CSC369 A2 Readme

The program we have provided is hashtable, which provides an implementation of a hashtable in C. The program then initializes a hashtable with 65536 values, inserts four key-value pairs, then reads the values at those four keys and prints them.

Program: simpleloop

memsize: 50

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	70.8558	7352	3024	2974	250	2724
FIFO	71.3088	7399	2977	2927	202	2725
LRU	73.1496	7590	2786	2736	86	2650
Clock	73.0821	7583	2793	2743	89	2654
OPT	74.1712	7696	2680	2630	24	2606

memsize: 100

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	73.2363	7599	2777	2677	60	2617
FIFO	73.3520	7611	2765	2665	44	2621
LRU	74.0266	7681	2695	2595	3	2592
Clock	73.9880	7677	2699	2599	5	2594
OPT	74.5181	7732	2644	2544	0	2544

memsize: 150

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	73.7760	7655	2721	2571	16	2555
FIFO	73.7375	7651	2725	2575	16	2559
LRU	74.0652	7685	2691	2541	0	2541
Clock	74.0555	7684	2692	2542	0	2542
OPT	74.5181	7732	2644	2494	0	2494

memsize: 200

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	73.7471	7652	2724	2524	17	2507
FIFO	73.8146	7659	2717	2517	12	2505
LRU	74.0652	7685	2691	2491	0	2491
Clock	74.0555	7684	2692	2492	0	2492
OPT	74.5181	7732	2644	2444	0	2444

Trend: OPT > LRU ≈ Clock > FIFO ≈ Rand

Program: matmul

memsize: 50

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	65.5165	1892047	995849	995799	956390	39409
FIFO	60.9660	1760634	1127262	1127212	1083219	43993
LRU	63.9453	1846673	1041223	1041173	1040065	1108
Clock	63.9451	1846668	1041228	1041178	1040069	1109
OPT	79.6577	2300432	587464	587414	586323	1091

memsize: 100

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	88.7860	2564046	323850	323750	316370	7380
FIFO	62.4795	1804342	1083554	1083454	1061228	22226
LRU	65.1490	1881434	1006462	1006362	1005277	1085
Clock	65.3103	1886093	1001803	1001703	1000616	1087
OPT	96.7865	2795093	92803	92703	91614	1089

memsize: 150

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	96.6634	2791539	96357	96207	93825	2382
FIFO	98.8083	2853481	34415	34265	32945	1320
LRU	98.8610	2855004	32892	32742	31658	1084
Clock	98.6037	2847573	40323	40173	39088	1085
OPT	99.0782	2861276	26620	26470	25381	1089

memsize: 200

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	98.0477	2831517	56379	56179	54548	1631
FIFO	98.8263	2854002	33894	33694	32435	1259
LRU	98.8615	2855016	32880	32680	31596	1084
Clock	98.8610	2855002	32894	32694	31610	1084
OPT	99.3327	2868626	19270	19070	17981	1089

Trend:

- **Memsize ≈ 50, 100:** OPT > Rand > LRU ≈ Clock > FIFO
- **Memsize ≈ 150, 200:** OPT > LRU ≈ Clock ≈ FIFO ≈ Rand

Program: blocked

memsize: 50

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	99.6618	2409822	8178	8128	5643	2485
FIFO	99.7342	2411574	6426	6376	4102	2274
LRU	99.7877	2412866	5134	5084	2747	2337
Clock	99.7627	2412262	5738	5688	3248	2440
OPT	99.8469	2414299	3701	3651	2562	1089

memsize: 100

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	99.7840	2412777	5223	5123	3400	1723
FIFO	99.8216	2413686	4314	4214	2731	1483
LRU	99.8432	2414208	3792	3692	2606	1086
Clock	99.8284	2413850	4150	4050	2613	1437
OPT	99.8758	2414996	3004	2904	1831	1073

memsize: 150

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	99.8199	2413644	4356	4206	2735	1471
FIFO	99.8257	2413785	4215	4065	2638	1427
LRU	99.8439	2414225	3775	3625	2560	1065
Clock	99.8434	2414213	3787	3637	2572	1065
OPT	99.8954	2415471	2529	2379	1296	1083

memsize: 200

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	99.8403	2414138	3862	3662	2294	1368
FIFO	99.8689	2414831	3169	2969	1867	1102
LRU	99.8469	2414298	3702	3502	2437	1065
Clock	99.8673	2414792	3208	3008	1943	1065
OPT	99.9057	2415721	2279	2079	1009	1070

Trend:

- **Memsize ≈ 50, 100:** OPT > LRU ≈ Clock ≈ FIFO > Rand
- **Memsize ≈ 150, 200:** OPT > LRU ≈ Clock ≈ FIFO ≈ Rand

Program: hashtable

memsize: 50

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	92.9881	6896	520	470	169	301
FIFO	93.4331	6929	487	437	135	302
LRU	94.7276	7025	391	341	72	269
Clock	94.5928	7015	401	351	85	266
OPT	96.0895	7126	290	240	14	226

memsize: 100

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	95.4693	7080	336	236	22	214
FIFO	95.4423	7078	338	238	10	228
LRU	95.6580	7094	322	7637	221	7416
Clock	95.5771	7088	328	228	5	223
OPT	96.3997	7149	267	167	0	167

memsize: 150

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	95.7389	7100	316	166	8	158
FIFO	95.6041	7090	326	176	1	175
LRU	95.7120	7098	318	168	0	168
Clock	95.6715	7095	321	171	0	171
OPT	96.3997	7149	267	117	0	117

memsize: 200

	Hit Rate	Hit Count	Miss Count	Total Evictions	Clean Evictions	Dirty Evictions
Rand	96.2513	7138	278	78	0	78
FIFO	95.8064	7105	311	111	0	111
LRU	96.1839	7133	283	83	0	83
Clock	95.9008	7112	304	104	0	104
OPT	96.3997	7149	267	67	0	67

Trend:

- **Memsize ≈ 50, 100:** OPT > LRU ≈ Clock > FIFO > Rand
- **Memsize ≈ 150, 200:** OPT > LRU ≈ Clock ≈ FIFO ≈ Rand

Algorithm Comparison:

In general, we see a trend of OPT > LRU ≈ Clock > FIFO in order of descending hit rates (always either OPT > LRU ≈ Clock > FIFO or OPT > LRU ≈ Clock ≈ FIFO), with Rand being roughly equal to FIFO or worse in most cases (with matmul being the exception). There are no cases in which any algorithm outperformed OPT, as expected, since OPT is proven to be theoretically optimal (assuming correct implementation). The hierarchy of the remaining algorithms aligns with our theoretical expectations as well. Since Clock approximates LRU by checking the usage of pages in the order they were paged in, it makes sense that the two are very close in performance, with LRU edging ahead slightly most of the time. LRU and Clock are both expected to perform better than FIFO on average, by taking greater advantage of temporal locality in their measures for predicting the future (LRU > Clock > FIFO > Rand). This relationship doesn't fully align with our results, however, since different programs have varying degrees of temporal locality. We can see that the general trend breaks down the most in matmul. The instructions in the loops in Get_matrices, Mat_mult, and Print_matrix access memory locations that are dependent on the innermost loop counter (with the exception of C[i*n + j].value in the innermost loop in Mat_mult()) in their context. Since these counters increment on each iteration, the locations accessed are close together spatially than temporally. In fact, for each row, we take the dot product with the columns of the other matrix in the same order every time, if memsize is insufficient to hold an entire matrix, the temporal algorithms will prioritize the eviction of elements more likely to be accessed sooner (the further back an element in the second matrix is accessed, the sooner it will be accessed again). This allows Rand to beat the other algorithms for low memsizes in matmult.

LRU Description:

Table of hit rates for LRU:

Program\Memsize	50	100	150	200
simpleloop	73.1496	74.0266	74.0652	74.0652
matmul	63.9453	65.1490	98.8610	98.8615
blocked	99.7877	99.8432	99.8439	99.8469
hashtable	94.7276	95.6580	95.7120	96.1839

In general, we can see that the performance for LRU increases as memsize increases. As memsize increases, more page frames are stored at any particular time. Since LRU always evicts the least recently used page, the pages in memory comprise a set of the most recently used frames. With a larger memsize, we have more pages, which means more recently used pages that are more likely to be accessed sooner (to a greater or lesser degree depending on the temporal locality of the memory accesses in the program). However, we can see that LRU does not particularly benefit from increased memsize more than the other algorithms, which also improve the accuracy of their measures. LRU also benefits from having more pages in memory at a time, decreasing the probability of faults since any page access is more likely to be present in memory in a similar way to the other algorithms. While the most noticeable jump is the large increase in hit rate from memsize 100 to 150 in matmul, this is not

LRU specific, as all the other algorithms jump to a similar hit rate as well. Perhaps more interesting here is the apparent limit on the hit rate of LRU in simpleloop, which remains exactly the same from memsize 150 to 200. This suggests that for simpleloop, 150 page frames is sufficient for LRU to achieve its optimal result (best possible result for LRU). This may be due to the breakpoint for being able to store the entire array in memory at once being between 100 and 150. The LRU algorithm will then actually store the entire array. Since it always evicts the least recently used page, when the array starts paging in, all the non-array pages existing in memory will be evicted before any pages holding array elements are. If memsize is large enough, the entire array of krecs will be paged in to memory upon being allocated. Then, since the rest of the program is simply iterating through this array, we will not encounter any more faults for the remainder of the program.