Dynamic Programming

Lecture by HNO2

Credit by yp155136, howard41436, boook, anj226

在課程開始之前

- 這週講的東西不會是第一階段階段考的內容
- 作業的 deadline 是 4/25(一)13:59:59 (多延後了兩天)
- 加分題統計的 deadline 是 4/25(一) 13:59:59 ，想加分的童鞋請把握時間(≥V≤)

Dynamic Programming(DP)

1. 回顧一下課前影片
2. 什麼是 DP？
3. DP 的一些細節
4. 練習

課前影片

1. 大家影片有看嗎!!!(Q&A)
2. 回憶一下:
   - 看的懂什麼是 DP 嗎？
   - 看的懂 LIS, LCS 嗎？
   - 如果有不懂的等課堂結束我會再講一次

什麼是 DP？

- 大家分享一下看完影片教學後，覺得什麼是 DP 吧！

什麼是 DP？

簡單來說，就是把大問題分成小問題：

- 用小問題**推出**大問題的答案

所以建 DP 時我們要想兩件事：

1. 這個大問題**如何改成**比較簡單的小問題？（怎麼訂狀態）
2. 大問題的答案**如何**從小問題的答案**推得**？（狀態間怎麼轉移）

如果想不到 DP 的作法時，可以考慮**最後一次/最後一格**會發生什麼事，通常作法都會和這個有關。

什麼是 DP？

DP 的幾個特性:

1. 重複子問題(Overlapping subproblems)
   - 一格會用到很多次，因為 DP 是一個用空間換取時間的演算法！
2. 最佳子結構(Optimal Substructure)
   - 講白話就是最佳解一定在所有考慮的子問題範圍內
   - 最佳化問題中，做了某個決定以後，變成一個比較小的問題
   - 計數問題中，我們考慮**所有**可能的最終決定，變成很多個小問題

什麼是 DP？

可是...這和前幾周學到的演算法有什麼不一樣？

1. 和分治的差異

- 通常分治的子問題只會出現一次

2. 和 Greedy 的差異

- Greedy 同樣也具有最佳子結構，但是 Greedy 可以透過**推理**排除絕對不可能的分支！

什麼是 DP？

哪些題目可能是 DP？

DP 通常拿來解決兩種問題:

1.最優解問題(通常是最大最小化問題)

2.計數問題

//大家想想，這兩種問題是不是很適合從小答案算出大答案呢？

DP 的一些細節

1. 怎麼估計 DP 的時間複雜度？

- 狀態複雜度：
  - 簡單來說就是陣列開了多少格
  - 每格都是一個狀態，都需要算出答案
- 轉移複雜度：
  - 轉移複雜度則是算出某一格的答案需要的時間複雜度
  - 可以觀察轉移式來得知
- DP 的總複雜度，就是**總共有幾格**乘上**一格需要計算的時間**！

舉個栗子

費氏數列(假設要算 n 個):

- 定義**狀態**:DP[i] 代表數列裡第 i 個數字。
- 狀態**轉移**:DP[i] = DP[i-1] + DP[i-2]。
- **狀態**複雜度:O(n)
- **轉移**複雜度:O(1)
- 總共:O(n)*O(1) = O(n)

舉個栗子

LCS（假設兩個字串的長度都是 n)：

- 定義**狀態**：DP[i][j] 代表 s1[1 : i] 跟 s2[1 : j] 的解
- 狀態**轉移**：DP[i] = max(DP[i-1][j], DP[i][j-1]) + 1

  DP[i-1][j-1] + 1 (s1[i]==s2[j])
- **狀態**複雜度：???（大家可以想想看！）
- **轉移**複雜度：???（大家可以想想看！）
- 總共：??? * ??? = ???

DP 的一些細節

因為 DP 總是分成這兩個部分，而壓複雜度有可能是從狀態下手，也有可能是從轉移下手，因此這兩件事是要分開討論的。

也就是「我的 DP 是n^3」這句話本身不夠表示你的 DP 演算法，必須要說「我的 DP 狀態 n^2，轉移 n」才夠精確。

我們通常用 nD/mD 來表示一個狀態 $O(N^n)$，轉移 $O(N^m)$ 的 DP 演算法。

在做每題 DP 時，都一定要好好寫出你的時間複雜度！

DP 的一些細節

2. 有關影片中有提到的 Bottom up 和 Top down:

- 絕大多數情況寫 Bottom up 就可以了，比較好寫，效率也比較高
- 不過也有 Top down 比較適用的時機，例如有很多狀態都用不到的時候(等等最後會有一個例子)

DP 的一些細節

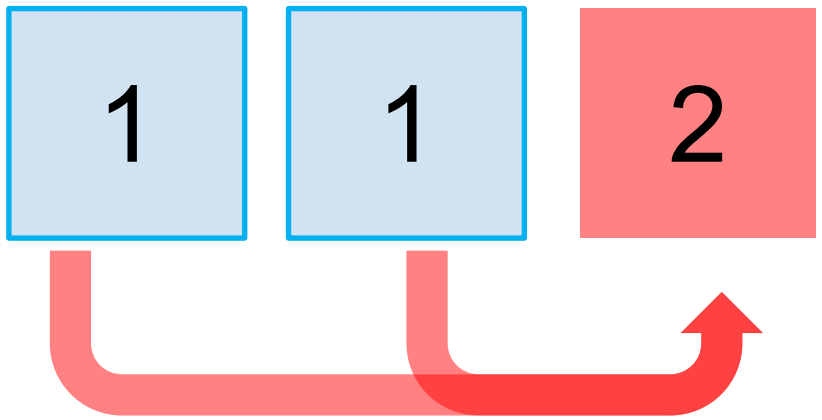
3. DP “bottom up” 的種類
以費氏數列舉例:

- 用「拉」的:
  - `for (int i = 2; i <= n; ++ i)`
    - `dp[i] = dp[i - 1] + dp[i - 2];`
- 用「推」的:
  - ```
    for (int i = 0; i <= n; ++ i) {
        if (i + 1 <= n) dp[i + 1] += dp[i];
        if (i + 2 <= n) dp[i + 2] += dp[i];
    }
    ```
    ```

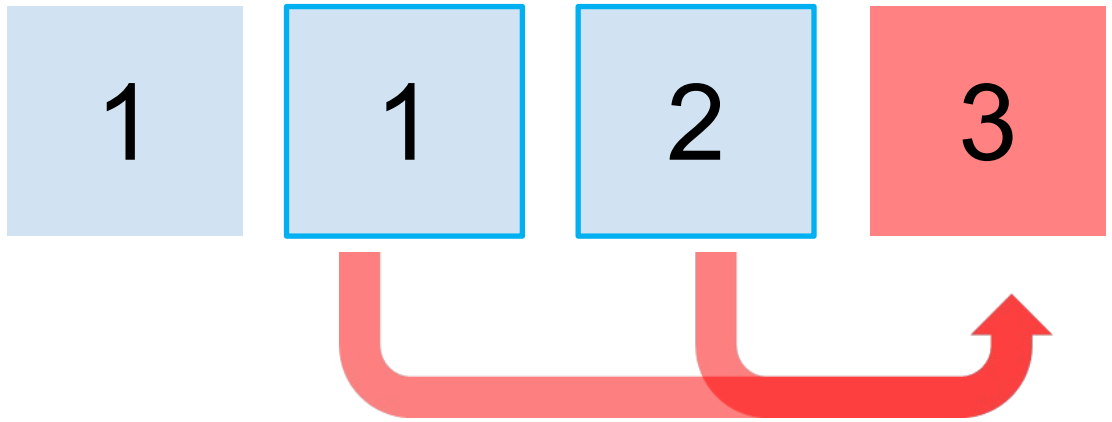
- 用「拉」的:

1	1

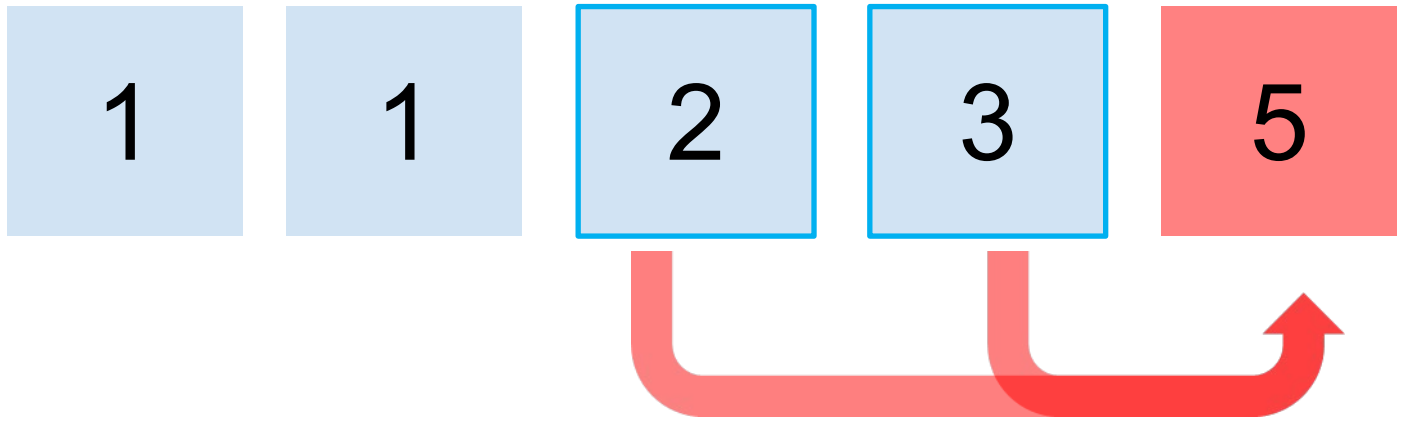
- 用「拉」的:



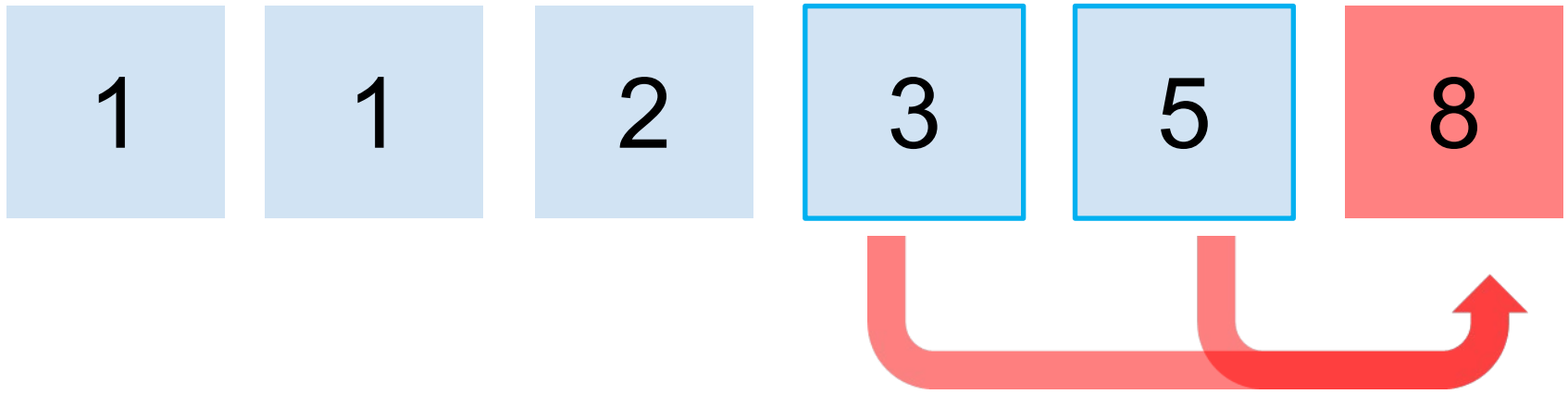
- 用「拉」的:



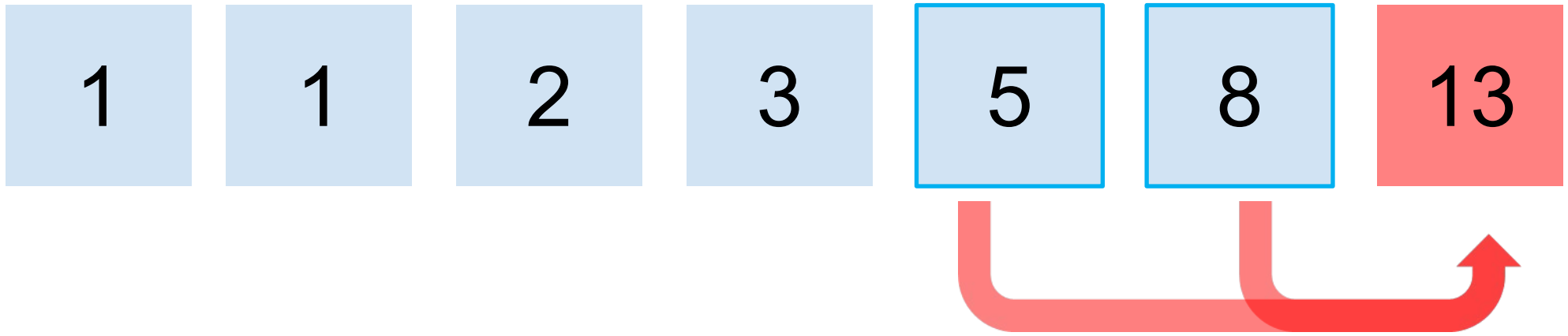
- 用「拉」的:



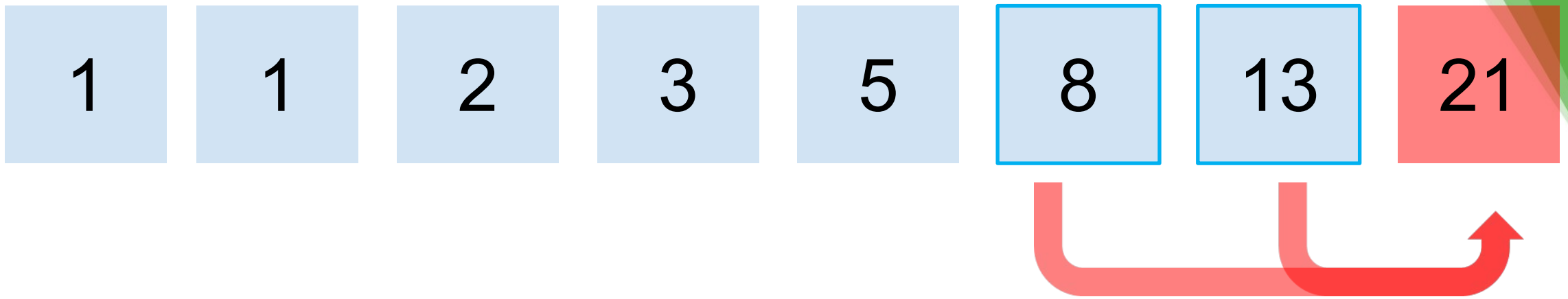
- 用「拉」的:



- 用「拉」的:



- 用「拉」的:



- 用「拉」的:

1	1	2	3	5	8	13	21

- 用「推」的:

1	0	0	0	0	0	0	0

- 用「拉」的:

1	1	2	3	5	8	13	21

- 用「推」的:

1	1	1	0	0	0	0	0

- 用「拉」的:

1	1	2	3	5	8	13	21

- 用「推」的:

1	1	2	1	0	0	0	0

- 用「拉」的:

1	1	2	3	5	8	13	21

- 用「推」的:

1	1	2	3	2	0	0	0

- 用「拉」的:

1	1	2	3	5	8	13	21

- 用「推」的:

1	1	2	3	5	3	0	0

- 用「拉」的:

1	1	2	3	5	8	13	21

- 用「推」的:

1	1	2	3	5	8	5	0

- 用「拉」的:

1	1	2	3	5	8	13	21

- 用「推」的:

1	1	2	3	5	8	13	8

- 用「拉」的:

1	1	2	3	5	8	13	21

- 用「推」的:

1	1	2	3	5	8	13	21

- 用「拉」的:

1	1	2	3	5	8	13	21

- 用「推」的:

1	1	2	3	5	8	13	21

DP 的一些細節

4. 模運算(mod)的一些細節

- 在很多 DP 問題中，很常看到「由於答案很大，請輸出答案 mod (10^9 + 7 / 998244353 / ???) 的結果」
- 假設現在是 mod 998244353，來想想看下面哪些轉移式的結果是正確的(先不考慮 DP 初始值)

```
const int MOD = 998244353;
int dp[101] = {0, 5, 100};
for (int i = 3; i <= 100; ++i) {
  dp[i] = dp[i - 1] + dp[i - 2] % MOD; // Q1
  dp[i] = dp[i - 1] * dp[i - 2] % MOD; // Q2
  dp[i] = (10LL * dp[i-1] - dp[i-2]) % MOD; // Q3
}
```

DP 的一些細節

```
dp[i] = dp[i - 1] + dp[i - 2] % MOD; // Q1
```

- C++ 是先**乘/除/取餘數**後**加減**
- `dp[i] = (dp[i - 1] + dp[i - 2]) % MOD;`

DP 的一些細節

```
dp[i] = dp[i - 1] * dp[i - 2] % MOD; // Q2
```

- 小心 dp 陣列是 int，兩個 < mod 的 int 相乘會 overflow!

- `dp[i] = 1LL * dp[i - 1] * dp[i - 2] % mod;`
- `dp[i] = (long long)dp[i - 1] * dp[i - 2] % mod;`

DP 的一些細節

```
dp[i] = (10LL * dp[i-1] - dp[i-2]) % MOD; // Q3
```

- 這次有記得先乘除後加減了，也小心 int overflow 的問題了
- 輸出第 90 項的時候，發現數字是 -611697889 !?!
- 在 C++ 中，如果一個負整數 mod 正整數，得到的結果是負數！((-1) mod 3 == -1)
- dp[i] = (10LL * dp[i-1] % MOD - dp[i-2] + MOD)%MOD

DP 的一些細節

5. 小提醒

debug 題：請問右邊這份 code 錯在哪（題目在此）

```cpp
#include <iostream>
using namespace std;

int grid[201][201], dp[201][201];

int main() {
  int m, n;
  cin >> m >> n;
  for (int i = 1; i <= m; i++)
    for (int j = 1; j <= n; j++)
      cin >> grid[i][j];

  for (int i = 1; i <= m; i++)
    for (int j = 1; j <= n; j++)
      dp[i][j] = grid[i][j] + max(dp[i][j - 1], dp[i - 1][j]);

  cout << dp[m][n] << '\n';
}
```

DP 的一些細節

5. 小提醒

問題就是出在，當 i = 1 或 j = 1 時，可能會從不存在的狀態轉移！

實作轉移時，需要確保**所有轉移來源都正確**，且**前面子問題不會用到後面子問題的答案**。

```cpp
#include <iostream>
using namespace std;

int grid[201][201], dp[201][201];

int main() {
  int m, n;
  cin >> m >> n;
  for (int i = 1; i <= m; i++)
    for (int j = 1; j <= n; j++)
      cin >> grid[i][j];

  for (int i = 1; i <= m; i++)
    for (int j = 1; j <= n; j++)
      dp[i][j] = grid[i][j] + max(dp[i][j - 1], dp[i - 1][j]);

  cout << dp[m][n] << '\n';
}
```

DP

- 那我們就來...練習!!!

跑步問題

Zerojudge b589

- 有 n 段路，每段路有一個分數 a_i，你每段路可以用其中一種速度
  1. 用走的：你不會得到任何分數
  2. 用跑的：你會得到 a_i 的分數
  3. 用衝的：你會得到 $2a_i$ 的分數，但你下一段路得用走的
- 請問你最多能得到多少分？

跑步問題

Zerojudge b589

- 有 n 段路，每段路有一個分數 a_i，你每段路可以用其中一種速度

  1.用走的:你不會得到任何分數

  2.用跑的:你會得到 a_i 的分數

  3.用衝的:你會得到 $2a_i$ 的分數，但你下一段路得用走的

  dp[i][0]:沒限制

  dp[i][1]:下一段路得用走的

跑步問題

Zerojudge b589

1.用走的:你不會得到任何分數

2.用跑的:你會得到 a_i 的分數

```
dp[i][0]
= max(dp[i - 1][0], dp[i - 1][0]+ a_i, dp[i - 1][1])
```

3.用衝的:你會得到 $2a_i$ 的分數, 但你下一段路得用走的

```
dp[i][1] = dp[i - 1][0] + 2a_i
```

pastebin.com/raw/DTY0cwzh

最大連續和問題

Zerojudge d784

- 有 n 個數字，每個數字可正可負。
- 請你選連續的一段數字，問最大總和可以是多少？

最大連續和問題

Zerojudge d784

- dp[i] : 以 i 為結尾的最大總和是多少
- 考慮要不要跟前面的接起來:
  dp[i] = max(dp[i - 1] + a[i], a[i]);
- 最後的答案為 dp 陣列的最大值
- 當然, 這題也有 greedy / 分治的作法！

最大和矩陣問題

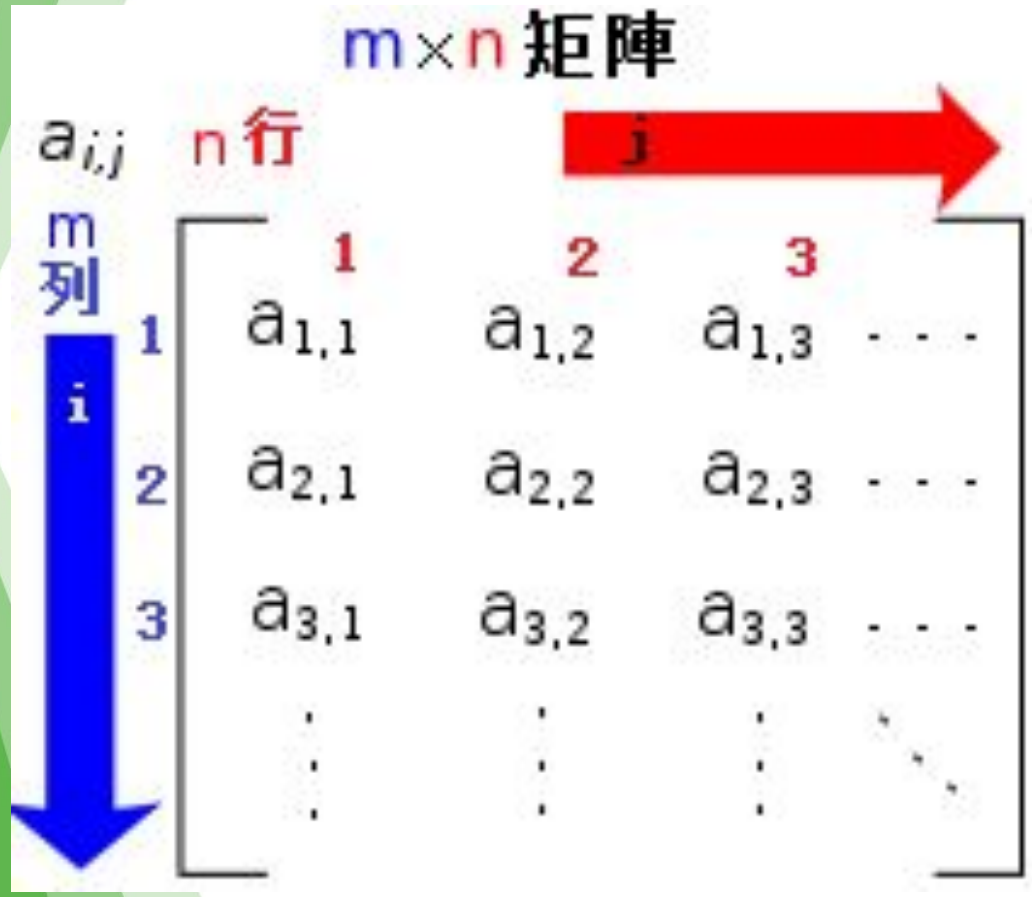
經典題:給你一個 $n \times m$ 的矩陣，所有的子矩陣中，最大的數字和是多少？($1 \le n,m \le 500$)

- 最裸最裸的做是 $O(n^3m^3)$
- 合理的裸做是 $O(n^2m^2)$

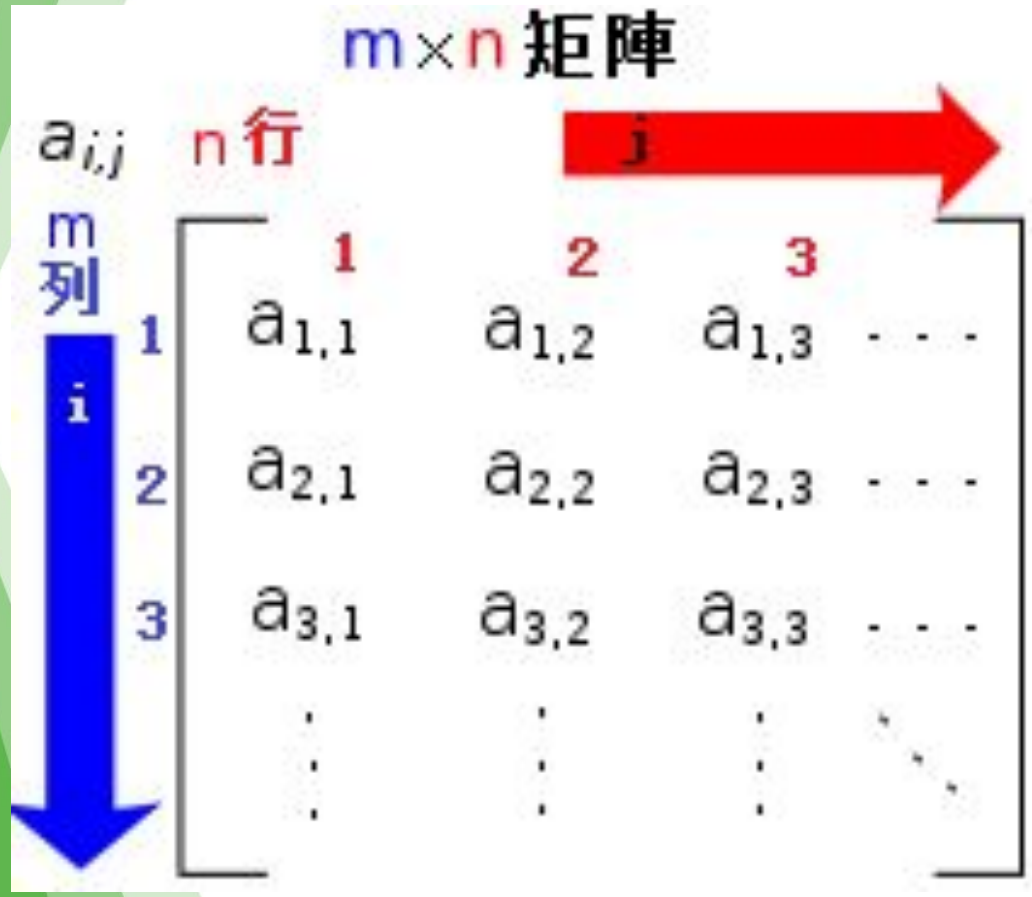
還能不能做得更好？

Hint: 跟上一題的作法有關係。

What is 矩陣？

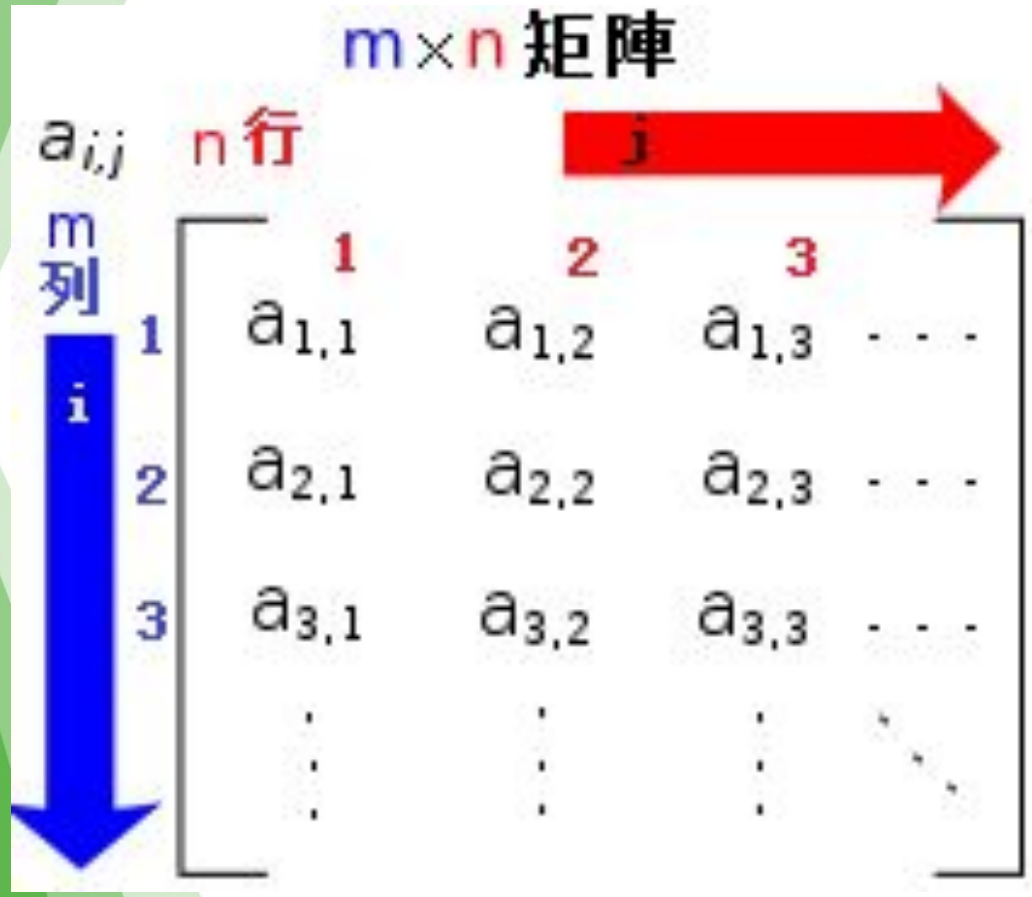


What is 矩陣？



1	-1	3	4	5
7	4	-5	7	19
21	13	8	0	0
-10	13	-19	-21	0

What is 矩陣？



1	-1	3	4	5
7	4	-5	7	19
21	13	8	0	0
-10	13	-19	-21	0

把他壓扁！！

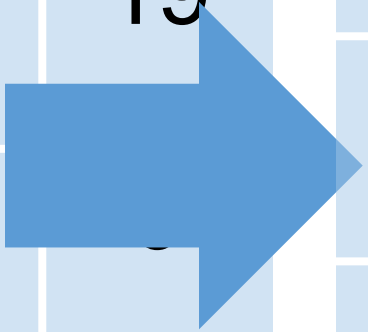
1	-1	3	4	5
7	4	-5	7	19
21	13	8	0	0
-10	13	-19	-21	0

假設我已經知道要取兩列了。

把他們兩列壓成一列！

把他壓扁！！

1	-1	3	4	5
7	4	-5	7	19
21	13	8	0	[illegible]
-10	13	-19	-21	0



8	3	-2	11	24
28	17	3	7	19
11	26	-11	-21	0

假設我已經知道要取兩列了。

把他們兩列壓成一列！

把他壓扁！！

8	3	-2	11	24
28	17	3	7	19
11	26	-11	-21	0

每一列做最大連續和。

這樣就等於做完所有 2×? 的子矩陣了！

對於高度 m、寬度 n 的矩陣：

- 枚舉高度，O(m)
- 對於壓完的每一行，O(m)
- 做一次最大連續和，O(n)
- $O(m^2n)$（嗎？）

最大和矩陣問題

- 當然也可以把行換成列、列換成行做事，所以實際時間複雜度是 $O(\min(n^2m, m^2n))$。
- 問題在於，怎麼快速的算把很多行壓扁以後的結果？

區間和

CSES 1646

- 給你一個陣列，並且有 q 次詢問，每次問某段區間 [L,R] 的數字和。
- 要求每次詢問時間複雜度 O(1)。

區間和

- 用 sum[i] 紀錄第 1 格至第 i 格的數字和，這樣要求區間和時可以用 sum[R]-sum[L-1]
- 這個 sum 叫做**前綴和**陣列

```
Sum[1] = a[1]                          = sum[0] + a[1]
Sum[2] = a[1] + a[2]                   = sum[1] + a[2]
Sum[3] = a[1] + a[2] + a[3]            = sum[2] + a[3]
Sum[4] = a[1] + a[2] + a[3] + a[4] = sum[3] + a[4]

a[3] + a[4] = Sum[4] - Sum[2]
```

區間和

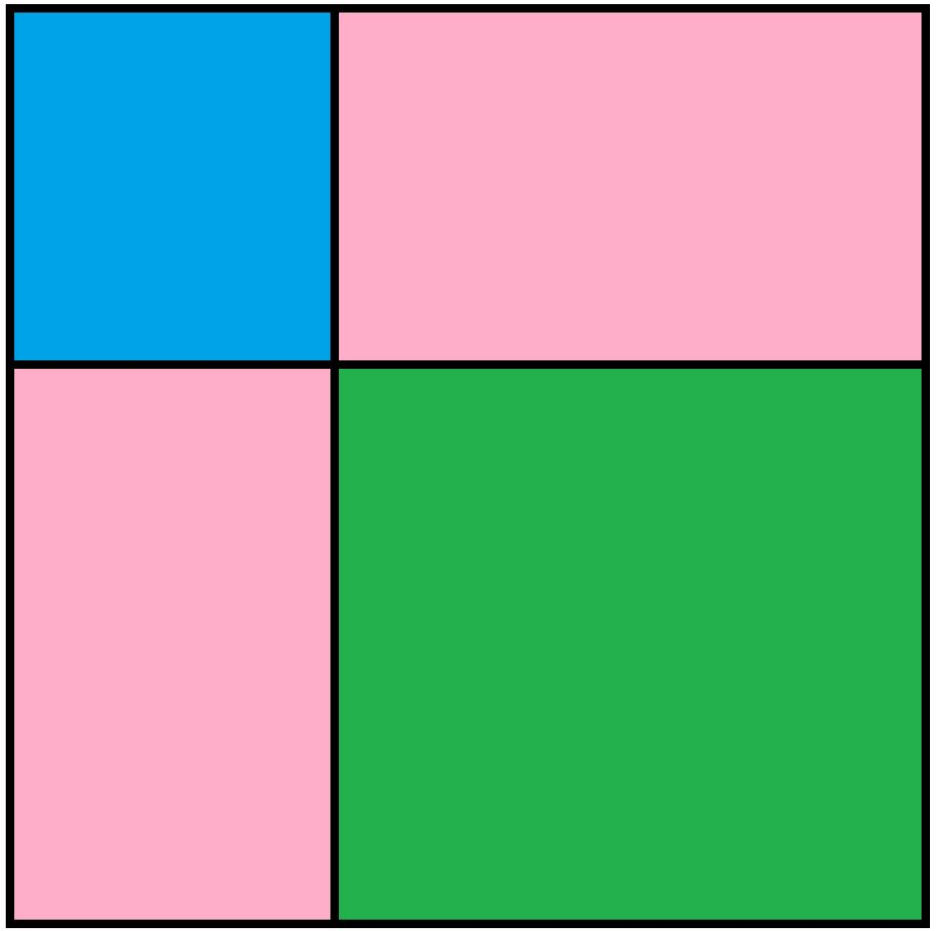
- 這是非常非常常用的技巧，請大家一定要記得，看到跟區間和有關的東西時常常可以這樣轉換：

  區間和 ⇔ 兩數的差

- 如果是二維的呢？（每次問你一個矩陣區塊的和）

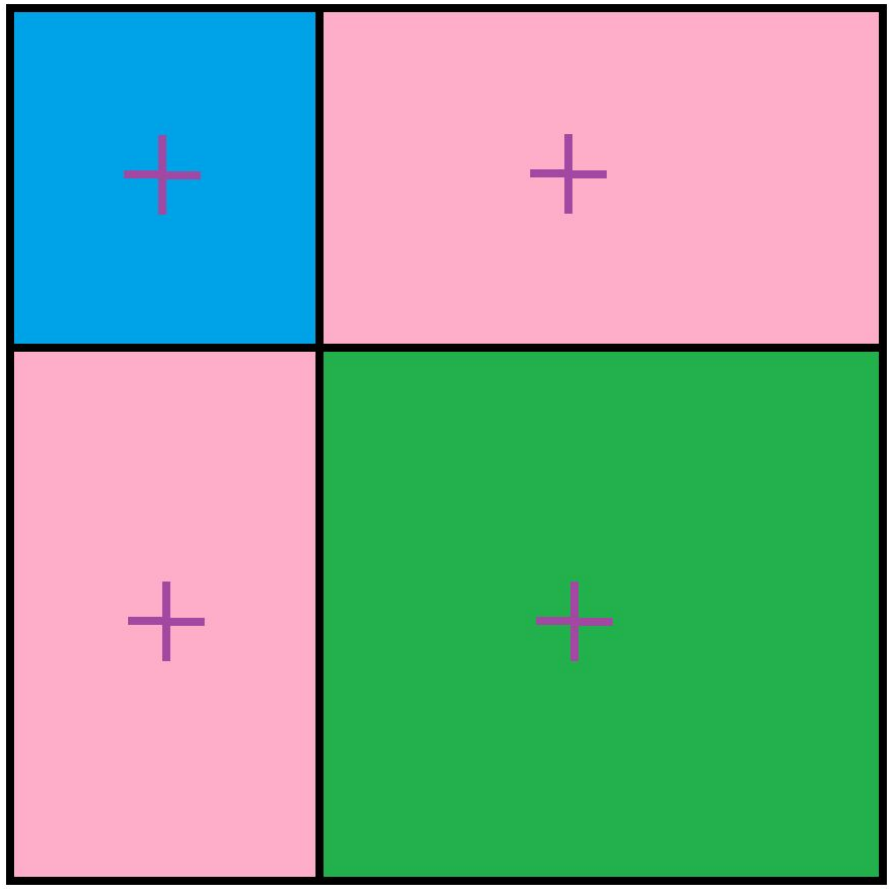
區段和

- 如果是二維的呢？(每次問你一個矩陣區塊的和)



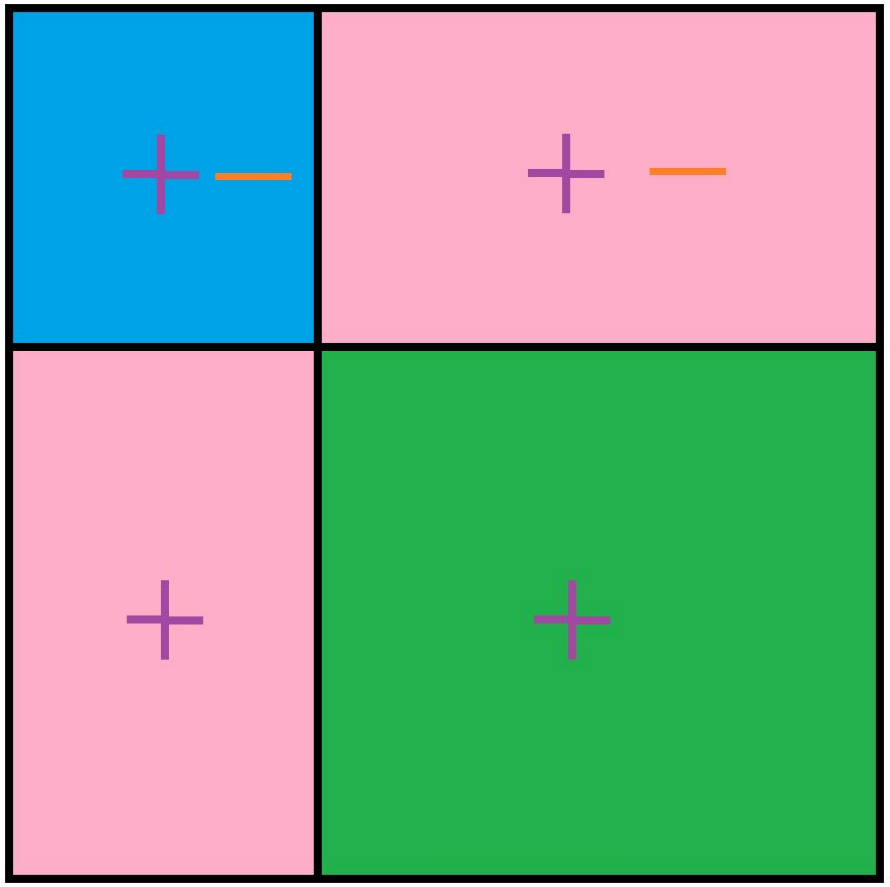
區段和

- 如果是二維的呢？(每次問你一個矩陣區塊的和)



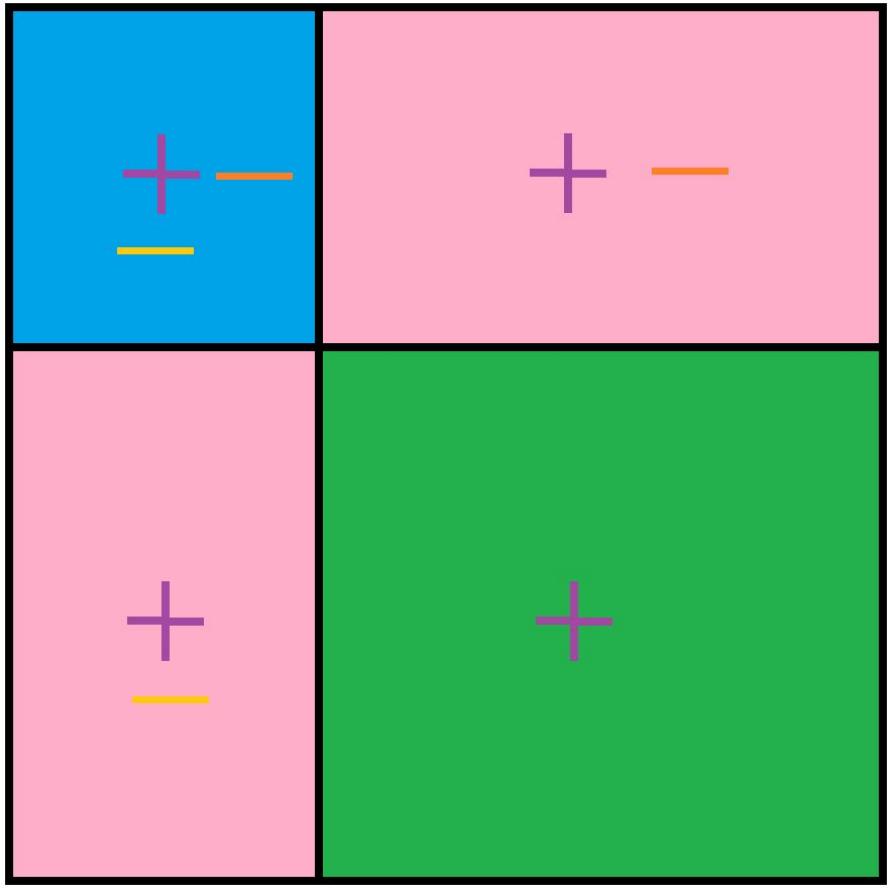
區段和

- 如果是二維的呢？(每次問你一個矩陣區塊的和)



區段和

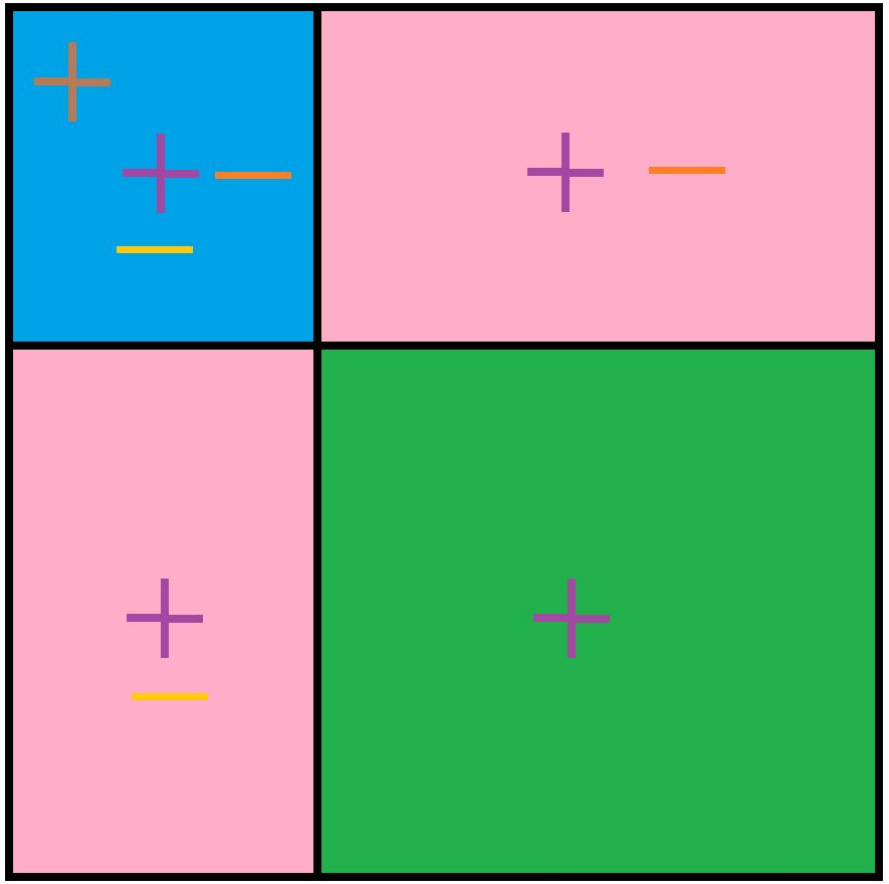
- 如果是二維的呢？(每次問你一個矩陣區塊的和)



區段和

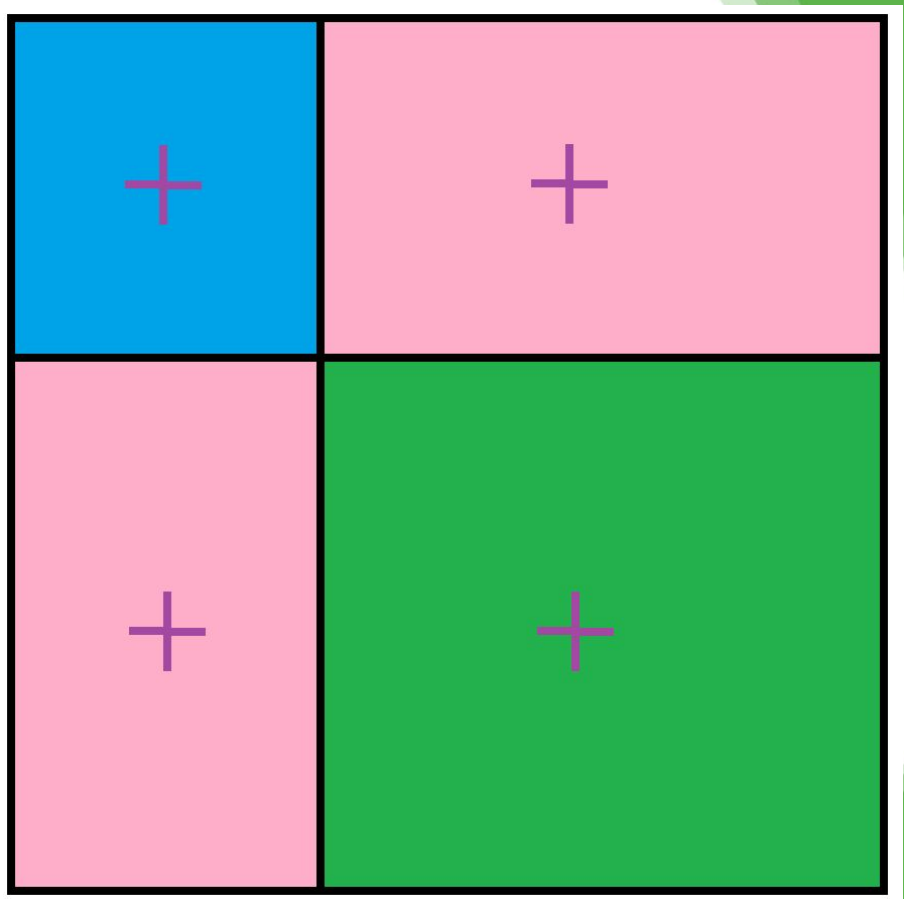
- 如果是二維的呢？(每次問你一個矩陣區塊的和)

```
A[l~L][r~R]
=    S[L][R]
   - S[L][r-1]
   - S[l-1][R]
   + S[l-1][r-1]
```



區段和

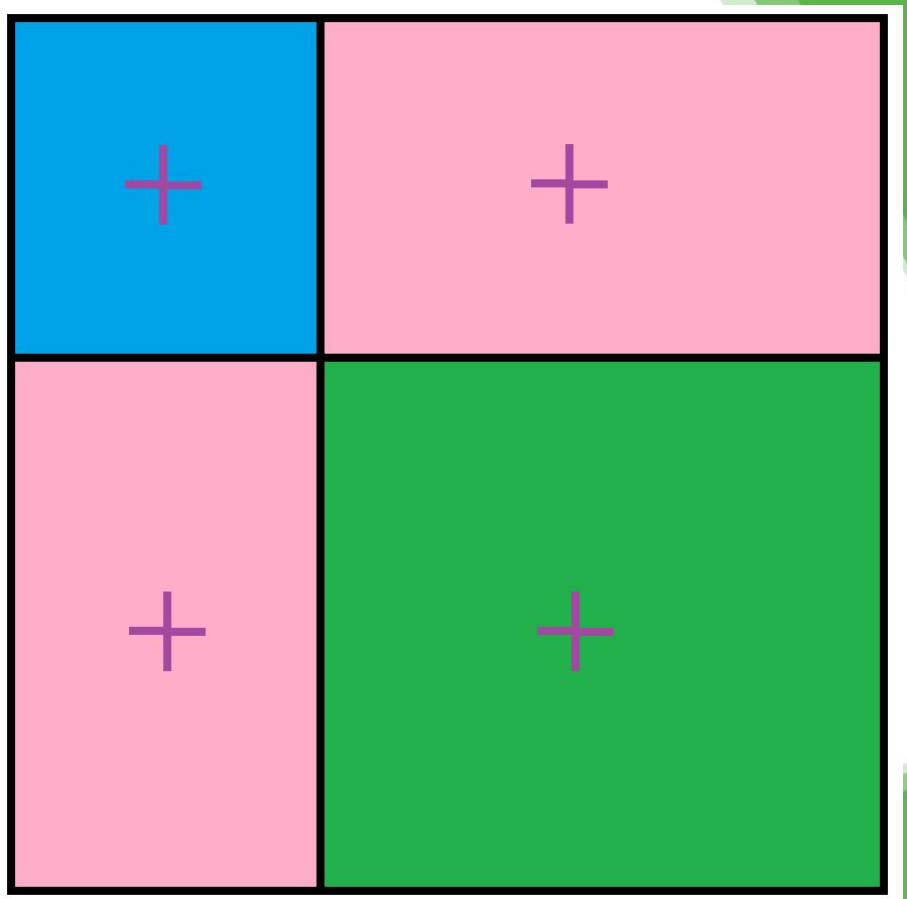
- 如果是二維的呢？(每次問你一個矩陣區塊的和)
- 要怎麼把 `S[i][j]` 蓋出來呢？



區段和

- 如果是二維的呢？(每次問你一個矩陣區塊的和)
- 要怎麼把 S[i][j] 蓋出來呢？

```
S[i][j] = S[i - 1][j] +
          S[i][j - 1] -
          S[i - 1][j - 1] +
          A[i][j]
```



矩陣最大空方形問題

- 給你一個 01 矩陣，請問裡面最大的全部都是 0 的正方形有多大？

矩陣最大空方形問題

- 給你一個 01 矩陣，請問裡面最大的全部都是 0 的方形有多大？
- Hint: 令 dp[i][j] 代表以 (i,j) 為右下角時正方形邊長可以多長。

0	0	1	0	0
0	0	0	0	0
0	0	0	0	1
0	0	0	0	1

矩陣最大空方形問題

- 給你一個 01 矩陣，請問裡面最大的全部都是 0 的方形有多大？

```
if (a[i][j] == 1)
  dp[i][j] = 0
if (a[i][j] == 0)
  dp[i][j] = min(dp[i - 1][j - 1]
           ,  dp[i][j - 1]
           ,  dp[i - 1][j]) + 1
```

- 想想看為什麼(只)需要從 dp[i - 1][j - 1],dp[i][j - 1], dp[i - 1][j] 三個狀態轉移？

矩陣最大空方形問題

- 給你一個 01 矩陣，請問裡面最大的全部都是 0 的方形有多大？

- O(nm) 狀態
- O(1) 轉移

0	0	1	0	0
0	0	0	0	0
0	0	0	0	1
0	0	0	0	1

矩陣乘法問題

經典題

- 給你一列矩陣，要從第一個乘到最後一個，保證兩相臨矩陣之間都是可以做乘法的。
  - 一個 a*b 的矩陣乘上一個 b*c 的矩陣需要做 a*b*c 次數字的乘法。
  - 矩陣有結合律，所以在不調換矩陣順序的狀況下可以用任意順序做乘法。
- 請問把所有矩陣乘起來最少需要做幾次數字乘法？

矩陣乘法問題

一個栗子:

[1x2] [2x4] [4x3] [3x5] [5x1]

[1x2] [2x4] [4x5] [5x1] 花 4*3*5

[1x4] [4x5] [5x1] 花 1*2*4

[1x5] [5x1] 花 1*4*5

[1X1] 花 1*5*1

矩陣乘法問題

還記得前面講過想不到解的時候可以嘗試從**最後一次**下手嗎？

- 在這題我們嘗試從「**最後一次合併**」的角度下手
- 由於序列本身的順序不能改變，因此如果我們以某個區間當成狀態，通常可以從左右的子區間轉移答案
- 例如本題，[L,R] 的矩陣要全部乘在一起，一定是先把 [L,K] 的矩陣乘在一起，以及 [K+1,R] 的矩陣乘在一起，再把剩下兩個矩陣相乘。
- 枚舉 K 後，剩下的部分是子問題！

- 狀態：2D
- 轉移：1D

矩陣乘法問題

- 寫成轉移式:
  DP[L][R] = max(DP[L][k]+DP[k+1][R]+兩個大矩陣相乘),
  for k in L ~ R

- 轉移順序？
  - 錯誤:

```
for (int i = 1; i <= n; i++)
   for (int j = i; j <= n; j++)
      for (int k = i; k <= j; k++)
         ...
```

  - 可以想想看為什麼是錯的
  - 正確作法:在轉移時, 要依照 R - L 從小到大轉移

轉移要小心捏

```
DP[L][R] = max( DP[L][k]+DP[k+1][R]+兩個大矩陣相乘 ),
           for k in L ~ R
```

	1	2	3	4
1	DP_{11}	DP_{12}	DP_{13}	DP_{14}
2	DP_{21}	DP_{22}	DP_{23}	DP_{24}
3	DP_{31}	DP_{32}	DP_{33}	DP_{34}
4	DP_{41}	DP_{42}	DP_{43}	DP_{44}

轉移要小心捏

```
DP[L][R] = max( DP[L][k]+DP[k+1][R]+兩個大矩陣相乘 ),
           for k in L ~ R
```

	1	2	3	4
1	DP_{11}	DP_{12}	DP_{13}	DP_{14}
2	DP_{21}	DP_{22}	DP_{23}	DP_{24}
3	DP_{31}	DP_{32}	DP_{33}	DP_{34}
4	DP_{41}	DP_{42}	DP_{43}	DP_{44}

轉移要小心捏

```
DP[L][R] = max( DP[L][k]+DP[k+1][R]+兩個大矩陣相乘 ),
           for k in L ~ R
```

	1	2	3	4
1	DP_{11}	DP_{12}	DP_{13}	DP_{14}
2	DP_{21}	DP_{22}	DP_{23}	DP_{24}
3	DP_{31}	DP_{32}	DP_{33}	DP_{34}
4	DP_{41}	DP_{42}	DP_{43}	DP_{44}

轉移要小心捏

```
DP[L][R] = max( DP[L][k]+DP[k+1][R]+兩個大矩陣相乘 ),
           for k in L ~ R
```

	1	2	3	4
1	DP_{11}	DP_{12}	DP_{13}	DP_{14}
2	DP_{21}	DP_{22}	DP_{23}	DP_{24}
3	DP_{31}	DP_{32}	DP_{33}	DP_{34}
4	DP_{41}	DP_{42}	DP_{43}	DP_{44}

轉移要小心捏

```
DP[L][R] = max( DP[L][k]+DP[k+1][R]+兩個大矩陣相乘 ),
           for k in L ~ R
```

消消樂

UVa 10559

- 有一排方塊，每個方塊都有顏色。
- 每次可以把連續顏色的一段消掉，得到 $(\text{消去長度})^2$ 的分數。
- 請問全部消完最多可以得到幾分？

跟剛剛那題很像的感覺，大家列列看狀態和轉移式吧！

消消樂

UVa 10559

- 有一排方塊，每個方塊都有顏色。
- 每次可以把連續顏色的一段消掉，得到（消去長度）^{2} 的分數。
- 請問全部消完最多可以得到幾分？

跟剛剛那題很像的感覺，大家列列看狀態和轉移式吧！
你確定你的演算法是對的嗎？

編輯距離 (edit distance)

- (CSES 1639) 給兩個字串 S, T, 你希望把 S 變成 T。

  在 S 插入一個字元要花 ins 塊錢, 刪除一個字元要花 del 塊錢, 取代一個字元要花 sub 塊錢。請問最小的花費為何？

  |S|, |T| <= 1000

- 想想看**狀態**、**轉移**、**初始值**要怎麼訂？

編輯距離 (edit distance)

- dp[i][j]: 把 S[1 : i], T[1 : j] 變一樣要花的最少費用
- 初始值:

```
dp[0][j] = j * ins
dp[i][0] = i * del
```

- 轉移:

```
dp[i][j] = min(dp[i - 1][j] + del,
               dp[i][j - 1] + ins,
               dp[i - 1][j - 1] + sub,
               dp[i - 1][j - 1](如果S[i] == T[j]))
```

關於 DP 大家要知道的

- 很多人覺得 DP 很難，但其實 DP 是簡單化問題的方法
- 看到 DP 題目時先建出可以轉移的狀態就好，先不管複雜度
- 找到不管時限會 AC 的 DP 算法，往往已經是成功的一半
  - 接下來的各種優化方式會在未來的 DP 課程教到
- DP 題目多練會進步得很快，因為從題目來建立狀態的方法在很多題目中是相似的，多接觸就會更加的熟悉

其他練習題

- CSES 1638 - Grid Paths
- TIOJ 2048 - 最大不連續和問題
- UVa 11420 - Chest of Drawers
- Zerojudge d652
- CSES 1639 - Edit Distance
- TIOJ 2173 - 搜集寶藏
- Optimal Binary Search Tree（非題目，參考用）

下課！ ><

- 第一階段的課就到這了。
- 下禮拜第一階段認證考加油 OwO