

JooHyun – Lee (comkiwer)

# TS개발협업도구

Hancom Education Co. Ltd.

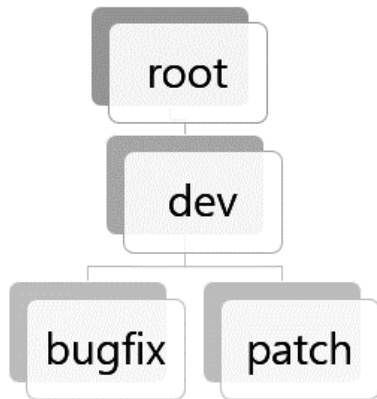
# Problem

프로젝트 형상 관리를 위해 개발 협업 도구를 사용한다.

개발 협업 도구의 프로젝트 관리 단위는 가지인데, 가지들은 Tree 형태로 분기된다.

각각의 가지는 개발 협업 도구 상에서, 파일들이 저장되는 공간이다.

가지 내 모든 파일들은 이름, 내용, 생성 시각, 최근 수정 시각 정보를 갖는다.



branch	name	content	created	recently edited
root	main	hi	1	6
	user	world	2	2
	module	mod	4	4
dev	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	paronly	par	13	13
bugfix	main	hello	1	1
	user	sword	2	17
	module	mod	5	5
	sononly	son	12	12
patch	main	hello	1	1
	user	lord	2	16
	module	mod	5	5
	paronly	par	13	13

[Fig. 1]

사용자는 개발 협업 도구 내 특정 가지에서 파일을 생성 또는 수정 하거나 해당 가지에서 자식 가지로 분기 또는 부모 가지로 병합하는 동작을 수행 할 수 있다. 개발 협업 도구의 초기 상태는, "root" 가지만 있고, 이 "root" 가지는 파일들이 없는 상태로 주어진다.

초기 상태에서, 시간 순서대로 아래 4 가지 동작을 수행하는 프로그램을 작성하라.

1. 새로운 파일 생성
2. 기존 파일 수정
3. 가지 분기
4. 부모 가지로 병합

각 동작에 대한 설명은 아래와 같다.

## 1. 새로운 파일 생성

가지 내 새로운 파일을 생성한다.

각각의 가지는 최대 50 개의 파일만 저장할 수 있다.

만약, 50 개의 파일이 저장된 가지에 새로운 파일을 생성해야 할 경우,  
가장 생성 시각이 오래된 파일을 하나 삭제한 후, 새로운 파일을 생성한다.

## 2. 기존 파일 수정

가지 내 특정 파일을 수정한다.

이때 수정된 파일의 최근 수정 시각이 `update` 된다.

## 3. 가지 분기

특정 가지에 자식 가지 생성 후, 자식 가지로 모든 파일을 복사한다.

복사된 파일의 이름, 내용, 생성 시각, 최근 수정 시각은 원본 파일과 같다.

새로운 가지는 분기를 통해서만 만들 수 있다.

따라서 "root" 가지를 제외한 모든 가지는 단 하나의 부모 가지를 갖지만,  
자식 가지는 여럿일 수 있다.

그리고 분기된 각 가지에 있는 파일 수정 사항은,

다른 가지에 있는 파일에 영향을 끼치지 않아야 함에 유의하라.

## 4. 부모 가지로 병합

"root" 가지가 아닌 모든 가지는, 부모 가지와 병합이 가능하다. 부모 가지로 병합 후, 해당 가지는 사라진다.

만약, 병합 하고자 하는 가지를 "x" 라 했을 때, "x" 의 자식 가지가 없을 경우, 가지에 있는 파일 병합 동작은 아래 규칙을 따른다.

- 1) "x" 에 있는 파일이 "x" 의 부모 가지에 없을 경우, 해당 파일은 부모 가지로 복사된다.
- 2) "x" 에 있는 파일이 "x" 의 부모 가지에 있을 경우,
  - 2-1) 부모 가지의 파일과 생성 시각이 다른 경우, "x" 의 파일은 무시된다.
  - 2-2) 부모 가지의 파일과 생성 시각이 같고, 최근 수정 시각이 "x" 의 파일이 더 최신인 경우, 부모 가지의 파일이 "x" 의 파일로 대체된다. (반대로 부모 가지에 속한 파일이 더 최근에 수정된 경우, "x" 의 파일은 무시된다.)
  - 2-3) 생성 시각, 수정 시각이 모두 같은 경우, "x" 의 파일은 무시된다.

## 4. 부모 가지로 병합 (계속)

위 과정을 마치고 부모 가지에 남은 파일의 개수가 50 개를 초과하면, 생성 시각이 오래 된 파일들을 삭제하여 50 개를 맞춘 후 "x" 가지를 삭제한다.

"x" 에게 자식 가지가 있을 경우, "x" 를 부모 가지로 병합하는 과정은 다음과 같다.

1) "x" 의 자식 가지들을 "x" 로 병합한다.

만약, "x" 의 자식 가지가 여러 개 일 경우, "x" 에서 분기된 순으로 자식 가지를 병합한다.

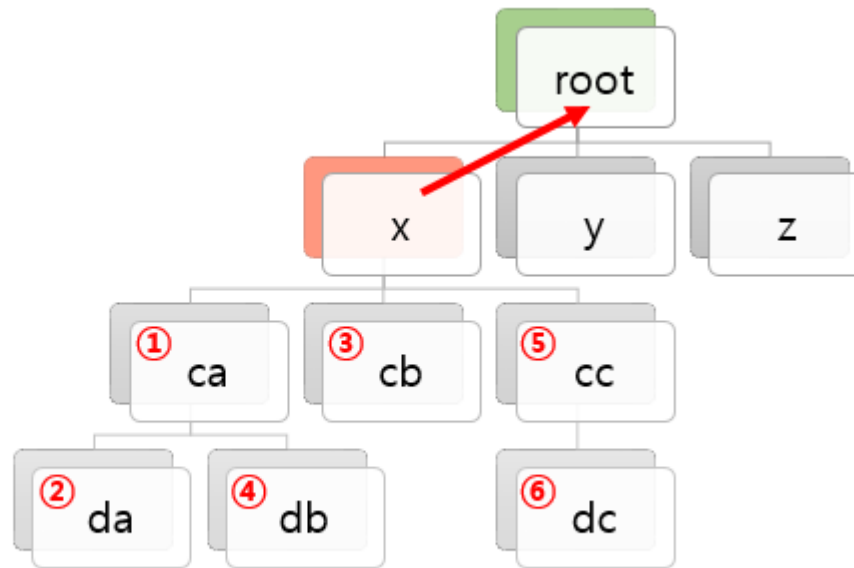
2) 자식 가지가 각각 분기된 가지를 가질 경우, 이 과정이 재귀적으로 반복된다.

3) "x" 가지의 자식 가지들의 병합이 모두 끝나면, "x" 를 "x" 의 부모 가지로 병합한다.



예를 들어 [Fig. 2] 와 같이, 가지들이 구성되어 있고 "x" 로부터 ①~⑥ 순서대로 분기되었을 때, "x" 를 부모 가지인 "root" 로 병합하는 경우를 생각해보자.

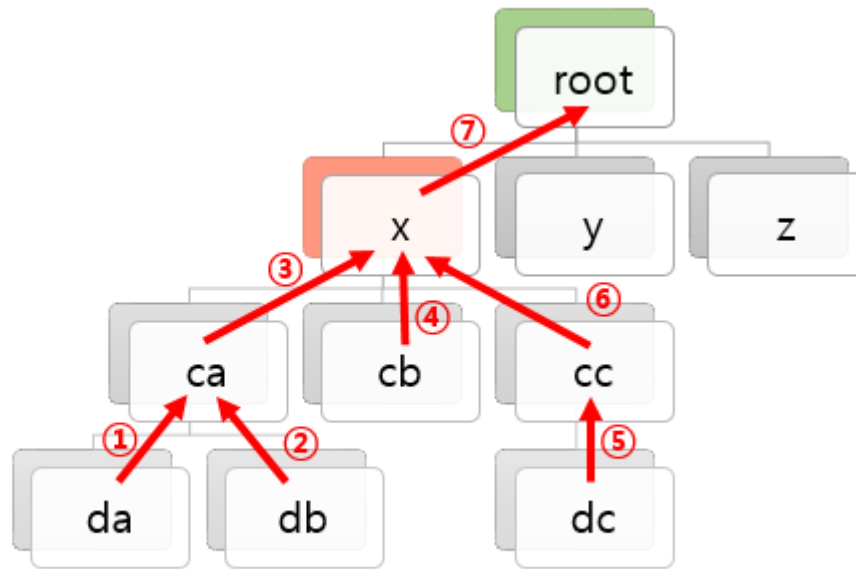
- ① "x" -> "ca"
- ② "ca" -> "da"
- ③ "x" -> "cb"
- ④ "ca" -> "db"
- ⑤ "x" -> "cc"
- ⑥ "cc" -> "dc"



[Fig. 2]

"x" 의 자손들을 "x" 가지로 병합하는 과정은 [Fig. 3]의 ①~⑥ 순으로 진행한다.  
"x" 의 자손들이 모두 병합되면, 그때 ⑦ 과 같이 "x" 의 부모인 "root" 로의 병합을 수행한다.

- ① "da" -> "ca"
- ② "db" -> "ca"
- ③ "ca" -> "x"
- ④ "cb" -> "x"
- ⑤ "dc" -> "cc"
- ⑥ "cc" -> "x"
- ⑦ "x" -> "root"



[Fig. 3]

※ 아래 API 설명을 참조하여 각 함수를 구현하라.

```
void init()
```

각 테스트 케이스의 처음에 한 번 호출된다.

호출 직후 "root" 이름의 가지만 존재하며,  
"root" 가지에는 아무 파일도 존재하지 않는다.

```
void create(int mTime, char mBranch[], char mFile[], char mData[])
```

mTime 시각, mBranch 가지에 이름이 mFile 인 파일을 생성한다.

생성된 파일의 내용은 mData 이다.

함수 호출 시점에 이름이 mBranch 인 가지가 존재한다.

함수 호출 시점에 mBranch 가지에는 이름이 mFile 인 파일이 없다.

mBranch, mFile, mData 는 모두 길이가 1 이상 10 이하인 문자열로,  
영문 소문자로만 이루어져 있으며 '\0' 으로 끝난다.

## *Parameters*

mTime : 함수가 호출되는 시각 ( $1 \leq \text{mTime} \leq 10,000$ )

mBranch : 파일을 생성될 가지의 이름

mFile : 생성된 파일의 이름

mData : 생성된 파일의 내용

```
void edit(int mTime, char mBranch[], char mFile[], char mData[])
```

mTime 시각, mBranch 가지 내 파일 중, 이름이 mFile 인 파일을 수정한다.

수정된 파일의 내용은 mData 이다.

함수 호출 시점에 이름이 mBranch 인 가지가 존재한다.

함수 호출 시점에 mBranch 가지에는 이름이 mFile 인 파일이 존재한다.

mBranch, mFile, mData 는 모두 길이가 1 이상 10 이하인 문자열로, 영문 소문자로만 이루어져 있으며 '\0' 으로 끝난다.

## *Parameters*

mTime : 함수가 호출되는 시각 ( $1 \leq mTime \leq 10,000$ )

mBranch : 수정할 파일이 위치한 가지의 이름

mFile : 수정하는 파일의 이름

mData : 수정된 파일의 내용

```
void branch(int mTime, char mBranch[], char mChild[])
```

mTime 시각에 이름이 mBranch 인 가지를 분기한다.

mBranch 로 부터 분기된 가지의 이름은 mChild 이다.

분기 후, mBranch 가지의 파일들을 모두 mChild 가지로 복사한다.

함수 호출 시점에 이름이 mBranch 인 가지가 존재한다.

mChild 는 init() 함수 호출 이후 생성된 모든 가지와 이름이 다르다. (생성 후, 삭제된 가지의 이름과도 다르다.)

mBranch, mChild 는 모두 길이가 1 이상 10 이하인 문자열로, 영문 소문자로만 이루어져 있으며 '\0' 으로 끝난다.

## *Parameters*

mTime : 함수가 호출되는 시각 ( $1 \leq mTime \leq 10,000$ )

mBranch : 분기할 가지의 이름

mChild : mBranch 로부터 분기 후 새로이 생성된 가지의 이름

```
void merge(int mTime, char mBranch[])
```

mTime 시각, 이름이 mBranch 인 가지를 해당 가지의 부모 가지로 병합한다.

mBranch 는 "root" 가 아니며,

함수 호출 시점에 이름이 mBranch 인 가지가 존재한다.

mBranch 는 길이가 1 이상 10 이하인 문자열로, 영문 소문자로만 이루어져 있으며  
'\0' 으로 끝난다.

## *Parameters*

mTime : 함수가 호출되는 시각 ( $1 \leq mTime \leq 10,000$ )

mBranch : 부모 가지로 병합할 가지의 이름

```
int readfile(int mTime, char mBranch[], char mFile[], char retString[])
```

mTime 시각, mBranch 가지에 있는 파일 중, 이름이 mFile 인 파일의 내용을 읽는다.

파일의 내용은 retString 문자 배열에 담고, 파일 내용의 길이를 반환한다.

함수 호출 시점에 이름이 mBranch 인 가지가 존재한다.

함수 호출 시점에 mBranch 가지에는 mFile 이름의 파일이 존재한다.

mBranch, mFile 은 모두 길이 1 이상 10 이하의 문자열로,  
영문 소문자로만 이루어져 있으며 '\0' 으로 끝난다.

## *Parameters*

mTime : 함수가 호출되는 시각 ( $1 \leq \text{mTime} \leq 10,000$ )

mBranch : 파일이 위치한 가지의 이름

mFile : 파일의 이름

retString : 반환할 파일 내용을 저장하는 문자열 변수

## *Returns*

읽은 파일 내용을 retString 배열에 담고, 그 길이를 반환한다.



## [제약사항]

1. 각 테스트 케이스 시작 시, `init()` 함수가 한 번 호출된다.
2. 각 테스트 케이스에서 `mTime` 은 1 부터 시작하며 `init()` 함수를 제외한, 매 함수 호출 시 마다 1 씩 증가한다.
3. 문자열은 항상 `'\0'` (널 문자)로 끝난다.
4. 각 테스트 케이스에서 `init()` 을 제외한 모든 함수 호출 횟수의 총합은 최대 10,000 이다.

# Problem analysis

1. `init()`
2. `create(1, "root", "main", "hello")`
3. `create(2, "root", "user", "world")`

root

branch	filename	data	creatTime	recentTime
root	main	hello	1	1
	user	world	2	2

4. `branch(3, "root", "dev")`

root

dev

동일하게  
복사

branch	filename	data	creatTime	recentTime
root	main	hello	1	1
	user	world	2	2
dev	main	hello	1	1
	user	world	2	2

5. `create(4, "root", "module", "mode")`

root

dev

branch	filename	data	creatTime	recentTime
root	main	hello	1	1
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	world	2	2

6. `create(5, "dev", "module", "mod")`

root

dev

branch	filename	data	creatTime	recentTime
root	main	hello	1	1
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	world	2	2
	module	mod	5	5

7. `edit(6, "root", "main", "hi")`

root

dev

branch	filename	data	creatTime	recentTime
root	main	<i>hello</i> -> hi	1	<i>1</i> -> 6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	world	2	2
	module	mod	5	5

8. `edit(7, "dev", "user", "word")`

root

dev

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	<i>world</i> -> word	2	<i>2</i> -> 7
	module	mod	5	5



9. `readfile(8, "root", "main")`

root

dev

hi와 길이 2를  
반환한다.

branch	filename	data	creatTime	recentTime
	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	word	2	7
	module	mod	5	5

10. `readfile(9, "root", "user")`

root

dev

world와 길이5를  
반환한다.

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	word	2	7
	module	mod	5	5

11. `readfile(10, "root", "module")`

root

dev

mode와 길이4를  
반환한다.

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	word	1	1
	user	world	2	7
	module	mod	5	5

12. `branch(11, "dev", "bugfix")`



dev와  
동일하게 복사.

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	word	2	7
	module	mod	5	5
bugfix	main	hello	1	1
	user	word	2	7
	module	mod	5	5

13. `create(12, "bugfix", "sononly", "son")`

root

dev

bugfix

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	word	2	7
	module	mod	5	5
bugfix	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	sononly	son	12	12

14. `create(13, "dev", "paronly", "par")`

root

dev

bugfix

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	paronly	par	13	13
bugfix	main	hello	1	7
	user	word	2	2
	module	mod	5	5
	sononly	son	12	12

15. `branch(14, "dev", "patch")`

root

dev

bugfix

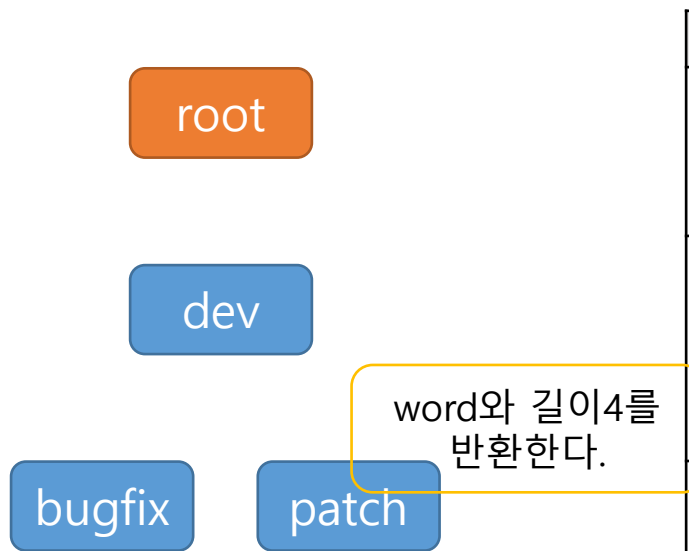
patch

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	paronly	par	13	13
bugfix	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	sononly	son	12	12
patch	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	paronly	par	13	13

# Problem analysis : 예제

TS개발협업도구

16. `readfile(15, "bugfix", "user")`



branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	paronly	par	13	13
bugfix	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	sononly	son	12	12
patch	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	paronly	par	13	13



17. `edit(16, "patch", "user", "lord")`

root

dev

bugfix

patch

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	paronly	par	13	13
bugfix	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	sononly	son	12	12
patch	main	hello	1	1
	user	<i>word</i> -> lord	2	16
	module	mod	5	5
	paronly	par	13	13

18. `edit(17, "bugfix", "user", "sword")`

root

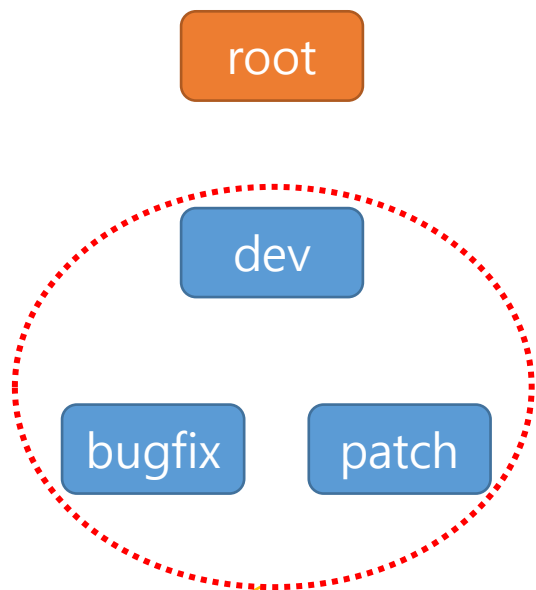
dev

bugfix

patch

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	paronly	par	13	13
bugfix	main	hello	1	1
	user	<i>word</i> -> sword	2	17
	module	mod	5	5
	sononly	son	12	12
patch	main	hello	1	1
	user	lord	2	16
	module	mod	5	5
	paronly	par	13	13

19. merge(18, "dev")



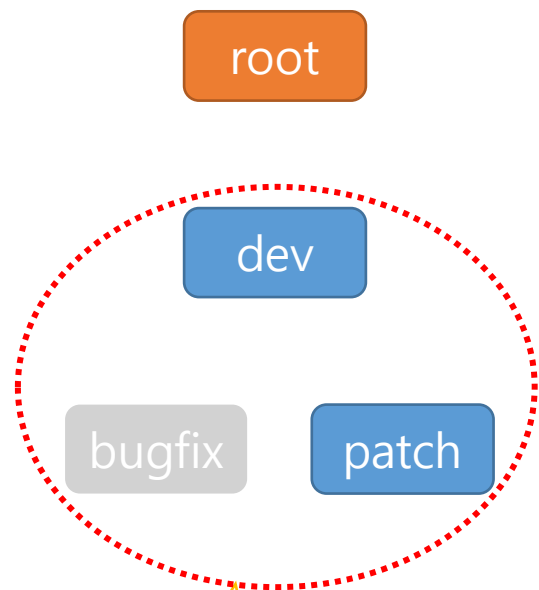
dev가지가 부모인 root로 병합된다.  
그런데 자식 가지가 있으므로 먼저  
자식가지를 dev에 병합한 후에  
dev를 root에 병합시킨다.

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	user	word	2	7
	module	mod	5	5
	paronly	par	13	13
bugfix	main	hello	1	1
	user	sword	2	17
	module	mod	5	5
	sononly	son	12	12
patch	main	hello	1	1
	user	lord	2	16
	module	mod	5	5
	paronly	par	13	13

# Problem analysis : 예제

TS개발협업도구

19\_1. merge(18, "dev")



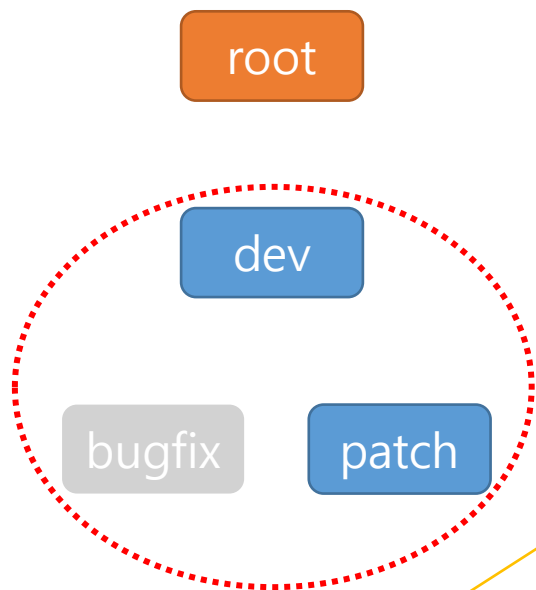
추가된 순서대로  
작업해야 한다.  
bugfix를 dev에  
병합시킨다.

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	<del>user</del>	<del>word</del>	<del>2</del>	<del>7</del>
	module	mod	5	5
	paronly	par	13	13
bugfix	<del>main</del>	<del>hello</del>	<del>1</del>	<del>1</del>
	user	sword	2	17
	<del>module</del>	<del>mod</del>	<del>5</del>	<del>5</del>
	sononly	son	12	12
patch	main	hello	1	1
	user	lord	2	16
	module	mod	5	5
	paronly	par	13	13

# Problem analysis : 예제

TS개발협업도구

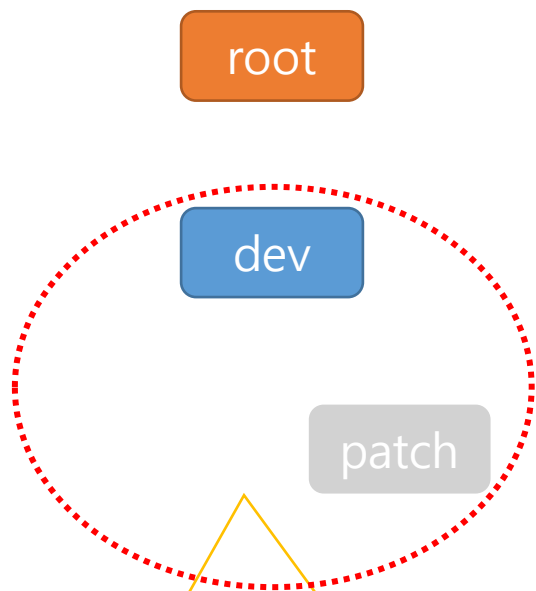
19\_2. merge(18, "dev")



bugfix를 dev에  
병합시킨 결과 이다.

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	module	mod	5	5
	paronly	par	13	13
	user	sword	2	17
	sononly	son	12	12
patch	main	hello	1	1
	user	lord	2	16
	module	mod	5	5
	paronly	par	13	13

19\_3. merge(18, "dev")



이제 patch를 dev에 병합시킨다.  
patch가지 에서 dev로 합쳐지는 파일은 없다.  
같거나 업데이트시간이 dev가지보다 최신이 아니다.

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	module	mod	5	5
	paronly	par	13	13
	user	sword	2	17
	sononly	son	12	12
patch	<del>main</del>	<del>hello</del>	<del>1</del>	<del>1</del>
	<del>user</del>	<del>lord</del>	<del>2</del>	<del>16</del>
	<del>module</del>	<del>mod</del>	<del>5</del>	<del>5</del>
	<del>paronly</del>	<del>par</del>	<del>13</del>	<del>13</del>

# Problem analysis : 예제

TS개발협업도구

19\_4. merge(18, "dev")

root

dev

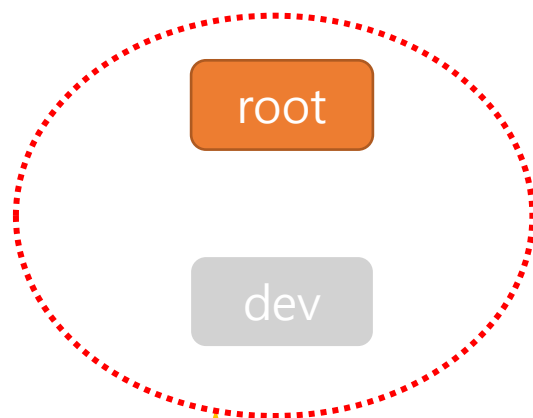
branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	world	2	2
	module	mode	4	4
dev	main	hello	1	1
	module	mod	5	5
	paronly	par	13	13
	user	sword	2	17
	sononly	son	12	12

patch를 dev에  
병합시킨 결과 이다.

# Problem analysis : 예제

TS개발협업도구

19\_5. merge(18, "dev")



이제 dev를 root에  
병합시킨다.

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	<del>user</del>	<del>world</del>	<del>2</del>	<del>2</del>
	module	mode	4	4
dev	<del>main</del>	<del>hello</del>	<del>1</del>	<del>1</del>
	<del>module</del>	<del>mod</del>	<del>5</del>	<del>5</del>
	paronly	par	13	13
	user	sword	2	17
	sononly	son	12	12

dev의 module파일의 수정 시각이  
root의 module파일의 수정 시각보다 더 최신이지만  
root의 module과 dev의 module은  
생성시각이 다르다.  
따라서 main의 module만 남고  
dev의 module은 삭제된다.



19\_6. merge(18, "dev")

root

branch	filename	data	creatTime	recentTime
root	main	hi	1	6
	user	sword	2	17
	module	mode	4	4
	sononly	son	12	12
	paronly	par	13	13

dev를 root에 병합시킨 후  
생성시간으로 오름차순  
정렬한 결과이다.

- 문제의 지문이 길고 설명이 복잡한 편이다.
- 시간에 따라 트리구조(노드를 가지:branch로 부른다.)를 관리하는 문제이다.
  - `init()` : 초기에는 root가지(branch) 하나만 있다.
  - `creat( tm, bid, fid, did)` : tm시각에 bid가지에 이름이 fid, 내용이 did인 파일을 생성한다.  
생성시각과 수정시각이 tm으로 설정된다. 하나의 가지가 갖는 **파일의 최대수는 50**이다.  
: 50개를 초과하는 경우 생성 시각이 가장 빠른 파일을 찾아 지워야 한다. 순차 탐색하면  $O(50)$
  - `edit(tm, bid, fid, did)` : tm시각에 bid가지에 fid파일의 내용을 did로 수정한다.  
수정시각이 tm으로 설정된다. : 순차 탐색하면  $O(50)$
  - `branch(tm, bid, cid)` : tm시각에 bid를 부모, cid를 자식으로 가지를 분기한다.  
cid는 bid의 속성(파일이름, 생성시각, 데이터, 수정시각)을 그대로 복사하여 생성된다.  
:  $O(50) * (\text{문자열길이}) \rightarrow$  문자열에 정수 id를 부여하고 처리하면  $O(50)$
  - `merge(tm, bid)` : Bid의 자손이 있으면 규칙에 따라 합쳐 Bid만 남긴 후 Bid를 Bid의 부모와 합친다. 규칙이 다소 복잡하다.  
: **생성된 만큼만 소멸될 수 있으므로  $O(50 * 50)$**
  - `readfile(tm, bid, fid, ret)` : bid가지의 fid파일의 내용을 ret에 복사하고 ret의 길이를 반환한다. : bid가지(노드)에서 fid를 순차탐색하면  $O(50)$

- 문자열이 등장하므로 인덱스 기반 프로그래밍을 위하여 정수로 치환하여 사용하는 전략을 선택할 수 있다. => 해시, unordered\_map 등
- 하나의 가지(노드)가 가지는 **파일의 수가 최대 50**이므로 새로운 가지(노드)가 생성될 때, 모두 복사하는 방법을 사용할 수 있다.
- 트리 구성을 위하여 vector 또는 링크드 리스트 등을 이용할 수 있다.
- 문제를 이해했다면 구현 복잡도가 가장 큰 부분은 merge부분이 될 것이다. 전체 시간복잡도가 여기서 결정된다.
- 문제 해결 전략을 생각해보자.

# Solution sketch

- branch, file, data 는 문자열로 주어진다.

각각을 정수로 바꾸어 사용하자.

`unordered_map<string, int>` 를 사용할 수 있다.

```
unordered_map<string, int> Bmap, Fmap, Dmap; // branch hash, file hash, data hash
```

- root를 제외하고 각 가지(노드)는 부모를 갖는다.

`merge()`를 수행하기 위해서는 부모가지(노드)번호를 보관할 필요가 있다.

```
int parents[LM]; // 각 노드의 부모 노드 번호(Bid)
```

- 트리를 구성할 필요가 있다.

여러가지 방법이 있으나 인접 배열을 이용한다면 `vector`를 이용할 수 있다.

```
vector<int> tree[LM]; // 트리
```

- 파일은 (파일이름, 생성시각, 내용(데이터), 수정시각) 이라는 속성을 갖는다.  
create()과 merge()에서의 우선순위를 고려하여  
연산자 오버로딩과 함께 클래스로 구현할 수 있다.

```
struct File {  
    int fid, ct, did, et; // file_id, createTime, data_id, editTime  
    bool operator<(const File&t)const { return ct < t.ct; } // create()에서 사용  
    bool operator>(const File&t)const { return ct > t.ct; } // merge()=>dfs()의 sort()에  
    // 사용  
};  
File frr[LM][101]; // 가지(노드)별 파일 목록 : 배열대신 vector를 사용할 수 있다.  
int flen[LM]; // 가지(노드)별 파일 목록 길이  
char str[LM][11]; // 데이터 내용: readfile()에서 사용
```

- 이제 각 API함수별 할 일을 정리해보자.

`void init()`

- 사용되는 모든 자료구조를 초기화한다.  
해시, 트리, 부모 노드 표시 배열 등.
- 트리의 초기에는 `root` 가지만 존재한다.

```
void create(int mTime, char mBranch[], char mFile[], char mData[])
```

- 각 문자열에 대한 정수 아이디 bid, fid, did 를 구한다.
- 새로운 파일을 생성한다.
- bid 가지(노드)에 파일수가 50미만이라면 단순히 추가한다.
- bid 가지(노드)에 파일수가 50이상이라면 생성시각이 최소인 파일을 찾아 현재 파일로 대체한다.

```
int Bid = getBID(mBranch);
```

```
int Fid = getFID(mFile);
```

```
int Did = getDID(mData); // 데이터 문자열은 백업해 둔다. readfile()에서 사용된다.
```

```
frr[Bid][flen[Bid]] = { Fid, mTime, Did, mTime }; // 새로운 파일 생성
```

```
if (flen[Bid] < 50) // 50개가 안된 경우 단수 추가
```

```
    flen[Bid]++;
```

```
else // 생성일이 가장 오래된 파일 위치에 새로운 파일을 저장
```

```
    *min_element(frr[Bid], frr[Bid] + flen[Bid]) = frr[Bid][flen[Bid]];
```



```
void edit(int mTime, char mBranch[], char mFile[], char mData[])
```

- 각 문자열에 대한 정수 아이디 bid, fid, did 를 구한다.
- bid가지에서 fid파일을 순차탐색으로 찾는다.  
(최대 50개 뿐이므로 단순 배열로 관리하자.)
- 찾은 파일(반드시 존재한다.)의 데이터와 수정시각을 수정한다.

```
int Bid = getBID(mBranch);
int Fid = getFID(mFile);
int Did = getDID(mData);
File*bp = frr[Bid];
for (int i = 0; i < flen[Bid]; ++i) {
    if (bp[i].fid == Fid) {
        bp[i].did = Did, bp[i].et = mTime; // 데이터와 수정시각 업데이트
        break;
    }
}
```

```
void branch(int mTime, char mBranch[], char mChild[])
```

- 각 문자열에 대한 정수 아이디 bid, cid 를 구한다.
- bid의 모든 파일을 복사하여 cid 가지(노드)를 생성한다.
- bid의 자식목록의 맨 뒤에 cid를 추가한다. (추가된 순서가 merge에서 중요하다.)
- cid의 부모로 bid를 설정한다.

```
int Bid = getBID(mBranch), Cid = getBID(mChild);
```

```
// Bid가지를 복사하여 Cid가지 생성
```

```
memcpy(frr[Cid], frr[Bid], sizeof(File) * flen[Bid]); // 반복문으로 처리할 수도 있다.
```

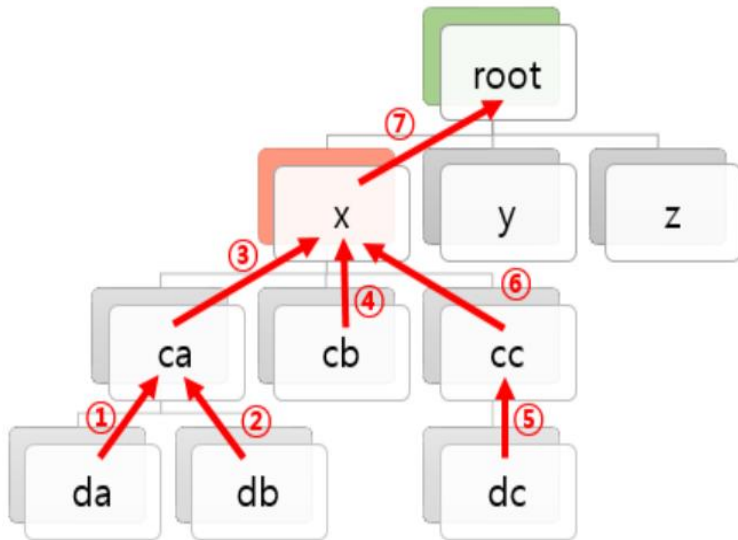
```
flen[Cid] = flen[Bid];
```

```
tree[Bid].push_back(Cid); // 트리 구성 업데이트
```

```
parents[Cid] = Bid; // 부모 노드 번호 저장
```

```
void merge(int mTime, char mBranch[])
```

- 문자열에 대한 정수 아이디 `bid`를 구한다.  
`bid`의 부모 아이디 `pid`를 구한다.
- 깊이우선탐색을 이용하여 규칙에 맞게 트리를 순회하며 합친다.
- 규칙의 디테일을 잘 적용하여야 한다.

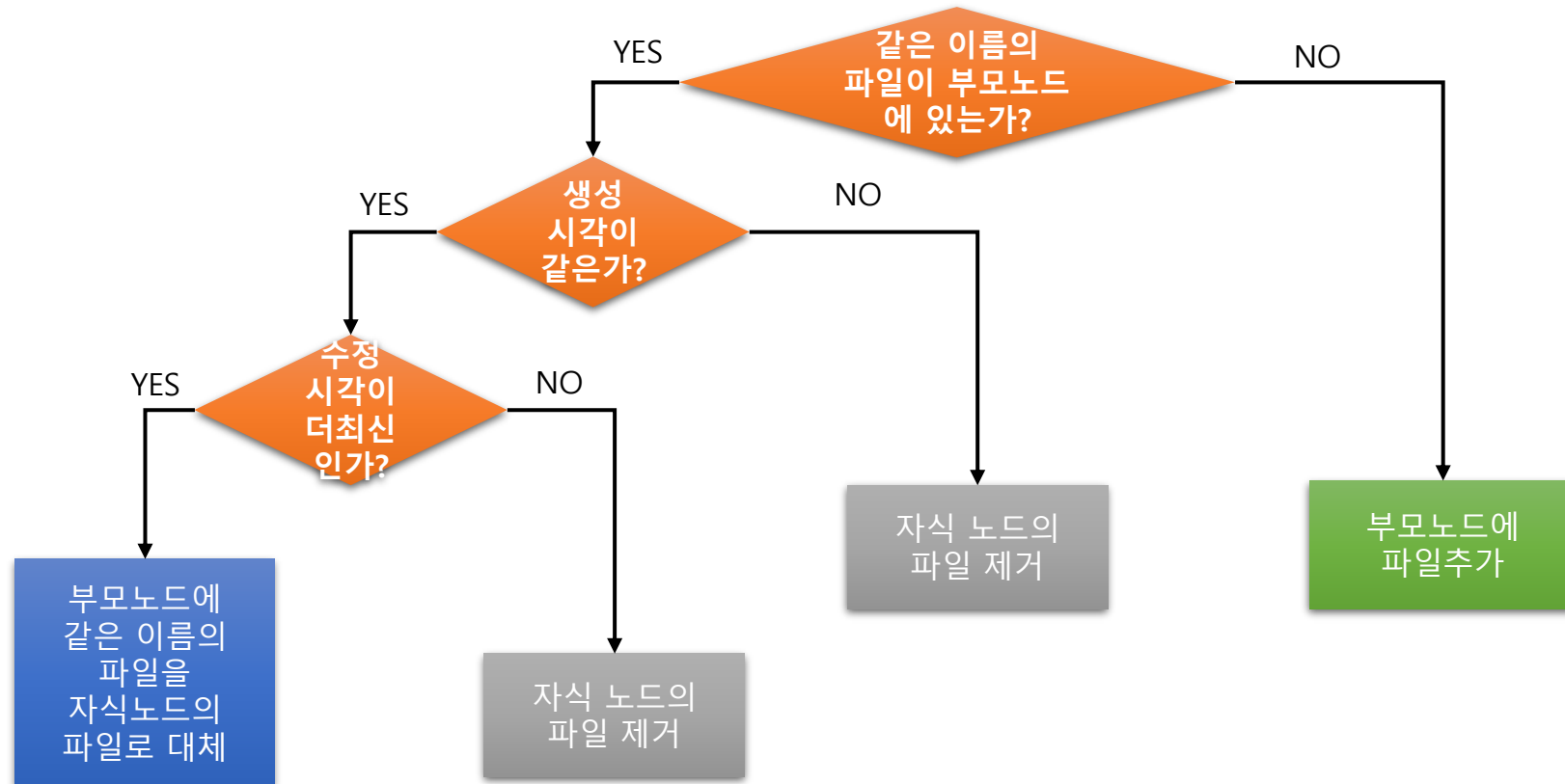
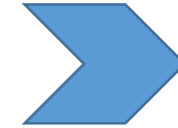


// 깊이우선탐색+생성시각 오름차순 순으로 방문하여 가지 합치기

```
void dfs(int cur, int par) {  
    for (auto child : tree[cur]) {  
        // 이전에 merge되어 제거된 경우는 제외시키기  
        if (flen[child])  
            dfs(child, cur);  
    }  
    .....  
}
```

`void merge(int mTime, char mBranch[])`: 계속

- 부모 노드와 자식 노드를 병합하기  
자식 가지(노드)의 각 파일에 대하여 아래 순서도를 따른다.



```
int readfile(int mTime, char mBranch[], char mFile[], char retString[])
```

- 각 문자열에 대한 정수 아이디 bid, fid 를 구한다.
- bid가지의 파일을 순회하며 fid파일을 찾는다.
- fid를 찾았다면 내용을 retString[]에 복사하고  
그 길이를 반환한다.

```
int Bid = getBID(mBranch), Fid = getFID(mFile);
for (auto t : frr[Bid]) {
    if (t.fid == Fid) {          // Bid가지에서 Fid파일을 찾기
        strcpy(retString, str[t.did]); // 데이터 내용 복사
        break;
    }
}
return (int)strlen(retString); // 데이터 길이 반환
```

## [Summary]

- 프로에서 필요한 기본 소양을 바탕으로 구현능력을 시험한 문제이다.
  - ✓ 문제를 읽고 분석하기
  - ✓ 문제를 재정의하기
  - ✓ 적절한 자료구조와 알고리즘(문제해결방법과 절차)을 선택하고 구현하기  
해시, 트리구성(인접배열, 인접리스트), 트리탐색  
두 배열을 규칙에 맞게 병합하기 등.
- 기본기를 탄탄히 하기에 좋은 문제이다.
- 다양한 방법으로 구현해보고 타인의 코드를 분석해보자.

# Code example1

# Code example1

TS개발협업도구

```
#ifndef _CRT_SECURE_NO_WARNINGS
#define _CRT_SECURE_NO_WARNINGS
#endif

#include <cstring>
#include <vector>
#include <string>
#include <algorithm>
#include <unordered_map>
using namespace std;

const int LM = 10001;

unordered_map<string, int> Bmap, Fmap, Dmap; // branch hash, file hash, data hash
int bcnt, fcnt, dcnt; // branch cnt, file cnt, data cnt;
int parents[LM]; // 각 노드의 부모노드 번호(Bid)
vector<int> tree[LM]; // 트리
char str[LM][11]; // 데이터 내용: readfile()에서 사용
struct File {
    int fid, ct, did, et; // file_id, creatTime, data_id, editTime
    bool operator<(const File&t)const { return ct < t.ct; } // create()에서 사용
    bool operator>(const File&t)const { return ct > t.ct; } // merge()=>dfs()의 sort()에서 사용
};
File frr[LM][101]; // 노드별 파일 목록
int flen[LM]; // 노드별 파일 목록 길이
```



```
void init() {                                     // 초기화
    for (int i = 0; i <= bcnt; ++i) {
        parents[i] = 0;
        flen[i] = 0;
        tree[i].clear();
    }
    Bmap.clear(), Fmap.clear(), Dmap.clear();
    bcnt = fcnt = dcnt = 0;
    Bmap["root"] = ++bcnt;                        // 초기 트리에는 root만 존재
}

int getBID(char*s) {                             // branch 정수 id 생성 또는 가져오기
    auto it = Bmap.find(s);
    if (it == Bmap.end()) it = Bmap.insert({ s, ++bcnt }).first;
    return it->second;
}

int getFID(char*s) {                             // file 정수 id 생성 또는 가져오기
    auto it = Fmap.find(s);
    if (it == Fmap.end()) it = Fmap.insert({ s, ++fcnt }).first;
    return it->second;
}
```

# Code example1

TS개발협업도구

```
int getDID(char*s) {                                     // data 정수 id 생성 또는 가져오기
    auto it = Dmap.find(s);
    if (it == Dmap.end()) {
        it = Dmap.insert({ s, ++dcnt }).first;
        strcpy(str[dcnt], s);                             // 처음 등장한 자료라면 문자열을 따로 백업하기
    }
    return it->second;
}

void create(int mTime, char mBranch[], char mFile[], char mData[]) { // 파일 생성
    int Bid = getBID(mBranch);
    int Fid = getFID(mFile);
    int Did = getDID(mData);
    frr[Bid][flen[Bid]] = { Fid, mTime, Did, mTime }; // 새로운 파일 생성
    if (flen[Bid] < 50)                               // 50개가 안된 경우 단수 추가
        flen[Bid]++;
    else // 생성일이 가장 오래된 파일 위치에 새로운 파일을 저장
        *min_element(frr[Bid], frr[Bid] + flen[Bid]) = frr[Bid][flen[Bid]];
}
```

# Code example1

TS개발협업도구

```
void edit(int mTime, char mBranch[], char mFile[], char mData[]) { // 파일 수정
    int Bid = getBID(mBranch);
    int Fid = getFID(mFile);
    int Did = getDID(mData);
    File*bp = frr[Bid];
    for (int i = 0; i < flen[Bid]; ++i) {
        if (bp[i].fid == Fid) {
            bp[i].did = Did, bp[i].et = mTime; // 데이터와 수정시각 업데이트
            break;
        }
    }
}

void branch(int mTime, char mBranch[], char mChild[]) { // 가지 생성
    int Bid = getBID(mBranch), Cid = getBID(mChild); // Bid가지를 복사하여 Cid가지 생성
    memcpy(frr[Cid], frr[Bid], sizeof(File) * flen[Bid]), flen[Cid] = flen[Bid];
    tree[Bid].push_back(Cid);
    parents[Cid] = Bid;
}
```

# Code example1

TS개발협업도구

```
void dfs(int cur, int par) { // 깊이우선탐색+생성시각오름차순 순으로 방문하여 가지 합치기
    for (auto child : tree[cur]) {
        if (flen[child]) // 이전에 merge되어 제거된 경우는 제외시키기
            dfs(child, cur);
    }
    File*pb = frr[par];
    int i, j, len = flen[par];
    for (j = 0; j < flen[cur]; ++j) {
        File&k = frr[cur][j];
        for (i = 0; i < len; ++i) {
            if (k.fid == pb[i].fid) { // 같은 이름의 파일을 찾은경우
                if (k.ct == pb[i].ct && k.et > pb[i].et) {
                    pb[i] = k; // 생성시각이 같고 cur가지의 파일 수정시각이 최신이라면
                }
                else; // 그렇지 않은 경우 무시됨
                break; // cur가지의 k파일이 선택되는 무시되든 임무 완료이므로 반복문 종료
            }
        }
        if (i == len) pb[flen[par]++] = k; // par가지에 존재하지 않는 새로운 파일은 par에 추가
    }
}
```



# Code example1

## TS개발협업도구

```
if (flen[par] > 50) { // 파일수가 50을 초과한 경우 50개로 맞추기
    //sort(pb, pb + flen[par], greater<File>()); // 생성시각의 내림차순으로 파일 전체를 정렬
    //partial_sort(pb, pb + 50, pb + flen[par], greater<File>()); // 생성시각의 내림차순으로 파
일을 50개만 정렬
    nth_element(pb, pb + 49, pb + flen[par], greater<File>()); // 생성시각의 내림차순으
로 50번째를 기준으로 파일들을 나누기
    flen[par] = 50;
}
flen[cur] = 0; // cur가지가 제거된것을 표시, 생성시각순서는 그대로 유지됨
}

void merge(int mTime, char mBranch[]) { // 가지 합치기
    int Bid = getBID(mBranch), pid = parents[Bid];
    dfs(Bid, pid); // 깊이우선탐색+생성시각오름차순 순으로 방문하여 가지 합치기
}

int readfile(int mTime, char mBranch[], char mFile[], char retString[]) {
    int Bid = getBID(mBranch), Fid = getFID(mFile);
    for (auto t : frr[Bid]) {
        if (t.fid == Fid) { // Bid가지에서 Fid파일을 찾기
            strcpy(retString, str[t.did]); // 데이터 내용 복사
            break;
        }
    }
    return (int)strlen(retString); // 데이터 길이 반환
}
```

# Code example2

# Code example2

TS개발협업도구

```
#ifndef _CRT_SECURE_NO_WARNINGS
#define _CRT_SECURE_NO_WARNINGS
#endif

#include <cstring>
#include <string>
#include <algorithm>
#include <unordered_map>
using namespace std;

const int LM = 10001;

unordered_map<string, int> Bmap, Fmap, Dmap; // branch hash, file hash, data hash
int bcnt, fcnt, dcnt; // branch cnt, file cnt, data cnt;
int parents[LM]; // 각 노드의 부모노드 번호(Bid)
char str[LM][11]; // 데이터 내용: readfile()에서 사용
struct File {
    int fid, ct, did, et; // file_id, creatTime, data_id, editTime
    bool operator<(const File&t)const { return ct < t.ct; } // create()에서 사용
    bool operator>(const File&t)const { return ct > t.ct; } // merge()=>dfs()의 sort()에서 사용
};
File frr[LM][101]; // 노드별 파일 목록
int flen[LM];
```

```
struct Tree {
    int id;
    Tree*prev, *next, *child;
    void alloc(int nid, Tree*np, Tree*nc) {
        id = nid, child = nc, child->init(), child->prev = child;
        if (np) np->next = this;
        next = nullptr;
    }
    void init() { prev = next = child = nullptr; }
}tree[LM + 5], tbuf[LM + 5];
int tcnt;

void init() {                                     // 초기화
    for (int i = 0; i <= bcnt; ++i) {
        parents[i] = 0;
        flen[i] = 0;
        tree[i].init();
    }
    Bmap.clear(), Fmap.clear(), Dmap.clear();
    bcnt = fcnt = dcnt = tcnt = 0;
    Bmap["root"] = ++bcnt;                       // 초기 트리에는 root만 존재
    tree[bcnt].alloc(bcnt, 0, tbuf + tcnt++ );
}
```



# Code example2

TS개발협업도구

```
int getBID(char*s) {                                // branch 정수 id 생성 또는 가져오기
    auto it = Bmap.find(s);
    if (it == Bmap.end()) it = Bmap.insert({ s, ++bcnt }).first;
    return it->second;
}
int getFID(char*s) {                                // file 정수 id 생성 또는 가져오기
    auto it = Fmap.find(s);
    if (it == Fmap.end()) it = Fmap.insert({ s, ++fcnt }).first;
    return it->second;
}
int getDID(char*s) {                                // data 정수 id 생성 또는 가져오기
    auto it = Dmap.find(s);
    if (it == Dmap.end()) {
        it = Dmap.insert({ s, ++dcnt }).first;
        strcpy(str[dcnt], s);                        // 처음 등장한 자료라면 문자열을 따로 백업하기
    }
    return it->second;
}
```

```
void create(int mTime, char mBranch[], char mFile[], char mData[]) { // 파일 생성
    int Bid = getBID(mBranch);
    int Fid = getFID(mFile);
    int Did = getDID(mData);
    frr[Bid][flen[Bid]] = { Fid, mTime, Did, mTime }; // 새로운 파일 생성
    if (flen[Bid] < 50) // 50개가 안된 경우 단수 추가
        flen[Bid]++;
    else // 생성일이 가장 오래된 파일 위치에 새로운 파일을 저장
        *min_element(frr[Bid], frr[Bid] + flen[Bid]) = frr[Bid][flen[Bid]];
}

void edit(int mTime, char mBranch[], char mFile[], char mData[]) { // 파일 수정
    int Bid = getBID(mBranch);
    int Fid = getFID(mFile);
    int Did = getDID(mData);
    File*bp = frr[Bid];
    for (int i = 0; i < flen[Bid]; ++i) {
        if (bp[i].fid == Fid) {
            bp[i].did = Did, bp[i].et = mTime; // 데이터와 수정시각 업데이트
            break;
        }
    }
}
```

# Code example2

TS개발협업도구

```
void branch(int mTime, char mBranch[], char mChild[]) { // 가지 생성
    int Bid = getBID(mBranch), Cid = getBID(mChild); // Bid가지를 복사하여 Cid가
    memcpy(frr[Cid], frr[Bid], sizeof(File) * flen[Bid]), flen[Cid] = flen[Bid];
    tree[Cid].alloc(Cid, tree[Bid].child->prev, tbuf + tcnt++); // tree[Bid]의 child목록 끝에 추가
    tree[Bid].child->prev = tree + Cid; // tree[Bid]의 child목록에서 마지막 추가된 노드는 Cid
    parents[Cid] = Bid;
}

void dfs(int cur, int par) { // 깊이우선탐색+생성시각오름차순 순으로 방문하여 가지 합치기
    Tree*p = tree[cur].child->next;
    for (; p;p = p->next) {
        if(flen[p->id]) // 이전에 merge되어 제거된 경우는 제외시키기
            dfs(p->id, cur);
    }
    File*pb = frr[par];
    int i, j, len = flen[par];
```

```
for (j = 0; j < flen[cur]; ++j) {
    File&k = frr[cur][j];
    for (i = 0; i < len; ++i) {
        if (k.fid == pb[i].fid) { // 같은 이름의 파일을 찾은경우
            if (k.ct == pb[i].ct && k.et > pb[i].et) {
                pb[i] = k; // 생성시각이 같고 cur가지의 파일 수정시각이 최신이라면
            }
            else; // 그렇지 않은 경우 무시됨
            break; // cur가지의 k파일이 선택되는 무시되든 임무 완료이므로 반복문 종료
        }
    }
    if (i == len) pb[flen[par]++] = k; // par가지에 존재하지 않는 새로운 파일은 par에 추가
}
if (flen[par] > 50) { // 파일수가 50을 초과한 경우 50개로 맞추기
    //sort(pb, pb + flen[par], greater<File>()); // 생성시각의 내림차순으로 파일을 정렬
    //partial_sort(pb, pb + 50, pb + flen[par], greater<File>()); // 생성시각의 내림차순으로 파
일을 50개만 정렬
    nth_element(pb, pb + 49, pb + flen[par], greater<File>()); // 생성시각의 내림차순으
로 50번째를 기준으로 파일들을 나누기
    flen[par] = 50;
}
flen[cur] = 0;
}
```

# Code example2

TS개발협업도구

```
void merge(int mTime, char mBranch[]) {                // 가지 합치기
    int Bid = getBID(mBranch), pid = parents[Bid];
    dfs(Bid, pid);                // 깊이우선탐색+생성시각오름차순 순으로 방문하여 가지 합치기
}

int readfile(int mTime, char mBranch[], char mFile[], char retString[]) {
    int Bid = getBID(mBranch), Fid = getFID(mFile);
    for (auto t : frr[Bid]) {
        if (t.fid == Fid) {        // Bid가지에서 Fid파일을 찾기
            strcpy(retString, str[t.did]); // 데이터 내용 복사
            break;
        }
    }
    return (int)strlen(retString); // 데이터 길이 반환
}
```

**Thank you.**