Санкт-Петербургский Национальный Исследовательский
Университет Информационных Технологий, Механики и Оптики
Мегафакультет компьютерных технологий и управления

Дисциплина
«Алгоритмы и структуры данных»

Лабораторная работа №1
“Жадные алгоритмы”

Выполнил:
Студент группы P3218
Рябов Сергей Витальевич
Преподаватель:
Муромцев Дмитрий Ильич

Санкт-Петербург,
2018

1. Покрыть отрезки точками

По данным n отрезкам необходимо найти множество точек минимального размера, для которого каждый из отрезков содержит хотя бы одну из точек.

В первой строке дано число $1 \le n \le 100$ отрезков. Каждая из последующих n строк содержит по два числа $0 \le l \le r \le 10^9$, задающих начало и конец отрезка. Выведите оптимальное число m точек и сами m точек. Если таких множеств точек несколько, выведите любое из них.

Исходный код (C#):

```
using System;
using System.Collections.Generic;
using System.IO;

namespace Stepik
{
    public struct LineSegment
    {
        public int Left { get; }
        public int Right { get; }

        public LineSegment(int left, int right)
        {
            Left = left;
            Right = right;
        }
    }

    public class DotsFiller
    {
        private LineSegment[] lines;

        public DotsFiller(LineSegment[] lines)
        {
            this.lines = lines;
        }

        public List<int> Fill()
        {
            Sort();

            List<int> dots = new List<int>();
            for (int i = 0; i < lines.Length; i++)
            {
                int dot = lines[i].Right;
                dots.Add(dot);
                while (i < lines.Length && lines[i].Left <= dot)
                    i++;
                i--;
            }
            return dots;
        }

        private void Sort()
        {
            LineSegment t;
            for (int p = 0; p <= lines.Length - 2; p++)
            {
                for (int i = 0; i <= lines.Length - 2; i++)
                {
                    if (lines[i].Right > lines[i + 1].Right)
                    {
```

```
                    t = lines[i + 1];
                    lines[i + 1] = lines[i];
                    lines[i] = t;
                }
            }
        }
    }
}

public class DotsFillerTask
{
    static void Main(string[] args)
    {
        int n = int.Parse(Console.ReadLine());
        LineSegment[] lines = new LineSegment[n];

        for (int i = 0; i < n; i++)
        {
            string[] str = Console.ReadLine().Split(' ');
            lines[i] = new LineSegment(int.Parse(str[0]), int.Parse(str[1]));
        }

        DotsFiller dotFiller = new DotsFiller(lines);
        List<int> dots = dotFiller.Fill();

        Console.WriteLine(dots.Count);
        for (int i = 0; i < dots.Count; i++)
            Console.Write("{0} ", dots[i]);
    }
}
}
```

2. Непрервный рюкзак

Первая строка содержит количество предметов $1 \le n \le 10^3$ и вместимость рюкзака $0 \le W \le 2 \cdot 10^6$. Каждая из следующих n строк задаёт стоимость $0 \le c_i \le 2 \cdot 10^6$ и объём $0 < w_i \le 2 \cdot 10^6$ предмета (n, W, ci, wi — целые числа). Выведите максимальную стоимость частей предметов (от каждого предмета можно отделить любую часть, стоимость и объём при этом пропорционально уменьшатся), помещающихся в данный рюкзак, с точностью не менее трёх знаков после запятой.

Исходный код (C#):

```
using System;
using System.Collections.Generic;
using System.IO;

namespace Stepik.GreedyAlgorithms
{
    public struct Item
    {
        public int Cost { get; }
        public int Size { get; }

        public double Value { get { return (float)Cost / Size; } }

        public Item(int cost, int size)
        {
            Cost = cost;
            Size = size;
        }
    }

    public class Backpack
```

```
{
    private Item[] items;
    private int maxSize;

    public Backpack(int maxSize, Item[] items)
    {
        this.items = items;
        this.maxSize = maxSize;
    }

    public double Calculate()
    {
        Quicksort(0, items.Length - 1);

        int size = maxSize;
        double cost = 0f;

        for (int i = items.Length - 1; i >= 0 && size > 0; i--)
        {
            if (items[i].Size > size)
            {
                cost += size * items[i].Value;
                break;
            }
            else
            {
                cost += items[i].Cost;
                size -= items[i].Size;
            }
        }

        return cost;
    }

    private int Partition (int start, int end)
    {
        Item temp;
        int marker = start;
        for ( int i = start; i <= end; i++ )
        {
            if (items[i].Value < items[end].Value)
            {
                temp = items[marker];
                items[marker] = items[i];
                items[i] = temp;
                marker += 1;
            }
        }

        temp = items[marker];
        items[marker] = items[end];
        items[end] = temp;
        return marker;
    }

    private void Quicksort(int start, int end)
    {
        if ( start >= end )
        {
            return;
        }
        int pivot = Partition(start, end);
        Quicksort(start, pivot-1);
        Quicksort(pivot+1, end);
```

```
        }
    }

    public class BackpackTask
    {
        static void Main(string[] args)
        {
            string[] str = Console.ReadLine().Split(' ');
            int n = int.Parse(str[0]);
            int size = int.Parse(str[1]);

            Item[] items = new Item[n];

            for (int i = 0; i < n; i++)
            {
                str = Console.ReadLine().Split(' ');
                items[i] = new Item(int.Parse(str[0]), int.Parse(str[1]));
            }

            Backpack backpack = new Backpack(size, items);
            Console.WriteLine(String.Format("{0:0.000}", backpack.Calculate()));
        }
    }
}
```

3. Различные слагаемые

По данному числу $1 \le n \le 10^9$ найдите максимальное число k, для которого nn можно представить как сумму k различных натуральных слагаемых. Выведите в первой строке число k, во второй — k слагаемых.

Исходный код (C#):

```
using System;
using System.Collections.Generic;
using System.IO;

namespace Stepik.GreedyAlgorithms
{
    public class Number
    {
        private int number;

        public Number(int number)
        {
            this.number = number;
        }

        public List<int> GetSum()
        {
            int sumLeft = number;
            List<int> sum = new List<int>();
            int currentNumber = 1;

            while (sumLeft > 0)
            {
                if (currentNumber * 2 < sumLeft)
                {
                    sumLeft -= currentNumber;
                    sum.Add(currentNumber);
                    currentNumber++;
                }
                else
```

```
                {
                    sum.Add(sumLeft);
                    sumLeft = 0;
                }
            }
            return sum;
        }
    }
    public class NumberSumTask
    {
        static void Main(string[] args)
        {
            Number number = new Number(int.Parse(Console.ReadLine()));

            List<int> sum = number.GetSum();
            Console.WriteLine(sum.Count);
            for (int i = 0; i < sum.Count; i++)
                Console.Write("{0} ", sum[i]);
        }
    }
}
```

4. Кодирование Хаффмана

По данной непустой строке ss длины не более 10^4, состоящей из строчных букв латинского алфавита, постройте оптимальный беспрефиксный код. В первой строке выведите количество различных букв k, встречающихся в строке, и размер получившейся закодированной строки. В следующих k строках запишите коды букв в формате "letter: code". В последней строке выведите закодированную строку.

Исходный код (C#):

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;

namespace Stepik.GreedyAlgorithms
{
    public class TreeBranch
    {
        public List<LetterCode> Letters { get; set; }
        public int Count { get; set; }

        public TreeBranch(List<LetterCode> lettersCode, int count)
        {
            Letters = lettersCode;
            Count = count;
        }

        public void AddPrefix(char c)
        {
            for (int i = 0; i < Letters.Count; i++)
                Letters[i] = new LetterCode(Letters[i].Letter, c + Letters[i].Code);
        }
    }

    public struct LetterCode
    {
        public char Letter { get; }
        public string Code { get; }
```

```
    public LetterCode(char letter, string code)
    {
        Letter = letter;
        Code = code;
    }
}

public class Huffman
{
    private string str;

    public string CodedString { get; private set; }
    public List<LetterCode> LettersCodes { get; private set; }

    public Huffman(string str)
    {
        this.str = str;
    }

    public void CodeString()
    {
        LettersCodes = GetCodes();
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < str.Length; i++)
            stringBuilder.Append(GetCode(str[i]));
        CodedString = stringBuilder.ToString();
    }

    private string GetCode(char c)
    {
        for (int i = 0; i < LettersCodes.Count; i++)
            if (LettersCodes[i].Letter == c)
                return LettersCodes[i].Code;
        return String.Empty;
    }

    private List<LetterCode> GetCodes()
    {
        List<TreeBranch> branches = CountLetters();
        if (branches.Count == 1)
            return new List<LetterCode>() { new LetterCode(branches[0].Letters[0].Letter, "0") };

        while (branches.Count > 1)
        {
            Sort(branches);
            TreeBranch left = branches[1];
            TreeBranch right = branches[0];

            left.AddPrefix('0');
            right.AddPrefix('1');

            List<LetterCode> letters = new List<LetterCode>(left.Letters);
            letters.AddRange(right.Letters);

            branches.RemoveAt(1);
            branches[0] = new TreeBranch(letters, left.Count + right.Count);
        }
        return branches[0].Letters;
    }
    private List<TreeBranch> CountLetters()
    {
        int[] count = new int[65536];
        for (int i = 0; i < str.Length; i++)
            count[str[i] - 'a']++;
```

```
            List<TreeBranch> lettersCount = new List<TreeBranch>();
            for (int i = 0; i <= 'z' - 'a'; i++)
                if (count[i] > 0)
                    lettersCount.Add(new TreeBranch( new List<LetterCode>() { new LetterCode((char)('a' +
i), "") }, count[i]));

            return lettersCount;
        }
        private void Sort(List<TreeBranch> brenches)
        {
                        TreeBranch t;
            for (int p = 0; p <= brenches.Count - 2; p++)
            {
                for (int i = 0; i <= brenches.Count - 2; i++)
                {
                    if (brenches[i].Count > brenches[i + 1].Count)
                    {
                        t = brenches[i + 1];
                        brenches[i + 1] = brenches[i];
                        brenches[i] = t;
                    }
                }
            }
        }
    }

    public class HuffmanTask
    {
        static void Main(string[] args)
        {
            Huffman huffman = new Huffman(Console.ReadLine());
            huffman.CodeString();

            Console.WriteLine("{0} {1}", huffman.LettersCodes.Count, huffman.CodedString.Length);
            foreach (LetterCode letterCode in huffman.LettersCodes)
                Console.WriteLine("{0}: {1}", letterCode.Letter, letterCode.Code);
            Console.WriteLine(huffman.CodedString);
        }
    }
}
```

1. Декодирование Хаффмана

Восстановите строку по её коду и беспрефиксному коду символов.

В первой строке входного файла заданы два целых числа k и l через пробел — количество различных букв, встречающихся в строке, и размер получившейся закодированной строки, соответственно. В следующих k строках записаны коды букв в формате "letter: code". Ни один код не является префиксом другого. Буквы могут быть перечислены в любом порядке. В качестве букв могут встречаться лишь строчные буквы латинского алфавита; каждая из этих букв встречается в строке хотя бы один раз. Наконец, в последней строке записана закодированная строка. Исходная строка и коды всех букв непусты. Заданный код таков, что закодированная строка имеет минимальный возможный размер.

В первой строке выходного файла выведите строку s. Она должна состоять из строчных букв латинского алфавита. Гарантируется, что длина правильного ответа не превосходит 10^4 символов.

Исходный код (C#):

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;

namespace Stepik.GreedyAlgorithms
{
    public class Node
    {
        public Node Left { get; set; }
        public Node Right { get; set; }

        public char Code { get; set; }
        public char Letter { get; set; }

        public void AddLetter(char letter, string code)
        {
            Code = code[0];
            if (code.Length > 1)
            {
                code = code.Substring(1);
                if (code[0] == '0')
                {
                    if (Left == null)
                        Left = new Node();
                    Left.AddLetter(letter, code);
                }
                else if (code[0] == '1')
                {
                    if (Right == null)
                        Right = new Node();
                    Right.AddLetter(letter, code);
                }
            }
            else
                Letter = letter;
        }
        public char FindLetter(string code, int index, out int resultIndex)
        {
            if (Letter != 0)
            {
                resultIndex = index - 1;
                return Letter;
            }

            if (Left != null && Left.Code == code[index])
            {
                return Left.FindLetter(code, index + 1, out resultIndex);
            }
            else
            {
                return Right.FindLetter(code, index + 1, out resultIndex);
            }
        }
    }
    public class ReverseHuffmanTask
    {
        static void Main(string[] args)
        {
            string[] str = Console.ReadLine().Split(' ');
            int n = int.Parse(str[0]);
            Node root = new Node();
```

```
            for (int i = 0; i < n; i++)
            {
                str = Console.ReadLine().Split(' ');
                root.AddLetter(str[0][0], " " + str[1]);
            }

            string code = Console.ReadLine();
            int index = 0;
            while (index < code.Length)
            {
                int resultIndex;
                Console.Write(root.FindLetter(code, index, out resultIndex));
                index = resultIndex + 1;
            }
        }
    }
}
```

6. Очередь с приоритетами

Первая строка входа содержит число операций $1 \le n \le 10^5$. Каждая из последующих nn строк задают операцию одного из следующих двух типов:

- InsertInsert xx, где $0 \le x \le 10^9$— целое число;
- ExtractMaxExtractMax.

Первая операция добавляет число xx в очередь с приоритетами, вторая — извлекает максимальное число и выводит его.

Исходный код (C#):

```
using System;
using System.IO;
using System.Collections.Generic;

namespace Stepik.GreedyAlgorithms
{
    public class PriorityQueue
    {
        public int Size { get { return elements.Count; } }
        private List<int> elements;

        public PriorityQueue(int baseSize = 1000000)
        {
            elements = new List<int>(baseSize);
        }

        public void Enqueue(int value)
        {
            elements.Add(value);
            LiftElement(Size - 1);
        }
        public int Dequeue()
        {
            int value = elements[0];
            Swap(0, Size - 1);
            elements.RemoveAt(Size - 1);
            Heapify(0);
            return value;
        }
        private void LiftElement(int index)
        {
            int parentIndex = (index + 1) / 2 - 1;
```

```
            while (index > 0 && elements[parentIndex] < elements[index])
            {
                Swap(index, parentIndex);
                index = parentIndex;
                parentIndex = (index + 1) / 2 - 1;
            }
        }

        private void Heapify(int index)
        {
            int leftIndex = (index + 1) * 2 - 1;
            int rightIndex = leftIndex + 1;
            int minIndex;
            if (leftIndex < elements.Count && elements[leftIndex] > elements[index])
                minIndex = leftIndex;
            else
                minIndex = index;
            if (rightIndex < elements.Count && elements[rightIndex] > elements[minIndex])
                minIndex = rightIndex;
            if (index != minIndex)
            {
                Swap(index, minIndex);
                Heapify(minIndex);
            }
        }

        private void Swap(int index1, int index2)
        {
            int buf = elements[index1];
            elements[index1] = elements[index2];
            elements[index2] = buf;
        }
    }
    public class PriorityQueueTask
    {
        public static void Main(string[] args)
        {
            int n = int.Parse(Console.ReadLine());
            PriorityQueue queue = new PriorityQueue(n);
            for (int i = 0; i < n; i++)
            {
                string[] cmd = Console.ReadLine().Split(' ');

                switch(cmd[0])
                {
                    case "Insert":
                        queue.Enqueue(int.Parse(cmd[1]));
                        break;
                    case "ExtractMax":
                        if (queue.Size > 0)
                            Console.WriteLine(queue.Dequeue());
                        else
                            Console.WriteLine('*');
                        break;
                }
            }
        }
    }
}
```