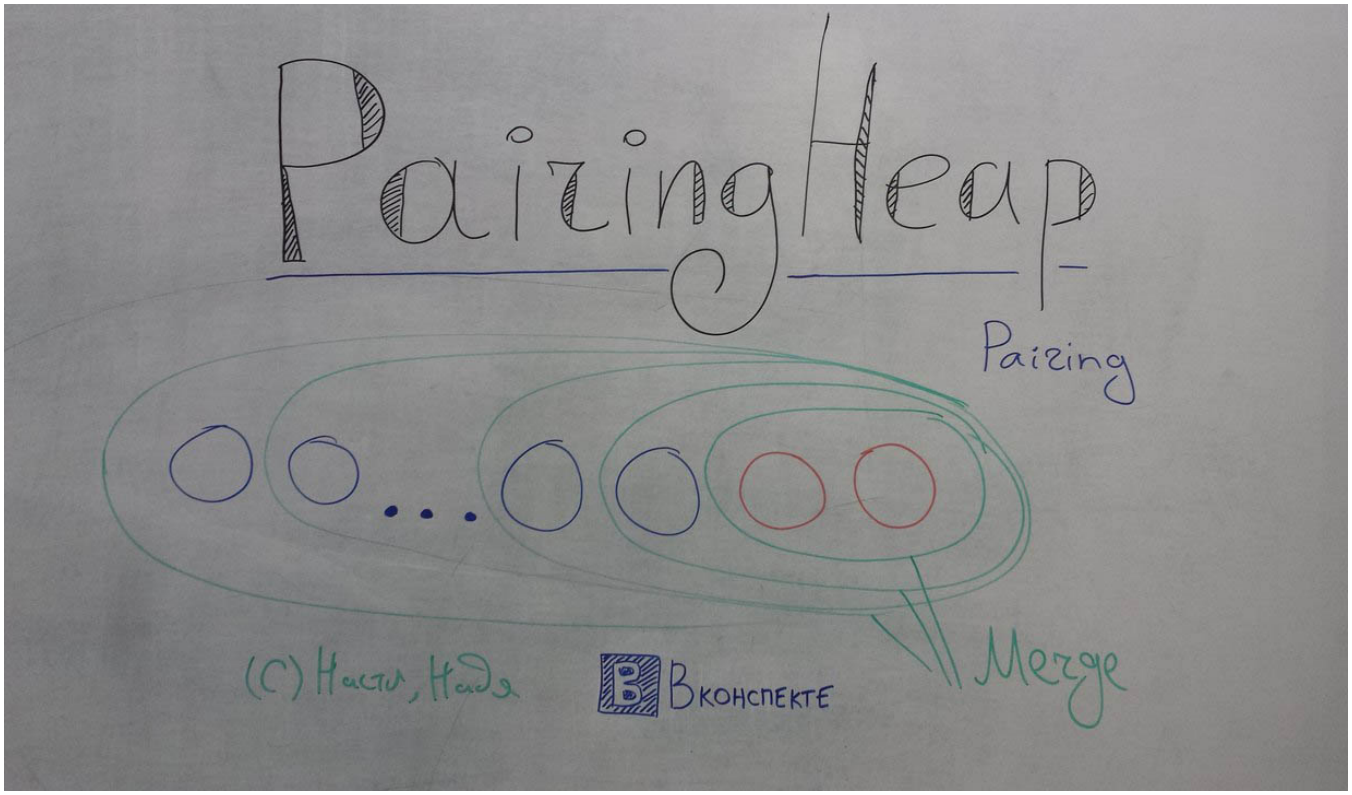
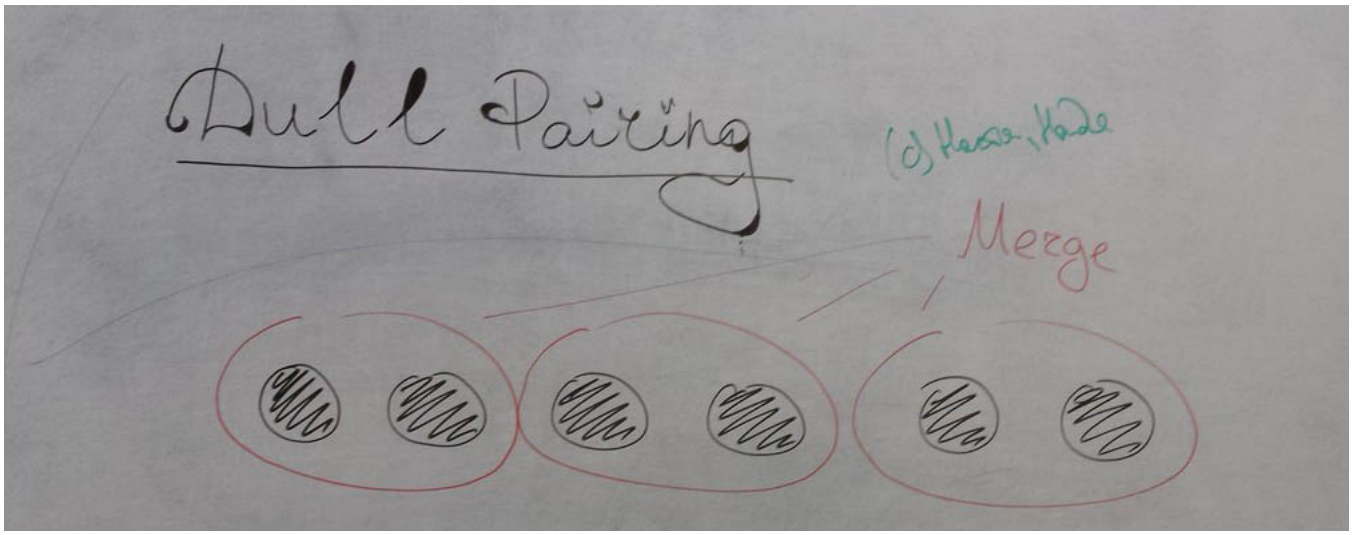
PairingHeap - амортизированное время операции DelMin за $\mathcal{O}(\sqrt{n})$

Рассмотрим немного неверный Pairing (доказательство амортизированного времени работы DelMin (с упрощением до DullPairing) всё равно катит).

Функция Pairing из n корней деревьев, хранящихся в списке, делает 1 корень, объединяя элементы. Сначала объединяет два последних элемента. А результат каждого объединения объединяет (Merge) со следующим элементом списка. При Pairing амортизированное время работы DelMin равно $\mathcal{O}(\log n)$.



Рассмотрим более простую процедуру: DullPairing. Докажем, что при применении DullPairing амортизированное время работы DelMin $\mathcal{O}(\sqrt{n})$. Она разбивает n элементов на произвольные пары, объединяя их (Merge) и получая $\lceil n/2 \rceil$ элементов. Это мы будем использовать в доказательстве того, что амортизированное время работы алгоритма DelMin на рассматриваемой куче равно $\sqrt{n}$.



Замечание: если необходимо разбить нечётное количество элементов, то последний останется без пары.

Пример: (0 1) (2 3) (4 5) 6
Для доказательства времени работы с помощью метода потенциалов введем следующие обозначения:

- t_i - реальное время работы операции.
- a_i - амортизированное время работы операции.
- φ - потенциал.

Тогда $a_i = t_i + \Delta\varphi$, где $\Delta\varphi = \varphi_{i+1} - \varphi_i$. Заметим, что в таком случае верна следующая формула: $\sum \frac{a_i}{m} = \sum \frac{t_i}{m} + \frac{\varphi_m - \varphi_0}{m}$, где m - количество операций. В этой формуле $\sum \frac{t_i}{m}$ - среднее время работы. Т.к. φ_m и φ_0 ограничены сверху константой, то если мы докажем, что $a_i \leq \sqrt{n}$, можно дать оценку $\sum \frac{t_i}{m}$, которая тоже будет $\mathcal{O}(\sqrt{n})$

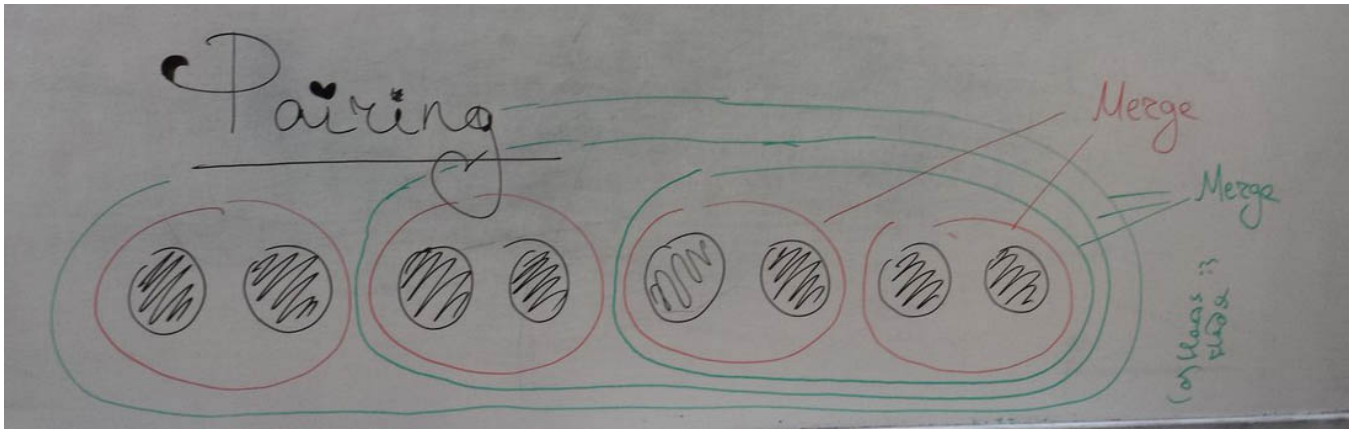
Теперь рассмотрим алгоритм DelMin. Он включает в себя вызовы Del, работающего за $\mathcal{O}(1)$, Merge - $\mathcal{O}(1)$ и Pairing - $\mathcal{O}(\#Roots)$.

Рассмотрим следующую функцию для φ: $\varphi = 2(\#Roots) + 2\sum max(deg, \sqrt{n})$, где deg - степень вершины очередного элемента в списке. Найдём чему равно $\Delta\varphi_i$ - изменение потенциала при удалении этого элемента.

	Δ(слагаемое)	
название функции	2(#Roots)	$2\sum \max(deg, \sqrt{n})$
Del	-2 Удаляем одну вершину (2 — это коэффициент, "−" т.к. вершин становится меньше)	$-2deg$ Из суммы всех степеней уходит степень вершины, которую удаляем
Merge	$+2deg$ Каждого из детей удаленной вершины заносим в список (их deg) (2 - коэффициент, "+" т.к. вершин становится больше)	0 Ничего не меняется, просто добавляем вершины в список
Pairing	$-roots - deg$ $-roots - deg = \frac{-2\cdot(roots+deg)}{2}$: было $roots$ корней, после merge стало $roots + deg$, и мы уменьшаем их количество вдвое	$2\sqrt{n}$ Теперь степень некоторых вершин увеличилась на 1. А количество вершин, для которых что-то изменилось не больше, чем $\sqrt{n}$, т.к. это могут быть только вершины, степень которых была не меньше $\sqrt{n}$(всего вершин n).

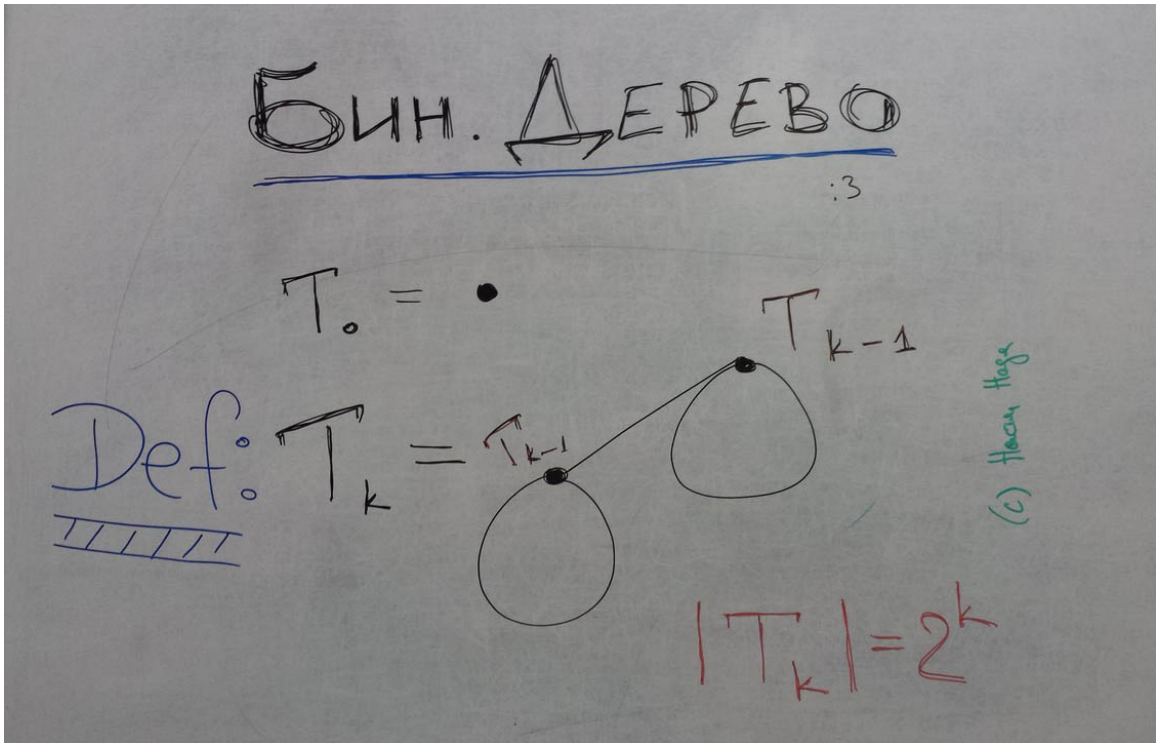
Получаем следующую оценку для φ: $0 \leq \varphi \leq cn\sqrt{n}$, где c - некая константа.
Результат: $\sum t_i = \sum a_i + \varphi_0 - \varphi_m$, $\sum a_i \approx m\sqrt{n}$, а $\varphi_0 - \varphi_m \approx n\sqrt{n}$. Таким образом, среднее время работы равно $\mathcal{O}(\sqrt{n})$, если $n \geq m$ (если $n < m$, то этого тоже добиваются, но немного другим потенциалом, на лекции рассказано не было).

Правильный Pairing будет в следующем конспете, а пока - вот вам картиночка!



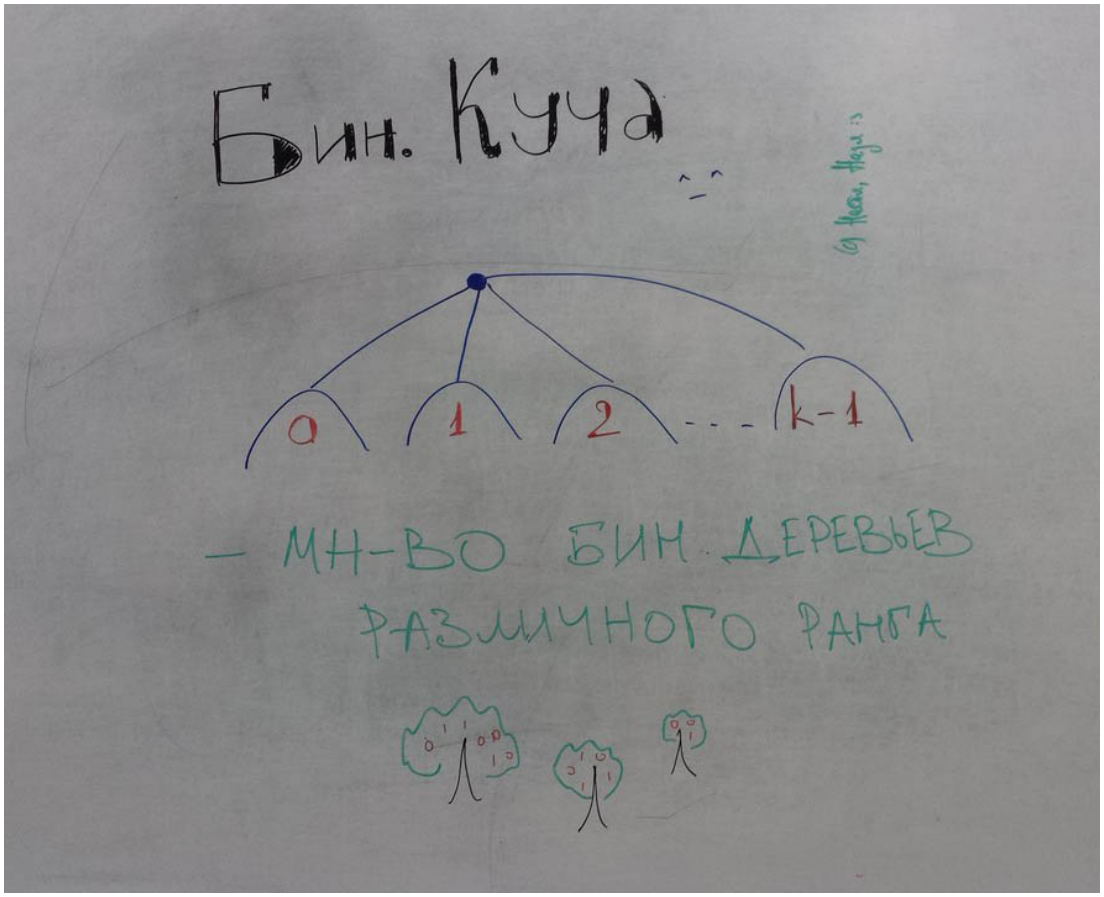
Биномиальные кучи

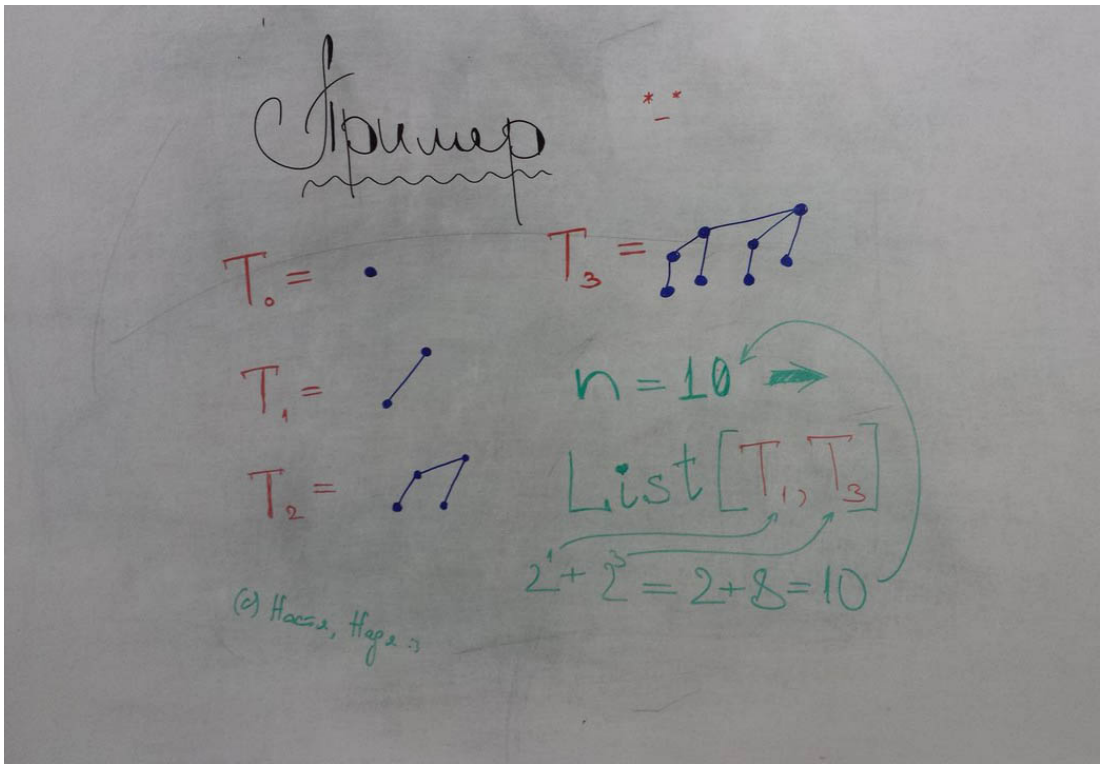
Биномиальное дерево - это дерево, задающееся следующим рекурентным способом.



За корень биномиального дерева ранга k (дерево T_k) подвешено соответственно k биномиальных деревьев с рангами от 0 до $k-1$.

Биномиальная куча - множество из n элементов, разбитое на множества, с мощностями, равными степени двойки, для каждого множества строится биномиальное дерево со свойством кучи. Т.е. куча $= \mathcal{O}(\log n)$ деревьев, или другими словами: биномиальная куча - множество биномиальных деревьев различного ранга. $n = \sum 2^{k_i}$; $|T_k| = 2^k$

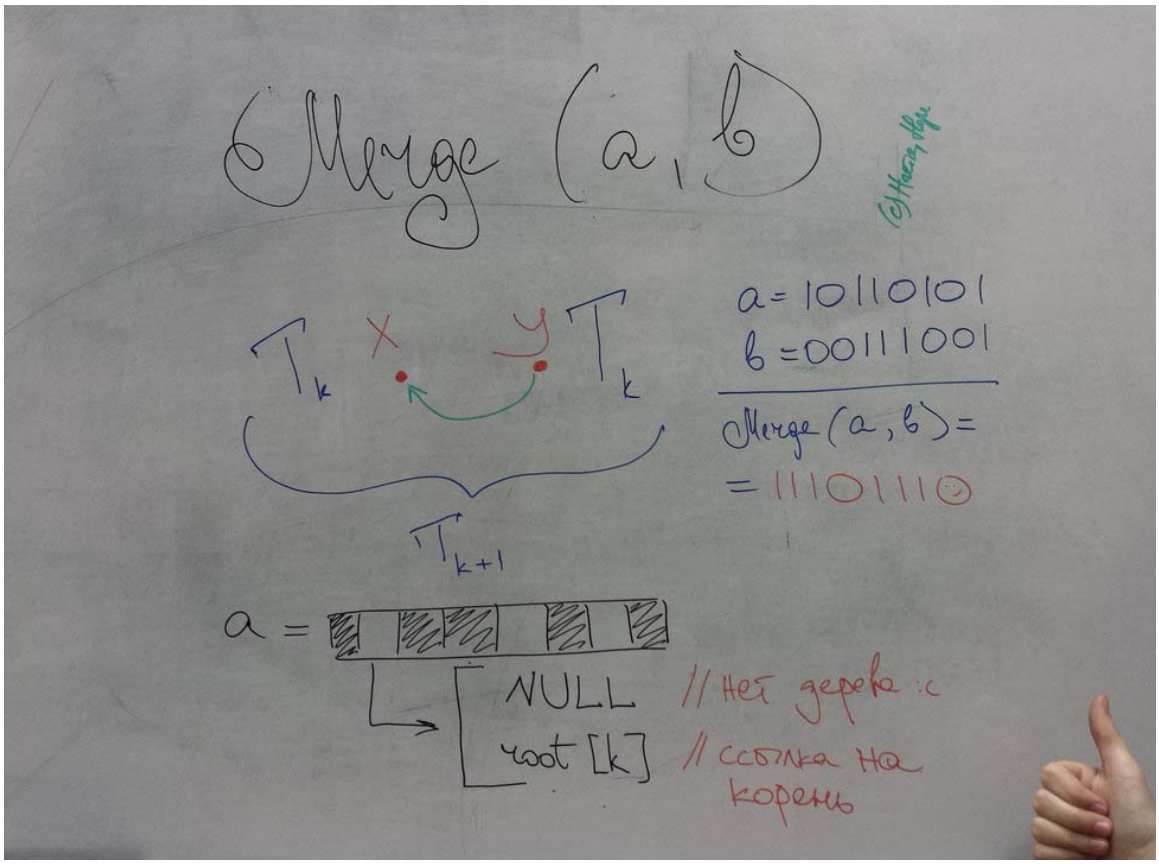




Операция Merge для Биномиальной кучи:

```
1  Merge (a, b) - O(log n)          // сливаем деревья
2  {
3    for k = 0..logN                // деревьев не может быть больше, чем log N
4      if b[k] != NULL              // проверка, есть ли дерево нужной размерности
5      for (i = k, c = b[k]; a[i] != NULL; i++)
6        c = c + a[i];              // получаем из двух T_k одну T_{k+1}
```

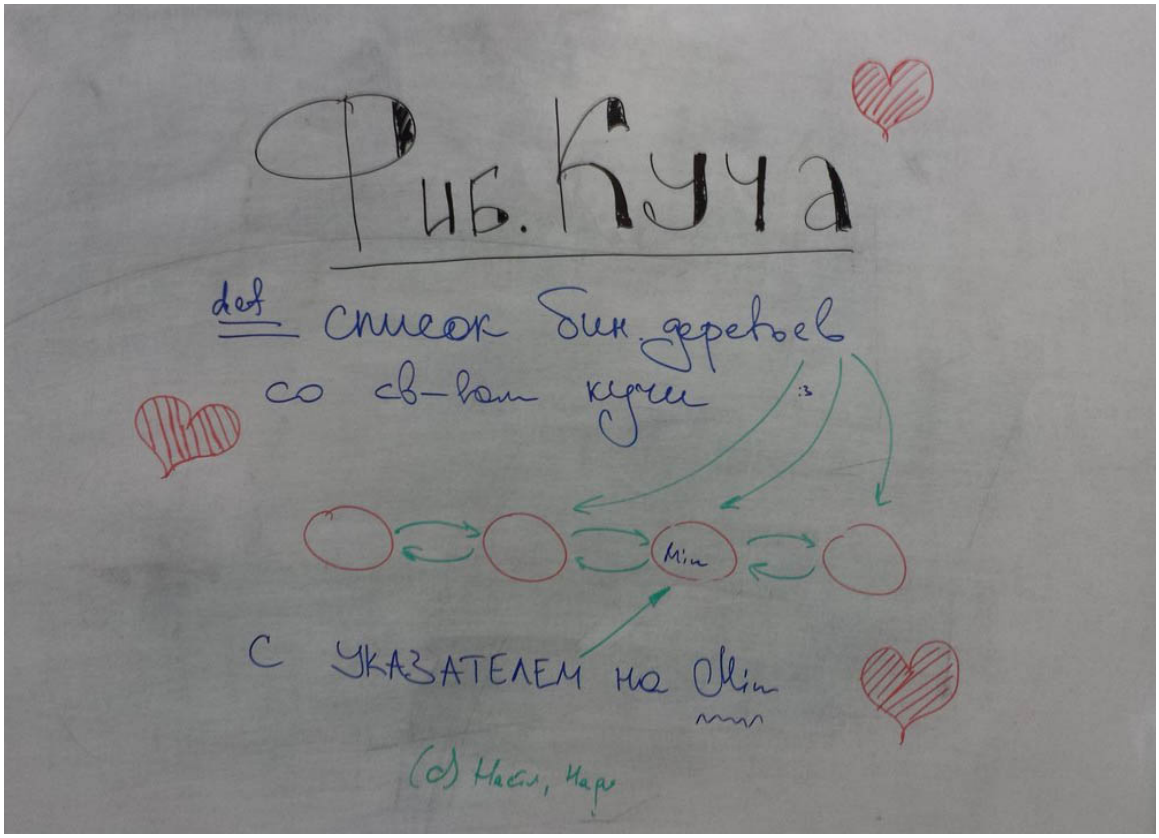
```
7         a[i] = NULL;              // удаляем ссылку на переписанное дерево
8       a[i] = c;                   // сохраняем полученное дерево
9   }
```

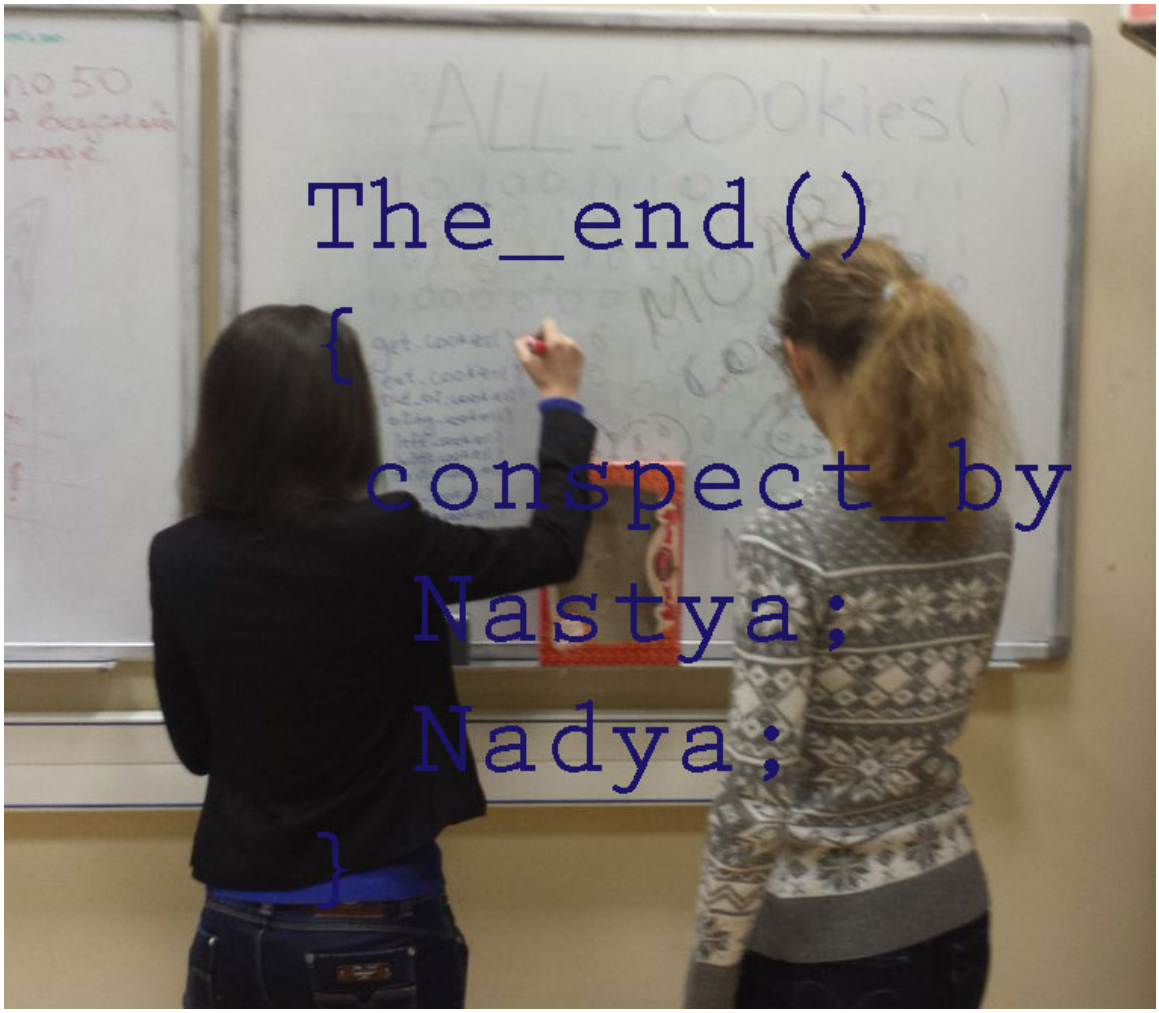


Фибоначчиева куча

Куча Фибоначчи - список биномиальных деревьев со свойством кучи (не обязательно различного ранга).
Позволяет выполнять следующие операции:

- Merge за O(1)
- Add за O(1)
- DelMin за O(logn)
- GetMin за O(1)
- DecreaseKey за O(1)


Фиб. Куча
def список бин. деревьев
со св-вом кучи
Min
с указателем на Min
(с) Настя, Надя


The_end()
{
conspect_by
Nastya;
Nadya;
}