Practice Questions

Problem 1

Which one of the following pairs are equal?

a) s = "aa", p = "a*"

b) s = "ab", p = ".*"

c) s = "aab", p = "c*a*b"

solution:

The given strings are equals in all three cases.

a) '*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it becomes "aa".

b) ".*" means "zero or more (*) of any character (.)".

c) c can be repeated 0 times, a can be repeated 1 time. Therefore, it matches "aab".

Problem 2

You are given a string s that consists of lower case English letters and brackets. Reverse the strings in each pair of matching parentheses, starting from the innermost one. Your result should not contain any brackets.

example 1:

```
Input: s = "(abcd)"
Output: "dcba"
```

example 2:

```
Input: s = "(gnimmargorp(love)i)"
Output: "iloveprogramming"
Explanation: The substring "love" is reversed first, then the
 whole string is reversed.
```

solution:

This problem can be solved using a stack. First, whenever a (is encountered then push the index of the element into the stack and whenever a) is encountered then get the top element of the stack as the latest index and reverse the string between the current index and index from the top of the stack. Follow this routine for the rest of the string and finally print the updated string.

```
1  #include "string"
2  #include "stack"
3  #include <iostream>
4  using namespace std;
5
6  // Function to return the modified string
7  string reverseParentheses(string str, int len) {
8      stack<char> st;
9
10     for (int i = 0; i < len; i++) {
11
12         // Push the index of the current
13         // opening bracket
14         if (str[i] == '(') {
15             st.push(i);
16         }
17
18         // Reverse the substring starting
19         // after the last encountered opening
20         // bracket till the current character
21         else if (str[i] == ')') {
22             reverse(str.begin() + st.top() + 1,
23                     str.begin() + i);
24             st.pop();
25         }
26     }
27
28     // To store the modified string
29     string res = "";
30     for (int i = 0; i < len; i++) {
31         if (str[i] != ')' && str[i] != '(')
32             res += (str[i]);
33     }
34     return res;
35 }
```

```
36
37 int main() {
38     string str = "(abc)";
39     int len = str.length();
40     string res =reverseParentheses(str, len);
41     cout << res;
42     return 0;
43 }
```

Problem 3

Describe as simply as possible in English the language corresponding to the regular expression $((a+b)^3)*(+a+b)$.

solution:

$((a+b)^3)$ represents the strings of length 3. Hence $((a+b)^3)*$ represents the strings of length a multiple of 3. Since $((a+b)^3)*(a+b)$ represents the strings of length 3n + 1, where n is a natural number, the given regular expression represents the strings of length $3n$ and $3n+1$, where $n \in \mathbb{N}$.

Problem 4

Find a regular expression corresponding to the language L defined recursively as follows:

Basis Clause: $\Lambda \in L$ and $a \in L$.
Inductive Clause: If $x \in L$, then $aabx \in L$ and $bbx \in L$.
Extremal Clause: Nothing is in L unless it can be obtained from the above two clauses.

solution:

Let us see what kind of strings are in L. First of all Λ and a are in L . Then starting with Λ or a, strings of L are generated one by one by prepending aab or bb to any of the already generated strings. Hence a string of L has zero or more of aab's and bb's in front possibly followed by a at the end. Thus $(aab+bb)*(a+\Lambda)$ is a regular expression for L.

Problem 5

Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.

example:

```
Input: [2,1,5,6,2,3]
Output: 10
```

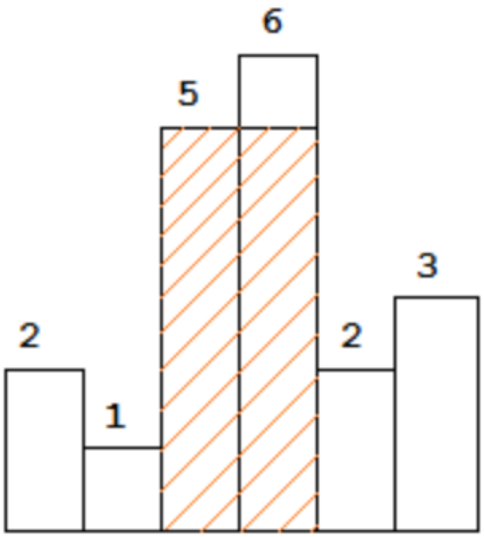

Figure 1: The largest rectangle is shown in the shaded area, which has area = 10 unit.

solution:

```
#include "iostream"
#include "vector"
#include "stack"
using namespace std;

class Solution {
public:
    int largestRectangleArea(vector<int>& height) {
        height.push_back(0);
```

```
        int n = height.size(), area = 0;
        stack<int> indexes;
        for (int i = 0; i < n; i++) {
            while (!indexes.empty() &&
                        height[indexes.top()] > height[i]) {
                int h = height[indexes.top()]; indexes.pop();
                int l = indexes.empty() ? -1 : indexes.top();
                area = max(area, h * (i - l - 1));
            }
            indexes.push(i);
        }
        return area;
    }
};


int main(){
    Solution obj;
    vector<int> hist = {2,1,5,6,2,3};
    int res = obj.largestRectangleArea(hist);
    cout << res;
};
```

Problem 6

Given N, Find the total number of unique BSTs that can be made using values from 1 to N.

example 1:

```
Input: n = 3
Output: 5
```

example 2:

```
Input: 4
Output: 14
```

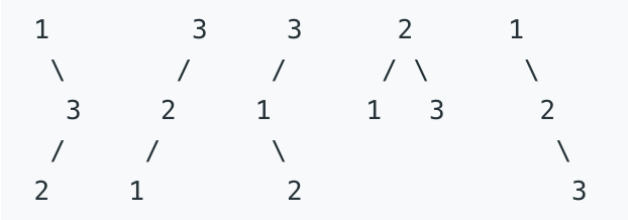

Figure 2: Unique BSTs for n = 3

solution:

```c
#include <stdio.h>

int numTrees(int n) {

  //dp[i] meaning total # of BST tree with i consecutive nodes
  int dp[n+1];
  memset(dp, 0, sizeof(int)*(n+1));
  dp[0] = 1; /* NULL is one tree */

  for(int i = 1; i<=n; i++) {
     /* the idea is for 1....i,  each node can be the
     root and count the full combination of left sub
     BST and right sub BST*/
    for(int j = 0; j<i ; j++) {
        dp[i]+= dp[j] * dp[i-j-1];
    }
  }
  return dp[n];
}

int main() {
    int n = 4;
    int res = numTrees(n);
    printf("%d", res);
    return 0;
}
```

Problem 7

draw a deterministic finite state machine which implements the specification.

1. Consider machines which accept a stream of 1 or more bits (the alphabet is limited to 1's and 0's), and whose output is 1 (accepting) or 0 (rejecting). Construct FSMs which implement the following operations:

a) OR
$c_1||c_2||\cdots||c_i$

b) AND
$c_1\&\&c_2\&\&\cdots\&\&c_i$

c) XOR
$c_1 \wedge c_2 \wedge \cdots \wedge c_i$

2.

d) Accepts strings containing CAT or DOG anywhere. [ACDGOT]

e) Accepts strings where $\#a > \#b$. [AB]

f) Accepts strings where $\#a = \#b$. [AB]

g) Accepts strings where ($\#a$ mod 3) = ($\#b$ mod 3). [AB]

solution:

a) Figure 3.

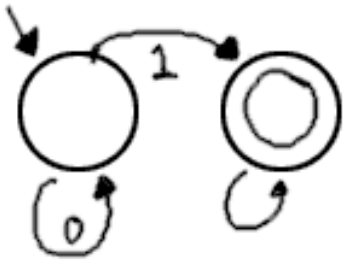

Figure 3: FSM for OR operation

b) Figure 4.

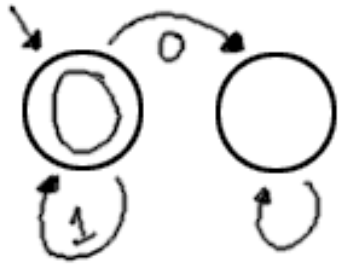

Figure 4: FSM for AND operation

c) Figure 5.

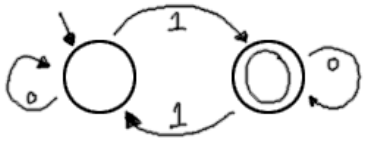

Figure 5: FSM for XOR operation

d) Figure 6.

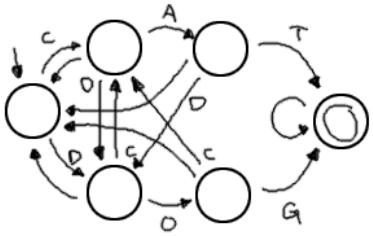

Figure 6: FSM for accepting CAT or DOG anywhere in the string

e) Impossible. Since $\#a$ is unbounded, this machine would need an infinite amount of states, or some infinite auxiliary storage.
A counting argument shows why: Since at any point in the processing of the string we can encounter any number of B's, we'd need to keep track of the number of A's encountered so far. This is an unbounded number. In order for a machine to keep track of exactly how many A's we've had, we'd need at least one state for each possibility.

But there are an infinite number of possibilities for $\#a$, so we can't do this with a finite state machine.

f) Impossible. Since the A's and B's can come in any order, this can't be done for the same reason as 4d.

g) In contrast to 4d and 4e, this machine is able to "count" since $\#a$ mod 3 and $\#b$ mod 3 are always bounded.

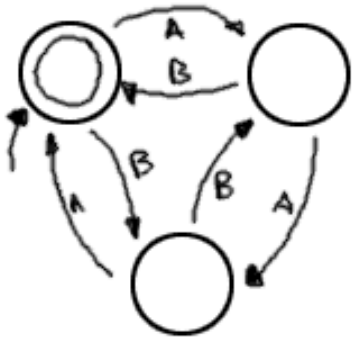

Figure 7: FSM that accept ($\#a$ mod 3) = ($\#b$ mod 3)

Problem 8

Given root of binary search tree and K as input, find K-th smallest element in BST.

For example, in the following BST, if k = 3, then output should be 10, and if k = 5, then output should be 14.

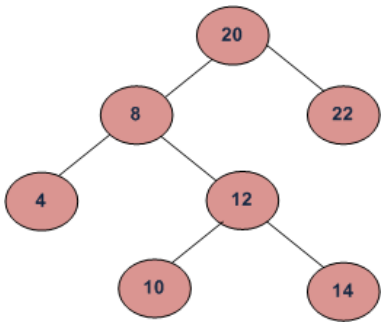

Figure 8: Third smallest value is 10.

solution:

Inorder traversal of BST retrieves elements of tree in the sorted order. The inorder traversal uses stack to store to be explored nodes of tree (threaded tree avoids stack and recursion for traversal, see this post). The idea is to keep track of popped elements which participate in the order statics.
Time complexity: O(n) where n is total nodes in tree.

Algorithm:

```
1  /* initialization */
2  pCrawl = root
3  set initial stack element as NULL (sentinal)
4
5  /* traverse upto left extreme */
6  while(pCrawl is valid )
7     stack.push(pCrawl)
8     pCrawl = pCrawl.left
9
10 /* process other nodes */
11 while( pCrawl = stack.pop() is valid )
12    stop if sufficient number of elements are popped.
13    if( pCrawl.right is valid )
14       pCrawl = pCrawl.right
15       while( pCrawl is valid )
16          stack.push(pCrawl)
17          pCrawl = pCrawl.left
```

Problem 9

Write the elements of the tree in the order they would be seen by pre-order, in-order, and post-order traversals.

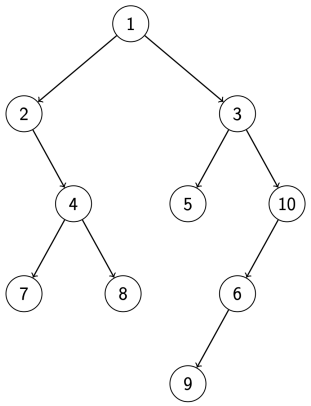

Figure 9: A binary Tree

solution:

Pre-order: 1, 2, 4, 7, 8, 3, 5, 10, 6, 9
In-order: 2, 7, 4, 8, 1, 5, 3, 9, 6, 10
Post-order: 7, 8, 4, 2, 5, 9, 6, 10, 3, 1