

SPM HW3 Group Part

- I. 隨著軟體的開發以及更新，程式中隱含的錯誤會不斷累積進而造成後續維護成本增加，因此，及早發現程式中的錯誤將有效減少開發成本。Code Review 是軟體開發中減少程式錯誤的常見手法，可以有效協助增進軟體品質並且減少測試成本。

- A. 請以 reference 資料夾中的 Formal Technical Review Forms.docx 為範本，完成 Code Review 的 Review Report，Planning 部分需記錄小組的成員以及 review 的檔案名稱，並自行定義 defect 的 severity。Code Review 部分除了 Reviewer 的名字外，每一個 Defect 皆需記錄：defect location (line number)、severity、type (type 可參考附檔 Classification-of-Fault-Types.pdf) 以及 description of the errors you found，並附上每一次 Code Review 所耗費的時間。在最後 Integration Review 階段，請整合小組所有成員找到的 defect 並計算 Integration Review 所耗費的時間。

詳見附檔：

SPM_HW3_CodeReview_1.pdf 與 SPM_HW3_CodeReview_2.pdf

- B. 承上題，請依據 Chapter 07 講義 p.84~88 計算：estimated total defects、yield、defect density、inspection rate 以及 defect finding efficiency，並將詳細公式列出。(30 points) Hint：計算 metric 之過程，size 請以行碼數 (LOC) 來計算。estimated total defects 之方法可參考 reference 資料夾內之附檔 Software-Inspections.pdf。

CodeReview_1：bst.cpp

根據 Chapter 07 講義 p.84~88，並參考 Software-Inspections.pdf 裡的 Three-engineer Estimating Example 中的敘述。發現最多 defect 的工程師的 defect number 為 A，然後將其他兩位工程師所發現的 defect number (已減掉交集的數量) 為 B，而 A 和 B 共同發現的 defect number (A 和 B 交集) 為 C。因此，我們 BSTCodeReview 小組，根據 Table 1 以及參考 Software-Inspections.pdf 裡的 A Three-engineer Estimating Example 中的敘述，得出 $A = 24$ ， $B = 25$ ， $C = 22$

而 $Total Defects Found = A + B - C = 24 + 25 - 22 = 27$

Line	張子姍	趙承彥	ChatGPT		A	B	C
11	1	1			1	1	1
16	1	1	1		1	1	1
19	1				1		
33	1	1			1	1	1
37	1	1	1		1	1	1
39	1				1		
41	1	1	1		1	1	1
56、57	1	1	1		1	1	1
67	1	1			1	1	1
70		1				1	
83	1	1	1		1	1	1
106	1		1		1	1	1
109	1	1	1		1	1	1
119		1	1			1	
132	1	1	1		1	1	1
141		1				1	
153	1	1	1		1	1	1
165	1	1	1		1	1	1
173	1	1	1		1	1	1
184	1	1			1	1	1
197	1	1	1		1	1	1
206	1	1			1	1	1
213	1	1	1		1	1	1
221	1	1	1		1	1	1
234	1	1	1		1	1	1
264	1	1	1		1	1	1
283	1		1		1	1	1
Totals	24	23	18		24	25	22

Table 1. Three-Engineer Inspection Report(BSTCodeReview)

根據下圖的公式：

$$Estimated\ Total\ Defects = \frac{A \times B}{C} = \frac{24 \times 25}{22} = 27.27$$

$$Yield = \frac{Total\ Defects\ Found}{Estimated\ Total\ Defects} \times 100\%$$

所以

$$Yield = \frac{27}{27.7} \times 100\% = 99.01\%$$

Estimating the Remaining Defects

After you complete the product review, follow the procedure described in the INS form instructions for calculating the number of defects that are likely to remain in the product after the inspection. Also calculate the inspection yield. The easiest way to estimate the inspection yield is as follows:

Estimated Inspection Yield = $100 * (\text{Total defects found}) / (\text{Total estimated defects})$

$$= 100 * (A+B-C) / (A*B/C)$$

$$Defect\ Density = Total\ Defects\ Found / Size$$

而 Size = 290 line of code

所以 Defect Density = $27 / 290 = 0.0931$ defects/lines of code

$$Inspection\ rate = size / total\ inspection\ hours$$

Total inspection hours = 5.583 hours (335 minutes)

所以 Inspection rate = $290 / 5.583 = 51.943$ lines of code / hour

$$Defect\ finding\ efficiency$$

$$= Total\ defects\ found / total\ inspection\ hours$$

所以 Defect finding efficiency = $27 / 5.583 = 4.836$ defects/ hour

CodeReview_2 : rbtrees.java

Line	李銘峰	王彥翔	GPT		A(GPT)	B	C
13		1		王		1	
18 等	1	1				1	
18 等		1		王		1	
31 等	1			李		1	
40	1			李		1	
43	1	1	1		1	1	1
61	1	1	1		1	1	1
84	1	1	1		1	1	1
106	1	1				1	
119			1	G	1		
129		1		王		1	
140	1	1	1		1	1	1
157			1	G	1		
159	1	1				1	
182			1	G	1		
199	1			李		1	
222			1	G	1		
223,224		1		王		1	
257		1	1		1	1	1
269			1	G	1		
275	1	1	1		1	1	1
297		1		王		1	
304			1	G	1		
321	1	1	1		1	1	1
341,345			1	G	1		
總計	12	15	14		14	18	7

以下計算使用到的變數有 3 個

A = 找到最多unique defects的reviewer所找到的defects數量,

B = 其他reviewers找到的defects數量(已經去掉交集),

C = A 與 B 找到defects的交集數量

其中 $A=14$, $B=18$, $C=7$

$$\text{Estimated total defects} = A \times B / C = 14 \times 18 / 7 = 36$$

$$\text{Yield} = \text{Total defects found} / \text{Estimated total defects} \times 100\%$$

$$= (14 + 18 - 7) / 36 \times 100\% = 69.4\%$$

$$\begin{aligned} \text{Defect density} &= \text{Total defects found} / \text{Size} \\ &= 25 / 366 = 0.07(\text{defects} / \text{loc}) \end{aligned}$$

$$\begin{aligned} \text{Inspection rate} &= \text{Size} / \text{Total inspection hours} \\ &= 366 / (520 / 60) = 42.2(\text{locs} / \text{hour}) \end{aligned}$$

$$\begin{aligned} \text{Defect finding efficiency} &= \text{Total defects found} / \text{Total inspection hours} \\ &= 25 / (520 / 60) = 2.9(\text{defects} / \text{hour}) \end{aligned}$$

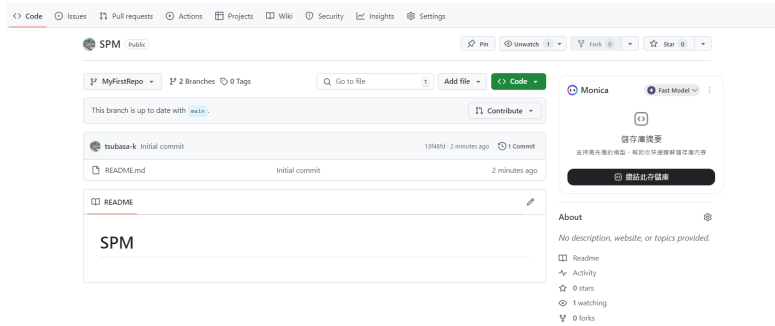
- C. 請使用一個開源工具來記錄/回報 Code Review 之過程，並以文字和實際操作的截圖來介紹該工具（組內小組之間可選擇不同的工具）。

CodeReview_1 : bst.cpp

我們 BSTCodeReview 小組決定用 GitHub 來記錄/回報 Code Review 過程。GitHub 除了可以用來管理程式碼版本以外，還可以用來記錄和回報 Code Review 的過程，並提升團隊的協作效率與程式碼品質。我們可以利用 GitHub flow 來進行 Code Review。

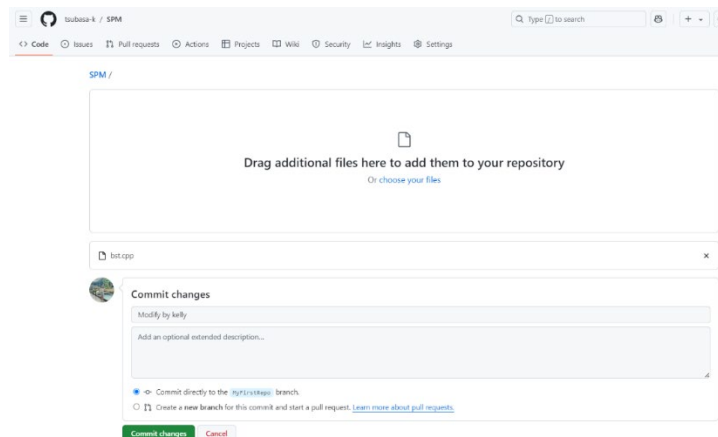
● Step 1. Create Branch

新增 MyFirstRepo 這個 Branch

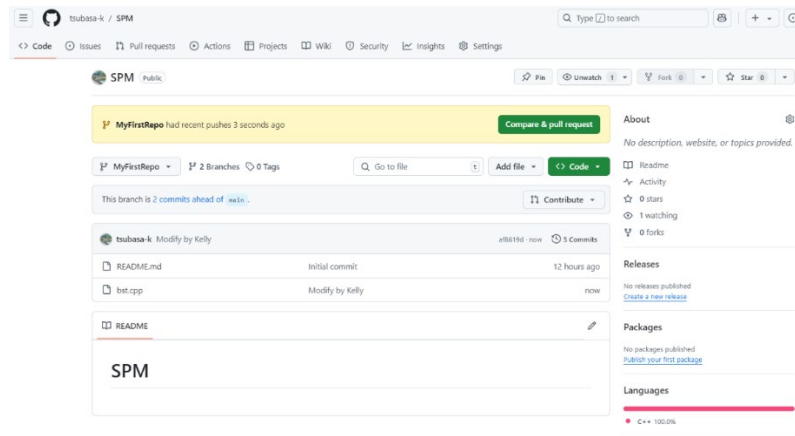


● Step 2. Add Commit

切換到這個 Branch，並上傳要 review 的程式檔案，並進行 Commit。

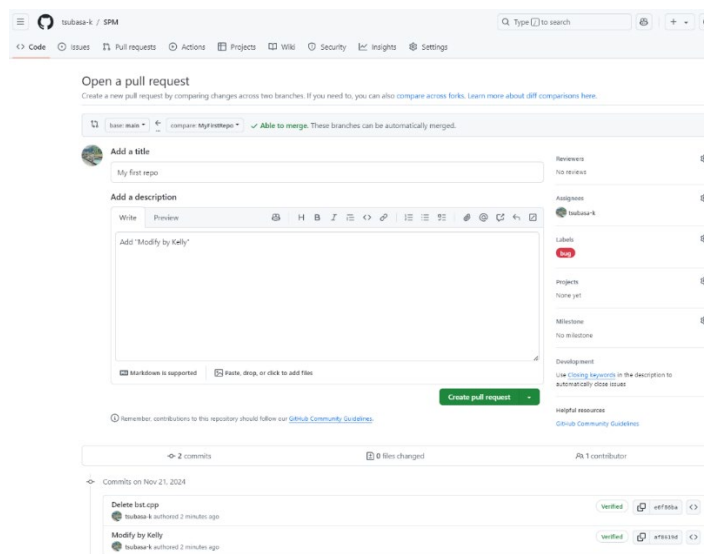


完成上傳程式碼後，回到 MyFirstRepo 主畫面，上方有提示通知某個 Branch 已經更新，如果需要比對內容和建立 Pull Request，可以點選 Compare & pull request



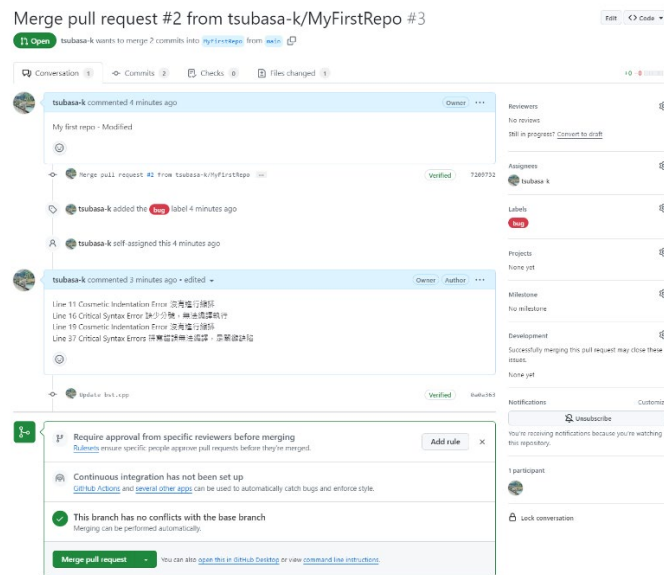
● Step 3. Pull Request

Open pull request，要先確認的是 Branch 合併來源與目的是否正確，並確認沒有衝突可以合併（下圖紅色框處），而在右邊選單可以指派 Reviewers 與 Assignees，設定 Labels、Project 與 Milestone，中間可以加入關於這個 pull request 的描述

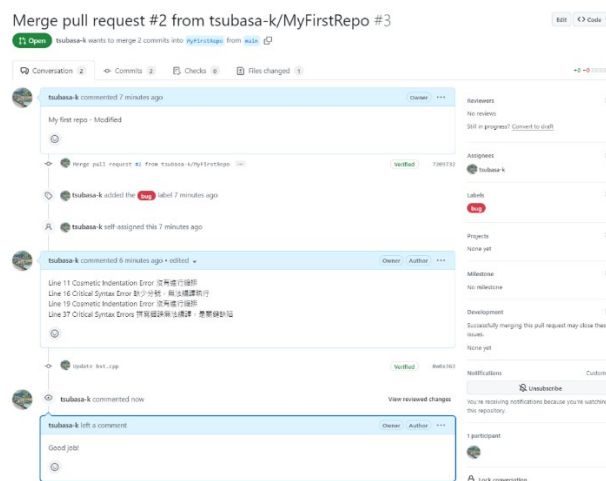
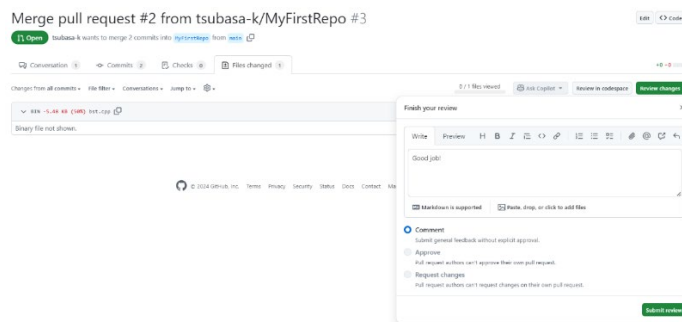


● Step 4. Code Review

可以看到所有的行為與討論都會在 Conversation 中被記錄，讓每個參與者可以清楚整個過程的來龍去脈



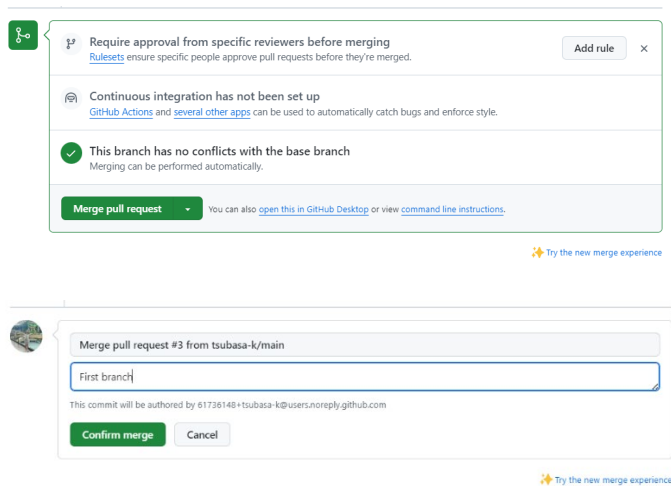
點選 Files Change 可以看見那些變更的內容，可以在程式碼上面輸入評論或 Review，讓其他人了解哪些內容需要變更才能符合需求



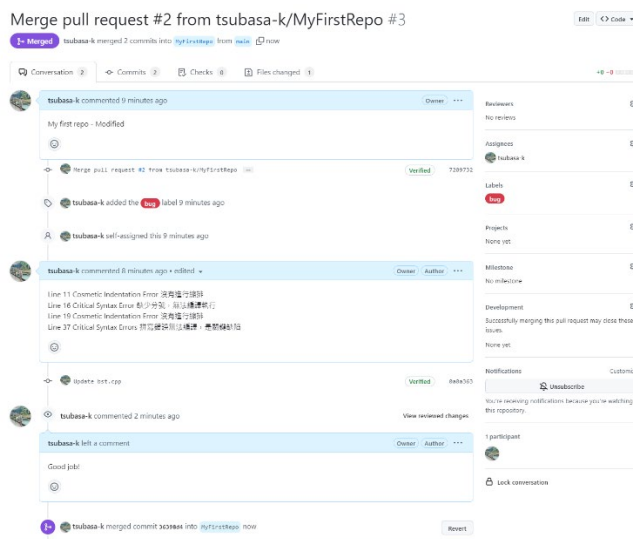
● Step 5. Merge pull request

修改許多內容之後，Author 確認沒問題，就可以在 Conversion 最下方點選 Merge pull request，輸入完相關內容後點選 confirm merge 進

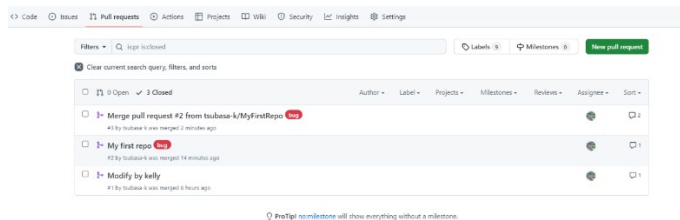
行合併



會顯示這個 Pull request 已經被 Merge，並且關閉這個 pull request



最後可以回到這個 MyFirstRepo 的 Pull Requests 畫面中，在 Close 的頁籤可以找到關閉的 pull request，也能查看主要分支(Main)是否已經完成修改



CodeReview_2：rbtrees.java

我們使用的工具是 GitHub，其基於 Git 提供強大的版本控制功能，能夠記錄每次程式變更，幫助開發者追蹤歷史記錄與回溯問題。

- 在使用之前要先做基本的帳號密碼申請，而在使用時必須先在電腦

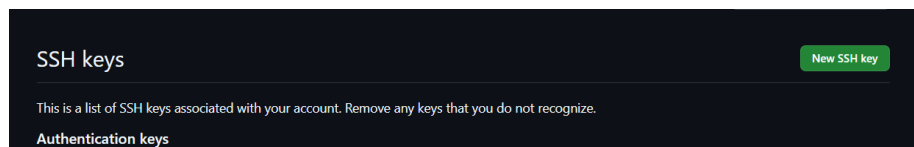
中安裝 ssh key 並且輸入到 Git 當中才能做後續操作。

```

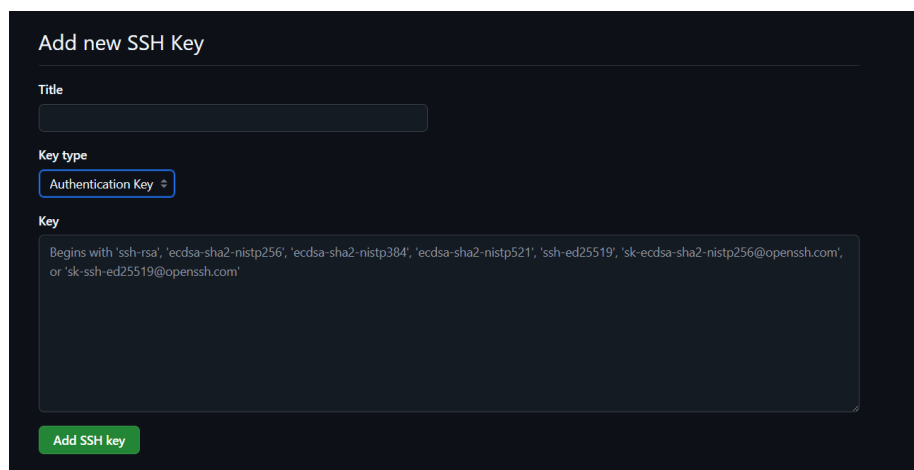
```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

```

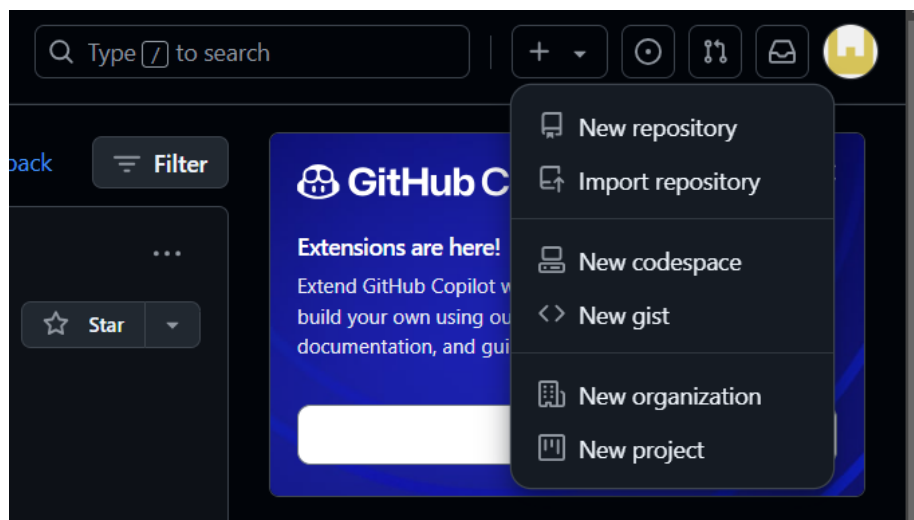
使用這個指令會在 ~/.ssh 這個資料夾底下產生 id_rsa 與 id_rsa.pub 兩個 key 作為後續驗證用的方法，我們需要把 id_rsa.pub 裡面中以 ssh- 為開頭自己的 email 為結尾的字串全部複製下來，接續點擊 github 右上角自己的頭像 > Setting > SSH and GPG keys，點擊畫面出現的 New SSH key。



Title 可以自行設定並把剛剛複製下來的字串完整貼到底下的 Key 並點擊 Add SSH key 即可建立完成。




接下來我們點擊畫面右上角的 "+" 並點擊 New repository。



輸入完基本資料即可點擊 Create repository。


Required fields are marked with an asterisk (*).


Owner * Repository name *

 RecorderZero /

Great repository names are short and memorable. Need inspiration? How about [ideal-spoon](#)?

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:


☐ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

 You are creating a public repository in your personal account.

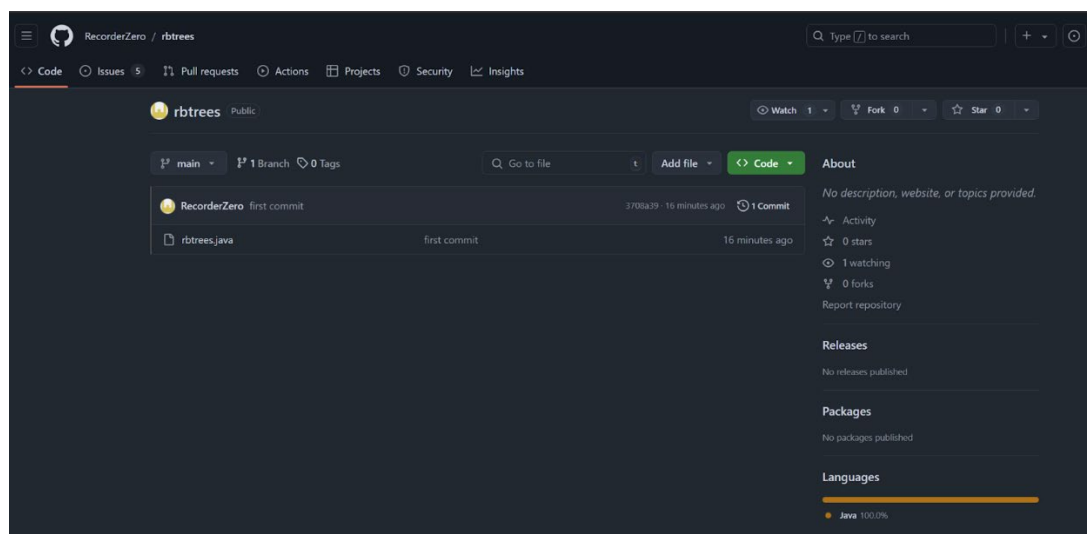
[Create repository](#)

使用 Git Bash 先移動到想要添加到 Git 上的資料夾，接續可以依照網頁所給的操作執行。

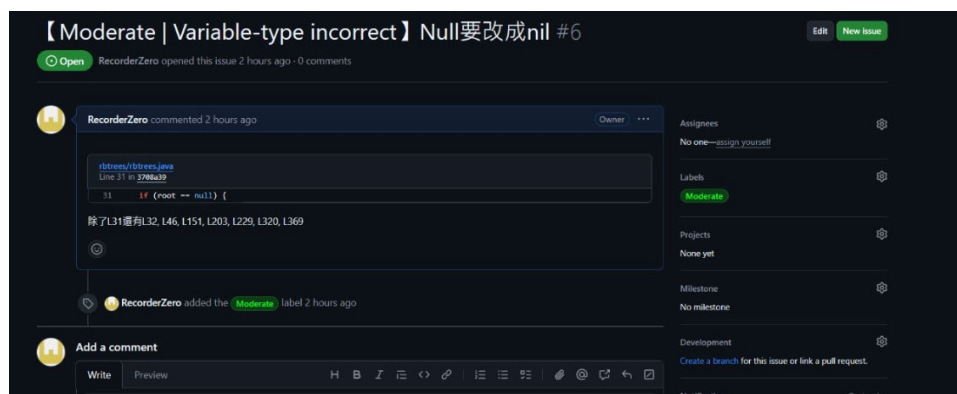
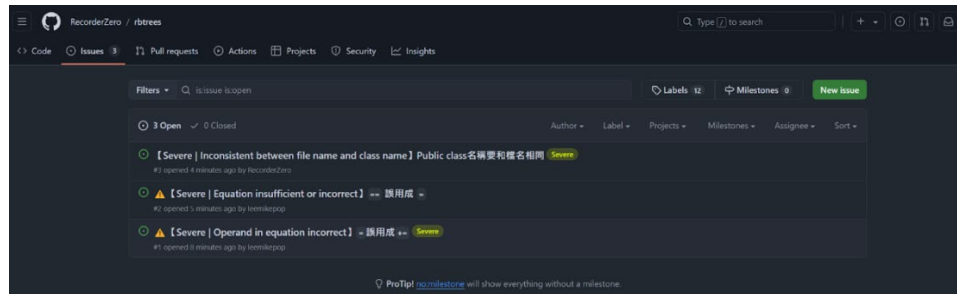
```
...or create a new repository on the command line

echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:RecorderZero/test.git
git push -u origin main
```

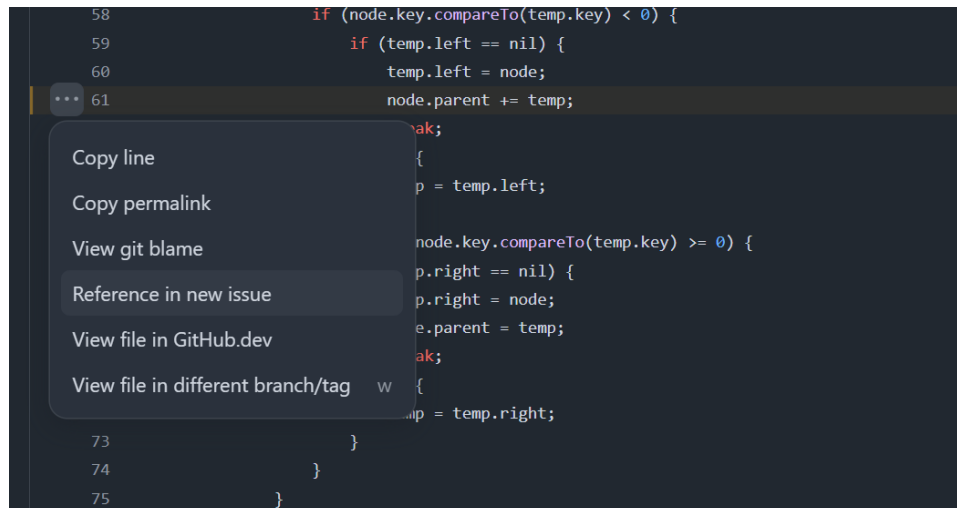
即可在 Git 上創建好自己的 repo。



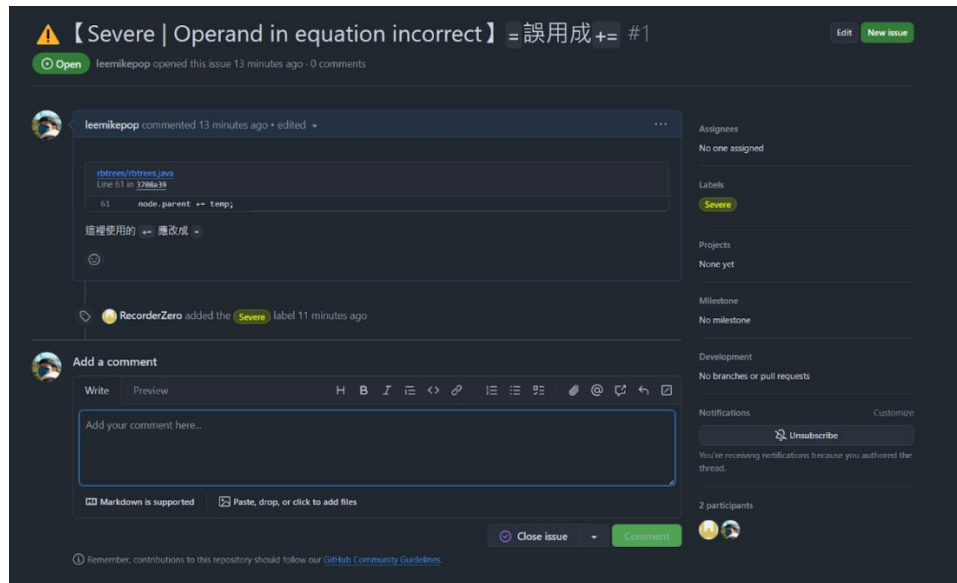
- Issue 是用來追蹤項目中的錯誤、功能需求或改進建議的工具。每個 Issue 都可以詳細描述問題、分配責任人、設置優先級、里程碑等標記，確保問題得到及時處理。



- Review 程式碼時若發現問題可以在程式碼左邊選擇”Reference in new issue”可以將問題加入到 issue 中。



- 參與者們以對話的方式討論並管理每一個 issue。



- 作者將程式修改好後可以透過 push 把原先開出來的 issue 給修正並關閉，具體流程如下：

```

...
git add .
git commit -m "some description fixes #1"
git push -u origin main
...

```

其中”fixes #1”是關鍵字，#後面的數字代表這次修改所關聯的 issue id，而 fixes 除了可以與 issue 關聯外還可以同時把 issue 關閉代表已經處理完成。

```

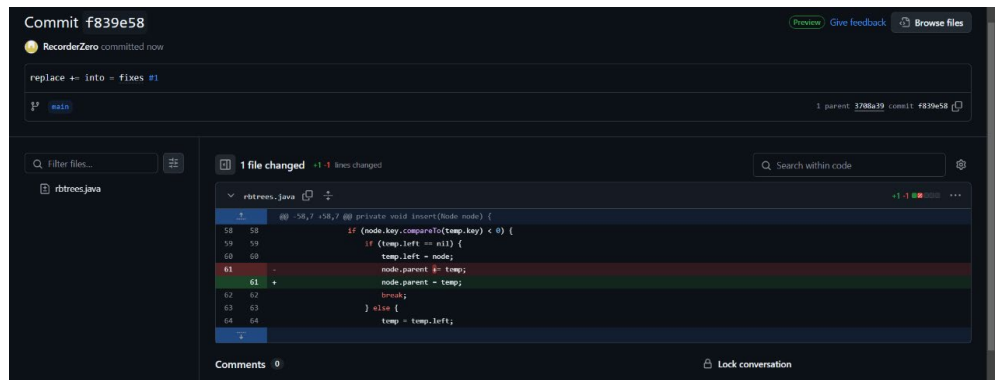
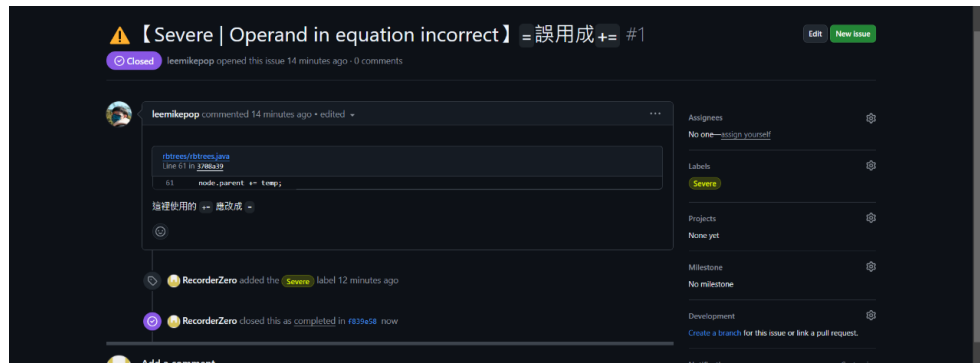
ACER@LAPTOP-IMRLMBOC MINGW64 /e/研究所/碩1上學期/軟體工程管理/2024_SPM_Hw3/Review/rbtree (main)
$ git add .

ACER@LAPTOP-IMRLMBOC MINGW64 /e/研究所/碩1上學期/軟體工程管理/2024_SPM_Hw3/Review/rbtree (main)
$ git commit -m "replace += into = fixes #1"
[main f839e58] replace += into = fixes #1
1 file changed, 1 insertion(+), 1 deletion(-)

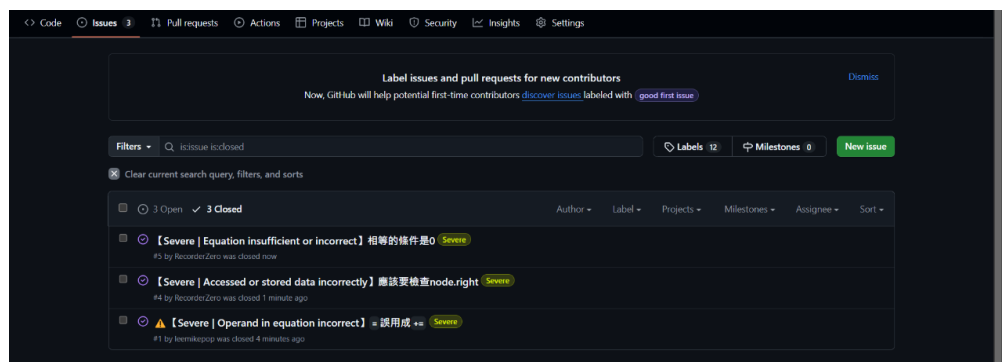
ACER@LAPTOP-IMRLMBOC MINGW64 /e/研究所/碩1上學期/軟體工程管理/2024_SPM_Hw3/Review/rbtree (main)
$ git push -u origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 20 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 300 bytes | 300.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:RecorderZero/rbtrees.git
  3708a39..f839e58  main -> main
branch 'main' set up to track 'origin/main'.

```

修改過後也會建立修改程式碼的連結，方便使用者點擊進入查看修改的內容。



最後所有正在處理以及處理完畢的 issue 均會詳列在 issue 這個選單當中。



- D. 請同學嘗試以任意形式，使用 ChatGPT 來協助理程式碼審查(同學們可參考網址：<https://github.com/anc95/ChatGPT-CodeReview>、<https://github.com/anc95/ChatGPT-CodeReview/blob/main/README.zh-TW.md> 或者 <https://studyhost.blogspot.com/2023/04/aigptprcode-reivew.html>，亦或直接使用 ChatGPT 聊天室來進程式碼審查)，並將其結果與手動程式碼審查(manual code review)的結果相互驗證及進行分析與討論。

CodeReview_1：bst.cpp

此部份我們直接使用 ChatGPT 聊天室來進程式碼審查

● Step 1. 首先將欲使用來 code review 的 Classification-of-Fault-Types 告訴 ChatGPT

你好~可以協助我進行code review嗎

當然可以！請告訴我你的程式碼內容或是提供程式檔案，我可以協助你進行以下幾方面的 code review：

1. **程式結構**：檢查程式碼的架構是否清晰，是否符合良好的設計模式。
2. **可讀性**：確保變數命名、註解和程式碼格式易於理解。
3. **效能**：找出可能影響效能的問題或潛在的優化空間。
4. **錯誤處理**：檢查是否妥善處理例外和錯誤情況。
5. **安全性**：檢查是否存在潛在的安全漏洞。
6. **符合性**：確認是否遵守既定的程式碼風格或標準。

請分享你需要檢查的程式碼，我會針對問題提出具體建議！

🔊 📄 🍌 🔄 📌



Classification of Fault Types

Logic problem
Forgotten cases or steps
Duplicate logic
Extreme conditions neglected
Unnecessary function
Misinterpretation
Missing condition test
Checking wrong variable
Iterating loop incorrectly
Computational problem
Equation insufficient or incorrect
Missing computation
Operand in equation incorrect
Operator in equation incorrect
Parentheses used incorrectly



這份「Fault Types Classification」清單列出了許多常見的錯誤類型，可以作為 code review 時評估和檢查的依據。當進行 code review 時，我會根據這些分類來檢查程式碼的問題，並對應每個類別進行詳細分析。以下是一些範例說明，如何對應這些錯誤類型：

1. 邏輯問題 (Logic Problem)

- **忘記的情況或步驟 (Forgotten cases or steps)**：檢查是否有遺漏的邏輯情況，特別是在條件判斷或分支結構中，可能會錯過處理某些特定情況。
- **重複邏輯 (Duplicate logic)**：查找是否有多餘的程式碼或功能，可能導致代碼重複或冗餘。
- **極端條件忽略 (Extreme conditions neglected)**：檢查代碼是否處理了所有的邊界情況或極端值，防止程式出現不可預期的錯誤。

2. 計算問題 (Computational Problem)

- **方程式不足或不正確 (Equation insufficient or incorrect)**：檢查公式是否正確，特別是數學計算中是否漏掉了某些必要的計算或邏輯。
- **操作數錯誤 (Operand in equation incorrect)**：檢查運算式中的變數是否正確，避免使用錯誤的資料。
- **四捨五入或截斷錯誤 (Rounding or truncation fault)**：確認數據處理中是否有精度損失，特別是浮點數運算中。

● Step 2. 將 Formal Technical Review Forms 格式告訴 GPT



待會按照以下格式
1. 發現錯誤項目順序
2. 所在行數
3. 嚴重程度分為critical、severe、moderate、minor
4. Chk/ref 從剛剛給的Classification of Fault Types挑選符合的
5. 錯誤內容描述
Num | Location | Severity | Chk/Ref | Description



了解！待會我會按照你提供的格式進行 code review。這是格式範例，當你提供程式碼後，我將根據錯誤項目進行檢查並填寫相關資訊：

Num	Location	Severity	Chk/Ref	Description
1	Line 12	Critical	Logic Problem - Forgotten cases or steps	程式邏輯遺漏了處理特定情況，可能導致錯誤結果。
2	Line 45	Severe	Data-handling Problem - Initialized data incorrectly	變數未正確初始化，導致後續運算錯誤。
3	Line 63	Moderate	Computational Problem - Operand in equation incorrect	計算式中的操作數錯誤，可能導致不正確的結果。
4	Line 88	Minor	Documentation Problem - Incomplete Item	函數註解不完整，未說明參數的邊界情況。

如果你準備好了程式碼，請提供，我會根據上述格式協助檢查並填寫。

● Step 3. 在開始進行前先讓 ChatGPT 替行數編號以利後續格式撰寫



以下是根據你提供的程式碼，已編號並準備進行 code review 的行數標註：

```
cpp 複製程式碼

1 //Binary Search Tree Program
2 //Author: Paul

3 //include "stdafx.h"
4 #include <iostream>
5 #include <cstdlib>
6 using namespace std;

7 class BinarySearchTree
8 {
9 private:
10     struct tree_node
11     {
12         tree_node* left;
13         tree_node* right;
14         int data;
15     };
16     tree_node* root;
17 public:
18     BinarySearchTree()
19     {

249         case 3: cout << endl;
250         cout << " Pre-Order Traversal " << endl;
251         cout << " -----" << endl;
252         b.print_preorder();
253         break;
254         case 4: cout << endl;
255         cout << " Post-Order Traversal " << endl;
256         cout << " -----" << endl;
257         b.print_postorder();
258         break;
259         case 5: cout << " Enter data to be deleted : ";
260         cin >> tmp1;
261         b.remove(tmp1);
262         break;
263         case 6: system("pause");
264         return 0;
265         break;
266     }
267 }
268 }
```

接下來，請告訴我是否可以開始進行錯誤檢查並根據格式填寫。

● Step 4. 嘗試進行 code review



可以
依照現在有錯誤的行數標記進行code review
不需要先更新成正確的程式碼



以下是根據您提供的格式進行的錯誤報告：

Num	Location	Severity	Chk/Ref	Description
1	Line 6	Severe	Syntax Error	缺少分號 (;) 結尾，應該是 <code>int data;</code>
2	Line 23	Moderate	Syntax Error	函式原型拼寫錯誤， <code>vioid</code> 應為 <code>void</code>
3	Line 24	Moderate	Memory Allocation	新建 <code>tree_node</code> 時，應檢查記憶體分配是否成功
4	Line 29	Moderate	Logic Error	邏輯錯誤，應將 <code>curr = curr->left</code> ，而不是 <code>curr = curr->right</code>
5	Line 35	Severe	Syntax Error	函式定義拼寫錯誤， <code>BinarySearchTree::remove(int d)</code> 應為 <code>BinarySearchTree::remove(int d)</code>
6	Line 38	Moderate	Type Error	<code>found</code> 應該是 <code>bool</code> 類型，而不是 <code>int</code>
7	Line 45	Moderate	Logic Error	訊息顯示錯誤，應該是 <code>Data not found!</code>
8	Line 52-53	Severe	Logic Error	邏輯錯誤， <code>&</code> 應該改為 <code>&&</code> 來進行邏輯比較
9	Line 58	Severe	Assignment Error	使用了賦值運算符 (<code>=</code>)，應該改為比較運算符 (<code>==</code>)
10	Line 61	Severe	Syntax Error	<code>delete cur;</code> 應該為 <code>delete curr;</code>
11	Line 65	Severe	Syntax Error	缺少分號 (;)，應加上 <code>delete curr;</code>
12	Line 81	Moderate	Logic Error	條件判斷括號錯誤，應該將 <code>)</code> 移到 <code>curr->right</code> 後面
13	Line 83	Minor	Syntax Error	需要注意縮排， <code>lcurrp</code> 變數名應該確認正確
14	Line 85	Moderate	Null Pointer Check	應檢查 <code>lcurr</code> 是否為 <code>NULL</code> ，避免空指針錯誤

● Step 5. 針對結果錯誤部分進行溝通討論



Location 似乎是錯誤的第一項應在第 16 行



感謝您的指正！根據您的要求，已經更新了錯誤所在行數，以下是修正後的錯誤報告：

Num	Location	Severity	Chk/Ref	Description
1	Line 16	Severe	Syntax Error	缺少分號 (;) 結尾，應該是 <code>int data;</code>
2	Line 23	Moderate	Syntax Error	函式原型拼寫錯誤， <code>vioid</code> 應為 <code>void</code>
3	Line 24	Moderate	Memory Allocation	新建 <code>tree_node</code> 時，應檢查記憶體分配是否成功
4	Line 29	Moderate	Logic Error	邏輯錯誤，應將 <code>curr = curr->left</code> ，而不是 <code>curr = curr->right</code>
5	Line 35	Severe	Syntax Error	函式定義拼寫錯誤， <code>BinarySearchTree::remove(int d)</code> 應為 <code>BinarySearchTree::remove(int d)</code>
6	Line 38	Moderate	Type Error	<code>found</code> 應該是 <code>bool</code> 類型，而不

● Step 6. 由於 ChatGPT 並無舉一反三修正錯誤持續進行討論並提出要求

其他項也是錯的

感謝您的耐心，這裡根據您提供的程式碼，重新檢查錯誤，並按照要求整理了正確的錯誤報告。這是修正後的版本：

Num	Location	Severity	Chk/Ref	Description
1	Line 16	Severe	Syntax Error	<code>int data</code> 應該以 <code>;</code> 結尾，應改為 <code>int data;</code>
2	Line 23	Moderate	Syntax Error	拼寫錯誤， <code>void</code> 應為 <code>void</code>
3	Line 24	Moderate	Memory Allocation	在創建 <code>tree_node</code> 時，沒有檢查 <code>new</code> 是否成功分配記憶體
4	Line 29	Severe	Logic Error	<code>if (t->data < curr->data)</code> 進行錯誤比較，應該改為 <code>if (t->data < curr->data)</code>
5	Line 35	Severe	Syntax Error	函式定義拼寫錯誤，應為 <code>BinarySearchTree::remove(int d)</code>
6	Line 38	Moderate	Logic Error	<code>found</code> 應該是 <code>bool</code> 類型，而不是 <code>int</code>
7	Line 45	Severe	Logic Error	在此條件中 <code>if (!found)</code> 顯示錯誤的訊息，應為 <code>Data not found!</code>
8	Line 52-53	Severe	Syntax Error	這裡有錯誤的邏輯，應使用 <code>&&</code> 而不是 <code>&</code>
9	Line 58	Severe	Assignment	使用了賦值運算符 <code>(=)</code> ，應該改

且chk/ref沒有照我給的type list

< 2/2 >

抱歉，之前的 Chk/Ref 欄位沒有依照您提供的 Classification of Fault Types 進行對應。這裡是基於您的要求，使用正確的 Chk/Ref 列表進行修正：

Num	Location	Severity	Chk/Ref	Description
1	Line 16	Severe	Syntax Error	<code>int data</code> 應該以 <code>;</code> 結尾，應改為 <code>int data;</code>
2	Line 23	Moderate	Syntax Error	拼寫錯誤， <code>void</code> 應為 <code>void</code>
3	Line 24	Moderate	Memory Management	在創建 <code>tree_node</code> 時，沒有檢查 <code>new</code> 是否成功分配記憶體
4	Line 29	Severe	Logic Error	<code>if (t->data < curr->data)</code> 進行錯誤比較，應該改為 <code>if (t->data < curr->data)</code>
5	Line 35	Severe	Syntax Error	函式定義拼寫錯誤，應為 <code>BinarySearchTree::remove(int d)</code>
6	Line 38	Moderate	Logic Error	<code>found</code> 應該是 <code>bool</code> 類型，而不是 <code>int</code>
7	Line 45	Severe	Syntax Error	在此條件中 <code>if (!found)</code> 顯示錯誤的訊息，應為 <code>Data not found!</code>
8	Line 52-53	Severe	Logic Error	這裡有錯誤的邏輯，應使用 <code>&&</code> 而不是 <code>&</code>

			Check	免空指針錯誤
15	Line 92	Moderate	Memory Management	應該檢查 <code>tmp</code> 是否為 <code>NULL</code> ，避免錯誤
16	Line 97	Minor	Function Call	<code>inorder</code> 函數呼叫時不一致，應該檢查使用的參數
17	Line 103	Minor	Syntax Error	拼寫錯誤， <code>root</code> 應該為 <code>root</code>
18	Line 106	Minor	Logic Error	應該檢查迴邏輯是否正確
19	Line 119	Severe	Assignment Error	使用賦值運算符 <code>(=)</code> 代替了比較運算符 <code>(==)</code>
20	Line 132	Moderate	Typo	<code>inssert</code> 應該為 <code>insert</code>
21	Line 139	Severe	Logic Error	<code>remove(ch)</code> 應該改為 <code>remove(tmp1)</code>

根據您的要求，我已經將 Chk/Ref 欄位修正為對應的錯誤類型，希望這次更符合您的期望！如果有進一步的修改需求，請隨時告訴我。

- Step 7. 驗證結果與討論

Location :

針對最後結果發現，ChatGPT 在錯誤發生位置給的行數一直是錯誤的，即便是先編輯編號給它，仍無法正確判定。

Severity :

在其他欄位排除錯誤時，Severity 會與前一次產出結果不一致。

Chk/Ref :

沒有與給定 list 的 type 名稱一致，會自行命名，且在繼續進行對話後結果也會變動。

Description :

此項目相對準確，能確實指出錯誤原因及如何修正，雖然每次也會有幾項不見或新增，但大致數量都相近，相比其他欄位，在 code review 中發揮最好也最有幫助。

結論：

最後在與其他兩位組員比對結果並修正些許錯誤內容後，也能產出完整 report，由此可見 ChatGPT 確實能在 code review 中有所幫助，但不能 100%採納其結果，還是需要後續人工檢查，但能提升效率及縮短程式碼審查的時間，除了格式上問題及主觀分類，程式碼錯誤定義和 debug 準確度也能完成 80~90%。

CodeReview_2 : rbtrees.java

ChatGPT 非常善於找到一些很細節的錯誤，例如：我們透過人工的方式已經找到了一些 operator 的錯誤，像是「==寫成=」或者「&&寫成&」但透過 GPT 重新分析後仍有一些被我們所遺漏的地方，如下圖所示。



GPT 也很善於講解一些程式的概念，如同我們這次挑選的 rbtree 程式

除了其本身本來就有許多操作方式，而且各個操作方式又有多個情境需要考慮，是個有點麻煩的程式，再加上這些操作方式又不只一種實現方法，有些人會從上到下加入時就開始檢查是否有不符合的狀況發生，有些人是先加入節點後再檢查，只要實作方法與我們認知的不一樣，往往就需要卡很久都還不見得可以理解其實作方法，像我在學習 `rbtree` 針對插入這個操作是在向下搜尋插入點時就一步步調整樹的模樣，和程式寫的先插入再慢慢向上調整的邏輯完全相反，因此主程式的部分總無法完全理解其實作的精妙處，但把這些程式一步步丟給 GPT 他就會很詳盡地說明每一步的操作邏輯，再搭配其他網站的說明就能夠理解作者實作的方式。當然 GPT 的說明不總是正確，如同現實中我們與其他人討論時也不一定每句話都是正確的，但在過程中我們至少可以慢慢去理解對方的想法一樣，下圖即是我們與 GPT 的討論過程。



這段程式碼實現了紅黑樹中節點的替換操作，稱為 `transplant`，其目的是將節點 `target` 用節點 `with` 替換，並更新它們的父子關係。然而，這段程式碼存在明顯的邏輯錯誤，導致樹的結構可能無法正確更新。以下是詳細解析、問題分析和修正版建議。

功能與目的

- `transplant` 是一個輔助函數，用於替換節點：
 - 如果 `target` 是根節點，直接將 `root` 指向 `with`。
 - 如果 `target` 是父節點的左子節點，則將父節點的左子節點設為 `with`。
 - 如果 `target` 是父節點的右子節點，則將父節點的右子節點設為 `with`。
- 在完成這些替換後，還需將 `with` 的 `parent` 更新為 `target` 的父節點。

問題分析

1. 邏輯錯誤：父節點的更新

```
java 複製程式碼  
  
} else if (target == target.parent.left) {  
    target.parent = with;  
}  

```

- 錯誤的行為：
 - 這段程式碼試圖更新父節點的左子節點，但錯誤地將 `target.parent` 直接設為 `with`，實際上應該是將 `target.parent.left` 設為 `with`。
- 修正：

```
java 複製程式碼  
  
} else if (target == target.parent.left) {  
    target.parent.left = with;  
}  

```

2. `with` 為 `nil` 時的父節點更新

```
java 複製程式碼  
  
with.parent = target.parent;  

```

- 問題：當 `with` 是 `nil` 時，這行代碼試圖訪問 `with.parent`，可能導致空指標異常。
- 修正：
 - 在執行 `with.parent = target.parent;` 之前，應確認 `with` 是否為 `nil`：

```
java 複製程式碼  
  
if (with != nil) {  
    with.parent = target.parent;  
}  

```

3. 縮排與代碼風格

- `else` 缺少花括號 `{}`，可能導致閱讀困難或邏輯錯誤。
- 修正：

```
java 複製程式碼  
  
} else {  
    target.parent.right = with;  
}  

```

邏輯流程

1. 如果 `target` 是根節點：
 - 將 `root` 替換為 `with`。
2. 如果 `target` 是父節點的左子節點：
 - 將父節點的左子節點設為 `with`。
3. 如果 `target` 是父節點的右子節點：
 - 將父節點的右子節點設為 `with`。
4. 如果 `with` 非空：
 - 更新 `with` 的父節點為 `target` 的父節點。

以下附上 GPT 額外找到我們沒找到的錯誤列表：

1	119	Severe	Operand in equation incorrect	node.parent.col or 應該改成 node.parent.parent.color
2	157	Moderate	Forgotten cases or steps	rotateLeft 應該 增加 node 為 nil 的判斷
3	182	Moderate	Forgotten cases or steps	rotateRight 應該 增加 node 為 nil 的判斷
4	222	Severe	Operand in equation incorrect	target.parent 應該改成 target.parent.left
5	269	Critical	Inconsistent subroutine arguments	x 應該改成 x.parent
6	304	Severe	Wrong subroutine called	rotateRight 應該改成 rotateLeft
7	341,345	Moderate	Missing condition test	未處理 nextLine 為空

或者用戶直接
按”Enter”，可
能導致系統崩
潰

可以發現 GPT 找到了許多主要程式碼上邏輯的錯誤，以及部分空指標未檢驗的錯誤，透過這次的作業我們發現雖然 GPT 偶有錯誤，例如 defect 的 type 可能會分類錯誤，大體上能夠大大地減輕 code review 的繁雜，可以先告訴我們哪幾段有潛在的問題，我們這時再專注於該處就能找到大部分的錯誤，只是給出程式碼時可能要一段一段一個 function 一個 function，因為上面有些錯誤是在一開始我們一次給出全部程式碼時 GPT 沒有提出的錯誤，而是到我們一個 function 慢慢深入追蹤時才提出的問題。實務上如果要透過 GPT 進行 code review 我們可能也需給 GPT 一些 function 的敘述文件，因此次作業是有名的 rbtree 使得 GPT 能夠很快進入狀況，但在自己撰寫的程式碼中我們可能要先將 high level 的程式邏輯說給 GPT 聽，才能更有效地找到問題所在。

- II. Time Recording 是軟體專案管理中，用來衡量 team member effort 以及專案品質的方法之一，請參考 Chapter 06 講義 p.96~p.99 關於 PSP time recording log 的說明，尋找一種開源 或閉源的 time log 工具，並詳細說明其功能、使用方法以及企業如何運用其增進專案效率。

我們討論後決定使用的開源工具是 **Clockify**，是類似 Toggl 的一款免費的時間追蹤工具，可以應用於軟體專案管理中，來衡量團隊成員的投入與專案品質。

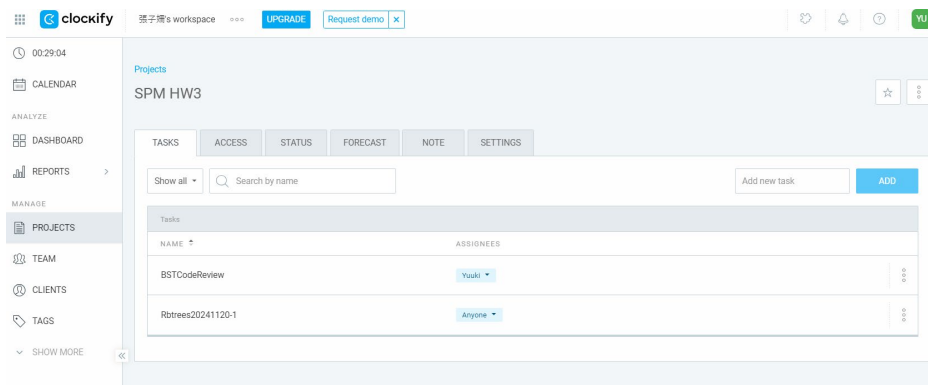
功能介紹：

1. Time Tracker:

Clockify 提供直觀的計時器功能，讓使用者能輕鬆開始和停止計時並且迅速記錄每項工作的耗時，無論是進行日常任務還是特定項目。

2. Projects:

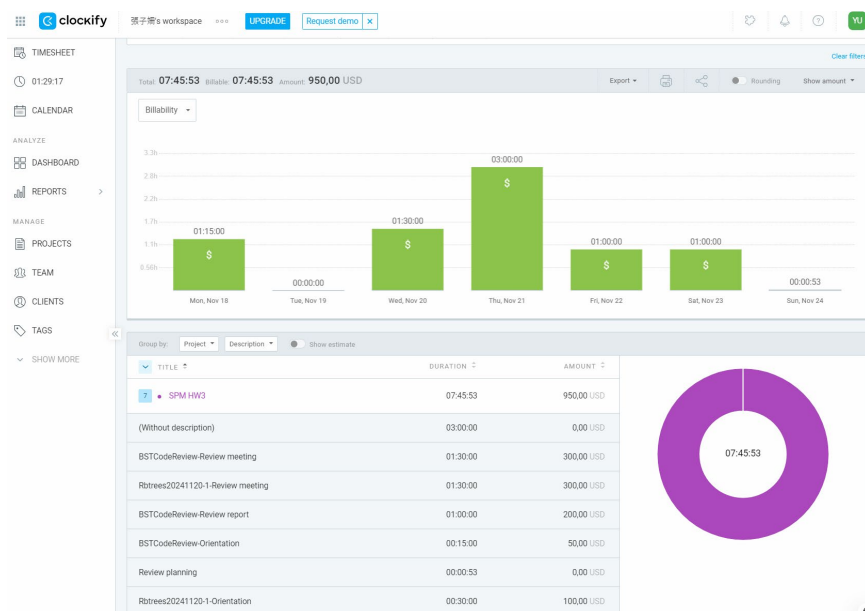
使用者可以建立專案，並在專案下細分任務，可以新增參與的人員，並方便追蹤每個任務的進度與時間分配。Clockify 的免費版也有協同合作功能，可以邀請最多 5 位團隊成員加入專案、分配工作任務、設定工時時間...等等。



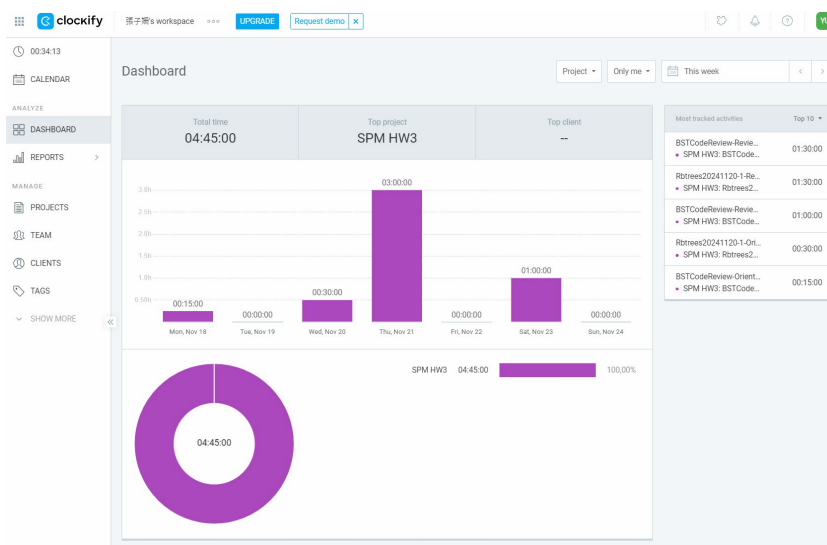
3. Reports & Dashboard:

Clockify 提供詳細的時間報告，展示每個專案和任務的時間分配，並提供可視化圖表，幫助使用者快速判斷工作中的瓶頸和改進空間

Reports:

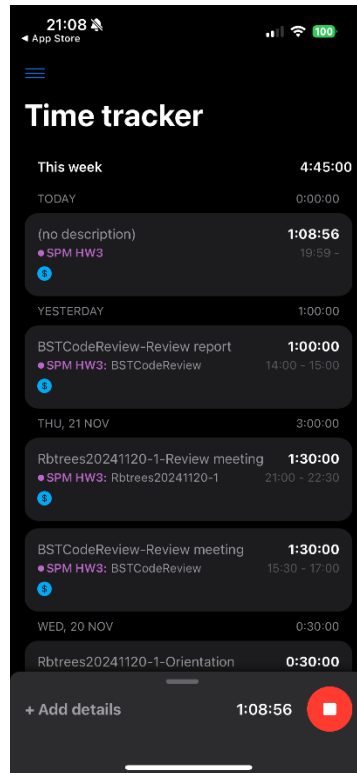


Dashboard:



4. 多平台支援：

Clockify 支援網頁、Windows、Mac、iOS、Android 等多種平台，讓使用者隨時隨地都能追蹤管理時間。但是 APP 能負責計時，無法查看統計結果，要看完整的結果還是必須透過網頁版的 Clockify 才行



使用方法：

(1) 註冊與登入：

前往 Clockify 官方網站，點擊右上方的「Sign up free」進行註冊，可使用自訂 email/password 或直接使用 Google 帳號進行註冊。

(2) 建立專案與任務：

登入後，進入「Projects」頁面，利用右上方的「新增專案」按鈕，新增專案。在專案的「Tasks」分頁，點擊「Add」按鈕，新增任務。

(3) 時間計時：

在「Time Tracker」頁面，輸入任務名稱，選擇對應的專案與任務，點擊「Start」開始計時。完成後，點擊「Stop」停止計時。

(4) 查看報告：

在「Reports」頁面，可設定不同條件來匯出工作時長報表，分析時間分配情況等。

企業如何運用其增進專案效率：

- **精確評估工時：**

透過 Clockify，企業可以精確記錄每個任務的實際耗時，為未來專案提供更準確的工時預估，避免低估或高估工時，提升專案規劃的準確性。

- **提升工作效率：**

可視化的時間流逝感會影響心理層面，刺激員工專心在眼前正在做的任務上，減少干擾，進而提升工作效率。

- **改善資源分配：**

透過分析團隊成員在不同任務上的時間投入，企業可以更有效地分配資源，確保關鍵任務獲得足夠的支持，避免資源浪費。

- **促進團隊協作：**

Clockify 的團隊功能允許成員之間共享時間追蹤資訊，增進透明度，促進協作，並有助於去了解及辨識需要支援的領域。

- **提升專案品質：**

透過詳細的時間報告，企業可以判斷在專案執行中遇到的瓶頸，針對需要的地方改進，提升專案品質。

III. 請同學用設計好的範本記錄本次會議，須至少包含會議日期、時間、地點、參與成員、討論事項、作業分工列表、組員討論合照。會議紀錄最多 4 頁。

詳見附檔 **SPM_HW3_MeetingMinutes.pdf**