HW06

409410050 資工二王謙靜

A

為簡化問題，假設 name2 必為 name1 的後繼者，因為未說明具題有多少種 relationship，無法得知每一種 reationship 彼此間誰是誰的後代，故做了這樣的假設。
另 $D(s)$ 為找 s 這個人的所有後繼者。
經過觀察可以發現此問題具有遞迴關係，要找 D(David) 就等於找所有 David的第一代後繼者 x 們的後繼者的聯集，而此遞迴的 base case 則是那一些沒有後繼者的人，即

$$D(s) = \begin{cases} \bigcup_{i \in \pi_{name_2}(\sigma_{name_1=\text{"david"}}(R))} D(i) \cup i & \text{if } \pi_{name_2}(\sigma_{name_1=\text{"david"}}(R)) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

由一個 relational algebra，每次只能找到一層的後繼者，故想得到答案要做非常多次的查詢(大概答案有多少筆，就要查詢幾次)，且我們無法在找第一個 $D(s)$ 的當下就知道要做幾次查詢，故無法使用 relational algebra 來列出 David 所有的後繼者。

除此問題之外，還有另一個較為簡單的問題，若 David 有兩個相同名字的後輩，則使用 relational model 會無法輸出兩次相同名字的人。

B

B1

先找出 Xmart 有賣的商品，之後用全部的商品去扣掉，就可以找出 Xmart 沒賣的商品了

$$\Pi_{\text{ProductName}}(\text{Product}) - \Pi_{\text{ProductName}}(\text{Inventory} \bowtie (\Pi_{\text{StoreID}}(\sigma_{\text{StoreName} = \text{'Xmart'}}(\text{Store}))))$$

B2

先找出有被兩間以上的商店販賣的商品，改名是為了做 join 時區分。

$I1 \leftarrow \rho(I1(ProductName \to name, StoreID \to id), Inventory)$
$I2 \leftarrow \rho(I2(ProductName \to name2, StoreID \to id2), Inventory)$
$tmp \leftarrow \Pi_{\text{name}}(I1 \bowtie_{name=name2 \wedge id \neq id2} I2)$

之後用全部有販賣的商品去扣掉前面找到的 table 就可以得到答案了。

$ans \leftarrow \Pi_{\text{ProductName}}(\text{Inventory}) - \rho(tmp(name \to ProductName), tmp)$

B3

先找單價大於 3 的商品
$\text{expensive\_product} \leftarrow \Pi_{\text{ProductName}}(\sigma_{\text{UnitPrice}>3}(Product))$

找有賣這些商品的店家 id
$\text{Target\_Inventory} \leftarrow \text{expensive\_product} \bowtie \text{Inventory}$

找那些店家的名字
$tmp \leftarrow \text{Target\_Inventroy} \bowtie Store$

改名，用來做下一行的 join 時可以區分。
$tmp2 \leftarrow \rho(tmp2(\text{StoreID} \rightarrow id2, \text{StoreName} \rightarrow \text{name2}), tmp)$

用商品名稱去做 join，之後挑選 Id 不一樣的店家
$ans \leftarrow \Pi_{\text{StoreName, name2}}(\sigma_{\text{StoreID}\neq\text{id2}}(tmp \bowtie tmp2))$

B4

找出所有 Store pair 對
$tmp \leftarrow \rho(S1(StoreID \rightarrow id, StoreName \rightarrow name), Store)$
$\quad \times \rho(S(StoreID \rightarrow id2, StoreName \rightarrow name2), Store)$

找出 id 不被 id2 包含的 record
$notsub \leftarrow (tmp \bowtie_{id=StoreID} Inventory) - (tmp \bowtie_{id2=StoreID} Inventory)$

全部減掉一定不對的
$ans \leftarrow \Pi_{name}(tmp - notsub)$

C

C1

先刪掉類別不是 laptop 的 record，之後挑 maker 這個 column。
$ans \leftarrow \Pi_{\text{maker}}(\sigma_{\text{category} = \text{'laptop'}}(\text{Computer}))$

C2

先刪掉類別不是 desktop 的 record，之後挑 maker 和 model 兩個 column。
$lm \leftarrow \Pi_{\text{maker, model}}(\sigma_{\text{category} = \text{'desktop'}}(Computer))$

用 maker 當 key，找出 model 不一樣的 record，此使我們可以知道至少製造兩種不同類型的 model 的 maker 有哪一些。
$two \leftarrow \sigma_{m1 \neq m2}(\rho(lm1(model \rightarrow m1), lm) \bowtie \rho(lm2(model \rightarrow m2), lm))$

再疊一層，確定三者 model 皆不相同，即可確定對應的maker至少做了三種model
$three \leftarrow \sigma_{m1 \neq m3 \wedge m2 \neq m3}(two \bowtie \rho(lm3(model \rightarrow m3), lm))$
$ans \leftarrow \Pi_{\text{maker}}(three)$

C3

先找出速度大於 3.2 的 desktop
$speed32 \leftarrow \sigma_{\text{speed}=3.2 \wedge \text{category} = \text{'desktop'}}(Computer \bowtie_{model=num} Model)$

找出這些 desktop 的製造者
$maker32 \leftarrow \rho(maker32(maker \rightarrow name), \Pi_{\text{maker}}(speed32))$

找製造者的電話號碼
$ans \leftarrow \Pi_{\text{name, phone}}(maker32 \bowtie Maker)$

C4

找 desktop 的製造者，和找 laptop 的製造者，之後取差集
$md \leftarrow \Pi_{\text{maker}}(\sigma_{\text{category} = \text{'desktop'}}(Computer))$
$ml \leftarrow \Pi_{\text{maker}}(\sigma_{\text{category} = \text{'laptop'}}(Computer))$
$ans \leftarrow ml - md$

C5

找所有速度大於 3.2 的設備
$fast \leftarrow \Pi_{\text{num}}(\sigma_{\text{speed}>3.2}(Model))$

用相除找有製造所有 fast 設備的人
$ans \leftarrow \Pi_{\text{maker}}(\Pi_{\text{maker, model}}(Computer)/fast))$

C6

$ans \leftarrow \Pi_{\text{model, maker}}(\sigma_{\text{category} = \text{'laptop'}}(Computer))$

C7

找server，和server的價錢
$serv \leftarrow \Pi_{\text{maker, model}}(\sigma_{\text{category} = \text{'server'}}(Computer))$
$sp \leftarrow \Pi_{\text{maker, price}}(Model \bowtie serv)$

找所有存在有record比自己大的record，找絕對不是最大值的record
$notmax \leftarrow \Pi_{m1,p1}(\rho(sp1(maker \to m1, price \to p1), sp) \bowtie_{p1<p2} \rho(sp2(maker \to m2, price \to p2), sp))$

用全部去扣掉不是最大值的record，就可以找到最大值
$ans \leftarrow \Pi_{\text{maker}}(sp - \rho(notmax(m1 \to model, p1 \to price), notmax))$

C8

找所有的 desktop，以及其價值與速度
$d \leftarrow \Pi_{\text{maker, model}}(\sigma_{\text{category='desktop'}}(Computer))$
$cpt \leftarrow \Pi_{\text{maker, model, speed, price}}(d \bowtie Model)$
改名方便 join 用
$cpt2 \leftarrow \rho(cpt2(maker \to maker2, model \to model2, speed \to sp2, price \to pri2)$

找所有存在有record比自己大的record，找絕對不是最大值的record，計算大小的方法是 cp 值
$notmax \leftarrow \Pi_{maker,model}(cpt \bowtie_{\text{speed/price<sp2/pri2}} cpt2)$

用全部去扣掉不是最大值的record，就可以找到最大值
$ans \leftarrow (\Pi_{maker,model}(cpt)) - notmax)$

D

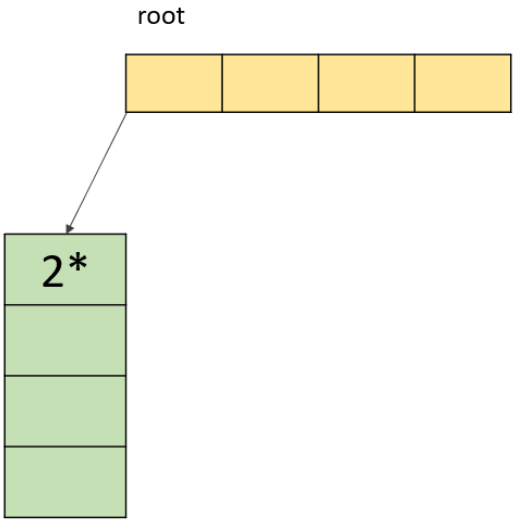
part D 皆假設沒有 redistribution

D1

step 1

在全空的時候 insert 2，樹的 order 為 2

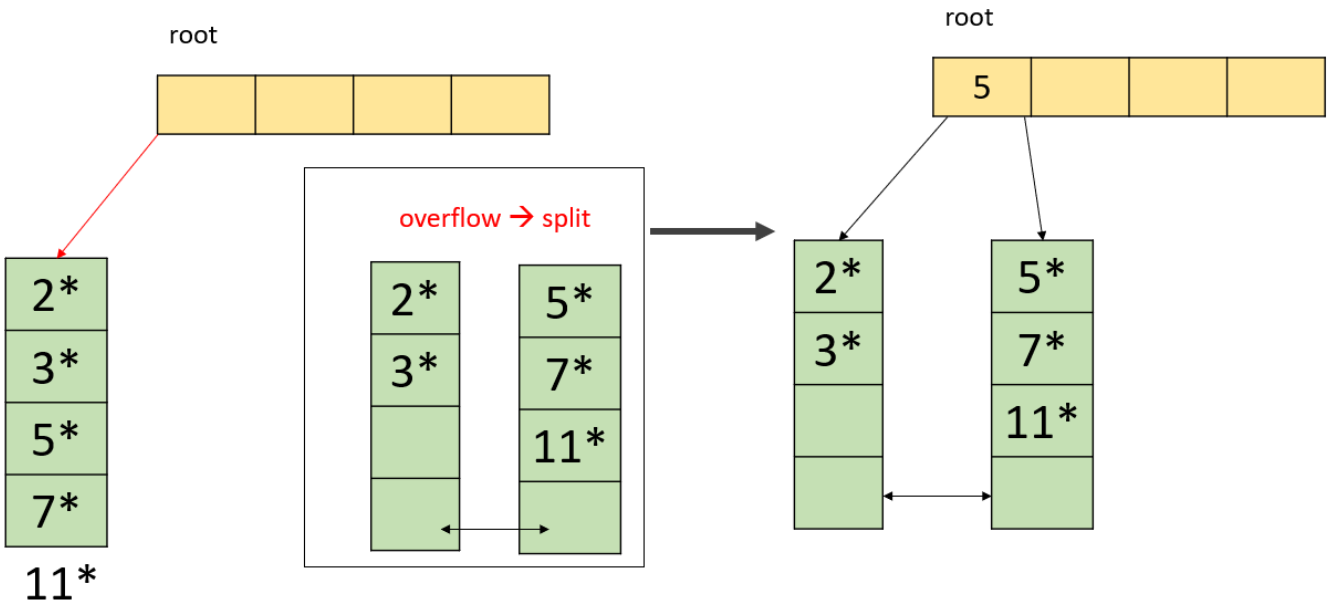
insert2



step 2

持續 INSERT 直到 11 時發生 overflow

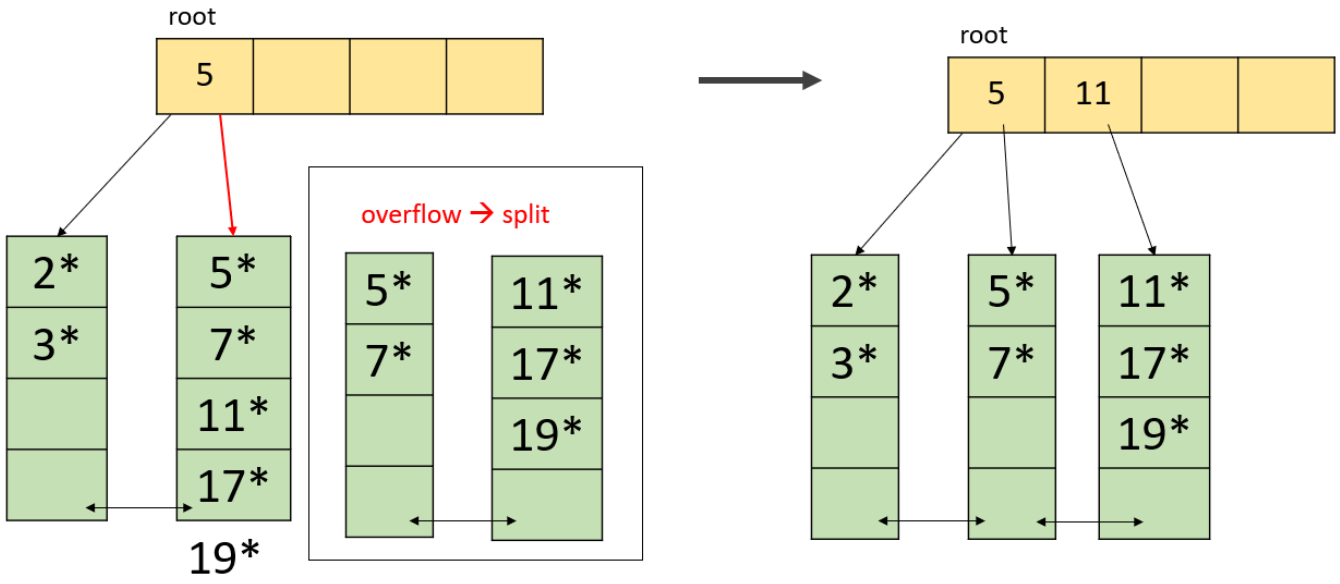
Insert 11



step 3

繼續 INSERT 直到 19 時發生 overflow

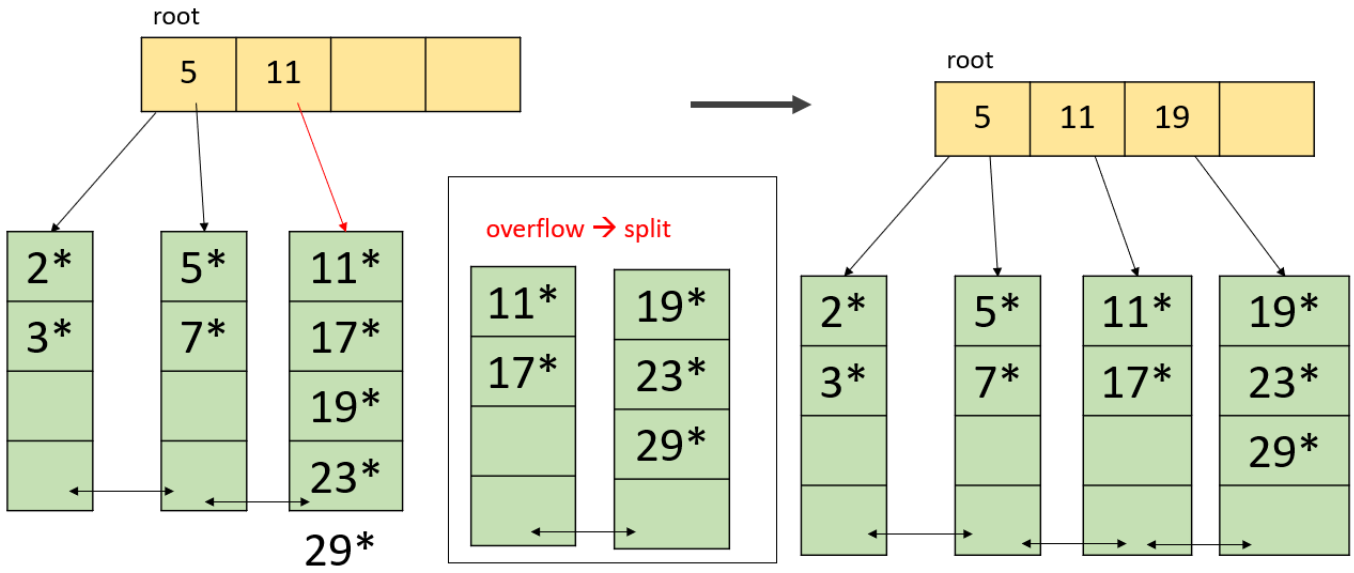
Insert 19



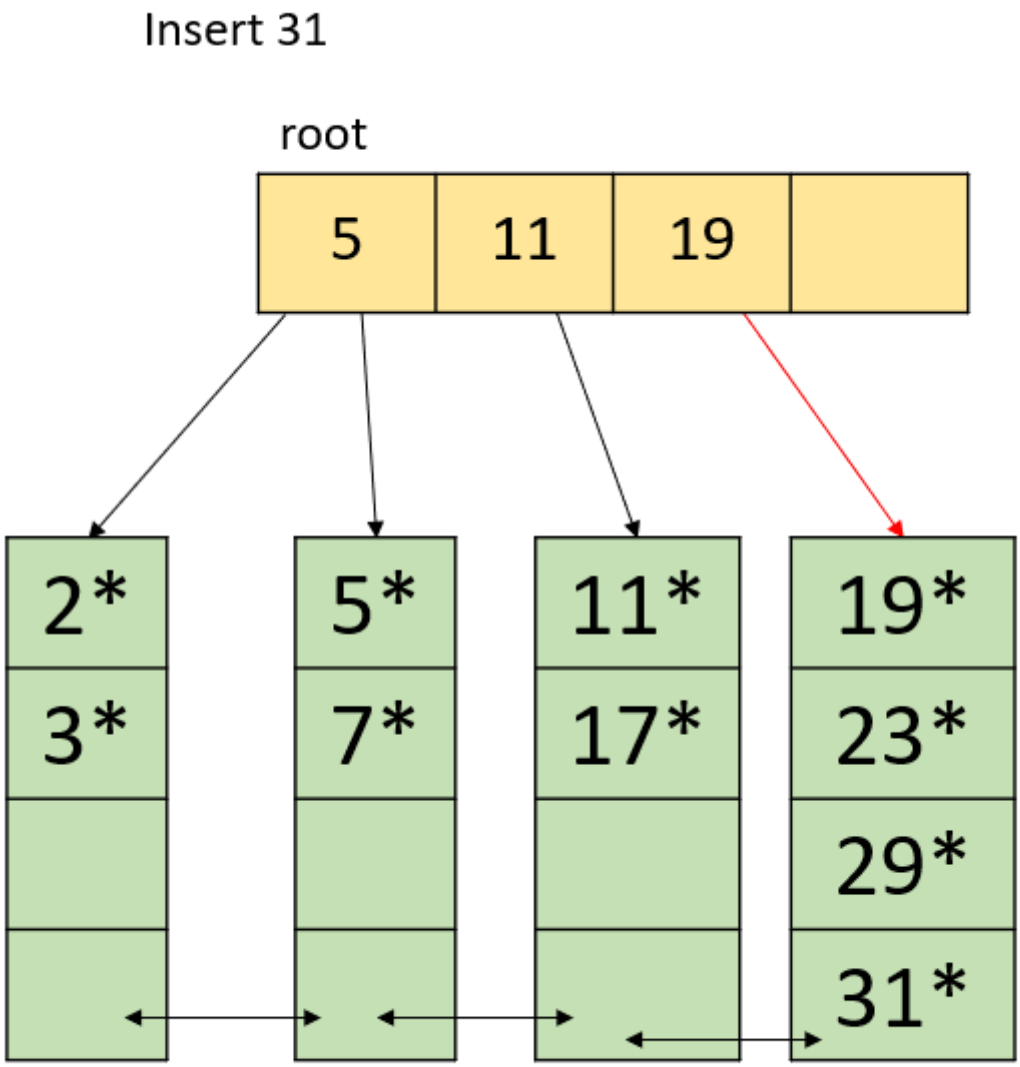
step 4

繼續 INSERT 直到 29 時發生 overflow

Insert 29



result

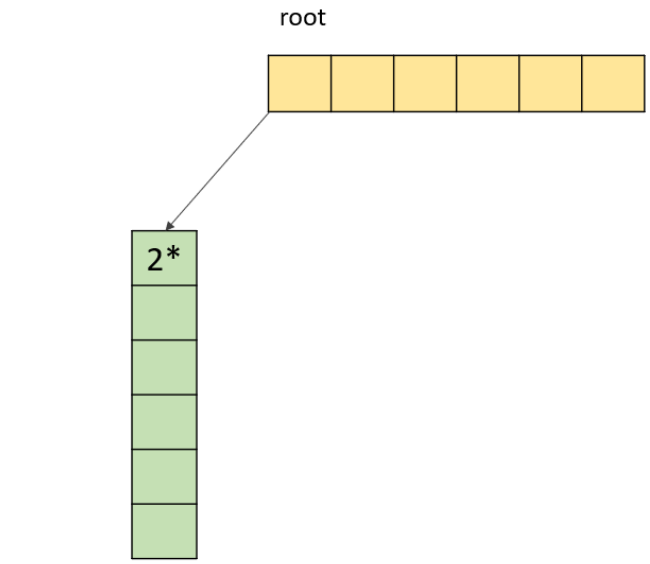


D2

step1

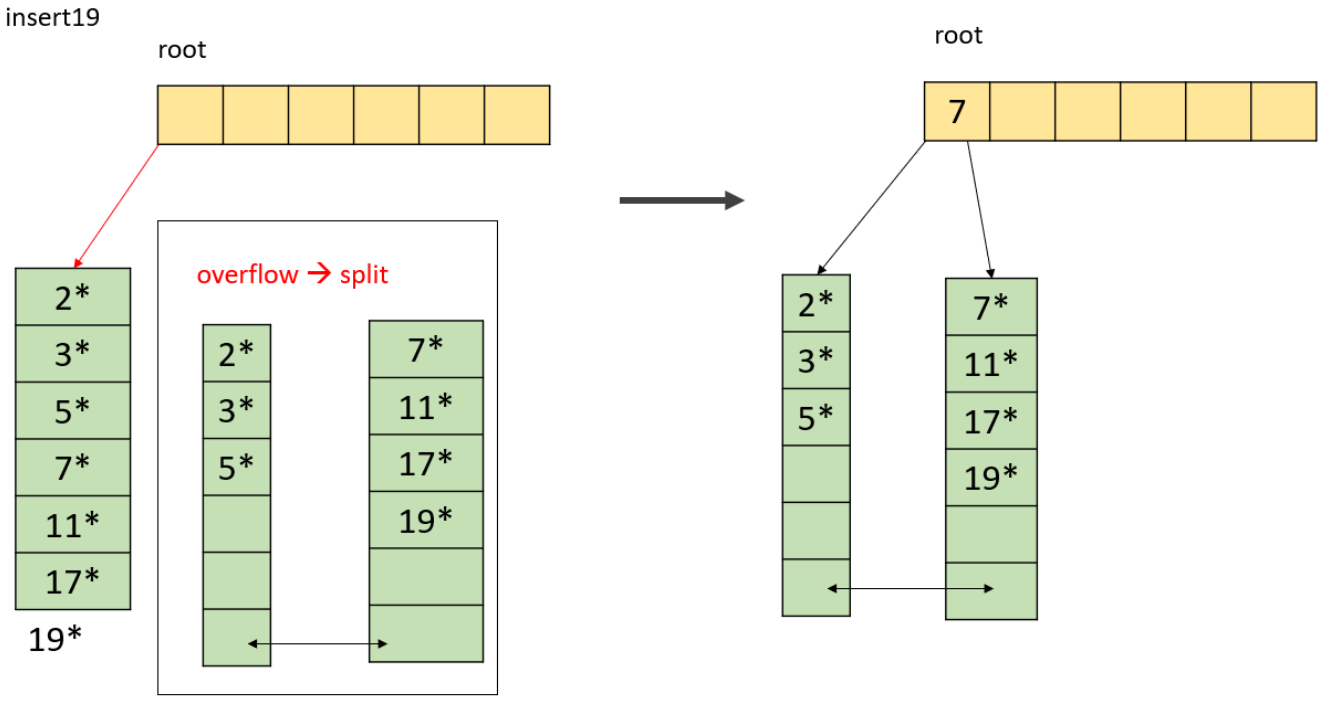
在全空的時候 insert 2，樹的 order 為 3

insert2



step2

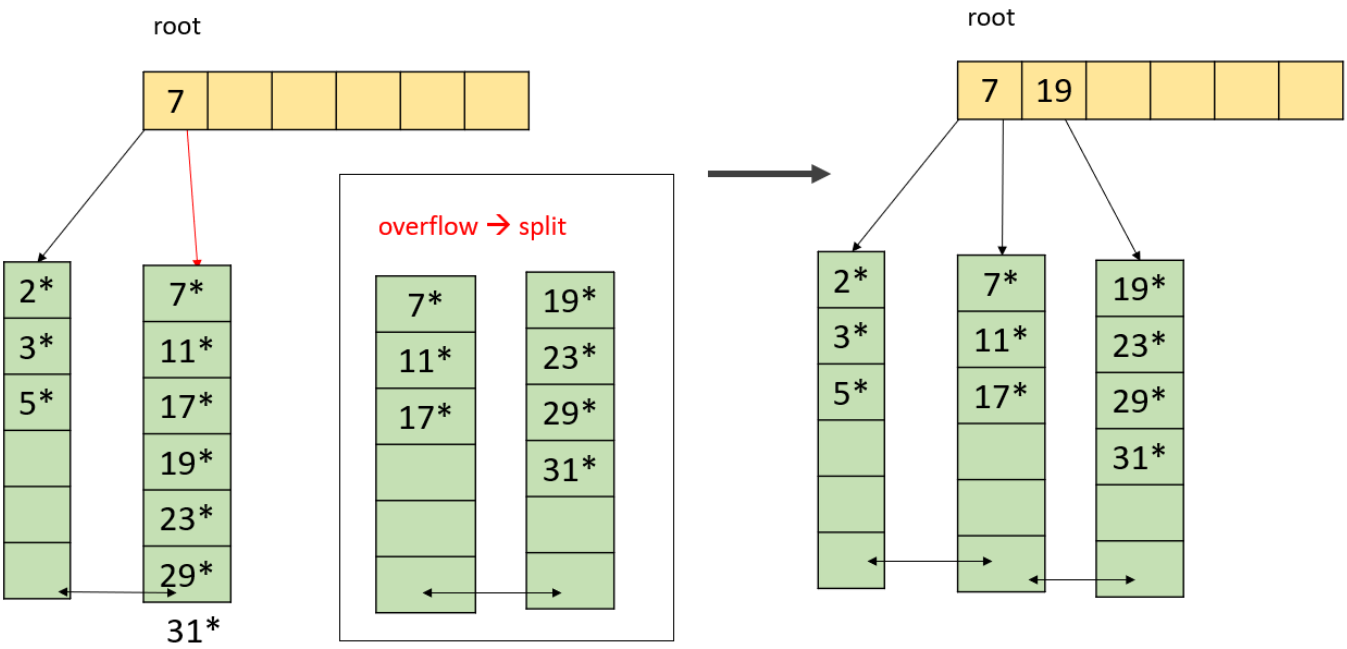
持續 INSERT 直到 19 時發生 overflow



result

繼續 INSERT 直到 31 時發生 overflow

insert31

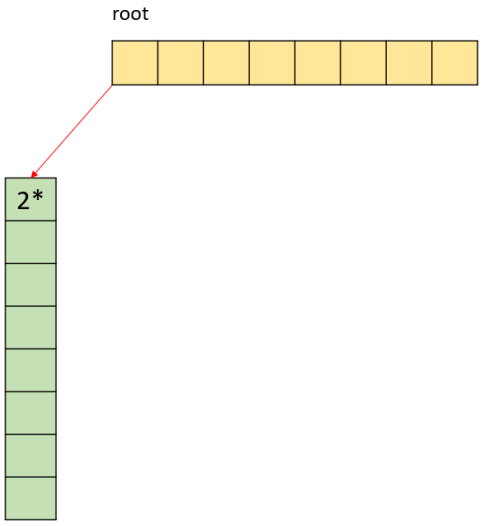


D3

step1

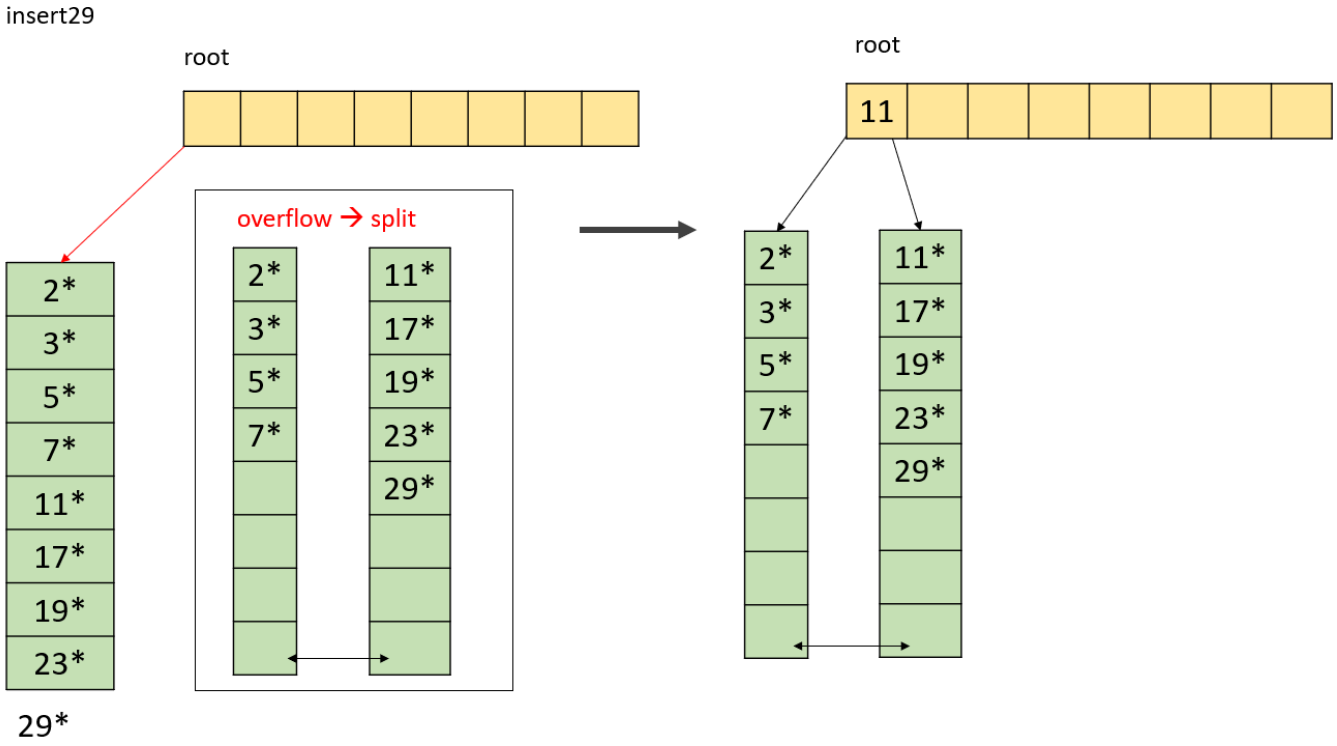
在全空的時候 insert 2，樹的 order 為 4

insert2

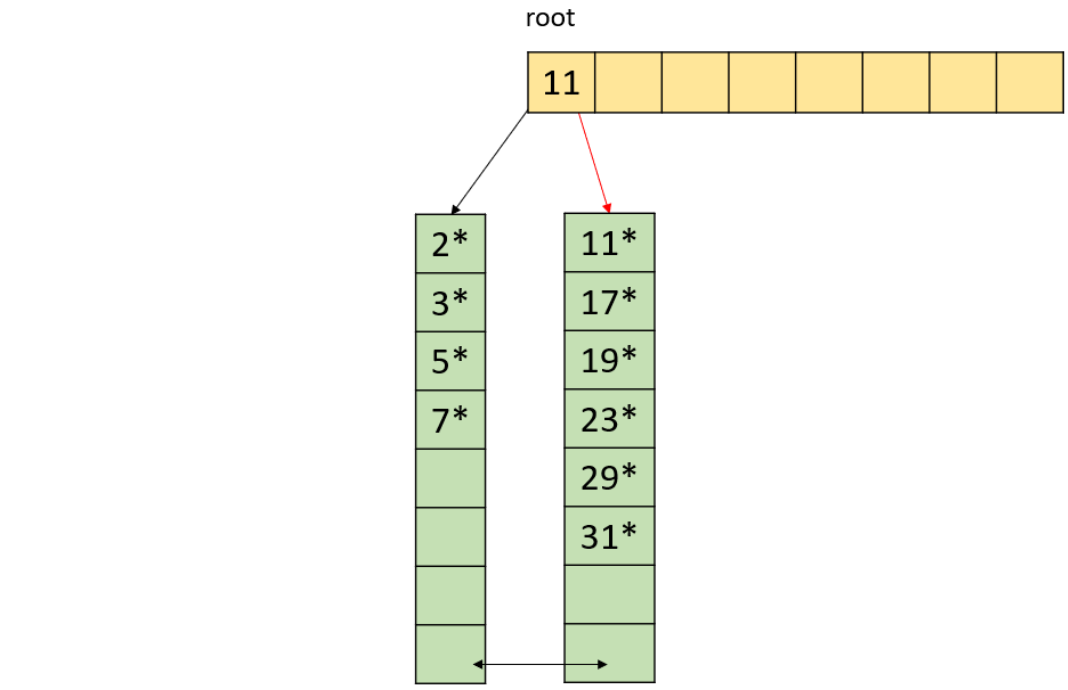


step2

持續 INSERT 直到 29 時發生 overflow



result

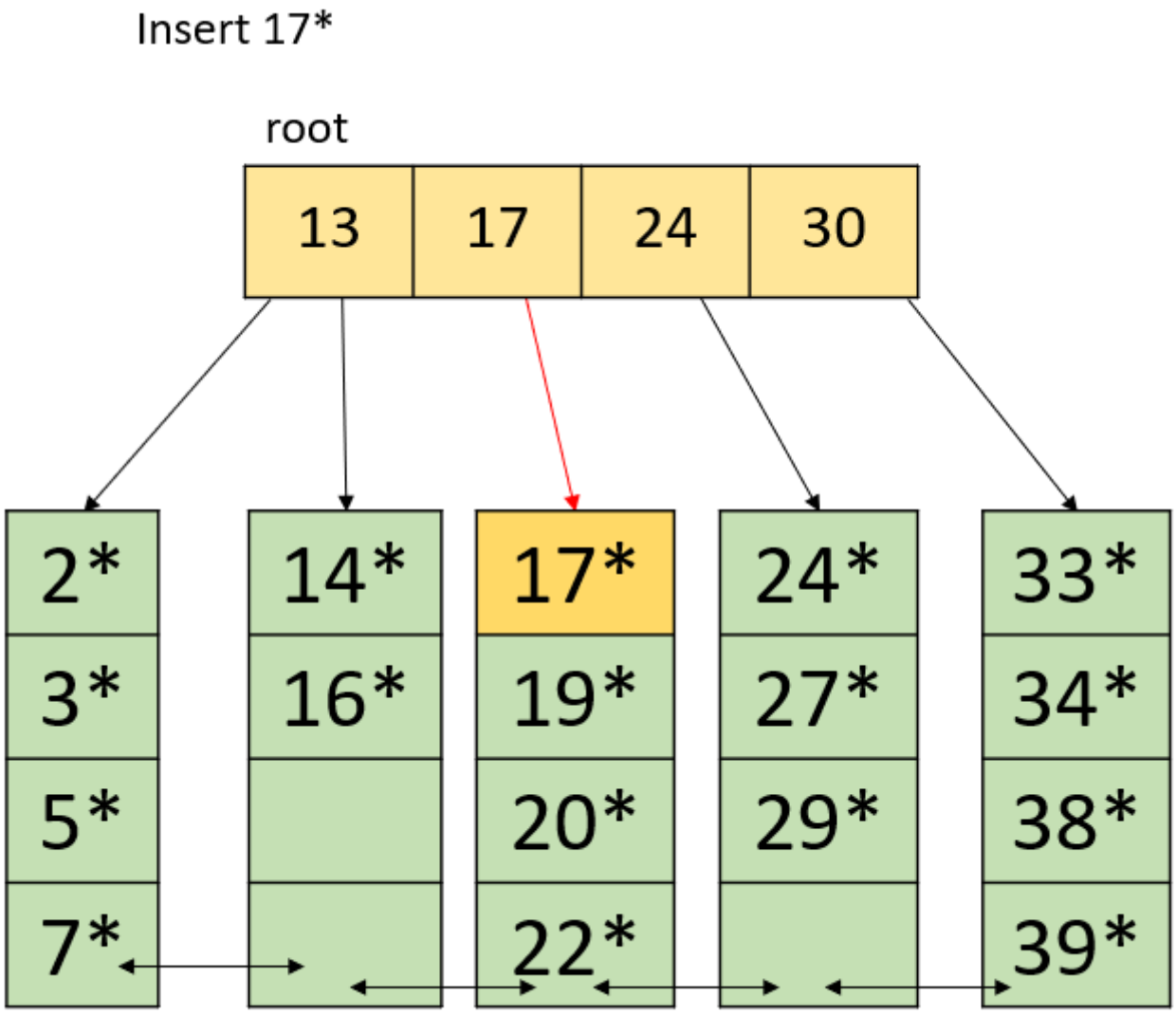


E

part E 假設在 insert 與delete 時皆有 redistribution

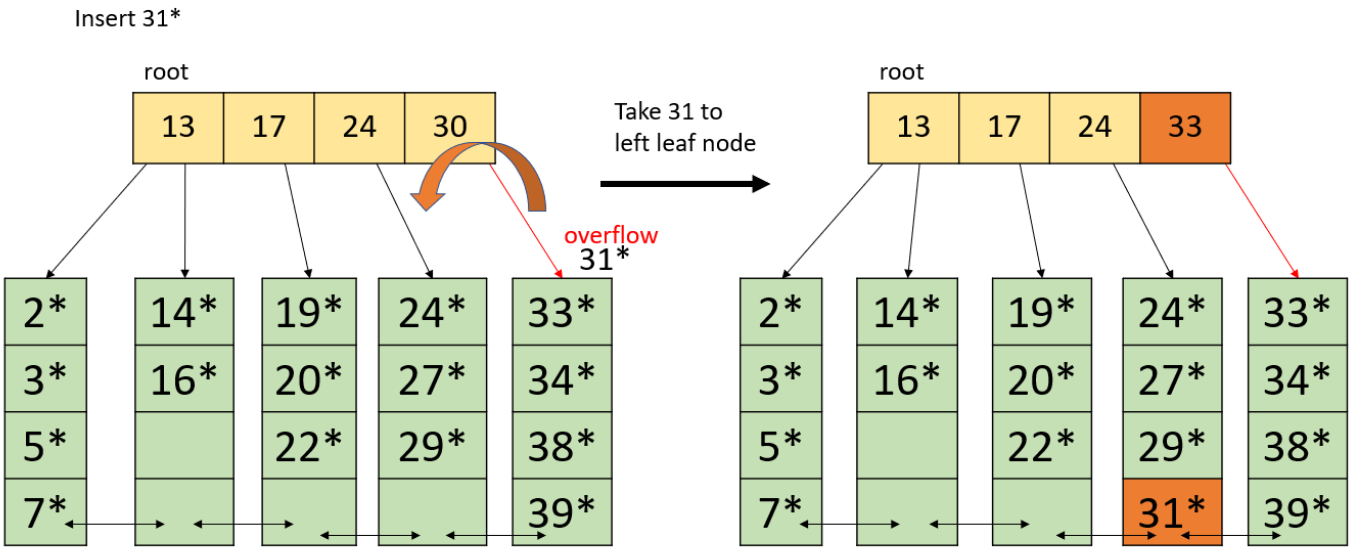
E1

用 index node 確定要放入的 leaf node 後，對leaf node 做 insert，最後確定沒有 overflow



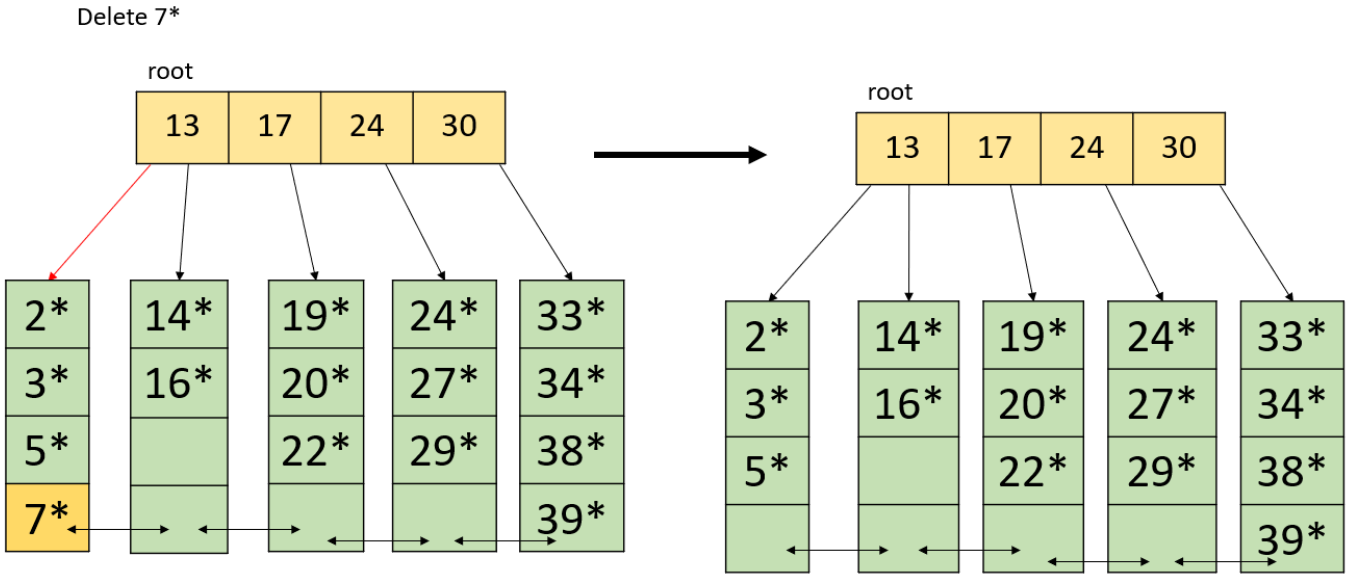
E2

insert 31 後發生 overflow，且左鄰居有空位，因此做 redistribution，把 31 放到左鄰居的 page 中



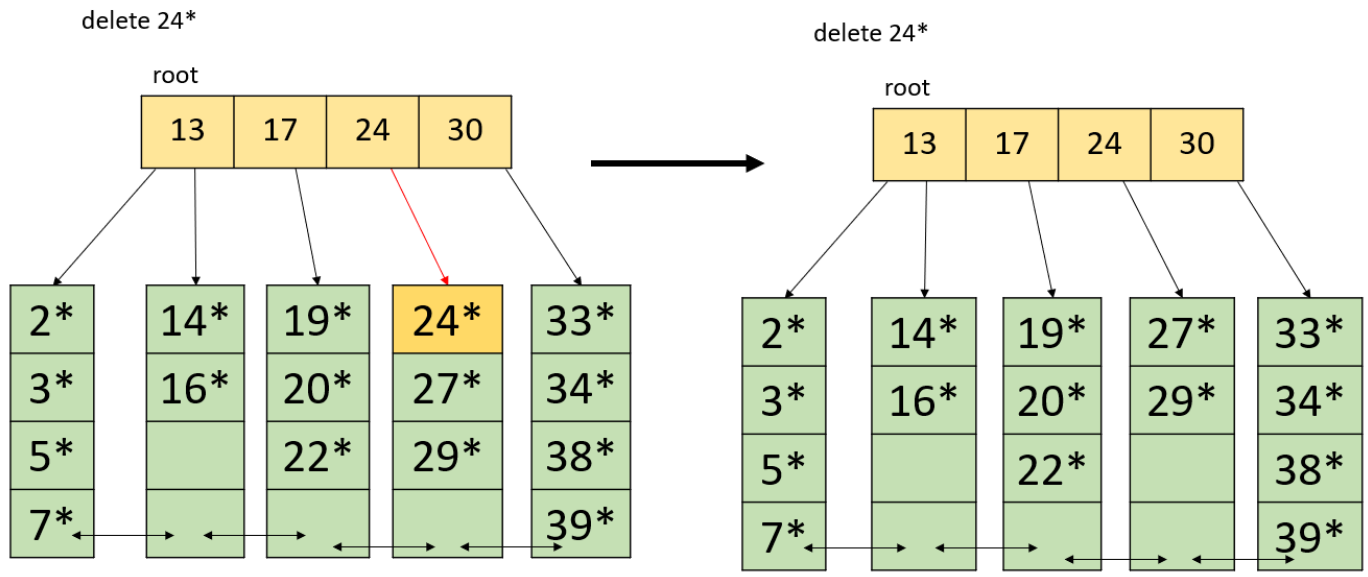
E3

找到 7* 之後把 7* 刪掉



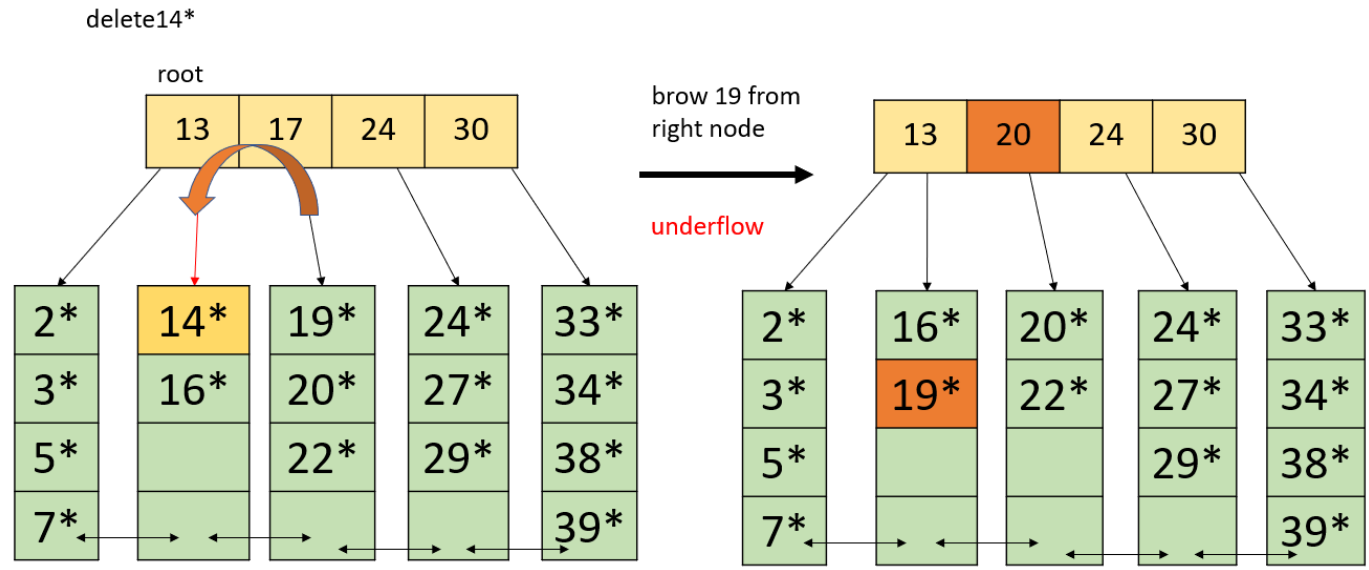
E4

找到 24* 之後把 24* 刪掉



E5

刪掉 14* 之後發生 underflow，因此做 redistribution，借右鄰居的值來避免 underflow



F

F1

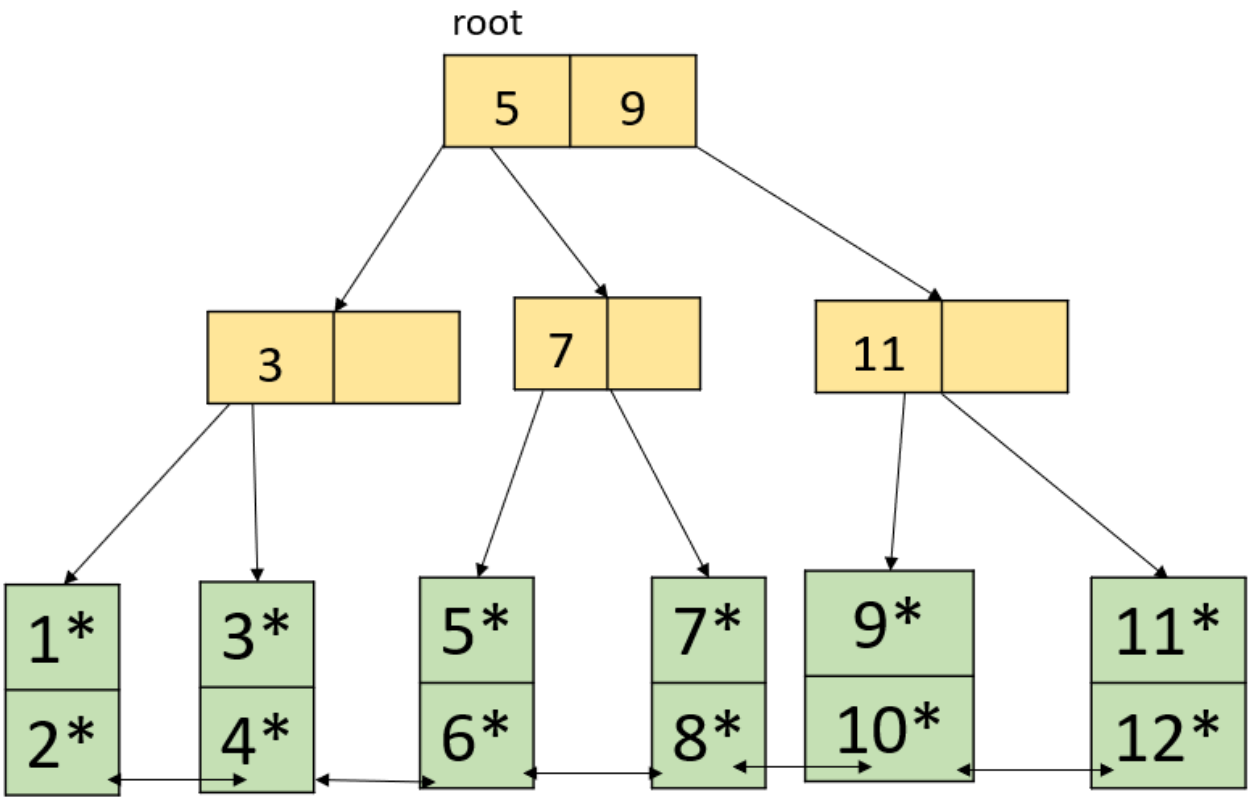
$d \times 2 \times 8 \ + \ (d \times 2 + 1) \times 4 \leq 28$
$16d + 8d + 4 \leq 28$
$d \leq 1$

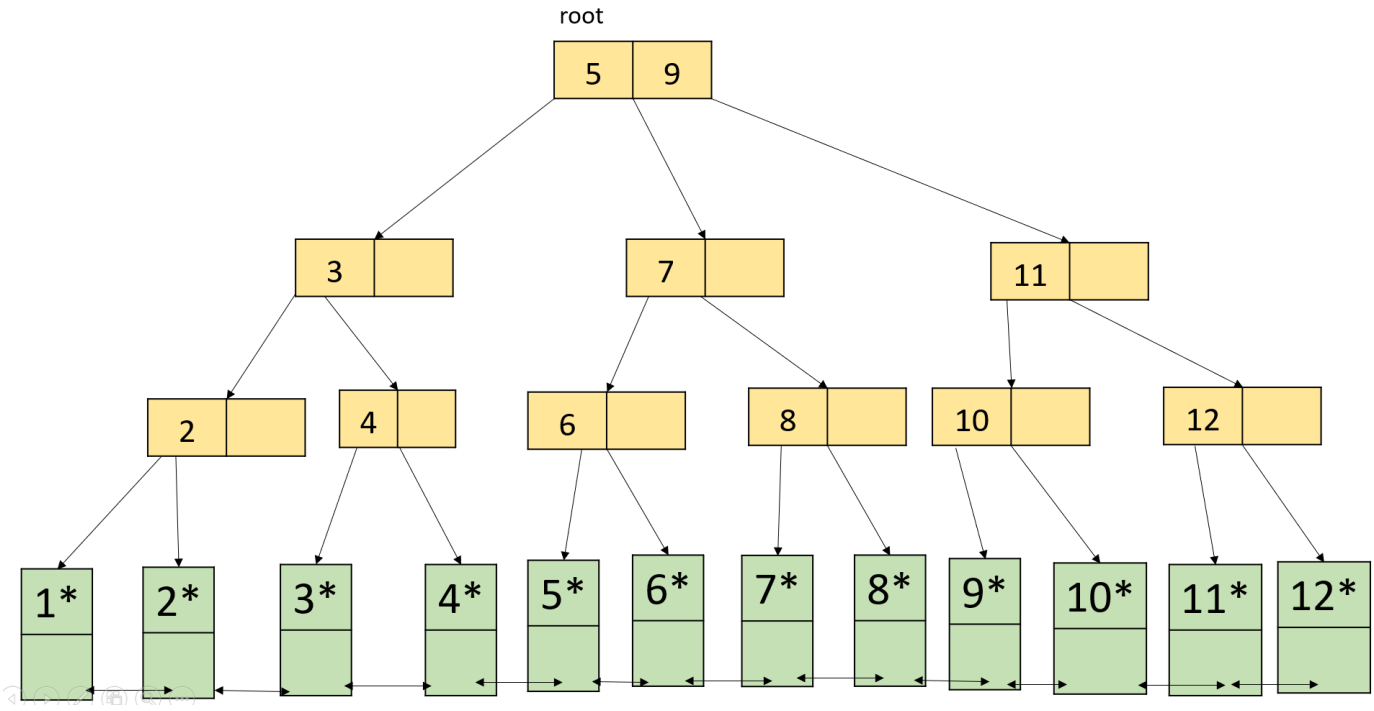
因為 d 越大時，樹高越低搜尋越快，故取 d 為最大值 1。
28 byte 的 Page 可以做出 order 為 1 的 B+ tree，且每個節點可以有 2 個 key 和 3 個 pointer。

F2

假設我們把 fill factor 設為 100%，則此 case bulk loading 生成的 B+ Tree 高度為 3
假設我們把 fill factor 設為 50%，則此 case bulk loading 生成的 B+ Tree 高度為 4

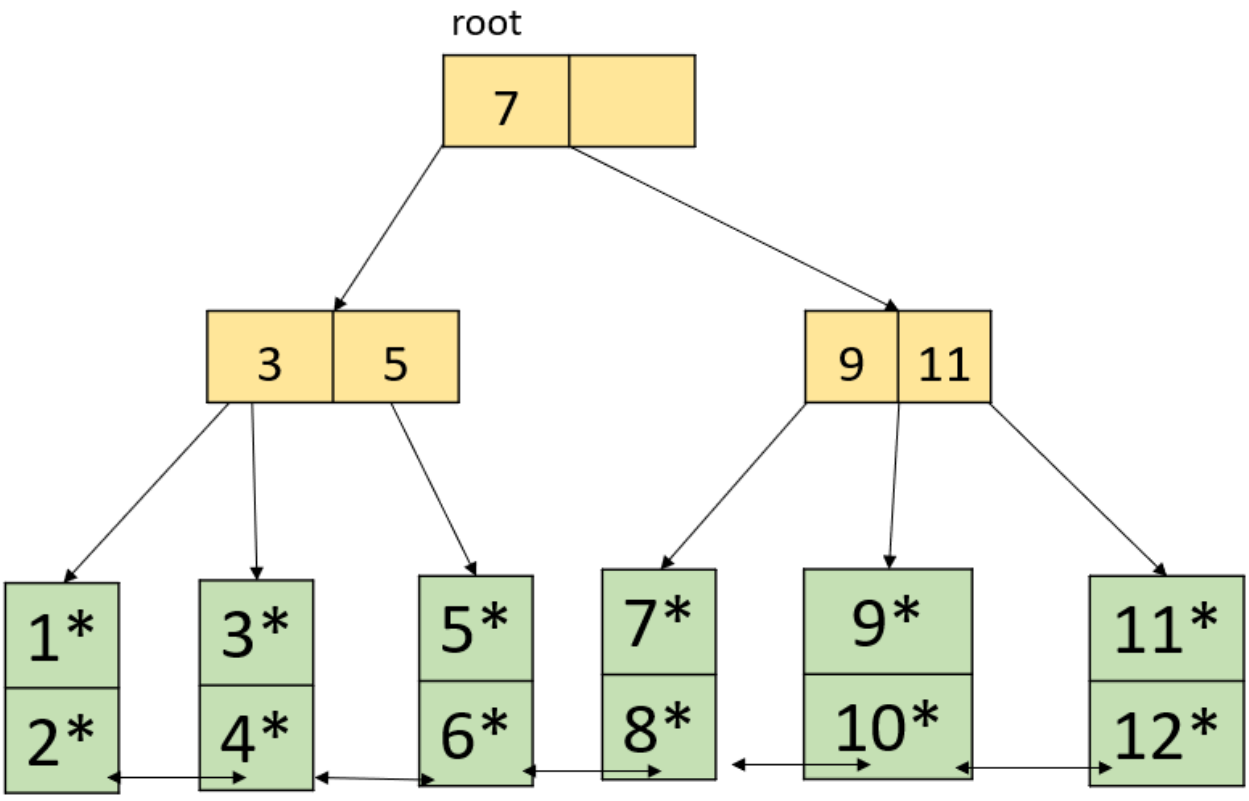
F3


root
5
9
3
7
11
1*
2*
3*
4*
5*
6*
7*
8*
9*
10*
11*
12*


root
5
9
3
7
11
2
4
6
8
10
12
1*
2*
3*
4*
5*
6*
7*
8*
9*
10*
11*
12*

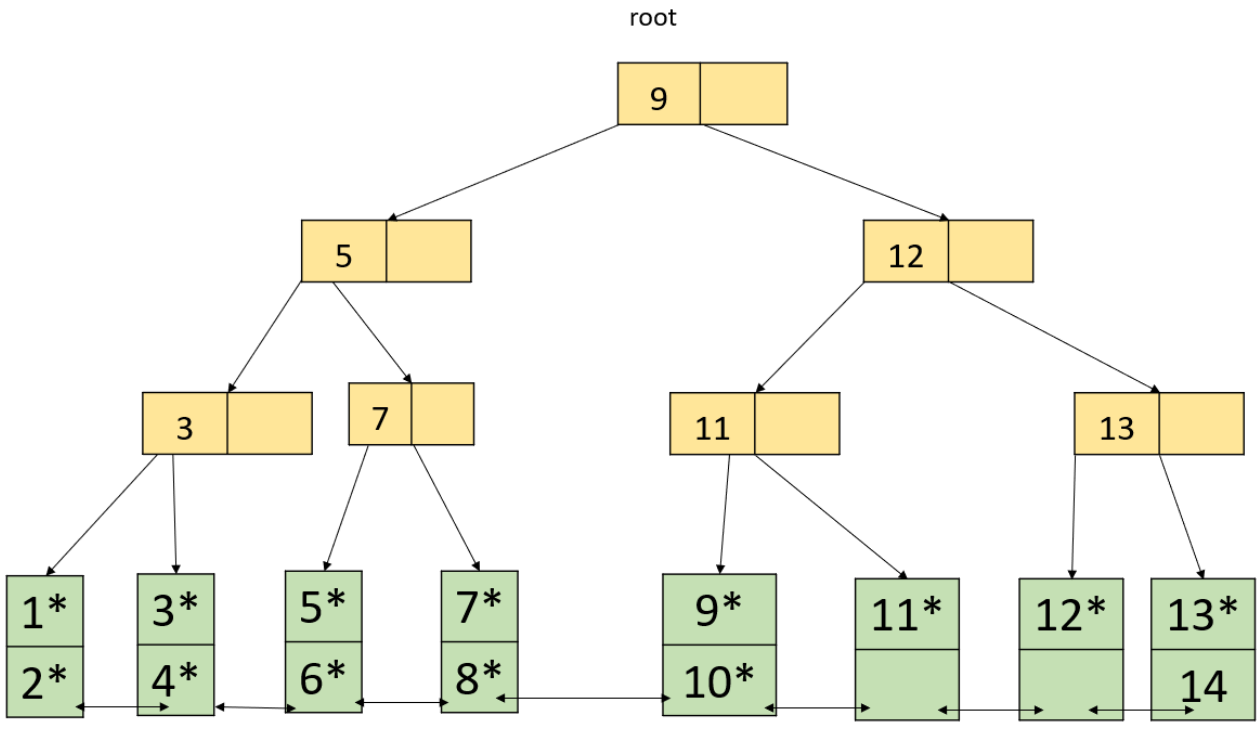
F4

NO



F5

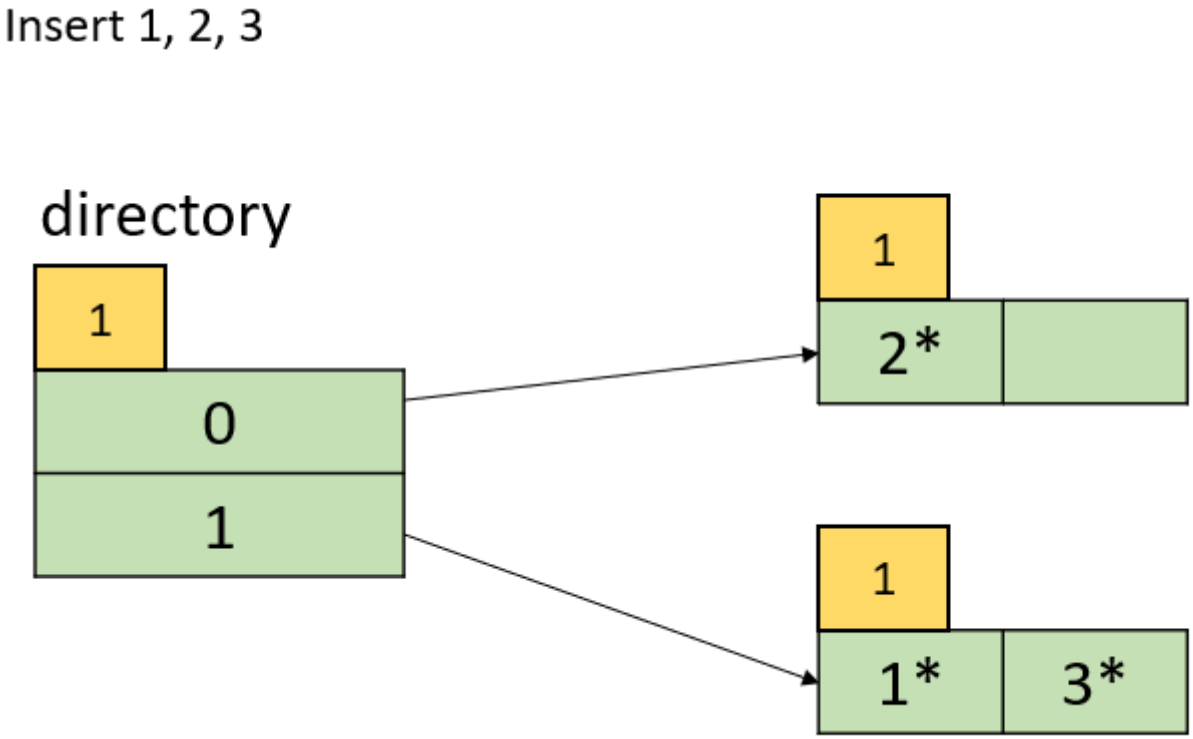
把 fill factor 設為 100%時，並假設在 insert 沒有 redistribution 的情況下，最少要加 2 個key，可以選擇任何大於 12 的兩個 key，例如 $13, 14$。



G

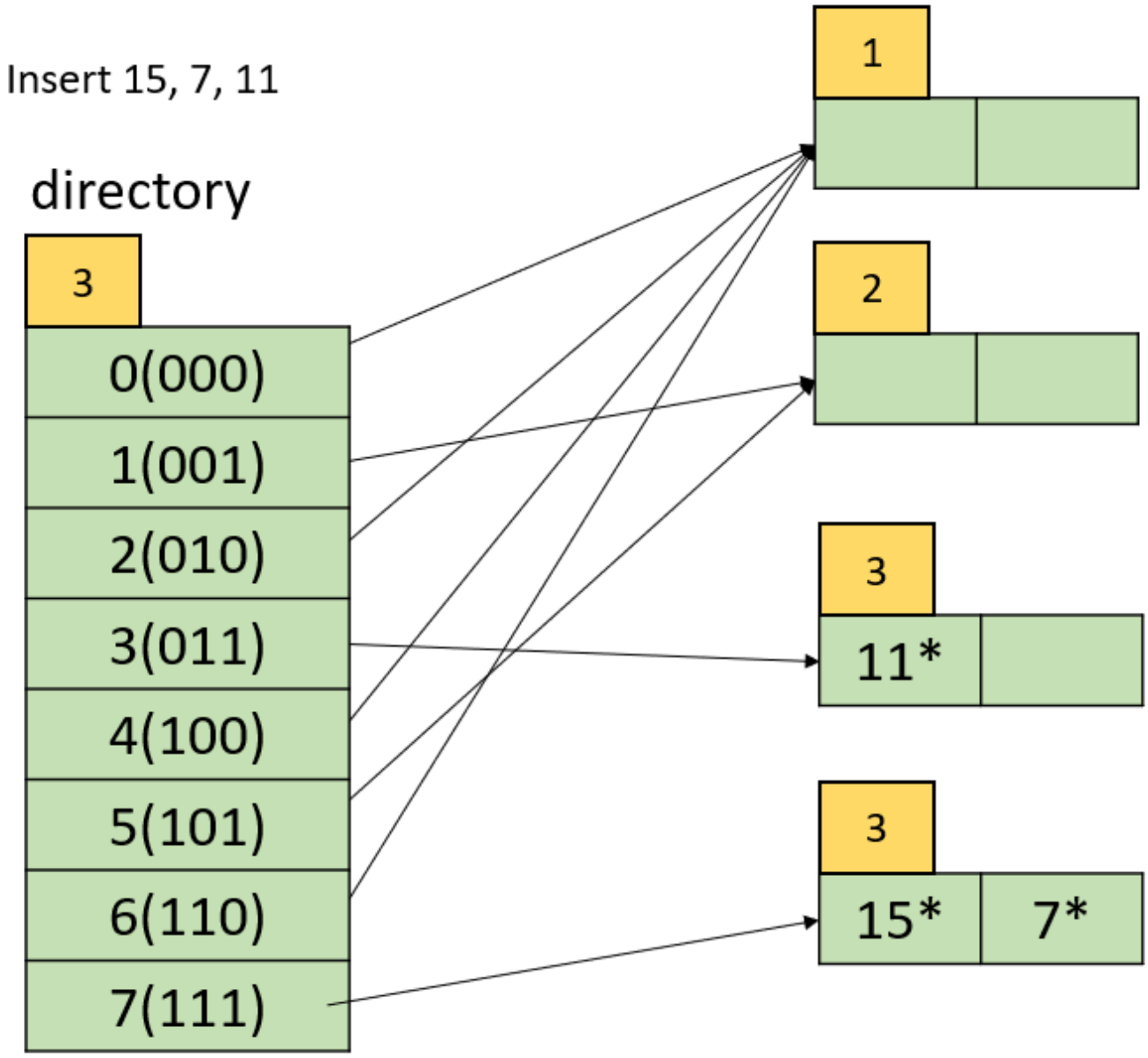
G1

在 insert 值為 $1, 2, 3$ 時，X 有最小值為 2



G2

在 insert 值為 $15, 7, 11$ 時，X 有最大值為 4



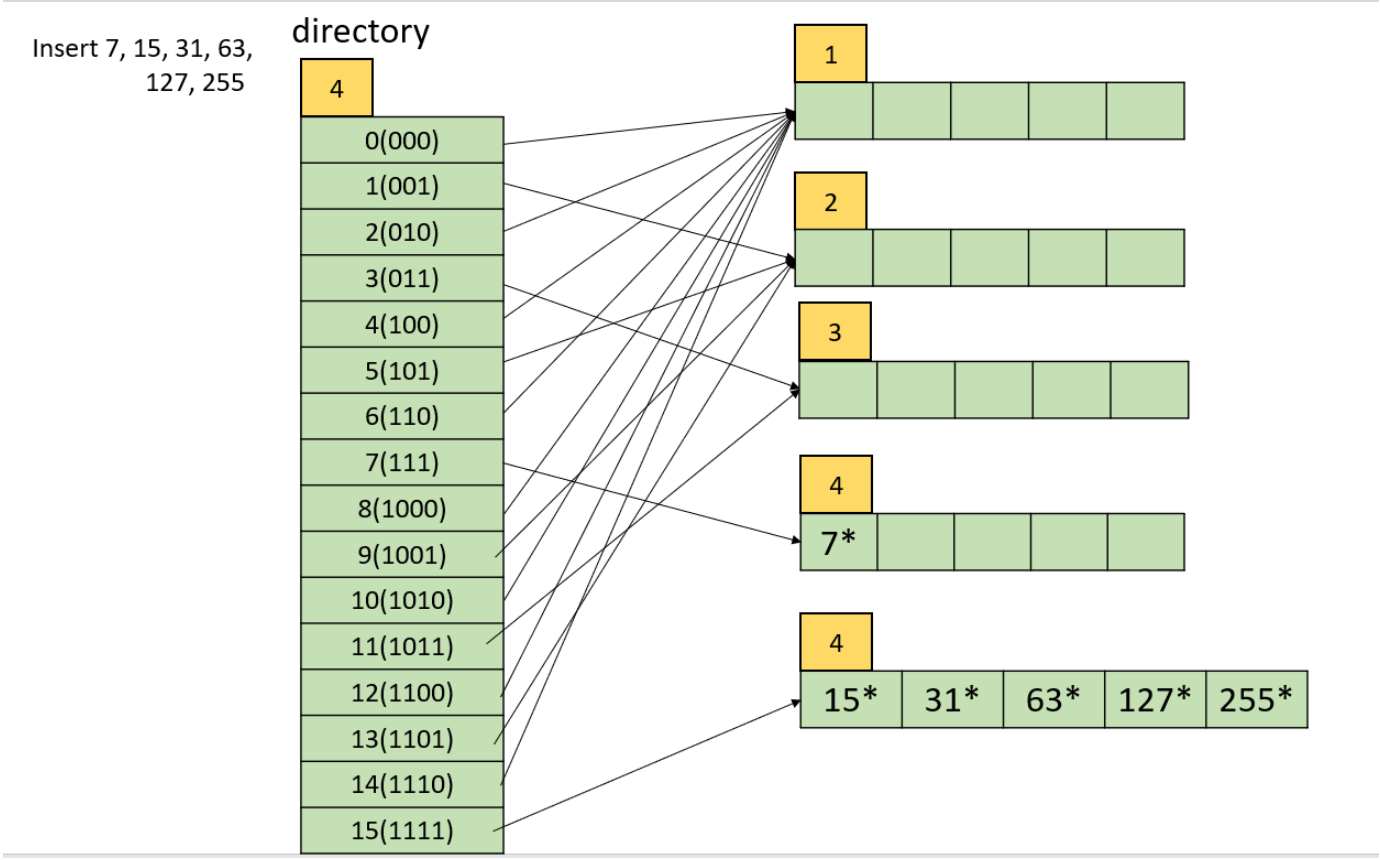
H

H1

在 insert 值為 $7, 15, 31, 63, 127, 255$ 時，bucket 有最小值為 5

H2

我們最少需要 insert 6 個數字，來使 directory 有 16 個 entries



H3

最大值出現在每個 entry 都有一個 bucket 時，因此最大 bucket 數為 16，可填入資料數的最大值為每個 bucket 皆滿，且 bucket 數為最大值時，因此最多可以存 80 筆 records