ENGG1330 Computer Programming I (22-23 Sem 2)
Assignment 1

Least Recently Used (LRU) Cache

Due date: **noon, 17-MAR-2023 (Friday)**
Late submission: 20% deduction per day

Coverage: Only Python libraries/features covered in this course up to and including Array (List). **Using other libraries, non-built-in functions, and features** (e.g., dictionary, set) **is not allowed** and you will be given zero marks. If you are unsure, feel free to contact us for clarification.

Grading: Evaluated against a set of private test cases on VPLs. While programming style is not graded, you are highly recommended to use function to factor your program and write clear and tidy codes, e.g., appropriate comments, concise and clear naming, appropriate spacing.

Introduction

You are having a technical interview for a software engineer internship at EEE Limited. You are asked to implement a least recently used (LRU) cache in Python. The interviewers know that you may be unfamiliar with LRU cache, so they also give you a description of LRU cache as follows.

A cache in programming is used to store data so that future requests for the data can be served faster. A data item in cache is stored as a *key-value* pair [*key*, *value*], where *key* is a unique identifier that identifies the data item and *value* is the data item to be stored. For example, after computing total marks of students, a cache can be used to store the total marks by using student ID as the key and his/her total mark as the value. As a result, a request for a student's total mark can be directly retrieved from the cache instead of re-computing it to reduce the computational time. However, since the cache also occupies storage/memory, its capacity is limited. When the cache is full, an existing item must be swapped out when a new item needs to be stored.

A LRU cache is a special kind of cache that defines the way of evicting item from the cache when it is full. Its policy is to evict the least recently used item. For example, a cache having capacity of 4 and is currently storing [[1, '101'], [2, '102'], [3, '103'], [4, '104']], where the 1st (leftmost) item [1, '101'] is the least recently used and the last (rightmost) item [4, '104'] is the most recently used. When the item [2, '102'] is accessed, the cache becomes [[1, '101'], [3, '103'], [4, '104'], [2, '102']] as the item [2, '102'] becomes the most recently used. When another item [10, '110'] is pushed into the cache, the cache cannot hold all of the items and must evict the least recently used item which is [1, '101'] and the cache becomes [[3, '103'], [4, '104'], [2, '102'], [10, '110']].

Level 1 (25%)

Your program will receive the capacity of the cache, a list of keys and a list of values as the inputs. You are then required to combine the keys and values to initialize a cache as a list of key-value pairs as follows and each key-value pair is stored as a sub-list of length 2.

$$\left[[K1, V1], [K2, V2], \ldots\ [Kn, Vn]\right], where\ n\ is\ the\ number\ of\ data\ items$$

Requirements:

1. Keys must be stored as integers.
2. Values must be stored as strings.
3. If the numbers of keys and values are different, prints "Warning: number of keys and values are not the same" and outputs an empty cache.
4. If the cache capacity is larger than or equal to the number of data items, stores all the data items in the cache. The order of data items stored in the cache must be the same as the order of the input key-value pairs.
5. If the cache capacity is smaller than the number of data items, keeps only those that are near the end of the inputs (right) up to the capacity. For example, if capacity is 3, keys = [1,2,3,4,5] and values = [-1,-2,-3,-4,-5], cache = [[3, '-3'], [4, '-4'], [5, '-5']].

Inputs:

1. Capacity of the cache C (integer). C = -1 means that the cache has unlimited capacity.
2. A string of integral keys $K1$, $K2$, …, K_{Nk} separated by "," where N_k is number of keys.
3. A string of values $V1$, $V2$, …, V_{Nv} separated by "," where N_v is number of values.

Outputs:

1. Warning message if $N_k \neq N_v$.
2. Cache as a list with format $\left[[K1, V1], [K2, V2], \ldots\ [Kn, Vn]\right]$.

Assumptions:

- $1 \leq C \leq 1000\ or\ C = -1$
- $1 \leq N_k, N_v \leq 1000$
- Input keys can be converted to integers
- Input keys are unique, no duplicates
- Input values are non-null string

Hints:

- You can use *"A_STRING".split(",")* to split a string separated by a comma "," into a list of strings, e.g., *"1,2,3".split(",")* → ["1", "2", "3"]
  https://docs.python.org/3/library/stdtypes.html#str.split

Examples:

Case	Sample input	Sample output
1	-1 1,2,3,4,5,6,7,8,9,10 101,102,103,104,105,106,107,108,109,110	[[1, '101'], [2, '102'], [3, '103'], [4, '104'], [5, '105'], [6, '106'], [7, '107'], [8, '108'], [9, '109'], [10, '110']]

2	3 1,2,3,4,5,6,7,8,9,10 101,102,103,104,105,106,107,108,109,110	[[8, '108'], [9, '109'], [10, '110']]
3	6 9,10 101,102,103,104,105,106,107,108,109,110	Warning: number of keys and values are not the same []
4	6 9,10 109,110	[[9, '109'], [10, '110']]

Level 2 (45%)

Next, you are going to implement two main functions, *get(key)* and *put(key, value)* to access a data item in the cache. The *get(key)* function retrieves the value of the data item given by the key parameter if the key exists in the cache. The *put(key, value)* function modifies the value of the data item, if the key exists, or inserts a new data item, if the key does not exist.

The ordering of data items in the cache must conform to the description in the Introduction, i.e., the 1st (leftmost) item is the least recently used while the last (rightmost) item is the most recently used. Every time a data item is accessed, whether using *get* or *put*, the data item has to be moved to the end (rightmost) of the cache to become the most recently used item. When a new data item is inserted to a full cache, the least recently used (leftmost) item needs to be evicted.

For example, given a cache of capacity 5: [[1,'100'], [2,'200'], [3,'300'], [4,'400']], the cache contents will be updated as follows.

1. *get(3)* → [[1,'100'], [2,'200'], [4,'400'], [3,'300']]
2. *put(2,'6')* → [[1,'100'], [4,'400'], [3,'300'], [2,'6']]
3. *put(10,'-1')* → [[1,'100'], [4,'400'], [3,'300'], [2,'6'], [10,'-1']]
4. *put(100,'-5')* → [[4,'400'], [3,'300'], [2,'6'], [10,'-1'], [100,'-5']]

In addition to the inputs you received from Level 1, you will also receive a list of commands. *"get,KEY"* denotes the get operation with key *KEY* and *"put,KEY,VALUE"* denotes the put operation with key *KEY* and value *VALUE*. Get operation prints the value of the key-value pair if the key exists, otherwise prints *"NULL"*. Put operation does not need to print anything. The command *"end"* denotes the end of the inputs and you should then print the final contents of the cache.

Inputs:

1. Capacity of the cache C (integer). C = -1 means that the cache has unlimited capacity.
2. A string of integral keys $K1$, $K2$, …, K_{Nk} separated by "," where N_k is number of keys.
3. A string of values $V1$, $V2$, …, V_{Nv} separated by "," where N_v is number of values.
4. A series of commands consisting of either *"get,KEY"* or *"put,KEY,VALUE"* separated by a newline character.
5. The input must be ended with *"end".*

Outputs:

1. Warning message if $N_k \neq N_v$.
2. For every get operation, print the value of the key-value pair if the key exists, otherwise print *"NULL".*
3. Cache as a list with format $\left[[K1, V1], [K2, V2], \ldots\ [Kn, Vn]\right]$ with the 1st (leftmost) item being the least recently used while the last (rightmost) item being the most recently used.

Assumptions:

- $1 \leq C \leq 1000\ or\ C = -1$
- $1 \leq N_k, N_v \leq 1000$
- Input keys can be converted to integers
- Input keys are unique, no duplicates
- Input values are non-null string
- None of the values will be equal to "NULL"

Examples:

Case	Sample input	Sample output
1	-1 1,2,3,4,5,6,7,8,9,10 101,102,103,104,105,106,107,108,109,110 get,3 get,6 get,1 get,100 get,5 end	103 106 101 NULL 105 [[2, '102'], [4, '104'], [7, '107'], [8, '108'], [9, '109'], [10, '110'], [3, '103'], [6, '106'], [1, '101'], [5, '105']]
2	3 1,2,3,4,5,6,7,8,9,10 101,102,103,104,105,106,107,108,109,110 get,1 get,2 get,3 get,10 get,9 end	NULL NULL NULL 110 109 [[8, '108'], [10, '110'], [9, '109']]
3	6 9,10 101,102,103,104,105,106,107,108,109,110 get,9 put,1,100 put,2,200 put,3,300 get,10 get,2 put,4,400 put,5,500	Warning: number of keys and values are not the same NULL NULL 200 NULL 200 [[4, '400'], [5, '500'], [6, '600'], [1, '10000'], [7, '700'], [2, '200']]

	put,6,600 put,1,10000 put,7,700 get,3 get,2 end	

Level 3 (30%)

After passing all the tests in Level 2, the interviewers proceed to ask a follow-up question to truly test your ability in programming and problem solving. They tell you that it is computationally expensive to move the most recently used item to the end of the list in order to conform to the description in the Introduction, i.e., the 1st (leftmost) item is the least recently used while the last (rightmost) item is the most recently used. It is because all the subsequent items need to be moved to the left by 1. For example, the steps of moving 2 in [1,2,3,4] to the end of the list is [1,2,3,4] → [1,3,3,4] → [1,3,4,4] → [1,3,4,2]. As a result, moving item to the end of a list should be avoided if possible.

The follow-up question is whether you can implement a solution without the need of moving items in the cache once they are inserted. Specifically, when accessing an item, **NO** movement should be made to any item but you are allowed to modify the item. When adding a new item to the list, it should be added to the end of the list if the cache is not full. Otherwise, the least recently used item is **REPLACED** by the new item and **NO** movement of other items are allowed. For example, given a cache of capacity 4: [[1,'100'], [2,'200'], [3,'300']], the cache contents will be updated as follows.

1. *get(2)* → [[1,'100'], [2,'200'], [3,'300']]
2. *put(4,'400')* → [[1,'100'], [2,'200'], [3,'300'], [4,'400']]
3. *get(1)* → [[1,'100'], [2,'200'], [3,'300'], [4,'400']]
4. *put(5,'500')* → [[1,'100'], [2,'200'], [5,'500'], [4,'400']]

As a result, there is **NO** limitation on how you are going to store the data. However, **ONLY** one list can be created and used for the cache and no other auxiliary list is allowed.

Note:
The test cases on VPL will be time limited. If you see timeout on VPL, it means your program is too slow and need further optimization. Nested loops are usually the most expensive operations and it is a good starting point for optimization.

Inputs:

1. Capacity of the cache C (integer). C = -1 means that the cache has unlimited capacity.
2. A string of integral keys $K1$, $K2$, …, K_{Nk} separated by "," where N_k is number of keys.
3. A string of values $V1$, $V2$, …, V_{Nv} separated by "," where N_v is number of values.
4. A series of commands consisting of either *"get,KEY"* or *"put,KEY,VALUE"* separated by a newline character.
5. The input must be ended with *"end"*.

Outputs:

1. Warning message if $N_k \neq N_v$.
2. For every get operation, print the value of the key-value pair if the key exists, otherwise print *"NULL".*
3. Cache as a list with format $\left[[K1, V1], [K2, V2], \ldots\ [Kn, Vn]\right]$ where the ordering is no longer based on recent usage but follows the specs listed in the question.

Assumptions:

- $1 \leq C \leq 10000\ or\ C = -1$
- $1 \leq N_k, N_v \leq 10000$
- Input keys can be converted to integers
- Input keys are unique, no duplicates
- Input values are non-null string
- None of the values will be equal to "NULL"
- Time constraint will be imposed on VPL and timeout is possible

Hints:

- You may want to change the structure of how you keep the data. For example, you can store extra information alongside with each data item if you find it useful.
- It is acceptable if *put* and/or other operations take more time to run as long as the total time is within the time constraint. Your program should be well within the time constraint unless it has multiple layers of loop.

Examples:

Case	Sample input	Sample output
1	-1 1,2,3,4,5,6,7,8,9,10 101,102,103,104,105,106,107,108,109,110 get,3 get,6 get,1 get,100 get,5 end	103 106 101 NULL 105 [[1, '101'], [2, '102'], [3, '103'], [4, '104'], [5, '105'], [6, '106'], [7, '107'], [8, '108'], [9, '109'], [10, '110']]
2	3 1,2,3,4,5,6,7,8,9,10 101,102,103,104,105,106,107,108,109,110 get,1 get,2 get,3 get,10 get,9 end	NULL NULL NULL 110 109 [[8, '108'], [9, '109'], [10, '110']]
3	6 9,10 101,102,103,104,105,106,107,108,109,110 get,9	Warning: number of keys and values are not the same NULL NULL

	put,1,100 put,2,200 put,3,300 get,10 get,2 put,4,400 put,5,500 put,6,600 put,1,10000 put,7,700 get,3 get,2 end	200 NULL 200 [[1, '10000'], [2, '200'], [7, '700'], [4, '400'], [5, '500'], [6, '600']]

Submission

- Virtual Programming Lab (VPL) will be set up for submission and testing. Only the last submission will be considered as the final submission.
- Test your program thoroughly before the deadline. Your program must generate output according to the given format and specifications, i.e., without extra text/space.
- The test cases in VPLs only check whether your programs meet the basic requirements. Your programs will be assessed with another set of private test cases.
- Program should only use the Python libraries/features covered in this course up to and including Array. Using other libraries, non-built-in functions, and features is not allowed and you will be given zero marks. If you are unsure, feel free to contact us for clarification.
- 20% will be deducted from the final mark for every 24 hours after the submission due date.
- Do not submit any program after the due date if your work is final. Any submission after the due date is regarded as late submission.

Plagiarism

- Confirmed plagiarism cases (detected by the system) will get zero mark and subject to disciplinary actions. The penalty applies to the source provider(s) as well. In other words, students who submit same/highly similar programs will all get zero mark. Students have full responsibility to protect their programs from being accessed by others.
- Last year, 6 students had found engaged in plagiarism. They all got zero mark for the assignment and a warning letter issued by the Department Head.

Questions

- If you have any questions, please send email (cky166@connect.hku.hk) to Mr. Ryan Chan.