Kick Start 2017 - Round F

Analysis: Catch Them All

Catch Them All: Analysis

Small dataset

We can start by computing the shortest distance between each pair of locations using the Floyd-Warshall algorithm. We will use dis[i, j] to represent the shortest distance between locations i and j.

Let dp[K, L] be the expected time needed to catch K Codejamons when starting from location L. Then we can use a dynamic programming algorithm with the following state transition equation:

```
if (K == 0):
  dp[K, L] = 0;
else:
  dp[K, L] = Σi!=L(dp[K-1, i] + dis[L, i]) / (N-1).
```

The algorithm above takes $O(\mathbf{N}^2\mathbf{P})$ time, which is fast enough to solve the Small dataset.

Large dataset

We can find that for each dp[K, L], the answer is a linear expression of dp[K-1, i] when K != 0. So, we can rewrite the state transition equation as the product of a matrix and a column vector, as shown below.

Let $S[i] = \Sigma_{j!=i}(dis[i, j])$.

```
+----------+   +----------------------------------------------+   +------------+
| dp[K, 1] |   |    0    , 1/(N-1), ... , 1/(N-1), S[1]/(N-1) |   | dp[K-1, 1] |
| dp[K, 2] |   | 1/(N-1),     0   , ... , 1/(N-1), S[2]/(N-1) |   | dp[K-1, 2] |
|   ...    | = |   ...  ,    ...  , ... ,   ...  ,     ...    | * |    ...     |
| dp[K, N] |   | 1/(N-1), 1/(N-1), ... ,    0   , S[N]/(N-1) |   | dp[K-1, N] |
|    1     |   |    0    ,    0    , ... ,    0   ,      1     |   |     1      |
+----------+   +----------------------------------------------+   +------------+
```

Let F_K denote the column vector of dp[K, i], and let A denote the transition matrix. Then we have $F_K = A * F_{K-1} = A^K * F_0$.

With the approach above, we can use exponentiation by squaring to accelerate the computation of A^K. This gives us an $O(\mathbf{N}^3\log\mathbf{P})$ algorithm which can solve the Large dataset.

Other solutions

Let Pr_t be the probability of being at location 1 after catching t Codejamons. Initially, $Pr_0 = 1$. Since at any time, the probabilities of being at locations 2, 3, ..., **N** (let's call them the "other locations") are the same, we can calculate the probability of the next Codejamon appearing at location 1 by multiplying the probability of being at any of the other locations by the probability of location 1 being chosen. Therefore, we have $Pr_t = (1 - Pr_{t-1}) / (\mathbf{N}-1)$.

After computing the values of Pr_i for i = 1, 2,..., **P**-1, the answer to the problem is:

$\Sigma(Pr_i$ * (expected distance from location 1) + $\Sigma(1 - Pr_i) / (N-1)$ * (expected distance from location j) for j = 2, 3, ..., N) for i = 0, 1, ..., **P**-1.

Note: the expected distance from location i equals $\Sigma dis[i, j] / (\mathbf{N}-1)$ for j = 1, 2, ..., **N**.

We might not have enough time to compute all **P** values of the sequence Pr_i, but one may notice that this sequence converges quickly (except for **N**=2, which we can handle separately). Intuitively, as the game progresses, the probabilities of you being at each of the **N** locations become more equal. For example, if **N**=4, the first few values for Pr_t are 1, 0, 0.333, 0.222, 0.259, 0.247, 0.251, 0.250, ... Once this value gets very close to 1 / **N**, after some threshold like i = 100 (depending on the numerical error allowed), we can simply approximate Pr_i = 1/**N** for all i larger than the threshold. Then i = 100, 101, ..., **P**-1 in the previous summation can be replaced by a multiplication: (**P**-100) * Σ(expected distance from location j) / **N** for j = 1, 2, ..., **N**.

The time complexity for the above algorithm is O($\mathbf{N}^3$ + C), where C depends on the numerical error allowed.

Using the sequence of Pr_t values, it is also possible to calculate the exact answer. Let's make another sequence $A_i = Pr_i - 1/\mathbf{N}$. This sequence converges to 0 and is a geometric progression because:

- $Pr_i = (1 - Pr_{i-1}) / (\mathbf{N}-1)$
- $A_i + 1/\mathbf{N} = (1 - A_{i-1} - 1/\mathbf{N}) / (\mathbf{N}-1)$
- $A_i + 1/\mathbf{N} = -A_{i-1} / (\mathbf{N}-1) + 1/\mathbf{N}$
- $A_i = -A_{i-1} / (\mathbf{N}-1)$.

ΣA_i for i = 0, 1, ..., **P**-1 can be calculated using the formula for the sum of a geometric series. With ΣA_i, we can obtain $\Sigma Pr_i = 1/\mathbf{N} * \mathbf{P} + \Sigma A_i$, and ultimately the answer. The time complexity for the above algorithm is O($\mathbf{N}^3$).