|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| http://ace.delos.com/usaco/cowhead2.gif | |  | | --- | | Contest: MAR11 **GOLD** Division | |  | |  | |  | |

**ANALYSIS MODE  
Submit solutions for your own enjoyment.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**GOLD PROBLEMS**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Three problems numbered 1 through 3**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Problem 1: Brownie Slicing [Haitao Mao, Jacob Steinhardt, and Paul Christiano, 2011]**

**Bessie has baked a rectangular brownie that can be thought of as**

**an RxC grid (1 <= R <= 500; 1 <= C <= 500) of little brownie squares.**

**The square at row i, column j contains N\_ij (0 <= N\_ij <= 4,000)**

**chocolate chips.**

**Bessie wants to partition the brownie up into A\*B chunks (1 <= A**

**<= R; 1 <= B <= C): one for each of the A\*B cows. The brownie is**

**cut by first making A-1 horizontal cuts (always along integer**

**coordinates) to divide the brownie into A strips. Then cut each**

**strip \*independently\* with B-1 vertical cuts, also on integer**

**boundaries. The other A\*B-1 cows then each choose a brownie piece,**

**leaving the last chunk for Bessie. Being greedy, they leave Bessie**

**the brownie that has the least number of chocolate chips on it.**

**Determine the maximum number of chocolate chips Bessie can receive,**

**assuming she cuts the brownies optimally.**

**As an example, consider a 5 row x 4 column brownie with chips**

**distributed like this:**

**1 2 2 1**

**3 1 1 1**

**2 0 1 3**

**1 1 1 1**

**1 1 1 1**

**Bessie must partition the brownie into 4 horizontal strips, each**

**with two pieces. Bessie can cut the brownie like this:**

**1 2 | 2 1**

**---------**

**3 | 1 1 1**

**---------**

**2 0 1 | 3**

**---------**

**1 1 | 1 1**

**1 1 | 1 1**

**Thus, when the other greedy cows take their brownie piece, Bessie**

**still gets 3 chocolate chips.**

**PROBLEM NAME: brownie**

**INPUT FORMAT:**

**\* Line 1: Four space-separated integers: R, C, A, and B**

**\* Lines 2..R+1: Line i+1 contains C space-separated integers: N\_i1,**

**..., N\_iC**

**SAMPLE INPUT (file brownie.in):**

**5 4 4 2**

**1 2 2 1**

**3 1 1 1**

**2 0 1 3**

**1 1 1 1**

**1 1 1 1**

**OUTPUT FORMAT:**

**\* Line 1: A single integer: the maximum number of chocolate chips that**

**Bessie guarantee on her brownie**

**SAMPLE OUTPUT (file brownie.out):**

**3**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Problem 2: Graph Discovery [Paul Christiano, 2011]**

**Bessie has a created a puzzle for Farmer John. In front of him there**

**is a lake with N (1 <= N <= 40) small islands, with bridges between**

**some pairs of islands. Bessie has agreed to tell FJ if it is possible**

**to safely get from any island to any other island \*without\* using**

**a specified set of bridges.**

**That is, if we think of the islands as a graph, with bridges**

**corresponding to edges, then Bessie will tell FJ if all N vertices**

**are connected \*after removing a specified subset of the edges\*. (It**

**is guaranteed that the initial graph is connected.)**

**FJ would like to determine exactly which bridges exist. Help FJ**

**figure this out using as few questions as possible (see below for**

**a detailed description of the scoring procedure).**

**An example dialogue between Bessie and FJ might be as follows,**

**assuming that the collection of islands corresponds to the graph**

**on 4 vertices consisting of edges {{1,2}, {1,3}, {1,4}, {2,3}}**

**(depicted below):**

**1--2**

**|\ |**

**| \|**

**4 3**

**FJ's query | Bessie's response | Comments**

**------------------------------------------------------------------------**

**{{1,2}} | Yes |**

**{{3,4}} | Yes |**

**{{1,4}, {4,3}} | No | {1,4} must be an edge**

**{{1,2}, {2,3}} | No | {2,3} must be an edge**

**{{1,2}, {3,1}} | No | {1,3} must be an edge**

**{{1,3}} | Yes | {1,2} must be an edge**

**{{1,4}} | No | {2,4} and {3,4} are not edges**

**FJ can then conclude that the graph has edges {{1,2}, {1,3}, {1,4},**

**{2,3}}, and no others.**

**This problem is a reactive problem, meaning that instead of reading**

**and writing to a file you will use stdin and stdout (in other words,**

**console input and output) to interact with a grader program. See**

**the input description for important information about interactive**

**problems.**

**At the beginning of execution, the grader program will write a**

**single integer, N, the number of vertices. You will then write lines**

**with one of the following three forms:**

**R i j**

**U i j**

**Q**

**where R, U, and Q are the characters 'R', 'U', and 'Q', and i and**

**j are integers in the range 1..N. The first sort of query removes**

**the edge between vertices i and j (if it exists). The second undoes**

**the previous removal of an edge between i and j. The third asks**

**whether the graph is connected; after you output Q, the grader will**

**output either 0 (for not connected) or 1 (for connected).**

**When your program is ready to output the graph, you should output**

**a line with the single character 'A', then N lines, each with N**

**characters. The jth number on the ith of these lines should be 1**

**if vertices i and j are adjacent, and 0 otherwise (a vertex is never**

**adjacent to itself).**

**If you output an incorrect graph at the end, you will receive 0**

**points. Otherwise, you will receive points based on the number of**

**times you output 'Q'. If you output 'Q' at most 900 times then you**

**will receive 100% of the points. If you output 'Q' K times for some**

**900 < K <= 5000, then you will receive a percentage of the points**

**equal to 20+80\*(900/K). If you output 'Q' more than 5000 times, you**

**will receive 0 points.**

**A dialogue corresponding to the example above is as follows (<**

**indicates the grader's output, > indicates your program's output;**

**these symbols are for clarity only and not part of the actual**

**output).**

**I/O | Set of removed edges**

**----------------------------------**

**< 4 |**

**> R 1 2 | {{1,2}}**

**> Q |**

**< 1 |**

**> U 1 2 | {}**

**> R 3 4 | {{3,4}}**

**> Q |**

**< 1 |**

**> R 4 1 | {{3,4}, {4,1}}**

**> Q |**

**< 0 |**

**> U 3 4 | {{4,1}}**

**> U 1 4 | {}**

**> R 1 2 | {{1,2}}**

**> R 2 3 | {{1,2}, {2,3}}**

**> Q |**

**< 0 |**

**> U 3 2 | {{1,2}}**

**> R 3 1 | {{1,2}, {3,1}}**

**> Q |**

**< 0 |**

**> U 1 2 | {{3,1}}**

**> Q |**

**< 1 |**

**> U 3 1 | {}**

**> R 1 4 | {{1,4}}**

**> Q |**

**< 0 |**

**> A |**

**> 0111 |**

**> 1010 |**

**> 1100 |**

**> 1000 |**

**TIME LIMIT: 2 seconds**

**Interactive programs usually require extra code that causes output**

**to be unbuffered -- to be written in real time instead of buffering**

**for faster (but later) output.**

**Those C/C++ users who use #include <stdio.h> should execute this**

**line before any input or output:**

**setlinebuf (stdout);**

**Users of <stdio.h> should also use fgets () to read from stdin.**

**Use of scanf is not recommended; do something like this to parse**

**input data:**

**char line[1000];**

**setlinebuf (stdout);**

**fgets (line, 1000, stdin);**

**sscanf (line, "..format..", &var1, ...);**

**/\* if the line contents need to be interpreted \*/**

**Those C++ users who use iostream should cout << flush after each**

**line (and also use cin in the normal manner):**

**cout << foo << endl;**

**cout << flush;**

**Be sure when you read the response from the computer that you read**

**\*all\* the letters, not just the first one. The response will never**

**be more than one letter + one newline + one string terminator ('\0').**

**Java users should use the following output scheme for output:**

**import java.io.\*;**

**...**

**PrintStream out = new PrintStream (System.out, true); // 'unbuffers' output**

**...**

**// sample integer print:**

**out.println (foo);**

**For Pascal, use the following scheme for writing:**

**writeln (stdout, ...your output here...); flush(stdout);**

**Be sure to read in the entire reply -- make room for the letter,**

**the newline, and the string terminator.**

**Despite the references to gdisc.in and gdisc.out here and below, no files are**

**used for input or output.**

**PROBLEM NAME: gdisc**

**INPUT FORMAT:**

**SAMPLE INPUT (file gdisc.in):**

**OUTPUT FORMAT:**

**SAMPLE OUTPUT (file gdisc.out):**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Problem 3: Tree Decoration [Jacob Steinhardt, 2011]**

**Farmer John is decorating his Spring Equinox Tree (like a Christmas**

**tree but popular about three months later). It can be modeled as a**

**rooted mathematical tree with N (1 <= N <= 100,000) elements, labeled**

**1...N, with element 1 as the root of the tree. Each tree element e**

**> 1 has a parent, P\_e (1 <= P\_e <= N). Element 1 has no parent**

**(denoted '-1' in the input), of course, because it is the root of**

**the tree.**

**Each element i has a corresponding subtree (potentially of size 1)**

**rooted there. FJ would like to make sure that the subtree corresponding**

**to element i has a total of at least C\_i (0 <= C\_i <= 10,000,000)**

**ornaments scattered among its members. He would also like to minimize**

**the total amount of time it takes him to place all the ornaments**

**(it takes time K\*T\_i to place K ornaments at element i (1 <= T\_i**

**<= 100)).**

**Help FJ determine the minimum amount of time it takes to place**

**ornaments that satisfy the constraints. Note that this answer might**

**not fit into a 32-bit integer, but it will fit into a signed 64-bit**

**integer.**

**For example, consider the tree below where nodes located higher on**

**the display are parents of connected lower nodes (1 is the root):**

**1**

**|**

**2**

**|**

**5**

**/ \**

**4 3**

**Suppose that FJ has the following subtree constraints:**

**Minimum ornaments the subtree requires**

**| Time to install an ornament**

**Subtree | |**

**root | C\_i | T\_i**

**--------+--------+-------**

**1 | 9 | 3**

**2 | 2 | 2**

**3 | 3 | 2**

**4 | 1 | 4**

**5 | 3 | 3**

**Then FJ can place all the ornaments as shown below, for a total**

**cost of 20:**

**1 [0/9(0)] legend: element# [ornaments here/**

**| total ornaments in subtree(node install time)]**

**2 [3/9(6)]**

**|**

**5 [0/6(0)]**

**/ \**

**[1/1(4)] 4 3 [5/5(10)]**

**PROBLEM NAME: tdec**

**INPUT FORMAT:**

**\* Line 1: A single integer: N**

**\* Lines 2..N+1: Line i+1 contains three space-separated integers: P\_i,**

**C\_i, and T\_i**

**SAMPLE INPUT (file tdec.in):**

**5**

**-1 9 3**

**1 2 2**

**5 3 2**

**5 1 4**

**2 3 3**

**OUTPUT FORMAT:**

**\* Line 1: A single integer: The minimum time to place all the**

**ornaments**

**SAMPLE OUTPUT (file tdec.out):**

**20**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***