USACO NOV12 Problem 'bbreeds' Analysis

by Jonathan Paulson

We'll call the two breeds A and B for convenience. Let the input string S = s_1s_2...s_n. We will give a dynamic programming algorithm working backwards from the end of the string.

Let f(i,A_open, B_open) be the number of ways to assign s_i...s_n to breeds such that the resulting two parentheses-strings are balanced, given that we have A_open unmatched left-parenthesis of type A and B_open unmatched left-parentheses of type B. If S[i]=='(', then f(i,A_open, B_open) = f(i+1, A_open+1, B_open) + f(i+1, A_open, B_open + 1), since we can assign the parenthesis to breed A or breed B. If S[i]==')', then we can assign the parenthesis to breed A as long as A_open > 0, and to B as long as B_open > 0.

The base case is i=n, in which case we processed the whole string without violating any invariants. As the total number of ')'s equals to the total number of '('s, we wil end up with two balanced strings of parentheses. Therefore we can start with so f(n, 0, 0) = 1. We have 0 <= i <= n, 0 <= A_open <= n, 0 <= B_open <= n, so the number of states is O(n^3), and there is O(1) non-recursive overhead for each state, so this leads to an O(n^3) solution.

Unfortunately, O(n^3) isn't fast enough with n=1,000. We can do better by noticing that B_open is uniquely determined by i and A_open (because A_open + B_open sums to the number of unmatched left-parentheses in s_1...s_{i-1}). So it suffices to keep track of (i, A_open), which gives O(n^2) states and an O(n^2) solution. This is fast enough. Here is my solution in Java:

```
import java.util.*;
import java.io.*;
import java.awt.Point;
import static java.lang.Math.*;

public class bbreeds {
    static int n;
    static char[] S;
    static int[] O;

    static void check(boolean b) { if(!b) throw new RuntimeException("data
invalid"); }
    public static void main(String[] args) throws Exception {
        Scanner in = new Scanner(new File("bbreeds.in"));
        PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter("bbreeds.out")));
        S = in.next().toCharArray();
        n = S.length;
        check(n <= 1000);
        for(int i=0; i<n; i++) check(S[i]=='(' || S[i]==')');
        O = new int[n+1];
        dp = new int[n][n];
        for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            dp[i][j] = -1;
```

```
        O[0] = 0;
        for(int i=0; i<n; i++)
            O[i+1] = O[i] + (S[i]=='('?1:-1);

        out.println(f(0, 0));
        out.flush();
    }

    static int[][] dp;
    static int f(int i, int A) {
        if(i == n) return 1;
        if(dp[i][A] >= 0) return dp[i][A];
        int B = O[i] - A;
        if(S[i] == '(') return dp[i][A] = (f(i+1,A+1)+f(i+1,A))%2012;
        else {
            int ans = 0;
            if(A > 0) ans += f(i+1, A-1);
            if(B > 0) ans += f(i+1, A);
            return dp[i][A] = ans%2012;
        }
    }
}
```

Mark Gordon's short C++ solution is also listed below:

```
#include <iostream>
#include <vector>
#include <cstring>
#include <cstdio>

using namespace std;

#define MOD 2012
#define MAXN 1010

int A[MAXN];

int main() {
  freopen("bbreeds.in", "r", stdin);
  freopen("bbreeds.out", "w", stdout);

  int L = A[1] = 1;
  for(int ch = cin.get(); L > 0 && ch == '(' || ch == ')'; ch = cin.get()) {
    int dir = ch == '(' ? 1 : -1;
    L += dir;
    for(int j = dir < 0 ? 1 : L; 1 <= j && j <= L; j -= dir) {
      A[j] += A[j - dir];
      if(A[j] >= MOD) A[j] -= MOD;
    }
    A[L + 1] = 0;
  }

  cout << (L == 1 ? A[1] : 0) << endl;
}
```