COMP3121/9101 Assignment 3
z5014567 Senlin Deng

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 build? You may assume that $x_i < x_{i+1}$.

   This problem can be solved by using **Greedy Method**.

   Since no other dams can be located near the i^{th} dam within r_i metres bidirectionally, the "safe" range for a certain dam i will be $[x_i - r_i, x_i + r_i]$. And we further denote $s_i = x_i - r_i$ and $e_i = x_i + r_i$ as the starting and ending spot, respectively, for certain i^{th} dam. And the interval can be rewritten as $[\boldsymbol{s_i}, \boldsymbol{e_i}]$**.**

   For each dam, its interval cannot have intersection with others'.

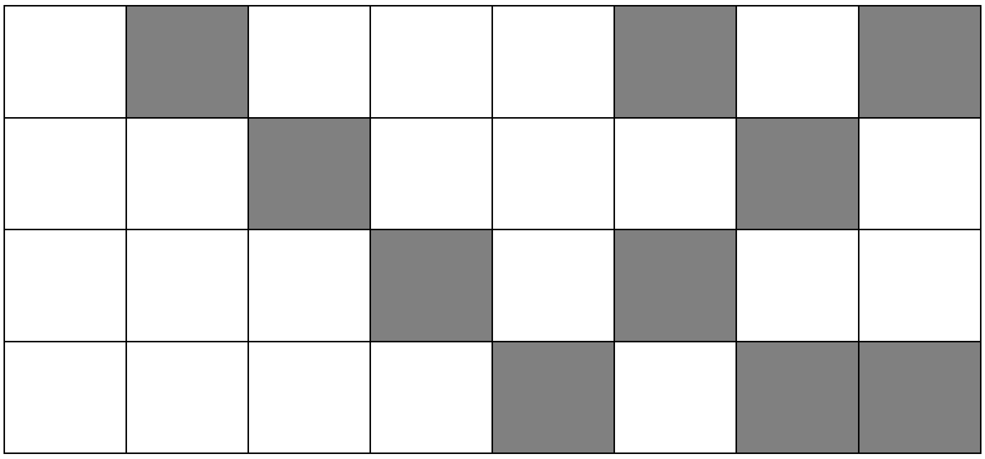
   The correct solution: **Among the dams which do not conflict with previously chosen dam always chose the one with the smallest end spot** (i.e. $e_i = x_i + r_i$).

   Proving: Assume there is an **alternatively optimal solution** violating the greedy choice and the first dam which violates the greedy choice has an interval $[s_i, e_i]$. And the last chosen dam is $[s_{i-1}, e_{i-1}]$. Since $[s_i, e_i]$ is not our candidate by using greedy method, we can find another dam $[s_j, e_j]$ and $e_j < e_i$. And we can choose the dam with the interval $[s_j, e_j]$ **with the same number of dams** which produces no conflicting. Continues in this manner until all dams have been traced. And we can prove that greedy method will give a solution which is optimal.

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 the 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.

   There are **8** legal patterns which are illustrated in the following. The grey tiles indicate the squares which are filled with pebble, while the blank ones are empty. (The enumeration is isolated, and the columns are not actually located next to each other.)



Call two patterns *compatible* if they can be placed on adjacent columns to form a legal placement. Let us consider sub-problems consisting of the first k columns $1 \leq k \leq n$. Each sub-problem can be assigned a type, which is the pattern occurring in the last column.

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

This problem can be solved by using **Dynamic Programming**.

For every $k \leq n$ we solve the following subproblems $P(k)$: find the **legal** combination of patterns which maximize the integers in the squares that are covered by pebbles **from the first column to $\boldsymbol{k^{th}}$ column.**

Firstly, we precompute the sum of each column upon all eight patterns and store the results in an $8 \times n$ two-dimensional array. And we denote $\boldsymbol{s(i,j)}$ as the element of the array (i.e. $s(3,6)$ is the sum of the 6th column with the third pattern). This step takes $O(n)$.

Due to the constraint, there are certain constant number of patterns that are compatible with it. For each pattern p, we denote p'as the compatible patterns of p. Note that one pattern can have multiple compatible patterns, so p' is not a single pattern in this case but a notation of all legal combinations of a certain pattern p.

Denote $\boldsymbol{opt(k-1,p)}$ as the optimal solution achieved by pebbling column $1 \ldots k-1$ with a certain pattern p. And the basic cases are $opt(0,p) = 0$. So that the optimal solution for column $1 \ldots k$ is $\boldsymbol{opt(k,p) = max\big(s(p,k)\ +\ opt(k-1,p')\big)}$. Since the k^{th} column might dominate the first $(k-1)^{th}$ columns, so that we need to subject to the pattern on $(k-1)^{th}$ column so that it is compatible with k^{th} column. And we can solve this problem recursively to get our final answer. To reconstruct the actual problem, we only need to substitute k with n and to get the $\boldsymbol{opt(n,p)}$. This can be simply proved by "cut-and-paste". And the time complexity for each process is constant, since we are only tracing the numbers which are stored in an array. So that time complexity for this algorithm is $O(n)$.

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

   This problem will be solved in two conditional cases based on the relation between m and n.

   Case 1: when $m = n$,

   We denote H and L as $\{h_1, h_2, \dots\dots, h_n\}$ and $\{l_1, l_2, \dots\dots, l_m\}$ respectively. Since the number of skiers is equal to the number of the skis, each person will be assigned with a ski with no remaining. Thus, this problem can be solved simply **with greedy method**. Firstly, we use **merge-sort** to arrange the elements in H and L with **non-decreasing order**. And we **match the skier and the ski with same index correlatively**. This will produce the minimal value of $\sum_{1 \leq i \leq n} |h_i - l_{j(i)}|$. To prove this is optimal, we assume there is another optimal solution which violates our greedy method. And we further assume (h_i, l_i) is the first pair which violates our greedy choice. Since our greedy method pick the shortest ski in the remaining set for a certain h_i, we can find another unpaired ski with a length l_i' and l_i' is definitely less than l_i and minimize the difference for certain h_i. So that, our assumption is not optimal compared to the greedy choice. Thus, this produces a contradiction with our assumption. So that greedy choice is the optimal solution.

   Case 2: when $m > n$,

   This problem can be solved with dynamic programming in a **two-dimensional** recursion, when m is generally greater than n. And we sort both H and L with non-decreasing order.

   For every $n' \leq n$ and $m' \leq m$ we solve the following subproblems $P(n', m')$: find the minimal sum of the absolute difference between the height of the skier and the length of the corresponding ski with the new set $\{h_1, h_2, \dots\dots, h_{n'}\}$ and $\{l_1, l_2, \dots\dots, l_{m'}\}$.

   Assume we have solved all the subproblems for $n'' < n'$ and $m'' < m'$. And we denote $opt(n', m')$ as the optimal cost of matching first n' skiers with first m' skis. Now we need to determine $opt(n', m')$ with solved optimal values. Note that the result is nontrivial when $m' < n'$, since we cannot have unpaired skiers. Thus, when $m' < n'$, we set $\boldsymbol{opt(n', m')} = \infty$ to skip nontrivial searching. And when $m' \geq n'$, $\boldsymbol{opt(n', m') = min(opt(n', m' - 1),\ opt(n' - 1, m' - 1) + |h_{m'} - l_{n'}|).}$ And the problem can be reconstructed by finding $opt(n, m)$. This can be proved simply with "cut-and-paste".

4. You know that $n + 2$ spies $S, s_1, s_2, s_3, \dots \dots, s_n,\ and\ T$ 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 s_i can send a secret message to spy s_j 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 on compromised channel.

   The question is a straightforward max flow problem. It can be form as a **flow network** with spies as vertices and channels as edges. Since we are interested in the message being passed between spy S and T. We can assign S as the **source** and T as the **sink**.

   If there is a connection between spy S and T directly, we can simply eavesdrop on that channel. And the smallest number in this case is one.

   Otherwise, we could cut the network into two sub-networks G_1 and G_2, which contain S and T respectively. And the number of channels we need to compromise is equal to the number of channels which start from G_1 and end at G_2, because all communications from S to T must pass though the cut.

   Thus, we can **assign ONE as the value of capacity of each channel**. Such that the number of compromised channels is equal to the cut of capacity. Now, we need to find the minimal compromised channels. Since the maximal amount of flow in a flow network is equal to the capacity of the cut of minimal capacity. This problem is equivalent to find the maximal amount of flow, and the flow network with spies as vertices and channels as edges with capacity of one. And we can further use Ford Fulkerson method to finalize the answer.

   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.

   In this question, we can treat the actual vertices as edges and vice versa.

   If there is a connection between spy S and T directly, there is no spy we can bribe. In this case, there is no solution.

   Otherwise, we reconstruct the graph to solve the problem. Firstly, we **duplicate** all spies except S and T. And we further connect the two spies with an **edge of a unit capacity**. And the **direction** of that edge is same as the direction flows in the vertex. Furthermore, we set the capacity of original edges as **infinity**. So that, the value dominates the cut and there will be no cut on the infinity edge.

Furthermore, we can solve this problem using the same method from a). And we can use the Ford Fulkerson method to finalize the answer.

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*.

   This question can be treated as a problem multiple sources and sinks. We can reduce the number of sources and sinks by adding a super-source and a super-sink.

   We denote **S as the super-source and T as the super-sink**. And we add the new connections $(S, s), (S, u), (t, T)\ and\ (v, T)$. We **set the capacity of these four links as infinity.** Since infinity dominates all other values of capacity, so that there will be no cut on these edges. Since the maximal amount of flow in a flow network is equal to the capacity of the cut of minimal capacity, we can use the **Ford Fulkerson** method to find the maximal flow of the new network. And the problem is solved.