J. P2P обновление

|  |  |
| --- | --- |
| Ограничение времени | 15 секунд |
| Ограничение памяти | 256Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

В системе умного дома под управлением голосового помощника Лариса *n* устройств, соединяющихся между собой по сети LoRaWAN. Устройство номер 1 подключено к интернету и на него было скачано обновление, которое необходимо передать на все устройства.

Сеть LoRaWAN очень медленная, поэтому для распространения протокола был придуман peer-to-peer (P2P) протокол. Файл обновления разбивается на *k* одинаковых по размеру частей, занумерованных от 1 до *k*.

Передача части обновления происходит во время таймслотов. Каждый таймслот занимает одну минуту. За один таймслот каждое устройство может получить и передать ровно одну часть обновления. То есть устройство во время таймслота может получать новую часть обновления и передавать уже имеющуюуся у него к началу таймслота часть обновления, или совершать только одно из этих действий, или вообще не осуществлять прием или передачу. После приема части обновления устройство может передавать эту часть обновления другим устройствам в следующих таймслотах.

Перед каждым таймслотом для каждой части обновления определяется, на скольких устройствах сети скачана эта часть. Каждое устройство выбирает отсутствующую на нем часть обновления, которая встречается в сети реже всего. Если таких частей несколько, то выбирается отсутствующая на устройстве часть обновления с наименьшим номером.

После этого устройство делает запрос выбранной части обновления у одного из устройств, на котором такая часть обновления уже скачана. Если таких устройств несколько — выбирается устройство, на котором скачано наименьшее количество частей обновления. Если и таких устройств оказалось несколько — выбирается устройство с минимальным номером.

После того, как все запросы отправлены, каждое устройство выбирает, чей запрос удовлетворить. Устройство *A* удовлетворяет тот запрос, который поступил от наиболее ценного для *A* устройства. Ценность устройства *B* для устройства *A* определяется как количество частей обновления, ранее полученных устройством *A* от устройства *B*. Если на устройство *A* пришло несколько запросов от одинаково ценных устройств, то удовлетворяется запрос того устройства, на котором меньше всего скачанных частей обновления. Если и таких запросов несколько, то среди них выбирается устройство с наименьшим номером.

Далее начинается новый таймслот. Устройства, чьи запросы удовлетворены, скачивают запрошенную часть обновления, а остальные не скачивают ничего.

Для каждого устройства определите, сколько таймслотов понадобится для скачивания всех частей обновления.

Формат ввода

Вводится два числа *n* и *k* (*2 ≤ n ≤ 100*, *1 ≤ k ≤ 200*).

Формат вывода

Выведите *n-1* число — количество таймслотов, необходимых для скачивания обновления на устройства с номерами от 2 до *n*.

Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 3 2 | 3 3 |

Примечания

Для удобства будем пользоваться обозначениями устройств буквами A, B, C (соответствует устройствам с номерами 1, 2 и 3). На устройстве A есть обе части обновления, а на устройствах B и C — ни одной.

Перед первым таймслотом для каждой части определяется количество устройств, на которых скачана каждая часть обновления: и 1 и 2 часть обновления присутствуют только на одном устройстве.

Устройства B и C выбирают самую редкую отсутствующую у них часть обновления с минимальным номером: самая редкая часть с минимальным номером — это часть 1. Она отсутствует и на устройстве B, и на устройстве С. Они запрашивают ее у устройства A. Ценность устройств B и C для устройства A равна нулю. Количество имеющихся у устройств B и C частей обновления одинакова и равно нулю. Поэтому устройство A выбирает устройство с минимальным номером (B). Во время первого таймслота выполняется передача части 1 с устройства A на устройство B. Ценность устройства A для устройства B становится равной 1.

Перед вторым таймслотом для каждой части определяется количество устройств, на которых скачана каждая часть обновления: самой редкой оказывается часть 2 (присутствует только на устройстве A), следующая по редкости часть 1 (присутствует на устройствах A и B).

Устройства B и C выбирают среди отсутствующих у них частей обновления самую редкую: для обоих устройств выбирается часть 2. Каждое из них делает запрос части 2 у единственного обладателя этой части — устройства A. Ценность устройств B и C для устройства A одинакова и равна нулю. Количество имеющихся у устройства C частей (0) меньше, чем у устройства B (1), поэтому выбирается устройство C. Во время второго таймслота выполняется передача части 2 с устройства A на устройство C. Ценность устройства A для устройства C становится равной 1.

Перед третьим таймслотом для каждой части определяется количество устройств, на которых скачана каждая часть обновления: обе части 1 и 2 присутствуют на двух устройствах (часть 1 на устройствах A и B, часть 2 — на устройствах A и C)

Устройство B может сделать запрос недостающей части 2 у обладающей ей устройств A и C, но выбирает устройство C, т.к. на устройстве C скачано меньше частей (1), чем у устройства A (2).

Устройство C может сделать запрос недостающей части 1 у обладающей ей устройств A и B, но выбирает устройство B, т.к. на устройстве B скачано меньше частей (1), чем у устройства A (2).

Во время третьего таймслота оба запроса оказываются единственными запросами у устройств B и C и удовлетворяются. Часть 2 передается с устройства C на устройство B. Часть 1 передается с устройства B на устройство C. Ценность устройства B для устройства C становится равной 1. Ценность устройства C для устройства B становится равной 1.

Все части обновления оказываются на всех устройствах и на этом обновление заканчивается.

using System;

using System.IO;

using System.Collections.Generic;

using System.Linq;

class Device

{

public int Name;

private Dictionary<int, int> value = new Dictionary<int, int>();

public int update\_counter = 0;

public bool[] parts;

public int

iterations = 0;

public int current\_needed;

public Device(int name, int k)

{

Name = name;

parts = new bool[k+1];

if (name == 1)

{

for (int i = 1; i < k + 1; i++)

{

parts[i] = true;

}

update\_counter = k;

iterations = 1;

}

}

public void Check\_updates()

{

int result = 0;

for (int i = 1; i < parts.Length; i++)

{

if (parts[i]) result++;

}

update\_counter = result;

}

public void Find\_current\_needed(int[] updates, int timeset)

{

int result = 0;

int min = 300;

for (int i = 1; i < parts.Length; i++)

{

if (parts[i] == false && updates[i] < min)

{

result = i;

min = updates[i];

}

}

current\_needed = result;

if (current\_needed == 0 && iterations == 0)

{

iterations = timeset;

}

}

public int Find\_send(Device[] devices)

{

int sender = 0;

int min = 300;

for (int i = 1; i < devices.Length; i++)

{

if (devices[i].parts[current\_needed] && devices[i].update\_counter < min)

{

min = devices[i].update\_counter;

sender = i;

}

}

return sender;

}

public void Send\_update(Device[] devices, Dictionary<int, HashSet<int>> sended\_devices, Dictionary<int, Sended\_update> sendedUpdates)

{

if (sended\_devices.ContainsKey(Name))

{

int max\_val = -1;

int min\_up = 300;

Device device\_sended = this;

Device devise\_taking = devices[0];

foreach (var sended\_dev in sended\_devices[Name])

{

int current\_value;

if (value.ContainsKey(sended\_dev)) current\_value = value[sended\_dev];

else current\_value = 0;

if (current\_value > max\_val)

{

devise\_taking = devices[sended\_dev];

max\_val = current\_value;

min\_up = devices[sended\_dev].update\_counter;

}

else if (current\_value == max\_val)

{

if (devices[sended\_dev].update\_counter < min\_up)

{

devise\_taking = devices[sended\_dev];

min\_up = devices[sended\_dev].update\_counter;

}

}

}

sendedUpdates.Add(devise\_taking.Name, new Sended\_update(devise\_taking, device\_sended, devise\_taking.current\_needed));

}

}

public void Take\_update(Sended\_update sendedUpdate)

{

if (value.ContainsKey(sendedUpdate.device\_sended.Name)) value[sendedUpdate.device\_sended.Name]++;

else value.Add(sendedUpdate.device\_sended.Name, 1);

parts[sendedUpdate.update] = true;

update\_counter++;

}

}

class Sended\_update

{

public Device devise\_taking;

public Device device\_sended;

public int update;

public Sended\_update(Device devise\_taking1, Device device\_sended1, int update1)

{

devise\_taking = devise\_taking1;

device\_sended = device\_sended1;

update = update1;

}

}

class Program

{

static void Main()

{

bool CheckFullUpdate(int[] updates, int N)

{

for (int i = 1; i < updates.Length; i++)

{

if (updates[i] != N) return false;

}

return true;

}

string[] input = Console.ReadLine().Split();

int N = int.Parse(input[0]);

int k = int.Parse(input[1]);

int[] updates = new int[k+1]; // счетчик всех обновлений по всем девасам

for (int i = 1; i < updates.Length; i++)

{

updates[i] = 1;

}

int time\_slot = 0; // счетчик таймслотов

Device[] devices = new Device[N + 1]; // заполнение всех девайсов

devices[1] = new Device(1, k);

for (int i = 2; i < N + 1; i++)

{

devices[i] = new Device(i, k);

}

while (!CheckFullUpdate(updates, N))

{

time\_slot++;

Dictionary<int, HashSet<int>> sended\_devices = new Dictionary<int, HashSet<int>>(); // кто кого просит отправить

Dictionary<int, Sended\_update> sendedUpdates = new Dictionary<int, Sended\_update>();

for (int i = 1; i < devices.Length; i++) // поиск нужной части обновления

{

if (devices[i].iterations == 0)

{

devices[i].Find\_current\_needed(updates, time\_slot-1);

}

}

for (int i = 1; i < devices.Length; i++) // поиск устройства на которое надо отправить

{

if (devices[i].iterations == 0)

{

int device\_to\_send = devices[i].Find\_send(devices);

if (sended\_devices.ContainsKey(device\_to\_send)) sended\_devices[device\_to\_send].Add(i);

else sended\_devices.Add(device\_to\_send, new HashSet<int>(){i});

}

}

for (int i = 1; i < devices.Length; i++) // отправка обновления с устройства

{

devices[i].Send\_update(devices, sended\_devices,sendedUpdates);

}

foreach (var pair in sendedUpdates)

{

devices[pair.Key].Take\_update(pair.Value);

updates[pair.Value.update]++;

}

}

for (int i = 1; i < devices.Length; i++) // поиск нужной части обновления

{

if (devices[i].iterations == 0)

{

devices[i].Find\_current\_needed(updates, time\_slot);

}

}

for (int i = 2; i < N + 1; i++)

{

Console.Write(devices[i].iterations + " ");

}

}

}