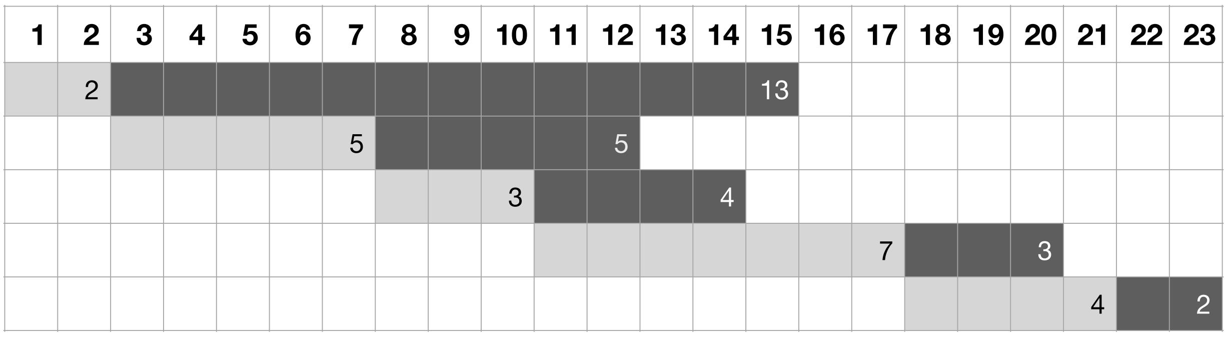
ALGORITHM DESIGN AND ANALYSIS - HOMEWORK 2

B07902143 陳正康

Problem 5

(1) The optimal arrangement yields a minimum time of 23 minute as:
(first row represents time in minutes;
following each row represents a person, composed of two segments,
the former of which is p_i , the latter, e_i)



(2)

a) Sort all people by their eating time e_i from long to short

b) Start from the first person of the sorted array, make Piepie process the next one's order as soon as the current pie finishes its preparation; the minimum time needed is $\sum_{i=1}^{N} p_i + \min\{e_i\}$
there is no time gap between p_i and p_{i+1} $(1 \le i \le N)$

c) **Time Complexity**: sort $O(N \lg N)$ + traversal of array $O(N) = O(N \lg N)$

(3) We'll prove this as a Greedy Algorithm

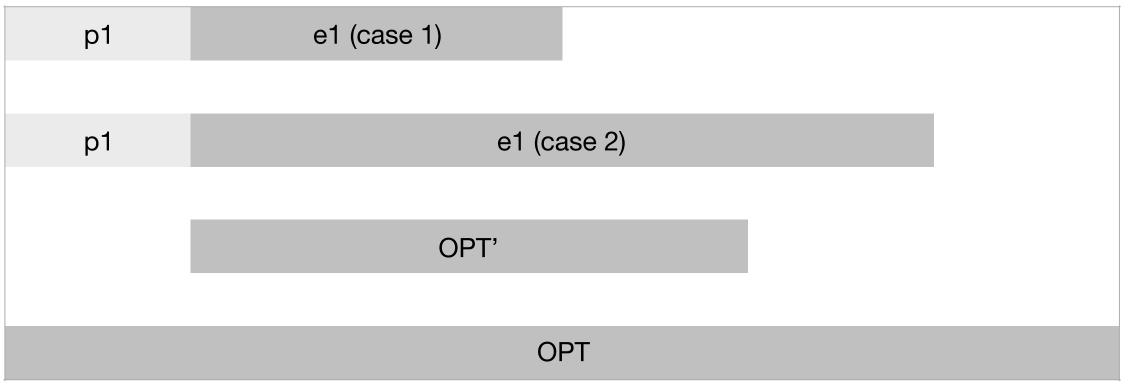
a) With sorting, we assume that $e_1 \ge e_2 \ge \cdots \ge e_N$

b) **Optimal Structure**:
We want to prove: if OPT is an optimal solution for people 1 to N, then OPT$\backslash\{(p_1, e_1)\}$ is an optimal solution for people 2 to N

Define:
$A_i := (p_i, e_i)$: the i-th person in sorted array
$time$(arrangement): the time interval between first one's order starts to be cooked and last one finishes his eating, of an arrangement of people

Suppose OPT is an optimal solution for people 1 to N, and there is OPT', which is an optimal solution for people 2 to N; $OPT' \cup \{A_1\} \neq OPT$
i.e. $time(OPT') < time(OPT \backslash \{A_1\})$

Then there is 2 cases:
Case 1: $e_1 < time(OPT')$
$\implies time(OPT' \cup \{A_1\}) = time(OPT') + p_1$
$< time(OPT \backslash \{A_1\}) + p_1 = time(OPT)$
This contradicts with the assumption.
Case 2: $time(OPT') < e_1 < time(OPT \backslash \{A_1\})$
$\implies time(OPT' \cup \{A_1\}) = p_1 + e_1$
$< p_1 + time(OPT \backslash \{A_1\}) = time(OPT)$
This contradicts with the assumption.



c) **Greedy-choice Property**:
Consider an arrangement with the first 2 person being the i-th and j-th person: $[A_i, A_j, \ldots]$, where $e_i \leq e_j$
to match the greedy choice, we would like to swap the two: $[A_j, A_i, \cdots]$
the ending time of preparation remains the same: $p_i + p_j = p_j + p_i$
We want to prove: the time when the 2 people finish does not become worse:
$p_j + \max\{p_i + e_i, e_j\} \leq p_i + \max\{p_j + e_j, e_i\}$
Since $e_j \geq e_i$, the inequality becomes:
$\implies p_j + \max\{p_i + e_i, e_j\} \leq p_i + p_j + e_j$
$\implies \max\{p_i + e_i, e_j\} \leq p_i + e_j$
$\implies p_i + e_i \leq p_i + e_j$, which is an identity, the greedy choice property is proved

(4) WRONG

Counter example: {(2,6), (3,3), (13,1)} with worker A and B

The following arrangement comply with the strategy but is not optimal:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	░	░ 2	▓	▓	▓	▓	▓	▓ 6								
			░	░	░	░	░	░	░	░	░	░	░	░	░ 13	▓ 1
B	░	░	░ 3	▓	▓	▓ 3										

The optimal solution is:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	░	░	░ 3	▓	▓	▓ 3										
				░	░ 2	▓	▓	▓	▓	▓	▓ 6					
B	░	░	░	░	░	░	░	░	░	░	░	░	░ 13	▓ 1		

In this case, the dispatch strategy between two workers should also be considered.

(5)

a) Run the algorithm stated in (2), producing an arrangement R;
   Time: $O(N \lg N)$
b) Construct an array T, where T[i] is the latest finish time among person i to person N in R; Time: $O(N)$
c) Iterate through each person i in R, calculate the minimum finishing time if i-th person is removed. With the information in T, this operation can be linear; Time: $O(N)$
d) Minimize the answer in c), and get the optimal person to kill.
e) **Correctness**: every possibility is checked
f) **Time Complexity**: $O(N \lg N)$

Problem 6

(1) Let the coordinate of i-th mobile diner be p_i $(1 \le i \le N)$ we've picked the diner $\{1,3\}$, where $p_1 = 1$, $p_3 = 12$
Thus, class 2 to class 5 (on coordinate 7, 11, 12, 17) can go to 3rd diner (on coordinate 12, with delicious rate 5), while class 1 (on coordinate 1) goes to 1st diner (on coordinate 1)

(2) Omitted. Refer to (3).

(3) Greedy Algorithm is used.
The first diner is always chosen. Maintain the largest coordinate that can be covered with chosen diner set.
Then for each class, if it's not covered, then choose another diner to cover it, the position p_j of the new diner j and the class's coordinate x_i have: $x_i + d_j = p_j$
Pseudo code:

```
last_idx := 0
last_coord := x[0] + d[0]
for i from 1 to N
    if last_coord < x[i] and last_idx < M
        last_idx += 1
        last_coord = x[i] + 2 * d[last_idx]
```

Time Complexity: $O(N + M)$
As i increases N times, and last_idx increases less than M times

Definition of Subproblems:
$prob(i)$: The subproblem for class i to class N
We let subproblem to $prob(1)$ be $prob(i)$
Where i is the class index such that the first diner of the optimal solution to $prob(1)$ covers exactly class 1 to class i-1

Proof of Optimal Structure:
Let OPT be an optimal solution diner set for class 1 to class N
Suppose OPT\{1} is not the optimal solution for class i to class N
Then there exists OPT' that is better for class i to class N
Thus OPT'∪{1} will be better than OPT => contradiction
So the subproblem we consider have optimal structure

Proof of Greedy Choice Property:
The greedy choice we made is, in $prob(i)$, place a unused diner j at p_j such that
$x_i + d_j = p_j$
If there exists another optimal solution whose first diner is j and at p_j', where
$x_i + d_j > p_j'$
Then we can always choose $p_j = x_i + d_j > p_j'$
Thus the range of coverage is wider, while minimum number of diner required remains the same.
The case where $x_i + d_j < p_j'$ is ignored as it is not a solution.
The greedy choice property is proved.

(4) Dynamic Programming is used.
Definition of Subproblems:
prob(i, j): a subproblem whose solution will minimize the mobile diners required to fulfill the need for class 1 to class i, with mobile diners indexed 1 to j available.
$dp[i][j]$: the value for prob(i, j)
k: for prob(i, j), k is the largest index of class such that $x_k \le x_i - 2d_j < x_{k+1}$

Initial State and State Transition:
$dp[1][1] = 1$
$dp[i][j] = \min\{dp[i][j-1],\ dp[k][j-1] + 1\}, \quad 1 \le i \le N,\ 2 \le j \le M, \exists k$
$dp[i][j] = \min\{dp[i][j-1],\ 1\}, \quad 1 \le i \le N,\ 2 \le j \le M, \nexists k$

Proof of Optimal structure:
To prove the optimal structure, consider 2 cases.
Case 1: the optimal solution OPT to prob(i, j) does not contain the j-th mobile diner
Goal: to proof that OPT is also an optimal solution to prob$(i, j-1)$
Suppose there is OPT' that is a better solution to prob$(i, j-1)$
Then OPT' must also be a better solution to prob(i, j) than OPT.
Contradiction.
Case 2: the optimal solution OPT to prob(i, j) contains the j-th mobile diner
Goal: to proof that $OPT \backslash \{j\}$ is the optimal solution to prob$(k, j-1)$
Suppose there exists a better solution OPT' to prob$(k, j-1)$
Since $x_k \le x_i - 2d_j$, exactly the j-th mobile dinner should be added to achieve prob(i, j). Thus, $OPT' \cup \{j\}$ is a better solution to prob(i, j). Contradiction.

Time Complexity:

a) Preprocessing: build an array K[N][M], where K[i][j] is defined as:

The class index k such that $x_k \leq x_i - 2d_j < x_{k+1}$

We can deduce that: $x_i - x_{k+1} < 2d_j \leq x_i - x_k$

Also it's trivial to see: $x_k < x_{k+1} \leq x_i$

So as we iterate over each d_j , we can iterate i from N-1 to 0, and find k in $O(N)$ time (similar to sliding window)

Pseudo code: (omit out-of-bounds)

```
for j from 1 to M
    k := N
    for i from N to 1
        while x[i] - x[k] < d[j]
            k -= 1
        K[i][j] := k
```

Time Complexity of a): $O(NM)$

b) fill DP table: aggregately $O(NM)$ states, each transition have 2 possibility ($O(1)$); so $O(NM)$ in total

In conclusion, the algorithm runs in $O(NM)$ time complexity.

Problem 7

(1) Omitted. Refer to (4)

(2) Omitted. Refer to (4)

(3) Omitted. Refer to (4)

(4) **Characterize the Problem**:

A circular route P from p_0 to p_N and back to p_0 is equivalent to the union of two routes, each from p_0 to p_N . The time required is the same. The two routes can only contain points where their vertical height is in ascending order.
Let two routes be U and D. U corresponds to the upward part of the original path, while D corresponds to the downward part.

Subproblem Definition:

$dp(i, j, S, TOP)$: the meaning depends on the value of TOP
$TOP = UP$: A route U from p_0 to p_i , and a route D from p_0 to p_j . U and D have minimum total time
$TOP = DOWN$: A route D from p_0 to p_i , and U from p_0 to p_j . U and D have minimum total time
where bit set S is the colors collected by U (total 7 bits, for 7 colours)

The original problem can be represented as:
$\min\{dp(N, j, \{1,2,3,4,5,6,7\}, UP) \mid 0 \leq j < N\}$

Initial State and State Transition:

$dp(0,0,\varnothing, UP) = 0$
$dp(0,0,\varnothing, DOWN) = 0$
$dp(i, j, S, UP) + f(i, i+1) \rightarrow dp(i+1, j, S \cup c_{i+1}, UP)$
$dp(i, j, S, UP) + f(j, i+1) \rightarrow dp(i+1, i, S, DOWN)$
$dp(i, j, S, DOWN) + f(i, i+1) \rightarrow dp(i+1, j, S, DOWN)$
$dp(i, j, S, DOWN) + f(j, i+1) \rightarrow dp(i+1, i, S \cup c_{i+1}, UP)$
[The syntax $a \rightarrow b$ is defined as: $b := \min(a, b)$]

Proof of Optimal Structure:

We use symbol D{i} and U{i} to represent that the i-th point is on path D or U, respectively.
Case 1: For a optimal solution OPT of $dp(i+1, j, S \cup c_{i+1}, UP)$, p_{i+1} is on route U in OPT
Suppose there is OPT' that is better than OPT\U{i+1} for $dp(i, j, S, UP)$

Then OPT' ∪ U{i+1} must be better than OPT
Contradiction

Case 2: For a optimal solution OPT of $dp(i+1, i, S, DOWN)$, p_{i+1} is on route D in OPT
Suppose there is OPT' that is better than OPT\D{i+1} for $dp(i, j, S, UP)$
Then OPT' ∪ D{i+1} must be better than OPT
Contradiction

Case 3: For a optimal solution OPT of $dp(i+1, j, S, DOWN)$, p_{i+1} is on route D in OPT
Suppose there is OPT' that is better than OPT\D{i+1} for $dp(i, j, S, DOWN)$
Then OPT' ∪ D{i+1} must be better than OPT
Contradiction

Case 4: For a optimal solution OPT of $dp(i+1, i, S \cup c_{i+1}, UP)$, p_{i+1} is on route U in OPT
Suppose there is OPT' that is better than OPT\U{i+1} for $dp(i, j, S, DOWN)$
Then OPT' ∪ U{i+1} must be better than OPT
Contradiction

Time Complexity:
Total $O(N \times N \times 2^7 \times 2) = O(N^2)$ states
Each state transition take $O(1)$ time
$\implies O(N^2)$

Space Complexity:
Each state transition use only i-th and (i+1)-th row
So we can use rolling array to store values of subproblems
i.e. arr[j][S][TOP] and tmp[j][S][TOP] stores value of $dp(i, j, S, TOP)$ and $dp(i+1, j, S, TOP)$ respectively
$\implies O(N \times 2^7 \times 2) = O(N)$