Homework #1

Due Time: 2025/03/05 (Wed.) 21:59
Contact TAs: `vegetable@csie.ntu.edu.tw`

Instructions and Announcements

- **NO LATE SUBMISSION OR PLAGIARISM IS ALLOWED.**
- Discussions with others are encouraged. However, you should write down your solutions **in your own words**. In addition, for **each and every** problem you have to specify the references (the URL of the web page you consulted or the people you discussed with) on the first page of your solution to that problem.
- Some problems below may not have standard solutions. We will give you the points if your answer is followed by reasonable explanations.

Submission

- Please place your answers in the same order as the problem sheet and do not repeat problem descriptions, just organize them by problem number in a tidy manner.
- Please ~~submit your pdf report via Gradescope. And,~~ zip the pdf file and shell script, name the zip file "`{your_student_id}.zip`", and submit it via NTU COOL. The directory layout should be the same as listed below:

```
{your_student_id}/
+-- merkle-dir.sh
+-- report.pdf
```

Grading

- There are some tests for each of the first to the fourth parts. You'll get the score of a part iff you pass all the tests of that part.
- We'll use `diff` command to compare your output and the answer of a test. Passing a test means that your output is the same as the answer.
- There are two kinds of tests: public tests and private tests. Public tests are released in `public-testcase.zip`, where you can use `test.sh` to test your script, while private tests and solution will (probably) be released after the homework deadline.
- You should list the references in the report in order to get full credits on it.
- The raw total score for the correctness and completeness of your shell script and report is 120 points, with the shell script and the report contributing 110 points and 10 points, respectively. However, the total score will be capped at 100 points if it exceeds 100.
- Tidiness score: 3 bonus points, graded by TA.
- Final score = $\min(100, \text{script score} + \text{report score}) + \text{tidiness score}$.

Shell Scripting: Merkle Tree

本題的目標是要寫出 shell script `merkle-dir.sh`。這個程式將能建立一個目錄 (directory) 的 (binary) Merkle tree，以及生成/驗證目錄內任意 regular file 的 inclusion proof。

0 注意事項

1. 請在 script 前加上 shebang 並開啟 script 的執行權限 (`chmod +x`)。
2. 你的 script 必須能使用下面幾種 shell 中其中之一執行: bash, sh, zsh, fish, tcsh, ksh。
3. 我們會在 NASA 工作站 (nasaws*) 測試你的 script，請確認所有使用到的指令在工作站上皆可以正常運作。
4. 本題除 report 外須全部使用 shell script 完成，不可在中途執行自己的 Python 或其他程式語言的腳本或者程式。
5. 請勿在 shell script 內新增題目要求之外的檔案，也請勿夾雜惡意程式碼，並在**一分鐘**內成功執行。**任何被判定為惡意程式碼、或者以不公平的方式作答或者影響批改，最嚴重的狀況可能導致作業 0 分、或甚至期末獲得不及格的成績**。
6. 如果不確定是否可以使用某些指令，請寄信 (或於 NTU COOL 討論區) 詢問助教。
7. 本作業中的檔案只會出現三種：regular file, symbolic link file, directory file (定義可以參考這個連結)。
8. 檔案名稱只含有 `[0-9a-zA-Z]` 、`.`、`-` 和 `_` 字元，且檔名開頭不會是 `-` 。
9. 以下子題 1-4 是以逐步加上功能的方式進行。也就是完成子題 1 的功能後，子題 2 將子題 1 的 `merkle-dir.sh` 再加上新的功能。最後只需要繳交完成版的 `merkle-dir.sh` (以及 report)，但是我們會針對每個子題的需求逐一驗證進行評分。
10. 如果沒有額外要求，你的 script 結束執行時請回傳 0 作為 exit status。

1 參數檢查 (30pts)

在這個小題中，主要的目標是希望你處理 `merkle-dir.sh` 這個指令的輸入是否符合格式。首先，請觀看下列方框中 `merkle-dir.sh` 指令的使用說明。

```
merkle-dir.sh - A tool for working with Merkle trees of directories.

Usage:
  merkle-dir.sh <subcommand> [options] [<argument>]
  merkle-dir.sh build <directory> --output <merkle-tree-file>
  merkle-dir.sh gen-proof <path-to-leaf-file> --tree <merkle-tree-file> --output <proof-
      ↪ file>
  merkle-dir.sh verify-proof <path-to-leaf-file> --proof <proof-file> --root <root-hash>

Subcommands:
  build           Construct a Merkle tree from a directory (requires --output).
  gen-proof       Generate a proof for a specific file in the Merkle tree (requires --tree
      ↪  and --output).
  verify-proof    Verify a proof against a Merkle root (requires --proof and --root).

Options:
  -h, --help      Show this help message and exit.
  --output FILE   Specify an output file (required for build and gen-proof).
  --tree FILE     Specify the Merkle tree file (required for gen-proof).
  --proof FILE    Specify the proof file (required for verify-proof).
  --root HASH     Specify the expected Merkle root hash (required for verify-proof).

Examples:
  merkle-dir.sh build dir1 --output dir1.mktree
  merkle-dir.sh gen-proof file1.txt --tree dir1.mktree --output file1.proof
  merkle-dir.sh verify-proof dir1/file1.txt --proof file1.proof --root abc123def456
```

`merkle-dir.sh` 指令中，`subcommand` 用來表示 script 現在應該執行的模式，而 `options` 則是表示針對該執行模式所需的額外的一些選項，針對那個模式有時候是一定需要，但也有可能有些是可有可無。最後，`argument` 則是表示針對該執行模式固定所需的輸入資料。

`merkle-dir.sh` 支援三種模式：(1) 針對給定的目錄生成 Merkle tree file、(2) 針對給定的檔案生成 inclusion proof、(3) 驗證 inclusion proof。你的 script 必須分別針對三種模式檢查輸入是否正確。

`merkle-dir.sh` 指令的格式為：

```
merkle-dir.sh <subcommand> [options] [<argument>]
```

其中 `argument` 可以在 `options` 之前、之間、或之後出現，但 `argument` 和 `options` 都必須出現在 `subcommand` 之後。

例如，在下面的例子中，`gen-proof` 是 `subcommand`，`--tree dir1.mktree` 和`--output file1.proof` 是 `options` ，`file1` 是 `argument`。則以下指令的格式皆合法：

```
merkle-dir.sh gen-proof file1 --tree dir1.mktree --output file1.proof
merkle-dir.sh gen-proof --output file1.proof file1 --tree dir1.mktree
merkle-dir.sh gen-proof --tree dir1.mktree --output file1.proof file1
```

`merkle-dir.sh <subcommand> [options] [<argument>]` 為合法指令若且唯若以下任一條件成立：

1. `subcommand` 與 `argument` 為空，且 `options` 只有`-h` 或`--help` 其中一個。
2. `subcommand` 為 `build`，且符合以下所有條件：
   - `options` 只有`--output <file>`，其中 `<file>` 不存在或為已存在之 regular file。

- `argument` 只有一個，且為已存在之 directory file。

3. `subcommand` 為 `gen-proof`，且符合以下所有條件：
   - `options` 同時有`--output <file1>` 與`--tree <file2>`，其中 `<file1>` 不存在或為已存在之 regular file，`<file2>` 為已存在之 regular file。
   - `argument` 只有一個。
4. `subcommand` 為 `verify-proof`，且符合以下所有條件：
   - `options` 只有`--proof <file>` 與`--root <hash>`，其中 `<file>` 為已存在之 regular file，`<hash>` 只含有數字 0 至 9 及字母 A 到 F，使用的字母為全大寫或全小寫。
   - `argument` 只有一個，且為已存在之 regular file。

本小題只要求你的 shell script 在對的時機輸出使用說明並回傳正確的 exit status。具體而言：

- 若輸入的參數不是合法指令，請輸出使用說明至 stdout，同時以 1 作為 exit status 結束執行程式。
- 若輸入的參數符合第 1 個合法指令的定義 (`subcommand` 與 `argument` 為空且 `options` 只有`-h` 或`--help` 其中一個)，則輸出使用說明至 stdout，同時以 0 作為 exit status 結束執行程式。
- 否則，不輸出使用說明並且繼續執行程式。
- 注意：在我們的測試中，不會有同樣的 `[options]` 重複出現的狀況。

2 Building a Merkle Tree (40pts)

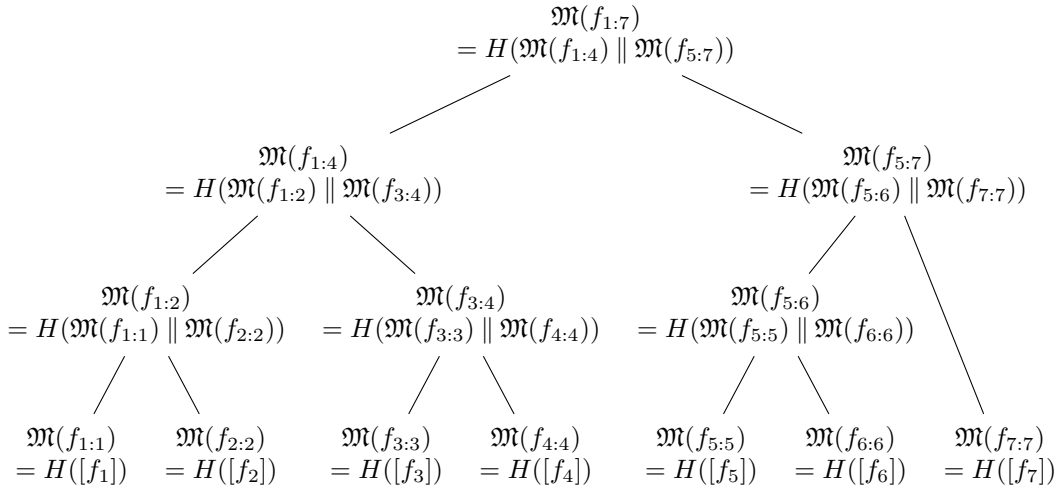
符號定義

- 對於每個檔案 f ，我們使用其路徑來表示它。因此本題中的" 檔案 f" 實際上指的是" 路徑 f 上的檔案"。
- $[f]$ 表示檔案 f 的內容。
- 定義 $F(p) \coloneqq (f_1, f_2, \ldots, f_n)$ 為 directory file p 底下的所有 **regular file**(包含隱藏的 regular file，不包含 symlink) 依字典序 排列而成的有限序列。本題所指的字典序 與 `LC_COLLATE=C sort` 的排列順序相同。
- H 為 SHA256 函數。例如：$H([f])$ 與 `sha256sum f | awk '{print $1}' | xxd -r -p` 的輸出結果相同，$H(\texttt{foo})$ 與 `echo -n foo | sha256sum | awk '{print $1}' | xxd -r -p` 的輸出結果相同。
- Hex 是將 bytes 以十六進制表示的函數 (字母全小寫)。例如：$\text{Hex}(\texttt{foo})$ 與 `echo -n foo | xxd -p -c 0` 的輸出結果相同，$\text{Hex}(H(\texttt{foo}))$ 與 `echo -n foo | sha256sum | awk '{print $1}'` 的輸出結果相同。
- $\|$ 為兩個字串的串接 (concatenation)。
- $+\!\!+$ 表示 list concatenation。例如：$[1,2,3] +\!\!+ [4,5,6] = [1,2,3,4,5,6]$。
- $f_{i:j}$ 表示 $(f_i, f_{i+1}, \ldots, f_j)$。

什麼是 Merkle Tree？

給定一個目錄底下的 n 個檔案 $T \coloneqq (f_1, f_2, \ldots, f_n)$，我們希望能計算出能一次代表這 n 個檔案的簡短 hash y，同時將 y 公佈給其他人。在一段時間過後，我們要能向任何知道 y 的人證明 T 中的任一檔案 f_i 的確就在 T 之中。

這次要實作的 Merkle tree 便是能幫助我們達到上述要求的其中一種資料結構。假設 $n = 7$，T 的 Merkle tree 如下圖所示：



其中 $\mathfrak{M}$ 代表 Merkle tree hash，是能以任意多個檔案 $f_1, f_2, \ldots, f_m$ 作為輸入值，並輸出一個 hash 值的 hash function。其定義如下：

$$\mathfrak{M}(f_1, f_2, \ldots, f_m) \triangleq \begin{cases} H([f_1]) & m = 1 \\ H(\mathfrak{M}(f_{1:k}) \| \mathfrak{M}(f_{k+1:m})) \text{ where } k = 2^{\lfloor \lg(m-1) \rfloor} & m > 1 \end{cases}$$

舉例來說：

$$\begin{aligned}
\mathfrak{M}(f_1, f_2, \ldots, f_5) &= \mathfrak{M}(f_{1:5}) \\
&= \left\{\because k = 2^{\lfloor \lg (5-1) \rfloor} = 4\right\} \\
&\quad H(\mathfrak{M}(f_{1:4}) \parallel \mathfrak{M}(f_{5:5})) \\
&= \left\{\because k = 2^{\lfloor \lg (4-1) \rfloor} = 2\right\} \\
&\quad H(\underbrace{H(\mathfrak{M}(f_{1:2}) \parallel \mathfrak{M}(f_{3:4}))}_{\mathfrak{M}(f_{1:4})} \parallel \mathfrak{M}(f_{5:5})) \\
&= \left\{\because k = 2^{\lfloor \lg (2-1) \rfloor} = 1\right\} \\
&\quad H(H(\underbrace{H(\mathfrak{M}(f_{1:1}) \parallel \mathfrak{M}(f_{2:2}))}_{\mathfrak{M}(f_{1:2})} \parallel \underbrace{H(\mathfrak{M}(f_{3:3}) \parallel \mathfrak{M}(f_{4:4}))}_{\mathfrak{M}(f_{3:4})}) \parallel \mathfrak{M}(f_{5:5})) \\
&= H(H(H(H([f_1]) \parallel H([f_2])) \parallel H(H([f_3]) \parallel H([f_4]))) \parallel H([f_5]))
\end{aligned}$$

我們稱 $\mathfrak{M}(f_{1:n})$ 為 *root hash*，即 T 的 Merkle tree 的根節點上存的 hash 值。另外，T 的 Merkle tree 中的所有節點，為遞迴展開 $\mathfrak{M}(f_{1:n})$ 的定義時出現的所有 $\mathfrak{M}(f_{i:j})$。例如，假設 $n = 2^l$，$l \in \mathbb{N}$，則 T 的 Merkle tree 的所有節點為：

$$\begin{gathered}
H([f_1]), H([f_2]), \ldots, H([f_n]), \\
\mathfrak{M}(f_{1:2}), \mathfrak{M}(f_{3:4}), \ldots, \mathfrak{M}(f_{n-(2^1-1):n}), \\
\mathfrak{M}(f_{1:4}), \mathfrak{M}(f_{5:8}), \ldots, \mathfrak{M}(f_{n-(2^2-1):n}), \\
\cdots \\
\mathfrak{M}(f_{1:\frac{n}{2}}), \mathfrak{M}(f_{\frac{n}{2}+1:n}), \\
\mathfrak{M}(f_{1:n})
\end{gathered}$$

題目要求

本小題的目標是實作 `merkle-dir.sh build` 的功能，因此使用的指令皆為

```
./merkle-dir.sh build <directory> --output <merkle-tree-file>
```

其中 `build` 是 `subcommand`，`--output <merkle-tree-file>` 是 `options`，`<directory>` 是 `argument`。

在判斷指令合法之後，請依以下規定輸出 $F(p) \coloneqq (f_1, f_2, \ldots, f_n)$ 的 Merkle tree 到檔案 `<merkle-tree-file>`，其中 p 為指令中 `<directory>` 的路徑：

- 輸出的前 n 行中的第 i 行，應為 $F(p)$ 中 f_i **相對於 p 的路徑**，$1 \le i \le n$。請注意 f_i 即路徑 p 的 directory file 底下第 i 個 regular file(以字典序排序)。注意 p 底下可能還有其他目錄，其中也會包含其他目錄和檔案，應進行遞迴處理。
- 第 $n+1$ 行為空行。
- 從第 $n+2$ 行開始的 $\lceil \lg n \rceil + 1$ 行，應輸出 $F(p)$ 的 Merkle tree 的各個高度上所有節點的值。這其中的第 k 行，也就是整個檔案的第 $(n+2)+k$ 行，$0 \le k \le \lceil \lg n \rceil$：
  - Case 1: 若 $k = 0$, 或者 $k \ge 1$ 且 $n \bmod 2^k \le 2^{k-1}$ 則為：

    $$\mathrm{Hex}(\mathfrak{M}(f_{1:2^k})) : \mathrm{Hex}(\mathfrak{M}(f_{2^k+1:2\cdot 2^k})) : \ldots : \mathrm{Hex}(\mathfrak{M}(f_{(\lfloor n/2^k \rfloor - 1)\cdot 2^k + 1:\lfloor n/2^k \rfloor \cdot 2^k})) \qquad (*)$$

    (換句話說，就是 $F(p)$ 的 Merkle tree 中所有符合 $2^{k-1} < j - i + 1 \le 2^k$ 的節點 $\mathfrak{M}(f_{i:j})$ 轉換成十六進位後依照 i 的大小排成的 array，各項間以 : 相連。)

– Case 2: 若 $k \geq 1$ 且 $n \bmod 2^k > 2^{k-1}$，則為：

$$(*) : \mathrm{Hex}(\mathfrak{M}(f_{\lfloor n/2^k \rfloor \cdot 2^k + 1:n}))$$

(一樣包含$(*)$，也就是 Case 1 的式子，但末端多一項。)

範例

假設 `dir1` 目錄的內容如 figure 1。(其中檔案的內容請參閱`public-testcase.zip`。)

Figure 1: 目錄 `dir1` 的結構

則指令

```
./merkle-dir.sh build dir1 --output dir1.mktree
```

會輸出以下內容至檔案 `dir1.mktree`:

```
code/hello.c
code/judge.py
code/run.sh
hi.txt
media/image1.jpg
media/inner/image2.png
media/inner/image3.gif

8e4615c432ba575455e36a56657920f242c4e6ef25553e72ce18fef83084c6d2:
    ↪ c1e4d9c7090927b06bf2d394cad89fb30213535719874f7fcfd224d6e427743c:1
    ↪ fb328ba2f65126f918c232cf6472c563a1f15e090d294051607b7743094682b:
    ↪ a7b271c8cec0e921cc624e0db21d864434a04c525903bc2df197c8516a8276b6:
    ↪ cc2921fb18759d587f7de99094a6c5417fdf4cf74ff08b92bbc3ec938efb4bf9:2
    ↪ a69eb97abd356255b6586b563f77d40e814bd41ae69a503ca22b10e49e67de9:
    ↪ ea58dc841d4df26457deff35e4e0e05935663c5489e8e4e5159a8cbdea74a8ef
```

```
10 027bf3cfe0826beba2cb24608ba43551b72988aa0babac896169cc877f59f7b9:8
     ↪ cbdd9b8a078a7c3ac8e02307e5ec41e51f9212593f6434599bd223b64416eb7:808
     ↪ c79213372250712763cf36ba3933ab73646505415df7e68a87b5d25941c77
11 9e233136e7e103e3ab852d16de432a37ccf934d8277cbf5b8bb516ac9a3db96e:3606
     ↪ efb5d124ab5308089042c46353f35d4deb443a3619330a6bd32ce7829c85
12 61972f8a5bf9925d70d34934dafced66756c5f9d7a80e3b265e61e5526155f83
```

檔案內容解釋：有 7 個 regular file，因此 $n = 7$。

- 第 1-7 行: `dir1` 內 7 個 regular file 相對於 `dir1` 的路徑
- 第 8 行: 沒東西
- 第 9 行: $k = 0$，為 Case 1。本行輸出 Merkle tree 中所有葉節點的值 (以 hex 表示)，即 $\mathrm{Hex}(\mathfrak{M}(f_{1:1})) : \ldots : \mathrm{Hex}(\mathfrak{M}(f_{7:7}))$
- 第 10 行: $k = 1$，此時 $7 \mod 2^1 = 1 \leq 2^{1-1}$，因此為 Case 1。本行輸出 Merkle tree 中所有底下有 2^1 個葉節點的節點的值 (以 hex 表示)，即 $\mathrm{Hex}(\mathfrak{M}(f_{1:2})) : \mathrm{Hex}(\mathfrak{M}(f_{3:4})) : \mathrm{Hex}(\mathfrak{M}(f_{5:6}))$。
- 第 11 行: $k = 2$，此時 $7 \mod 2^2 = 3 > 2^{2-1}$, 因此為 Case 2。本行輸出 Merkle tree 中所有底下有 $2^1 + 1 \sim 2^2$ 個葉節點的節點的值 (以 hex 表示)，即 $\mathrm{Hex}(\mathfrak{M}(f_{1:4})) : \mathrm{Hex}(\mathfrak{M}(f_{5:7}))$。
- 第 12 行: $k = 3$，此時 $7 \mod 2^3 = 7 > 2^{3-1}$, 因此為 Case 2。本行輸出 Merkle tree 中所有底下有 $2^2 + 1 \sim 2^3$ 個葉節點的節點的值 (以 hex 表示)，即 $\mathrm{Hex}(\mathfrak{M}(f_{1:7}))$。

注意事項與提示

- `<directory>` 底下的檔案不超過 1000 個。
- 測資中不會有 `<merkle-tree-file>` 在 `<directory>` 底下的情況。
- 只要指令是合法的 (關於合法的定義請見參數檢查)，我們會確保使用者擁有足夠權限寫入或創建參數中指定的檔案。
- 如果輸出格式正確，`<merkle-tree-file>` 的最後一行將是 root hash 的十六進位表示。
- 記得確認 $F(p)$ 的定義！
- `find path/to/dir | LC_COLLATE=C sort` 列出檔案的順序可能與 `tree path/to/dir` 不同。

3 Generating an Inclusion Proof (20pts)

什麼是 Inclusion Proof？

給定 n 個檔案 $(f_1, f_2, \ldots, f_n)$，第 i 個檔案 f_i $(1 \le i \le n)$ 在 $f_{1:n}$ 的 Merkle tree 中的 *inclusion proof* 為 $\pi(i, f_{1:n})$ 。而 $\pi()$ 的定義如下：

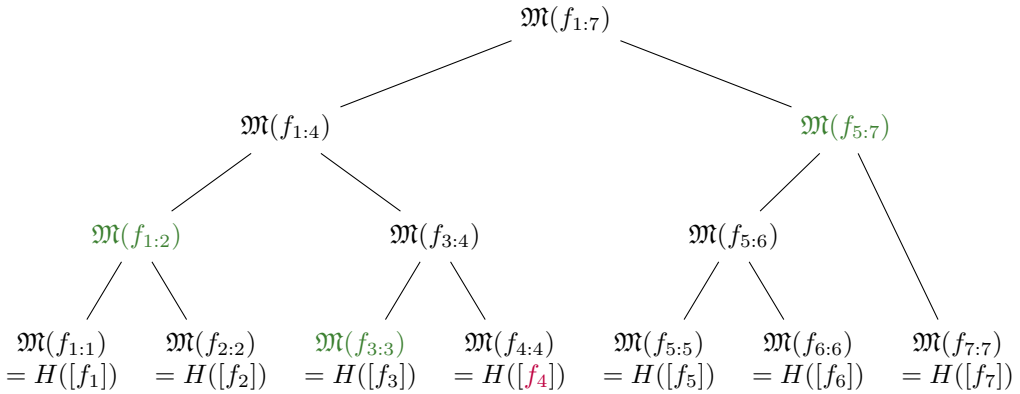
$$\pi(j, f_{s:s+m-1}) \triangleq \begin{cases} [\,] & m = 1 \\ \pi(j, f_{s:s+k-1}) + [\mathfrak{M}(f_{s+k:s+m-1})] & m \ge 2 \text{ and } j \le k \\ \pi(j-k, f_{s+k:s+m-1}) + [\mathfrak{M}(f_{s:s+k-1})] & m \ge 2 \text{ and } j > k \end{cases}$$

where $k = 2^{\lfloor \lg (m-1) \rfloor}$ and $[\,]$ 表示 empty list。**注意這是個遞迴定義。**其他符號的定義請見符號定義。

舉例來說，f_4 在 $(f_1, f_2, \ldots, f_7))$ 的 Merkle tree 中的 inclusion proof $\pi(4, f_{1:7})$ 為：

$$\begin{aligned}
\pi(4, (f_1, f_2, \ldots, f_7)) &= \pi(4, f_{1:7}) \\
&= \left\{ \because k = 2^{\lfloor \lg (7-1) \rfloor} = 4 \text{ and } 4 \le k \right\} \\
&\quad \pi(4, f_{1:4}) + [\mathfrak{M}(f_{5:7})] \\
&= \left\{ \because k = 2^{\lfloor \lg (4-1) \rfloor} = 2 \text{ and } 4 > k \right\} \\
&\quad \underbrace{\pi(2, f_{3:4}) + [\mathfrak{M}(f_{1:2})]}_{\pi(4, f_{1:4})} + [\mathfrak{M}(f_{5:7})] \\
&= \left\{ \because k = 2^{\lfloor \lg (2-1) \rfloor} = 1 \text{ and } 2 > 1 \right\} \\
&\quad \underbrace{\pi(1, f_{4:4}) + [\mathfrak{M}(f_{3:3})]}_{\pi(2, f_{3:4})} + [\mathfrak{M}(f_{1:2})] + [\mathfrak{M}(f_{5:7})] \\
&= \{ \because n = 1 \} \\
&\quad \underbrace{[\,]}_{\pi(1, f_{4:4})} + [\mathfrak{M}(f_{3:3})] + [\mathfrak{M}(f_{1:2})] + [\mathfrak{M}(f_{5:7})] \\
&= [\, \mathfrak{M}(f_{3:3}) \,,\, \mathfrak{M}(f_{1:2}) \,,\, \mathfrak{M}(f_{5:7}) \,]
\end{aligned}$$

由 $f_{1:7}$ 的 Merkle tree (下圖) 可以看出，任何人只要有了f_4與它的 inclusion proof $\pi(4, f_{1:7})$，他就可以算出這個 Merkle tree 的 root hash $\mathfrak{M}(f_{1:7})$：



因此，透過提供檔案 f 與 inclusion proof $\pi(i, f_{1:n})$ ，我們能向其他人證明 $f = f_i$。

題目要求

本小題的目標是實作 `merkle-dir.sh gen-proof` 的功能，因此使用的指令皆為

```
./merkle-dir.sh gen-proof <path-to-leaf-file> --tree <merkle-tree-file>
  ↪ --output <proof-file>
```

`gen-proof` 是 subcommand，`--tree <merkle-tree-file> --output <proof-file>` 是 options，`<directory>` 是 argument。其中 `<merkle-tree-file>` 為上一小題輸出的合法檔案，包含了某個 directory file p 底下所有 regular file (即 $F(p)$) 的相對路徑與 $F(p)$ 的 Merkle tree。令 $n := |F(p)|$，即 $F(p)$ 的 Merkle tree 的檔案數量。

在判斷參數合法之後，請先判斷 `<path-to-leaf-file>` 是否剛好與檔案 `<merkle-tree-file>` 的前 n 行裡的其中一行相同。

- 如果不是，表示 `<path-to-leaf-file>` 不屬於 $F(p)$，也就不屬於 `<merkle-tree-file>` 表示的 Merkle tree。因此，請輸出 `ERROR: file not found in tree` 至 stdout，並以 1 作為 exit status 結束執行程式。
- 如果是，假設 `<path-to-leaf-file>` 出現於第 i 行，表示這個檔案為 $F(p)$ 中第 i 個檔案。假設這個檔案在 $F(p)$ 的 Merkle tree 中的 inclusion proof $\pi(i, F(p)) := [h_1, h_2, \ldots, h_m]$ ，請依以下格式輸出 inclusion proof 至檔案 `<proof-file>`：
  - 第 1 行為 `leaf_index:`i`,tree_size:`n。
  - 接下來的 m 行輸出 $[h_1, h_2, \ldots, h_m]$ 的 hex 值，其中第 i 行為 $\mathrm{Hex}(h_i)$。(換句話說：對 $\pi(i, F(p))$ 裡的每一個 hash 值都各輸出一行，並以十六進位表示之，字母全小寫。)

範例

指令
`./merkle-dir.sh gen-proof hi.txt --tree dir1.mktree --output hi.txt.proof`
會輸出以下內容至檔案 `hi.txt.proof`:

```
1 leaf_index:4,tree_size:7
2 1fb328ba2f65126f918c232cf6472c563a1f15e090d294051607b7743094682b
3 027bf3cfe0826beba2cb24608ba43551b72988aa0babac896169cc877f59f7b9
4 3606efb5d124ab5308089042c46353f35d4deb443a3619330a6bd32ce7829c85
```

注意事項與提示

- 只要參數是合法的 (關於合法的定義請見參數檢查)，我們會確保使用者擁有足夠權限寫入或創建參數中指定的檔案。
- 就算不知道原先的目錄 p，也可以透過數 `<merkle-tree-file>` 裡面空行前有幾行來得知 $|F(p)|$。

4 Verifying an Inclusion Proof (20pts)

如何驗證 Inclusion Proof？

給定檔案數量 n、index $k(1 \le k \le n)$、n 個檔案 $(f_1, f_2, \ldots, f_n)$ 的 Merkle tree 的 root hash h^*、檔案 f 與 inclusion proof $\pi' \coloneqq [h_1, h_2, \ldots, h_m]$，我們可以利用以下演算法驗證 π' 為 f_k 在 $f_{1:n}$ 的 Merkle tree 中的 inclusion proof $\pi(k, f_{1:n})$，從而驗證 $f = f_k$：

Algorithm 1 Verification of an inclusion proof

Input: f: a file; $\pi' = [h_1, h_2, \ldots, h_m]$: an inclusion proof; n: a positive integer; $k \in \{1, 2, \ldots, n\}$; h^*: the root hash of the Merkle tree of files $(f_1, f_2, \ldots, f_n)$

Output: `Valid` if $\pi' = \pi(k, f_{1:n}) \wedge f = f_k$, `Invalid` otherwise

1: $k', n' \leftarrow k - 1, n - 1$
2: $h \leftarrow H([f])$
3: **for** $i \leftarrow 1$ to m **do**
4: **if** $n' = 0$ **then**
5: **return** `Invalid`
6: **end if**
7: **if** $LSB(k') = 1$ **or** $k' = n'$ **then**
8: $h \leftarrow H(h_i \parallel h)$
9: **while** $LSB(k') = 0$ **do**
10: right-shift both k' and n' by 1
11: **end while**
12: **else**
13: $h \leftarrow H(h \parallel h_i)$
14: **end if**
15: right-shift both k' and n' by 1
16: **end for**
17: **if** $n' = 0$ **and** $h = h^*$ **then**
18: **return** `Valid`
19: **else**
20: **return** `Invalid`
21: **end if**

(LSB 指的是 least significant bit。)

以下為此演算法的說明，可以忽略：

1. 初始化：
   - 輸入參數：
     - f: 待驗證的檔案，而 $H([f])$ 為 Merkle tree 的一個 leaf node
     - $\pi' = [h_1, h_2, \ldots, h_m]$: inclusion proof，即從該 leaf node 到 root node 每一層的 sibling node 所存的 hash 值
     - n: Merkle tree 的總檔案數 (即 leaf node 個數)
     - k: 檔案 f 在所有檔案中的 index (1-based indexing)，表示 f 是第 k 個檔案，即 $H([f])$ 為 Merkle tree 的第 k 個 leaf node
     - h^*: 已知的 Merkle tree 的 root hash
   - 第 1 行: 將 f 的 index k 由 1-based indexing 轉換為 0-based indexing，同時令 n' 為最後一個檔案 (或最後一個 leaf node) 的 index
   - 第 2 行: 令 h 為第 k 個 leaf node 的值 $H([f])$

2. 透過 for 迴圈，由 leaf node $h = H([f])$ 開始往上計算每一個 parent node 的值直到 root node ，每次計算時會依序用到 inclusion proof π' 裡的每一個 h_i。過程中 k' 表示目前節點 h 在該層的 index，n' 則表示該層最後一個節點的 index。

   - 第 4 $\sim$ 6 行: 如果這時 $n' = 0$，表示已計算到 root node 但 π' 中仍有剩餘的 h_i，則 $\pi' \neq \pi(k, f_{1:n})$ (即 π' 不合法)。此時回傳 `Invalid`。
   - 第 7 $\sim$ 8、12 $\sim$ 14 行: 由目前節點 (所存的值) h 與 sibling node 的值 h_i 來計算出 parent node 的值，並存回 h 。這裡根據目前節點 h 在該層的位置來決定 h 該如何與 h_i 串接：
     - Case 1: 若 $LSB(k') = 1$ 或 $k' = n'$，表示 h 為其 parent node 的 right child ，那麼 h_i 就是 h 左側的 sibling node。因此更新 h 的值為 $H(h_i \| h)$。
       此外，若 $LSB(k') = 0$，我們可以知道 $k' = n'$，這表示在更新 h 前，h 為該層最後一個節點，且 h 的 parent node 並不在上一層。因此，我們透過 right-shift k', n' (相當於除以 2 並捨棄餘數) 來往上一層移動，直到 $LSB(k') = 1$ ，這表示到達了更新前的 h 的 parent node 所在層的下面一層。
     - Case 2: 若 $LSB(k') = 0$ 且 $k' \neq n'$，表示 h 為其 parent node 的 left child ，那麼 h_i 就是 h 右側的 sibling node。因此更新 h 的值為 $H(h \| h_i)$。
   - 第 15 行: 將 h 更新為其 parent node 的值後，由於該 parent node 在目前所在層的上面一層，我們透過 right-shift k', n' (相當於除以 2 並捨棄餘數) 來往上一層移動。

3. 驗證 (第 17 $\sim$ 21 行)：
   當 inclusion proof 中的所有 h_i 都被處理完後，檢查以下兩者是否成立：

   - $n' = 0$: 由於 n' 表示節點 h 所在的層的最後一個節點的 index，這表示節點 h 為 root node ，即我們已經算到 Merkle tree 的最頂層
   - 算出來的 root hash h 與已知的 root hash h^* 相等

   如果兩者都滿足，則檔案 f 及其 inclusion proof π' 皆正確合法，因此回傳 `Valid`。反之，若有一條件不合法，則表示 $f \neq f_k$ 或 $\pi' \neq \pi(k, f_{1:n})$ ，即輸入的檔案或 inclusion proof 不合法，因此回傳 `Invalid`。

題目要求

本小題的目標是實作 `merkle-dir.sh verify-proof` 的功能，因此使用的指令皆為

```
./merkle-dir.sh verify-proof <path-to-leaf-file> --proof <proof-file> --
    ↪ root <root-hash>
```

令 f = `<path-to-leaf-file>`。此處 `<proof-file>` 的內容為 index k、檔案數量 n、與某個 inclusion proof π'，格式與上一小題的輸出格式相同，且 $1 \leq k \leq n$。另外，`<root-hash>` 為某 n 個檔案 $(f_1, f_2, \ldots, f_n)$ 的 Merkle tree 的 root hash h^*，以十六進位表示，其中字母為全大寫或全小寫。注意到 π' 可能不等於 $\pi(k, f_{1:n})$。

請利用 f, π', n, k, h^* 驗證 $f = f_k$。若驗證失敗，請輸出 `Verification Failed` 至 stdout，並以 1 作為 exit status。若驗證成功，請輸出 `OK` 至 stdout，並以 0 作為 exit status。

範例

指令

```
./merkle-dir.sh verify-proof dir1/hi.txt --proof hi.txt.proof --root
    ↪ 61972f8a5bf9925d70d34934dafced66756c5f9d7a80e3b265e61e5526155f83
```

會輸出 `OK` 至 stdout。

5 Report (10+3pts)

請繳交一份 `report.pdf`，寫下你做這次作業用到的 references，例如：

- 你所參考的網站的 URL
- 與你討論的同學 (學號或姓名皆可)
- 你與大語言模型或者生成式 AI（包含 ChatGPT 及其他類似工具）的完整對話截圖（如果有使用到）