COMP 9101

Assignment 3

Student ID : z5174741

Student name : GUANQUN ZHOU

Q1:

For this question, I think the method mentioned in “Activity Selection with maximal total duration”in lecture is the best option.

I will define **subproblem**:

$P(i)$: for $i^{th}(\,for\ all\ 1 < i < n)$ proposal, find a subset S_i of dams= $\{d_1, d_2, \ldots, d_{i-1}, d_i\}$ such that:

1. Satisfy the requirement that there is not another dam within r_i meteres (upstream or downstream).
2. S_i ends with dam d_i.
3. S_i is of maximum total number of dams among all subset of dams= $\{d_1, d_2, \ldots, d_{i-1}, d_i\}$ which satisfy requirement 1 and 2

Let σ_i be the number of the dams of the optimal solution of subproblem $P(i)$.

So the **recursion** is:

$$\sigma_i = \boldsymbol{max}(\,\sigma_i + 1, for\ all\ j < i, if\ x_j + r_j < x_i\ and\ x_i - r_i > x_j\,)$$

After we find optimal solution for every $i^{th}(\,for\ all\ 1 < i < n)$ proposal ,we need to go back to look for the maximum among all proposals and finally find the optimal solution:

$$Result = \mathbf{max}(\sigma_i\ , 1 \le i \le n)$$

Prove of the optimality:

Let the optimal solution of subproblem $P(i)$ be the subset:

$$S_i = \left(d_{k_1}, d_{k_2}, \ldots, d_{k_{m-1}}, d_{k_m}\right)\ where\ k_m = i$$

We claim: the truncated subset $S' = (d_{k_1}, d_{k_2}, \ldots, d_{k_{m-1}})$ is an optimal solution to subproblem $P(k_{m-1})$, where $k_{m-1} < i$

If there were a subset S^* of a larger total number of the dams than the number of subset S' and also ending with dam $d_{k_{m-1}}$, we could obtain a sequence $\hat{S}$ by extending the sequence S^* with dam d_{k_m} and obtain a solution for subproblem $P(i)$ with a longer total number of the dams than the total number of dams of sequence S_i , contradicting the optimality of S_i .

Time complexity:

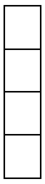
the algorithm computes n proposals and for each i^{th} proposal, it needs to compute all preceding solutions in order to find the optimal solution.

Thus, the time complexity is $O(n^2)$.

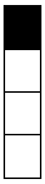
Q2:

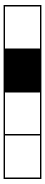
(a) There are **8** possible patterns:

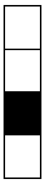
1. The empty patterns:

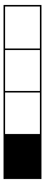
1 

2. **4** patterns where each pattern has exactly one pebble:

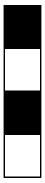
2 

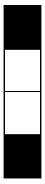
3 

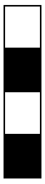
4 

5 

3. **3** patterns where each pattern has exactly two pebbles (on the first and third square, on the first and forth square, one the second and forth square):

6 

7 

8 

(b) For this question, the 2D recursion method is the optimal option.
So I will build a 2D table with 8 legal patterns as the table rows and n columns.
Then I define the **subproblem**:
$P(i,j)$: find the i^{th} pattern which is the optimal solution for the j^{th} column such that:

1. i is range from $1\ to\ 8$, j is range from $1\ to\ n$
2. The pattern is compatible with the pattern of $(j-1)^{th}$ column.
3. We can obtain the maximal sum of the integers in the squares that are covered by pebbles.

So the **recursion** is:
First we need to define v_j is the sum of the integers that are covered by pebbles of i^{th} pattern
we compute all possible patterns for j^{th} column. In this way, we can obtain all possible values of v_j and we can compute all sums of v_j and corresponding optimal solution for $P(i,j-1)$. Then we can choose the maximal sum from the results we obtain.
In other words:
In every recursion, we will compute a $r(i,j)$ for every possible pattern for j^{th} column.
And every $r(i,j)$ contains:
1. The sum of the v_j and corresponding optimal solution for $P(i,j-1)$.
2. The index of the chosen pattern of $P(i,j-1)$ so that for every possible pattern for j^{th} column we can obtain the maximal sum.

$$r(i,j) = (sum(i,j), index)$$

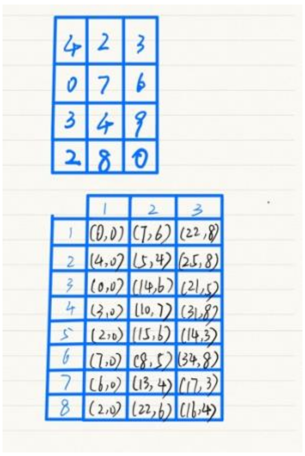
We define that $sum(i,j)$ is the maximal sum of first j columns and the index of the pattern of j^{th} column is i.
We will compute $sum(i,j)$ in this way:

$$sum(i,j) = \max(sum(k,j-1)\ \ for\ all\ 1 \le k \le 8\ and\ k\ is\ compatible\ with\ i) + v_j$$

We will compute $index$ in this way:

$$index\ = \ argmax_{1\le k\le 8}(sum(k,j-1)\ \ for\ all\ 1 \le k \le 8\ and\ k\ is\ compatible\ with\ i)$$

After obtaining the optimal solutions for the last column, we need to go back to look for the maximal sum among all the optimal solutions for the possible patterns for the last column and finally obtain the optimal placement.

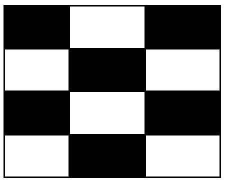


4	2	3
0	7	6
3	4	9
2	8	0

	1	2	3
1	(0,0)	(7,6)	(22,8)
2	(4,0)	(5,4)	(25,8)
3	(0,0)	(14,6)	(21,5)
4	(3,0)	(10,7)	(31,8)
5	(2,0)	(15,6)	(14,3)
6	(7,0)	(8,5)	(34,8)
7	(6,0)	(13,4)	(17,3)
8	(2,0)	(22,6)	(16,4)

The graph above is an example, and we say that the maximal sum of the integers in the squares that are covered by pebbles is 34 and the patterns chosen are 6-8-6.

The covered checkerboard should be like this:



Time Complexity:
This algorithm go through all n columns. On each column, we need to consider 8 patterns.
For every possible pattern, we need to calculate the optimal solution for this pattern by computing the sum of the value and corresponding previous optimal solution.
In the end, we need to go back to look for the maximal sum among all the optimal solutions for the possible patterns for the last column and finally obtain the optimal placement.
Thus, the time complexity is $O(n)$.

Q3:
For this question, the method mentioned in "Integer Knapsack Problem (no duplicate items)"would be the optimal option.
We need to build a 2D recursion to find the optimal solution.
First we define $h = (h_1, h_2, .., h_n)$ for the height of skiers and $l = (l_1, l_2, .., l_m)$ for the length of skis.
Note that $m \geq n$
Then we sort $h\ and\ l$ in increasing order.
Notice that if an assignment is optimal and $h_m < h_n$ then the skis assigned to skier h_m should be shorter than the skis assigned to skier h_n , otherwise you could swap their skis without increasing the sum of the absolute values of the difference between the heights of skiers and length of skis.
In other words:
If there exist $h_m, h_n\ and\ l_m, l_n$ and $h_m < h_n\ and\ l_m < l_n$, then the optimal assignment is: $assign\ l_m\ to\ h_m$ $and\ assign\ l_n\ to\ h_n$.

So the **subproblem** is :
$P(i, j)$: finding the optimal assignment of first j^{th} skis to first i^{th} skiers such that:
i is range from $1\ to\ n$, j is range from $1\ to\ m$

So the **recursion** is:
If $j = i$ then there is only one assignment that assigns skis according to the skier's height.

If $j > i$:
We need to compute the subproblem:
$P(i, j)$: = first j^{th} skis and first i^{th} skiers, such that $\sum_{1 \leq i < n} |h_i - l_{j(i)}|$ is minimized.

$$P(i,j) = \begin{cases} \min\left(P(i,j-1), P(i-1,j-1) + \left|h_i - l_{j(i)}\right|\right), if\ 1 \le i \le j \\ \infty \qquad\qquad\qquad\qquad\qquad\qquad if\ 1 \le j \le i \end{cases}$$

So now we have two options:

If $P(i-1,j-1) + \left|h_i - l_{j(i)}\right|) < P(i,j-1)$:

then $P(i,j) = P(i-1,j-1) + \left|h_i - l_{j(i)}\right|$

else $P(i,j) = P(i,j-1)$

If $P(i,j) = P(i-1,j-1) + \left|h_i - l_{j(i)}\right|$, assign $l_j\ to\ h_i$.

If $P(i,j) = P(i,j-1)$, continue to $P(i,j-1)$

Final solution will be given by $P(i,j)$

Q4:

(a) For this problem, we can treat it as a max flow problem.

Step 1:

We can treat S as source and treat T as sink.

Step 2:

We set the capacity of the edge starts from S as infinity capacity.

And we do the same to the weight of the edge ends in T.

Step 3:

We set the capacity of each other edge as unit capacity(1).

Step 4:

We use Edmond Karp algorithm to find the max flow. Meanwhile, we can obtain the minimal cut. As we set the capacity of each other edge as unit capacity (1), so the minimal cut is equal to the number of edges which the minimal cut goes through, which means the minimal cut is equal to the number of channels we need to comprise.

(b) For this problem, the method we need is similar to the one mentioned in (a).

What different is that we need to duplicate each spy vertex and connect the two vertices with an edge of a unit capacity(1).

And we can set the capacity of all other edges as 0.(In other word, we can treat the edges in original graph as vertices transformed from duplicate vertices).

Then we repeat the 4 steps in (a) and we will obtain the minimal cut finally.

After we have already obtained the minimal cut, since we duplicate each spy vertex and connect the two vertices with an edge of a unit capacity(1), so according to the minimal cut ,we can obtain the number of cut edges which connect two duplicate vertices. And that number is the number of the spies we have to bribe.

Q5:

For this problem, we need 3 steps.

Step1:

We add two edges: $s \longrightarrow u$ and $v \longrightarrow t$.

Step2:

We set the capacity of both edges $s \longrightarrow u$ and $v \longrightarrow t$ as infinity capacity.

Step3:

We treat this problem as max flow problem. We use Edmond Karp. algorithm to find the max flow. Meanwhile, we can obtain the minimal cut.