АТД, Абстрактный тип данных

Примеры:
- Стек
- Очередь
- Дек
- Дерево поиска

≡ АТД - это интерфейс и набор контрактов

Пример: стек

Интерфейс:
- pop ()
- push (e)
- top ()
- size ()

Пример: очередь с приоритетами

Реализации: куча, d-ичная куча, массив

Амортизационный анализ

$c_1 \dots c_n$ - стоимость операций над структурой данных

$\exists C: \forall i \; c_i \le C \Rightarrow$ сложность всех операций $\le nC$, а каждой в отдельности $\le C$

Пример: расширяющийся массив

c_i - стоимость добавления i-го эл-та в массив

$\exists i: c_i \le O(n) \Rightarrow \sum_{i=1}^{n} c_i = n \cdot O(n) = O(n^2)$?

На $\forall$ эл-т будем тратить $O(n)$

$\varphi_0, \varphi_1 \dots \varphi_m$ – потенциал после операции i

$\varphi_i \in \mathbb{R}$

$\varphi_m \geq \varphi_0$

$\equiv \tilde{c}_i$ – амортизационная стоимость операции i

$$\tilde{c}_i = c_i + \varphi_i - \varphi_{i-1} = c_i + \Delta\varphi_i$$

$$\sum_{i=1}^{m} \tilde{c}_i = \sum_i c_i + \sum_i \Delta\varphi_i = \sum_{i=1}^{m} c_i + \underbrace{\varphi_m - \varphi_0}_{\geq 0}$$

$$\sum_{i=1}^{m} c_i \leq \sum_{i=1}^{m} \tilde{c}_i$$

$$\exists\, \tilde{C} : \forall\, \tilde{c}_i \leq \tilde{C} \Rightarrow \sum_{i=1}^{m} c_i \leq \tilde{C} \cdot m$$

Пример: расширяющийся массив

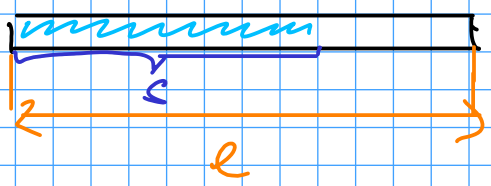
c_i – стоимость добавления i-го эл-та.

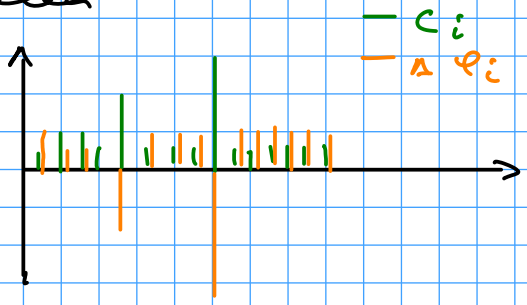
Используем мультипликативную схему

$\varphi_i = 2S - \ell$

где S – кол-во эл-ов в массиве

ℓ – длина массива





Утв: $\tilde{c}_i \leq \text{const} = 3$

1. Пусть $c_i = 1$, $\ell > S$

$$\tilde{c}_i = c_i + (2(S+1) - \ell) - (2S - \ell) = 3$$

($c_i = 1$)

2. Пусть $c_i = S + 1$, $\ell = S$

$$\tilde{c}_i = S + 1 + (2(S+1) - \ell \cdot 2) - (2S - \ell) =$$

$$= S + 1 + 2 - \ell = 3 \qquad (\approx -S)$$

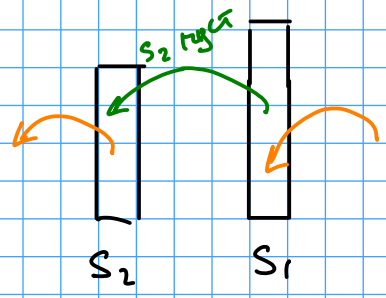
$$\sum c_i \leq \sum \tilde{c}_i = 3 \cdot n$$

$\Rightarrow$ Аморт. стоим. $\#$ операции $= 3$

Метод "потенциала"

Идея: находим эл-ту структуры данных выдаётся C_i монеток, которыми этот эл-т "оплачивает" операции с ним.
Общая стоимость = $\sum C_i$

Пример: очередь на двух стеках



$C_i = 4$

С ∀ эл-ом происходит 4 операции:

S_1.push, S_1.pop
S_2.push, S_2.pop

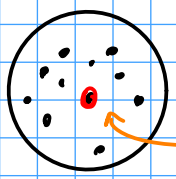
⇒ Стоимость обработки n эл-ов $\le 4 \cdot n = O(n)$

Аморт. стоимость ∀ операции $= O(1)$

NB. $\varphi_i = S_1.\text{size}()$

Система непересекающихся множеств

Запросы:
- make_set // создать множество
- component / find // найти предок.
- union // объединить 2 множества.



$\forall y \in S$: find(y) → x

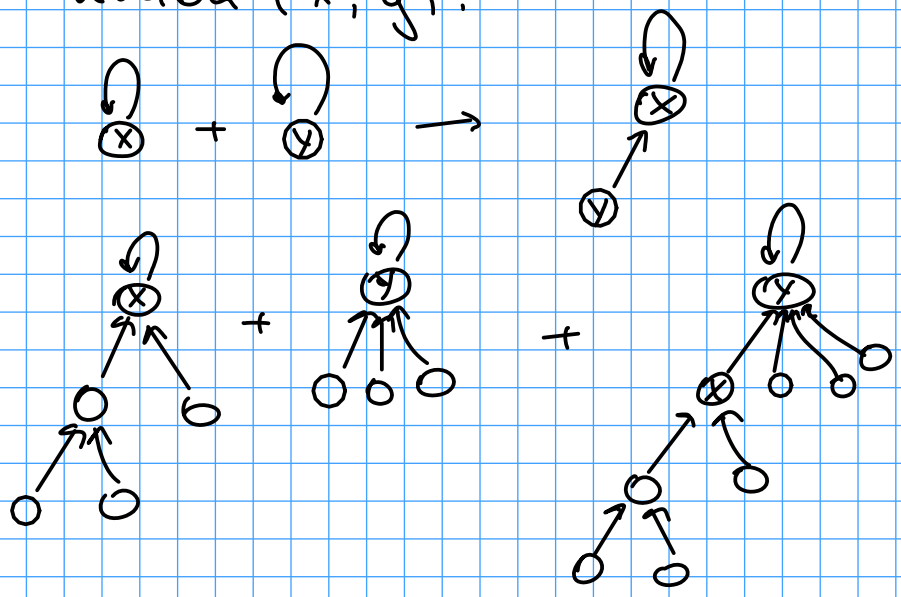
S

---

make_set(x)

find(v):
"иди вверх пока не дашь".

union(x, y):



```
make_set(x):
    parent[x] = x
union(x, y):
    parent[x] = y
find(x):
    while x ≠ parent[x]
        x = parent[x]
    return x
```

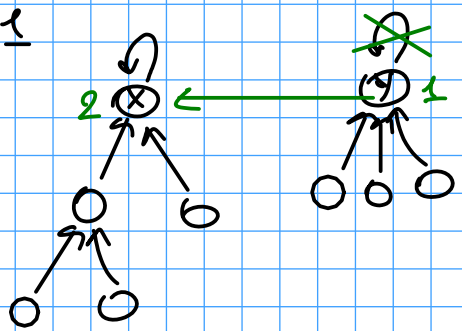
Проблема: find работает $O(h)$

Решение: подвешивать маленькое к большому

```
make_set(x):
    parent[x] = x
    rank[x] = 0

union(x, y):
    x = find(x)   // представитель дер. x
    y = find(y)   //        ———"———      y
    if rank[y] > rank[x]
        parent[x] = y
    else // rank[x] ≥ rank[y]
        parent[y] = x
        if rank[x] = rank[y]
            rank[x] = rank[x] + 1

find(x):
    while x ≠ parent[x]
        x = parent[x]
    return x
```



Утв. 1. Высота дерева с корнем ранга $k = k$

▷ по индукции ◁

Утв. 2. В дереве с корнем ранга k не менее 2^k элементов

▷ База $k = 0$

Переход: union для двух деревьев ранга k.

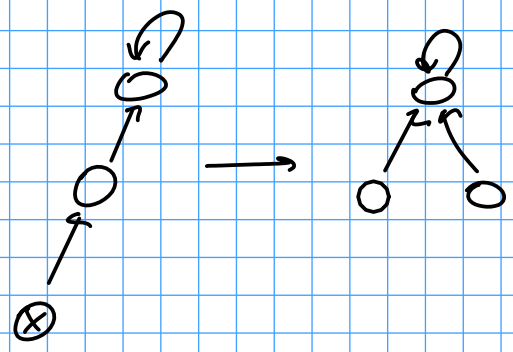
В новом дереве будет $\geq 2^k + 2^k = 2^{k+1}$ элемент. ◁

Утв: Максимальный ранг $= \log_2 n$.

$\Rightarrow$ find работает за $O(\log n)$

Эвристика сжатия путей

```
find (x):
    if x ≠ parent[x]
        parent[x] = find(parent[x])
    return parent[x]
```



Замеч: ранг больше не соотв. высоте, но является оценкой сверху.

Замеч: Ур-ня 2. остались верными.

Все возможные ранги $[0, 1, \dots, \log_2 n]$

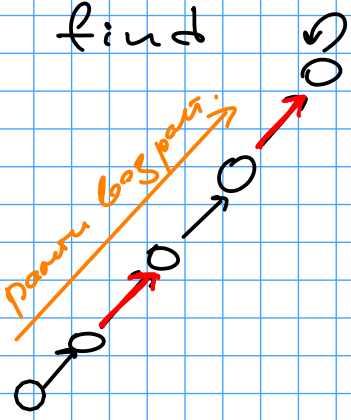
Разобьём его на отрезки $[k+1, 2^k]$

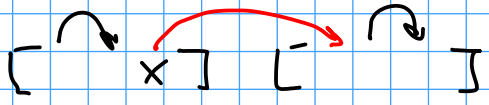
$[0], [1], [2], [3,4], [5, 16], [17, 65536], [65537, 2^{65536}] \dots$

Соответствуют $\log^* n$

Замеч: когда вершина перестаёт быть рутом, её ранг фиксируется.

Вызов find





ребро красное, если

$[\log^*(\text{rank}[x])] < [\log^*(\text{rank}[\text{parent}[x]])]$

Утв: В каждом find-е мы проходим $\le \log^* n$ красных рёбер.

Утв: $rank[x] < rank[parent[x]]$

Амортизация:

Вершина (не корень) ранга из $[k+1, 2^k]$ мы дадим 2^k монеток

длина $< 2^k$

Всего монеток в отрезке $[k+1, 2^k]$

$$\sum_{i=k+1}^{2^k} \frac{n}{2^i} \cdot 2^k \le 2^k \cdot n \cdot \sum_{i=k+1}^{\infty} \frac{1}{2^i} = n$$

вершин с рангом i | кол-во монеток | $= \frac{1}{2^k}$

Всего монеток будет $n \cdot \log^* n$

интервалов

Итого: Если мы сделаем m find-ов, то мы пройдем $\le m \cdot \log^* n$ красных рёбер и в сумме потратим $\le n \log^* n$ монеток

$\Rightarrow$ суммарная сложность $O(m \cdot \log^* n)$, $m \ge n$

Утв: $\exists \le \frac{n}{2^k}$ вершин ранга k (Следствие Утв. 2).

Следствие: Алгоритм Краскала за $O(E \log^* V)$ (при условии целых весов из отрезка $[0, |E|]$ или подобных случаях)