Жадные алгоритмы: введение

Александр Куликов

Онлайн-курс «Алгоритмы: теория и практика. Методы»
http://stepic.org/217

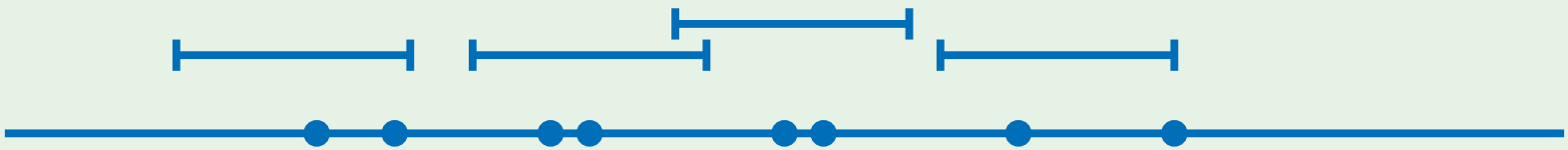
Покрытие точек отрезками

Вход: множество n точек на прямой $x_1, \ldots, x_n \in \mathbb{R}$.

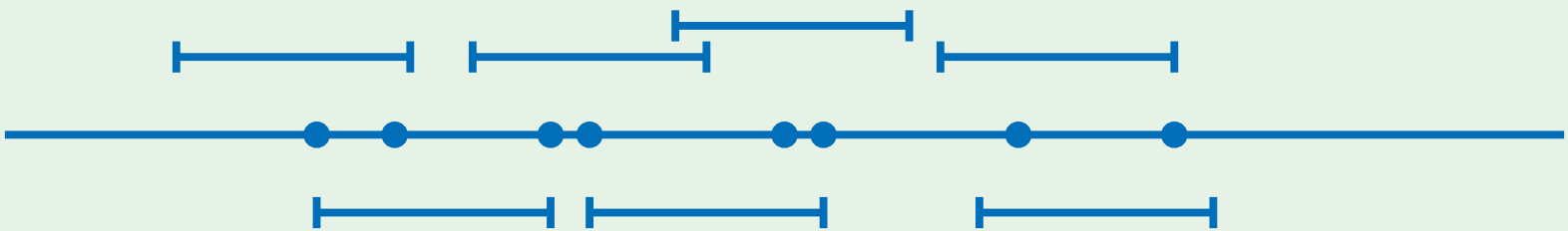
Выход: минимальное количество отрезков единичной длины, которыми можно покрыть все точки.

Пример

Пример

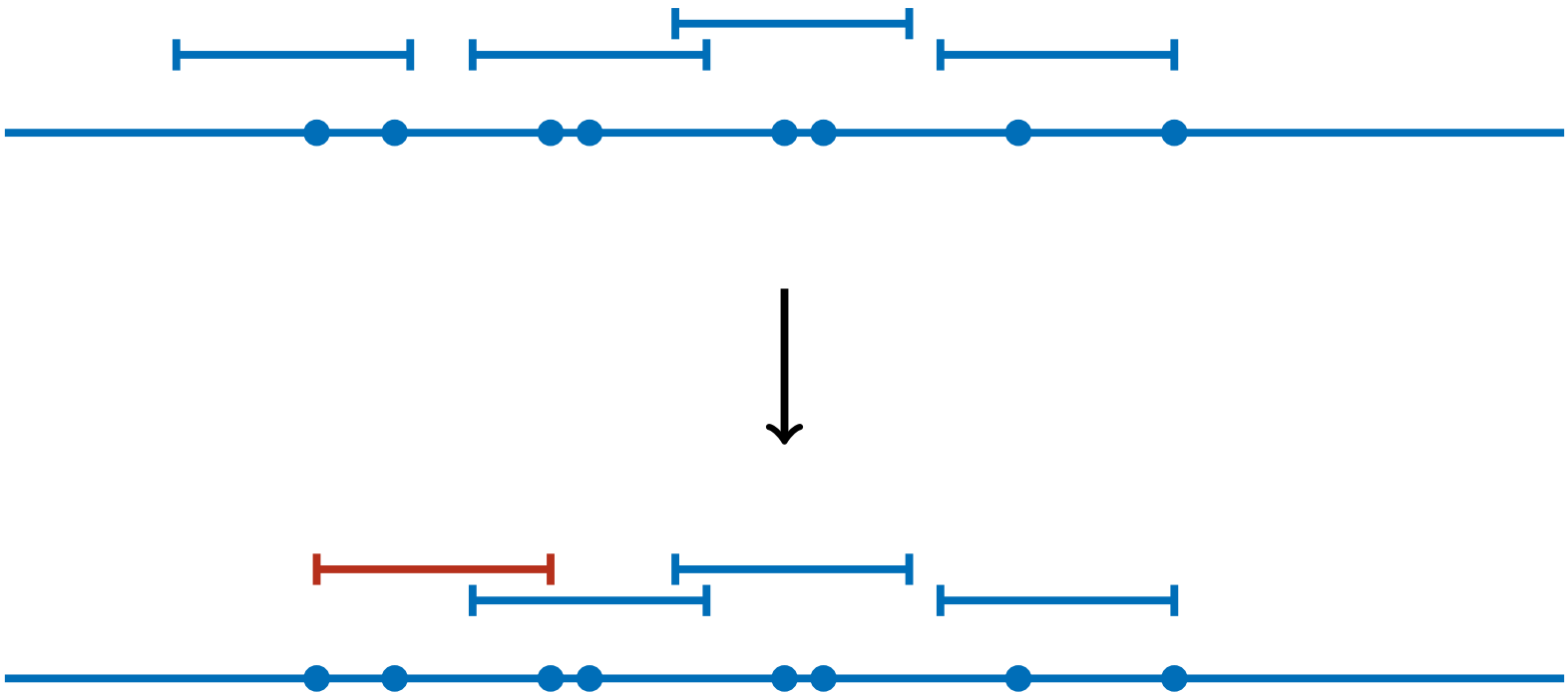


Пример



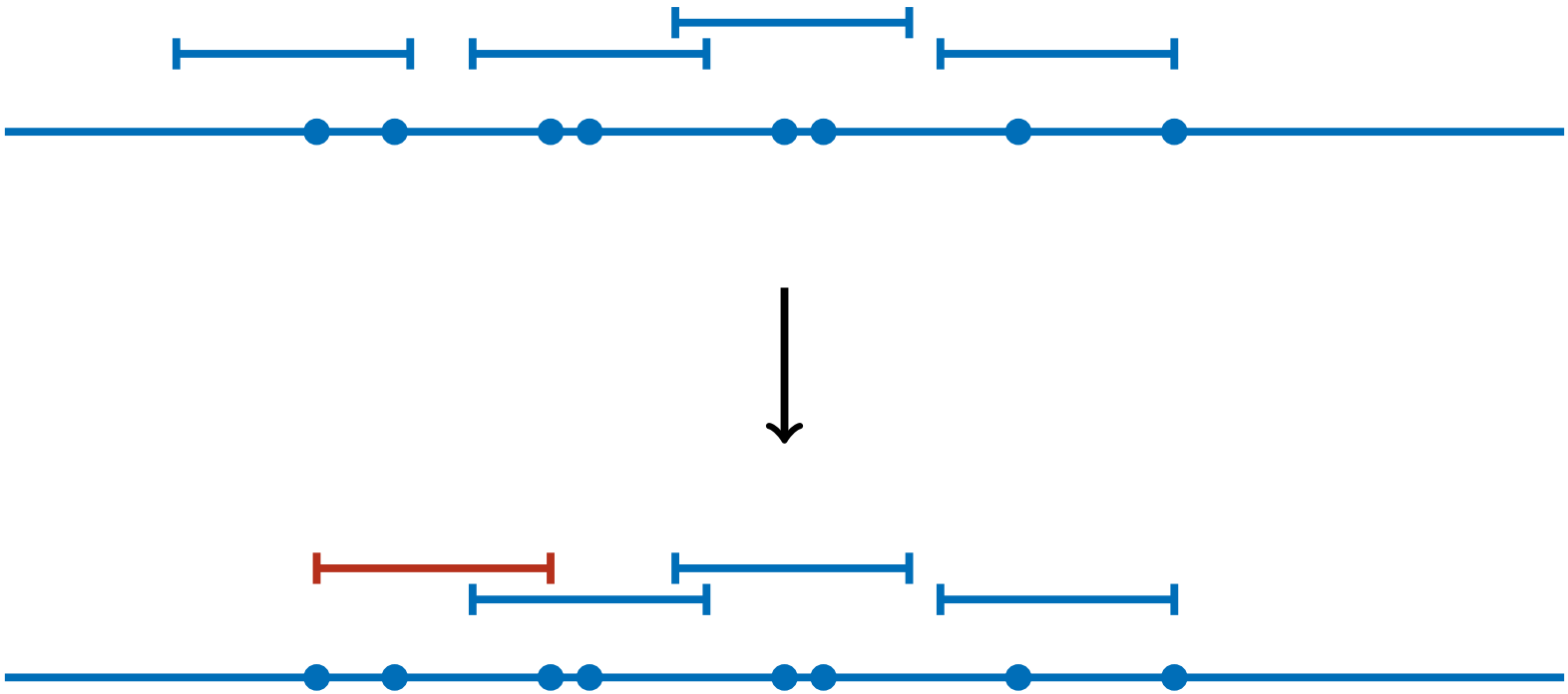
Надёжный шаг

Существует оптимальное покрытие, в котором самая левая точка покрыта левым концом отрезка.



Надёжный шаг

Существует оптимальное покрытие, в котором самая левая точка покрыта левым концом отрезка.



Поэтому можно сразу добавить в решение отрезок, левый конец которого совпадает с самой левой точкой.

Алгоритм

Функция PointsCover($x_1, \ldots, x_n$)

$S \leftarrow \{x_1, \ldots, x_n\}$
пока S не пусто:
  $x_m \leftarrow$ минимальная точка S
  добавить к решению отрезок $[\ell, r] = [x_m, x_m + 1]$
  выкинуть из S точки, покрытые отрезком $[\ell, r]$
вернуть построенное решение

Алгоритм

Функция PointsCover($x_1, \ldots, x_n$)

$S \leftarrow \{x_1, \ldots, x_n\}$
пока S не пусто:
  $x_m \leftarrow$ минимальная точка S
  добавить к решению отрезок $[\ell, r] = [x_m, x_m + 1]$
  выкинуть из S точки, покрытые отрезком $[\ell, r]$
вернуть построенное решение

Время работы: $O(n^2)$.

Улучшенный алгоритм

Функция PointsCover($x_1, \ldots, x_n$)

$x_1, \ldots, x_n \leftarrow \text{Sort}(x_1, \ldots, x_n)$
$i \leftarrow 1$
пока $i \leq n$:
  добавить к решению отрезок $[\ell, r] = [x_i, x_i + 1]$
  $i \leftarrow i + 1$
  пока $i \leq n$ и $x_i \leq r$:
    $i \leftarrow i + 1$
вернуть построенное решение

Улучшенный алгоритм

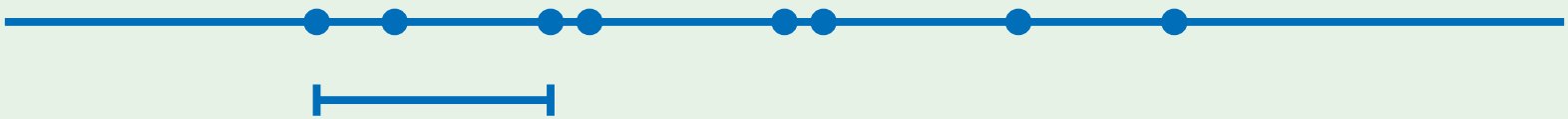
Функция PointsCover($x_1, \ldots, x_n$)

$x_1, \ldots, x_n \leftarrow \text{Sort}(x_1, \ldots, x_n)$
$i \leftarrow 1$
пока $i \le n$:
  добавить к решению отрезок $[\ell, r] = [x_i, x_i + 1]$
  $i \leftarrow i + 1$
  пока $i \le n$ и $x_i \le r$:
    $i \leftarrow i + 1$
вернуть построенное решение

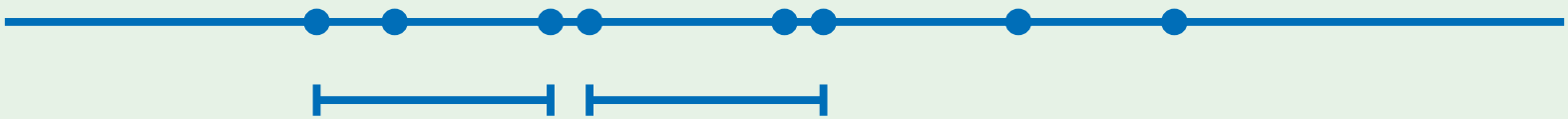
Время работы: $T(\text{Sort}) + O(n) = O(n \log n)$.

Пример

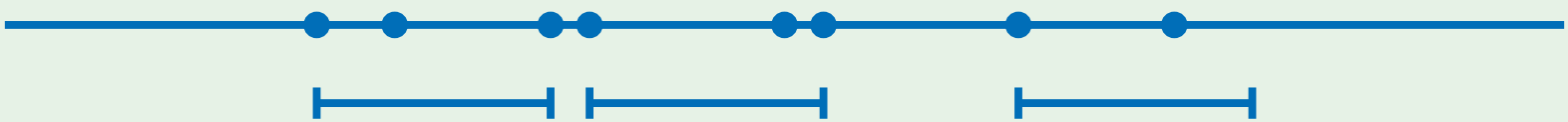
Пример



Пример



Пример

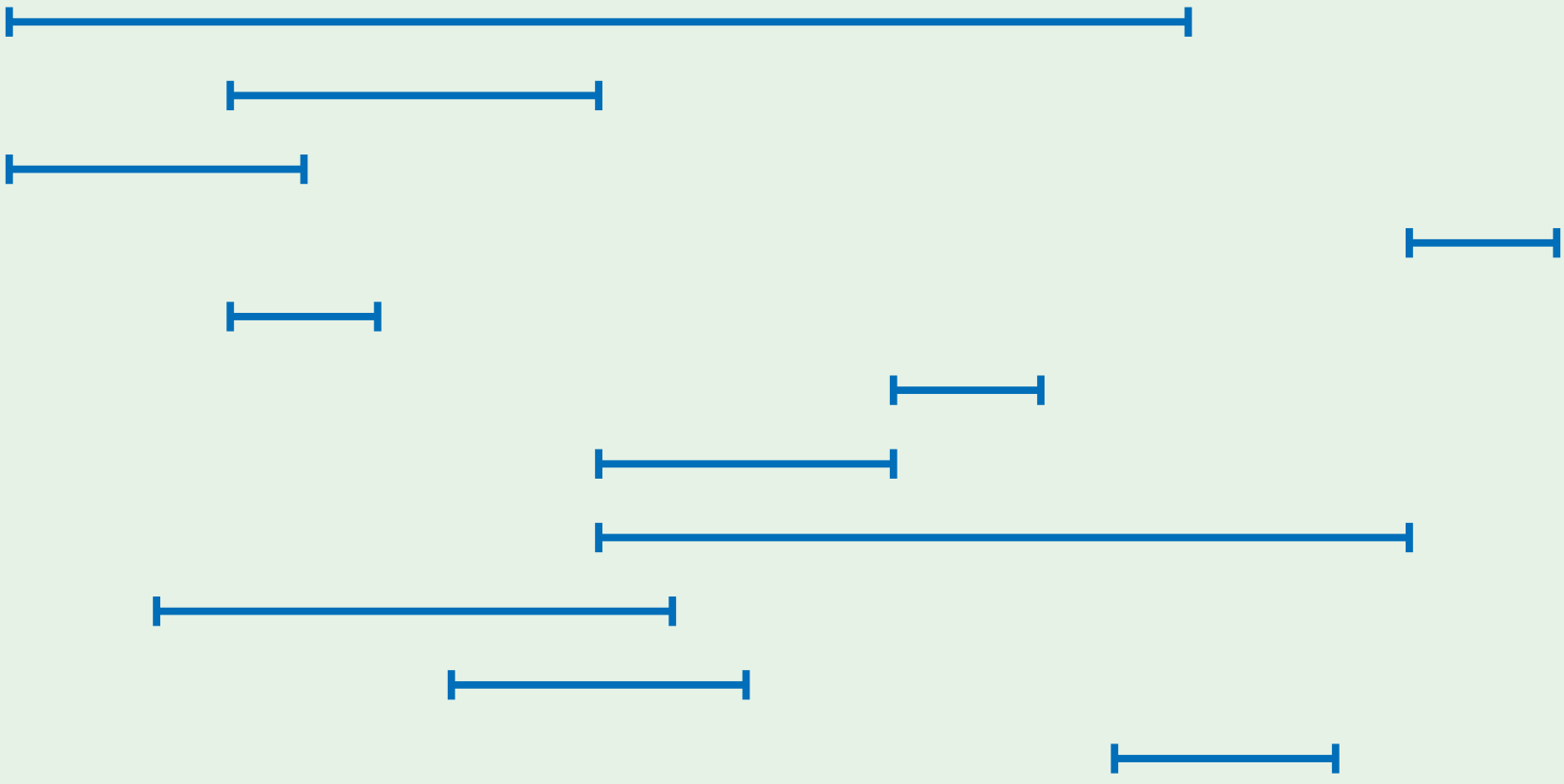


Задача о выборе заявок

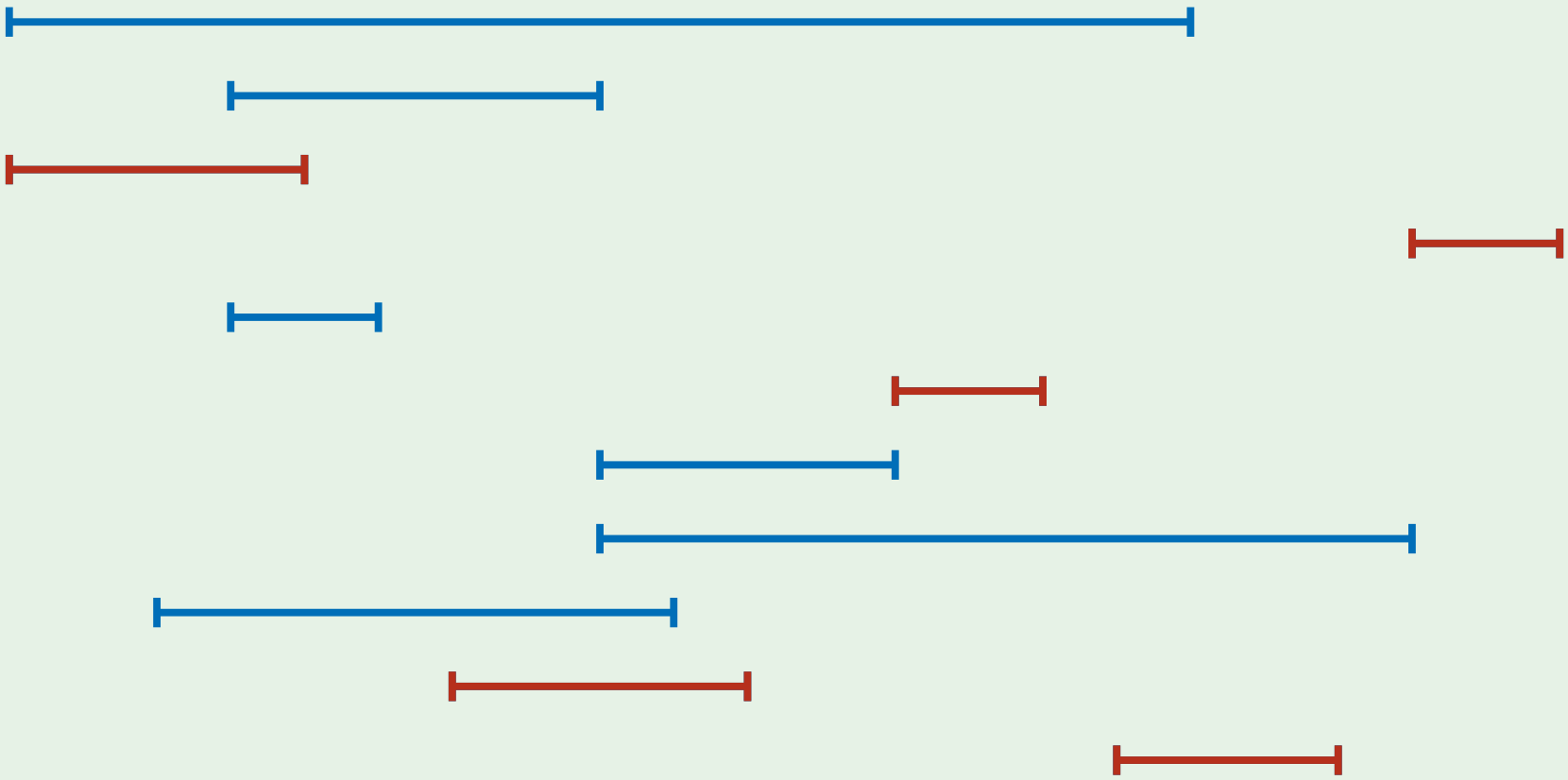
Вход: множество n отрезков на прямой.

Выход: максимальное количество попарно не пересекающихся отрезков.

Пример

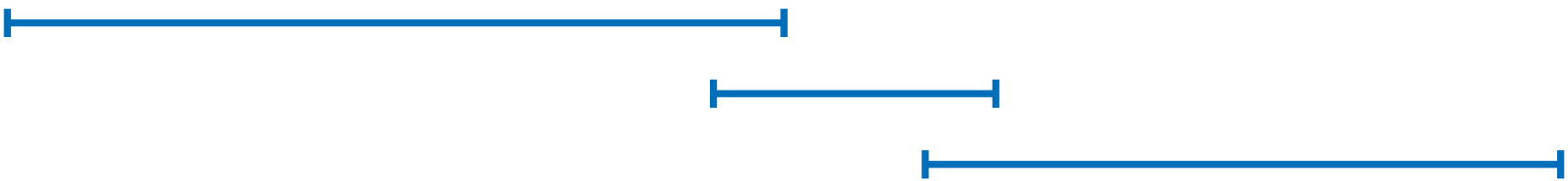


Пример



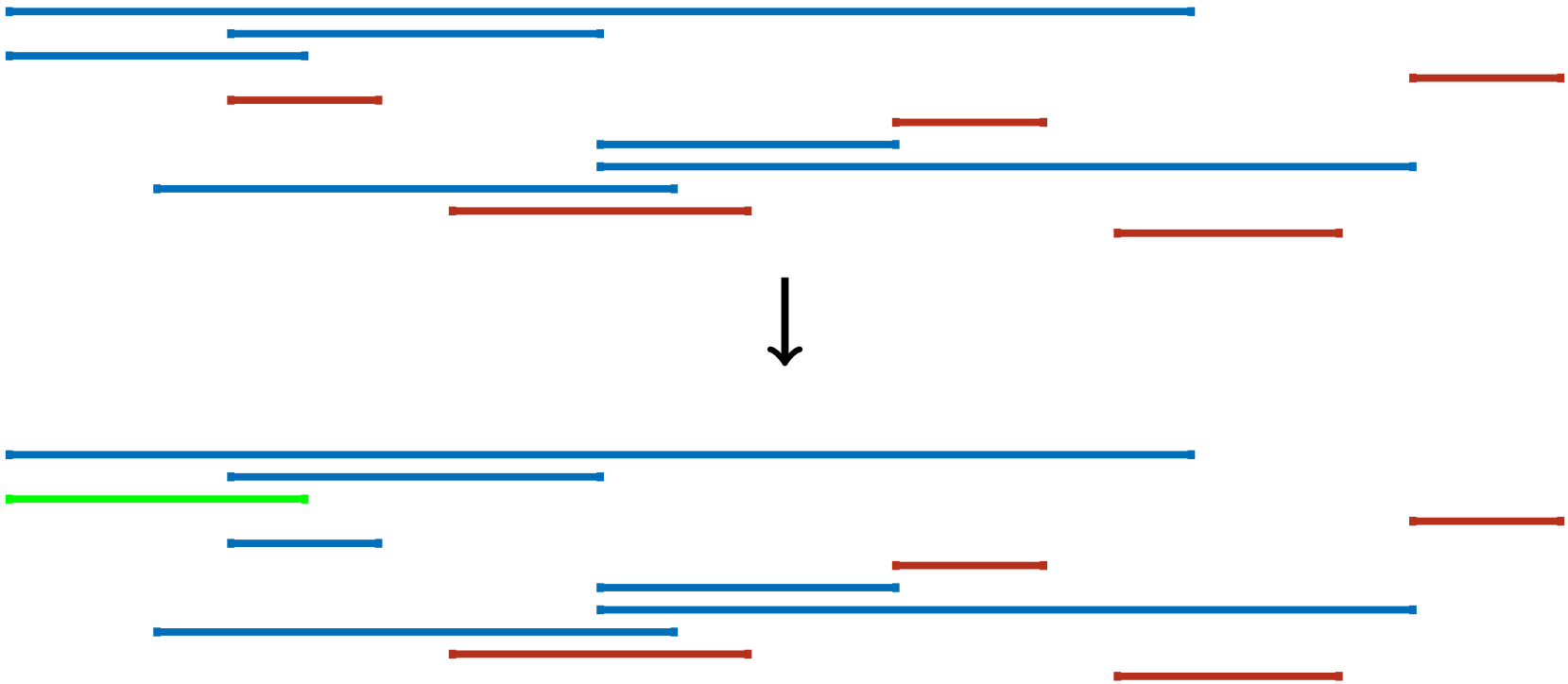
Замечание

Выбирая в первую очередь более короткие отрезки, можно получить неоптимальное решение.



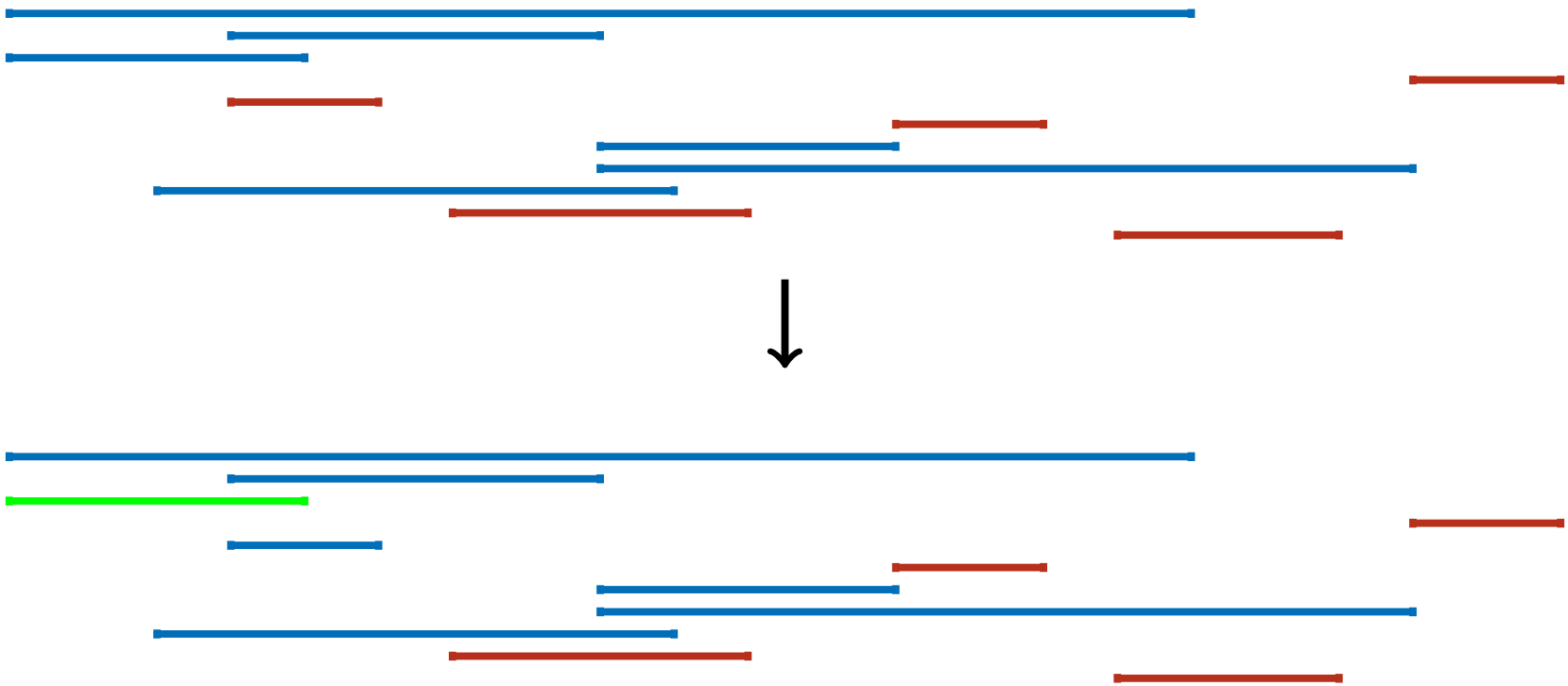
Надёжный шаг

Существует оптимальное решение, содержащее отрезок, правый конец которого минимален.



Надёжный шаг

Существует оптимальное решение, содержащее отрезок, правый конец которого минимален.



Можно сразу добавить в решение отрезок, правый конец которого минимален.

Алгоритм

Функция ACTSEL$(\ell_1, r_1, \ldots, \ell_n, r_n)$

$S \leftarrow \{[\ell_1, r_1], \ldots, [\ell_n, r_n]\}$
пока S не пусто:
  $[\ell_m, r_m] \leftarrow$ отрезок из S с мин. правым концом
  добавить $[\ell_m, r_m]$ к решению
  выкинуть из S отрезки, пересекающиеся с $[\ell_m, r_m]$
вернуть построенное решение

Алгоритм

Функция ACTSEL$(\ell_1, r_1, \ldots, \ell_n, r_n)$

$S \leftarrow \{[\ell_1, r_1], \ldots, [\ell_n, r_n]\}$
пока S не пусто:
  $[\ell_m, r_m] \leftarrow$ отрезок из S с мин. правым концом
  добавить $[\ell_m, r_m]$ к решению
  выкинуть из S отрезки, пересекающиеся с $[\ell_m, r_m]$
вернуть построенное решение

Время работы: $O(n^2)$.

Улучшенный алгоритм

Функция ACTSEL($\ell_1, r_1, \ldots, \ell_n, r_n$)

```
отсортировать n отрезков по правым концам
для всех отрезков в полученном порядке:
  если текущий отрезок не пересекает
    последний добавленный:
    взять его в решение
вернуть построенное решение
```

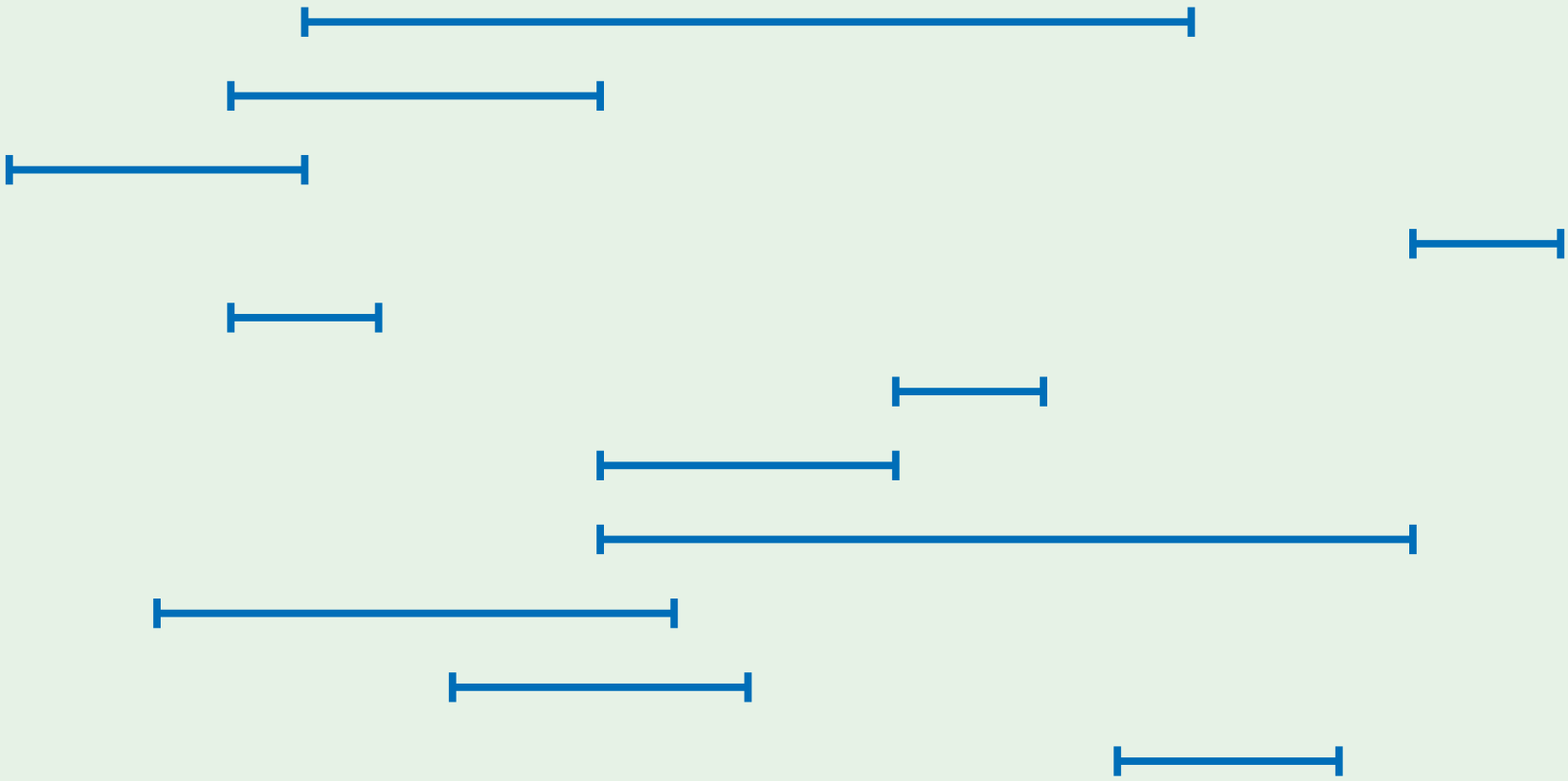
Улучшенный алгоритм

Функция ACTSEL$(\ell_1, r_1, \ldots, \ell_n, r_n)$

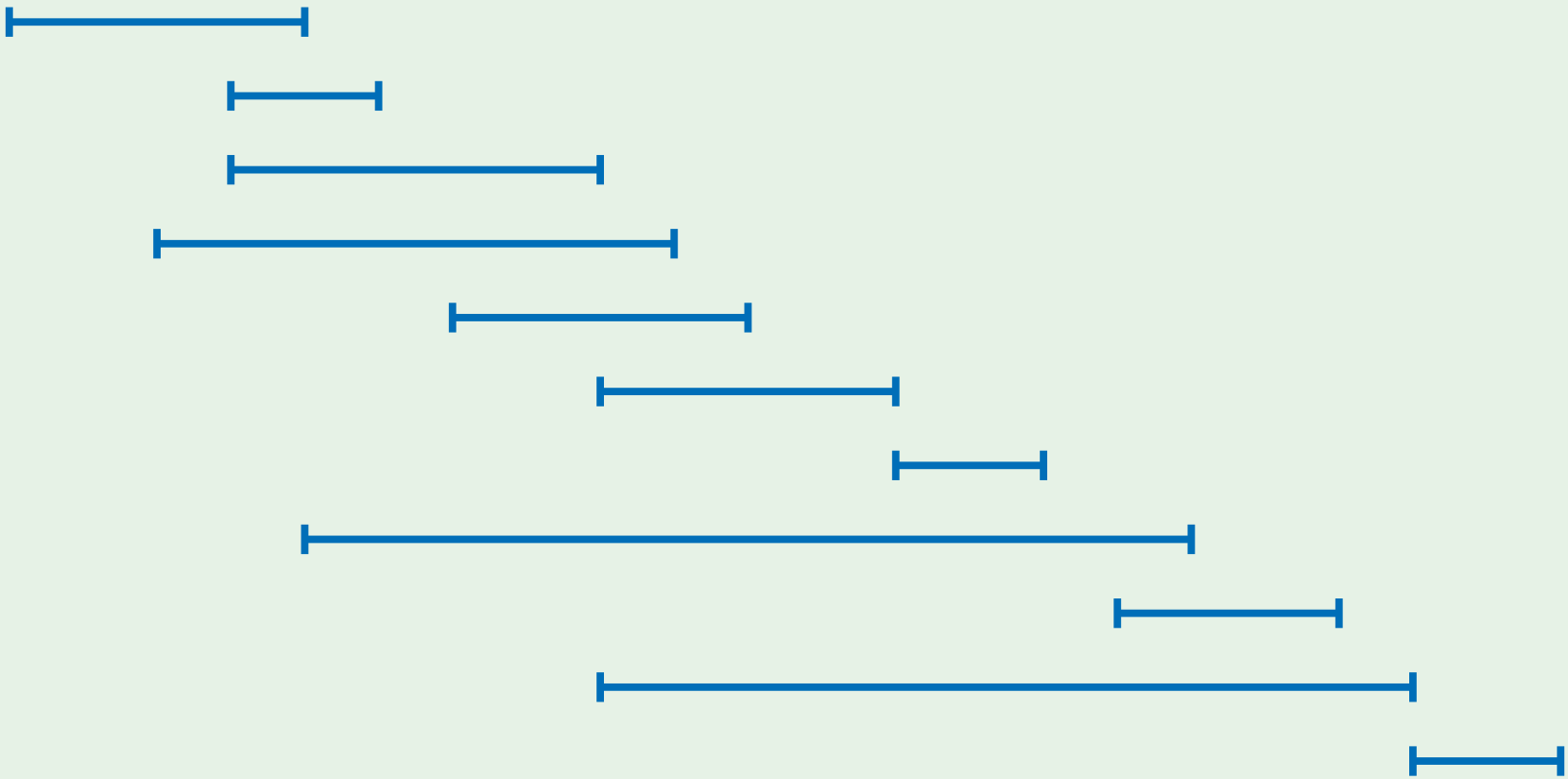
```
отсортировать n отрезков по правым концам
для всех отрезков в полученном порядке:
  если текущий отрезок не пересекает
    последний добавленный:
    взять его в решение
вернуть построенное решение
```

Время работы: $T(\text{SORT}) + O(n) = O(n \log n)$.

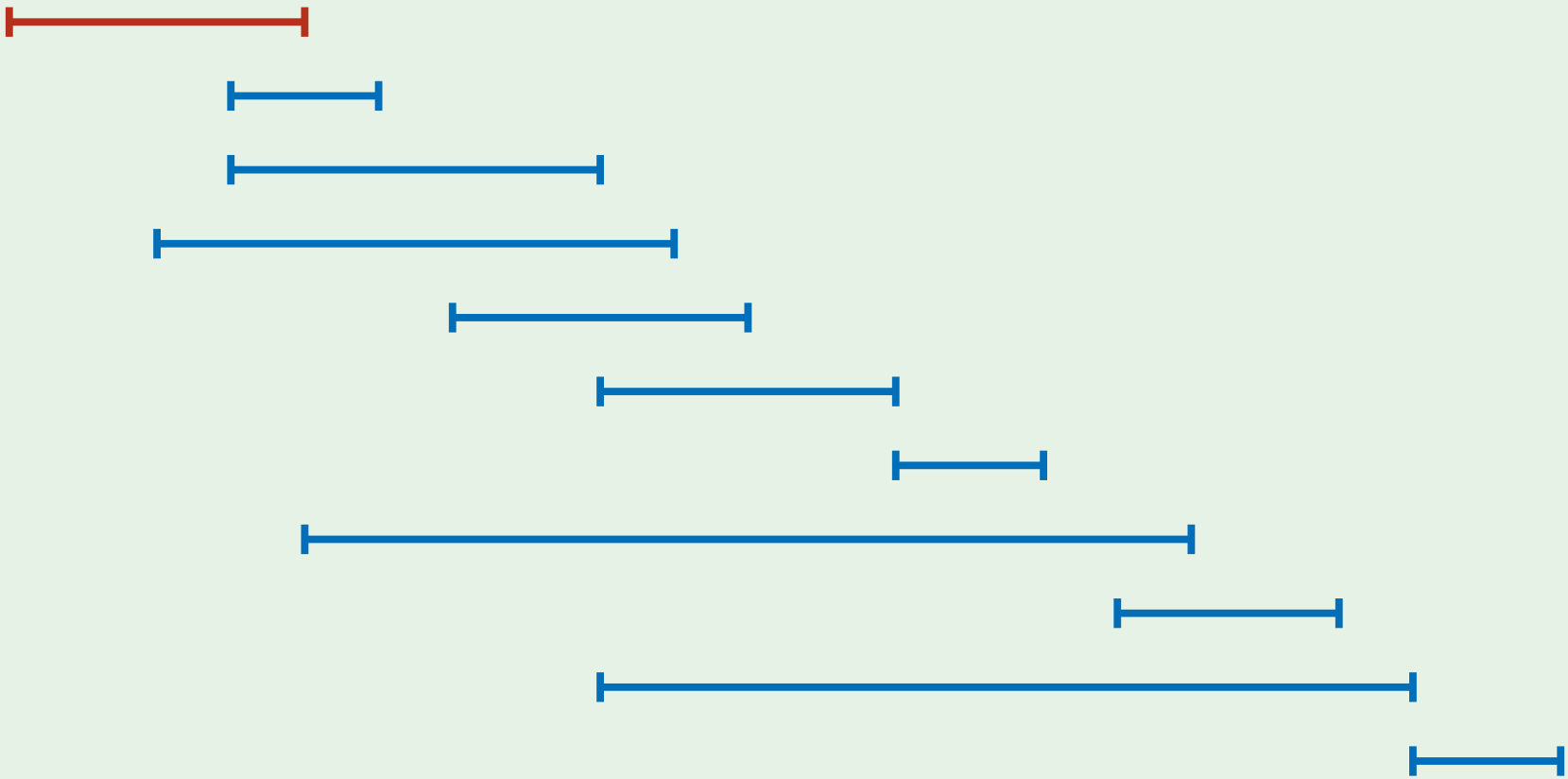
Пример



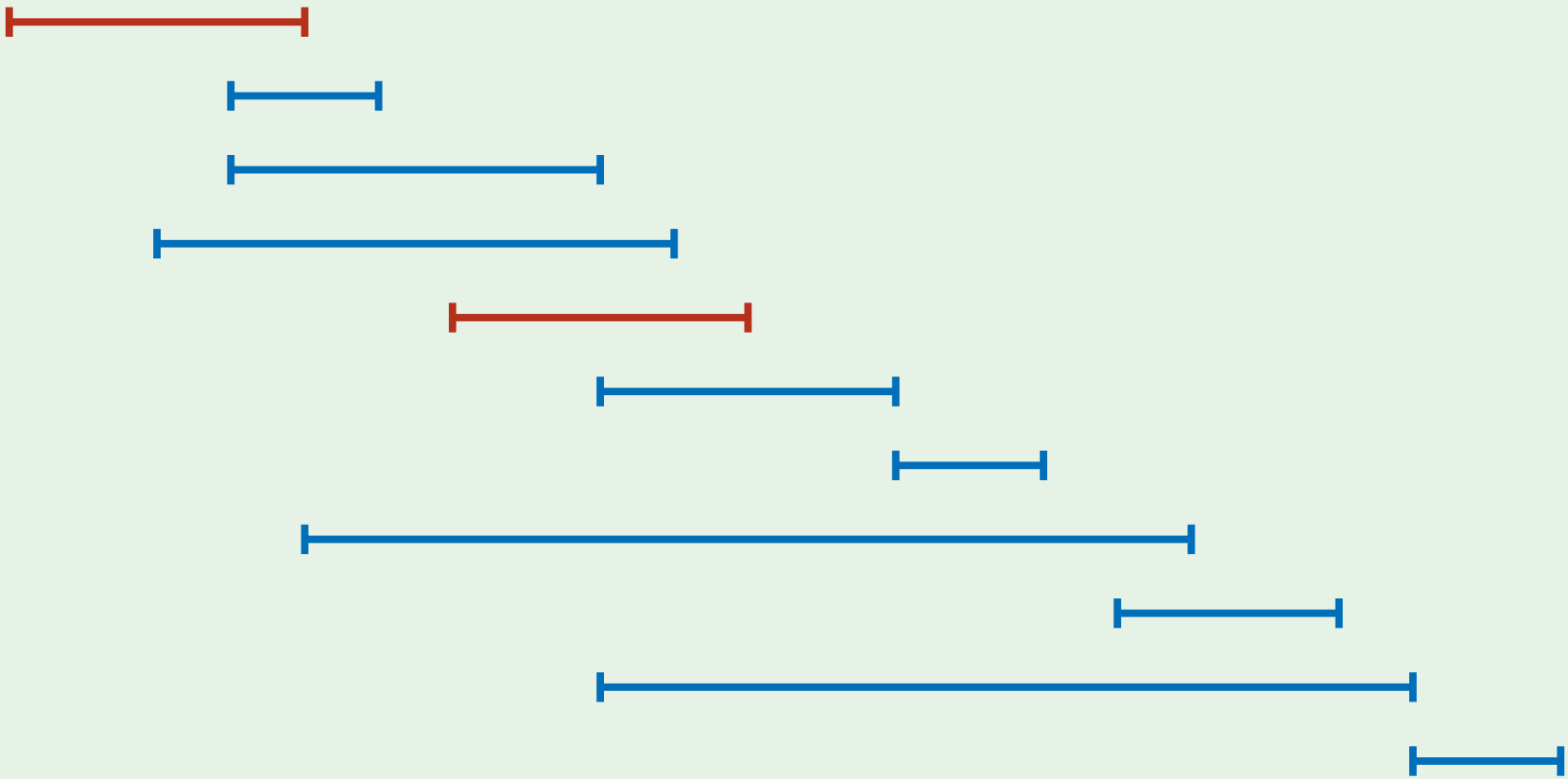
Пример



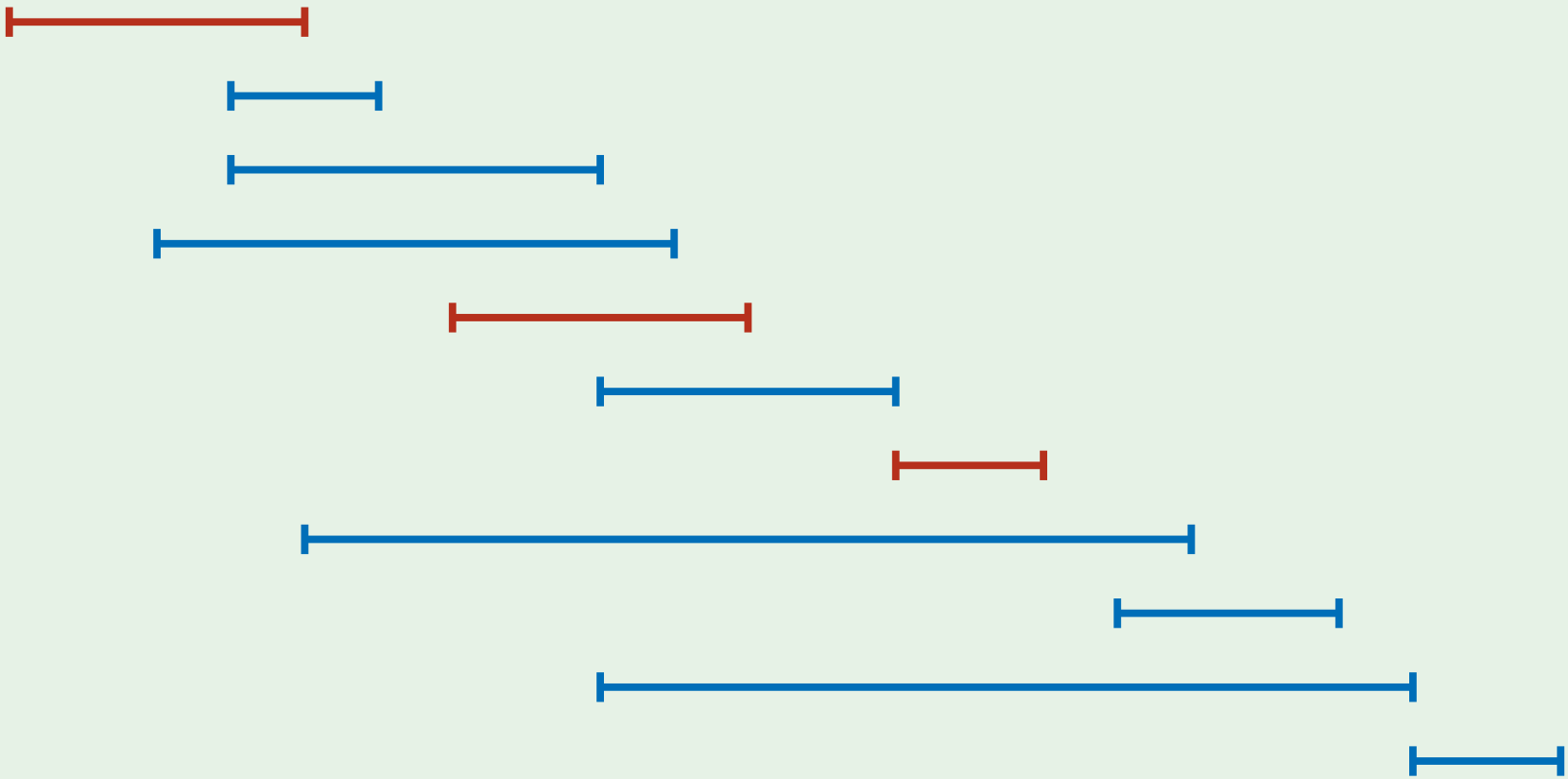
Пример



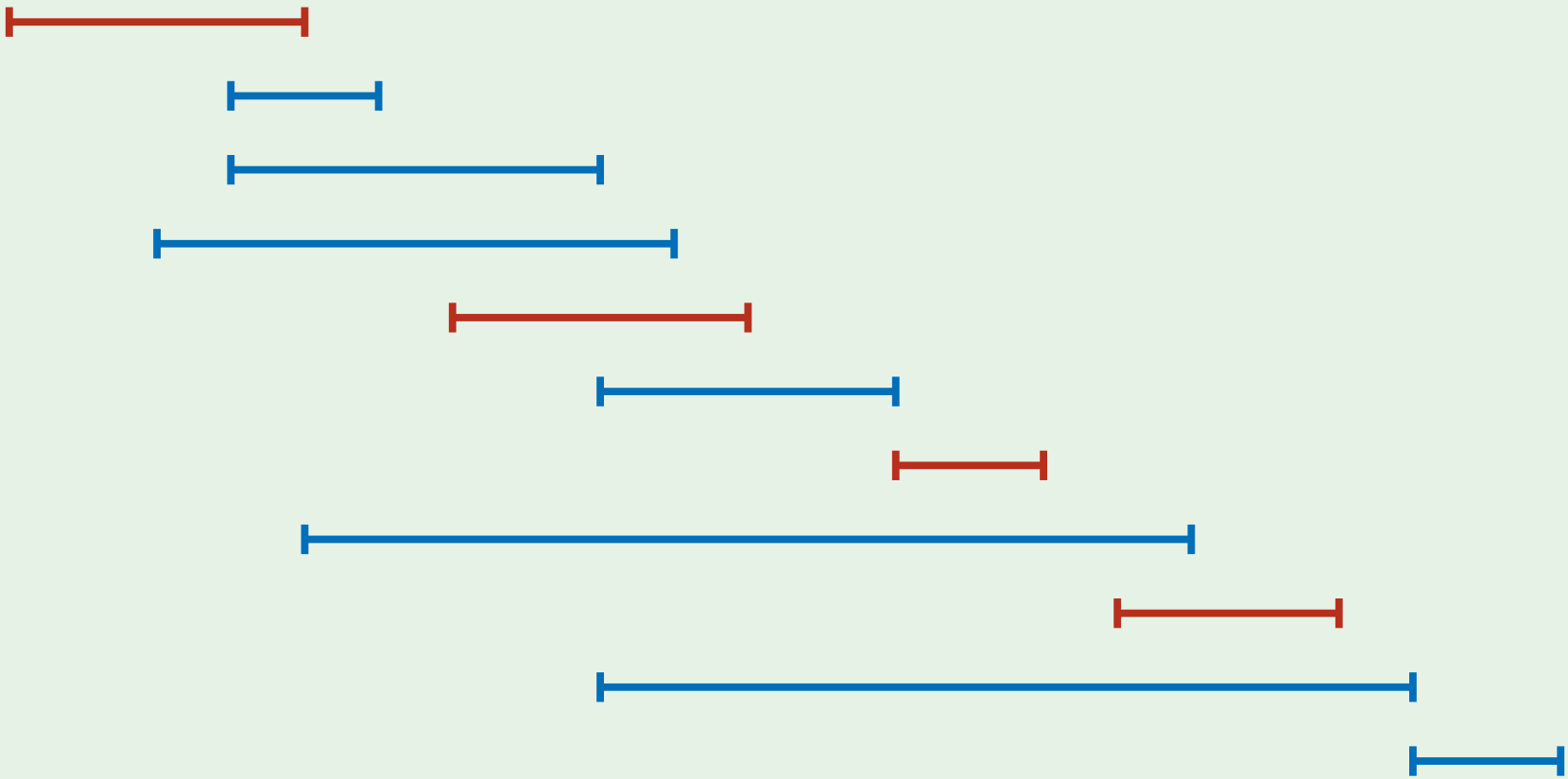
Пример



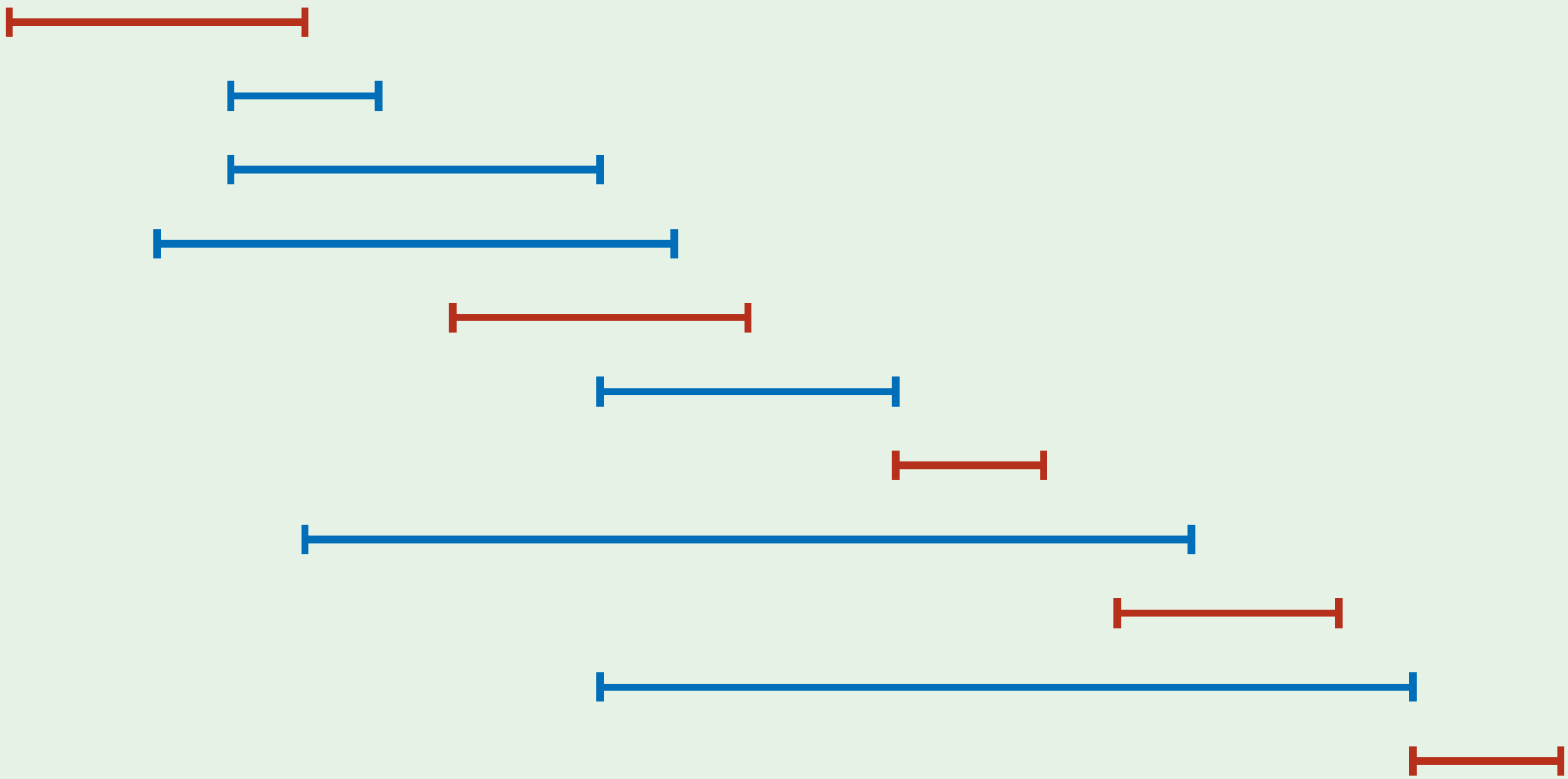
Пример



Пример



Пример

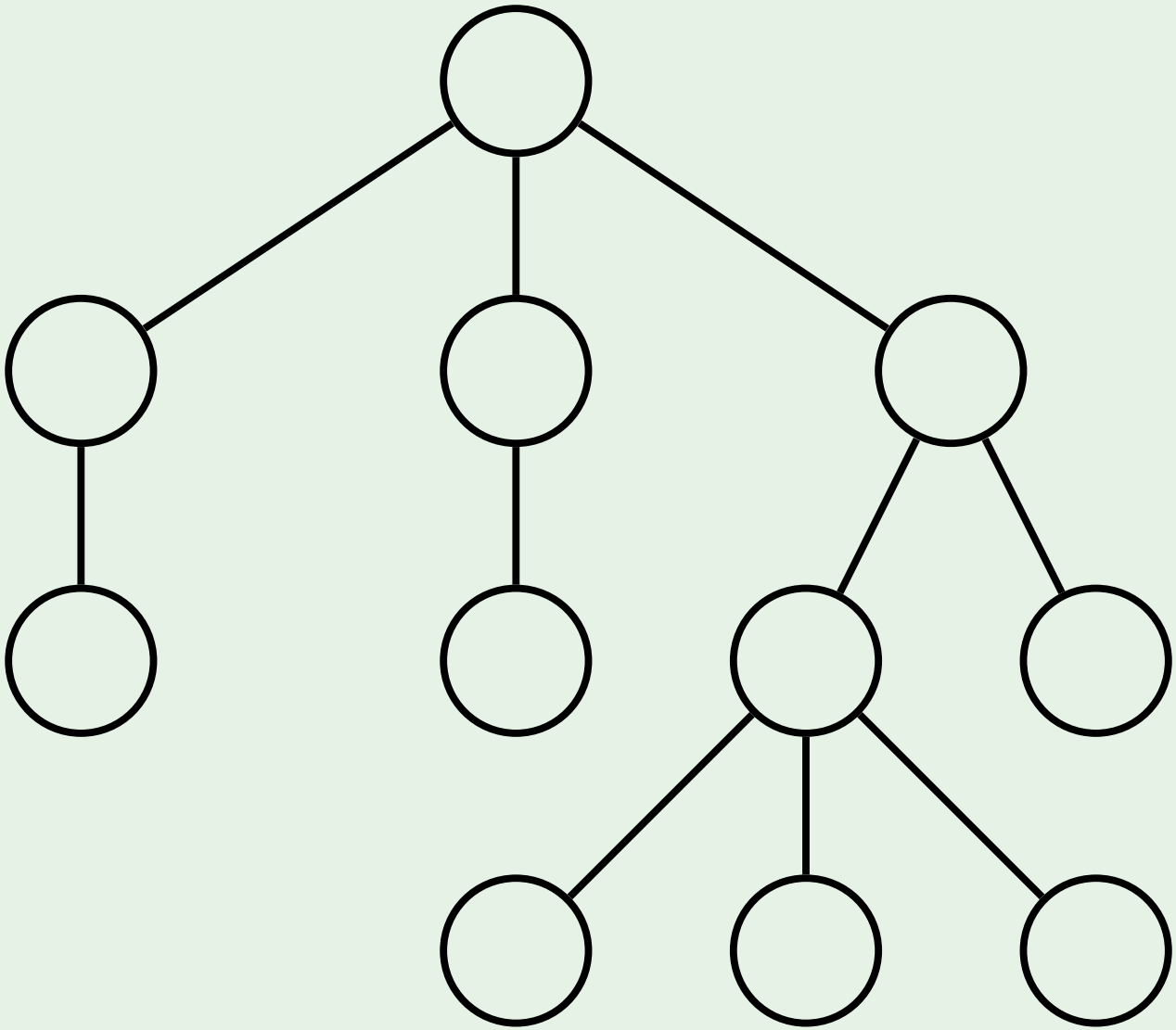


Планирование вечеринки в компании

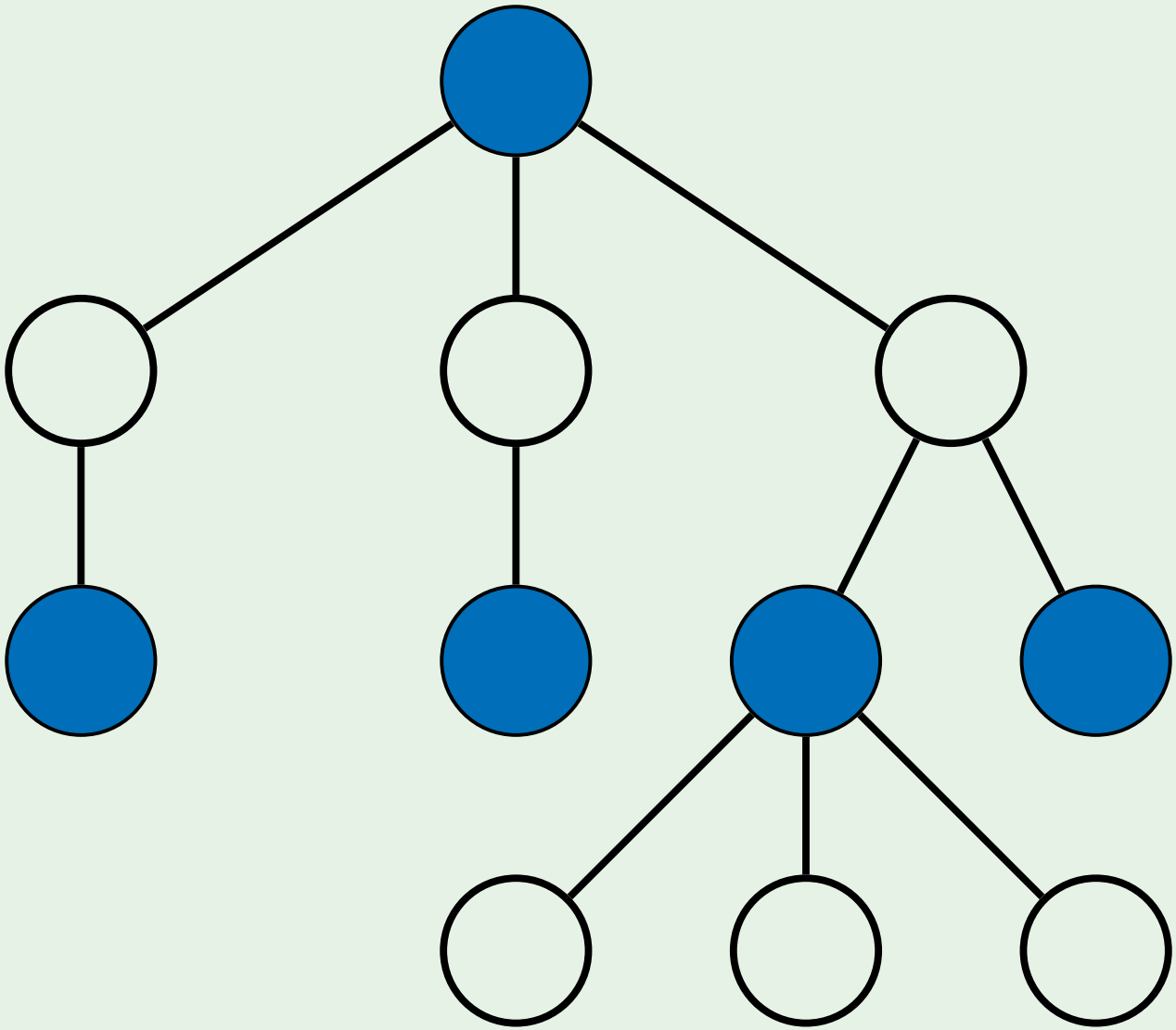
Вход: дерево.

Выход: независимое множество (множество не соединённых друг с другом вершин) максимального размера.

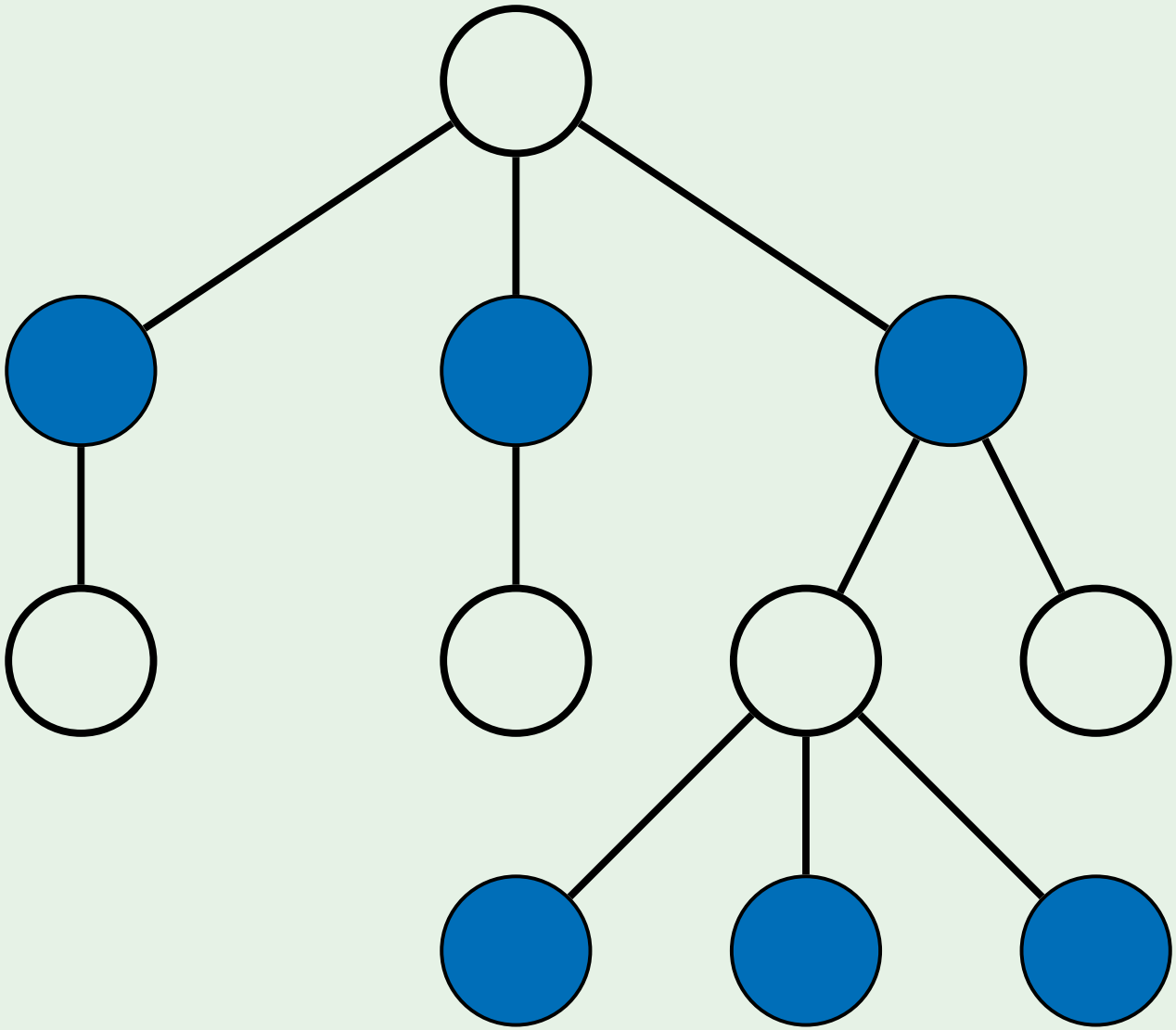
Пример



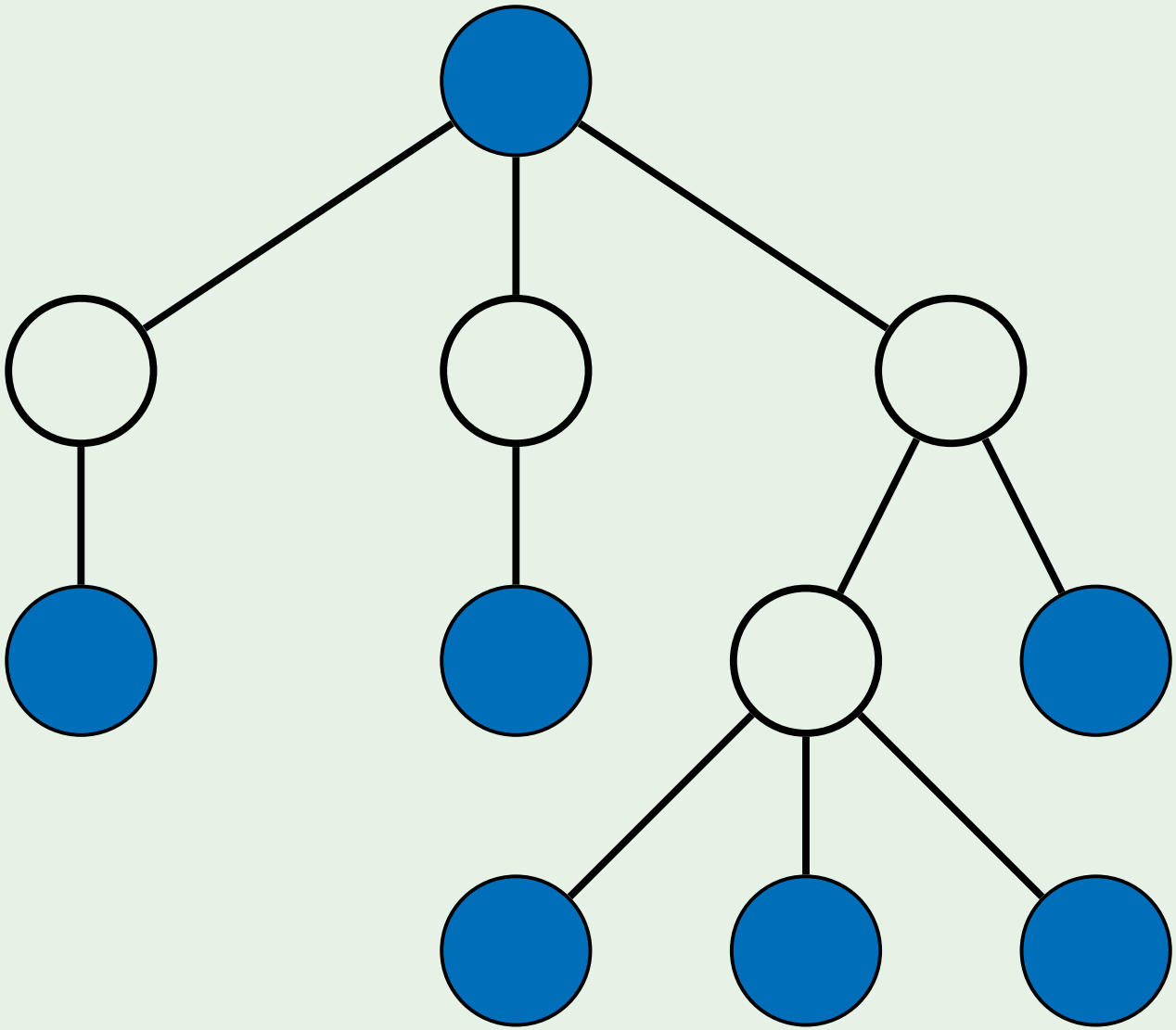
Пример



Пример

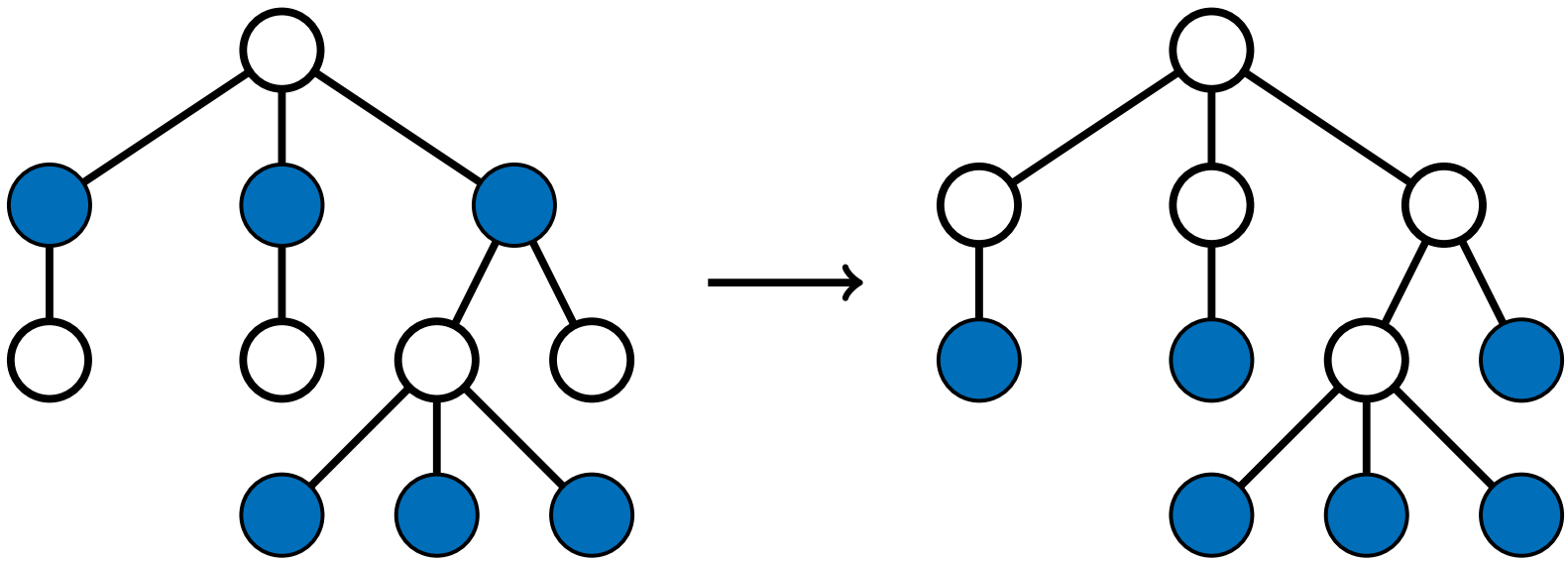


Пример



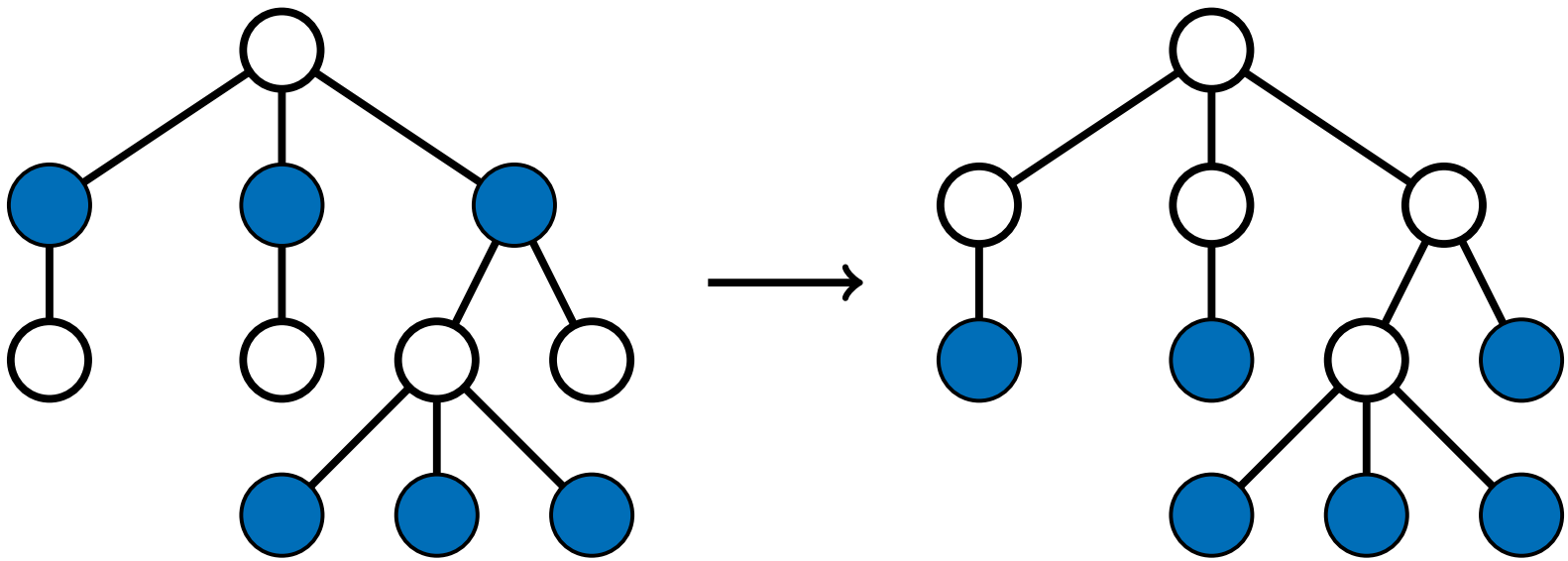
Надёжный шаг

Существует оптимальное решение, содержащее каждый лист дерева.



Надёжный шаг

Существует оптимальное решение, содержащее каждый лист дерева.



Можно взять в решение все листья.

Алгоритм

Функция MaxIndependentSet(T)

```
пока T не пусто:
  взять в решение все листья
  выкинуть их и их родителей из T
вернуть построенное решение
```

Алгоритм

Функция MaxIndependentSet(T)

```
пока T не пусто:
  взять в решение все листья
  выкинуть их и их родителей из T
вернуть построенное решение
```

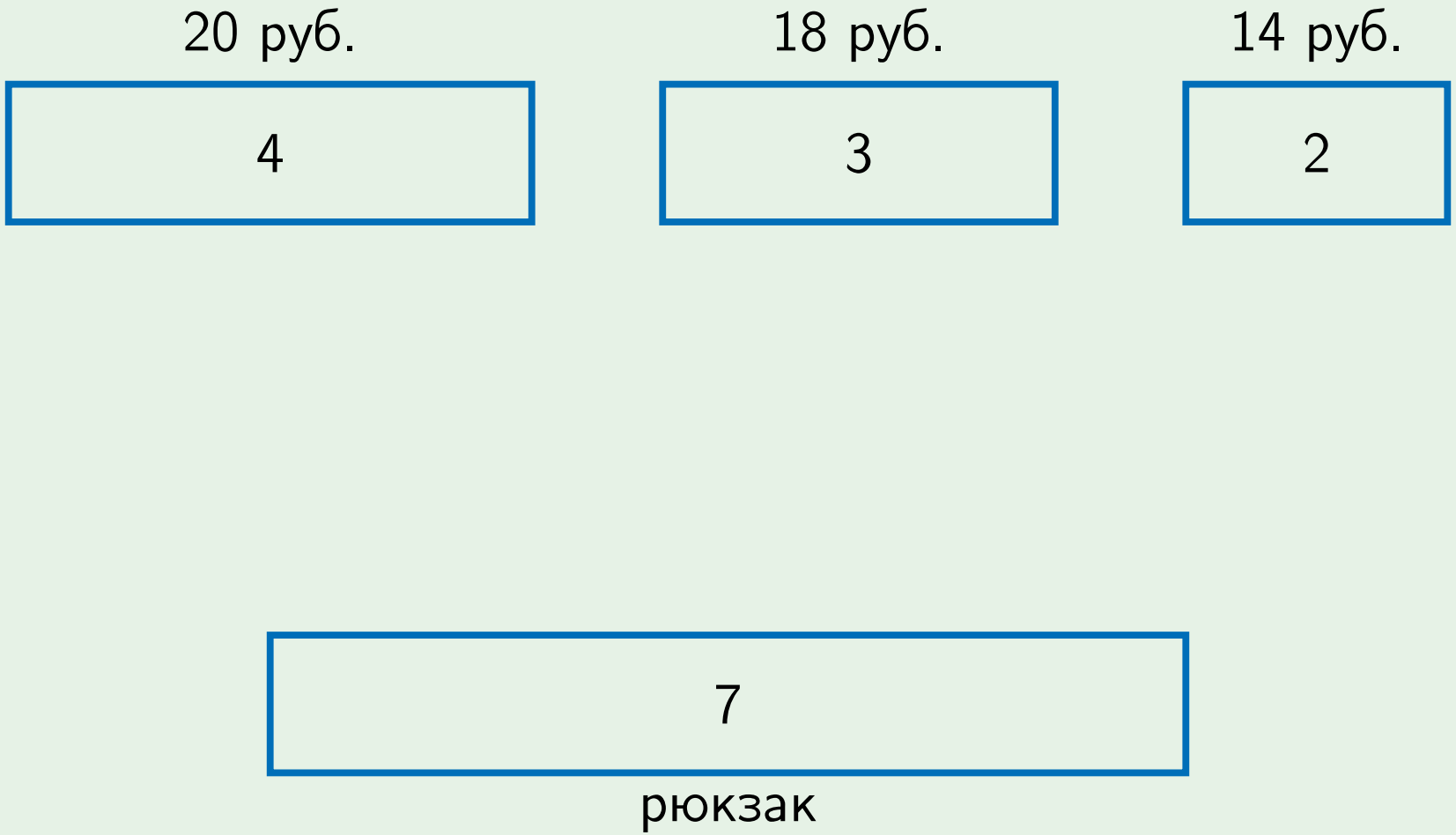
Время работы: $O(|T|)$.

Непрерывный рюкзак

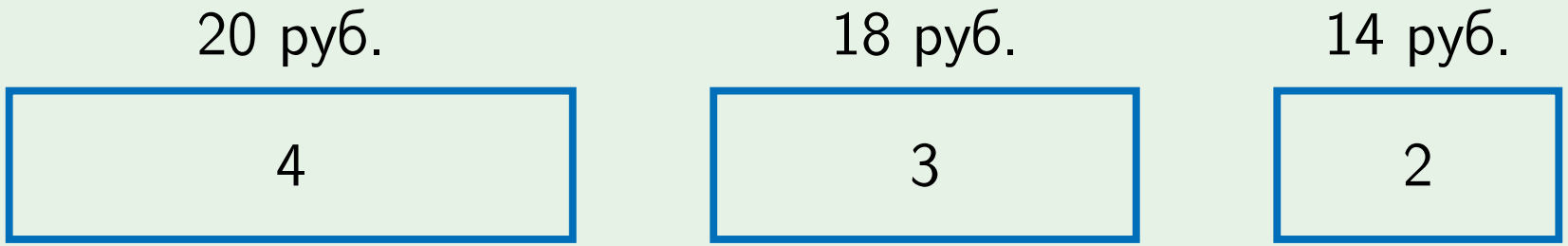
Вход: веса $w_1, \ldots, w_n$ и стоимости $c_1, \ldots, c_n$ данных n предметов; вместимость рюкзака W.

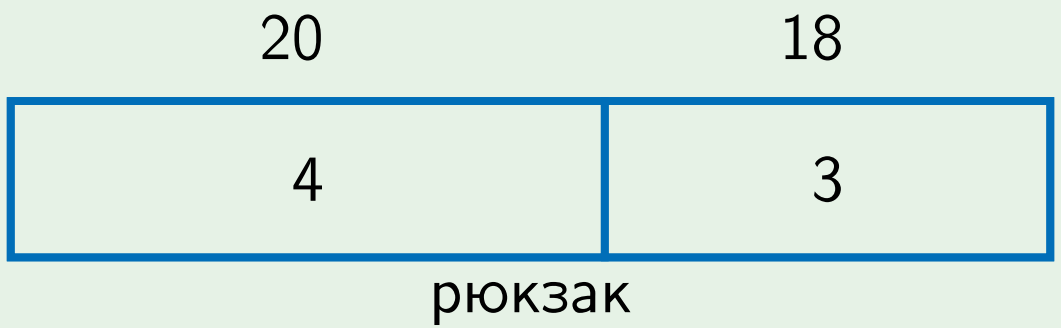
Выход: максимальная стоимость частей предметов суммарного веса не более W.

Пример



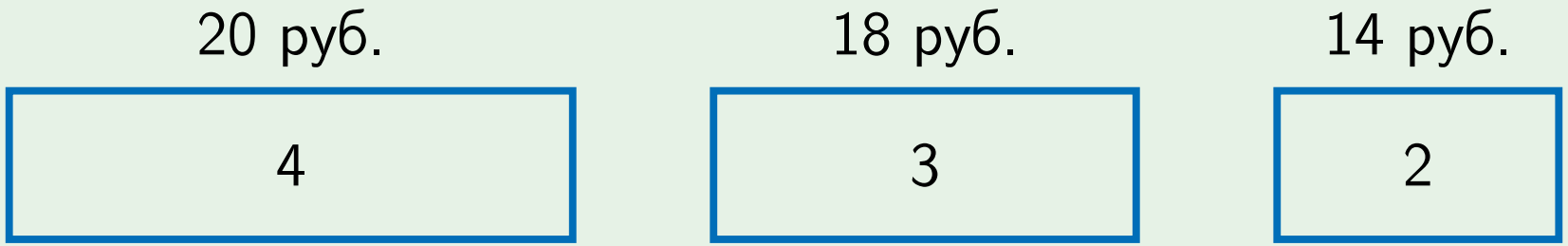
Пример

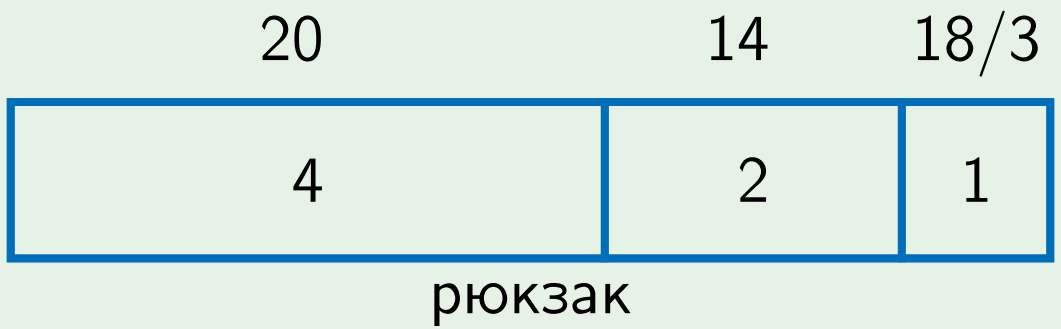




всего: 38 руб.

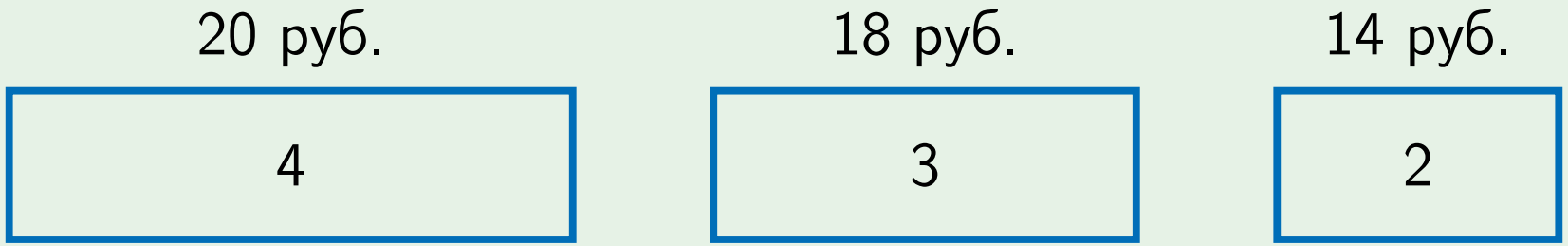
Пример

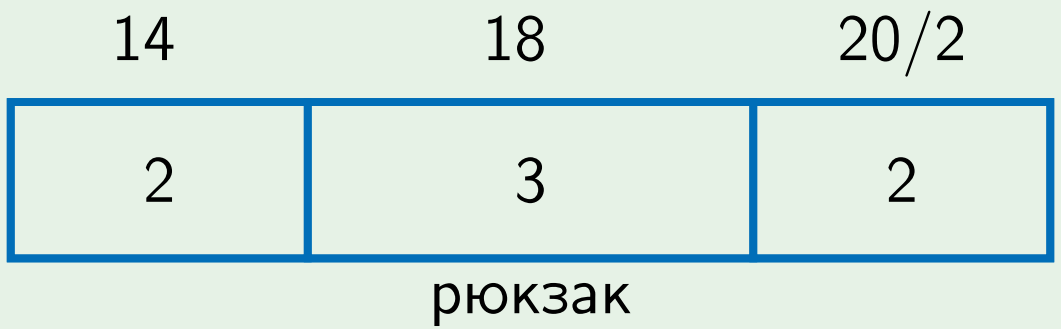




всего: 40 руб.

Пример

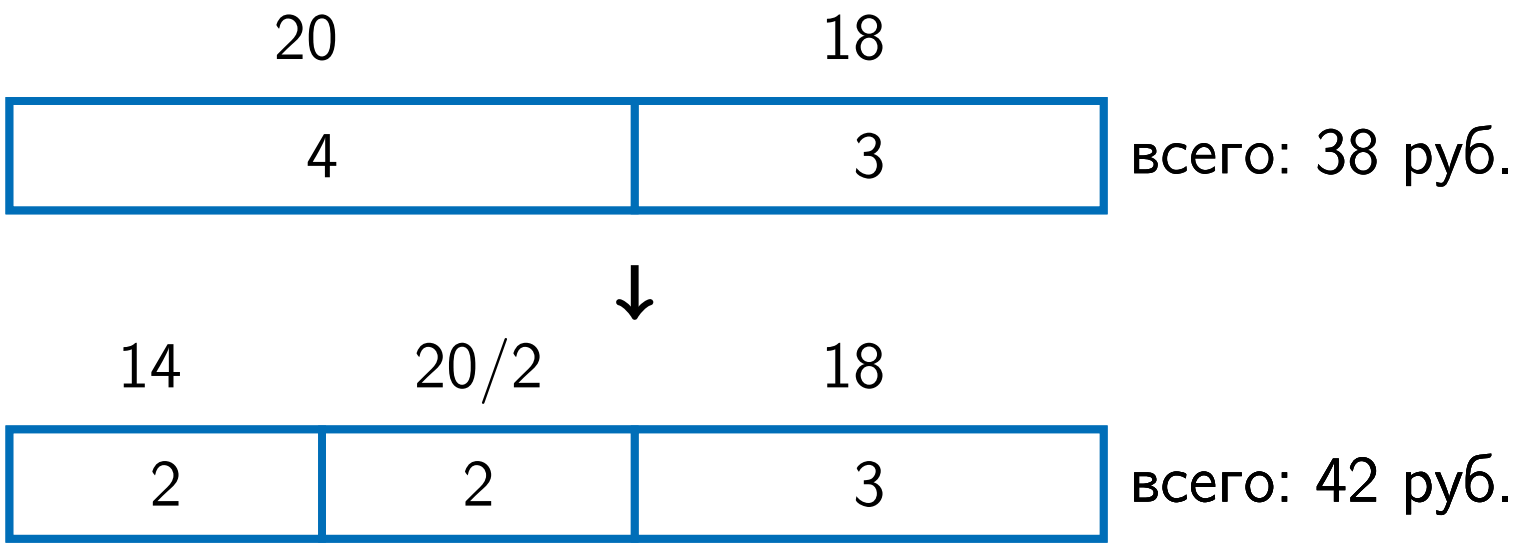




всего: 42 руб.

Надёжный шаг

Существует оптимальное решение, содержащее максимально возможную часть предмета, стоимость которого за килограмм максимальна.



Алгоритм

Функция KNAPSACK($w_1, c_1, \ldots, w_n, c_n$)

```
отсортировать предметы по убыванию c/w
для всех предметов в полученном порядке:
  взять по максимуму текущего предмета
вернуть построенное решение
```

Алгоритм

Функция KNAPSACK$(w_1, c_1, \ldots, w_n, c_n)$

```
отсортировать предметы по убыванию c/w
для всех предметов в полученном порядке:
  взять по максимуму текущего предмета
вернуть построенное решение
```

Время работы: $T(\text{SORT}) + O(n) = O(n \log n)$.

Основные идеи

Надёжный шаг. Существует оптимальное решение, согласованное с локальным жадным шагом.

Оптимальность подзадач. Задача, остающаяся после жадного шага, имеет тот же тип.