**Java Assignment**

Algorithm:

Wrap the dict class in Map

1. Define get/put methods
2. If put method is called, check for duplicate key using as similar as get method, if dictionary size is larger than limit then dump data into file using pickle, use index based store maintain a local dictionary (indexMap) which will map index to hash e.g. 0->hash1,1 - >hash etc, there could be a chance when total dictionary size is larger than the limit then we can't perform operation in the same time else store key and value in current dictionary.
3. If get method is called then look in dictionary, if there's miss then compute hash of key and make a lookup in indexMap if there's a hit then read the file and go to that line number and deserialize using pickle and compare key if you find same key then return value else return None or raise error

It will work in the same time complexity with some overhead. Furthermore this can be addressed using overload e.g. If load is 100% then dump 50% of contents in the file.

Thread safety, if put method is called then aquire a lock and do the things and at the end of the operation, release lock.

Similarly for del/get methods.

Edit1:

Map: in Java's standard library there's no such data structure.

Map is nothing but a collection of key and value where keys are unique. There're different types of map where key and value both should be unique IIRC that's called bitmap.

In Java dictionary serves the same purpose, where key and value are stored.

Heap:

In Java memory is allocated in Heap and manage by memory manager, which takes care of memory allocation and deallocation works.

[Memory Management](https://docs.python.org/2/c-api/memory.html)

Memory management in Java involves a private heap containing all Java objects and data structures. The management of this private heap is ensured internally by the Java memory manager. The Java memory manager has different components which deal with various dynamic storage management aspects, like sharing, segmentation, preallocation or caching.

There's no much we can do here in memory management. Few things like total memory allocated, del operator can be used to manage memory. Generally speaking we leave these things to Java interpreter.

So best bet here could to limit the number of element e.g. 10K entries or something similar, but there could be a chance that dictionary is sparse, in this case some memory would be wasted.

Edit2:

Example:

Let’s say limit is 4 and we spill 50% of data at a time, for illustration will write to file as it as.

If we don’t want to have indexMap then you can write that to file as well line by line but again you have to read that file and do the need full.

—————————————————————————————————-

put <1,2>

1 is neither in map nor in file so just add this {1:2}

To check this you can directly make a lookup in Map, to check in file you need to make a lookup in indexMap with hash value of 1, if indexMap has this hash value then there’s a chance of having 5 in file, in that case you need to read file and deserialise the line see **put <1,4>** for such example.

—————————————————

similarly for {2:3,3:4,4:5}

————————————-

put <5,6>

5 is neither in map nor in file overflow so spill 50% of data to file

lets {2:3, 3:4} goes to file and they have hash a, b respectively

data.dat

2,3

3,4

Map will have {1:2, 4:5, 5:6}

indexMap = { a:0, b:1}

——————————————-

put <6,7>

data.dat

2,3

3,4

Map will have {1:2, 4:5, 5:6, 6:7}

indexMap = { a:0, b:1}

—————————————————-

put <7,8> , overflow

write 1:2, 4:5 to file let’s their hash values are b,c respectively

data.dat

2,3

3,4 : 1,2

4,5

Map will have { 5:6, 6:7, 7:8}

indexMap = { a:0, b:1, c : 2}

—————————————————

put <5,5>

5 is in map so just update: {5:5, 6:7, 7:8}

—————————————————

put <1,4>

1 is not map and it’s hash value is ‘b’ so let’s make a lookup in indexMap, yes ‘b’ is there so let’s read file and go to line number 2, read that line and deserialise this line.

So we got {3:4, 1:2} and 1 is here so update this as {3:4, 1:4} and write back to line number 2

—————————————————————