

Problem 5:

言語對象：
 b07207063 廖政華 b06303131 沈家睿 b05504066 李曼翰 b08501098 柯晨緯
 b09902068 凌 暉 b08502041 李芸芳 p10922001 黃佳琪

Subproblem(a) :

(1) 我們可假設 $E(a, b) = E(b, a)$ (即對稱的),

所以可知 $dp(i, \bar{j}) = dp(\bar{j}, i)$, 且 $i \neq \bar{j}$

因 $dp(i, \bar{j}) = dp(\bar{j}, i)$, 故只列出當 $i > \bar{j}$ 時的轉移式,

$i < \bar{j}$ 時的 $dp(i, \bar{j})$ 可由 $dp(\bar{j}, i)$ 得到。以下為 dp 的轉移式：

$$dp(i, \bar{j}) = \begin{cases} E(0, 1), & \text{if } (i, \bar{j}) = (1, 0) \\ dp(i-1, \bar{j}) + E(i-1, \bar{j}), & \text{if } i > \bar{j} + 1 \\ \min_{0 \leq i' < \bar{j}} (dp(i', \bar{j}) + E(i', \bar{j})), & \text{if } i = \bar{j} + 1 \end{cases}$$

定義 $\text{state}(i, \bar{j})$ 為去程最遠取到 S_i , 回程最遠取到 $S_{\bar{j}}$

① $dp(1, 0) = E(1, 0) \Rightarrow \text{trivial}$

② 當 $i > \bar{j}$ 時, $S_{\bar{j}+1}$ 到 S_i 的每塊石頭皆在去程時被足采
 可知當 $i > \bar{j} + 1$ 時, 前一個 state 必是 $(i-1, \bar{j})$
 故 $dp(i, \bar{j}) = dp(i-1, \bar{j}) + E(i-1, \bar{j})$

③ 當 $i = \bar{j} + 1$ 時, 前一個 state 有可能是 $\{(0, \bar{j}), (1, \bar{j}), \dots, (\bar{j}-1, \bar{j})\}$
 其中 $i -$, 故 $dp(i, \bar{j}) = \min_{0 \leq i' < \bar{j}} (dp(i', \bar{j}) + E(i', \bar{j}))$

(因為要 minimal cost, $\bar{j}+1$ 以上一個 state 的花費加上 $E(i', \bar{j})$ 一定要是
 是所有可能的 previous state 中最小的)

($i = \bar{j}$ 的 state 不符題目邏輯, 故不列出)

Correctness proof: ($i > j$)

(i) base case: $dp(1, 0) = E(1, 0)$ is the optimal solution to the state $(1, 0)$ (trivial)

(ii) $(i, j) \neq (1, 0)$:

case 1. $i > j + 1$:

If $dp(i, j) = dp(i-1, j) + E(i-1, i)$ is not optimal then $dp(i-1, j)$ is not the optimal solution to the state $(i-1, j)$ (代表 state $(i-1, j)$ 的 dp 值不是 minimal cost 會有更小的值)

So if $dp(i-1, j)$ is the optimal to the state $(i-1, j)$ then $dp(i, j)$ is the optimal solution to the state (i, j)

case 2. $i = j + 1$

Assume $dp(i', j)$ be the optimal sol. to the state (i', j) ($0 \leq i' < j$)

Let $\min_{0 \leq i' < j} (dp(i', j) + E(i', i)) = dp(i'', i) + E(i'', i)$

If $dp(i, j) = dp(i'', i) + E(i'', i)$ is not optimal then $dp(i'', i)$ is not the optimal solution to the state (i'', i) (代表 state (i'', i) 的 dp 值不是 minimum cost 會有更小的值)

So if $dp(i'', i)$ is the optimal to the state (i'', i) then $dp(i, j)$ is the optimal solution to the state (i, j)

\Rightarrow 由(i)(ii), 可知當小問題的 optimal solution 是其 dp 值

則大問題的 optimal solution 也是其 dp 值

且 base case 的 dp 值是 optimal 的

故任一 state 的 dp 值也是 optimal

Time Complexity:

固定 j , i 由 $j+1$ 到 N 的所有 state 中, 除了 $(j+1, j)$ 這個 state 需 $O(N)$ 時間轉換, 其他皆只要 $O(1)$, 平均佳得 $(j+1, j) \sim (N, j)$ 的 dp 值計算時間加總為 $O(N)$, 而 i 由 $0 \rightarrow N-2$, 故其需 $O(N^2)$ 時間完成

(2)

我們需要兩個長度為 N 的 array

arr1 來記錄目前計算到的 dp 值

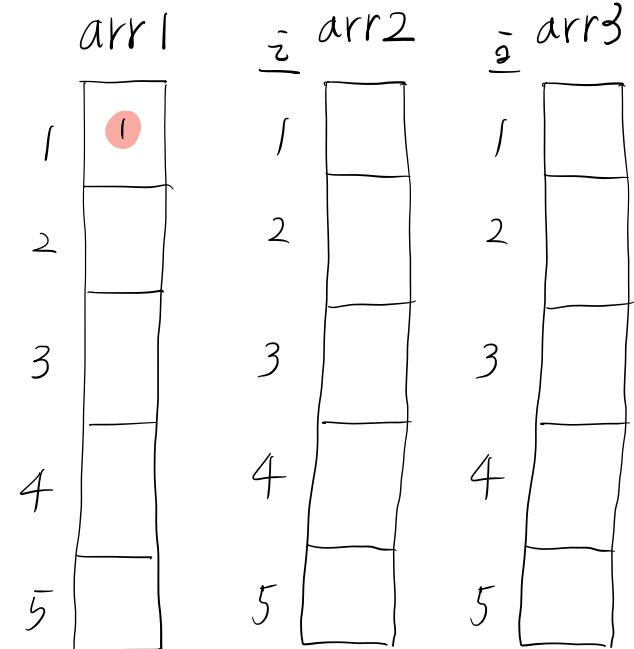
(由以下示意圖可知，只需求紀錄前 i^{th} column 的值即可)

arr2 來記錄當 $i = j + 1$ 時，由哪個 previous state 推得 current state 的 dp 值

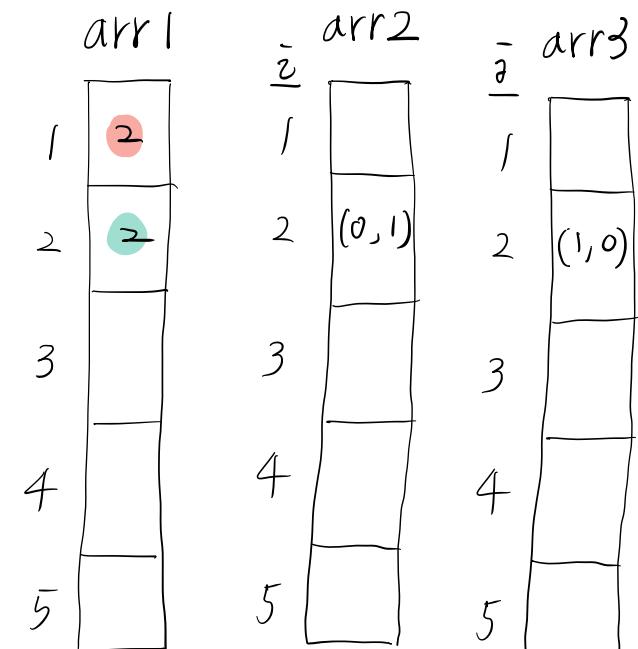
arr3 是 arr2 一樣，只是用來記錄 $j = i + 1$ 時的 previous state.

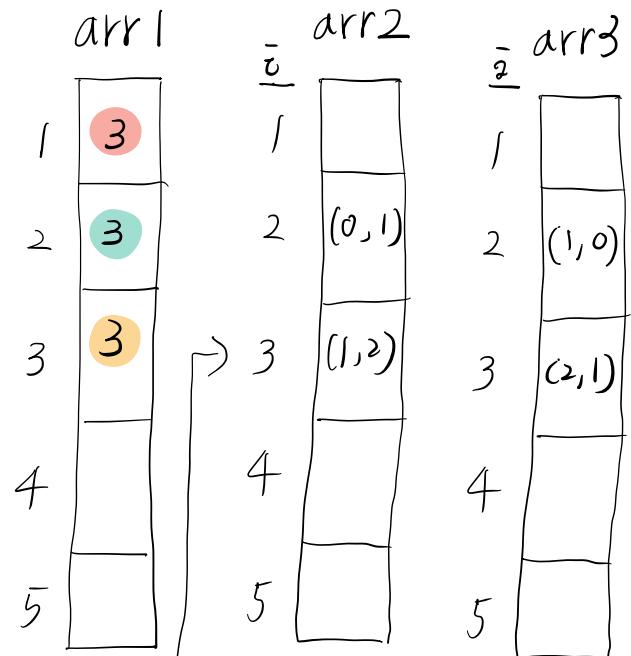
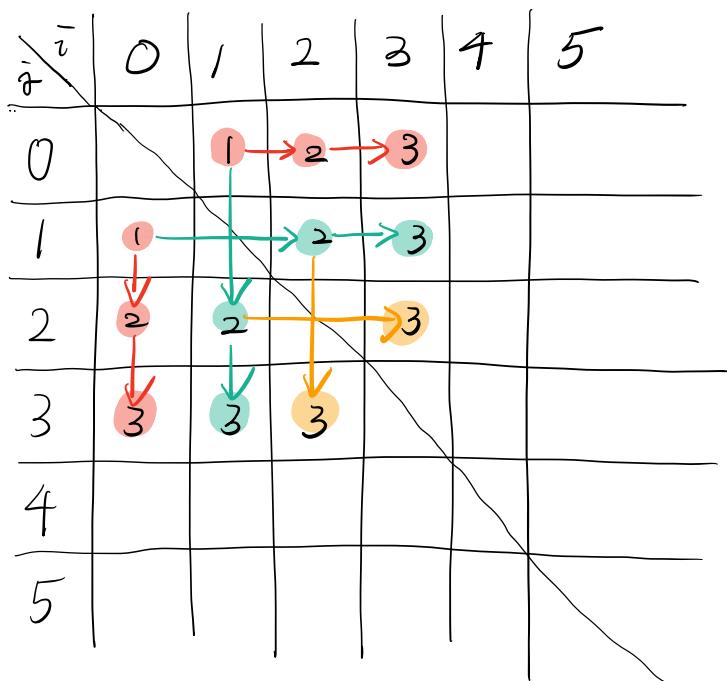
$1 \times N = 6$ 為例 (dp table)

i	0	1	2	3	4	5
j						
0		1				
1	1					
2						
3						
4						
5						

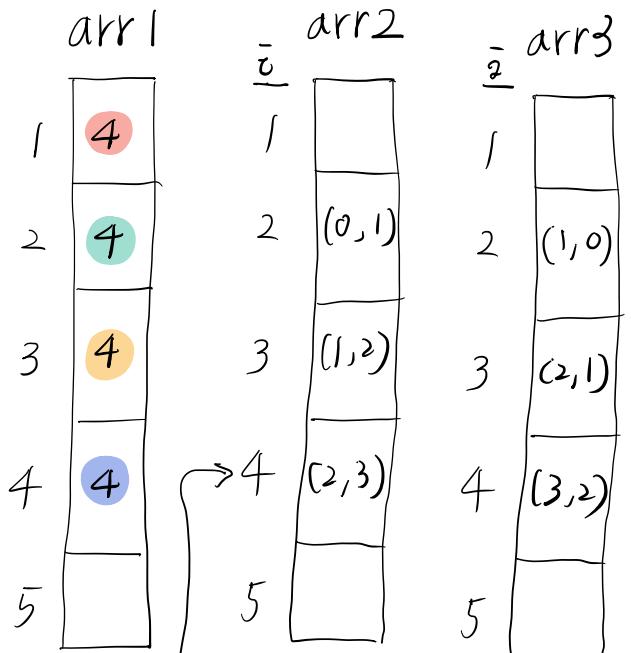
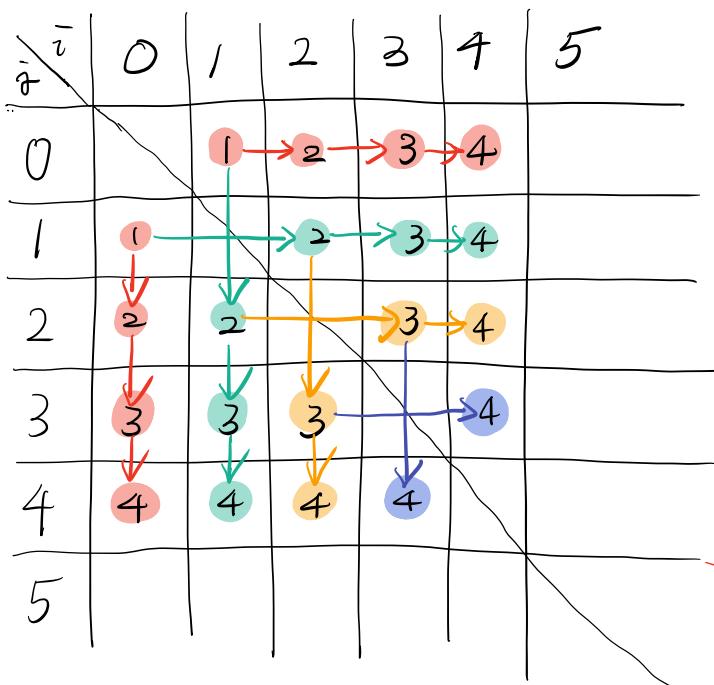


i	0	1	2	3	4	5
j						
0		1	1 → 2			
1	1		2 → 2			
2	2	2				
3						
4						
5						

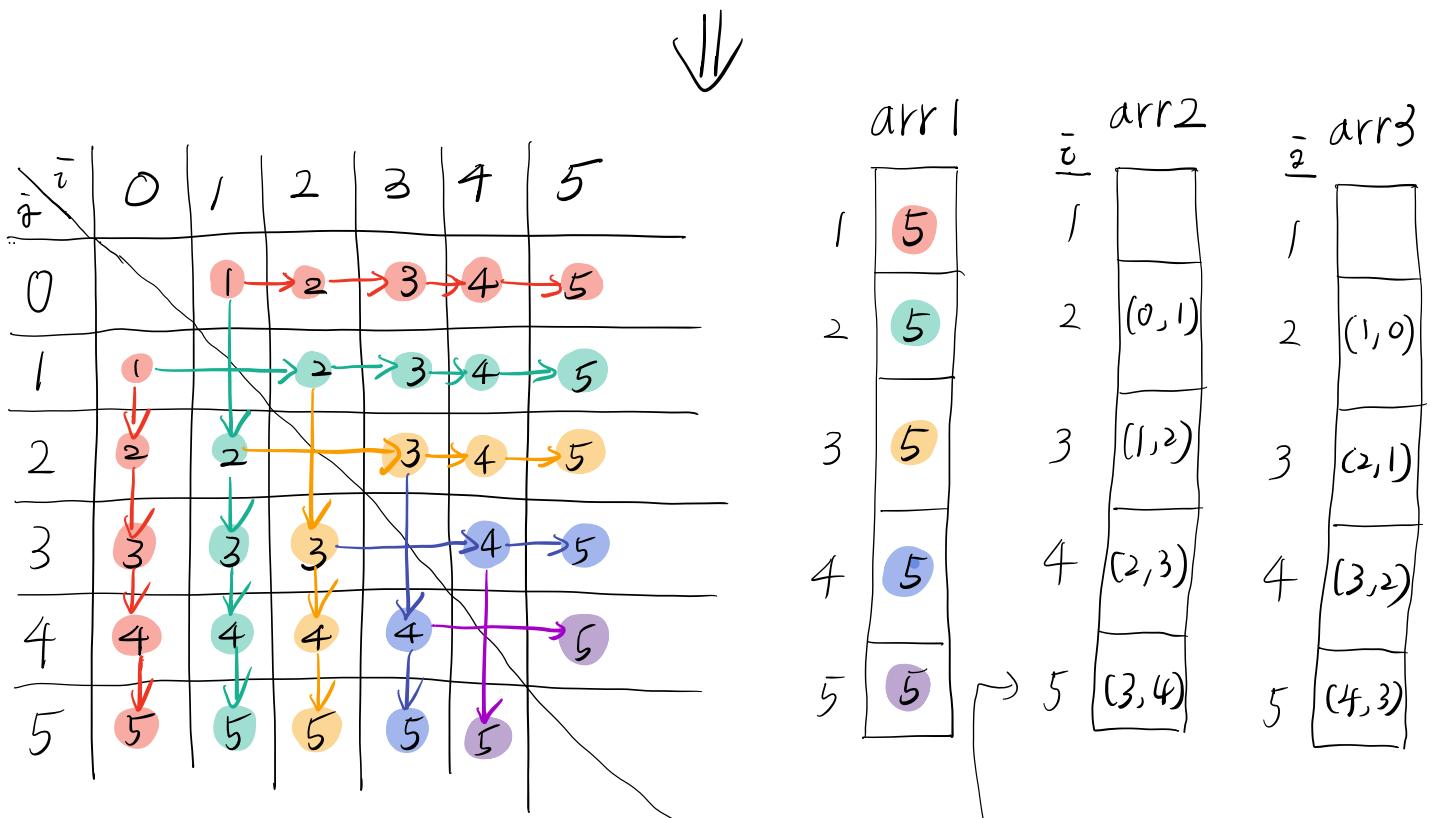




$$\min(dp(0,2) + E(0,3), dp(1,2) + E(1,3)) \\ = dp(1,2) + E(1,3)$$



$$\min(dp(0,3) + E(0,4), dp(1,3) + E(1,4), dp(2,3) + E(2,4)) \\ = dp(2,3) + E(2,4)$$



假設 $\min(dp(0,4) + E(0,5), dp(1,4) + E(1,5), dp(2,4) + E(2,5), dp(3,4) + E(3,5))$
 $= dp(3,4) + E(3,5)$

這樣一來，arr1 內所存的，就是 $dp(N-1,0) \sim dp(N-1,N-2)$ 的所有值
 接著找出

$\min(dp(N-1,0) + E(N,0) + E(N-1,N), \dots, dp(N-1,N-2) + E(N,N-2) + E(N-1,N))$
 即可得 minimum cost.

只用了 3 個 N -sized array，故 space complexity: $O(N)$

(3) 接續(2)所用的 arr2, arr3

當我們確定最後的 minimum cost 是由那個狀態的 dp 值
 而得後，就可以追往而得上一個 state：

if $i > j+1$ or $j > i+1$ ，則上一個 state 便是：

$(i-1, j)$ if $i > j+1$

$(i, j-1)$ if $j > i+1$

(ii) 如果 $\bar{i} = \bar{j} + 1$, 則上一個 state 就儲存在 $\text{arr}_2[\bar{i}]$

(iii) 如果 $\bar{j} = \bar{i} + 1$, 則上一個 state 就儲存在 $\text{arr}_3[\bar{j}]$

從每次的 state 變更就能知道哪塊石頭在去程、哪塊在回程

因為只用到 arr_2 和 arr_3 , 故共花 $O(N)$ space complexity

(如果 back-trace 到 (\bar{i}, \bar{j}) , 代表 $S_{\bar{i}}$ 在去程, $S_{\bar{j}}$ 在回程)

回程 Path 需要花 $O(N)$ 時間 (因為 go through 每一顆石頭)

故整體演算法複雜度仍是 $O(N^2)$

subproblem (b) :

(1)

• If $\bar{i} > \bar{j}$:

$$dp(\bar{i}, \bar{j}, h) = \begin{cases} \infty, & \text{if } h \leq 0 \\ E(0, 1), & \text{if } h > 0 \text{ and } (\bar{i}, \bar{j}) = (1, 0) \\ \infty, & \text{if } h \leq D_1 \text{ and } (\bar{i}, \bar{j}) = (1, 0) \\ dp(\bar{i}-1, \bar{j}, h-D_{\bar{i}}) + E(\bar{i}-1, \bar{i}), & \text{if } h > 0 \text{ and } \bar{i} > \bar{j} + 1 \\ \min_{0 \leq i' < j} (dp(\bar{i}', \bar{j}, h-D_{\bar{i}}) + E(\bar{i}', \bar{i})), & \text{if } h > 0 \text{ and } \bar{i} = \bar{j} + 1 \end{cases}$$

• If $\bar{j} > \bar{i}$:

$$dp(\bar{i}, \bar{j}, h) = \begin{cases} \infty, & \text{if } h \leq 0 \\ E(1, 0), & \text{if } h > 0 \text{ and } (\bar{i}, \bar{j}) = (0, 1) \\ dp(\bar{i}, \bar{j}-1, h) + E(\bar{j}, \bar{j}-1), & \text{if } h > 0 \text{ and } \bar{j} > \bar{i} + 1 \\ \min_{0 \leq j' < i} (dp(\bar{i}, \bar{j}', h) + E(\bar{j}', \bar{j}')) & \text{if } h > 0 \text{ and } \bar{j} = \bar{i} + 1 \end{cases}$$

($dp(\bar{i}, \bar{j}, h)$ 為當可用面量是 h 時, 去程最遠達到 $S_{\bar{i}}$, 回程最遠達到 $S_{\bar{j}}$ 所耗費的 minimal cost)

最後找出:

$$\min(dp(N-1, 0, H) + E(N-1, N) + E(N, 0), \dots, dp(N-1, N-2, H) + E(N-1, N) + E(N, N-2),$$

$$dp(0, N-1, H) + E(0, N) + E(N, N-1), \dots, dp(N-2, N-1, H) + E(N-2, N) + E(N, N-1))$$

就是最小花費了。(如果 minimal cost = ∞ 代表到達不了對面)

(2) Correctness proof:

當 $h \leq 0$, 因此狀態不可能達成, 故 cost 是 ∞

$$\begin{aligned} \text{當 } (\bar{i}, \bar{j}) = (1, 0) : & \left\{ \begin{array}{l} \text{if } h > D_1, dp(\bar{i}, \bar{j}, h) = E(0, 1) \\ \text{if } h \leq D_1, dp(\bar{i}, \bar{j}, h) = \infty \end{array} \right. \quad \left. \begin{array}{l} \text{這些 base case} \\ \Rightarrow \text{該 dp 值為其 optimal solution} \\ \text{(Trivial)} \end{array} \right. \\ \text{當 } (\bar{i}, \bar{j}) = (0, 1) : & dp(\bar{i}, \bar{j}, h) = E(1, 0) \end{aligned}$$

當 $h > 0, \bar{i} > \bar{j}$:

(i) $\bar{i} > \bar{j} + 1$, 因為 $S_{\bar{j}+1} \sim S_{\bar{i}}$ 都是算入步驟, 故 (\bar{i}, \bar{j}, h) 的上一個 state 只有可能是 $(\bar{i}-1, \bar{j}, h-D_{\bar{i}})$ 這個 state.

\Rightarrow if $h - D_{\bar{i}} < 0, dp(\bar{i}-1, \bar{j}, h-D_{\bar{i}}) = \infty$ is the optimal sol. to the state $(\bar{i}-1, \bar{j}, h-D_{\bar{i}})$, then $dp(\bar{i}, \bar{j}, h)$ will be ∞ and is the optimal sol. to the state (\bar{i}, \bar{j}, h) .

if $h - D_{\bar{i}} > 0$, if $dp(\bar{i}-1, \bar{j}, h-D_{\bar{i}})$ is the optimal sol. to the state $(\bar{i}-1, \bar{j}, h-D_{\bar{i}})$, then $dp(\bar{i}, \bar{j}, h)$ is the optimal sol. to the state (\bar{i}, \bar{j}, h) , otherwise there must exist a lower cost in state $(\bar{i}-1, \bar{j}, h-D_{\bar{i}})$ and $dp(\bar{i}-1, \bar{j}, h-D_{\bar{i}})$ will not be the optimal sol.

(ii) $\bar{i} = \bar{j} + 1$:

假設 $dp(\bar{i}', \bar{j}, h-D_{\bar{i}})$ 是 state $(\bar{i}', \bar{j}, h-D_{\bar{i}})$ 的 optimal sol. ($0 \leq \bar{i}' < \bar{j}$)

若 $\min_{0 \leq \bar{i}' < \bar{j}} (dp(\bar{i}', \bar{j}, h-D_{\bar{i}}) + E(\bar{i}', \bar{i})) = dp(\bar{i}'', \bar{j}, h') + E(\bar{i}'', \bar{i})$

則 $dp(\bar{i}, \bar{j}, h) = dp(\bar{i}'', \bar{j}, h') + E(\bar{i}'', \bar{i})$ 會是 optimal sol. to the state (\bar{i}, \bar{j}, h) , 否則代表在 state (\bar{i}'', \bar{j}, h') 有比 $dp(\bar{i}'', \bar{j}, h')$ 更小的 cost, $dp(\bar{i}'', \bar{j}, h')$ 非 optimal sol.

由(i)(ii)可知因 base case 該 dp 值為其 optimal solution, 且 previous state 該 dp 值是其 optimal solution 時, current state 該 dp 值也是同理可證 $\bar{i} < \bar{j}$ 的所有 cases, 故此迴歸式正確。

Time complexity:

當 $i > h$ ，在固定 \bar{x}, h 時，除了 $dp(\bar{x}+1, \bar{x}, h)$ 需花 $O(N)$ ，其他都只花 $O(1)$ 時間。
且換往後在 \bar{x}, h 固定時的 dp 值共花 $O(N)$ 時間計算，且 \bar{x} 由 $0 \rightarrow N-2$ ， h 由 $0 \rightarrow H$ 。
故共花 $O(HN^2)$ 。同理可得 $i < \bar{x}$ 時的複雜度也是 $O(HN^2)$ ，故全部共花 $O(HN^2)$ 。

Problem 6:

討論題子題：

b07207063 廖政華 b06303131 沈家睿 b05504066 李曼翰 b08501098 柯晨緯
b09902068 凌 暱 b08502041 李芸芳 p10922001 黃佳琪

(1) Under no assumption: 98141210

Under Assumption 3: 981120

(2) 以 merge sort 去 sort，由大到小排序 input 的數字
再由大到小輸出每一個數，串接起來即是答案，只花 $O(N \log N)$ 時間
(\because merge sort)

greedy choice：把大的數先填入，若有 OPT 先把小的數填入
將它換 greedy choice 的填入方法會得更大的數，故 optimal solution
必照此 greedy choice。

(3) 以 merge sort 以下次序以下次序排序，並由前至後串接字串即是答案
只花 $O(N \log N)$ 時間 (\because merge sort)

次序：若字串 S_1, S_2 串接， $S_1 + S_2$ 的數值不 $\leq S_2 + S_1$
則 S_1 排前 S_2 排後 ex. "99", "996" \Rightarrow "99" 應排前
(每次比較只會花 $O(1)$, 因為字長 ≤ 4) "11", "112" \Rightarrow "112" 應排前

此方法為 divide & conquer

每次都把能變成最大的數的字串往前擺，因此最後得到的數
會是最大的。

(4) 先以counting sort 排序每一個 input digit，並由大到小串接成一串數字，令其為 N ，在串接的同時先加高恩每個 digit 的值，令其為 S

Case 1：如果 $S \% 3 = 0$ ，則 N 為答案

Case 2：如果 $S \% 3 = 1$ ，則從 N 的 LSD 開始往 MSD 找要刪的 digit
先宣告兩變數 $\text{tmp1} = -1$, $\text{tmp2} = -1$ (初始化)

① 若遇到 $\text{digit \% 3} = 0$ ，直接跳過該 digit 繼續往前找

② 若遇到 $\text{digit \% 3} = 1$ ，直接刪掉該 digit 得到之字符串為答案

③ 若遇到 $\text{digit \% 3} = 2$ ，如果 tmp1 或 tmp2 其中一個仍為初始值就令該 digit 為其值，否則就跳過，如果到整串字符串檢查結束前都沒有找到 mod 3 為 1 的其他 digit，就刪掉 tmp1 和 tmp2 所紀錄的兩個 digit 即為答案

Case 3：如果 $S \% 3 = 2$ ，則從 N 的 LSD 開始往 MSD 找要刪的 digit
先宣告兩變數 $\text{tmp1} = -1$, $\text{tmp2} = -1$ (初始化)

① 若遇到 $\text{digit \% 3} = 0$ ，直接跳過該 digit 繼續往前找

② 若遇到 $\text{digit \% 3} = 2$ ，直接刪掉該 digit 得到之字符串為答案

③ 若遇到 $\text{digit \% 3} = 1$ ，如果 tmp1 或 tmp2 其中一個仍為初始值就令該 digit 為其值，否則就跳過，如果到整串字符串檢查結束前都沒有找到 mod 3 為 2 的其他 digit，就刪掉 tmp1 和 tmp2 所紀錄的兩個 digit 即為答案

以上如果被先是空字符串，就 output 0.

因為是用 counting sort 排序，只需 $O(N)$ 時間。刪的字元最多 2 個故也只需 $O(N)$ 時間刪掉字元，故加高恩時間為 $O(N)$

以上是使用greedy, greedy choice 是盡量將較大的數字往前排
如果有一OP_T所得之答案，在計算過程中若插入較小的數，那先插入
較大的數會得更大的結果，故所有的OP_T都包含此greedy choice。
拔數字的過程也是greedy，就是盡量拔取较少且較小的數，如果有OP_T
不是照此greedy choice，那將其換成此greedy choice 會得較大的數，因為會
拔掉比較少或一樣少但比較小的 digits.

(5) 98653

(6).

(以下所說的字典序是如果一個string是另一個string的prefix，較長者字典序較大；否則就一個字元一個字元由左至右比較，當比較到兩個不同字元時，具有較大值的字元之string是字典序較大者)

(MSD=most significant digit; LSD=last significant digit)

(以下陣列index都由1開始)

首先找在維持原order的條件下， P 可構成的最大 x 位數，以及 M 可構成的最大 $k - x$ 位數，並將他們combine成最大的 k 位數，再比較當 $\max(0, k - n) \leq x \leq \min(k, n)$ ，所有可能的 k 位數中哪個最大即為答案。

1. 在找維持原order的條件下所能找到的最大 x 位數時，我們先建立一個空的string叫 arr ， arr 的每個digit，我們都要盡可能填入較大的數。所以在檢查 P (或 M ，這裡以 P 舉例)中的每個數能不能填入的時候，要從LSD of arr 檢查到 arr 中由右往左第 j 個digit ($j = \min(n - i + 1, k)$)，如果 P_i 大於某個digit就用 P_i 取代該digit然後持續往左檢查 arr ；若最終取代的是 arr 中由右至左第 h 個digit，那 $arr[h]$ 以後的digits都會移出 arr (也就是 $arr[h]$ 變成LSD)；如果都沒有取代且 $arr.size \neq k$ 那就將 P_i 串接在 arr 之後，最後得到的 arr 就是最大的 x 位數。由於填入每個 P_i 都要往 arr 的左邊檢查最多 k 個digit，所以在建構 P 、 M 所各自得到的數時會花 $O(kn)$ 的複雜度。
 2. 在combine P 所得到的 x 位數和 M 所得到的 $k - x$ 位數時，我們先設一個空的string叫 arr ，我們要不斷比較兩者的字典序，字典序較大者的MSD要push進 arr ，並且拔掉該MSD，然後持續上面步驟直到所有數都被push進去，最終得到的 arr 就是 P 所得到的 x 位數和 M 所得到的 $k - x$ 位數combine起來能得到的最大 k 位數。每個數再插入之前都得比較字典序，每次比較都得花 $O(n)$ 的時間，所以共花 $O(n^2)$ 的時間combine。
- 由於取數字的可能一定是 P 取 x 個， M 取 $k - x$ 個($\max(0, k - n) \leq x \leq \min(k, n)$)，所以每種可能的組合都有考慮到。因為至多 k 種組合，每個組合在建構 P 、 M 所能構成的最大數並combine所花的時間是 $O(n^2)$ ，所以演算法總時間是 $O(kn^2)$ 。
 - 從 P 中取 x 位數的greedy choice是盡量將較大的數往前填入，如果有令一個最佳解 OPT 不是這樣子取，則該 OPT 所得之解必有某位digit可以換成更大的數，故該 OPT 便非最佳解，可知此greedy choice可以讓我們得大最大的 x 位數。
 - Combine時的greedy choice是把字典序較大者的MSD先排入，如果不按照此方式combine，將排入方法換成greedy choice將得到至少一樣好的結果。例如以下範例：

```
[6, 8, 7], [6, 8, 8] --(不照greedy choice)--> 687688
```

```
[6, 8, 7], [6, 8, 8] --(greedy choice)--> 688687
```