверить, содержит ли он ключ; Для каждого ключа словарь хранит только одно значение
* return 42; return Surname.GetHashCode(); return Surname.GetHashCode() \* 31 + Name.GetHashCode(); return (Surname.GetHashCode() \* 31 + Name.GetHashCode()) \* 31 + Patronymic.GetHashCode()

**Практика «Readonly bytes»**

namespace hashes

{

public class ReadonlyBytes : IEnumerable<byte>

{

private byte[] Bytes;

private int hash;

private bool hashSet = false;

public ReadonlyBytes(params byte[] input)

{

Bytes = input;

try

{

Length = input.Length;

}

catch (Exception error)

{

throw new ArgumentNullException();

}

}

public byte this[int index]

{

get

{

if (index < 0 || index >= Bytes.Length) throw new IndexOutOfRangeException();

return Bytes[index];

}

}

public int Length { get; }

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public override string ToString()

{

var result = new StringBuilder();

result.Append("[");

if (Bytes.Length == 0) return "[]";

for (int i = 0; i < Bytes.Length; i++)

result.Append(Bytes[i].ToString() + ", ");

result.Remove(result.Length-2, 2);

result.Append("]");

return result.ToString();

}

public override bool Equals(object obj)

{

var num = obj as ReadonlyBytes;

if (num.Length != Bytes.Length) return false;

for (int i = 0; i < Bytes.Length; i++)

if (num[i] != Bytes[i]) return false;

return true;

}

public override int GetHashCode()

{

unchecked

{

if (hashSet) return hash;

hash = 1000;

var num = 322;

for (int i = 0; i < Bytes.Length; i++)

{

hash \*= num;

hash ^= Bytes[i];

}

hashSet = true;

}

return hash;

}

public IEnumerator<byte> GetEnumerator()

{

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

yield return Bytes[i];

}

}

}

**Практика «Ghosts»**

namespace hashes

{

public class GhostsTask :

IFactory<Document>, IFactory<Vector>, IFactory<Segment>, IFactory<Cat>, IFactory<Robot>,

IMagic

{

private byte[] array = {12, 47, 32, 64};

private Vector vector = new Vector(1, 2);

private Segment segment = new Segment(new Vector(0,0),new Vector(1,1));

private Cat cat = new Cat("Adolf Hitler","Aryan",new DateTime(1889,4,20));

private Robot robot = new Robot("Down",47);

public void DoMagic()

{

vector = vector.Add(new Vector(12, 47));

segment.End.Add(new Vector(12, 47));

array[3] = array[1];

cat.Rename("Kek");

Robot.BatteryCapacity -= 10;

}

Vector IFactory<Vector>.Create()

{

return vector;

}

Segment IFactory<Segment>.Create()

{

return segment;

}

Document IFactory<Document>.Create()

{

return new Document("Kek",Encoding.Unicode, array);;

}

Cat IFactory<Cat>.Create()

{

return cat;

}

Robot IFactory<Robot>.Create()

{

return robot;

}

}

}

# 4. Делегаты

**Синтаксис делегатов:** delegate void TellUser(string text);

**Синтаксис делегатов 2:** public delegate bool TryGet<T1,T2>(T1 x,Action<T1> z,out T2 y);

**Синтаксис лямбд:**

private static readonly Func<int> zero = () => 0;

private static readonly Func<int, string> toString = x => x.ToString();

private static readonly Func<double, double, double> add = (a, b) => a+b;

private static readonly Action<string> print = Console.WriteLine;

**Синтаксис лямбд 2:**

static Func<T1, T3> Combine<T1, T2, T3>(Func<T1, T2> f, Func<T2, T3> g) { return x => g(f(x)); }

**Делегаты:**

* Func<int, string, double>
* Func<int, List<int>>[]
* символ 'Z'

**Практика «Виртуальная машина Brainfuck»**

namespace func.brainfuck

{

public class VirtualMachine : IVirtualMachine

{

public VirtualMachine(string program, int memorySize)

{

Instructions = program;

Memory = new byte[memorySize];

dict = new Dictionary<char, Action<IVirtualMachine>>(program.Length);

}

public void RegisterCommand(char symbol, Action<IVirtualMachine> execute)

{

if(!dict.ContainsKey(symbol)) dict.Add(symbol,execute);

}

private Dictionary<char, Action<IVirtualMachine>> dict;

public string Instructions { get; }

public int InstructionPointer { get; set; }

public byte[] Memory { get; }

public int MemoryPointer { get; set; }

public void Run()

{

for (; InstructionPointer < Instructions.Length; InstructionPointer++)

if(dict.ContainsKey(Instructions[InstructionPointer]))

dict[Instructions[InstructionPointer]](this);

}

}

}

**Практика «Простые команды Brainfuck»**

namespace func.brainfuck

{

public class BrainfuckBasicCommands

{

public static void RegisterTo(IVirtualMachine vm, Func<int> read, Action<char> write)

{

vm.RegisterCommand('.', b => { write(Convert.ToChar(b.Memory[b.MemoryPointer])); });

vm.RegisterCommand(',', b => { b.Memory[b.MemoryPointer] = Convert.ToByte(read()); });

vm.RegisterCommand('+', b =>

{

if(b.Memory[b.MemoryPointer] == 255) b.Memory[b.MemoryPointer] = 0;

else b.Memory[b.MemoryPointer]++;

});

vm.RegisterCommand('-', b =>

{

if(b.Memory[b.MemoryPointer] == 0) b.Memory[b.MemoryPointer] = 255;

else b.Memory[b.MemoryPointer]--;

});

vm.RegisterCommand('>', b =>

{

if(b.MemoryPointer == b.Memory.Length-1) b.MemoryPointer = 0;

else b.MemoryPointer++;

});

vm.RegisterCommand('<', b =>

{

if(b.MemoryPointer == 0) b.MemoryPointer = b.Memory.Length-1;

else b.MemoryPointer--;

});

RegisterSymbols(vm);

}

public static void RegisterSymbols(IVirtualMachine vm)

{

var symbols =

"QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm1234567890".ToCharArray();

foreach (var e in symbols)

vm.RegisterCommand(e, b => { b.Memory[b.MemoryPointer] = Convert.ToByte(e);});

}

}

}

**Практика «Циклы Brainfuck»**

namespace func.brainfuck

{

public class BrainfuckLoopCommands

{

private static Dictionary<int, int> cycleOpenClose = new Dictionary<int, int>();

private static Dictionary<int, int> cycleCloseOpen = new Dictionary<int, int>();

private static Stack<int> stack = new Stack<int>();

private static void Search(IVirtualMachine b)

{

for (int i = 0; i < b.Instructions.Length; i++)

{

if (b.Instructions[i] == '[') stack.Push(i);

if (b.Instructions[i] == ']'){

var index = stack.Pop();

cycleCloseOpen[i]=index;

cycleOpenClose[index]= i;}

}

}

public static void RegisterTo(IVirtualMachine vm)

{

Search(vm);

vm.RegisterCommand('[', b => {

if (b.Memory[b.MemoryPointer] == 0)

b.InstructionPointer = cycleOpenClose[b.InstructionPointer];});

vm.RegisterCommand(']', b => {

if (b.Memory[b.MemoryPointer] != 0)

b.InstructionPointer = cycleCloseOpen[b.InstructionPointer];});

}

}

}

# 5. Элементы функционального программирования

**Частичное применение функций:**

* 5
* 111
* 53

**Реализация метода FirstOrDefault**

private static T FirstOrDefault<T>(IEnumerable<T> source, Func<T, bool> filter)

{

return source.FirstOrDefault(e => filter(e));

}

**Реализация метода Take**

private static IEnumerable<T> Take<T>(IEnumerable<T> source, int count) => source.Take(count);

**Последовательность вызовов**

* s f s f
* s s f f
* s f
* s s f

**Практика «Лямбды и делегаты»**

namespace func\_rocket

{

public class ForcesTask

{

public static RocketForce GetThrustForce(double forceValue)

{

return r =>

{

var vector = new Vector(forceValue,0);

var x = Math.Cos(r.Direction)\* vector.X - Math.Sin(r.Direction)\*vector.Y;

var y = Math.Cos(r.Direction)\* vector.Y + Math.Sin(r.Direction)\*vector.X;

return new Vector(x,y);

};

}

public static RocketForce ConvertGravityToForce(Gravity gravity, Size spaceSize)

{

return r => gravity(spaceSize, r.Location);

}

public static RocketForce Sum(params RocketForce[] forces)

{

return r =>

{

var vector = new Vector(0,0);

var vec = new Vector(0,0);

foreach(var e in forces)

vector += e.Invoke(new Rocket(vec,vec,0));

return vector;

};

}

}

}

**Практика «Уровни»**

namespace func\_rocket

{

public class LevelsTask

{

static readonly Physics standardPhysics = new Physics();

public static IEnumerable<Level> CreateLevels()

{

var endVector = new Vector(700, 500);

var rocket = new Rocket(new Vector(200, 500), Vector.Zero, -0.5 \* Math.PI);

yield return new Level("Zero", rocket, endVector,

(size, v) => Vector.Zero, standardPhysics);

yield return new Level("Heavy", rocket, endVector,

(size, v) => new Vector(0, 0.9), standardPhysics);

yield return new Level("Up", rocket, endVector,

(size, v) => new Vector(0, -300/(300 + (size.Height-v.Y))), standardPhysics);

yield return new Level("WhiteHole", rocket, endVector,

(size,v)=> WhiteHoleVector(v,endVector),standardPhysics);

yield return new Level("BlackHole", rocket, endVector,

(size, v) => BlackHoleVector(v,endVector,rocket.Location), standardPhysics);

yield return new Level("BlackAndWhite", rocket, endVector,

(size, v) =>

{

var vectorWhite = WhiteHoleVector(v, endVector);

var vectorBlack = BlackHoleVector(v, endVector, rocket.Location);

return (vectorWhite + vectorBlack) / 2;

}, standardPhysics);

}

public static Vector WhiteHoleVector(Vector v, Vector end)

{

var d = (v - end).Length;

return (v - end).Normalize() \* 140 \* d / (d \* d + 1);

}

public static Vector BlackHoleVector(Vector v, Vector end, Vector rocketLocation)

{

var blackHoleLocation = (end + rocketLocation) / 2;

var d = (blackHoleLocation - v).Length;

return (blackHoleLocation - v).Normalize() \* 300 \* d / (d \* d + 1);

}

}

}

**Практика «Управление»**

namespace func\_rocket

{

public class ControlTask

{

private static double angle = 0.0;

public static Turn ControlRocket(Rocket rocket, Vector target)

{

Calculate(rocket, target);

if (angle > 0)

return Turn.Right;

else if (angle < 0)

return Turn.Left;

return Turn.None;

}

private static void Calculate(Rocket rocket, Vector target)

{

var distance = target-rocket.Location;

if (Check(distance, rocket))

angle = (distance.Angle \* 2 - rocket.Velocity.Angle - rocket.Direction) / 2;

else angle = distance.Angle - rocket.Direction;

}

private static bool Check(Vector distance, Rocket rocket)

{

var a = Math.Abs(distance.Angle - rocket.Velocity.Angle)<0.5;

var b = Math.Abs(distance.Angle - rocket.Direction) < 0.5;

return a||b;

}

}

}

# 6. LINQ

**Чтение массива чисел**

public static int[] ParseNumbers(IEnumerable<string> lines)

{

return lines.Where(x => !string.IsNullOrEmpty(x)).Select(int.Parse).ToArray();

}

**Чтение списка точек**

public static List<Point> ParsePoints(IEnumerable<string> lines)

{

return lines

.Select(x => new Point(int.Parse(x.Split()[0]), int.Parse(x.Split()[1])))

.ToList();

}

**Объединение коллекций**

public static string[] GetAllStudents(Classroom[] classes)

{

return classes

.SelectMany(x => x.Students)

.ToArray();

}

**Декартово произведение**

public static IEnumerable<Point> GetNeighbours(Point p)

{

int[] d = {-1, 0, 1};

return d.SelectMany(x => d.Select(y => new Point(p.X + x, p.Y + y))).Where(point => !p.Equals(point));

}

**Составление словаря**

public static string[] GetSortedWords(params string[] textLines)

{

return textLines

.SelectMany(s => Regex.Split(s.ToLower(), @"\W+"))

.OrderBy(x => x)

.Distinct()

.ToArray();

}

**Сортировка кортежей**

public static List<string> GetSortedWords(string text)

{

return Regex.Split(text.ToLower(), @"\W+")

.Where(x => x != "")

.OrderBy(x => (x.Length, x))

.Distinct()

.ToList();

}

**Поиск самого длинного слова**

public static string GetLongest(IEnumerable<string> words)

{

return words

.Min(word => (-word.Length, word))

.Item2;

}

**Создание частотного словаря**

public static Tuple<string, int>[] GetMostFrequentWords(string text, int count)

{

return Regex.Split(text, @"\W+")

.Where(word => word != "")

.Select(word => word.ToLower())

.GroupBy(word => word)

.Select(x => Tuple.Create(x.Key, x.Count()))

.OrderByDescending(x => x.Item2)

.ThenBy(x => x.Item1)

.Take(count)

.ToArray();

}

**Создание обратного индекса**

public static ILookup<string, int> BuildInvertedIndex(Document[] documents)

{

return documents

.SelectMany(w => Regex.Split(w.Text.ToLower(), @"\W+")

.Distinct()

.Where(x => x != "")

.Select(x => Tuple.Create(x, w.Id)))

.ToLookup(w => w.Item1, w => w.Item2);

}

**Практика «Median & Bigrams»**

namespace linq\_slideviews

{

public static class ExtensionsTask

{

public static double Median(this IEnumerable<double> items)

{

var list = items.ToList();

if(list.Count == 0)

throw new InvalidOperationException();

list.Sort();

if (list.Count % 2 != 0)

return list[list.Count / 2];

var result = (list[list.Count / 2] + list[(list.Count / 2) - 1]) / 2;

return result;

}

public static IEnumerable<Tuple<T, T>> Bigrams<T>(this IEnumerable<T> items)

{

T prevItem = default(T);

var flag = true;

foreach(var item in items)

{

if (flag)

{

prevItem = item;

flag = false;

continue;

}

yield return Tuple.Create(prevItem, item);

prevItem = item;

}

}

}

}

**Практика «Чтение файла»**

namespace linq\_slideviews

{

public class ParsingTask

{

private static Dictionary<int, SlideRecord> createSlides = new Dictionary<int, SlideRecord>();

public static IDictionary<int, SlideRecord> ParseSlideRecords(IEnumerable<string> lines)

{

return lines.Select(k => k.Split(';'))

.Where(s => s.Length == 3 && giveSlideType(s[1]) != -1)

.Select(m => SlideCreate(m))

.Where(a => a != null)

.ToDictionary(a => a.SlideId, a => a);

}

private static SlideRecord SlideCreate(string[] part)

{

int id;

if(int.TryParse(part[0], out id))

return new SlideRecord(id, (SlideType)giveSlideType(part[1]),part[2]);

return null;

}

private static int giveSlideType(string s)

{

switch (s)

{

case "theory":

return 0;

case "exercise":

return 1;

case "quiz":

return 2;

default:

return -1;

}

}

public static IEnumerable<VisitRecord> ParseVisitRecords

(IEnumerable<string> lines, IDictionary<int, SlideRecord> slides)

{

return lines.Where(s => s != "UserId;SlideId;Date;Time")

.Select(s => ParseVisit(s,slides));

}

private static VisitRecord ParseVisit(string line, IDictionary<int, SlideRecord> slides)

{

var part = line.Split(';');

int userId;

int slideId;

DateTime d;

if (part.Length == 4 &&

int.TryParse(part[0], out userId)&&

int.TryParse(part[1], out slideId)&&

DateTime.TryParse(part[2],out d)&&

DateTime.TryParse(part[3],out d)&&

slides.ContainsKey(slideId))

return new VisitRecord(userId,

slideId, GetDateTime(part[2],part[3]),

slides[slideId].SlideType);

throw new FormatException("Wrong line [" + line + "]");

}

private static DateTime GetDateTime(string date, string time)

{

var dateString = $"{date} {time}";

return DateTime.Parse(dateString);

}

}

}

**Практика «Статистика»**

namespace linq\_slideviews

{

public class StatisticsTask

{

public static double GetMedianTimePerSlide(List<VisitRecord> visits, SlideType slideType)

{

return visits.OrderBy(x => x.DateTime)

.GroupBy(x => x.UserId)

.SelectMany(group =>

{

var result = group.Bigrams();

return result.Where(i => i.Item1.SlideType == slideType);

})

.Select(x =>

{

var date1 = x.Item1.DateTime;

var date2 = x.Item2.DateTime;

return date2.Subtract(date1).TotalMinutes;

})

.Where(i => i >= 1 && i <= 120)

.DefaultIfEmpty(0)

.Median();

}

}

}

# 7. Графы и обходы

**Графы**

* 2
* 1
* В нем есть цикл; Он связен; В нем есть путь из вершины 0 в вершину 4

**Сложность обхода лабиринта (ЭТО СТЕПЕНЬ N)**

* 2
* 2
* 3

**Сложность обходов графа**

* O(V+E)
* O(V)
* O(V+E)
* O(V)

**Поиск цикла в неориентированном графе**

public static bool HasCycle(List<Node> graph)

{

var visited = new HashSet<Node>();

var finished = new HashSet<Node>();

var stack = new Stack<Node>();

visited.Add(graph.First());

stack.Push(graph.First());

while (stack.Count != 0)

{

var node = stack.Pop();

foreach (var nextNode in node.IncidentNodes)

{

if(finished.Contains(nextNode)) continue;

if (visited.Contains(nextNode)) return true;

stack.Push(nextNode);

visited.Add(nextNode);

}

finished.Add(node);

}

return false;

}

**Практика «Поиск в ширину»**

namespace Dungeon

{

public class BfsTask

{

public static IEnumerable<SinglyLinkedList<Point>> FindPaths(Map map, Point start, Point[] chests)

{

var queue = new Queue<SinglyLinkedList<Point>>();

var visited = new HashSet<Point>(){start};

var pointStart = new SinglyLinkedList<Point>(start, null);

var route = Walker.PossibleDirections.Select(p => new Point(p));

queue.Enqueue(pointStart);

while(queue.Count != 0)

{

var result = queue.Dequeue();

var point = result.Value;

if(!map.InBounds(point)) continue;

if(map.Dungeon[point.X,point.Y]== 0) continue;

if (chests.Contains(point)) yield return result;

foreach(var e in route)

{

var p = new Point(point.X + e.X, point.Y + e.Y);

if (!visited.Contains(p))

{

queue.Enqueue(new SinglyLinkedList<Point>(p, result));

visited.Add(p);

}

}

}

}

}

}

**Практика «Вынести клад!»**

namespace Dungeon

{

public class DungeonTask

{

public static MoveDirection[] FindShortestPath(Map map)

{

var start = map.InitialPosition;

var exit = map.Exit;

var chests = map.Chests;

var routeNoChest = BfsTask.FindPaths(map, start, new Point[] {exit}).FirstOrDefault();

if (routeNoChest == null) return new MoveDirection[0];

var moveToExit = routeNoChest.ToList();

moveToExit.Reverse();

if (chests.Any(c=>moveToExit.Contains(c)))

return moveToExit.Zip(moveToExit.Skip(1), (Move)).ToArray();

var routeStartToChests = BfsTask.FindPaths(map, start, chests);

var routeExitToChests = BfsTask.FindPaths(map, exit, chests).DefaultIfEmpty();

if (routeStartToChests.FirstOrDefault() == null)

return moveToExit.Zip(moveToExit.Skip(1), (Move)).ToArray();

var routeStartToExit = routeStartToChests.Join(routeExitToChests,f=>f.Value,s=>s.Value, (f, s) => new {

Length = f.Length + s.Length,

listFinish = f.ToList(),

listStart = s.ToList() });

var listsTuple = routeStartToExit.OrderBy(l => l.Length)

.Select(v => Tuple.Create(v.listFinish, v.listStart)).First();

listsTuple.Item1.Reverse();

listsTuple.Item1.AddRange(listsTuple.Item2.Skip(1));

return listsTuple.Item1.Zip(listsTuple.Item1.Skip(1), (Move)).ToArray();

}

private static MoveDirection Move(Point start, Point finish)

{

var d = new Point(finish.X - start.X, finish.Y - start.Y);

if (d.X == 1) return MoveDirection.Right;

else if (d.X == -1) return MoveDirection.Left;

return d.Y == 1 ? MoveDirection.Down : MoveDirection.Up;

}

}

}

**Практика «Поделить территорию!»**

namespace Rivals

{

public class RivalsTask

{

public static IEnumerable<OwnedLocation> AssignOwners(Map map)

{

var queue = new Queue<OwnedLocation>();

var visited = new HashSet<Point>();

var players = map.Players;

for(var i = 0; i <players.Length;i++)

queue.Enqueue(new OwnedLocation(i, players[i], 0));

while(queue.Count>0)

{

var location = queue.Dequeue();

if (visited.Contains(location.Location) ||

!map.InBounds(location.Location) ||

map.Maze[location.Location.X, location.Location.Y] != MapCell.Empty)

continue;

visited.Add(location.Location);

yield return location;

AddToQueue(queue, location);

}

}

private static void AddToQueue(Queue<OwnedLocation> queue, OwnedLocation location)

{

for (var dy = -1; dy <= 1; dy++)

for (var dx = -1; dx <= 1; dx++)

if ((dy == 0 || dx == 0) && dy != dx)

{

var newLocation = new OwnedLocation(location.Owner,

new Point(location.Location.X + dx, location.Location.Y + dy),

location.Distance + 1);

queue.Enqueue(newLocation);

}

}

}

}

# 8. Жадные алгоритмы

**Комбинаторные задачи**

* +
* +
* -

**Реализация планировщика**

public static IEnumerable<Tuple<int, int>> PlanSchedule(IEnumerable<Tuple<int, int>> meetings)

{

var leftEdge = int.MinValue;

foreach (var meeting in meetings.OrderBy(m => m.Item2))

if (meeting.Item1 >= leftEdge)

{

leftEdge = meeting.Item2;

yield return meeting;

}

}

**Реализация алгоритма Краскала**

public static IEnumerable<Edge> FindMinimumSpanningTree(IEnumerable<Edge> edges)

{

var tree = new List<Edge>();

foreach (var edge in edges.OrderBy(x => x.Weight))

if (!HasCycle(new List<Edge>(tree){edge}))

tree.Add(edge);

return tree;

}

**Размен монет**

* -
* К задаче поиска кратчайшего пути

**Практика «Путь в лабиринте»**

**–**

**Практика «Жадина в лабиринте»**

namespace Greedy

{

public class GreedyPathFinder : IPathFinder

{

public List<Point> FindPathToCompleteGoal(State state)

{

if (state.Chests.Count < state.Goal)

{

return new List<Point>();

}

var unpickedChests = new HashSet<Point>(state.Chests);

var pickedChestsNumber = 0;

var path = new List<Point>();

var currentEnergy = state.InitialEnergy;

var currentPosition = state.Position;

while (pickedChestsNumber < state.Goal)

{

int wastedEnergy;

var pathToNextChest = GetPathToNextChest(unpickedChests, currentPosition, state, out wastedEnergy);

currentEnergy -= wastedEnergy;

if (currentEnergy < 0 || pathToNextChest == null)

{

return new List<Point>();

}

path.AddRange(pathToNextChest);

if (pathToNextChest.Count > 0)

{

currentPosition = pathToNextChest[pathToNextChest.Count - 1];

}

unpickedChests.Remove(currentPosition);

pickedChestsNumber++;

}

return path;

}

private List<Point> GetPathToNextChest(HashSet<Point> chests,

Point startPoint, State state, out int wastedEnergy)

{

var passedPoints = new Dictionary<Point, MovingData>();

var markedPoints = new Dictionary<Point, MovingData>();

markedPoints.Add(startPoint, new MovingData(startPoint, 0));

Point lastPoint;

while (true)

{

if (markedPoints.Count == 0)

{

wastedEnergy = 0;

return null;

}

var openingPoint = GetOpeningPoint(markedPoints);

passedPoints.Add(openingPoint, markedPoints[openingPoint]);

markedPoints.Remove(openingPoint);

if (chests.Contains(openingPoint))

{

lastPoint = openingPoint;

break;

}

foreach (var nextPoint in Environs(openingPoint))

{

if (passedPoints.ContainsKey(nextPoint) || !state.InsideMap(nextPoint) || state.IsWallAt(nextPoint))

{

continue;

}

var energyForNextPoint = state.CellCost[nextPoint.X, nextPoint.Y]

+ passedPoints[openingPoint].WastedEnergy;

UpdateMarkedPoints(markedPoints, nextPoint, energyForNextPoint, openingPoint);

}

}

wastedEnergy = passedPoints[lastPoint].WastedEnergy;

return GetResult(passedPoints, startPoint, lastPoint);

}

private Point GetOpeningPoint(Dictionary<Point, MovingData> markedPoints)

{

var openingPoint = default(Point);

var energyToPoint = int.MaxValue;

foreach (var point in markedPoints.Keys)

{

if (markedPoints[point].WastedEnergy < energyToPoint)

{

openingPoint = point;

energyToPoint = markedPoints[point].WastedEnergy;

}

}

return openingPoint;

}

private IEnumerable<Point> Environs(Point current)

{

for (int x = current.X - 1; x < current.X + 2; x++)

{

for (int y = current.Y - 1; y < current.Y + 2; y++)

{

if (x == current.X || y == current.Y)

{

yield return new Point(x, y);

}

}

}

}

private void UpdateMarkedPoints(Dictionary<Point, MovingData> markedPoints,

Point nextPoint, int energyForNextPoint, Point previousPoint)

{

if (markedPoints.TryGetValue(nextPoint, out var alreadyMarkedInfo))

{

if (alreadyMarkedInfo.WastedEnergy <= energyForNextPoint)

{

return;

}

}

markedPoints[nextPoint] = new MovingData(previousPoint, energyForNextPoint);

}

private List<Point> GetResult(Dictionary<Point, MovingData> passedPoints, Point startPoint, Point lastPoint)

{

var result = new List<Point>();

while (lastPoint != startPoint)

{

result.Add(lastPoint);

lastPoint = passedPoints[lastPoint].Previous;

}

result.Reverse();

return result;

}

class MovingData

{

public Point Previous { get; set; }

public int WastedEnergy { get; set; }

public MovingData(Point previous, int wastedEnergy)

{

Previous = previous;

WastedEnergy = wastedEnergy;

}

}

}

}

**Практика «Оптимальный маршрут»**

namespace Greedy

{

public class NotGreedyPathFinder : IPathFinder

{

public const int Limiter = 8;

public List<Point> FindPathToCompleteGoal(State state)

{

var graph = new Graph(state.MapHeight \* state.MapWidth);

var weights = new Dictionary<Edge, double>();

var chests = state.Chests.Take(Limiter);

var pathsBetweenChests = new Dictionary<Point, Dictionary<Point, List<Point>>>();

InitializeGraphOnCells(state, weights, graph);

pathsBetweenChests[state.Position] = GetPathToAllChests(graph, weights, state.Position, state);

foreach (var chest in chests)

pathsBetweenChests[chest] = GetPathToAllChests(graph, weights, chest, state);

var bestPath = new List<Point>();

for (var i = 1; i <= Limiter; i++)

{

var reachableChestPermutations = GetPermutations(chests, i)

.Where(perm => PathsPermutationStaminaCost(state, perm, pathsBetweenChests) <= state.Energy);

if (!reachableChestPermutations.Any())

break;

bestPath = reachableChestPermutations

.First()

.ToList();

}

return GetTotalPathByPermutation(state, bestPath, pathsBetweenChests);

}

private static Dictionary<Point, List<Point>> GetPathToAllChests(Graph graph, Dictionary<Edge, double> weights,

Point start, State state)

{

var result = new Dictionary<Point, List<Point>>();

var chests = state.Chests.Where(chest => chest != start);

foreach (var chest in chests)

result[chest] = GetPathToChest(graph, weights, state, start, chest);

return result;

}

private static List<Point> GetPathToChest(Graph graph, Dictionary<Edge, double> weights,

State state, Point start, Point chest)

{

var initialPositionNumber = GetPointNumber(start, state.MapWidth);

var chestPositionNumber = GetPointNumber(chest, state.MapWidth);

var path = Dijkstra(graph, weights, graph[initialPositionNumber], graph[chestPositionNumber]);

return path?.Select(n => CreatePointByNumber(n.NodeNumber, state.MapWidth)).Skip(1).ToList() ??

new List<Point>();

}

private static int PathStaminaCost(State state, IEnumerable<Point> path)

{

return path.Sum(point => state.CellCost[point.X, point.Y]);

}

private static int PathsPermutationStaminaCost(State state, IEnumerable<Point> chests,

Dictionary<Point, Dictionary<Point, List<Point>>> pathsBetweenChests)

{

var current = state.Position;

var total = 0;

foreach (var chest in chests)

{

total += PathStaminaCost(state, pathsBetweenChests[current][chest]);

current = chest;

}

return total;

}

private static List<Point> GetTotalPathByPermutation(State state, ICollection<Point> pointsPermutation,

Dictionary<Point, Dictionary<Point, List<Point>>> pathsBetweenChests)

{

var current = state.Position;

var result = new List<Point>();

foreach (var point in pointsPermutation)

{

result.AddRange(pathsBetweenChests[current][point]);

current = point;

}

return result;

}

private static IEnumerable<IEnumerable<T>> GetPermutations<T>(IEnumerable<T> list, int length)

{

if (length == 1) return list.Select(t => new[] { t });

return GetPermutations(list, length - 1)

.SelectMany(t => list.Where(o => !t.Contains(o)),

(t1, t2) => t1.Concat(new[] { t2 }));

}

private static void InitializeGraphOnCells(State state, IDictionary<Edge, double> weights, Graph graph)

{

for (var y = 0; y < state.MapHeight; y++)

for (var x = 0; x < state.MapWidth; x++)

{

var point = new Point(x, y);

for (var dy = -1; dy <= 1; dy++)

for (var dx = -1; dx <= 1; dx++)

{

if (dx != 0 && dy != 0) continue;

var neighbour = new Point(x + dx, y + dy);

if (!state.InsideMap(neighbour)) continue;

if (state.IsWallAt(neighbour)) continue;

var pointNumber = GetPointNumber(point, state.MapWidth);

var neighbourNumber = GetPointNumber(neighbour, state.MapWidth);

weights[graph.Connect(pointNumber, neighbourNumber)] = state.CellCost[neighbour.X, neighbour.Y];

}

}

}

private static int GetPointNumber(Point point, int mapWidth)

{

return point.Y \* mapWidth + point.X;

}

private static Point CreatePointByNumber(int pointNumber, int mapWidth)

{

return new Point(pointNumber % mapWidth, pointNumber / mapWidth);

}

private static List<Node> Dijkstra(Graph graph, Dictionary<Edge, double> weights, Node start, Node end)

{

var notVisited = graph.Nodes.ToList();

var track = new Dictionary<Node, DijkstraData>();

track[start] = new DijkstraData { Price = 0, Previous = null };

while (true)

{

Node toOpen = null;

var bestPrice = double.PositiveInfinity;

foreach (var e in notVisited)

if (track.ContainsKey(e) && track[e].Price < bestPrice)

{

bestPrice = track[e].Price;

toOpen = e;

}

if (toOpen == null) return null;

if (toOpen == end) break;

foreach (var e in toOpen.IncidentEdges.Where(z => z.From == toOpen))

{

var currentPrice = track[toOpen].Price + weights[e];

var nextNode = e.OtherNode(toOpen);

if (!track.ContainsKey(nextNode) || track[nextNode].Price > currentPrice)

track[nextNode] = new DijkstraData { Previous = toOpen, Price = currentPrice };

}

notVisited.Remove(toOpen);

}

var result = new List<Node>();

while (end != null)

{

result.Add(end);

end = track[end].Previous;

}

result.Reverse();

return result;

}

internal class DijkstraData

{

public Node Previous { get; set; }

public double Price { get; set; }

}

internal class Edge

{

public readonly Node From;

public readonly Node To;

public Edge(Node first, Node second)

{

From = first;

To = second;

}

public bool IsIncident(Node node)

{

return From == node || To == node;

}

public Node OtherNode(Node node)

{

if (!IsIncident(node)) throw new ArgumentException();

if (From == node) return To;

return From;

}

}

internal class Node

{

private readonly List<Edge> edges = new List<Edge>();

public readonly int NodeNumber;

public Node(int number)

{

NodeNumber = number;

}

public IEnumerable<Node> IncidentNodes

{

get { return edges.Select(z => z.OtherNode(this)); }

}

public IEnumerable<Edge> IncidentEdges

{

get

{

foreach (var e in edges) yield return e;

}

}

public static Edge Connect(Node node1, Node node2, Graph graph)

{

if (!graph.Nodes.Contains(node1) || !graph.Nodes.Contains(node2)) throw new ArgumentException();

var edge = new Edge(node1, node2);

node1.edges.Add(edge);

return edge;

}

}

internal class Graph

{

private readonly Node[] nodes;

public Graph(int nodesCount)

{

nodes = Enumerable.Range(0, nodesCount).Select(z => new Node(z)).ToArray();

}

public int Length => nodes.Length;

public Node this[int index] => nodes[index];

public IEnumerable<Node> Nodes

{

get

{

foreach (var node in nodes) yield return node;

}

}

public IEnumerable<Edge> Edges

{

get { return nodes.SelectMany(z => z.IncidentEdges).Distinct(); }

}

public Edge Connect(int index1, int index2)

{

return Node.Connect(nodes[index1], nodes[index2], this);

}

public static Graph MakeGraph(params int[] incidentNodes)

{

var graph = new Graph(incidentNodes.Max() + 1);

for (var i = 0; i < incidentNodes.Length - 1; i += 2)

graph.Connect(incidentNodes[i], incidentNodes[i + 1]);

return graph;

}

}

}

}

# 9. Динамическое программирование

**==//==**

* Перебрать все подмножества мероприятий и выбрать наилучшую комбинацию; С помощью динамического программирования
* основы программирования и математика

**Планирование встреч**

public static int GetOptimalScheduleGain(params Event[] events)

{

var fakeBorderEvent = new Event { StartTime = int.MinValue, FinishTime = int.MinValue, Price = 0 };

events = events.Concat(new[] { fakeBorderEvent }).OrderBy(e => e.FinishTime).ToArray();

var opt = new int[events.Length];

opt[0] = 0;

for (var k = 1; k < events.Length; k++)

{

int indexOfCompatibleEvent = Array.FindLastIndex(events, x => x.FinishTime < events[k].StartTime);

opt[k] = Math.Max(opt[k - 1], events[k].Price+opt[indexOfCompatibleEvent]);

}

return opt.Last();

}

**Расстояние Левенштейна**

* 2
* 7

**==//==**

public static int LevenshteinDistance(string first, string second)

{

var opt = new int[first.Length + 1, second.Length + 1];

for (var i = 0; i <= first.Length; ++i) opt[i, 0] = i;

for (var i = 0; i <= second.Length; ++i) opt[0, i] = i;

for (var i = 1; i <= first.Length; ++i)

for (var j = 1; j <= second.Length; ++j)

{

if (first[i - 1] == second[j - 1])

opt[i, j] = opt[i - 1, j - 1];

else

opt[i, j] = 1 + Math.Min(Math.Min(opt[i - 1, j], opt[i, j - 1]), opt[i - 1, j - 1]);

}

return opt[first.Length, second.Length];

}

**Реализация Форда-Беллмана**

public static int GetMinPathCost(List<Edge> edges, int startNode, int finalNode)

{

var maxNodeIndex =

edges.SelectMany(e => new[] { e.From, e.To })

.Concat(new[] { startNode, finalNode })

.Max();

int[] opt = Enumerable.Repeat(int.MaxValue, maxNodeIndex + 1).ToArray();

opt[startNode] = 0;

for (var pathSize = 1; pathSize <= maxNodeIndex; pathSize++)

foreach (var edge in edges)

if (opt[edge.From] != int.MaxValue)

opt[edge.To] = Math.Min(opt[edge.To], opt[edge.From] + edge.Cost);

return opt[finalNode];

}

**Практика «Антиплагиат»**

**–**

**Практика «Diff Tool»**

**–**

**Практика «Счастливые билеты»**

namespace Tickets

{

internal class TicketsTask

{

private const int MaxLen = 100;

private const int MaxSum = 2000;

public static BigInteger Solve(int halfLen, int totalSum)

{

if (totalSum % 2 != 0)

return 0;

var happyTickets = InitializeHappyTicketsContainer();

var halfResult = CountHappyTickets(happyTickets, halfLen, totalSum / 2);

return halfResult \* halfResult;

}

private static BigInteger[,] InitializeHappyTicketsContainer()

{

var happyTickets = new BigInteger[MaxLen + 1, MaxSum + 1];

for (var i = 0; i < MaxLen; i++)

for (var j = 0; j < MaxSum; j++)

{

happyTickets[i, j] = -1;

}

return happyTickets;

}

private static BigInteger CountHappyTickets(BigInteger[,] happyTickets, int len, int sum)

{

if (happyTickets[len, sum] >= 0) return happyTickets[len, sum];

if (sum == 0) return 1;

if (len == 0) return 0;

happyTickets[len, sum] = 0;

for (var i = 0; i < 10; i++)

{

if (sum - i >= 0)

{

happyTickets[len, sum] += CountHappyTickets(happyTickets, len - 1, sum - i);

}

}

return happyTickets[len, sum];

}

}

}

# 10. Структуры данных

**HeapifyUp**

public static void HeapifyUp(List<int> heap)

{

var itemIndex = heap.Count-1;

while (heap[itemIndex] < heap[itemIndex / 2])

{

var parentIndex = itemIndex / 2;

var t = heap[itemIndex];

heap[itemIndex] = heap[parentIndex];

heap[parentIndex] = t;

itemIndex = parentIndex;

}

}

**GetMinValue**

public static int GetMinValue(TreeNode root) { return root.Left == null ? root.Value : GetMinValue(root.Left); }

**Реализация бинарного дерева**

public static TreeNode Search(TreeNode root, int element)

{

if (root == null) return null;

if (element == root.Value) return root;

return Search(element < root.Value ? root.Left : root.Right, element);

}

**Структуры данных**

* Это может быть не тем и не другим
* При добавлении N чисел по порядку начиная с меньших к большим, высота дерева окажется порядка N; Вставка элемента в бинарное дерево поиска имеет сложность Θ(h), где h — высота дерева
* Это куча
* +

**Практика «Add и Contains»**

**–**

**Практика «Enumerable и Индексатор»**

**–**

**Практика «Disk Tree»**

namespace DiskTree

{

public class DiskTreeTask

{

public class Root

{

public string Name;

public Dictionary<string, Root> Nodes = new Dictionary<string, Root>();

public Root(string name)

{

Name = name;

}

public Root GetDirection(string subRoot)

{

return Nodes.TryGetValue(subRoot, out Root node)

? node : Nodes[subRoot] = new Root(subRoot);

}

public List<string> MakeConcluson(int i, List<string> list)

{

if (i >= 0)

list.Add(new string(' ', i) + Name);

i++;

foreach (var child in Nodes.Values.OrderBy(root => root.Name,

StringComparer.Ordinal))

list = child.MakeConcluson(i, list);

return list;

}

}

public static List<string> Solve(List<string> input)

{

var root = new Root("");

foreach (var name in input)

{

var path = name.Split('\\');

var node = root;

foreach (var item in path)

node = node.GetDirection(item);

}

return root.MakeConcluson(-1, new List<string>());

}

}

}

# 11. События

**События и делегаты**

* Мультикаст-делегаты позволяют комбинировать несколько делегатов; Мультикаст-делегаты можно вызывать так же, как и обычные делегаты; Мультикаст-делегаты позволяют комбинировать даже делегаты, возвращающие значения
* Событие — обертка над делегатом, которая помогает обеспечивать целостность; Событие класса нельзя вызвать из метода другого класса

# 13. Многопоточное программирование

**Процессы и потоки**

* +
* -
* +
* +
* -
* -

**Потоки и блокировки**

* Чтобы обеспечить корректное использование разделяемого между потоками ресурса
* +
* +
* -
* +
* ... все остальные потоки не окажутся вне этой секции; ... все остальные потоки не окажутся вне секций, заключенных в `lock(obj)`, с тем же `obj`
* Сверяться с документацией; По умолчанию считать, что все операции не являются потокобезопасными

**Практика «Поток для AI»**

**–**

**Практика «Параллельный AI»**

**–**

# 14. Рефлексия типов

**Практика «Документация»**

**–**