Цифровой поиск

Дискретный анализ 2012/13

Андрей Калинин, Татьяна Романова

1 октября 2012 г.

Литература

- Д. Э. Кнут, Искусство программирования, том 3, п. 6.3 Цифровой поиск, с. 527-535.

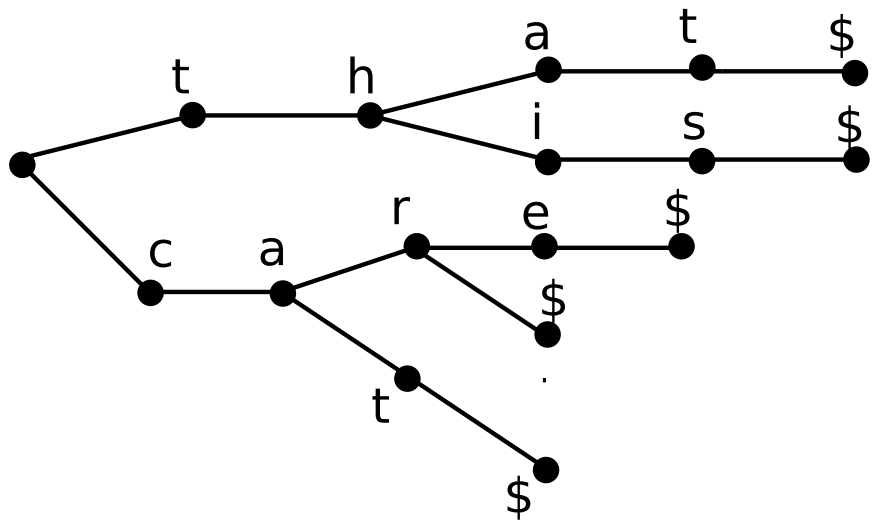
Раздел

Способ построения

- Цифровой поиск основан не на сравнении ключей, а на их представлении в виде последовательности букв или цифр.
- Пример — побуквенные метки в больших словарях.
- Простейшая реализация — дерево ключей. Пусть M — мощность алфавита, N — количество ключей. В каждом узле может храниться массив указателей размером M или, если M велико, а N нет, линейный список.
- Ни один ключ, хранящийся в дереве, не является префиксом другого.

Пример

Дерево ключей со словами this, that, car, care, cat:



Недостатки

- Нерациональный расход памяти.
- Считывание из памяти строки при каждом обращении к узлу.
- Невозможность хранить строки на диске из-за большого количества чтений при поиске.

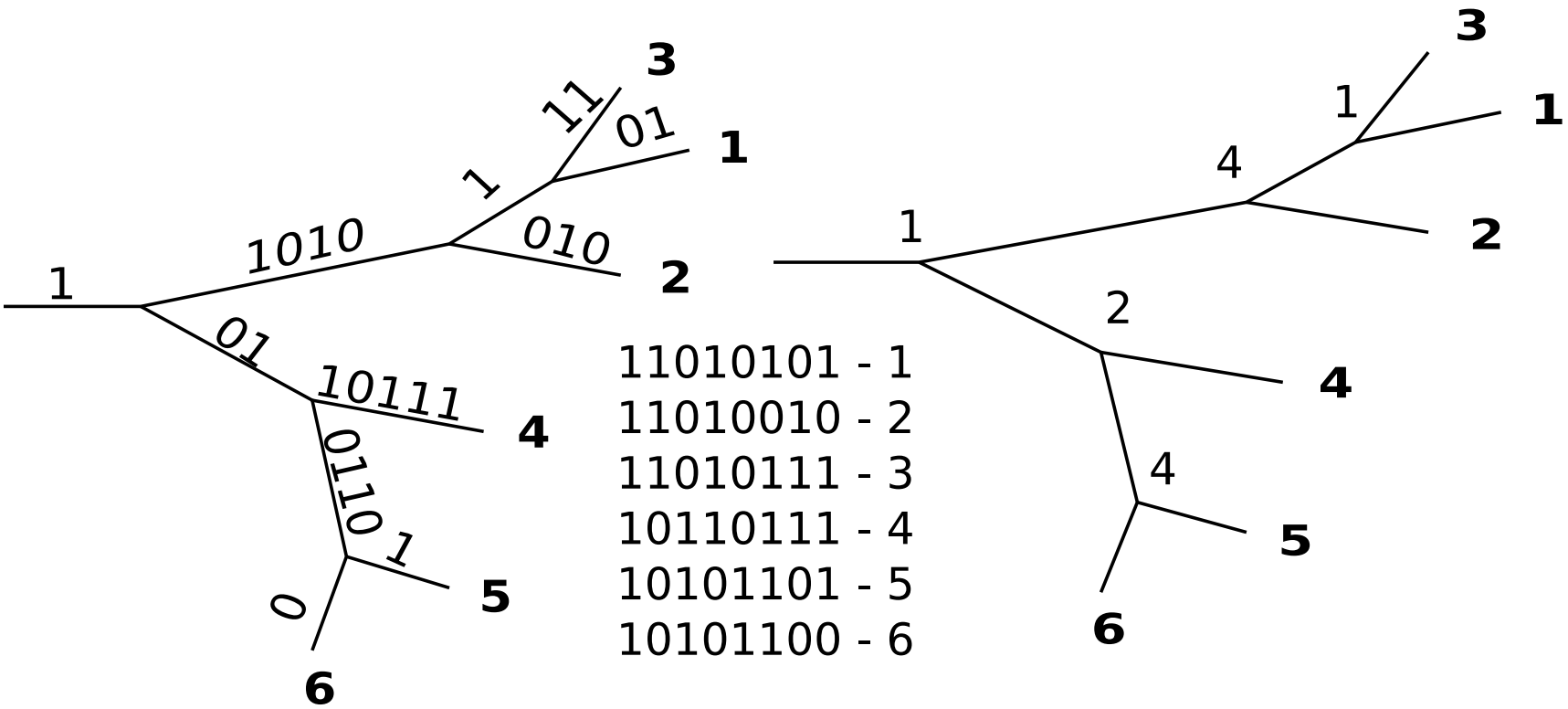
Раздел

Основные идеи

- Practical Algorithm to Retrieve Information Coded in Alphanumeric.
- Используется бинарный алфавит.
- Во внутреннем узле хранится количество битов, которые можно пропустить до проверки очередного бита. Эти биты общие для всех потомков.
- Дерево не содержит однопутевых веток.
- Указатель на ключ хранится в листовом узле.
- При поиске находится ключ, совпадающий с искомым по маске. Затем производится чтение строки (возможно, с диска) и сравнение с искомым ключом.
- Все узлы хранятся в памяти, все данные могут храниться на диске.

Пример

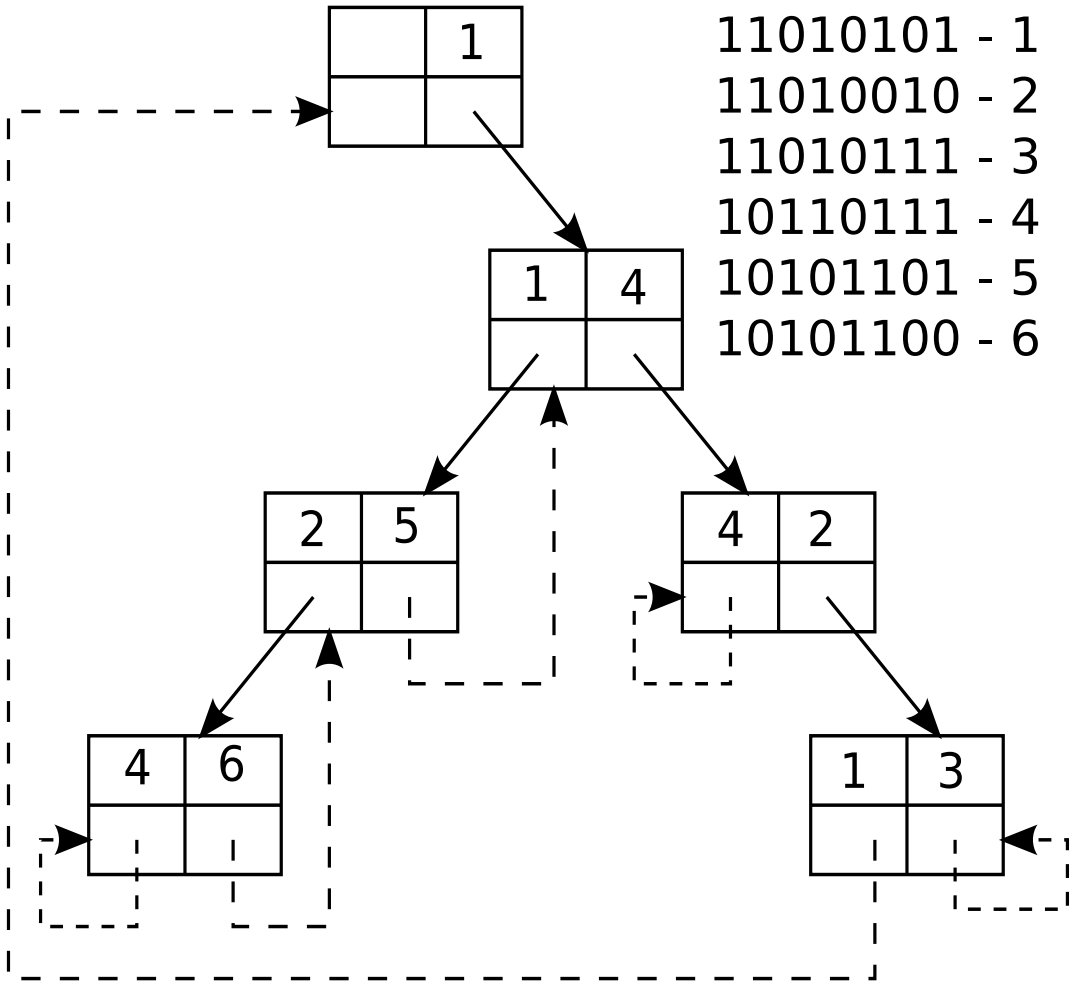
Переход от дерева ключей к PATRICIA:



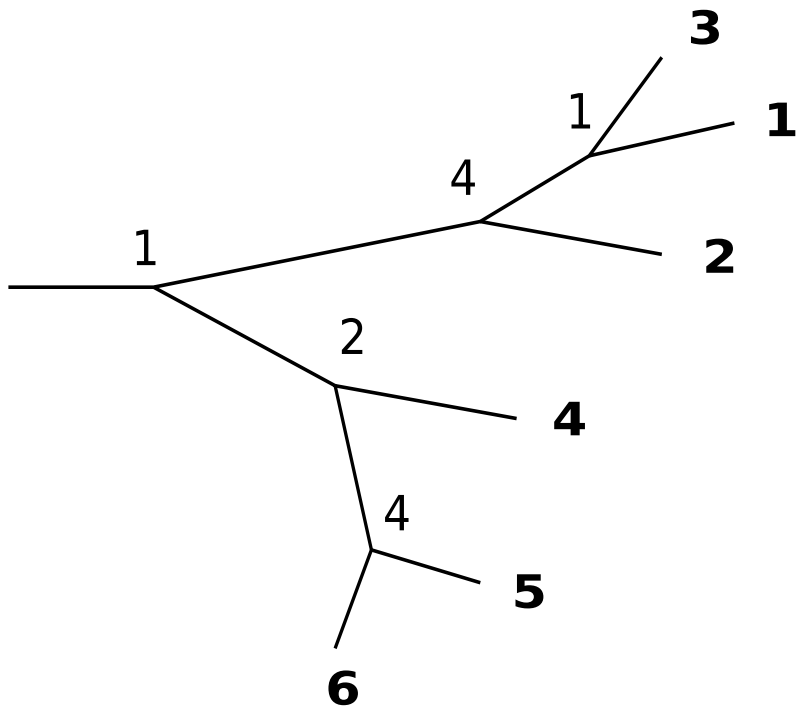
Улучшение

- Количество «развилок» (внутренних узлов, считая корень) совпадает с количеством листовых узлов.
- Ключи могут храниться вместе с «развилками».
- Для указания на хранящийся в листе ключ используется ссылка «вверх», помечаемая специальным образом.

Пример



11010101 - 1
11010010 - 2
11010111 - 3
10110111 - 4
10101101 - 5
10101100 - 6



Структуры данных

- Ключ:
  1. bitlen — длина ключа в битах.
  2. data — массив с данными, можно получить нужный бит функцией bit(key,n)
- Узел дерева:
  1. key — номер соответствующего ключа.
  2. skip — количество пропускаемых бит.
  3. links[2] — ссылки на левое (0) и правое (1) поддерево, состоящие из:
     - 3.1 link — указателя.
     - 3.2 up — признака, что этот указатель ведёт «наверх».

Раздел

Основная идея

1. Проходим по всему дереву, последовательно проверяя нужные биты, используя поля skip.
2. При проходе по ссылке, помеченной флагом up, считываем из узла номер ключа.
3. Достаём ключ из базы и сверяем его с искомым: если он равен, то мы его нашли, если не равен, то длина совпадающего префикса будет самым длинным совпадением из всей базы ключей PATRICIA.

Алгоритм

PAT-SEARCH(T, key)

1 $ref \leftarrow \&T.root.links[1], pref \leftarrow \&T.root$
2 $nd \leftarrow ref.link, j \leftarrow 0$
3 **while not** $ref.up$
4 $\quad j \leftarrow j + nd.skip$
5 $\quad pref \leftarrow ref$
6 $\quad$ **if** $j < key.bitlen$ **then** $dir \leftarrow bit(key, j)$ **else** $dir \leftarrow 0$
7 $\quad ref \leftarrow \&nd.links[dir]$
8 $\quad nd \leftarrow ref.link$
9 $m =$ COMMON-BIT-PREFIX$(nd.key, key)$
10 **return** $(pref, dir, m)$

Раздел

Основная идея

1. Ищем ключ в дереве, вычисляем самый длинный совпадающий префикс.
2. Нужно найти связь, на которую приходится конец этого префикса, и разбить её новым внутренним узлом дерева.
3. При этом одна из связей нового узла должа будет показывать на него самого, а другая — продолжать старую связь.

Алгоритм

PAT-INSERT(T, key)

```
 1  if T.empty()
 2        T.root ← NEW-NODE(key), T.root.point-to(1, T.root)
 3        return (TRUE, T.root)
 4  (pref, dir, m) ← PAT-SEARCH(T, key)
 5  if m = key.bitlen
 6        return (FALSE, ref.link)
 7  ref ← &T.root.links[1], nd ← ref.link, j ← 0
 8  while not ref.up and j + nd.skip < m
 9        j ← j + nd.skip
10        ref ← &nd.links[bit(key, j)]
11        nd ← ref.link
12  return (TRUE, SPLIT-INSERT(ref, key, m, m − j))
```

Алгоритм

SPLIT-INSERT(ref, key, m, k)

```
1   N ← NEW-NODE(key)
2   dir ← bit(key, m)
3   N.point-to(dir, N)
4   N.links[1-dir] ← *ref
5   N.skip ← k
6   if not ref.up
7         ref.link.skip ← ref.link.skip − k
8   ref.link ← N
9   ref.up ← FALSE
10  return N
```

Раздел

Основная идея

- В худшем случае нужно модифицировать два узла: узел со ссылкой «вверх» и узел, где хранится номер удаляемого ключа.
- Выполняем удаление в два этапа:
  1. Перемещаем ключ из листа в узел, где содержится удаляемый ключ, тем самым ссылка «вверх» будет вести на тот же узел.
  2. После чего удаляем лист.

Алгоритм

```
PAT-DELETE(T, key)
 1  (pref, dir, m) ← PAT-SEARCH(T, key)
 2  if m ≠ key.len
 3        return (FALSE, NIL)
 4  if *pref = T.root
 5        tmp ← T.root, T.root ← NIL
 6        return (TRUE, tmp)
 7  R ← pref.link, D ← R.links[dir]
 8  if D = R
 9        *pref ← D.links[1-dir]
10        if not pref.up
11              pref.link.skip+ = D.skip
12        return (TRUE, D)
```

Алгоритм

```
PAT-DELETE(T, key)
 1  if R.links[1-dir].link = R
 2        SWAP-KEYS(D, R)
 3        *pref ← LINK(TRUE, D)
 4  else
 5        *pref ← R.links[1-dir]
 6        if not pref.up
 7              pref.link.skip+ = R.skip
 8        (pref1, dir1, m1) ← PAT-SEARCH(T, R.key)
 9        pref1.link.point-to(dir1, D)
10        SWAP-KEYS(D, R)
11  return (TRUE, R)
```