DP 3

Lecture by LittleCube

開始之前

- 影片看了嗎？

目錄

- 更多矩陣
- 更多狀態
- 更多的故事

更多矩陣

鐵路鋪設

Problem NHSPC 2021 I 鐵路鋪設

有一個 $2 \times L$ 的棋盤網格，每一格的正中央有一個火車站，市政府想要蓋一些環狀線連接這些鐵路，並且要滿足：

- 每一條路線最多只能有一段長鐵路
- 每條路線都必須是環狀的
- 任兩條路線不會有任何的重疊或交叉

請問有多少種方法可以連接全部這 L 座火車站。$L \leq 10^{10}$

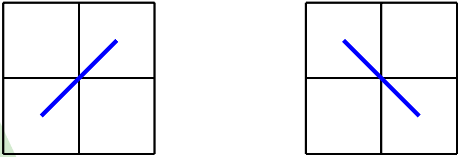


Figure: 兩種長鐵路

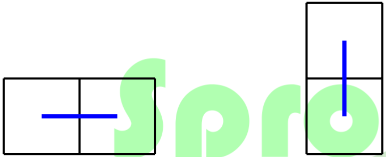
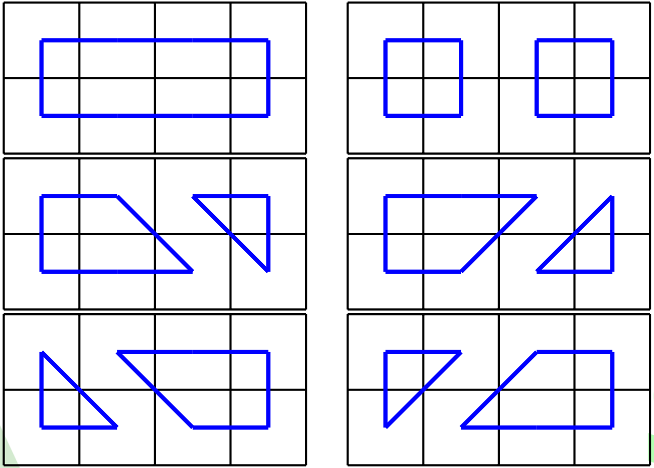
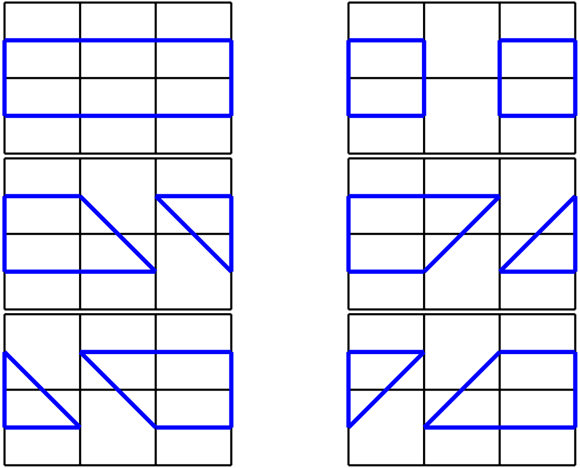


Figure: 兩種短鐵路

鐵路鋪設



鐵路鋪設



鐵路鋪設

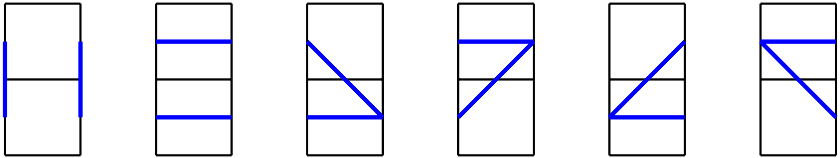


Figure: 六種拼塊，七種狀態

- 第二種需要多紀錄有沒有用過斜的鐵路

鐵路鋪設

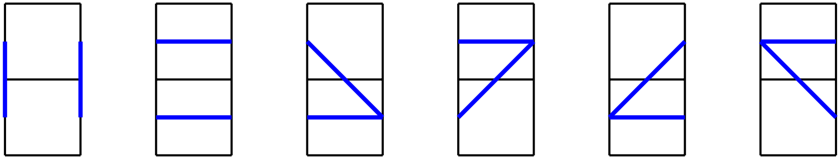


Figure: 六種拼塊，七種狀態

- 第二種需要多紀錄有沒有用過斜的鐵路
- $dp_{i,j}$ 表示拼了 i 塊，現在的結尾狀態是 j

鐵路鋪設

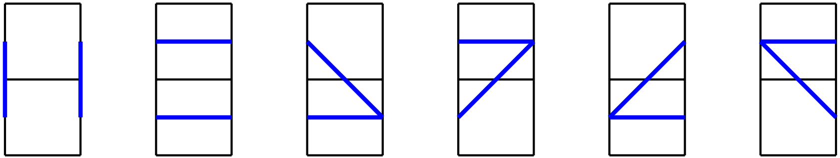


Figure: 六種拼塊，七種狀態

- 第二種需要多紀錄有沒有用過斜的鐵路
- $dp_{i,j}$ 表示拼了 i 塊，現在的結尾狀態是 j
- 起始不能拼右邊兩塊，答案是 dp_{L-1} 結尾在除了第一三四種拚塊的方法數

鐵路鋪設

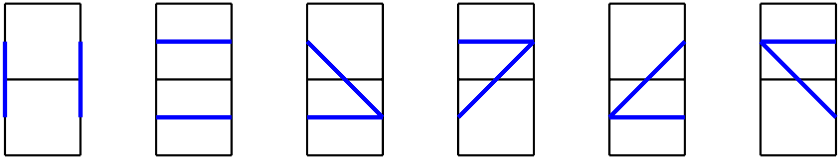


Figure: 六種拼塊，七種狀態

- 第二種需要多紀錄有沒有用過斜的鐵路
- $dp_{i,j}$ 表示拼了 i 塊，現在的結尾狀態是 j
- 起始不能拼右邊兩塊，答案是 dp_{L-1} 結尾在除了第一三四種拚塊的方法數
- $dp_i \to dp_{i+1}$ 都是同樣的轉移，而且都是固定的狀態加起來

鐵路鋪設

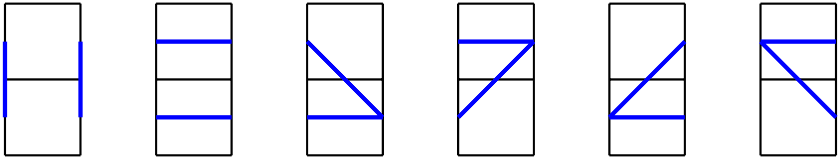


Figure: 六種拼塊，七種狀態

- 第二種需要多紀錄有沒有用過斜的鐵路
- $dp_{i,j}$ 表示拼了 i 塊，現在的結尾狀態是 j
- 起始不能拼右邊兩塊，答案是 dp_{L-1} 結尾在除了第一三四種拚塊的方法數
- $dp_i \to dp_{i+1}$ 都是同樣的轉移，而且都是固定的狀態加起來
- 造出轉移矩陣，時間複雜度變為 $O(7^3 \log L)$

Problem AtCoder DP Contest R Walk

有一張 N 點簡單有向圖，問有多少條長度為 K 的路徑，輸出除以 10^9+7 的餘數。
$N \leq 50, K \leq 10^{18}$

Graph Reachability

- $dp_{i,j,k}$：從 $i \rightsquigarrow j$ 走 k 條邊的方法數

Graph Reachability

- $dp_{i,j,k}$：從 $i \rightsquigarrow j$ 走 k 條邊的方法數
- $dp_{i,j,k+1} = \sum dp_{i,t,k} \cdot G_{t,j}$

Graph Reachability

- $dp_{i,j,k}$：從 $i \rightsquigarrow j$ 走 k 條邊的方法數
- $dp_{i,j,k+1} = \sum dp_{i,t,k} \cdot G_{t,j}$
- $dp_{i,j,2k} = \sum dp_{i,t,k} \cdot dp_{t,j,k}$

Graph Reachability

- $dp_{i,j,k}$：從 $i \rightsquigarrow j$ 走 k 條邊的方法數
- $dp_{i,j,k+1} = \sum dp_{i,t,k} \cdot G_{t,j}$
- $dp_{i,j,2k} = \sum dp_{i,t,k} \cdot dp_{t,j,k}$
- 看似很像矩陣快速冪，實際上我們在算的就是 Adjacency Matrix G 的 K 次方 G^K！

Graph Reachability

Problem 經典問題

有一張 N 點簡單帶權有向圖，問從 u 到 v 走剛好長度為 K 的最短路徑有多長。
$N \leq 300, K \leq 10^9$

Graph Reachability

- $dp_{i,j,k}$：從 $i \rightsquigarrow j$ 走 k 條邊的最短路徑

Graph Reachability

- $dp_{i,j,k}$：從 $i \rightsquigarrow j$ 走 k 條邊的最短路徑
- $dp_{i,j,k+1} = \min_t dp_{i,t,k} + G_{t,j}$

Graph Reachability

- $dp_{i,j,k}$：從 $i \rightsquigarrow j$ 走 k 條邊的最短路徑
- $dp_{i,j,k+1} = \min_t dp_{i,t,k} + G_{t,j}$
- $dp_{i,j,2k} = \min_t dp_{i,t,k} + dp_{t,j,k}$

Graph Reachability

- $dp_{i,j,k}$：從 $i \rightsquigarrow j$ 走 k 條邊的最短路徑
- $dp_{i,j,k+1} = \min_t dp_{i,t,k} + G_{t,j}$
- $dp_{i,j,2k} = \min_t dp_{i,t,k} + dp_{t,j,k}$
- 也是矩陣快速冪的樣子，只是把 $(+, \times)$ 換成 $(\min, +)$

Graph Reachability

- $dp_{i,j,k}$：從 $i \rightsquigarrow j$ 走 k 條邊的最短路徑
- $dp_{i,j,k+1} = \min_t dp_{i,t,k} + G_{t,j}$
- $dp_{i,j,2k} = \min_t dp_{i,t,k} + dp_{t,j,k}$
- 也是矩陣快速冪的樣子，只是把 $(+, \times)$ 換成 $(\min, +)$
- Useful Insight: 矩陣快速冪只不過是一種倍增，事實上也有很多 dp 跟倍增有關係（但不一定可以被寫成矩陣）

Mr. Kitayuta's Gift

Problem Codeforces 506E Mr. Kitayuta's Gift

你有一個字串 s，你要插入 n 個字母讓這個字串變成回文，有多少種**結束的樣子**是造得出來的？ 輸出這個種類數除以 10007 的餘數。

$|s| \leq 200, n \leq 10^9$

Mr. Kitayuta's Gift

Problem Codeforces 506E Mr. Kitayuta's Gift

你有一個字串 s，你要插入 n 個字母讓這個字串變成回文，有多少種**結束的樣子**是造得出來的？ 輸出這個種類數除以 10007 的餘數。
$|s| \leq 200, n \leq 10^9$

Left as an excercise. (It's tough.)

更多狀態

CSES Counting Tilings

Problem Counting Tilings

有多少種方法用 1×2 的骨牌（可以旋轉）不重疊的拼滿 $n \times m$ 的棋盤格？

$n \leq 10, m \leq 1000$

CSES Counting Tilings

方法 1：一排一排拼

CSES Counting Tilings

方法 1：一排一排拼

- 這一排有空位的一定要拼一塊上去

CSES Counting Tilings

方法 1：一排一排拼

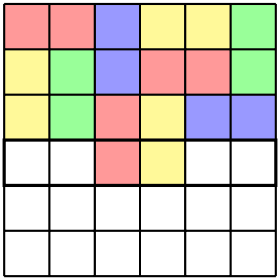
- 這一排有空位的一定要拼一塊上去
- 連續兩個空位可以拼橫的，一個空位拼直的會往下一排凸

CSES Counting Tilings

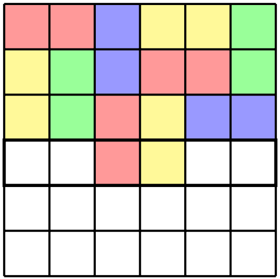
方法 1：一排一排拼

- 這一排有空位的一定要拼一塊上去
- 連續兩個空位可以拼橫的，一個空位拼直的會往下一排凸
- 需要知道哪些資訊？

CSES Counting Tilings

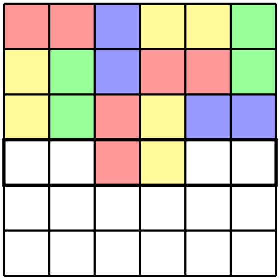


CSES Counting Tilings



- 上一排至多只會凸到這一排的一些格子，嘗試直接紀錄長相

CSES Counting Tilings



- 上一排至多只會凸到這一排的一些格子，嘗試直接紀錄長相
- $dp_{i,mask}$ 代表第 i 列的長相是 $mask$（在這個例子中就是 001100_2）的拼法數量

CSES Counting Tilings

- 嘗試預處理哪些 $mask \to mask'$ 是合法的
  - 直接枚舉：$O(n4^n)$

CSES Counting Tilings

- 嘗試預處理哪些 $mask \to mask'$ 是合法的
  - 直接枚舉：$O(n4^n)$
- 轉移
  - 直接枚舉：$O(m4^n)$

CSES Counting Tilings

- 嘗試預處理哪些 $mask \to mask'$ 是合法的
  - 直接枚舉：$O(n4^n)$
  - 只枚舉可能的狀況：每個已經被上一排凸下來的就不用放了，所以每一格只有被凸過來、放直的跟放橫的，其實只有 $O(3^n)$ 個轉移是合法的
- 轉移
  - 直接枚舉：$O(m4^n)$

CSES Counting Tilings

- 嘗試預處理哪些 $mask \to mask'$ 是合法的
  - 直接枚舉：$O(n4^n)$
  - 只枚舉可能的狀況：每個已經被上一排凸下來的就不用放了，所以每一格只有被凸過來、放直的跟放橫的，其實只有 $O(3^n)$ 個轉移是合法的
- 轉移
  - 直接枚舉：$O(m4^n)$
  - 只轉移合法狀態：$O(m3^n)$，複雜度還算能通過

CSES Counting Tilings

- 嘗試預處理哪些 $mask \to mask'$ 是合法的
  - 直接枚舉：$O(n4^n)$
  - 只枚舉可能的狀況：每個已經被上一排凸下來的就不用放了，所以每一格只有被凸過來、放直的跟放橫的，其實只有 $O(3^n)$ 個轉移是合法的
  - 事實上可能更少
- 轉移
  - 直接枚舉：$O(m4^n)$
  - 只轉移合法狀態：$O(m3^n)$，複雜度還算能通過

CSES Counting Tilings

方法 2：一格一格拼

CSES Counting Tilings

方法 2：一格一格拼

- 這一格是空位的一定要拼一塊上去

CSES Counting Tilings

方法 2：一格一格拼

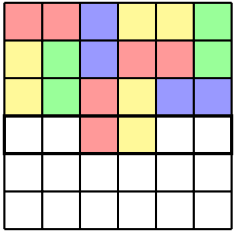
- 這一格是空位的一定要拼一塊上去
- 右邊是空位可以拼橫的，一個空位拼直的會往下一排凸

CSES Counting Tilings

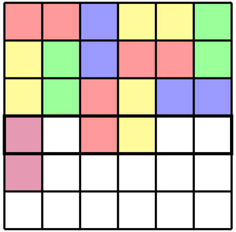
方法 2：一格一格拼

- 這一格是空位的一定要拼一塊上去
- 右邊是空位可以拼橫的，一個空位拼直的會往下一排凸
- 需要知道／紀錄哪些資訊？

CSES Counting Tilings

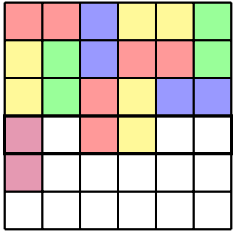


CSES Counting Tilings



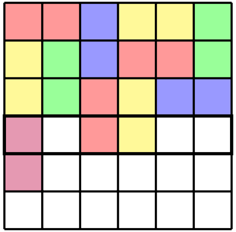
- 拼上紫色的拼塊之後，需要多紀錄他凸出了一塊

CSES Counting Tilings



- 拼上紫色的拼塊之後，需要多紀錄他凸出了一塊
- 幸好紫色的上半部不再重要，可以用新的資訊覆蓋掉！

CSES Counting Tilings



- 拼上紫色的拼塊之後，需要多紀錄他凸出了一塊
- 幸好紫色的上半部不再重要，可以用新的資訊覆蓋掉！
- $dp_{i,j,mask}$ 代表拚到第 i 列第 j 塊時的長相是 $mask$（在這個例子中就是 $001100_2 \to 101100_2$）的拼法數量

CSES Counting Tilings

- 轉移

CSES Counting Tilings

- 轉移
  - 三個 case：不用放、放直的、放橫的

CSES Counting Tilings

- 轉移
  - 三個 case：不用放、放直的、放橫的
  - 時間複雜度只有 $O(nm2^n)$，比剛剛快多了

CSES Counting Tilings

- 方法 1：一排一排拼 $O(m3^n)$
  - 狀態少一點點，轉移多很多
- 方法 2：一格一格拼 $O(nm2^n)$
  - 狀態多一點點，轉移少很多

CSES Counting Tilings

- 方法 1：一排一排拼 $O(m3^n)$
  - 狀態少一點點，轉移多很多
- 方法 2：一格一格拼 $O(nm2^n)$
  - 狀態多一點點，轉移少很多
- 如果 n 小一點點（$n \leq 8$）但是 m 非常大（$m \leq 10^9$），要怎麼做？

CSES Counting Tilings

- 方法 1：一排一排拼 $O(m3^n)$
  - 狀態少一點點，轉移多很多
- 方法 2：一格一格拼 $O(nm2^n)$
  - 狀態多一點點，轉移少很多
- 如果 n 小一點點（$n \leq 8$）但是 m 非常大（$m \leq 10^9$），要怎麼做？
  - 用方法 1 可以套矩陣做到 $O(2^{3n} \log m) = O(8^n \log m)$

Sprout

更多的故事

LCS - Extra

Problem Longest Common Subsequence

你有兩個陣列 A, B，長度分別是 n, m，求他們的最長共同子序列（可以不連續的那種）
時間複雜度 $O(nm)$，

LCS - Extra

Problem Longest Common Subsequence

你有兩個陣列 A, B，長度分別是 n, m，求他們的最長共同子序列（可以不連續的那種）

時間複雜度 $O(nm)$，空間複雜度 $O(n + m)$，

LCS - Extra

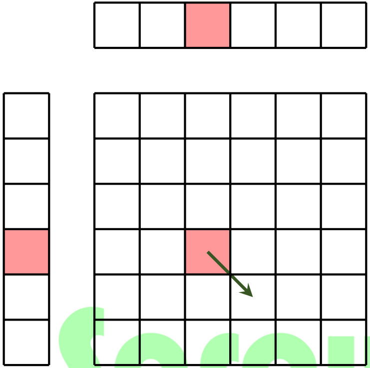
Problem Longest Common Subsequence

你有兩個陣列 A, B，長度分別是 n, m，求他們的最長共同子序列（可以不連續的那種）

時間複雜度 $O(nm)$，空間複雜度 $O(n+m)$，**要回溯解**。

LCS - Extra

如果有一天我們知道（某一個）最佳解當中，A_i 要對應到 B_j，這能告訴我們什麼資訊？

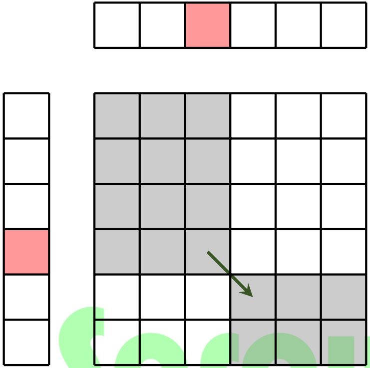


LCS - Extra

因為這個轉移必定會保證一個最佳解，而這個最佳解可以拆成

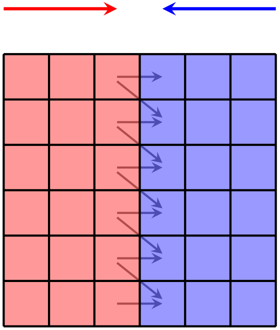
- $(0,0) \rightsquigarrow (i,j)$
- $(i,j) \to (i+1,j+1)$
- $(i+1,j+1) \rightsquigarrow (n,m)$

三個部份的最佳解，所以我們只需要在乎這兩塊灰色的部分就好了！



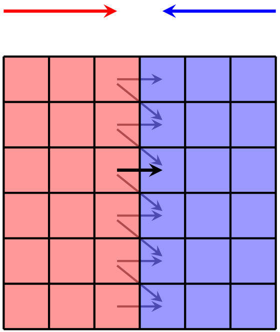
LCS - Extra

我們希望得知的這個轉移越中間越好，所以乾脆切成兩半解！



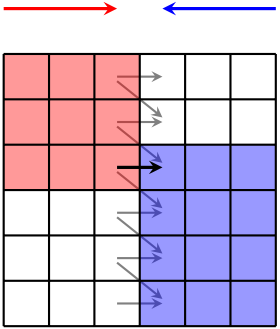
LCS - Extra

一旦得知中間兩排的 dp 值，就能直接 $O(n)$ 找出在中間最佳的其中一個轉移



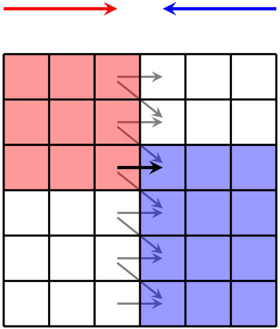
LCS - Extra

一旦得知中間兩排的 dp 值，就能直接 $O(n)$ 找出在中間最佳的其中一個轉移



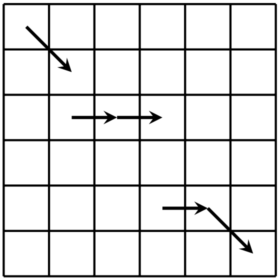
LCS - Extra

接下來只需要遞迴求解就好。



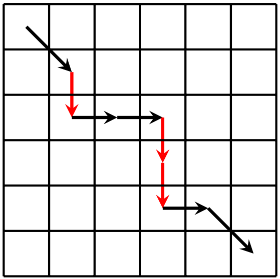
LCS - Extra

假設最終的解答是，



LCS - Extra

就可以回溯出這樣的解答。



LCS - Extra

時間複雜度：

$$T(nm) = T\left(\frac{m}{2} \cdot n_1\right) + T\left(\frac{m}{2} \cdot n_2\right) \quad (n_1 + n_2 = n)$$

$$T(nm) = O(nm)$$

空間複雜度：
每次計算只需要花 $O(nm)$，空間只要 $O(n)$ 因為兩側都可以滾動，留下來的資訊是 $O(1)$，所以遞迴總空間也不過 $O(n+m)$。

Undecodable Codes

Problem ACM-ICPC World Finals 2002 Undecodable Codes

要把文字儲存在電腦裡面需要把字元做編碼，也就是把每個字元對應到一個二進位的序列，這樣編碼的結果就會是各個字元轉換成編碼串接起來的結果。一個好的編碼除了長度要盡量短，也要保證一個編碼的解碼方法是唯一的，否則就會發生問題。現在你有一些有問題的編碼組合，也就是說在這些組合中，有編碼後的結果會有兩種以上的解讀，請你找出長度最短而字典序最小而有問題的編碼。

- 字元集合 $m \leq 20$
- 編碼的長度都不超過 20

Undecodable Codes

範例：

a	010
c	01
j	001
l	10
p	0
s	1
v	101

- pascal $=$ 001010101010 $=$ java
- c $=$ 01 $=$ ps

Undecodable Codes

- 如何定狀態？

Undecodable Codes

- pascal
- 001010101010
- 001010101010
- java

Undecodable Codes

- p
- 0
- 001
- j

Undecodable Codes

- pa
- 0010
- 001
- j

Undecodable Codes

- pa
- 0010
- 001010
- ja

Undecodable Codes

- pas
- 00101
- 001010
- ja

Undecodable Codes

- pasc
- 0010101
- 001010
- ja

Undecodable Codes

- pasc
- 0010101
- 001010101
- jav

Undecodable Codes

- pasca
- 0010101010
- 001010101
- jav

Undecodable Codes

- pasca
- 0010101010
- 001010101010
- java

Undecodable Codes

- pascal
- 001010101010
- 001010101010
- java

Undecodable Codes

- dp_s：兩個不同的編碼前面相同，比較長的多的字串是 s，這樣狀況下前綴最小的長度

Undecodable Codes

- dp_s：兩個不同的編碼前面相同，比較長的多的字串是 s，這樣狀況下前綴最小的長度
- 如果每次我們增加都是增加小的部分的編碼，s 的長度不過 19

Undecodable Codes

- dp_s：兩個不同的編碼前面相同，比較長的多的字串是 s，這樣狀況下前綴最小的長度
- 如果每次我們增加都是增加小的部分的編碼，s 的長度不過 19
- 至多只有 2^{19} 個狀態

Undecodable Codes

- dp_s：兩個不同的編碼前面相同，比較長的多的字串是 s，這樣狀況下前綴最小的長度
- 如果每次我們增加都是增加小的部分的編碼，s 的長度不過 19
- 至多只有 2^{19} 個狀態
- 初始狀態是取兩個前綴一樣的編碼重疊
- 答案是 $dp_{\varnothing}$

Undecodable Codes

- 轉移的順序不知道要怎麼處理
  - 可以長變短

Undecodable Codes

- 轉移的順序不知道要怎麼處理
  - 可以長變短
  - 甚至可以有環

Undecodable Codes

- 轉移的順序不知道要怎麼處理
  - 可以長變短
  - 甚至可以有環
- 轉移的形式都是 $dp_i = \min_j dp_j + f(j,i)$

Undecodable Codes

- 轉移的順序不知道要怎麼處理
  - 可以長變短
  - 甚至可以有環
- 轉移的形式都是 $dp_i = \min_j dp_j + f(j, i)$
- 換句話說，$dp_j + f(j, i) \geq dp_i$

Undecodable Codes

- 轉移的順序不知道要怎麼處理
  - 可以長變短
  - 甚至可以有環
- 轉移的形式都是 $dp_i = \min_j dp_j + f(j,i)$
- 換句話說，$dp_j + f(j,i) \geq dp_i$
- 看起來像什麼？

Undecodable Codes

- 對式子敏感的人：$dp_j + f(j,i) \geq dp_i$ 就是最短路！
- 比較直覺的想法：

Undecodable Codes

- 對式子敏感的人：$dp_j + f(j,i) \geq dp_i$ 就是最短路！
- 比較直覺的想法：
- 每次把最小還沒確定的 dp 值取出來，這個值不可能變更小，因為轉移都是正權

Undecodable Codes

- 對式子敏感的人：$dp_j + f(j,i) \geq dp_i$ 就是最短路！
- 比較直覺的想法：
- 每次把最小還沒確定的 dp 值取出來，這個值不可能變更小，因為轉移都是正權
- 因此，只需要把最小的 dp 值取出來確定，並且處理與他有關的所有轉移

Undecodable Codes

- 對式子敏感的人：$dp_j + f(j,i) \geq dp_i$ 就是最短路！
- 比較直覺的想法：
- 每次把最小還沒確定的 dp 值取出來，這個值不可能變更小，因為轉移都是正權
- 因此，只需要把最小的 dp 值取出來確定，並且處理與他有關的所有轉移
- 整個作法就是一個 Dijkstra 演算法，換句話說我們就是在求最短從基礎狀態到終點的距離！

Undecodable Codes

- 對式子敏感的人：$dp_j + f(j,i) \geq dp_i$ 就是最短路！
- 比較直覺的想法：
- 每次把最小還沒確定的 dp 值取出來，這個值不可能變更小，因為轉移都是正權
- 因此，只需要把最小的 dp 值取出來確定，並且處理與他有關的所有轉移
- 整個作法就是一個 Dijkstra 演算法，換句話說我們就是在求最短從基礎狀態到終點的距離！
- 跟最短路相同，回溯解就是找出最短路的路徑

Undecodable Codes

- 對式子敏感的人：$dp_j + f(j,i) \geq dp_i$ 就是最短路！
- 比較直覺的想法：
- 每次把最小還沒確定的 dp 值取出來，這個值不可能變更小，因為轉移都是正權
- 因此，只需要把最小的 dp 值取出來確定，並且處理與他有關的所有轉移
- 整個作法就是一個 Dijkstra 演算法，換句話說我們就是在求最短從基礎狀態到終點的距離！
- 跟最短路相同，回溯解就是找出最短路的路徑
- 時間複雜度是 $O(20m^2 + 2^{20} m \log 2^{20})$

Undecodable Codes

- 對式子敏感的人：$dp_j + f(j,i) \geq dp_i$ 就是最短路！
- 比較直覺的想法：
- 每次把最小還沒確定的 dp 值取出來，這個值不可能變更小，因為轉移都是正權
- 因此，只需要把最小的 dp 值取出來確定，並且處理與他有關的所有轉移
- 整個作法就是一個 Dijkstra 演算法，換句話說我們就是在求最短從基礎狀態到終點的距離！
- 跟最短路相同，回溯解就是找出最短路的路徑
- 時間複雜度是 $O(20m^2 + 2^{20}m\log 2^{20})$
- Useful insight: DP 轉移就是一張圖，有時候是 DAG，有時候是一般圖

The Great Julya Calendar

Problem Codeforces 331C3 The Great Julya Calendar

你有一個神奇數字 n，每次你可以把 n 減掉他其中的一個位數。要讓 n 變成 0 需要多少次？

- $n \leq 10^{18}$

The Great Julya Calendar

思考簡單版本：$n \leq 10^6$

The Great Julya Calendar

思考簡單版本：$n \leq 10^6$

- dp_n 表示最少次數，轉移 $O(\log_{10} n)$

The Great Julya Calendar

思考簡單版本：$n \leq 10^6$

- dp_n 表示最少次數，轉移 $O(\log_{10} n)$
- 仔細一想真的需要 DP 嗎？

The Great Julya Calendar

思考簡單版本：$n \leq 10^6$

- dp_n 表示最少次數，轉移 $O(\log_{10} n)$
- 仔細一想真的需要 DP 嗎？
- Greedy：每次減最大的數字？

The Great Julya Calendar

Greedy 每次減最大的數字是對的。

Proof

直接對題目進行數學歸納法，我們假設 a_n 是對於 n 最少需要的操作次數，想要證明 $\langle a_n \rangle$ 非嚴格遞增。

基底：$a_0 = 0, a_1 = a_2 = \cdots = a_9 = 1$。

一次對 10 個數字歸納，假設 $\langle a_n \rangle$ 對於 $\lfloor \frac{n}{10} \rfloor < k$ 是遞增的，而且對於十位數以上一樣的數字，他們的 a_n 相差不超過 1。

考慮 $10k, 10k+1, \ldots, 10k+9$。假設 d 是 k 的最大位數，那

- 最後一位為 0：$a_{10k} = a_{10k-d} + 1 \geq a_{10k-1}$
- 最後一位小於 d：$a_{10k+i} = a_{10k+i-d} + 1 \geq a_{10k+i-d-1} + 1 = a_{10k+i-1}$
- 最後一位大於等於 d：$a_{10k+i} = a_{10k} + 1 \geq a_{10k-1} + 1 = a_{10k+i-1}$

根據數學歸納法得證。 $\square$

The Great Julya Calendar

- 一個位數會失效如果下面的人都被扣光然後借位了

The Great Julya Calendar

- 一個位數會失效如果下面的人都被扣光然後借位了
- `solve`(n, d)：如果在 n 最高位前面的位數最大是 d，扣到不是正的需要幾步（以及扣成什麼樣子）

The Great Julya Calendar

- 一個位數會失效如果下面的人都被扣光然後借位了
- `solve`(n, d)：如果在 n 最高位前面的位數最大是 d，扣到不是正的需要幾步（以及扣成什麼樣子）
- $n < 10$：扣一次必定非正
- n 去掉最高位都是 0：先扣一次，然後遞迴算
- n 去掉最高位非 0：答案是先把去掉最高位扣光，加上剩下的扣光

The Great Julya Calendar

- 一個位數會失效如果下面的人都被扣光然後借位了
- solve(n, d)：如果在 n 最高位前面的位數最大是 d，扣到不是正的需要幾步（以及扣成什麼樣子）
- $n < 10$：扣一次必定非正
- n 去掉最高位都是 0：先扣一次，然後遞迴算
- n 去掉最高位非 0：答案是先把去掉最高位扣光，加上剩下的扣光
- DP 精神：算過的不要再算，複雜度是多少？

有三類數字會被算到：

有三類數字會被算到：

- n 只有最高位非 0：$10 \times 10 \times \log_{10} n$ 個狀態
- n 是最高位非 0 扣掉 $0 \sim 9$：$10 \times 10 \times 10 \times \log_{10} n$ 個狀態
- n 是原本數字的後綴：$10 \times \log_{10} n$ 個狀態

The Great Julya Calendar

有三類數字會被算到：

- n 只有最高位非 0：$10 \times 10 \times \log_{10} n$ 個狀態
- n 是最高位非 0 扣掉 $0 \sim 9$：$10 \times 10 \times 10 \times \log_{10} n$ 個狀態
- n 是原本數字的後綴：$10 \times \log_{10} n$ 個狀態

總共不過 $O(1000 \log n) = O(b^3 \log n)$ 個狀態，直接用 `map` 存就足夠了！

Ciel and Gondolas

Problem Ciel and Gondolas

有 n 個人排隊要搭船，一般船來的時候你可以叫隊伍前面的一些人上船，也就是在隊伍中從頭開始連續的人，不過所有人都要搭到船，所以在全部 k 台船來之後所有人都要在某台船上。
不過，跟陌生人坐船難免會感到有點尷尬，所以不僅所有人都要搭到船，所有同船任意兩個人 i, j 的**陌生程度** $u_{i,j}$ 加起來要盡量小。

- $n \leq 4000, k \leq \min(n, 800), 0 \leq u_{i,j} \leq 9$

Ciel and Gondolas

- $dp_{i,j}$ 表示前 i 個人已經分了 j 組

Ciel and Gondolas

- $dp_{i,j}$ 表示前 i 個人已經分了 j 組
- 時間複雜度是 $O(n^2k)$

Ciel and Gondolas

- $dp_{i,j}$ 表示前 i 個人已經分了 j 組
- 時間複雜度是 $O(n^2k)$
- ?????

Ciel and Gondolas

- 直覺上會覺得船分得越均勻越好，但未必是剛好均勻的

Ciel and Gondolas

- 直覺上會覺得船分得越均勻越好，但未必是剛好均勻的
- 考慮下一個人要上船的時候，如果現在的船越大，上去增加的陌生程度會越多

Ciel and Gondolas

- 直覺上會覺得船分得越均勻越好，但未必是剛好均勻的
- 考慮下一個人要上船的時候，如果現在的船越大，上去增加的陌生程度會越多
- 讓 $f(i, j)$ 表示 i 到 j 都同船的時候整艘船任意兩人的陌生度總和，

$$f(i, j+1) - f(i, j) \leq f(i-1, j+1) - f(i-1, j)$$

或是說

$$f(r, j+1) - f(r, j) \leq f(l, j+1) - f(l, j) \quad (l < r)$$

Ciel and Gondolas

- 直覺上會覺得船分得越均勻越好，但未必是剛好均勻的
- 考慮下一個人要上船的時候，如果現在的船越大，上去增加的陌生程度會越多
- 讓 $f(i,j)$ 表示 i 到 j 都同船的時候整艘船任意兩人的陌生度總和，

$$f(i,j+1)-f(i,j)\le f(i-1,j+1)-f(i-1,j)$$

或是說

$$f(r,j+1)-f(r,j)\le f(l,j+1)-f(l,j)\quad (l<r)$$

- 假設說 $dp_{j,k}$ 是從 $dpi-1,k-1+f(i,j)$ 來的，這代表說

$$dp_{i-1,k-1}+f(i,j)\le dp_{l-1,k-1}+f(l,j)\quad (l<i)$$

Ciel and Gondolas

- 直覺上會覺得船分得越均勻越好，但未必是剛好均勻的
- 考慮下一個人要上船的時候，如果現在的船越大，上去增加的陌生程度會越多
- 讓 $f(i,j)$ 表示 i 到 j 都同船的時候整艘船任意兩人的陌生度總和，

  $$f(i,j+1)-f(i,j)\le f(i-1,j+1)-f(i-1,j)$$

  或是說

  $$f(r,j+1)-f(r,j)\le f(l,j+1)-f(l,j)\quad (l<r)$$

- 假設說 $dp_{j,k}$ 是從 $dpi-1,k-1+f(i,j)$ 來的，這代表說

  $$dp_{i-1,k-1}+f(i,j)\le dp_{l-1,k-1}+f(l,j)\quad (l<i)$$

  $$dp_{i-1,k-1}+f(i,j+1)\le dp_{l-1,k-1}+f(l,j+1)\quad (l<i)$$

Ciel and Gondolas

- 直覺上會覺得船分得越均勻越好，但未必是剛好均勻的
- 考慮下一個人要上船的時候，如果現在的船越大，上去增加的陌生程度會越多
- 讓 $f(i,j)$ 表示 i 到 j 都同船的時候整艘船任意兩人的陌生度總和，

$$f(i,j+1)-f(i,j)\leq f(i-1,j+1)-f(i-1,j)$$

  或是說

$$f(r,j+1)-f(r,j)\leq f(l,j+1)-f(l,j) \quad (l<r)$$

- 假設說 $dp_{j,k}$ 是從 $dpi-1,k-1+f(i,j)$ 來的，這代表說

$$dp_{i-1,k-1}+f(i,j)\leq dp_{l-1,k-1}+f(l,j) \quad (l<i)$$

$$dp_{i-1,k-1}+f(i,j+1)\leq dp_{l-1,k-1}+f(l,j+1) \quad (l<i)$$

- 也就是說，$dp_{j+1,k}$ 不可能從 $dp_{i-1,k-1}$ 的左邊來。換句話說位置與最好的轉移來源會一起向右遞增！

Ciel and Gondolas

- 嘗試：每次從上一個最好的點開始算，只要遞增就停下來

Ciel and Gondolas

- 嘗試：每次從上一個最好的點開始算，只要遞增就停下來
- 錯誤作法 — 我們只有保證最好的點是往右增加，沒有保證任何 dp 值長的樣子

Ciel and Gondolas

- 嘗試：每次從上一個最好的點開始算，只要遞增就停下來
- 錯誤作法 — 我們只有保證最好的點是往右增加，沒有保證任何 dp 值長的樣子
- 如果我們知道某個位置的最佳解，就可以把剩下的點切成兩半！

Ciel and Gondolas

- solve(l, r, l', r') 表示我們要處理 l, r 的所有 $dp_{i,k}$ 計算，而且我們知道的額外資訊是這些的最佳來源只會在 $dp_{l',k}$ 到 $dp_{r',k}$

Ciel and Gondolas

- solve(l, r, l', r') 表示我們要處理 l, r 的所有 $dp_{i,k}$ 計算，而且我們知道的額外資訊是這些的最佳來源只會在 $dp_{l',k}$ 到 $dp_{r',k}$
- 一開始呼叫 solve$(1, n, 1, n)$

Ciel and Gondolas

- solve(l, r, l', r') 表示我們要處理 l, r 的所有 $dp_{i,k}$ 計算，而且我們知道的額外資訊是這些的最佳來源只會在 $dp_{l',k}$ 到 $dp_{r',k}$
- 一開始呼叫 solve$(1, n, 1, n)$
- 選某個點 m，然後暴力嘗試 m 對於 l' 到 r'，算出最好的位置

Ciel and Gondolas

- solve(l, r, l', r') 表示我們要處理 l, r 的所有 $dp_{i,k}$ 計算，而且我們知道的額外資訊是這些的最佳來源只會在 $dp_{l',k}$ 到 $dp_{r',k}$
- 一開始呼叫 solve$(1, n, 1, n)$
- 選某個點 m，然後暴力嘗試 m 對於 l' 到 r'，算出最好的位置
- 假設最好的位置是 m'，直接呼叫 solve$(l, m-1, l', m')$ 跟 solve$(m+1, r, m', r')$

Ciel and Gondolas

- solve(l, r, l', r') 表示我們要處理 l, r 的所有 $dp_{i,k}$ 計算，而且我們知道的額外資訊是這些的最佳來源只會在 $dp_{l',k}$ 到 $dp_{r',k}$
- 一開始呼叫 solve$(1, n, 1, n)$
- 選某個點 m，然後暴力嘗試 m 對於 l' 到 r'，算出最好的位置
- 假設最好的位置是 m'，直接呼叫 solve$(l, m-1, l', m')$ 跟 solve$(m+1, r, m', r')$
- 終止條件：$r < l$（沒有東西好算）

Ciel and Gondolas

- solve(l, r, l', r') 表示我們要處理 l, r 的所有 $dp_{i,k}$ 計算，而且我們知道的額外資訊是這些的最佳來源只會在 $dp_{l',k}$ 到 $dp_{r',k}$
- 一開始呼叫 solve$(1, n, 1, n)$
- 選某個點 m，然後暴力嘗試 m 對於 l' 到 r'，算出最好的位置
- 假設最好的位置是 m'，直接呼叫 solve$(l, m-1, l', m')$ 跟 solve$(m+1, r, m', r')$
- 終止條件：$r < l$（沒有東西好算）
- 時間複雜度：???

Ciel and Gondolas

- solve(l, r, l', r') 表示我們要處理 l, r 的所有 $dp_{i,k}$ 計算，而且我們知道的額外資訊是這些的最佳來源只會在 $dp_{l',k}$ 到 $dp_{r',k}$
- 一開始呼叫 solve$(1, n, 1, n)$
- **選中點** $m = \lfloor \frac{l+r}{2} \rfloor$，然後暴力嘗試 m 對於 l' 到 r'，算出最好的位置
- 假設最好的位置是 m'，直接呼叫 solve$(l, m-1, l', m')$ 跟 solve$(m+1, r, m', r')$
- 終止條件：$r < l$（沒有東西好算）
- 時間複雜度：???

Ciel and Gondolas

- solve(l, r, l', r') 表示我們要處理 l, r 的所有 $dp_{i,k}$ 計算，而且我們知道的額外資訊是這些的最佳來源只會在 $dp_{l',k}$ 到 $dp_{r',k}$
- 一開始呼叫 solve$(1, n, 1, n)$
- **選中點** $m = \lfloor \frac{l+r}{2} \rfloor$，然後暴力嘗試 m 對於 l' 到 r'，算出最好的位置
- 假設最好的位置是 m'，直接呼叫 solve$(l, m-1, l', m')$ 跟 solve$(m+1, r, m', r')$
- 終止條件：$r < l$（沒有東西好算）
- 時間複雜度：$O(n \log n)$

Ciel and Gondolas

- 這個技巧被稱為「分治優化」

Ciel and Gondolas

- 這個技巧被稱為「分治優化」
- 永遠可以用？ 不見得，要看函數性質

Ciel and Gondolas

- 這個技巧被稱為「分治優化」
- 永遠可以用？ 不見得，要看函數性質
- Useful Insight: DP 轉移或 DP 值其實就是一些數字，有很多可以用來操作這些數字的東西
  - 各種資料結構
  - 各種函數性質偷懶少算某些點
  - 四邊形優化、Aliens Trick、Slope Trick
  - Seems fancy? 他們只是你學過的二分搜、二分搜跟 STL

How to DP?

How to DP?

- 定狀態
  - 什麼東西可以是狀態？
  - 哪些狀態不需要？

How to DP?

- 定狀態
  - 什麼東西可以是狀態？ 什麼都可以是狀態！
  - 哪些狀態不需要？

How to DP?

- 定狀態
  - 什麼東西可以是狀態？ 什麼都可以是狀態！
  - 哪些狀態不需要？ 有時候甚至是反過來 — 放寬狀態限制會不會比較好

How to DP?

- 定狀態
  - 什麼東西可以是狀態？ 什麼都可以是狀態！
  - 哪些狀態不需要？ 有時候甚至是反過來 — 放寬狀態限制會不會比較好
- 定基底 — 這個應該不是難事

How to DP?

- 定狀態
  - 什麼東西可以是狀態？ 什麼都可以是狀態！
  - 哪些狀態不需要？ 有時候甚至是反過來 — 放寬狀態限制會不會比較好
- 定基底 — 這個應該不是難事
- 定轉移
  - 轉移有哪些性質？ — DP 變成數值方法或資料結構問題
  - 怎麼轉移？ — DP 其實也是圖論問題