Assignment 3

z5142340 Haiyu LYU

1. We build a hash table called $prev()$ that can save previous dam. For example. x_i is a dam place and x_j is a dam just in front of x_i. $x_j$need to meet the conditions: $x_j + r_j < x_i$ and $x_i - r_i > x_j$. If x_j can not satisfy the conditions. We check if x_{j-1} meet the above restrictions. Repeat this algorithm until you find a dam that meets the restrictions or no such dam. There are two result in $prev(x_i)$. The first one is $prev(x_i) = j$ (j is the number of dams in front of x_i and meeting distance conditions). The second result is that there is no x_j meet x_i, so $prev(x_i) = 0$. For each dam x_i, we use this algorithm to get hash table $prev(x_i)$.
   We set optimal subset as table $opt[]$. We set $opt[0] = 0$ and the first dam $opt[1] = 1$. The pseudocode of the algorithm is as follows:

```
BEGIN
for i ← 2 to n do:
    if prev(xi) == 0:
        Return 1
    else:
        A = 1 + opt[prev(xi)]
        B = opt[i − 1]
        opt(i) = max(A, B)
    end if
end for
return opt[N]
END
```

2.
(a). Case1: There is no pebble in column. (1 pattern)
Case2: The column only contains 1 pebble (1000, 0100,0010,0001). (4 patterns)
Case3: The column may contain 2 pebbles (1010,1001,0101). (3 patterns)
Hence, there are 8 legal patterns that can occur in any column.

(b). Because there are 8 ways in total. We assume they are $x_1, x_2, \cdots, x_8$ and those value is $v_1, v_2, \cdots, v_8$.
Suppose we have two ways, x_i and x_j, x_i are to the left of x_j and both of them are legal.
Suppose we have a table C, which is our best placement solution. We assume that x_i is the optimal placement for $C[i, n]$ and then we should put the value of x_i in column $C[i, n]$. The best $C[j, n + 1]$ is the max x_j(x_j is legal for x_i). We have:

$$C[i, n] = \max(C[j, n + 1]) + v_i$$

Calculating each x_i and v_i take $O(n)$, and the backtracking take $O(1)$. Therefore, this algorithm runs in $O(n)$ overall.

3. Firstly, we should use merge sort to sort heights $h_1, h_2, \cdots, h_n$ and sort $l_1, l_2, \cdots, l_m$. Then we should create a 2-D table called subset which size is $(n+1)*(m+1)$. It is necessary to initialize this table before running the algorithm. we put 0 into first row and first column, like this:

lengths \ heights	0	1	……	m
0	0	0	0	0
1	0			
……	0			
n	0			

The pseudocode of the algorithm is as follows:

$BEGIN$
$for\ i \leftarrow 1\ to\ n\ do$:
    $for\ j \leftarrow 1\ to\ m\ do$:
        $if\ j < i$:
            $subset[i, j] = \infty$
        $else$:
            $if\ j = i$:
                $subset[i, j] = subset[i-1, j-1] + |h_i - l_j|$
            $else$:
                $A = subset[i, j-1]$
                $B = subset[i-1, j-1] + |h_i - l_j|$
                $subset[i, j] = \min(A, B)$
            $end\ if$
        $end\ if$
    $end\ for$
$end\ for$
$return\ subset[n, m]$
END

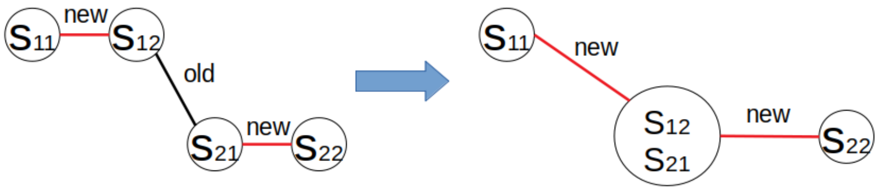
4.

(a). We can see each spy as a vertex and each channel as edge in network. The original problem that find the fewest number of channels becomes that finding the min-cut. In order to find it, we just need to use Edmonds-Karp algorithm to calculate max-flow. We assume the capacity of each edge as 1. After calculate the max-flow we can confirm min-cut. Because the edges in network flow is the channel. We need to compromise the edges which are crossed by min-cut. Finally, we get the fewest number of channels that satisfies the conditions.

(b). At first, we should check if there is an edge between. If there is such an edge, then we will have no solution.

We assume that there are N vertex (except S and T), and M edge in network flow. We separate one vertex to two vertexes. For instance, vertex S_1 is separated to S_{11} and S_{12}. We add a new edge between two new vertexes and the capacity of this edge is 1.

Now we have N new edges and M old edges. Old edges connect original vertexes, like s1-s2. So we need to delete old edges by merging 2 vertexes. The method is shown below.



After merging vertexes, we only consider the new edge in network flow. We can also use E-K algorithm to get the max-flow and min-cut. Because those edges represent vertexes. Therefore, we can bribe those vertexes(spies).

5. Firstly, we add two edges $s \rightarrow v$ and $u \rightarrow t$. Then we can use Fold-Fulkerson method for finding maximal flow and minimal cut one the new network. Using this method to achieve that vertex u is at the same side of the cut as the source s and vertex v is at the same side as sink t.