Double Array Trie 木構造機概要書

作成者	作成日	バージョン
丁志剛	2007/09/01	0.1

目次

１、概要……3
２、トライ木（Trie）とは……3
３、基本構造……4
・構造体定義……4
・UTF8 コードテーブル……5
・データタイプ定義……5
・文字のＡＮＳＩ／ＵＴＦ８数値を取得する……6
・ＤＦＡ構造図……7
・ＤＦＡ構造テーブル……8
・Double Array Trie 木を構造する必須条件……9
・Double Array Trie 木探索のＣ＋＋実現……9
・Double Array Trie 木構造結果……12
・Double Array Trie 辞書構造スクリーンショット……13
４、利点と欠点……16
・利点……16
・欠点……17
５、附録……18
・形態素解析……18

テーマ:	作成日	作成者
Double Array Trie 木構造機概要書	09-01-2007	丁　志剛

１、概要

下記の Double Array Trie 構造プログラムは Minidx2.0 の一部である

２、　トライ木（Trie）とは

順序付き木構造 (データ構造)の一種。プレフィックス木（Prefix Tree）とも呼ばれる。キーが文字列である連想配列の実装構造として使われる。Trie という名称は "retrieval"（探索、検索）が語源であるため、"tree" と同じ発音を用いる（リトゥリーヴァル)。しかし、ツリー構造との混同を避けるために「トライ」という読み方を奨励する人もいる。日本語では、「トライ木」という呼び方がほぼ定着している。Ｔｒｉｅの本質はＤＦＡ[注①]である。

注①:「ＤＦＡ」 決定性有限オートマトン（けっていせい-、Deterministic Finite Automaton）または決定性有限状態機械（けっていせいゆうげんじょうたいきかい、Deterministic Finite State Machine）は、状態と入力によって次に遷移すべき状態が一意に定まる有限オートマトンである。DFA と略記される。

３、基本構造

- 構造体定義

```
struct ST_STATE
{
    uint16_t code;
    uint16_t final;
    uint32_t state;
};

struct ST_TRIE_ITEM
{
    int32_t base;
    int32_t check;
    int32_t handle;
};
struct ST_STAT_BASE
{
    int32_t offset;
    uint32_t state;
    uint16_t final;
    uint16_t child_count;

    ST_STAT_BASE(){}
    ST_STAT_BASE(int32_t o, uint16_t f, uint32_t t, int c) :
        offset(o),
        final(f),
        state(t),
        child_count(c),
        next(NULL)
           {
           }
           ST_STAT_BASE *next;
};
```

◆ UTF8 コードテーブル:

U+00000000 - U+0000007F:	0 xxxxxxx	0x - 7x
U+00000080 - U+000007FF:	110 xxxxx　10 xxxxxx	Cx 8x - Dx Bx
U+00000800 - U+0000FFFF:	1110 xxxx　10 xxxxxx　10 xxxxxx	Ex 8x 8x - Ex Bx Bx
U+00010000 - U+001FFFFF:	11110 xxx　10 xxxxxx　10 xxxxxx　10 xxxxxx	F0 8x 8x 8x - F7 Bx Bx Bx
U+00200000 - U+03FFFFFF:	111110 xx　10 xxxxxx　10 xxxxxx　10 xxxxxx　10 xxxxxx	F8 8x 8x 8x 8x - FB Bx Bx Bx Bx
U+04000000 - U+7FFFFFFF:	1111110 x　10 xxxxxx　10 xxxxxx　10 xxxxxx　10 xxxxxx　10 xxxxxx	FC 8x 8x 8x 8x 8x - FD Bx Bx Bx Bx Bx

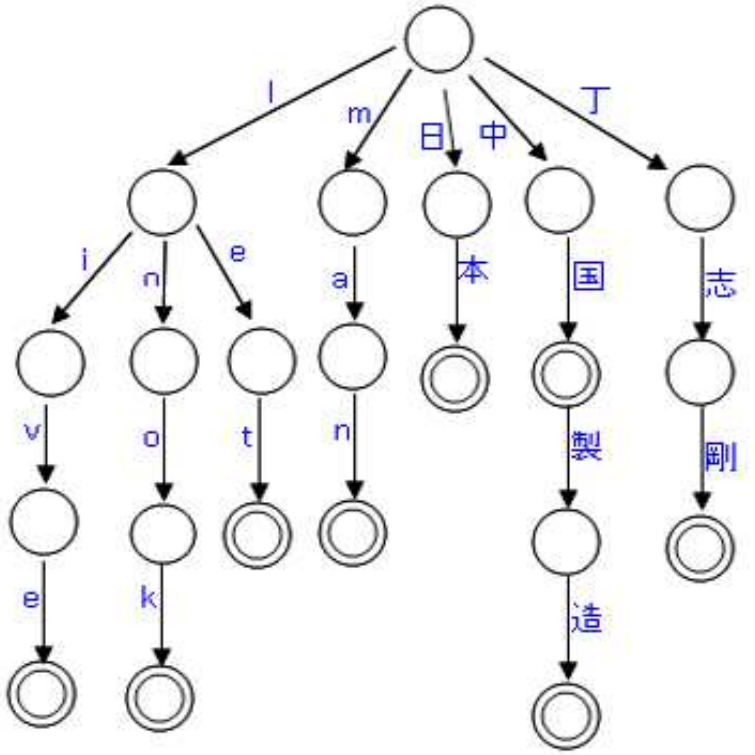
◆ タイプ定義：

```
#ifndef WIN32
        #include <inttypes.h>
        typedef uint8_t                 byte;
#else   //for Win32
        #include <sys/types.h>
        #include <wchar.h>
        typedef signed char             int8_t;
        typedef short                   int16_t;
        typedef long                    int32_t;
        typedef __int64                 int64_t;
        typedef unsigned char           uint8_t;
        typedef unsigned short          uint16_t;
        typedef unsigned long           uint32_t;
        typedef unsigned __int64        uint64_t;
        typedef unsigned char           byte;
        typedef __int64                 intmax_t;
        typedef unsigned __int64        uintmax_t;
#endif
```

◆　文字のＡＮＳＩ／ＵＴＦ８数値を取得する

```
//uint16_t    slnput;
//const char    *plnput = NULL;
switch (MINIDX_DEF_ENCODE)
{
case MINIDX_DEF_ENCODE_ANSI:    // ANSIモード
    if ( (*plnput < 0) && *(plnput+1) )
    {
        slnput = ((byte)(*(plnput++)))<<8;
        slnput |= (byte)(*(plnput++));
    }
    else
    {
        slnput = (byte)*(plnput++);
    }
    break;
case MINIDX_DEF_ENCODE_UTF8:    // UTF8モード
    if (((*(byte *)plnput) & 0x80) == 0) {
        slnput = *(byte *)plnput;
        plnput += 1;
    } else if (((*(byte *)plnput) & 0xe0) == 0xc0) {
        slnput = ((*(byte *)plnput & 0x1f) << 6) + (*(byte *)(plnput + 1) & 0x3f);
        plnput += 2;
    } else if (((*(byte *)plnput) & 0xf0) == 0xe0) {
        slnput = ((*(byte *)plnput & 0x0f) << 12) + ((*(byte *)(plnput + 1) & 0x3f) << 6) + (*(byte
*)(plnput + 2) & 0x3f);
        plnput += 3;
    }
    break;
default:
    //
    break;
}
```

◆　ＤＦＡ構造図



単語：

live	日本
look	中国
let	中国製造
man	丁志剛

◆　ＤＦＡ構造テーブル

character	code	final	state	count	charCode
l	108	0	1	1	1
i	105	0	2	1	2
v	118	0	3	1	3
e	101	1	4	1	4
l	×	－	－	－	－
o	111	0	5	2	5
o	111	0	6	1	5
k	107	1	7	1	6
l	×	－	－	－	－
e	101	0	8	3	4
t	116	1	9	1	7
m	109	0	10	2	8
a	97	0	11	1	9
n	110	1	12	1	10
日	26085	0	13	3	11
本	26412	1	14	1	12
中	20013	0	15	4	13
国	22269	1	16	1	14
中	×	－	－	－	－
国	×	－	－	－	－
製	35069	0	17	1	15
造	36896	1	18	1	16
丁	19969	0	19	5	17
志	24535	0	20	1	18
剛	21083	1	21	1	19

- ＤＦＡから Double Array Trie 木を構造するのは、下記の条件は必須である
  - base[s] + c = t
  - check[t] = s

- DAT の検索
  - Step1:「c」を入力する
  - Step2:

    t = base[s] + c

    if (check[t] == s) then

        next state: = t

    else

        fail

    endif
  - Step3:

    if(base[t] > 0)　then

        goto Step1

    else

        end state

下記は DAT 検索のＣ＋＋実現：

```
//
// @param pcszWord: key word(utf8)
// @return bool
//
bool DATCreator::IsWord(const char* pcszWord)
{
    int i = 0;
    int nPos;
    int32_t base=1,check=0;
    uint16_t input;

    while(*pcszWord)
    {
        // 文字のＵＴＦ８数値を取得する
        switch (MINIDX_DEF_ENCODE)
        {
        case MINIDX_DEF_ENCODE_ANSI:
```

```
            if ( (pcszWord[i] < 0) )
            {
                input = ((byte)pcszWord[i] << 8) | (byte)pcszWord[i+1];
                ++i;
            }
            else
            {
                input = (byte)pcszWord[i];
            }
            break;
        case MINIDX_DEF_ENCODE_UTF8:
            if (((*(byte *)pcszWord) & 0x80) == 0) {
                input = *(byte *)pcszWord;
                pcszWord += 1;
                i += 1;
            } else if (((*(byte *)pcszWord) & 0xe0) == 0xc0) {
                input = ((*(byte *)pcszWord & 0x1f) << 6) + (*(byte *)(pcszWord + 1) & 0x3f);
                pcszWord += 2;
                i += 2;
            } else if (((*(byte *)pcszWord) & 0xf0) == 0xe0) {
                input = ((*(byte *)pcszWord & 0x0f) << 12) + ((*(byte *)(pcszWord + 1) & 0x3f) << 6)
+ (*(byte *)(pcszWord + 2) & 0x3f);
                pcszWord += 3;
                i += 3;
            }
            break;
        default:
            //
            break;
        }
        // 存在しない
        if ( !m_charCode[input] )
        {
            return false;
        }
        // inputが存在する場合、遷移位置を計算する
        nPos=base+m_charCode[input];
        //  配列位置範囲外、ＤＡＴ遷移条件が満足していなければ存在しない
```

```
        if (nPos>m_nTotal||m_pTrie[nPos].check!=check)
        {
            return false;
        }
        //  上記の条件を満足する場合、ＤＡＴ遷移する
        check = nPos;

        base = m_pTrie[nPos].base & (~MINIDX_DEF_FINAL_TAG);
    }
    //  存在する場合
    if (nPos<=m_nTotal && (m_pTrie[nPos].base & MINIDX_DEF_FINAL_TAG))
    {
        return true;
    }
    //  ここまで実行すれば存在しない
    return false;
}
```

◆ DAT 構造結果

base[]={1,0,6,0,0,0,0,0,19,6,14,8,4,13,3,15,-1,-214783635,6,2,-1,12,-1,19,-1,0,-1,0,0,14,0,-1,-1}

check[]={-1,0,0, 0, 0, 0, 0, 0,2,0,2,2,0,11,0,9,12,14,0,13,18,10,8,22,0,15,0,0,17,0,28,20}

※ -214783635=13 | MINIDX_DEF_FINAL_TAG; 13 は元 base 値、MINIDX_DEF_FINAL_TAG= 0x80000000

No.	base	check	handle
1	1	-1	0
2	0	0	0
3	6	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	19	0	0
9	6	2	0
10	14	0	0
11	8	2	0
12	4	2	0
13	13	0	0
14	3	11	0
15	15	0	0
16	-1	9	0
17	-214783635	12	5
18	6	14	6
19	2	0	0
20	-1	13	2
21	12	18	0
22	-1	10	3
23	19	8	0
24	-1	22	1
25	0	0	0
26	-1	15	4
27	0	0	0
28	0	0	0
29	14	17	0
30	0	0	0
31	-1	28	7
32	-1	20	8

◆ DAT 辞書構造スクリーンショット


Double Array Trie(DAT)辞書構造
辞書作成
オリジナル辞書ファイル
C:¥MinidxSrc¥Minidx2.0¥core¥bin_release¥dict¥src.txt
ブラウザ...
ターゲット辞書パス:
C:¥MinidxSrc¥Minidx2.0¥core¥bin_release¥dict¥MinidxDictA.dat
ブラウザ...
パラメータ設定(パラメータによって、生成した辞書サイズが違う)
キーの最大衝突数:
500
衝突後baseの値上:
3
辞書作成
辞書検索
検索単語:
汽车
検索
辞書に存在しています。word ID:[283264]
形態素解析
強い台風9号は6日午後11時現在、静岡県・石廊崎付近の海上を時速約20キロで北へ進んでおり、東京都や神奈川県、房総半島の南部などが暴風域に入った。
7日午前3時ごろまでに静岡県から神奈川県の沿岸に上陸し、東日本を北上する見通しで、気象庁は大雨や暴風、高波に厳重な警戒を呼び掛けている。
解析結果:
強い/台風/9/号/は/6/日/午後/1/1/時/現在/、/静岡/県/・/石廊崎/付近/の/海上/を/時速/約/2/0/キロ/で/北/へ/進/んで/おり/、/東京/都/や/神奈川/県/、/房総半島/の/南部/など/が/暴風/域/に/入/っ/た/。/// /7/日/午前/3/時/ごろ/まで/に/静岡/県/から/神奈川/県/の/沿岸/に/上陸/し/、/東日本/を/北上/する/見通し/で/、/気象庁/は/大雨/や/暴風/、/高波/に/厳重/な/警戒/を/呼び掛け/てい/る/。/
形態素解析

（日本語解析）


Double Array Trie(DAT)辞書構造
辞書作成
オリジナル辞書ファイル
C:¥MinidxSrc¥Minidx2.0¥core¥bin_release¥dict¥src.txt
ブラウザ...
ターゲット辞書パス:
C:¥MinidxSrc¥Minidx2.0¥core¥bin_release¥dict¥MinidxDictA.dat
ブラウザ...
パラメータ設定(パラメータによって、生成した辞書サイズが違う)
キーの最大衝突数:
500
衝突後baseの値上:
3
辞書作成
辞書検索
検索単語:
汽车
検索
辞書に存在しています。word ID:[283264]
形態素解析
该新规名为《节能环保型小排量汽车技术条件》，是上海地方标准，由上海市经委牵头，上海市汽车行业协会、上海市技术监督局等多个部门共同制定，目前已进入了国家质检总局的备案程序。该标准不仅规定了八大指标突出节能、环保和安全，而且还要求排放标准、排放限值、燃料消耗限值和噪声限值等达到国家最新的标准。记者获悉，该标准将在两个月后出台并正式实施，并可能在全国推广
解析結果:
名为/《/节能/环保/型/小排/量/汽车/技术/条件/》/，/是/上海/地方/标准/，/由/上海市/经委/牵头/，/上海市/汽车/行业/协会/、/上海市/技术监督局/等/多/个/部门/共同/制定/，/目前/已/进入/了/国家/质检/总局/的/备案/程序/。/该/标准/不仅/规定/了/八大/指标/突出/节能/、/环保/和/安全/，/而且/还要/求/排放/标准/、/排放/限/值/、/燃料/消耗/限/值/和/噪声/限/值/等/达到/国家/最新/的/标准/。/记者/获悉/，/该/标准/将在/两/个月/后/出台/并/正式/实施/，/并可/能在/全国/推广
形態素解析

（中国語解析）

4、利点と欠点

利点：

- キー検索が高速である

長さ m のキー検索は最悪で O(m) の時間がかかる。2 分探索木では O(log n) の時間であり、n は木を構成するノード数である（木の深さに応じた時間がかかり、2 分探索木の深さは n の対数となる)。トライ木が検索処理で行う文字でインデックス付けした配列の操作なども、実際のマシンでは高速である。

- 多数の短い文字列を格納する場合にはトライ木の方がメモリを節約できる

これはキーが明示的に格納されないためであり、複数のキーによってノードが共有されるためである。

- 辞書引き終了のタイミングが自動的にわかる

欠点：

トライ木はキーの順序を与えるが、それは辞書式順序でなければならない。

- トライ木は状況によっては極めて巨大になる。例えば、少数の非常に長い文字列を格納するトライ木などである（この場合はパトリシア木が適している)。

- トライ木のアルゴリズムは単純な 2 分探索木よりも複雑である。

- データを文字列として表すのは常に簡単とは言えない。例えば、複雑なデータ構造や浮動小数点数などをキーとする場合、工夫が必要となる。

５、附録

形態素解析

（http://ja.wikipedia.org/wiki/%E5%BD%A2%E6%85%8B%E7%B4%A0%E8%A7%A3%E6%9E%90）

形態素解析（けいたいそかいせき、*Morphological Analysis*）とは、コンピュータ等の計算機を用いた自然言語処理の基礎技術のひとつ。かな漢字変換等にも応用されている。

対象言語の文法の知識（文法のルールの集まり）や辞書（品詞等の情報付きの単語リスト）を情報源として用い、自然言語で書かれた文を形態素（Morpheme，おおまかにいえば、言語で意味を持つ最小単位）の列に分割し、それぞれの品詞を判別する作業を指す。

以下は「お待ちしております」という文を形態素解析した例である（形態素解析ツールには「茶筌」を使用した）。

文字列	**読み**	**原形**	**品詞の種類**	**活用の種類**	**活用形**
お待ち	オマチ	お待ち	名詞-サ変接続		
し	シ	する	動詞-自立	サ変・スル	連用形
て	テ	て	助詞-接続助詞		
おり	オリ	おる	動詞-非自立	五段・ラ行	連用形
ます	マス	ます	助動詞	特殊・マス	基本形
。	。	。	記号-句点		