COMP3121 Assignment1

Fiona Lin z5131048

April 22, 2019

1. [20 marks] Solution: For this question it can be approached by dynamic programming techquie.
Firstly, sort the proposals by their position x_i plus radius r_i, $x_i + r_i$ in increasing order so we can denote $f_i = x_i + r_i,\ s_i = x_i - r_i$
For every $i < N$ we are going to resolve the following subproblems:
subproblem $P(i)$: find a subsequence σ_i of the sequence of dams $D_i = \langle x_1, x_2, ..., x_i \rangle$ such that:

(1) σ_i consist of non-overleaping dams with their position x_i plus/minus radius r_i are disjoint interval, which means $f_{i-1} < s_i$.

(2) it ends with dam at x_i with $f_i = x_i + r_i$

(3) σ has the largest number of dams amoung all subsequence of D_i which satisfy 1 and 2.

Let $D(i)$ be the number of dams of the optimal solution $\sigma(i)$ of the subproblem $P(i)$
For $\sigma(1)$, we choose x_1; then $N(1) = 1$
Make an array with N slots to represent those proposals to store the $D(i)$
Recursion: assuming that we have solved subproblems for all $j < i$ and obtain the solution $N(j)$ and stored them in the array

$$D(i) = max\{N(j) + 1 : j < i \ \& \ x_j + r_j < x_i - r_i\}$$

Time complexity: to examine N proposals in the role of the last dame in an optimal sub-sequence and for each such dam, we have to find all preceeding compatible dams and their optimal solutions(to look up in an array). Thus the time complexity is $O(N^2)$.

2. Solution:

(a) **[10 marks]** There are 4 rows in every column, let's denote these 4 rows as sequence $S = \langle r_1, r_2, r_3, r_4 \rangle$. According to the question assuming the integers in each squares could be negative, and the adjacent squares are not allowed, therefore only (), (r_1), (r_2), (r_3), (r_4), (r_1, r_3), (r_1, r_4) and (r_2, r_4) are legal pattern that can occur in any column.

(b) **[10 marks]** As considering 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 occuring in the last column. Then we have known that there are 8 legal pattern that can be compatible with each other; hence denote these 8 types in a set T.
For $1 \leq k \leq n$
subproblem $P(k)$: find a type $T_k \in T$ to assigned in the column at k, such that:

(1) T_k provides the maximum sum of the integers in the squares in column k

(2) T_k is compatible with T_{k-1}

Let $V(k)$ be the maximum sum of the integers in the squares of first k columns, and $sum(T_k)$ be the sum of the integers in the squares of a type T_k at column k.
Make an array with n slots to represent those n columns to store the $V(k)$
Recursion: assuming that we have solved subproblems for all $1 \leq k \leq n$ and obtain the solution $V(k)$ and stored them in the array

$$V(k) = max\{sum(T_k) : T_k \in T\} + V(k-1)$$

Time complexity: to examine n columns in the row of the last column with maximum sum for every column, we have to find all preceeding compatible type and their maximum sum (look up in an array). Thus the overall time complexity is $O(8n)$ which is same as $O(n)$.

3. [20 marks] Solution: As given $\sum_{1 \leq i \leq n} |h_i - l_j(i)|$ as S.
To minimise S, we need to minimise the difference of between all pairs $(h_i, l_j(i))$.
Now, let's order all skiers S_i $1 \leq i \leq n$, by increasing height $h(S_i)$ and all skis $s_j, 1 \leq j \leq m$, by increasing length $l(s_j)$.
Because if an assignment is optimal and $h(S_i) < h(S_j)$ then the skis assigned to skier S_i are shorter than the skis assigned to skier S_j, otherwise you could swap their skis without increasing the sum of the absolute values of the differences between the heights of skiers and length of skis.
Since there are n skiers and m skis and $n \leq m$, we need to work out $d_{i,j} = |h_i - l_j(i)|$ for each assignment for n skiers and m skis and fill each $d_{i,j}$ in the n columns x m rows table. This takes $O(nm)time$
Therefore, for all i and j satisfying $1 \leq i \leq n$ and $i \leq j \leq m$,
subproblems $P(i, j)$ we need to find the sequence of assignment of skis $S = \langle l_1(1), l_2(2), ..., l_j(i) \rangle$ produce the minimum sum S.
Let $opt(i, S_i)$ be the optimal solution of subproblems $P(i, j)$ for all i and j satisfying $1 \leq i \leq n$ and $i \leq j \leq m$: Find optimal assignment of the first i skiers to chose from the first j skis. If $j = i$ then there is only one assignment that assigns skis according to the skiers height. If $j > i$ then there are two choices: either you assign to the i th skier skis j or you do not. It should now be easy now to do a recursion.
Recursion assuming we have solved all subproblems for all $i < j \leq m$ and obtain the optimal solution $opt(i, S_i)$ and store the sequence of assignment of skis in an Array S of n elements.

For ski at j row, we find the minimum value $v_{r,j}$ at column r, which is corresponding to skier ($v_{r,j}$ is the difference the absolute values of the differences between the height of skier at column r and length of skis at row j). Then we can assign j at position r in the Array S if the elements at position r in the Array S is not assigned before. When the element has been assigned value k before, in order to decide whether to replaces k with j at r position in Array S; we need to look up $v_{r,k}$ and compare $v_{r,k}$ and $v_{r,j}$. If $v_{r,k} > v_{r,j}$, we replaces k with j at r position in Array S.
The **Recursion** will go m times for all skis, which are $O(m)$ time. Hence the overall complexity would be $O((n+1)m)$ for this algorithm.

4. Solution: According to the question a), we know S and T spies do not send message via direct communication channels, otherwise the question become trivial.

(a) **[10 marks]** Given $n+2$ spies i.e. $S, s_1, s_2, ..., s_n, T$, and they are communicating through certain number of communications channels. We can consider the spies as vertices and the communication channels as edges in a graph.
In order to cut the communications flow with the fewest number of channels, we can apply the Edmonds-Karp Max Flow Algorithm. Therefore, the proposed algorithm to cut the communications flow with the fewest number of channels as following:

1) First form $n+2$ vertices V the graph and connect E pairs vertex i and vertex j with a edge of a unit capacity if there is a communication between them.
2) Knowing the flow through the cut $f(S,T) = \sum_{(u,v)\in E}\{f(u,v) : u \in S \ \& \ v \in T\} - \sum_{(u,v)\in E}\{f(u,v) : u \in T \ \& \ v \in S\}$.
Thus, Divide n intermediary spies(vertices) into one set contains vertex S, another set contains vertex T. We repeat the following steps 3) and 4).
3) For each $s \in S$ and each $t \in T$ preform breadth-first search to find the shorstest path between s and t vertices.
4) Start from the shorstest path as the argumenting path, we keep adding the flow $f(S,T)$ of the cut (s,t) to the argumenting path until there is no more argumenting path when the flow through the cut equal to the minimum capacity of the cut.

As each edge has one as the unit capacity, therefore, the min cut equals to the number of edges crossing the cut.
Time complexity: for each $s \in S$ and each $t \in T$, each BFS takes $O(E)$ time and each max flow finding takes $O((n+2)E)$ time. So the overall complexity is $O((n+2)E^2)$ time

(b) **[10 marks]** Since we bribe the spies instead of compromising the communication channels, we can model the question in another way which similar the above approach. We can consider the spies as edges and the communication channels as vertices in a graph.
In order to cut the communications flow with the smallest number of spies, we can

apply the Edmonds-Karp Max Flow Algorithm. Therefore, the proposed algorithm to cut the communications flow with the fewest number of channels as following:

1) First transform the previous $n + 2$ vertices V and E edges the graph into a new graph with $E + 2$ vertices(including S and T) and n edges of a unit capacity. We need to exclude S and T, because we can't bribe them.
2) Divide E intermediary spies(vertices) into one set contains vertex S, another set contains T. We repeat the following steps 3) and 4).
3) For each $s \in S$ and each $t \in T$ preform breadth-first search to find the shorstest path between s and t vertices.
4) Start from the shorstest path as the argumenting path, we keep adding the flow $f(S, T)$ of the cut (s, t) to the argumenting path until there is no more argumenting path when the flow through the cut equal to the minimum capacity of the cut.

As each edge has one as the unit weight, therefore, the min cut equals to the number of edges crossing the cut. Time complexity: for each $s \in S$ and each $t \in T$, each BFS takes $O(n)$ time and each max flow finding takes $O((E + 2)n)$ time. So the overall complexity is $O((E + 2)n^2)$ time

5. [20 marks] Solution: In order to find a cut of the smallest possible capacity among all cuts in which vertex u is at the same side as the source s and vertex v is at the same side as sink t, we can apply the Edmonds-Karp Max Flow Algorithm to find the minimum cut again.
Firstly, add the dummy super source s' and the dummy super sink t' to the flow network G, then connect (s', s) and (s', u) and (t, t') and (v, t') as edge and assign the infinity capacity to these edges. It can ensure $\{s, u\}$ and $\{v, t\}$ are always at the same side after the cut, because the flow is not block/cut on the infinity capacity edges.

Hence, we have a graph $G = (V, E)$, where $V = n + 2 > 6$ & $E \in [4, \frac{n(n-1)}{2} + 2]$
Then Divide V vertices into two Sets $s, s', u \in S$ and $t, t', v \in T$. We repeat the following steps 1) and 2).

1) For each $s \in S$ and each $t \in T$ preform breadth-first search to find the shorstest path between s and t vertices.
2) Start from the shorstest path as the argumenting path, we keep adding the flow $f(S, T)$ of the cut (s, t) to the argumenting path until there is no more argumenting path when the flow through the cut equal to the minimum capacity of the cut.

Time complexity in worst case each BFS each max flow finding for each $s \in S$ and each $t \in T$ take

$O(E) = O(\frac{n(n-1)}{2} + 2) = O(n^2)$ and $O(VE) = O(\frac{n(n-1)(n+2)}{2} + 2) = O(n^3)$

So the overall complexity is

$O(VE^2) = O((n+2)\big(\frac{n(n-1)}{2} + 2\big)^2) = O(n^5)$