Мини(?)-задача #21 (2 балла)

Реализовать биномиальную кучу со всеми рассмотренными операциями.

Подготовить набор тестов, демонстрирующих корректность решения.

Алгоритмы и структуры данных

Сложность алгоритма Дейкстры, Биномиальная пирамида

Абстрактный тип данных: очередь с приоритетами

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(priority, value) - добавить в очередь задание с указанным приоритетом
2. peek_max() -> T - взять элемент с максимальным* приоритетом (без изменения очереди)
3. extract_max() -> T - взять элемент с максимальным* приоритетом (удалив его из очереди)

Абстрактный тип данных: очередь с приоритетами

Множество значений: S = {<priority: int, value: T>}

Операции:

1. insert(priority, value)
2. peek_max() -> T
3. extract_max() -> T

4. increase_priority(s, new_priority),

   где s ∈ S - элемент из очереди,
   new_priority ⩾ s.priority

Абстрактный тип данных: очередь с приоритетами

Множество значений: S = {<priority: int, value: T>}

Операции:

1. insert(priority, value)
2. peek_max() -> T
3. extract_max() -> T
4. increase_priority(s, new_priority),

   где s ∈ S - элемент из очереди,
   new_priority ⩾ s.priority

Именно сам элемент, поиск по значению не подразумевается

Очередь с приоритетами через пирамиду

Множество значений: пары <priority: int, value: T>

Будем хранить массив из пар <priority, value>
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. insert(priority, value)
2. peek_max() -> T
3. extract_max() -> T

Очередь с приоритетами через пирамиду

Множество значений: пары <priority: int, value: T>

Будем хранить массив из пар <priority, value>
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. insert(priority, value)
2. peek_max() -> T
3. extract_max() -> T

4. increase_priority(s, new_priority)

Очередь с приоритетами через пирамиду

Множество значений: пары <priority: int, value: T>

Будем хранить массив из пар <priority, value>
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. insert(priority, value)
2. peek_max() -> T
3. extract_max() -> T

4. increase_priority(s, new_priority)

В этом случае, в качестве s используется текущий индекс элемента в массиве

Очередь с приоритетами через пирамиду

Множество значений: пары <priority: int, value: T>

Будем хранить массив из пар <priority, value>
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. insert(priority, value)
2. peek_max() -> T
3. extract_max() -> T

4. increase_priority(s, new_priority)

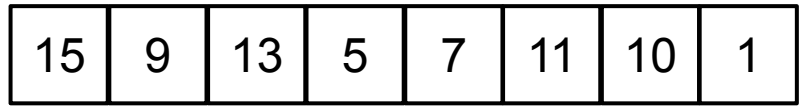
В этом случае, в качестве s используется текущий индекс элемента в массиве

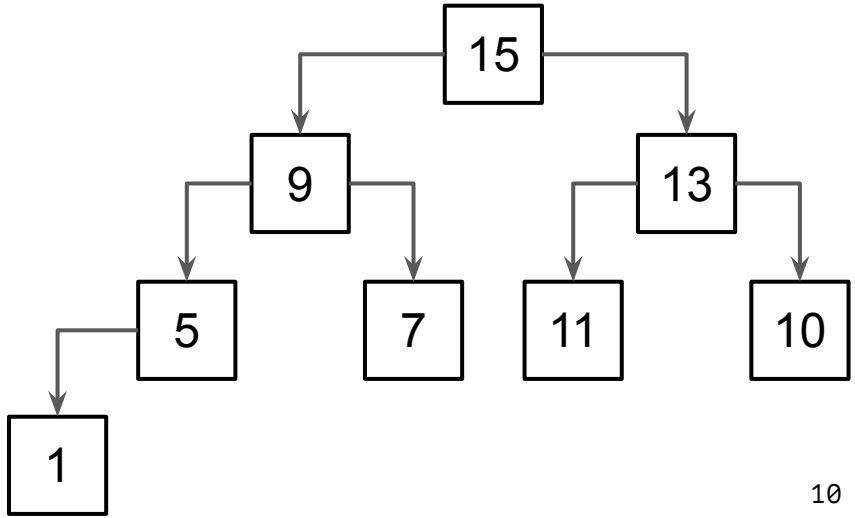
Как реализовать?

Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i+1] \; \forall i : 2i+1 < N,$$

$$A[i] \geq A[2i+2] \; \forall i : 2i+2 < N.$$

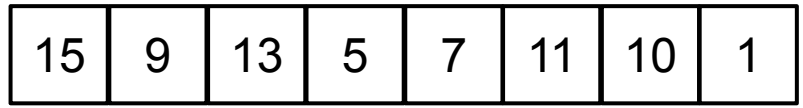




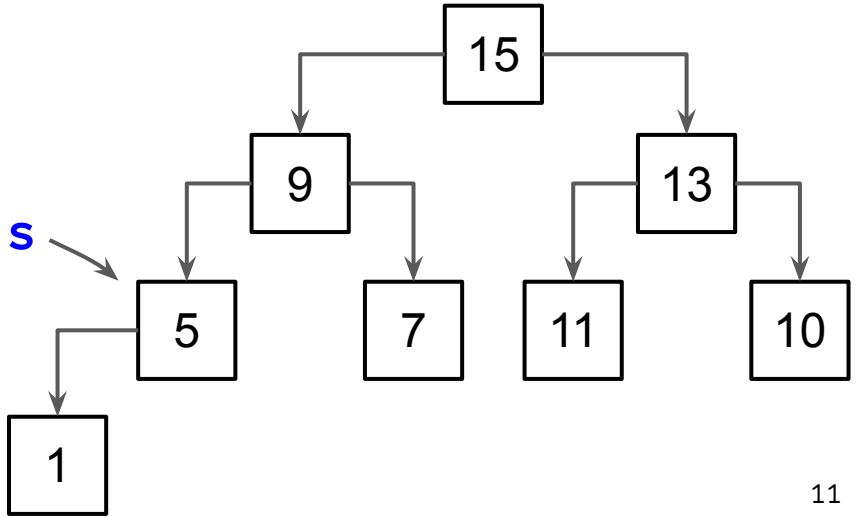
Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i+1] \; \forall i : 2i+1 < N,$$

$$A[i] \geq A[2i+2] \; \forall i : 2i+2 < N.$$



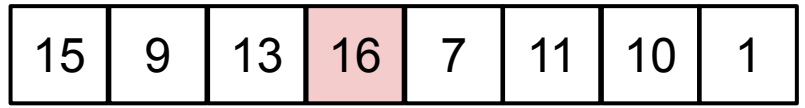
Запрос: `increase_priority(s, 16)`



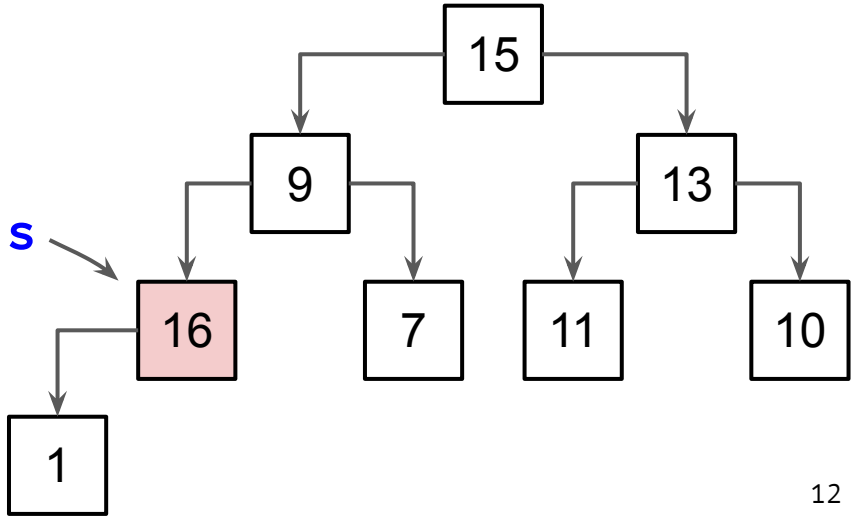
Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i+1] \; \forall i : 2i+1 < N,$$

$$A[i] \geq A[2i+2] \; \forall i : 2i+2 < N.$$



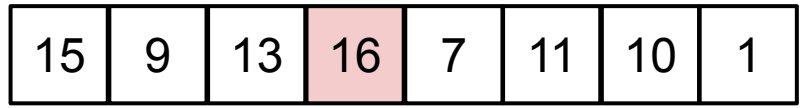
Запрос: `increase_priority(s, 16)`



Структура данных: невозрастающая пирамида

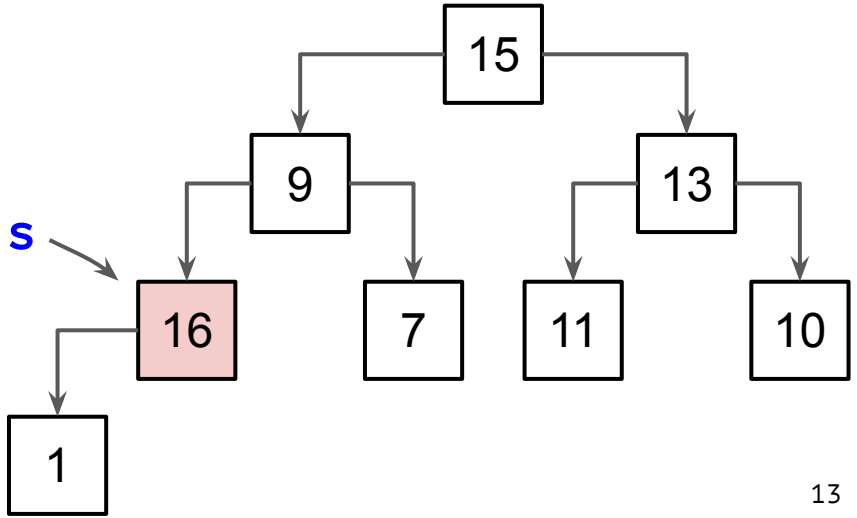
$$A[i] \geq A[2i+1] \; \forall i : 2i+1 < N,$$

$$A[i] \geq A[2i+2] \; \forall i : 2i+2 < N.$$



Запрос: `increase_priority(s, 16)`

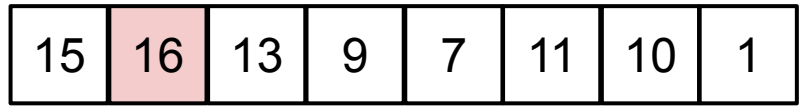
Решение: просеивание в верх!



Структура данных: невозрастающая пирамида

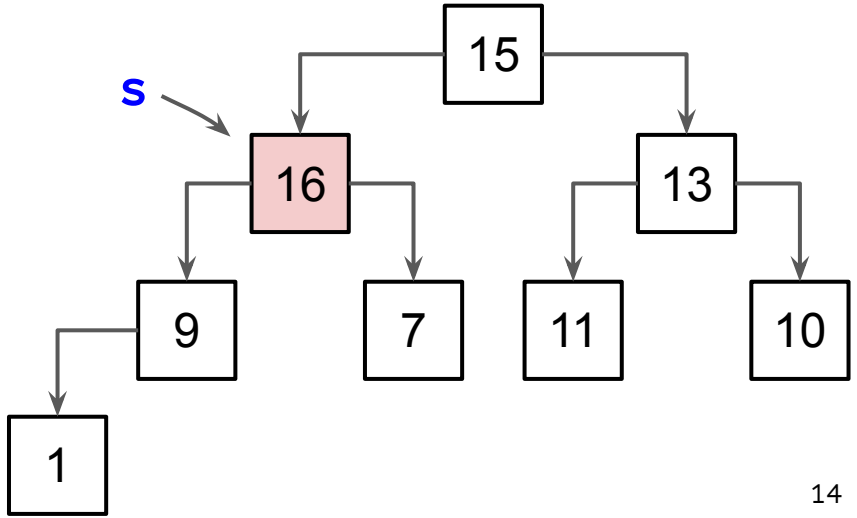
$$A[i] \geq A[2i+1] \; \forall i : 2i+1 < N,$$

$$A[i] \geq A[2i+2] \; \forall i : 2i+2 < N.$$



Запрос: `increase_priority(s, 16)`

Решение: просеивание в верх!



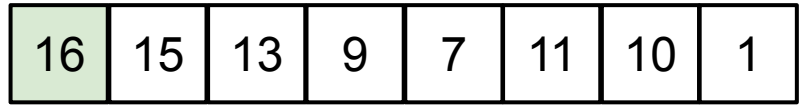
Структура данных: невозрастающая пирамида

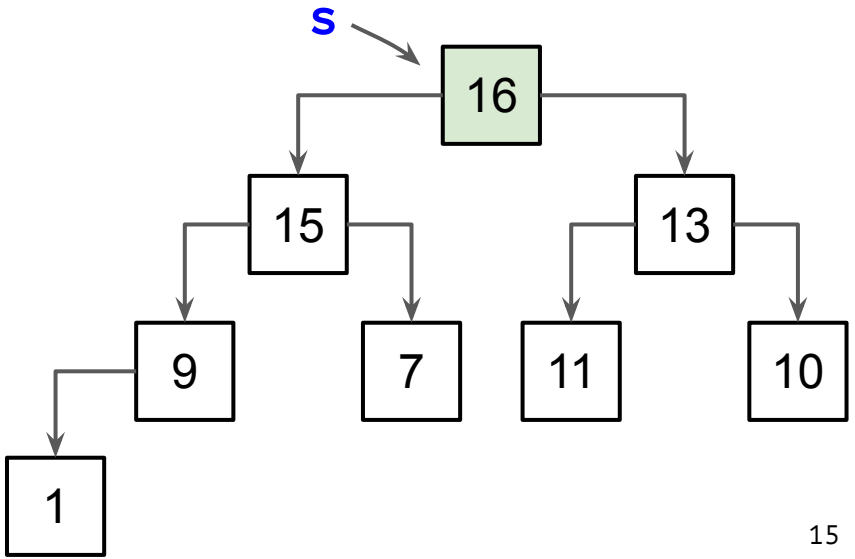
$$A[i] \geq A[2i+1] \; \forall i : 2i+1 < N,$$

$$A[i] \geq A[2i+2] \; \forall i : 2i+2 < N.$$

Запрос: increase_priority(s, 16)

Решение: просеивание в верх!

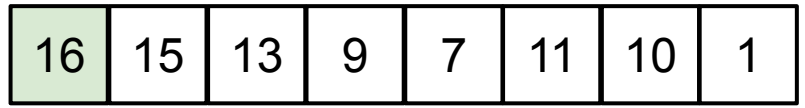




Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i+1] \; \forall i : 2i+1 < N,$$

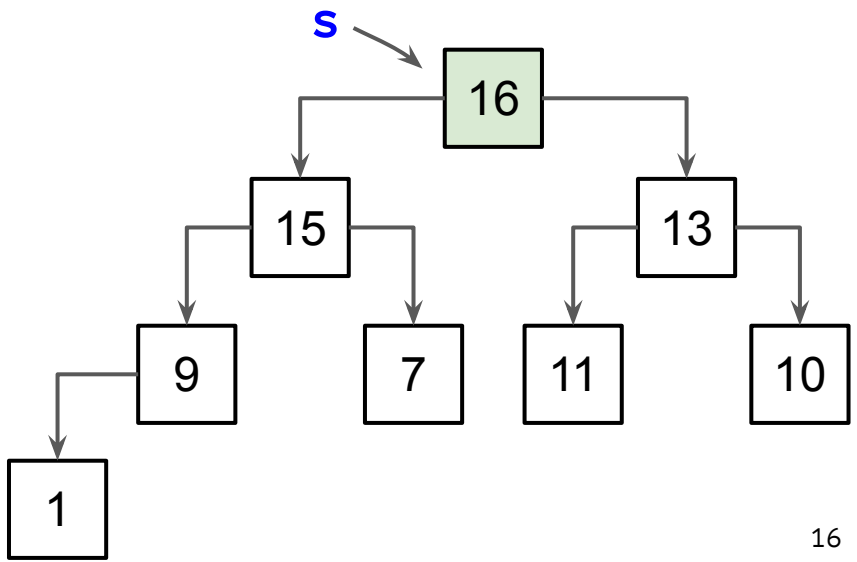
$$A[i] \geq A[2i+2] \; \forall i : 2i+2 < N.$$



Запрос: increase_priority(s, 16)

Решение: просеивание в верх!

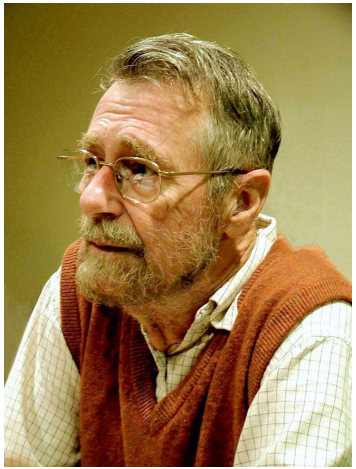
Сложность: O(logN)



Алгоритм Дейкстры поиска кратчайших путей

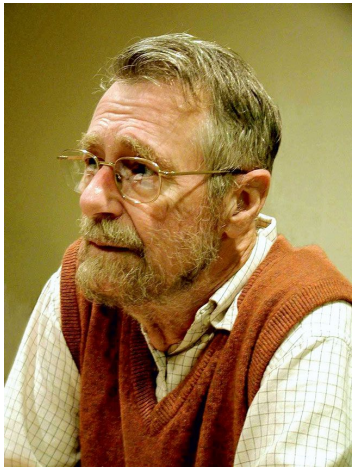
Алгоритм Дейкстры поиска кратчайших путей

Задача: пусть дан взвешенный граф $G = \langle V, E \rangle$ без рёбер отрицательного веса. Найти длины кратчайших путей от заданной вершины до всех остальных.



Алгоритм Дейкстры поиска кратчайших путей

Задача: пусть дан взвешенный граф $G = \langle V, E \rangle$ без рёбер отрицательного веса. Найти кратчайшие пути от заданной вершины до всех остальных.



Алгоритм Дейкстры поиска кратчайших путей

Задача: пусть дан взвешенный граф $G = \langle V, E \rangle$ без рёбер отрицательного веса. Найти кратчайшие пути от заданной вершины до всех остальных.

Идея алгоритма:

1. Поддерживаем множество "пометок" - текущее вычисленное расстояние до каждой вершины

Алгоритм Дейкстры поиска кратчайших путей

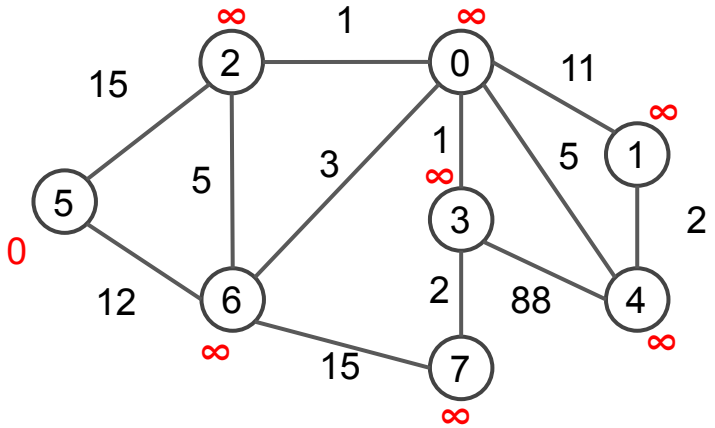
Задача: пусть дан взвешенный граф $G = \langle V, E \rangle$ без рёбер отрицательного веса. Найти кратчайшие пути от заданной вершины до всех остальных.

Идея алгоритма:

1. Поддерживаем множество "пометок" - текущее вычисленное расстояние до каждой вершины

2. На каждой итерации выбираем необработанную еще вершину с минимальной пометкой в качестве текущей, обновляем пометки ее соседей

Алгоритм Дейкстры поиска кратчайших путей



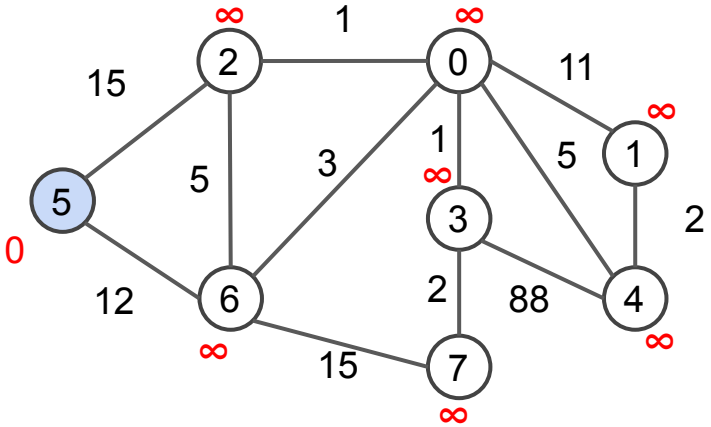
visited = | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

dists = | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ | ∞ |

Ищем расстояние от пятой вершины до всех остальных.

Изначально все остальные пометки - бесконечность

Алгоритм Дейкстры поиска кратчайших путей



visited = | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

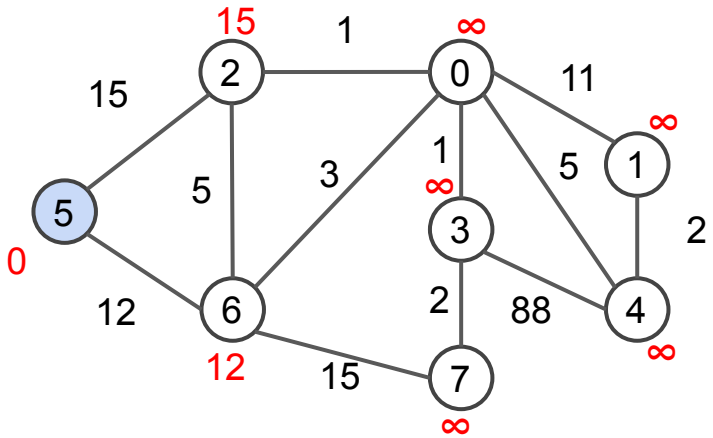
dists = | ∞ | ∞ | ∞ | ∞ | ∞ | 0 | ∞ | ∞ |

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах

Ищем расстояние от пятой вершины до всех остальных.

Изначально все остальные пометки - бесконечность

Алгоритм Дейкстры поиска кратчайших путей



visited = | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

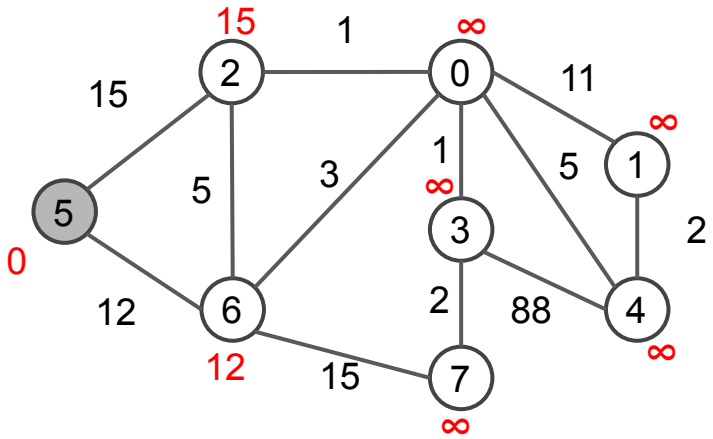
dists = | ∞ | ∞ | 15 | ∞ | ∞ | 0 | 12 | ∞ |

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах

Ищем расстояние от пятой вершины до всех остальных.

Изначально все остальные пометки - бесконечность

Алгоритм Дейкстры поиска кратчайших путей



Ищем расстояние от пятой вершины до всех остальных.

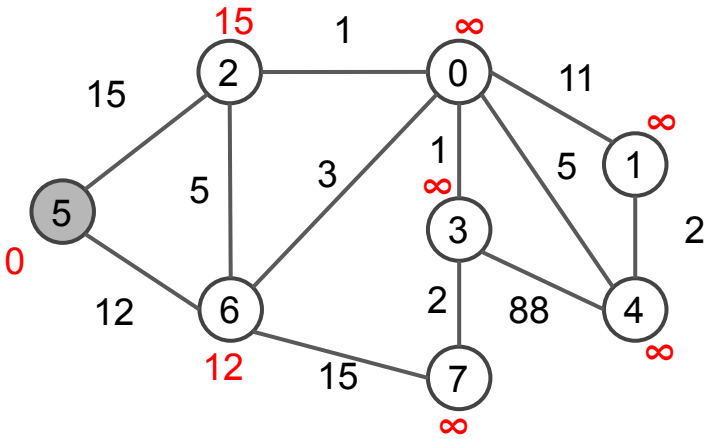
Изначально все остальные пометки - бесконечность

visited = | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

dists = | ∞ | ∞ | 15 | ∞ | ∞ | 0 | 12 | ∞ |

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах
2. Пометить текущую, как обработанную

Алгоритм Дейкстры поиска кратчайших путей



Ищем расстояние от пятой вершины до всех остальных.

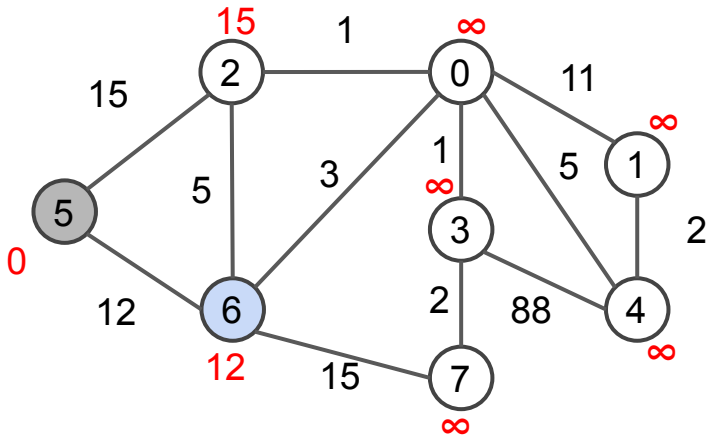
Изначально все остальные пометки - бесконечность

visited = | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

dists = | ∞ | ∞ | 15 | ∞ | ∞ | 0 | 12 | ∞ |

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах
2. Пометить текущую, как обработанную
3. В качестве следующей вершины берем ту, в которой еще не были, и у которой минимальный dist

Алгоритм Дейкстры поиска кратчайших путей



Ищем расстояние от пятой вершины до всех остальных.

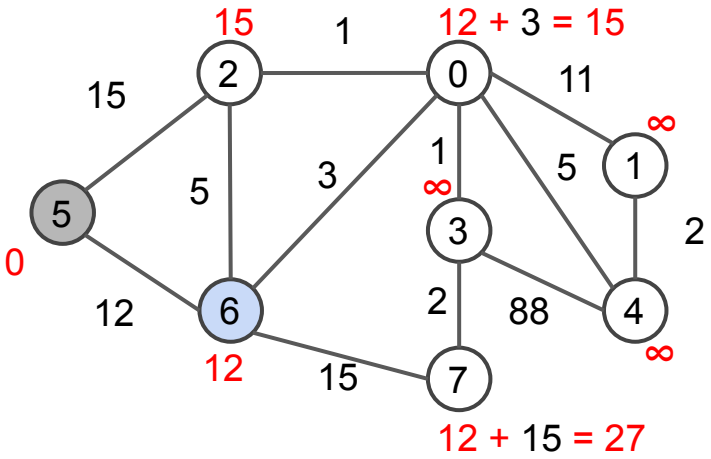
Изначально все остальные пометки - бесконечность

visited = | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

dists = | ∞ | ∞ | 15 | ∞ | ∞ | 0 | 12 | ∞ |

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах
2. Пометить текущую, как обработанную
3. В качестве следующей вершины берем ту, в которой еще не были, и у которой минимальный dist

Алгоритм Дейкстры поиска кратчайших путей



Ищем расстояние от пятой вершины до всех остальных.

Изначально все остальные пометки - бесконечность

visited =

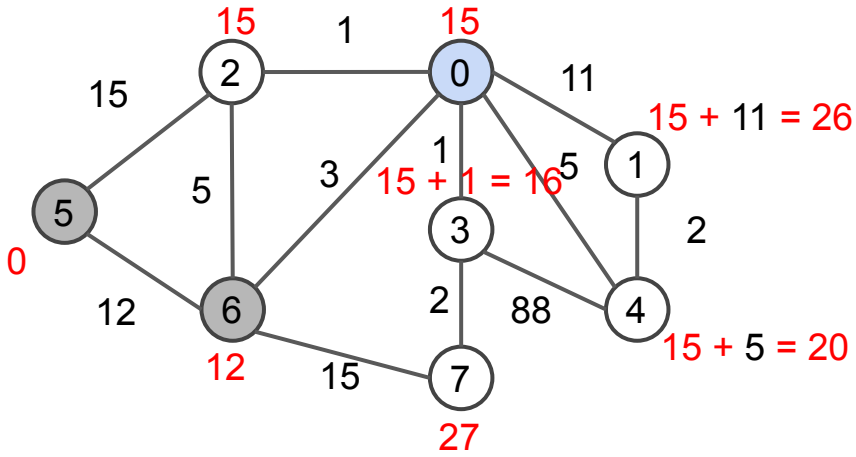
0	0	0	0	0	1	0	0

dists =

15	∞	15	∞	∞	0	12	27

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах
2. Пометить текущую, как обработанную
3. В качестве следующей вершины берем ту, в которой еще не были, и у которой минимальный dist

Алгоритм Дейкстры поиска кратчайших путей



Ищем расстояние от пятой вершины до всех остальных.

Изначально все остальные пометки - бесконечность

visited =

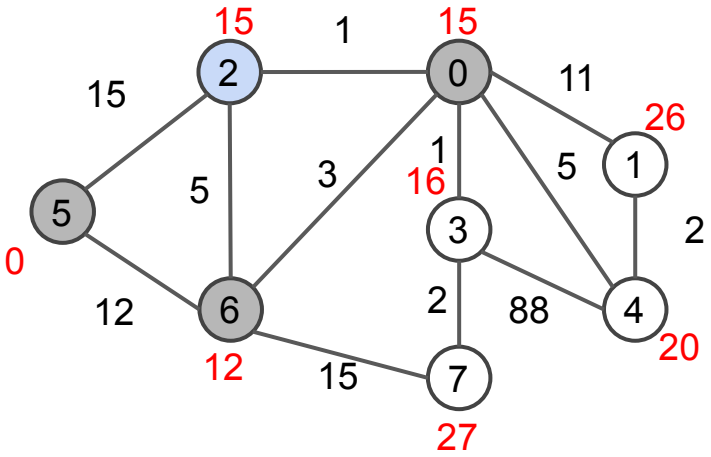
0	0	0	0	0	1	1	0

dists =

15	26	15	16	20	0	12	27

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах
2. Пометить текущую, как обработанную
3. В качестве следующей вершины берем ту, в которой еще не были, и у которой минимальный dist

Алгоритм Дейкстры поиска кратчайших путей



Ищем расстояние от пятой вершины до всех остальных.

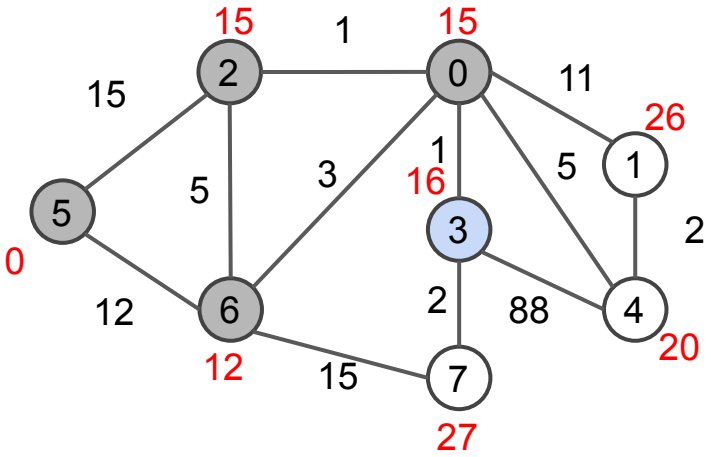
Изначально все остальные пометки - бесконечность

visited = | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

dists = | 15 | 26 | 15 | 16 | 20 | 0 | 12 | 27 |

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах
2. Пометить текущую, как обработанную
3. В качестве следующей вершины берем ту, в которой еще не были, и у которой минимальный dist

Алгоритм Дейкстры поиска кратчайших путей



Ищем расстояние от пятой вершины до всех остальных.

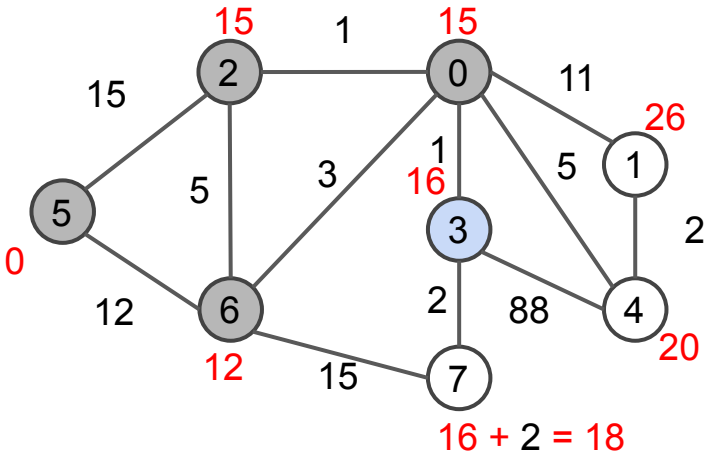
Изначально все остальные пометки - бесконечность

visited = | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |

dists = | 15 | 26 | 15 | 16 | 20 | 0 | 12 | 27 |

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах
2. Пометить текущую, как обработанную
3. В качестве следующей вершины берем ту, в которой еще не были, и у которой минимальный dist

Алгоритм Дейкстры поиска кратчайших путей



Ищем расстояние от пятой вершины до всех остальных.

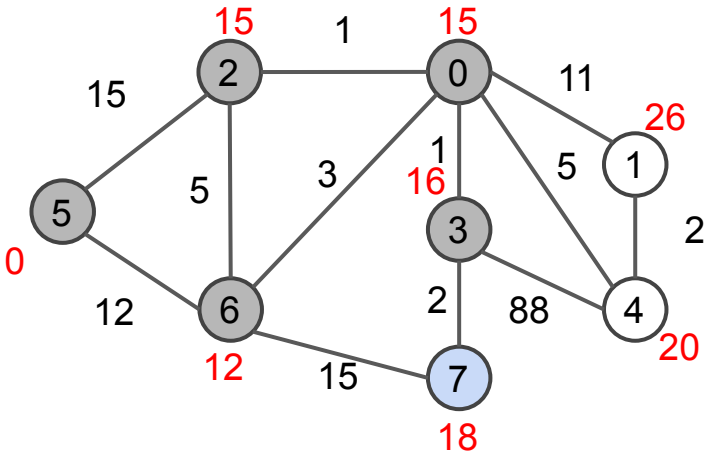
Изначально все остальные пометки - бесконечность

visited = | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |

dists = | 15 | 26 | 15 | 16 | 20 | 0 | 12 | 18 |

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах
2. Пометить текущую, как обработанную
3. В качестве следующей вершины берем ту, в которой еще не были, и у которой минимальный dist

Алгоритм Дейкстры поиска кратчайших путей



Ищем расстояние от пятой вершины до всех остальных.

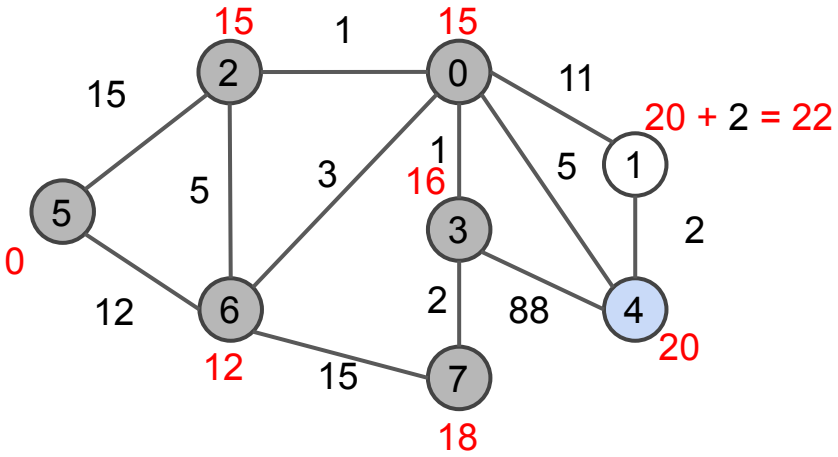
Изначально все остальные пометки - бесконечность

visited = | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |

dists = | 15 | 26 | 15 | 16 | 20 | 0 | 12 | 18 |

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах
2. Пометить текущую, как обработанную
3. В качестве следующей вершины берем ту, в которой еще не были, и у которой минимальный dist

Алгоритм Дейкстры поиска кратчайших путей



Ищем расстояние от пятой вершины до всех остальных.

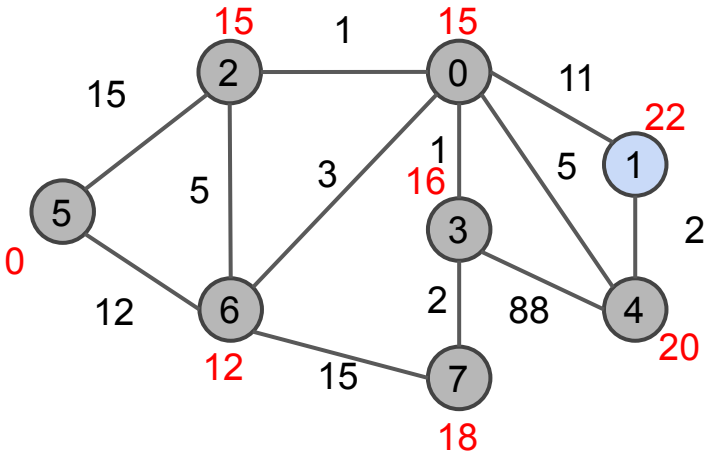
Изначально все остальные пометки - бесконечность

visited = | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |

dists = | 15 | 22 | 15 | 16 | 20 | 0 | 12 | 18 |

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах
2. Пометить текущую, как обработанную
3. В качестве следующей вершины берем ту, в которой еще не были, и у которой минимальный dist

Алгоритм Дейкстры поиска кратчайших путей



Ищем расстояние от пятой вершины до всех остальных.

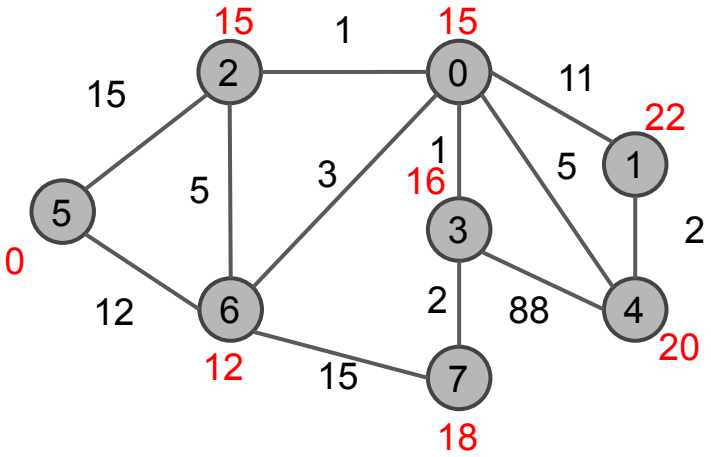
Изначально все остальные пометки - бесконечность

visited = | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

dists = | 15 | 22 | 15 | 16 | 20 | 0 | 12 | 18 |

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах
2. Пометить текущую, как обработанную
3. В качестве следующей вершины берем ту, в которой еще не были, и у которой минимальный dist

Алгоритм Дейкстры поиска кратчайших путей



Ищем расстояние от пятой вершины до всех остальных.

Изначально все остальные пометки - бесконечность

visited = | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

dists = | 15 | 22 | 15 | 16 | 20 | 0 | 12 | 18 |

1. Улучшаем текущее расстояние до соседних с текущей (еще не обработанных) вершинах
2. Пометить текущую, как обработанную
3. В качестве следующей вершины берем ту, в которой еще не были, и у которой минимальный dist

Алгоритм Дейкстры поиска кратчайших путей

Идея алгоритма:

1. Поддерживаем множество "пометок" - текущее вычисленное расстояние до каждой вершины

2. На каждой итерации выбираем необработанную еще вершину с минимальной пометкой в качестве текущей, обновляем пометки ее соседей

Сложность?

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум из пометок

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум из пометок

$$O(|V| * T + |E|)$$

T — сложность поиска минимальной из пометок

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум из пометок

$$O(|V| * T + |E|) = O(|V|^2 + |E|)$$

T — сложность поиска минимальной из пометок

Для случая линейного поиска - T = O(|V|)

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум из пометок

$$O(|V| * T + |E|) = O(|V|^2 + |E|) = O(|V|^2)$$
$$|E| \leq |V| * (|V| - 1)$$

T — сложность поиска минимальной из пометок

Для случая линейного поиска - T = O(|V|)

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O(|V| * T + |E|) = O(|V|^2 + |E|) = O(|V|^2)$$

$$|E| \leq |V| * (|V| - 1)$$

T — сложность поиска минимальной из пометок

Для случая линейного поиска - T = O(|V|)

Можем ли мы лучше?

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

Можем ли мы лучше?

$$O(|V| * T + |E|) = O(|V|^2 + |E|) = O(|V|^2)$$

$$|E| \leq |V| * (|V| - 1)$$

T — сложность поиска минимальной из пометок

Для случая линейного поиска - T = O(|V|)

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O(|V| * T + |E|) = O(|V|^2 + |E|) = O(|V|^2)$$

$$|E| \leq |V| * (|V| - 1)$$

T — сложность поиска минимальной из пометок

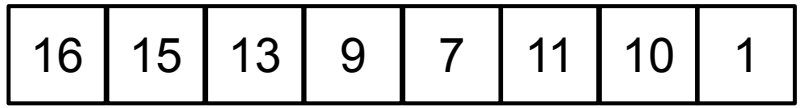
Для случая линейного поиска - T = O(|V|)

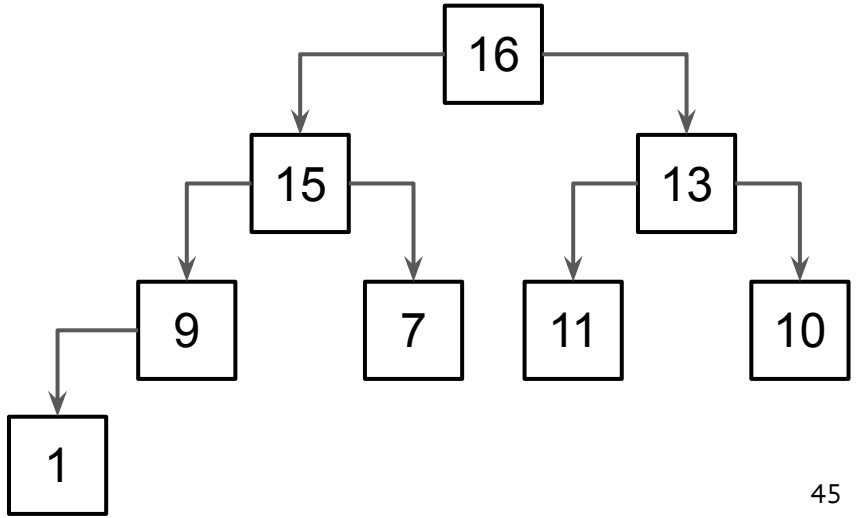
Можем ли мы лучше?

Структура данных: невозрастающая пирамида

$$A[i] \geq A[2i+1] \; \forall i : 2i+1 < N,$$

$$A[i] \geq A[2i+2] \; \forall i : 2i+2 < N.$$



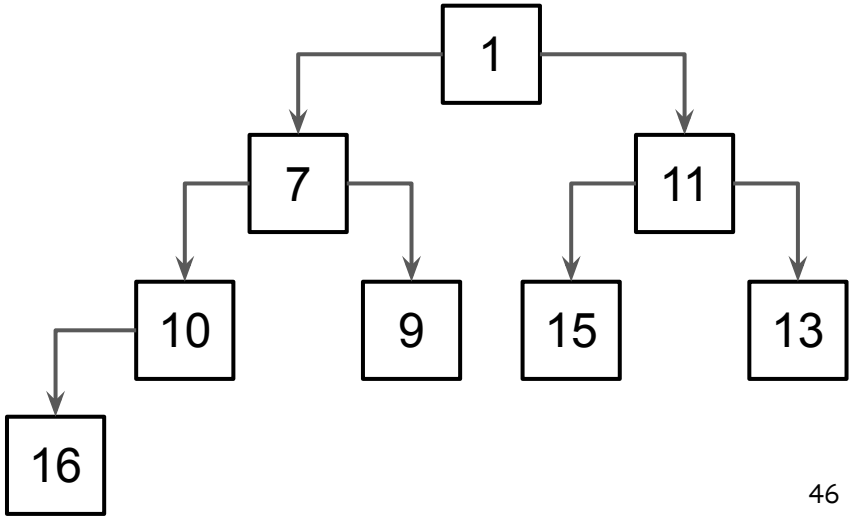


Структура данных: неубывающая пирамида

$$A[i] \leq A[2i+1] \; \forall i : 2i+1 < N,$$

$$A[i] \leq A[2i+2] \; \forall i : 2i+2 < N.$$

1	7	11	10	9	15	13	16



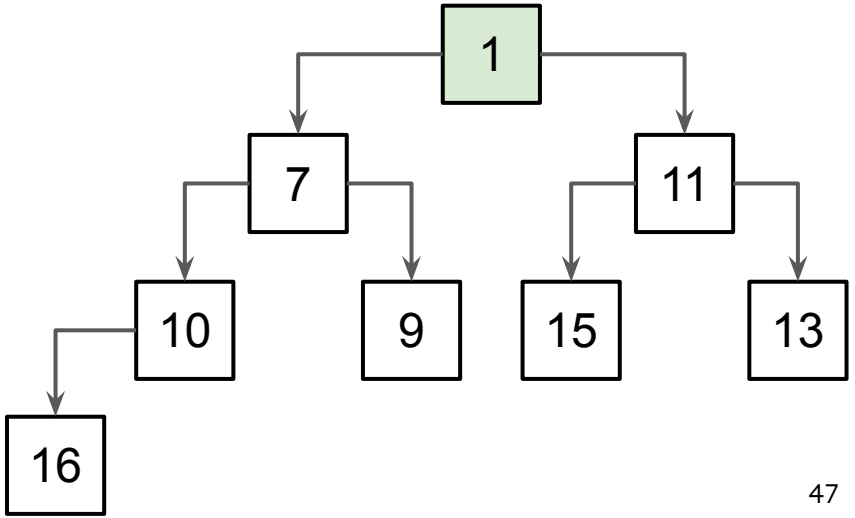
Структура данных: неубывающая пирамида

$$A[i] \leq A[2i+1] \; \forall i : 2i+1 < N,$$

$$A[i] \leq A[2i+2] \; \forall i : 2i+2 < N.$$

1	7	11	10	9	15	13	16

Теперь у такого массива
минимальный элемент стоит первым



Очередь с приоритетами через пирамиду

Множество значений: пары <priority: int, value: T>

Будем хранить массив из пар <priority, value>
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. insert(priority, value)
2. peek_max() -> T
3. extract_max() -> T
4. increase_priority(s, new_priority),
   где s ∈ S - элемент из очереди,
   new_priority ⩾ s.priority

Неубывающие очереди с приоритетами

Множество значений: пары <priority: int, value: T>

Будем хранить массив из пар <priority, value>
(но относимся к этому массиву, как к пирамиде из ключей)

Операции:

1. insert(priority, value)
2. peek_min() -> T
3. extract_min() -> T
4. decrease_priority(s, new_priority),
   где s ∈ S - элемент из очереди,
   new_priority ⩽ s.priority

Реализация абсолютно зеркальна предыдущей

Алгоритм Дейкстры поиска кратчайших путей

Идея алгоритма:

1. Поддерживаем множество "пометок" - текущее вычисленное расстояние до каждой вершины

2. На каждой итерации выбираем необработанную еще вершину с минимальной пометкой в качестве текущей, обновляем пометки ее соседей

Алгоритм Дейкстры поиска кратчайших путей

Идея алгоритма:

1. Поддерживаем множество "пометок" - текущее вычисленное расстояние до каждой вершины

2. На каждой итерации выбираем необработанную еще вершину с минимальной пометкой в качестве текущей, обновляем пометки ее соседей

Реализация: будем хранить пометки в неубывающей пирамиде

Алгоритм Дейкстры поиска кратчайших путей

Идея алгоритма:

1. Поддерживаем множество "пометок" - текущее вычисленное расстояние до каждой вершины

2. На каждой итерации выбираем необработанную еще вершину с минимальной пометкой в качестве текущей, обновляем пометки ее соседей

Реализация: будем хранить пометки в неубывающей пирамиде
При этом каждая пометка связана с вершиной и наоборот.

Алгоритм Дейкстры поиска кратчайших путей

Идея алгоритма:

1. Поддерживаем множество "пометок" - текущее вычисленное расстояние до каждой вершины

2. На каждой итерации выбираем необработанную еще вершину с минимальной пометкой в качестве текущей, обновляем пометки ее соседей

Реализация: будем хранить пометки в неубывающей пирамиде
При этом каждая пометка связана с вершиной и наоборот.

Можно сделать указателями, можно параллельными массивами.

Алгоритм Дейкстры поиска кратчайших путей

Идея алгоритма:

1. Поддерживаем множество "пометок" - текущее вычисленное расстояние до каждой вершины

2. На каждой итерации выбираем необработанную еще вершину с минимальной пометкой в качестве текущей, обновляем пометки ее соседей

Реализация: будем хранить пометки в неубывающей пирамиде

Сложность?

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O(|V| * T + |E|) = O(|V|^2 + |E|) = O(|V|^2)$$

Для случая линейного поиска - T = O(|V|)

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O(|V| * T + |E|) = O(|V|^2 + |E|) = O(|V|^2)$$

Для случая пирамиды - T = O(log(|V|))

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O(|V| * T + |E|) = O(|V|^2 + |E|) = O(|V|^2)$$

Для случая пирамиды - T = O(log(|V|))

Но нужно еще и пометки обновить, а это как раз decrease_priority(...) - каждая тоже за O(log(|V|))

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O(|V| * log(|V|) + |E| * log(|V|))$$

Для случая пирамиды - T = O(log(|V|))

Но нужно еще и пометки обновить, а это как раз decrease_priority(...) - каждая тоже за O(log(|V|))

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
   Перебор не может быть за O(V), иначе получите квадрат
3. На каждом шаге алгоритма ищем минимум

$$O(|V| * log(|V|) + |E| * log(|V|))$$

Для случая пирамиды - T = O(log(|V|))

Но нужно еще и пометки обновить, а это как раз decrease_priority(...) - каждая тоже за O(log(|V|))

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
   Перебор не может быть за O(V), иначе получите квадрат
3. На каждом шаге алгоритма ищем минимум

Попрощайтесь с матрицей смежности 😩

$$O(|V| * log(|V|) + |E| * log(|V|))$$

Для случая пирамиды - T = O(log(|V|))

Но нужно еще и пометки обновить, а это как раз decrease_priority(...) - каждая тоже за O(log(|V|))

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности аккуратной реализации ч/р бинарные пирамиды:

$$O(|V| * log(|V|) + |E| * log(|V|))$$

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности аккуратной реализации ч/р бинарные пирамиды:

$$O(|V| * log(|V|) + |E| * log(|V|))$$

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности аккуратной реализации ч/р бинарные пирамиды:

$$O(|V| * log(|V|) + |E| * log(|V|))$$

Но можем ли мы еще лучше?

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности аккуратной реализации ч/р бинарные пирамиды:

$$O(|V| * log(|V|) + |E| * log(|V|))$$

Но можем ли мы еще лучше?

Да! Но начнем чуть издалека.

Сливаемые пирамиды

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value)
2. peek_min()
3. extract_min()
4. decrease_key(s, k)

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value)
2. peek_min()
3. extract_min()
4. decrease_key(s, k)
5. merge(H1, H2) - взять две сливаемые пирамиды и превратить в одну, объединить

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value)
2. peek_min()
3. extract_min()
4. decrease_key(s, k)
5. merge(H1, H2)
6. delete(s) - удалить данный элемент

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value)
2. peek_min()
3. extract_min()
4. decrease_key(s, k)
5. merge(H1, H2)
6. delete(s)

Как реализовать?

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value)
2. peek_min()
3. extract_min()
4. decrease_key(s, k)
5. merge(H1, H2)
6. delete(s)

Как реализовать?
Бинарная куча!

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(logN)
2. peek_min() -> O(1) 🥰
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2)
6. delete(s)

Как реализовать?
Бинарная куча!

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(logN)
2. peek_min() -> O(1) 🥰
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(???)
6. delete(s)

Как реализовать?
Бинарная куча!

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(logN)
2. peek_min() -> O(1) 🥰
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(N)
6. delete(s)

Как реализовать?
Бинарная куча!

Просто строим новую пирамиду из элементов из обеих старых пирамид

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(logN)
2. peek_min() -> O(1) 🥰
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(N)
6. delete(s) -> O(logN)

Как реализовать?
Бинарная куча!

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(logN)
2. peek_min() -> O(1) 🥰
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(N)
6. delete(s) -> O(logN)

Меняем с последним, сокращаем массив, просеиваем

Как реализовать? Бинарная куча!

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(logN)
2. peek_min() -> O(1) 🥰
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(N)
6. delete(s) -> O(logN)

Как реализовать?
Бинарная куча!

O(N) – неприятно, хочется оптимизировать merge

Биномиальные деревья

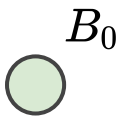
Биномиальные деревья

B_i — введем i-ое биномиальное дерево рекурсивно:

Биномиальные деревья

B_i — введем i-ое биномиальное дерево рекурсивно:

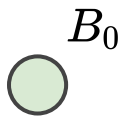
B_0 — дерево, состоящее из одного узла



Биномиальные деревья

B_i — введем i-ое биномиальное дерево рекурсивно:

B_0 — дерево, состоящее из одного узла

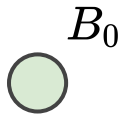


B_{k+1} — дерево, строится из двух деревьев B_k следующим образом: корень одного из деревьев становится левым сыном корня второго из деревьев.

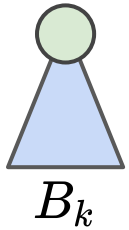
Биномиальные деревья

B_i — введем i-ое биномиальное дерево рекурсивно:

B_0 — дерево, состоящее из одного узла



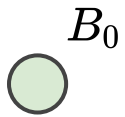
B_{k+1} — дерево, строится из двух деревьев B_k следующим образом: корень одного из деревьев становится левым сыном корня второго из деревьев.



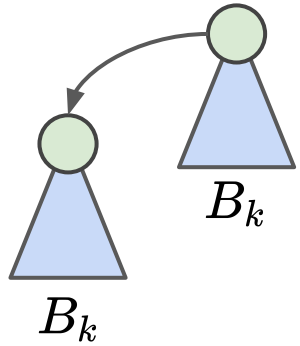
Биномиальные деревья

B_i — введем i-ое биномиальное дерево рекурсивно:

B_0 — дерево, состоящее из одного узла



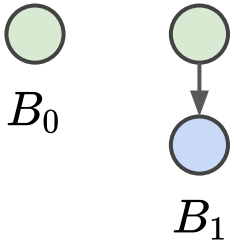
B_{k+1} — дерево, строится из двух деревьев B_k следующим образом: корень одного из деревьев становится левым сыном корня второго из деревьев.



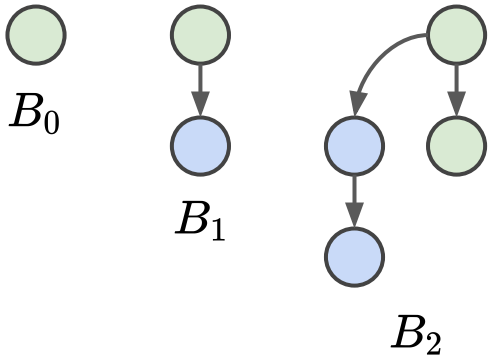
Биномиальные деревья

B_0

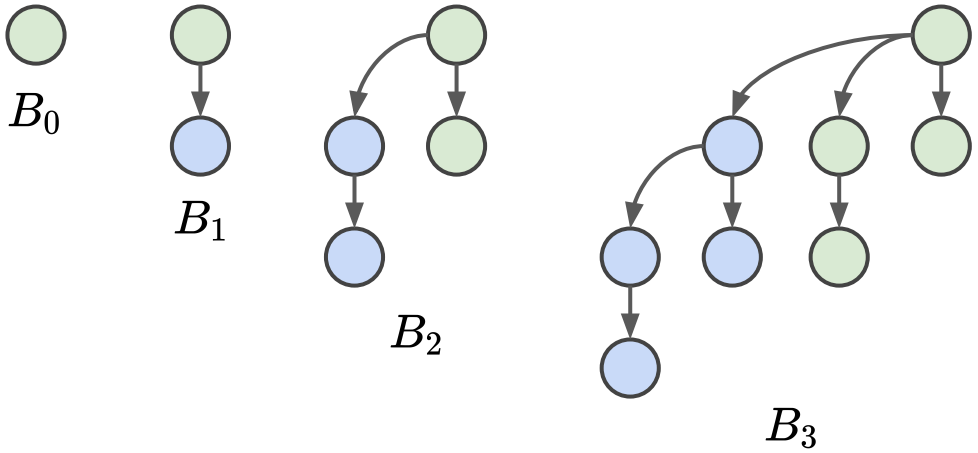
Биномиальные деревья



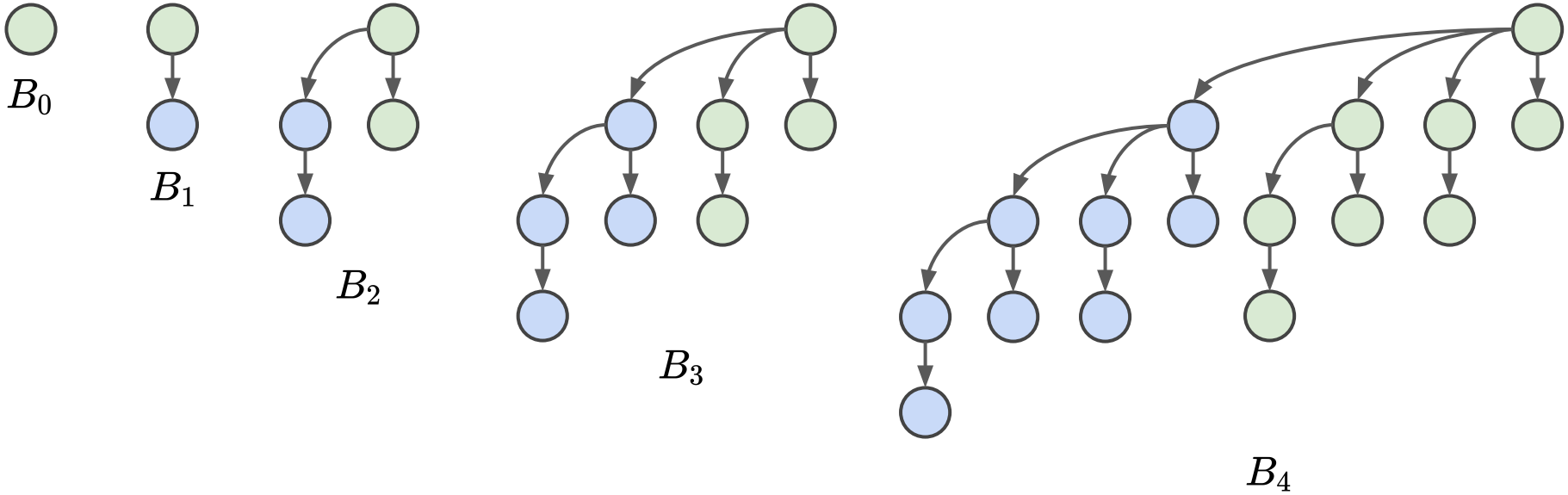
Биномиальные деревья



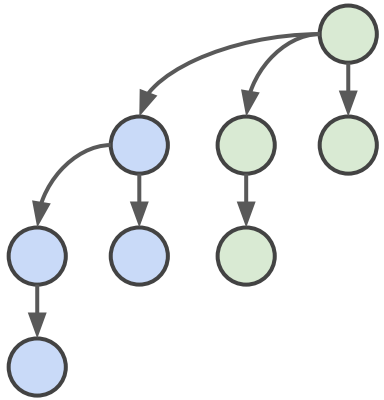
Биномиальные деревья



Биномиальные деревья



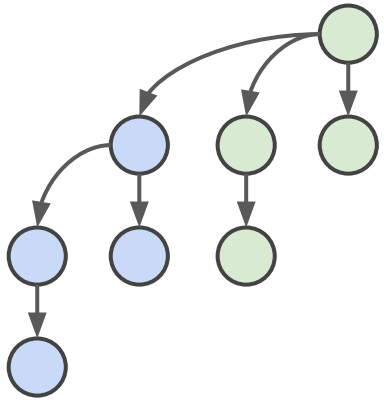
Биномиальные деревья: свойства



B_3

Биномиальные деревья: свойства

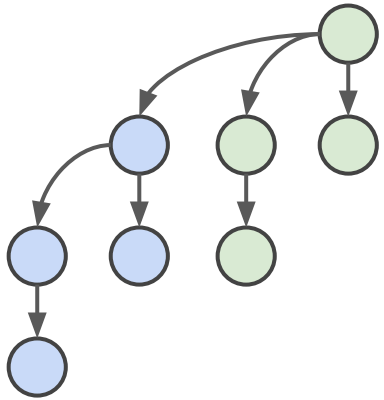
1. Сколько вершин в B_k?



B_3

Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k



B_3

Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k

   Доказательство: по индукции.

   Для k = 0 - верно.

B_0

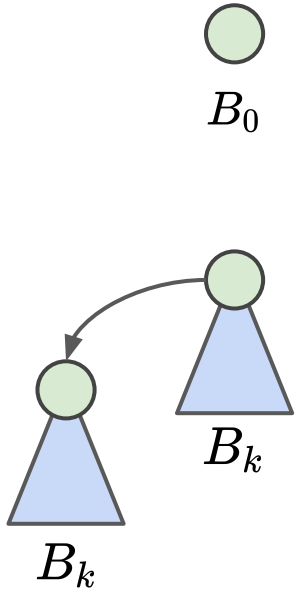
Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k

   Доказательство: по индукции.

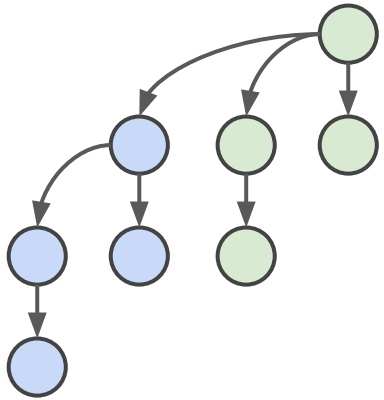
   Для k = 0 - верно.

   Если верно для B_k, то
   $|B_{k+1}| = |B_k| + |B_K| = 2^k + 2^k = 2^{k+1}$



Биномиальные деревья: свойства

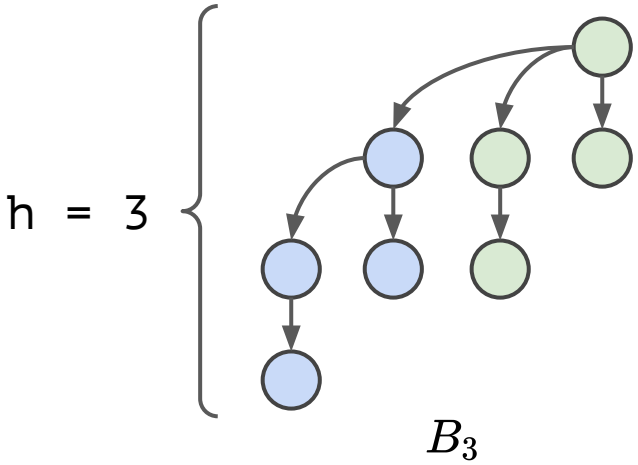
1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k?



B_3

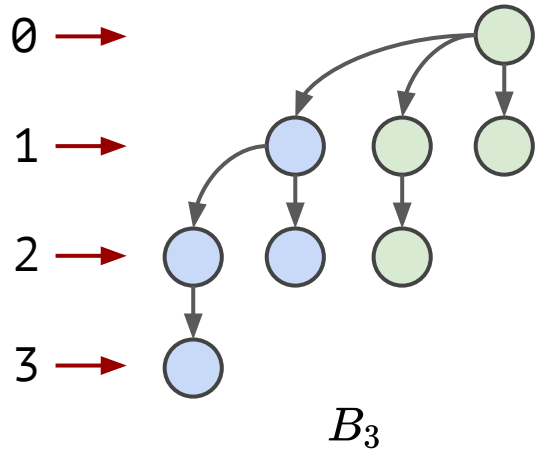
Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k



Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$?

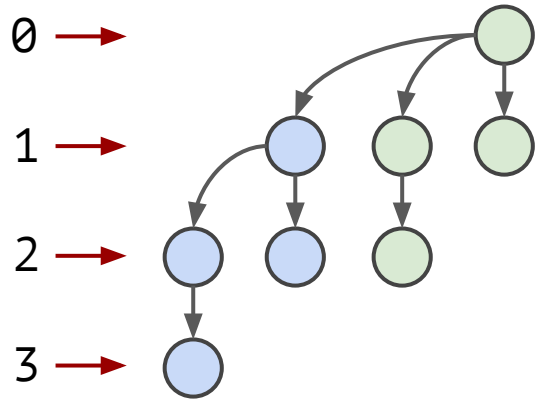


B_3

Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i

Именно поэтому они биномиальные!

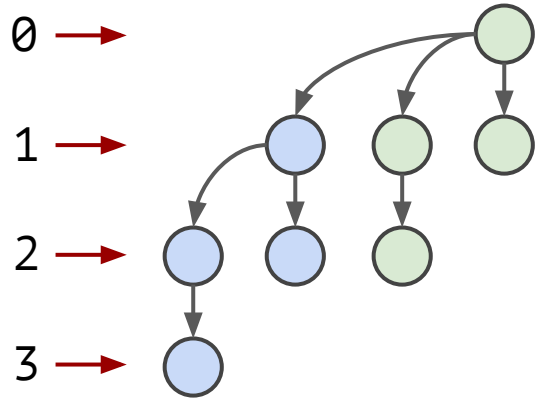


B_3

Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i

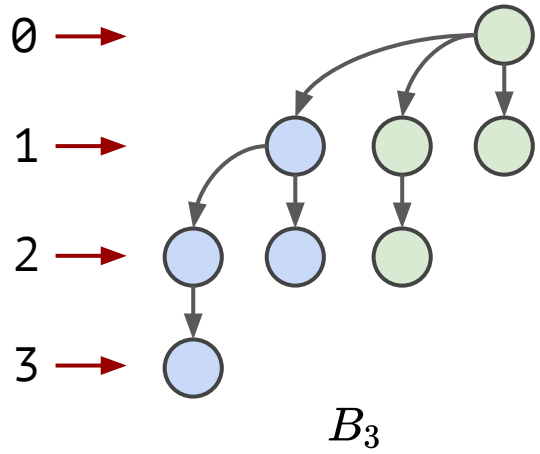
Док-во: индукция по k.
Для нуля верно (там только корень).



B_3

Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i



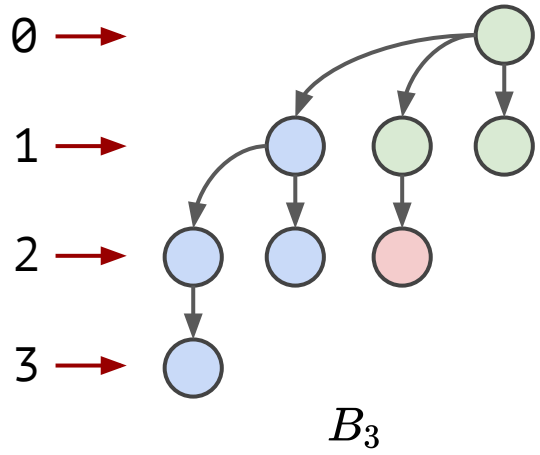
B_3

Док-во: индукция по k.
Для нуля верно (там только корень).

D(k, i) - количество вершин на i-ом уровне B_k

Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i



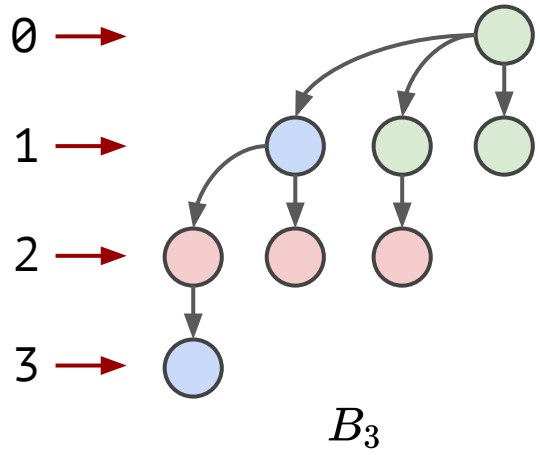
B_3

Док-во: индукция по k.
Для нуля верно (там только корень).

D(k, i) - количество вершин на i-ом уровне B_k. Заметим, что по построению дерева i-ый уровень состоит из i-ого уровня B_{k-1}

Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i



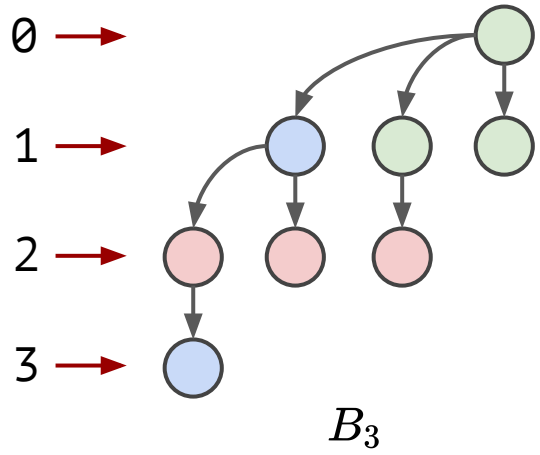
B_3

Док-во: индукция по k.
Для нуля верно (там только корень).

D(k, i) - количество вершин на i-ом уровне B_k. Заметим, что по построению дерева i-ый уровень состоит из i-ого уровня B_{k-1} и (i-1)-ого уровня B_{k-1}.

Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i



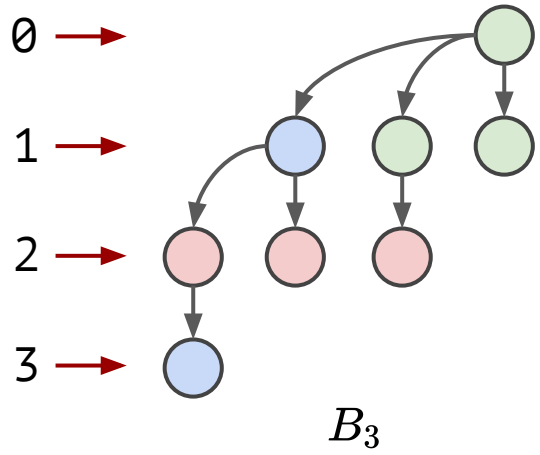
B_3

Док-во: индукция по k.
Для нуля верно (там только корень).

D(k, i) - количество вершин на i-ом уровне B_k. Тогда:
D(k, i) = D(k-1, i) + D(k-1, i-1) =

Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i



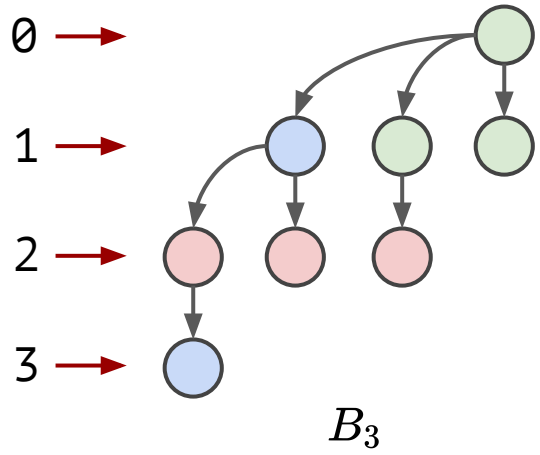
B_3

Док-во: индукция по k.
Для нуля верно (там только корень).

D(k, i) - количество вершин на i-ом уровне B_k. Тогда:
D(k, i) = D(k-1, i) + D(k-1, i-1) = $C_{k-1}^i + C_{k-1}^{i-1}$

Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i



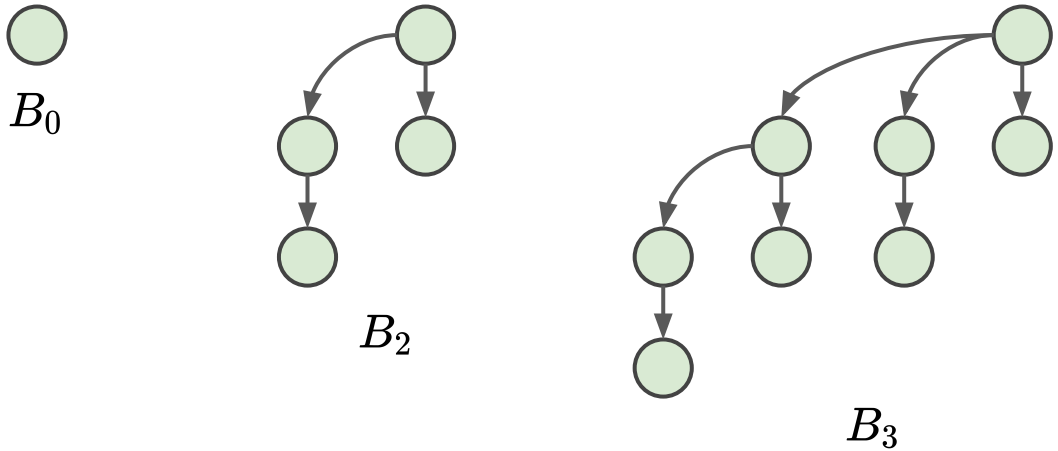
B_3

Док-во: индукция по k.
Для нуля верно (там только корень).

D(k, i) - количество вершин на i-ом уровне B_k. Тогда:
D(k, i) = D(k-1, i) + D(k-1, i-1) = $C_{k-1}^i + C_{k-1}^{i-1} = \binom{k-1}{i} + \binom{k-1}{i-1} = \binom{k}{i}$

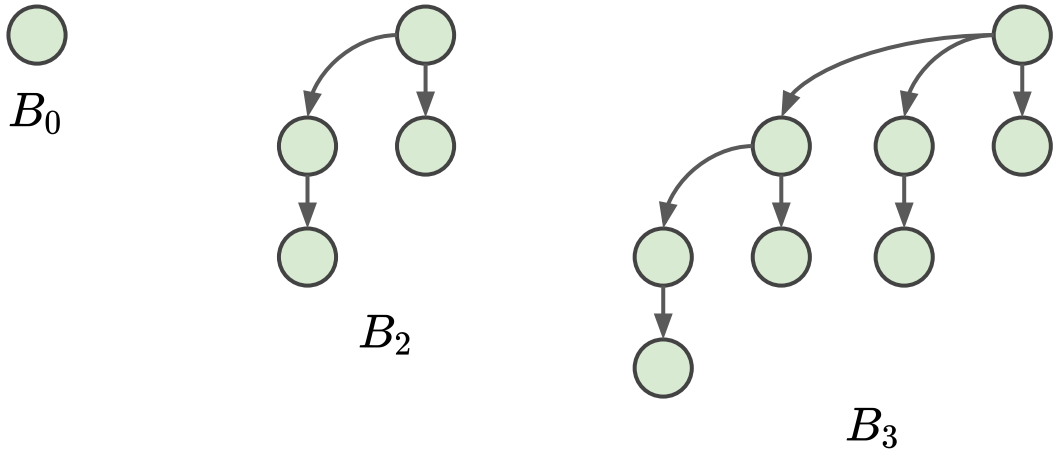
Биномиальная пирамида

Биномиальной пирамидой назовем множество биномиальных деревьев со следующими свойствами:



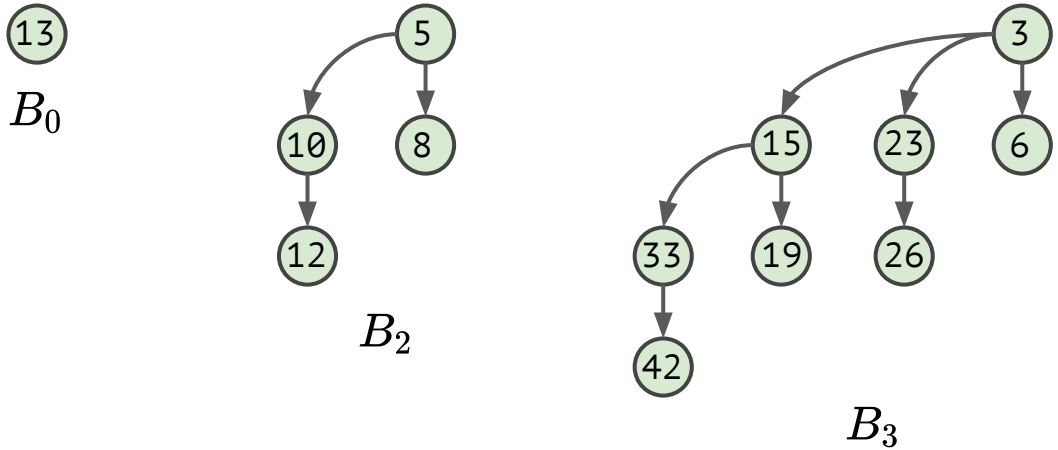
Биномиальной пирамидой назовем множество биномиальных деревьев со следующими свойствами:

1) Каждое биномиальное дерево подчиняется свойству неубывающей пирамиды: ключ узла не меньше ключа его родителя.



Биномиальной пирамидой назовем множество биномиальных деревьев со следующими свойствами:

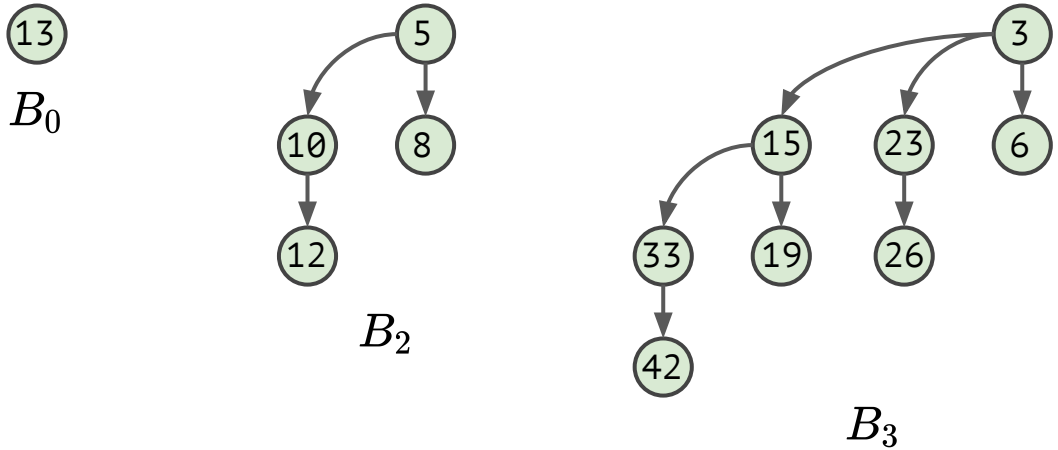
1) Каждое биномиальное дерево подчиняется свойству неубывающей пирамиды: ключ узла не меньше ключа его родителя.



Биномиальной пирамидой назовем множество биномиальных деревьев со следующими свойствами:

1) Каждое биномиальное дерево подчиняется свойству неубывающей пирамиды: ключ узла не меньше ключа его родителя.

2) Порядки всех биномиальных деревьев B_k, из которых состоит пирамида, попарно различны.

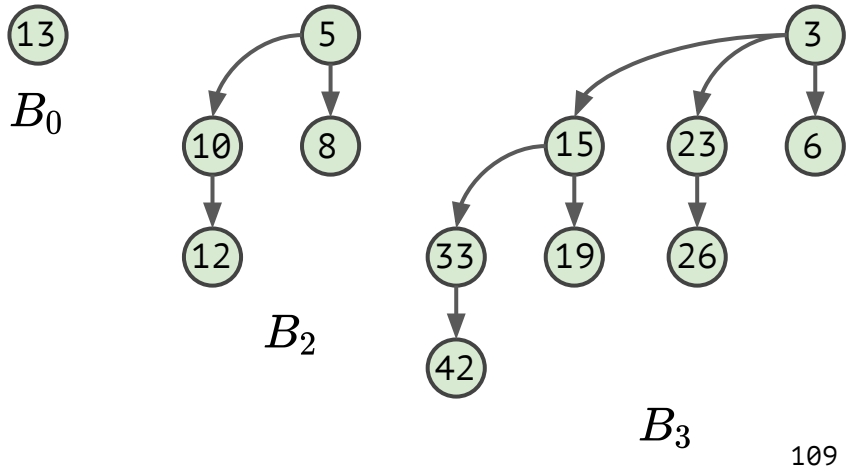


Биномиальной пирамидой назовем множество биномиальных деревьев со следующими свойствами:

1) Каждое биномиальное дерево подчиняется свойству неубывающей пирамиды: ключ узла не меньше ключа его родителя.

2) Порядки всех биномиальных деревьев B_k, из которых состоит пирамида, попарно различны.

Сколько всего может быть деревьев в пирамиде из N узлов?

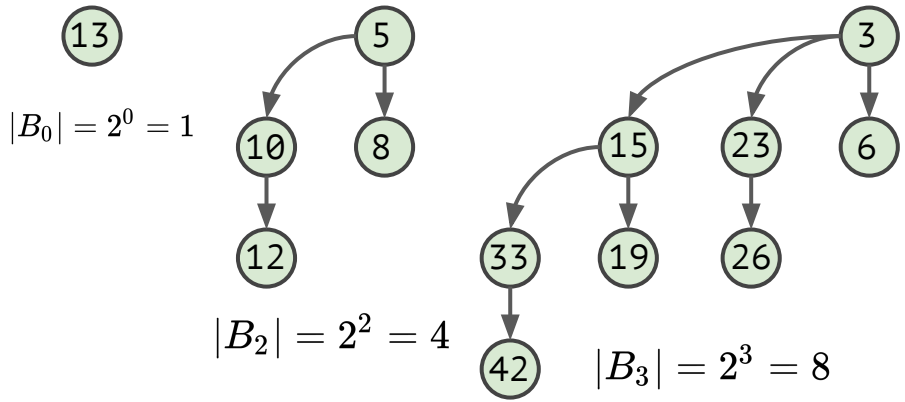


Биномиальной пирамидой назовем множество биномиальных деревьев со следующими свойствами:

1) Каждое биномиальное дерево подчиняется свойству неубывающей пирамиды: ключ узла не меньше ключа его родителя.

2) Порядки всех биномиальных деревьев B_k, из которых состоит пирамида, попарно различны.

Сколько всего может быть деревьев в пирамиде из N узлов?

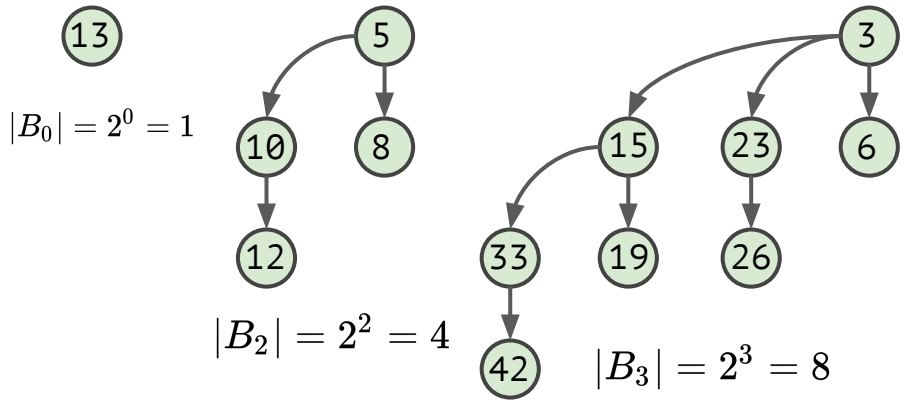


Биномиальной пирамидой назовем множество биномиальных деревьев со следующими свойствами:

1) Каждое биномиальное дерево подчиняется свойству неубывающей пирамиды: ключ узла не меньше ключа его родителя.

2) Порядки всех биномиальных деревьев B_k, из которых состоит пирамида, попарно различны.

Сколько всего может быть деревьев в пирамиде из N узлов?



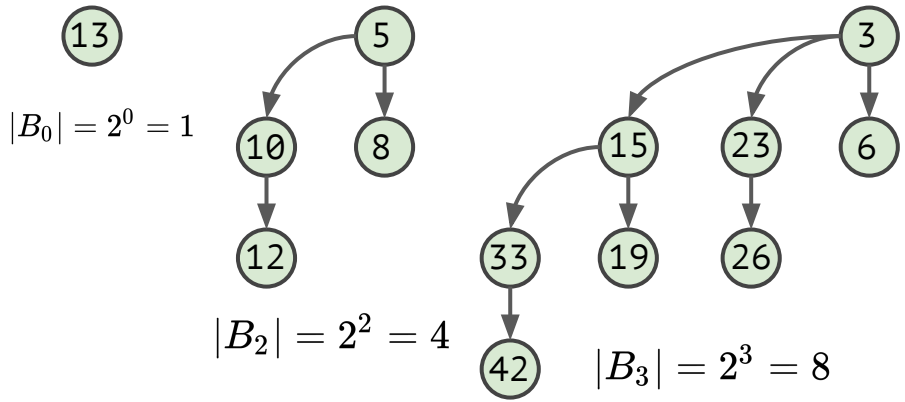
$$N = 1 * |B_0| + 0 * |B_1| + 1 * |B_2| + 1 * |B_3|$$

Биномиальной пирамидой назовем множество биномиальных деревьев со следующими свойствами:

1) Каждое биномиальное дерево подчиняется свойству неубывающей пирамиды: ключ узла не меньше ключа его родителя.

2) Порядки всех биномиальных деревьев B_k, из которых состоит пирамида, попарно различны.

Сколько всего может быть деревьев в пирамиде из N узлов?



$$N = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 1 * 2^3 = 13$$

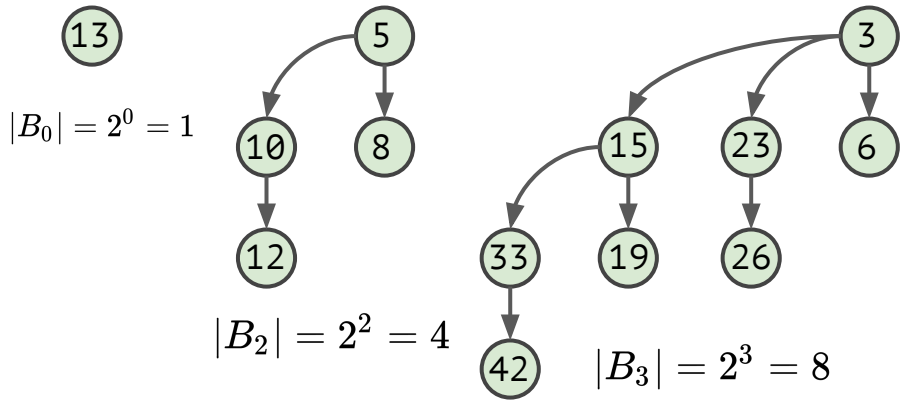
Биномиальной пирамидой назовем множество биномиальных деревьев со следующими свойствами:

1) Каждое биномиальное дерево подчиняется свойству неубывающей пирамиды: ключ узла не меньше ключа его родителя.

2) Порядки всех биномиальных деревьев B_k, из которых состоит пирамида, попарно различны.

Сколько всего может быть деревьев в пирамиде из N узлов?

Общее число узлов раскладывается по степеням двойки на размеры биномиальных деревьев.



$$N = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 1 * 2^3 = 13$$

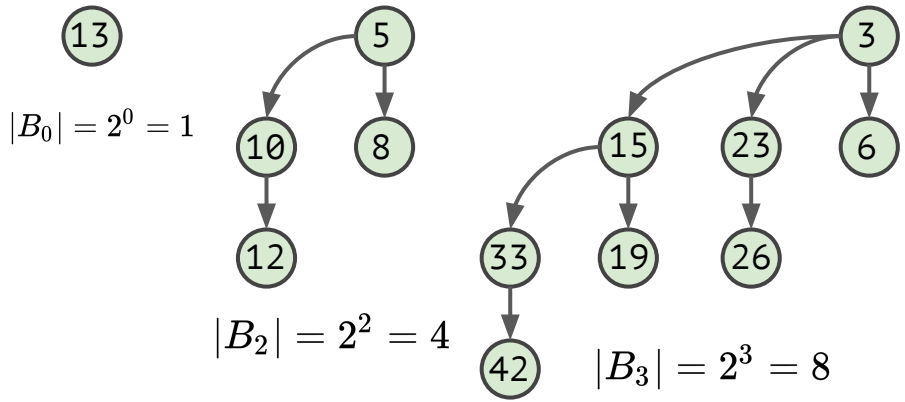
Биномиальной пирамидой назовем множество биномиальных деревьев со следующими свойствами:

1) Каждое биномиальное дерево подчиняется свойству неубывающей пирамиды: ключ узла не меньше ключа его родителя.

2) Порядки всех биномиальных деревьев B_k, из которых состоит пирамида, попарно различны.

Сколько всего может быть деревьев в пирамиде из N узлов?

Общее число узлов раскладывается по степеням двойки на размеры биномиальных деревьев. Тогда их общее количество $\leq \lfloor logN \rfloor + 1$



$$N = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 1 * 2^3 = 13$$

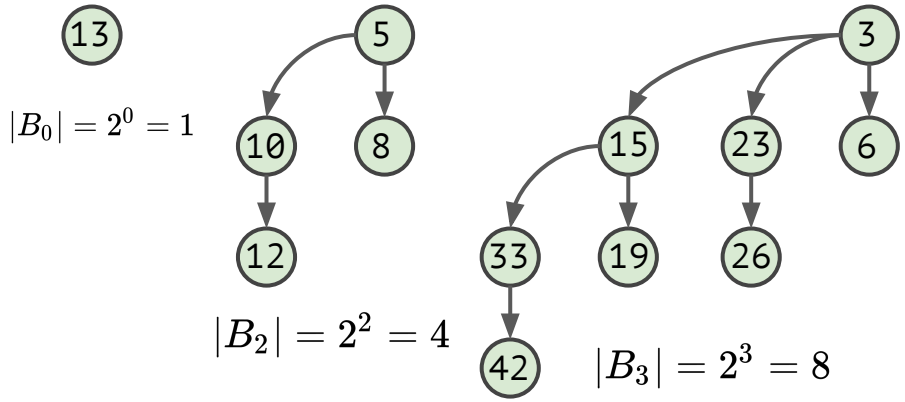
Биномиальной пирамидой назовем множество биномиальных деревьев со следующими свойствами:

1) Каждое биномиальное дерево подчиняется свойству неубывающей пирамиды: ключ узла не меньше ключа его родителя.

2) Порядки всех биномиальных деревьев B_k, из которых состоит пирамида, попарно различны.

Сколько всего может быть деревьев в пирамиде из N узлов?

Общее число узлов раскладывается по степеням двойки на размеры биномиальных деревьев. Тогда их общее количество $\leq \lfloor logN \rfloor + 1$ (как и битов в представлении числа N)



$$N = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 1 * 2^3 = 13$$

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(logN)
2. peek_min() -> O(1) 🥰
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(N)
6. delete(s) -> O(logN)

Как реализовать?
Бинарная куча!

O(N) – неприятно, хочется оптимизировать merge

Биномиальная пирамида

Множество значений: пары <priority: int, value: T>

Операции:

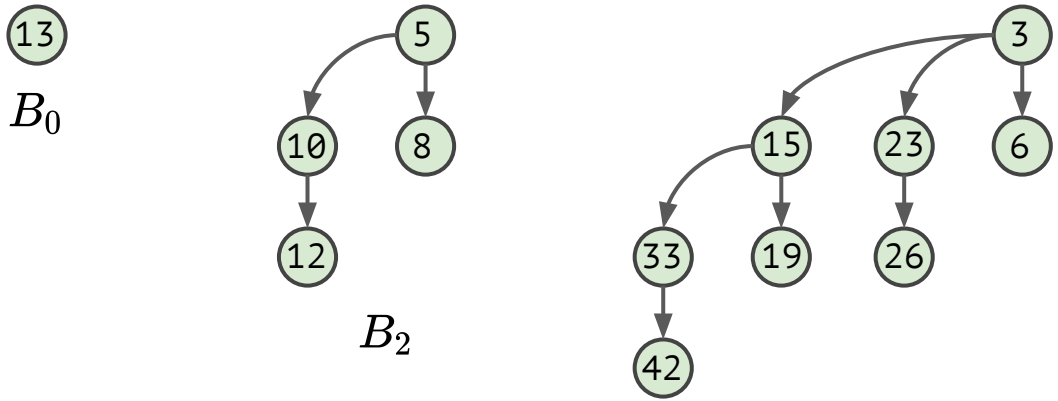
1. insert(value) -> O(???)
2. peek_min() -> O(???)
3. extract_min() -> O(???)
4. decrease_key(s, k) -> O(???)
5. merge(H1, H2) -> O(???)
6. delete(s) -> O(???)

Как реализовать?

~~Бинарная куча!~~

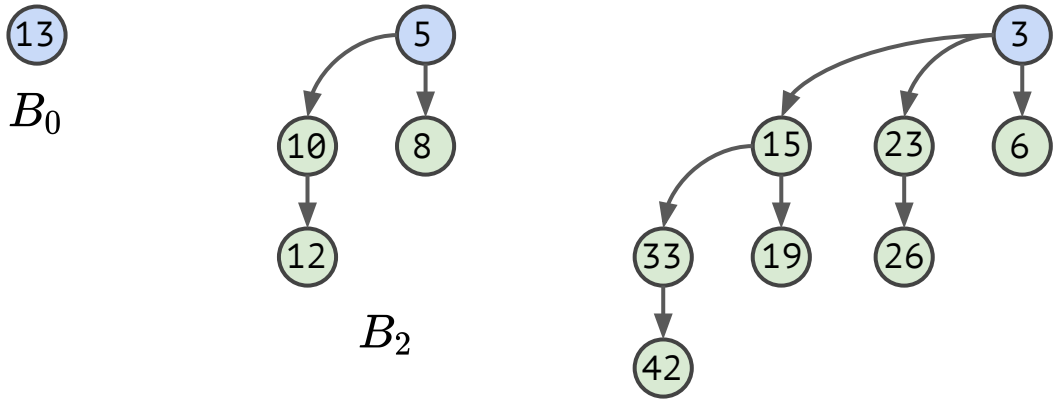
Биномиальная!

Биномиальная пирамида: peek_min



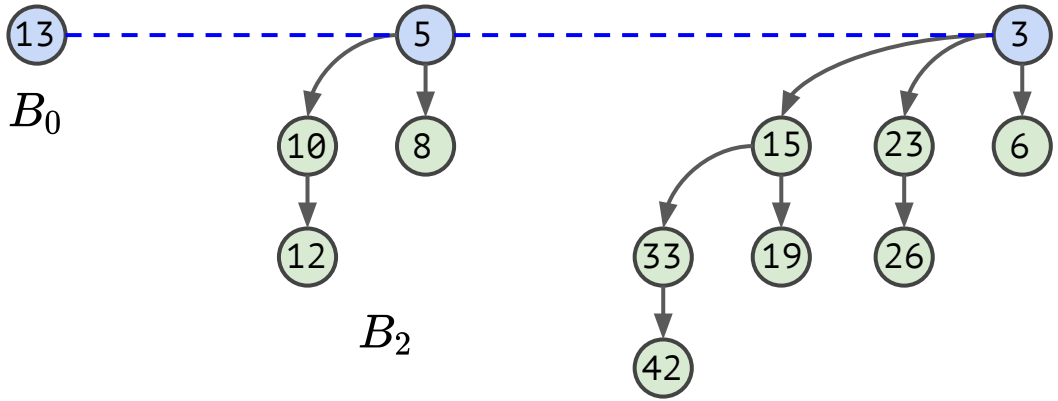
Где минимум?

Биномиальная пирамида: peek_min



Где минимум?
В одном из корней!

Биномиальная пирамида: peek_min

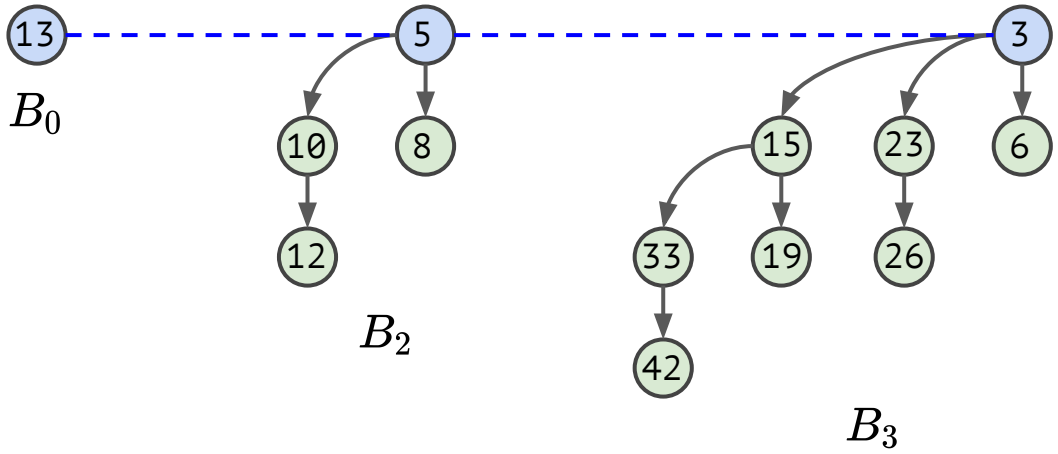


Где минимум?

В одном из корней!

Ищем линейным поиском.

Биномиальная пирамида: peek_min



Где минимум?
В одном из корней!
Ищем линейным поиском.
Сложность?

Биномиальная пирамида

Множество значений: пары <priority: int, value: T>

Операции:

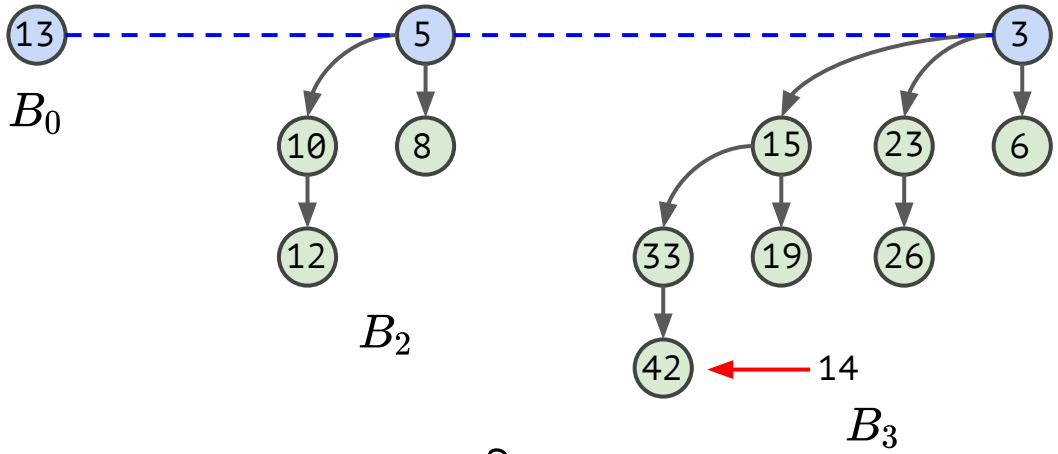
1. insert(value) -> O(???)
2. peek_min() -> O(logN)
3. extract_min() -> O(???)
4. decrease_key(s, k) -> O(???)
5. merge(H1, H2) -> O(???)
6. delete(s) -> O(???)

Как реализовать?

~~Бинарная куча!~~

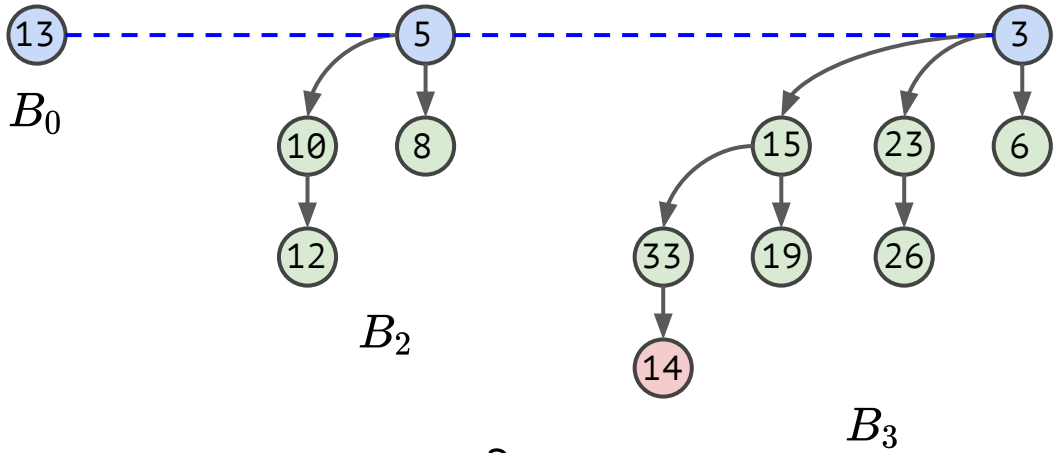
Биномиальная!

Биномиальная пирамида: decrease_key



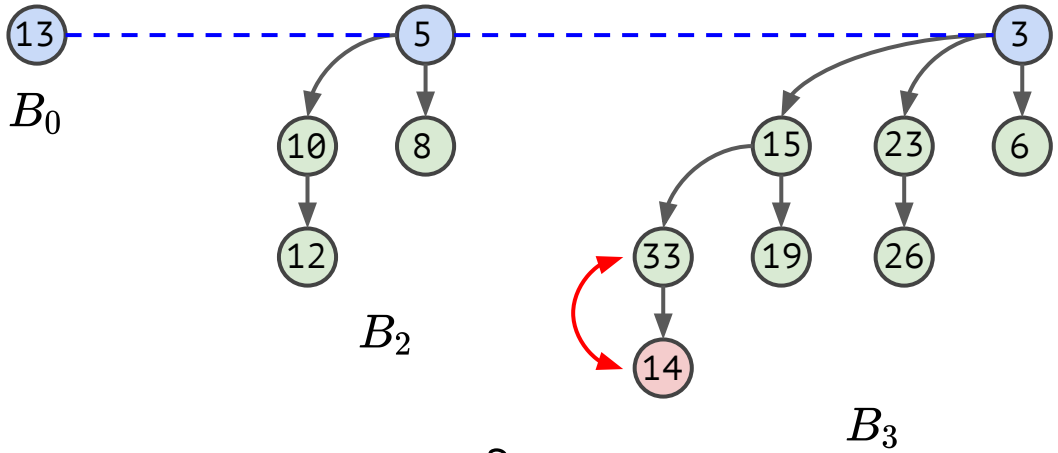
Как уменьшить ключ заданного элемента?

Биномиальная пирамида: decrease_key



Как уменьшить ключ заданного элемента?
Если просто заменить, пирамида сломается.

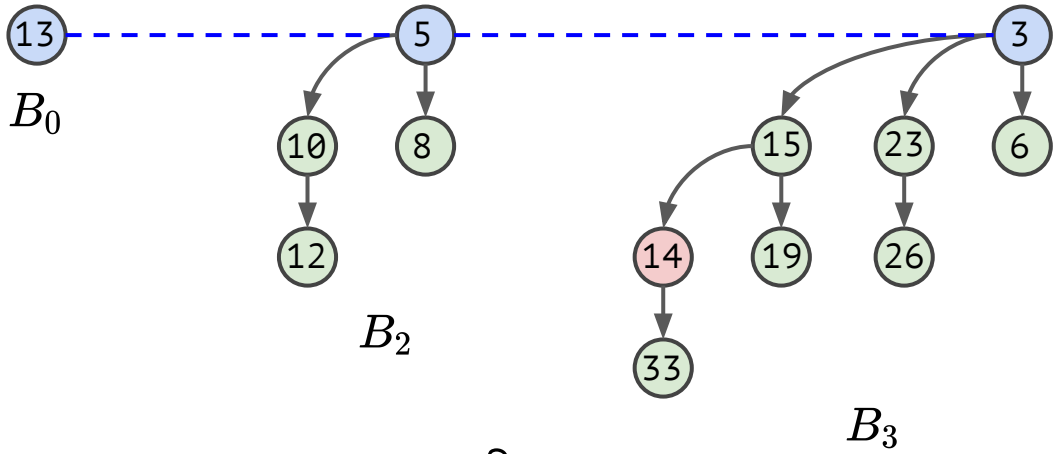
Биномиальная пирамида: decrease_key



Как уменьшить ключ заданного элемента?

Если просто заменить, пирамида сломается. Чиним просеиванием вверх.

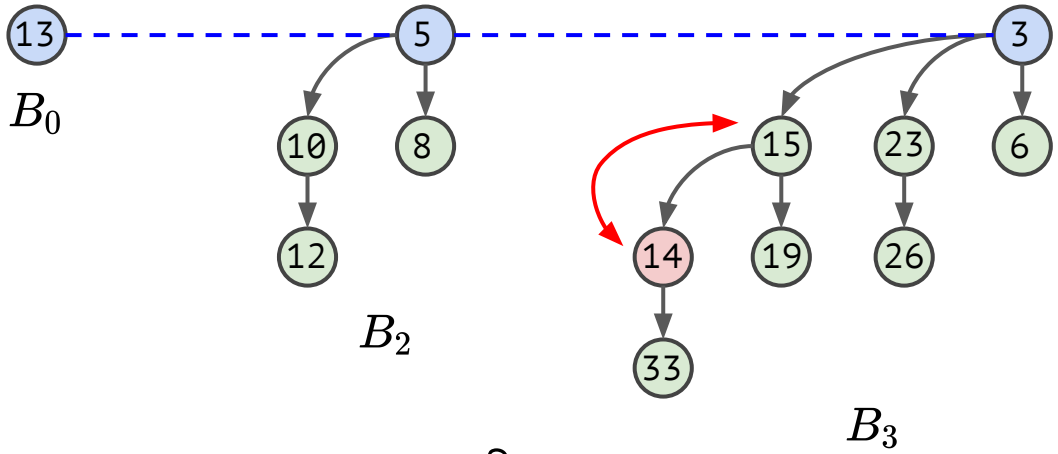
Биномиальная пирамида: decrease_key



Как уменьшить ключ заданного элемента?

Если просто заменить, пирамида сломается. Чиним просеиванием вверх.

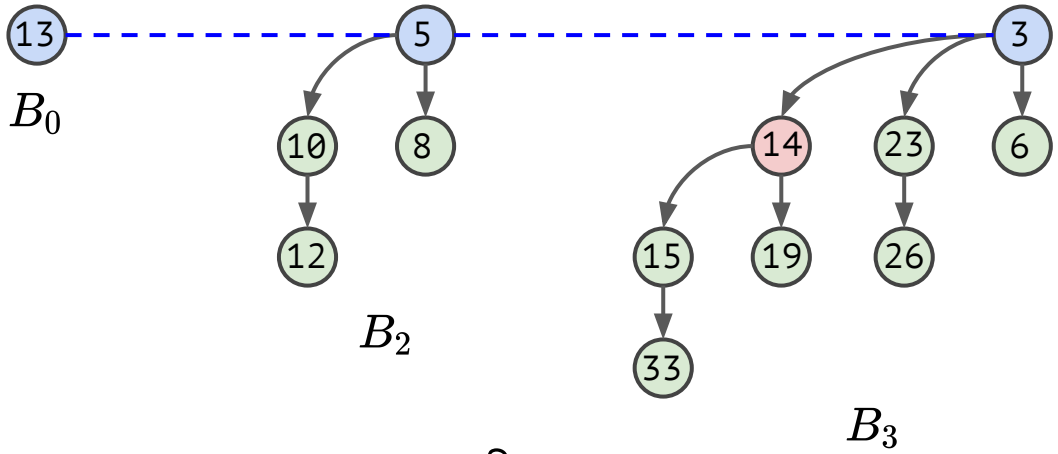
Биномиальная пирамида: decrease_key



Как уменьшить ключ заданного элемента?

Если просто заменить, пирамида сломается. Чиним просеиванием вверх.

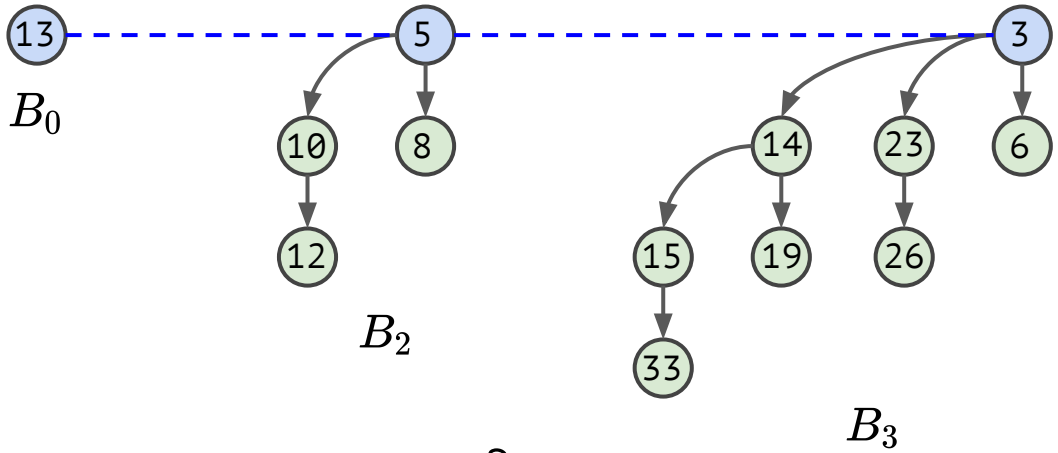
Биномиальная пирамида: decrease_key



Как уменьшить ключ заданного элемента?

Если просто заменить, пирамида сломается. Чиним просеиванием вверх.

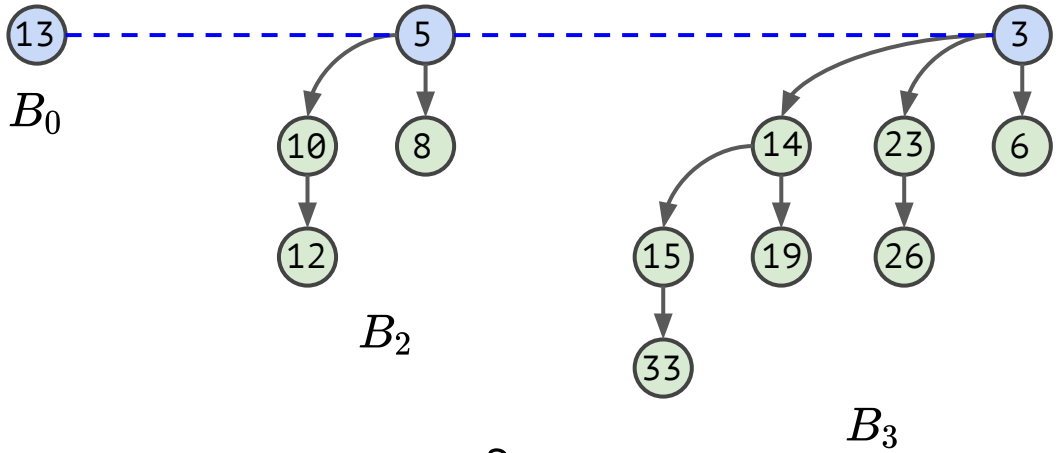
Биномиальная пирамида: decrease_key



Как уменьшить ключ заданного элемента?

Если просто заменить, пирамида сломается. Чиним просеиванием вверх.

Биномиальная пирамида: decrease_key

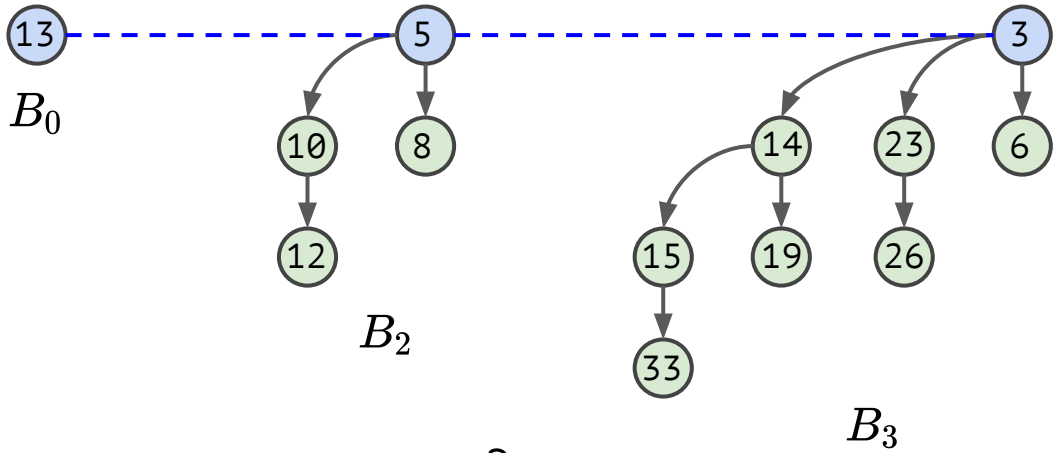


Как уменьшить ключ заданного элемента?

Если просто заменить, пирамида сломается. Чиним просеиванием вверх.

Сложность?

Биномиальная пирамида: decrease_key

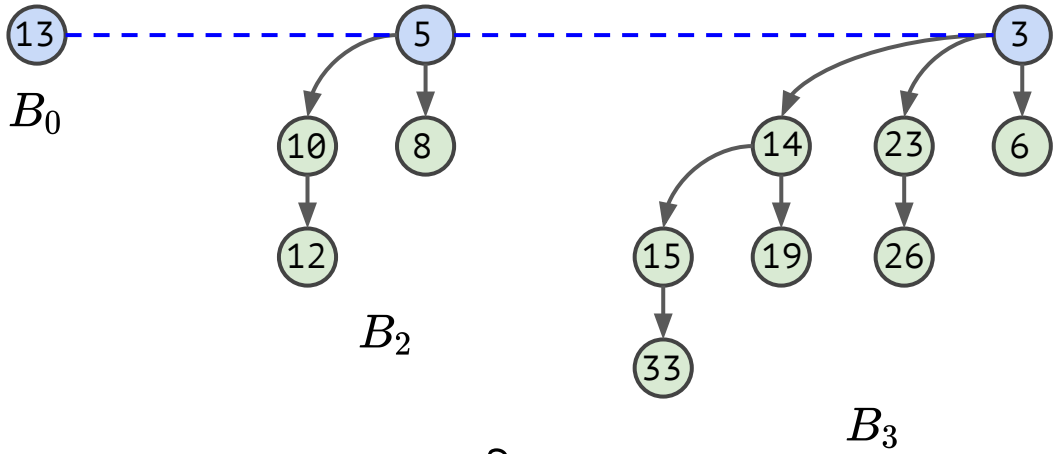


Как уменьшить ключ заданного элемента?

Если просто заменить, пирамида сломается. Чиним просеиванием вверх.

Сложность? Высота B_k – k

Биномиальная пирамида: decrease_key

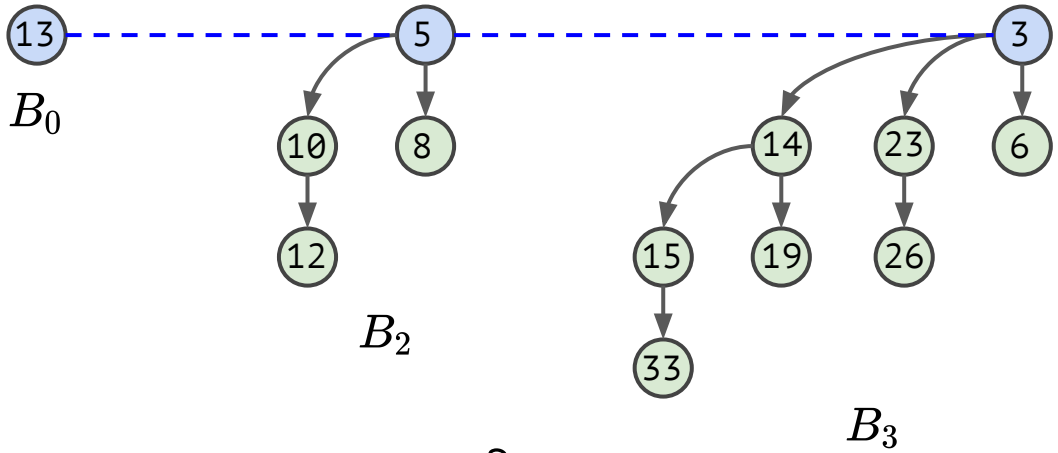


Как уменьшить ключ заданного элемента?

Если просто заменить, пирамида сломается. Чиним просеиванием вверх.

Сложность? Высота B_k – k; а самое большое k – logN (вспоминаем про разложение на степени двойки).

Биномиальная пирамида: decrease_key



Как уменьшить ключ заданного элемента?

Если просто заменить, пирамида сломается. Чиним просеиванием вверх.

Сложность? Высота B_k – k; а самое большое k – logN (вспоминаем про разложение на степени двойки) => сложность – O(logN)

Биномиальная пирамида

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(???)
2. peek_min() -> O(logN)
3. extract_min() -> O(???)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(???)
6. delete(s) -> O(???)

Как реализовать?

~~Бинарная куча!~~

Биномиальная!

Биномиальная пирамида: merge

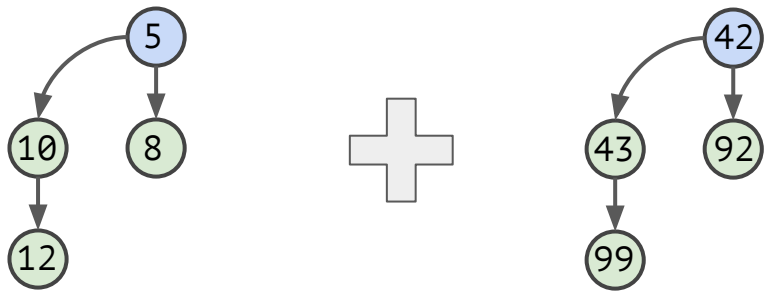
Наблюдения:

1. Проблема merge в том, что пирамидах могут быть биномиальные деревья одинакового порядка.

Биномиальная пирамида: merge

Наблюдения:

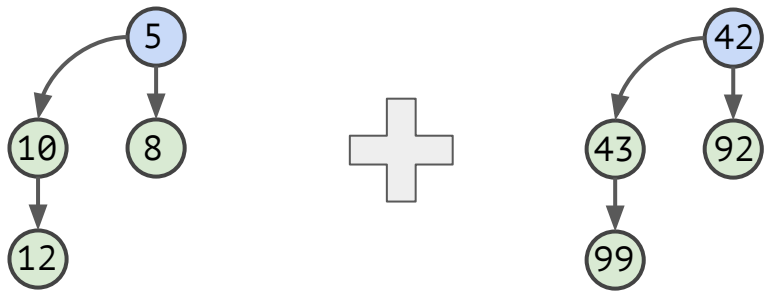
1. Проблема merge в том, что пирамидах могут быть биномиальные деревья одинакового порядка.



Биномиальная пирамида: merge

Наблюдения:

1. Проблема merge в том, что пирамидах могут быть биномиальные деревья одинакового порядка.

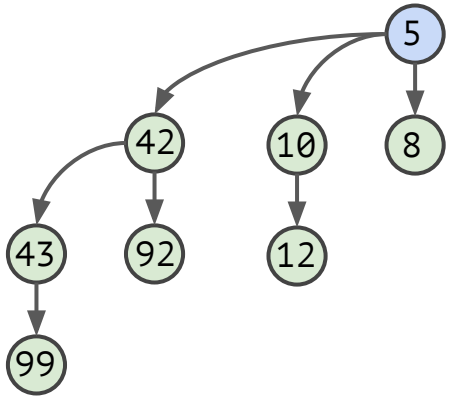


Как слить два дерева порядка k?

Биномиальная пирамида: merge

Наблюдения:

1. Проблема merge в том, что пирамидах могут быть биномиальные деревья одинакового порядка.



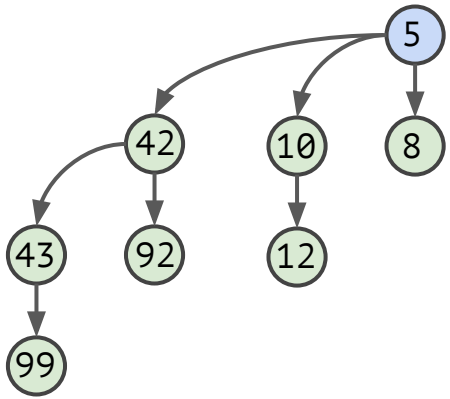
Делаем дерево с большим ключом корня левым сыном корня с меньшим ключом.

Как слить два дерева порядка k?

Биномиальная пирамида: merge

Наблюдения:

1. Проблема merge в том, что пирамидах могут быть биномиальные деревья одинакового порядка.



Делаем дерево с большим ключом корня левым сыном корня с меньшим ключом.

Как слить два дерева порядка k?

Назовем это merge_tree(t1, t2), работает за O(1), возвращает дерево порядка k+1

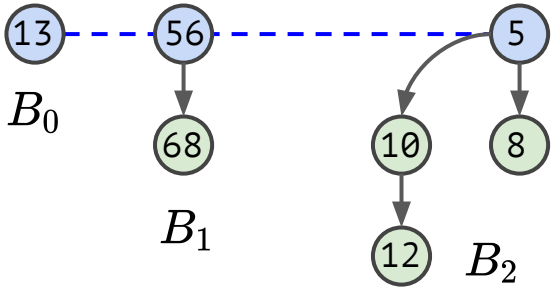
Биномиальная пирамида: merge

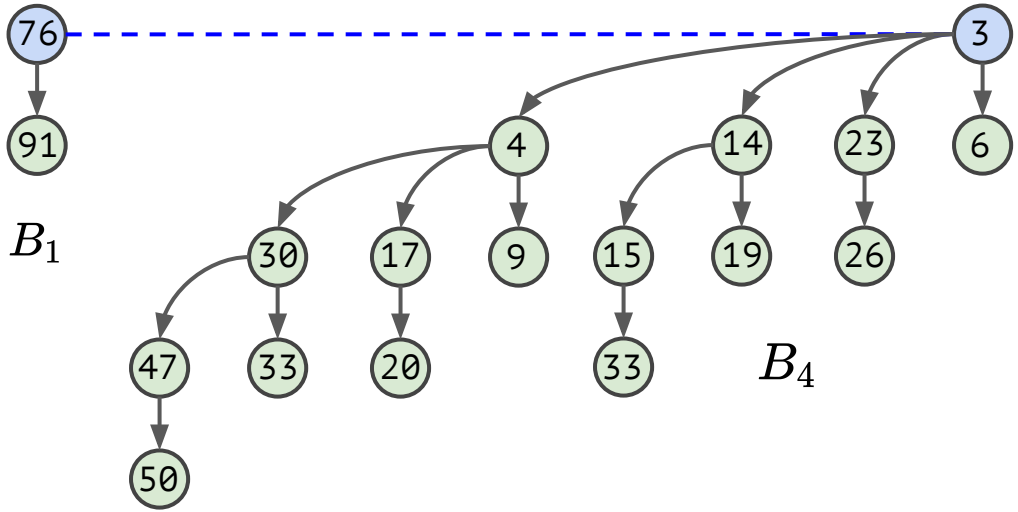
Наблюдения:

1. Проблема merge в том, что пирамидах могут быть биномиальные деревья одинакового порядка.

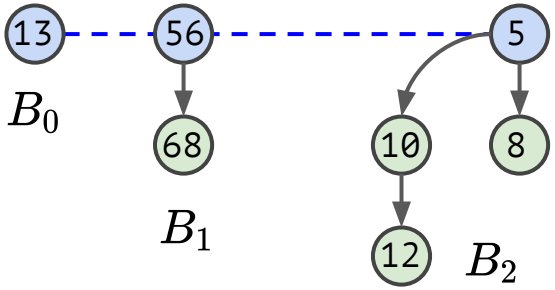
2. Слияние двух пирамид очень похоже на сложение двух чисел в двоичной записи (в столбик).

Слияние … похоже на сложение чисел в двоичной записи (в столбик)



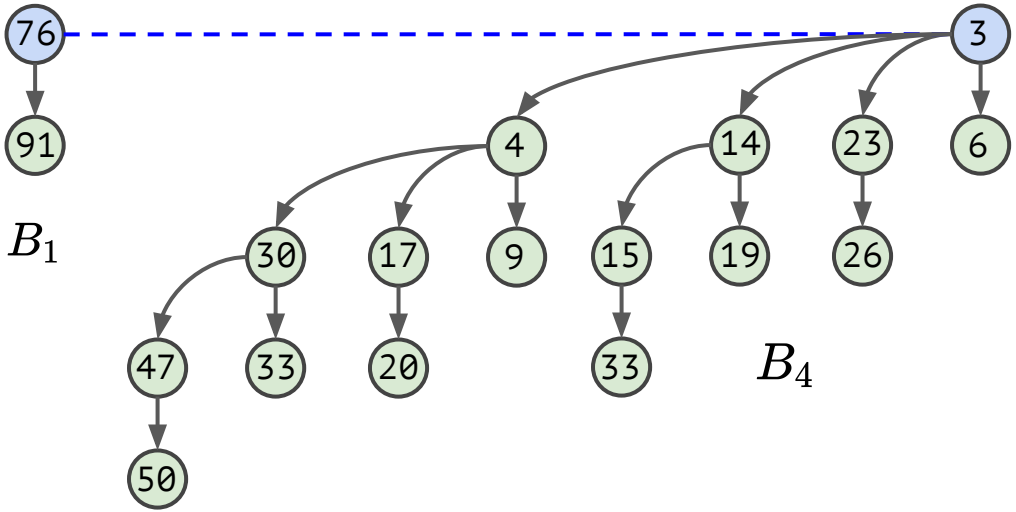


Слияние … похоже на сложение чисел в двоичной записи (в столбик)



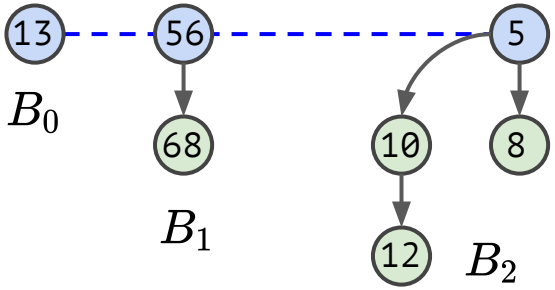
lsb слева

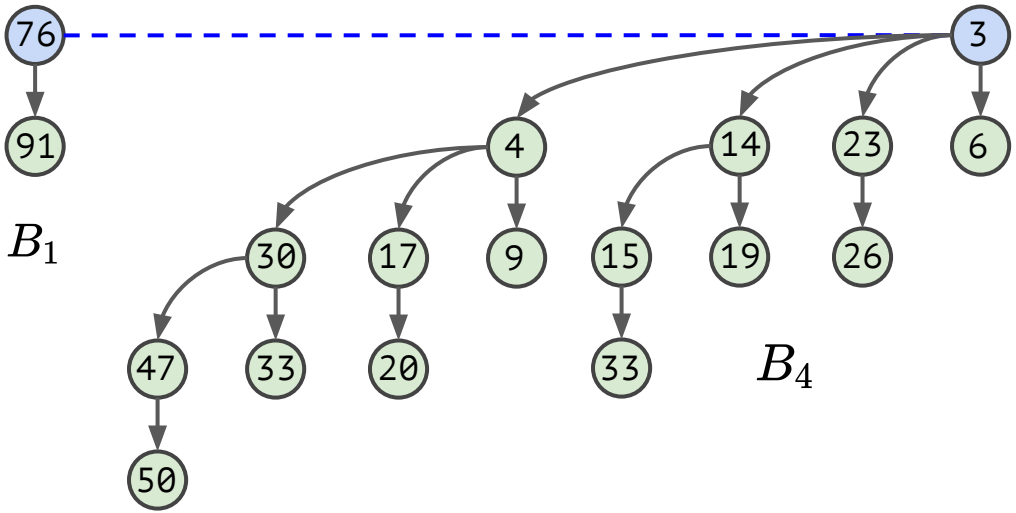
1	1	1	0	0

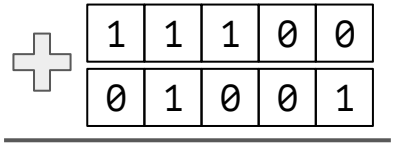


0	1	0	0	1

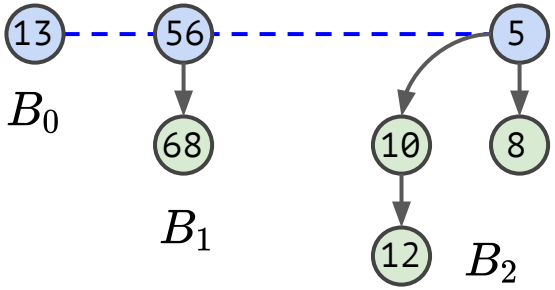
Слияние … похоже на сложение чисел в двоичной записи (в столбик)

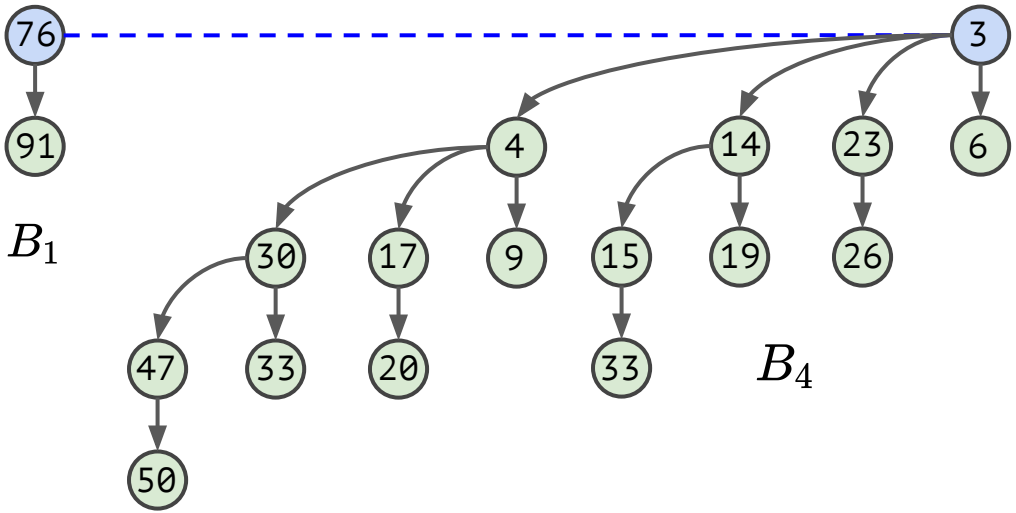


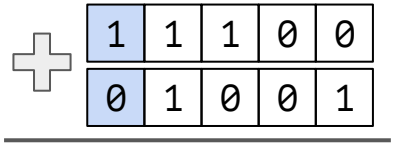




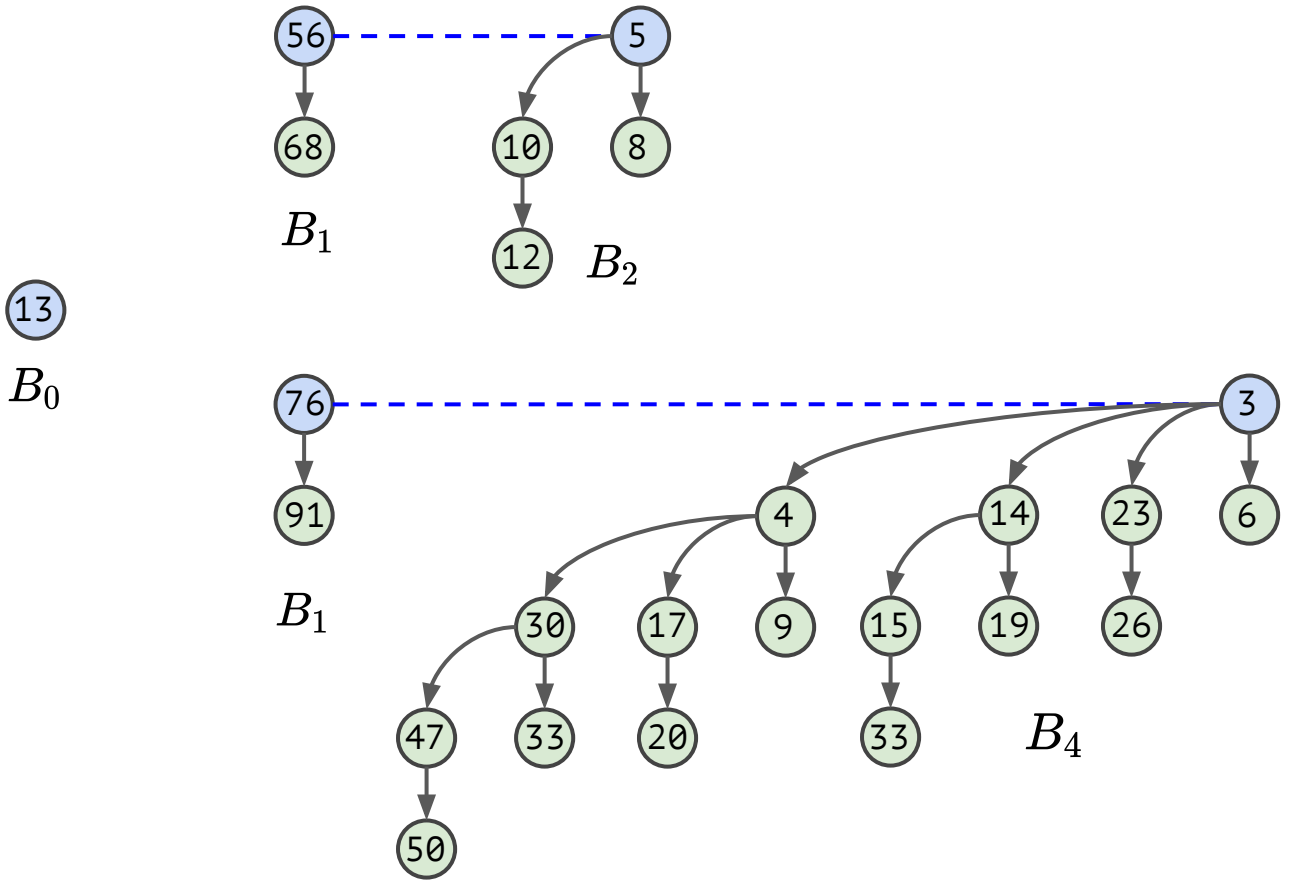
Слияние … похоже на сложение чисел в двоичной записи (в столбик)





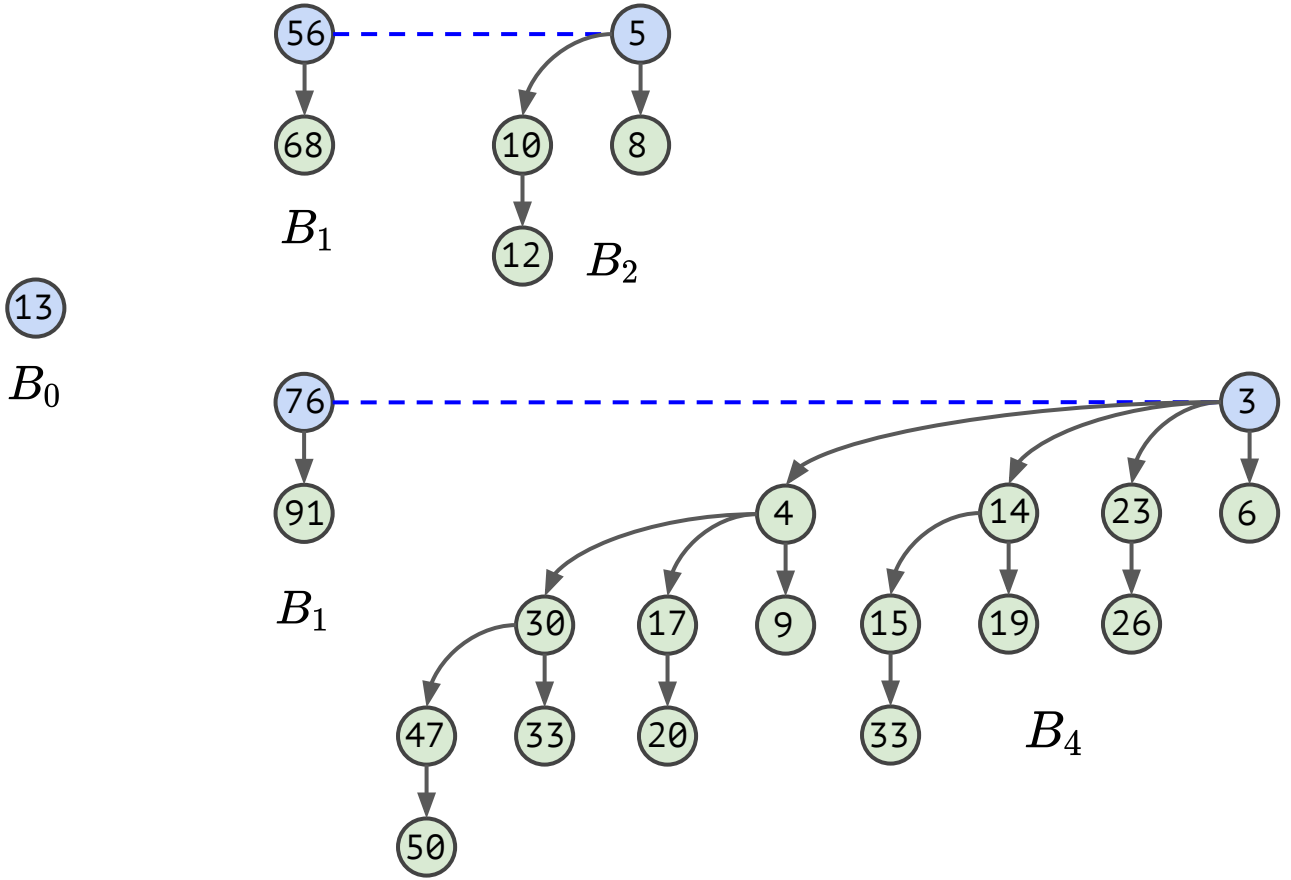


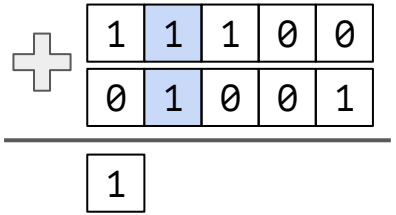
Слияние … похоже на сложение чисел в двоичной записи (в столбик)



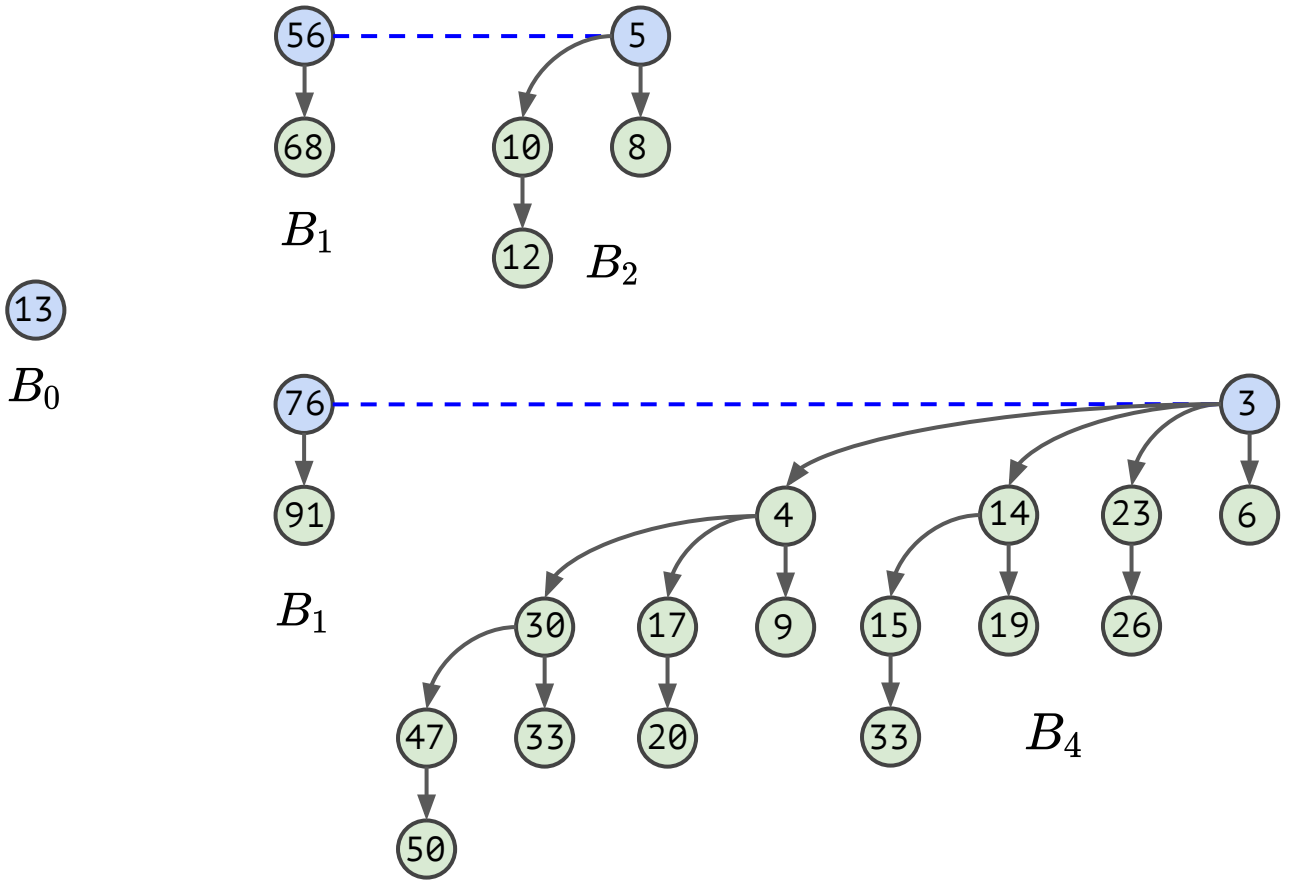
+	1	1	1	0	0
	0	1	0	0	1
	1				

Слияние … похоже на сложение чисел в двоичной записи (в столбик)





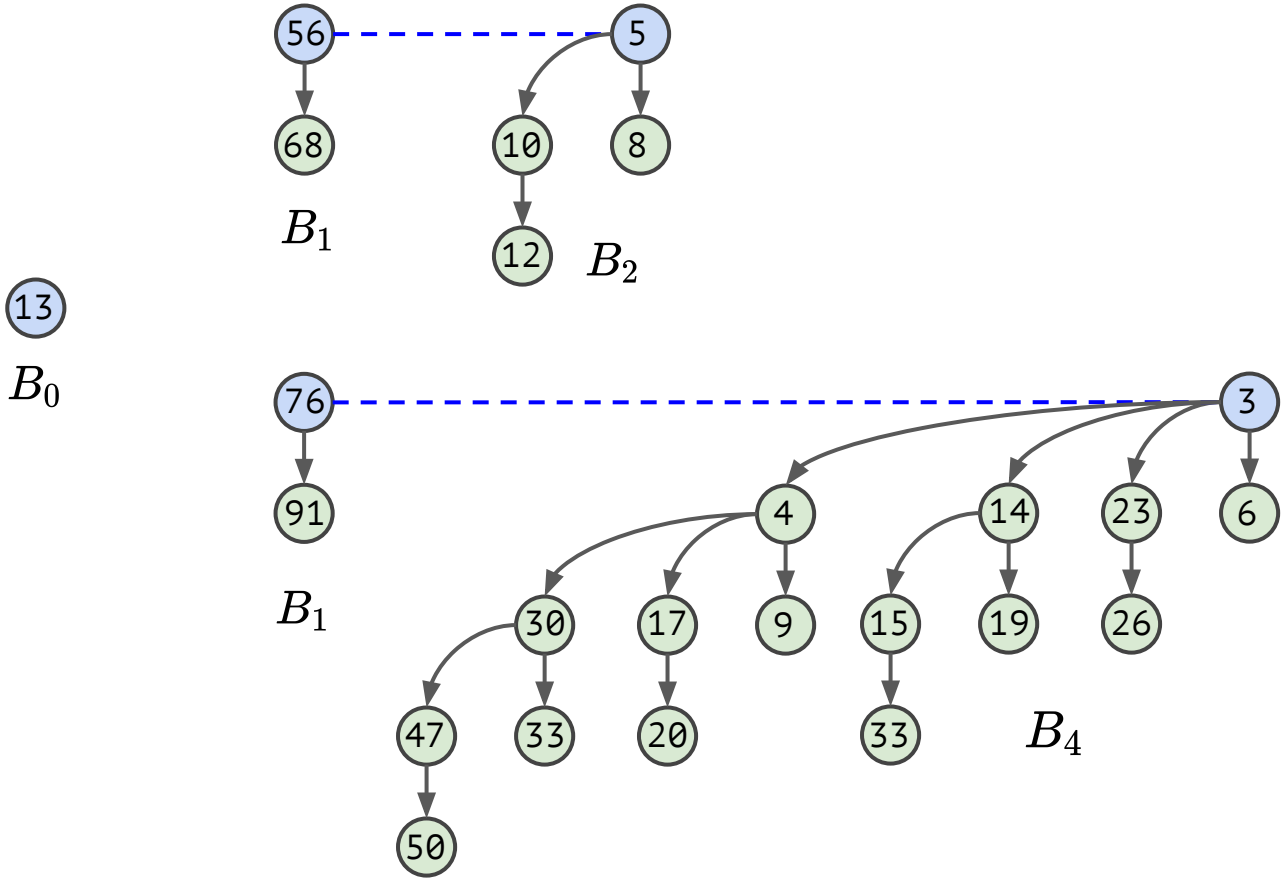
Слияние … похоже на сложение чисел в двоичной записи (в столбик)

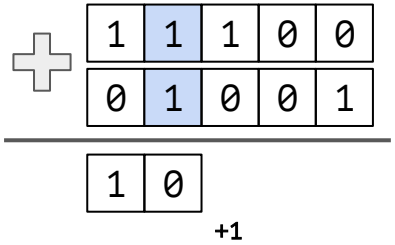


	1	1	1	0	0
+	0	1	0	0	1
	1	0			

Деревьев порядка 1 у нас точно не будет.

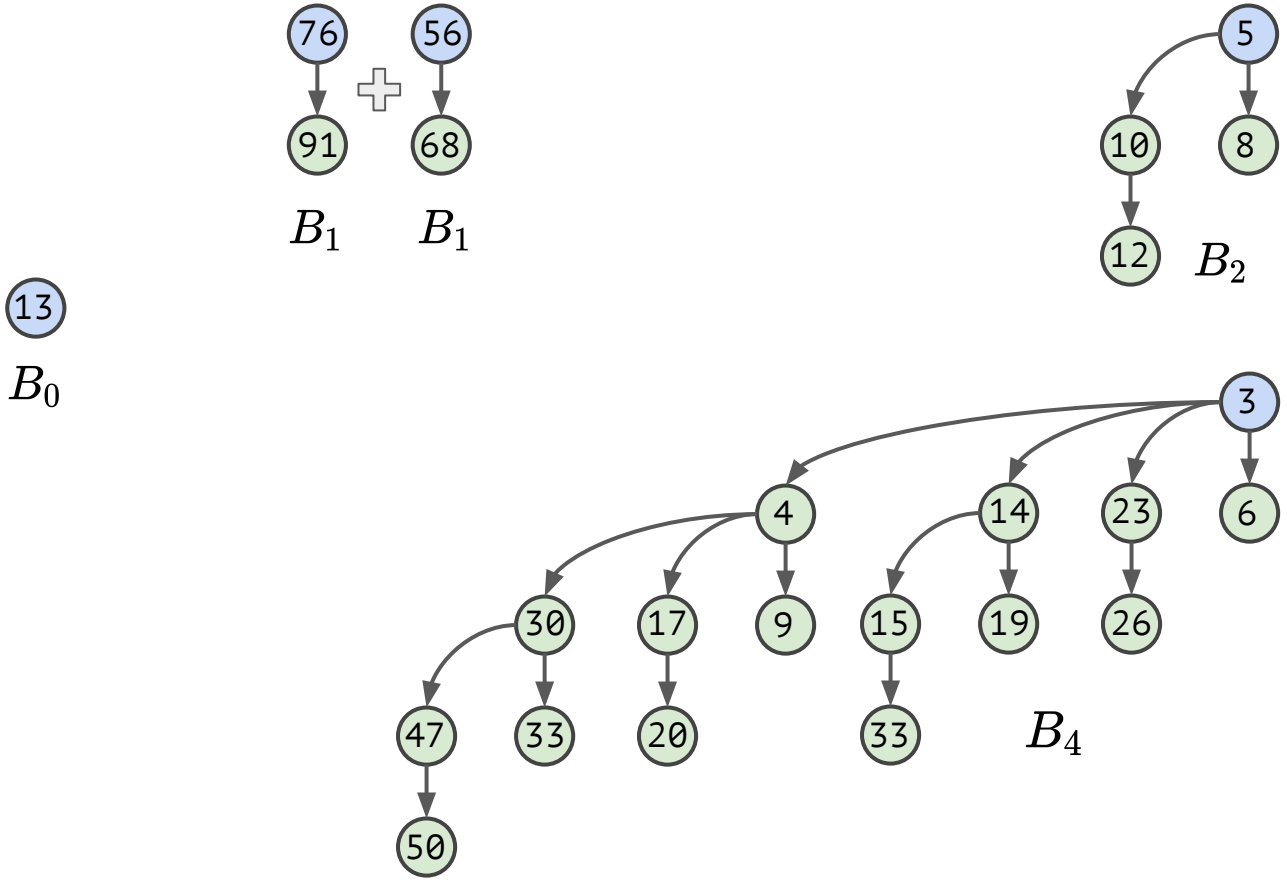
Слияние … похоже на сложение чисел в двоичной записи (в столбик)





Деревьев порядка 1 у нас точно не будет. Но еще держим единичку в уме!

Слияние … похоже на сложение чисел в двоичной записи (в столбик)

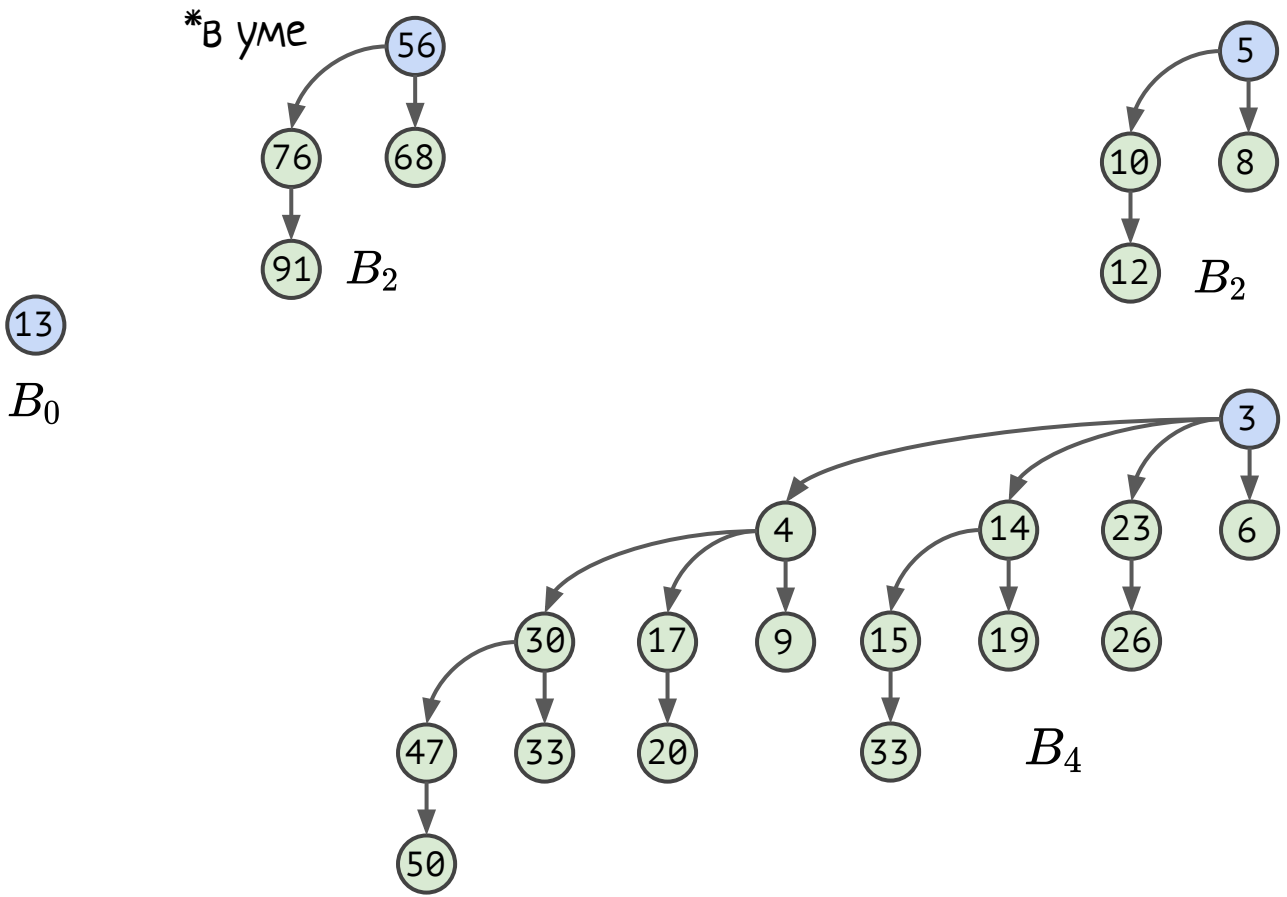


	1	1	1	0	0
+	0	1	0	0	1
	1	0			

+1

Деревьев порядка 1 у нас точно не будет. Но еще держим единичку в уме!

Слияние … похоже на сложение чисел в двоичной записи (в столбик)

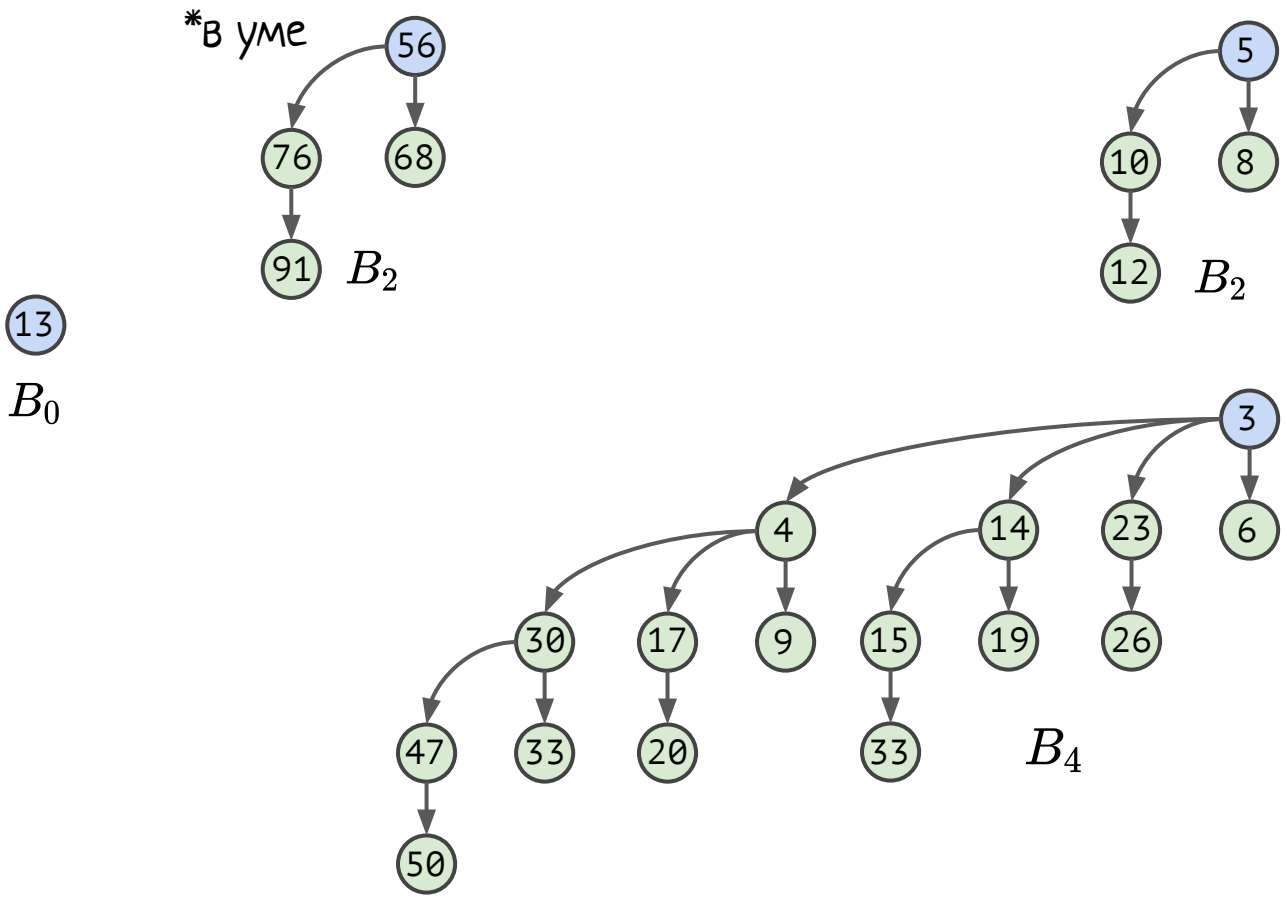


	1	1	1	0	0
+	0	1	0	0	1
	1	0			

+1

Деревьев порядка 1 у нас точно не будет. Но еще держим единичку в уме!

Слияние … похоже на сложение чисел в двоичной записи (в столбик)



	1	1	1	0	0
+	0	1	0	0	1
	1	0	0		

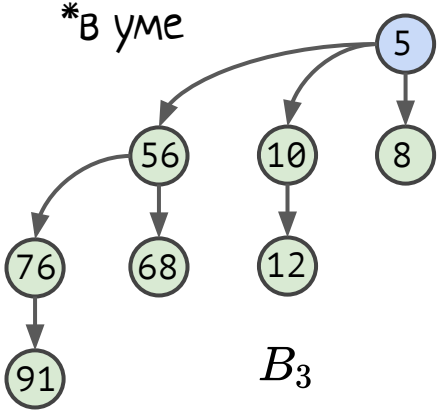
+1

Деревьев порядка 2 у нас точно не будет. Да еще держим единичку в уме!

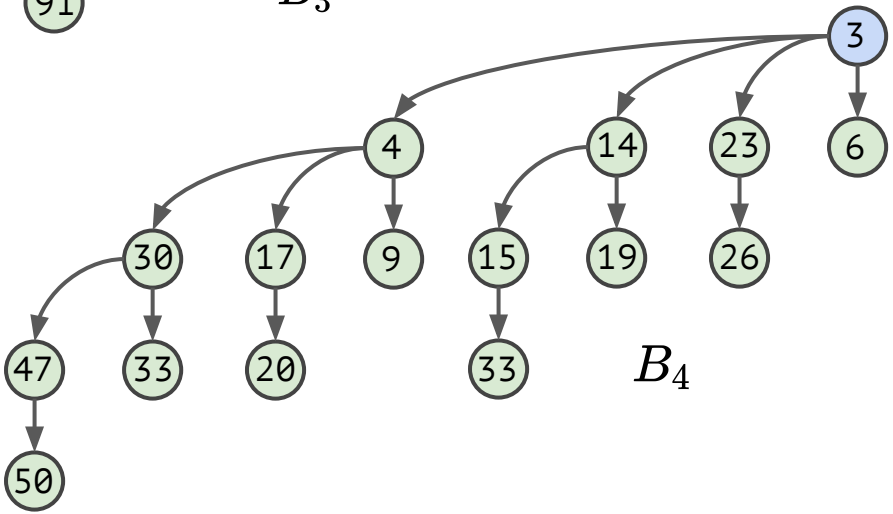
Слияние … похоже на сложение чисел в двоичной записи (в столбик)

B_0

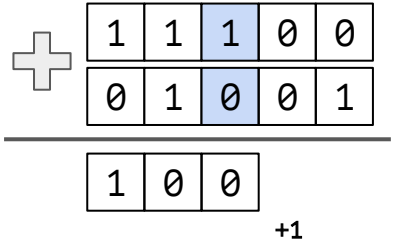
*в уме



B_3



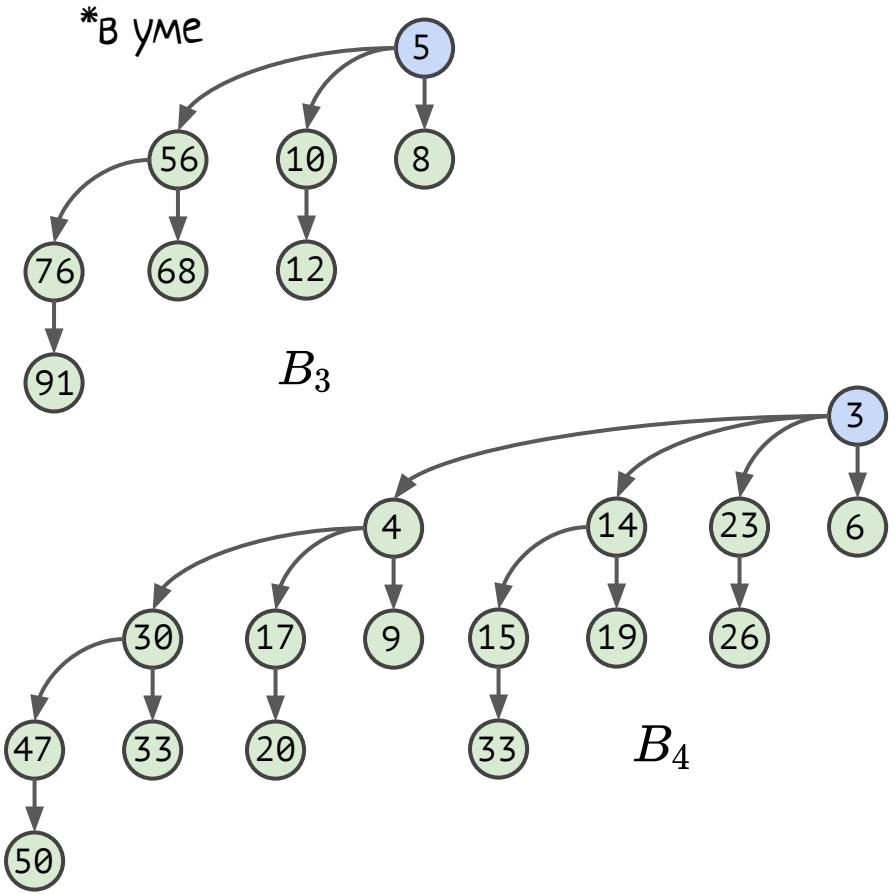
B_4

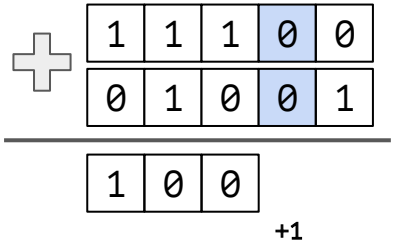


Деревьев порядка 2 у нас точно не будет. Да еще держим единичку в уме!

Слияние … похоже на сложение чисел в двоичной записи (в столбик)

B_0

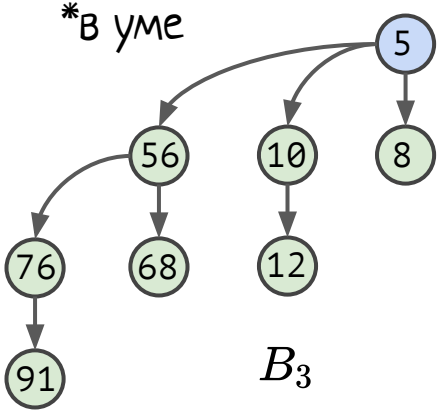




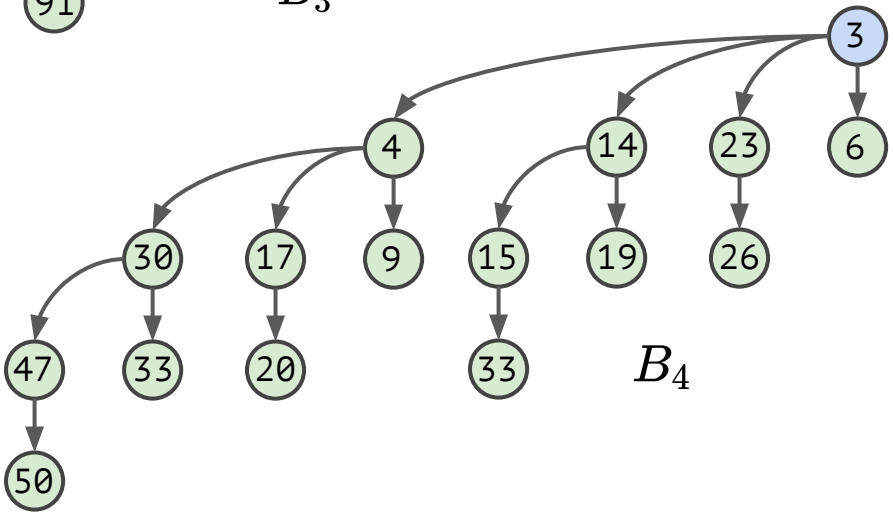
Слияние … похоже на сложение чисел в двоичной записи (в столбик)

B_0

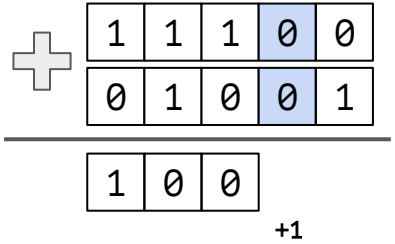
*в уме



B_3

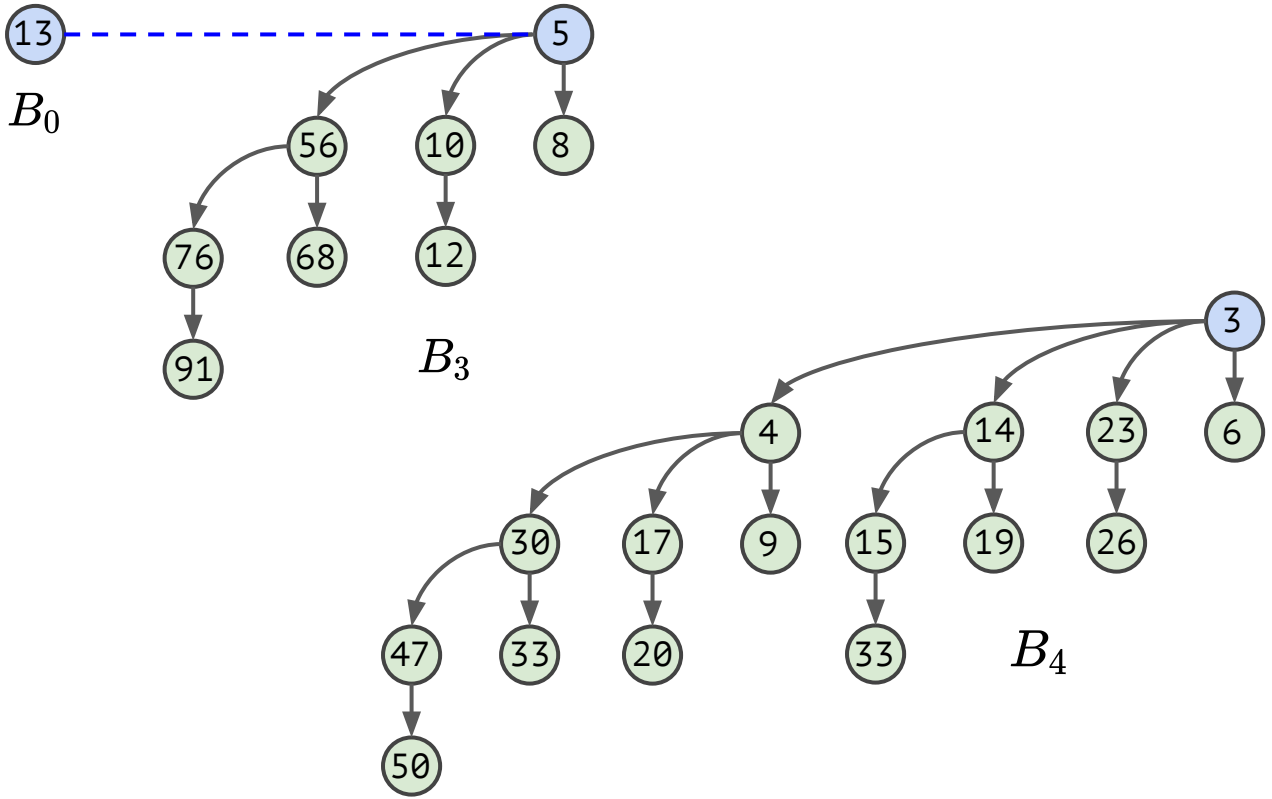


B_4



Вспоминаем, что у нас один в уме, добавляем его в ответ

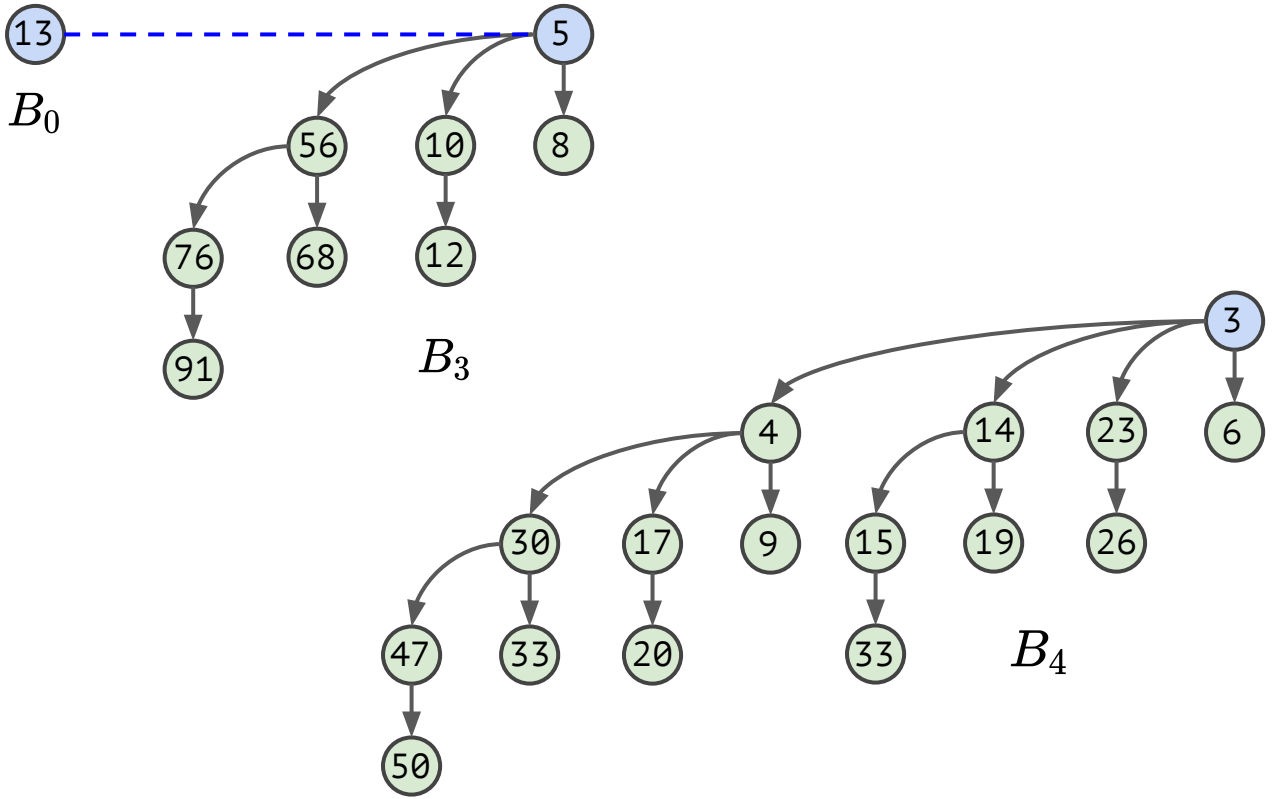
Слияние … похоже на сложение чисел в двоичной записи (в столбик)



+	1	1	1	0	0
	0	1	0	0	1
	1	0	0	1	

Вспоминаем, что у нас один в уме, добавляем его в ответ

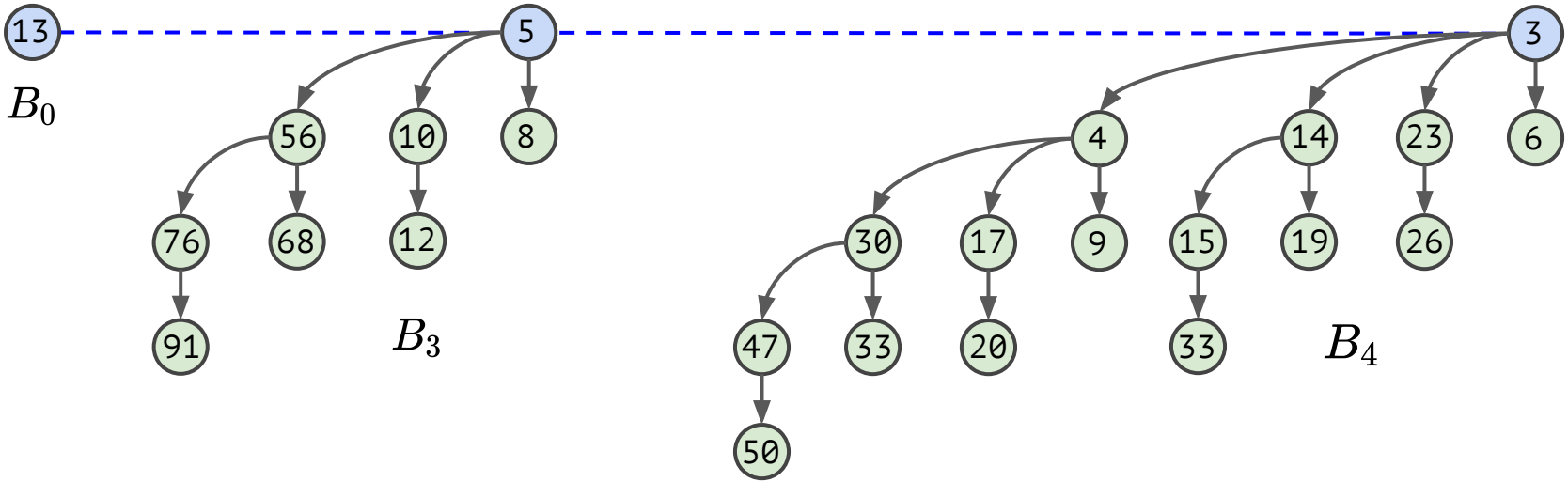
Слияние … похоже на сложение чисел в двоичной записи (в столбик)



+	1	1	1	0	0
	0	1	0	0	1
	1	0	0	1	1

Последнюю единичку добавляем в ответ

Слияние … похоже на сложение чисел в двоичной записи (в столбик)



Биномиальная пирамида: merge

Наблюдения:

1. Проблема merge в том, что пирамидах могут быть биномиальные деревья одинакового порядка.

2. Слияние двух пирамид очень похоже на сложение двух чисел в двоичной записи (в столбик).

3. Осталось написать алгоритм!

$$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$$

$$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$$

$$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$$

$$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$

$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

И пусть:

carry = корень дерева, которое держим в уме или None, если его нет.

$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$

$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

И пусть:

carry = корень дерева, которое держим в уме или None, если его нет.

r[j] - корень биномиального дерева порядка j в получившейся пирамиде или None, если его нет.

$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$

$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

И пусть:

carry = корень дерева, которое держим в уме или None, если его нет.

r[j] - корень биномиального дерева порядка j в получившейся пирамиде или None, если его нет.

```
carry = None

for i in [0, logN + 1]:
```

$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$

$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

И пусть:

carry = корень дерева, которое держим в уме или None, если его нет.

r[j] - корень биномиального дерева порядка j в получившейся пирамиде или None, если его нет.

```
carry = None

for i in [0, logN + 1]:
  if t1[i] and t2[i] and carry:
```

$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$

$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

И пусть:

carry = корень дерева, которое держим в уме или None, если его нет.

r[j] - корень биномиального дерева порядка j в получившейся пирамиде или None, если его нет.

```
carry = None

for i in [0, logN + 1]:
  if t1[i] and t2[i] and carry:
    r[i] = carry
    carry = merge_tree(t1[i], t2[i])
```

$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$

$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

И пусть:

carry = корень дерева, которое держим в уме или None, если его нет.

r[j] - корень биномиального дерева порядка j в получившейся пирамиде или None, если его нет.

```
carry = None

for i in [0, logN + 1]:
  if t1[i] and t2[i] and carry:
    r[i] = carry
    carry = merge_tree(t1[i], t2[i])
  elif t1[i] and t2[i]:
    carry = merge_tree(t1[i], t2[i])
    r[i] = None
```

$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$

$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

И пусть:

carry = корень дерева, которое держим в уме или None, если его нет.

r[j] - корень биномиального дерева порядка j в получившейся пирамиде или None, если его нет.

```
carry = None

for i in [0, logN + 1]:
  if t1[i] and t2[i] and carry:
    r[i] = carry
    carry = merge_tree(t1[i], t2[i])
  elif t1[i] and t2[i]:
    carry = merge_tree(t1[i], t2[i])
    r[i] = None
  elif t1[i] and carry:
    carry = merge_tree(t1[i], carry)
    r[i] = None
  elif carry and t2[i]:
    carry = merge_tree(carry, t2[i])
    r[i] = None
```

$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$

$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

И пусть:

carry = корень дерева, которое держим в уме или None, если его нет.

r[j] - корень биномиального дерева порядка j в получившейся пирамиде или None, если его нет.

```
carry = None

for i in [0, logN + 1]:
  if t1[i] and t2[i] and carry:
    r[i] = carry
    carry = merge_tree(t1[i], t2[i])
  elif t1[i] and t2[i]:
    carry = merge_tree(t1[i], t2[i])
    r[i] = None
  ...
  elif t1[i]:
    r[i] = t1[i]
```

$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$

$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

И пусть:

carry = корень дерева, которое держим в уме или None, если его нет.

r[j] - корень биномиального дерева порядка j в получившейся пирамиде или None, если его нет.

```
carry = None

for i in [0, logN + 1]:
  if t1[i] and t2[i] and carry:
    r[i] = carry
    carry = merge_tree(t1[i], t2[i])
  elif t1[i] and t2[i]:
    carry = merge_tree(t1[i], t2[i])
    r[i] = None
  ...
  elif t1[i]:
    r[i] = t1[i]
  elif t2[i]:
    r[i] = t2[i]
  else:
    r[i] = carry
    carry = None
```

$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$

$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

И пусть:

carry = корень дерева, которое держим в уме или None, если его нет.

r[j] - корень биномиального дерева порядка j в получившейся пирамиде или None, если его нет.

Аккуратнее с последним разрядом

```
carry = None

for i in [0, logN + 1]:
  if t1[i] and t2[i] and carry:
    r[i] = carry
    carry = merge_tree(t1[i], t2[i])
  elif t1[i] and t2[i]:
    carry = merge_tree(t1[i], t2[i])
    r[i] = None
  ...
  elif t1[i]:
    r[i] = t1[i]
  elif t2[i]:
    r[i] = t2[i]
  else:
    r[i] = carry
    carry = None
```

$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$

$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

И пусть:

carry = корень дерева, которое держим в уме или None, если его нет.

r[j] - корень биномиального дерева порядка j в получившейся пирамиде или None, если его нет.

```
carry = None

for i in [0, logN + 1]:
  if t1[i] and t2[i] and carry:
    r[i] = carry
    carry = merge_tree(t1[i], t2[i])
  elif t1[i] and t2[i]:
    carry = merge_tree(t1[i], t2[i])
    r[i] = None
  ...
  elif t1[i]:
    r[i] = t1[i]
  elif t2[i]:
    r[i] = t2[i]
  else:
    r[i] = carry
    carry = None
```

Аккуратнее с последним разрядом

Сложность?

$H_1 = t_1[0], t_1[1], \ldots, t_1[logN]$

$H_2 = t_2[0], t_2[1], \ldots, t_2[logN]$

Пусть t_i[j] - корень биномиального дерева порядка j из пирамиды i, или None, если дерева такого порядка не было

И пусть:

carry = корень дерева, которое держим в уме или None, если его нет.

r[j] - корень биномиального дерева порядка j в получившейся пирамиде или None, если его нет.

Аккуратнее с последним разрядом

```
carry = None

for i in [0, logN + 1]:
  if t1[i] and t2[i] and carry:
    r[i] = carry
    carry = merge_tree(t1[i], t2[i])
  elif t1[i] and t2[i]:
    carry = merge_tree(t1[i], t2[i])
    r[i] = None
  ...
  elif t1[i]:
    r[i] = t1[i]
  elif t2[i]:
    r[i] = t2[i]
  else:
    r[i] = carry
    carry = None
```

Сложность?
O(logN)

Биномиальная пирамида

Множество значений: пары <priority: int, value: T>

Операции:

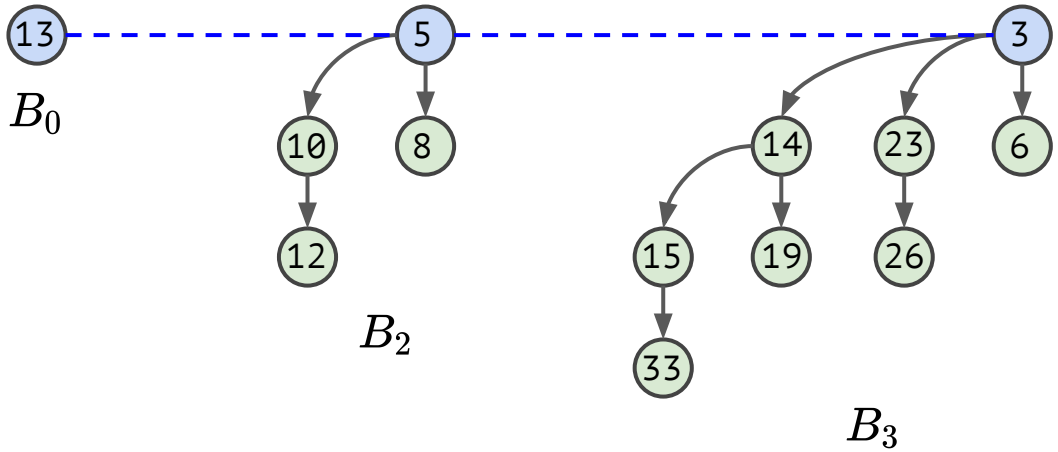
1. insert(value) -> O(???)
2. peek_min() -> O(logN)
3. extract_min() -> O(???)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(logN)
6. delete(s) -> O(???)

Как реализовать?

~~Бинарная куча!~~

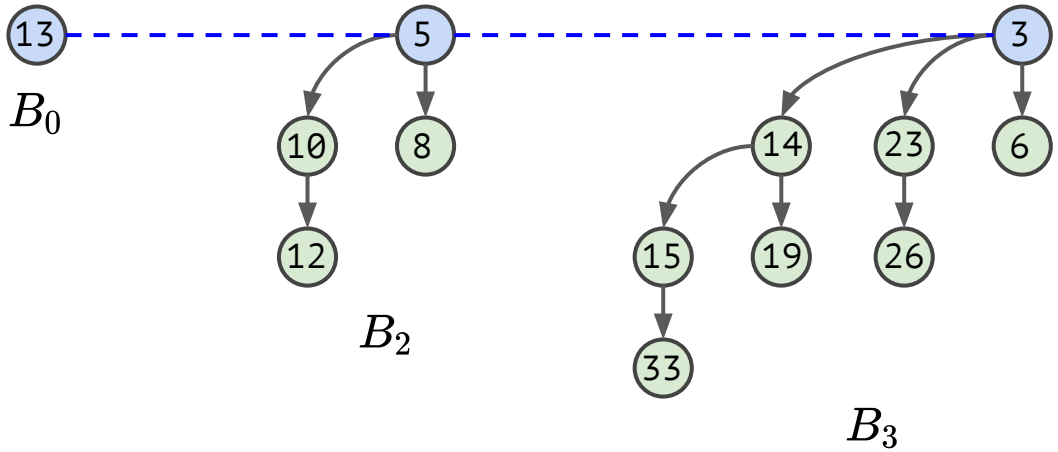
Биномиальная!

Биномиальная пирамида: insert



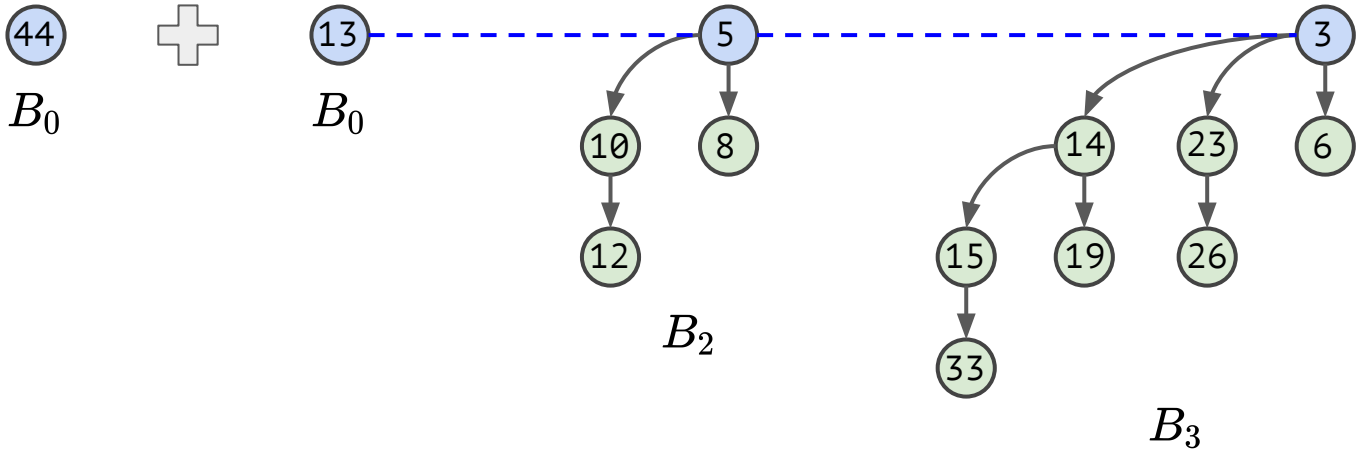
Как вставить новый элемент?

Биномиальная пирамида: insert



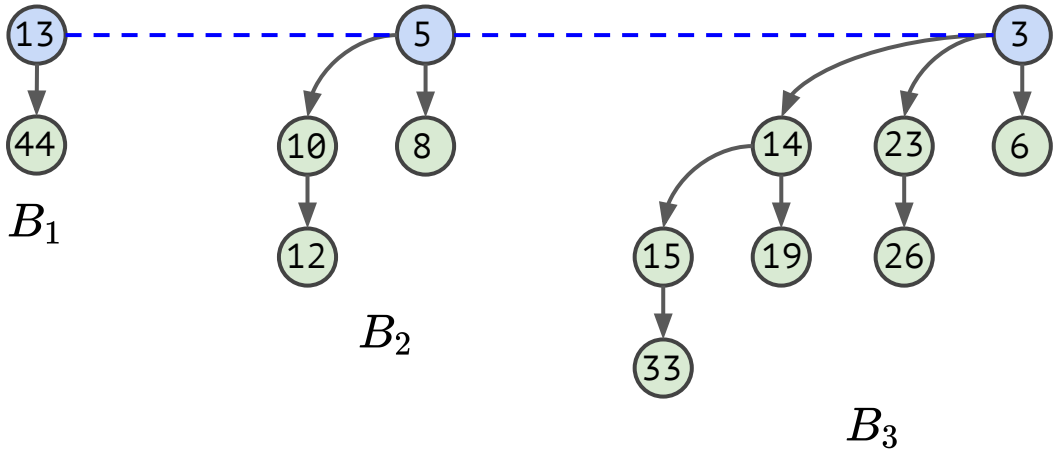
Как вставить новый элемент?
Так это же просто merge с пирамидой из одного элемента!

Биномиальная пирамида: insert



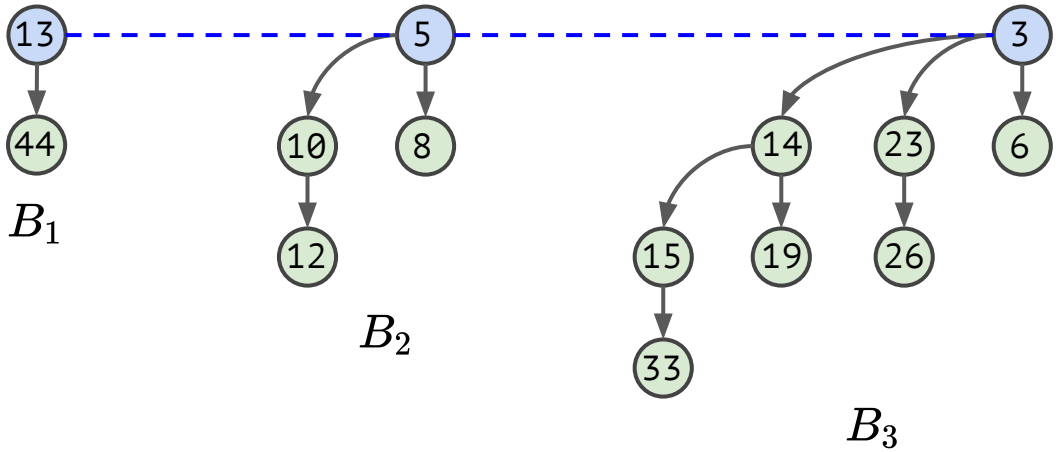
Как вставить новый элемент?
Так это же просто merge с пирамидой из одного элемента!

Биномиальная пирамида: insert



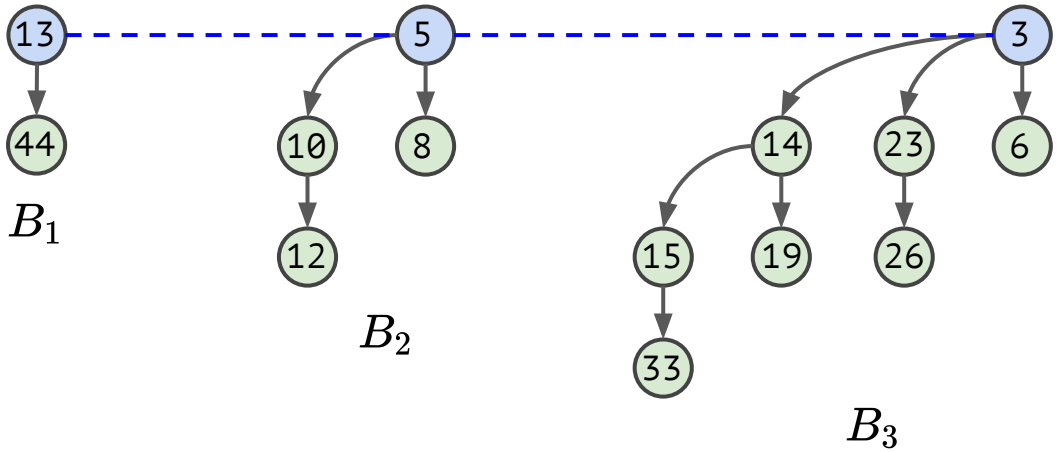
Как вставить новый элемент?
Так это же просто merge с пирамидой из одного элемента!

Биномиальная пирамида: insert



Как вставить новый элемент?
Так это же просто merge с пирамидой из одного элемента!
Сложность?

Биномиальная пирамида: insert



Как вставить новый элемент?
Так это же просто merge с пирамидой из одного элемента!
Сложность? O(logN)

Биномиальная пирамида

Множество значений: пары <priority: int, value: T>

Операции:

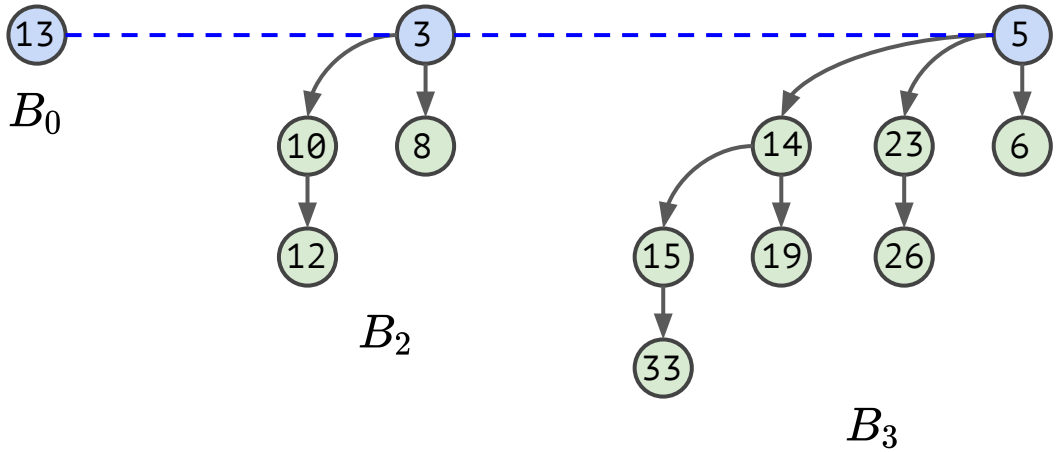
1. insert(value) -> O(logN)
2. peek_min() -> O(logN)
3. extract_min() -> O(???)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(logN)
6. delete(s) -> O(???)

Как реализовать?

~~Бинарная куча!~~

Биномиальная!

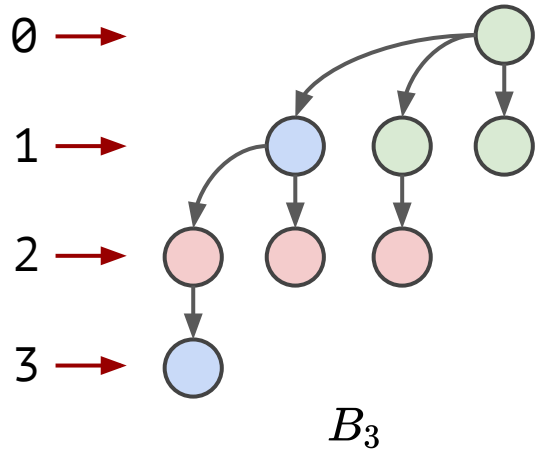
Биномиальная пирамида: extract_min



Как достать минимум, т.е. в нашем случае 3?

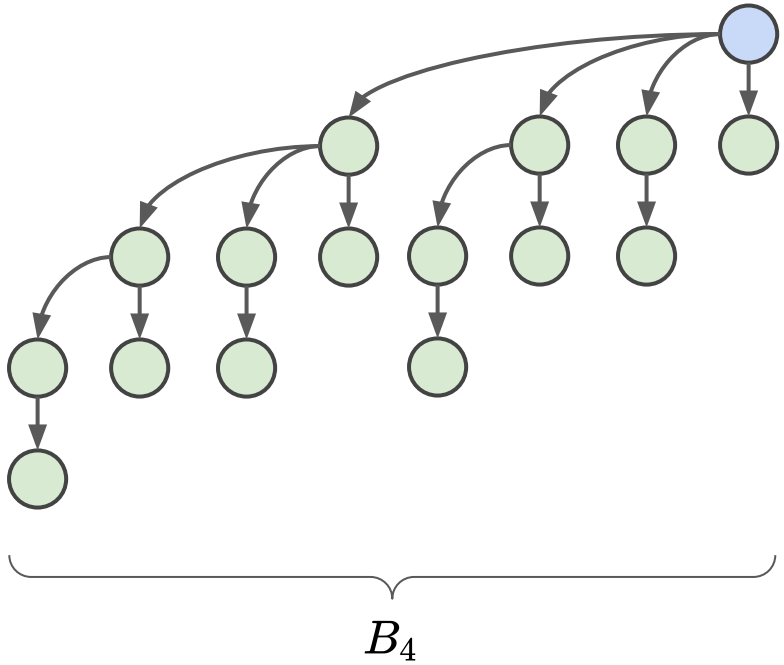
Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i



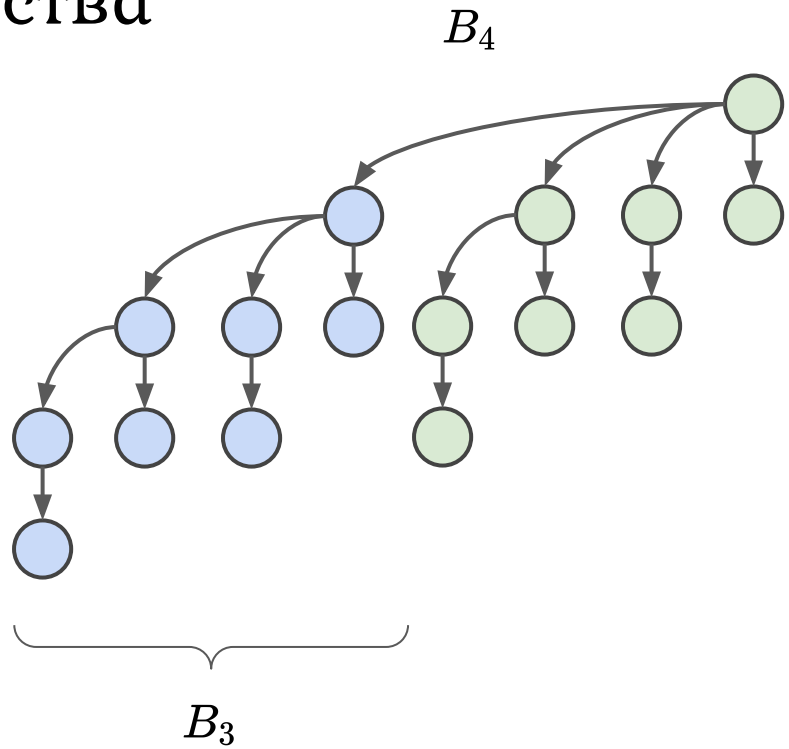
Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i



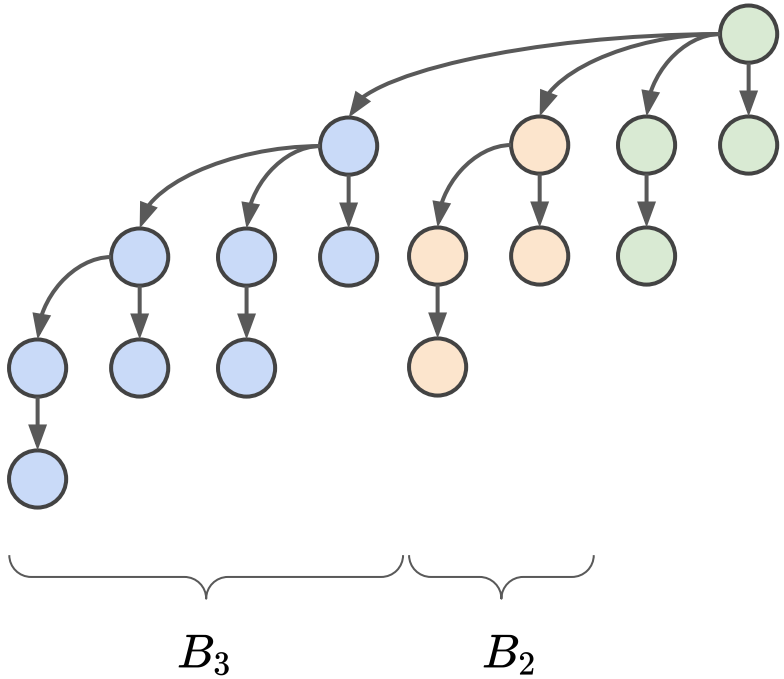
Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i



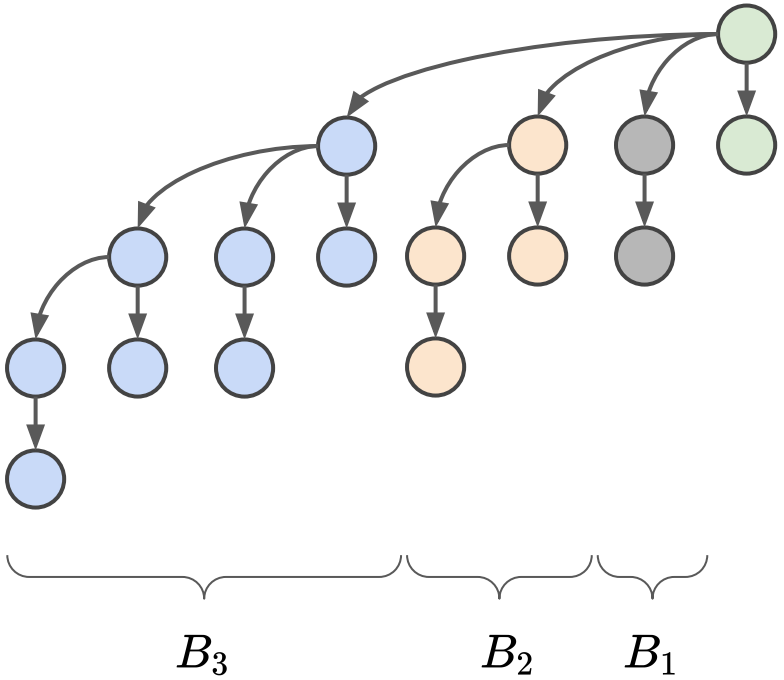
Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i



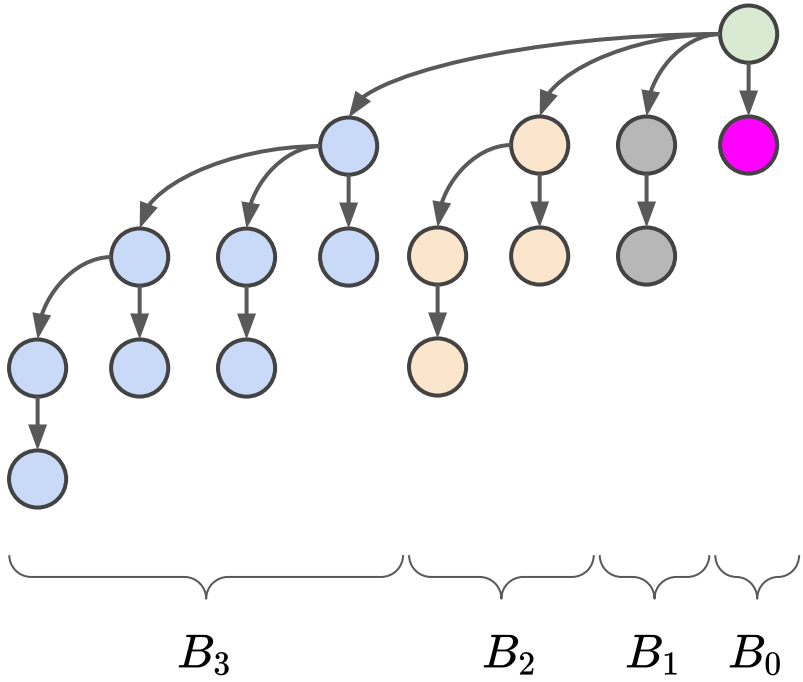
Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i



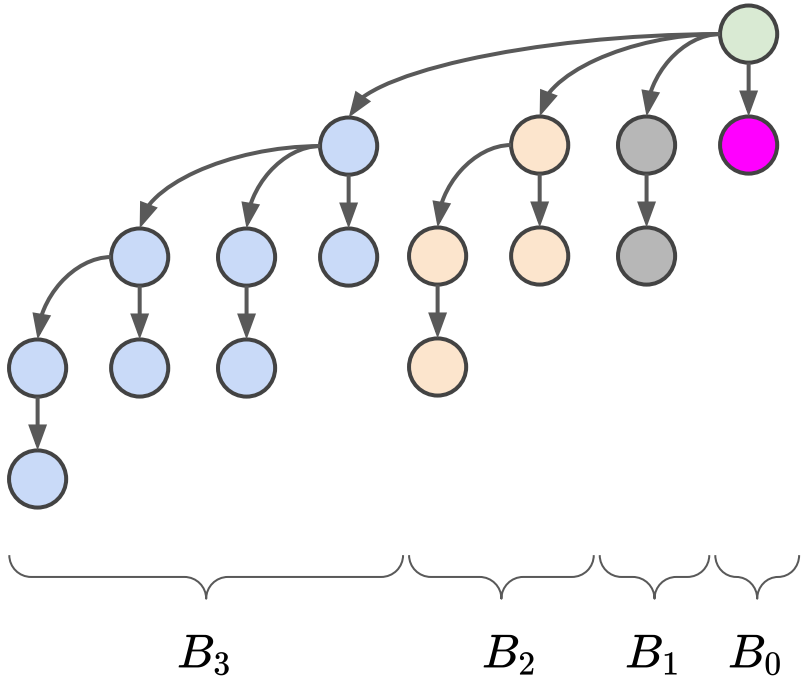
Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i



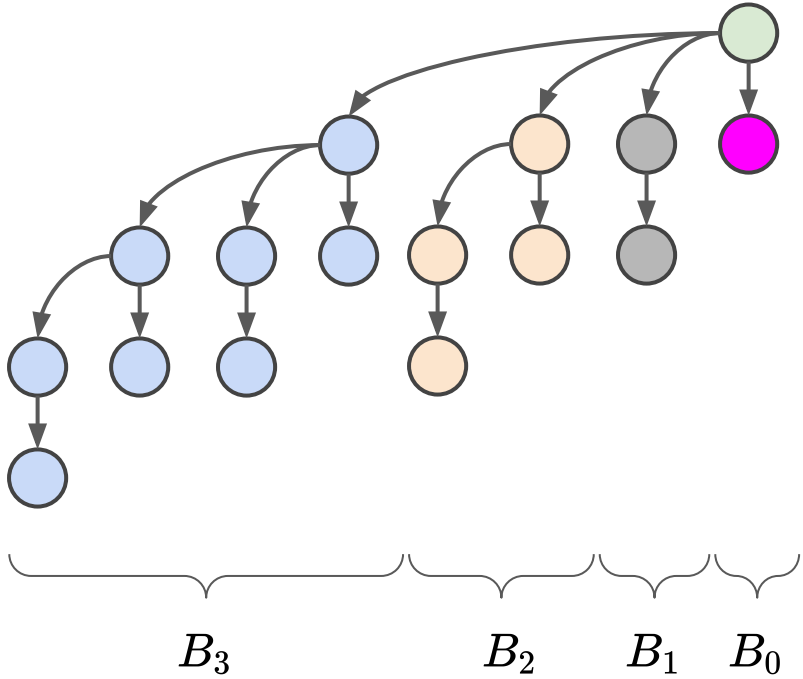
Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i
4. Наследники корня дерева B_k сами являются корнями деревьев $B_0, B_1, \ldots B_{k-1}$



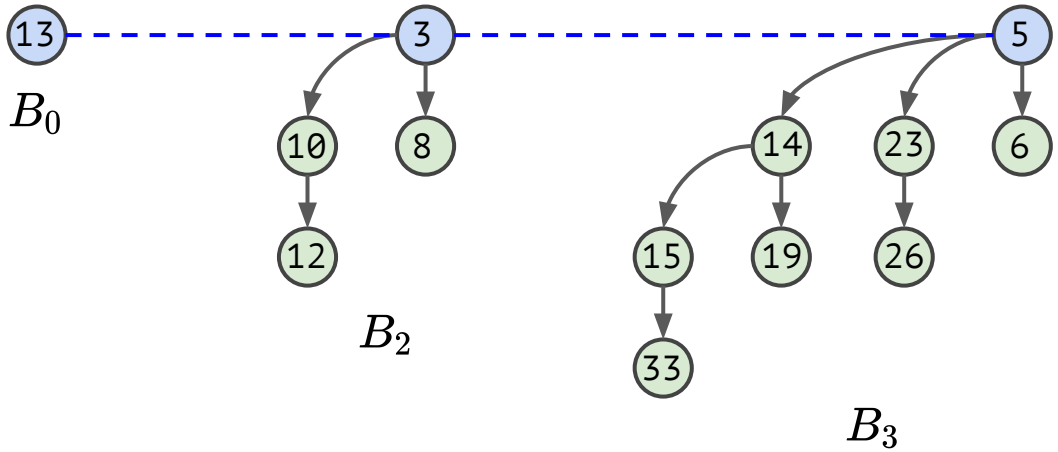
Биномиальные деревья: свойства

1. Сколько вершин в B_k? 2^k
2. Высота у дерева B_k? k
3. Сколько вершин на глубине $i = 0, 1, \ldots, k$? C_k^i
4. Наследники корня дерева B_k сами являются корнями деревьев $B_0, B_1, \ldots B_{k-1}$



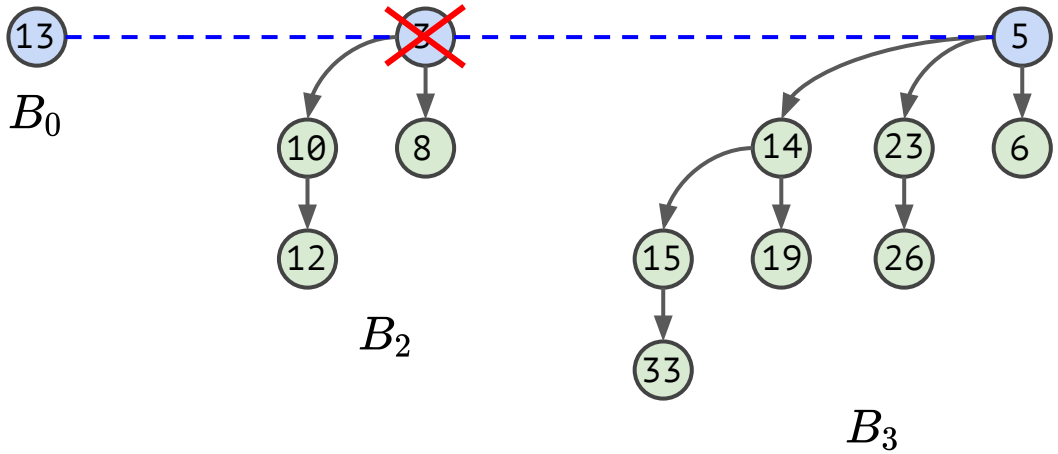
Док-во: по индукции. Для базы верно, если верно для k, то k+1 первый строим как раз добавлением слева сына k-ого порядка.

Биномиальная пирамида: extract_min



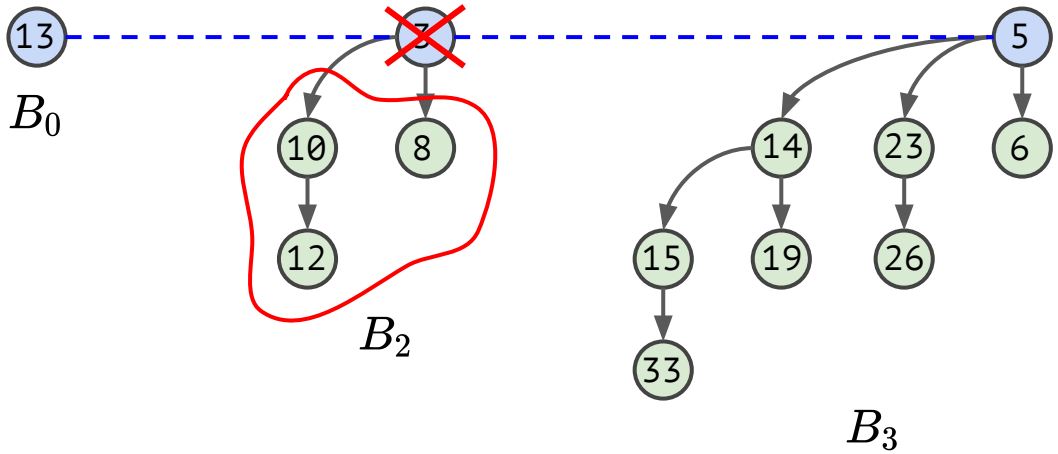
Как достать минимум, т.е. в нашем случае 3?

Биномиальная пирамида: extract_min



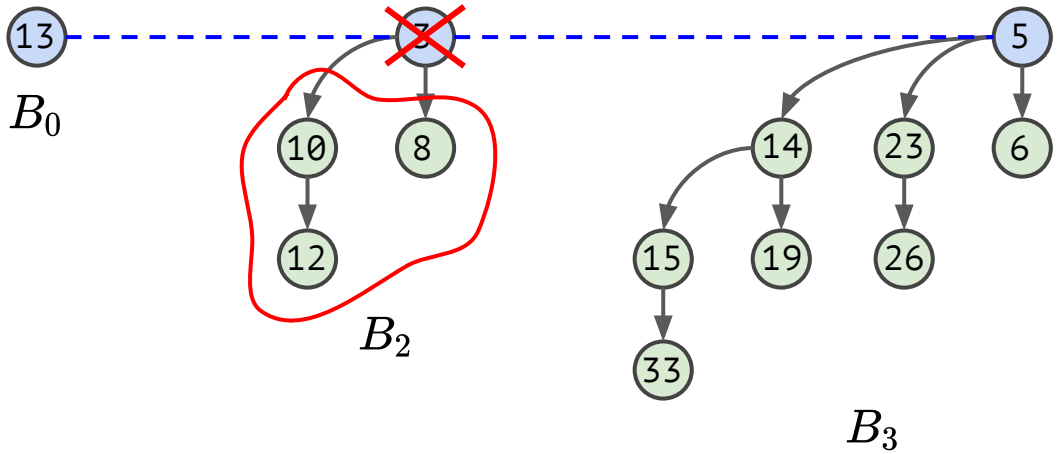
Как достать минимум, т.е. в нашем случае 3?
Убираем минимум (возвращаем овтет),

Биномиальная пирамида: extract_min



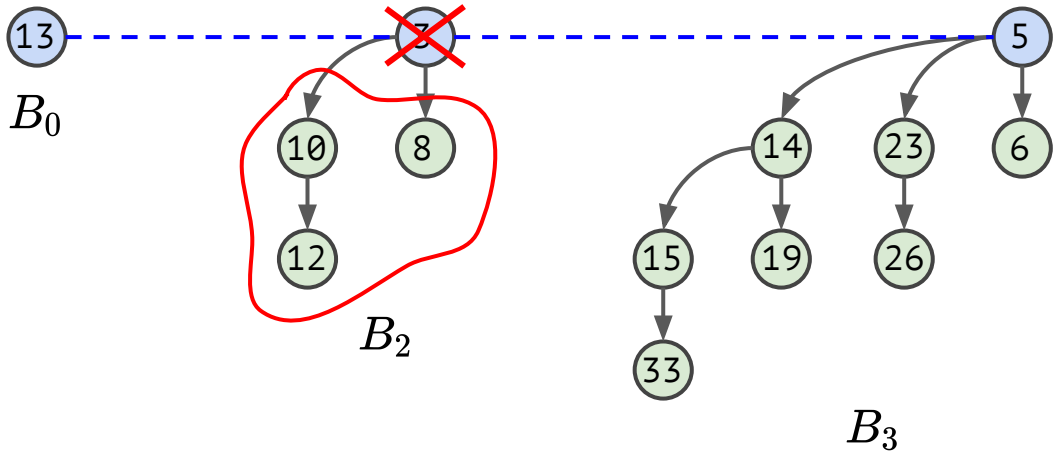
Как достать минимум, т.е. в нашем случае 3?
Убираем минимум (возвращаем овтет), все его дети образуют корректную биномиальную пирамиду!

Биномиальная пирамида: extract_min



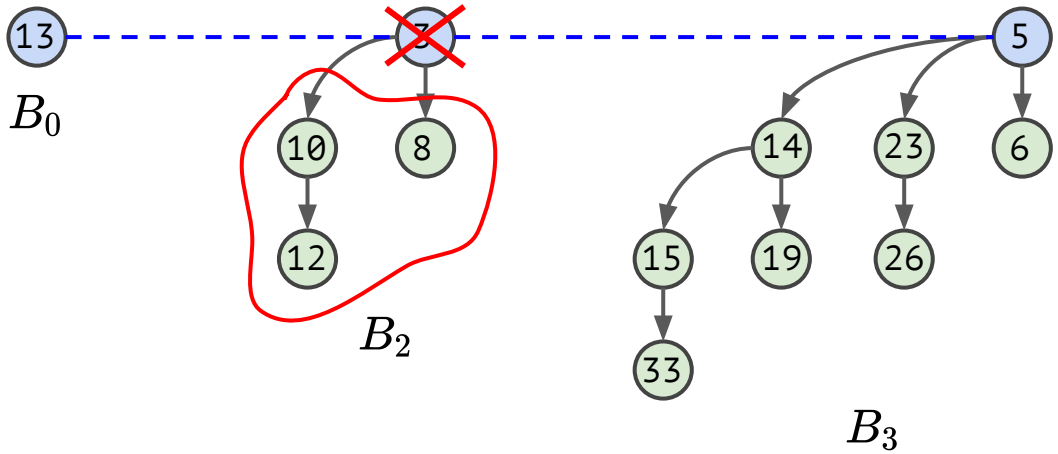
Как достать минимум, т.е. в нашем случае 3?
Убираем минимум (возвращаем овтет), все его дети образуют корректную биномиальную пирамиду! Вызываем для нее merge с остальной пирамидой.

Биномиальная пирамида: extract_min



Как достать минимум, т.е. в нашем случае 3?
Убираем минимум (возвращаем овтет), все его дети образуют корректную биномиальную пирамиду! Вызываем для нее merge с остальной пирамидой.
Сложность?

Биномиальная пирамида: extract_min



Как достать минимум, т.е. в нашем случае 3?
Убираем минимум (возвращаем овтет), все его дети образуют корректную биномиальную пирамиду! Вызываем для нее merge с остальной пирамидой.
Сложность? O(logN)

Биномиальная пирамида

Множество значений: пары <priority: int, value: T>

Операции:

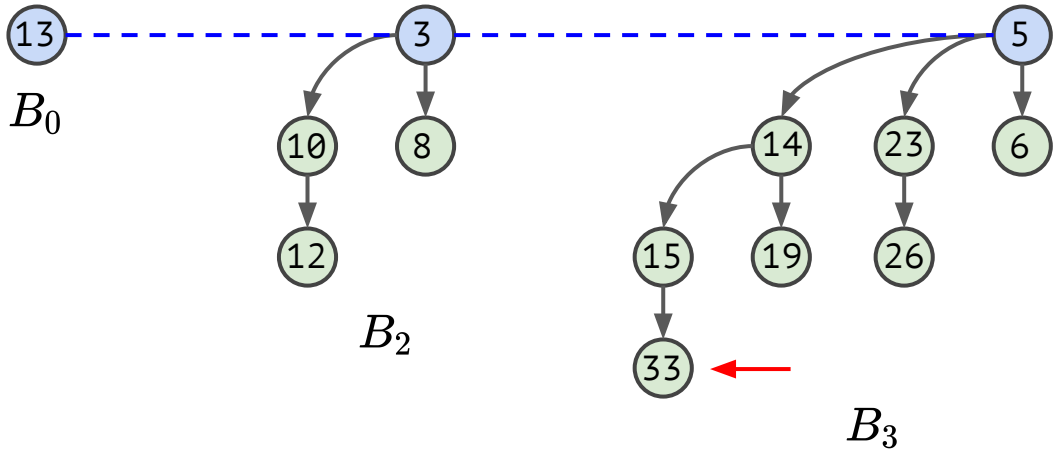
1. insert(value) -> O(logN)
2. peek_min() -> O(logN)
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(logN)
6. delete(s) -> O(???)

Как реализовать?

~~Бинарная куча!~~

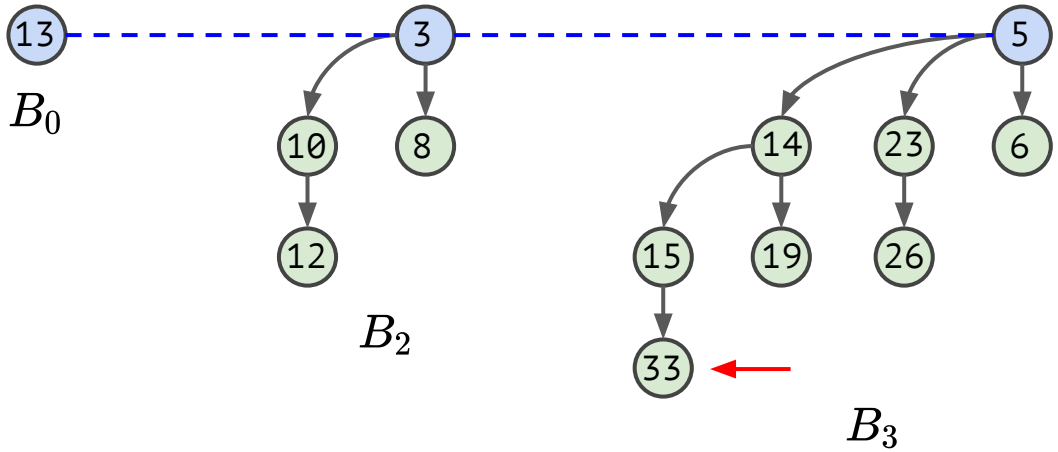
Биномиальная!

Биномиальная пирамида: delete



Как удалить данный элемент, например, тот, где ключ 33?

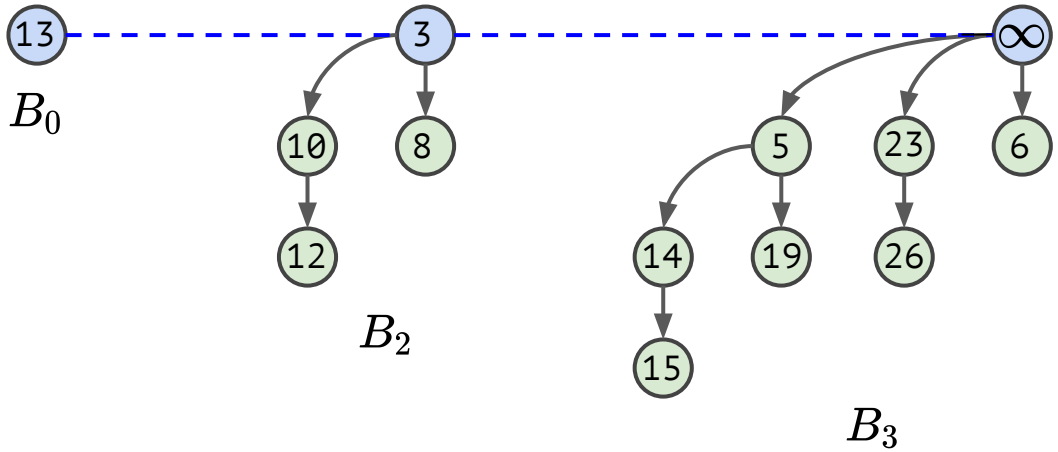
Биномиальная пирамида: delete



Как удалить данный элемент, например, тот, где ключ 33?

1. decrease_key(s, $-\infty$)

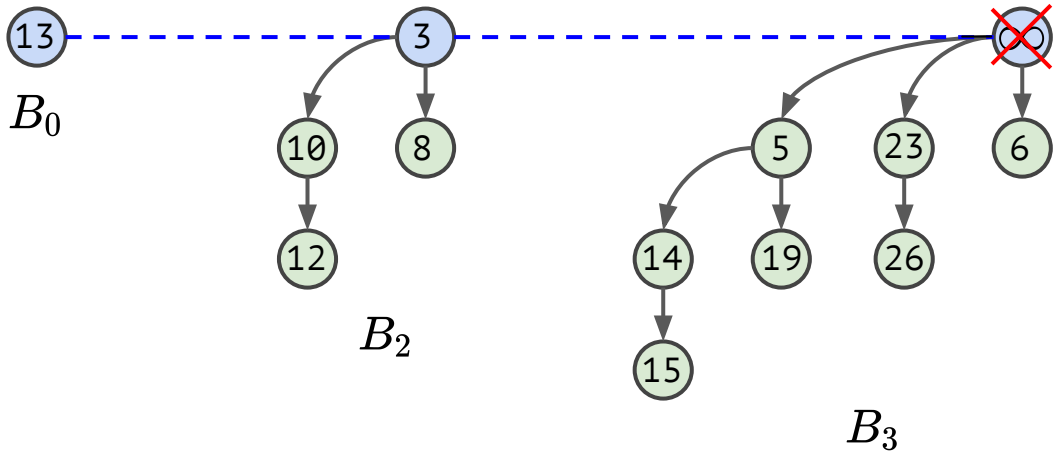
Биномиальная пирамида: delete



Как удалить данный элемент, например, тот, где ключ 33?

1. decrease_key(s, $-\infty$)

Биномиальная пирамида: delete



Как удалить данный элемент, например, тот, где ключ 33?

1. decrease_key(s, $-\infty$)
2. extract_min()

Биномиальная пирамида

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(logN)
2. peek_min() -> O(logN)
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(logN)
6. delete(s) -> O(logN)

Как реализовать?

~~Бинарная куча!~~

Биномиальная!

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(logN)
2. peek_min() -> O(1) 🥰
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(N)
6. delete(s) -> O(logN)

Как реализовать?
Бинарная куча!

O(N) – неприятно, хочется оптимизировать merge

Биномиальная пирамида

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(logN)
2. peek_min() -> O(logN) 😉
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(logN) 🥰
6. delete(s) -> O(logN)

Как реализовать?

~~Бинарная куча!~~

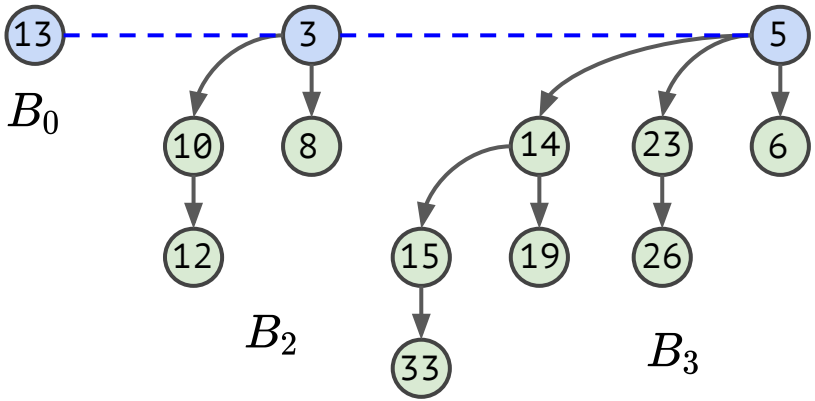
Биномиальная!

Биномиальная пирамида

Пара слов про реализацию:

Биномиальная пирамида

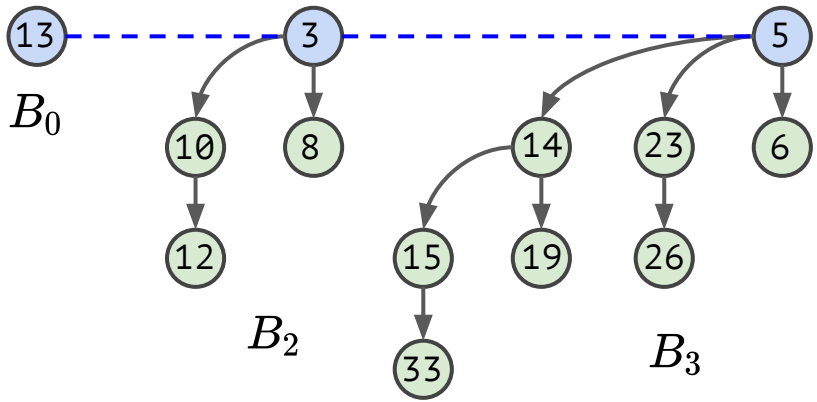
Пара слов про реализацию:



Биномиальная пирамида

Пара слов про реализацию:

1. Корни обычно провязывают в связный список

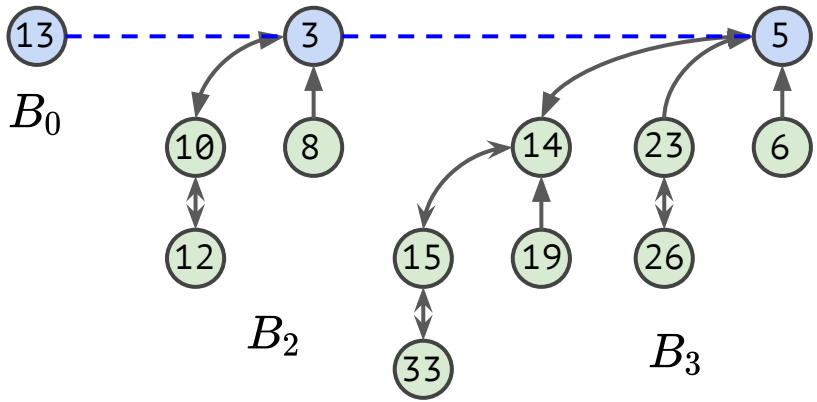


Биномиальная пирамида

Пара слов про реализацию:

1. Корни обычно провязывают в связный список

2. Каждому узлу стоит знать своего предка

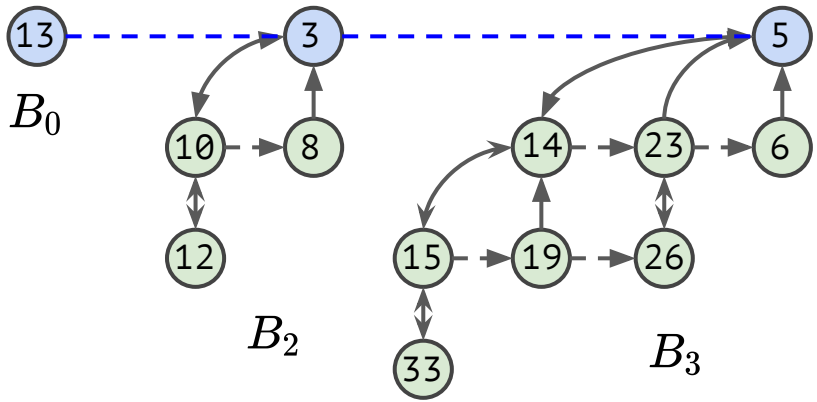


Биномиальная пирамида

Пара слов про реализацию:

1. Корни обычно провязывают в связный список

2. Каждому узлу стоит знать своего предка и соседа (справа).

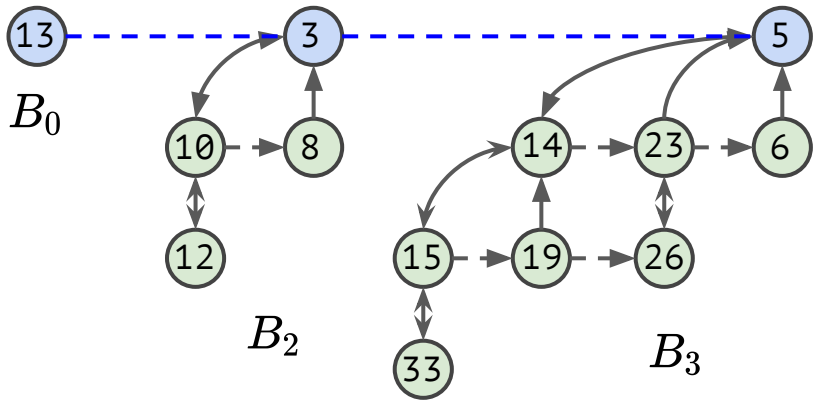


Биномиальная пирамида

Пара слов про реализацию:

1. Корни обычно провязывают в связный список

2. Каждому узлу стоит знать своего предка и соседа (справа). Всех своих детей при этом знать не нужно.



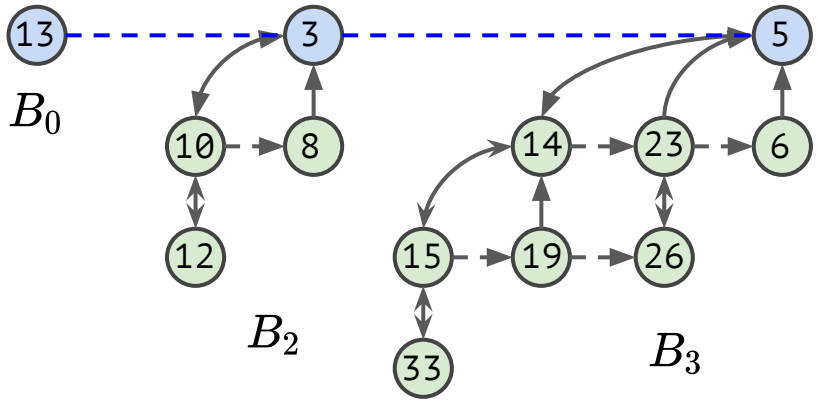
Биномиальная пирамида

Пара слов про реализацию:

1. Корни обычно провязывают в связный список

2. Каждому узлу стоит знать своего предка и соседа (справа). Всех своих детей при этом знать не нужно.

3. Еще полезно (для merge) в каждом узле хранить его степень.



Мини(?)-задача #21 (2 балла)

Реализовать биномиальную кучу со всеми рассмотренными операциями.

Подготовить набор тестов, демонстрирующих корректность решения.

Алгоритм Дейкстры поиска кратчайших путей

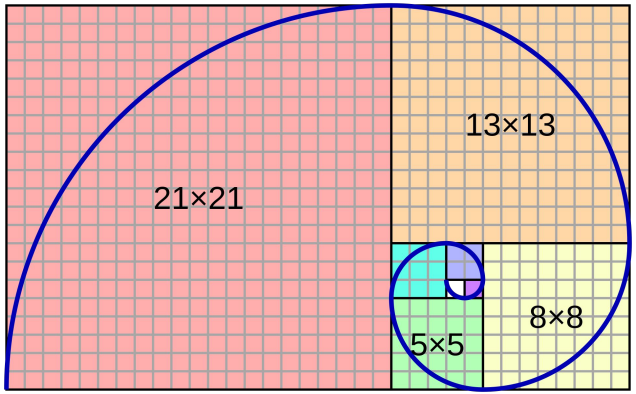
Оценка сложности аккуратной реализации ч/р бинарные пирамиды:

$$O(|V| * log(|V|) + |E| * log(|V|))$$

Но можем ли мы еще лучше?

Да! Но начнем чуть издалека.

Фибоначчиева пирамида



Takeaways

- Алгоритм Дейкстры можно и нужно оптимизировать через пирамиды.

Takeaways

- Алгоритм Дейкстры можно и нужно оптимизировать через пирамиды.

- Кроме бинарных существуют и другие пирамиды, например, Фибоначчиева и Биномиальная.

Takeaways

- Алгоритм Дейкстры можно и нужно оптимизировать через пирамиды.
- Кроме бинарных существуют и другие пирамиды, например, Фибоначчиева и Биномиальная.
- Пока познакомились с биномиальной, она хороша своей предсказуемостью и хорошей сложностью merge.