チベット語のソートキー生成ソフトについて

田浦宏太郎

目　次

——チベット語のソートキー生成ソフトについて——

1　序論

(1) 何を作るか

私が所属するゼミのテーマは「人の役に立つ Web アプリケーションを作成する」というものである。そこでテーマに沿った内容を考えた際に、福田先生の助言もあり先生の研究分野であるチベットに関連して、チベットに関するデータベースで使用するためのソートキーを生成するプログラムを作ることとなった。そうして私の卒業論文のテーマは「チベット語のソートキー生成ソフトについて」となった。

(2) 誰のために作るか

この制作物はすべてのチベット研究者を対象に作成している。「単語を辞書の順番に並び替えたい」「ファイル名を並び替えたい」という風に思った際に、制作物を使用することでチベット文字列をソート文字列に変換しソートできるように作成した。そうすることで、少しでも研究の役に立てればというのが私の思いである。

(3) これまでに似たようなものがあったのか、なかったのか

私が制作するよりもずっと以前に福田先生が C 言語を使用して、ソートするものを作成していた。しかしそのプログラムは unicode(1)という文字コードに対応していなかったことや、特定のデータベースで使用するものであったことなどから、あらゆる場面で使えるソートプログラムを新しく作った方が良いのではないか、ということで python(2)を使用して新しくソートプログラムを制作することとなった。

2　企画

(1) 構想や規模

この制作物の使用方法として、主に以下の三つを想定している。

- コマンドラインからチベット文字を直接入力し、プログラムを実行することでソートキーを作成する方法
- コマンドラインでファイルを指定し読み込ませプログラムを実行することで、そのファイルの中のチベット文字をソートキーに変換する方法
- PHP[3]などの他の言語の中に組み込んでプログラムを実行し、変数にソート文字列を返す方法

プログラムの構想として、プログラムを実行することで元の文字列からソートキーを生成するというものを考えていた。生成する時は、元の文字列は書き出さずソートキーのみ書き出すという構図を想定した。

制作物の規模を考える際、まずはチベット語に対応させ、ソートキーを作成できるようにすることを目標にした。そしてチベット語が対応した後、そこから派生して、サンスクリット語に対応させることとした。サンスクリット語とはインドの古語であり、チベット語と文字の組み合わせ方が違う。そのため、プログラムをチベット語のものとは別に追加する必要がある。チベット語だけでなく、サンスクリット語に対応させる理由として、このサンスクリット語はチベット文献の文章中にチベット語と一緒に使われていることがあるため、サンスクリット語に対応させることが制作物の完成には欠かせないのである。また、チベット語の後にサンスクリット語に対応させる理由は、まずチベット語に対応させることで中心となるものを作成し、それが動くことを確認してから機能を拡張する手順を踏むことで円滑に制作が進むと考えたからである。

(2) 利用環境

今回、制作に使用した言語は python である。python を使用する理由として、多くのプラットフォーム(4)で動作することを挙げられる。

python が動作するプラットフォームは、

・Windows, Windows CE（9x 系および NT 系は最新版、Windows 3.1 および MS-DOS は旧版のみ）

・Macintosh（OS 9 以前および OS X 以降ともに）

・各種 UNIX

・Linux（Linux Standard Base3.2 で標準仕様となった）

・Java プラットフォーム (Jython)

・.NET Framework プラットフォーム (IronPython)

などがある。このように多くのプラットフォームで動作するため、様々な環境で使うことができると判断し、python を使用することとした。

(3) 必要な技術

今回の制作には python を使用している。python は、システム管理からデスクトップアプリケーション、サーバアプリケーションまで、幅広く応用可能なプログラミング言語である。様々な分野で高く評価されており、python が実用的で生産性に優れたプログラミング言語であることを実証している。また、python は非常に習得しやすい言語で初心者が学習するのに最適な言語であり、福田ゼミの授業でも扱ったことがある。さらに、今回の制作ではチベット文字を、「正規表現」を用いて取り出すため unicode による文字列操作をサポートしている python を使用するのが最適だと判断した。

制作において、python での関数定義や for 文、if 文の技術が必要であ

る。しかし、この２点の技術に関しては授業で取り扱ったことがあるため、ゼロからのスタートではなかった。なので、２点の技術を扱うのに問題はなかった。その他の技術の中で、最も重要な技術に「正規表現」を挙げられる。正規表現は、文字列のパターンを表現する表記法で、文字列の検索・置換を行うときに利用されている。これを使うことで、文字列を直接指定せず、パターンを指定して検索することができるため、チベット文字の検索・置換を容易に行うことができる。つまりこの正規表現の操作を正確にしなければ、正しく文字列を取ることができず制作が完成しないため、重要な技術であると考えられる。

正規表現操作の中でも、「re.compile()」という操作が重要であると私は考える。re.compile() は正規表現パターンを正規表現オブジェクトにコンパイルするものである。このオブジェクトは、match() や search() メソッドを使って、マッチングに使うことができる。つまり、「変数名=re.compile(パターン)」とすることで、正規表現オブジェクトに変換されたチベット文字が変数の中に代入されるのである。そうすることで match() や search() メソッドを使う際に何度もパターンを入力せずに、変数を入力するだけで済むのである。また、この操作をすることでプログラムのソースが見やすくなるので、やはり制作を行う上で必要な技術であると考える。ここで入力するパターンを間違えると正確に文字列を取り出すことができなくなるため、パターンをどのようにするかはよく考えなければいけない。また、こういった正規表現に関する技術を習得するのに、python の日本版の公式サイト[(5)]を利用した。

(4) 準備

制作に取り掛かる前に、私はチベット文字に関する知識が全く無かったため、福田先生から頂いたチベット語のテキストを参考に勉強をした。私が学んだものはチベット語文語で、チベット語には口語もある。しかし文語と口語の違いは単語や文法、言い回しが違うだけであるので、音節の仕組みだけを考慮している今回の制作には、文語と口語の違いは関係がない。

(5) 制作に必要なチベット語の知識

ここでは、制作を行う上で必要なチベット語に関する知識について述べようと思う。

(i) 子音字について

チベット語の基本子音字は３０字母ある。字母とは基本となる文字のことで、それらの文字を組み合わせて１音節を作る。そのとき上下に子音字を積み重ねるときには、文字の形が若干変化する。そのため、３０字母だけではなく、上下に積み重ねるときの変化形も覚える必要がある。

(ii) 音節について

音節は「ツェック」と呼ばれる記号で区切られている。これはちょうど英語のスペースのような役割を果たしている。チベット語の１音節は、基本的には一つの母音の前後に子音がいくつか付け加わってできている。一番多くの文字からなる音節の例は、前置字＋頭＋基字＋足＋母音＋後置字＋再後置字、である。この中の基字と呼ばれる子音字が音節の中心である。辞書は基字の順に配列されているので、どの子音字が基字であるかを判別することは制作には必要である。

(iii) 基字を探す

チベット語には、

・一つの子音に母音記号が付いているときは、その子音字が基字である

・足または頭がついている子音字が基字である

・足・頭・母音記号が重なっていてもその子音字が基字である

などのような決まり事がある。これらの情報を元に基字を探すのである。また、頭につく子音字や足がつく子音字などは限られているため、基字を探す際の目安になる。それらを以下に記す。

(iv) 頭について

頭になることができる子音字は「r, l, s」の三文字に限られ、それらが頭にくる基字もそれぞれ限られている。以下の通りである。

・頭「r」がつきうる子音字は「k, g, ng, j, ny, t, d, n, b, m, ts, dz」の 12 文字である。

・頭「l」がつきうる子音字は「k, g, ng, c, j, t, d, p, b, h」の 10 文字である。

・頭「s」がつきうる子音字は「k, n, ng, ny, t, d, n, p, b, m, ts」の 11 文字である。

(v) 足について

同様に足になることができる子音字は限られており、「y, r, l, w」の四文字である。それらが足が付く基字も頭同様それぞれ限られている。以下の通りである。

・足「y」がつきうる子音字は「k, kh, g, p, ph, b, m」の 7 文字である。

・足「r」がつきうる子音字は「k, kh, g, t, th, d, p, ph, b, m, sh, s,

h」の 13 文字である。

・足「l」がつきうる子音字は「k, g, b, z, r, s」の 6 文字である。

・足「w」がつきうる子音字は「k, kh, g, c, ny, t, d, ts, tsh, zh, z, r, sh, s, h」の 16 文字である。

（vi）頭と足について

頭も足もついている組み合わせも限られている。「rky, rgy, sky, sgy, spy, sby, smy, skr, sgr, spr, sbr, smr」の 12 通りである。三文字のアルファベットの中央の文字が基字である。

（vii）前置字について

基字もしくは頭や足がついた子音字の前に置かれる子音字を前置字といい、前置字になりうる子音字は「g, d, b, m, 'a」の五文字である。そしてそれぞれの前置字の後に置かれる子音字もしくは頭や足がついた子音字も、限られている。以下の通りである。

・前置字「g」がつく子音字は「c, ny, t, d, n, ts, zh, z, y, sh, s」の 11 文字である。

・前置字「d」がつく子音字は「k, g, ng, p, b, m」の 6 文字、
足のついた子音字は「ky, gy, by, my, kr, pr, br」の 7 文字である。

・前置字「b」がつく子音字は「k, g, c, t, d, ts, zh, z, sh, s」の 10 文字、
足のついた子音字は「ky, gy, kr, gr, rl, sl」の 6 文字、
頭のついた子音字は「rk, rg, rng, rj, rny, rt, rd, rn, rts, rdz, lt, sk, sg, sng, sny, st, sd, sn, sts」の 19 文字、
足と頭のついた子音字は「rky, rgy, sky, sgy, skr, sgr」の 6 文字である。

・前置字「m」がつく子音字は「kh, g, ng, ch, j, ny, th, d, n, tsh,

dz」の 11 文字、

足のついた子音字は「khy, gy, khr, gr」の 4 文字である。

・前置字「'a」がつく子音字は「kh, g, ch, j, th, d, ph, b, tsh, dz」の 10 文字、

足のついた子音字は「khy, gy, phy, by, khr, gr, dr, phr, br」9 文字である。

これらは、どれが基字かを判断するために必要な知識である。

(viii) 後置字・再後置字について

基字もしくは頭や足がついた基字の後に置かれる子音字を後置字といい、後置字になりうる子音字は「g, ng, d, n, b, m, 'a, r, l, s」の 10 文字である。これらはどの子音字の後にも置かれる可能性がある。そして、後置字の後にさらに子音字「s」が後置されることがある。これを再後置字という。また、この再後置字「s」は後置字「g, ng, b, m」の後にしかつかない。子音字「s」だけでなく子音字「d」も再後置字になりうる可能性があるが、主に再後置字として置かれるのは「s」であることから、今回の制作では再後置字「d」を正規表現で取り出していない。

3 本論

(1) 制作過程

(i) 作業 1

正規表現の練習として、チベット文字を取り出してみることにした。尚、チベット文字を取り出す時 unicode 文字コードを使用している。unicode 文字コードを用いるために、先生にチベット文字の unicode 文字コード表を頂き、制作の際に使用した。

re_tibetan=re.compile(u"[\u0F40-\u0FBC]")

これは、unicode 文字コードの 0F40 から 0FBC の間のどれか一文字が正規表現オブジェクトに変換され、re_tibetan の中に代入されるというものである。次に、for 文を用いてコマンドラインからファイルを読み込むプログラムをつくる。そして、文字列を書き出すための関数をつくる。以下の通りである。

```
for file_name in sys.argv[1:]:
    for line in open(file_name, ”rU”):
      line=unicode(line, ”utf-8”)
      line=re_tibetan.sub(one_onsetsu, line)
def get_onsetsu(m):
    print m.group(0).encode(”utf-8”)
```

簡単にプログラムの流れを説明すると、コマンドラインのファイルを file_name に代入した後、そのファイルをオープンし、1 行ごと line に代入する。そして変数 line を、コンパイルの際に使用した unicode 文字コードに対応させる。その後.sub() を用いて、指定したチベット文字のパターンにマッチしたものを、関数を使用して書き出している。この一連の流れが、ファイルの中のチベット文字列からソート文字列を作成する流れである。尚今回の制作では、この方法を主に使用して、起動の確認を行った。また、for 文の中で用いられている.sub() メソッドは、文字列を置換する働きをするものである。

ここまでプログラムを作成したところで正常に動くかを試したところ、チベット文字を一文字ずつではあるが、取り出すことができた。しかし最終的には音節を抜き取らなければいけないので、「直前の正規表現の一回以上の繰り返しに一致する」という操作をする「+」を、先ほど指定した正規表現の最後に追加した。すると、

re.compile(u”[\u0F40-\u0FBC]+”)
となる。これで「チベット文字が続く限り」という正規表現になるので、文字列を取り出す際にツェックで区切られ、音節を抜き取ることができるのである。

(ii) 作業２

音節が取り出せるようになり、次に頭のついた子音字を取り出す。前述したように、頭になることができる子音字とそれらが頭にくる子音字は決まっている。なので、頭になることができる文字とそれぞれにつきうる子音字を、正規表現を 3 つに分けて取り出すことにした。以下の通りである。

・頭「r」とそれにつきうる子音字
re.compile(u”\u0F62[\u0F90\u0F92\u0F94\u0F97\u0F99\u0F9F\u0FA1\u0FA3\u0FA6\u0FA8\u0FA9\u0FAB]”,re.X)

・頭「l」とそれにつきうる子音字
re.compile(u”\u0F63[\u0F90\u0F92\u0F94\u0F95\u0F97\u0F9F\u0FA1\u0FA4\u0FA6\u0FB7]”,re.X)

・頭「s」とそれにつきうる子音字
re.compile(u”\u0F66[\u0F90\u0F92\u0F94\u0F99\u0F9F\u0FA1\u0FA3\u0FA4\u0FA6\u0FA8\u0FA9]”,re.X)

以上が頭と頭のついた子音字を取り出すための正規表現である。当初、頭のついた子音字を取り出す際、子音字「k」ならば、0F40 を正規表現に記述していた。しかし、それでは取り出すことができなかったので、unicode 文字コード表を確認したところ、0F90 から 0FBC が頭のついた場合の子音字であることが分かったため、頭がつきうる子音字はそこから記述することにした。そうしてできた正規表現を用いて、頭と頭の

ついた子音字を取り出してみたところ、正しく取り出すことができた。

(iii) 作業３

次に足と足がつきうる子音字を取り出す。頭と同様、足になることができる子音字とそれらがつきうる子音字は決まっている為、足に関しても正規表現を分けて取り出す。足の「ｙ」と「ｒ」を例に挙げると以下のようになる。

・足「y」とそれがつきうる子音字

re.compile(u"[\u0F40\u0F41\u0F42\u0F54\u0F55\u0F56\u0F58]\u0FB1",re.X)

・足「r」とそれがつきうる子音字

re.compile(u"[\u0F40\u0F41\u0F42\u0F4F\u0F50\u0F51\u0F54\u0F55\u0F56\u0F58\u0F64\u0F66\u0F67]\u0FB2",re.X)

以上が足と足がつきうる子音字を取り出すための正規表現である。足を取り出す際に、当初子音字「y」ならば、0F61 を記述していた。しかしそれでは足の「y」として取り出されなかったため、足に関しても頭と同様 0F90 から 0FBC の中から正規表現に記述することにした。そして、できた正規表現を用いて、足と足がついた子音字を取り出してみたところ、正しく取り出すことができた。

(iv) 作業４

次に、頭と足のついた文字を取り出す。頭と足のついた文字列は限られているため、12 通りの組み合わせすべてを一つの正規表現に記述した。以下の通りである。

re.compile(u"\u0F62\u0F90\u0FB1 | \u0F62\u0F92\u0FB1 | \u0F66\u0F90\u0FB1|\u0F66\u0F92\u0FB1|\u0F66\u0FA4\u0FB1|\u0F66\u0FA6\u0FB1|\u0F66\u0FA8\u0FB1|\u0F66\u0F90\u0FB2

|\u0F66\u0F92\u0FB2|\u0F66\u0FA4\u0FB2|\u0F66\u0FA6\u0FB2|\u0F66\u0FA8\u0FB2",re.X) ↓

「|」は「この前後にある正規表現のどちらかと一致する」という正規表現なので、いずれかの組み合わせに HIT した場合のみ文字列を取り出すことになる。この正規表現で取り出しを行ったところ、正しく取り出すことができた。

(v) 作業５

次に、前置字と前置字の後に置かれる子音字もしくは頭や足がついた子音字を取り出す。前置字についても、前置字になりうる子音字や後にくる子音字などは決まっている為、それぞれ正規表現を分けて取り出す。以下の通りである。

・前置字「b」について

・前置字「b」とその後にくる子音字

re.compile(u"\u0F56[\u0F40\u0F42\u0F45\u0F4F\u0F51\u0F59\u0F5E\u0F5F\u0F64\u0F66]",re.X)

・前置字「b」とその後にくる足「y」がつく子音字

re.compile(u"\u0F56[\u0F40\u0F42]\u0FB1",re.X)

・前置字「b」とその後にくる頭「l」がつく子音字

re.compile(u"\u0F56\u0F63\u0F9F",re.X)

・前置字「b」とその後にくる頭と足がつく子音字

re.compile(u"\u0F56(\u0F62\u0F90\u0FB1|\u0F62\u0F92\u0FB1|\u0F66\u0F90\u0FB1|\u0F66\u0F92\u0FB1|\u0F66\u0F90\u0FB2|\u0F66\u0F92\u0FB2)",re.X)

前置字についてすべての正規表現を述べると長くなってしまうため、一部の正規表現を記述している。上記の正規表現で前置字と前置字の後に

置かれる子音字もしくは頭や足がついた子音字を取り出すことができる。しかし、前置字については、文字列を取り出す際に気をつけなければいけないことがある。それは取り出す順番である。つまり前置字のついた子音字の組み合わせのような「複雑な文字列の正規表現」を、基字のみの文字列の正規表現のような、「複雑でない正規表現」よりも先に取り出さなければならないのである。なぜなら、取り出す際に複雑でない文字列の正規表現を、複雑な文字列の正規表現よりも先に記述してしまうと、複雑でない文字列を取り出すことで文字列の取り出しが終了してしまい、複雑な文字列を取り出すことができないということが起こるからである。

文字列を取り出す方法として、for 文の中に

line=re_zen_ashi_y.sub(one_onsetsu, line)

という行を、それぞれの正規表現ごとに書いている。このとき、

line=re_zen_ashi_y.sub(one_onsetsu, line)

line=re_kiji_only.sub(one_onsetsu, line)

というように、複雑な文字列の正規表現を先に記述するのである。こうすることで、複雑な文字列を、複雑でない文字列として取り出してしまうというようなことは起こらなくなるのである。

（vi）作業６

こうして、母音の前の、子音字の組み合わせの正規表現をつくることができた。次は母音と後置字、再後置字を取り出すのだが、その前に、取り出した文字列がどの正規表現によって取り出されたのかを分かるようにする。そうすることで、正規表現が正しく書かれているか、文字列が正しく取り出されているのかを確認することができるからである。

どの正規表現によって取り出されたかを分かるようにするには、書き

出しのための関数を書き換える必要がある。そこで、新たに one_onsetsu という関数をつくり、そこに書くことにした。以下の通りである。

```
def one_onsetsu(m):
    this_onsetsu = m.group()
##
    mx = re_onsetsu_zen_b_atama_r.match(this_onsetsu)
    if mx:
        print mx.group().encode("utf-8") + ": 前 b +頭 r +基+母"
        return ""
```

this_onsetsu に、検索に HIT して取り出された文字列が代入される。そして、.match() を用いて正規表現とマッチさせ、マッチした場合変数 mx に代入され、if 以下のプログラムが実行される。そして、プリントする際に + ": 前 b +頭 r +基+母" とすることで、取り出した文字列の右にどの正規表現によって取り出されたかを分かるようにした。このプログラムをそれぞれの正規表現ごとに追加することで、どの正規表現にマッチしたのかが分かるようになった。

（vii）作業 7

次に母音、後置字、再後置字を取り出す。母音の取り出しに取り掛かった当初、それぞれの正規表現に母音の正規表現を書き加え、正規表現を新たに追加していた。足「y」のつく子音字を取り出す正規表現を例に説明すると、

```
re.compile(u"[\u0F40\u0F41\u0F42\u0F54\u0F55\u0F56\u0F58]
\u0FB1[\u0F72\u0F74\u0F7A\u0F7C\u0F7E]",re.X)
```

このように、正規表現の終わりに母音の正規表現を書き加えるのである。しかし、正規表現ごとに追加していてはあまりにも数が多く

なってしまう。そこで、先生の助言もあり、母音、後置字、再後置字を、母音より前の子音字部分とは別に取り出すことにした。そのためにまず、line=re_onsetsu_zen_b_atama_ashi.sub(one_onsetsu, line) のように子音字の組み合わせごとに文字列を取り出しているのを、line=re_onsetsu.sub(one_onsetsu, line) の音節を取り出す正規表現のみを記述するようにした。そして、音節を取り出し後書き出すための関数 one_onsetsu に、nokori=one_onsetsu[mx.end():] というプログラムを追加する。こうすることで、.match() にマッチした、音節の母音以降の文字列が変数 nokori に代入されるようになる。そして、母音以降の文字列を書き出す関数 boin_kochiji を新たに作成し、母音以降を書き出すことで音節すべてのチベット文字を書き出すことができるようになる。また、boin_kochiji の中で「マップ型オブジェクト」を用いて文字列をソート文字列に書き換えている。boin_kochiji は資料５ページの１６４行から１７４行に書かれている。マップ型オブジェクトは資料４ページ１３３行から１３９行に書かれている。チベット文字を書き換える際、書き換え後のソート文字にはチベット文字を用い、母音はチベット文字の数字に書き換えている。母音、後置字、再後置字は、関数 boin_kochiji の中で正規表現でマッチさせ、マップ型オブジェクトを用いてソート文字列に書き換えている。

（viii）作業８

次に、母音以前の子音字の文字列を、マップ型オブジェクトを用いてソート文字列に書き換えられるようにする。そのために、文字列を取り出す際にグループごとに取り出せるよう正規表現を変更する。足「ｙ」の場合を例に説明すると、

re.compile(u”(?< kiji > [\u0F40\u0F41\u0F42\u0F54

\u0F55\u0F56\u0F58])(?< ashi > \u0FB1)”,re.X)
となる。(?＜＞) を用いることで () によって囲まれた部分がグループになり、＜＞の中にそのグループ名を設定できる。正規表現で設定したグループは「前置字＋頭、前置字、頭、基字、足」である。「前置字＋頭」をひとつのグループとしたのは、ソート文字列に書き換えたとき、前置字、頭それぞれをソート文字に書き換えるとソート文字列が長くなってしまう、という先生の助言があったからである。

グループを設定したので、書き出す際にも設定したグループを使えるようにする。新たに関数を作成し、そこで、if m.gorup(”ashi”) とすることで () の中に記述したグループがある場合 if 以下のプログラムを実行させるようにした。しかしこれでは、取り出した音節に、() の中に記述したグループがない場合エラーが出てしまった。そこで、「前置字、頭、基字、足」で構成されている文字列は、関数 kiji_pt4 を用いて書き出し、「前置字、基字、足」で構成されている文字列は、関数 kiji_pt3_1 を用いて書き出す、という風に子音字の組み合わせごとに書き出す関数を分けることにした。関数を分けることでエラーは無くなったが、関数を分けたことでプログラムが長くなってしまい、若干見にくくなってしまった。そこで、先生の助言もあり、新たに.has_key() を用いて関数を作成することで、一つの関数で書き出せるようにした。

```
if tango.has_key('zen_atama') and tango['zen_atama']:
    map=tango['zen_atama']
    k_sk+=zen_atama_map[map]
```

このようにすることで、取り出した音節に、() の中で指定したグループがある場合、マップ型オブジェクトを用いてソート文字列に書き換える、というプログラムになる。そして if 文をグループごとに追加する。

ただし、グループ kiji に関しては、if 文にはしない。なぜなら、音節に基字は必ず存在しているため、if 文を用いて、ある場合、ない場合という風に場合分けする必要がないからである。また、関数の中で.groupdict() を新たに用いているが、これはグループを使用して書き出す際に必要なものである。

グループを設定することができ、次にソート文字に書き換えるために、マップ型オブジェクトを新たにそれぞれのグループごとに追加する。このとき、書き換え後のソート文字は取り出したチベット文字と同じにした。そうすることで、文字列が正しく取り出され正しく書き換えられたかを確認することができると考えたからである。こうして一つの関数でソート文字列に書き換え、書き出せるようになった。

（ix）作業９

音節を取り出し、それぞれの子音字をソート文字列に書き換えることができるようになった。次に、ソート文字列を実際にソートした時、正しく辞書の順序に並び替えられるようにしなければならない。そのために、取り出した文字列のグループを書き出す順番と、マップ型オブジェクトで書き換える際のソート文字を変更する。

ソート文字列に書き換える際、取り出した子音字と同じ子音字に書き換えていたが、これでは順番に違いが生じてしまう。例を挙げると、「基字 (k)+ 足 (y)」と「前置字 (b)+ 基字 (k)」を辞書の順番に並び替えると、本来は「基字 (k)+ 足 (y)」が「前置字 (b)+ 基字 (k)」よりも先にこなければいけないが、ソートしたときに「前置字 (b)+ 基字 (k)」が先に並び替えられてしまうのである。この問題を解消するために、書き換えた後のソート文字をグループごとに変更する。基字のグループに関しては、現状のまま取り出した子音字と同じ子音字で書き出す。ただ

し、頭がついた基字は、通常の基字と unicode 文字コードが違うため、通常の基字に書き換えるようにしている。基字 k を例に説明すると、通常の基字 k の unicode 文字コードは 0F40 であるが、頭がついた場合の基字 k は 0F90 となる。そこで、頭がついた基字はそのまま書き出すのでなく、通常の頭がついていない基字に書き換えているのである。足のグループは、unicode 文字コードの 0F40、0F41、0F42 と順番に書き換え、足の書き換えが終わったその続きから、前置字のグループを書き換える。続いて前置字、頭、前置字＋頭と書き換える。こうすることで、先ほど述べたような順番の違いは生じなくなった。

次にグループを書き出す順番を正しくする。辞書の順番の通りに「基字→足→前置字→頭→前置字＋頭」というようにしているが、これでは、並び替えたとき順番に違いが生じてしまう。例を挙げると、「頭＋基字＋足」と「前置字＋基字」、それぞれの組み合わせの文字を並び替えたときに「頭＋基字＋足」が先に並び替えられてしまうのである。この問題を解消するために、書き出す順番を変更する。そこで、「基→前＋頭→頭→前→足」と順番を変更した。こうすることで、順番の違いが解消された。

(x) 作業１０

ソート文字と書き出す順番を正しく変更することができた。次は、「子音字が二つ続いた場合前の子音字が基字である」というような基字に関しての例外を取り出せるようにする。そのために正規表現に基字に関しての例外を新たに追加した。追加した例外は以下の通りである。

・「子音字が二つ続いた場合前の子音字が基字である」
・「子音字三つで最後が「s」でない場合、真ん中の子音字が基字である
・「子音字三つで最後が「s」で、真ん中が「g,ng,b,m」でない場合、真

ん中が基字である」

・「子音字三つで最後が「s」で、真ん中が「g,ng,b,m」である場合、最初の文字が基字である」

・「子音字の組み合わせ「dgs」は g が基字である」

・「子音字の組み合わせ「dms」は m が基字である」

・「子音字の組み合わせ「ags」は g が基字である」

・「子音字の組み合わせ「mngs」は ng が基字である」

正規表現を追加し、次に、今ある関数では例外をうまく書き出すことができないため、例外用に書き出しの関数 kiji_pt_reigai を新たに作成する。母音に関しては、正規表現にに含めていなかったため、母音が a の場合チベット数字の 0 を書き出すようにした。こうして例外にも対応できるようになった。

(xi) 作業１１

ここまでで、例外に対応させることができ、ソート文字の順番も正しくさせることができた。ここで、先生からチベット文字の単語ファイルを頂き、これを制作物を用いてソート文字列に書き換え、そのソート文字列を並び替えた。すると、所々順番がおかしくなっている箇所が発見できた。調べたところその箇所はサンスクリット語であることが分かった。サンスクリット語以外の部分は正しく並び替えられていたため、チベット語に関しては対応させることができた。そして次に、サンスクリット語に対応させることにした。しかし、サンスクリット語はチベット語と子音字の組み合わせが異なるため、新たに正規表現などを追加しなければならない。

サンスクリット語は基字で始まり、基字に足が必ずつく、そしてそれに母音がつく。つまり「基字＋足＋母音」という構成である。稀に、足

にさらに足の「w」がつく場合がある。そして、「基字＋足＋母音＋基字＋足＋母音」という風に「基字＋足＋母音」の後にさらに子音字が続くこともある。

これらの情報を元に正規表現を作成した (資料３ページの１１２行から１２１行)。次に書き出すためのプログラムを書かなくてはならないのだが、サンスクリット語に関してはこれまでしてきたように関数を新たに作成するのではなく、音節をマッチさせる関数 one_onsetsu の中に、サンスクリット語をマッチさせた後書き出すプログラムを書くことにした。なぜなら、関数を作るのは、何度も同じプログラムを入力する手間を省くためであるため、サンスクリット語の為だけに、１度しか使わない関数を作るのは無駄であると判断したからである (資料１０ページの４８１行から１１ページの５０５行)。次にサンスクリット語用に、足のマップ型オブジェクトに子音字 30 字を追加した。

先生の助言もあり、サンスクリット語に関しては足を書き出す際、

```
for ch in map:
    skt += ashi_map[ch]
```

とすることで、足の後にさらに足がつく場合にも対応できるようにした。母音に関して、0F71 をチベット数字の 0 に書き換えていたが、これではソート文字列を書き出し並び替えた時、順番に違いが生じてしまったため、0F71 を空文字列に書き換えるようにしたところ、順番の違いは解消された。

こうして、サンスクリット語にも対応させることができた。

(2) 制作の過程で出会った問題

チベット語や正規表現について全く知識の無かったため、私にとってとても大変な制作となった。さらに、必要な知識を集め、試行錯誤を繰り返しながらの制作であったため、様々な問題に直面することが多々あった。その中でも特に解決するのに時間の掛かった問題をいくつか記述する。

まずは、「文字列をマッチさせる順番」である。つまり、音節を取り出した後に、子音字の組み合わせをマッチさせる順番である。「基字」→「基字＋足」→「前置字＋基字」というように辞書の順番の通りマッチさせていた。しかしこれでは、複雑な組み合わせにマッチする前に複雑でない文字列にマッチしてしまい、文字列を正しく取り出すことができないという問題が発生してしまったのである。この問題は複雑な組み合わせを先にマッチさせることで解消することができたが、問題に直面した当初は、何故うまく取り出すことができないのか原因が分からずとても困惑した。

次に、「母音以前の子音字の書き出しの順番」を挙げることができる。辞書の順番通りに並び替えられるようにするには、グループを書き出す際の順番が重要である。当初、書き出す順番は辞書の通りにしていた。しかしそれでは、実際に書き出した際に順番に違いが生じてしまった。そこで、基字の後に母音を書き出すようにするなど、試行錯誤を繰り返したところ、辞書の順番とは逆に書き出すと正しく並び替えることができることが分かったのである。この問題に関しても大変苦しめられた。書き出す順番を考える際、ソート文字について注目するため、ソート文字をどのように書き換えるかについても同時に考えなければならなかったからである。

次に、「ソート文字に書き換える際の子音字の指定」に関しても問題に直面した。辞書の順番通りに並び替えるには、ソート文字をどのように指定するかが重要であった。完成形として、「基字」はそのまま取り出した子音字と同じ子音字を書き出し、「足、前置字、頭、前置字＋頭」は書き換え後の子音字が被らないよう、順番に指定した。「後置字、再後置字」に関しては、他と被ることがあっても順番に問題はなかったため、0F40 から順番に指定することとした。また、「母音」はチベット数字に書き換えることとした。この完成形に至るまでには時間がかかり、当初は後置字、再後置字を 0F40 から順番に書き換え、足、前置字などはその続きから書き換えていたが、しかしこれでは、並び替えに違いが生じてしまったことがある。このように、ここで指定した子音字がソート文字になるため、辞書の順番通りに並び替えるためにはとても重要であることが分かる。

問題に直面した際原因が分かっていれば、それについて調べ解決することができるが、原因が分からない時は、原因を探すことから始まるため、大変時間をとられることとなった。

4　総括

(1) プログラムのソースについて

プログラムが正しく実行されることが何よりも重要であるが、プログラムのソースを見やすくすることで、どこがどの操作をしている部分かを分かりやすくすることは、制作を円滑に進める上で重要である。そこで、書き出しのための関数の中でソート文字列を print を用いて書き出していたのを、return を用いてソート文字列を返すようにしたことで、プログラムを短くすることができ、また、書き出す際は for 文の中に、

print line を記述するだけで書き出しができるため、非常に見やすくすることができたのである。

プログラムを記述する際に、「#」を用いてコメントアウトを至る所に使用した。コメントアウトを使用することで、プログラムに反映させることを防げるため、どういった操作をしているか、などのような注意書きを記述することができる。こうすることで、一目でプログラムの内容を理解することができるため、制作を行う上で非常に役に立つのである。それだけでなく、制作物の利用者がプログラムのソースを確認する際にも、目安となることができるため、コメントアウトを積極的に用いた。

(2) 制作物の使用方法などについて

最終的なプログラムの仕様は、プログラムを実行することで、ソート文字列のみが生成されるようにした。制作を行っている時は、ソート文字の右側に取り出した文字列を書き出していた。しかし、この制作物を使用する人が求めているのはソート文字だけであると考えたため、ソート文字の元の文字列を書き出すことはやめたのである。

主な使用方法として、コマンドラインからファイルを読み込ませ、ファイルの中のチベット文字をソート文字列に書き換える、という使用方法を想定している。また、PHP などのように他の言語に組み込んで使用することもできる。PHP の場合は、「shell_exec()」を使用することで、ソート文字列を取り出すことができる。コマンドラインに直接チベット文字を入力し、プログラムを実行することでもソート文字を生成することができる。このように、当初予定していた使用方法でソート文字を生成することができるようなり、目的を達成することができたと言

えるだろう。

(3) 今後の課題などについて

チベット語はいくつかの子音字が組み合わさって構成されているため、文字列を取り出したり、書き出す際の順番などがとても難しい。制作を始めた当初は、チベット語や正規表現に関する知識が全く無く、ゼロからのスタートであったが、チベット語を取り出し、ソート文字列に書き換え、さらにサンスクリット語にも対応できるようにすることができた。福田先生の助言があったからこそできたことではあったが、チベット語に対応させることができたことは、当初の目的を達成できたと言えるだろう。しかしサンスクリット語に関しては、ある程度対応させることができたが、まだ 100 %ではない。チベット文献にはサンスクリット語が使われることがよくあるため、今後の課題として、サンスクリット語に完全に対応させることが挙げられるだろう。

注

(1) すべての文字を 16 ビット (2 バイト) で表現し、1 つの文字コード体系で多国語処理を可能にしようとするもの。世界の主要な言語のほとんどの文字を収録している

(2) フリーなオブジェクト指向プログラミング言語。

(3) 動的に Web ページを生成する Web サーバの拡張機能の一つ。また、そこで使われるスクリプト言語。

(4) ソフトウェアやハードウェアを動作させるために必要な、基盤となるハードウェアや OS、ミドルウェアなどのこと。

(5) http://www.python.jp/Zope

文献表

福田洋一「チベット語文語文法テキスト」