2013 Multi-University Training Contest 9

watashi

ゆっくりでいいさ

September 5, 2013

Arc of Dream

$$
\begin{cases}
1 & = & 1 \\
a_i & = & AX * a_{i-1} + AY * 1 \\
b_i & = & BX * b_{i-1} + BY * 1 \\
a_i b_i & = & (AX * a_{i-1} + AY)(BX * b_{i-1} + BY) \\
 & = & AX * BX * a_{i-1} b_{i-1} + AY * BY * 1 \\
 & & AX * BY * a_{i-1} + BX * AY * b_{i-1} + \\
AoD(i) & = & AoD(i-1) + a_{i-1} b_{i-1}
\end{cases}
$$

Arc of Dream

$$\begin{cases} 1 &= 1 \\ a_i &= AX * a_{i-1} + AY * 1 \\ b_i &= BX * b_{i-1} + BY * 1 \\ a_i b_i &= (AX * a_{i-1} + AY)(BX * b_{i-1} + BY) \\ &= AX * BX * a_{i-1} b_{i-1} + AY * BY * 1 \\ & \quad AX * BY * a_{i-1} + BX * AY * b_{i-1} + \\ AoD(i) &= AoD(i-1) + a_{i-1} b_{i-1} \end{cases}$$

$$\begin{pmatrix} 1 \\ a_n \\ b_n \\ a_n b_n \\ AoD(n) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ AY & AX & 0 & 0 & 0 \\ BY & 0 & BX & 0 & 0 \\ AY \cdot BY & AX \cdot BY & AY \cdot BX & AX \cdot BX & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}^n \begin{pmatrix} 1 \\ a_0 \\ b_0 \\ a_0 b_0 \\ 0 \end{pmatrix}$$

Boke and Tsukkomi

问题

找出所给一般图中一定不属于任何一个最大匹配的边。

Boke and Tsukkomi

问题

找出所给一般图中一定不属于任何一个最大匹配的边。

- 首先计算一般图最大匹配数 c
- 枚举每条边，计算删去该边两个顶点后的最大匹配数
- 如果匹配数小于 $c-1$，则说明该边不属于任何一个最大匹配

Cut the Cake

- 如果 $v_{i-1}v_i$ 和 v_jv_{j+1} 已经切好，那么切 v_kv_{k+1} 的时候，由于所切区域是凸的，新的切割线只会和 $v_{i-1}v_i$、v_jv_{j+1} 或切割前的切糕边缘相交，于是确定了 i, j, k 就唯一确定了切割线的长度 $cost_{i,j,k}$。

Cut the Cake

- 如果 $v_{i-1}v_i$ 和 v_jv_{j+1} 已经切好，那么切 v_kv_{k+1} 的时候，由于所切区域是凸的，新的切割线只会和 $v_{i-1}v_i$、v_jv_{j+1} 或切割前的切糕边缘相交，于是确定了 i, j, k 就唯一确定了切割线的长度 $cost_{i,j,k}$。
- 记 $dp_{i,j}$ 为果 $v_{i-1}v_i$ 和 v_jv_{j+1} 已经切好后，将折线 $v_iv_{i+1}\dots v_{j-1}v_j$ 切好的最小花费，则有
$$dp_{i,j} = \min\{dp_{i,k} + cost_{i,j,k} + dp_{k,j}\}.$$
于是通过 $O(n^3)$ 的动态规划可求得最优解。

Derangement

- 从第一个位置开始依次填数字，$N_{i,j}$ 表示处理到第 k 个位置时，前 k 个位置有 i 个是空余的，而前 k 个数字还有 j 个要填到后面的位置的方案数。一开始 $k = 0$ 时 $N_{0,0} = 1$。

Derangement

0 从第一个位置开始依次填数字，$N_{i,j}$ 表示处理到第 k 个位置时，前 k 个位置有 i 个是空余的，而前 k 个数字还有 j 个要填到后面的位置的方案数。一开始 $k=0$ 时 $N_{0,0}=1$。

+ 遇到'+'时有两种选择：
  - 暂时空着第 k 个位置：$N'_{i+1,j+1} += N_{i,j}$
  - 填入之前未填的数字：$N'_{i,j} += j * N_{i,j}$

Derangement

0 从第一个位置开始依次填数字，$N_{i,j}$ 表示处理到第 k 个位置时，前 k 个位置有 i 个是空余的，而前 k 个数字还有 j 个要填到后面的位置的方案数。一开始 $k = 0$ 时 $N_{0,0} = 1$。

+ 遇到 '+' 时有两种选择：
  - 暂时空着第 k 个位置：$N'_{i+1,j+1} += N_{i,j}$
  - 填入之前未填的数字：$N'_{i,j} += j * N_{i,j}$

− 遇到 '-' 时，我们总是要将第 k 个数填入之前的空余位置，同时对第 k 个位置还有以下两种选择：
  - 暂时空着第 k 个位置：$N'_{i,j} += i * N_{i,j}$
  - 填入之前未填的数字：$N'_{i-1,j-1} += i * j * N_{i,j}$

Derangement

0 从第一个位置开始依次填数字，$N_{i,j}$ 表示处理到第 k 个位置时，前 k 个位置有 i 个是空余的，而前 k 个数字还有 j 个要填到后面的位置的方案数。一开始 $k = 0$ 时 $N_{0,0} = 1$。

+ 遇到 `'+'` 时有两种选择：
  - 暂时空着第 k 个位置：$N'_{i+1,j+1} += N_{i,j}$
  - 填入之前未填的数字：$N'_{i,j} += j * N_{i,j}$

− 遇到 `'-'` 时，我们总是要将第 k 个数填入之前的空余位置，同时对第 k 个位置还有以下两种选择：
  - 暂时空着第 k 个位置：$N'_{i,j} += i * N_{i,j}$
  - 填入之前未填的数字：$N'_{i-1,j-1} += i * j * N_{i,j}$

? 最后的答案就是 $k = N$ 时的 $N_{0,0}$。

EBCDIC

直接构造映射表处理

```
while (scanf("%2X", &ord) != EOF) {
  printf("%02X", table[ord]);
}
puts("");
```

不建议人肉构造映射表。可以通过简单的程序或者较高级的文本编辑器处理网页得到映射表。当然也可以通过别的途径获得映射表。

Front compression

问题的核心在于求相邻两行的LCP。由于每一行都是字符串S的一个子串，所以

$$\mathrm{LCP}(S[s_i, t_i), S[s_j, t_j)) = \min\{\mathrm{LCP}(S[s_i, end), S[s_j, end)), t_i - s_i, t_j - s_j\}.$$

而计算两个后缀 $S[s_i, end)$ 和 $S[s_j, end)$ 的LCP，是后缀数组的一个基本应用。

Great Sequence

- 在数据范围内，`digit_sum` 不会超过 200，而且很容易计算出在 L 到 R 的范围内不同 `digit_sum` 的数各有多少个。

Great Sequence

- 在数据范围内，`digit_sum` 不会超过 200，而且很容易计算出在 L 到 R 的范围内不同 `digit_sum` 的数各有多少个。

- 在此基础之上，只要通过一个简单的 $O(max\_i * max\_digit\_sum)$ 复杂度的动态规划 `dp[last_i][last_digit_sum]` 就可以计算出有多少合法的序列了。

Great Sequence

- 在数据范围内，digit_sum 不会超过 200，而且很容易计算出在 L 到 R 的范围内不同 digit_sum 的数各有多少个。
- 在此基础之上，只要通过一个简单的 $O(max\_i * max\_digit\_sum)$ 复杂度的动态规划 dp[last_i][last_digit_sum] 就可以计算出有多少合法的序列了。
- 能够计算合法序列的个数，就可以通过二分求得字典序第 k 小的合法序列了。

Huge String

Implementation, BigInteger, String.Empty

Important Sisters

首先，对于那些从 #N 不可达的妹妹们，答案为 0，而其余的妹妹们则构成了一个以 #N 为起点的控制流程图。如果途中所有从 #N 到 #I 的路径都经过了 #K，那么我们称 #K 是 #I 的 dominator。

要求出所有的 dominator 关系，可以类似 Bellman-Ford 算法，对 N 个 `bitset` 进行迭代。这题用这种算法，并保证最后求和部分也利用 `bitset` 优化的话，是有可能在时限内 AC 的。

Important Sisters

首先，对于那些从#N不可达的妹妹们，答案为0，而其余的妹妹们则构成了一个以#N为起点的控制流程图。如果途中所有从#N到#I的路径都经过了#K，那么我们称#K是#I的dominator。

要求出所有的dominator关系，可以类似Bellman-Ford算法，对N个`bitset`进行迭代。这题用这种算法，并保证最后求和部分也利用`bitset`优化的话，是有可能在时限内AC的。

这种dominator的关系构成了一个dominator tree，树上的父亲节点是儿子节点的immediate dominator。

如果我们已经得到了dominator tree或者所有节点的immediate dominator，那么问题所要求的和只需一个简单的DFS即可实现。

dominator tree在很多领域都用重要运用，Lengauer-Tarjan算法可以在$O(m\alpha(m,n))$时间求得dominator tree，不过这题只要求$O(m\log(m))$的算法就足够了。

Jumping Frog

首先，青蛙子在往下跳的时候总要经过往上跳的时候的台阶。那么可以按往下跳所经过的台阶分为若干段，每段里，青蛙子都要通过往上跳一至多次完成。跨度为 k 的台阶的一段，对应的方案数 c_k 可以通过简单的动态规划求得。

有了这些 c_k 后，求总的方案数又是一个完全一样的动态规划

$$dp_n = \sum_k c_k dp_{n-k}.$$

Jumping Frog

首先，青蛙子在往下跳的时候总要经过往上跳的时候的台阶。那么可以按往下跳所经过的台阶分为若干段，每段里，青蛙子都要通过往上跳一至多次完成。跨度为 k 的台阶的一段，对应的方案数 c_k 可以通过简单的动态规划求得。

有了这些 c_k 后，求总的方案数又是一个完全一样的动态规划

$$dp_n = \sum_k c_k dp_{n-k}.$$

只不过这里的 n 太大，$O(nm)$ 的动态规划显然是行不通的。而像这样的线性递推式，我们已经非常熟悉将它转为矩阵乘法，在利用快速幂运算把复杂度降到 $(m^3 \log(n)$ 的处理方法了。但这里的 m 和 n 还是太大了，事实上，对于线性递推式的情况，我们还可以做得更好。

专栏 (更快地计算递推式——《挑战程序设计竞赛》P201)

事实上，对于 m 项递推式的第 n 项的求解可以不使用矩阵，而是使用初项的线性表示，通过快速幂在 $O(m^2 \log(n))$ 的时间内求出答案。有兴趣的读者可以试着思考看看。

Q&A

$\Rightarrow$ *Comments*