First of all I define where is the mark phase in order to know all the reachable and allocated inheritance of the root nodes, followed by a sweep phase, which frees each unmarked allocated mem which is in our case is walk_region_and_mark(). This will work in two stage

1 + Mark Phase

During this phase, the Garbage Collector identifies objects that are still in use and marks their "mark bit" to true. The search begins with a root set of references held in local variables on the Stack, or global variables. Starting from the root references, the Garbage Collector will perform a deep search for objects that have references to these roots(using traversing to root node), any object that holds a reference to another object, the GC keeps that object alive(since out chunk is a 8 bit so I increase the count 8 ). It is important to note that during the Mark Phase, application threads need to be stopped to avoid possible changes to the object's state during this marking phase.

Circular references are not a problem for Mark and Sweep, if you follow the diagram above, there is a circular reference represented by a square but it is not directly accessible from root, hence the references type. It will not be marked as existing and allow the GC to collect it as a garbage resource

2 + Sweep Phase

In the scan phase, all unmarked objects from the Mark stage are removed from memory, freeing up HEAP space. Due to this fragmentation, the next memory allocation may fail if it needs more memory than the remaining free area. To solve this problem, we added a new phase to solve it “is_pointer” to “ determine if what "looks" like a pointer actually points to an in use block in the heap. if so, return a pointer to its header” because the isPtr(ptr p) function uses the tree to perform a binary search of the allocated blocks.

The marker phase calls the marker function shown in line 125 once for each root node. The walk_region_and_mark function returns immediately if *temp does not point to an allocated and unsigned heap block. If not, it marks the block and calls itself recursively over each word in the block. Each call to the marker function will mark any unmarked and accessible child of some root node. At the end of the mark stage, any allocated blocks that are not marked are guaranteed to be unreachable and, therefore, garbage can be recovered during the sweep phase.