Name: PEIGUO GUAN
zID: z5143964

Solution:
1.
Assume the dam has head as H, and Tail as T:
let $x_h = x_i, x_t = x_i$
Set D as the number of dam

So, there are two directions, downstream and up stream. If the point with the distance larger than r_i from point x, then we can set a dam there.

Firstly, we got a point x_i , $x_h = x_i,\ x_t = x_i$
With the recursion, we can refresh the point if satisfied the condition:

Upstream: $\mathrm{P}(x_h)_1 = 1 + P(x_h = x_h - r_i)\,,(x_h - H > r_i)$

Downstream:$\mathrm{P}(x_t)_2 = 1 + P(x_t = x_t + r_i\,), (T - x_t > r_i)$

Finally, we can sum up P_1 and P_2 to get the maximum number of dams.

2.
(a). In isolation and ignoring the pebbles in adjacent columns, the patterns of the column which does not contain the vertically adjacent pebbles look for the column also does not contain the vertically adjacent and do not contain pebbles in the rows already containing a pebble in column previous column.

There are 4 squares in a column, so the number of the legal patterns that can occur in a column is that:
If do not put any pebble in a column, it could be compatible to any column: 1
If put 1 pebble in a column, there will be 4 different ways.
If put 2 pebbles in a column, there will be 3 different ways.
And if over 2 pebbles in a column, it cannot be a legal pattern.
So in total, there is 1+4+3 = 8 of legal patterns.

Two patterns compatible:
Since the first k columns 1<=k <=n, so let say that there are *k* columns and the last column k^{th} column is c, so considering the sub-problems, so the pattern occurring in the last column is:
Assuming all k^{th-1} columns are legal and compatible, so the last k^{th} column should be legal and compatible with previous column so:

Legal(k^{th-1}), *Legal*(k^{th})

Legal(k^{th}) = Compatible(*Legal*(k^{th-1}), *Legal*(k^{th}))

The total pattern is

$$\text{Pattern[i]} = \begin{cases} legal(k^i)\,, i = 1; \\ Pattern[i-1] + Compatible\left(legal(k^{i-1}), legal(k^i)\right), 1 < i \leq n \end{cases}$$

(b).

The bond edge is when the i is 1, then just choose the max legal pattern, and then when i is larger than 1, it needs to always choose the max in previous pattern and add the max compatible pattern between i-1 and I column.

$$\text{Opt[i]} = \begin{cases} \max\,(pattern[i])\,, i = 1; \\ \max\,(Pattern[i-1] + \max\,(Compatible\left(legal(k^{i-1}), legal(k^i)\right)), 1 < i \leq n \end{cases}$$

Every step of Opt[i] is the optimal solution, so it cost O(n) time for computing the optimal replacement.

(What I think is simply sum up the previous optimal solution with the next column which is the max value with the previous column. But I think this question is more complicated, since though the previous optimal solution is the max, it doesn't mean adding the max column value which is compatible to the previous column is optimal as well. So what I do is to max the previous possible pattern and choose the largest one.)

(3):

Firstly, order skiers' height as S_i and order the length of all skis by increasing order as S_j. We can assume that if the task is optimal, then if skis l_j assigned to S_i and the $|h_i - l_{j(i)}|$ is shorter than l_j assigned to S_j, which is $|h_j - l_{j(j)}|$ then we assigned l_j to S_i, otherwise we can compare the sum of differences between the heights of skiers and length of skis.

Find the first optimal assignment of skiers i from skis j:
If j = i, because the number of them is equal then assigned skiers according to their height,
If j > i, the subproblem is that assume I people already get optimal solution with i skies, but when the j skies appear, it needs to reassigned the skies, so here will be two situation:

1. The I people keep on previous solution without using the new skies
2. assigned skiers i with skis j.

The bond situation is opt(i=1,j).

Let $D[i,j]$= $|h_i - l_{i(j)}|$

$$1 \leq \text{i} \leq \text{n}, \text{i} \leq \text{j} \leq \text{m}$$

$$\text{Opt(i, j)} = \begin{cases} Opt(i,j), j = i\,; \\ \min(Opt(i,j-1), Opt(i-1,j-1) + D[i,j])\,, j > i \end{cases}$$

(4):

(a):

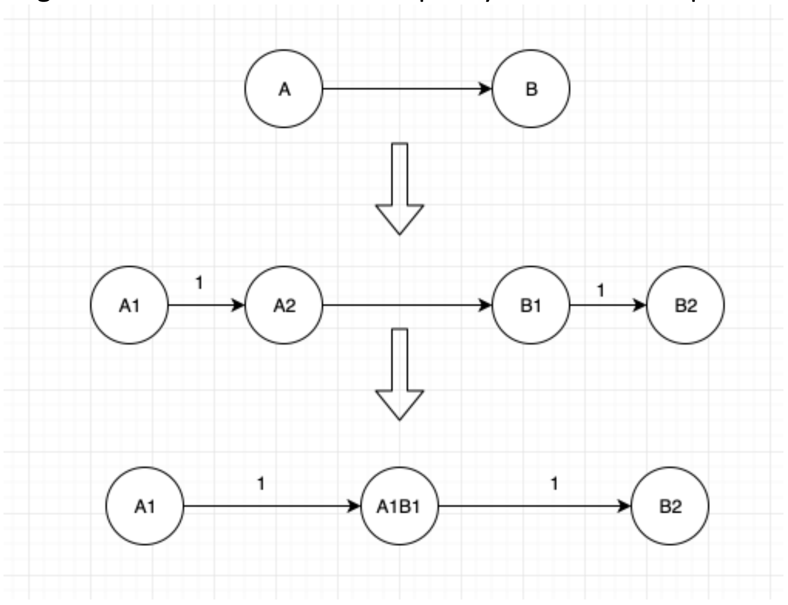
Assume the spies are vertices and the channels as the edges of the graph, so the task of question is to find the max flow and find the min number of edges that need to cut.

We can add two edges from S->s_i and the edges s_j->T, since there are n spies, so we can assign S->s_i and s_j->T with n capacity and assign 1 capacity of edges among all the edge from s_i to s_j. Among the two vertices s_i and s_j, we run Max-Flow – Min- Cut Algorithm on the network, and finding the minimum S-T cut. So that the min cut is equal to the number of edges crossing the cut. We can place the listing device on the edges to stop the communication.

(b):

In this problem, we can use the same trick as the question(a), but there is a little different. In question(a), we need to find the minimum cut of edges, but in this problem we need to find the minimum vertices that pass message from S to T. So the main problem is that how can we change the vertices to edges, so as to use the Max-Flow and Min-Cut Algorithm and find the minimum edges(which is the vertices actually) to bribe, so as to stop the communication?

There is a way which can transfer the vertices to edges and edges to vertices without changing the structure of the graph. We can separate a vertice to two vertices and build a edge between them with the capacity of 1. For example:



By this way, we can change the vertices to edge instead of changing the original meaning of graph. Then we can use the Max-Flow and Min-Cut algorithm to find the minimum number of edges which is the vertices of spies to bribe.

Q5:

Modifying the graph by adding two edges from $s \rightarrow u$ and $v \rightarrow t$, and set the capacity of them as infinite. Then we use Max-Flow and Min-Cut Algorithm on the network, finding a minimum $s - t$ cut.