Летняя школа по компьютерным наукам – 2016
НИУ ВШЭ, Центр «Стратегия»
Параллель В, занятие 1: Сортировки и их применение
Автор: Михаил Густокашин

Сортировка

Возьмем последовательность чисел, которую нужно считать, упорядочить и вывести. Вот как решается эта задача:

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    int n;
    cin >> n;
    vector <int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    sort(a.begin(), a.end());
    for (auto now : a) {
        cout << now << " ";
    }
    return 0;
}
```

В этой программе есть функция sort. Она принимает на вход два параметра: начало и конец сортируемой области. В нашем случае это начало вектора (begin) и его конец (end).Функцию sort можно применять для векторов, которые состоят из сравнимых величин: целых и вещественных чисел, строк и чего угодно другого, для чего можно ввести операцию «меньше».

Сортировка пар

Очень часто требуется отсортировать пары элементов. Мы уже столкнулись с парами во время изучения словарей, а сейчас остановимся на них подробнее. Для их использования нужно подключить библиотеку utility.

Например, мы хотим отсортировать пары «число — номер в исходной последовательности». В качестве ответа нужно вывести позиции отсортированных элементов в исходном массиве.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <utility>

using namespace std;

int main() {
    int n;
    cin >> n;
    vector <pair <int, int>> a(n);
    for (int i = 0; i < n; i++) {
        int temp;
        cin >> temp;
        a[i] = {temp, i}; // создание пары значение - номер
    }
    sort(a.begin(), a.end());
    for (auto now : a) {
        cout << now.second << " ";
    }
    return 0;
}
```

Чтобы создать пару, пишем ключевое слово pair, затем в треугольных скобках указываем через запятую два типа (для полей first и second соответственно). Ещё один интересный трюк — это создание пары из двух значений. Достаточно в фигурных скобках написать оба значения через запятую, чтобы получилась. Отличие от предыдущей задачи состоит в том, что мы выводим только поля second — номера элементов в исходной последовательности.

При сравнении двух пар сначала сравниваются первые элементы, а если они равны — то вторые. Как время: сначала по часам, затем по минутам.

Сортировка по убыванию

Сортировать по убыванию очень просто — нужно сначала выстроить массив по возрастанию, а потом его развернуть. Для этого используется функция reverse. В ней задаются те же параметры, что и у sort — начало и конец области.

Структуры

Не все сложные объекты можно описать с помощью пар. Чтобы описывать объекты, которые характеризуются несколькими разными значениями, нужны структуры. Фактически, структура — это новый тип переменных. Сначала нужно описать структуру, а после этого можно создавать переменные, вектора и прочие элементы такого типа. Описание структуры должно следовать сразу после using namespace std и находиться вне функций. Например, мы хотим описать человека при помощи двух характеристик: рост и имя. Структура создается так:

```
struct man {
    int height;
    string name;
};
```

Сначала пишем ключевое слово struct, затем — название структуры, а в фигурных скобках перечислим поля структуры вместе с указанием типа каждого поля. Обратите внимание на точку с запятой после фигурной скобки — она обязательно нужна.

Посмотрим полное решение задачи, в которой нужно считать информацию о людях, упорядочить их по росту и вывести имена.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>

using namespace std;

struct man {
    int height;
    string name;
};

bool cmp(man &a, man &b) {
```

```
    return a.height < b.height;
}

int main() {
    int n;
    cin >> n;
    vector <man> a(n);
    for (int i = 0; i < n; i++) {
        int temp;
        string s_temp;
        cin << temp << s_temp;
        man struct_temp; // временная структура
        struct_temp.height = temp;
        struct_temp.name = s_temp;
        a[i] = struct_temp; // создание пары значение - номер
    }
    sort(a.begin(), a.end(), cmp);
    for (auto now : a) {
        cout << now.name << endl;
    }
    return 0;
}
```

В этой программе, во-первых, мы научились создавать переменные для описания людей, а также вектор из таких переменных. Во-вторых, теперь мы можем обращаться к отдельным полям структуры. Как и в парах, это делается через точку, то есть first и second в паре — это тоже поля структуры. И, наконец, мы научились сортировать структуры. Для этого в программе используется функция sort с тремя параметрами. Первые два из них обозначают начало и конец сортируемой области, а третий — имя функции для сравнения. В нашем случае это функция cmp, которая принимает на вход две наших структуры и возвращает истину, если первая должны стоять в отсортированном массиве раньше второй. Если заменить в ней знак «меньше» на «больше», то массив будет отсортирован по убыванию.

Устойчивая сортировка

Сортировка называется устойчивой, если она сохраняет взаимный порядок одинаковых элементов. Если Вася и Петя одного роста, но в исходной

последовательности Вася стоял раньше Пети, то после устойчивой сортировки Вася также должен идти перед Петей.

При использовании sort взаимный порядок одинаковых элементов может нарушиться (Петя окажется перед Васей). Чтобы этого не произошло, нужно использовать функцию устойчивой сортировки stable_sort. Она принимает те же параметры, что и sort, но работает немного медленнее и потребляет больше памяти.

Сканирующая прямая

Очень часто в олимпиадных задачах возникает необходимость обработать некоторые события по порядку. В этом случае события упорядочиваются по оси «времени» и сканирующая прямая движется вдоль этой прямой, переходя от события к следующему. Рассмотрим, например, такую классическую задачу: на прямой задано N отрезков двумя числами: координатами начала и конца отрезка. Требуется определить, какое максимальное количество отрезков покрывает некоторую точку.

Будем двигаться по «событиям» (в нашем случае событием считается достижение точки, которая является началом или конца отрезка) слева направо. Если очередная точка – начало отрезка, то прибавим к счетчику начавшихся, но не кончившихся отрезков единицу. Если же очередная точка – конец отрезка, то вычтем из счетчика единицу. В случае, если в одной точке происходит несколько событий, то будем учитывать сначала начала отрезков, а затем их концы: в нашей задаче точка считается покрытой отрезком, если она покрыта хотя бы одной его точкой, а значит, в случае если в некоторой точке предыдущий отрезок кончается, а новый – начинается, то необходимо учесть, что точка покрыта двумя отрезками. В других задачах упорядочивания одновременных событий определяется из условия.

Для реализации такого решения, необходимо «слить» координаты начал и концов отрезков в один массив, при этом дополнив их признаком того, является ли точка началом или концом отрезка. Затем этот массив сортируется в первую очередь по координате, а в случае их совпадения– по признаку начала или конца. После этого необходим цикл, который проходит по всем упорядоченным точкам и прибавляет единицу к счетчику, в случае если очередная точка имеет признак начала и вычитает единицу, если очередная точка является концом отрезка. Достигнутый во время этого прохода максимум и будет являться ответом на задачу.

Задача становится интереснее, если кроме самого максимального количества отрезков, которыми покрыта точка, необходимо вывести и номера этих отрезков.

Рассмотрим решение «в лоб». Список начавшихся, но не закончившихся, отрезков легко поддерживать в виде булевского массива. Однако при нахождении очередного максимума надо совершить проход по этому массиву, чтобы сохранить ответ. Один проход по такому массиву займет O(N) времени, ответ также может обновляться порядка N раз (когда все отрезки последовательно вложены друг в друга). Таким образом, сложность такого решения составит $O(N^2)$, что слишком много для большинства задач такого рода.

Более разумно сделать двухпроходное решение. Первый проход будет решать исходную задачу – искать максимальное количество отрезков, покрывающих точку. Второй проход будет незначительно отличаться от первого, а именно: когда значение счетчика начавшихся, но не закончившихся отрезков станет равно максимальному значению, следует прекратить дальнейший поиск и пройти по булевскому массиву, в котором помечены эти отрезки и вывести их. Сложность такого решения составит O(N). Первый и второй проход займут порядка N операций, проход по массиву открытых отрезков также займет O(N) операций и будет выполнен один раз.

Метод двух указателей

Рассмотрим метод двух указателей на примере такой задачи: дана последовательность из N чисел, нужно найти в ней несколько подряд идущих чисел, чтобы их сумма была больше либо равна K, а количество чисел было минимальным.

Простое решение представляет собой два цикла, перебирающих левую и правую границу отрезка. Сложность такого решения будет $O(N^2)$. Однако, можно заметить, что при сдвиге левой границы на единицу вправо правая граница может либо остаться на том же месте, либо сдвинутся вправо. Будем хранить сумму на отрезке от левой до правой границы и поддерживать границы таким образом, чтобы эта сумма всегда была больше либо равна K. При сдвиге левой границы элемент будет «выпадать» из окна и мы будем вычитать его значение из суммы. Затем будем двигать правую границу и прибавлять значение очередного добавленного элемента до тех пор, пока не дойдем до конца или сумма не станет больше либо равна K. Суммарно за все время работы программы правая граница сдвинется на N и, таким образом, сложность всего алгоритма будет равна O(N).

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
	int n, k;
	cin >> n >> k;
	vector <int> a(n);
	for (int i = 0; i < n; ++i)
		cin >> a[i];
	long long sum = 0;
	int R = 0;
	int minLen = n + 1;
	for (int L = 0; L < n; ++L) { // перебираем левую границу
		while (R < n && sum < k) { // двигаем правую
			sum += a[R];
			R++;
		}
		if (sum >= k && R - L < minLen) // если сумма подходит и длина минимальная
			minLen = R - L;
		sum -= a[L];
	}
	cout << minLen;
	return 0;
}
```