CS 577: Introduction to Algorithms — Homework 7

Out: 03/31/17 — **Due: 04/07/17**

Ground Rules

- Review problems will be discussed at the following week's review.
- Self-graded problems will be discussed at the following week's review, and are to be self-graded by you during the review. Only students present at the review session and turning in their graded solutions in person will earn points.
- Graded problems are to be turned in on the due date at the beginning of the lecture.
- Self-graded problems should be done individually.
- Graded problems should be done in pairs.
- Please follow page limits for all self-graded and graded problems.
- Write your name(s) clearly on your submissions, and turn in each problem on **a separate sheet of paper**.

Review Problems

1. Consider the following family of hash functions. We will think of every element in the universe as being a k-bit vector (so, $k = \log |U|$), and every index in the hash table as being an ℓ-bit vector (so, $\ell = \log m$ where m is the size of the hash table).

   Each function h_M in the hash family is parameterized by a $\{0, 1\}$ matrix M of size $k \times \ell$. The function h_M maps an element x in U to the vector $h_M(x) = M \cdot x$ where $\cdot$ is the matrix product operation with multiplications and additions performed modulo 2.

   Prove that this family is a universal family of hash functions.

2. You are given a sorted circular linked list containing n integers, where every element has a "next" pointer to the next larger element. (The largest element's "next" pointer points to the smallest element.) You are asked to determine whether a given target element belongs to the list. The only way you can access an element of the list is to follow the next pointer from a previously accessed element, or via the function RAND that returns a random element of the list.

   Develop a randomized algorithm for finding the target that accesses at most $O(\sqrt{n})$ elements in expectation. If the target is present in the list, your algorithm should return a pointer to it, and if it is not, your algorithm should return "Error".

Self-graded Problems

3. **(Page limit: 1 sheet; 2 sides)** Let us modify binary search in the following way. Given a sorted list of n elements and a target to search for, instead of querying the median of the list, we compare the target against a uniformly random element, and then recurse. Prove that in the worst case over the input, the expected number of comparisons the algorithm makes is still $O(\log n)$.

   Hint: Use indicator random variables to count the number of comparisons. The probability that the target is compared against a particular element should depend on the location of the element.

Graded Problems

4. **(Page limit: 1 sheet; 2 sides)** (Taken from "Algorithm Design") A small business—say, a photocopying service with a single large machine—faces the following scheduling problem. Each morning they get a set of jobs from customers. They want to do the jobs on their single machine in an order that keeps their customers happiest. Customer i's job will take t_i time to complete. Given a schedule (i.e., an ordering of the jobs), let C_i denote the finishing time of job i. For example, if job j is the first to be done, we would have $C_j = t_j$; and if the job j is done right after job i, we would have $C_j = C_i + t_j$. Each customer i also has a given weight w_i that represents his or her importance to the business. The happiness of customer i is expected to be dependent on the finishing time of i's job. So the company decides that they want to order the jobs to minimize the weighted sum of the completion times, $\sum_{i=1}^{n} w_i C_i$.

   Design a polynomial time algorithm to solve this problem. Your algorithm should take as input the set of n jobs with processing times t_i and weights w_i for each, and return as output the ordering that minimizes the weighted sum $\sum_{i=1}^{n} w_i C_i$. Prove the correctness of your algorithm and analyze its running time.

   Hint: Use a greedy strategy.

5. **(Page limit: 1 sheet; 2 sides)** Give a polynomial time algorithm for the attached ICPC problem "The Lost House". We are asking for an algorithm, not a working program. Describe your algorithm concisely. Provide a proof of correctness and an argument for its running time.

3141 - The Lost House

Asia - Beijing - 2004/2005

One day a snail climbed up to a big tree and finally came to the end of a branch. What a different feeling to look down from such a high place he had never been to before! However, he was very tired due to the long time of climbing, and fell asleep. An unbelievable thing happened when he woke up he found himself lying in a meadow and his house originally on his back disappeared! Immediately he realized that he fell off the branch when he was sleeping! He was sure that his house must still be on the branch he had been sleeping on. The snail began to climb the tree again, since he could not live without his house.

When reaching the first fork of the tree, he sadly found that he could not remember the route that he climbed before. In order to find his lovely house, the snail decided to go to the end of every branch. It was dangerous to walk without the protection of the house, so he wished to search the tree in the best way.

Fortunately, there lived many warm-hearted worms in the tree that could accurately tell the snail whether he had ever passed their places or not before he fell off.

Now our job is to help the snail. We pay most of our attention to two parts of the tree the forks of the branches and the ends of the branches, which we call them key points because key events always happen there, such as choosing a path, getting the help from a worm and arriving at the house he is searching for.

Assume all worms live at key points, and all the branches between two neighboring key points have the same distance of 1. The snail is now at the first fork of the tree.

Our purpose is to find a proper route along which he can find his house as soon as possible, through the analysis of the structure of the tree and the locations of the worms. The only restriction on the route is that he must not go down from a fork until he has reached all the ends grown from this fork.

The house may be left at the end of any branches in an equal probability. We focus on the mathematical expectation of the distance the snail has to cover before arriving his house. We wish the value to be as small as possible.

As illustrated in Figure-1, the snail is at the key point 1 and his house is at a certain point among 2, 4 and 5. A worm lives at point 3, who can tell the snail whether his house is at one of point 4 and 5 or not. Therefore, the snail can choose two strategies. He can go to point 2 first. If he cannot find the house there, he should go back to point 1, and then reaches point 4 (or 5) by point 3. If still not, he has to return point 3, then go to point 5 (or 4), where he will undoubtedly find his house. In this choice, the snail covers distances of 1, 4, 6 corresponding to the circumstances under which the house is located at point 2, 4 (or 5), 5 (or 4) respectively. So the expectation value is $(1 + 4 + 6) / 3 = 11 / 3$. Obviously, this strategy does not make full use of the information from the worm. If the snail goes to point 3 and gets useful information from the worm first, and then chooses to go back to point 1 then towards point 2, or go to point 4 or 5 to take his chance, the distances he covers will be 2, 3, 4 corresponding to the different locations of the house. In such a strategy, the mathematical expectation will be $(2 + 3 + 4) / 3 = 3$, and it is the very route along which the snail should search the tree.

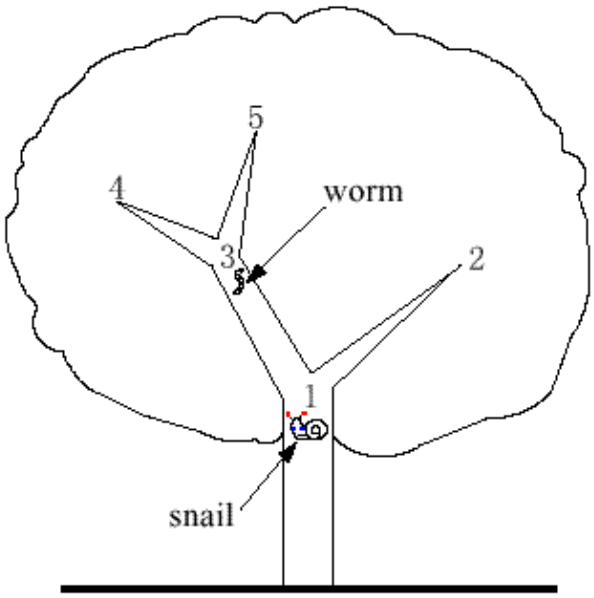

Figure-1

Input

The input contains several sets of test data. Each set begins with a line containing one integer N, no more than 1000, which indicates the number of key points in the tree. Then follow N lines describing the N key points. For convenience, we number all the key points from 1 to N. The key point numbered with 1 is always the first fork of the tree. Other numbers may be any key points in the tree except the first fork. The i-th line in these N lines describes the key point with number i. Each line consists of one integer and one uppercase character 'Y' or 'N' separated by a single space, which represents the number of the previous key point and whether there lives a worm ('Y' means lives and 'N' means not). The previous key point means the neighboring key point in the shortest path between this key point and the key point numbered 1. In the above illustration, the previous key point of point 2 or 3 is point 1, while the previous key point of point 4 or 5 is point 3. This integer is -1 for the key point 1, means it has no previous key point. You can assume a fork has at most eight branches. The first set in the sample input describes the above illustration.

A test case of N = 0 indicates the end of input, and should not be processed.

Output

Output one line for each set of input data. The line contains one float number with exactly four digits after the decimal point, which is the mathematical expectation value.

Sample Input

```
5
-1 N
1 N
1 Y
3 N
3 N
10
-1 N
1 Y
1 N
2 N
```

```
2 N
2 N
3 N
3 Y
8 N
8 N
6
-1 N
1 N
1 Y
1 N
3 N
3 N
0
```

Sample Output

```
3.0000
5.0000
3.5000
```

Beijing 2004-2005