HyunTaek Oh (ohhyun@oregonstate.edu)

Rob Hess

CS 261_020

Dec. 13, 2023

Final Exam

I choose the first scenario “Undo/redo in a 3D world building application”. In the scenario, undo/reo feature with this application will allow a user to undo actions they’ve already taken and to redo actions they’ve undone, so we need to focus on four following requirements.

· Undo/redo the most recent action taken by the user.

· Undo/redo function can be continued by users as many times as they want.

· Each subsequent “undo/redo” will undo/re-execute the most recent actions.

· The corresponding action can be quickly and easily accessed and used to undo or redo the action it represents.

From these requirements, every instance of actions generated by a user should be stored in the order of arrival and the oldest data would not be considered currently since undo/redo is affected by the most recent action. It means we only care about the most recent action taken, not the oldest one, and then the next recent action would be processed. Moreover, we do not need to use direct access like using indices to find a specific data because we only concentrate on handling the most recent data. Then, when user did many actions, the storage of user actions would need to undo/redo function as many times as users want. If there are no instances of action, user cannot utilize undo/redo function. Furthermore, each data should be linked, processing one by one to access quickly and easily.

To implement with the software features above, I will use stack with linked list. According to the lecture note “Stack, Queues, and Deques”, a stack is a linear ADT that imposes a last in, first out (or LIFO) order on elements. Due to this order, the oldest data will be removed in the last order. Stack has a pointer to point to a top element, which is the most recently inserted, and the pointer will be moved to point to next data after insertion (PUSH) and removal (POP). We can implement a stack using a singly-linked list, where the head of the list corresponds to the top of the stack. When insertion or removal happen, the head of linked list will point to the most recent added or previously pointed data. The most recently added data will be linked with the next added data, caring only the next step. The operations of this data structure have O(1) operations (best-case, worst-case, and average), which can access quickly and easily.

By using the features of a stack, all the requirements of undo/redo feature can be handled. Since a stack only manipulates the top element, which is the most recently inserted, the first requirement can be satisfied. In the case of second requirement, if there are data generated by user in the stack, user can undo/redo as many times as they want, maintaining the order of data. The third requirement can also be solved by using the stack because each data is in order, and then each subsequent undo/redo will undo/re-execute the next most recent actions. Due to the runtime complexity of stack operations, data insertion and removal have O(1) time complexity, which means they are not affected by the number of data and access quickly and easily.

However, there is a possible trade-off that when a user would like to search a specific action, this data structure would be failed to find it quickly because every time user have to find a specific data from the top element to the bottom element in order, not direct access like using index.