1 ADA HW2 P5 Answer

1.1 a

1.1.1

$$dp(i,j) = \left\{ \begin{array}{lr} 0, & \text{for } i,j = 0,0 \\ dp(i-1,j) + E(i-1,i), & \text{for } i-j > 1 \\ dp(i,j-1) + E(j,j-1), & \text{for } j-i > 1 \\ \min\limits_{0 \le i^* \le j} dp(i^*,j) + E(i^*,i), & \text{for } i-j = 1 \\ \min\limits_{0 \le j^* \le i} dp(i,j^*) + E(j,j^*), & \text{for } j-i = 1 \\ \infty, & \text{otherwise} \end{array} \right\}$$

dp(0,0) is the cost of $\{0 \Rightarrow N \Rightarrow 0\}$ and then minus $E(0,N) + E(N,0)$ which is equal to 0.

In the case $i - j > 1$, if there is a path $\hat{P}$ with cost $\hat{C}$ which is smaller than dp(i,j), we can find the last stone $\hat{i}$ that is on the $\hat{P}$ and behind i. $\hat{i}$ is $i-1$ because $i - j > 1$ and all the stone behind S_i should be chosen. We can construct a P' with cost C' is $\hat{P}$ without i.

$C' = \hat{C} - E(i-1,i) < dp(i,j) - E(i-1,i) \le dp(i-1,j)$

In P', the last stone on the way go is $i-1$ and the first stone on the way left is j, so C' most greater or equal than $dp(i-1,j)$. $\Rightarrow\Leftarrow$.

In the same way, we can proof the correctness of the case $j - 1 > 1$

In the case $i - j = 1$, if there is a path $\hat{P}$ with cost $\hat{C}$ which is smaller than dp(i,j), we can find the last stone $\hat{i}$ that is on the $\hat{P}$ and behind i. $\hat{i}$ can be the number from 0 to j. We can construct a P' with cost C' is $\hat{P}$ without i.

$C' = \hat{C} - E(\hat{i},i) < dp(i,j) - E(\hat{i},i) \le dp(\hat{i},j) + E(\hat{i},i) - E(\hat{i},i) \le dp(\hat{i},j)$

In P', the last stone on the way go is $\hat{i}$ and the first stone on the way left is j, so C' most greater or equal than $dp(\hat{i},j)$. $\Rightarrow\Leftarrow$.

In the same way, we can proof the correctness of the case $j - 1 = 1$

1.1.2

Because we need to choose all the stone, the last stone on the way go or the first stone on the way left should be $N-1$, so we can find the answer:

$\min(\min\limits_{j^*} dp(N-1,j^*) + E(N,j^*) + E(N-1,N), \min\limits_{i^*} dp(i^*,N-1) + E(i^*,N) + E(N,N-1))$

In the recursive function, there are N^2 status needs to count. It cost $O(N)$ times in $2N$ cases ($|i-j| = 1$) and $O(1)$ times in other cases, and then count the final answer in $O(2N)$ times. So, we can solve it in $O(N^2)$ time complexity.

Define two 2D array DP_i and DP_j to save the value of $dp(i,j)$ for $i >= j$ and $i <= j$. In the recursive function, $DP_i[i][]$ needs $DP_i[i-1][]$ and $DP_j[i-1][]$, and so is $DP_j[i][]$. So we can roll the first dimension, the size of DP_i and DP_j would be $(2, N-1)$. It takes $O(N)$ space.

1.1.3

For all dp state $|i-j| > 1$, we can directly find the previous status that is $i-1$ behind i or $j-1$ after j, so we only need to label the previous state when $|i-j| = 1$. It needs extra $O(N)$ space.

1.2 b

1.2.1

$dp(i,j,h)$ is the same status of sub-problem(a) that add a constraint that the health point is greater or equal then h.

$$dp(i,j,h) = \begin{cases} 0, & \text{for } i,j = 0,0, h \leq H \\ dp(i-1,j,h+D(i)) + E(i-1,i), & \text{for } i-j > 1 \\ dp(i,j-1,h) + E(j,j-1), & \text{for } j-i > 1 \\ \min\limits_{0\leq i^* \leq j} dp(i^*,j,h+D(i)) + E(i^*,i), & \text{for } i-j = 1 \\ \min\limits_{0\leq j^* \leq i} dp(i,j^*,h) + E(j,j^*), & \text{for } j-i = 1 \\ \infty, & \text{otherwise} \end{cases}$$

Answer is
$\min(\min\limits_{j^*} dp(N-1,j^*,1) + E(N,j^*) + E(N-1,N), \min\limits_{i^*} dp(i^*,N-1,1) + E(i^*,N) + E(N,N-1))$

1.2.2

For $h \leq H$, dp(0,0, h) is the cost of $\{0 \Rightarrow N \Rightarrow 0\}$ and then minus $E(0,N) + E(N,0)$ which is equal to 0.

In the case $i-j > 1$ and the health point h, if there is a path $\hat{P}$ with cost $\hat{C}$ which is smaller than dp(i,j,h), we can find the last stone $\hat{i}$ that is on the $\hat{P}$ and behind i. $\hat{i}$ is $i-1$ because $i-j>1$ and all the stone behind S_i should be chosen. We can construct a P' with cost C' is $\hat{P}$ without i. In P', the health point is greater than $h + D(i)$, or you cannot keep the health point greater than h in current path.

$C' = \hat{C} - E(i-1,i) < dp(i,j,h) - E(i-1,i) \leq dp(i-1,j,h+D(i))$

In P', the last stone on the way go is $i-1$ and the first stone on the way left is j, so C' most greater or equal than $dp(i-1,j,h+D(i))$. $\Rightarrow\!\Leftarrow$.

In the same way, we can proof the correctness of the case $j-1>1$(In this case, your health point keep the same because it is no bomb on the way left).

In the case $i-j=1$ and the health point h, if there is a path $\hat{P}$ with cost $\hat{C}$ which is smaller than dp(i,j), we can find the last stone $\hat{i}$ that is on the $\hat{P}$ and behind i. $\hat{i}$ can be the number from 0 to j. We can construct a P' with cost C' is $\hat{P}$ without i. In P', the health point is greater than $h + D(i)$.

$C' = \hat{C} - E(\hat{i},i) < dp(i,j,h) - E(\hat{i},i) \leq dp(\hat{i},j,h+D(i)) + E(\hat{i},i) - E(\hat{i},i) \leq dp(\hat{i},j,h+D(i))$

In P', the last stone on the way go is $\hat{i}$ and the first stone on the way left is j, so C' most greater or equal than $dp(\hat{i},j,h+D(i))$. $\Rightarrow\!\Leftarrow$.

In the same way, we can proof the correctness of the case $j-1=1$(In this case, your health point keep the same because it is no bomb on the way left).

There are HN^2 status. It costs $O(HN^2)$ times in HN^2 cases and costs $O(1)$ in other cases. So, it takes $O(HN^2)$ times.

2 ADA HW2 P6 Answer

The answer are shown in the following content.

1. (2%) Given preference value of 6 courses $P = [1,\ 0,\ 9,\ 81,\ 4,\ 12]$, can you find the maximum satisfying value under no assumption and under **Assumption 3**, respectively?
   9814120, 981120

2. (2%) Please give a $O(n\log(n))$ algorithm to find the maximum satisfying value under **Assumption 1**.
   First, Sort array P in descending order, it takes $O(n\log(n))$. Next, choose the number from the first one to the last one and push them into the answer string, it takes $O(n)$, and we get the answer.

   Time Complexity: $O(n\log(n)) + O(n) = O(n\log(n))$

   Correctness: Suppose the string create by above algorithm is S. For any other strings S' create by P and isn't sorted in descending order, we can always find two numbers S'_i and S'_j in it which $i < j$ and $S'_i < S'_j$. In this case, we swap S'_i and S'_j will make S' a bigger number. Therefore, any other strings are not the optimal answer, S is the maximum satisfying value.

3. (2%) Please give a $O(n\log(n))$ algorithm to find the maximum satisfying value under **Assumption 2**.
   First, Sort array P in alphabetical descending order. In other words, sort P so that the one has larger alphabetical order will be in front. It takes $O(n\log(n))$. Next, choose the number from the first one to the last one and push them into the answer string, it takes $O(n)$, and we get the answer.

   Time Complexity: $O(n\log(n)) + O(n) = O(n\log(n))$

   Correctness: Suppose the string create by above algorithm is S. For any other strings S' create by P and isn't sorted in alphabetical descending order, we can always find two numbers S'_i and S'_j in it which $i < j$ and in alphabetical order, $S'_i < S'_j$. In this case, we swap S'_i and S'_j will make S' a bigger number. Therefore, any other strings are not the optimal answer, S is the maximum satisfying value.

4. (4%) Please give a $O(n)$ algorithm to find the maximum satisfying value under both **Assumption 1 and 3** (If you get full points on this problem, you will automatically gain full points of (2).)
   Create a array of size 10, $C[10]$. In $C[i]$, we go through P and we can just count the occurrences of each digit and save the occurrences of i it, it takes $O(n)$. Second, we calculate the sum of $R[0] = \Sigma\ C[i], i \in \{0,3,6,9\}$, $R[1] = \Sigma\ C[i], i \in \{1,4,7\}$, $R[2] = \Sigma\ C[i], i \in \{2,5,8\}$, to save the number of those i module 3, it takes $O(1)$.

   Next, split it into five cases.
   If $R[1] + 2 * R[2] \equiv 0\ mod\ 3$, construct the string S from $C[9]$ to $C[0]$. Hence we get the answer S, it takes $O(n)$.

   Else if $R[1] + 2 * R[2] \equiv 1\ mod\ 3$ and $R[1] > 0$, choose the smallest i in $\{1,4,7\}$ and that $C[i] > 0$, then subtract $C[i]$ by 1. Next, do the same thing as the first case and get the answer.

   Else if $R[1] + 2 * R[2] \equiv 1\ mod\ 3$ and $R[1] = 0$, choose the smallest i in $\{2,5,8\}$ that $C[i] > 0$ and subtract $C[i]$ by 1 twice. Next, do the same thing as the first case and get the answer.

   Else if $R[1] + 2 * R[2] \equiv 2\ mod\ 3$ and $R[2] > 0$, choose the smallest i in $\{2,5,8\}$ that $C[i] > 0$ and subtract $C[i]$ by 1. Next, do the same thing as the first case and get the answer.

   Else if $R[1] + 2 * R[2] \equiv 2\ mod\ 3$ and $R[2] = 0$, choose the smallest i in $\{1,4,7\}$ that $C[i] > 0$ and subtract $C[i]$ by 1 twice. Next, do the same thing as the first case and get the answer.

   Time Complexity: $O(n) + O(1) + O(n) = O(n)$

   Correctness: To be a multiple of 3, the sum of digits should also be a multiple of three. Moreover, for those sum of digits that $\equiv 1\ mod\ 3$ or $\equiv 2\ mod\ 3$, remove one number will always make value greater than remove two numbers. Thus, we always delete least number to reach the maximum satisfying value. Then, sort them in descending order, same as (2), which lead us to the correct answer.

5. (2%) Assuming $n = 5$, $k = 5$, given preference value of courses $P_i = [3,\ 4,\ 6,\ 5,\ 0]$, $M_i = [9,\ 0,\ 5,\ 8,\ 3]$, can you find the maximum satisfying value?
98653

6. (8%) Please give a $O(kn^2)$ algorithm to find the maximum satisfying value given P_i and M_i.
Note that there may be other valid solutions that have same or better time complexity. Those who explain correctly will also get full mark.

Divide the problem into two parts.
First, $int*\ MaxNumber(A[\ ]\ ,\ k)$. Given one array, return an array of k digits which is maximum number of length k. To do this, maintain a stack R, go through the array, keep popping back if the new one is greater then the back, and push the new element in to R. Pick the first k elements of R as the return array and return. This takes $O(n)$ time.

Next, $int*\ Merge(A[\ ]\ ,\ B[\ ])$. Given two array, return the maximum number merged array. To do this, merge two arrays like merge sort. When $A[i] = B[j]$, select whether i or j by continuing to compare later numbers follow the arrays. By doing so, the worse case time complexity would be $O(n^2)$.

Finally, we use these two functions to solve the problem. Try different length MaxNumber of the two arrays to find the final answer. Which means to calculate maximum satisfying value equals to
$max\{Merge\ (MaxNumber(P_i, j),\ MaxNumber(M_i, k-j))\}$, where $0 \leq j, k-j \leq n$, it runs k iterations.

Time Complexity: We can build MaxNumber table of $k \in [0,\ n]$ of P_i and M_i in $O(n^2)$ time. Next, for $max\{Merge\ (MaxNumber(P_i, j),\ MaxNumber(M_i, k-j))\}$ step, which takes k iterations to run merge which is $O(n^2)$, is a $O(kn^2)$ step.
Thus, the total time complexity is $O(n^2) + O(kn^2) = O(kn^2)$

Correctness: First, We go through all possibility of choosing $i, k-i$ numbers from P_i and M_i from each department respectively, and choose the biggest one from them. Next, in each case, we choose the largest $i\ or\ k-i$ numbers from each array. Third, we merge them to make the greatest value that is possible. Thus, we have considered all the possibilities and choose the greatest one, which is the optimal answer.