Complexity

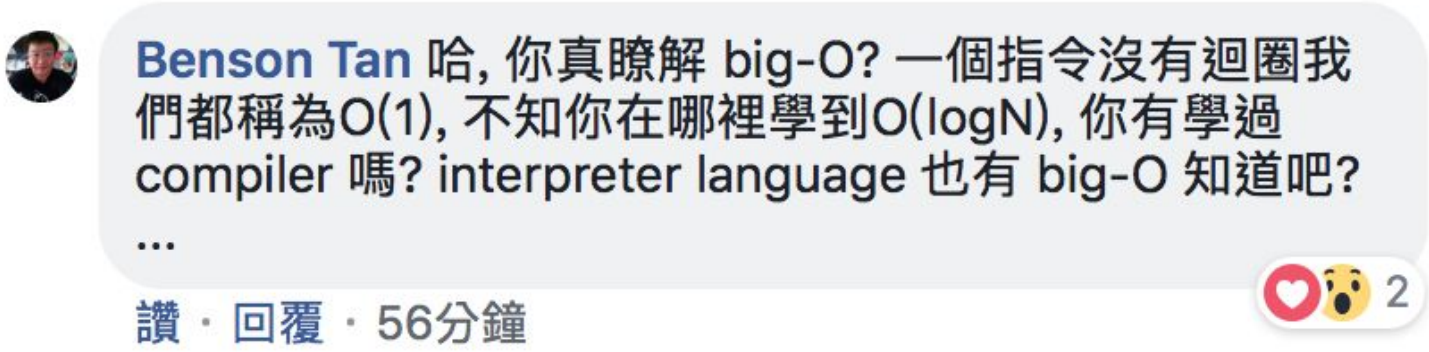
上課補充 by erichung

Credit by PixelCat

Sprout

今天要學的東西

- 今天的課以理論為主，不太會寫到題目
- 但是對於複雜度的知識絕對是寫程式最重要的一環
- 大家都怎麼算複雜度？



今天要學的東西

- 今天的課以理論為主，不太會寫到題目
- 但是對於複雜度的知識絕對是寫程式最重要的一環
- 大家都怎麼算複雜度？
- “沒有迴圈就是O(1)，一層迴圈就是O(n)，兩層就O(n^2)”

- 真的是如此嗎？

複雜度的正確算法

- 數迴圈有幾層的優缺點:
- 優點:大部分的程式我們大概都可以這樣算啦
- 缺點:
- 1.有些操作可能不是O(1)
- 2.無法計算遞迴的時間複雜度
- 3.可能會錯估均攤複雜度

複雜度的正確算法

- 首先，先來討論複雜度的正確算法
- 正式的數學定義手寫作業有，也會讓大家證一些東西
- 不過在99%的時候，複雜度都這樣算就好了：
- 計算總共需要的操作數，留下量級最大那一項，常數去掉
- ex:$O(3n^2\log n + 2n^2 + 4n + \log n) = O(n^2\log n)$

漸近複雜度

small-O	$f(n) = o(g(n))$	$g(n)$ 是 $f(n)$ 的**上界**（嚴格大於）
big-O	$f(n) = O(g(n))$	$g(n)$ 是 $f(n)$ 的**上界**（可以一樣）
big-theta	$f(n) = \Theta(g(n))$	$f(n)$ 跟 $g(n)$ 長**一樣快**
big-omega	$f(n) = \Omega(g(n))$	$g(n)$ 是 $f(n)$ 的**下界**（可以一樣）
small-omega	$f(n) = \omega(g(n))$	$g(n)$ 是 $f(n)$ 的**下界**（嚴格小於）

分析複雜度

分析複雜度

假設某些**基本操作**需要的時間都差不多

1. 計算演算法需要做幾次**基本操作**
2. 留下複雜度最大的那一項

計算複雜度相當仰賴 case by case 討論，不只是數迴圈！

分析複雜度：數迴圈（一）

```
#define rep(i, n) for(int i = 0; i < n; i++)

void mult(int n, int a[N][N], int b[N][N], int c[N][N]) {
  rep(i, n) rep(j, n) {
    c[i][j] = 0;
  }
  rep(i, n) rep(j, n) rep(k, n) {
    tmp[i][j] += (a[i][k] * b[k][j]);
  }
  rep(i, n) rep(j, n) {
    c[i][j] = tmp[i][j] % MOD;
  }
}
```

樸實的矩陣乘法

分析複雜度：數迴圈（一）

- $N^3 + 2N^2$ 次賦值
- N^3 次加法
- N^3 次乘法
- N^2 次除法（模運算）
- ??? 次陣列取值、指標和位址計算...

確切不知道，大概是 $? \times N^3 + ? \times N^2 + \ldots$ 次基本操作

分析複雜度：數迴圈（一）

- $N^3 + 2N^2$ 次賦值
- N^3 次加法
- N^3 次乘法
- N^2 次除法（模運算）
- ??? 次陣列取值、指標和位址計算...

確切不知道，大概是 $? \times N^3 + ? \times N^2 + \ldots$ 次基本操作

複雜度告訴你，N 很大的時候常數和複雜度小的項沒什麼大影響

複雜度就是妥妥的 $O(N^3)$

分析複雜度：數迴圈（二）

```
void f(int n) {
  int ans = 0;
  for(int i = 0; i < n; i++) {
    for(int j = 0; j < (1 << n); j++) {
      ans += i * j;
    }
  }
}
```

Sprout

分析複雜度：數迴圈（二）

```
int f(int n) {
  int ans = 0;
  for(int i = 0; i < n; i++) {  // 0 ... (n - 1)
    for(int j = 0; j < (1 << n); j++) {  // 0 ... (2^n - 1)
      ans += i * j;
    }
  }
  return ans;
}
```

時間複雜度：$O(n2^n)$

分析複雜度：數迴圈（三）

```
int g() {
  int ans = 0;
  for(int i = 0; i < 100; i++) {
    ans += i;
  }
  return ans;
}
```

分析複雜度：數迴圈（三）

```
int g() {
  int ans = 0;
  for(int i = 0; i < 100; i++) {
    ans += i;
  }
  return ans;
}
```

雖然有點不甘願，但是 $O(100) = O(1)$ 確實是常數時間

分析複雜度：數迴圈（四）

```
int my_lower_bound(int n, int key, int arr[]) {
  int lo = -1, hi = n;
  while(hi - lo > 1) {
    int mi = (hi + lo) / 2;
    if(arr[mi] >= key) hi = mi;
    else lo = mi;
  }
  return hi;
}
```

分析複雜度：數迴圈（四）

```
int my_lower_bound(int n, int key, int arr[]) {
  int lo = -1, hi = n;
  while(hi - lo > 1) {
    int mi = (hi + lo) / 2;
    if(arr[mi] >= key) hi = mi;
    else lo = mi;
  }
  return hi;
}
```

(hi - lo) 一開始是 $n+1$，每次都被砍一半，砍個 $\log N$ 次之後變 1 退出迴圈

二分搜尋，時間複雜度 $O(\log N)$

分析複雜度：被藏起來的複雜度

```
std::sort(a + 1, a + n + 1);
```

沒有迴圈，總共 $O(1)$ (??)

分析複雜度：被藏起來的複雜度

```
std::sort(a + 1, a + n + 1);
```

沒有迴圈，總共 $O(1)$（??）

呼叫別的函數當然需要時間，cppreference 之類的通常會告訴你各個內建函數的時間複雜度

分析複雜度：均攤分析

```
for(int idx = 0; idx < n; idx++) {
  while(stk.size() && value[stk.top()] > value[idx]) {
    ans[stk.top()] = idx;
    stk.pop();
  }
  stk.push(idx);
}
```

上週教過的單調堆疊

分析複雜度：均攤分析

```
for(int idx = 0; idx < n; idx++) {
  while(stk.size() && value[stk.top()] > value[idx]) {
    ans[stk.top()] = idx;
    stk.pop();
  }
  stk.push(idx);
}
```

for 迴圈執行 N 次

while 迴圈每一輪最多執行 N 次（stack 最多裝 N 個元素）

總複雜度是 $O(N^2)$，真的那麼糟糕嗎

分析複雜度：均攤分析

```
for(int idx = 0; idx < n; idx++) {
  while(stk.size() && value[stk.top()] > value[idx]) {
    ans[stk.top()] = idx;
    stk.pop();
  }
  stk.push(idx);
}
```

認真聽上週課程的你知道，不會有那麼多元素讓你 pop，從頭到尾 **while** 迴圈總共最多跑 N 次。時間複雜度 $O(N)$

均攤分析「偶爾會跑很慢，但是不可能每次都跑很慢，平均起來還是跑很快」

分析複雜度

複雜度分析不單純是數迴圈、還需要豐富的經驗和數學和數學和數學

分析複雜度

算完複雜度之後呢？

- 把題目給的變數範圍代進去，看看會不會超時
  - 我的電腦可以一秒跑 4×10^9 次加法
  - 綜合考量其他因素，代入複雜度後在 $10^7 \sim 10^8$ 通常算合理不超時範圍
- 有沒有複雜度差、但夠快而且好寫的作法？
- 超時了，優化演算法的哪個地方可以改進複雜度？
  - 例：少用一層迴圈？

複雜度之外的現實因素

「常數」

複雜度的計算會忽略常數

在線上評測系統，你不只要考慮演算法的複雜度，還要把他實做出來

- N 次加法和 $2N$ 次加法，誰比較快？
- N 次加法和 N 次除法，誰比較快？
- ...？

實驗一

```cpp
const int MAXN = 100'000'000;
int a[MAXN + 10];  // is assigned random value

for(int i = 1; i <= MAXN; i++) ans = ans ^ a[i];
// 0.021 s

for(int i = 1; i <= MAXN; i++) ans = ans + a[i];
// 0.022 s

for(int i = 1; i <= MAXN; i++) ans = ans * a[i];
// 0.065 s

for(int i = 1; i <= MAXN; i++) ans = (ans * a[i]) % MOD;
// 0.292 s
```

實驗二

```cpp
const int MAXN = 100'000'000;
int a[MAXN + 10];    // is assigned random value in [0, 2^16)
int ord[MAXN + 10];  // is assigned 1 ... MAXN

for(int i = 1; i <= MAXN; i++) ans += a[ord[i]];
// 0.035 s

shuffle(ord + 1, ord + MAXN + 1);
for(int i = 1; i <= MAXN; i++) ans += a[ord[i]];
// 0.758 s
```

「cache miss」

實驗三

```
const int MAXN = 10'000;
int a[MAXN + 10][MAXN + 10];  // is assigned random value in [0, 2^16)

for(int i = 1; i <= MAXN; i++)
  for(int j = 1; j <= MAXN; j++)
    ans += a[i][j];
// 0.085 s

for(int j = 1; j <= MAXN; j++)
  for(int i = 1; i <= MAXN; i++)
    ans += a[i][j];
// 1.741 s
```

也是 cache miss

實驗四

```cpp
const int MAXN = 100'000'000;
int a[MAXN + 10];  // is assigned random value in [0, 2^16)

for(int i = 1; i <= MAXN; i++) ans = ans + a[i];
// g++ main.cpp -O0
// 0.166 s

for(int i = 1; i <= MAXN; i++) ans = ans + a[i];
// g++ main.cpp -O4
// 0.045 s
```

編譯器的邪惡優化