Разработка и анализ алгоритмов

Лекция 2
Предподсчёт. Рекурсия. Конечный автомат. Два указателя.

Сергей Леонидович Бабичев

Амортизационный анализ. Предподсчёт.

Предподсчёт

Задача. Дан массив A из N элементов. Дано M запросов $R(l, r)$ вида: подсчитать сумму элементов от l_i до r_i включительно, $0 \leqslant l \leqslant r < N$.

- Примитивное решение: Подсчитывать каждый раз запрашиваемую сумму. Сложность алгоритма: по памяти $O(1)$, по времени $O\left(\sum_{i=1}^{N}(r_i - l_i)\right)$.
- Решение с предподсчётом. Сформируем новый массив $B[0 \ldots N]$ из $N + 1$ элементов такого вида:

$$B_k = \begin{cases} 0, & \text{если } k = 0, \\ \sum_{i=0}^{k} A_i, & \text{если } k > 0. \end{cases}$$

Тогда значение запроса $R(l, r)$ есть значение $B_{r+1} - B_l$.
Сложность алгоритма: по памяти $O(N)$, по времени $O(N + M)$.

Предподсчёт

- Амортизированная сложность примитивного алгоритма на одну операцию

$$T_1 = O(\mathrm{Avg}_{i=1,M}\, r_i - l_i).$$

- Амортизированная сложность алгоритма с предподсчётом

$$T_1 = O(1).$$

Рекурсия.

Рекурсивное построение перестановок

- Мы пользовались рекурсией для разбиения задачи на подзадачи.
- Рекурсия — универсальный механизм для решения большого количества задач.

Задача. Построить все перестановки заданного множества S из N элементов.
Решение: Если имеются все перестановки P для n элементов, то получить все перестановки для $n+1$ элемента можно добавив в каждую их перестановок P_n элемент S_{n+1} во всевозможные позиции (которых $n+1$).

- Взяв заданное множество S, мы можем зафиксировать первый элемент, построить на оставшейся части все перестановки подмножества $S' = S \setminus S_1$, затем все перестановки $S' = S \setminus S_2$ и так далее.
- Если всё проводить рекурсивно, то получаем все перестановки S.

Рекурсивное построение перестановок

```
void all_permutations(char *a, int left, int right) {
    if (left == right) puts(a);
    else {
        for (int i = left; i < right; i++) {
            swap(a+left, a+i);
            all_permutations(a, left+1, right);
            swap(a+left, a+i);
        }
    }
}
```

Перебор с возвратом (backtraking)

- Ряд задач явным образом имеет подзадачи, решающиеся тем же способом.
- Для решения задачи требуется решить все подзадачи и обработать их результаты (консолидировать).
- Много таких подзадач будет в следующем семестре в теме «динамическое программирование».

Классическая задача

Задача. Имеется шахматная доска размером $N \times N$. Шахматный конь стоит на поле $a1$. Требуется обойти доску таким образом, чтобы конь побывал на каждом поле ровно по одному разу.

Решение:

- Что есть подзадача? В текущей позиции выполнить очередной ход или обнаружить, что ход невозможен. Если ход не возможен и доска не заполнена, то нужно вернуться назад и сделать другой ход.
- Что есть позиция? Например, массив $N \times N$, содержащий номер хода.
- Как делается ход? Нужная клетка заполняется номером хода.
- Как ход берётся назад? Клетка объявляется незанятой. Мы должны вернуться назад и сделать очередной ход, который мы не рассматривали.
- Что нужно помнить? Историю продвижения к цели.
- Какова сложность? Большая. Неполиномиальная.

Автоматы.

Понятие автомата

- *Автоматы* — произведение множеств *состояний* P и *переходов* T.
- Имеются *начальное состояние автомата* и *заключительное состояние*.
- *Конечный автомат* — автомат с ограниченными множествами состояний и переходов.
- *Вход автомата* — события, вызывающие переходы.
- *Детерминированный конечный автомат* — конечный автомат, в котором одна и та же последовательность входных данных приводит при одном и том же начальном состоянии к одному и тому же заключительному.

Применение автоматов

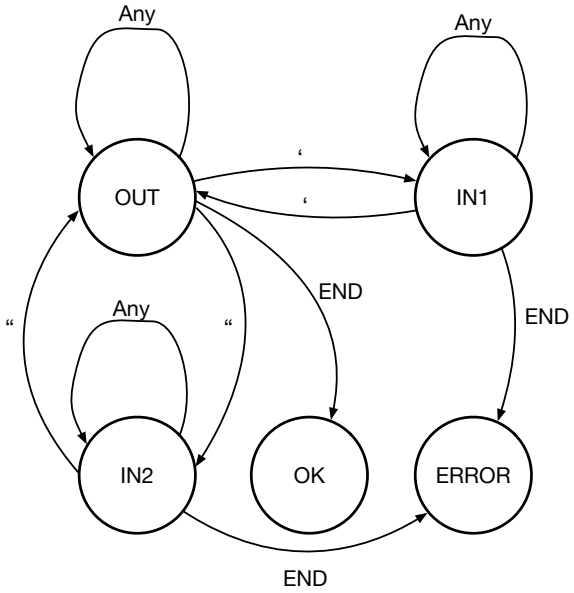
Задача. На вход алгоритма подаётся последовательность символов. Назовём *строкой* любую подпоследовательность символов, начинающуюся на знак одиночной кавычки или двойной кавычки и заканчивающейся ей же. Внутри строк могут находиться любые символы, кроме завершающего.
Нужно определить корректность входной последовательности.
Примеры:

```
'abracadabra' - OK
'abra"shvabra cadabra' - OK
"" - OK
"abra'shravra' - Fail
```

Традиционный способ решения

```
int check(const char *s) {
    const char *ps = s;
    while (*ps != 0) {
        if (*ps == '\'') {
            while (*++ps != 0 && *ps != '\'') {}
            if (*ps == '\'') ps++;
            else if (*ps == 0) return 0;
        } else if (*ps == '"') {
            while (*++ps != 0 && *ps != '"') {}
            if (*ps == '"') ps++;
            else if (*ps == 0) return 0;
        } else {
            ps++;
        }
    }
    return 1;
}
```

Конечный автомат



Конечный автомат

```
int DFA(const char *s) {
    enum {OUT, IN1, IN2} state = OUT;
    for (const char *ps = s; *ps != 0; ps++) {
        if (state == IN1 && *ps == '\'') state = OUT;
        else if (state == IN2 && *ps == '"')  state = OUT;
        else if (state == OUT && *ps == '\'') state = IN1;
        else if (state == OUT && *ps == '"')  state = IN2;
    }
    return state == OUT;
}
```

Конечный автомат

Можно построить таблицу переходов.

		Any	'	"	END
OUT		OUT	IN1	IN2	OK
IN1		IN1	OUT	IN1	ERROR
IN2		IN2	IN2	OUT	ERROR

Два указателя.

Два указателя

- Алгоритмы двух указателей напоминают алгоритмы конечных автоматов.
- Конечный автомат последовательно обходит последовательность, изменяя состояния автомата.
- Два указателя последовательно обходят одну или две последовательности, изменяя состояния автомата.
- После начального состояния проверяются условия. Проверка условия приводит либо к продвижению первого указателя, либо к продвижению второго.
- Алгоритм заканчивается либо по достижению результата, либо по завершению продвижения указателей.

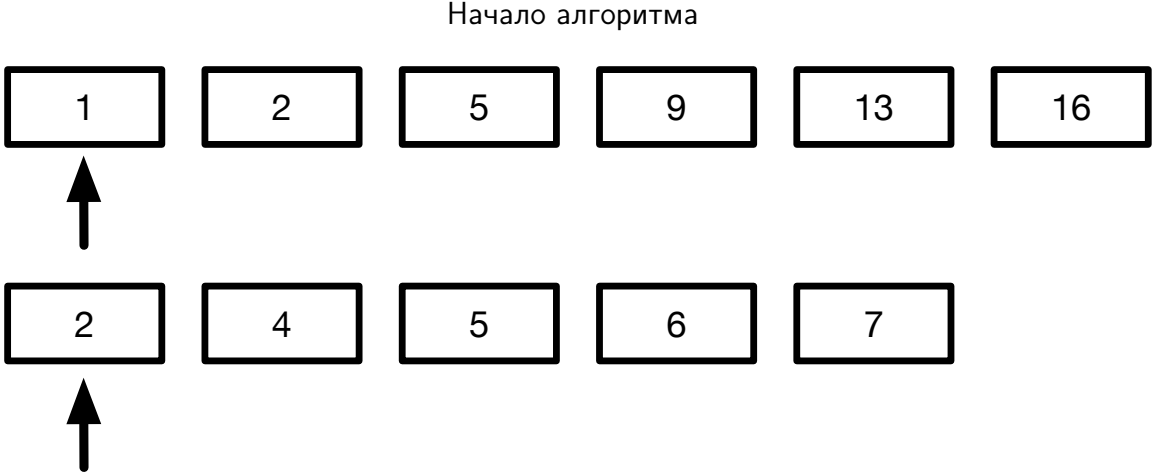
Симметрическая разность множеств

Задача. Над элементами двух множеств A и B определена операция сравнения и они заданы своими элементами, расположенными согласно этой операции. Имеется операция нахождения следующего элемента множества, исполняемая за время $O(f(|A|)$. Требуется за время $O(|A| \cdot f(|A|) + |B| \cdot f(|B|))$ найти множество $C = A \triangle B$.

Решение: Воспользуемся алгоритмом двух указателей.

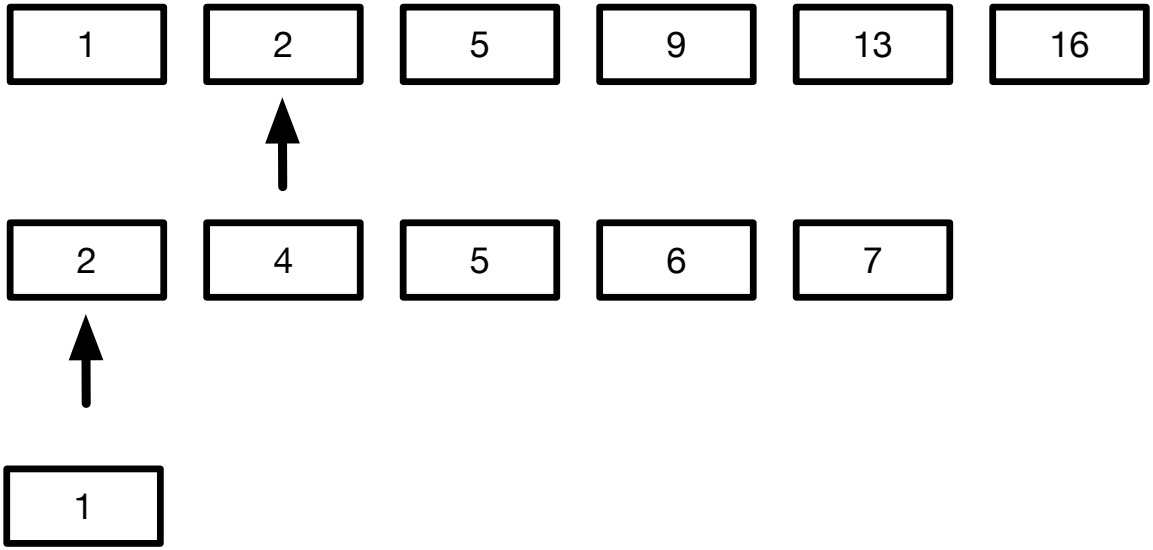
1. Найдём первые элементы множества и установим на них виртуальные указатели.
2. Если в каком-то множестве обработаны все элементы — добавить в C необработанные элементы другого множества и затем завершить алгоритм.
3. Сравним элементы. Тот элемент, которые меньше отправим элемент в C и продвинем соответствующий указатель.
4. Если элементы равны — продвинем оба указателя. Возвращаемся к п. 3.

Симметрическая разность множеств



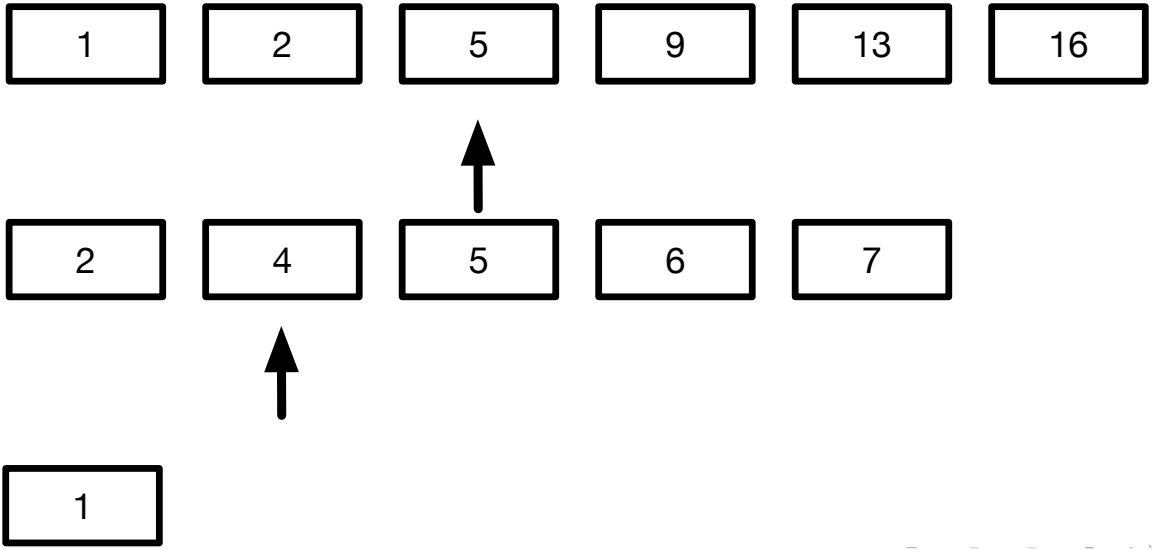
Симметрическая разность множеств

Элемент первого множества меньше — отправляем в C.



Симметрическая разность множеств

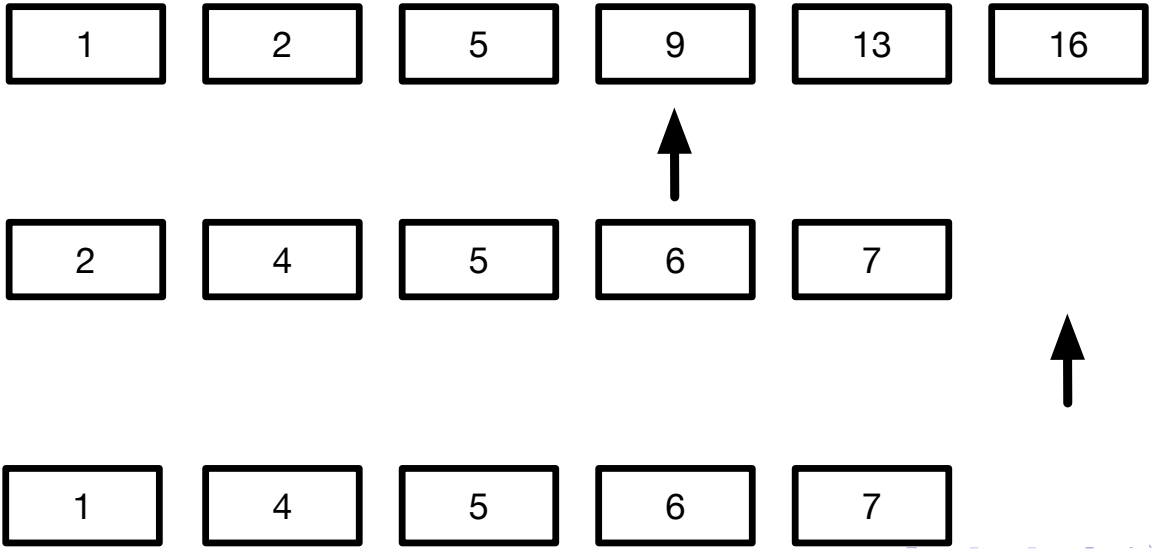
Передвинули указатель первого множества.



Симметрическая разность множеств

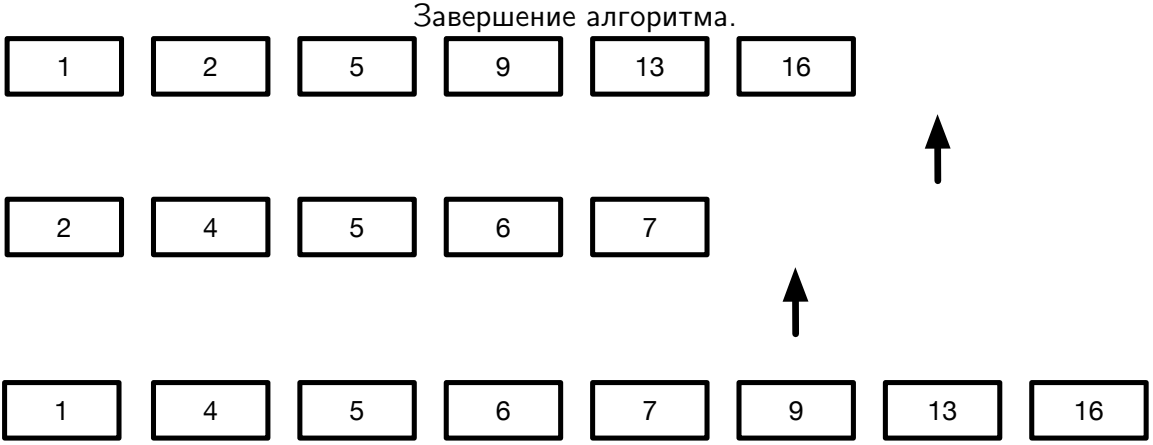
Передвинули указатель первого множества.

Одно множество исчерпано — будем копировать остаток другого.



Симметрическая разность множеств

Передвинули указатель первого множества.



Симметрическая разность множеств — сложность

- Каждый элемент множества S потребовал ровно $|S|$ операций поиска следующего, каждая со сложностью $O(f(|S|))$.
- Общая сложность совпадает с требуемой.