Отчет по дисциплине «Структуры и алгоритмы обработки данных»

Студента [скрыто], группы [скрыто]

Часть 2

Оглавление

[1 Базовые структуры данных 3](#_Toc158548193)

[1.2.1 Расстановка скобок в коде 3](#_Toc158548194)

[1.2.2 Высота дерева 4](#_Toc158548195)

[1.2.3 Симуляция обработки сетевых пакетов 4](#_Toc158548196)

[1.2.4 Стек с поддержкой максимума 6](#_Toc158548197)

[1.2.5 Максимум в скользящем окне 8](#_Toc158548198)

[2 Очередь с приоритетом и система непересекающихся множеств 12](#_Toc158548199)

[2.3.1 Построение кучи 12](#_Toc158548200)

[2.3.2 Параллельная обработка 12](#_Toc158548201)

[2.3.4 Объединение таблиц 14](#_Toc158548202)

[2.3.5 Автоматический анализ программ 20](#_Toc158548203)

[Сертификат 26](#_Toc158548204)

[[скрыто, сертификат 53%] 26](#_Toc158548205)

# 1 Базовые структуры данных

## 1.2.1 Расстановка скобок в коде

**Решение:**

#include <iostream>

#include <iterator>

#include <stack>

#include <string>

int get\_bracket\_type**(**const char symbol**)**

**{**

**switch(**symbol**)** **{**

**case** '('**:**

**return** 1**;**

**case** '['**:**

**return** 2**;**

**case** '{'**:**

**return** 3**;**

**case** ')'**:**

**return** **-**1**;**

**case** ']'**:**

**return** **-**2**;**

**case** '}'**:**

**return** **-**3**;**

**default:**

**return** 0**;**

**}**

**}**

template **<**typename InputIt**>** InputIt check\_brackets**(**InputIt first**,** InputIt last**)**

**{**

std**::**stack**<**int**>** stack**;**

InputIt first\_open**;**

**for(**auto it **=** first**;** it **!=** last**;** **++**it**)** **{**

const int bk **=** get\_bracket\_type**(\***it**);**

**if(**bk **>** 0**)** **{**

**if(**stack**.**empty**())**

first\_open **=** it**;**

stack**.**push**(**bk**);**

**}** **else** **if(**bk **<** 0**)** **{**

**if(!**stack**.**empty**()** **&&** stack**.**top**()** **==** **-**bk**)** **{**

stack**.**pop**();**

**}** **else**

**return** it**;**

**}**

**}**

**if(!**stack**.**empty**())**

**return** first\_open**;**

**return** last**;**

**}**

int main**()**

**{**

const std**::**string input**(**std**::**istreambuf\_iterator**<**char**>(**std**::**cin**),**std**::**istreambuf\_iterator**<**char**>());**

auto end**=**input**.**cend**();**

auto it **=** check\_brackets**(**input**.**cbegin**(),** end**);**

**if(**it **!=** end**)**

std**::**cout **<<** std**::**distance**(**input**.**cbegin**(),** it**)** **+** 1**;**

**else**

std**::**cout **<<** "Success"**;**

**return** 0**;**

**}**

## 1.2.2 Высота дерева

**Решение**

#include <iostream>

#include <unordered\_set>

#include <vector>

#include <iterator>

**using** ind\_t **=** std**::**size\_t**;**

**using** len\_t **=** std**::**size\_t**;**

template **<**typename InputIt**>** len\_t tree\_height**(**InputIt first**,** InputIt last**,** ind\_t count**)**

**{**

constexpr ind\_t root\_mark**=static\_cast<**ind\_t**>(-**1**);**

struct node **{**ind\_t child\_ind**;**len\_t path**=**0**;};**

std**::**vector**<**node**>** parents**(**count**);**

std**::**unordered\_set**<**ind\_t**>** parents\_set**;**

node**\*** root**;**

**for** **(**auto**&** parent**:**parents**){**

parent**.**child\_ind**=\***first**;**

parents\_set**.**insert**(\***first**);**

**if(\***first**++** **==** root\_mark**)**

root **=** **&**parent**;**

**}**

**for(**ind\_t i **=** 0**;** i **!=** parents**.**size**();** **++**i**)** **{**

**if(**parents\_set**.**find**(**i**)==**parents\_set**.**end**())** **{**

ind\_t ind **=** i**;**

len\_t path **=** 1**;**

**while(**ind **!=** root\_mark **&&** parents**[**ind**].**path **<** path**)** **{**

parents**[**ind**].**path **=** path**++;**

ind **=** parents**[**ind**].**child\_ind**;**

**}**

**}**

**}**

**return** root**->**path**;**

**}**

int main**()**

**{**

ind\_t count**;**

std**::**cin**>>**count**;**

std**::**cout **<<** tree\_height**(**std**::**istream\_iterator**<**ind\_t**>(**std**::**cin**),**std**::**istream\_iterator**<**ind\_t**>(),**count**);**

**return** 0**;**

**}**

## 1.2.3 Симуляция обработки сетевых пакетов

**Решение:**

#include <cstdint>

#include <iostream>

#include <queue>

#include <iterator>

template **<**typename TimeType **=** unsigned long**,**

TimeType PackIgnored **=** **(**TimeType**)(-**1**),**

typename Queue **=** std**::**queue**<**TimeType**>>**

class network\_pack\_simulator

**{**

public**:**

**using** time\_type **=** TimeType**;**

**using** size\_type**=** typename Queue**::**size\_type**;**

**using** queue\_type **=** Queue**;**

constexpr static TimeType pack\_ignored **=** PackIgnored**;**

protected**:**

Queue queue**;**

time\_type past\_time **=** 0**;**

public**:**

const size\_type queue\_size**;**

network\_pack\_simulator**(**const size\_type queue\_size\_**)**

**:** queue\_size**(**queue\_size\_**){}**

time\_type simulate\_pack**(**const time\_type arrival**,** const time\_type duraction**)**

**{**

**while(!**queue**.**empty**()** **&&** arrival **>=** queue**.**front**())**

queue**.**pop**();**

**if(**arrival **>** past\_time**)**

past\_time **=** arrival**;**

**if(**queue**.**size**()** **<** queue\_size**)** **{**

time\_type start\_tm **=** past\_time**;**

past\_time **+=** duraction**;**

queue**.**push**(**past\_time**);**

**return** start\_tm**;**

**}**

**return** pack\_ignored**;**

**}**

void reset**(**const size\_type queue\_size\_**)**

**{**

past\_time **=** 0**;**

queue\_size**=**queue\_size\_**;**

**while(!**queue**.**empty**())** **{**

queue**.**pop**();**

**}**

**}**

**};**

**using** num\_t **=** std**::**uint\_fast32\_t**;**

struct pack **{**

num\_t arrival**,** duraction**;**

**};**

std**::**istream**&** **operator>>** **(**std**::**istream**&** is**,** pack**&** pk**)**

**{**

is**>>**pk**.**arrival**>>**pk**.**duraction**;**

**return** is**;**

**}**

int main**()**

**{**

**using** simulator\_t**=**network\_pack\_simulator**<**num\_t**>;**

simulator\_t**::**size\_type buf\_size**,**size**;**

std**::**cin**>>**buf\_size**>>**size**;**

simulator\_t simulator**(**buf\_size**);**

auto it**=**std**::**istream\_iterator**<**pack**>(**std**::**cin**);**

**for** **(**decltype**(**size**)** i **=** 0**;** i **<** size**;** **++**it**,** **++**i**)** **{**

simulator\_t**::**time\_type tm**=**simulator**.**simulate\_pack**(**it**->**arrival**,**it**->**duraction**);**

**if** **(**tm**!=**simulator\_t**::**pack\_ignored**)**

std**::**cout**<<**tm**<<**std**::**endl**;**

**else**

std**::**cout**<<**"-1"**<<**std**::**endl**;**

**}**

**}**

## 1.2.4 Стек с поддержкой максимума

**Решение:**

#include <cstddef>

#include <iostream>

#include <memory>

#include <stack>

#include <string>

#include <type\_traits>

#include <utility>

template **<**typename Elem**,** typename Container **=** std**::**stack**<**std**::**pair**<**Elem**,** const Elem**\*>>>** class max\_stack

**{**

public**:**

**using** size\_type **=** typename Container**::**size\_type**;**

**using** container\_type **=** Container**;**

**using** value\_type **=** Elem**;**

**using** reference **=** Elem**&;**

**using** const\_reference **=** const Elem**&;**

protected**:**

Container container**;**

template **<**typename Alloc**>**

**using** uses\_alloc **=** typename std**::**enable\_if**<**std**::**uses\_allocator**<**Container**,** Alloc**>::**value**>::**type**;**

public**:**

template **<**typename Alloc**,** typename **=** uses\_alloc**<**Alloc**>>**

max\_stack**(**const max\_stack**&** src**,** const Alloc**&** alloc**)**

**:** container**(**src**.**container**,** alloc**){}**

template **<**typename Alloc**,** typename **=** uses\_alloc**<**Alloc**>>**

max\_stack**(**max\_stack**&&** src**,** const Alloc**&** alloc**)**

**:** container**(**std**::**move**(**src**.**container**),** alloc**){}**

template **<**typename **=** typename std**::**enable\_if**<**std**::**is\_default\_constructible**<**Elem**>::**value**>::**type**>**

max\_stack**()** **:** container**(){}**

bool empty**()** const

**{**

**return** container**.**empty**();**

**}**

size\_type size**()** const

**{**

**return** container**.**size**();**

**}**

reference top**()**

**{**

**return** container**.**back**().**first**;**

**}**

const\_reference top**()** const

**{**

**return** container**.**back**().**first**;**

**}**

const\_reference max**()** const

**{**

**return** **\***container**.**top**().**second**;**

**}**

void push**(**const value\_type**&** item**)**

**{**

**if(!**empty**())** **{**

const Elem**\*** cur **=** **&**max**();**

container**.**push**(**std**::**make\_pair**(**item**,** **nullptr));**

**if(\***cur **<** item**)**

cur **=** **&**container**.**top**().**first**;**

container**.**top**().**second **=** cur**;**

**}** **else** **{**

container**.**push**(**std**::**make\_pair**(**item**,** **nullptr));**

container**.**top**().**second **=** **&**container**.**top**().**first**;**

**}**

**}**

void push**(**value\_type**&&** item**)**

**{**

**if(!**empty**())** **{**

const Elem**\*** cur **=** **&**max**();**

container**.**push**(**std**::**make\_pair**(**std**::**move**(**item**),** **nullptr));**

const Elem**&** pushed **=** container**.**top**().**first**;**

**if(\***cur **<** pushed**)**

cur **=** **&**pushed**;**

container**.**top**().**second **=** cur**;**

**}** **else** **{**

container**.**push**(**std**::**make\_pair**(**std**::**move**(**item**),** **nullptr));**

container**.**top**().**second **=** **&**container**.**top**().**first**;**

**}**

**}**

void pop**()**

**{**

container**.**pop**();**

**}**

**};**

**using** num\_t **=** std**::**uint\_least32\_t**;**

int main**()**

**{**

num\_t count**;**

std**::**cin **>>** count**;**

max\_stack**<**num\_t**>** stack**;**

std**::**string str**;**

**while(**count**--)** **{**

std**::**cin **>>** str**;**

**if(**str **==** "push"**)** **{**

num\_t num**;** std**::**cin **>>** num**;**

stack**.**push**(**num**);**

**}** **else** **if(**str **==** "pop"**)**

stack**.**pop**();**

**else** **if(**str **==** "max"**)**

std**::**cout **<<** stack**.**max**()<<**std**::**endl**;**

**}**

**return** 0**;**

**}**

## 1.2.5 Максимум в скользящем окне

**Решение:**

#include <algorithm>

#include <cstddef>

#include <cstdint>

#include <deque>

#include <stack>

#include <iostream>

#include <memory>

#include <type\_traits>

#include <utility>

#include <iterator>

#include <vector>

template **<**typename Elem**,** typename Container **=** std**::**stack**<**std**::**pair**<**Elem**,** const Elem**\*>>>** class max\_stack

**{**

public**:**

**using** size\_type **=** typename Container**::**size\_type**;**

**using** container\_type **=** Container**;**

**using** value\_type **=** Elem**;**

**using** reference **=** Elem**&;**

**using** const\_reference **=** const Elem**&;**

protected**:**

Container container**;**

template **<**typename Alloc**>**

**using** uses\_alloc **=** typename std**::**enable\_if**<**std**::**uses\_allocator**<**Container**,** Alloc**>::**value**>::**type**;**

public**:**

template **<**typename Alloc**,** typename **=** uses\_alloc**<**Alloc**>>**

max\_stack**(**const max\_stack**&** src**,** const Alloc**&** alloc**)**

**:** container**(**src**.**container**,** alloc**)**

**{**

**}**

template **<**typename Alloc**,** typename **=** uses\_alloc**<**Alloc**>>**

max\_stack**(**max\_stack**&&** src**,** const Alloc**&** alloc**)**

**:** container**(**std**::**move**(**src**.**container**),** alloc**)**

**{**

**}**

template **<**typename **=** typename std**::**enable\_if**<**std**::**is\_default\_constructible**<**Elem**>::**value**>::**type**>**

max\_stack**()**

**:** container**()**

**{**

**}**

bool empty**()** const

**{**

**return** container**.**empty**();**

**}**

size\_type size**()** const

**{**

**return** container**.**size**();**

**}**

reference top**()**

**{**

**return** container**.**top**().**first**;**

**}**

const\_reference top**()** const

**{**

**return** container**.**top**().**first**;**

**}**

const\_reference max**()** const

**{**

**return** **\***container**.**top**().**second**;**

**}**

void push**(**const value\_type**&** item**)**

**{**

**if(!**empty**())** **{**

const Elem**\*** cur **=** **&**max**();**

container**.**push**(**std**::**make\_pair**(**item**,** **nullptr));**

**if(\***cur **<** item**)**

cur **=** **&**container**.**top**().**first**;**

container**.**top**().**second **=** cur**;**

**}** **else** **{**

container**.**push**(**std**::**make\_pair**(**item**,** **nullptr));**

container**.**top**().**second **=** **&**container**.**top**().**first**;**

**}**

**}**

void push**(**value\_type**&&** item**)**

**{**

**if(!**empty**())** **{**

const Elem**\*** cur **=** **&**max**();**

container**.**push**(**std**::**make\_pair**(**std**::**move**(**item**),** **nullptr));**

const Elem**&** pushed **=** container**.**top**().**first**;**

**if(\***cur **<** pushed**)**

cur **=** **&**pushed**;**

container**.**top**().**second **=** cur**;**

**}** **else** **{**

container**.**push**(**std**::**make\_pair**(**std**::**move**(**item**),** **nullptr));**

container**.**top**().**second **=** **&**container**.**top**().**first**;**

**}**

**}**

void pop**()**

**{**

container**.**pop**();**

**}**

**};**

template **<**typename Elem**,** typename PushContainer **=** std**::**deque**<**Elem**>,** typename PopContainer **=** max\_stack**<**Elem**>>**

class max\_queue

**{**

public**:**

**static\_assert(**std**::**is\_same**<**typename PushContainer**::**size\_type**,** typename PopContainer**::**size\_type**>::**value**,**

"size\_type of both containers should be the same"**);**

**using** size\_type **=** typename PushContainer**::**size\_type**;**

**using** push\_container\_type **=** PushContainer**;**

**using** pop\_container\_type **=** PopContainer**;**

**using** value\_type **=** Elem**;**

**using** reference **=** Elem**&;**

**using** const\_reference **=** const Elem**&;**

protected**:**

PushContainer push\_container**;**

PopContainer pop\_container**;**

value\_type**\*** push\_max **=** **nullptr;**

public**:**

template **<**typename **=** typename std**::**enable\_if**<**std**::**is\_default\_constructible**<**Elem**>::**value**>::**type**>**

max\_queue**()**

**:** push\_container**()**

**,** pop\_container**()**

**{**

**}**

bool empty**()** const

**{**

**return** push\_container**.**empty**()** **&&** pop\_container**.**empty**();**

**}**

size\_type size**()** const

**{**

**return** push\_container**.**size**()** **+** pop\_container**.**size**();**

**}**

reference front**()**

**{**

**return** **!**pop\_container**.**empty**()** **?** pop\_container**.**top**()** **:** push\_container**.**front**();**

**}**

const\_reference front**()** const

**{**

**return** **!**pop\_container**.**empty**()** **?** pop\_container**.**top**()** **:** push\_container**.**front**();**

**}**

const\_reference max**()** const

**{**

const bool pop\_container\_empty **=** pop\_container**.**empty**();**

**if(!**push\_container**.**empty**()** **&&** **!**pop\_container\_empty**)**

**return** std**::**max**(\***push\_max**,** pop\_container**.**max**());**

**return** **!**pop\_container\_empty **?** pop\_container**.**max**()** **:** **\***push\_max**;**

**}**

void push\_back**(**const value\_type**&** item**)**

**{**

const bool upd\_max **=** push\_container**.**empty**()** **||** **\***push\_max **<** item**;**

push\_container**.**push\_back**(**item**);**

**if(**upd\_max**)**

push\_max **=** **&**push\_container**.**back**();**

**}**

void push\_back**(**value\_type**&&** item**)**

**{**

const bool upd\_max **=** push\_container**.**empty**()** **||** **\***push\_max **<** item**;**

push\_container**.**push\_back**(**std**::**move**(**item**));**

**if(**upd\_max**)**

push\_max **=** **&**push\_container**.**back**();**

**}**

void pop\_front**()**

**{**

**if(**pop\_container**.**empty**())** **{**

**while(!**push\_container**.**empty**())** **{**

pop\_container**.**push**(**push\_container**.**back**());**

push\_container**.**pop\_back**();**

**}**

**}**

pop\_container**.**pop**();**

**}**

**};**

template **<**typename InputIt**,** typename Size**,** typename Callback**,** typename MaxQueue**=**max\_queue**<**typename std**::**iterator\_traits**<**InputIt**>::**value\_type**>>**

void maximum\_in\_sliding\_window**(**InputIt first**,** Size size**,** typename MaxQueue**::**size\_type win\_size**,** const Callback callback**)**

**{**

MaxQueue window**;**

**for(**size **-=** win\_size **-** 1**;** size **!=** 0**;** **++**first**)** **{**

window**.**push\_back**(\***first**);**

**if(**window**.**size**()** **==** win\_size**)** **{**

callback**(**window**.**max**());**

window**.**pop\_front**();**

**--**size**;**

**}**

**}**

**}**

**using** num\_t **=** std**::**uint\_least32\_t**;**

int main**()**

**{**

std**::**size\_t count**;**

std**::**cin**>>**count**;**

std**::**vector**<**num\_t**>** data**(**count**);**

std**::**for\_each**(**data**.**begin**(),**data**.**end**(),** **[](**num\_t**&** num**){**std**::**cin**>>**num**;});**

std**::**size\_t win\_size**;**

std**::**cin**>>**win\_size**;**

maximum\_in\_sliding\_window**(**data**.**begin**(),** data**.**size**(),** win\_size**,** **[](**const num\_t max**)** **{** std**::**cout **<<** max**<<**' '**;** **});**

**return** 0**;**

**}**

# 2 Очередь с приоритетом и система непересекающихся множеств

## 2.3.1 Построение кучи

**Решение:**

#include <iostream>

#include <vector>

#include <cstddef>

#include <cstdint>

#include <iterator>

#include <algorithm>

#include <utility>

template **<**typename InputIt**,** typename Callback**>**

void build\_heap**(**const InputIt first**,** const InputIt last**,** const Callback callback**)**

**{**

InputIt it **=** first **+** **(**std**::**distance**(**first**,** last**)** **+** 1**)** **/** 2 **-** 1**;**

**for(;** it **+** 1 **!=** first**;** **--**it**)** **{**

InputIt ind **=** it**;**

InputIt child **=** ind **+** std**::**distance**(**first**,** ind**)** **+** 1**;**

**while(**child **<** last**)** **{**

**if((**child **+** 1**)** **<** last **&&** **\*(**child **+** 1**)** **<** **\***child**)**

**++**child**;**

**if(\***child **<** **\***ind**){}else** **break;**

callback**(**ind**,** child**);**

std**::**swap**(\***ind**,\***child**);**

ind **=** child**;**

child **=** ind **+** std**::**distance**(**first**,** ind**)** **+** 1**;**

**}**

**}**

**}**

**using** num\_t**=**std**::**uint\_least32\_t**;**

int main**()** **{**

std**::**size\_t count**;**

std**::**cin**>>**count**;**

std**::**vector**<**num\_t**>** data**(**count**);**

std**::**for\_each**(**data**.**begin**(),**data**.**end**(),[](**num\_t**&** num**){**std**::**cin**>>**num**;});**

**using** it\_t**=**decltype**(**data**)::**iterator**;**

std**::**vector**<**std**::**pair**<**std**::**iterator\_traits**<**it\_t**>::**value\_type**,**std**::**iterator\_traits**<**it\_t**>::**value\_type**>>** swaps**;**

const it\_t beg**=**data**.**begin**();**

build\_heap**(**beg**,**data**.**end**(),[&**swaps**,&**beg**](**it\_t a**,** it\_t b**){**

swaps**.**push\_back**(**std**::**make\_pair**(**std**::**distance**(**beg**,**a**),**std**::**distance**(**beg**,**b**)));**

**});**

std**::**cout**<<**swaps**.**size**()<<**std**::**endl**;**

**for(**const auto**&** item**:** swaps**){**

std**::**cout**<<**item**.**first**<<**' '**<<**item**.**second**<<**std**::**endl**;**

**}**

**return** 0**;**

**}**

## 2.3.2 Параллельная обработка

**Решение:**

#include <cstdint>

#include <iostream>

#include <queue>

#include <vector>

#include <algorithm>

#include <iterator>

**using** num\_t**=**uint\_least64\_t**;**

struct new\_task **{**

num\_t id**;**

num\_t length**;**

**};**

struct completed\_task **{**

new\_task src**;**

num\_t start\_time**;**

num\_t processor**;**

num\_t get\_end\_time**()** const**{**

**return** start\_time **+** src**.**length**;**

**}**

**};**

bool **operator<(**const completed\_task**&** lhs**,** const completed\_task**&** rhs**)**

**{**

const num\_t l\_end **=** lhs**.**get\_end\_time**();**

const num\_t r\_end **=** rhs**.**get\_end\_time**();**

**if(**r\_end **!=** l\_end**)**

**return** r\_end **<** l\_end**;**

**return** rhs**.**processor **<** lhs**.**processor**;**

**}**

template **<**typename TaskCompleted**,**

typename CompletedTask **=** completed\_task**,**

typename Compare **=** std**::**less**<**CompletedTask**>,**

typename PriorQueue **=** std**::**priority\_queue**<**CompletedTask**,** std**::**vector**<**CompletedTask**>,** Compare**>>**

class parallel\_simulator

**{**

public**:**

**using** size\_type **=** typename PriorQueue**::**size\_type**;**

**using** container\_type **=** PriorQueue**;**

**using** value\_type **=** typename PriorQueue**::**value\_type**;**

**using** reference **=** typename PriorQueue**::**reference**;**

**using** const\_reference **=** typename PriorQueue**::**const\_reference**;**

private**:**

PriorQueue queue**;**

public**:**

const size\_type processors**;**

const TaskCompleted**&** task\_completed**;**

explicit parallel\_simulator**(**size\_type processors\_**,**

const TaskCompleted**&** task\_completed\_**,**

const Compare**&** comp **=** Compare**())**

**:** queue**(**comp**),** processors**(**processors\_**)**

**,** task\_completed**(**task\_completed\_**){}**

void clear**()**

**{**

**while(!**queue**.**empty**())**

queue**.**pop**();**

**}**

template **<**typename NewTask**>** void add\_task**(**const NewTask**&** nt**)**

**{**

**if(**queue**.**size**()** **>=** processors**)** **{**

CompletedTask ct**(**queue**.**top**());**

task\_completed**(**ct**);**

queue**.**pop**();**

queue**.**push**(**CompletedTask **{** nt**,** ct**.**get\_end\_time**(),** ct**.**processor **});**

**}** **else** **{**

const CompletedTask ct **{** nt**,** 0**,** queue**.**size**()** **};**

**if(**nt**.**length **!=** 0**)**

queue**.**push**(**ct**);**

**else**

task\_completed**(**ct**);**

**}**

**}**

void extract\_task**()**

**{**

task\_completed**(**queue**.**top**());**

queue**.**pop**();**

**}**

bool empty**()** const

**{**

**return** queue**.**empty**();**

**}**

size\_type size**()** const

**{**

**return** queue**.**size**();**

**}**

**};**

int main**()**

**{**

num\_t procs**;**

std**::**cin**>>**procs**;**

**using** tasks\_container **=** std**::**vector**<**num\_t**>;**

tasks\_container**::**size\_type size**;**

std**::**cin**>>**size**;**

tasks\_container tasks**(**size**);**

std**::**copy**(**std**::**istream\_iterator**<**tasks\_container**::**value\_type**>(**std**::**cin**),**

std**::**istream\_iterator**<**tasks\_container**::**value\_type**>(),**

tasks**.**begin**());**

std**::**vector**<**completed\_task**>** result**(**tasks**.**size**());**

auto on\_completed **=** **[&**result**](**const completed\_task**&** ctask**)** **{** result**[**ctask**.**src**.**id**]** **=** ctask**;** **};**

parallel\_simulator**<**decltype**(**on\_completed**)>** simulator**(**procs**,** on\_completed**);**

**for(**tasks\_container**::**size\_type i **=** 0**;** i **!=** tasks**.**size**();** **++**i**)**

simulator**.**add\_task**(**new\_task **{** i**,** tasks**[**i**]** **});**

**while(!**simulator**.**empty**())**

simulator**.**extract\_task**();**

**for(**const auto**&** task **:** result**)** **{**

std**::**cout **<<** task**.**processor **<<** ' ' **<<** task**.**start\_time **<<** std**::**endl**;**

**}**

**return** 0**;**

**}**

## 2.3.4 Объединение таблиц

**Решение:**

#include <cstddef>

#include <cstdint>

#include <iostream>

#include <iterator>

#include <type\_traits>

#include <utility>

#include <vector>

#include <unordered\_map>

**namespace** \_disjoint\_set\_implementation\_details

**{**

template **<**typename Ind**,** typename Item**>** struct node **{**

Ind parent**;**

Ind rank**;**

Item item**;**

node**(**Ind parent\_**,** Ind rank\_**,** Item item\_**)**

**:** parent**(**parent\_**)**

**,** rank**(**rank\_**)**

**,** item**(**item\_**)**

**{**

**}**

**};**

**}**

template **<**typename Element**,**

typename Ind **=** std**::**size\_t**,**

class MapContainer **=** std**::**unordered\_map**<**Element**,** Ind**>,**

class TreeContainer **=** std**::**vector**<**\_disjoint\_set\_implementation\_details**::**node**<**Ind**,** const Element**\*>>>**

class disjoint\_set

**{**

public**:**

**using** map\_container **=** MapContainer**;**

**using** tree\_container **=** TreeContainer**;**

**using** value\_type **=** Element**;**

**using** reference **=** Element**&;**

**using** const\_reference **=** const Element**&;**

**using** ind\_type **=** Ind**;**

protected**:**

template **<**bool IsConstIt**>** class internal\_iterator

**{**

friend disjoint\_set**;**

private**:**

**using** base\_iterator **=** typename map\_container**::**iterator**;**

base\_iterator it**;**

static inline typename std**::**iterator\_traits**<**base\_iterator**>::**value\_type**::**first\_type**&** reflect\_func**(**

base\_iterator it**)**

**{**

**return** it**->**first**;**

**}**

public**:**

**using** reference **=** decltype**(**reflect\_func**(**it**));**

**using** pointer **=** std**::**remove\_reference**<**reference**>\*;**

**using** value\_type **=** std**::**remove\_reference**<**reference**>;**

**using** iterator\_category **=** std**::**forward\_iterator\_tag**;**

private**:**

**using** mbc\_reference **=** typename std**::**conditional**<**IsConstIt**,** const reference**,** reference**>::**type**;**

**using** mbc\_pointer **=** typename std**::**conditional**<**IsConstIt**,** const pointer**,** pointer**>::**type**;**

internal\_iterator**(**const base\_iterator it\_**)**

**:** it**(**it\_**)**

**{**

**}**

public**:**

internal\_iterator**()**

**{**

**}**

internal\_iterator**(**const internal\_iterator**<false>&** other**)**

**:** it**(**other**.**it**)**

**{**

**}**

internal\_iterator**&** **operator=(**const internal\_iterator**&** other**)**

**{**

it **=** other**.**it**;**

**return** **\*this;**

**}**

internal\_iterator**&** **operator++()**

**{**

**++**it**;**

**return** **\*this;**

**}**

internal\_iterator **operator++(**int**)**

**{**

iterator tmp**(\*this);**

**operator++();**

**return** tmp**;**

**}**

const reference **operator\*()** const

**{**

**return** reflect\_func**(**it**);**

**}**

mbc\_reference **operator\*()**

**{**

**return** reflect\_func**(**it**);**

**}**

const pointer **operator->()** const

**{**

**return** **&**reflect\_func**(**it**);**

**}**

mbc\_pointer **operator->()**

**{**

**return** **&**reflect\_func**(**it**);**

**}**

bool **operator!=(**const internal\_iterator**&** other**)** const

**{**

**return** it **!=** other**.**it**;**

**}**

bool **operator==(**const internal\_iterator**&** other**)** const

**{**

**return** it **==** other**.**it**;**

**}**

**};**

mutable tree\_container tree**;**

mutable map\_container map**;**

ind\_type \_sets\_count**;**

**using** map\_iterator **=** typename map\_container**::**iterator**;**

ind\_type collapse**(**ind\_type ind**)** const

**{**

ind\_type root **=** ind**;**

**while(**tree**[**root**].**parent **!=** root**)** **{**

root **=** tree**[**root**].**parent**;**

**}**

**while(**tree**[**ind**].**parent **!=** ind**)** **{**

ind\_type oind **=** ind**;**

ind **=** tree**[**ind**].**parent**;**

tree**[**oind**].**parent **=** root**;**

**}**

**return** root**;**

**}**

public**:**

**using** iterator **=** internal\_iterator**<false>;**

**using** const\_iterator **=** internal\_iterator**<true>;**

protected**:**

inline iterator create\_it**(**ind\_type ind**)**

**{**

**return** iterator**(**map**.**find**(\***tree**[**ind**].**item**));**

**}**

template **<**bool find\_mode**,** typename VT**>** iterator \_make\_set**(**VT**&&** item**)**

**{**

**static\_assert(**std**::**is\_convertible**<**VT**,** value\_type**>::**value**,** "VT should be convertible to value\_type"**);**

const ind\_type size **=** tree**.**size**();**

auto emplace\_result **=** map**.**emplace**(**std**::**forward**<**VT**>(**item**),** size**);**

**if(!**emplace\_result**.**second**)** **{**

**if(**find\_mode**)**

**return** find\_set**(**emplace\_result**.**first**);**

**return** end**();**

**}**

tree**.**emplace\_back**(**size**,** 0**,** **&(**emplace\_result**.**first**->**first**));**

**++**\_sets\_count**;**

**return** emplace\_result**.**first**;**

**}**

public**:**

disjoint\_set**()**

**:** tree**()**

**,** map**()**

**,** \_sets\_count**(**0**)**

**{**

**}**

disjoint\_set**(**const disjoint\_set**&** other**)**

**:** tree**(**other**.**tree**)**

**,** map**(**other**.**map**)**

**,** \_sets\_count**(**other**.**sets\_count**())**

**{**

**for(**auto it **=** map**.**begin**();** it **!=** map**.**end**();** **++**it**)** **{**

tree**[**it**->**second**].**item **=** **&**it**->**first**;**

**}**

**}**

disjoint\_set**&** **operator=(**disjoint\_set other**)**

**{**

swap**(\*this,** other**);**

**return** **\*this;**

**}**

friend void swap**(**disjoint\_set**&** lhs**,** disjoint\_set**&** rhs**)**

**{**

lhs**.**tree**.**swap**(**rhs**.**tree**);**

lhs**.**map**.**swap**(**rhs**.**map**);**

std**::**swap**(**lhs**.**\_sets\_count**,** rhs**.**\_sets\_count**);**

**}**

inline void reserve**(**ind\_type size**)**

**{**

tree**.**reserve**(**size**);**

map**.**reserve**(**size**);**

**}**

inline iterator begin**()**

**{**

**return** map**.**begin**();**

**}**

inline iterator end**()**

**{**

**return** map**.**end**();**

**}**

inline const\_iterator cbegin**()** const

**{**

**return** map**.**begin**();**

**}**

inline const\_iterator cend**()** const

**{**

**return** map**.**end**();**

**}**

template **<**typename VT**>** inline iterator make\_set**(**VT**&&** item**)**

**{**

**return** \_make\_set**<false>(**std**::**forward**<**VT**>(**item**));**

**}**

template **<**typename VT**>** inline iterator find\_or\_make\_set**(**VT**&&** item**)**

**{**

**return** \_make\_set**<true>(**std**::**forward**<**VT**>(**item**));**

**}**

inline iterator find\_set**(**const\_iterator item**)**

**{**

**return** create\_it**(**collapse**(**tree**[**item**.**it**->**second**].**parent**));**

**}**

inline const\_iterator find\_set**(**const\_iterator item**)** const

**{**

**return** map**.**find**(\***tree**[**collapse**(**tree**[**item**.**it**->**second**].**parent**)].**item**);**

**}**

template **<**typename VT**,** typename std**::**enable\_if**<**std**::**is\_convertible**<**VT**,** value\_type**>::**value**,** bool**>::**type **=** **true>**

iterator find\_set**(**VT**&&** item**)**

**{**

const iterator it**(**map**.**find**(**std**::**forward**<**VT**>(**item**)));**

**if(**it **!=** end**())**

**return** find\_set**(**it**);**

**return** it**;**

**}**

template **<**typename VT**,** typename std**::**enable\_if**<**std**::**is\_convertible**<**VT**,** value\_type**>::**value**,** bool**>::**type **=** **true>**

const\_iterator find\_set**(**VT**&&** item**)** const

**{**

const const\_iterator it**(**map**.**find**(**std**::**forward**<**VT**>(**item**)));**

**if(**it **!=** cend**())**

**return** find\_set**(**it**);**

**return** it**;**

**}**

inline void swap\_values**(**const\_iterator lhs**,** const\_iterator rhs**)**

**{**

std**::**swap**(**tree**[**lhs**.**it**->**second**].**item**,** tree**[**rhs**.**it**->**second**].**item**);**

std**::**swap**(**lhs**.**it**->**second**,** rhs**.**it**->**second**);**

**}**

enum class agent\_kind **{** same\_set**,** left\_set**,** right\_set **};**

std**::**pair**<**agent\_kind**,** iterator**>** link**(**const\_iterator lhs**,** const\_iterator rhs**)**

**{**

const ind\_type lind **=** collapse**(**tree**[**lhs**.**it**->**second**].**parent**);**

const ind\_type rind **=** collapse**(**tree**[**rhs**.**it**->**second**].**parent**);**

**if(**lind **==** rind**)**

**return** std**::**make\_pair**(**agent\_kind**::**same\_set**,** create\_it**(**lind**));**

**--**\_sets\_count**;**

**if(**tree**[**lind**].**rank **>** tree**[**rind**].**rank**)** **{**

tree**[**rind**].**parent **=** lind**;**

**return** std**::**make\_pair**(**agent\_kind**::**left\_set**,** create\_it**(**lind**));**

**}** **else** **{**

tree**[**lind**].**parent **=** rind**;**

**if(**tree**[**lind**].**rank **==** tree**[**rind**].**rank**)**

**++**tree**[**rind**].**rank**;**

**return** std**::**make\_pair**(**agent\_kind**::**right\_set**,** create\_it**(**rind**));**

**}**

**}**

inline void left\_link**(**const\_iterator agent**,** const\_iterator item**)**

**{**

auto pair **=** link**(**agent**,** item**);**

**if(**const\_iterator**(**pair**.**second**)** **!=** agent**)** **{**

swap\_values**(**agent**,** pair**.**second**);**

**}**

**}**

inline void right\_link**(**const\_iterator item**,** const\_iterator agent**)**

**{**

left\_link**(**agent**,** item**);**

**}**

inline ind\_type sets\_count**()** const

**{**

**return** \_sets\_count**;**

**}**

inline ind\_type size**()** const

**{**

**return** **(**ind\_type**)**tree**.**size**();**

**}**

inline void clear**()**

**{**

map**.**clear**();**

tree**.**clear**();**

\_sets\_count **=** 0**;**

**}**

**};**

int main**()**

**{**

**using** num\_t **=** std**::**uint\_fast32\_t**;**

num\_t table\_size**,** query\_count**;**

std**::**cin **>>** table\_size **>>** query\_count**;**

std**::**vector**<**num\_t**>** tables**(**table\_size**);**

disjoint\_set**<**num\_t**>** disjoint**;**

disjoint**.**reserve**(**table\_size**);**

num\_t max\_sz**=**0**;**

**for(**num\_t**&** sz**:**tables**){**

std**::**cin**>>**sz**;**

**if** **(**sz**>**max\_sz**)** max\_sz**=**sz**;**

**}**

**while(**query\_count**--){**

num\_t dest**,**src**;**

std**::**cin**>>**dest**>>**src**;**

**--**dest**;--**src**;**

auto dit**=**disjoint**.**find\_or\_make\_set**(**dest**);**

auto sit**=**disjoint**.**find\_or\_make\_set**(**src**);**

**if** **(**dit**!=**sit**){**

tables**[\***dit**]+=**tables**[\***sit**];**

disjoint**.**left\_link**(**dit**,**sit**);**

**if** **(**tables**[\***dit**]>**max\_sz**)** max\_sz**=**tables**[\***dit**];**

**}**

std**::**cout**<<**max\_sz**<<**std**::**endl**;**

**}**

**}**

## 2.3.5 Автоматический анализ программ

**Решение:**

#include <cstddef>

#include <cstdint>

#include <iostream>

#include <iterator>

#include <type\_traits>

#include <unordered\_map>

#include <utility>

#include <vector>

**namespace** \_disjoint\_set\_implementation\_details

**{**

template **<**typename Ind**,** typename Item**>** struct node **{**

Ind parent**;**

Ind rank**;**

Item item**;**

node**(**Ind parent\_**,** Ind rank\_**,** Item item\_**)**

**:** parent**(**parent\_**)**

**,** rank**(**rank\_**)**

**,** item**(**item\_**)**

**{**

**}**

**};**

**}**

template **<**typename Element**,**

typename Ind **=** std**::**size\_t**,**

class MapContainer **=** std**::**unordered\_map**<**Element**,** Ind**>,**

class TreeContainer **=** std**::**vector**<**\_disjoint\_set\_implementation\_details**::**node**<**Ind**,** const Element**\*>>>**

class disjoint\_set

**{**

public**:**

**using** map\_container **=** MapContainer**;**

**using** tree\_container **=** TreeContainer**;**

**using** value\_type **=** Element**;**

**using** reference **=** Element**&;**

**using** const\_reference **=** const Element**&;**

**using** ind\_type **=** Ind**;**

protected**:**

template **<**bool IsConstIt**>** class internal\_iterator

**{**

friend disjoint\_set**;**

private**:**

**using** base\_iterator **=** typename map\_container**::**iterator**;**

base\_iterator it**;**

static inline typename std**::**iterator\_traits**<**base\_iterator**>::**value\_type**::**first\_type**&** reflect\_func**(**

base\_iterator it**)**

**{**

**return** it**->**first**;**

**}**

public**:**

**using** reference **=** decltype**(**reflect\_func**(**it**));**

**using** pointer **=** std**::**remove\_reference**<**reference**>\*;**

**using** value\_type **=** std**::**remove\_reference**<**reference**>;**

**using** iterator\_category **=** std**::**forward\_iterator\_tag**;**

private**:**

**using** mbc\_reference **=** typename std**::**conditional**<**IsConstIt**,** const reference**,** reference**>::**type**;**

**using** mbc\_pointer **=** typename std**::**conditional**<**IsConstIt**,** const pointer**,** pointer**>::**type**;**

internal\_iterator**(**const base\_iterator it\_**)**

**:** it**(**it\_**)**

**{**

**}**

public**:**

internal\_iterator**()**

**{**

**}**

internal\_iterator**(**const internal\_iterator**<false>&** other**)**

**:** it**(**other**.**it**)**

**{**

**}**

internal\_iterator**&** **operator=(**const internal\_iterator**&** other**)**

**{**

it **=** other**.**it**;**

**return** **\*this;**

**}**

internal\_iterator**&** **operator++()**

**{**

**++**it**;**

**return** **\*this;**

**}**

internal\_iterator **operator++(**int**)**

**{**

iterator tmp**(\*this);**

**operator++();**

**return** tmp**;**

**}**

const reference **operator\*()** const

**{**

**return** reflect\_func**(**it**);**

**}**

mbc\_reference **operator\*()**

**{**

**return** reflect\_func**(**it**);**

**}**

const pointer **operator->()** const

**{**

**return** **&**reflect\_func**(**it**);**

**}**

mbc\_pointer **operator->()**

**{**

**return** **&**reflect\_func**(**it**);**

**}**

bool **operator!=(**const internal\_iterator**&** other**)** const

**{**

**return** it **!=** other**.**it**;**

**}**

bool **operator==(**const internal\_iterator**&** other**)** const

**{**

**return** it **==** other**.**it**;**

**}**

**};**

mutable tree\_container tree**;**

mutable map\_container map**;**

ind\_type \_sets\_count**;**

**using** map\_iterator **=** typename map\_container**::**iterator**;**

ind\_type collapse**(**ind\_type ind**)** const

**{**

ind\_type root **=** ind**;**

**while(**tree**[**root**].**parent **!=** root**)** **{**

root **=** tree**[**root**].**parent**;**

**}**

**while(**tree**[**ind**].**parent **!=** ind**)** **{**

ind\_type oind **=** ind**;**

ind **=** tree**[**ind**].**parent**;**

tree**[**oind**].**parent **=** root**;**

**}**

**return** root**;**

**}**

public**:**

**using** iterator **=** internal\_iterator**<false>;**

**using** const\_iterator **=** internal\_iterator**<true>;**

private**:**

inline iterator create\_it**(**ind\_type ind**)**

**{**

**return** iterator**(**map**.**find**(\***tree**[**ind**].**item**));**

**}**

template **<**bool find\_mode**,** typename VT**>** iterator \_make\_set**(**VT**&&** item**)**

**{**

**static\_assert(**std**::**is\_convertible**<**VT**,** value\_type**>::**value**,** "VT should be convertible to value\_type"**);**

const ind\_type size **=** tree**.**size**();**

auto emplace\_result **=** map**.**emplace**(**std**::**forward**<**VT**>(**item**),** size**);**

**if(!**emplace\_result**.**second**)** **{**

**if(**find\_mode**)**

**return** find\_set**(**emplace\_result**.**first**);**

**return** end**();**

**}**

tree**.**emplace\_back**(**size**,** 0**,** **&(**emplace\_result**.**first**->**first**));**

**++**\_sets\_count**;**

**return** emplace\_result**.**first**;**

**}**

public**:**

disjoint\_set**()**

**:** tree**()**

**,** map**()**

**,** \_sets\_count**(**0**)**

**{**

**}**

disjoint\_set**(**const disjoint\_set**&** other**)**

**:** tree**(**other**.**tree**)**

**,** map**(**other**.**map**)**

**,** \_sets\_count**(**other**.**sets\_count**())**

**{**

**for(**auto it **=** map**.**begin**();** it **!=** map**.**end**();** **++**it**)** **{**

tree**[**it**->**second**].**item **=** **&**it**->**first**;**

**}**

**}**

disjoint\_set**&** **operator=(**disjoint\_set other**)**

**{**

swap**(\*this,** other**);**

**return** **\*this;**

**}**

friend void swap**(**disjoint\_set**&** lhs**,** disjoint\_set**&** rhs**)**

**{**

lhs**.**tree**.**swap**(**rhs**.**tree**);**

lhs**.**map**.**swap**(**rhs**.**map**);**

std**::**swap**(**lhs**.**\_sets\_count**,** rhs**.**\_sets\_count**);**

**}**

inline void reserve**(**ind\_type size**)**

**{**

tree**.**reserve**(**size**);**

map**.**reserve**(**size**);**

**}**

inline iterator begin**()**

**{**

**return** map**.**begin**();**

**}**

inline iterator end**()**

**{**

**return** map**.**end**();**

**}**

inline const\_iterator cbegin**()** const

**{**

**return** map**.**begin**();**

**}**

inline const\_iterator cend**()** const

**{**

**return** map**.**end**();**

**}**

template **<**typename VT**>** inline iterator make\_set**(**VT**&&** item**)**

**{**

**return** \_make\_set**<false>(**std**::**forward**<**VT**>(**item**));**

**}**

template **<**typename VT**>** inline iterator find\_or\_make\_set**(**VT**&&** item**)**

**{**

**return** \_make\_set**<true>(**std**::**forward**<**VT**>(**item**));**

**}**

inline iterator find\_set**(**const\_iterator item**)**

**{**

**return** create\_it**(**collapse**(**tree**[**item**.**it**->**second**].**parent**));**

**}**

inline const\_iterator find\_set**(**const\_iterator item**)** const

**{**

**return** map**.**find**(\***tree**[**collapse**(**tree**[**item**.**it**->**second**].**parent**)].**item**);**

**}**

template **<**typename VT**,** typename std**::**enable\_if**<**std**::**is\_convertible**<**VT**,** value\_type**>::**value**,** bool**>::**type **=** **true>**

iterator find\_set**(**VT**&&** item**)**

**{**

const iterator it**(**map**.**find**(**std**::**forward**<**VT**>(**item**)));**

**if(**it **!=** end**())**

**return** find\_set**(**it**);**

**return** it**;**

**}**

template **<**typename VT**,** typename std**::**enable\_if**<**std**::**is\_convertible**<**VT**,** value\_type**>::**value**,** bool**>::**type **=** **true>**

const\_iterator find\_set**(**VT**&&** item**)** const

**{**

const const\_iterator it**(**map**.**find**(**std**::**forward**<**VT**>(**item**)));**

**if(**it **!=** cend**())**

**return** find\_set**(**it**);**

**return** it**;**

**}**

inline void swap\_values**(**const\_iterator lhs**,** const\_iterator rhs**)**

**{**

std**::**swap**(**tree**[**lhs**.**it**->**second**].**item**,** tree**[**rhs**.**it**->**second**].**item**);**

std**::**swap**(**lhs**.**it**->**second**,** rhs**.**it**->**second**);**

**}**

enum class agent\_kind **{** same\_set**,** left\_set**,** right\_set **};**

std**::**pair**<**agent\_kind**,** iterator**>** link**(**const\_iterator lhs**,** const\_iterator rhs**)**

**{**

const ind\_type lind **=** collapse**(**tree**[**lhs**.**it**->**second**].**parent**);**

const ind\_type rind **=** collapse**(**tree**[**rhs**.**it**->**second**].**parent**);**

**if(**lind **==** rind**)**

**return** std**::**make\_pair**(**agent\_kind**::**same\_set**,** create\_it**(**lind**));**

**--**\_sets\_count**;**

**if(**tree**[**lind**].**rank **>** tree**[**rind**].**rank**)** **{**

tree**[**rind**].**parent **=** lind**;**

**return** std**::**make\_pair**(**agent\_kind**::**left\_set**,** create\_it**(**lind**));**

**}** **else** **{**

tree**[**lind**].**parent **=** rind**;**

**if(**tree**[**lind**].**rank **==** tree**[**rind**].**rank**)**

**++**tree**[**rind**].**rank**;**

**return** std**::**make\_pair**(**agent\_kind**::**right\_set**,** create\_it**(**rind**));**

**}**

**}**

inline void left\_link**(**const\_iterator agent**,** const\_iterator item**)**

**{**

auto pair **=** link**(**agent**,** item**);**

**if(**const\_iterator**(**pair**.**second**)** **!=** agent**)** **{**

swap\_values**(**agent**,** pair**.**second**);**

**}**

**}**

inline void right\_link**(**const\_iterator item**,** const\_iterator agent**)**

**{**

left\_link**(**agent**,** item**);**

**}**

inline ind\_type sets\_count**()** const

**{**

**return** \_sets\_count**;**

**}**

inline ind\_type size**()** const

**{**

**return** **(**ind\_type**)**tree**.**size**();**

**}**

inline void clear**()**

**{**

map**.**clear**();**

tree**.**clear**();**

\_sets\_count **=** 0**;**

**}**

**};**

int main**()**

**{**

**using** num\_t **=** std**::**uint\_fast32\_t**;**

num\_t n**,** eq**,** neq**;**

std**::**cin **>>** n **>>** eq **>>** neq**;**

disjoint\_set**<**num\_t**>** disset**;**

disset**.**reserve**(**n**);**

**while(**eq**--){**

num\_t lv**,**rv**;**

std**::**cin**>>**lv**>>**rv**;**

disset**.**link**(**disset**.**find\_or\_make\_set**(**lv**),** disset**.**find\_or\_make\_set**(**rv**));**

**}**

**while(**neq**--){**

num\_t lv**,**rv**;**

std**::**cin**>>**lv**>>**rv**;**

**if** **(**disset**.**find\_or\_make\_set**(**lv**)==**disset**.**find\_or\_make\_set**(**rv**)){**

std**::**cout**<<**'0'**;**

**return** 0**;**

**}**

**}**

std**::**cout**<<**'1'**;**

**return** 0**;**

**}**

# Сертификат

## [скрыто, сертификат 53%, <https://stepik.org/course/1547/syllabus>]