**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЯ ОБРАЗОВАНИЯ**

**“ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

Факультет информационных технологий

Кафедра технологий программирования

**Алгоритмы: теория и практика. Структуры данных**

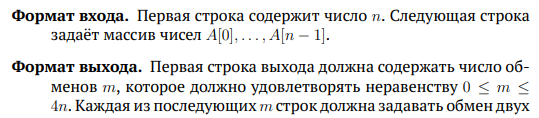
**Модуль №2**

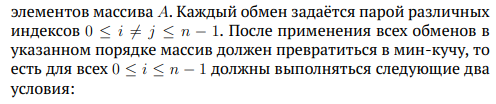
Выполнила студент 2 курса, группа 21-ИТ-1 Макеёнок Д.И.

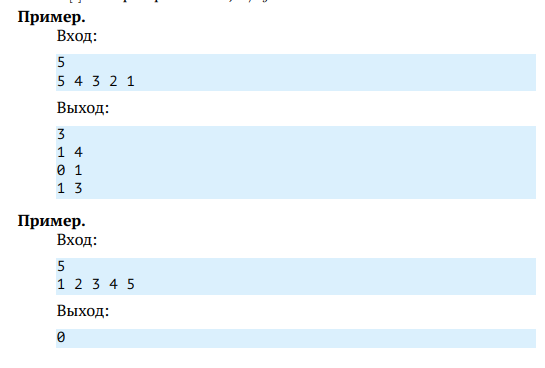
Проверила Виноградова А.Д

Полоцк, 2022 г.

Построение кучи







Решение

using System;

using System.Collections.Generic;

namespace HeapBuild

{

class Program

{

public static void Main(string[] args)

{

var size = Int32.Parse(Console.ReadLine());

//var size = 6;

var heapData = GetArray(Console.ReadLine());

//var heapData = GetArray("7 6 5 4 3 2");

var heap = new Heap(heapData, size);

Console.WriteLine(heap.countOfPermutation);

foreach (var x in heap.permutations)

Console.WriteLine("{0} ", x);

//Console.ReadKey();

}

private static int[] GetArray(string v)

{

var numbers = v.Split(' ');

var array = new int[numbers.Length];

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

array[i] = Int32.Parse(numbers[i]);

return array;

}

}

public class Heap

{

public int countOfPermutation = 0;

public int[] heap;

public List<string> permutations = new List<string>();

public Heap(int[] inputData, int sz)

{

if (inputData.Length == sz) this.Size = sz;

else throw new InvalidOperationException("Size missmatch");

heap = inputData;

BuildHeap(this.heap);

}

private void BuildHeap(int[] heap)

{

for (int i = (int)(this.heap.Length \* .5 - 1); i >= 0; i--)

{

#if DEBUG

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

#endif

SiftDown(i);

}

//throw new NotImplementedException();

}

public int Parent(int point)

{

return (int)(0.5 \* (point - 1));

}

public int LeftChild(int point)

{

return 2 \* point + 1;

}

public int RightChild(int point)

{

return 2 \* (point + 1);

}

int size;

int maxSize;

public int Size { get => size; set => size = value; }

public int MaxSize { get => maxSize; set => maxSize = value; }

public void SiftUp(int i)

{

if (i > 0 && heap[i] < heap[Parent(i)])

{

Swap(i, Parent(i));

SiftUp(Parent(i));

}

}

public void SiftDown(int i)

{

int iMin = i;

if (LeftChild(i) < heap.Length)

{

if (heap[i] < heap[LeftChild(i)]) iMin = i;

else iMin = LeftChild(i);

}

if (RightChild(i) < heap.Length)

{

if (heap[RightChild(i)] < heap[iMin]) iMin = RightChild(i);

//else iMin = RightChild(i);

}

if (i != iMin)

{

Swap(i, iMin);

SiftDown(iMin);

}

}

private void Swap(int index1, int index2)

{

int tmp = heap[index1];

heap[index1] = heap[index2];

heap[index2] = tmp;

permutations.Add(index1.ToString() + " " + index2.ToString());

this.countOfPermutation++;

#if DEBUG

Console.WriteLine(countOfPermutation);

Console.WriteLine("{0} {1}", index1, index2);

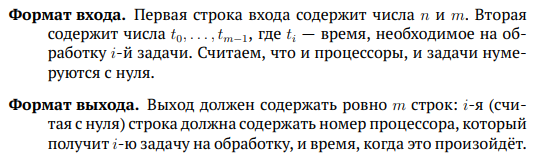
#endif

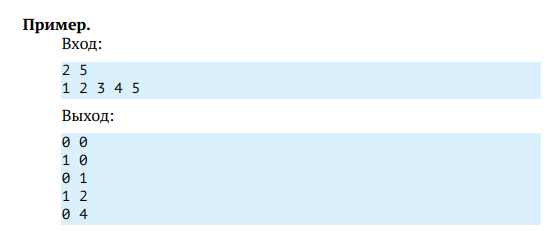
}

}

}

Параллельная обработка





Решение

using System;

namespace ParallelProcessing

{

class Program

{

static void Main(string[] args)

{

long time = 0;

var inp1 = GetArray(Console.ReadLine());

var size = inp1[1];

var numCores = inp1[0];

var tasks = GetArray(Console.ReadLine());

if (size != tasks.Length) throw new InvalidOperationException("Size missmatch");

var cores = new Heap(numCores);

var processes = new Heap(numCores);

for (int j = 0; j < numCores; j++)

cores.Insert(new long[] { j });

int i = 0;

long core;

while (i < tasks.Length)

{

if (processes.Full)

{

time = GetFromHeap(ref processes, ref cores);

}

while (!processes.Full && i < tasks.Length)

{

if (tasks[i] > 0)

{

core = cores.ExtractTop()[0];

processes.Insert(new long[] { tasks[i] + time, core });

}

else core = cores.Peek()[0];

Console.WriteLine("{0} {1}", core, time);

i++;

}

}

}

private static long GetFromHeap(ref Heap processes, ref Heap cores)

{

long finishedTask = processes.Peek()[0];

while (processes.Peek()[0] == finishedTask && processes.Size > 0)

{

var fromHeap = processes.ExtractTop();

long finishedCore = fromHeap[1];

cores.Insert(new long[] { finishedCore });

}

return finishedTask;

}

private static long[] GetArray(string v)

{

var numbers = v.Split(' ');

var array = new long[numbers.Length];

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

array[i] = long.Parse(numbers[i]);

return array;

}

}

public class Heap

{

private long[][] heap;

int size;

public int Size

{

get

{

return size;

}

}

public Heap(long size)

{

this.size = 0;

heap = new long[size][];

}

private int Parent(int point)

{

return (int)(0.5 \* (point - 1));

}

private int LeftChild(int point)

{

return 2 \* point + 1;

}

private int RightChild(int point)

{

return 2 \* (point + 1);

}

public void SiftUp(int i)

{

if (i > 0 && heap[i][0] < heap[Parent(i)][0])

{

Swap(i, Parent(i));

SiftUp(Parent(i));

}

}

public void SiftDown(int i)

{

int iMin = i;

if (LeftChild(i) < this.size)

{

if (heap[i][0] < heap[LeftChild(i)][0]) iMin = i;

else iMin = LeftChild(i);

}

if (RightChild(i) < this.size)

{

if (heap[RightChild(i)][0] < heap[iMin][0]) iMin = RightChild(i);

}

if (i != iMin)

{

Swap(i, iMin);

SiftDown(iMin);

}

}

public void Insert(long[] value)

{

if (this.size == heap.Length)

throw new InvalidOperationException("Heap overflow");

heap[this.size] = value;

SiftUp(this.size);

this.size++;

}

public long[] ExtractTop()

{

var result = heap[0];

size--;

heap[0] = heap[size];

SiftDown(0);

return result;

}

public long[] Peek()

{

return heap[0];

}

public bool Full

{

get

{

if (heap.Length == this.size) return true;

else return false;

}

}

private void Swap(int index1, int index2)

{

long[] tmp = heap[index1];

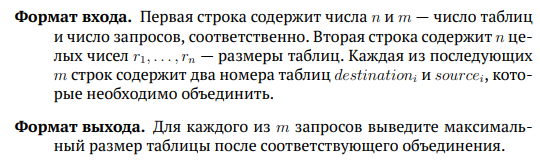
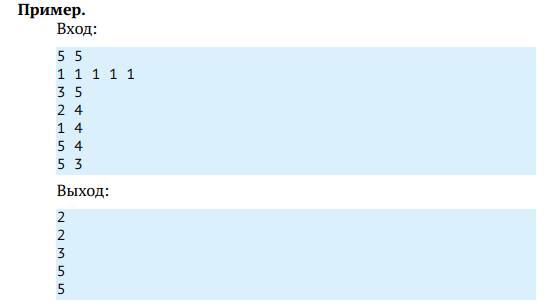
heap[index1] = heap[index2];

heap[index2] = tmp;

}

}

}

Объединение таблиц 

Решение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MergeTables

{

class Program

{

static void Main(string[] args)

{

var arg1 = GetArray(Console.ReadLine());

var arg2 = GetArray(Console.ReadLine());

if (arg2.Length != arg1[0]) throw new InvalidOperationException("Size missmatch");

var djset = new DisjointSet(arg2);

for (int i = 0; i < arg1[1]; i++)

{

var merge = GetArray(Console.ReadLine());

djset.Union(merge[0] - 1, merge[1] - 1);

Console.WriteLine(djset.Max);

}

}

private static int[] GetArray(string v)

{

var numbers = v.Split(' ');

var array = new int[numbers.Length];

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

array[i] = int.Parse(numbers[i]);

return array;

}

}

public class DisjointSet

{

public int Count { get; private set; }

private int[] Parent;

private int[] Rank;

public int Max;

public DisjointSet(int[] rank)

{

this.Rank = rank;

this.Count = Rank.Length;

this.Parent = new int[this.Count];

this.Max = int.MinValue;

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

{

this.Parent[i] = i;

this.Max = Math.Max(this.Max, Rank[i]);

}

}

public int Find(int i)

{

if (Parent[i] == i) return i;

else

{

int result = Find(Parent[i]);

Parent[i] = result;

return result;

}

}

public void Union(int destination, int source)

{

int dRep = this.Find(destination),

sRep = this.Find(source),

dRank = Rank[dRep],

sRank = Rank[sRep];

if (dRep == sRep)

return;

this.Parent[sRep] = dRep;

Rank[dRep] = dRank + sRank;

this.Max = Math.Max(this.Max, Rank[dRep]);

Rank[sRep] = 0;

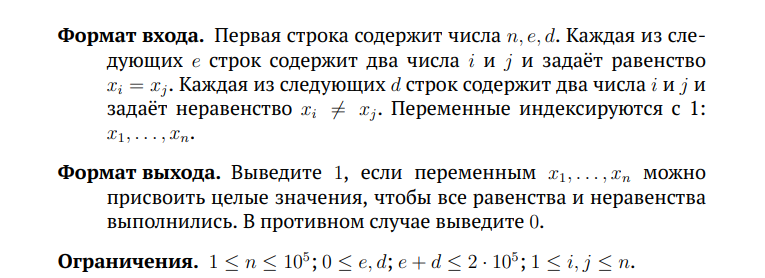
//this.Parent[dRep] = sRep;

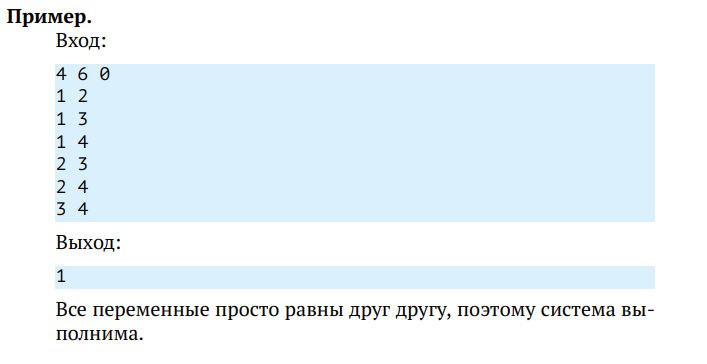
}

}

}

Автоматический анализ программ





Решение

# include <iostream>

# include <vector>

int main()

{

int n, e, d;

std::cin >> n >> e >> d;

std::vector<int> parent(n);

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

{

parent[i] = i;

}

int var1, var2;

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

{

std::cin >> var1 >> var2;

parent[--var2] = parent[--var1];

}

int result = 1;

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

{

std::cin >> var1 >> var2;

if (parent[--var1] == parent[--var2])

{

result = 0;

break;

}

}

std::cout << result << std::endl;

return 0;

}