Problem A
k-ary Tree Game
Input file: *pa.txt*

Problem Statement

The k-ary tree game is played by two players, you and your opponent. The two players alternate in hacking away edges from a complete k-ary tree and removing those pieces of the tree that are no longer connected to the root node. Initially, You move first and you are given a complete k-ary tree with n edges, numbered from1 to n, where an edge at index i has (at most) k children-edges at indexes (k·i+1), (k·i+2), …, and (k·i+k) and a parent-edge at index $\lfloor (i-1)/k \rfloor$. All leaf nodes of this k-ary tree are at some depth d or d-1, and all leaves at depth d are toward the left. The first player who has no legal move (no edge to remove) loses. You may assume that your opponent play optimally. Your task is to find all winning moves for the initial configuration.

For example, the following figure shows the game tree for a complete 2-ary (i.e. binary) tree with n=5 edges. Initially, you have 5 possible moves, that is, removing edges 1, 2, 3, 4, or 5. Your opponent moves next. In this figure, we only show the complete results for the leftmost 3 subtrees. The remaining 2 subtrees can be derived in a similar way. From this figure, we can see that you have exactly 3 winning moves: removing edges 3, 4, or 5. The remaining two moves (removing edges 1 or 2) will let you lose the game. In this figure, we use bold arrows to denote the winning moves for any player in that configuration.

Given a complete k-ary tree with n edges, please write a program to find all of your winning moves in the initial configuration. We assume that $2 \le k \le 5$ and $k \le n \le 1000$.

Input File Format

The first line of the input file contains an integer N, $1 \le N \le 10$, indicating the number of test cases. There is a single line containing a '/' character separating two consecutive test cases. The last line of the input file contains a '.' character denoting the end of the input file.

For each of the test cases, a complete k-ary tree is given. Each tree is described in terms of *k* and *n*, where k is the number of degree and *n* is the number of edges.

Output Format

The output data contains the answers for all test sets. Each test set ends with a "/". However, the last test set ends with double slashes '//'. For each test set, output all the winning moves in a line, from small edge index to large edge index. In case of no winning moves, please just output a number "0" in a line.

Sample Input

```
4
2 5
/
2 4
/
3 7
```

```
/
5 7
.
```

Sample Output

```
3   4   5
/
0
/
3
/
1   2   3   4   5
//
```

Problem B
Conflict Analysis
Input file: *pb.txt*

Problem Statement

Conflict analysis is a field whose importance is increasing as distributed computer systems becomes more and more prevalent. Conflict analysis may be applied to many different areas where a conflict can arise, such as business, government, political and military operations, and labor-management negotiations. A typical conflict situation consists of a set of agents (parties) and a set of issues. Each agent may be *for* or *against* an issue. Assume that we have n agents and m issues. A conflict situation can be represented as an $n \times m$ Boolean-valued matrix C, where $C(i,j)=1$ means that agent i is *for* issue j, and $C(i,j)=0$ means that agent i is *against* issue j.

For example, in the Middle-East conflict, let us consider six parties: 1. Israel, 2. Egypt, 3.Palestinians, 4.Jordan, 5.Syria, and 6.Saudi Arabia.

Five issues that effect the relationship between these parties are::

1. Autonomous Palestinian state on the West Bank and Gaza
2. Israeli military outpost along the Jordan river
3. Israel retaining East Jerusalem
4. Israeli military outposts on the Golan Heights
5. Arab countries granting citizenship to Palestinians who choose to remain within their borders.

The attitudes of these six parties toward these five issues are represented in the following matrix:

0	1	1	1	1
1	1	0	1	0
1	0	0	1	1
1	0	0	1	0
1	0	0	0	0
1	1	0	1	1

To analyze the causes of the conflict, we must consider the importance of each issue. For a subset of issues $S \subseteq \{1, 2, \ldots, m\}$ and an issue $j \in S$, we said that j is a secondary issue in S if any two agents who agree on all issues in $S-\{j\}$ also agree on issue j. Otherwise, j is a primary issue in S. Formally, j is a secondary issue in S if

for any $1 \le i_1, i_2 \le n$, $C(i_1, k) = C(i_2, k)$ for all $k \in S-\{j\}$ implies $C(i_1, j) = C(i_2, j)$.

A subset of issues $S \subseteq \{1, 2, \ldots, m\}$ is called an *independent set* if each issue in S is a primary issue in S. An independent set S is called a *reduct* if any two agents who agree on all issues in S also agree on all issues not in S. In general, a conflict situation may have several different reducts. Let R_1 and R_2 be two reducts of a conflict situation, we say that R_1 is better than R_2 if

$$\min(R_1 - R_2) > \min(R_2 - R_1).$$

For example, in the Middle-East situation, we have two reducts {1,2,4,5} and {2,3,4,5}. {2,3,4,5} is better than {1,2,4,5} since

$$\min(\{2,3,4,5\}-\{1,2,4,5\})=\min(\{3\})=3> \min(\{1,2,4,5\}-\{2,3,4,5\})=\min(\{1\}).$$

This problem is to compute the best reduct for a given conflict situation.

Definitions and Constraints:

1. Let $S \subseteq \{1, 2, \ldots, m\}$ be a subset of issues, then an issue $j \in S$ is a secondary issue in S if for any $1 \le i_1, i_2 \le n$,

   $$C(i_1, k) = C(i_2, k) \text{ for all } k \in S-\{j\} \text{ implies } C(i_1, j) = C(i_2, j).$$

2. j is a primary issue in S if it is not a secondary issue in S.
3. A subset of issues $S \subseteq \{1, 2, \ldots, m\}$ is called an *independent set* if each issue in S is a primary issue in S.
4. An independent set S is called a *reduct* if for any $1 \le i_1, i_2 \le n$,

   $C(i_1, k) = C(i_2, k)$ for all $k \in S$ implies $C(i_1, j) = C(i_2, j)$ for all $j \in \{1, 2, \ldots, m\}$

5. Let R_1 and R_2 be two reducts of a conflict situation, R_1 is better than R_2 if

   $$\min(R_1 - R_2) > \min(R_2 - R_1)$$

Input File Format

The first line of the input file contains an integer N, $1 \le N \le 10$, indicating the number of test cases. There is a single line containing a '/' character separating two consecutive test cases. The last line of the input file contains a '.' character denoting the end of the input file.

For each of the test cases, the first line contains two positive integers n and m (separated by blank) denoting the number of agents and the number of issues respectively. Following the first line, there are n lines that form the conflict situation matrix. Each line contains a sequence of 0s and/or 1s of length m, representing the attitudes of the agent toward each issue.

Output Format

For each test case of the input, compute the best reduct, which is a subset of $\{1,2,\ldots,m\}$, for the conflict situation.

Sample Input

```
2
6 5
01111
11010
10011
10010
10000
11011
/
10 7
```

```
1111110
0110000
1101101
1111001
0110011
1011011
1011111
1110000
1111111
1111011
.
```

Sample Output

```
{2,3,4,5}
{1,2,5,6,7}
```

Problem C
Plan Reduction
Input file: *pc.txt*

Problem Statement

A U.S. company X has many subdivisions in the world. After the tragedy of 911, company X wants every subdivision plots an emergency plan when terrorists strike again. Later, these plans are collected at X's global site. A manager is responsible for merging these plans and removing the redundant portions of the merged plan to ensure the company exercises this plan at its minimum cost.

Initially, every subdivision is in a status '0'. A status 0 indicates that subdivision is operating at its site peacefully. Company X categorizes the emergency events into 26 kinds, ranging from 'a' to 'z'. Some emergency events may only happen to some subdivisions because of their locations or environment. Some events, on the other hand, may be included in many subdivision plans.

For example, a subdivision X1 comes up with a plan like Fig. 1.

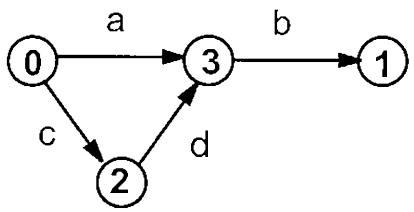

Figure 1: plan X1.

In plan X1, when an emergency event *a* happens, subdivision X1 shall evacuate its employees to nearby site 3. If the terrorists strike hard and trigger another emergency event *b,* subdivision X1 shall evacuate all its employees from site 3 to X in U.S. Status '1' indicate all the employees are moved back to U.S. and all the functions of subdivisions have been ceased. So, a subdivision plan should begin with status 0 and end with status 1.

Now, suppose subdivision X2 comes up with a plan X2 as in Fig. 2.

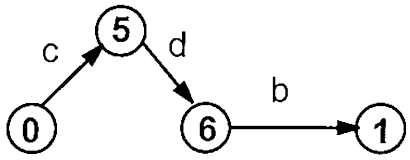

Figure 2: plan X2.

By merging the two plans into one plan (see Fig. 3), the manager in X soon finds some problems in the merged plan. For example, if site 3 and site 6 are merged, when event *b* occurs, the cost of transportation of personnel can be reduced because an airplane can be filled with maximum load and fewer flights are needed. So, in Fig. 3, site 3 and site 6 can be merged and site 2 and site 5 can be merged.

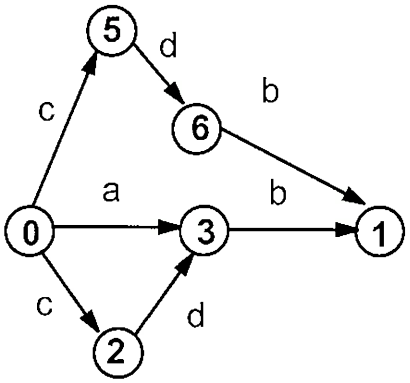

Figure 3: A merged plan from X1 and X2.

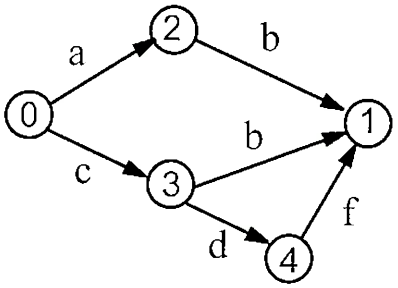

Figure 4: A plan that cannot be merged.

The manager also discovers a case like Figure 4, where site 2 and site 3 are considered to be merged. However, company X determines that it is better not to merge site 2 and site 3 because site 3 is planned to deal with more emergency events such as *d* and *f*. Merging 2 and 3 causes too much management cost. Your goal is to write a program for the manager to merge the sites of plans.

To give you some hints, in Fig. 3, site 3 and site 6 can be merged because after them, the remained emergency plans are the same. In Fig. 4, site 2 and site 3 cannot be merged because their remained plans are not the same.

Note that, you can assume that there are not sites shared by the two subdivision plans. In subdivision plans, cycles can be presented.

Input File Format

The first line of the input file contains an integer N, $1 \le N \le 10$, indicating the number of test cases. There is a single line containing a '/' character separating two consecutive test cases. The last line of the input file contains a '.' character denoting the end of the input file.

In each test case, it begins with an integer *np* which is the number of plans to be merged. In each plan, it begins with a number *e* where *e* is the number of edges and $1 \le e \le 100000$. Each edge is represented by (*s k t*) where *s* is the index of source site and *t* is the index of destination site. The index value is between 0 and 10000, where 0 and 1 are preserved for initial status and final status. *k* is the name of emergency event. It is a single character ranging from 'a'- 'z'.

Output Format

The number of sites for each test case (please ***exclude*** the status 0 and status 1).

Sample Input

```
2
2
4
0 a 3
0 c 2
2 d 3
3 b 1
3
0 c 5
5 d 6
6 b 1
/
3
2
0 c 2
2 b 1
2
0 a 3
3 b 1
3
0 a 4
0 c 4
4 b 1
.
```

Sample Output

```
2
1
```

Problem D
Transfer Optimizer
Input file: *pd.txt*

Problem Statement

Metro Taipei is planning the metro system schedule for the year 2030. At that time there will be N lines (N <=10), numbered from one to N, and M stations (M <=50) numbered from one to M. There will be no circular line so each line will have two directions.

By then the system will be fully automated and all trains will arrive and depart at precisely scheduled times.

The following rules have been decided in previous board meetings:

1. First trains for every line and direction must leave the starting station at 6am or 6:05am. Last trains must leave the starting station no later than 11:30pm.
2. For each line both directions must follow the same schedule.
3. Each train must stop at each station for exactly one minute.
4. Trains must arrive at the starting station two minutes before the scheduled departure; this leaves a two-minute window for passengers to transfer to this train.

The travel time between any two stations for each line is an integer in minutes. Facilitating line transfers is an important issue. To simplify the problem let us assume instantaneously transfer, which means if train A arrives at 10 and train B leaves at 10 then all passengers can still transfer from A to B or B to A. Given a description of the metro system (i.e., the sequences of stations of each line and the time needed to travel from one station to the next), you are going to write a program that finds the best starting time for each metro line so that the *daily transfer count* of the system is maximized. The *transfer count* of train A at stop S is the number of other trains (a train going in opposite direction does not count) that train A passengers can transfer to at stop S. As there would be no transfer to trains reached final station, no one would transfer to a train, which has arrived its destination, the count should exclude trains that have reached its final stop. The *daily transfer count* is the sum of all transfer count of all the trains at all stops from one day.

Input File Format

The first line of the input file contains an integer T, $1 \le T \le 10$, indicating the number of test cases. There is a single line containing a '/' character separating two consecutive test cases. The last line of the input file contains a '.' character denoting the end of the input file.

For each test case, the first line contains two space-separated integers N and M. The next N lines contain descriptions of each metro line arranged according to their metro line numbers, starting from one to N. There are at most fifteen stations for one metro line, and the format of descriptions is $f, s_1, t_1, s_2, t_2, \ldots, s_k$ where f $(2 \le f \le 20)$ is the number of minutes between two consecutive trains, s_1 is the starting station, s_k is the final station and t_i is the time needed to travel from s_i to s_{i+1}.

Output Format

The output contains answers for test cases ordered by their input ordering. Use '/' to separate two consecutive answers. For each answer, the first line contains an integer, which is the maximum achievable daily transfer count. The second line contains N space-separated bits, b_1, b_2, ..., b_n, where bi=1 means the i^{th} metro line start at 6 am, and b_i=0 means the i^{th} metro starts at 6:05.

Sample Input

```
2
2 5
10 1 10 2 10 3
10 4 10 2 10 5
/
2 5
10 1 10 2 10 3
10 4 6 2 6 5
.
```

Sample Output

```
840
0 0
/
840
1 0
```

Problem E
String Decomposition
Input file: *pe.txt*

Problem Statement

We give scores to those 26 lower-case English characters by letting score('a') = 1, score('b') = 2, ..., and score('z') = 26. For any string S composed of only lower-case characters, we define score(S) to be the overall scores of the characters divided by the length of S. For example, score("toon") = (20 + 15 + 15 + 14) / 4 = 16 and score("car") = (3 + 1 + 18) / 3 = 22/3.

We say that a string S is *fair* if score(R) $\leqq$ score(T) holds for any non-empty strings R and T with $S = R\ T$. For example, "car" is a fair string, because score("c") = 3 $\leqq$ 19/2 = score("ar") and score("ca") = 2 $\leqq$ 18 = score("r"). However, "toon" is not a fair string, because score("t") = 20 > 44/3 = score("oon").

Let $S_1, S_2, \ldots, S_m$ be a decomposition of a string S, i.e., $S = S_1\ S_2\ \ldots\ S_m$. We say that $S_1, S_2, \ldots, S_m$ is a *fair decomposition* of S if (a) each S_i with $1 \leqq i \leqq m$ is a fair string, and (b) score(S_1) > score(S_2) > ... > score(S_m). For example, "t", "oo", "n", "car" is a fair partition for "tooncar", because one can easily verify that all four of them are fair strings, and score("t") = 20 > score("oo") = 15 > score("n") = 14 > score("car") = 22/3. One can also verify that "cartoon" has exactly one fair partition, i.e., the word "cartoon" itself.

As a matter of fact, each string has *exactly one* fair decomposition. Given a string S consisting of lower-case characters, you are asked to compute *the* fair decomposition of S.

You may assume that (i) S contains only those 26 lower-case English characters, and (ii) the length of S is at most 200.

Input File Format

The first line of the input file contains an integer N, $1 \leq N \leq 10$, indicating the number of test cases. There is a single line containing a '/' character separating two consecutive test cases. The last line of the input file contains a '.' character denoting the end of the input file. For each of the test cases, the input string is given in a single line.

Output Format

For each of the test cases, print on one line the ending indices of each substring of a fair decomposition separated by space characters.

Sample Input

```
3
cartoon
/
tooncar
/
ccccccccccbbbbbbbbbbaaaaaaaaaa
```

Sample Output

```
7
1 3 4 7
10 20 30
```

Problem F
Interval Grouping
Input file: *pf.txt*

Problem Statement

Let $[a_1, b_1]$, $[a_2, b_2]$, ..., $[a_n, b_n]$ be n closed intervals where a_i and b_i denote the left and right endpoints of $[a_i, b_i]$, respectively. Assume all a_i's and b_i's are positive integers and distinct. Two intervals $[a_i, b_i]$ and $[a_j, b_j]$ are said to *overlap* if there exists a positive integer k with $a_i \le k \le b_i$ and $a_j \le k \le b_j$. A partitioning of the intervals into groups is said to be a *legal grouping* if the following three conditions are satisfied.

(1) Each group has at least one interval.

(2) Each interval is exactly in one group.

(3) For each group having two or more intervals, any two intervals in it do not overlap.

The problem asks to find the minimum number of groups among all legal groupings.

Example

Suppose 7 intervals, [1, 4], [12, 15], [7, 13], [3, 8], [5, 10], [2, 6] and [9, 14], are given. The 3 groups, <[1, 4], [12, 15], [5, 10]>, <[7, 13], [2, 6]> and <[3, 8], [9,14]>, comprise a legal grouping. Moreover the number of groups, 3, is minimum among all legal groupings.

Input File Format

The first line of the input file consists of a single number denoting the number of test cases in the file. There is a single line containing a '/' character separating two consecutive test cases. The end of the file is marked with a line containing a '.' character.

For each test case, the first line specifies n intervals, with $n \le 10000$. The next n lines give n pairs of positive integers, each denoting the left and right endpoints of an interval. (Note: the $2n$ integers are all distinct.)

Output Format

For each test case, print out the minimum number of groups on a single line

Sample Input

```
2
5
5 7
6 10
2 3
4 8
1 9
/
5
4 7
2 5
```

```
1 3
8 10
6 9
.
```

Sample Output

```
4
2
```

Problem G
Two-Goal Scheduler
Input file: *pg.txt*

Problem Statement

Scheduler in an operating system is used to schedule a set of tasks for some scheduling goals. There are periodic and aperiodic tasks. A periodic task is executed exactly once in every constant interval, period. For simplicity, we assume a periodic task is ready at the beginning of a period and the deadline is at the end of each period where a ready task is a task ready to run as long as it gets the right to use CPU. The periodic task will be ready again at the beginning of the next period. Tasks running not in the periodic manner are aperiodic tasks, that is, the aperiodic tasks arrive randomly and only run once. For simplicity, we assume an aperiodic task is ready when it arrives. However, a ready task might not be always running the CPU because there might be some other ready task with higher priority. Priority-driven scheduling approach is commonly used in modern computer operating systems where the systems always execute the task with the highest priority. A set of tasks are said to be feasible if every task finishes execution before its deadline. In priority-driven scheduling, a preemptive approach states that a task with lower priority may be preempted by a ready task with a higher priority, yielding CPU to the higher-priority task, and resuming later.

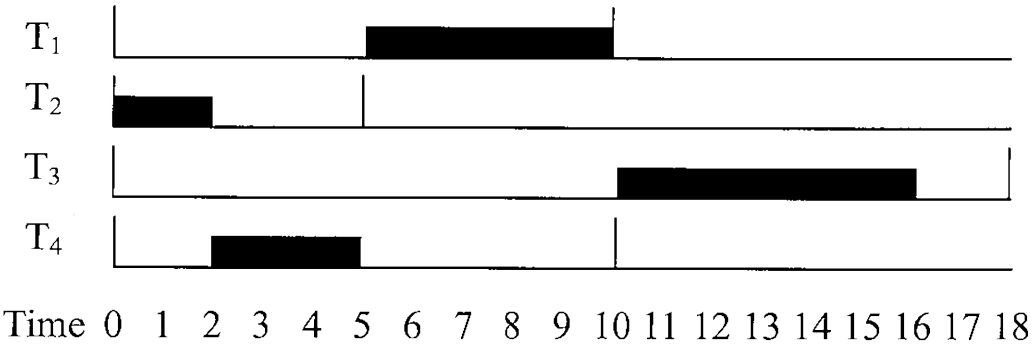

Figure 1 Sample Task Schedule

Waiting time is defined to be the sum of the time task spent waiting in the ready queue, i.e. the time a ready task can not run. Figure 1 shows that tasks T1, T2, T3 , and T4 which all arrive at time 0 with execution times 5, 2, 6, and 3 and deadlines 10, 5, 18, and 10 respectively are feasible using shortest-job-first scheduling. T_1, T_2, T_3, and T_4 finish execution at time 10, 2, 16, 5 with waiting time 5, 0, 10, 2 respectively. Note that using the shortest-job-first scheduling might not always find a feasible schedule with the shortest total waiting time.

The tasks all arrive at time 0 and are preemptive. The first scheduling goal is to be feasible and the second goal is to have the shortest total waiting time. Both goals need to be achieved.

Input File Format

The first line of the input file contains an integer N, $1 \leq N \leq 10$, indicating the number of test cases. There is a single line containing a '/' character separating two

consecutive test cases. The last line of the input file contains a '.' character denoting the end of the input file.

Each test case is listed by a line of the number of tasks, ≤ 100, and lines of task execution time and deadline. All the execution times and deadlines are positive integers, ≤ 500000.

Output Format

For each task set, print out a line of the shortest total waiting time, if feasible, and -1, if not feasible.

Sample Input

```
3
4
5 10
2 5
6 18
3 10
/
3
1 6
2 2
3 6
/
3
1 2
2 2
4 7
.
```

Sample Output

```
17
5
-1
```

Problem H
Smallest Partition
Input file: *ph.txt*

Problem Statement

Let a and b be two relatively prime positive integers. We are interested in the partition P={p[1],....,p[N]} of (a + b) that satisfies the following properties:

p[1]+...+p[N] = a + b, where each p[i] is a positive integer, and

If S is any subset of P, either

(1) for some subset T of S with the sum equal to a, or

(2) for some subset U of P-S (the difference set of P and S) with the sum equal to b.

There is a trivial partition for any a and b, i.e., (a + b) 1's, which satisfies the above properties and can be checked easily. However, here we want to find the smallest partition, which contains the minimum number of elements. For example, {2, 2, 2, 2, 1, 1, 1} is a minimum partition, for 2 and 9, whose size is 7. Given two relatively prime positive integer a and b (smaller than 2^32), you are asked to write a program to find the size of the smallest partition that satisfies the above properties.

Input File Format

The first line of the input file contains an integer N, $1 \le N \le 10$, indicating the number of test cases. There is a single line containing a '/' character separating two consecutive test cases. The last line of the input file contains a '.' character denoting the end of the input file.

For each of the test cases, there is one line of input, that consists of the values of a and b, which are both less than 2^32.

Output Format

For each test case, the output contains a line with the size of the minimum partition as described above.

Sample Input

```
3
1 2
/
1 9
/
2 9
.
```

Sample Output

```
3
10
7
```