Простые структуры данных.

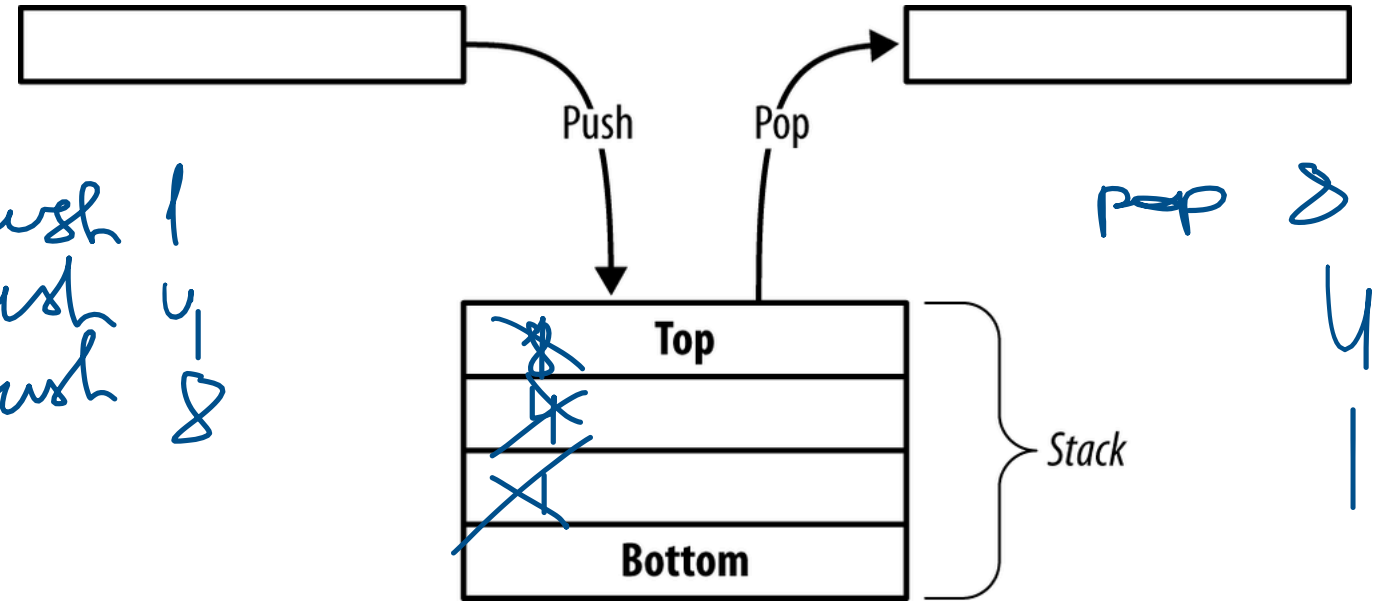
М.К. Горденко mgordenko@hse.ru

Структуры данных

- Структуры данных - это одна из составляющих успешной и эффективной имплементации алгоритмов
- Обычная реализация со списками очень часто проседает (например, если нам надо искать максимальный элемент, то это можно сделать минимум за O(n) операций)
- Кроме того, структуры данных позволяют сильно экономить память (и хотя мы в данном курсе редко говорим про оптимизацию с точки зрения памяти, на больших данных это очень большая и существенная проблема)
- Многие структуры данных имплементированы в Python

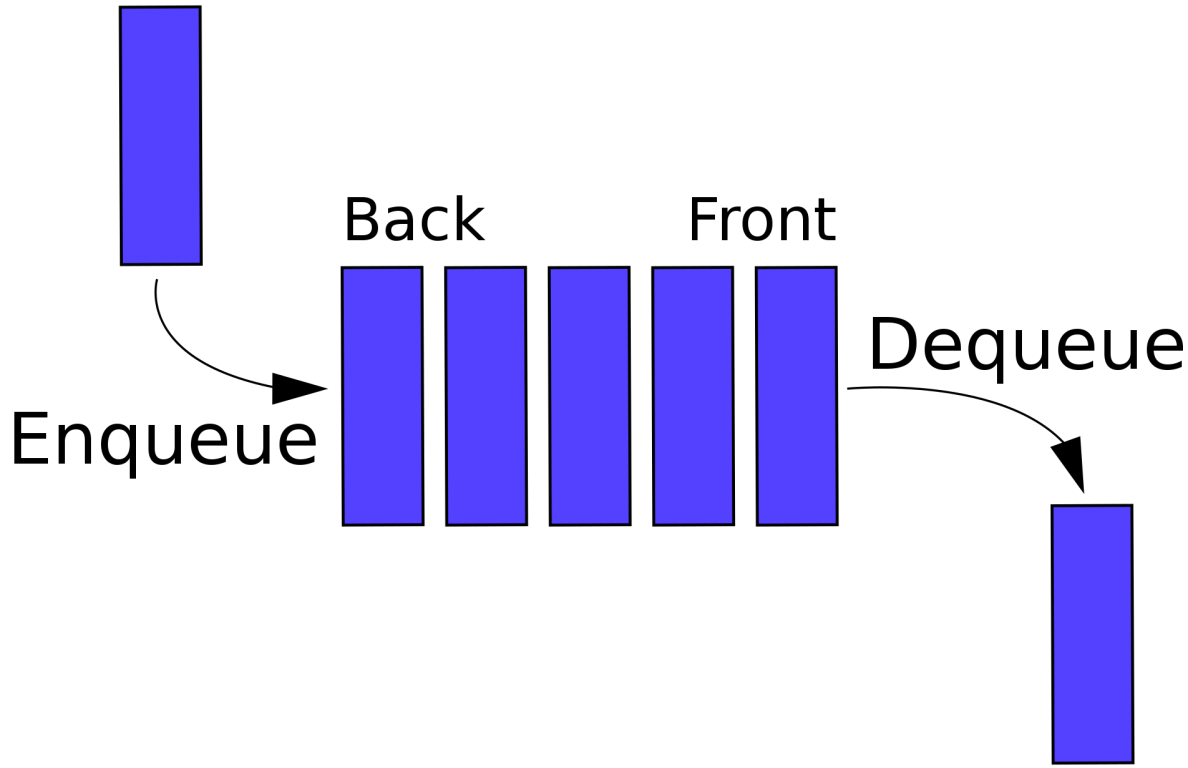
Стек

- Стек - это структура данных, которая реализована по принципу LIFO (last in - first out)
- Это значит, что мы добавляем элементы только в конец и выкидываем элементы тоже из конца (аналогия со стопкой тарелок)
- Данная структура данныз используется на уровне архитектуры памяти, а также потребуется при алгоритмах на графах



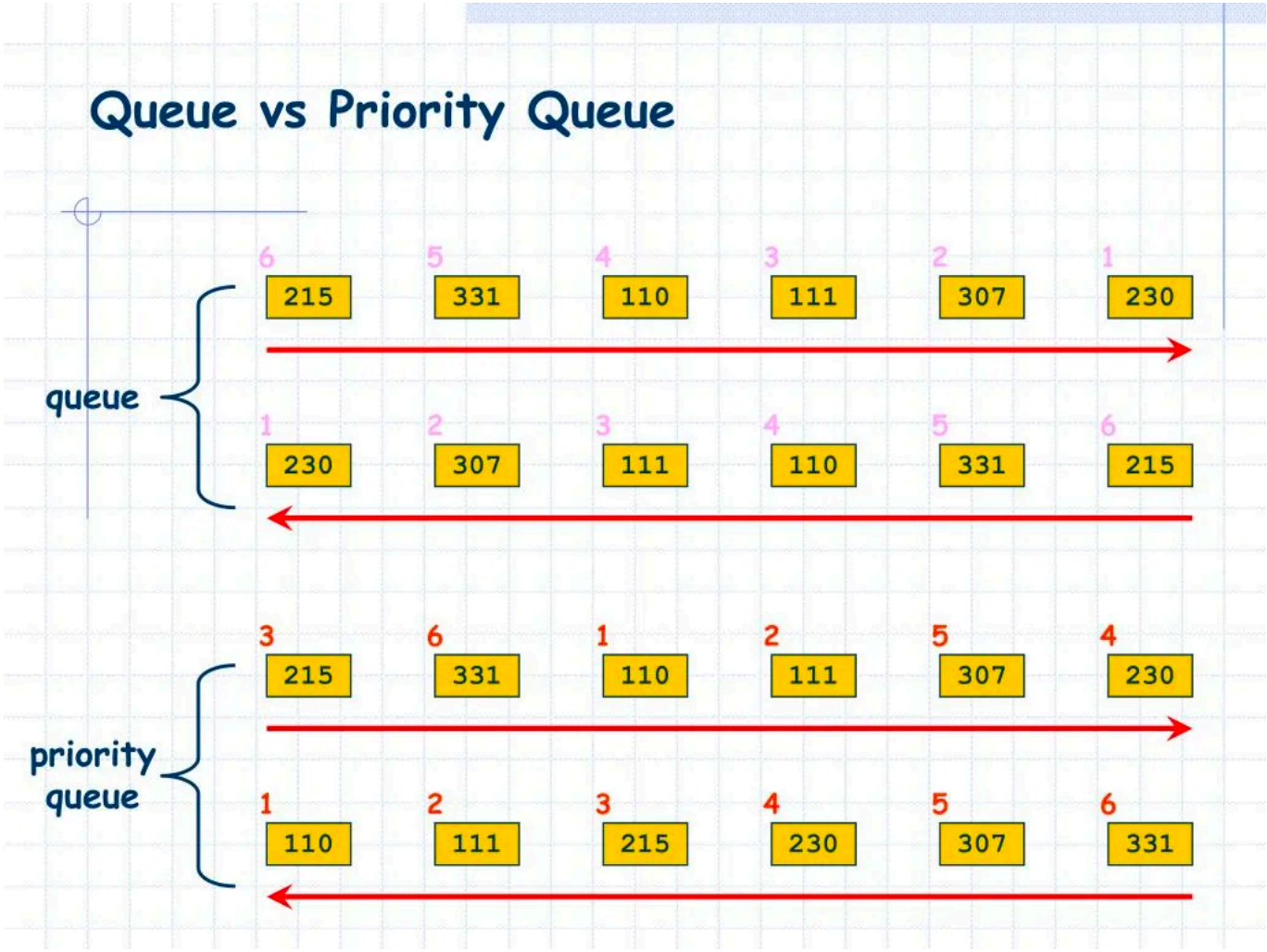
Очередь

- Очередь - это структура данных, реализуемая по принципу FIFO (first in - first out)
- Это означает, что элементы добавляются в конец, а забираются из начала (как в очереди в банке)
- Используется во многих случаях последовательных задач (добавляем задачу, если появляется новая, то ставим в конец)



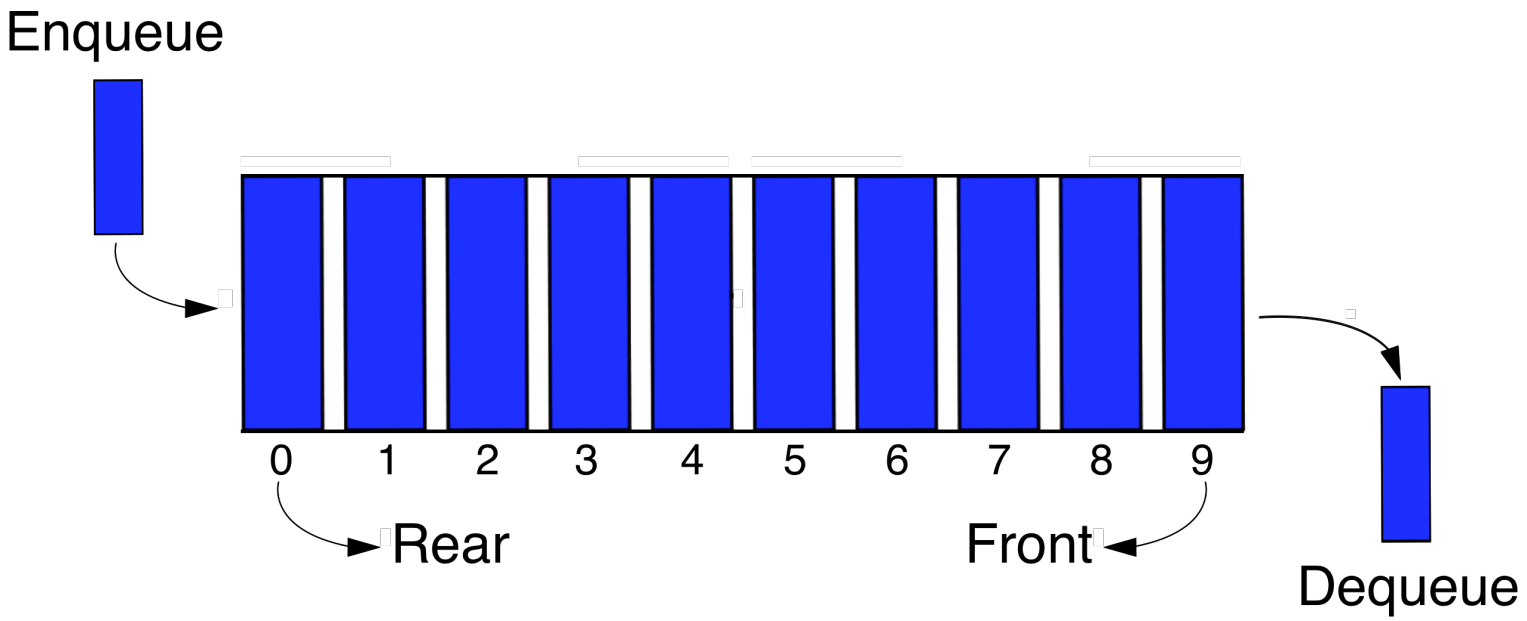
Очередь с приоритетами

- В отличие от обычной очереди, очередь с приоритетами сортирует внутри себя значения по “приоритету”
- Аналогия: в очереди людей у некоторых есть льготы, которых пропускают вне очереди, то есть проходят первыми
- Очередь с приоритетами реализуется на базе двоичной кучи



Дека

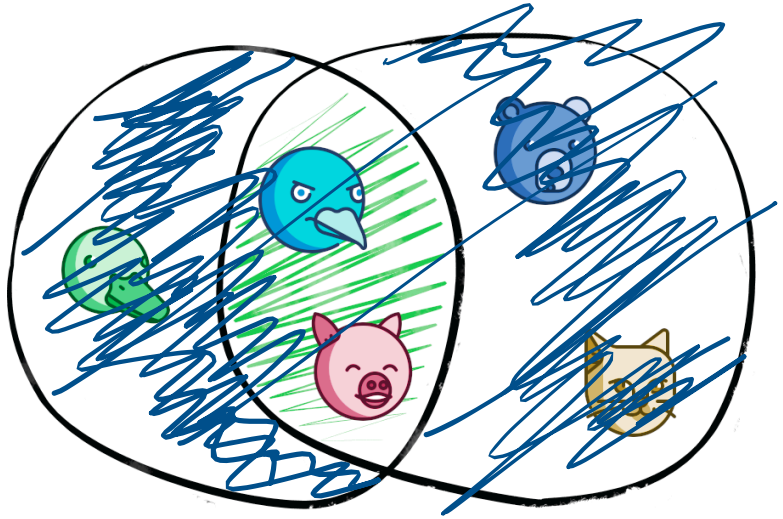
- Дека - это двусторонняя очередь
- Это обозначает, что теперь добавлять мы можем как в начало, так и в конец, и удалять мы можем с обоих сторон
- Очень часто используется в параллельных вычислениях (например)



set 1 2 1 → 1 2

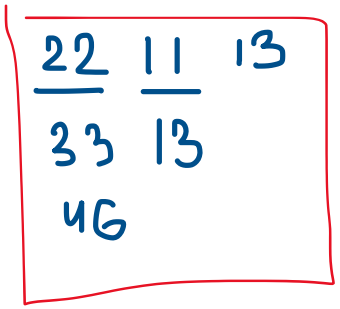
Множество

- Множество - это контейнер, который содержит в себе неупорядоченные уникальные элементы
- Это очень удобно, когда нам не важно, сколько раз встретился элемент, а важно просто его наличие
- Благодаря внутреннему устройству множеств, они позволяют делать многие операции сильно быстрее, чем обычные списки (например, поиск элемента во множестве составляет O(1) за счет внутренней имплементации)
- Кроме этого, множества можно объединять, искать пересечение, разность и так далее



4

10 11 12 13

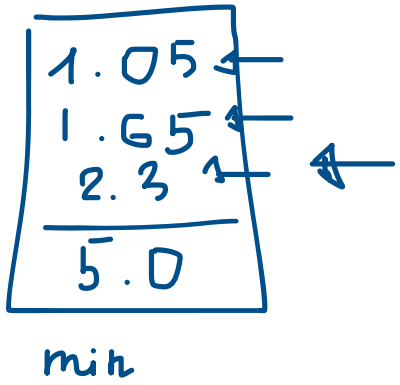


12 13 22

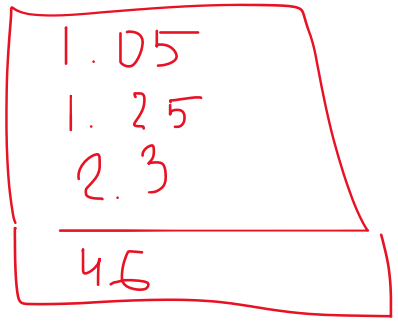
22 25

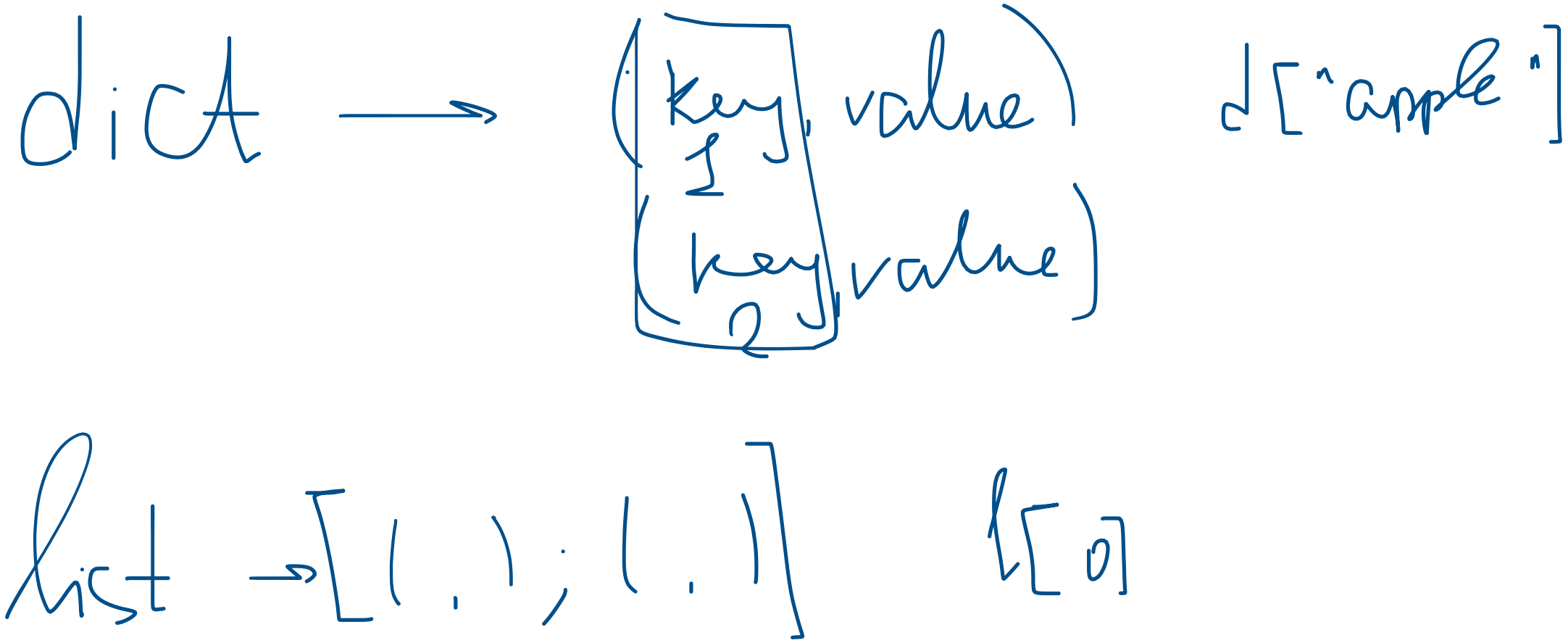
46

10

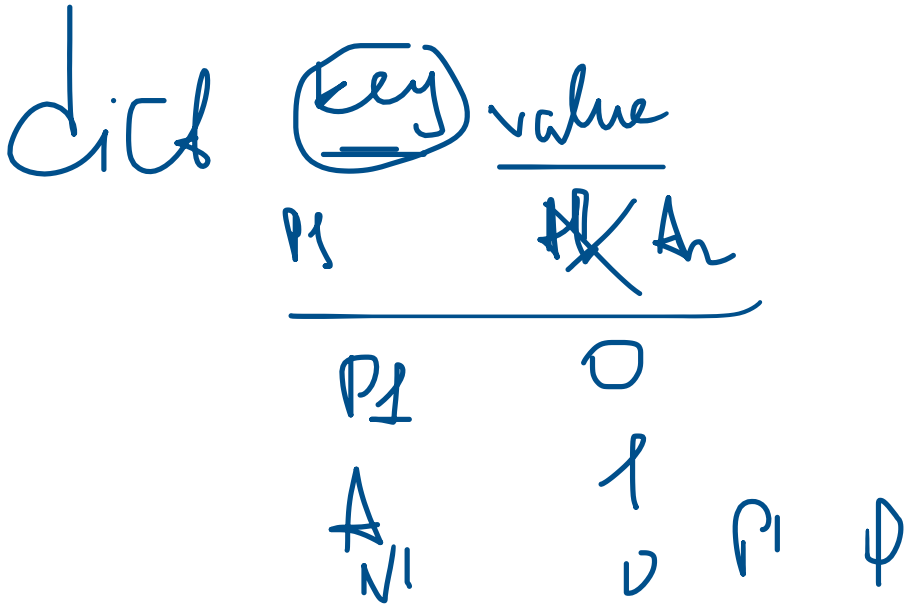


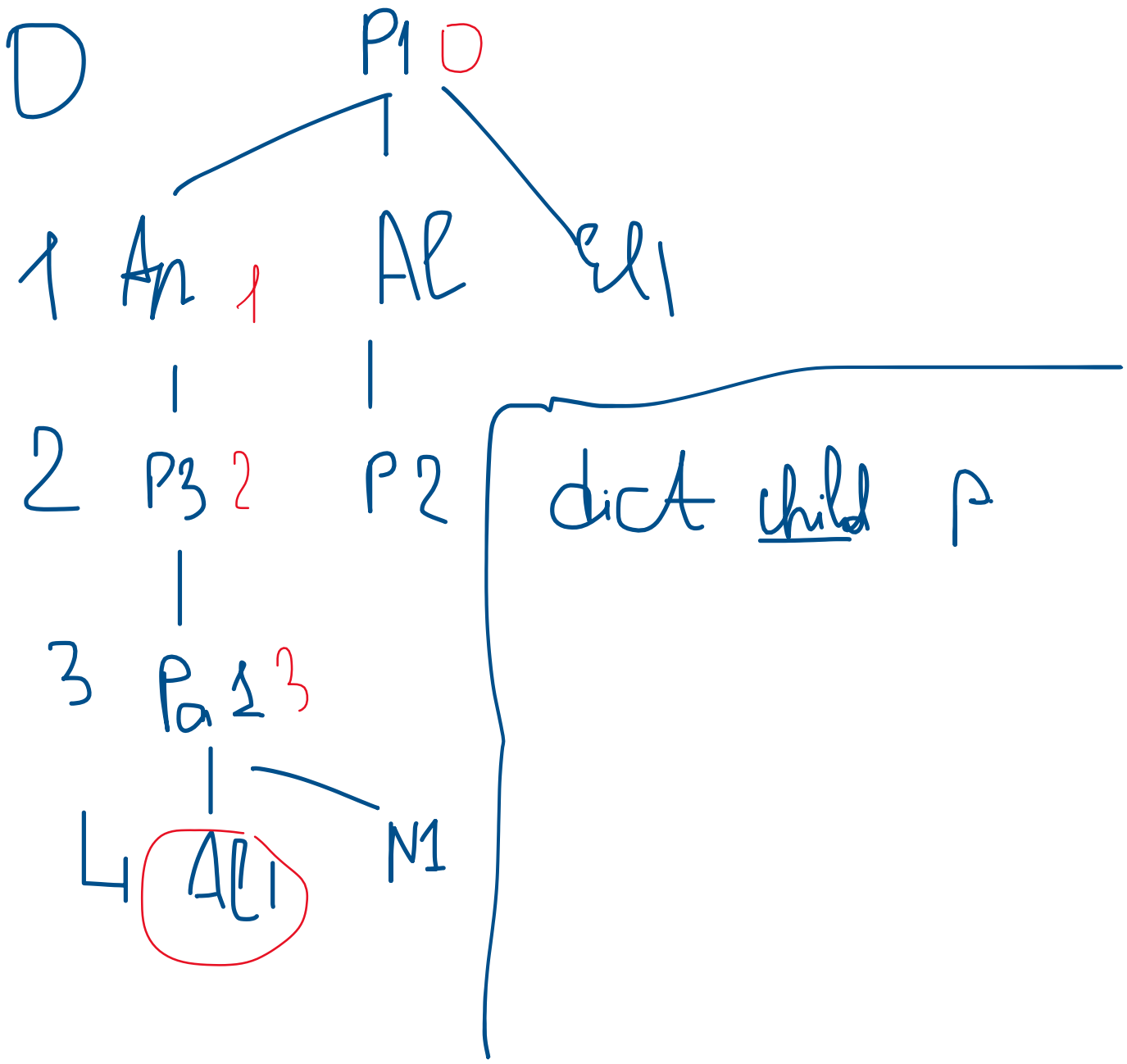
min




dict
(key, value)
1
(key, value)
2
d["apple"]
list
[(,); (,)]
l[0]

Ввод	Вывод
9	Alexander_I 4
Alexei Peter_I	Alexei 1
Anna Peter_I	Anna 1
Elizabeth Peter_I	Elizabeth 1
Peter_II Alexei	Nicholaus_I 4
Peter_III Anna	Paul_I 3
Paul_I Peter_III	Peter_I 0
Alexander_I Paul_I	Peter_II 2
Nicholaus_I Paul_I	Peter_III 2

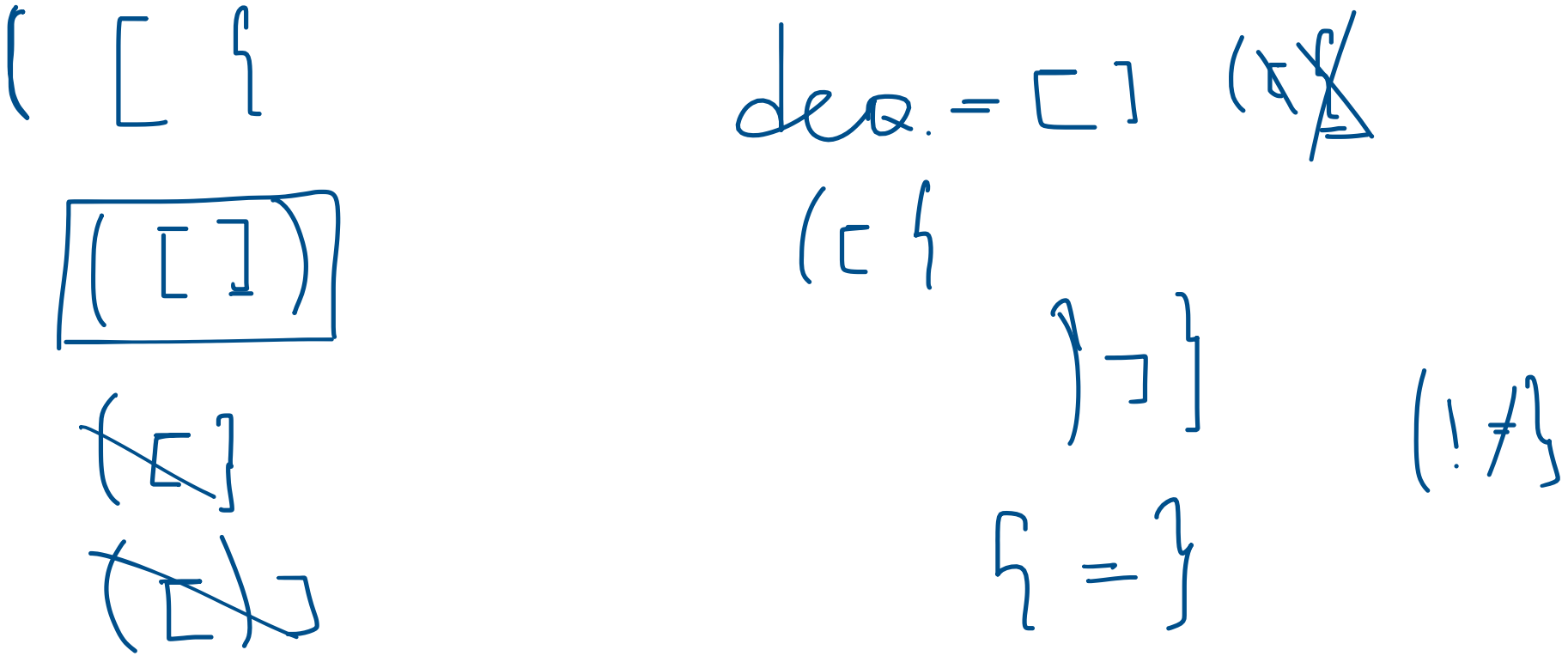




```
n = int(input())
tree = {}
for i in range(n - 1):
    child, parent = input().split()
    tree[child] = parent
print(tree)
s = {}
all_tree = tree.keys() | tree.values()
type(all_tree)
for i in all_tree:
    h = 0
    p = i
    for l in range(n):
        if p in tree:
            h += 1
            p = tree[p]
        else:
            break
    s[i] = h
for i in sorted(s):
    print(i, s[i])
```

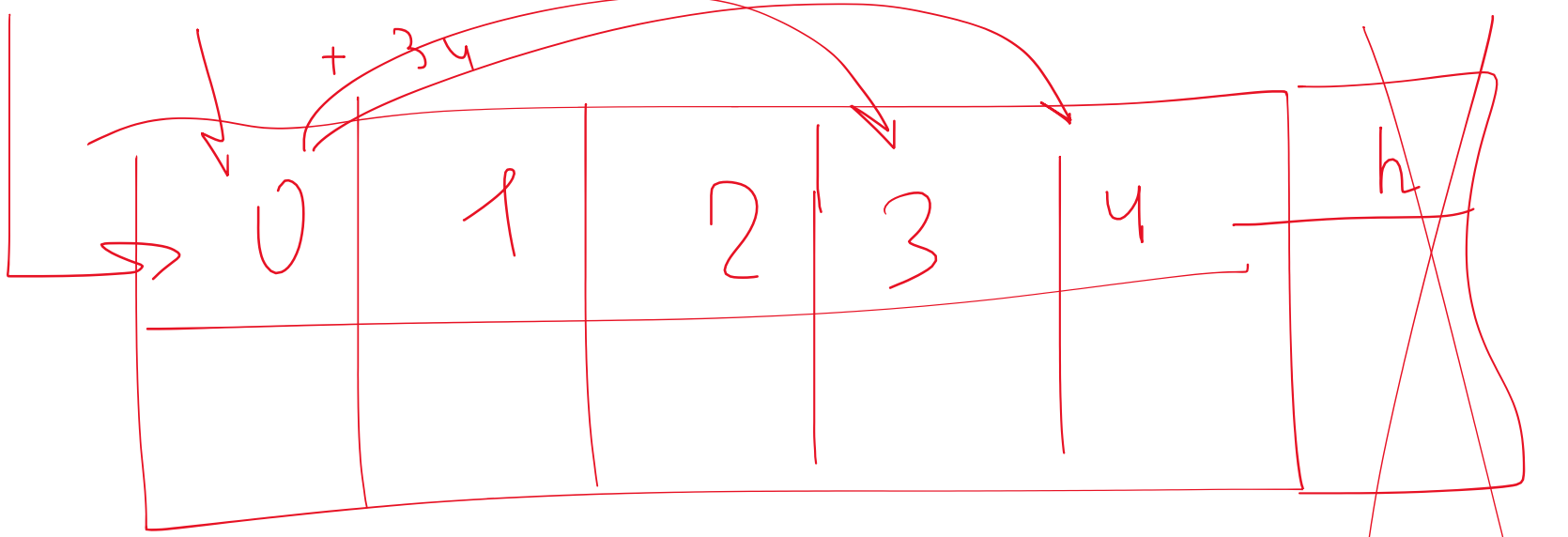
Заключение

- Использование структур данных при эффективной имплементации позволяют выполнять те или иные операции быстрее, что позволяет ускорить работу алгоритмов
- Многие из структур данных, которые были пройдены, будут использоваться дальше для алгоритмов (например, на графах и на строках)
- В зависимости от задачи можно выбирать тут или иную структуру (или представление графа)



8 9 + 1 7 − *

(8 + 9) * (1 − 7) − *


+ 3 4
0
1
2
3
4
h