HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING
BBM204 PROGRAMMING LAB. II
ASSIGNMENT #2

Subject : Binary Search Trees and Heaps
Submission date: 09.04.2013
Deadline: April 22, 2013
Programming Language: ANSI C (89)
Advisors: R.A. Pelin Aktaş, Dr. Erkut Erdem

1. Introduction/Aim

In this experiment, you will practice using binary search tree and heap data structures. You are expected to design and implement a program that will work like a very basic cache memory. The algorithm that you are expected to implement will be a simpler version of the Least Recently Used (LRU) replacement algorithm, which is one of the well known cache algorithms.

2. Background Information

2.1. Cache Memory

Cache memory is an extremely fast memory that is built into a computer's central processing unit (CPU), or located next to it on a separate chip (see Figure 1). The CPU uses cache memory to store instructions that are repeatedly required to run programs, improving overall system speed. The advantage of cache memory is that the CPU does not have to use the motherboard's system bus for data transfer. Whenever data must be passed through the system bus, the data transfer speed slows to the motherboard's capability. The CPU can process data much faster by avoiding the bottleneck created by the system bus.

Cache hit means that the data is found in cache. When the data is found in cache, the data transfer has the maximum speed. Cache miss means that the data is not found in cache. In that case, processor loads the data from main memory and copies into cache. This results in extra delay, called miss penalty.

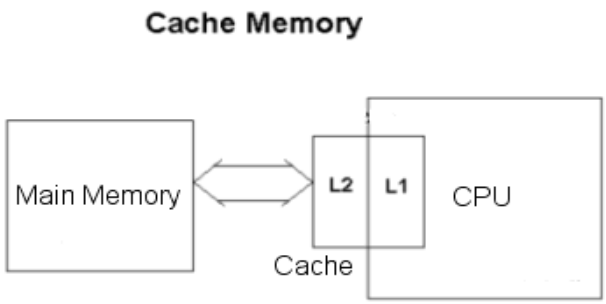

Figure 1: Structure of cache memories

2.2. Replacement Algorithm

When a cache memory is full and the data you have to read is not in the cache memory, then you have to delete an item to make room for the new item. The main problem regarding the memory replacement is deciding which section is to be deleted. The LRU replacement algorithm discards the least recently used items first. This algorithm requires keeping track of what was used when, which is expensive if one wants to make sure that the algorithm always discards the least recently used item. **In order to recognize the least recently used item, each item has to have a sequence number (seq no).**

Imagine we have a cache memory that can have 4 items maximum and we have the following order of items that are read from the memory:

A	B	C	D	A	B	E	A	B	C	D	E

The structure of the cache memory flows as given in Figure 2. Once the cache memory is full (after having ‘A’, ‘B’, ‘C’, ‘D’), to make room for the new item ‘E’ the least recently used item ‘C’ is removed from the cache memory and replaced with ‘E’. The same process is repeated when ‘C’, ‘D’ and ‘E’ are read.

A	A	A	A	A	A	A	A	A	A	A	E
	B	B	B	B	B	B	B	B	B	B	B
		C	C	C	C	E	E	E	E	D	D
			D	D	D	D	D	D	C	C	C

Figure 2: Structure of Least recently used replacement algorithm

3. Experiment

In this experiment, you are expected to design a simpler version of the LRU algorithm. Assume that you have a 2-level cache memory. Let the first cache be L0 and let the second level cache be L1. Imagine that L0 has 3 sections and L1 has 7 sections. The CPU wants to read a stream of data items via memories.

L0 Cache

L1 Cache

Main Memory

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

The data stream and Sequence numbers

A	C	A	B	K	C	D	P	Y	B	P	E	T	C
1	2	3	4	5	6	7	8	9	10	11	12	13	14

Figure 3: An example

3.1. Remarks about the Experiment

- The data must be read from L0 initially. If the data item does not exist in L0, it must be read from L1. Finally, if the data item still does not exist in L1, it must be read from the main memory.
- Once a data item is read from the main memory, first it should be inserted into L1 and then it should be inserted into L0.
- For both L0 and L1 a binary search tree and a heap have to be created.
- Binary search trees will be used for searching for the data item that will be read.
- Heap data structure will be used for the replacement algorithm to decide which data item to remove from the cache to insert a new data item.
- Reading cost for L0 is 10t, for L1 it is 20t, for the main memory reading cost is 50t (travelling cost between tree levels is 2t **except the main memory**).
- The costs of deleting nodes and inserting nodes into the binary search trees and heaps will be neglected.

4. Scenario Samples

For example, CPU wants to read the data ‘A’.

Scenario 1:

- Check L0, if A is not there, check L1,
- If A is not in both caches and both have empty section,
  - Duplicate A to L1 cache from main memory then duplicate it to L0 from L1.
  - Adjust the heaps according to new value of A’s sequence number.

Scenario 2:

- Check L0, if A is not there, check L1,
- If A is not in both caches and L0 full and L1 has empty room,
  - Duplicate A to L1’s suitable room (according to binary search tree)
  - You have to delete a node from L0 due to LRU algorithm and duplicate A to L0’s suitable room (according to the binary search tree).
  - Adjust the heaps according to the new value of A’s sequence number.

Scenario 3:

- Check L0, if A is not there, check L1,
- If A is not in both caches and both caches are full,
  - You have to delete a node from L1 due to LRU algorithm and duplicate A to L1’s suitable room (according to binary search tree).
  - Set binary heap trees according to new A’s sequence no
  - You have to delete a node from L0 due to LRU algorithm and duplicate A to L0’s suitable room (according to binary search tree).
  - Adjust the heaps according to the new value of A’s sequence number.

Scenario 4:

- Check L0, if A is not there, check L1.
- If L1 has A and L0 is full,
  - You have to delete a node from L0 due to LRU algorithm and duplicate A to L0’s suitable room (according to the binary search tree).
  - Adjust the heaps according to the new value of A’s sequence number.

Scenario 5:

- Check L0, if L0 has A
  - Adjust heap trees according to the new value of A’s sequence number.

5. Commands

Set Cache Size: In order to set cache size use this command

Command: **Set** <tree_id> <node_count>

Read Node: Find node wherever it is and read it from L0. Return total cost of reading.

Command: **Read** <node->data>

Output: Cost of the reading <node->data>, <node->cost>

List of tree: This command returns all nodes in L0 according to inorder traversal.

Command: **ListOnTree** <tree_id>

Output: <tree_nodes-->data>

Exit: Output the last stuation of caches and close the system

Command: **exit**

Output: <tree0_nodes-->data> <tree1_nodes-->data>

6. Input and Output File (A Example)

An example is given in Figure 3.

Input	**Output**
Set L0 3	L0 has 3 nodes
Set L1 7	L1 has 7 nodes
Read A	Cost of reading A is 80t { L0=10t + L1=20t + main memory=50t -->80t }
Read C	Cost of reading C is 80Tt {L0=10t + L0's 0.level = 0*2t + L1=20t + main memory = 50t -->80t }
Read A	Cost of reading A is 10t { L0 = 10t }
Read B	Cost of reading B is 82t {L0=10t + L0's 1.level = 1*2t + L1=20t + L1's 1.level=1*2t + main memory = 50t -->84t}
Read K	Cost of reading K is 84t {{L0=10t + L0's 2.level = 2*2t + L1=20t + L1's 2.level=2*2t + main memory = 50t -->88t}
Read C	Cost of reading C is 36t { L0=10 + L0's 2.level=2*2t + L1=20 + L1's 1.level=1*2t --> 36t }
Read D	Cost of reading D is 88t { L0=10t + L0's 2.level = 2*2t + L1= 20t + L1's 2.level = 2*2t + main memory = 50t --->88t}
Read P	Cost of reading P is 90t { L0=10t + L0's 2.level = 2*2t + L1= 20t + L1's 3.level = 3*2t + main memory = 50t --->90t}

Read Y	Cost of reading Y is 86t { L0=10t + L0's 2.level = 2*2t + L1= 20t + L1's 3.level = 3*2t + main memory = 50t --->90t}
Read B	Cost of reading B is 24t{ L0=10t + L0's 2.level = 2*2t + L1= 20t + L1's 2.level = 2*2t --->38t}
Read P	Cost of reading P is 10t { L0=10t + L0's 0.level = 0*2t -->10t }
ListOfTree 0	B,P,Y
ListOfTree 1	A,B,C,D,K,P,Y
Read E	Cost of reading E is 90t{ L0=10t + L0's 1.level = 1*2t + L1= 20t + L1's 4.level = 4*2t + main memory = 50t --->90t}
ListOfTree 0	B,E,P
LİstOfTree 1	B,C,D,E,K,P,Y
Read T	Cost of reading T is 90t
LİstOfTree 0	E,P,T
ListOfTree 1	B,C,D,E,P,T,Y
Read C	Cost of reading C is 32t { L0=10t + L0's 1.level = 1*2t + L1= 20t + L1's 0.level = 0*2t --->32t}
ListOfTree 0	C,E,T
ListOfTree 1	B,C,D,E,P,T,Y
exit	C,E,T B,C,D,E,P,T,Y

Input format:	**Desired output format:**
Set L0 3	L0 3
Set L1 7	L1 7
Read A	80
Read C	80
Read A	10
Read B	84
Read K	88
Read C	36
Read D	88
Read P	90
Read Y	90
Read B	38
Read P	10
ListOfTree 0	B,P,Y
ListOfTree 1	A,B,C,D,K,P,Y
Read E	90
ListOfTree 0	B,E,P
ListOfTree 1	B,C,D,E,K,P,Y
Read T	90
ListOfTree 0	E,P,T
ListOfTree 1	B,C,D,E,P,T,Y
Read C	32
exit	C,E,T B,C,D,E,P,T,Y

7. Submission Format

exp1.zip/ (*Required*)
report/; (*Required*)
report/*.pdf; (*Required*)
src/;(*Required*)
src/Makefile; (*Required*)
src/main.c; (*Required*)
src/*.c; (*Optional*)
src/*.h; (*Optional*)

8. Evaluation

- Your application will be executed by a Linux script, and evaluated automatically. It is important to apply the rules of output format. **Misformatted outputs will not be evaluated. There will be no toleration in evaluation process.** The format file and sample file given below.
- You have to design binary search tree structure.
- You can save the caches in stacks
- You are supposed to compare sequential search to binary search
- You have to explain the algorithm detailed in your experiment report. Describe your work with your own sentences as a pseudo code in a 1-2 pages report.
- Your work have to be compiled in reference system: dev.cs.hacettepe.edu.tr. Application should take input and output files as arguments.

9. Notes and Restrictions

1- Your experiment must be submitted before the due date. Late submissions will be penalized. Providing at most 3 late days, for each late day, your submission is evaluated with % 10 points less.

2- All work on assignments must be done individually unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudo code) will not be tolerated.

In short, turning in someone else's work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.

10. References

[1]http://www.mathcs.emory.edu/~cheung/Courses/355/Syllabus/9-virtual-mem/LRU-replace.html
[2] http://www.wisegeek.org/what-is-cache-memory.htm
[3] http://en.wikipedia.org/wiki/Cache_algorithms