大規模コーパスを扱うためのツール群

岡野原　大輔　辻井　潤一　東京大学

- Google Nグラムコーパスや Wikipediaなど大規模コーパスを利用した自然言語処理を支援するためのツールを開発した
- tx：木の簡潔表現を利用したtrie
- bep：最小完全ハッシュ関数による連想配列
- oll：複数のオンライン学習手法を備えた学習器
- データ構造や学習手法の最新の研究成果を取り入れ言語処理向けに最適化
- 全て修正BSDライセンスでダウンロード可能
  - 複数の企業や研究所での利用実績有

準備：Rank/Select辞書
ビット配列に対する簡潔データ構造

- ビット配列 B[0…n-1]に対し，次の二つの操作
  - $\mathrm{rank}_b(B,i)$: B[0…i]中のb∈{0,1}の数を返す
  - $\mathrm{select}_b(B,i)$: (i+1)番目のb∈{0,1}の位置を返す
- 例

	0	1	2	3	4	5	6	7	8	9	10
B	1	0	1	1	0	1	1	1	0	1	1

$\mathrm{rank}_1(B, 6) = 5$　　$\mathrm{select}_0(B, 1) = 4$

- Rank/Select辞書は$nH_0+o(n)$ bitsの作業領域量で定数時間で実現可能 実装例などは[Okanohara+ 07]

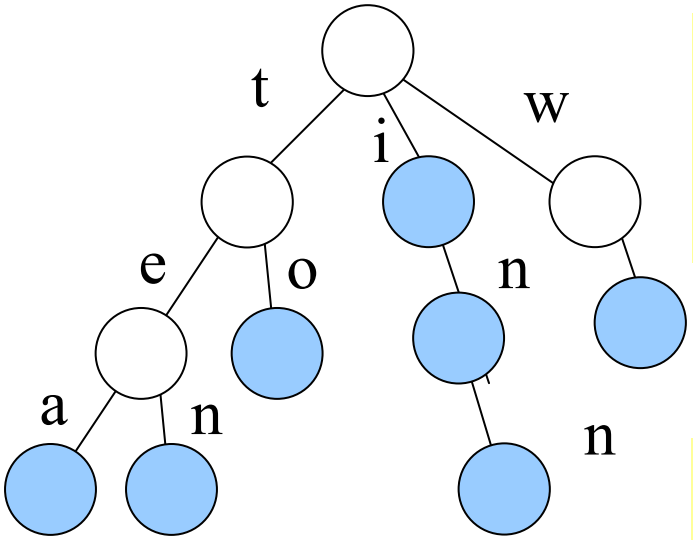
tx : 簡潔表現を用いたコンパクトなtrie

- 大規模キー集合を扱うためのライブラリ
- 与えれたクエリ**q**に対し次をサポート
  - lookup: **q**が辞書に含まれているか
  - predictive search : **q**を接頭辞として含むキーが辞書に含まれているか
  - common prefix search : **q**の接頭辞からなるキーが辞書に含まれているか

q=”東京”でpredictive search ⇒
“東京都” “東京都庁” “東京ドーム”…
q=“熱海温泉は日本三大温泉”でcommon prefix search ⇒
“熱海” “熱海温泉”

trieのLOUDS簡潔表現

- trieはキー集合を木構造で管理
  - 各枝に文字が付随し、つなげるとキーになる



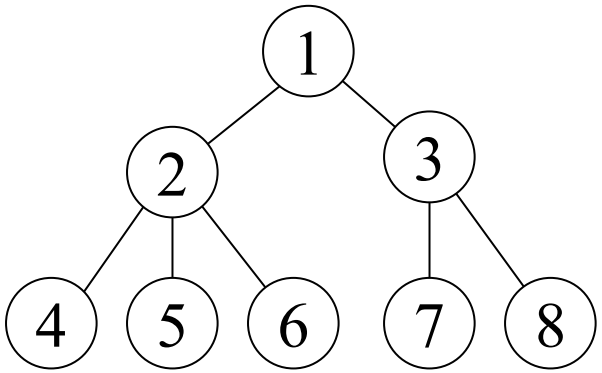
木のポインタ表現は1ノードあたり
約96bit必要
- 親、最初の子、次の兄弟を指す場合

LOUDS表現は1ノードあたり約2bit！
- 木を辿る操作は定数時間

キー : "to", "tea", "ten", "i",
"in", "inn“, and “we”.

LOUDS: Level-Ordered Unary Degree Sequence [Jacobson 89, O'Neil Delpratt 06]

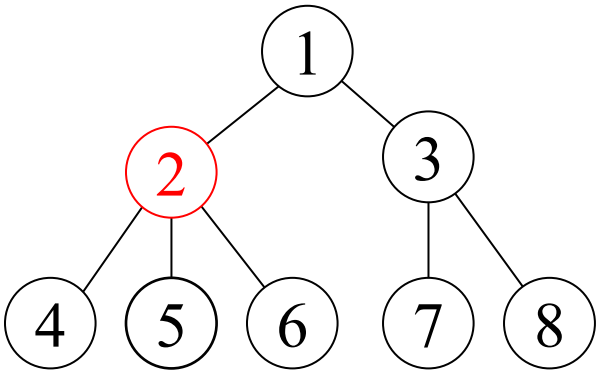
- 木をBFSで辿り，k次のノードを行きがけに k個の'**1**'と1個の'**0**'で表す
  - 先頭にSuper Root S（番人）を追加
  - n個の節点からなる順序木にはn個の**1**, n+1個の**0**が出現する
  - Bi番目の節点とi番目の1, (i+1)番目のoが対応

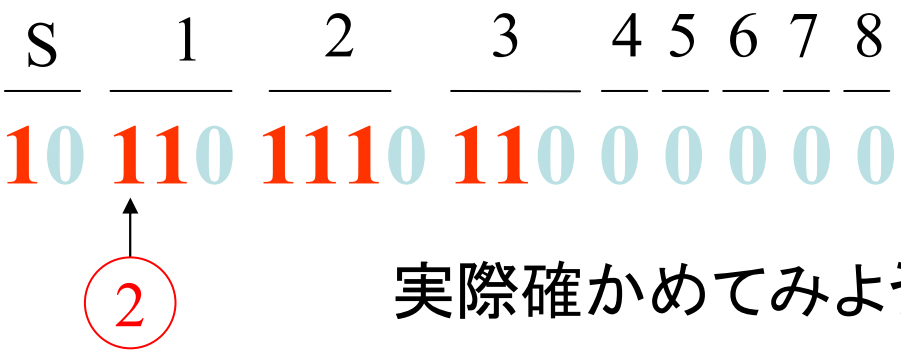


S	1	2	3	4	5	6	7	8
10	**1**1**0**	**1**1**1**0	**1**1**0**	0	0	0	0	0

LOUDS上での操作

- B[i]に対応するノードに対する各操作
  - 最初の子 = $\mathrm{select}_0(B, \mathrm{rank}_1(B,i)-1) + 1$
  - 最後の子 = $\mathrm{select}_0(B, \mathrm{rank}_1(B,i)) - 1$
  - 親 = $\mathrm{select}_1(B, \mathrm{rank}_0(B,i)-1)$
  - 次の兄弟 = i+1





実際確かめてみよう

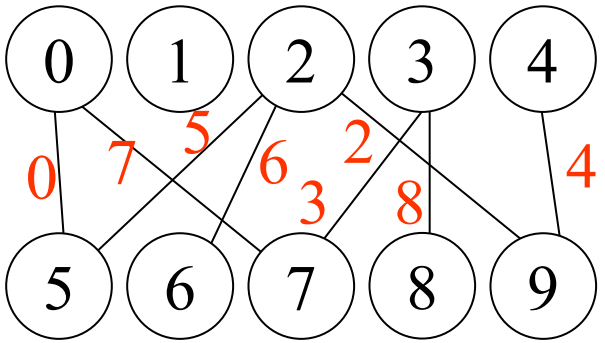
枝に付随する文字情報，各節点に付随するデータも配列で保存可能

bep : 最小完全ハッシュ関数による連想配列

- 大規模キーを扱うための連想配列
  - lookupのみサポートしキー順序は保存しない
- 最小完全ハッシュ関数を利用
  - PHF: キー集合Sに対する完全ハッシュ関数
    ⇔S中のキー同士は衝突しないことが保障
  - MPHF: Sに対する最小完全ハッシュ関数
    ⇔ハッシュ関数が完全かつ、ハッシュ値の範囲が
    [0, |S|-1]　（重複の無い連番を返す）
  - 単純なアプローチでは作れない
    - n個のキーで偶然できる確率は $n! / n^n \fallingdotseq e^{-n}$

PHFの構築手法
[Ziviani+, WADS 07]

- n個のキー k_1 k_2… k_nに対し，[0..m-1] (n≦m)を値域とするPHFを構築する
- 二つの独立なハッシュ関数h_1, h_2を用意
  - h_1は[0..m/2) に，　h_2は[m/2 .. m)に写像するようなハッシュ関数
- グラフG={V,E}を考える.
  - 頂点 V = {0,…,m-1}
  - 枝　$e_i = (h_1(k_i), h_2(k_i))$ (i=1…n)
  - 2部グラフであり、枝の数はn個

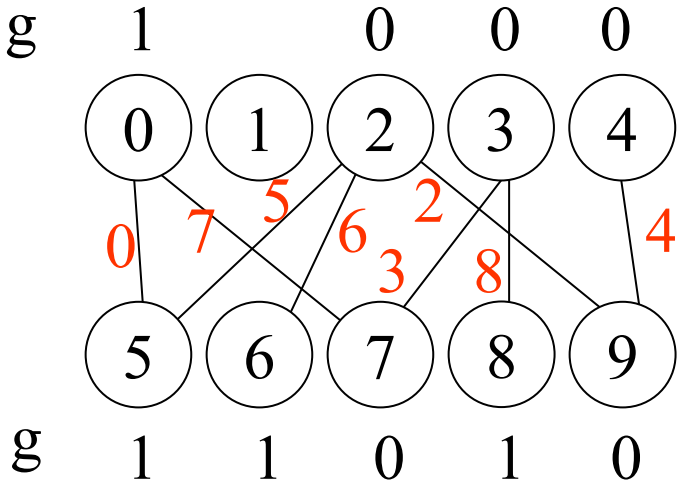


PHFの構築手法 (続)

- 各枝につき，頂点を1つずつ割り当てる
  - この時，割り当てた頂点が重複しないようにする．グラフGに閉路が無いなら可能
  - 閉路があったらハッシュを作り直す
  - 各頂点vに番号g(v)∈{0,1}を与え，割り当てられるようにする
- PHF: pは次の通り

$$p(k) = h_i(k)$$

$$i = g(h_0(k)) + g(h_1(k)) \bmod 2$$



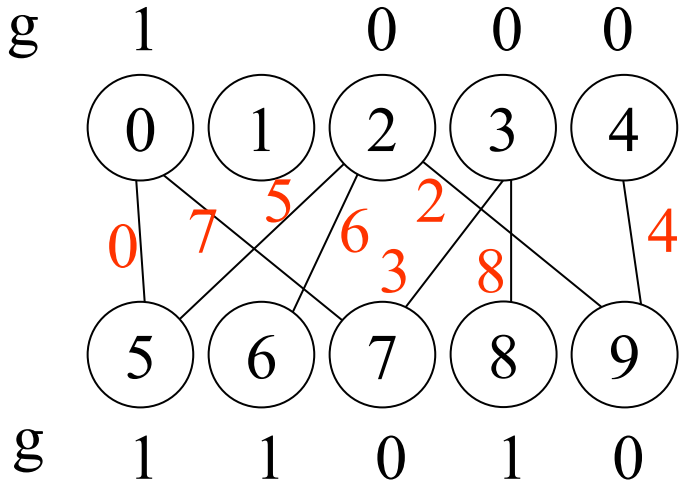
PHFからMPHF

- 今回構築したPHFは既にかなり密
  - m=1.23n, 値域のうち1/1.23はキー
- rank関数を利用しMPHFにする
  - B[0…m-1]：B[i]=1: iにキーがある

$$p(k) = rank_1(B, h_i(k))$$

$$i = g(h_0(k)) + g(h_1(k)) \bmod 2$$

例 キー k が $h_0(k)=2$, $h_1(k)=6$の場合
$i = g(2)+g(6) = 0+1 = 1$
$p(k)=6, rank_1(B, 6) = 5$



OLL: オンライン学習ライブラリ

- 様々なオンライン学習手法をサポートした機械学習ライブラリ
- バッチ学習の10倍～1000倍で学習可能
  - データを一度しか見ないストリーミングも可能
- 次の学習手法をサポート
  - Perceptron [F. Rosenblatt, 58]
  - Averaged Perceptron [M. Collins 02]
  - Passive-Aggressive (PA, PA-I, PAII)[K. Crammer 06]
  - Confidence-Weighted [M. Dredze　08]

オンライン学習

- 入力$\Phi(x) \in R^m$から出力$y \in \{-1,1\}$を学習
- 全ての手法は線形識別器を利用
  - パラメータは重みベクトル$\mathbf{w} \in R^m$
  - $\mathbf{w}^T\Phi(x)$を推定結果として利用
- 毎回、一つの訓練例(x,y)に対し$\mathbf{w}$を更新
  - 全ての訓練例を保存する必要はない
  - 間違えた時、または正解していても$\mathbf{w}^T\Phi(x)$が小さい場合、$\mathbf{w}$を更新

入力(x,y)が与えられた時の更新式

手法	更新条件	更新式	分類式
Perceptron [7]	$s < 0$	$\boldsymbol{w} += y\phi(x)$	$\boldsymbol{w}^T\phi(x)$
Averaged Perceptron [8]	$s < 0$	$\boldsymbol{w} += y\phi(x) \quad \boldsymbol{w}_a += y\frac{\phi(x)}{t}$	$(\boldsymbol{w} - \frac{\boldsymbol{w}_a}{t})^T\phi(x)$
PA [9]	$s < 1$	$\boldsymbol{w} += y\frac{1-s}{\lvert\phi(x)\rvert}\phi(x)$	$\boldsymbol{w}^T\phi(x)$
PA-I [9]	$s < 1$	$\boldsymbol{w} += y\min(C, \frac{1-s}{\lvert\phi(x)\rvert}\phi(x))$	$\boldsymbol{w}^T\phi(x)$
PA-II [9]	$s < 1$	$\boldsymbol{w} += y\frac{1-s}{\lvert\phi(x)\rvert+2C}\phi(x)$	$\boldsymbol{w}^T\phi(x)$
CW [10]	$\gamma > 0$	$\boldsymbol{w} += y\gamma\Sigma\phi(x) \quad \Sigma^{-1} += 2\gamma C\boldsymbol{diag}(\phi(x))$	$\boldsymbol{w}^T\phi(x)$

- どの手法もほぼ同じフレームワークで扱える
  - 目的関数はそれぞれ全く違う
- $s = y\mathbf{w}^T\Phi(x)$
  - 正解した時は正、不正解の時は負の値
- t それまでに観測した訓練例数
- γ 分散も考慮した更新幅
  - 詳細は[M. Dredze+, ICML 08]

$$\gamma_i = \frac{-(1+2\phi M_i) + \sqrt{(1+2\phi M_i)^2 - 8\phi(M_i - \phi V_i)}}{4\phi V_i}$$

実装手法	入力サイズ比 (1≒100MB)	構築時間 (秒)	lookup 10^6回, 秒	complex search*
tx	0.46	30.5	10.3	○
bep	2.03	44.5	0.49	×
bep(キー無し)	0.07	44.5	0.46	×
darts	3.62	16.7	1.25	○
stl::set	8.86	39.5	3.30	○
stl::hash_set	8.18	32.4	0.54	○

*common prefix searchなどをサポートしているか

- 実験データは Web 1T 5-gram Version 1 の 1-gram
  - キーワード種類数は13588391
  - キーワードを全てつなげた時の総長は112349445 (約100MB)
- darts: double arrayによりtrieを実装したライブラリ
- stl::set stl::hash_setはC++ STLライブラリ
- tx, bepともにに非常にコンパクトにキーを管理可能であった

学習の繰り返し回数が10回の時（SVMを除く）

手法	P	AP	PA	PA-I	PA-II	CW	SVM
学習時間 (s)	0.54	0.56	0.58	0.59	0.60	1.39	1122.6
正解精度 (%)	94.7	95.3	96.5	96.5	96.5	96.5	96.2

学習の繰り返し回数が1回の時

手法	P	AP	PA	PA-I	PA-II	CW
学習時間 (s)	0.05	0.09	0.07	0.08	0.08	0.21
正解精度 (%)	93.4	94.0	96.1	96.1	95.9	96.4

- テストデータとしてnews20.binaryを利用
  - クラス数2 データ数19996 素性種類数約135万
  - シャッフルし、3:1の訓練データ、テストデータを作成
- 多くの手法では訓練例一つあたり約３マイクロ秒
- データを一度しか見ない場合も非常に高精度であった