Разберём ещё одну задачу, решаемую жадным алгоритмом, — задачу о выборе заявок.

Задача о выборе заявок
Имеется аудитория и n заявок на проведение занятий в ней. Каждая заявка — это временной интервал $[l_i, r_i)$, причём, как обычно бывает при 0-индексации, левая граница включается, а правая — нет. Нужно выбрать наибольшее число заявок, чтобы они не пересекались.

На рис. 1 приведён пример с тремя заявками. Здесь оптимальным решением будет выбрать две заявки, одна из которых начинается в тот момент, когда другая заканчивается.

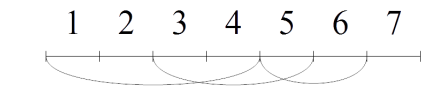

Рис. 1. Пример к задаче о выборе заявок

Чтобы решить задачу жадным алгоритмом, нужно действовать оптимально на каждом шаге. Рассмотрим ситуацию на рис. 2. Обе эти заявки мы взять не сможем. Если мы просматриваем заявки слева направо, нам выгоднее брать заявку, которая раньше заканчивается. Это может помочь взять какие-то заявки в дальнейшем.

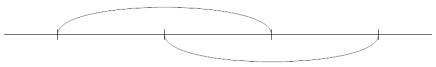

Рис. 2

Из этих рассуждений получается жадный алгоритм: сортируем заявки по возрастанию правой границы, идём по заявкам слева направо и берём текущую заявку, если она не пересекается с предыдущей взятой.

Разберём код программы на рис. 3.

Здесь cnt — это количество удовлетворённых заявок, last — последняя удовлетворённая заявка. Мы считаем, что заявки уже отсортированы по возрастанию правой границы. Сначала мы удовлетворяем самую первую заявку с индексом 0:

```
int cnt = 1;
int last = 0;

for (int i = 1; i < n; i++)
    if (l[i] >= r[last])
    {
        cnt++;
        last = i;
    }
```

Рис. 3. Решение задачи о выборе заявок

int cnt = 1;
int last = 0;

Затем проходим циклом по остальным заявкам, начиная с номера 1.

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

Проверяем: если левая граница новой заявки больше или равна правой границе прошлой заявки, мы можем добавить к ответу новую заявку.

if (l[i] >= r[last])

Для этого мы увеличиваем количество заявок в ответе cnt.

cnt++;

И изменяем номер последней заявки:

last = i;

В результате в переменной cnt будет наибольшее число заявок, которое мы можем удовлетворить. Если нужно запоминать сами удовлетворённые заявки, то это тоже нетрудно сделать.

Оценим время работы алгоритма. Код на рис. 3 выполняет $O(n)$ операций: в нём только один цикл до n. Ещё $O(n \log n)$ операций требуется для сортировки. Получаем общее время $O(n \log n)$.

Ответим на вопрос о том, почему жадное решение правильно работает и всегда находит максимальный ответ. Предположим, что это не так, и воспользуемся методом минимального контрпримера. Пусть существут пример, на котором жадное решение не совпадает с оптимальным, причём выберем пример с минимальным числом заявок. Рассмотрим жадное решение и оптимальное. Первая заявка в жадном решении — та, которая раньше всех заканчивается. Предположим, что в оптимальном решении её нет. Тогда первая заявка в оптимальном решении заканчивается позже или, во всяком случае, не раньше заявки из жадного решения, и мы можем заменить заявку из оптимального решения на заявку из жадного. Ясно, что такая замена не добавит пересечения с другими заявками, и оптимальное решение останется оптимальным. После этого мы можем удалить первую заявку и перейти к примеру с меньшим количеством заявок, на котором жадное решение не совпадает с оптимальным. Но

наш контрпример по предположению был минимальным, а значит, мы пришли к противоречию. Мы доказали, что жадное решение этой задачи всегда оптимальное.

Таким образом, мы разобрали жадное решение задачи о выборе заявок и доказали его оптимальность.