1 Instructions

- This is a programming project on machine translation but we assume no prior knowledge in any of the languages involved in the examples (Chinese, French, or Spanish), nor any familiarity with computational linguistics or any linguistic theory.
- Please refer to the slides for high-level intuitions and visualizations.
- The submission deadline is Friday Dec 21st at midnight. No late submissions are accepted.
- There are other testcases on the course homepage.

2 The Problem - Syntax-Directed Translation

In this project, you will be building a machine translation system that translates from a source language to a target language, say, English-to-French. The translation method we use here is syntax-directed translation, which is originated in compiling, where the source program in a high-level language (say, C or Java) is first parsed into a syntax or expression tree which guides the generation of object program (say, in Assembly or Byte-code). We adapt this technique to the translation of human languages, where the source input is first parsed into a parse-tree and then we recursively convert the tree into a string in the target language.

2.1 An English-to-Chinese Example

Consider the following English sentence and its Chinese translation (note the reordering in the passive construction):

(1) the gunman was killed by the police.

Figure 1 shows how the translation works. The English sentence (a) is first parsed into the tree in (b)\(^1\) which is then recursively converted into the Chinese string in (e) through five steps. First, at the root node,

\(^1\)The internal nodes in the parse tree are called non-terminal tags or syntactic categories. For example, VP stands for verb phrase and PP prepositional phrase. The tags right above leaf nodes are called preterminals or part-of-speech tags. For example, VBD and VBN correspond to verbs in past tense and past participle, respectively. Our tagset and grammar follows the Penn Treebank Style. For further information, please visit:
http://www.cis.upenn.edu/~treebank
http://bulba.sdsu.edu/jeanette/thesis/PennTags.html
(a) the gunman was killed by the police.

```
S
  NP-C  VP  PUNC
    DT  NN  VBD  VP-C
        |        |
        killed  by  NP-C
          IN  DT  NN
            the  police
```

(r1, r2)↓

```
VP
  VBD  VP-C
    was  VBN  PP
          |
          killed  by  NP-C
            IN  DT  NN
                  the  police
```

(r3)↓

```
NP-C  VBN
  qiangshou  killed
    was  by  NP-C
      IN  DT  NN
            the  police
```

(r5)↓  (r4)↓

```
qiangshou  bei  VBN
  DT  NN  killed
    the  police
```

(e) qiangshou bei jingfang jibi

Figure 1: A syntax-directed translation process.
we apply the rule $r_1$ which preserves the top-level word-order and translates the English period into its Chinese counterpart:

$$(r_1) \ S (x_1{:}\text{NP-C} \ x_2{:}\text{VP} \ \text{PUNC} \ (\cdot) ) \rightarrow x_1 \ x_2 \ .$$

Then, the rule $r_2$ grabs the whole sub-tree for “the gunman” and translates it as a phrase:

$$(r_2) \ \text{NP-C} \ (\ \text{DT} \ (\text{the}) \ \text{NN} \ (\text{gunman}) \ ) \rightarrow qiangshou$$

Now we get a “partial Chinese, partial English” sentence “$qiangshou$ $\text{VP}$” as shown in Fig. 1 (c). Our recursion goes on to translate the VP sub-tree. Here we use the rule $r_3$ for the passive construction:

$$
\begin{array}{c}
\text{VP} \\
\text{VBD} \\
\text{VP-C} \\
\text{was} \\
\text{x}_1{:}\text{VBN} \\
\text{PP} \\
\text{IN} \\
\text{x}_2{:}\text{NP-C} \\
\text{by} \\
\end{array}
\rightarrow \text{bei} \ x_2 \ x_1
$$

which captures the fact that the agent (NP-C, “the police”) and the verb (VBN, “killed”) are always inverted between English and Chinese in a passive voice. Finally, we apply rules:

$$(r_4) \ \text{VBN} \ (\text{killed}) \rightarrow jibi$$

$$(r_5) \ \text{NP-C} \ (\ \text{DT} \ (\text{the}) \ \text{NN} \ (\text{police}) \ ) \rightarrow jingfang$$

which perform phrasal translations for the two remaining sub-trees in (d), respectively, and get the completed Chinese string in (e).

You might also have other rules which lead to a different translation, say, replacing $r_5$ with another Chinese word for “police”:

$$(r_5) \ \text{NP-C} \ (\ \text{DT} \ (\text{the}) \ \text{NN} \ (\text{police}) \ ) \rightarrow jingcha$$

In this framework, every rule is associated with a probability, and your job is to search for the best translation (the one with the highest probability) given a parse tree.

### 2.2 English-to-French/Spanish Example

If you have no idea what the previous example is all about, here is another example, but from English to French and Spanish (it’s not a whole sentence).

(2) my friend’s black cat

| le chat noir de mon ami |
| el gato negro de mi amigo |
| [the] [cat] [black] [of] [my] [friend] |
Again, assume the parse tree for the English phrase is:

```
NP
  /\                          /\  
DP  NP
  /\  /\    PRP   JJ   NN
NP  POS  J\   black  cat
  /\     / \    / \  
my  friend
```

We can first have a rule which takes care of the reordering of possessive (’s) construction between English and French/Spanish:

\[(r'_1)\] NP  \rightarrow  x_1  de  x_0

Then we use a rule to translate the English phrase “my friend”:

\[(r'_2)\] NP( PRP (my) NN (friend) ) \rightarrow  mon ami
       mi amigo
and a rule for reordering the adjective (black) with the noun (cat):

\[(r'_3)\] NP( x_0:JJ  x_1:NN ) \rightarrow  x_1  x_0

finally, we finish the translation with the following two lexical rules:

\[(r'_4)\] black \rightarrow  noir
       negro

\[(r'_5)\] cat \rightarrow  le chat
       el gato

Of course, you may decompose \(r'_2\) into three smaller rules and get the same translations.

### 2.3 Translation Algorithm

Given a fixed parse-tree \(\tau^*\), we are to search for the best derivation with the highest probability. This can be done by a simple top-down traversal (or depth-first search) from the root of \(\tau^*\): at each node \(\eta\) in \(\tau^*\), try each possible rule \(r\) whose source-side pattern \(t(r)\) matches the subtree \(\tau^*\) rooted at \(\eta\), and recursively visit each descendant node \(\eta_i\) in \(\tau^*\) that corresponds to a variable in \(t(r)\). We then collect the resulting target-language
strings and plug them into the Chinese-side $s(r)$ of rule $r$, getting a translation for the subtree $\tau^*_r$. We finally take the best of all translations.

With the extended LHS of our transducer, there may be many different rules applicable at one tree node. As a result, the number of derivations is exponential in the size of the tree, since there are exponentially many decompositions of the tree for a given set of rules. This problem can be solved by memoization: we cache each subtree that has been visited before, so that every tree node is visited at most once. This results in a dynamic programming algorithm that is guaranteed to run in $O(npq)$ time where $n$ is the size of the parse tree, $p$ is the maximum number of rules applicable to one tree node, and $q$ is the maximum size of an applicable rule. For a given rule-set, this algorithm runs in time linear to the length of the input sentence, since $p$ and $q$ are considered grammar constants, and $n$ is proportional to the input length. The full pseudo-code is worked out in Algorithm 1.

Algorithm 1 Top-down Memoized Recursion

1: function TRANSLATE($\eta$)  
2:     if cache[$\eta$] defined then  
3:         return cache[$\eta$]  \> this sub-tree visited before?  
4:     best ← 0  
5:     for $r \in \mathcal{R}$ do  \> try each rule $r$  
6:         matched, sublist ← PATTERNMATCH($t(r), \eta$)  \> tree pattern matching  
7:         if matched then  \> if matched, sublist contains a list of matched subtrees  
8:             prob ← Pr($r$)  \> the probability of rule $r$  
9:             for $\eta_i \in \text{sublist}$ do  \> recursively solve each sub-problem  
10:                $p_i, s_i \leftarrow$ TRANSLATE($\eta_i$)  
11:                prob ← prob · $p_i$  
12:         if prob > best then  
13:             best ← prob  
14:             str ← $[x_i \mapsto s_i]s(r)$  \> plug in the results  
15:     cache[$\eta$] ← best, str  \> caching the best solution for future use  
16:     return cache[$\eta$]  \> returns the best string with its prob.

3 I/O Specifications and Samples

Command-line format:

```bash
cat <tree_file> | ./translate.py <rule_file> [-d|-k <K>]
```

where <...> and [ ...] denote required and optional arguments, respectively.

There are two input files:

- a tree file (through `sys.stdin`): a list of source-language parse trees, which contains several lines, with each line representing one parse tree.

- a rule file (filename specified by the command-line): one line for each rule in the format of

  \[lhs \rightarrow rhs \quad \text{### prob}=p\]
where \(lhs\) is the left-hand-side (source-language) tree pattern (trees with variables), and \(rhs\) is the right-hand-side (target-language) string (with the same set of variables, possibly permuted). The variables will be in the form of \(x_0, x_1, \) and so on. The real number \(p\) is the probability of this rule.

Sample input files:

input1.txt:

\[
\begin{align*}
\text{NP} & (\text{DP} (\text{NP} (\text{PRP} ("my") \text{ NN} ("friend"))) \text{ POS} ("'s")) \text{ NP} (\text{JJ} ("black") \text{ NN} ("cat"))) \\
\text{NP} & (\text{DP} (\text{NP} (\text{PRP} ("my") \text{ NN} ("friend"))) \text{ POS} ("'s")) \text{ NP} (\text{JJ} ("white") \text{ NN} ("cat")))
\end{align*}
\]

rules1.txt:

\[
\begin{align*}
\text{NP} (\text{DP} (x_0: \text{NP} \text{ POS} ("'s")) x_1: \text{NP}) & \rightarrow x_1 \ "de" \ x_0 \ ### \ prob=1.0 \\
\text{NP} (\text{PRP} ("my") \text{ NN} ("friend")) & \rightarrow "mon" \ "ami" \ ### \ prob=0.51 \\
\text{NP} (\text{PRP} ("my") \text{ NN} ("friend")) & \rightarrow "mon" \ "amie" \ ### \ prob=0.49 \\
\text{NP} (x_0: \text{JJ} x_1: \text{NN}) & \rightarrow x_1 \ x_0 \ ### \ prob=0.7 \\
\text{NP} (x_0: \text{JJ} x_1: \text{NN}) & \rightarrow x_0 \ x_1 \ ### \ prob=0.3 \\
\text{JJ} ("black") & \rightarrow "noir" \ ### \ prob=0.6 \\
\text{JJ} ("black") & \rightarrow "noire" \ ### \ prob=0.4 \\
\text{NN} ("cat") & \rightarrow "le" \ "chat" \ ### \ prob=1.0 \\
\text{NP} (x_0: \text{JJ} \text{ NN} ("cat")) & \rightarrow "le" \ "chat" \ x_0 \ ### \ prob=1 \\
\text{NP} (\text{JJ} ("black")) x_0: \text{NN}) & \rightarrow x_0 \ "noir" \ ### \ prob=0.55 \\
\text{NP} (\text{JJ} ("black")) x_0: \text{NN}) & \rightarrow x_0 \ "noire" \ ### \ prob=0.45
\end{align*}
\]

To run the program, type in the terminal

```
cat input1.txt | ./translate.py rules1.txt
```

and you will get the output

```
my friend 's black cat -> le chat noir de mon ami ### prob=0.306
my friend 's white cat -> *** failed ***
```

The output contains one line for each source-language tree in input.txt. As shown in the above example, the format is quite similar to the rules file, except there are no quotes, and the probability of the best translation is printed to the 3rd decimal digit (i.e., \(\% .3f\) \ % prob).

Note:

1. if the search fails, simply print \*** failed ***\ in place of the best translation. (see the above example).

2. each individual word in the input (e.g., "my") is surrounded by a pair of double quotes, and we assume that white spaces, double quotes, or parentheses are not part of any word.

3. The sample case is available online as input1.txt, rules1.txt, and output1.txt under the homework page, where you can also find a more complicated example.
4 Recommended Design

- a Tree class
  A separate class for trees is highly recommended. It should work for both parse-trees and tree patterns (trees with variables), and implements the following two functions:

  - Tree Parsing
    the static method from_string(s) implements the parsing of a tree represented in string input into a Tree instance. This might be the most difficult part of this project. You may want to write a “pretty-print” function inside the Tree class to help you debug this part.

  - Tree Pattern Matching
    the method patternmatch(self, pattern) takes a tree pattern as input and returns a pair (matched, sublist), where matched is a boolean (whether the matching succeeds or not), and sublist is a list of matched subtrees in case of success. See Algorithm 1.

- a Rule class: This is optional. In fact, a tuple of (pattern, rhs, prob) would suffice.

- search: This module implements the memoized depth-first search (Algorithm 1).

5 Extensions (Extra Credit Problems)

5.1 Outputing the derivation (easy)

What if we want to look into the derivation (see Figure 1)? How is the source sentence translated to the foreign sentence and what rules are used? We add another option -d to the command line for outputing the derivation of the best translation (if any). For example,

    cat input1.txt | ./translate.py rules1.txt -d

should output the following:

my friend ’s black cat -> le chat noir de mon ami ### prob=0.306
NP (DP (x0:NP POS (’s)) x1:NP) -> x1 de x0 ### prob=1.000
  | x0: NP (PRP (my) NN (friend)) -> mon ami ### prob=0.510
  | x1: NP (x0:JJ NN (cat)) -> le chat x0 ### prob=1.000
  | x0: JJ (black) -> noir ### prob=0.600
| black cat -> le chat noir ### prob=0.600
my friend ’s black cat -> le chat noir de mon ami ### prob=0.306
my friend ’s white cat -> *** failed ***

This output is available online as output1-d.txt. A derivation basically outputs all the rules involved in a tree-like format. For each subtree, we first output the top-level rule, e.g.

NP (DP (x0:NP POS (’s)) x1:NP) -> x1 de x0 ### prob=1.000

and then recursively output the sub-derivations for the subtrees corresponding to the two variables (x0 and x1) with one more level of indentation. Finally, we output the best translation for this subtree in the standard format.

There is a more complicated example online.
5.2 Rule Indexing (easy)

In line 5 of Algorithm 1, we enumerate each rule and check if its LHS tree-pattern matches the current tree. This is rather inefficient in general when we have a very large rule set. However, you can index the rules using a dictionary, where the keys are the top-level productions of the tree pattern. For example, for the rule

\[
\text{NP (DP (x0:NP POS ('s)) x1:NP) } \rightarrow \text{ x1 de x0 } \text{ ### prob=1.000}
\]

the top-level production is

\[
\text{NP } \rightarrow \text{ DP NP}
\]

Now when you are translating a tree \( T \) whose top-level production is \( \gamma \), you only need to look at rules indexed under \( \gamma \) (because other rules are certainly incompatible with \( T \)).

You can further group rules with same LHS tree pattern together, as a second-level of indexing, which will save some time by reducing redundant calls of PatternMatch().

5.3 \( k \)-best translations (challenging but very interesting)

Clearly, there are many possible translations given a ruleset and an input tree. Can you also output the 2nd-best, 3rd-best, or up to \( k \)th-best translations? We add another option \(-k\) to the command line to request “top-\( k \)” translations. For example,

\[
\text{cat input1.txt | ./translate.py rules1.txt } -k 3
\]

will output

my friend ’s black cat -> le chat noir de mon ami ### prob=0.306
my friend ’s black cat -> le chat noir de mon amie ### prob=0.294
my friend ’s black cat -> le chat noire de mon ami ### prob=0.204
my friend ’s white cat -> *** failed ***

Note: do not print duplicate translations (say, there might be another way of deriving the best translated string). Be sure to include in debrief.txt your algorithm or thoughts if you attempted this question.

References: see (Huang and Chiang, 2005) and (Huang et al., 2006).

6 debrief.txt

1. How many hours did you spend on this project?

2. If you did any extra credit problem, describe your algorithms or methods.

3. If you had discussed with another student, what’s his or her name?

References
