Problem A:

Link:
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=3778
Type: Implementation
Hints:
You can keep two array: L[] & R[], where array L[i] will represent i-th soldier's left neighbour & R[i] will represent i-th soldier's right neighbour. So, when you are given that A to B soldiers are killed, then you will just have to print from these two array: L[A] & R[B] and also check for the print of *. After that, the update will be: L[R[B]] = L[A] & R[L[A]] = R[B], as A to B are killed, then left of A will be neighbour of right of B and vice versa.

Problem B:

Link:
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=3517
Type: Bitmask DP + Aho Corasick
Hints:
You can keep an aho corasick for the given substrings. Then do a bitmask dp. Here you have to try to build strings by adding new letters at the end and check how many substrings have been covered by this created string. The bitmask should contain the information of the substrings which are already contained by the newly created string. You also have to keep the node of aho corasick on which you are currently on after adding a new letter. So, dp functions parameter will be: string length already made, mask for substrings & node number of aho corasick.

Problem C:

Link:
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=2229
Type: Binary Search + BPM
Hints:
Lets divide the poles into two sets: Red and Blue. Now, do a binary search in between 0 to max distance of the poles. Now for each mid value of the binary search, create a graph: if i-th red and j-th blue poles have distance less than mid, then add an edge between them and run BPM on the graph. Then you will find maximum match of X. Now compare this X with the given K to move the low and high of the binary search and get the result.

Problem D:

Link:
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=5043
Type: Number Theory
Hints:
You will just have to calculate the euler's phi for each number for each input. If you don't know the formula of euler's phi, then here it is: $\phi(n) = n * (1 - 1/p_1) * (1 - 1/p_2) * ...$ where p_1, p_2, ... are the prime factors of the number.

Problem E:

Link: http://codeforces.com/problemset/problem/653/D
Type: BS + Flow
Hints:
This problem can be easily solved using the combination of binary search and maximum flow. So, you will just need to binary search on the maximum amount of weight. Taking the mid value as the weight you can build the graph using the given edges as the capacity of C_i/mid. Then you will find how many bears you will need and then find the result by shifting the values of low and high.

Problem F:

Link:
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=4629
Type: Number Theory + DS
Hints:
This problem's solution is offline processing DS. You can find every number's divisor and store them in vectors using sieve. Then for every number, find L[i] & R[i], which means within this range - L[i] to R[i], there is no number which can divide i-th number. Now, we store the queries and sort them according to the ending of the query range. Lets now try to find out the answers. We will try to find the result using a DS, for simplicity lets use BIT. Loop through the array of numbers from 1 to N. On the i-th number, put -1 on the index of L[i]-1 and +1 on the index i of the BIT. Also put -1 on those indices whose R occurs on this index- i, that means, you will put -1 on index j if R[j]+1 == i. Now, for queries those ends at index i, just query on the BIT and find out those answers.

Problem G:

Link:
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=3767
Type: Bruteforce
Hints:
May be most of the people didn't looked into the time limit of the problem. This was the actual intention of givin this problem. The TL was 10sec. Usually 10^9 complexity runs in 3 sec in UVA, and if you just do bruteforce in this problem, the complexity will be: 2.5*10^9

(Only one test case in this problem! ;)) :D Confusion!! Those who didn't solve this problem, you didn't look into each constraint in the problem then :P

Problem H:

Link: http://codeforces.com/problemset/problem/696/E
Type: DS - HLD + Segment Tree
Hints:
The main tricks here in this problem is: each girl will be selected at most 1 times and after that she will not be considered for next queries. So, in each query, you can simple loop through the girls for picking up. The idea is: for finding the result of a query, you just find the best girl from the existing ones which will occur on the way of u to v. Then remove her from the existing girls set. Then again, find the next best girl and remove her, and so on. That means, in a query, you can just loop over the given value K to find the girls one by one. For doing so, you will just now need to have a HLD and a segment tree. I think just after revealing this tricks, the problem becomes easier for you, now just you need to implement the solution.

Problem I:

Link:
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=4948
Type: DP
Hints:
Initially precalculate the number of ways you can use to go from (0,0) to (1000, 1000). Now in each test case, you can easily find the result for the queries. Now, can you find out how many ways are there which will pass through the cell (i, j) for going from (0,0) to (R,C)? This is: [number of ways from (0,0) to (i, j)] * [number of ways from (i, j) to (R, C)]. Using the precalculation you should be able to answer this thing in O(1). Now, for answering the queries: you just find answer: number of ways from (0, 0) to (R, C) - (summation of number of ways from (0, 0) to (R, C) that goes through (i, j)). This summation for the rectangular answers can also be precalculated using DP. So, overall the complexity become 1000*1000*2 precalculations and O(1) for answering the queries.

Problem J:

Link: http://codeforces.com/problemset/problem/406/C
Type: Dfs + Implementation
Hints:
This problem only has solution when the number of edges are even, otherwise partition isn't possible. For finding the solution we can use dfs and use non-used edges. So, lets try to make pairs from the tail of dfs calls. Go the the deepest nodes and find the pair of edges to make them together. After that, on a node, if there still remains some edges, collect them together, and join them in pairs for making new pairs. A deep visualization and implementation in this problem can help you figure out the answers.

Problem K:

Link: http://codeforces.com/problemset/problem/95/D
Type: Digit DP
Hints:
This is a very traditional digit dp problem. It should have been solved by more that 5 teams. You just need to keep track of the previously used lucky digit's position and then continue building numbers keeping the constraints of K.

Problem L:

Link:
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=3890
Type: Adhoc / Number Theory
Hints:
This the hardest problem for me to solve. :(Still I have no idea how should I solve this problem :'(
Jokes apart, this problem can be solved just calculating the fibonacci series.