15-122 : Principles of Imperative Computation

Summer 1 2012

Assignment 4

(Theory Part)

Due: Monday, June 11, 2012 in class

Name: _____

Andrew ID: _____

Recitation: _____

The written portion of this weeks homework will give you some practice working with amortized analysis, memory management, hashtables, and recursion. You can either type up your solutions or write them *neatly* by hand in the spaces provided. You should submit your work in class on the due date just before lecture or recitation begins. Please remember to *staple* your written homework before submission.

Question	Points	Score
1	4	
2	2	
3	4	
4	6	
5	4	
Total:	20	

1. **Hash Tables.**

In Java, strings are hashed using the following function:

$$(s[0] * 31^{n-1} + s[1] * 31^{n-2} + ... + s[n-2] * 31^1 + s[n-1] * 31^0) \ \% \ m$$

where $s[i]$ is the ASCII code for the ith character of string s, n is the length of the string, and m is the hash table size.

(2) (a) If 15122 strings were stored in a hash table of size 4401, what would the load factor of the table be?

Solution:

(2) (b) Using the hash function above with a table size of 4401, give an example of two strings that would "collide" and would be stored in the same chain. Show your work. (Think of short strings please!)

Solution:

(2) 2. **Invariant of Hashtables.** In lecture, we wrote a hash table specification function, `is_ht`, to check that a given `ht` is actually a hash table.

```
bool is_ht(ht H)
{
    if (H == NULL) return false;
    if (!(H->size > 0)) return false;
    return true;
}
```

This specification function is incomplete. Extend `is_ht` from above, implementing code to check that every element of a chain in the table hashes to the index of that chain.

Solution:

3. **Amortized Analysis.**

(2) (a) There are n ECE students who want to get into the Gates-Hillman building after 6pm. Unfortunately 122 TAs control the building and charge a toll for entrance. The toll policy is the following: the i-th student is charged k^2 when $i \equiv 0 \bmod k$, or zero otherwise. What is the amortized cost *per student* for entering the building? Assume that $n > k$. Explain your answer.

Solution:

(2) (b) Let us recall that when we analyze the amortized cost we are averaging the running time of an algorithm over a *worst-case* sequence of executions. In this task you don't need to compute the amortized cost but rather provide an example of such sequence.

Consider a linked list of items, where searching for k-th element takes $O(k)$ cost (this is the cost of traversing the list from the head to the kth location in the list). We shall use n to denote the number of items in the list. Our goal is to find a simple rule for updating the list (by performing exchanges) that will make the total cost of a sequence of n operations as small as possible. Famous Fred Hacker suggests the following rule: after accessing an item in the list, exchange it with the immediately preceding item. He claims that such technique will improve the amortized cost of single searching to $O(1)$. To prove him wrong, you need to create a counterexample, namely, find a sequence of n search operations such that the amortized cost of a single search (under Hacker's rule) will be linear. Explain your answer.

Solution:

4. **Memory Management.**

Consider the following two implementations of a stack. The first implementation uses a linked list to create the stack data structure

```
struct list_stack {
  struct list_node* top;
  struct list_node* bottom;
};

struct list_node {
  char data;
  struct list_node* next;
};
```

and the second implementation uses an unbounded array:

```
struct uba_stack {
    struct ubarray* top;
};

struct ubarray {
  int limit;        /* limit > 0 */
  int size;         /* 0 <= size && size <= limit */
  char[] elems;     /* \length(elems) == limit */
};
```

Recall that in C_0,

- a `char` is represented using 8 bits
- a pointer is represented using 64 bits
- an `int` is represented using 32 bits

An array of characters of size n takes exactly $8n + 64$ bits, where $8n$ is for the characters and 32 bits each are for two `int`s: one to record the length of the array and one to record the size of its elements.

The size of a struct is the sum of the sizes of its elements.

Consider a program that uses a stack of characters and can allocate only 1 MB (2^{20} bytes, or 8×2^{20} bits) of memory for this stack. Answer the questions below.

(3) (a) What is the maximum number of characters that can be stored using a `list_stack`? Show your work.

Solution:

(3) (b) What is the maximum number of characters that can be stored using a `uba_stack`? Show your work.

Solution:

(4) 5. **Recursion.** Recursive functions can be used to construct data structures of recursive type, like linked lists. For the following exercise, you assume a linked list defined by

```
typedef struct list_node* list;
struct list_node {
  int data;
  list next;
};
```

Write a recursive function `deleteAll` matching the following specification:

```
list deleteAll(int key, list start, list end)
```

The function deletes all occurrences of `key` in the given linked list between `start` and `end` nodes inclusively. If deletion cannot be performed (e.g., the item to delete is not in the list), it should leave the list unchanged.

Solution: