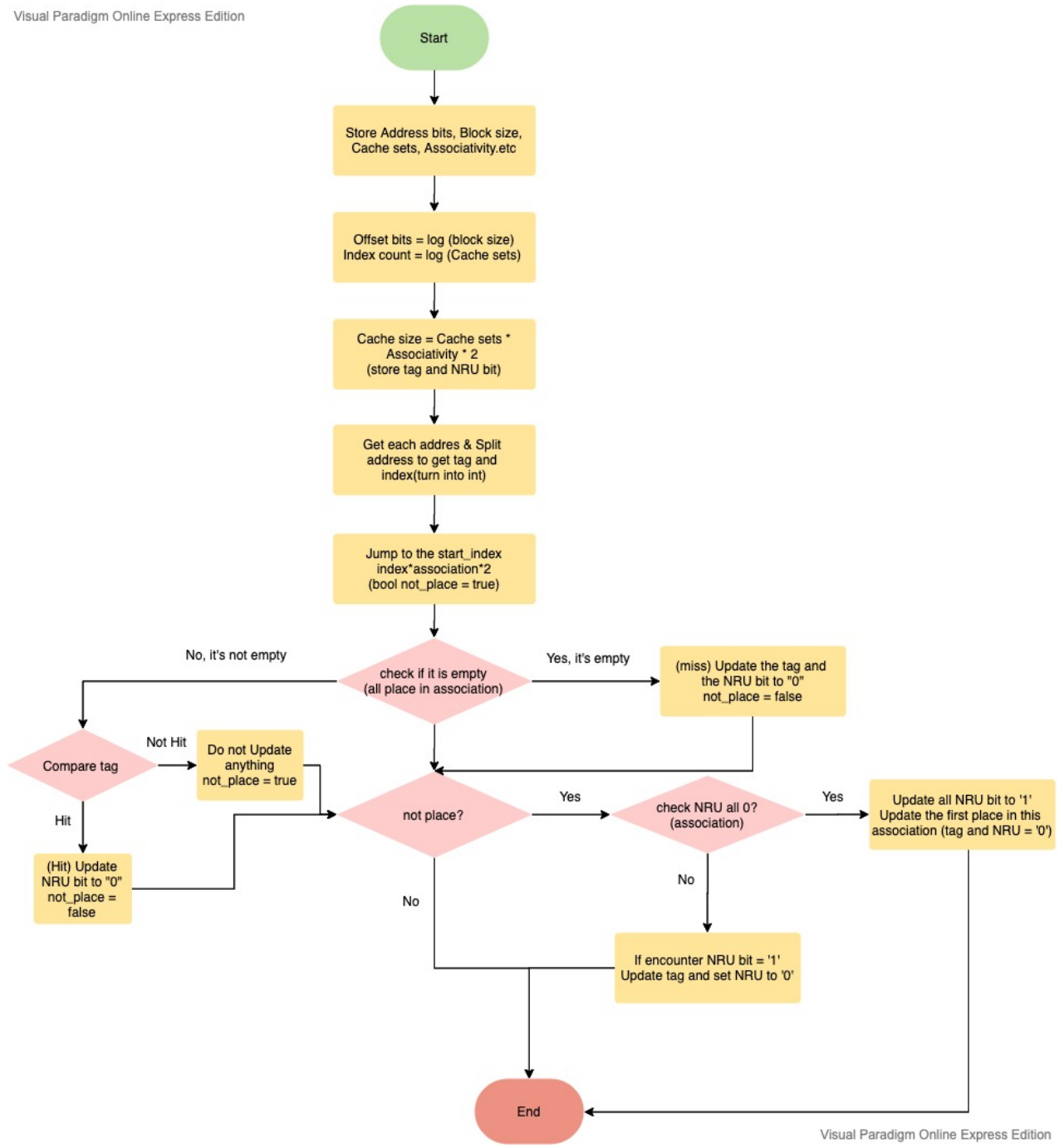
The cache behavior simulation Report

一、演算法流程圖



二、找 offset_count, index_count 跟 index_bit （output file 需要）

Address_bits: 8 （存在 cache_info[0]）

Block_size: 4 （存在 cache_info[1]） offset_bit = log2(cache_info[1])

Cache_sets: 4 （存在 cahce_info[2]） index_count = log2(cache_info[2])

Associativity: 2 （存在 cahce_info[3]）

Address index 7654 32 | 10

如果 index_count = 2 (cache set 有 4 個)，等於要求 32

tag 長度 = address_bit - offset_bit - index_bit

從 temp = address_bit -tag 長度開始，到 temp>offset_bit

每次輸出 temp-1 即可得 index_bit

三、一次讀入一個地址並切出 tag 和 index

有一串地址以後，譬如 00010000，每個地址需要 2 格來分別存: tag, NRU bit

以上一點的例子來說，需要一個 vector 4*2*2 格

(set * associativity * 2 格拿來存，2 是固定不變的 for tag and NRU bit)

int cache_size = cache_info[2]*cache_info[3]*2;

vector<string> whole_cache(cache_size, "1"); // set cache_size and initiate all data to '1'

從 all_address 取出地址以後（第一行不是地址要跳過）：

1. 用 line.substr(temp, index_count) 取 index

   字串從 temp = 4 索引開始，取 index_count=2 個字元，如下：

   0123 | 45 | 67

   0001 | 00 | 00

2. 用 line.substr(0, tag_length) 取 tag

   int tag_length = temp; // address_length-offset_bit-index_count

四、確定 miss 或 hit

先跳到 set 的起始位置，起始+2 = 下一格存tag，起始+1 = NRU bit

只會檢查同個 set 裡

（會有個 bool 變數 not_place 來紀錄，如果這個地址的資料沒被存過 = true）

if （tag 一樣）:

hit，並寫入 output file

NRU bit 設為 0

not_place = false 代表 hit

跳出迴圈，不用繼續檢查下一個位置

else if 為空:

miss，並寫入 output file

儲存 tag

NRU bit 設為 0

not_place = false 代表被存好了

跳出迴圈，不用繼續檢查下一個位置

最後檢查 not_place

如果 not_place 為 true 代表還沒被存

這時候要從同個 set 的起始位置+1 開始看 NRU bit

if 有 NRU bit 為 1 的：

選他當替換的，把 tag 存進去，並把 NRU bit 設為 0

else:

NRU bit 全部改為 1，再選 set 中最靠近 head 的替換

五、怎麼找哪幾個 bit 當 index miss 次數比較少？

找到 index bit 之後統計次數，每個 set 都有用到，而且次數分配比較平均的

miss 次數應該會比較少