1. Write regular expressions for the following informally described languages:

a. All strings of a's and b's with the subsequence abb.
   \( \Sigma = \{a, b\}; \)
   \((a|b)^*a(a|b)^*b(a|b)^*b(a|b)^*\)

b. All strings of a's and b's with an even number of a's and an odd number of b's.
   The expression \((00|11)^*((01|10)(00|11)^* (01|10)(00|11)^*)^*\) given in the notes is for going from a state back to itself. Let R denote this regular expression, replacing 0 by a and 1 by b. There are two paths from the starting state (let's call it S) to the state representing even number of a's and odd number of b's (let's call it T): S b T or S aba T. From each state (starting, intermediate, and ending), one can go arbitrarily and back to that state. So, you can replace each state by R. So the answer is \(R \ b \ R \ | \ R \ a \ R \ b \ R \ a \ R\)
   Note that regular expressions do not have a minimized form. So, there is no need to further minimize anything.
2. Consider the regular expression $\text{aac}^* \mid \text{b}(\text{a} \mid \text{b})\text{c}^*$ defined on $\Sigma = \{a, b, c\}$.

a. Construct the NFA for the regular expression. You can directly draw the NFA without going through the RE-to-NFA steps.

b. Convert the NFA to DFA. Show the conversion steps.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S1</td>
<td>S3</td>
<td>X</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>x</td>
<td>X</td>
</tr>
<tr>
<td>S2</td>
<td>x</td>
<td>x</td>
<td>S2</td>
</tr>
<tr>
<td>S3</td>
<td>S4</td>
<td>S5</td>
<td>X</td>
</tr>
<tr>
<td>S4</td>
<td>x</td>
<td>x</td>
<td>S4</td>
</tr>
<tr>
<td>S5</td>
<td>x</td>
<td>x</td>
<td>S5</td>
</tr>
</tbody>
</table>

c. Minimize the DFA. Show the minimization steps.
3. Consider $\Sigma = \{a, b\}$. Answer the following DFA related questions. When constructing DFA, there is no need to show your construction steps, but you need to informally state how you get the DFAs.

a. Construct a DFA that accepts $(a|b)^*$ except for $abb$.

b. Construct a DFA that accepts $(a|b)^*$ except for $b^*a^*$.

c. Based on the techniques you use in (a) and (b), can you come up with a DFA construction algorithm for the "except for" type of languages? (Just focus on the main idea.)

1) Draw the DFA of the RE in the exception statement. The DFA should be a complete DFA, i.e., every state should have a transition for each input symbol.
2) Each accepting state becomes non-accepting state and each non-accepting state becomes accepting state.
4. A token recognizer is designed to handle the following tokens, where \( \mathbb{A} = \{a, b, c, d\} \).

- \( T_1 = \text{aab} \)
- \( T_2 = \text{bcc}^* \)
- \( T_3 = \text{bcd}^* \)
- \( T_4 = \text{aab}\text{cd}^* \)

a. Construct a minimized DFA for token recognition.

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>b</th>
<th>c</th>
<th>D</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td>A,1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>B,2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>C,3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td>D,4, T1</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td>I, 8, T4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>6</td>
<td>7</td>
<td></td>
<td>E, 5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
<td>7</td>
<td></td>
<td>F, 6, T2</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td>G, 6, T2</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td>H, 7, T3</td>
</tr>
</tbody>
</table>
```
b. Execute your DFA to process the following string to identify tokens. List every token string and its token name. Also, describe all the backtracking actions taken during the process.

```
Aabaabcdddbcbccccaaabcddd
aab| aabcdddbcbccccaaabcddd        -- got token aab(T1)
aab| aabcddd| bcbccccaaabcddd        -- got token aabddd (T4)
aab| aabcddd| bc| bccccaaabcddd        -- got token bc(T2)
aab| aabcddd| bc| bcccc| aabbcddd        -- got token bcccc(T2)
aab| aabddd| bc| bcccc| aab| bcddd        -- got token aab(T1)
aab| aabcddd| bc| bcccc| aab| bcddd        -- got token bcddd (T3)
```

5. Consider the grammar $S \rightarrow 0S11 | S1 | \varepsilon$

(a) Briefly describe the language generated by this grammar.

$0^n1^m2^n+m$

(b) Show that this grammar is ambiguous by giving a string that can be parsed in two different ways and showing the two corresponding parse trees.

```
      S
     /\  \
    0 /  \ 11
   / \   \\
  S 1  \\
 / \   \\
\varepsilon
```

```
      S
     /\  \
    0 /  \ 11
   / \   \\
  S 1  \\
 / \   \\
\varepsilon
```

(c) Rewrite the grammar to eliminate the ambiguity.

$S \rightarrow 0511 | T$  $T \rightarrow T1 | \varepsilon$
6. Let $L$ be a language defined over $\Sigma = \{0, 1\}$ and $L$ consists of all strings with the same and even number of 0's and 1's.

   a. Give a context free grammar for $L$.
   
   $A \rightarrow 01 B | 10 B | X A | Y B | \epsilon$
   $B \rightarrow 01 A | 10 A | Y A | X B$
   $X \rightarrow 00 X 11 | 11 X 00 | A$
   $Y \rightarrow 00 Y 11 | 11 Y 00 | B$

   b. Show a parse tree for the string 0110100.

   ![Parse Tree]

   c. Leftmost derivation.
   
   $A \Rightarrow 01 B \Rightarrow 01 Y A \Rightarrow 01 11 Y 00 A \Rightarrow 01 11 B 00 A \Rightarrow 01 11 01 A 00 A \Rightarrow 01 11 01 00 A \Rightarrow 01 11 01 00 01$

7. Construct a regular grammar for the language $L$, where $L$ accepts $(a|b)^* \text{ except for } b^*a^*$ (start from your answer for 3).

   $S \rightarrow bS | aT$
   $T \rightarrow aT | b U$
   $U \rightarrow aU | bU | \epsilon$
8. Construct a type-0/1 grammar for the language $\delta \sigma \delta$, where $\delta$ can be any string of a's and b's. Use some examples to illustrate how your grammar would work.

$$\begin{align*}
S & \rightarrow ACME \mid BCNE \\
C & \rightarrow ACM \mid BCN \\
MA & \rightarrow AM \\
MB & \rightarrow BM \\
NA & \rightarrow AN \\
NB & \rightarrow BN \\
ME & \rightarrow AE \\
NE & \rightarrow BE \\
A & \rightarrow a \\
B & \rightarrow b \\
C & \rightarrow c \\
E & \rightarrow \epsilon
\end{align*}$$

Generate $\delta \sigma \delta^{-1}$ with M equivalent to A and N equivalent to B
An E at the end to help with the conversion of M to A and N to B
$$S \Rightarrow ACME \Rightarrow ABCNME \Rightarrow ABBCNNME$$

Convert the last symbol, M to A or N to B, and move it to the front
In parallel, other symbols can be converted and moved, but does not matter
$$ABBCNNME \Rightarrow ABBCNNAE \Rightarrow ABBCNA \Rightarrow ABBCANNNNE$$

Continue with the conversion and move
$$ABBCANNNNE \Rightarrow ABBCANBE \Rightarrow ABBCABNE \Rightarrow ABBCABBE$$

The rest is to substitute nonterminals to terminals.
Not working. Consider to use A/B to clear the pairs 01/10. If only encounter one, then go to B, when meet another then come back to A.

A → 01 B | 10 B | X A | Y B | ε
B → 01 A | 10 A | Y A | X B
X → 00 X 11 | 11 X 00 | A
Y → 00 Y 11 | 11 Y 00 | B

Fails for: 10 1010 1010 1010 1010 1010 10

S → 00 S 11 | 11 S 00 | A | ε
T → 00 T 11 | 11 T 00 | B
A → 01 B | 10 B | B 01 | B 10 | S
B → 01 A | 10 A | A 01 | A 10 | T

Fails for: 1001 1010 0110 1010 1011

Not working. Consider each pair in the string, 01/10/11/00, extend the above analysis

S → 00 S 11 | 11 S 00 | A | ε
T → 00 T 11 | 11 T 00 | B
A → 01 B | 10 B | B 01 | B 10 | S
B → 01 A | 10 A | A 01 | A 10 | T

It will be endless

Not working. Consider the string as pairs of 00/11/01/10.

S → X S Y | Y S X | C S C | ε
X → 00, Y → 11, C → 10 | 11

Fails for: 10 11 01, XY pattern and C pattern got interleaved

Not working: The following is the method for even number of x y c
Match it to even number of X/Y pair (e.g., going out of a state as X, coming back as Y), and even number of C