

LRU CACHE Implementation

Bugs

- None Detected

Datastructures

- `class Node` : Class which specifies the node in the linked list for the LRU Cache
- `class Cache_Linked_List` :
 - Contains the pointer references for the head and tail `Node` . This itself can be used as the LRU cache but for each page check, It will have to traverse the complete linked list each time. Thereby to optimise a hashmap is used.
 - Contains a hashmap `map<int, *Node> page_indexes` : which helps check if the node is present in the linked list or not. This adds for the optimisation for the cache to make it faster. Reduces list traversal from $O(n)$ to $O(1)$ by directly identifying the node for the `page_id`
- `struct lis_input` : To store the values of `page_indexes` from each line in this `.lis` file provided.
- `class Arc_Window` : The object which will maintain the Arc Cache. It will contain 4 `Cache_Linked_List` objects which are based on two types : *recency*, *frequency* . Both the recency and frequency cache have two caches each. The Arc window consists of the top of the recency and frequency cache.

Process Flow LRU

- The LRU cache (`Cache_Linked_List`) consists of a doubly linked-list which consists of `Node` references for the head and tail. It also consists of hashmap (`map<int, *Node> page_indexes`) which contains the reference to the node according to the page id.
- `.lis` file is read and for each line in the file the LRU cache is accessed for the `page_ids` pertaining to each line.
 - If there is a hit in the cache the page id than the item is removed linked list and placed at the head of the list.
 - If there is a miss we add the item to the linked-list and also to the hashmap. In this case if the cache is full it removes the last item from the list and the same item from the linked list.
- When the file is completely processed from the cache the stats about the cache are printed.

Caching Results.

File Name	Cache Size	LRU Hit Ratio	LRU Hit %	Arc Hit Ratio	Arc Hit %
P6.lis	1024	0.007082	0.7082%	0.00839	0.8390%
P6.lis	2048	0.008593	0.8593%	0.01517	1.5175%
P6.lis	4096	0.010936	1.0936%	0.03173	3.1726%
P6.lis	8192	0.012641	1.2641%	0.05942	5.9415%
P6.lis	16384	0.016933	1.6933%	0.13011	13.0108%

File Name	Cache Size	LRU Hit Ratio	LRU Hit %	Arc Hit Ratio	Arc Hit %
OLTP.lis	1024	0.332185	33.2185%	0.39174	39.1738%
OLTP.lis	2048	0.427774	42.7774%	0.46349	46.3491%
OLTP.lis	4096	0.512405	51.2405%	0.53256	53.2563%
OLTP.lis	8192	0.588611	58.8611%	0.59617	59.6171%
OLTP.lis	16384	0.654246	65.4246%	0.66362	66.3622%
P3.lis	1024	0.010493	1.0493%	0.01129	1.1290%
P3.lis	2048	0.011516	1.1516%	0.01515	1.5154%
P3.lis	4096	0.013188	1.3188%	0.02332	2.3323%
P3.lis	8192	0.016204	1.6204%	0.04020	4.0198%
P3.lis	16384	0.020739	2.0739%	0.07003	7.0032%
P4.lis	1024	0.026851	2.6851%	0.02688	2.6880%
P4.lis	2048	0.029639	2.9639%	0.02977	2.9767%
P4.lis	4096	0.033160	3.3160%	0.03496	3.4965%
P4.lis	8192	0.036504	3.6504%	0.04165	4.1652%
P4.lis	16384	0.040738	4.0738%	0.05761	5.7610%

Contributions

- Discussion of Datastructure and optimisation with William in class.

References

1. [Erasing a key in Map Datatype](#)
2. [Searching Values in Map Datatype](#)