**Теоретическое решение задачи B.**

**Алгоритм решения и доказательство его правильности.**

В данной задаче, от нас требуется отвечать на 2 типа запросов:

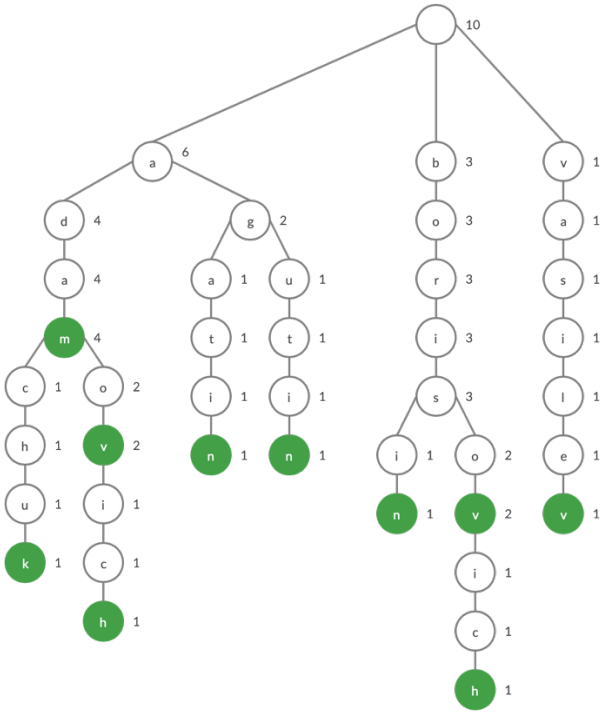
1) Добавление нового кандидата для розыгрыша лотереи
2) Выбор победителя, за номером в списке лексикографического порядка фамилии кандидатов

Данные запросы достаточно просты, однако временные ограничения задачи и количество запросов до $10^5$, не позволяют использовать алгоритмы даже с линейной с асимптотической оценкой времени. Поэтому было выбрано структуру данных – Бор. Бор - это структура данных для эффективного хранения и обработки строк. Вершины в боре соответствуют отдельным символам. Строки представляются в виде путей по бору от корня до последнего символа.

**Хранение данных**.

Данные в Боре хранятся в виде дерева, поэтому для каждого символа был разработан узел node:

- letter – одна буква из фамилии (символ)
- finally – булевая переменная, которая соответствует истине, когда слово заканчивается на этой букве
- count – количество слов, оканчивающихся в этом поддереве
- childs – вектор детей узла (последующие буквы слова)




Пример нашего Бора

Зеленым обозначены узлы с finally = true.
Цифра справа – состояние переменной count.
В примере мы добавили такие фамилии:

- adam
- adamchuk
- adamov
- adamovich
- agatin
- agutin
- borisin
- borisov
- borisovich
- vasiliev

**Используемые методы:**

- *search_letter* – Ищет букву в векторе childs структуры node, возвращает ее индекс в векторе или -1, если там она отсутствует
- *add_letter* – Добавляет букву в вектор childs структуры node, соблюдая алфавитный порядок возвращает ее индекс в векторе или -1, если там она отсутствует
- *add_people* – Добавляет фамилию человека в наш Бор list_of_people. Опускается вниз по дереву на каждом уровне инкрементирует значение count узла. Переходит на следующий уровень с помощью метода search_letter в вектор узла childs и в случае необходимости метода add_letter. В конце слова меняет значение finally на true, последнему узлу.
- *search_winner* – Находит фамилию человека по номеру в списке лексикографического порядка. Опускается вниз по дереву запоминая буквы по которым проходит. Выбирает букву из списка childs таким образом, что бы сумма значений переменных count левых поддеревьев была меньше искомого значения, однако добавляя следующую ветку, значение превышает или соответствует значение. При проходе через вершину со значением finally = true, значение sum (в котором мы храним сумму фамилий в левых поддеревьях) инкрементируется.

**Временная сложность.**

Наш алгоритм включает такие функции:

1. **add_people.** По алгоритму опускается по дереву от корня, на длину слова – n и на каждой итерации выполняется поиск или добавление нужного узла, которые выполняются линейно, количество букв в массиве childs – $\Sigma$ (max: 26). Поэтому конечная асимптотика – $O(n\Sigma)$.
2. **search_winner.** Аналогично предыдущему методу проход от корня дерева к концу фамилии и на каждой итерации поиск в списке узлов. Итого - $O(n\Sigma)$.

Видим, что все функции реализованы с одной асимптотикой $O(n\Sigma)$.

Если заменим длину одной строки n на сумму длин всех строк w, на всех запросах – **получим конечную асимптотическую оценку всего алгоритма – O(wΣ).**

**Затраты памяти.**

Для реализации описанного выше алгоритма, требуется постоянно хранить все узлы дерева. Так как, многие символы слов будут совмещены в одни узлы, то точно оценить затраты памяти невозможно. Худшие случаи – когда, узлы пересекаются крайне редко - O(w), где w – суммарное количество символов всех фамилий

**Таким образом, итоговые затраты памяти худшего случая= O(w).**