REPORT OF CS3230 PJ1 TASKA

Simulation of Cache Replacement Policy

I. algorithm design

- Since my comment are sufficient for any coder to understand my program, I'll only note the important points here.

A. FIFO policy

1. data-structure
   Queue is a natural implementation of FIFO sequence.
2. algorithm
   Maintain a queue as cache. If a cache miss happens, the new entry is insert to the end of the queue. And choose the front as the victim if the queue is larger than allowed.

B. LIFO policy

1. data-structure
   Stack is a natural implementation of LIFO sequence.
2. algorithm
   Maintain a stack as cache. When a cache miss happens, choose the top as the victim if the stack is as high as allowed. And the new entry is always insert to the top of the stack.

C. LRU policy

1. data-structure
   List is a good implementation because it's easy to modify the sequence.
2. algorithm
   The farer from the front an entry is, the more recently it's used. Thus, the less likely it's the victim.
   If a cache hit happens, the recently used entry is moved to the end. Otherwise, choose the first one as victim, and insert the new one to the end.

D. OPT policy

1. data-structure
   Use set to implement cache because the choice of victim has nothing to do with the order of cache entries.
2. algorithm
   If a cache miss happens, insert the new entry into the set. If the set is bigger than allowed, choose the one that is least soon to be used in the future as the victim.
   As for the detail, first we can make a copy of the cache as the candidates for the victim. While scanning the future usage, pardon the one that's used. Do that until there's only one candidate so that we can determine the victim.

II. time complexity analysis of OPT

A. cold down loop

$$\Theta(n)$$

B. body loop

1. inner loop

   In the worst case, the number of candidates never reaches 1.

   $$\mathrm{O}(m-i)$$

2. outer loop

   $$\Theta(m)$$

3. sum up

   $$\mathrm{O}(m^2)$$

C. sum up

Usually $\mathrm{O}(m^2)$ dominates, so that's the final time complexity in all.