2022 年
花蓮高中
校內賽
題解

A. 顛倒國 題解

by SorahISA

0 問題概要

輸入一行左右顛倒、只包含加減乘除的運算式。

保證計算途中只會出現整數，請輸出原本計算式的結果。

0 子任務們

- 子任務 1 (20 分)
  - 只有 1 個運算元，只會出現 +
- 子任務 2 (20 分)
  - 只有 1 個運算元，只會出現 +、−
- 子任務 3 (30 分)
  - 只有 1 個運算元
- 子任務 4 (30 分)
  - 至多有 4 個運算元

其它輸入限制：

- 輸入的數字 $0 \leq d \leq 10^6$
- 輸出的答案 $-10^6 \leq A \leq 10^6$
- 運算符號有 +、−、*、\

子任務 1-3

只有 1 個運算元

1 子任務 1-3 的解法

輸入只有一個運算元，因此算式的樣子會是 $A' \oplus B'$

分別輸入 A' 跟 B' 之後將其反轉並計算 $B \oplus A$

注意：

- 因為輸入是反著的，因此越先輸入的會在越後面
- 輸入中，除號是反斜線 \
  需要在前面加反斜線來 escape。

```cpp
string A, B;
char op;
cin >> A >> op >> B;

reverse(A.begin(), A.end());
reverse(B.begin(), B.end());

/// 計算 B op A ///

int ans = 0;

if (op == '+')  ans = B + A;
if (op == '-')  ans = B - A;
if (op == '*')  ans = B * A;
if (op == '\\') ans = B / A;

cout << ans << "\n";
```

1 子任務 1-3 的解法

輸入只有一個運算元，因此算式的樣子會是 $A' \oplus B'$

分別輸入 A' 跟 B' 之後將其反轉並計算 $B \oplus A$

Accepted (partially)

得分：70 分

注意：

- 因為輸入是反著的，因此越先輸入的會在越後面
- 輸入中，除號是反斜線 \
  需要在前面加反斜線來 escape。

```
1 string A, B;
2 char op;
3 cin >> A >> op >> B;
4
5 reverse(A.begin(), A.end());
6 reverse(B.begin(), B.end());
7
8 /// 計算 B op A ///
9
10 int ans = 0;
11
12 if (op == '+')  ans = B + A;
13 if (op == '-')  ans = B - A;
14 if (op == '*')  ans = B * A;
15 if (op == '\\') ans = B / A;
16
17 cout << ans << "\n";
```

子任務 4

至多有 4 個運算元

2 子任務 4 的解法

輸入的部分其實可以直接 `getline` 讀進來並一次 `reverse`

上課應該有學過用 stack 去處理四則運算？

不過這樣太麻煩了，直接暴力分兩次處理就好！

第一次忽略掉加跟減，只做乘除

第二次就只剩加減法了

2 子任務 4 的實作

以 stringstream 儲存計算式再依序輸入進來

儲存「上一個數字、運算元、這一個數字」並依據讀到的運算元做操作

第一遍（包含加減乘除）：

- 遇到 *、\ 就把上一個數字變成運算的結果
- 遇到 +、- 就把上一個數字跟運算元塞進其他字串暫存

第二遍（只剩加減）：

- 遇到 +、- 就把上一個數字變成運算的結果

2 子任務 4 的實作

以 stringstream 儲存計算式

儲存「L、op、R」

第一遍（包含加減乘除）：

- 遇到 *、\：運算
- 遇到 +、−：塞進其他字串暫存

第二遍（只剩加減）：

- 遇到 +、−：運算

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    string formula;
    getline(cin, formula);
    reverse(formula.begin(), formula.end());

    stringstream ss;
    string formula_add_sub, L, op, R;

    ss << formula;
    ss >> L;
    while (ss >> op >> R) {
        if (op == "+" or op == "-") formula_add_sub += L + " " + op + " ", L = R;
        else if (op == "*")  L = to_string( stoi(L) * stoi(R) );
        else if (op == "\\") L = to_string( stoi(L) / stoi(R) );
    }
    formula_add_sub += L;

    ss.clear();
    ss << formula_add_sub;
    ss >> L;
    while (ss >> op >> R) {
            if (op == "+") L = to_string( stoi(L) + stoi(R) );
        else if (op == "-") L = to_string( stoi(L) - stoi(R) );
    }

    cout << L << "\n";

    return 0;
}
```

2 子任務 4 的實作

以 stringstream 儲存計算式

儲存「L、op、R」

第一遍（包含加減乘除）：

- 遇到 *、\：運算
- 遇到 +、−：塞進其他字串暫存

第二遍（只剩加減）：

- 遇到 +、−：運算

Accepted

得分：100 分

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    string formula;
    getline(cin, formula);
    reverse(formula.begin(), formula.end());

    stringstream ss;
    string formula_add_sub, L, op, R;

    ss << formula;
    ss >> L;
    while (ss >> op >> R) {
        if (op == "+" or op == "-") formula_add_sub += L + " " + op + " ", L = R;
        else if (op == "*") L = to_string( stoi(L) * stoi(R) );
        else if (op == "\\") L = to_string( stoi(L) / stoi(R) );
    }
    formula_add_sub += L;

    ss.clear();
    ss << formula_add_sub;
    ss >> L;
    while (ss >> op >> R) {
        if (op == "+") L = to_string( stoi(L) + stoi(R) );
        else if (op == "-") L = to_string( stoi(L) - stoi(R) );
    }

    cout << L << "\n";

    return 0;
}
```

結論

3 結論

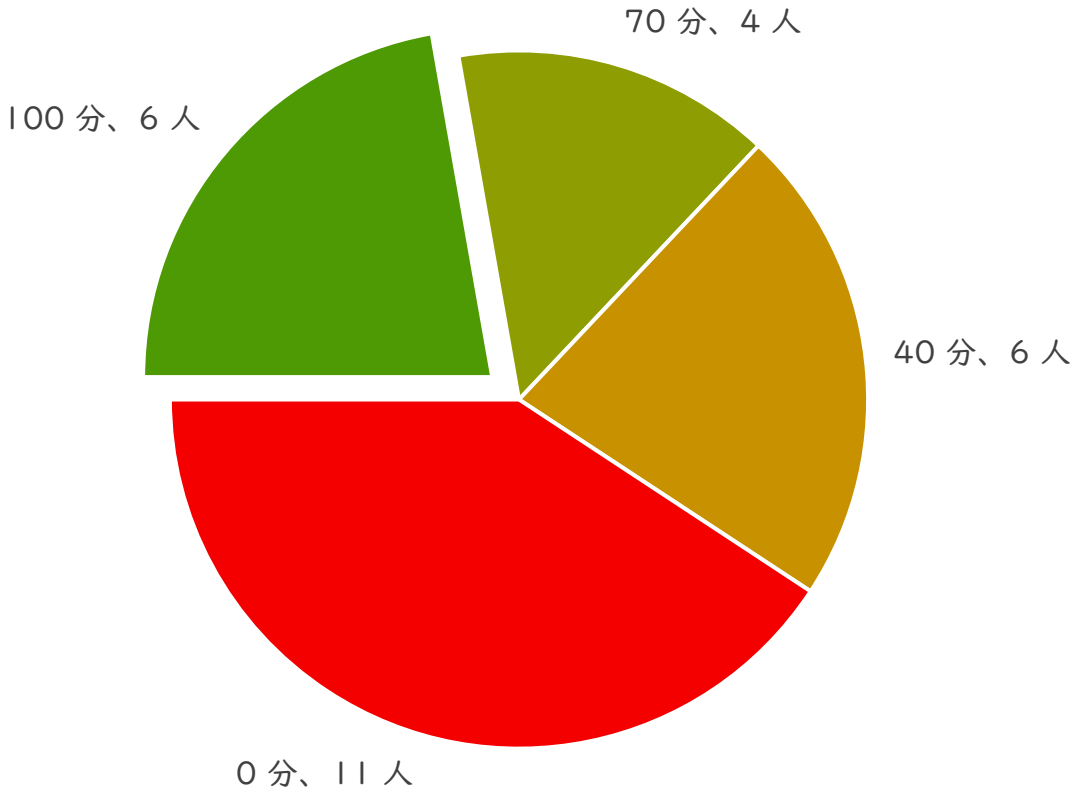
有括弧的四則運算會難上不少，不過也是可以利用遞迴 + 做兩遍來解決。

多記一些常用的方便函式！雖然多數函式都可以自己完成，但仍然是對時間跟精神的很大一筆開銷。在這裡用上了以下幾個函式：

- `getline`：讀進一整行字串
- `stringstream`：通常搭配 `getline` 一起使用，可以模擬 `cin` 跟 `cout`
- `reverse`：將指定的區間反轉
- `stoi`、`to_string`：將字串跟 `int` 互相轉換

AC Code 連結

3 成績分布



B. 密碼強度評估 題解

by SorahISA

0 問題概要

給汝 n 個字串，汝需要依規則計算它們的分數，並按照分數由大到小、字典序由大到小輸出。

規則如下：

1. 不得出現空白、長度不得 < 4。
2. 長度 ≥ 4 可以得 10 分，長度 > 8 的部分每超過一個可以多拿 2 分。
3. 「特殊字元」數量要 ≥ 3，否則分數 -6。
4. 有出現「大寫字母」、「小寫字母」、「數字字元」各加 2 分。
5. 「特殊字元」每出現 5 個就能多 10 分。
6. 「英文字母」、「數字字元」、「特殊字元」相鄰則一組加 2 分。

0 子任務們

- 子任務 1 (35 分)
  - 密碼由「英文字母」構成，輸入由小到大排好。
- 子任務 2 (35 分)
  - 可以忽略規則 6，輸入由小到大排好。
- 子任務 3 (30 分)
  - 無額外限制。

其它輸入限制：

- 密碼組數 $1 \leq n \leq 40$
- 密碼長度 $1 \leq |\text{pass}| \leq 32$
- 密碼只包含 ASCII $[32, 126]$ 的字元

満分解

1 滿分解法－規則 1、2

要讀輸入進來一樣是使用 `getline`。

規則 1【空白、長度】：應該很容易判斷。

規則 2【長度】：得到的分數是 $2 \times \max\{|\text{pass}| - 8, 0\} + 10$，或是使用 if-else 判斷。

```cpp
getline(cin, pass);

/// rule 1: |pass| >= 4 and no space ///
/// 找空白也可以使用 pass.find(' ') != string::npos ///
bool has_space = false;
for (char c : pass) if (c == ' ') has_space = true;
if ((int)pass.size() < 4 or has_space) {++cnt_0; continue;}

/// rule 2: score += 10 if |pass| >= 4; score += 2 * max(|pass| - 8, 0) ///
/// 注意 rule 1 已經判過長度, 所以 |pass| >= 4 永遠是 true ///
score += 2 * max((int)pass.size() - 8, 0) + 10;
```

1 滿分解法－字元種類判斷

剩下的四個規則都圍繞在判斷字元種類上，在這裡字元種類被分為幾種：

1. 大寫字母（isupper）：也就是 'A' <= ch && ch <= 'Z'。
2. 小寫字母（islower）：也就是 'a' <= ch && ch <= 'z'。
3. 英文字母（isalpha）：大寫或小寫，isupper(ch) || islower(ch)。
4. 數字字元（isdigit）：也就是 '0' <= ch && ch <= '9'。
5. 特殊字元（!isalnum）：並非英文也非數字，!isalpha(ch) && !isdigit(ch)。

把這些東西都寫成函式會讓 code 更精簡～

左處的函式其實都是在 <cctype> 裡面內建的函式ㄛ！

1 滿分解法－規則 3、4、5

使用變數分別紀錄「大寫字母」、「小寫字母」、「數字字元」、「特殊字元」數量。

`isextra(char)` 是自己定義的函式，方便後面統一規格。

code 寫整齊、

重複用的東西就定義成 function。

```
1  bool isextra(char c) {return !isalnum(c);}
2
3  /// 計算每種字元出現次數 ///
4  int cnt_upper = 0, cnt_lower = 0, cnt_digit = 0, cnt_extra = 0;
5  for (char c : pass) {
6      if (isupper(c)) ++cnt_upper; if (islower(c)) ++cnt_lower;
7      if (isdigit(c)) ++cnt_digit; if (isextra(c)) ++cnt_extra;
8  }
9
10 /// rule 3: score -= 6 if cnt_extra < 3 ///
11 if (cnt_extra < 3) score -= 6;
12
13 /// rule 4: type = {upper, lower, digit}, score += 2 with each type found ///
14 if (cnt_upper) score += 2;
15 if (cnt_lower) score += 2;
16 if (cnt_digit) score += 2;
17
18 /// rule 5: score += 10 * floor(cnt_extra / 5) ///
19 score += 10 * (cnt_extra / 5);
```

1 滿分解法－規則 6

判斷相鄰兩個字元是否為相同組別。

只有三種 case：當前字元為「組別 X」且上一字元不為「組別 X」。

這三種 case 互斥且包含了所有可能性。

```
1 /// rule 6: each alternate {alpha, digit, extra} let score += 2 ///
2 for (int i = 1; i < (int)pass.size(); ++i) {
3     if (
4             (isalpha(pass[i]) and !isalpha(pass[i-1]))
5         or  (isdigit(pass[i]) and !isdigit(pass[i-1]))
6         or  (isextra(pass[i]) and !isextra(pass[i-1]))
7     ) score += 2;
8 }
```

1 滿分解法－排序

計算完每個字串的分數，就剩下對這些字串進行排序了！

比較好的實作方法是用一個 **pair<int, string>** 的陣列包住 {score, pass}。

std::string 的預設排序就是按照字典序排序，

所以只要將這個陣列由大到小排序就能 AC 了！

```
1 /// 方法一：sort 完再 reverse ///
2 sort(pwd.begin(), pwd.end());
3 reverse(pwd.begin(), pwd.end());
4
5 /// 方法二：用大到小的方法排序 ///
6 sort(pwd.begin(), pwd.end(), greater<>());
7
8 /// 方法三：讓 sort 看到的陣列是反過來的 ///
9 sort(pwd.rbegin(), pwd.rend());
```

1 滿分解法－排序

計算完每個字串的分數，就剩下對這些字串進行排序了！

比較好的實作方法是用 pair<int, string> 的陣列包住 {score, pass}。

Accepted

std::string 的預設排序就是按照字典序排序，

所以只要將這個陣列由大到小排序就能 AC 了！

得分：100 分

```
/// 方法一：sort 完再 reverse ///
sort(pwd.begin(), pwd.end());
reverse(pwd.begin(), pwd.end());

/// 方法二：用大到小的方法排序 ///
sort(pwd.begin(), pwd.end(), greater<>());

/// 方法三：讓 sort 看到的陣列是反過來的 ///
sort(pwd.rbegin(), pwd.rend());
```

結論

2 結論

實作題常常會重複用到一些判斷式，這時候寫成 function 重複呼叫會讓 code 更好理解。

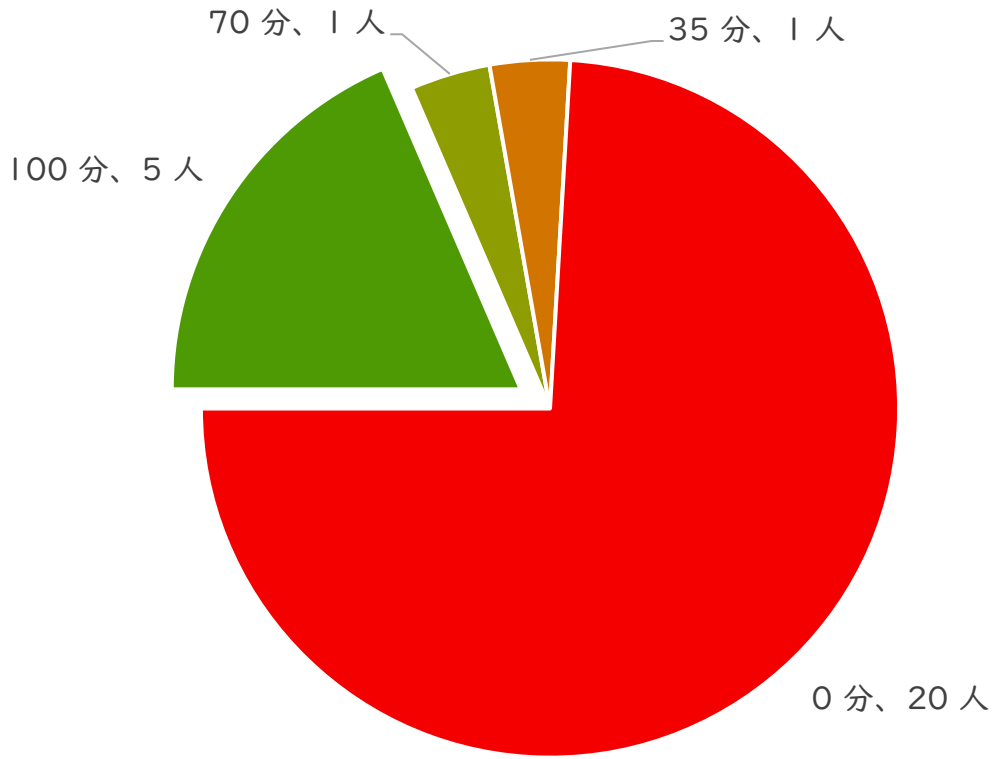
不同的規則分開判斷、加上註解。

如果不放心自己的實作能力，可以利用子題來驗證：

- 子題 1 只會有規則 1、2、4
- 子題 2 只會有規則 1、2、3、4、5

AC Code 連結

2 成績分布



C. 討論室 題解

by SorahISA

0 問題概要

現在有 n 個區間（時段）$[S, E)$ 要被放到 k 條數線（討論室）上，每個區間可以：

1. 被放在某條數線上，他必須跟該數線上的所有區間都沒有交集。
2. 不放在任何數線上。

請問最多可以放幾個區間？

0 子任務們

- 子任務 1 (20 分)
  - $n \leq 10$、$k = 1$。
- 子任務 2 (30 分)
  - $n \leq 100$、$k \leq 5$。
- 子任務 3 (50 分)
  - $n \leq 1\,000\,000$、$k \leq 100$。

其它輸入限制：

- 區間數量 $1 \leq n \leq 1\,000\,000$
- 數線數量 $1 \leq k \leq 100$
- 區間端點 $8 \leq S < E \leq 21$

子任務 1

$n \leq 10$、$k = 1$

1 子任務 1 的暴力解法

因為 n 只到 10，那要不要試試看暴力解？

每個區間只會有「選」或是「不選」兩種狀態，可以透過遞迴來枚舉。

紀錄下每個時段有沒有被借過，
如果當前區間內有時段被借走了就只能放棄。

遞迴下去之前跟之後記得要把該時段設為
「有人借了」跟「借的人取消預約了」。

時間複雜度是 $\mathcal{O}(2^n) \times \mathcal{O}(C)$，分別是
枚舉跟檢查的複雜度，其中 $C = 14$ 是值域。

```cpp
int ans = 0, N, K;
vector<pair<int, int>> rec;
vector<bool> have(24, false);

void dfs(int now, int id) {
    if (id == N) {ans = max(ans, now); return;}
    auto [L, R] = rec[id];

    int can_put = true;
    for (int t = L; t < R; ++t) if (have[t]) can_put = false;

    if (can_put) {
        for (int t = L; t < R; ++t) have[t] = true;
        dfs(now+1, id+1);
        for (int t = L; t < R; ++t) have[t] = false;
    }
    dfs(now, id+1);
}
```

1 子任務 1 的暴力解法

因為 n 只到 10，那要不要試試看暴力解？

每個區間只會有「選」或是「不選」兩種狀態，可以透過遞迴來做。

(partially) Accepted

紀錄下每個時段有沒有被借過，
如果當前區間內有時段被借走了就只能放棄。

遞迴下去之前跟之後記得要把時段設為
「有人借了」跟「借的人取消預約了」。

時間複雜度是 $O(2^n) \times O(C)$，分別是
枚舉跟檢查的複雜度，其中 $C = 14$ 是值域。

得分：20 分

```
1  int ans = 0, N, K;
2  vector<pair<int, int>> rec;
3  vector<bool> have(24, false);
4
5  void dfs(int now, int id) {
6      if (id == K) {ans = max(ans, now); return;}
7      auto [L, R] = rec[id];
8
9      int can_put = true;
10     for (int t = L; t < R; ++t) if (have[t]) can_put = false;
11
12     if (can_put) {
13         for (int t = L; t < R; ++t) have[t] = true;
14         dfs(now+1, id+1);
15         for (int t = L; t < R; ++t) have[t] = false;
16     }
17     dfs(now, id+1);
18 }
```

1 子任務 1 的 greedy 解法

「最大不相交覆蓋」是經典的 greedy 題。

做法是先對區間的右界排序，再依序能取就取。

不過，greedy 最重要的是如何證明正確性！

如果證明不出正確性那它就只是一種虎爛而已 QQ

1 子任務 1 的 greedy 證明

Claim. 先對區間的右界排序，再依序能取就取，會得到最佳解。

Proof.

不失公正性的先將所有區間以右界排序，使 $R_1 \le R_2 \le \cdots \le R_n$，並令 $i_1 < i_2 < \cdots < i_m$ 為其中一組最佳解 S。

對於所有 $i'_k \le i_k$，都有 $R_{i'_k} \le R_{i_k} \le L_{i_{k+1}} < L_{i_{k+2}} < \cdots < L_{i_m}$，也就是後面的區間不會被現在的解所影響到，所以將 i_1 變成 1 一定還滿足區間不相交的性質。

我們也可以找到一個最小的 $i'_2 \le i_2$ 使 $L_{i'_2} \ge R_{i_1}$（$i'_2 = i_2$ 本身也是一組解）。

以此類推，每次將 i_k 往左移動時都仍保證還是一組合法解，而且不可能限制到後面的區間選法。

這代表對於任何一組最佳解 S 我們都可以將其改為 greedy 的解 S'，並滿足 $|S| = |S'|$。■

1 子任務 1 的 greedy 實作

要對右界排序，簡單的實作方法就是使用 pair<int, int> 來儲存區間。

讓 p.first 存右界、p.second 存左界，就可以直接呼叫 sort 而不用自己寫比較函式！

```cpp
vector<pair<int, int>> rec(N);
for (auto &[r, l] : rec) cin >> l >> r;
sort(rec.begin(), rec.end());

int ans = 0, lst = 0;
for (auto [r, l] : rec) {
    /// 直接選遇到的第一個沒有相交的區間 ///
    if (l >= lst) lst = r, ++ans;
}
cout << ans << "\n";
```

1 子任務 1 的 greedy 實作

要對右界排序，簡單的實作方法就是使用 pair<int, int> 來儲存區間。

讓 p.first 存右界、p.second 存左界，就可以直接呼叫 sort 而不用自己寫比較函式！

(partially) Accepted

得分：20 分

```
1 vector<pair<int, int>> rec(n);
2 for (auto &[r, l] : rec) cin >> l >> r;
3 sort(rec.begin(), rec.end());
4
5 int ans = 0, lst = 0;
6 for (auto [r, l] : rec) {
7     /// 直接選遇到的第一個沒有相交的區間 ///
8     if (l >= lst) lst = r, ++ans;
9 }
10 cout << ans << "\n";
```

子任務 2

$n \leq 100$、$k \leq 5$

2 子任務 2 的 DP 解法

從剛剛的 greedy 出發，有一個重要的觀察是：

如果將區間對右界排序，那麼一間討論室的狀態會從 2^{13} 種降低到 14 種。

從「每個時間段有沒有被借走」變成「最後被借走的時間」。

所以，如果把每間討論室被借用的狀態列出來，就會發現只有 $14^k = 14^5 = 537\,824$ 種狀態。

於是我們可以考慮看看使用動態規劃（DP）來解決這個子問題。

2 子任務 2 的 DP 解法

只有 $14^5 = 537\,824$ 種狀態，考慮使用 DP 來解決。

令 $\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5]$ 代表前 i 個區間、五間討論室可以開放使用的時間。

如果第一間討論室在借給第 i 個區間 $[L, R)$ 之後，在時間 t_1 時仍可以使用，那麼就會有

$$\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5] \leftarrow \mathrm{dp}[i-1][L][t_2][t_3][t_4][t_5] + 1, \qquad \textbf{if } R \le t_1$$

註：這裡的 $a \leftarrow b$ 是取 max（$a = \max\{a, b\}$）的意思。

意思是：要借給第 i 個區間我們只能考慮在時間 L 就能使用的狀態。

對第二、三、四、五間討論室亦同。

2 子任務 2 的 DP 解法

令 $\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5]$ 代表前 i 個區間、五間討論室可以開放使用的時間。

轉移式 1-5：$\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5] \leftarrow \mathrm{dp}[i-1][L][t_2][t_3][t_4][t_5] + 1,\ \textbf{if}\ R \le t_1$

若是在 $(t_1, t_2, t_3, t_4, t_5)$ 時選擇不借，則會是跟上一時間取 max：

$$\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5] \leftarrow \mathrm{dp}[i-1][t_1-1][t_2][t_3][t_4][t_5]$$

一樣可以從五間討論室來轉移。

2 子任務 2 的 DP 解法

令 $\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5]$ 代表前 i 個區間、五間討論室可以開放使用的時間。

轉移式 1-5：$\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5] \leftarrow \mathrm{dp}[i-1][L][t_2][t_3][t_4][t_5] + 1,\ \textbf{if}\ R \le t_1$

轉移式 6-10：$\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5] \leftarrow \mathrm{dp}[i-1][t_1-1][t_2][t_3][t_4][t_5]$

空間有點大，雖然應該還是開的下，不過可以使用滾動 DP 來把第一維的 n 倍空間壓掉。

在轉移式 1-5 要從大到小轉移、轉移式 6-10 要從小到大轉移。

詳細的部分可以參考 code。

2 子任務 2 的 DP 解法

令 $\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5]$ 代表前 i 個區間、五間討論室可以開放使用的時間。

轉移式 1-5：$\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5] \leftarrow \mathrm{dp}[i-1][L][t_2][t_3][t_4][t_5] + 1,\ \textbf{if}\ R \le t_1$

轉移式 6-10：$\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5] \leftarrow \mathrm{dp}[i-1][t_1-1][t_2][t_3][t_4][t_5]$

```cpp
vector<pair<int, int>> itv(N);
for (auto &[L, R] : itv) cin >> L >> R, L -= 8, R -= 8;
sort(itv.begin(), itv.end());

int dp[14][14][14][14][14];
memset(dp, 0x00, sizeof(dp));

for (auto [L, R] : itv) { /* 轉移放在下一頁 */ }
```

```cpp
if (K == 1) cout << dp[13][ 0][ 0][ 0][ 0] << "\n";
if (K == 2) cout << dp[13][13][ 0][ 0][ 0] << "\n";
if (K == 3) cout << dp[13][13][13][ 0][ 0] << "\n";
if (K == 4) cout << dp[13][13][13][13][ 0] << "\n";
if (K == 5) cout << dp[13][13][13][13][13] << "\n";
```

2 子任務 2 的 DP 解法

轉移式 1-5：$\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5] \leftarrow \mathrm{dp}[i-1][L][t_2][t_3][t_4][t_5] + 1,\ \textbf{if}\ R \le t_1$

轉移式 6-10：$\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5] \leftarrow \mathrm{dp}[i-1][t_1-1][t_2][t_3][t_4][t_5]$

```
for (int c1 = 13; c1 >= 0; --c1) for (int c2 = 13; c2 >= 0; --c2) for (int c3 = 13; c3 >= 0; --c3)
for (int c4 = 13; c4 >= 0; --c4) for (int c5 = 13; c5 >= 0; --c5) {
    dp[c1][c2][c3][c4][c5] = max({
        dp[c1][c2][c3][c4][c5],                    (c1 >= R ? dp[ L][c2][c3][c4][c5] + 1 : 0),
        (c2 >= R ? dp[c1][ L][c3][c4][c5] + 1 : 0), (c3 >= R ? dp[c1][c2][ L][c4][c5] + 1 : 0),
        (c4 >= R ? dp[c1][c2][c3][ L][c5] + 1 : 0), (c5 >= R ? dp[c1][c2][c3][c4][ L] + 1 : 0),
    }); /// put in 1-5
}

for (int c5 = 0; c5 <= 13; ++c5) for (int c4 = 0; c4 <= 13; ++c4) for (int c3 = 0; c3 <= 13; ++c3)
for (int c2 = 0; c2 <= 13; ++c2) for (int c1 = 0; c1 <= 13; ++c1) {
    dp[c1][c2][c3][c4][c5] = max({
        dp[c1][c2][c3][c4][c5],              (c1 ? dp[c1-1][c2][c3][c4][c5] : 0),
        (c2 ? dp[c1][c2-1][c3][c4][c5] : 0), (c3 ? dp[c1][c2][c3-1][c4][c5] : 0),
        (c4 ? dp[c1][c2][c3][c4-1][c5] : 0), (c5 ? dp[c1][c2][c3][c4][c5-1] : 0),
    }); /// do nothing
}
```

2 子任務 2 的 DP 解法

轉移式 1-5：$\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5] \leftarrow \mathrm{dp}[i-1][L][t_2][t_3][t_4][t_5] + 1,\ \textbf{if}\ R \le t_1$

轉移式 6-10：$\mathrm{dp}[i][t_1][t_2][t_3][t_4][t_5] \leftarrow \mathrm{dp}[i-1][t_1-1][t_2][t_3][t_4][t_5]$

(partially) Accepted

得分：50分

```
for (int c1 = 13; c1 >= 0; --c1) for (int c2 = 13; c2 >= 0; --c2) for (int c3 = 13; c3 >= 0; --c3)
for (int c4 = 13; c4 >= 0; --c4) for (int c5 = 13; c5 >= 0; --c5) {
    dp[c1][c2][c3][c4][c5] = max({
        dp[c1][c2][c3][c4][c5],                 (c1 >= R ? dp[ L][c2][c3][c4][c5] + 1 : 0),
        (c2 >= R ? dp[c1][ L][c3][c4][c5] + 1 : 0), (c3 >= R ? dp[c1][c2][ L][c4][c5] + 1 : 0),
        (c4 >= R ? dp[c1][c2][c3][ L][c5] + 1 : 0), (c5 >= R ? dp[c1][c2][c3][c4][ L] + 1 : 0),
    }); /// put in 1-5
}

for (int c5 = 0; c5 <= 13; ++c5) for (int c4 = 0; c4 <= 13; ++c4) for (int c3 = 0; c3 <= 13; ++c3)
for (int c2 = 0; c2 <= 13; ++c2) for (int c1 = 0; c1 <= 13; ++c1) {
    dp[c1][c2][c3][c4][c5] = max({
        dp[c1][c2][c3][c4][c5],                 (c1 ? dp[c1-1][c2][c3][c4][c5] : 0),
        (c2 ? dp[c1][c2-1][c3][c4][c5] : 0), (c3 ? dp[c1][c2][c3-1][c4][c5] : 0),
        (c4 ? dp[c1][c2][c3][c4-1][c5] : 0), (c5 ? dp[c1][c2][c3][c4][c5-1] : 0),
    }); /// do nothing
}
```

2 子任務 2 的 DP 加速

其實有很多狀態是重複的，例如 $(3, 11, 7, 3, 6)$ 跟 $(11, 7, 6, 3, 3)$ 就是同一種狀態。

所以我們可以只考慮 $t_1 \geq t_2 \geq t_3 \geq t_4 \geq t_5$ 的狀態，而這樣的狀態總共有 $\binom{19}{5} = 11\,628$ 種。

時間複雜度是 $\mathcal{O}\left(n \times \frac{C^k}{k!}\right) \times \mathcal{O}(k)$，其中值域 $C = 14$。

這兩者分別是狀態跟轉移的複雜度。此處假設 $\binom{C}{k} \approx \frac{C^k}{k!}$。

空間複雜度是 $\mathcal{O}(C^k)$。

子任務 3

$n \leq 1\,000\,000$、$k \leq 100$

3 滿分的 greedy 解法

回到正題，這題的滿分解就是使用 greedy。

若一個區間 $[L, R)$ 可以被借用，就代表存在一間討論室上一次歸還的時間 $\leq L$。

跟 $k = 1$ 的情況只有一點點差別！

假設每間討論室可以開放使用的時間由小至大是 $t_1, t_2, t_3, \ldots, t_n$。

那要借的討論室就最好是在 $t_i \leq L$ 的討論室之間選擇 t_i 最大的（可以嘗試證明看看）。

實作上可以使用一個 `multiset` 維護小至大的 t_i、用 `upper_bound` 來找到目標數字。

這方法聽起來好到不真實，但是要怎麼證明它是對的？

3 滿分的 greedy 證明

Claim. 先對區間的 $(r_i, -l_i)$ 排序，再依序能取就取，會得到最佳解。

Proof.

定義大小相同的多重集合 s 被 t 支配（$s \preccurlyeq t$），僅當將 s 跟 t 內的元素排序後每一項 $t_i \leq s_i$。

不失公正性的先將所有區間以 $(r_i, -l_i)$ 排序，對於一組 $\{取, 不取\}^k$ 的決策，前面已經給出了一種最佳的擺放方式，於是這裡也只需要在意每個區間到底要不要取。

考慮前 $i-1$ 個區間中最末尾一段不取的區間們，把「取區間 i」改成「取區間 j」有三種 case：

- 放不下：那就不要管它。
- $r_j < r_i$ 右界向左移動：取 (l_j, r_j) 的序列一定被 (l_i, r_i) 的序列支配。
- $r_j = r_i$、$l_j > l_i$ 右界不變左界遞增：顯然只要不變位置就至少一樣好了。

3 滿分的 greedy 證明

Claim. $r_j < r_i$ 右界向左移動：取 (l_j, r_j) 的序列一定被 (l_i, r_i) 的序列支配。

Proof.

如果 $l_j \geq l_i$ 那顯然至少可以維持原樣，以下考慮 $l_j < l_i$ 的情形。

令還沒有選區間 i 或 j 時的序列是 $s = \{s_1, s_2, \dots, s_{k-1}, s_k\}$。

選區間 i，把 s 中 $\leq l_i$ 的最大值（令其為 s_a）換成 r_i：

$$S_i = \{s_1, \dots, s_{a-1}, r_i, s_{a+1}, \dots, s_k\} = \{r_i, s_1, \dots, s_{a-1}, s_{a+1}, \dots, s_k\}$$

選區間 j，把 s 中 $\leq l_j$ 的最大值（令其為 s_b）換成 r_j：

$$S_j = \{s_1, \dots, s_{b-1}, r_j, s_{b+1}, \dots, s_k\} = \{r_j, s_1, \dots, s_{b-1}, s_{b+1}, \dots, s_k\}$$

若 $a = b$ 則顯然，以下考慮 $a < b$。先將位置相同的項 $s_1, s_2, \dots, s_{a-1}$、$s_{b+1}, s_{b+2}, \dots, s_k$ 忽略掉：

$$\{r_i, s_{a+1}, s_{a+2}, \dots, s_{b-1}, s_b\} \preccurlyeq \{r_j, s_a, s_{a+1}, \dots, s_{b-2}, s_{b-1}\}$$

因為 $\{s_{i+1}\} \preccurlyeq \{s_i\}$ 且 $r_i > r_j$，所以 $S_i \preccurlyeq S_j$。 ■

3 滿分的 greedy 證明

Claim. $r_j < r_i$ 右界向左移動：取 (l_j, r_j) 的序列一定被 (l_i, r_i) 的序列支配。

Proof.

如果 $l_j \geq l_i$ 那顯然至少可以維持原樣，以下考慮 $l_j < l_i$ 的情形。

令還沒有選區間 i 或 j 時的序列是 $s = \{s_1, s_2, \dots, s_{k-1}, s_k\}$。

選區間 i，把 s 中 $\leq l_i$ 的最大值（令其為 s_a）換成 r_i：

$$S_i = \{s_1, \dots, s_{a-1}, r_i, s_{a+1}, \dots, s_k\} = \{r_i, s_1, \dots, s_{a-1}, s_{a+1}, \dots, s_k\}$$

選區間 j，把 s 中 $\leq l_j$ 的最大值（令其為 s_b）換成 r_j：

$$S_j = \{s_1, \dots, s_{b-1}, r_j, s_{b+1}, \dots, s_k\} = \{r_j, s_1, \dots, s_{b-1}, s_{b+1}, \dots, s_k\}$$

若 $a = b$ 則顯然，以下考慮 $a < b$。先將位置相同的項 $s_1, s_2, \dots, s_{a-1}$、$s_{b+1}, s_{b+2}, \dots, s_k$ 忽略掉：

$$\{r_i, s_{a+1}, s_{a+2}, \dots, s_{b-1}, s_b\} \preccurlyeq \{r_j, s_a, s_{a+1}, \dots, s_{b-2}, s_{b-1}\}$$

因為 $\{s_{i+1}\} \preccurlyeq \{s_i\}$ 且 $r_i > r_j$，所以 $S_i \preccurlyeq S_j$。 ■

Accepted

得分：100 分

3 滿分的 greedy 證明

證明的沒有很仔細 QQ。

還有一些細節請思考看看，網路上應該也有許多寫的更好的證明方法。

結論

4 結論

greedy 不能解決所有題目（ex. 背包問題），於是最重要的就是證明他的正確性。

而常見的證明方法有幾種：

1. 對任一組最佳解 S，都可以變成用 greedy 構出來的最佳解 S'。
2. 對不滿足貪心的解 T，都可以找到更好的解 T'。
3. 歸納法：先證明邊界情況 F_1 是最佳解，再證明 F_n 可以由 $F_1, F_2, \ldots, F_{n-1}$ 推導出。

greedy 常見的樣子有兩種，分別是「對 XXX 排序」以及「每次取最大的 YYY」。

而他跟動態規劃最大的差別就是，greedy 會直接選擇，不能反悔；而 DP 會保留以前的結果。

4 結論

不過，因為比賽中 greedy「通常」都很好寫，於是會有許多人使用「Proof by AC」的方法。

簡單來說，就是直接寫、直接傳、直接 AC。拿到 AC 就代表汝的想法是對的。

（但也有可能是測資沒出好，2020 年的 TOI 初選 pB 就有相同的狀況發生。）

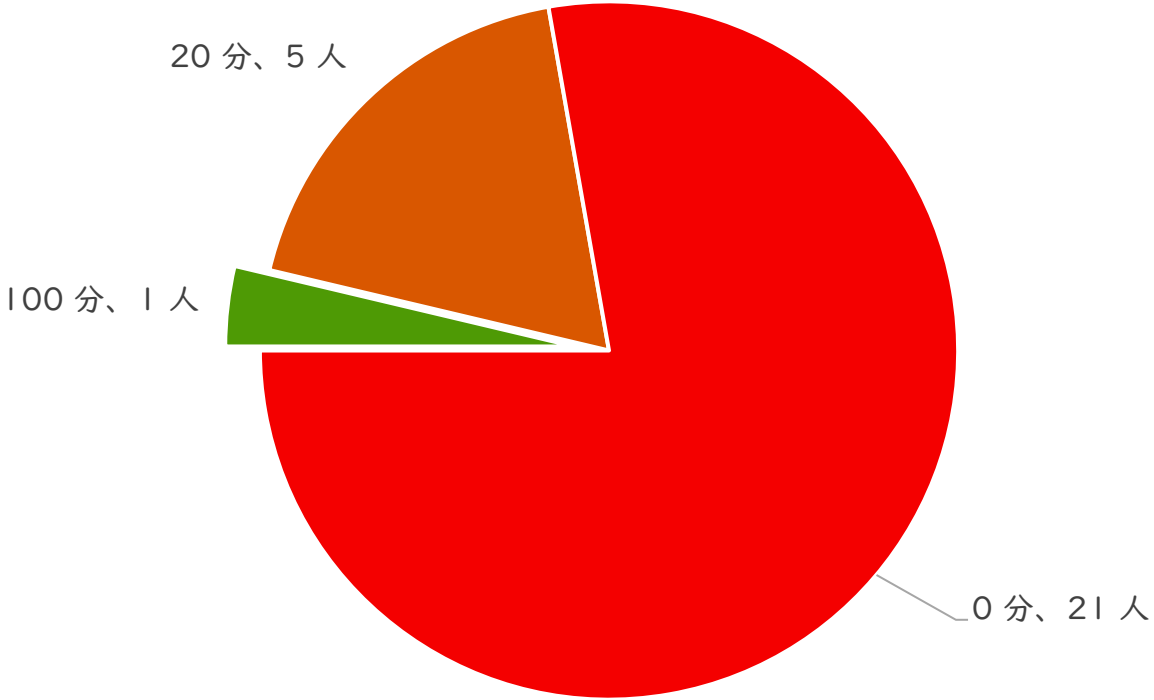
在練習的時候如果汝 claim 了一個 greedy 作法，請嘗試看看證明他ㄛ！

通常題解也都會附上 greedy 的證明方法。

Subtask 2

AC Code 連結

4 成績分布

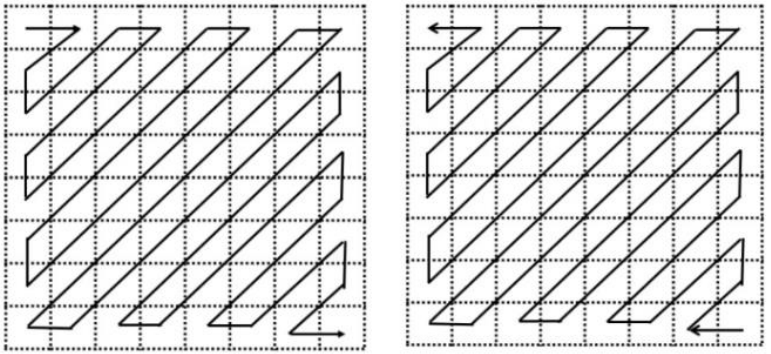


D. ZZPooling 密碼池產生器 題解

by SorahISA

0 問題概要

給汝一個長度 n^2 的字串，汝要將其按照順序填進 $n \times n$ 表格後
以下圖方式來讀取成第二個字串。



a Forward direction　　**b** Reverse direction

接下來汝要把這個表格切成 $\left(\frac{n}{d}\right)^2$ 個 $d \times d$ 的子矩陣，
並對每個子矩陣求出 ASCII Code 的「和」或
「平均」值。

最後，將這 $\left(\frac{n}{d}\right)^2$ 個字元輸出。

0 子任務們

- 子任務 1 (15 分)
  - $n = 8$、$d = 2$、Zigzag = "a"、Pooling = "m"。
- 子任務 2 (25 分)
  - $n = 8$、$d = 2$、Zigzag = "b"。
- 子任務 3 (30 分)
  - $n \leq 64$。
- 子任務 4 (30 分)
  - $n \leq 1024$。

其它輸入限制：

- Zigzag = "a" or "b"
- Pooling = "m" or "a"
- d 是 n 的因數
- 字串只包含 ASCII $[33, 126]$ 的字元

満分解

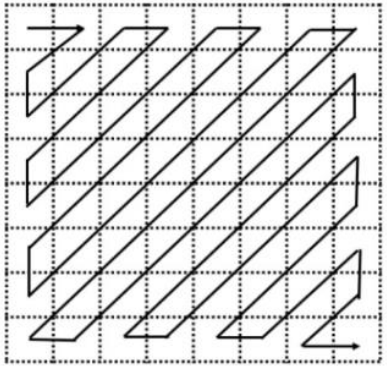
1 滿分解法－輸入

先把輸入讀進字串裡。

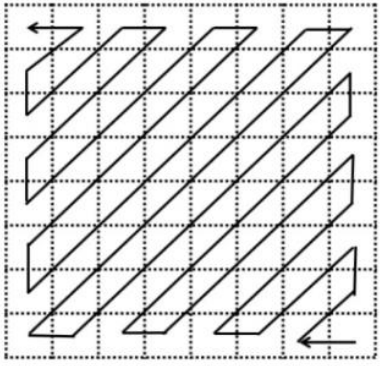
有沒有發現其實「方案 a」跟「方案 b」只是讀取的方向相反？

如果填表是用「方案 b」就直接 reverse 讀進來的字串！

這樣就解決掉其中一個限制了～



a Forward direction



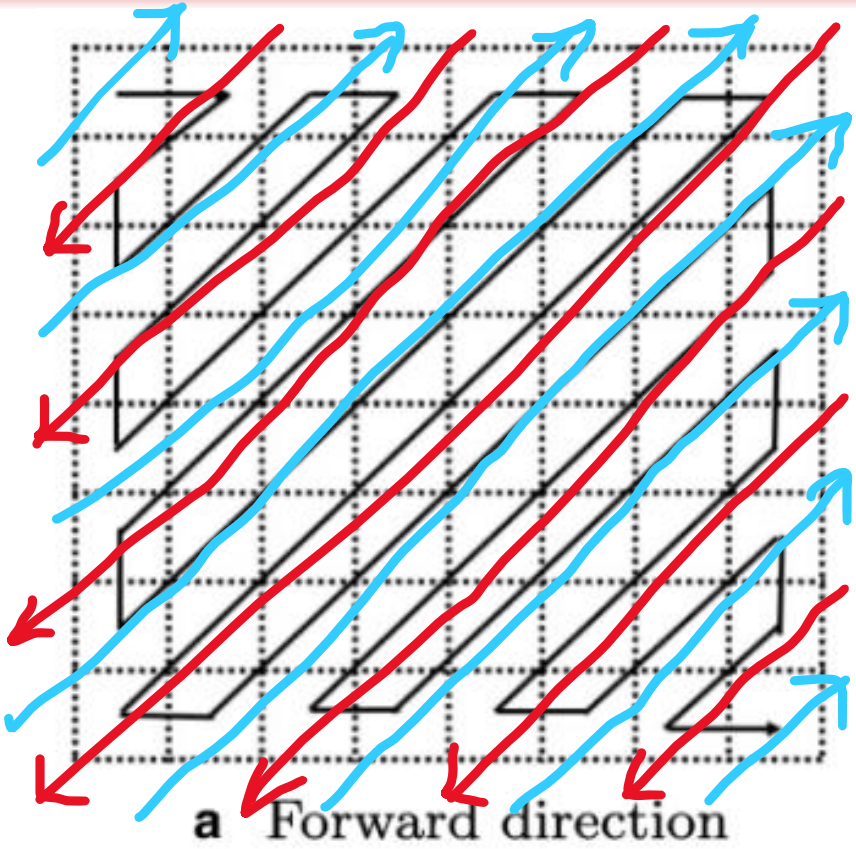
b Reverse direction

1 滿分解法－掃描

掃描的方向是依反對角線左上到右下，並且

在 $r+c$ 是偶數的情況下是由左下到右上、

在 $r+c$ 是奇數的情況下是由右上到左下。



1 滿分解法－掃描實作

掃描的方向是依反對角線左上到右下：讓 $\text{sum} = r + c$ 由 0 走到 $2(N-1)$。

在 sum 是偶數的情況下是由左下到右上：從 $c = 0$ 走到 $r = 0$。

在 sum 是奇數的情況下是由右上到左下：從 $r = 0$ 走到 $c = 0$。

這裡的計算次數大約是 $\binom{2n}{2}$，比好好填慢約一倍。

當然，汝也可以計算出精確的起點跟終點，

此處是用還可以接受的效率換取實作的時間。

P.S.：精確的起點是 $\max(\text{sum} - N + 1, 0)$。

```cpp
bool valid(int r, int c) {
    return (0 <= r and r < N and 0 <= c and c < N);
};

string zigstr;
for (int sum = 0; sum <= 2*N-2; ++sum) {
    if (sum % 2 == 0) {
        for (int c = 0, r = sum; r >= 0; ++c, --r)
            if (valid(r, c)) zigstr += board[r][c];
    }
    else {
        for (int r = 0, c = sum; c >= 0; ++r, --c)
            if (valid(r, c)) zigstr += board[r][c];
    }
}
```

1 滿分解法－池化

這部分很單純，就是一塊一塊的掃描過矩陣。

看是要把答案存到對應的區塊（如右）、

還是要直接算完一塊的答案（如下）都可以，

此處提供這兩種寫法供參考。

```cpp
int ans[N/d][N/d] = {};
for (int i = 0; i < N*N; ++i) {
    /// 因為重複寫這個位置 code 會充滿算式，所以先用 now 代替 ///
    int &now = ans[(i/N) / d][(i%N) / d];
    if (pooling == 'm') now = max(now, (int)zigstr[i]);
    if (pooling == 'a') now += (int)zigstr[i];
}

for (int i = 0; i < N/d; ++i) {
    for (int j = 0; j < N/d; ++j) {
        if (pooling == 'a') ans[i][j] /= d * d;
        cout << (char)ans[i][j];
    }
}
```

```cpp
for (int i = 0; i < N; i += d) for (int j = 0; j < N; j += d) {
    /// 依序掃描每一塊，當前 row = i+x 且 col = j+y ///
    int ans = 0;
    for (int x = 0; x < d; ++x) for (int y = 0; y < d; ++y) {
        if (pooling == 'm') ans = max(ans, (int)zigstr[(i+x)*N + (j+y)]);
        if (pooling == 'a') ans += (int)zigstr[(i+x)*N + (j+y)];
    }
    if (pooling == 'a') ans /= d * d;
    cout << (char)ans;
}
```

1 滿分解法－池化

這部分很單純，就是一塊一塊的掃描過矩陣。

看是要把答案存到對應的區塊（如右）、

還是要直接算完一塊的答案（如下）都可以，

此處提供這兩種寫法供參考。

Accepted

得分：100 分

```
1 int ans[N/d][N/d] = {};
2 for (int i = 0; i < N*N; ++i) {
3     /// 依序掃描每一塊，當前 row = i/N 且 col = i%N ///
4     int &now = ans[(i/N) / d][(i%N) / d];
5     if (pooling == 'm') now = max(now, (int)zigstr[i]);
6     if (pooling == 'a') now += (int)zigstr[i];
7 }
8
9 for (int i = 0; i < N/d; ++i) {
10     for (int j = 0; j < N/d; ++j) {
11         if (pooling == 'a') ans[i][j] /= d * d;
12         cout << (char)ans[i][j];
13     }
14 }
```

```
1 for (int i = 0; i < N; i += d) for (int j = 0; j < N; j += d) {
2     /// 依序掃描每一塊，當前 row = i+x 且 col = j+y ///
3     int ans = 0;
4     for (int x = 0; x < d; ++x) for (int y = 0; y < d; ++y) {
5         if (pooling == 'm') ans = max(ans, (int)zigstr[(i+x)*N + (j+y)]);
6         if (pooling == 'a') ans += (int)zigstr[(i+x)*N + (j+y)];
7     }
8     if (pooling == 'a') ans /= d * d;
9     cout << (char)ans;
10 }
```

結論

2 結論

競賽中，題目通常不會特別去要求微小的常數。

- 大多數比賽中都有規定 Time Limit 至少要是官解執行時間的兩倍。

往往在不影響複雜度的前提下寫的簡潔一點會有利於 debug ㄛ～

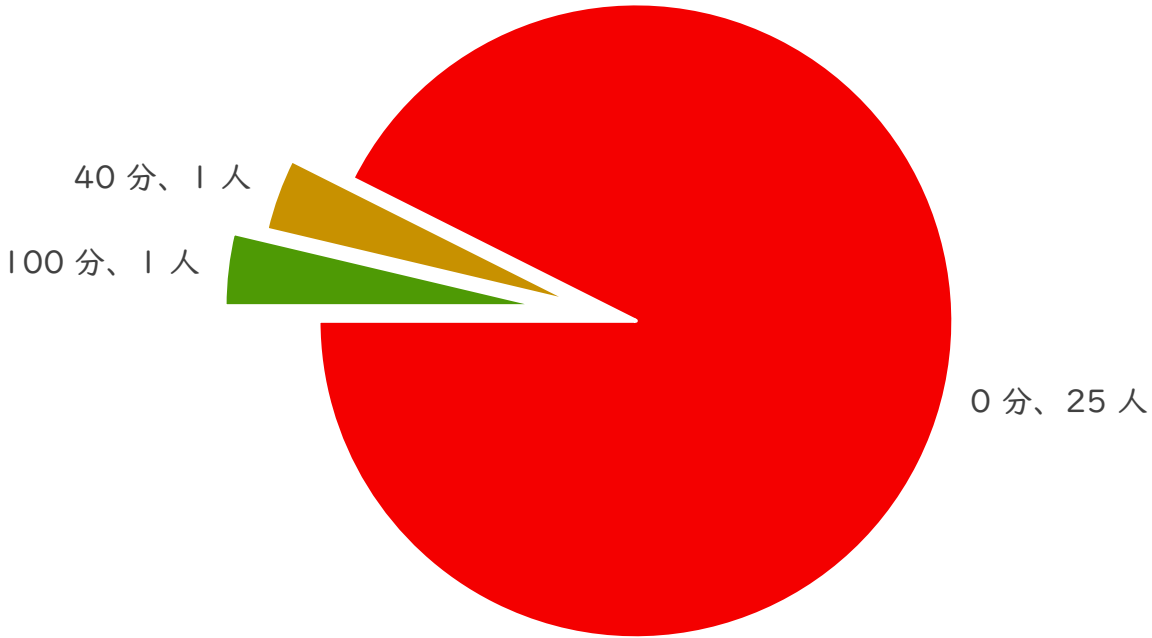
先在紙上畫一畫、多想一些實作細節也能讓 code 變的可看許多。

如果汝發現有地方會讓汝不斷複製貼上，那多半是有地方沒想清楚、條件沒看清楚。

畢竟出題者跟驗題者也都不會想寫那種東西（X）。

AC Code 連結

2 成績分布



E. 方格迷宮 題解

by SorahISA

0 問題概要

有一個 $N \times M$ 的矩陣，每個位置上有「業力」$v_{i,j}$。

汝要找到一條從 (x_0, y_0) 不斷往上下左右移動，最後達到 (x_1, y_1) 的路徑，

使得途中經過的數字位元 OR 的結果最小。

在所有位元 OR 的結果最小的路徑中，請輸出最短的一組解。

特別的，如果只輸出最小的位元 OR 結果 V 可以拿 40% 的分數；

只輸出 V 以及最短路徑長 D 可以拿 60% 的分數；

輸出 V、D、以及一組解 C 可以拿 100% 的分數。

0 子任務們

- 子任務 1 (0 分)：範例測資
- 子任務 2 (10 分)：$N = 1$
- 子任務 3 (25 分)：$N \leq 2$
- 子任務 4 (20 分)：$v_{i,j} \leq 1$
- 子任務 5 (15 分)：$v_{i,j} \leq 31$
- 子任務 6 (30 分)：無額外限制

其它輸入限制：

- 測資數量 $1 \leq T \leq 100$
- 地圖長寬 $1 \leq N, M \leq 100$
- 業力範圍 $0 \leq v_{i,j} \leq 2^{30} - 1$

子任務 2

$N = 1$

1 子任務 2 的解法

$N = 1$ 代表只有一行：

- 從起點走到終點只有一種方法。
- 把路徑上的數字 OR 起來。

```
3

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

1 5
0 0 1073741823 0 0
1 1
1 5

1 7
27 3 7 19 14 18 19
1 4
1 2
```

1 子任務 2 的解法

$N = 1$ 代表只有一行：

- 從起點走到終點只有一種方法。
- 把路徑上的數字 OR 起來。

Accepted (partially)

得分：10 分

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

1 5
0 0 1073741823 0 0
1 1
1 5

1 7
27 3 7 19 14 18 19
1 4
1 2
```

子任務 3

$N \leq 2$

2 子任務 3 的解法

$N \leq 2$ 代表不會同時有「往左走」跟「往右走」，因為這樣就會有 2×2 的方格都被走過，而也一定存在一種走法只會經過其中至多 3 格。

這個 subtask 沒有預期的作法，主要是給有滿分想法但是不會 BFS 的人拿分。

子任務 4

$v_{i,j} \leq 1$

3 子任務 4 的解法

迷宮的數值只包含 0 跟 1，當汝走到是 1 的格子那業力就會永遠變成 1。

也就是說，如果能從起點到終點都只走 0 那麼答案就是 0，否則就是 1。

這就是 BFS 的經典題！

如果汝會判斷只走 0 能不能走的到終點，至少能拿到 20 × 0.4 = 8 分。

最短路長度要怎麼計算？

3 子任務 4 的解法

最短路長度要怎麼計算？

如果答案是 0，那當汝做 BFS 的時候就會算出最短距離。

- 要構造路徑就是 BFS 的時候紀錄來的點，最後回溯得出。

如果答案是 1，那麼汝可以在整張地圖上隨便走。

- 直接走直線距離 $|x_0 - x_1| + |y_0 - y_1|$ 就是最短。
- 其中一種構造路徑的方法是先走 x 方向再走 y 方向，應該很簡單。

3 子任務 4 的解法

最短路長度要怎麼計算？

(partially) Accepted

如果答案是 0，那當汝做 BFS 的時候就會算出最短距離。

- 要構造路徑就是 BFS 的時候紀錄來的點，最後回溯得出。

得分：10 + 20 分

如果答案是 1，那麼汝可以在整張地圖上隨便走。

- 直接走直線距離 $|x_0 - x_1| + |y_0 - y_1|$ 就是最短。
- 其中一種構造路徑的方法是先走 x 方向再走 y 方向，應該很簡單。

子任務 5

$v_{i,j} \leq 31$

4 子任務 5 的解法

就算汝把所有業力為 $0, 1, 2, \ldots, 31$ 的格子都走一遍答案也只會是 31，所以可以依序檢查答案能不能是 $K = 0, 1, 2, \ldots, 31$。

那要怎麼檢查？是只要走過去身上的業力不會超過 K 就繼續走下去？

- 下面是一個反例，當汝走到紅色的 1 就會認為 $(2, 6)$ 這個位置最小的業力是 1，
  而後走到 $(2, 5)$ 的時候就會記下最小值是 1，但實際上要走只有 0 跟 2 的路徑才是最佳解。

```
2 10
3 2 0 3 2 2 2 1 1 3
1 3 2 2 0 1 0 0 0 2
2 9
1 2
```

4 子任務 5 的解法

觀察：位元 OR 可以看做是集合的聯集運算。

把每個數字以二進位表示 $11_{10} = 01011_2$、$31_{10} = 11111_2$，並將一個數字看作一個集合 s_i，滿足

$$s_i = \{x \mid i \text{ 在二進位下的第 } x \text{ 位是 } 1\}$$

也就是 $s_{11} = \{0, 1, 3\}$、$s_{31} = \{0, 1, 2, 3, 4\}$、$s_0 = \emptyset$。

而兩個數字的位元 OR 就是聯集：

$$6_{10} \text{ OR } 18_{10} = 0110_2 \text{ OR } 10010_2 \Longrightarrow \{1, 2\} \cup \{1, 4\} = \{1, 2, 4\} \Longrightarrow 10110_2 = 22_{10}$$

```
2 10
{0,1} {  1} {   } {0,1} {  1} {  1} {  1} {0  } {0  } {0,1}
{0  } {0,1} {  1} {  1} {   } {0  } {   } {   } {   } {  1}
2 9
1 2
```

4 子任務 5 的解法

觀察：位元 OR 可以看做是集合的聯集運算。

當汝枚舉 $K = 0, 1, 2, \ldots, 31$ 的時候，實際上就是在枚舉所有 $\{0, 1, 2, 3, 4\}$ 的子集合 s_K，
並且檢查有沒有一條路徑上經過的數字（集合）們聯集起來是 s_K。

要找剛好是 s_K 的解太難了，不如找存不存在聯集起來是 s_K 的子集就好？

證明：如果找到的聯集集合是 s_v，那麼因為 $s_v \subseteq s_K \Longrightarrow v \leq K$，所以在 $K' = v$ 時就會計算到。

```
2 10
{0,1} {  1} {   } {0,1} {  1} {  1} {  1} {0  } {0  } {0,1}
{0  } {0,1} {  1} {  1} {   } {0  } {   } {   } {   } {  1}
2 9
1 2
```

4 子任務 5 的實作

枚舉 $K = 0, 1, 2, \ldots, 31$，找存不存在聯集起來是 s_K 的子集的路徑。

剛剛「集合」只是為了方便理解，
在 C++ 的實作會直接使用 int 儲存，
並以 (a | b) == b 來判斷 a 是不是 b 的子集。

右邊的函數會被一直呼叫並傳入 (x0, y0, K)，
直到得出的 dis[x1][y1] != -1 為止。

根據剛剛的分析，最多只會呼叫 32 次該函數。

```cpp
const pair<int, int> dir[4] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
int val[maxn][maxn], dis[maxn][maxn];

void bfs(int sx, int sy, int mx) {
    /// 起點就爆了 ///
    if ((val[sx][sy] | mx) != mx) return;
    /// 初始化起點距離為 0，其他為 -1 ///
    memset(dis, 0xFF, sizeof(dis));
    dis[sx][sy] = 0;
    /// 一般的 BFS ///
    queue<pair<int, int>> que; que.emplace(sx, sy);
    while (!que.empty()) {
        auto [nx, ny] = que.front(); que.pop();
        for (auto [dx, dy] : dir) {
            /// 如果 dis[][] 不是 -1 代表已經走過了 ///
            if (dis[nx+dx][ny+dy] != -1) continue;
            /// 如果 val[][] 不是 mx 的子集就 continue ///
            if ((val[nx+dx][ny+dy] | mx) != mx) continue;
            dis[nx+dx][ny+dy] = dis[nx][ny] + 1;
            que.emplace(nx+dx, ny+dy);
        }
    }
}
```

4 子任務 5 的實作

枚舉 $K = 0, 1, 2, \ldots, 31$，找存不存在聯集起來是 s_K 的子集的路徑。

(partially) Accepted

剛剛「集合」只是為了方便理解，
在 C++ 的實作會直接使用
`(a | b) == b` 來判斷 a 是不是 b 的子集。

得分：10 + 35 分

右邊的函數會被一直呼叫並傳入 (x0, y0, K)，
直到得出的 `dis[x1][y1] != -1` 為止。

根據剛剛的分析，最多只會呼叫 32 次該函數。

```
1  pair<int, int> dir[4] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
2  int val[maxn][maxn], dis[maxn][maxn];
3
4  void bfs(int sx, int sy, int mx) {
5      /// 起點就壞了 ///
6      if ((val[sx][sy] | mx) != mx) return;
7      /// 初始化起點距離為 0，其他為 -1 ///
8      memset(dis, 0xFF, sizeof(dis));
9      dis[sx][sy] = 0;
10     /// 標準的 BFS ///
11     queue<pair<int, int>> que; que.emplace(sx, sy);
12     while (!que.empty()) {
13         auto [nx, ny] = que.front(); que.pop();
14         for (auto [dx, dy] : dir) {
15             /// 如果 dis[][] 不是 -1 代表已經走過了 ///
16             if (dis[nx+dx][ny+dy] != -1) continue;
17             /// 如果 val[][] 不是 mx 的子集就 continue ///
18             if ((val[nx+dx][ny+dy] | mx) != mx) continue;
19             dis[nx+dx][ny+dy] = dis[nx][ny] + 1;
20             que.emplace(nx+dx, ny+dy);
21         }
22     }
23 }
```

子任務 6

$v_{i,j} \leq 2^{30} - 1$

5 子任務 6 的解法

一個一個集合去判斷實在是太慢了，有沒有什麼性質可以利用的？

根據 s_k 的定義 $x \in s_k$ 的代價是 2^x。

如果 $K = v$ 時可以走到終點，那麼代價就是 $v = \sum_{x \in s_v} 2^x$。

注意到 $2^x > 2^x - 1 = \sum_{i=0}^{x-1} 2^i$（就是前面的位數越大數字就越大），

代表我們可以 greedy 的由大到小判斷存不存在可行的 K 使 $x \notin s_K$。

5 子任務 6 的解法

我們可以 greedy 的由大到小判斷存不存在可行的 K 使 $x \notin s_K$，x 的範圍是 $[0, 29]$。

也就是先把 $x = 29 \in s_{v_{i,j}}$ 的格子都當成不能走，跟子任務 4 一樣做 BFS 判斷存不存在答案。

- 如果存在 $x \notin s_K$ 的路徑：那麼所有 $x \in s_{v_{i,j}}$ 的格子都會永遠變成障礙物。
- 如果不存在 $x \notin s_K$ 的路徑：那麼一定有 $x \in s_V$，且汝之後可以忽略掉 x。

做完 $x = 29$ 就接著做 $28, 27, \ldots, 0$，最後就會得到答案了！

時間複雜度是 $O(TNM \lg C)$，其中 C 是業力的範圍 2^{30}。

5 子任務 6 的實作

把 $x \in s_{v_{i,j}}$ 的格子都當成不能走，做 BFS 判斷存不存在答案。

- 使用 a & (1 << x) 或是 (a >> x) & 1 可以快速判斷 $x \in s_a$。

如果存在 $x \notin s_K$ 的路徑：那麼所有 $x \in s_{v_{i,j}}$ 的格子都會永遠變成障礙物。

- 簡單的實作方法是對所有 $x \in s_{v_{i,j}}$ 的格子都讓 $v_{i,j} = 2^{30} - 1$，這樣就會包含所有元素，後續的 BFS 也不會走到上面。

如果不存在 $x \notin s_K$ 的路徑：那麼一定有 $x \in s_V$，且汝之後可以忽略掉 x。

- 答案加上去就好。

5 子任務 6 的實作

把 $x \in s_{v_{i,j}}$ 的格子都當成不能走，做 BFS 判斷存不存在答案。

- 使用 a & (1 << x) 或是 (a >> x) & 1 可以快速判斷 $x \in s_a$。

如果存在 $x \notin s_K$ 的路徑：那麼所有 $x \in s_{v_{i,j}}$ 的格子都會永遠變成障礙物。

- 簡單的實作方法是對所有 $x \in s_{v_{i,j}}$ 的格子都讓 $v_i = 2^{30} - 1$，這樣就會包含所有元素，後續的 BFS 也不會走到上面。

如果不存在 $x \notin s_K$ 的路徑：那麼一定有 $x \in s_V$，且汝之後可以忽略掉 x。

- 答案加上去就好。

Accepted

得分：100 分

結論

6 結論

子任務 2（$N = 1$）的做法真的很單純，只要有好好看完題目就應該要能拿到這 10 分。

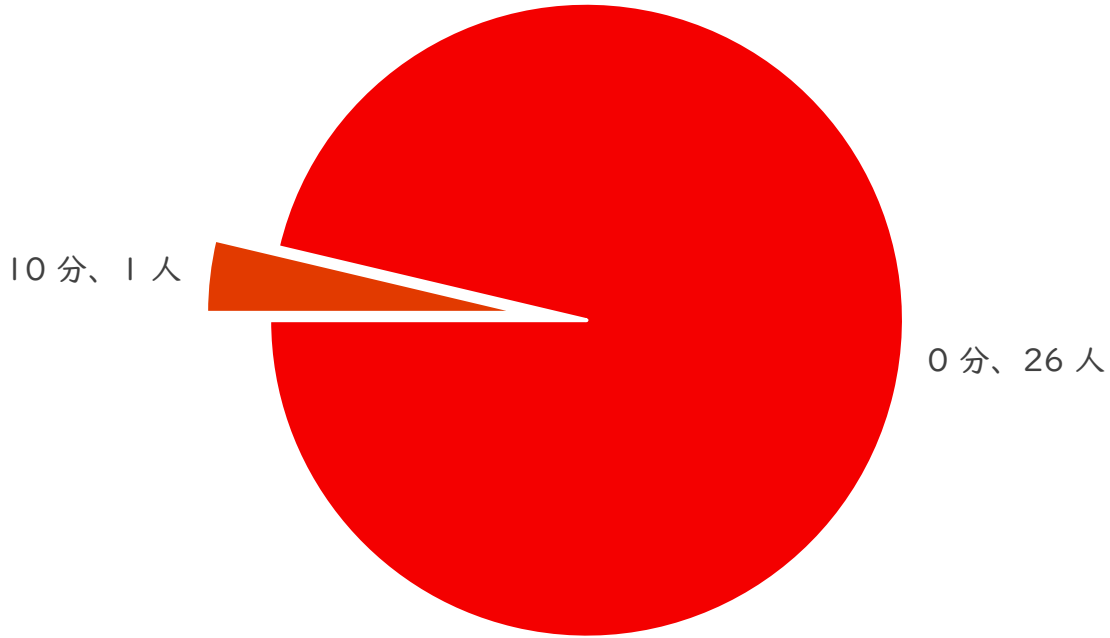
子任務 4（$v_{ij} \leq 1$）也只是單純的 BFS 而已，相信大家也都有學過了。

注意位元運算的優先順序，如果寫出 a | b == b 那他會變成 a | (b == b)。

最好的避免方法就是加上括弧增加可讀性。

AC Code 連結

6 成績分布



F. 質數家族 題解

by SorahISA

0 問題概要

定義兩個質數 p、q 有連結關係 $p \leftrightarrow q$ 若且唯若 p、q 符合以下任意一個規則：

- p 跟 q 長度相同（$\lfloor \log_{10} p \rfloor = \lfloor \log_{10} q \rfloor$），且 p 跟 q 僅有一個位數不同。
- 新增一個位數到 p 的最左邊會得到 q。
- 新增一個位數到 q 的最左邊會得到 p。

定義一個質數 p 屬於 2 的質數家族 $p \in \mathcal{F}(2)$ 若且唯若：

- 存在一或多個質數 $p_1, p_2, \ldots, p_k$ 滿足 $p_k = p$ 及 $p_1, p_2, \ldots, p_k \leq p$，且存在連結關係 $2 \leftrightarrow p_1 \leftrightarrow p_2 \leftrightarrow \cdots \leftrightarrow p_k$。

給定 N，請計算

$$\sum_{\substack{p \text{ is prime} \\ p \leq N \\ p \notin \mathcal{F}(2)}} p$$

0 子任務們

- 子任務 1 (20 分)：$12 \leq N \leq 1000$
- 子任務 2 (30 分)：$1001 \leq N \leq 1060$
- 子任務 3 (50 分)：$1061 \leq N \leq 10\,000\,000$

子任務 1-2

$N \leq 1060$

1 子任務 1-2 的解法

整道題目大致上分為三個部分，分別是

1. 找到 $[1, N]$ 的所有質數。
2. 找到所有連結關係。
3. 找到所有不在 $\mathcal{F}(2)$ 且 $\leq N$ 的質數。

1 子任務 1-2 的解法

1. 找到 $[1, N]$ 的所有質數。

要怎麼知道一個數字 x 是不是質數？回想質數的定義：「> 1 且因數只有 1 跟自己」。

所以可以枚舉 $i = 2, 3, \ldots, x-1$，如果發現 $i \mid x$ 那麼 x 就不是質數。

- 複雜度是 $O(x)$，對 $1, 2, \ldots, N$ 各判斷一次大約是 $O(N^2)$。

注意到如果正整數 $ab = x$ 那麼 $\min\{a, b\} \leq \sqrt{x}$，所以如果 $[2, \sqrt{x}]$ 之間沒有 x 的因數那 $[\sqrt{x}, x-1]$ 之間也不會有，可以只枚舉 $i = 2, 3, \ldots, \lfloor \sqrt{x} \rfloor$ 就好。

- 複雜度是 $O(\sqrt{x})$，對 $1, 2, \ldots, N$ 各判斷一次大約是 $O(N\sqrt{N})$。

1 子任務 1-2 的解法

2. 找到所有連結關係。

要怎麼檢查兩個質數能不能互相連結？

分成位數相同（僅有一個位置不同）以及位數差一（在最前面加一位數）兩種情況分開判斷。

- 當位數相同的時候就依序檢查每個位數是否相同，能連結若且唯若在恰一個位置相異。
- 當位數不同的時候就檢查把大的最高位拔掉會不會變成小的。

如果沒有判斷位數差一可能會多找到 $3 \leftrightarrow 103$ 這種連結。

1 子任務 1-2 的解法

3. 找到所有不在 $\mathcal{F}(2)$ 且 $\leq N$ 的質數。

將連結關係建成一張無向圖之後，對每個質數 p 做一次 DFS。

檢查能不能在只走數字 $\leq p$ 的點的情況下從 2 走到 p，不行就代表 $p \notin \mathcal{F}(2)$。

實作方法大致如右，每次設置一個上限 limit，
只有在要拜訪的點 $\leq$ limit 時才能繼續 DFS。

```
1 void dfs(int now, int limit) {
2     vis[now] = 1;
3     for (int x : adj[now]) {
4         if (!vis[x] and x <= limit) dfs(x, limit);
5     }
6 }
```

1 子任務 1-2 的解法

整道題目大致上分為三個部分，分別是

1. 找到 $[1, N]$ 的所有質數，$\mathcal{O}(N\sqrt{N})$。

2. 找到所有連結關係，$\mathcal{O}\left(\left(\frac{N}{\log N}\right)^2 \cdot \log N\right) = \mathcal{O}\left(\frac{N^2}{\log N}\right)$。

3. 找到所有不在 $\mathcal{F}(2)$ 且 $\leq N$ 的質數，$\mathcal{O}\left(\left(\frac{N}{\log N}\right) \cdot \left(\frac{N}{\log N} + \frac{N}{\log N} \cdot \log N\right)\right) = \mathcal{O}\left(\frac{N^2}{\log N}\right)$。

註：N 以下的質數數量大約是 $\mathcal{O}\left(\frac{N}{\log N}\right)$，且一個數字修改其中一個位數只會有 $\mathcal{O}(\log N)$ 種可能。

1 子任務 1-2 的解法

整道題目大致上分為三個部分，分別是

1. 找到 $[1, N]$ 的所有質數，$\mathcal{O}(N\sqrt{N})$。

2. 找到所有連結關係，$\mathcal{O}\left(\left(\frac{N}{\log N}\right)^2 \cdot \log N\right) = \mathcal{O}\left(\frac{N^2}{\log N}\right)$。

3. 找到所有不在 $\mathcal{F}(2)$ 且 $\leq N$ 的質數，$\mathcal{O}\left(\left(\frac{N}{\log N}\right) \cdot \left(\frac{N}{\log N} + \frac{N}{\log^2 N} \cdot \log N\right)\right) = \mathcal{O}\left(\frac{N^2}{\log N}\right)$。

Accepted (partially)

得分：50 分

註：N 以下的質數數量大約是 $\mathcal{O}\left(\frac{N}{\log N}\right)$，且一個數字修改其中一個位數只會有 $\mathcal{O}(\log N)$ 種可能。

子任務 3

$N \leq 10\ 000\ 000$

2 子任務 3 的解法

整道題目大致上分為三個部分，分別是

1. 找到 $[1, N]$ 的所有質數，$\mathcal{O}\left(N\sqrt{N}\right)$。
2. 找到所有連結關係，$\mathcal{O}\left(\left(\frac{N}{\log N}\right)^2 \cdot \log N\right) = \mathcal{O}\left(\frac{N^2}{\log N}\right)$。
3. 找到所有不在 $\mathcal{F}(2)$ 且 $\leq N$ 的質數，$\mathcal{O}\left(\left(\frac{N}{\log N}\right) \cdot \left(\frac{N}{\log N} + \frac{N}{\log N} \cdot \log N\right)\right) = \mathcal{O}\left(\frac{N^2}{\log N}\right)$。

剛剛的這三項都會在 $N \sim 10\,000\,000$ 超時，我們需要找到更快的作法。

2 子任務 3 的解法

1. 找到 $[1, N]$ 的所有質數，~~$\mathcal{O}(N\sqrt{N})$~~ $\mathcal{O}(N \log N)$。

不要一個數字一個數字慢慢查，不如一次把所有數字做完？

用 bool 陣列記錄 $[1, N]$ 每個數字是不是質數，再依序把 $2, 3, 4, \ldots, N$（兩倍以上）的倍數刪掉。

複雜度是 $\mathcal{O}\left(\sum_i \frac{N}{i}\right) = \mathcal{O}(N \log N)$。

調和級數的複雜度也是競賽中非常常見的主題之一。

2 子任務 3 的解法

1. 找到 $[1, N]$ 的所有質數，~~$\mathcal{O}(N\sqrt{N})$~~ ~~$\mathcal{O}(N \log N)$~~ $\mathcal{O}(N \log\log N)$。

只有質數的倍數才重要，例如：6 能刪的數字都已經被 2 跟 3 刪掉了。

用 bool 陣列記錄 $[1, N]$ 每個數字是不是質數，再依序把 $2, 3, 5, \ldots, N$ 等質數（兩倍以上）的倍數刪掉。

複雜度是 $\mathcal{O}\left(\sum_p \frac{N}{p}\right) = \mathcal{O}(N \log\log N)$。

質數倒數和的複雜度記起來，證明就先不管他了。

2 子任務 3 的解法

1. 找到 $[1, N]$ 的所有質數，~~$\mathcal{O}(N\sqrt{N})$~~ ~~$\mathcal{O}(N \log N)$~~ ~~$\mathcal{O}(N \log \log N)$~~ $\mathcal{O}(N)$。

如果一個數字只會被算到一次那就可以做到 $\mathcal{O}(N)$ 了！

維護一個質數陣列以及每個數字的最小質因數 lp_i，並更新 $2i, 3i, 5i, \ldots, lp_i \cdot i$ 的最小質因數。

雖然複雜度較小，但因為操作常數較大，競賽上通常是使用 $\mathcal{O}(N \log \log N)$ 的版本。

參考資料

2 子任務 3 的解法

2. 找到所有連結關係，~~$\mathcal{O}(N^2/\log N)$~~ $\mathcal{O}(N)$。

之前的做法是「找出所有質數」再「驗證連結是否合法」，
現在可以試試「找出所有連結」再「驗證連結的數是否是質數」。

驗證質數這點才剛處理完，已經可以 $\mathcal{O}(1)$ 做到了，那麼找出連結要如何？
（這裡只要對 p 找出那些 $q < p$ 可以連結的就好，讓大的去找小的。）

一樣分成長度少一以及長度不變的兩種情況：

- 把最高位拔掉（需要次高位非 0）形成的數字，總共有 1 種。
- 每個位數分別代換成比該位數小的數字，總共有 $\mathcal{O}(9\log N)$ 種。

2 子任務 3 的解法

3. 找到所有不在 $\mathcal{F}(2)$ 且 $\leq N$ 的質數，~~$\mathcal{O}(N^2/\log N)$~~ $\mathcal{O}\big(N\,\alpha(N)\big)$。

想要知道 $p \in \mathcal{F}(2)$，我們在意的不只是直接有連結關係的點對，也包含間接連接著的點對們。

如果將連結關係們由小到大加入這張圖，那我們需要的就是詢問「在加入了所有兩端都 $\leq p$ 的邊之後，2 和 p 是不是連通的」。

很巧的，方才 (2.) 建立連結的順序就是依照兩端點最大值由小到大排好的！

要不斷的加邊、判斷兩點在不在同個連通塊內，自然的就會想到使用並查集了！

2 子任務 3 的解法

整道題目大致上分為三個部分，分別是

1. 找到 $[1, N]$ 的所有質數，$\mathcal{O}(N)$。
2. 找到並建立所有連結關係，$\mathcal{O}\big(N\ \alpha(N)\big)$。
3. 找到所有不在 $\mathcal{F}(2)$ 且 $\leq N$ 的質數，$\mathcal{O}\big(N\ \alpha(N)\big)$。

建立連結關係需要使用並查集的 Union 操作，其複雜度是 $\mathcal{O}\big(\alpha(N)\big)$。

現在這三項在 $N \sim 10\,000\,000$ 時都可以在一秒左右執行完，可以拿到 AC 了！

2 子任務 3 的解法

整道題目大致上分為三個部分，分別是

1. 找到 $[1, N]$ 的所有質數，$\mathcal{O}(N)$。
2. 找到並建立所有連結關係，$\mathcal{O}(N\,\alpha(N))$。
3. 找到所有不在 $\mathcal{F}(2)$ 且 $\leq N$ 的質數，$\mathcal{O}(N\,\alpha(N))$。

Accepted

得分：100 分

建立連結關係需要使用並查集的 Union 操作，其複雜度是 $\mathcal{O}(\alpha(N))$。

現在這三項在 $N \sim 10\,000\,000$ 時都可以在一秒左右執行完，可以拿到 AC 了！

2 並查集

如果汝沒聽過、或是不熟悉並查集，那麼這裡是它的介紹。

並查集是支援「合併兩個集合 Union(x, y)」、「查詢一個元素所在集合 Find(x)」的資料結構。

汝可以維護 N 個集合暴力實作，又或是使用以下介紹的「集團老大法」。

定義 B_x 為 x 的老大，一開始每個人的老大都是自己，也就是 $B_x = x$。

2 並查集－Union(x, y)

想要讓兩個人所在的集團合併，當然首先要問過他們最上級的老大 $boss_x$ 跟 $boss_y$。

- 如果他們的老大其實是同一人，那就不需要合併。
- 如果不是同一人，接下來就讓那兩位老大談合併的事情就好
  （此處直接讓 $boss_x$ 認 $boss_y$ 為老大）。

```
1 vector<int> B(N);
2 for (int i = 0; i < N; ++i) B[i] = i;
3
4 void Union(int x, int y) {
5     int bossX = Find(x), bossY = Find(y);
6     if (bossX != bossY) B[bossX] = bossY;
7 }
```

2 並查集－Find(x)

要查詢某個人最上級的老大 $boss_x$，就一直問「汝的老大 B_x 是誰」。

- 如果他的回覆是「我的老大就是我自己（$B_x = x$）」，那就找到答案了。
- 不然，就繼續去問他的老大 B_x「汝的老大 B_{B_x} 是誰」。

這樣在最糟情況下每次 Find(x) 都是 $O(N)$，
於是需要一點小優化。

```
1 int Find(int x) {
2     if (B[x] == x) return x;
3     else return Find(B[x]);
4 }
```

2 並查集－優化

第一種優化是「路徑壓縮優化」，核心思路是：帶著他去問他的老大「汝的老大是誰」。

重點就在這個「帶著他」，讓中途問過的每個人都知道他最上級的老大是誰。

這樣就不會出現「不管問了幾次，他都只能帶汝見他上一層的老大」了。

第二種優化是「啓發式合併」，核心思路是：當兩個集團合併時，把小的搬到大的會更省時。

通常使用的方法是紀錄每個集團的人數，人多的一方當最終的老大，別忘記還要更新集團人數。

資訊都只需要紀錄在老大那裡，其他人負責記得自己老大是誰就好。

2 並查集－優化

第一種優化是「路徑壓縮優化」，核心思路是：帶著他去問他的老大「汝的老大是誰」。

第二種優化是「啓發式合併」，核心思路是：當兩個集團合併時，把小的搬到大的會更省時。

單獨使用任一種優化的複雜度都是 $\mathcal{O}(Q \log N)$，同時使用兩種優化則會降至 $\mathcal{O}\big(Q \cdot \alpha(Q,N)\big)$，其中 Q 代表操作次數，而 $\alpha(x,y)$ 是 Inverse Ackermann 函數，可以視為大約是 4 的常數。

只使用「啓發式合併」的複雜度較容易證明，因為每次向上詢問都會保證集團大小變成兩倍，於是 $\mathrm{Find}(x)$ 的遞迴深度頂多只會是 $\mathcal{O}(\log N)$。

通常競賽中都以「路徑壓縮優化」為主，但也有需要額外紀錄資訊的題目存在，所以兩種優化都要會。

2 並查集－優化

講了這麼多，汝會不會以為他的程式碼很長？不，其實他非常的短！

並查集還有許多進階的應用，這麼棒的資料結構還不學起來？

```cpp
vector<int> p;
p.resize(N); /// 指定大小為 N
iota(p.begin(), p.end(), 0); /// 依序給 0, 1, ..., N-1
/// R 是 Find()、U 是 Union(x, y)，此處回傳的是兩個集團有沒有成功合併 ///
int R(int x) {return x ^ p[x] ? p[x] = R(p[x]) : x;}
int U(int x, int y) {x = R(x), y = R(y); return x ^ y ? p[x] = y, 1 : 0;}
```

結論

3 結論

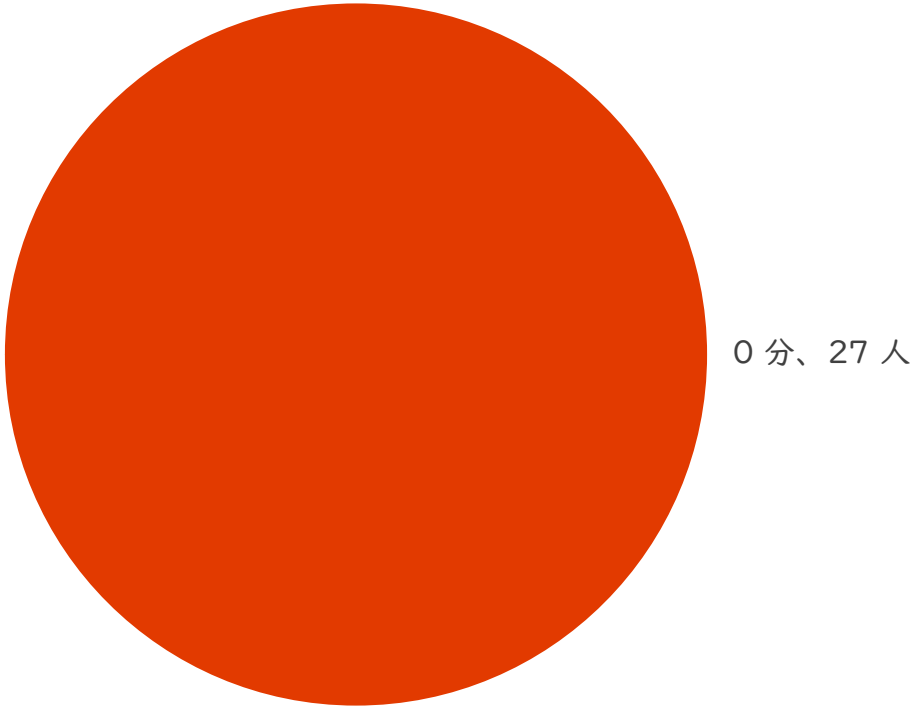
50 分跟滿分基本上是兩個完全不同的天地（而且這題時間有點緊），
不過 50 分需要的知識只有「知道質數是什麼」還有「DFS」而已。

希望汝在這題中學到的知識：

- $\mathcal{O}(N)$ 或 $\mathcal{O}(N \log\log N)$ 質數篩。
- 並查集跟基本的兩種優化。
- 時間複雜度概念：倒數和 $\approx \mathcal{O}(\log N)$。
- 時間複雜度概念：隨便合併的深度 $\mathcal{O}(N)$、
  啓發式（小到大）合併的深度 $\mathcal{O}(\log N)$。

Subtask 1+2
AC Code 連結

3 成績分布



3 額外補充

1. 找到 $[1, N]$ 的所有質數，~~$O(N\sqrt{N})$~~ ~~$O(N \log N)$~~ ~~$O(N \log \log N)$~~ ~~$O(N)$~~ $O\left(\frac{N}{\log \log N}\right)$。

參見 Sieve of Pritchard，沒用過。

G. 最短生成樹 題解

by SorahISA

0 問題概要

給定一張帶權無向圖，汝要先構造出字典序最小的最短路徑生成樹。

接著請汝求出

1. 該樹上所有經過 P 個點的簡單路徑中最長的路徑長度 D 為何，
2. 有多少對 (u, v) 滿足兩端點在 u、v 的簡單路徑恰經過 P 個點且長度為 D。

0 子任務們

- 子任務 1 (30 分)：$N \leq 5000$
- 子任務 2 (70 分)：$N \leq 30\,000$

其它輸入限制：

- 點的數量 $2 \leq N \leq 30\,000$
- 邊的數量 $N - 1 \leq M \leq 60\,000$
- 目標路徑節點數 $2 \leq P \leq N$
- 邊的長度 $1 \leq w_i \leq 10\,000$
- 圖一定連通、但不保證是簡單圖

子任務 I

$N \leq 5000$

1 子任務 1 的解法

整題可以分成「建出最短生成樹」以及「找出樹上經過 P 個點的路徑們」兩個部分。

1 子任務 1 的解法

整題可以分成「建出最短生成樹」以及「找出樹上經過 P 個點的路徑們」兩個部分。

因為邊權都是正的，因此我們可以直接使用 Dijkstra 算法求出起點到所有點的最短路。

我們會維護「已經求出最短路」的集合 S 以及剩下的點 $T = V \setminus S$，一開始 $S = \emptyset$：

1. 先初始化起點 s 距離 $\mathrm{dis}[s] = 0$，而其他點的距離 $\mathrm{dis}[v] = \infty$。
2. 求出集合 T 裡距離最小的點 v（$v = \arg\min_{t \in T}\{\mathrm{dis}[t]\}$），並把 v 移到 S。
3. 對所有起點是 v 的邊去更新其他點的最短路。
4. 回到 (2.)

1 子任務 1 的解法

這個算法一定會找到最短路徑的長度，為什麼？

什麼情況下他不會找到最短路徑？只有在把 v 加進 S 之後仍然能更新 S 裡面某點的距離。

由於將 v 加進 S 的時候 $\mathrm{dis}[v] = \max_{s \in S}\{\mathrm{dis}[s]\}$ 且邊權皆 ≥ 0，不可能出現這種情況。

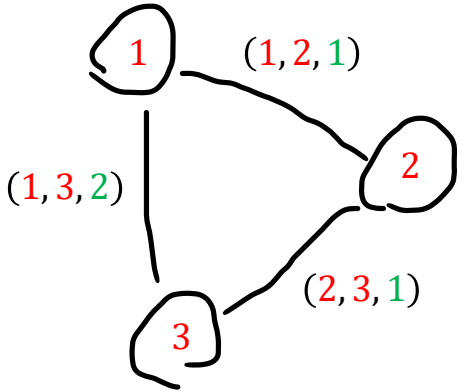
整個算法直接暴力實作的複雜度是 $O(N^2 + M)$：

- 每次找 $\arg\min_{t \in T}\{\mathrm{dis}[t]\}$ 是 $O(N)$，總共會做 N 次所以是 $O(N^2)$。
- 每條邊只會在將兩個端點加入 S 時分別被看一次，所以 (3.) 總共是 $O(M)$。

1 子任務 1 的解法

把剛剛找到的路都找出來就會是樹了ㄇ？

沒錯，不過依照汝更新的順序可能會有很多種結果的樹。



如右圖，$\text{dis}[1] = 0$、$\text{dis}[2] = 1$、$\text{dis}[3] = 2$。

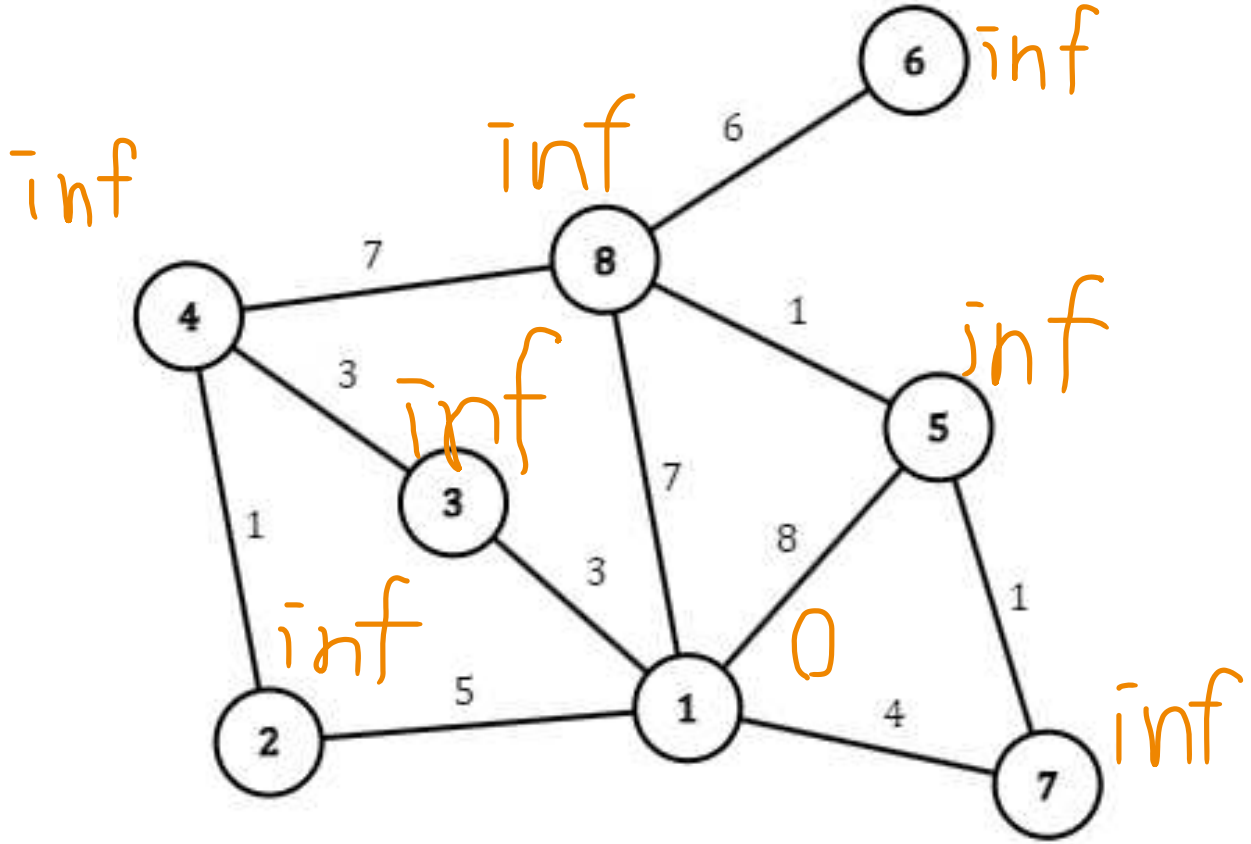
選擇 $(1,2),(2,3)$ 是一種、選擇 $(1,2),(1,3)$ 也是一種。

於是我們需要把所有可能在最短路徑樹上的邊都找出來，
再去 DFS 找出字典序最小的一組。

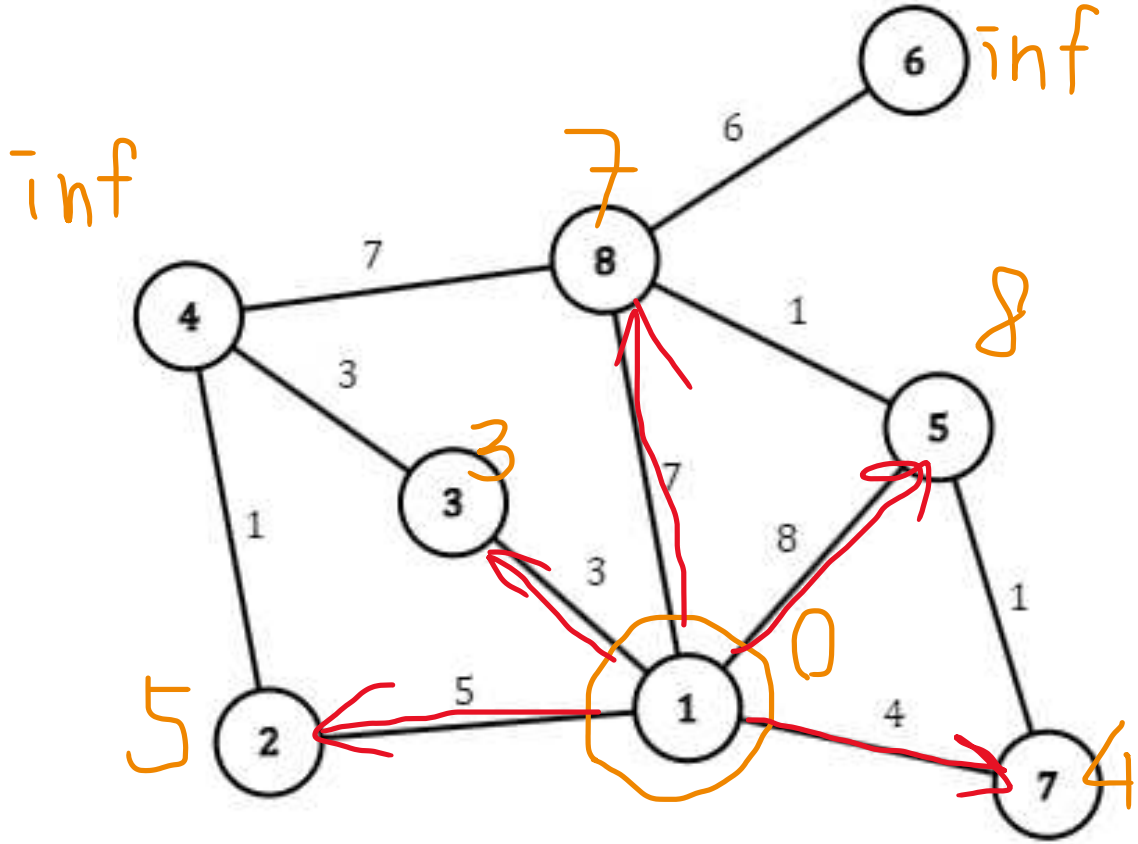
可能在最短路徑樹上的邊 (u,v,w) 就是符合 $\text{dis}[v] = \text{dis}[u] + w$ 的邊。

找最小字典序就是在所有出邊中先往編號小的去 DFS。

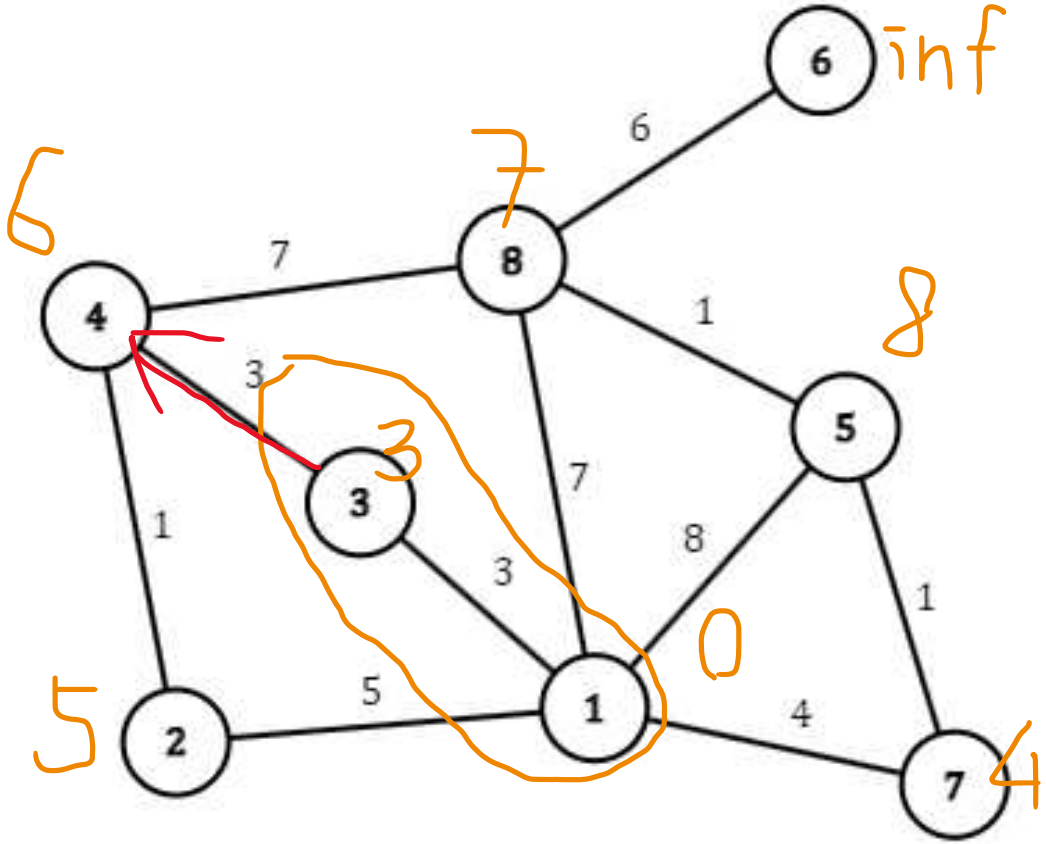
1 舉例－建最短路徑樹



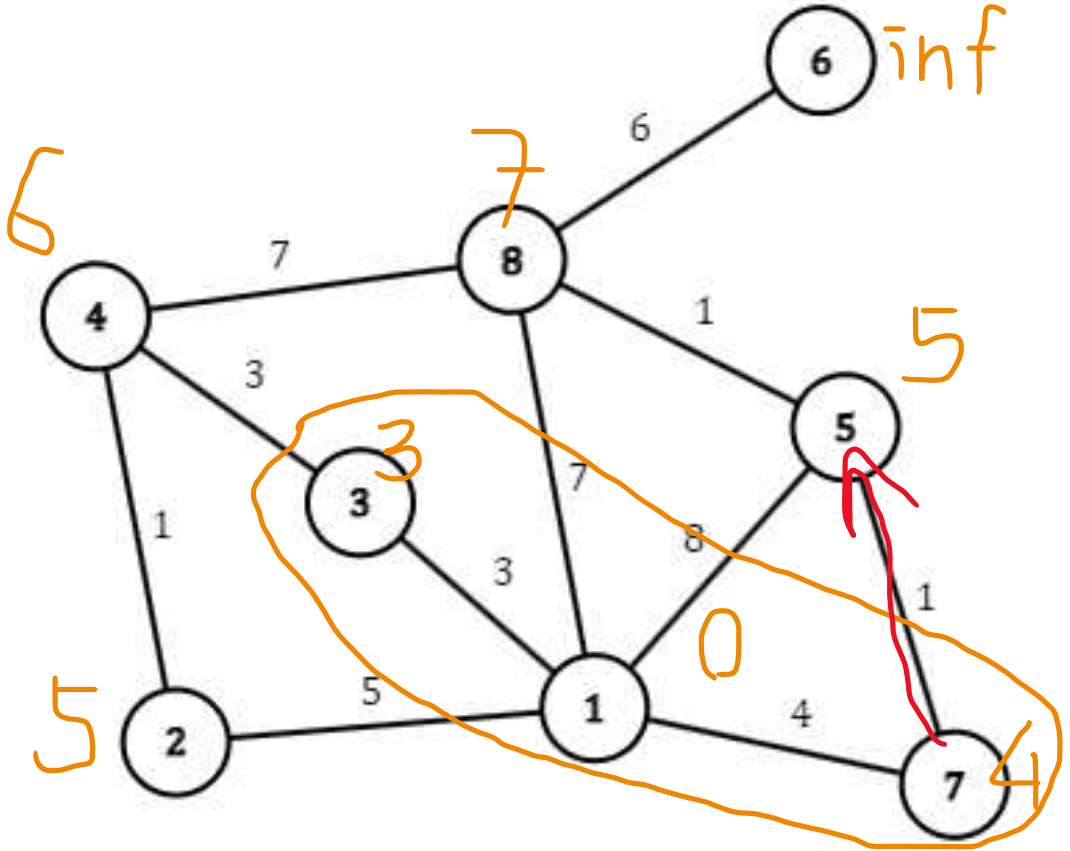
1 舉例－建最短路徑樹



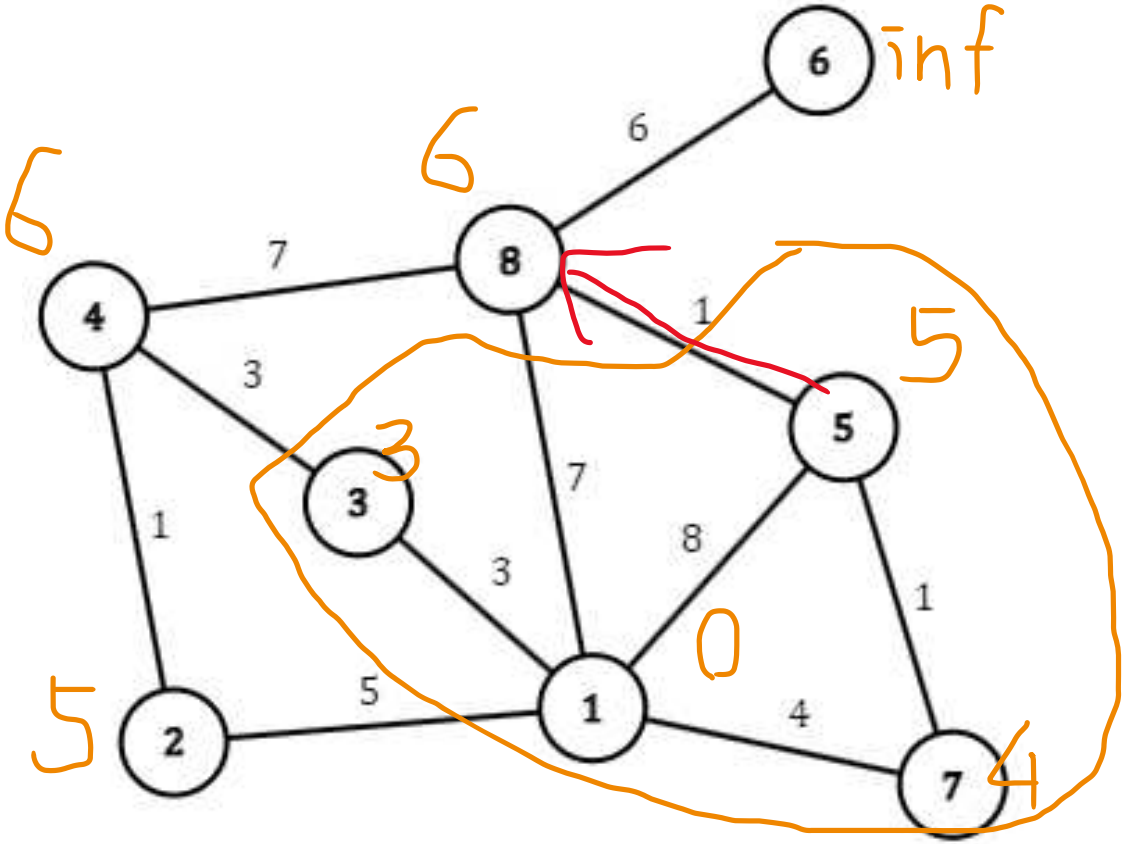
1 舉例－建最短路徑樹



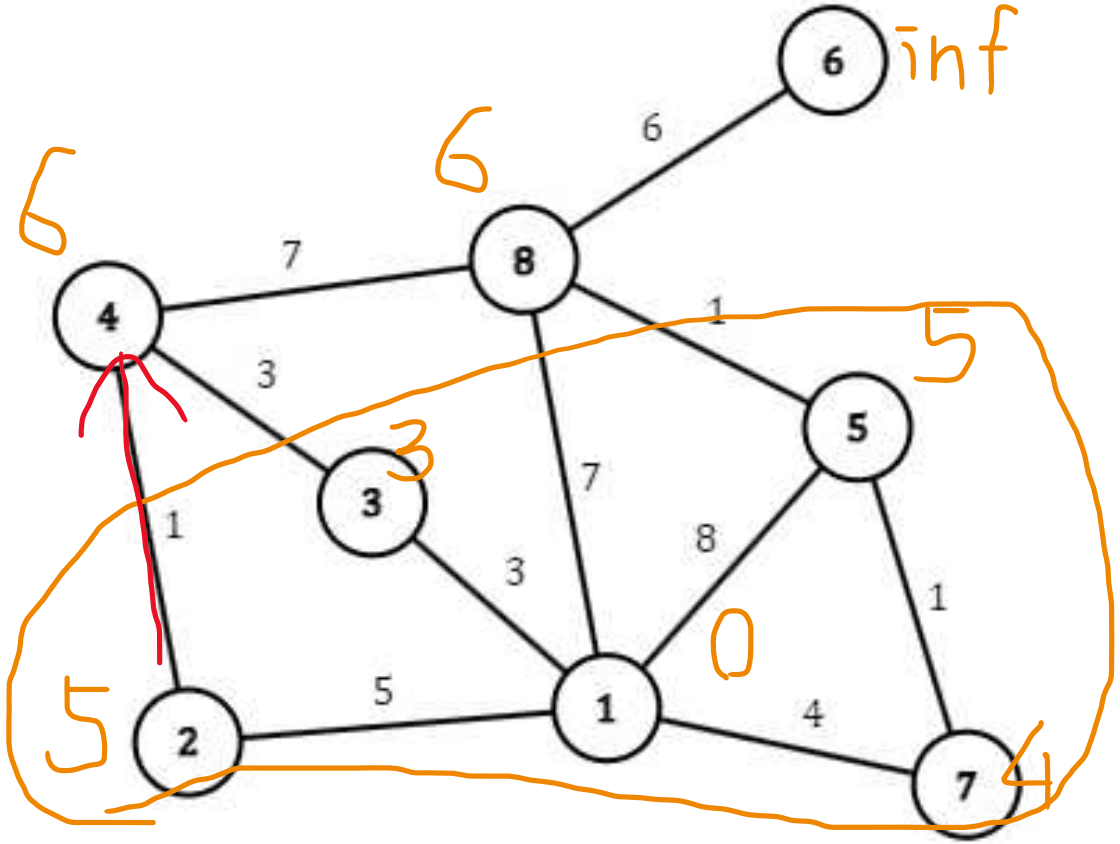
1 舉例－建最短路徑樹



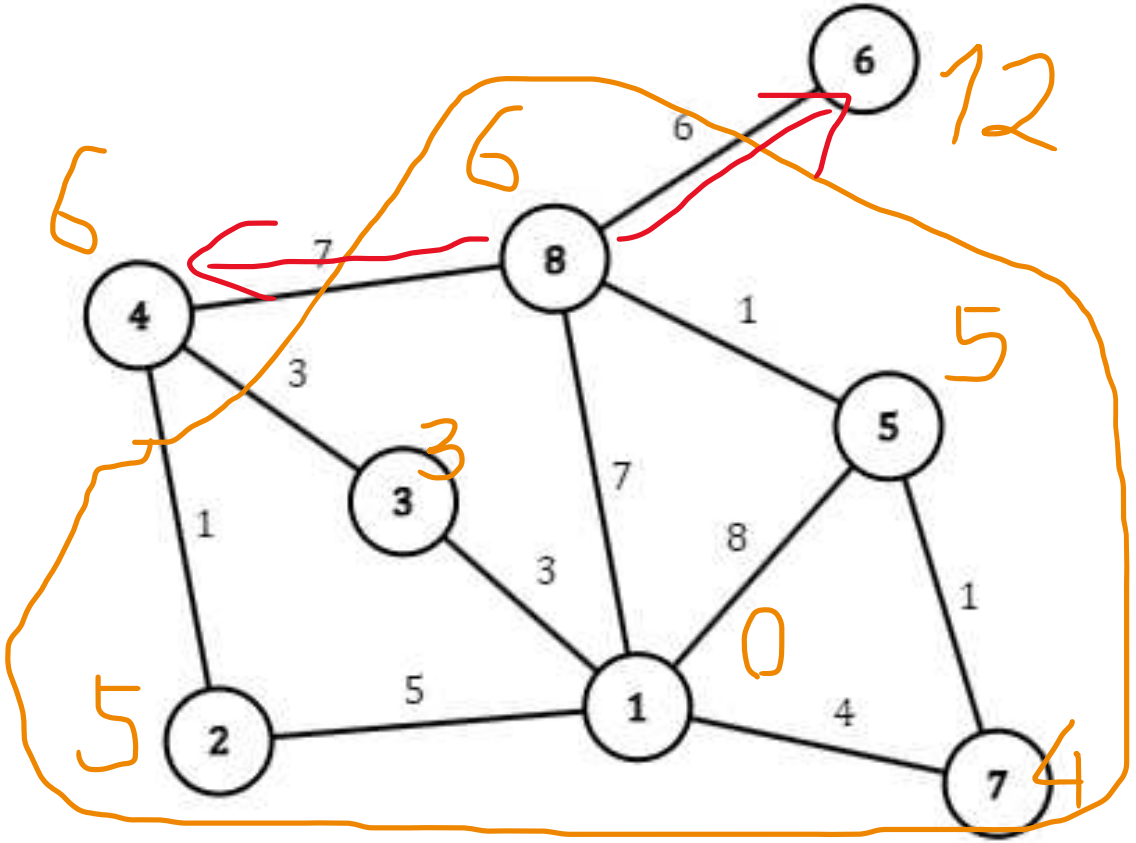
1 舉例－建最短路徑樹



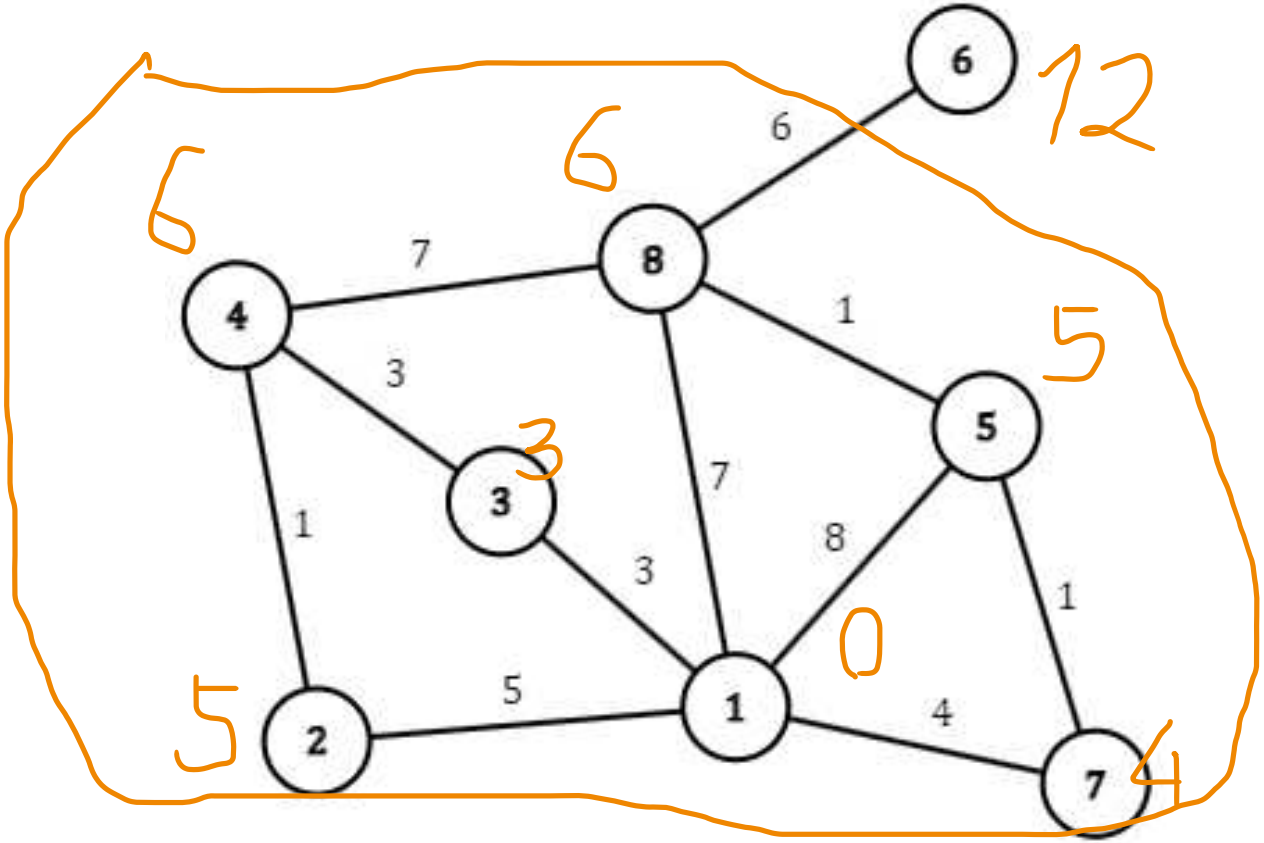
1 舉例－建最短路徑樹



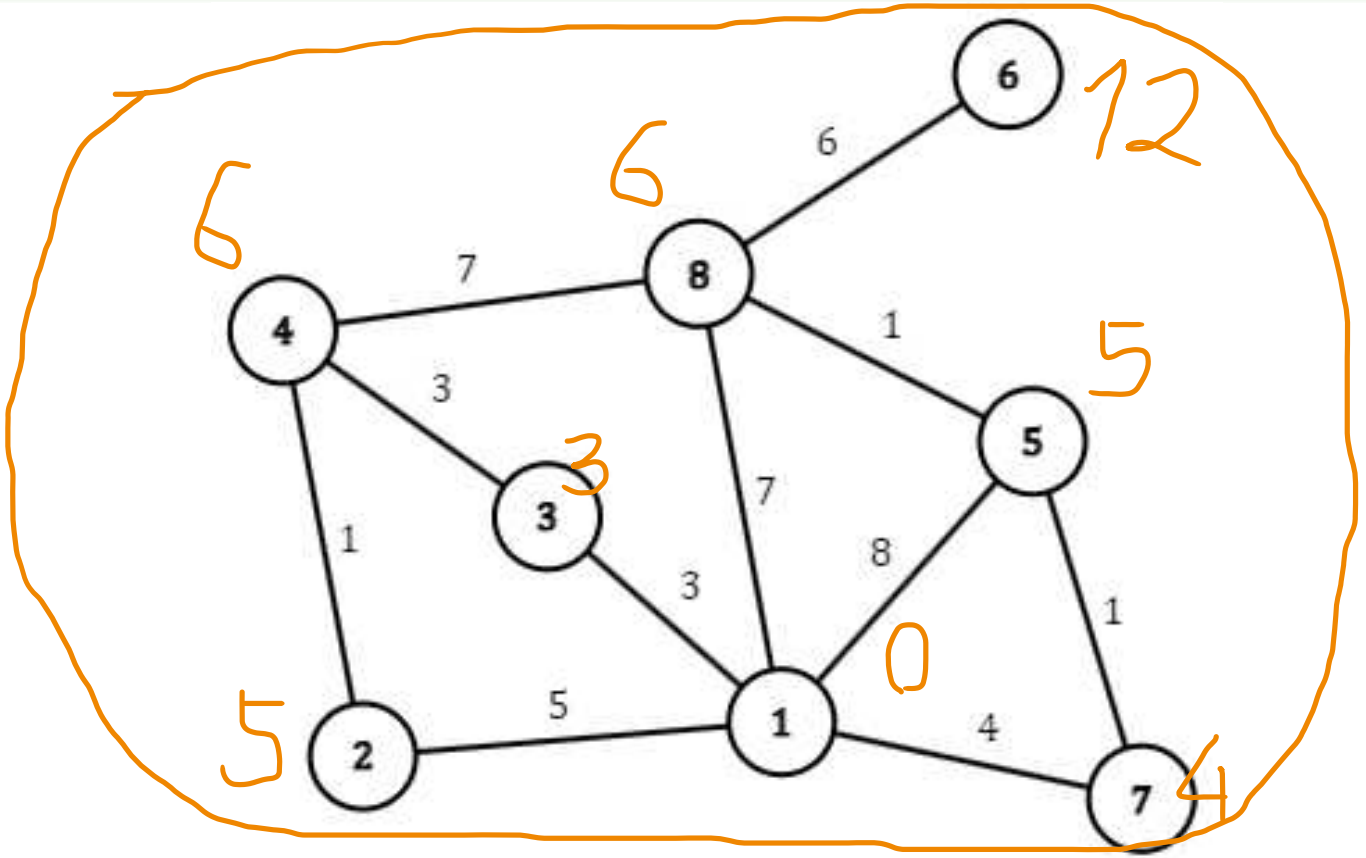
1 舉例－建最短路徑樹



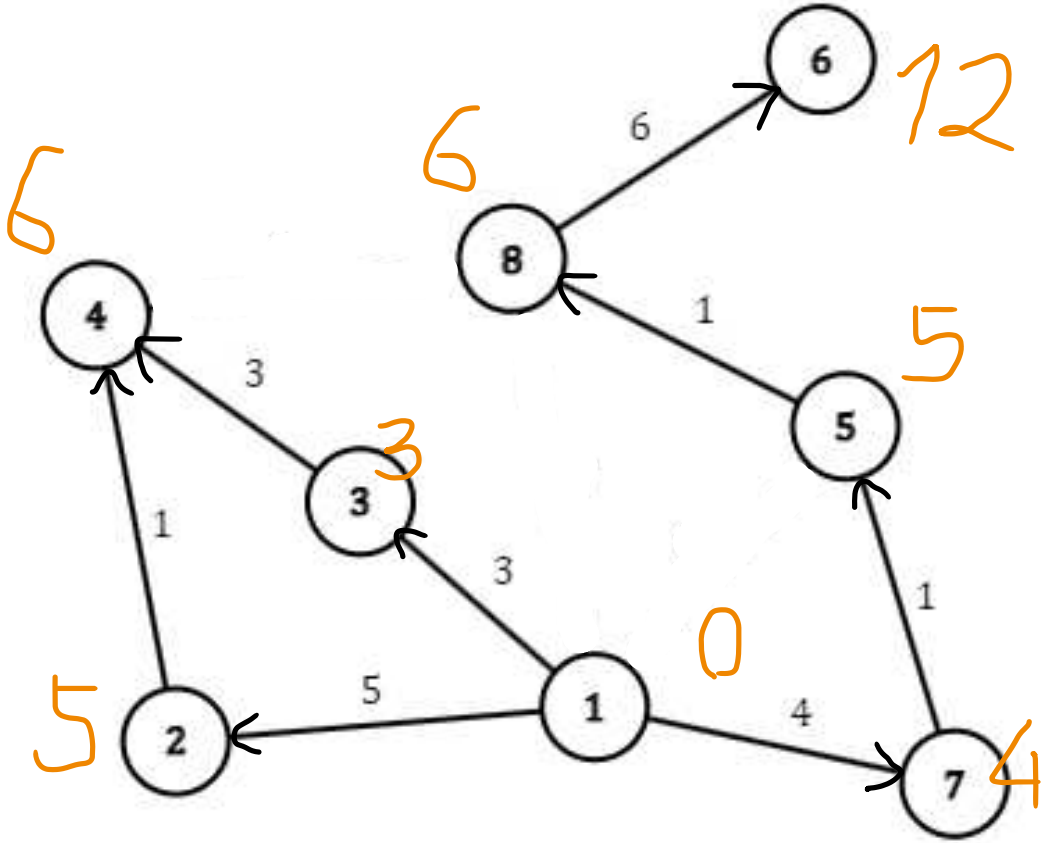
1 舉例－建最短路徑樹



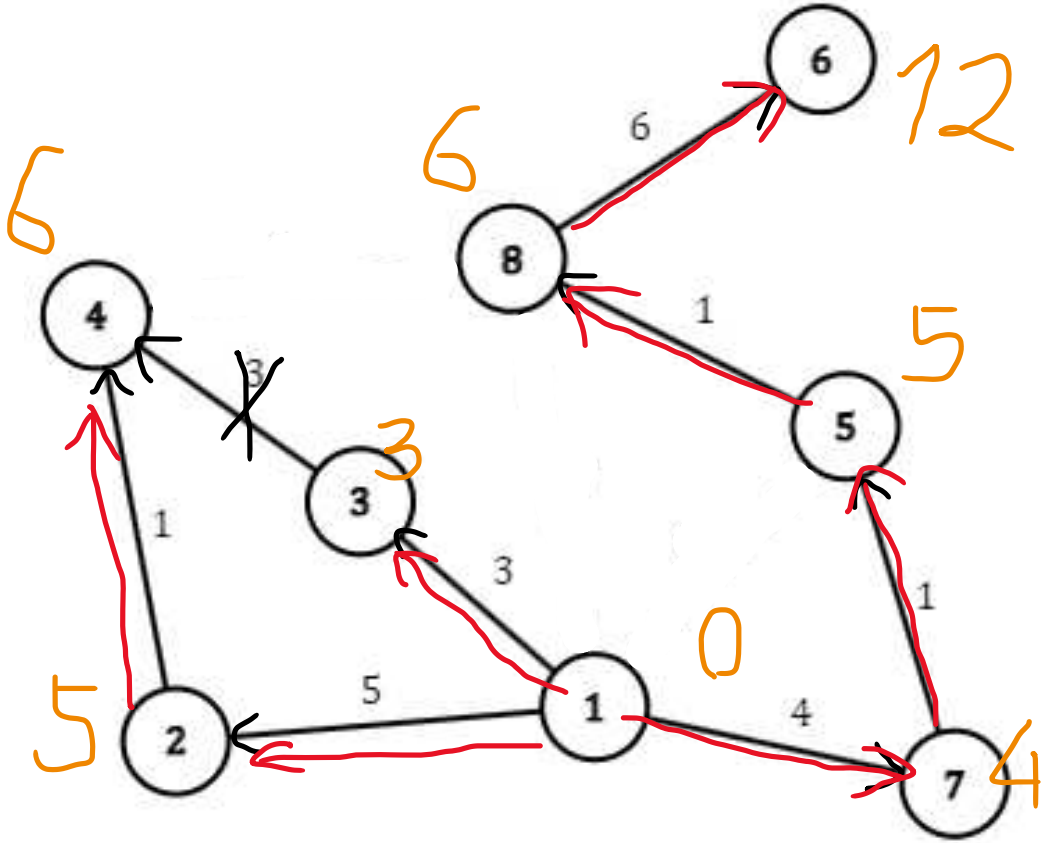
1 舉例－建最短路徑樹



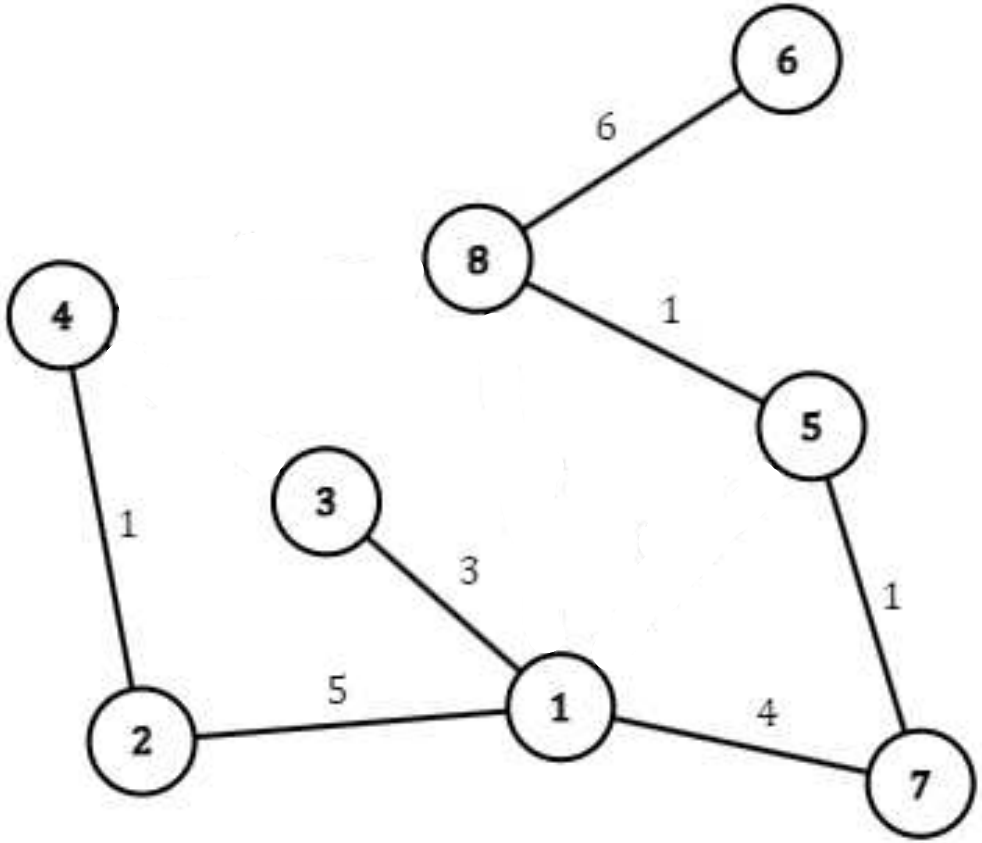
1 舉例－建最短路徑樹



1 舉例－建最短路徑樹



1 舉例－建最短路徑樹



1 子任務 1 的解法

建出樹之後就可以在上面以每個點當起點進行一遍 DFS，得到跟每個點的距離還有路徑長了。

- 距離：代表經過幾條邊。
- 路徑長：代表經過的邊的權重之和。

最後用所有距離是 $P-1$ 的點去以路徑長來更新答案即可。

1 子任務 1 的實作

G 是原圖、adj 是樹。兩者的型態為 vector<vector<pair<int, int>>>，儲存 {目標,權重}。

左下是求出最短路徑長的部分、右下是建出最短路徑樹的部分。

```cpp
/// dis[1] = 0、其他都是 INF，in_S 代表在不在集合 S 裡 ///
vector<int> dis, in_S;
for (int _round = 1; _round <= N; ++_round) {
    int now = 0; /// dis[0] = INF
    /// 找到 T 裡距離最近的點 now ///
    for (int i = 1; i <= N; ++i) {
        if (!in_S[i] and dis[i] < dis[now]) now = i;
    }
    /// 把 now 加進集合 S 裡 ///
    in_S[now] = 1;
    /// 從 now 更新其他點的距離 ///
    for (auto [x, cost] : G[now]) {
        dis[x] = min(dis[x], dis[now] + cost);
    }
}
```

```cpp
vector<int> vis, dis;
void dfs_sptree(int now) {
    vis[now] = 1;
    for (auto [x, cost] : G[now]) {
        /// 如果走 (now, x) 這條邊是最短路徑 ///
        if (!vis[x] and dis[x] == dis[now] + cost) {
            /// 建最短路徑樹 ///
            adj[now].emplace_back(x, cost);
            adj[x].emplace_back(now, cost);
            dfs_sptree(x);
        }
    }
}
```

1 子任務 1 的實作

樹建完之後，對每個點作為起點做一次 DFS。

找到所有起點為 i 且經過 P 個點的終點 j 們，更新最長路徑以及數量。

```cpp
vector<int> len, dis;
void dfs_ans(int now, int par) {
    for (auto [x, cost] : adj[now]) {
        if (x == par) continue;
        len[x] = len[now] + 1;
        dis[x] = dis[now] + cost;
        dfs_ans(x, now);
    }
}
```

```cpp
/// 紀錄最長的路徑長度以及數量 ///
int max_dis = 0, max_cnt = 0;
for (int i = 1; i <= N; ++i) {
    len.assign(N+1, 0);
    dis.assign(N+1, 0);
    dfs_ans(i, 0);
    for (int j = 1; j <= N; ++j) {
        if (len[j] == P - 1) {
            if (dis[j] > max_dis) {
                max_dis = dis[j];
                max_cnt = 1;
            }
            else if (dis[j] == max_dis) {
                ++max_cnt;
            }
        }
    }
}
```

1 子任務 1 的實作

樹建完之後，對每個點作為起點做一次 DFS。

找到所有起點為 i 且經過 P 個點的終點們，更新最長路徑以及數量。

Accepted (partially)

得分：30 分

```
1 vector<int> len, dis;
2 void dfs_ans(int now, int par) {
3     for (auto [x, cost] : adj[now]) {
4         if (x == par) continue;
5         len[x] = len[now] + 1;
6         dis[x] = dis[now] + cost;
7         dfs_ans(x, now);
8     }
9 }
```

```
1  // 紀錄最長路徑及數量 ///
2  int max_dis = 0, max_cnt = 0;
3  for (int i = 1; i <= N; ++i) {
4      len.assign(N+1, 0);
5      dis.assign(N+1, 0);
6      dfs_ans(i, 0);
7      for (int j = 1; j <= N; ++j) {
8          if (len[j] == P - 1) {
9              if (dis[j] > max_dis) {
10                 max_dis = dis[j];
11                 max_cnt = 1;
12             }
13             else if (dis[j] == max_dis) {
14                 ++max_cnt;
15             }
16         }
17     }
18 }
```

子任務 2

$N \leq 30\,000$

2 子任務 2 的解法

「建出最短生成樹」以及「找出樹上經過 P 個點的路徑們」兩個部分的複雜度皆為 $\mathcal{O}(N^2)$。

跟上一題一樣，我們可以將兩個部分都進行優化來降低複雜度。

2 子任務 2 的解法

「建出最短生成樹」的複雜度可以降低至 $O(M \log M)$。

回想一下，最短路的 Dijkstra 算法長怎樣？最容易超時的步驟是什麼？

2 子任務 2 的解法

「建出最短生成樹」的複雜度可以降低至 $\mathcal{O}(M \log M)$。

我們會維護「已經求出最短路」的集合 S 以及剩下的點 $T = V \setminus S$，一開始 $S = \emptyset$：

1. 先初始化起點 s 距離 $\mathrm{dis}[s] = 0$，而其他點的距離 $\mathrm{dis}[v] = \infty$。
2. 求出集合 T 裡距離最小的點 v（$v = \arg\min_{t \in T}\{\mathrm{dis}[t]\}$），並把 v 移到 S。
3. 對所有起點是 v 的邊去更新其他點的最短路。
4. 回到 (2.)

之前我們用了迴圈暴力枚舉，可是我們明明有 `std::priority_queue`（堆）可以用阿！

2 子任務 2 的 Dijkstra 實作

如果沒有前面的 using，那麼我會需要打下面那一長串東西來宣告一個先按照第一個再按照第二個元素由小排到大的小根堆。

```
priority_queue<
    pair<int, int>,
    vector<pair<int, int>>,
    greater<pair<int, int>>
> pq;
```

珍愛生命，使用 using（？）

```cpp
/// 因為 priority_queue 的宣告很長，所以先定義 prior 是小根堆 ///
template <typename T>
using prior = std::priority_queue<T, vector<T>, greater<T>>;

vector<int> dis; /// dis[1] = 0; else INF
prior<pair<int, int>> pq;

pq.emplace(dis[0], 0);
while (!pq.empty()) {
    /// 把距離最短的點取出來 ///
    auto [_d, now] = pq.top(); pq.pop();
    /// 但是他可能已經被加進 S 了 ///
    if (dis[now] != _d) continue;
    /// 枚舉出邊，如果能更新最短距離就把新的距離跟點推進去 ///
    for (auto [x, cost] : G[now]) {
        if (dis[x] > dis[now] + cost) {
            dis[x] = dis[now] + cost;
            pq.emplace(dis[x], x);
        }
    }
}
```

2 子任務 2 的解法

「找出樹上經過 P 個點的路徑們」的複雜度可以降低至 $\mathcal{O}(N \log N)$。

先把問題稍微簡化一點，只考慮「有多少組點對的距離是 $P-1$」。

當然可以暴力，但此處我們考慮使用樹 DP：

- 以 1 號節點為根，對每個點 v 以 $\mathrm{cnt}_v[\ell]$ 紀錄從 v 往下走長度為 ℓ 的路徑有幾條。
  - $\mathrm{cnt}_v[\ell] = \sum_{c \in \mathrm{ch}[v]} cnt_c[\ell - 1]$，其中 $\mathrm{ch}[v]$ 代表 v 的兒子們。
- 在要從一個點離開時計算有幾條長度為 $P-1$ 的路徑上深度最小的點是自己。
  - 因為一條路徑上的最高點唯一，所以所有路徑一定會被算到恰一次。

2 子任務 2 的解法

以 1 號節點為根，對每個點 v 以 $\mathrm{cnt}_v[\ell]$ 紀錄從 v 往下走長度為 ℓ 的路徑有幾條。

在要從一個點離開時計算有幾條長度為 $P-1$ 的路徑上深度最小的點是自己。

- 對每組子樹們的 pair，計算左邊走 p 條右邊走 $P-p-3$ 條邊的答案數量。
  - 總共經過的邊是「左邊 p 條」、「右邊 $P-p-3$ 條」、「左邊到自己到右邊 2 條」。
  - 這樣是 $\mathcal{O}(\sum_v \deg[v]^2 \cdot P) = \mathcal{O}(N^3)$。

2 子任務 2 的解法

以 1 號節點為根，對每個點 v 以 $\mathrm{cnt}_v[\ell]$ 紀錄從 v 往下走長度為 ℓ 的路徑有幾條。

在要從一個點離開時計算有幾條長度為 $P-1$ 的路徑上深度最小的點是自己。

- 對每組子樹們的 pair，計算左邊走 p 條右邊走 $P-p-3$ 條邊的答案數量。
  - 總共經過的邊是「左邊 p 條」、「右邊 $P-p-3$ 條」、「左邊到自己到右邊 2 條」。
  - 這樣是 $\mathcal{O}(\sum_v \deg[v]^2 \cdot P) = \mathcal{O}(N^3)$。
- 對每個子樹，計算該子樹走 p 條其他子樹走 $P-p-3$ 條邊的答案數量。
  - 這樣是 $\mathcal{O}(\sum_v \deg[v] \cdot P) = \mathcal{O}(N^2)$。

2 子任務 2 的解法

以 1 號節點為根，對每個點 v 以 $\mathrm{cnt}_v[\ell]$ 紀錄從 v 往下走長度為 ℓ 的路徑有幾條。

在要從一個點離開時計算有幾條長度為 $P-1$ 的路徑上深度最小的點是自己。

- 對每組子樹們的 pair，計算左邊走 p 條右邊走 $P-p-3$ 條邊的答案數量。
  - 總共經過的邊是「左邊 p 條」、「右邊 $P-p-3$ 條」、「左邊到自己到右邊 2 條」。
  - 這樣是 $\mathcal{O}(\sum_v \deg[v]^2 \cdot P) = \mathcal{O}(N^3)$。
- 對每個子樹，計算該子樹走 p 條其他子樹走 $P-p-3$ 條邊的答案數量。
  - 這樣是 $\mathcal{O}(\sum_v \deg[v] \cdot P) = \mathcal{O}(N^2)$。
- 還有些小 case 要判，像是只走一邊、答案會被算到兩次等等。

2 子任務 2 的解法

以 1 號節點為根，對每個點 v 以 $\mathrm{cnt}_v[\ell]$ 紀錄從 v 往下走長度為 ℓ 的路徑有幾條。

在要從一個點離開時計算有幾條長度為 $P-1$ 的路徑上深度最小的點是自己。

介紹一個酷東西：樹上啓發式合併。英文叫 DSU on Tree 但我不知道他怎麼像 DSU 了。

2 樹上啓發式合併

定義重兒子 heavy_v 是所有 v 的兒子裡子樹最大的（任選）一個，其他則叫輕兒子。

1. 算完輕兒子們分別內部的答案。

   他的概念是對每個點 v 先往輕兒子們遞迴處理，但是在處理完之後會把該子樹的所有資訊都刪掉。

2. 算完重兒子內部的答案。

   接著往重兒子遞迴，並把重兒子的資料都留下來。

3. 計算剩下的答案。

   依序往每個輕兒子遞迴，這次資料都會被存下來。

2 樹上啓發式合併

而這個做法的複雜度是 $\mathcal{O}(N \log N)$ ！因為他利用了啓發式合併每次大小會變至少兩倍的性質。

這樣輕兒子只會被往上合併 $\mathcal{O}(\log N)$ 次，總複雜度是 $\mathcal{O}(N \log N)$。

很難理解？沒關係，來看看 code。

2 樹上啓發式合併－實作

計算答案的 dfs_calc、

加子樹資料的 dfs_add、

還有一開始計算重兒子的 dfs_heavy、

這個啓發式合併 dfs、

以及建最短路徑樹的 dfs_sptree，

這題總共寫了五種不同的 DFS。

（可以更少但可能不會比較好看？）

```cpp
deque<int> dp; /// dp[i] 存長度為 i 的路徑數量
/// 用 bool keep 來記錄「要不要存下資料」 ///
void dfs(int now, bool keep) {
    /// 第一遍：往輕兒子遞迴，不要留下資料 ///
    for (auto [x, cost] : adj[now]) {
        if (x == heavy[now]) continue;
        dfs(x, false);
    }
    /// 第二遍：往重兒子遞迴，留下資料 ///
    if (heavy[now] != -1) dfs(heavy[now], true);
    /// 從 heavy[now] 來的路徑長都 + 1，並為長度 0 的答案（自己）+ 1 ///
    dp.emplace_front(1);
    /// 如果存在長度為 P-1 的路徑就更新答案 ///
    if ((int)dp.size() > P-1) ans += dp[P-1];
    /// 第三遍：往輕兒子遞迴，把資料加進去 ///
    for (auto [x, cost] : adj[now]) {
        if (x == heavy[now]) continue;
        dfs_calc(x, 1, cost); /// 計算該子樹跟目前的 dp 可以得到的答案
        dfs_add(x, 1, cost); /// 把子樹的資料加進去
    }
    /// 如果 keep == false 就代表不需要這個子樹的資料 ///
    /// 本題中可以直接 clear 掉，但有些題目需要再寫一個 dfs_del() ///
    if (!keep) dp.clear();
}
```

2 樹上啓發式合併－實作

計算答案的 dfs_calc、

加子樹資料的 dfs_add、

還有一開始計算重兒子的 dfs_heavy、

這個啓發式合併 dfs、

以及建最短路徑樹的 dfs_build_tree，

這題總共寫了五種不同的 DFS。

（可以更少但可能不會比較好看？）

Accepted

得分：100分

```
deque<int> dp; /// dp[i] 存長度為 i 的路徑數量
/// global 變數小心不要算錯 ///
void dfs(int now, bool keep) {
    /// 第一遍：往輕兒子遞迴，不保留資料 ///
    for (auto [x, cost] : adj[now]) {
        if (x == heavy[now]) continue;
        dfs(x, false);
    }
    /// 第二遍：往重兒子遞迴，留下資料 ///
    if (heavy[now] != -1) dfs(heavy[now], true);
    /// 從 heavy[now] 來的路長都 + 1，並為長度 0 的答案（自己）+ 1 ///
    dp.emplace_front(1);
    /// 如果存在長度 P-1 的路徑就更新答案 ///
    if ((int)dp.size() > P-1) ans += dp[P-1];
    /// 第三遍：往輕兒子遞迴，把資料加進去 ///
    for (auto [x, cost] : adj[now]) {
        if (x == heavy[now]) continue;
        dfs_calc(x, 1, cost); /// 計算該子樹跟目前的 dp 可以得到的答案
        dfs_add(x, 1, cost); /// 把子樹的資料加進去
    }
    /// 如果 keep == false 就代表不需要這個子樹的資料 ///
    /// 本題中可以直接 clear 掉，但有些題目需要再寫一個 dfs_del() ///
    if (!keep) dp.clear();
}
```

2 子任務 2 的解法

ㄟ等等，剛剛不是只有考慮到「有多少組點對的距離是 $P-1$」的 case？

實際上要計算最長路徑的數量，只要用 {長度,數量} 來儲存 DP 值跟答案，每當修改了長度就把數量歸零即可。

不過這樣 code 就會多出一堆 `.first`、`.second` 變得擁擠難以閱讀，所以具體實作還請參考後面的 AC Code。

結論

3 結論

這題能 30 分就已經很有難度了，有能力在比賽時間內破台的應該都能進全國賽前二等獎勹。

最短路是校內、東區、全國賽等等地方的常考題，倒不如說在圖論裡他是除了遍歷方法之外能考出來最簡單的東西了（？）

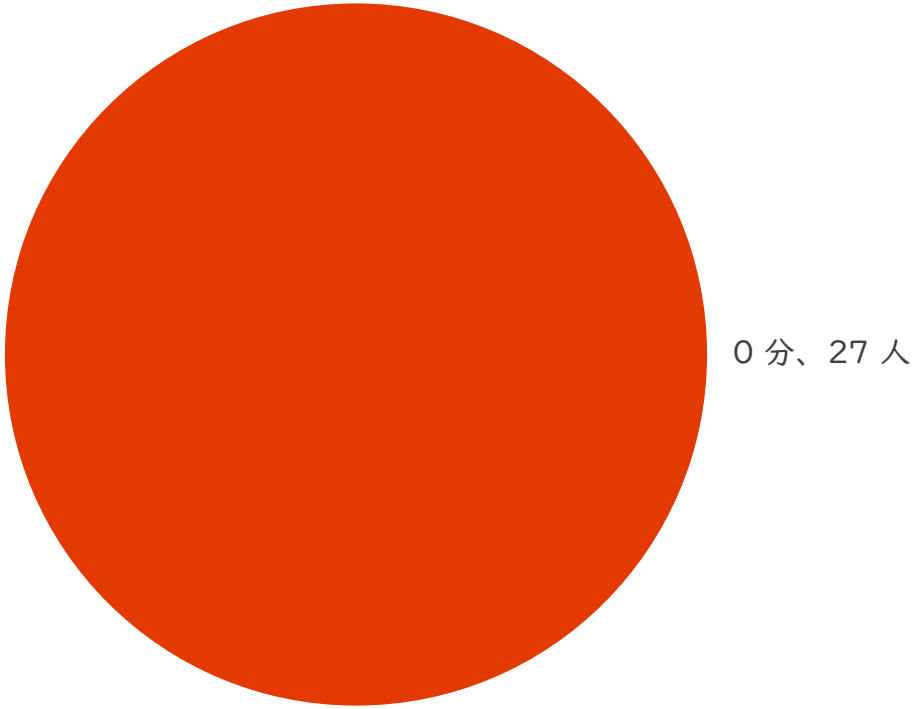
樹上啓發式合併則算是中偏難的等級，建議可以自己從頭寫寫看模板題。

希望汝在這題中學到的知識：

- 最短路徑的求法。不只有 Dijkstra，還有 Bellman-Ford 跟 Floyd-Warshall。
- 啓發式合併的又一種用法。

AC Code 連結

3 成績分布



3 額外補充

「建出最短生成樹」的複雜度可以降低至 $\mathcal{O}(N \log N + M)$。

詳情請查看 Fibonacci Heap。

此外，並不是在任何情況下使用 heap 來實作 Dijkstra 都是最好的。當圖接近完全圖，也就是 $m = \mathcal{O}(n^2)$ 的時候，使用暴力來實作會比用 heap 少一個 $\log m$。