Generate Parentheses

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given n = 3, a solution set is:

```
[
        "((()))",
        "(()())",
        "(())()",
        "()(())",
        "()()()"
]
```

Solution in C++

```
class Solution {
public:

    void push_unique_pattern(vector<string> &r, map<string,int> &mp, string pattern) {
        if (!mp[pattern]) {
            r.push_back(pattern);
            mp[pattern]=1;
        }
    }

    vector<string> generateParenthesis(int n) {
        vector<string> r;
        map<string,int> mp;
        string s;

        if (n==1) r.push_back("()");
        else if (n>1) {
            vector<string> tmp=generateParenthesis(n-1);

            for(int i=0; i<tmp.size(); i++) {
                push_unique_pattern( r, mp, "(" + tmp[i] + ")");
            }

            for(int i=1; i<n; i++) {
                vector<string> tmp1=generateParenthesis(i);
                vector<string> tmp2=generateParenthesis(n-i);
                for(int j=0; j<tmp1.size(); j++)
                    for(int k=0; k<tmp2.size(); k++) {
                        push_unique_pattern( r, mp, tmp1[j] + tmp2[k]);
                        push_unique_pattern( r, mp, tmp2[k] + tmp1[j]);
                    }
            }
        }
        sort(r.begin(), r.end());
        return r;
    }
};
```

Solution in Java

```
class Solution {
    public List<String> generateParenthesis(int n) {
        List<String> combinations = new ArrayList();
        generateAll(new char[2 * n], 0, combinations);
        return combinations;
    }

    public void generateAll(char[] current, int pos, List<String> result) {
        if (pos == current.length) {
            if (valid(current))
                result.add(new String(current));
        } else {
            current[pos] = '(';
            generateAll(current, pos+1, result);
            current[pos] = ')';
            generateAll(current, pos+1, result);
        }
    }

    public boolean valid(char[] current) {
        int balance = 0;
        for (char c: current) {
            if (c == '(') balance++;
            else balance--;
            if (balance < 0) return false;
        }
        return (balance == 0);
    }
}
```

```
def generateParenthesis(self, N):
    if N == 0: return ['']
    ans = []
    for c in xrange(N):
        for left in self.generateParenthesis(c):
            for right in self.generateParenthesis(N-1-c):
                ans.append('({}){}'.format(left, right))
    return ans
```

Solution in Python

```
class Solution(object):
    def generateParenthesis(self, n):
        def generate(A = []):
            if len(A) == 2*n:
                if valid(A):
                    ans.append("".join(A))
            else:
                A.append('(')
                generate(A)
                A.pop()
                A.append(')')
                generate(A)
                A.pop()

        def valid(A):
            bal = 0
            for c in A:
                if c == '(': bal += 1
                else: bal -= 1
                if bal < 0: return False
            return bal == 0

        ans = []
        generate()
        return ans
```