QUESTION 1

Because of the recent droughts, N proposals have been made to dam the Murray river. The i^{th} proposal asks to place a dam x_i meters from the head of the river (i.e., from the source of the river) and requires that there is not another dam within r_i metres (upstream or downstream). What is the largest number of dams that can be built? You may assume that $x_i < x_i + 1$.

- To start this question, we need to make sure that the starting point of the dam from each proposal x_i are in non-decreasing order, so we get $x_1 \leq x_2 \leq \cdots \leq x_i$, when $i \leq n$, however because of there are "$x_i < x_i + 1$" in the question content so we assume that it is placing by non-decreasing/increasing order.
- Think: no matter how we place the dams, the last step of placing the dam would be place a dam in position x_i, and x_i should be greater or equals to r_{i-1}.

- For every $i \leq n$, we solve the following sub-problems:

  P(i) : find a subsequence A_i of the sequence of the proposals $P_i =< P_1 + P_2 + P_3 + \cdots + P_i >$:

  1) A_i consisting of non-overlapping proposal, which means $r_{i-1} \leq x_i$;
  2) A_i ends with proposal P_i, so (i) means the current one in recursion;
  3) A_i is the largest number of dam (optimal solution) that can be built among all subsequence of P_i which satisfies step 1 and 2

- Let opt(i) be the largest (optimal solution) number of dam that we can built when we are up to P_i.
- When its P_1, which means when $(i = 1)$, there is only one proposal, so opt(i) = 1.
- Recursion: assuming that we solved the subproblems for all $j < i$ and they are placed in order in the table:

  $$\text{opt}(i) = \max\{opt(j) + 1 : (j < i)\ and\ (r_j \leq x_i)\}$$

  in the table, besides opt(i) we also store j for which the above max is achieved.
- The time complexity would be $O(n^2)$

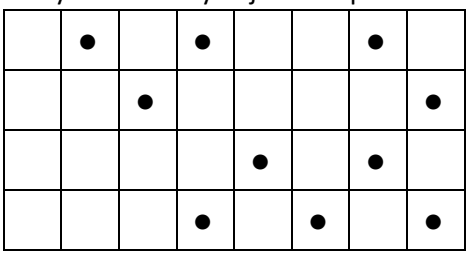
QUESTION 2 -> NEXT PAGE

QUESTION 2

We are given a checkerboard which has 4 rows and n columns, and has an integer written in each square. We are also given a set of 2n pebbles, and we want to place some or all of these on the checkerboard (each pebble can be placed on exactly one square) so as to maximize the sum of the integers in the squares that are covered by pebbles. There is one constraint: for a placement of pebbles to be legal, no two of them can be on horizontally or vertically adjacent squares (diagonal adjacency is fine).

(a) Determine the number of legal patterns that can occur in any column (in isolation, ignoring the pebbles in adjacent columns) and describe these patterns.

Legal = no two pebbles be on horizontally or vertically adjacent squares



There are total 8 patterns:

1) Empty column	5) One pebble in fourth row
2) One pebble in first row	6) One pebble in first row + one pebble in third row
3) One pebble in second row	7) One pebble in first row + one pebble in fourth row
4) One pebble in third row	8) One pebble in second row + one pebble in fourth row

(b) Using the notions of compatibility and type, give an O(n)-time algorithm for computing an optimal placement.

- For every pattern, there is a constant number of patterns that are compatible with it, such as, every pattern is compatible with empty column, one pebble in the first row is compatible with every pattern except (pebbles in first + third row) and (pebbles in first row + fourth row), etc.
- About the number of pebbles, 2*n, because of the maximum number of pebble we can place in one column is 2 to make it legal so for total n columns we can only place maximum 2*n pebbles in the checkerboard
- Use a 2D array opt[i][j]:
    - [i] means the 8 patterns that we get from the previous question;
    - [j] means the maximum value (sum of single/multiple square/integers has been covered) that we can cover for each column, which means j <= n, as there is total n columns.
- Therefore, for every column j, we choose the row i with the maximum value, and check if it is compatible with the previous column:

$$\text{opt[i][j]} = \max\{\text{opt[m][j} - 1] + \text{opt}[n][j] : \text{m, n} \in \text{i \& m and n are compatible cases \& j} \leq \text{n}\}$$

base case: when the column equals to 0, the optimal answer would be zero.

- The time complexity would be $O(8 * n) = O(n)$.

QUESTION 3

Skiers go fastest with skis whose length is about their height. Your team consists of n members, with heights $h_1, h_2, \ldots, h_n$. Your team gets a delivery of m ≥ n pairs of skis, with lengths $l_1, l_2, \ldots, l_m$. Your goal is to write an algorithm to assign to each skier one pair of skis to minimize the sum of the absolute differences between the height h_i of the skier and the length of the corresponding ski he got, i.e., to minimize $\sum_{1 \le i \le n} |h_i - l_j(i)|$ where $l_j(i)$ is the length of the ski assigned to the skier of height h_i.

- Think: because we need to assign to each skier one pair of skis to minimize the sum of the absolute difference between the height h_i of the skier and the length of the ski he got, and we now have total m ≥ n pairs of skis, thus we have two possibilities to think about, the first one is m = n, the second one is m > n. when m = n, we sort both heights of skiers and lengths of skis by non-decreasing order, and assign the skis to the skiers diagonally. For m > n, after we sort the heights and the lengths by non-decreasing order, we also need to consider if we should assign the current ski to the skier or not, just like the no duplicate knapsack problem. Also if we choose not to assign the ski to the current skier, which means we skip current one ski, how many skis we can skip, and we can still assign skis to every skier?

- The last step that we can do is to assign the j^{th} ski to the i^{th} skier, and we need to make sure we get the minimum, $\sum_{1 \le i \le n} |h_i - l_j(i)|$, for this equation we can understand it by another way, which is, $\left|\sum h_i - \sum l_j(i)\right|$, which is the minimum difference between the sum of the total height that we currently have and the sum of the total length of assigned skis that we currently have.
- For the case m > n, we can now get a table with size m*n, use the variable i ≤ n and j ≤ m. for every i, which we can say for every row or for every skier (that sorted by heights in non-decreasing order), we need to check which j, or which ski length, can minimize the function, $\sum_{1 \le i \le n} |h_i - l_j(i)|$, with current skier. Thus, every time we find a better choice, we store the result in table, once, no more better options, we remain the best option.

$$\min\{\mathrm{M[i][j-1]}, \mathrm{M[i-1][j-1]} + |h_i - l_j|\}$$

and for this one we need to make sure 1 ≤ i ≤ j, and do this statement for every M[i][1..j] till we reach j, and do another row M[i+1][1..j]. However, the case we still need to think about is what if i > j, because of we can only match the ski and skiers diagonally or skip one/some skis and assign skis diagonally, so when i > j we get infinity. And while there is no skier, we don't need to assign any skis so while i = 0, M[i][j] would equals to zero. Therefore we can get the following three cases:

M[i][j]=
- if i = 0, then M[i][j] = 0
- if i > j, then M[i][j] = ∞
- while $1 \le i \le j$, then M[i][j] =$\min\{M[i][j-1], M[i-1][j-1] + |h_i - l_j|\}$

- After we fill the table we can consider about the skis assign problem, because of the three cases that we listed above, we can say that, in one row, once find the current best/optimal option, all the column after are remaining the same number. Thus we finding from the last column x last row, which is now i x j, or we can say its m x n. from the last row, last column, we check if current [j] is equals to [j-1], if [j] = [j-1] then we move to left, till we find the [j] != [j-1] we assign the [j] to the current [i] and move to the [i+1] row.

- The time complexity would be nlogn and mlogm for sorting the two array/sequence, and check every row, total n rows, maximum m times, so m*n. however when we checking back we would only check maximum (m-n)*n times.

QUESTION 4

You know that n+ 2 spies S, s1, s2, . . . , sn and T are communicating through certain number of communication channels; in fact, for each i and each j you know if there is a channel through which spy si can send a secret message to spy sj or if there is no such a channel (i.e., you know what the graph with spies as vertices and communication channels as edges looks like).

(a) Your task is to design an algorithm which finds the fewest number of channels which you need to compromise (for example, by placing a listening device on that channel) so that spy S cannot send a message to spy T through a sequence of intermediary spies without the message being passed through at least one compromised channel.

- As the task is to find the fewest number of channels which we need to compromise, which means, the number of channels that we are going to compromise should be as few as possible and also need to make sure spy S can no longer send message to spy T.
- To solve this question, we can view the spies as vertices in a flow network and the spy S as the source node, spy T as the sink node. We then can use the maxflow-mincut to solve the question, whenever the flow network reach the max flow, which means it can no longer find any augmenting path it stops, however, the problem now becomes what should be the capacity.
- As we need to make sure the number of channels that we are going to compromise should be as small as possible, thus we can view it as we compromise the edge/channels that can help S and T connect each other at most once. So we set the capacity of every edge to 1, once the max flow algorithm terminated we can find the min cut, which are the edges that we need to compromise.

(b) Assume now that you cannot compromise channels because they are encrypted, so the only thing you can do is bribe some of the spies. Design an algorithm which finds the smallest number of spies which you need to bribe so that S cannot send a message to T without the message going through at least one of the bribed spies as an intermediary.

- Similar to the previous question we set the spy S as source and spy T as sink as we need to make sure S cannot send message to T without message going through at least one of the bribed spies as intermediary.
- As we cannot compromise channels anymore, so we have to ignore the channels which is the edge in the 'flow network', what we can do is to set the capacity of the channels/ or edges to infinity. And what we can do now is to bribe spies, thus we make the capacity of the vertex/spy becomes 1, then there will be some new vertexes, because if one vertex V, that has capacity it will become two vertexes connected by the edge e(Vin, Vout) with weight (the weight of the edge is depends on the capacity), then we do the max flow algorithm, after it terminates, based on the maxflow-minmax theorem, we can do a min cut. Then we can find the minimum number of spies that we need to bribe in this question.
- If S can directly connect to T then no needed to bribe any other spies, there will be no solution.

QUESTION 5

You are given a flow network G with n > 4 vertices. Besides the source s and the sink t, you are also given two other special vertices u and v belonging to G. Describe an algorithm which finds a cut of the smallest possible capacity among all cuts in which 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.

- As we need to find the cut with smallest possible capacity among all the cutes in which vertex u is at the same size of the cut as source s, and v is at the same side as sink t.
- If we want to avoid to find the cut between s to u, and v to t. As s is the source node, so the only possibility is from source s to u, s->u, and from v to sink t, v->t.
- Therefore, we can add two edges, s->u, and v->t with both infinity capacity, then run the max flow algorithm, after it terminated, we do a minimum cut and find the solution.