TECHNICAL UNIVERSITY OF DENMARK - DTU COMPUTE -

[SUPERSTRINGWITHEXPANSION] Problem

Salik Lennert Pedersen
Anders Holmgaard Opstrup
Federico Bergamin

1 DESCRIPTION OF THE PROBLEM

In SUPERSTRINGWITHEXPANSION decision problem we have two different alphabet Σ and Γ, where $\Gamma = \{\gamma_1, ..., \gamma_m\}$ contains m symbol, a string s which is made up of symbols that belongs to the Σ alphabet (formally $s \in \Sigma^*$), k strings $t_1, ..., t_k$, which is made up of symbols from both alphabets ($t \in (\Sigma \cup \Gamma)^*$), and in the end, we have m subsets $R_1, ..., R_m \subseteq \Sigma^*$. These subsets contain symbols of Σ alphabet, and as we can notice, they are as many as the symbols in the alphabet Γ. That's because for each γ_i there is a subset R_i. We can understand better this relation explaining the core of our problem.
The output of the problem will be YES if exits a sequence of words $r_1 \in R_1$, $r_2 \in R_2$,...,$r_m \in R_m$, such that for all $1 \leq i \leq k$ the so-called *expansion* $e(t_i)$ is substring of s; the expansion of a string t_i consists in the replacement of every symbols $\gamma_j \in \Gamma$ that appears in the string t_i with its expansion, where the expansion of a symbol is defined as follow: $e(\gamma_j) := r_j$. That is, we have to choose for every symbol $\gamma_j \in \Gamma$ a symbol $r_j \in R_j$, and we know that $r_j \in \Sigma^*$. We have to replace the symbol γ_j in each string t_i with the symbol r_j that was chosen. We have to do that for each $\gamma \in \Gamma$. In this way, using these replacements, we are transforming our string t_i in its expansion $e(t_i)$.
In the end we will obtain that every expansion-string is made up of symbols of the alphabet Σ as our string s. Hence, if every new constructed string $e(t_i)$ is a substring of s, the answer for our problem will be YES, otherwise NO.
In other words, the answer of our problem will be YES if there is a selection of the $r_i \in R_i$ such that, replacing each symbols $\gamma_i \in \Gamma$ with the respective symbol r_i (we have to remember that for each γ_i we have to chose one symbol from the subset $R_i \subseteq \Sigma^*$) in every string $t \in T$, where

$|T| = k$, we will have that every new string that we obtain with this replacement is a substring of the string s. If a selection of $r_i \in R_i$ like this doesn't exist, the answer will be NO.

1.1 Simple examples

With the text of the problem we receive also some problem instances on the alphabets $\Sigma = \{\texttt{a,b,...,z}\}$, $\Gamma = \{\texttt{A,B,...,Z}\}$. The first line of the file contains the number k, which is the number of strigs t we have. The second line contains the string s and the following k lines the strings $t_1, ..., t_k$. Finally, the last lines (at most 26, the number of letters in the alphabet) start with a letter $\gamma_j \in \Gamma$ followed by a colon and the contents of the set R_j belonging to the letter, where the elements of the set are separated by commas. Hence, we can clearly see that for each symbols $\gamma_i \in \Gamma$, in this case letter, there is a subset R_i and in this subset we have to chose one symbol.

For example:

```
4
abdde
ABD
DDE
AAB
ABd
A:a,b,c,d,e,f,dd
B:a,b,c,d,e,f,dd
C:a,b,c,d,e,f,dd
D:a,b,c,d,e,f,dd
E:aa,bd,c,d,e
```

In this example we can observe that there is NO possible selection of symbols r_i such that every string t would be a substring of `s=abdde`. In fact if we have a look to t_2=`DDE` and t_3=`AAB`, we can notice that they have the same structure, and an the same time, in our string s the only letter that is repeated twice is the `d`. So we conclude that `A=d` and `D=d`. Moreover from t_4=`ABd` we see that, since t_4 will be a substring, or `B=d` or `B=b`. If we try to choose our selection of r_i following these rules, we can noticed that there is no possible solution, such that, at the same time, all $e(t_i)$ are substring of s.

2 SuperStringWithExpansion is in $\mathcal{NP}$

We have to show and prove that our problem is in $\mathcal{NP}$. To do that we use the so called *guess-verify algorithms* proof, so we have to start designing a deterministic algorithm A which takes as input a problem instance X and a random sequence R.
Important: In this case, since we already use R to indicates the subset of the problem, we are going to call the random sequence with the capital letter **G**.

Proof. We have to show that there is a polynomial p and a randomized p-bounded algorithm A which satisfies the condition of the class $\mathcal{NP}$.

1. Let Σ be an alphabet and Γ another alphabet. We know that $|\Gamma| = m$ (cardinality of the alphabet Γ is m). Let s be a string made up of symbols of Σ^* and let $t_1, ..., t_k$ be k strings made up of symbols of $(\Sigma \cup \Gamma)^*$. Then let $R_1, ..., R_m$ be subsets that contains symbols of Σ^*. So the first part of the input is:

   $$X = (\Sigma, \Gamma, s, t_1, ..., t_k, R_1, ..., R_m)$$

   We could simplify this symbolism, considering the set $T = \{t_1, ..., t_k\}$ and $R = \{R_1, ..., R_m\}$. In this way we obtain that the first part of the input for our algorithm A now is:

   $$X = (\Sigma, \Gamma, s, T, R)$$

   a) The random sequence G consists of **integers** in the range $\{1, ..., n\}$, where $n = max(|R_i|)$ for $1 \leq i \leq m$.

   b) Now on input $((\Sigma, \Gamma, s, T, R), \mathbf{G})$, algorithm A checks whether G contains at least m integers, and for every integer it has to check whether:

   $$int_j \leq |R_j|$$

   which means that A has to check if the integer contains in the position j, where $1 \leq j \leq m$, is less or equal than the cardinality of the subset R_j. The reason of that will be clear immediately.
   If G is shorter, or the first m integer don't satisfy the condition above, A will return NO. Otherwise A uses the first m integers $g_1, ..., g_m$ of G to select a symbols in every subsets $R_1, ..., R_m$. In this way:

   $$r_{g_i} \in R_i$$

   That is, the integer g_i, which is in the position i inside the random string G, indicates the position of the symbol that we have to choose inside the subset R_i. That's why it has to satisfy the condition above, in fact if the integer g_i is greater than the cardinality of R_i we are not able to select a symbol.

   c) Now we have to choose an element for every subset R_i, where $1 \leq i \leq m$, and then using the expansion for the symbols of the alphabet Γ, we are able to "link" a letter $\gamma_i \in \Gamma$ with a symbol in R_i. (That's because for each letter γ_i we have to choose one and only one symbol in the subset R_i, the i is the same, because the subset is linked to the letter).

   $$\gamma_i \in \Gamma \Longrightarrow e(\gamma_i) = r_j \in R_i$$

   where $j = g_i$ of the string G

Now we could substitute every letters $\gamma_i \in \Gamma$, $1 \le i \le m$, which are in the k strings t with their expansion that we obtained thanks to the formula above. After that, we will have k strings $e(t_i)$ (which are the expansion of our strings) made up of only symbols in Σ^*.
Then we have to check if every string that we obtain $e(t_l)$, $1 \le l \le k$, is a substring of our string s. If every string is a substring the algorithm A returns YES, otherwise it returns NO.

2. Now we have to show that the two conditions of the class $\mathcal{NP}$ are met:

   a) Let us first assume that the true answer is YES, so there is a selection of symbols $r_1, ..., r_m$ from the subsets $R_1, ..., R_m$ such that, using the expansion for the symbols $\gamma \in \Gamma$ first, and then for the k strings t, we will obtain that every string $e(t)$ (the expanded one) is a substring of s. So the symbols $r_1, ..., r_m$ is our solution. That means, that our subsets $R_1, ..., R_m$ contains one symbol each to expand $\gamma_1, ..., \gamma_m \in \Gamma$ in the right way, and we have to choose exactly that symbol in every subset R_i. Hence, there will be a string $G = g_1...g_m$ (which is the random string and contains the position of the element we will select) such that we will select the right symbol inside every subset R, so we are able to obtain that every string $e(t_i)$ is a substring of s. In this case if G is given to A, A will correctly return YES. Hence, there is a string of length at most m which lead to a correct answer YES. To calculate the probability to obtain this string we try to see how to built this string that satisfy our problem: we show this probability in the two cases: the first, where all the subsets have exactly n elements (which is useful to understand how change the probability in regard to the number of elements of the subsets and the number of the subsets) and the second, in which we consider every cardinality of the subset. We have to choose exactly one element in each subset, so our g_i has to be the exactly position of the right symbols. In the first case for each g_i we have to choose 1 position between n, ans so we have:

$$G = g_1 g_2 ... g_{m-1} g_m$$

$$\mathbf{P}[\text{G is the right string}] = \underbrace{\left(\frac{1}{n}\right) \times ... \times \left(\frac{1}{n}\right)}_{m \text{ times}} = \left(\tfrac{1}{n}\right)^m$$

   Otherwise, if we want to use subset with no fixed cardinality, we have this probability:

$$G = g_1 g_2 ... g_{m-1} g_m$$

$$\mathbf{P}[\text{G is the right string}] = \left(\tfrac{1}{|R_1|}\right) \times ... \times \left(\tfrac{1}{|R_m|}\right)$$

   We could see that this probability is small, especially if n and m are big, but it's positive. Hence, the first condition is satisfied.

b) Now, we assume that the true answer is NO, so there is no possible selection of symbols $r_1,...,r_m$ from $R_1,...,R_m$ such that every expanded-strings $e(t_i)$ are substring of s. In other words, for each random string G we pass to A, there is no possible sequence of position $g_1...g_m$ such that let us select the right symbols to obtain that every expanded-strings $e(t_i)$ are substring of s. Hence, regardless the random string G, the algorithm A will always answer NO in this case, so also the second condition is satisfied.

3. Now we have to show that the running time of the algorithm A that we've built is p-bounded for some polynomial p. We could see that our algorithm have to:

   - check if G contains at least m integer: *O(m)*;
   - for each integer check if the condition is satisfied: so for each integer has to calculate the cardinality of the subset R_i related to the position g_i and compare to the integer value. If we are going to use arrays we have to compare an integer to the `length` of the array. So it will be *O(1)* for each element in G. For m element it will be *O(m)*;
   - select the exact element in R_i using the position g_i for every subset. If we will work with arrays we have to enter one cell, so *O(1)*, but for all the letters of the alphabet Γ is $O(m \times 1) = O(m)$;
   - read all the k strings and substitute all the γ with their expansions. The worst case is when the length of the strings are almost n, so we have to read all the k strings and substitute. If we think that a substitution is *O(1)*, then it will be $O(k \times n)$;
   - in the end check if the all k expanded-strings are substring of s. Using the simplest algorithm, we have that it will be *O(n+q)*, where n is the length of the string s and q the length of strings t. Hence, for all k strings t it will be *O(k(n+q))*

   Therefore, we can see that our algorithm give the result in time:

$$m + m + m + kn + k(n+q) = 3m + kn + k(n+q)$$

   In the worst case, we could have n symbols in Γ, and n strings t, so the algorithm will be $O(n^2)$, which is polynomial.

□

3 SuperStringWithExpansion is $\mathcal{NP}$−complete

In this section we are going to prove that our SuperStringWithExpansion problem is $\mathcal{NP}$−complete.

Proof. We could reduce from Graph-3-Coloring, which is a $\mathcal{NP}$−complete problem. In other words we have to define a *transformation T*, that will create from each instance **X** of Graph-3-Coloring an instance of our problem SuperStringWithExpansion.

Let $G = (V, E)$ be the input graph for Graph-3-Coloring, where:

$$V = \{v_1, v_2, ..., v_n\} \text{ is the set of nodes}$$
$$E = \{e_1, e_2, ..., e_k\} \text{ is the set of edges}$$

we label the 3 colours red, blue, yellow in this way: $\{R, G, B\}$, and the transformation T works as follows.
Firstly we have to define two alphabets Σ and Γ: we choose:

$$\Sigma = \{R, G, B\} \text{ and } \Gamma = \{v_1, v_2, ..., v_n\}, \text{ that is, the nodes of the graph G.}$$

Then, for each edge $e_i \in E$ we consider the two nodes that it links, and we create a string:

$$t_i = v_a v_b \text{ (where } v_a \text{ and } v_b \text{ are linked by } e_i\text{)}$$

In this way, our strings t contains only letters of Γ. Hence, we satisfy the rule that $t \in (\Sigma \cup \Gamma)^*$
Then we have to create the subsets relative to each letters of Γ, and in other words, to all nodes $v_1, ..., v_n$. Therefore for each letter in Γ we define:

$$R_1 = R_2 = ... = R_n = \{R, G, B\}$$

In this way, using the structure of the SWE file to understand better, we obtain:

$$v_1 : \text{R,G,B and this is } R_1$$
$$v_2 : \text{R,G,B and this is } R_2$$
.
.
.
$$v_n : \text{R,G,B and this is } R_n$$

We could notice that $R_i \in \Sigma^*$ for $1 \leq i \leq n$.
In the end we have to define the string s, and we do like this:

$$s = BRGRBGB$$

We could see that the running time of this transformation T is linear in the input size, in fact for each edge $e \in E$, it creates a string with two literal (the nodes that the edge links). Hence, in total it crates $|E|$ string of length 2. Then for each node $\{v_1, ..., v_n\}$ it creates a subset with three element. Hence, we obtain n subset of size two. In the end it creates a string s.

Now suppose that there is a true answer to our GRAPH-3-COLORING instance **X**, then for definition of the problem, there is a "3-colours assignment" to the vertices:

$$\{v_1, v_2, ..., v_n\} \rightarrow \{R, G, B\}$$

such that all adjacent nodes don't have the same colour. We claim that, then, there exists a selection of one element for each subset $R_1, ..., R_n$, which are linked to the symbols $\{v_1, ..., v_n\}$, such that after the substitution of v_i with $r_i \in R_i$ (the chosen element) in all the strings t, we obtain that all the strings t are substring of the string s. Consider that the expansion of each symbol $v_i \in \Gamma$ is equal to the colour of the "colour-assignment" to the vertices $v_i \in V$ that satisfies the GRAPH-3-COLORING problem instance. In other words every v_i has the same colour of the node v_i, and in this way every symbol in Γ could have one of these expansion: R,G or B. We could see that this will satisfy the rules, because our subsets $R_i = \{R, G, B\}$.

Because of the way we construct our strings t, which contain each one two adjacent nodes, we obtain that through the expansion and the substitution of the symbols in Γ, each strings t have to be the form of:

$$RG, RB, GB, GR, BG, BR$$

That's because the "colour-assignment" satisfies the GRAPH-3-COLORING problem ans so every pair of adjacent nodes doesn't have the same colour. Hence, every string t has to be substring of our string s, so we constructed a satisfying selection for the instance of SUPERSTRINGWITHEXPANSION.

On the other hand, let us assume that there exists a selection of one element for each subset $R_1, ..., R_n$, such that, after the expansion of each symbols in Γ and its substitute inside the strings t, in this way:

$$\begin{gathered} v_1 \rightarrowtail r_1 \in R_1 \\ v_2 \rightarrowtail r_2 \in R_2 \\ . \\ . \\ . \\ v_n \rightarrowtail r_n \in R_n \\ \text{where } R_1, ..., R_n = \{R, G, B\} \end{gathered}$$

we obtain that all strings t are substring of string s. We claim, then, that there exist a "colour-assignment" to the vertices of $G = (V, E)$, such that all adjacent nodes don't have the same colour. Because of the construction of our strings t, which represent all the pairs of adjacent nodes, if we have a YES answer to the SUPERSTRINGWITHEXPANSION problem, it means that every string t has one of these forms:

$$RG, RB, GB, GR, BG, BR$$

that's because, these are all the possible substring of length 2 of our string s. Hence, we can see that, since we have a satisfying selection, we cannot have a string t that contains two equal literals. That means that every two adjacent nodes have different colours. Therefore we find also a satisfying "colour-assignment" for the GRAPH-3-COLORING problem.

That shows the we can reduce from GRAPH-3-COLORING, then our problem SUPERSTRINGWITHEXPANSION is in $\mathcal{NP}$-complete.

$\square$

These are two example about how our transformation works: in the first example there is a YES answer, in the second, instead, is not possible.

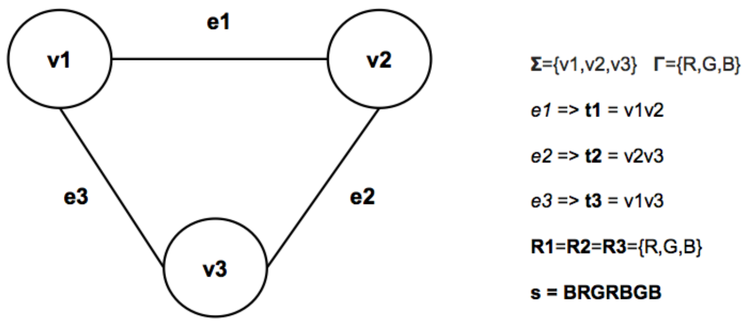


Figure 3.1: Example of a YES answer

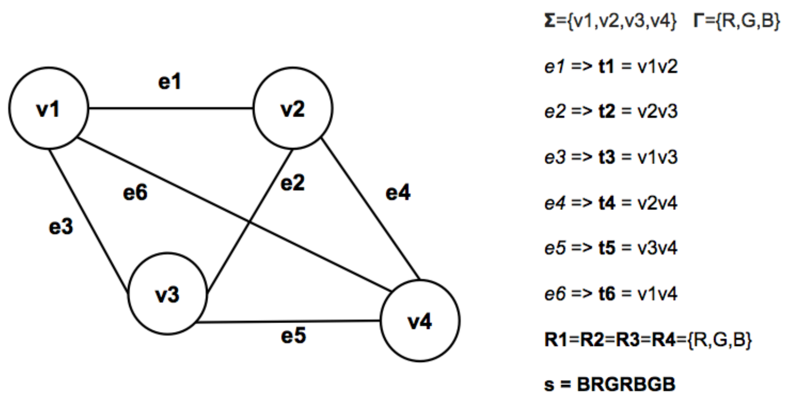


Figure 3.2: Example of a NO answer

4 Implementation of the decoder

We have to implement a decoder that read a SWE files, which contains an instance of our SuperStringWithExpansion problems (where Σ is the lower-case alphabet and Γ the upper-case one), that check if these are written in a correctly way, and in particular rejected the files in which the subset R_i are not specified in a proper way.
Our program has to read line to line the SWE file and check if everything is correct. Hence, we have to check these things:

- the first line has to be a number, which represents the number of the strings t;
- the second line has to be a string made up of only lower-case characters, we cannot accept numbers, symbols and upper-case letters;
- each string t has to be a string which could make up of both upper-case and lower-case letters, but we cannot accept numbers and symbols. Moreover, the length of the string t has to be less or equal to the length of the string s;

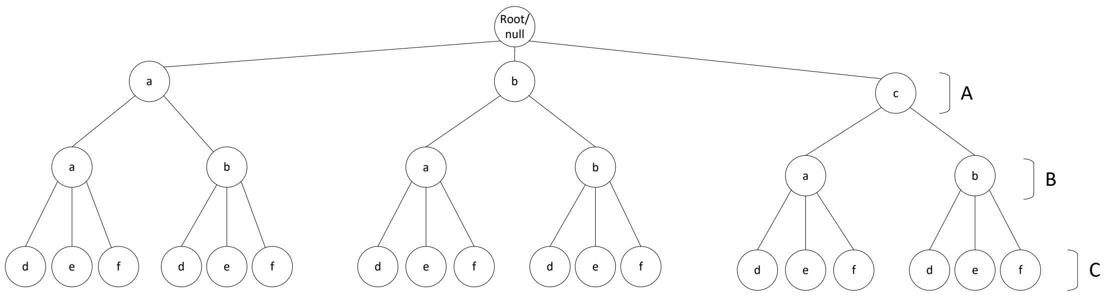
- each subset has to begin with a capital letter followed by the column and then there has to be lower-case letter separated by the comma. We have to check if this "grammar" is respected, hence, we cannot accept symbols, blank spaces, upper-case letter except at the beginning. Moreover we have to check that there exists a subset for all the upper-case letter that appears on the strings t.

The program code (in Java) is attached in the project folder.

5 Heuristic algorithm

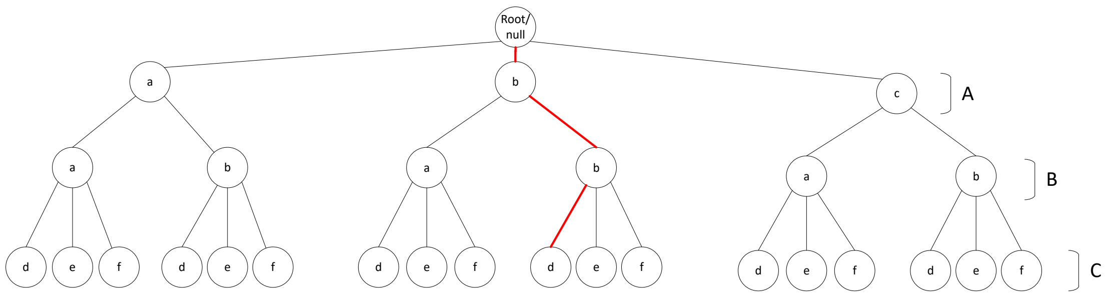
In order to construct a heuristic algorithm for the SuperStringWithExpansion problem we decided to implement an algorithm that works in several steps.

1. Decoding the file and storing the values in appropiate datastructures. This is done in $O(n)$ time complexity.
2. Filtering the input thereby potentially making the problem smaller is done in multiple steps.
    a) The strings $t_1 \dots t_k \in (\Sigma \cup \Gamma)^*$ are filtered by removing duplicates and t strings that are substrings of other t strings for instance if $t_1 = C$, $t_2 = DAC$ and $t_3 = DACD$ then both t_1 and t_2 will be removed as they are substrings of t_3. This is done in $O(n^2)$ time complexity as we need to run through all t strings once and place them in a HashSet which has to to be run through each time to check whether or not the HashSet contains the value or if the value is a substring of a previously added t string.
    b) When filtering t strings each $\gamma \in \Gamma$ is stored in a HashSet in order to keep track of which subsets $R_1 \dots R_m \subseteq \Sigma^*$ should be used. This is done in $O(n)$ time complexity.
    c) We filter the subsets $R_1 \dots R_m \subseteq \Sigma^*$ by checking if the subset should be used. If the symbol γ is found in the HashSet from the previous step then we move on to filter the actual set. This is done by checking if each element is a substring of the string s. If an element is not a substring of s, the element is removed as it is not an option. Otherwise it is kept in the set. This can be done in $O(mn^2)$ by running through all r elements of of all $R_1 \dots R_m$ subsets and checking if the string s contains the string as a substring.
3. In order to be able to search through all the possible combinations we create a tree that follows this pattern:



This is done by recursively taking the $R_1 \dots R_m$ subsets and then adding each string r in the subset as a child of all nodes in the the previous layer. For instance in the picture the elements of $\gamma = A$ is $\{a, b, c\}$, $\gamma = B$ is $\{a, b\}$ and $\gamma = C$ is $\{d, e, f\}$. With m sets this gives a time complexity of $O(n^m)$

4. Searching the tree is basically a pre-order Depth-First-Search done in a couple of steps.
   a) Check if the node is a leaf and a solution is not found already. If the node is a leaf and a solution has not been found then traverse up the tree visiting all parents to get their γ value and r value and store them in a Hashtable as a possible solution.



   This represents a possible solution where $A = b, B = b$ and $C = d$

   b) Check the solution stored in the Hashtable against all the t strings to see if the solution works for all t strings.
   c) If the node is not a leaf and a solution has not been found then call traverseTree function recursively on all children of the node. This means exponential time complexity as each node potentially has n children. Assuming that our alphabet Γ is limited to m this gives $O(n^m)$ time complexity.
   d) When/if a solution is found it is stored in a separate Hashtable.

   The process of searching the tree can be made iteratively. This can be done such that when creating the tree and adding leafs to the tree, then adding them to a List as well which can be iterated over. This still gives a worst case time complexity of $O(n^m)$ for searching all possible combinations.

5. Add the unused $R_1 \dots R_m \subseteq \Sigma^*$ subsets to the solution. This can be done in $O(n)$ time complexity.

6. Sort the solution by γ and print to a .SOL file. Using the java built in Collections.sort which uses a modified mergesort this gives a time complexity of $O(n \log(n))$

This gives a combined worst-case running time of $O(n^m)$

6 More about the worst case running time

As we can see, thanks to the explanation of the heuristic algorithm, for our problem, which is based on the alphabet letters, that are 26, we have a worst case running time of $O(n^{26})$, or better $O(n^m)$, where the max value of m is 26 . That's because we have 26 letters, and for each letter we have a subset linked to it, that could contains all the possible expansion for that letter, and so n possibilities.
In other words, we obtain the worst case, if, after the filtering process of our algorithm, we cannot delete anything, and so we have to use all the 26 subsets and all the possibilities that they contain, which in the worst case are n. Hence, the tree that our algorithm build have 26 levels, and each level k contain n^k nodes. The last level contains n^{26} nodes.
Since the pre-order DFS algorithm is usually $O(n)$, where n is the total number of the nodes, we have that in our case the complexity is: $O(n^{26} + n^{25} + ... + n + 1)$, and so we could say that it will be $O(n^{26})$.

We have this fact, only because we are using a finite alphabet. If we are using a alphabet that contains n letters, instead, where n is big, we could see that the complexity of our algorithm, in the worst case is very bad, because we will obtain $O(n^n)$. But it's not the case of our exercise, and moreover, thanks to the filtering process we are able to obtain an answer in a good time.

7 Appendix A: Run the algorithm

Instructions to run the algorithm: There are two options to run the program.

1. Run the *.jar file in the following way: java -jar chp-project-assignment.jar <testfile> where <testfile> could be test01.SWE if the file is in the same folder as the jar file.

2. Run the *.jar file in the following way: java -jar chp-project-assignment.jar. Then the program will ask for a relative path to a <testfile> so test01.SWE could be entered.

After completion the *.SOL file can be found in the same directory as the jar file. Furthermore the program will ask for another relative path in case you want to test with another file. Once you wish to exit the program enter N instead of a path to a file.