CS39002 Operating Systems Laboratory
Spring 2025

Lab Assignment: 10
Date: 02–April –2025

Demand Paging with Page Replacement

In this assignment, we continue with the embedded device of LA9. The specifications of this system are reproduced below. The input is to be read from *search.txt* which is organized in the same format as in LA9.

Total memory = 64 MB
Page/Frame size = 4 KB
OS usage = 16 MB (higher side)
User space = 48 MB
Number of user frames = 12288 (numbered 0, 1, 2, . . . , 12287)
Number of processes = n (in the range [50, 500], PID's are numbered 0, 1, 2, . . . , n – 1)
Number of binary searches per process = m (in the range [10, 100])
Virtual memory size = 2048 pages (this is also the page-table size of each process)
Number of essential pages = 10 (page numbers 0, 1, 2, . . . , 9)
Number of pages for storing the array A = 2038 (page numbers 10, 11, 12, . . . , 2047)
Size of each page-table entry = 16 bits (unsigned short int)
14-bit frame addresses (bits 0 – 13), Bit 15 is the valid bit, Bit 14 to be used as the reference bit

In LA9, processes are swapped out when memory is full. In this assignment, we keep all the processes in memory. When memory is *full*, we replace valid pages by new pages. Approximate LRU policy is used for page replacement.

Global frame allocation

Maintain a list of free frames (FFLIST). Each entry in the FFLIST consists of (1) a frame number, (2) the last owner (PID) of the frame (–1 if none), and (3) the page number of the last owner stored in the frame (–1 if no last owner). In LA9, the FFLIST consisted only of free-frame numbers, and was implemented as a FIFO queue. Here, you need to make passes through the FFLIST, so it is technically no longer a queue. Use an array of triples (frame number, last owner, last owner's page number) as the FFLIST. Also maintain a counter NFF (number of free frames). Make your own implementation.

To start with, add all the 12288 user frames to the FFLIST. When you start the n processes, allocate 10 essential frames to each from the FFLIST. Subsequently, free frames are allocated on demand. However, we will never allow NFF (the number of free frames) to fall below NFFMIN = 1000. So long as we have NFF > NFFMIN, any process requesting a frame is granted a free frame. You do not need to check how many frames are already owned by the process. In our application, processes carry out binary searches in a round-robin fashion, so this frame-allocation strategy will ensure more or less equal distribution of frames to the processes. Although the sizes of A for different processes vary within a factor of 2, a proportional frame-preallocation scheme based on the sizes of A may be more appropriate, but you do not need to do this.

When NFF = NFFMIN, and a process requests a frame, a free frame in the FFLIST is no longer allocated straightaway to the process. On the contrary, a free frame in the FFLIST is swapped with a frame currently allocated to the process. This situation is described in detail in the next section.

When a process exits (after completing its m searches), all the frames held by that process at that time are returned to the FFLIST. These frames join the FFLIST with no last owner, because the exiting process will never try to use/reclaim these (or other) frames. Free frames released by an exiting process will be allocated to the existing processes on demand.

Local page replacement

A process i, in each iteration of its binary-search loop, makes a memory access of $A[M]$. Assume that this element is stored in logical page p of the process. If the page table of process i indicates the valid bit to be 1 for the p-th page, that page is already residing in a frame, so the process proceeds to the next iteration of the binary-search loop. In the other case, that is, the valid bit is 0, this memory access encounters a page fault. If NFF > NFFMIN, the process gets a free frame from the FFLIST, and continues. If NFF = NFFMIN, then a page replacement is necessary.

We use the approximate LRU replacement algorithm. The 14-th bit in each page-table entry is used as the **reference bit**. Moreover, each page-table entry should additionally contain a 16-bit counter for storing the history of accesses made in the recent past. The 10 essential pages always have `0xffff` as this history, and will never be replaced. A page q with valid bit 1 and with the minimum history will be selected from the page table for replacement. When a new page is loaded (after a page fault or otherwise), its history is set to `0xffff` (it is the most recently used page at that moment, and must not be replaced in the next iteration of the binary-search loop).

Any access of $A[M]$ (with or without page fault) will set the reference bit of the corresponding page. After the end of the binary-search loop, the history of each valid page is updated by a 1-bit right shift followed by inserting the reference bit at the MSB (15-th) position. The reference bits of the valid pages are then cleared.

Once a page q (a valid page of minimum history) is located for replacement (to accommodate page p that caused the page fault), the following steps should be carried out in the given order. Recall again that page replacement is done only when NFF = NFFMIN.

1. First check in the FFLIST whether there exists a free frame f with last owner i and page number p. If that is the case, page p of process i has been replaced in the recent past. Use this frame f to store the new page p. No reloading of the page from the hard disk is needed in this case. In your program, you actually do not load pages, but follow this strategy to simulate a real-life situation correctly.

2. If Attempt 1 fails, check in the FFLIST whether there is a free frame with no owner. If such a frame f is found, allocate f to page p.

3. If both Attempts 1 and 2 fail, try to locate in the FFLIST a free frame whose last owner was i (of course, the page number stored will be different from p, otherwise Attempt 1 would have succeeded). If such a frame f is found, allocate f to page p. If there are multiple free frames with i as the last owner, pick any one of them.

4. If all the above three attempts fail, pick a *random* free frame f from FFLIST for allocating to page p.

Once a free frame f is obtained to accommodate page p of process i, you need to free the frame g storing the victim page q of process i. Add (g, i, q) to FFLIST. Also update the page table of process i to reflect these changes.

When a process exits, add, to the FFLIST, all the frames currently held by that process, after clearing the ownership information in all these frames.

Use a compile-time flag `VERBOSE` for the type of output. In the default (non-verbose) mode, print for each process (and also the aggregate) the number of page accesses, the number and percentage of page faults, the number and percentage of page replacements, and the number (and percentage) of page replacements handled by each of Attempts 1 – 4. In the verbose mode, supply more detailed paging information. See the sample provided.

Submit a single C/C++ file LRU.c(pp).

Sample Output

We use the same *search.txt* as in LA9, with $n = 128$ process, each making $m = 64$ binary searches. The non-verbose output (statistics only) is given below. An archive (storing other files) will be provided to you. The percentages of faults and replacements are with respect to the number of memory accesses, whereas the percentages in attempt statistics are with respect to the number of page replacements.

```
gcc -Wall -o runsearch LRU.c
./runsearch
+++ Page access summary
    PID     Accesses        Faults         Replacements                    Attempts
    0         1339      409   (30.55%)    335   (25.02%)      6 +   5 + 324 +   0  (1.79% + 1.49% + 96.72% + 0.00%)
    1         1326      356   (26.85%)    266   (20.06%)      1 +   4 + 261 +   0  (0.38% + 1.50% + 98.12% + 0.00%)
    2         1322      379   (28.67%)    291   (22.01%)      9 +   7 + 275 +   0  (3.09% + 2.41% + 94.50% + 0.00%)
    3         1282      336   (26.21%)    266   (20.75%)      8 +   9 + 249 +   0  (3.01% + 3.38% + 93.61% + 0.00%)
    4         1280      367   (28.67%)    287   (22.42%)     13 +  11 + 263 +   0  (4.53% + 3.83% + 91.64% + 0.00%)
    5         1290      331   (25.66%)    252   (19.53%)      4 +   4 + 244 +   0  (1.59% + 1.59% + 96.83% + 0.00%)
    6         1288      373   (28.96%)    293   (22.75%)      4 +  14 + 275 +   0  (1.37% + 4.78% + 93.86% + 0.00%)
    7         1341      408   (30.43%)    314   (23.42%)     11 +  15 + 288 +   0  (3.50% + 4.78% + 91.72% + 0.00%)
    8         1328      387   (29.14%)    298   (22.44%)      3 +  13 + 282 +   0  (1.01% + 4.36% + 94.63% + 0.00%)
    9         1302      381   (29.26%)    293   (22.50%)      0 +   9 + 284 +   0  (0.00% + 3.07% + 96.93% + 0.00%)
    10        1334      403   (30.21%)    316   (23.69%)     12 +   8 + 296 +   0  (3.80% + 2.53% + 93.67% + 0.00%)
    11        1339      402   (30.02%)    319   (23.82%)     11 +  12 + 296 +   0  (3.45% + 3.76% + 92.79% + 0.00%)
    12        1342      417   (31.07%)    322   (23.99%)     15 +  15 + 292 +   0  (4.66% + 4.66% + 90.68% + 0.00%)
    13        1334      390   (29.24%)    304   (22.79%)      6 +  13 + 285 +   0  (1.97% + 4.28% + 93.75% + 0.00%)
    14        1310      371   (28.32%)    291   (22.21%)      2 +   5 + 284 +   0  (0.69% + 1.72% + 97.59% + 0.00%)
    15        1320      381   (28.86%)    300   (22.73%)     10 +  16 + 274 +   0  (3.33% + 5.33% + 91.33% + 0.00%)
    16        1337      385   (28.80%)    292   (21.84%)      4 +  12 + 276 +   0  (1.37% + 4.11% + 94.52% + 0.00%)
    17        1334      393   (29.46%)    313   (23.46%)      6 +  15 + 292 +   0  (1.92% + 4.79% + 93.29% + 0.00%)
    18        1295      362   (27.95%)    284   (21.93%)      8 +  12 + 264 +   0  (2.82% + 4.23% + 92.96% + 0.00%)
    19        1334      384   (28.79%)    298   (22.34%)     12 +  12 + 274 +   0  (4.03% + 4.03% + 91.95% + 0.00%)
    20        1335      409   (30.64%)    318   (23.82%)      7 +  12 + 299 +   0  (2.20% + 3.77% + 94.03% + 0.00%)
    21        1304      360   (27.61%)    270   (20.71%)     10 +   7 + 253 +   0  (3.70% + 2.59% + 93.70% + 0.00%)
    22        1291      367   (28.43%)    285   (22.08%)      8 +  11 + 266 +   0  (2.81% + 3.86% + 93.33% + 0.00%)
    23        1326      409   (30.84%)    325   (24.51%)      8 +  12 + 305 +   0  (2.46% + 3.69% + 93.85% + 0.00%)
    24        1318      395   (29.97%)    311   (23.60%)      1 +  16 + 294 +   0  (0.32% + 5.14% + 94.53% + 0.00%)
    25        1338      411   (30.72%)    323   (24.14%)      6 +  12 + 305 +   0  (1.86% + 3.72% + 94.43% + 0.00%)
    26        1279      339   (26.51%)    262   (20.48%)      4 +   7 + 251 +   0  (1.53% + 2.67% + 95.80% + 0.00%)
    27        1331      397   (29.83%)    306   (22.99%)      5 +   7 + 294 +   0  (1.63% + 2.29% + 96.08% + 0.00%)
    28        1312      371   (28.28%)    287   (21.88%)     15 +  13 + 259 +   0  (5.23% + 4.53% + 90.24% + 0.00%)
    29        1296      355   (27.39%)    273   (21.06%)     10 +   8 + 255 +   0  (3.66% + 2.93% + 93.41% + 0.00%)
    30        1318      384   (29.14%)    304   (23.07%)     11 +  11 + 282 +   0  (3.62% + 3.62% + 92.76% + 0.00%)
    31        1337      394   (29.47%)    305   (22.81%)      3 +  10 + 292 +   0  (0.98% + 3.28% + 95.74% + 0.00%)
    32        1301      361   (27.75%)    275   (21.14%)      9 +  12 + 254 +   0  (3.27% + 4.36% + 92.36% + 0.00%)
    33        1299      350   (26.94%)    266   (20.48%)     11 +   8 + 247 +   0  (4.14% + 3.01% + 92.86% + 0.00%)
    34        1282      349   (27.22%)    273   (21.29%)      8 +  10 + 255 +   0  (2.93% + 3.66% + 93.41% + 0.00%)
    35        1325      365   (27.55%)    280   (21.13%)      1 +  11 + 268 +   0  (0.36% + 3.93% + 95.71% + 0.00%)
    36        1339      422   (31.52%)    341   (25.47%)     10 +  13 + 318 +   0  (2.93% + 3.81% + 93.26% + 0.00%)
    37        1337      420   (31.41%)    330   (24.68%)      5 +  11 + 314 +   0  (1.52% + 3.33% + 95.15% + 0.00%)
    38        1305      367   (28.12%)    287   (21.99%)     11 +   9 + 267 +   0  (3.83% + 3.14% + 93.03% + 0.00%)
    39        1338      376   (28.10%)    290   (21.67%)      5 +  16 + 269 +   0  (1.72% + 5.52% + 92.76% + 0.00%)
    40        1306      391   (29.94%)    308   (23.58%)      8 +  12 + 288 +   0  (2.60% + 3.90% + 93.51% + 0.00%)
    41        1341      390   (29.08%)    307   (22.89%)      5 +  11 + 291 +   0  (1.63% + 3.58% + 94.79% + 0.00%)
    42        1321      388   (29.37%)    303   (22.94%)     14 +  13 + 276 +   0  (4.62% + 4.29% + 91.09% + 0.00%)
    43        1342      390   (29.06%)    304   (22.65%)      7 +   7 + 290 +   0  (2.30% + 2.30% + 95.39% + 0.00%)
    44        1294      381   (29.44%)    304   (23.49%)      6 +  13 + 285 +   0  (1.97% + 4.28% + 93.75% + 0.00%)
    45        1335      394   (29.51%)    310   (23.22%)      0 +   8 + 302 +   0  (0.00% + 2.58% + 97.42% + 0.00%)
    46        1343      422   (31.42%)    336   (25.02%)      7 +  11 + 318 +   0  (2.08% + 3.27% + 94.64% + 0.00%)
    47        1288      366   (28.42%)    280   (21.74%)      9 +  12 + 259 +   0  (3.21% + 4.29% + 92.50% + 0.00%)
    48        1316      355   (26.98%)    272   (20.67%)      8 +  12 + 252 +   0  (2.94% + 4.41% + 92.65% + 0.00%)
    49        1334      398   (29.84%)    311   (23.31%)      9 +  14 + 288 +   0  (2.89% + 4.50% + 92.60% + 0.00%)
    50        1311      379   (28.91%)    289   (22.04%)      9 +  10 + 270 +   0  (3.11% + 3.46% + 93.43% + 0.00%)
    51        1314      368   (28.01%)    281   (21.39%)      8 +  13 + 260 +   0  (2.85% + 4.63% + 92.53% + 0.00%)
    52        1335      417   (31.24%)    342   (25.62%)      7 +  12 + 323 +   0  (2.05% + 3.51% + 94.44% + 0.00%)
    53        1318      358   (27.16%)    276   (20.94%)      2 +   6 + 268 +   0  (0.72% + 2.17% + 97.10% + 0.00%)
    54        1326      391   (29.49%)    311   (23.45%)      5 +  12 + 294 +   0  (1.61% + 3.86% + 94.53% + 0.00%)
    55        1336      373   (27.92%)    286   (21.41%)      8 +  10 + 268 +   0  (2.80% + 3.50% + 93.71% + 0.00%)
    56        1317      378   (28.70%)    296   (22.48%)      6 +  10 + 280 +   0  (2.03% + 3.38% + 94.59% + 0.00%)
    57        1334      402   (30.13%)    308   (23.09%)      0 +   5 + 303 +   0  (0.00% + 1.62% + 98.38% + 0.00%)
    58        1335      396   (29.66%)    309   (23.15%)      6 +   7 + 296 +   0  (1.94% + 2.27% + 95.79% + 0.00%)
    59        1321      374   (28.31%)    287   (21.73%)      4 +   7 + 276 +   0  (1.39% + 2.44% + 96.17% + 0.00%)
    60        1338      397   (29.67%)    316   (23.62%)      1 +   3 + 312 +   0  (0.32% + 0.95% + 98.73% + 0.00%)
    61        1330      391   (29.40%)    308   (23.16%)      7 +   6 + 295 +   0  (2.27% + 1.95% + 95.78% + 0.00%)
    62        1295      374   (28.88%)    301   (23.24%)      0 +   5 + 296 +   0  (0.00% + 1.66% + 98.34% + 0.00%)
    63        1312      375   (28.58%)    290   (22.10%)      1 +   3 + 286 +   0  (0.34% + 1.03% + 98.62% + 0.00%)
    64        1339      419   (31.29%)    330   (24.65%)      3 +   7 + 320 +   0  (0.91% + 2.12% + 96.97% + 0.00%)
    65        1314      381   (29.00%)    295   (22.45%)      9 +   6 + 280 +   0  (3.05% + 2.03% + 94.92% + 0.00%)
    66        1325      404   (30.49%)    314   (23.70%)      2 +   6 + 306 +   0  (0.64% + 1.91% + 97.45% + 0.00%)
    67        1307      361   (27.62%)    282   (21.58%)      0 +   0 + 281 +   1  (0.00% + 0.00% + 99.65% + 0.35%)
    68        1311      352   (26.85%)    272   (20.75%)      6 +   6 + 260 +   0  (2.21% + 2.21% + 95.59% + 0.00%)
    69        1333      414   (31.06%)    323   (24.23%)      6 +   8 + 309 +   0  (1.86% + 2.48% + 95.67% + 0.00%)
    70        1310      389   (29.69%)    305   (23.28%)      0 +   4 + 301 +   0  (0.00% + 1.31% + 98.69% + 0.00%)
    71        1285      344   (26.77%)    274   (21.32%)      5 +   7 + 262 +   0  (1.82% + 2.55% + 95.62% + 0.00%)
    72        1333      403   (30.23%)    315   (23.63%)      3 +   5 + 307 +   0  (0.95% + 1.59% + 97.46% + 0.00%)
    73        1280      315   (24.61%)    233   (18.20%)      1 +   0 + 231 +   1  (0.43% + 0.00% + 99.14% + 0.43%)
```

```
74        1315      367   (27.91%)    282   (21.44%)      3 +   6 + 273 +   0  (1.06% + 2.13% + 96.81% + 0.00%)
75        1321      368   (27.86%)    278   (21.04%)      0 +   0 + 277 +   1  (0.00% + 0.00% + 99.64% + 0.36%)
76        1305      361   (27.66%)    294   (22.53%)      2 +   7 + 285 +   0  (0.68% + 2.38% + 96.94% + 0.00%)
77        1326      409   (30.84%)    320   (24.13%)      2 +   5 + 313 +   0  (0.62% + 1.56% + 97.81% + 0.00%)
78        1286      345   (26.83%)    265   (20.61%)      4 +   7 + 254 +   0  (1.51% + 2.64% + 95.85% + 0.00%)
79        1331      390   (29.30%)    306   (22.99%)      1 +   7 + 298 +   0  (0.33% + 2.29% + 97.39% + 0.00%)
80        1313      372   (28.33%)    284   (21.63%)      4 +   6 + 274 +   0  (1.41% + 2.11% + 96.48% + 0.00%)
81        1331      377   (28.32%)    286   (21.49%)     11 +   7 + 268 +   0  (3.85% + 2.45% + 93.71% + 0.00%)
82        1315      379   (28.82%)    293   (22.28%)      6 +   6 + 281 +   0  (2.05% + 2.05% + 95.90% + 0.00%)
83        1310      375   (28.63%)    290   (22.14%)      3 +   3 + 284 +   0  (1.03% + 1.03% + 97.93% + 0.00%)
84        1315      369   (28.06%)    286   (21.75%)      0 +   3 + 283 +   0  (0.00% + 1.05% + 98.95% + 0.00%)
85        1319      353   (26.76%)    265   (20.09%)      2 +   9 + 254 +   0  (0.75% + 3.40% + 95.85% + 0.00%)
86        1340      407   (30.37%)    316   (23.58%)      2 +   5 + 309 +   0  (0.63% + 1.58% + 97.78% + 0.00%)
87        1297      351   (27.06%)    263   (20.28%)      2 +   3 + 258 +   0  (0.76% + 1.14% + 98.10% + 0.00%)
88        1289      373   (28.94%)    291   (22.58%)      0 +   0 + 290 +   1  (0.00% + 0.00% + 99.66% + 0.34%)
89        1309      384   (29.34%)    305   (23.30%)      4 +   7 + 294 +   0  (1.31% + 2.30% + 96.39% + 0.00%)
90        1308      361   (27.60%)    281   (21.48%)      0 +   5 + 276 +   0  (0.00% + 1.78% + 98.22% + 0.00%)
91        1320      385   (29.17%)    302   (22.88%)      5 +   5 + 292 +   0  (1.66% + 1.66% + 96.69% + 0.00%)
92        1280      339   (26.48%)    259   (20.23%)      0 +   5 + 254 +   0  (0.00% + 1.93% + 98.07% + 0.00%)
93        1299      369   (28.41%)    298   (22.94%)      3 +   7 + 288 +   0  (1.01% + 2.35% + 96.64% + 0.00%)
94        1315      382   (29.05%)    299   (22.74%)      1 +   4 + 294 +   0  (0.33% + 1.34% + 98.33% + 0.00%)
95        1325      396   (29.89%)    313   (23.62%)      3 +   7 + 303 +   0  (0.96% + 2.24% + 96.81% + 0.00%)
96        1334      409   (30.66%)    316   (23.69%)      1 +   3 + 312 +   0  (0.32% + 0.95% + 98.73% + 0.00%)
97        1312      376   (28.66%)    294   (22.41%)      4 +   7 + 283 +   0  (1.36% + 2.38% + 96.26% + 0.00%)
98        1341      392   (29.23%)    304   (22.67%)      1 +   7 + 296 +   0  (0.33% + 2.30% + 97.37% + 0.00%)
99        1337      402   (30.07%)    316   (23.64%)      2 +   6 + 308 +   0  (0.63% + 1.90% + 97.47% + 0.00%)
100       1328      385   (28.99%)    317   (23.87%)      7 +   8 + 302 +   0  (2.21% + 2.52% + 95.27% + 0.00%)
101       1336      412   (30.84%)    330   (24.70%)      5 +   4 + 321 +   0  (1.52% + 1.21% + 97.27% + 0.00%)
102       1305      345   (26.44%)    269   (20.61%)      7 +   9 + 253 +   0  (2.60% + 3.35% + 94.05% + 0.00%)
103       1280      353   (27.58%)    275   (21.48%)      2 +   7 + 266 +   0  (0.73% + 2.55% + 96.73% + 0.00%)
104       1337      388   (29.02%)    298   (22.29%)      1 +   6 + 291 +   0  (0.34% + 2.01% + 97.65% + 0.00%)
105       1339      397   (29.65%)    310   (23.15%)      3 +   7 + 300 +   0  (0.97% + 2.26% + 96.77% + 0.00%)
106       1289      370   (28.70%)    295   (22.89%)      3 +   5 + 287 +   0  (1.02% + 1.69% + 97.29% + 0.00%)
107       1285      374   (29.11%)    300   (23.35%)      5 +   7 + 288 +   0  (1.67% + 2.33% + 96.00% + 0.00%)
108       1339      405   (30.25%)    319   (23.82%)      6 +   7 + 306 +   0  (1.88% + 2.19% + 95.92% + 0.00%)
109       1314      402   (30.59%)    319   (24.28%)      2 +   5 + 312 +   0  (0.63% + 1.57% + 97.81% + 0.00%)
110       1338      406   (30.34%)    316   (23.62%)      1 +   3 + 312 +   0  (0.32% + 0.95% + 98.73% + 0.00%)
111       1282      364   (28.39%)    283   (22.07%)      2 +   4 + 277 +   0  (0.71% + 1.41% + 97.88% + 0.00%)
112       1339      408   (30.47%)    310   (23.15%)      1 +   6 + 303 +   0  (0.32% + 1.94% + 97.74% + 0.00%)
113       1335      401   (30.04%)    312   (23.37%)      3 +   4 + 305 +   0  (0.96% + 1.28% + 97.76% + 0.00%)
114       1337      406   (30.37%)    319   (23.86%)      3 +   6 + 310 +   0  (0.94% + 1.88% + 97.18% + 0.00%)
115       1303      377   (28.93%)    303   (23.25%)     10 +   9 + 284 +   0  (3.30% + 2.97% + 93.73% + 0.00%)
116       1311      372   (28.38%)    283   (21.59%)      3 +   5 + 275 +   0  (1.06% + 1.77% + 97.17% + 0.00%)
117       1322      372   (28.14%)    281   (21.26%)      1 +   5 + 275 +   0  (0.36% + 1.78% + 97.86% + 0.00%)
118       1317      347   (26.35%)    260   (19.74%)      4 +   6 + 250 +   0  (1.54% + 2.31% + 96.15% + 0.00%)
119       1316      372   (28.27%)    288   (21.88%)      2 +   8 + 278 +   0  (0.69% + 2.78% + 96.53% + 0.00%)
120       1314      371   (28.23%)    284   (21.61%)      2 +   3 + 279 +   0  (0.70% + 1.06% + 98.24% + 0.00%)
121       1300      370   (28.46%)    287   (22.08%)      6 +   5 + 276 +   0  (2.09% + 1.74% + 96.17% + 0.00%)
122       1285      352   (27.39%)    274   (21.32%)      7 +   5 + 262 +   0  (2.55% + 1.82% + 95.62% + 0.00%)
123       1296      368   (28.40%)    281   (21.68%)      8 +   7 + 266 +   0  (2.85% + 2.49% + 94.66% + 0.00%)
124       1321      357   (27.02%)    276   (20.89%)      8 +   7 + 261 +   0  (2.90% + 2.54% + 94.57% + 0.00%)
125       1326      404   (30.47%)    313   (23.60%)      2 +   6 + 305 +   0  (0.64% + 1.92% + 97.44% + 0.00%)
126       1318      389   (29.51%)    310   (23.52%)      1 +   4 + 305 +   0  (0.32% + 1.29% + 98.39% + 0.00%)
127       1316      387   (29.41%)    302   (22.95%)      6 +   7 + 289 +   0  (1.99% + 2.32% + 95.70% + 0.00%)

Total     168588    48691 (28.88%)    37943 (22.51%)    642 + 1000 + 36297 + 4  (1.69% + 2.64% + 95.66% + 0.01%)
```

You may use the following `makefile`.

```
run: LRU.c
        gcc -Wall -o runsearch LRU.c
        ./runsearch

vrun: LRU.c
        gcc -Wall -DVERBOSE -o runsearch LRU.c
        ./runsearch

db: gensearch.c
        gcc -Wall -o gensearch gensearch.c
        ./gensearch

clean:
        -rm -f runsearch gensearch
```
