MP6 Report. Chen Yanghonghui 3210110208 yc47

A).Analysis of memory allocation and running time in original MP5 implementation:

1.Check the Running Time of the original mp5 implementation

abcd@cyhh mp6 % time ./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png
Loading Tile Images... (4730/4730)... 4479 unique images loaded
Populating Mosaic: setting tile (399, 532)
Drawing Mosaic: resizing tiles (213200/213200)
Saving Output Image... Done
./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png 4.90s user 1.19s system 9% cpu 1:05.55 total

2.Check the Space Utilization of the original mp5 implementation

valgrind . /mp6 tests/source,png ../mp5/mp5_pngs/ 400 5 mosaic .png
Loading Tile Images...(4730/4730)...4479 unique images loadedPopulating Mosaic;

setting tile (399， 532)Drawing Mosaic: resizing tiles (213200/213200)

saving Output Image...Done

HEAP SUMMARY:
in use at exit: 5,461,692,867 bytes in 4 480 blockstotal heap usage: 738,217 allocs, 733,737 frees,5,460 , 578,627 bytes allocated

LEAK SUMMARY:
definitely lost: 16 bytes in 1 blocks
indirectly lost: 5,461,692,851 bytes in 4,479 blocks
possibly lost: bytes in blocks
still reachable: 0 bytes in 0 blocks
suppressed: 0 bytes in 0 blocks

B)What I have done to imporve the memory utilization :

When it comes to the too much space utilization, I think it is mainly due to the storage of the pixels of the tileimage
for every block of the source image in the original mp5 implementation. And it is also a great number of space utilization
after using the pointers for each block of the source image. So, I think of the data structure of hash table to store the
key of each tileimage for each block of source image because each pointer occupies 8 bytes while each unsigned short occupies
2 bytes. Therefore, I modified function “mapTiles”. I designed a hash table, matching

index number to pointers of tile images, and
replaced pointers in myImages with index numbers. Specifically, every tile image is assigned an unsigned short int number,
from 0 to total number of tile images. The hash table is declared as a member variable in class “MosaicCanvas”.
Then in function “mapTiles”, the returned MosaicCanvas has an unsigned short int representing which tile image at each block of source
image and has a hash table matching these unsigned short ints to pointers to tile images.

C) Analysis of memory allocation and running time after optimization:

1.Check the Running Time of the optimized MP6 implementation

abcd@cyhh mp6 % time ./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png
Loading Tile Images... (4730/4730)... 4479 unique images loaded
Populating Mosaic: setting tile (399, 532)
Drawing Mosaic: resizing tiles (213200/213200)
Saving Output Image... Done
./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png 4.99s user 1.19s system 20% cpu 30.557 total

*** So, by comparison, the time consumption has increased a little unexpectedly

2.Check the Space Utilization of the optimized MP6 implementation

valgrind . /mp6 tests/source,png ../mp5/mp5_pngs/ 400 5 mosaic .png
Loading Tile Images...(4730/4730)...4479 unique images loadedPopulating Mosaic;

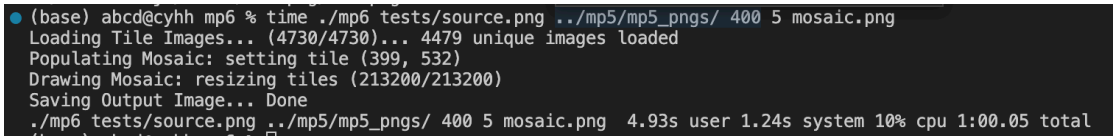
setting tile (399， 532)Drawing Mosaic: resizing tiles (213200/213200)

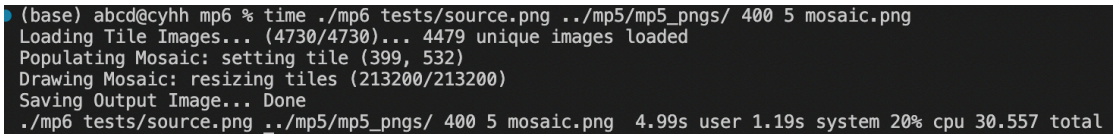
saving Output Image...Done

HEAP SUMMARY:
in use at exit: 4,460,579,651 bytes in 4 480 blockstotal heap usage: 738,217 allocs, 733,737 frees,5,460 , 578,627 bytes allocated

LEAK SUMMARY:
definitely lost: 16 bytes in 1 blocks
indirectly lost: 4,460,579,651 bytes in 4,479 blocks
possibly lost: bytes in blocks
still reachable: 0 bytes in 0 blocks
suppressed: 0 bytes in 0 blocks

**** So, by comparison, the memory utilization has decreased by 2113216 bytes
**** The memory space has decreated, which means the optimization is good





D)Analysis of the running time of the drawing mosaic image

Drawing a mosaic image has an O(w*h*w'*h') running time, because it firstly loops through all small blocks of source image from left to right, up to down, which takes O(w*h) time, then for each block, it also loops through all pixels of tile images from left to right, up to down, which takes O(w*h*w'*h') eventually.