Homework #2

資工二 陳盈如 B05902118

November 8, 2017

Reference

Problem 1

0.0.1 b05902074

0.0.2 b05902038

Problem 2

0.0.3 b05902074

0.0.4 b05902041

Problem 3

0.0.5 助教周忠毅

0.0.6 b05902041

Problem 4

0.0.7 b05902012

Problem 5

0.0.8 b05902120

0.0.9 b05902031

0.0.10 b05902117

Problem 6

0.0.11 助教周忠毅

0.0.12 b05902007

Problem 5 - Dynamic Programming

(1)

Algorithm 1: the maximum sum of non-decreasing subsequence

Input: S_i
Output: max

```
dp[0] = S[0];
for i = 1; i < len; i++ do
    max = -1;
    for j = 0; j < i; j++ do
        if (S[i] ≥ S[j]) ∧ (dp[j] + S[i] > max) then
            max = dp[j] + S[i];
        else if S[i] > max then
            max = S[i];
    dp[i] = max;
for i = 0; i < len; j++ do
    if dp[i] > max then
        max = dp[i];
return max;
```

Use $dp[]$ to store the maximum sum of non-decreaing subsequence before S_i. Go through S and check which $dp[S_j]$ before S_i plus S_i is the biggest. Then record the biggest value in $dp[S_i]$. Seeing that every time we get the value of $dp[]$, we need to go through every j before i. It takes

$$\sum_{i=0}^{n} i = 0 + 1 + 2 + ... + n = \frac{n(n-1)}{2}$$

to get it.

And then we go through each $dp[]$ to find the maximum. In the meantime, the maximum is the answer. Therefore,

$$T(n) = \frac{n(n-1)}{2} + n$$

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

(2)

Algorithm 2: $valid(a, b)$

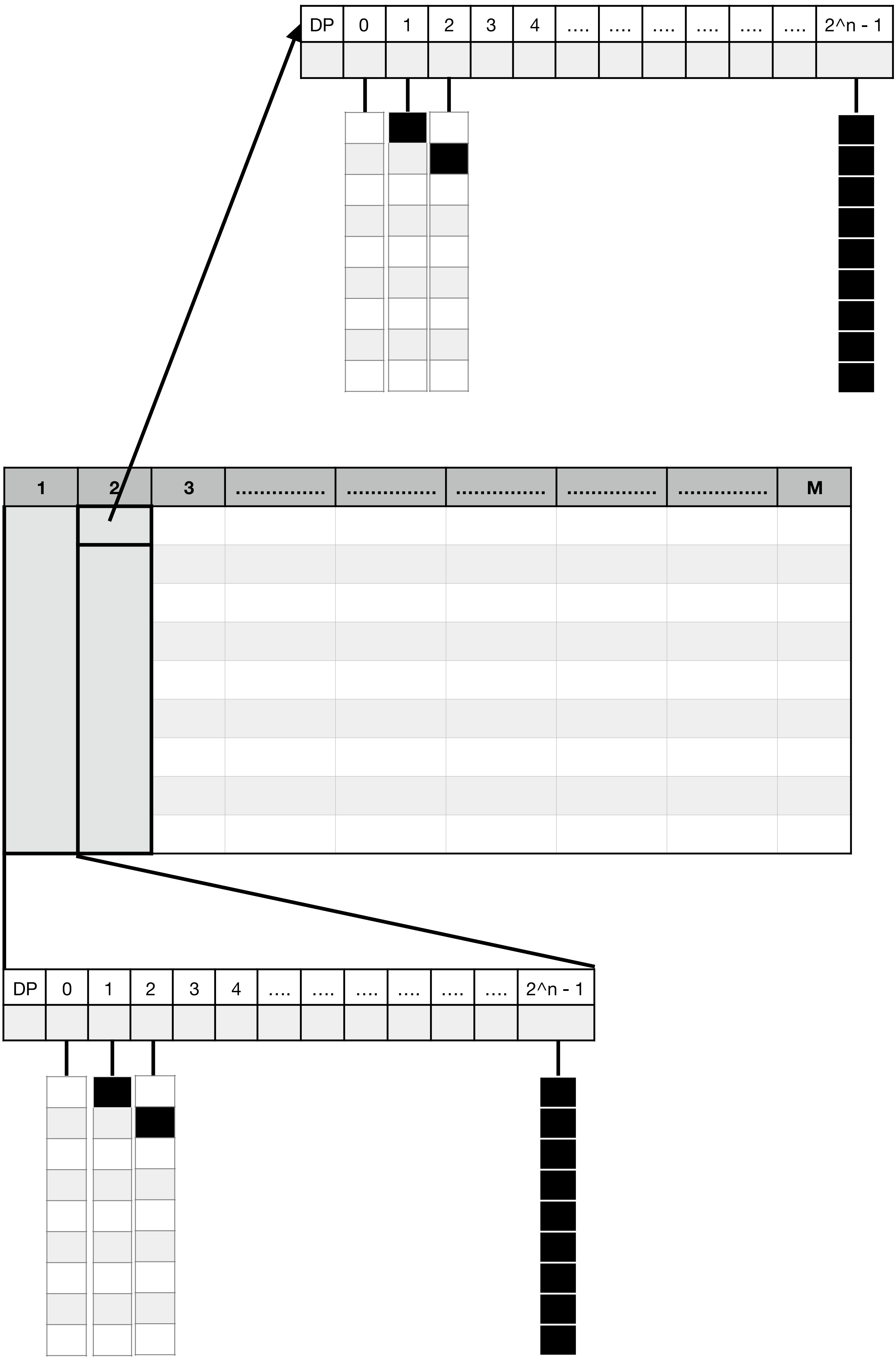
Input: a, b
Output: $TRUE/FALSE$

for $i = 0; i < n; i++$ **do**
    $ok[i] = ((a >> i)\&1) + ((b >> i)\&1);$
for $i = 0; i < n; i++$ **do**
    if $ok[i] > 1$ **then**
        return $FALSE$;
    else if $i = 0\&\&ok[i] == 1\&\&ok[i-1] == 1$ **then**
        return $FALSE$;
return $TRUE$;

Algorithm 3: valid ways to put objects into the $n * m$ grid

Input: n, m
Output: $DP[m+1][0]$

$memset(DP, 0, sizeof(DP))$;
$DP[0][0] = 1$;
for $i = 1; i \leq m+1; i++$ **do**
    for $j = 0; j < 2^n; j++$ **do**
        for $k = 0; k < 2^n; k++$ **do**
            if $valid(j, k)$ **then**
                $DP[i][j] += DP[i-1][k]$;
return $DP[m+1][0]$;


DP 0 1 2 3 4 .... 2^n - 1
1 2 3 .... M
DP 0 1 2 3 4 .... 2^n - 1

(a)

- 利用二進位來表示，0代表不放物品，1代表放置物品
- 輸入的 $n * m$ grid，確保 m 必大於 n，若 n 大於 m 就 $swap(n, m)$
- 在高度為 n 的 grid 裡面，是否放置物品總共有 2^n 種可能，用二進位表示也就是從 0 到 $2^n - 1$ ，舉例：0 二進位表示為 0，也就是不放任何物品、2 二進位表示為 1 0，也就是只在第二格放物品、$2^n - 1$ 二進位表示為1 1 1 1......共 n 個 1，也就是 n 格裡面都放物品。（第四頁的示意圖，黑色區域代表放置物品）
- $DP[][]$ 用來存放此格以前的有效方法數，在判斷下一行的時候（每行有 2^n 種情況），就從 0 開始假設，此行可能的情況為 0 到 $2^n - 1$ ，若此行為 0 時，前一行為 0 到 $2^n - 1$ 時共有幾種是有效的，並將有效的DP值加到此DP值裡（等同於 Algorithm 3 的第七行），若此行為 1 時，前一行為 0 到 $2^n - 1$ 時......以此類推。

(b)

- The third line of Algorithm 3 takes O(m).
- The fourth line of Algorithm 3 takes O(2^n).
- The fifth line of Algorithm 3 takes O(2^n).
- By looking into Algorithm 2, we can know that the *valid* function in sixth line of Algorithm 3 takes O(n).
- In Algorithm 3, each for_loop contains another for_loop or a function. Therefore, the time complexity of this solution is O($2^n \times 2^n \times m \times n$), that is O($4^n \times n \times m$).

Problem 6 - Greedy Algorithm

(1)

The minimum number of shots is 3, and they are 6 to 8, 11 to 12, and 14 to 16 respectively. We only need to choose one y to shoot in each range.

(2)

Algorithm 4: the minimum number of shots

Input: $balloon[][2]$
Output: ans

```
MergeSort(balloon);
ceiling = balloon[0][1];
for i = 1; i < len do
    ans++;
    while balloon[i][0] ≤ ceiling do
        if balloon[i][1] < ceiling then
            ceiling = balloon[i][1];
        i++;
    ceiling = balloon[i][1];
    i++;
    if i ≥ len then
        ans++;
return ans;
```

Use $MergeSort()$ to sort $balloon[][]$ by $balloon[][0]$ from small one to big one. ($balloon[][0]$ represents the lower position of a balloon, and $balloon[][1]$ represents the upper position) Set the $balloon[][1]$ of the remained balloons who has the smallest $balloon[][0]$ as upper bound. Never stop reading $balloon[][]$ and start the following balloon until there is a balloon whose lower position exceed the upper bound. Besides, when we read $balloon[][]$, we need to update the upper bound if there is a balloon who is inside the range, and its upper position is lower than the upper bound. Do all the things above, and count the number of shots meanwhile.

$MergeSort()$'s time complexity is $O(nlogn)$. Then go through the number of balloons. Therefore,

$$T(n) = nlogn + n$$

and the time complexity is $O(nlogn)$.

(3)

以下證明以範例為例：

- Greedy Choice Property:
  - 先 MergeSort 完之後，從通過最低點的氣球開始找，找其他氣球跟他有交集的 y 軸範圍，此 y 軸範圍為 greedy choice (圖*Figure 1*：4 到 5 是其中一個 greedy choice)
  - 利用反證法：假設不選擇 Greedy Choice，有更好的解。
  - 考慮不射 $y = 4, 5$ 的情況，前三顆氣球一定要花兩次以上才能射完，而 Greedy Choice 可以花一次就射完三顆氣球(圖*Figure 2*)
  - 沒有更佳解，出現矛盾！
- Optimal Substructure:
  - Optimal Solution = Greedy Choice + Optimal Solution to subproblem，每做完一個 Greedy Choice，被射掉的氣球就不用管他，剩下的氣球再繼續做 Greedy Choice，所以所有的最佳解的集合就是最後答案的最佳解。

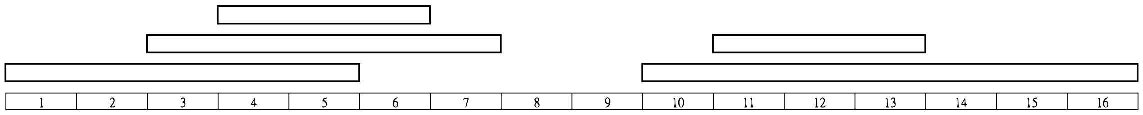


Figure 1: 範例題

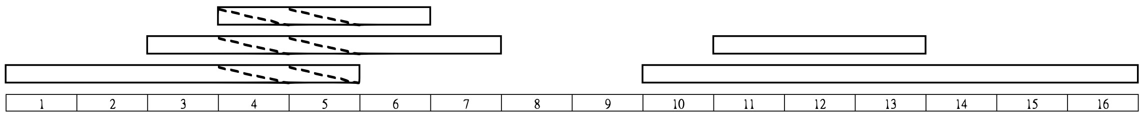


Figure 2: 範例題