大量データをいかに扱うか

圧縮全文索引を用いた文字列処理

発表者：定兼　邦彦　（情報学プリンシプル研究系）

概要

大量の文書からの検索や知識発見を高速に行うには，文字列処理を高速に行う必要がある．しかし，Web検索エンジン等で用いられている転置ファイルなどの索引を用いる手法では細かな検索ができない．また，接尾辞配列や接尾辞木などの索引を用いれば細かな検索は可能になるが，索引のサイズが大きくなってしまう．本研究では圧縮全文索引を用いた効率的文字列処理手法を提案する．

Abstract

In order to search large scale documents and discover some knowledge form them, fast string processing is necessary. However, methods based on inverted files, which are used in Web search engines, do not support detailed search. Though detailed searches are possible by using indices such as suffix arrays and suffix trees, there is a problem of large index size. In this research, we propose efficient string processing methods based on compressed full-text indices.

CSA (Compressed Suffix Arrays, 圧縮接尾辞配列) ライブラリ

文字列を圧縮して保存し，各種検索処理を高速に実現
search(key): keyの順位を検索
lookup(i): 順位 i の部分文字列の出現位置を求める
text(s, t): s 文字目から t 文字目までの部分文字列を復元
child_l(i,j): 順位 i から j の部分文字列の左に出現する文字集合
child_r(i,j): 順位 i から j の部分文字列の右に出現する文字集合

接尾辞配列

0	9	$
1	1	acagcagg$
2	3	agcagg$
3	6	agg$
4	2	cagcagg$
5	5	cagg$
6	8	g$
7	4	gcagg$
8	7	gg$

ライブラリの特徴
- 様々な圧縮法を利用可能
- 新しい圧縮法の開発も容易
  - 圧縮の基本操作のみ実装すればいい．検索処理はライブラリの関数を利用できる．
  - 圧縮法に特化した高速検索アルゴリズムがあれば，ライブラリの関数と置き換えられる
- 接尾辞木 (suffix trees) の模倣も可能

search(“a”) = [1,3]
search(“ca”) = [4,5]
lookup(4) = 2, lookup(5) = 5
text(6,8) = “agg”
child_l(1,3) = {c}
child_l(4,5) = {a, g}

文書データ

NTCIR-4 PATENT (日本語特許公報全文　1993-1997)
文書数 3,496,252

サイズ (タグ除去, utf8)	113.8G
bzip2での圧縮サイズ	15.2G
圧縮索引のサイズ	21.6G
従来の索引 (接尾辞配列)	680.4G

検索処理
文書中の頻出パタンの列挙　　36秒
(文字列長の0.01%以上の頻度)
出力例
[99828894586,99842145324] key = 図である。
[89135087764,89147590131] key = ることを特徴とする
[82281197875,82293217686] key = することができる。

ソースコード (Ruby)

```
require 'csa'
require 'benchmark'

def traverse(csa, range, th, key)
#  print("traverse [", range[0], ",", range[1], "] key = ", key, "¥n")
  f = 0
  csa.child_l(range).each {|c, range2|
    num = range2[1] - range2[0] + 1
    if num >= th then
      newkey = "%c" % c;
      newkey += key
      traverse(csa, range2, th, newkey)
      f = 1
    end
  }
  if f == 0 then
    print("[", range[0], ",", range[1], "] key = ", key, "¥n")
  end
end

csa = CSA.new(ARGV[0],ARGV[1])

n = csa.getn()
th = n / 10000

puts Benchmark.measure{
  traverse(csa, [0, n], th, "")
}
```