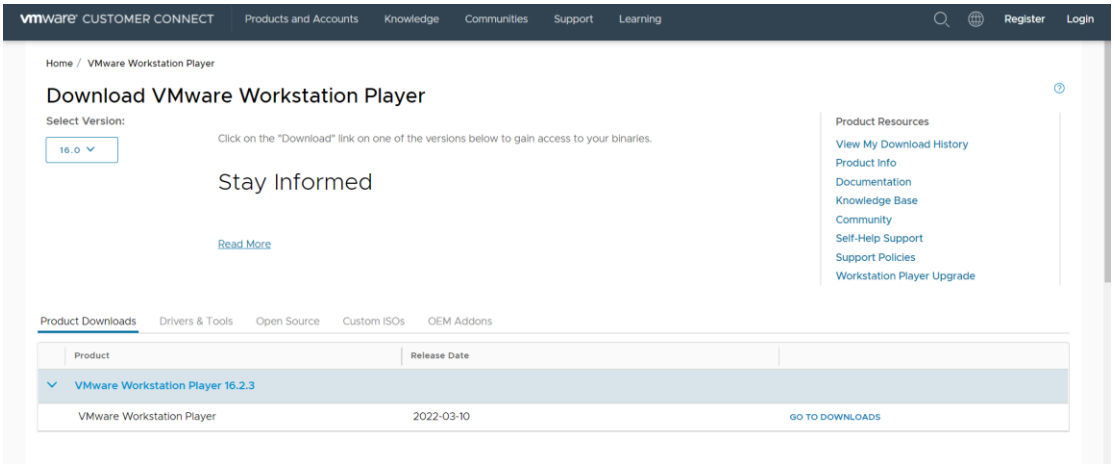
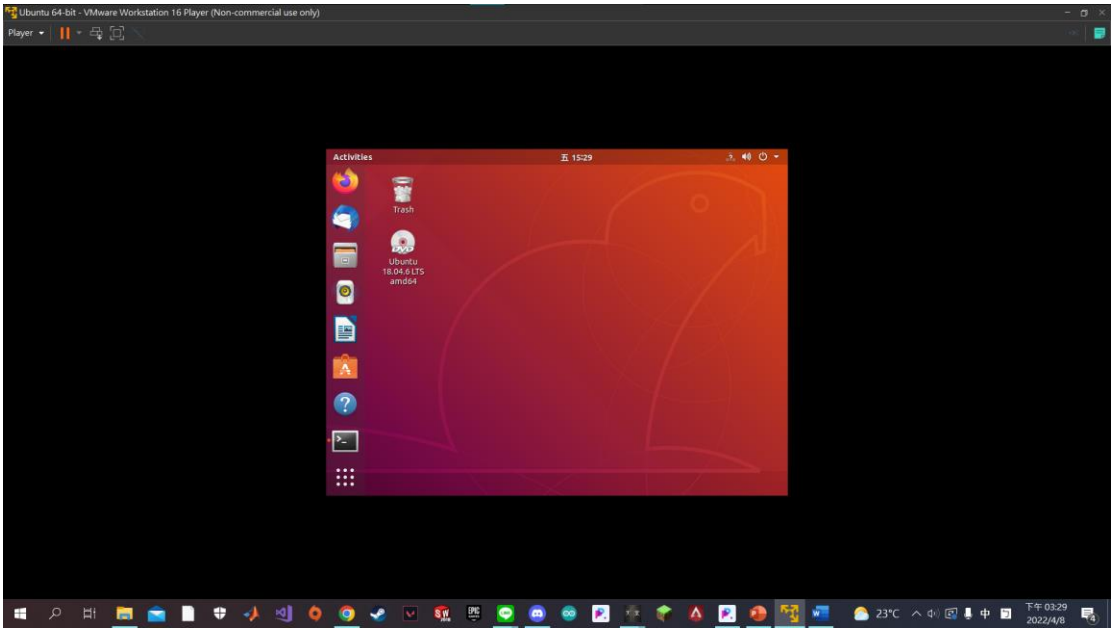
PART A



使用 VMware 下載

並且輸入使用者與地方 在設定切割其所需的 swap ,boot ,跟主空間。



登入後打開 terminal(ctrl + alt + t)

並創建資料夾

```
e14073043@E14073043:~$ mkdir operating_system
```

更換現在空間

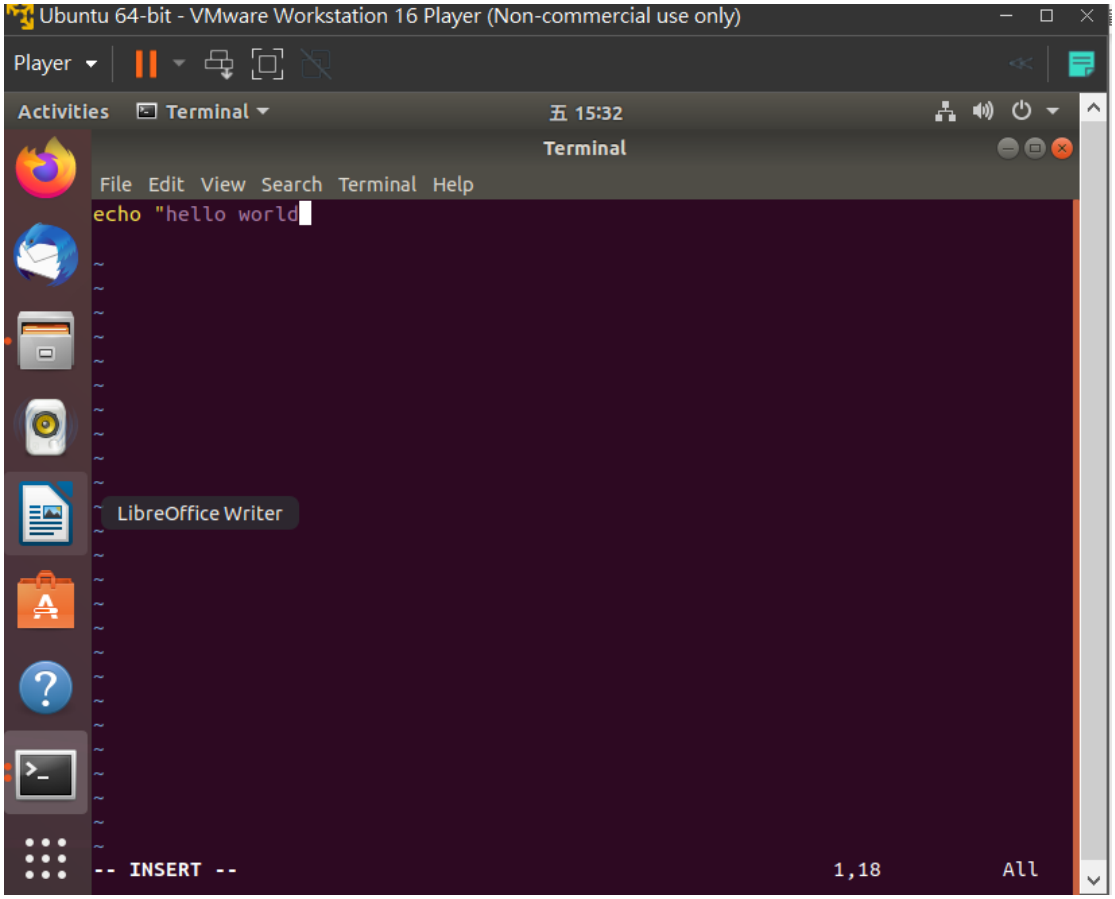
```
e14073043@E14073043:~$ cd operating_system/
```

創建檔案

```
e14073043@E14073043:~/operating_system$ vim hello.sh
```

輸入 echo “hello world”

Esc => :wq



輸出 hello world

```
e14073043@E14073043:~/operating_system$ ./hello.sh
hello world
```

PART B

基本規則:

1. 左邊子樹延伸下去的數值會比根(也就是第一個黑色球)還來的小
2. 右半邊子樹會比根來的大
3. 看子樹的分支，也可以看出左邊會比子樹的根來的小，右邊會比子樹來的大，簡而言之，就是無論從哪裡看進去，都符合第一點與第二點。
4. 子節點最多兩個
5. 根總是黑色
6. 節點不是黑就是紅
7. 節點為紅，則必定會有兩個黑子節點，不會出現紅節點連紅子節點
8. 所有的葉節點都必定是黑色(葉節點會用 NIL 表示，如果不使用這樣的方式有時會感覺與規則互相矛盾，比如說紅色的節點下必接兩個黑色節點)
9. 從根開始計算，到最下面的節點(又稱為葉節點或是空子節點)，都包含相同數量的黑色節點。

紅黑樹-搜尋演算法

紅黑樹的搜尋演算法與二元搜尋樹是一樣的。

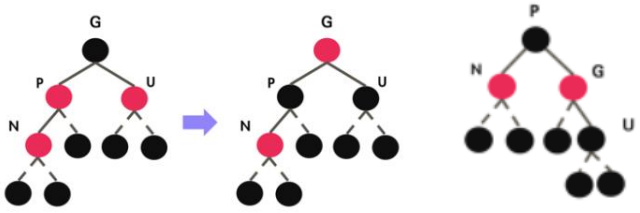
搜尋演算法:

會先從根部開始(就是樹的最上端開始)，如果需要搜尋的數值小於當下的值，則向左尋找，反之，要找較大的數值則向右尋找，每次與到分支都需要做一樣的動作，看搜尋的值是否比當下的值大還小，就可以判斷出要向右還是向左尋找，找到後就回傳 true。

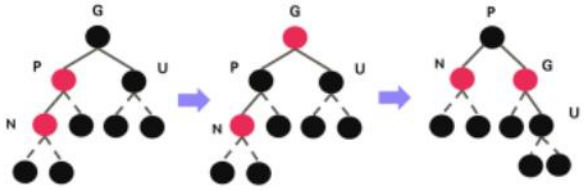
紅黑樹-插入演算法

從根結點開始向下尋找位置插入，插入節點後，並須先檢查紅黑樹是否有符合規則，如果發現不平衡，就要開始修復，將其符合規則。

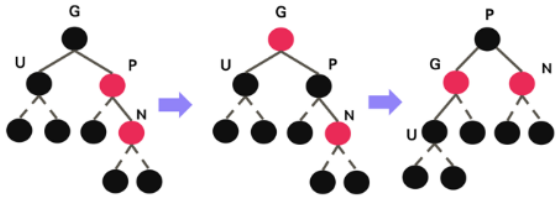
1. 如果插入的是根節點，則違反根為黑色，則直接將插入的改成黑色即可
2. 若插入的節點其父節點為黑色，也不需要做任何更動。
3. 若插入的節點其父節點為紅色，且其同層的節點也為紅色，則先將所有紅色的父節點變成黑色，並將根節點先轉成紅色，這時候會發現其根與插入的節點皆為紅色，做右旋的動作，就可以將黑色父節點變成根，然後子節點往上提，原始的根則轉變成子節點，就符合規則。



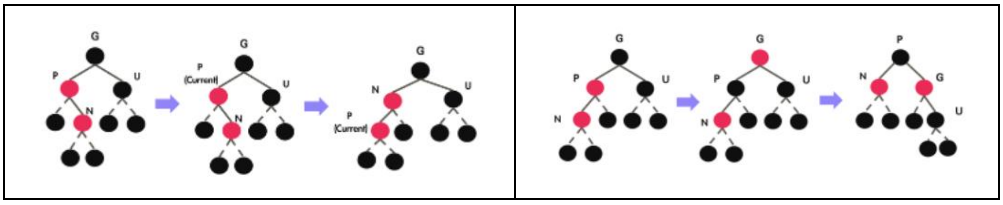
4. 若插入的節點其父節點為紅色，但其另一個父節點不為紅色，則將紅色的父節點變成黑色，再將根轉成紅色節點，這時候會發現其與上述第一次轉變後的結果一樣，所以只要做右旋動作即可。



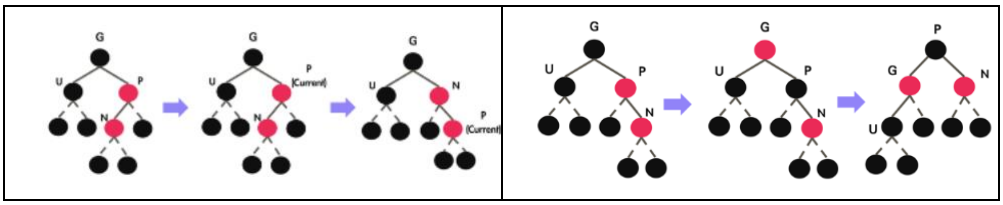
5. 若插入的節點其父節點為紅色，違反紅接紅，要重新調整，而另一個同層父節點是黑色，則先將插入的父節點(為紅)改成黑色，並且把根改成紅色，就會發現他與上述的長相雷同，只要做一次左旋，就可以完成調整。



6. 若插入的節點其父節點為紅色，違反紅接紅，而且其插入的位置為父節點的右邊，則先對他作左旋，讓其與第四個情況相同，然後再根據第四個步驟完成重建。



7. 若插入的節點其父節點為紅色，違反紅接紅，且同層左側的父節點為黑色，先對其插入節點的父節點做右旋，將紅色節點放在最右翼，這樣就與第五個相同，在做重建。



紅黑樹-刪除演算法

1. 如果刪除最下面的節點(葉節點)剛好是紅色節點，則不須要做平衡，因為並不影響規則。
2. 如果刪除的是最下面為黑色節點，則必須對其建立一個空節點，並做**調整**，來符合規則。
3. 如果刪除的節點只有一個子節點，則必須先判斷子節點是甚麼顏色，如果為黑色，則就先直接頂替當前節點，在對其**調整**。
4. 如果刪除的節點有兩個子節點，把這兩個子節點的資料進行交換，不要複製顏色，也不要改變其原有的父子等關係，然後重新進行刪除的動作，此處的刪除是指對交換資料後的中序後繼節點做刪除，就會與前面三點一樣的情況，所以只要對前面三個做**調整**即可。

調整演算法:

1. 刪除後，前的節點為紅色，則將其直接變成黑色即可
2. 刪除後，當前節點為黑色，剛好也為根節點，則這樣即可。
3. 刪除後，當前節點是黑色且兄弟節點為紅色，當前節點為父節點的左子節點，則把兄弟節點變成父節點的顏色，把父節點變成紅色，然後於父節點上執行左旋，重新開始判斷。
4. 刪除後，當前節點是黑色且同層節點為紅色，當前節點為父節點的右子節點，則把同層節點變成父節點的顏色，把父節點變成紅色，然後於父節點上右旋，重新開始判斷。
5. 刪除後，當前節點是黑色且其父節點和同層節點皆為黑色，同層節點的兩個子節點全為黑色，則把同層節點變紅，然後把父節點當成新的當前節點，重新開始判斷。
6. 當前節點如果為黑色且同層節點也為黑色，同層其他節點的兩個子節點也全為黑色，但其父節點為紅色，則將同層其他節點轉成紅色，並將其父節點轉成黑色。
7. 當前節點如果為黑且其同層節點也為黑色，但其同層其他節點的左邊節點為紅色，右邊節點為黑色，且當前節點是父節點的左邊，則把同層其他節點轉成紅，並將同層其他節點的左邊子節點轉成黑，然後對同層其他節點做右旋，重新開始判斷。
8. 當前節點如果為黑且其同層節點(兄弟節點)也為黑，兄弟節點的左邊節點為黑，右邊為紅，且當前節點是父節點的右邊子節點，則把兄弟節點染紅，再將兄弟節點的右側子節點轉黑，然後對兄弟節點右旋，重新開始判斷。
9. 當前節點為黑且其兄弟節點也為黑色，其兄弟節點的右側節點為紅，但其左側可以黑或紅，且當前節點是父節點的左側子節點，則把兄弟節點的顏色轉為當前節點的父節點顏色，如果當前的父節點

為紅，則轉黑，並把兄弟節點的右側節點轉黑，然後以當前節點的父節點做左旋，即可完成。

10. 當前節點為黑且其兄弟節點也為黑色，兄弟節點的左子節點為紅，右側節點顏色可以為黑或紅，當前節點為父節點的右側子節點，則把兄弟節點轉成與當前節點的父節點的顏色，如果當前節點的父節點不是黑色，則讓其變色，再讓兄弟節點左側節點轉黑，然後對當前節點的父節點做右旋，即可完成。

紅黑樹與其他演算法的比較

	RB Tree	AVL Tree	Splay Tree
規則	較為不嚴格的規則使其旋轉次樹降低，任意動作都可以再三次旋轉與變色中完成	較為嚴格，追求完全平衡	將最近加入的資料或搜索的資料放在接近根節點的附近
搜索	較慢	較快	較快
插入	較快	較慢	較快
刪除	較快	較慢	未知
應用情境	如果需要大量的異動資料，此法較為方便。	查詢速度較為快速，因為其一直保持平衡狀態	如果需要對於較近的資料進行變更或是要找剛放入的資料位置，會相比其他樹來的快速

紅黑樹程式註解

已經將註解打在程式中，請助教可以點開程式註解閱讀。

紅黑樹在 Linux 的應用

在 Linux 的 kernel 中有許多 Red-Black trees 的應用，比如說 deadline 與 CFQ I/O 排程器使用 RBtrees 追蹤請求；ext3 檔案系統使用 RBtrees 紀錄 directory entries。Virtual memory areas (VMAs) 也是以 RBtrees 追蹤、Epoll 的檔案描述器、密碼學用的 key 值、以及在"階層式 token bucket 的網路封包。

可以看出有 Linux 系統有相當多的地方使用 RBtrees，但並不代表 RBtrees 就是最好的方法，當使用者所輸入的 indexes 之間差距都是相當大的時候，則不會使用 RBtrees，可以想像，因為 RBtrees 本身是一個平衡樹，當 indexes 都偏向一邊，RBtrees 必須一直做 rotations，將其平衡，這樣就需要消耗大量的時間進行調整，這時候使用 radix tree 就會相對於 RBtrees 來的好，因為它只對有用到的 indexes 創造其空間陣列，這樣在查找上就可以省下許多時間。

參考資料

資料結構與演算法筆記
https://clu.gitbook.io/data-structure-note/1.4.1.1-rb-tree-insert

紅黑樹
https://www.cnblogs.com/sunankang/p/14309507.html

軟體學徒 FOREVER
http://reborn2266.blogspot.com/2012/01/red-black-trees-rbtree-in-linux.html

Torvalds github
https://github.com/torvalds/linux