Реализация ARTful деревьев на C и Go

Антон Гардер

Рассмотрим АТД Map

Операции:

- Добавить пару (ключ, значение)
- Удалить ключ и значение
- Получить значение по ключу

Обычно реализуют на деревьях поиска (Red-black tree) или хеш-таблицах

Стандартные решения

Деревья поиска:

- Добавление, удаление и поиск за $O(\log N \cdot \text{size}(key))$

Хеш-таблицы:

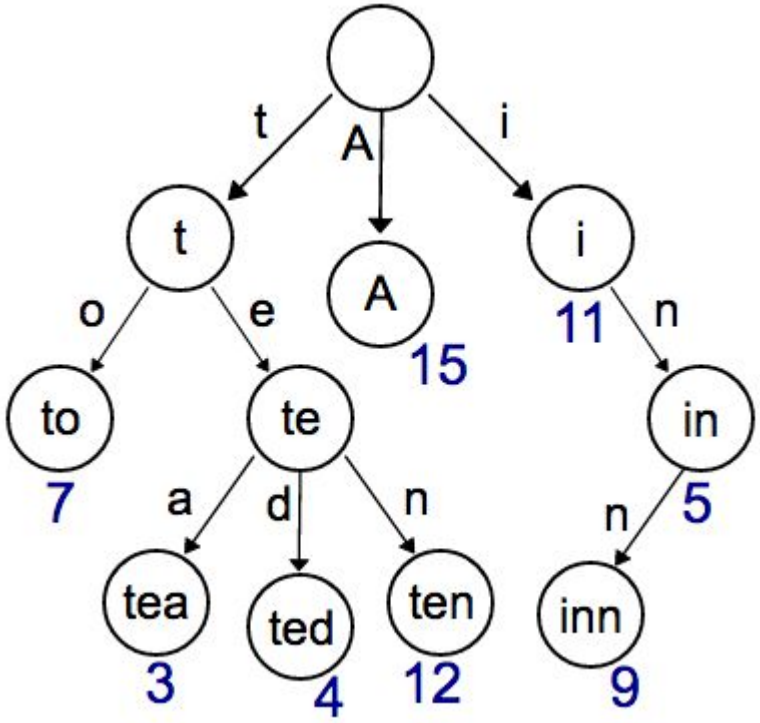
- Добавление, удаление и поиск амортизированно за $O(\text{size}(key))$

Другие решения

Цифровой бор (trie, radix tree):
Разбиваем ключи “посимвольно”, храним значения в конце путей в дереве.

Асимптотика операций:
O(len(key))

Память:
O(N • len(key) • размер алфавита)



Модификации: быстрый цифровой бор (x-fast trie), сверхбыстрый цифровой бор (y-fast trie)

Проблемы бора

Эквивалентные вещи:

- Ускорение операций
- Увеличение размера алфавита
- Увеличение количества потребляемой памяти

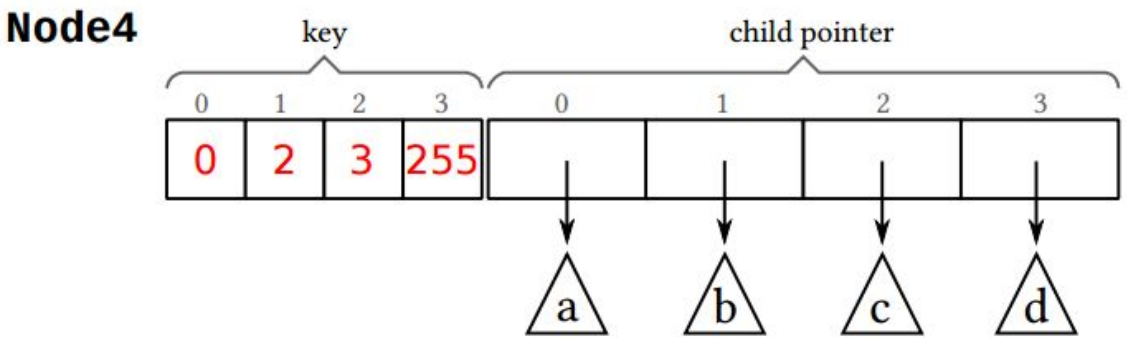
Как бороться?

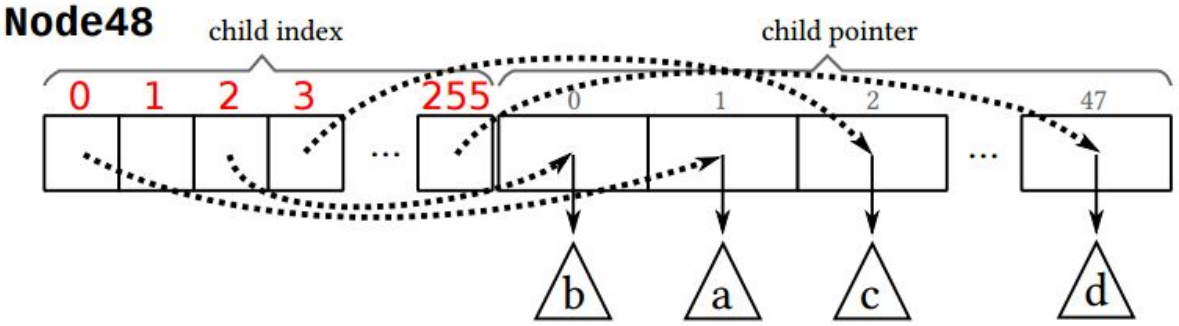
Перестать считать размер вершины константой

Adaptive Radix Tree

Введем четыре типа вершин и будем менять тип вершины в процессе работы

- Node4
- Node16
- Node48
- Node256





Написали, запускаем

Взлетело? Нет :(

len(key)	1e3	1e4	1e5	1e6
art	0.0288	0.2795	1.2909	8.1344
map	0.0008	0.0020	0.0149	0.2552
unordered	0.0008	0.0033	0.0289	0.3471

Оцениваем среднее время тысячи операций в зависимости от длины ключа

Может быть на трудных тестах?

Все равно нет :(

len(key)	1e3	1e4	1e5	1e6
art	0.0054	0.0063	0.824	8.134
map	0.0018	0.0054	0.068	0.785

Когда ключи долго сравнивать (aaa….aaxyz), начиная с какой-то длины ключа art мог бы обгонять по скорости map, но нет

Чему я научился

- Использовать новую интересную структуру данных
- Писать код на C без боли
- Отлаживать код на C (не без боли)
- Думать дважды прежде чем использовать интересную структуру данных вместо стандартной

Спасибо за внимание

Вопросы?