Федеральное государственное автономное образовательное учреждение  
высшего образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий математики и механики

**Отчёт по лабораторной работе**

**«Разделённые множества. Задача Percolation»**

**Выполнил:**

студент группы 0823-2

Чапыгин С.А.

**Проверил:**

ассистент каф. МОСТ института ИТММ

Сиднев А.А.

Нижний Новгород

2016 г.

**Оглавление**

Оглавление

[Введение 3](#_Toc453010700)

[Постановка задачи 4](#_Toc453010701)

[Руководство пользователя 5](#_Toc453010702)

[Руководство программиста 6](#_Toc453010703)

[*Структура программы* 6](#_Toc453010704)

[*Структуры данных* 6](#_Toc453010705)

[*Описание алгоритмов* 7](#_Toc453010706)

[*Описание классов* 8](#_Toc453010707)

[*Google Test* 8](#_Toc453010708)

[Заключение 9](#_Toc453010709)

[Приложение 10](#_Toc453010710)

[Литература 13](#_Toc453010711)

# Введение

Разделенные множества — это абстрактный тип данных, предназначенный для представления коллекции, состоящей из некоторого числа k попарно непересекающихся подмножеств U_1, U_2\dts
U_k заданного множества U.

В физике и химии явлением перколяции (от [лат.](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D1%82%D0%B8%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) percōlāre, просачиваться, протекать) называется явление протекания или не протекания жидкостей через пористые материалы, электричества через смесь проводящих и непроводящих частиц и другие подобные процессы. Теория перколяции находит применение в описании разнообразных систем и явлений, в том числе таких, как распространение эпидемий и надежность компьютерных сетей.

# Постановка задачи

Имеется прямоугольная решетка, состоящая из N×M открытых и закрытых ячеек. Будем называть кластером соседние открытые ячейки и стягивающим кластером (перколяционным кластером) такой кластер, который начинается на одной границе и заканчивается на противоположной границе решетки. Необходимо установить долю открытых ячеек, при которой возникает стягивающий кластер, то есть имеется путь от верхней до нижней границы решётки.

Таким образом, необходимо решить следующую задачу: реализовать структуру данных разделённые множества с использованием древесной структуры с рангами. С помощью разделённых множеств реализовать алгоритм поиска числа открытых ячеек, при котором в решётке имеется стягивающий кластер. Провести эксперименты, определяющий долю открытых ячеек. Для структуры данных разделённые множества написать тесты с помощью Google C++ Testing Framework. Разработать консольное приложение для демонстрации результатов эксперимента.

# Руководство пользователя

Для того чтобы решить задачу Percolation пользователю необходимо выполнить следующие действия:

Шаг 1. Задать размеры матрицы M x N.

Шаг 2. Задать количество проводимых экспериментов K.

Шаг 3. Запустить программу.

В окне консоли будет выведен ответ в виде protsent = x.

# Руководство программиста

## *Структура программы*

Проекты:

* **gtest** – библиотека Google Test
* **MyList** – проект реализации программы
* **MyList-test** – проект с тестами для программы

Файлы:

* List.h – объявление класса Sset разделенные множества
* List.cpp – реализация Sset
* Perc.h – объявление функций
* Perc.cpp – реализация функций
* Main.cpp – функция проведения экспериментов
* Test\_MyList.cpp – тесты для класса Sset
* Test\_main.cpp – запуск всех тестов для Sset

## *Структуры данных*

Разделенные множества– это абстрактный тип данных, предназначенный для представления коллекции, состоящей из некоторого числа *k*попарно непересекающихся подмножеств *U*1, *U*2, ..., *Uk*заданного множества *U*.

Этот тип данных применяется в таких задачах, как отыскание минимального остовного дерева для заданного взвешенного неориентированного графа, построение компонент связности графа, минимизация конечного автомата, и многих других, требующих динамического поддержания некоторого отношения эквивалентности. Примеры таких задач будут рассмотрены ниже.

Как правило, в таких задачах вычисления начинаются с пустой коллекции подмножеств (*k*= 0)*.*Затем по мере вычислений формируются новые подмножества, включаемые в коллекцию. Формирование новых подмножеств происходит либо путем создания одноэлементного подмножества, либо путем объединения уже существующих в коллекции подмножеств. Для осуществления таких действий используются имена включенных в коллекцию подмножеств. В качестве имени подмножества будем использовать один из его элементов (главный элемент), выбираемый по определенному правилу. Поскольку в коллекции всегда будут находиться попарно непересекающиеся подмножества множества *U*, такое имя будет однозначно определять требуемое подмножество.

Операции над разделенными множествами:

1. *Создать* (*x*). Эта операция предназначена для введения в коллекцию нового подмножества, состоящего из одного элемента *x*, при этом предполагается, что*x*не входит ни в одно из подмножеств коллекции, созданной к моменту выполнения этой операции. Элемент *x*указывается в качестве параметра. Именем созданного подмножества будет считаться сам элемент *x*.
2. *Объединить*(*x*, *y*). С помощью этой операции можно объединить два подмножества коллекции, имеющие, соответственно, имена *x*и *y*, в одно новое подмножество, при этом оба объединяемые подмножества удаляются из коллекции, а вновь построенное подмножество получает некоторое имя. Во всех рассматриваемых нами случаях именем нового полученного в результате этой операции подмножества будет одно из имен *x* или *y*. Имена объединяемых подмножеств указываются в качестве параметров.
3. *Найти*(*x*, *y*). Эта операция позволяет определить имя *y*того подмножества коллекции, которому принадлежит элемент *x*. Если элемент *x*до выполнения операции не входил ни в одно из подмножеств коллекции, то в качестве *y*берется0*.*

Последовательность σ, составленную из операций типа СОЗДАТЬ, ОБЪЕДИНИТЬ, НАЙТИ, назовем *корректной*, если перед выполнением каждой операции из последовательностиσ выполнены условия ее применения. Например, перед выполнением очередной операции вида ОБЪЕДИНИТЬ (*x*,*y*) подмножества с именами *x*и *y*должны быть уже созданы. Перед выполнением операции СОЗДАТЬ (*x*) элемент *x*не должен принадлежать ни одному из подмножеств коллекции. Операция НАЙТИ (*x*, *y*) применима при любом значении аргумента *x U*. Следует только помнить, что если *x*не принадлежит ни одному из подмножеств коллекции, то получим *y*= 0*.*

Способы представления:

•с помощью массива;

•с помощью древовидной структуры;

•с помощью древовидной структуры с использованием рангов вершин;

•с помощью древовидной структуры с использованием рангов вер-

шин и сжатия путей.

Последний из перечисленных способов является наиболее эффективным по времени выполнения произвольных корректных последовательностей операций типа СОЗДАТЬ, ОБЪЕДИНИТЬ, НАЙТИ. Строго говоря, во всех перечисленных случаях будут использоваться массивы, но интерпретации их содержимого будут различными. Каждый раз при описании очередной реализации мы будем обсуждать оценки трудоемкости рассматриваемых операций.

## *Описание алгоритмов*

На входе программа имеет матрицу ячеек размера М х N. Часть этих ячеек преобразуются в открытые, часть остаются закрытыми. Программа запускает циклическую последовательность действий: проверка соседних ячеек на предмет открытости -> если ячейка свободна, то присоединение её к множеству найденной ячейки. Когда найден путь из верхней ячейки до нижней, процесс прекращается. Далее производится подсчет процента открытых ячеек.

## *Описание классов*

Класс Sset – класс разделенных множеств.

1. **\*Val** – массив данных

2. **\*h** – служебный массив(связанности элементов)

3. **Size** – размер

4**. Sset()** – конструктор

5. **Sset(size)** – конструктор с параметром

6. **Merge(I, j)** – функция слияния

7. **Singleton(i)** – создание синглтона

8. **Search(i)** – функция поиска

9. **Clear()** – функция удаления

10. **~Sset()** - деструктор

## *Google Test*

Test(Sset, can\_create) – тест на создание множества

Test(Sset, can\_singleton) – проверка метода singleton

Test(Sset, singleton\_throws\_when\_bad\_idx) – проверка корректности ввода в методе singleton

Test(Sset, searches) – проверка метода search

# Заключение

В качестве итоговой проверки было проведено 3 эксперимента, демонстрирующие и проверяющие работу программы. Входные данные и результаты приведены ниже:

|  |  |  |
| --- | --- | --- |
| Размер матрицы | Количество экспериментов | Результат |
| 200×200 | 100 | 0.587943 |
| 400×400 | 500 | 0.592375 |
| 400×400 | 1000 | 0.592497 |

Тесты класса Sset пройдены успешно.

# Приложение

List.cpp

#include "List.h"

#include "stdio.h"

#include <exception>

Sset::Sset(int \_size) {

size = \_size;

val = new int[size];

h = new int[size];

for (int i = 0; i < size; i++) {

val[i] = -1;

h[i] = -1;

}

}

void Sset::singleton(int i) {

if (i < 0 || i > size || val[i] != -1)

throw "BAD INDEX";

val[i] = i;

h[i] = 0;

}

void Sset::merge(int i, int j) {

if (i < 0 || i > size || j < 0 || j > size || val[i] == -1 || val[j] == -1)

throw "BAD INDEX";

if (i != j) {

int a = i;

int b = j;

while (val[a] != a)

a = val[a];

while (val[b] != b)

b = val[b];

if (a != b) {

if (h[a] < h[b]) {

val[a] = b;

} else {

val[b] = a;

if (h[a] == h[b])

++h[a];

}

}

}

}

int Sset::search(int i) {

if (i < 0 || i > size || val[i] == -1)

throw "BAD INDEX";

while (val[i] != i)

i = val[i];

return val[i];

}

void Sset::clear() {

for (int i = 0; i < size; i++) {

val[i] = -1;

h[i] = -1;

}

}

Sset::~Sset() {

delete[] val;

delete[] h;

}

Perc.cpp

#include "perc.h"

void countino(Sset \*set, int \*co, int n, int m) {

bool f = false;

int x;

int y;

srand(time(0));

int coord;

while (f == false && (\*co) < n\*m) {

do {

x = rand() % n;

y = rand() % m;

} while (set -> val[x + (y + 1)\*n] != -1);

(\*co)++;

coord = x + (y + 1)\*n;

set -> singleton(coord);

if (x != 0)

if (set -> val[coord - 1] != -1)

set -> merge(coord, coord - 1);

if (set -> val[coord - n] != -1)

set -> merge(coord, coord - n);

if (x != n - 1)

if (set -> val[coord + 1] != -1)

set -> merge(coord, coord + 1);

if (set -> val[coord + n] != -1)

set -> merge(coord, coord + n);

f = (set -> search(0) == set -> search(set -> size - 1));

}

}

void preparation(Sset \*set, int n, int m) {

set -> clear();

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

set -> singleton(i);

set -> singleton(set -> size - i - 1);

}

for (int i = 1; i < n; i++) {

set -> merge(0, i);

set -> merge((\*set).size - 1, set -> size - i - 1);

}

}

Test\_MyList.cpp

#include "List.h"

#include <gtest.h>

TEST(SSet, can\_create) {

EXPECT\_NO\_THROW(Sset a(23));

}

TEST(SSet, can\_singleton) {

Sset a(23);

a.singleton(22);

EXPECT\_EQ(a.val[22], 22);

}

TEST(SSet, singleton\_throws\_bad\_idx) {

Sset a(23);

EXPECT\_ANY\_THROW(a.singleton(-3));

}

TEST(SSet, searches) {

Sset a(23);

a.singleton(4);

a.singleton(0);

a.singleton(11);

a.singleton(22);

a.singleton(13);

a.merge(0, 22);

a.merge(11, 13);

EXPECT\_EQ(a.search(22), 0);

}

Main.cpp

#include "perc.h"

void main() {

int k = 500;

int n = 400;

int m = 400;

int size = n \* m + 2 \* n;

Sset set(size);

int count;

double res = 0;

for (int i = 0; i < k; i++) {

preparation(&set, n, m);

count = 0;

countino(&set, &count, n, m);

res += count \* 1.0 / n / m;

}

printf("protsent = %f", (res / k));

}

# Литература

1. Б. Страуструп. Язык программирования C++ – The C++ Programming Language / Пер. с англ. – 3-е изд. – СПб.; М.: Невский диалект – Бином, 1999
2. 1999 Тарасевич Ю.Ю. Перколяция: теория, приложения, алгоритмы. - М.: УРСС, 2002.