**Словари и множества**

Сегодня мы поговорим о множествах. Множества — это математические структуры, которые могут хранить в себе уникальные элементы (то есть, каждый элемент может входить в множество только один раз).

**Работа с элементами множества**

Решим следующую задачу: даны N запросов трёх типов:

1. добавить элемент во множество;
2. проверить, входит ли элемент во множество;
3. удалить элемент из множества.

Сначала задается число N, а затем сами запросы. Каждый из них состоит из двух чисел. Первое обозначает тип запроса, а второе — элемент, с которым нужно произвести операцию.

Решать эту задачу удобнее всего с помощью типа set (множество). Для начала нужно подключить одноимённую библиотеку set. Давайте напишем программу, а потом я объясню новые понятия:

#include <iostream>

#include <set>

using namespace std;

int main() {

    set <int> s;

    int n;

    cin >> n;

    for (int i = 0; i < n; i++) {

        int type, x;

        cin >> type >> x;

        if (type == 1) { // добавление

            s.insert(x);

        } else if (type == 2) { // проверка

            if (s.find(x) == s.end()) {

                  cout << "NO\n";

            } else {

                cout << "YES\n";

            }

        } else { // удаление

              s.erase(x);

        }

    }

    return 0;

}

Множества создаются по аналогии с векторами. Пишем ключевое слово set, за ним — название типа каждого из элементов множества (в треугольных скобках), а после этого указываем имя для нового множества. Так мы получаем пустое множество. Добавление элементов в него происходит с помощью метода insert. Чтобы проверить, входит ли элемент во множество, используется метод find. Если элемент в множестве не нашелся, то он выдает то же значение, что и метод end. Удаление отдельного элемента из множества выполняется с помощью метода erase.

**Вывод всех элементов множества**

Вот как можно пройти по всем элементам, которые есть во множестве. Сначала посчитаем N чисел и «положим» их во множество S:

set <int> s;

int n;

cin >> n;

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

    int x;

    cin >> x;

    s.insert(x);

}

Вывести всё содержимое множества можно двумя способами. Первый из них новый:

for (auto now = s.begin(); now != s.end(); now++) {

    cout << \*now << ' ';

}

В данном случае now — это не очередной элемент, а указатель на него. Метод begin возвращает указатель на самый маленький элемент множества, end — это конец множества (он идёт после самого большого элемента), а операция ++ осуществляет переход к указателю на следующий элемент. Чтобы посмотреть, что за элемент хранится по указателю, нужно перед его именем написать символ \*.

Второй способ позволяет выводить элементы множества аналогично выводу всех элементов в векторе:

for (auto now : s) {

    cout << now << ' ';

}

Если ввести одинаковые элементы, то каждый из них окажется во множестве (и будет выведен) только один раз.

**Сортировка с помощью множества**

Поскольку проход по элементам множества осуществляется в возрастающем порядке, то велик соблазн использовать его для сортировки последовательностей. Увы, всё портит тот факт, что с одинаковыми элементами множество работает иначе.

Тем не менее, задачу сортировки решить можно и с их помощью. В C++ есть структура multiset, которая может хранить в себе одинаковые элементы. Multiset умеет все то же самое, что и обычный set, и лежит в той же библиотеке. Если в предыдущей программе вывода всех элементов заменить set на multiset, то мы как раз и получим элементы по возрастанию.

**Количество разных элементов**

С помощью set очень легко подсчитать число различных элементов в последовательности. Для этого нужно просто добавить все элементы последовательности во множество, а затем посмотреть на его размер. Вот как решить эту задачу с помощью того, что мы уже знаем. При обработке очередного элемента последовательности нужно сначала проверить, есть ли элемент во множестве. Если его там нет — увеличиваем счётчик на единицу, а затем добавляем элемент во множество. Но есть и более простой способ. У set есть метод size(), который возвращает количество элементов во множестве. Достаточно просто добавить в него все элементы подряд, а затем вывести size() от этого множества.

**Подсчет количества вхождений элемента в последовательность**

Поскольку во множестве все элементы упорядочены, с его помощью легко подсчитать количество вхождений элемента в последовательность. Например, мы хотим посчитать, сколько раз встречается единица. Для решения этой задачи мы будем использовать multiset.

multiset <int> s;

int n;

cin >> n;

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

    int x;

    cin >> x;

    s.insert(x);

}

int cnt = 0;

for (auto now = s.lower\_bound(1); now != s.upper\_bound(1); now++) {

    cnt++;

}

cout << cnt;

Здесь появилось два новых метода: lower\_bound и upper\_bound. Lower\_bound возвращает указатель на первый элемент, значение которого больше либо равно переданному параметру. Upper\_bound — на первый элемент, который строго больше. Так мы пробежим от первой единицы до первого элемента (или end’а нашего set’а), на каждом шаге увеличивая значение счётчика вхождений. Если ни одной единицы в последовательности нет, оба метода вернут указатели на больший элемент; будет выполнено 0 шагов.

**Словари**

В C++ есть ещё одна структура, похожая на множество, которая называется «словарь». Она ставит в соответствие ключу значение, совсем как в обычном словаре, где каждому русскому слову ставится в соответствие иностранное.

Рассмотрим такую задачу: на телефон поступает входящий звонок с известного номера телефона. Нужно проверить, есть ли он в телефонной книге. Для этого понадобится словарь, в котором числу (номеру) соответствует строка (имя абонента).

#include <iostream>

#include <map>

#include <string>

using namespace std;

int main() {

      map <int, string> s;

      s[112] = "sos";

      if (s.find(112) != s.end()) {

            cout << "YES\n";

      }

      return 0;

}

Словарь в C++ называется map (карта). Чтобы пользоваться словарями, нужно подключить библиотеку map. Чтобы создать словарь, мы пишем map, затем в треугольных скобках через запятую указываем тип ключа и значения. После этого идёт имя нового словаря.

Создавать элементы в словаре очень просто. Достаточно написать имя словаря, а затем в квадратных скобках указать ключ. Нужно сразу приравнять этот элемент к какому-либо значению. Тогда к нему можно будет обращаться, просто указав имя словаря и ключ в квадратных скобках. Проверка существования элемента делается с помощью метода find, как и во множествах.

**Проход по элементам словаря**

Проход по всем элементам в словаре делается почти так же, как и во множестве. Рассмотрим первый способ:

map <int, string> s;

s[112] = "sos";

s[102] = "emergency";

for (auto now : s) {

    cout << now.first << " " << now.second << "\n";

}

В отличие от множества, в словаре на место now подставляются пары «ключ-значение». Пара — это особый тип данных, состоящий из двух элементов. Обратиться к первому из них можно как к now.first (где now — название пары), а ко второму — now.second. Это обращение к полям очень похоже на вызов метода, но после него не нужно писать скобок. Отдельные элементы пары также можно менять как обычные переменные.

Как и во множестве, ключи в словаре упорядочены по возрастанию. Для поиска ключей можно также пользоваться методами find, lower\_bound и upper\_bound.

**Сопоставление нескольких значений**

Часто требуется сопоставить одному ключу несколько значений. Например, в словаре иностранных слов может быть несколько переводов для одного слова, а в телефонной книге — несколько номеров у одного и того же человека. Чтобы решить задачу такого сопоставления, мы будем использовать в качестве значения вектор.

Вернёмся к примеру с телефонной книгой. Допустим, нам нужно сохранить для одного абонента все телефонные номера (их мы тоже будем хранить в строке).

#include <iostream>

#include <map>

#include <string>

#include <vector>

using namespace std;

int main() {

    map <string, vector <string>> s;

    s["Vasya"] = { "112133", "12341" };

    for (auto now : s["Vasya"]) {

        cout << now << " ";

    }

    return 0;

}

В этой программе мы сразу инициализировали вектор конкретными значениями, используя фигурные скобки. В принципе, можно создать пустой вектор и добавлять в него элементы с помощью метода push\_back. Это удобно в том случае, если мы не знаем заранее, сколько номеров в нашей телефонной книге может быть сопоставлено человеку. Если ключ ещё не встречался, нужно создать пустой вектор, а при каждом его обнаружении просто добавлять элемент в вектор.