Advanced Competitive Programming

國立成功大學ACM-ICPC程式競賽培訓隊
nckuacm@imslab.org

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

如何解題?

解題

看懂問題

觀察問題

提出作法

實作程式

演算法的設計思維

演算法的設計思維

- 枚舉
- 動態規劃
- 分治法
- 貪心法

最大連續和問題

考慮**整數**數列: $a_1, a_2, \ldots, a_N$

讓 $a_L + a_{L+1} + \ldots + a_R$ 盡量大，其中 $1 \leqq L \leqq R \leqq N$

例如 -4, 2, 3, -1, 0, 4, -5, 6, -7
的最大連續和為 9

[2, 3, -1, 0, 4, -5, 6]

演算法的設計思維

- 枚舉
- 動態規劃
- 分治法
- 貪心法

枚舉

將有限個已知狀況全盤計算出來 湊出最後答案

- 解題的起手式
- 知道越多，能做得越多
- 可將**求值**問題轉為**判定**問題 e.g. $x^2 + x = 0$

枚舉

應用在問題中
將每個 L 與 R 配對舉出來

接著 `for(int k = L; k <= R; k++) sum += a[k];`

就能找出最大的 sum

枚舉: 最大連續和問題

```cpp
int best = a[1];

for (int L = 1; L <= N; L++) {
  for (int R = L; R <= N; R++) {
    int sum = 0;
    for (int k = L; k <= R; k++) sum += a[k];
    best = max(best, sum);
  }
}
```

其時間複雜度為 $O(N^3)$

演算法的設計思維

- 枚舉
- 動態規劃
- 分治法
- 貪心法

動態規劃

部份朋友可能知道若令 $S_i = A_1 + A_2 + \ldots + A_i$

$A_1 + A_2 + \ldots + A_{i-1} + A_i + \ldots + A_j = S_j$

$A_1 + A_2 + \ldots + A_{i-1} = S_{i-1}$

而 $A_i + A_{i+1} + \ldots + A_j = S_j - S_{i-1}$

有了 S_i 就可將連續和的計算從 O(N) 降為 O(1)

動態規劃: 最大連續和問題

```
S[0] = 0;
for (int i = 1; i <= N; i++)
  S[i] = S[i-1] + A[i];
```

從邊界遞推地紀錄所有問題的解，
且一個項用到前一項的最佳結果

動態規劃

```
for(int L = 1; L <= N; L++)
  for(int R = L; R <= N; R++)
    best = max(best, S[R] - S[L-1]);
```

動態規劃

好處是將會重複使用到的解都保存下來了，
就能省下不少時間

不用像枚舉一樣重新計算

```
for(int k = L; k <= R; k++) sum += A[k];
```

演算法的設計思維

- 枚舉
- 動態規劃
- 分治法
- 貪心法

分治法

- 分治 (divide & conquer) 簡稱 D&C
- 將一個大的問題
- 分成幾個**互相獨立**的子問題
- 然後再將子問題分成子子問題
- 一直重複分割的動作直到最小問題 (邊界)
- 接著讓子問題合併求出父問題。

分治法: 最大連續和問題

- 將數列切一半 (分割)

- 左半的最大連續和為何 (子問題)
- 右半的最大連續和為何 (子問題)
- 包含”切開的分水嶺”的最大連續和 (子問題☆)

- 選出三者中最大值，就是整個數列的解 (合併)

分治法: 原大小的問題

P [a(1), a(2), a(3), a(4), a(5)]

假設 N = 5

分治法: 分割問題

[a(1), a(2), a(3), a(4), a(5)]

分治法: 子問題

[a(1), a(2), a(3), a(4), a(5)]

P [a(1), a(2)]   P [a(3), a(4), a(5)]

分治法: 分割問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

P [a(1)]

P [a(2)]

分治法: 最小子問題(邊界)

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

[a(1)]

P [a(2)]

Return a(1)

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

L=P [a(1)]

P [a(2)]

分治法: 最小子問題(邊界)

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

L=P [a(1)]

[a(2)]

Return a(2)

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

L=P [a(1)]

R=P [a(2)]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

L=P [a(1)]

R=P [a(2)]

P [..., a(1), a(2), ...]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

L=P [a(1)]

R=P [a(2)]

maxSum [..., a(1), a(2), ...]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

L=P [a(1)]

R=P [a(2)]

M=maxSum [..., a(1), a(2), ...]

分治法: 合併問題 (回傳解)

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

Return max(L, M, R)

分治法: 子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

P [a(3), a(4), a(5)]

分治法: 分割問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

P [a(3)]

P [a(4), a(5)]

分治法: 最小子問題 (邊界)

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

[a(3)]

P [a(4), a(5)]

Return a(3)

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

L=P [a(3)]

P [a(4), a(5)]

分治法: 合併問題 (回傳解)

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

L=P [a(3)]

[a(4), a(5)]

Return max(L, M, R)

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

L=P [a(3)]

R=P [a(4), a(5)]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

L=P [a(3)]

R=P [a(4), a(5)]

P [..., a(3), a(4), ...]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

L=P [a(3)]

R=P [a(4), a(5)]

maxSum [... a(3), a(4), ...]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

L=P [a(3)]

R=P [a(4), a(5)]

M=maxSum [... a(3), a(4), ...]

分治法: 合併問題 (回傳解)

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

Return max(L, M, R)

分治法: 子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

R=P [a(3), a(4), a(5)]

分治法: 子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

R=P [a(3), a(4), a(5)]

P [..., a(2), a(3),...]

分治法: 子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

R=P [a(3), a(4), a(5)]

maxSum [..., a(2), a(3),...]

分治法: 子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

R=P [a(3), a(4), a(5)]

M=maxSum [..., a(2), a(3),...]

分治法: 合併問題 (回傳解)

[a(1), a(2), a(3), a(4), a(5)]

Return max(L, M, R)

分治法: 原問題

P [a(1), a(2), a(3), a(4), a(5)]

分治法: 原問題

G=P [a(1), a(2), a(3), a(4), a(5)]

得到原問題的解了

分治法: 複雜度

假設原問題大小為: N

考慮實際時間花費

分治法: 時間花費

原問題時間花費為: T(N)

分割問題後為: T(N/2) + T(N/2)

maxSum: N

分治法: 時間花費

合併問題得 $T(N) = 2*T(N/2) + N$

並且最小子問題 $T(1) = 1$

分治法: 時間花費

$$T(N)$$

$$= 2^1 * T(N/2^1) + 1 * N = 2^1 * (2 * T(N/2^2) + N/2^1) + 1 * N$$

$$= 2^2 * T(N/2^2) + 2 * N = 2^2 * (2 * T(N/2^3) + N/2^2) + 2 * N$$

$$= 2^3 * T(N/2^3) + 3 * N = 2^3 * (2 * T(N/2^4) + N/2^3) + 3 * N$$

$$\ldots$$

$$= 2^? * T(1) + ? * N$$

想想看 “問號” 為多少？

分治法: 複雜度

$T(N) = 2^{\lg N} * T(1) + (\lg N) * N$

$\wedge\ T(1) = 1$

$\Rightarrow T(N) = 2^{\lg N} + N\lg N$

$\Rightarrow T(N) = O(N\lg N)$

演算法的設計思維

- 枚舉
- 動態規劃
- 分治法
- 貪心法

貪心法

- 每次做一個在當下看起來最佳的決策
- 進而漸漸求出全局最佳解

- 貪心法是動態規劃的特例

觀察問題

把樸素的過程仔細攤開來看

每次固定一個左界，接著遞增右界以枚舉子序列

也可以看做是，固定一個右界，一直遞減左界

$$
\begin{array}{ccccccccc}
 & & & & a_{L+2} & + & \dots & + & a_R \\
 & & a_{L+1} & + & a_{L+2} & + & \dots & + & a_R \\
a_L & + & a_{L+1} & + & a_{L+2} & + & \dots & + & a_R
\end{array}
$$

觀察問題

下列三者，誰較大?

$$a_{L+2} + \ldots + a_R$$
$$a_{L+1} + a_{L+2} + \ldots + a_R$$
$$a_L + a_{L+1} + a_{L+2} + \ldots + a_R$$

這似乎引導我們，去重新提問

固定一個右界，其所有子序列中的最大和為何?

觀察問題

固定一個右界 p，其所有子序列中的最大和 f(p)
對於**每一種右界**，{ f(1), f(2), ..., f(N) }

算出其最大值 max { f(1), f(2), ..., f(N) }

這跟原問題是等價的!

觀察問題

對於**每一種右界**，f(i)

f(i) 與 f(i+1) 只差在是否含有 a_{i+1}

但 $f(i+1) = f(i) + a_{i+1}$ 嗎?

觀察問題

對於**每一種右界**，f(i)

f(i) 與 f(i+1) 只差在是否含有 a_{i+1}

但 $f(i+1) = f(i) + a_{i+1}$ 嗎?

那倒未必，如果 $f(i) < 0$ 的話，$f(i+1) = a_{i+1}$

提出作法

```
int best = a[1], sum = 0;

for(int R = 1; R <= N; R++) {
  sum = max(a[R], sum + a[R]);
  best = max(best, sum);
}
```

複雜度為 O(N)

枚舉範例

回文子串

回文子串

給定一長度 N 字串 A，算出有幾個回文子字串

例如: aabab
有 a, a, b, a, b, aa, aba, bab 共 6 個回文子字串

回文子串 (樸素解)

將 $O(N^2)$ 個子字串算出來，
接著**判定**其是否為回文串

共 $O(N^3)$ 複雜度

觀察問題

觀察得知，
回文是前後相同的結構

觀察問題

觀察得知，

子字串 $A_{i:j} = A_iA_{i+1}...A_j$ 若**是回文**，可以為空字串

且 $A_{i-1} = A_{j+1}$

則$A_{i-1}A_{i:j}A_{j+1}$ 是回文

例如 cbc 是, 則 pcbcp 是, 但 qcbcp 不是

觀察問題

觀察得知，

子字串 $A_{i:j}$ 若**不是回文**

則 $A_{i-1}A_{i:j}A_{j+1}$ 必不是回文

例如 abc 不是, 則 pabcp 不是

提出作法 ($O(N^2)$ 解)

也就是說，能從**每個**字元**往外**找回文

挑 A_i 就檢查 $\mathbf{A_{i-1}}A_i\mathbf{A_{i+1}}$ 是否為回文
是的話繼續往外看 $\mathbf{A_{i-2}}A_{i-1}A_iA_{i+1}\mathbf{A_{i+2}}$

依此類推，每個字元做一遍

提出作法 ($O(N^2)$ 解)

```
int ans = 1;

// 從單個字元擴充
for(int i = 0; i < N; i++)
  for(int l = 1; i-l >= 0 && i+l < N; l++) {
    if(S[i-l] != S[i+l]) break;
    ans++;
  }
```

提出作法 ($O(N^2)$ 解)

但如果只用這個方法
abba 這樣的子字串就沒辦法判斷到

所以除了從**每個**字元往外擴充檢查以外
還要從字元跟字元之間往外擴充檢查

提出作法 (O(N²) 解)

```
// 從字元跟字元間擴充
for(int i = 0; i < N; i++)
  for(int l = 1; i-l >= 0 && i+l-1 < N; l++) {
    if(S[i-l] != S[i+l-1]) break;
    ans++;
  }
```

XOR equation

XOR equation

給定 N, V 及 a_i，其中 $1 \le i \le N$，
算出有多少組 $1 \le x_i \le a_i$ 滿足 $x_1 \oplus x_2 \oplus \ldots \oplus x_N = V$

限制 $N \le 15, a_i \le 9$, $\prod_1^N a_i \le 40000$

N = 3 可能寫法

```
for(int i = 1; i <= a1; i++)
   for(int j = 1; j <= a2; j++)
      for(int k = 1; k <= a3; k++)
         if(i^j^k == V) cnt++;
```

N 太大

不過當 N 夠大
程式碼就有點難寫了

廢話不多說，遞迴真好用

遞迴

```
void dfs(int i, int res) {
   if(i > N) {
      if(res == V) cnt++;
      return;
   }

   for(int j = 1; j <= a[i]; j++)
      dfs(i+1, res ^ j);
}
```

分治範例

合併排序法

合併排序法

給長度為 n 的數列 $a_0,a_1,\cdots,a_{n-1}$

將數列依照數字大小從小排到大

觀察問題

為了應用分治法
先將數列分成兩區

觀察問題

為了應用分治法
先將數列分成兩區

依照先前的經驗，要假設這兩區已經解完(排列)了
利用這兩區合併成整體排序完的數列

觀察問題

為了應用分治法
先將數列分成兩區

依照先前的經驗，要假設這兩區已經解完(排列)了
利用這兩區合併成整體排序完的數列

例如 2, 3, 7, 9 和 4, 6, 8, 10
要併成 2, 3, 4, 6, 7, 8, 9, 10

提出作法

左右區比較一下，看目前誰的數字小
小的那個就加入排列好的隊伍中

例如 2, 3, 7, 9 和 4, 6, 8, 10
要併成 2, 3, 4, 6, 7, 8, 9, 10

提出作法

```
void split(int l, int r) {// [l, r)
   if (r-l == 1) return;

   int m = (l+r)/2;
   split(l, m);
   split(m, r);

   merge(l, r);
}
```

逆序數對

逆序數對

給長度為 n 的數列 $a_0,a_1,\cdots,a_{n-1}$若 $i<j$ 且 $a_i>a_j$
則 (a_i,a_j) 稱為**逆序**數對

請計算出該數列中有多少逆序數對

提出作法 (樸素解)

```
int cnt = 0;

for(int i = 0; i < n; i++)
  for(int j = i+1; j < n; j++)
    if(a[i] > a[j]) cnt++;
```

觀察問題

N = 2，數列 4, 3 則有 (4, 3)

N = 4，數列 5, 8, 2, 9 則有 (5, 2) (8, 2)

…

觀察問題

逆序數對的構成就是右邊數字比左邊大 廢話

在數這些數對時
可以發覺到 當數完一逆序對 (a, b) 時
若數列中 a 跟 b 之間有 $c > a$
那不就推得 $c > b$ 嗎?

提出作法 1

將 b 之前的數字排序 (升序)

這樣一來，當 a 數完之後，直到 b 以前

所有比 a 大的那些數字們都不需要重複計算 a 配過的數

提出作法 1

例如 b = 5
那麼 7, 3, 6, **5**, 9, 13, 1, 2
則 b 以前的數字排序，得 3, 6, 7, **5**, 9, 13, 1, 2

拿 3 去配 b 後的數字，(3, 1), (3, 2)
那麼輪到 6 和 7 時，(6, 1), (6, 2), (7, 1), (7, 2) 不需計算

觀察問題

並且若 (a, b) 不是逆序對
表示任意 x > b，(a, x) 都不是逆序對

提出作法 2

將 a 之後的數字排序 (升序)

這樣一來，當遇到 b 或是比 a 大的數字
a 不需要再繼續往後配對，可以把 a 換下來

提出作法 2

例如 a = 6

那麼 7, 3, **6**, 5, 9, 13, 1, 2

則 a 以後的數字排序，得 7, 3, **6**, 1, 2, 5, 9, 13

拿 3 去配，(3, 1), (3, 2)

接著碰到 (3, 5) 不是逆序對，則可以換下個數字 6 繼續配

提出作法 3

將兩種做法綜合起來
會發現，可以將數列分成兩堆，分別排序

這樣一來能省下很多計算量

提出作法 3

假設分成兩個區間 [l, m), [m, r)
分別都已經排序

```
for(int i = l, j = m; i < m; i++) {
   while(j < r && a[i] > a[j]) j++;

   cnt += j-m;
}
```

觀察問題

但這樣只有將兩個區間之間的逆序數對找出來而已

區間本身中的逆序對怎麼數？

提出作法 4

區間怎麼來的，就如法炮製

也對這兩個區間分別切割！
這樣就成了四個區間，四個區間再變八個區間
依此類推..

提出作法 4

接著會切到不能再切
例如數列 4, 3，再切一次分別變成 4 和 3

但 4 和 3 本身就是排好序的 (只有一個元素)
回到數列 4, 3 這個問題，當他數完區間間的數對
也要將他排序，變為 3, 4

提出作法 4 (分而治之)

```
int count(int l, int r) { // [l, r)
  if(r-l == 1) return 0;

  int m = (l+r)/2, cnt = 0;
  cnt += count(l, m);
  cnt += count(m, r);

  vector<int> b; // 保存升序數列

  /* 下一頁 */

  copy(b.begin(), b.end(), a+l);

  return cnt;
}
```

提出作法 4 (分而治之)

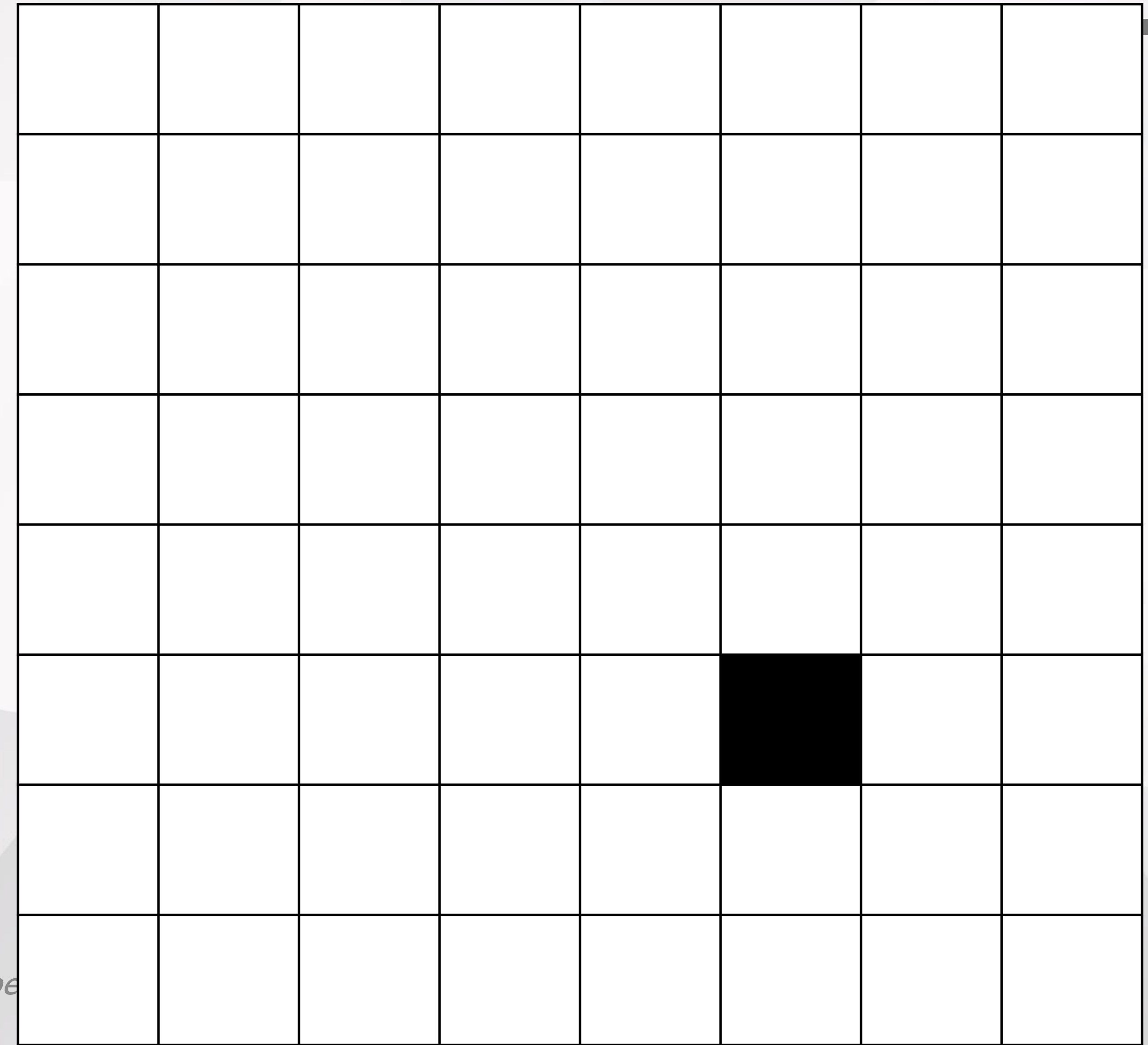
```
int j = m;
for(int i = l; i < m; i++) {
   while(j < r && a[i] > a[j])
      b.push_back(a[j++]);

   b.push_back(a[i]);

   cnt += j-m;
}
while(j < r) b.push_back(a[j++]);
```

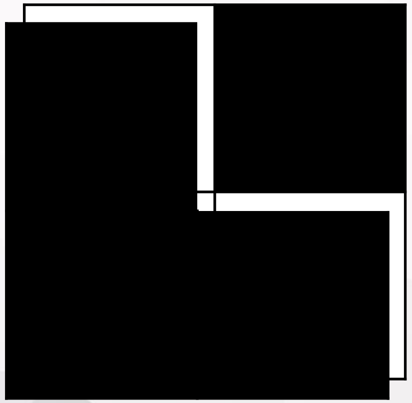
填 L 型格子

填 L 型格子

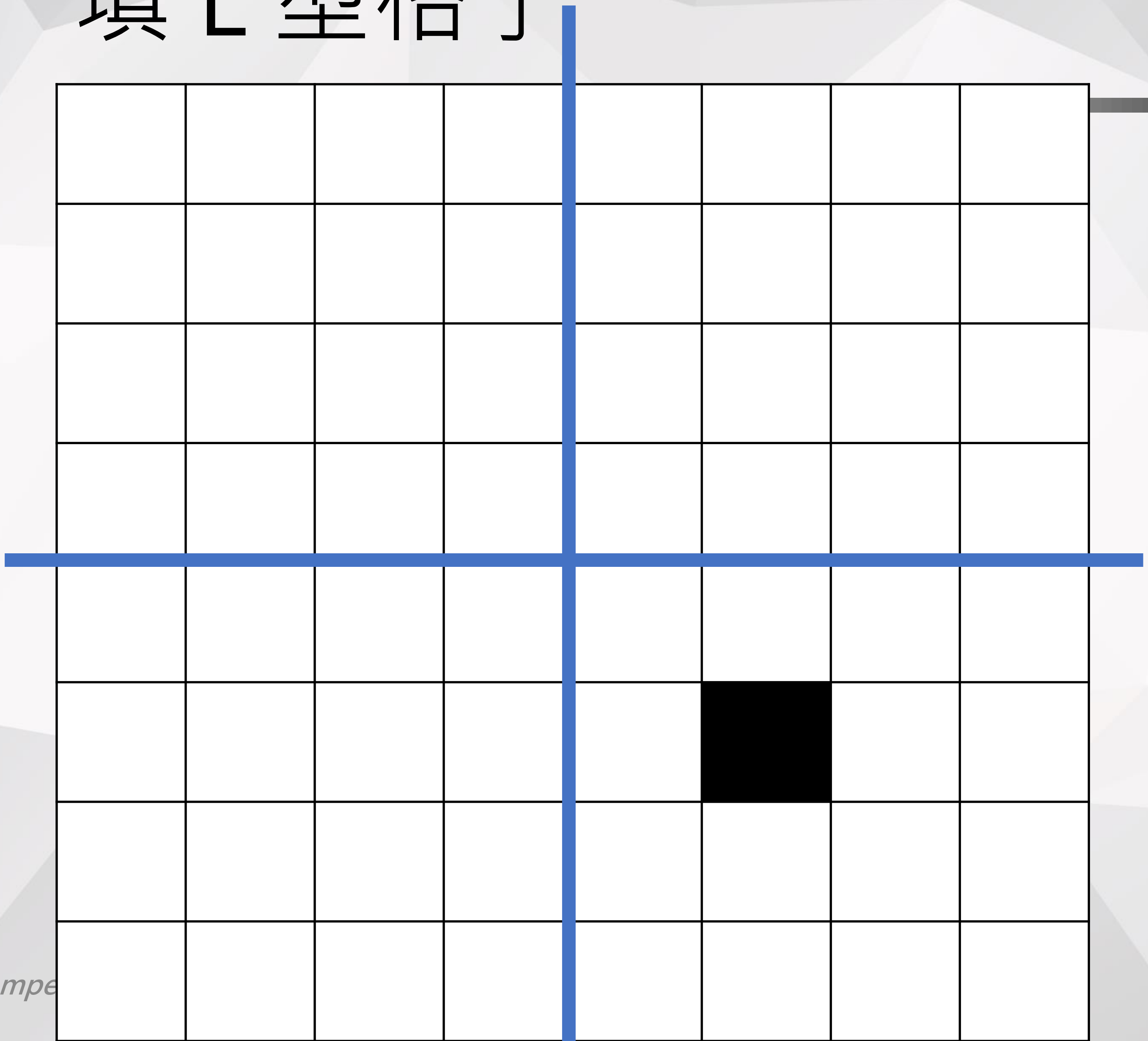


填 L 型格子

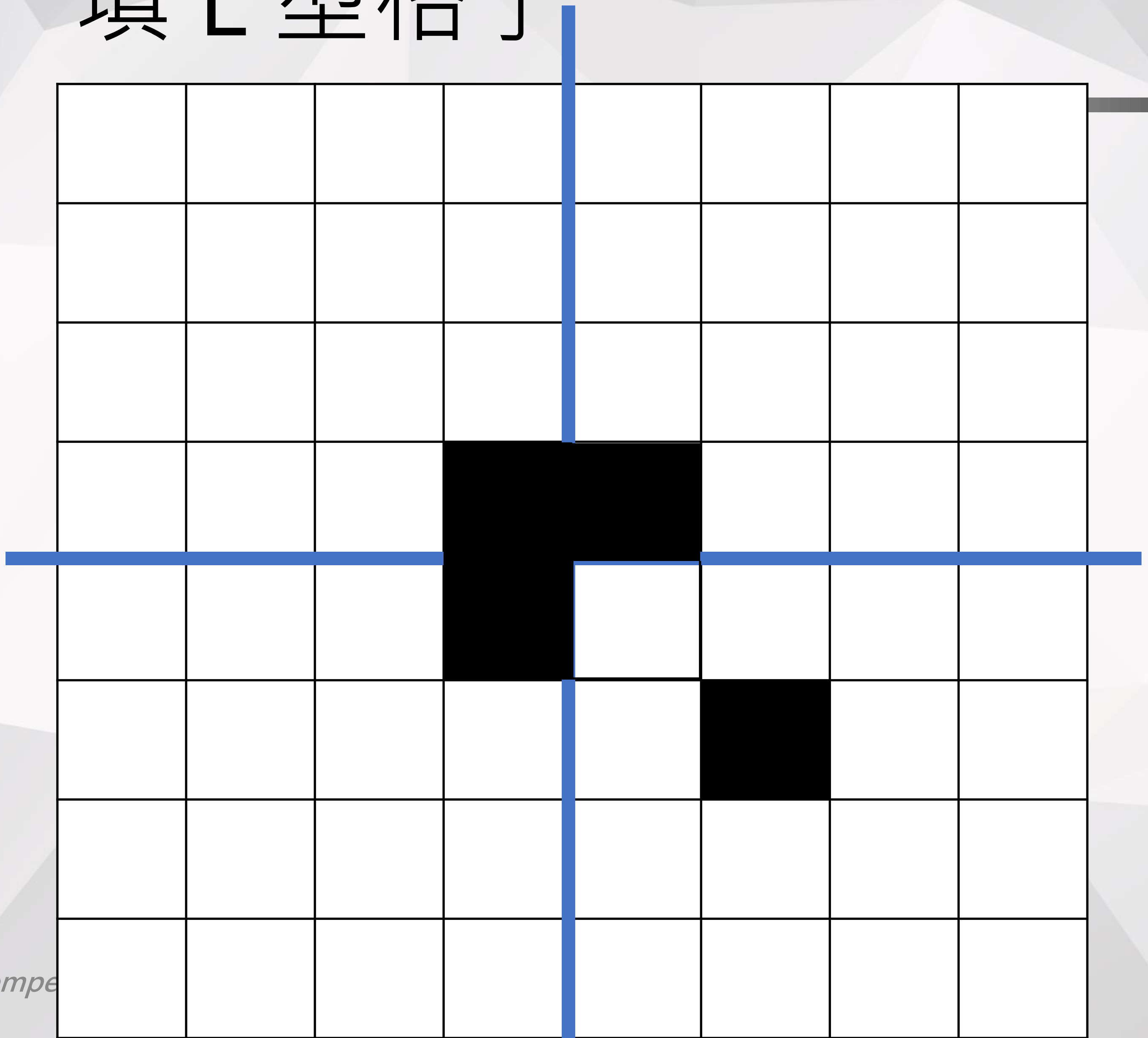
填 L 型格子



填 L 型格子



填 L 型格子



動態規劃範例

上樓梯

上樓梯

每走一次可以走 1 或 2 階

從 0 階(地板)開始走到 n 階的走法有幾種？

觀察問題

先從小問題觀察起

若 n=0，則答案直接為 1
若 n=1，則答案為 1

若 n=2，
由於可以每次都走 1 階，
或直接走 2 階，則答案為 2

觀察問題

根據題目

會造成方法數變動的就只有兩種**決策**
並且每使用一種走法
就能夠改變走到某階的答案

觀察問題

走到 i 階有兩種走法，從 i-1 和 i-2 走過來

那麼
走到 i-1 又有幾種走法?
走到 i-2 又有幾種走法?

提出作法

狀態 dp(i) 代表從 0 階走到 i 階有幾種走法

則**狀態轉移** $dp(i) = dp(i-1) + dp(i-2)$

提出作法

```
dp[0] = dp[1] = 1;

for(int i = 2; i < n; i++)
  dp[i] = dp[i-1] + dp[i-2];
```

最小路徑和

最小路徑和

從左上角走到右下角，
每次只往右或下走的路徑**最小總和**為何？

1	2	4	9
6	8	3	4
5	3	2	7

觀察問題

往右，那麼位置 (i, j−1) 就會變成 (i, j)

往下，那麼位置 (i−1, j) 就會變成 (i, j)

提出作法

定義 dp(i, j) 為從起點到 (i, j) 的最小和

直接的，$dp(i,j)=\min(dp(i-1, j),dp(i, j-1)) + a_{i,j}$

提出作法

```
for(int t = 1; t <= max(M, N); t++)
  dp[0][t] = dp[t][0] = 0;

for(int i = 1; i <= M; i++)
  for(int j = 1; j <= N; j++)
    dp[i][j] = min(dp[i-1][j],
                   dp[i][j-1]) + a[i][j];
```

Longest Increasing Subsequence (LIS)

Longest Increasing Subsequence (LIS)

在給定 N 長度序列 a，找到一個子序列，
為**嚴格遞增**且長度**最長**

例如 a = (1, 4, 2, 3, 8, 3, 4, 1, 9)
則 LIS 為 (1, 2, 3, 4, 9) 或 (1, 2, 3, 8, 9)

觀察問題

若某數字在某遞增子序列**後**出現，
且比此序列末項還**大**，加入它就能成更長的遞增子序列！

例如子序列 1, 2, 3, 4
接著在 4 以後有出現 9
加上去成 1, 2, 3, 4, 9

提出作法

定義狀態 dp(n) 為以第 n 個數為**結尾**的 LIS **長度**

狀態轉移方程為，對於所有 $i < j$ 且 $a_i < a_j$
$dp(j) = max(dp(i)+1)$

邊界為 $dp(1) = 1$

提出作法

```
for(int j = 1; j <= N; j++) {
  dp[j] = 1;
  for(int i = 1; i < j; i++)
    if(a[i] < a[j])
      dp[j] = max(dp[j], dp[i] + 1);
}
```

觀察問題

剛剛的作法只能記錄以某數結尾的 LIS 長度

但問題還需要 LIS 具體的樣子
所以需要一個方法去做到這件事

提出作法

```
for(int j = 1; j <= N; j++) {
  dp[j] = 1, f[j] = j;
  for(int i = 1; i < j; i++)
    if(a[i] < a[j] && dp[j] < dp[i] + 1)
      dp[j] = dp[i] + 1, f[j] = i;
}
```

Longest Increasing Subsequence (LIS)

那麼有了 dp 記錄以某個數為結尾是最長的 LIS 後

那只要跑過 dp 陣列後，就可以找到某數位置為何

```
int pos;
for(int i = 1, mx = 0; i <= N; i++)
   if(mx < dp[i]) mx = dp[i], pos = i;
```

Longest Increasing Subsequence (LIS)

例如 $a = (1, 4, 2, 3, 8, 3, 4, 1, 9)$
得出 $f = (1, 1, 1, 3, 4, 3, 4, 8, 5)$

假設 $a_9 = 9$ 為末項的 LIS 為例：

$f_9 = 5 \rightarrow a_5 = 8$
$f_5 = 4 \rightarrow a_4 = 3$
$f_4 = 3 \rightarrow a_3 = 2$
$f_3 = 1 \rightarrow a_1 = 1$
$f_1 = 1$

CF 1033C Permutation Game

觀察問題

從邊界觀察

當 token 為 n 那麼**該局**移動者必輸
因為沒有比 n 更大的數

觀察問題

定義 w(i) 表示該局 token 在 i 的操作者輸贏狀態

那麼假設 w(i) 為輸，
前一手只要想辦法把 token 移到 i，他就可以獲勝

提出作法

從開局 token 為 n-1 開始推敲 (已知 w(n) = 輸)

若發現 token 開局就不能移動那麼 w(i) = 輸
也就是 Alice 輸了

若發現可以移動，那就看是否能移到一個 i，其 w(i) = 輸
這樣 Alice 就可以贏

CF 429B Working out

觀察問題

又是路徑和問題
除了見面的點以外，其他都是要求最小路經和

所以沿用之前最小路徑和的作法

提出作法

```
for(int i = 1; i <= n; i++)
  for(int j = 1; j <= m; j++)
    LT_mt[i][j] = a[i][j] + max(LT_mt[i-1][j],
                                LT_mt[i][j-1]);

for(int i = n; i >= 1; i--)
  for(int j = m; j >= 1; j--)
    RB_mt[i][j] = a[i][j] + max(RB_mt[i+1][j],
                                RB_mt[i][j+1]);
```

提出作法

```
for(int i = n; i >= 1; i--)
  for(int j = 1; j <= m; j++)
    LB_mt[i][j] = a[i][j] + max(LB_mt[i+1][j],
                                LB_mt[i][j-1]);

for(int i = 1; i <= n; i++)
  for(int j = m; j >= 1; j--)
    RT_mt[i][j] = a[i][j] + max(RT_mt[i-1][j],
                                RT_mt[i][j+1]);
```

觀察問題

若 Iahub 走 $(x-1,y) \to (x,y) \to (x,y+1)$

則 Iahubina 從 (x,y) 往上或右
分別會碰到 $(x-1,y),(x,y+1)$，就不是恰好碰面一次

提出作法

```
int best = 0;
for(int i = 2; i < n; i++)
  for(int j = 2; j < m; j++)
    best = max(best,
        LT_mt[i-1][j]+RB_mt[i+1][j] +
        LB_mt[i][j-1]+RT_mt[i][j+1],
        LT_mt[i][j-1]+RB_mt[i][j+1] +
        LB_mt[i+1][j]+RT_mt[i-1][j]});
```

結語