Hash Lab COSC 3319 Fall 2019 Burris

Due: Monday December 2 for MWF classes and Tuesday December 3 for TTH classes.

For your convenience a file of 200+ random words and names (Words200D16) has been provided on Blackboard. Each line of text in the file is exactly 16 characters in length with the word left justified. You may utilize any code developed in class directly in the lab. While not initially apparent, this lab will be more of a thought problem than coding problem. A careful examination of the sample code utilized in class will reveal a great deal of the code required for the lab has already been developed in class. Indeed, with so many examples*, I assume you should be able to complete the “A” option code with little trouble. Hence grading will emphasize evaluation and presentation of results for correctly coded programs.* ***Be professional grade!***

**"C" Option:**

Assume a main memory hash table of size 128. For convenience, assume the characters in the key are numbered 1 through 16. The hash address must be calculated as follows using Wallgol:

HA = {abs[ slice(4..5) ] + abs[slice(13..14) ] } / 65,535 + abs[slice(10..10)]

All alphabetic/numeric characters in a key are left justified. Keys containing less than 16 alphabetic/numeric characters are padded with spaces on the right to a total of 16 characters. For example, it the string contains the value “ABCDEFGHIJKLMNO” then the hash address would be HA = {abs[ “DE” ] + abs[ “MN” ] } / 65536 + abs[“J”]. I have never used this function but anticipate it will be substantially less than optimal generating a plethora of collisions. *All characters must be treated as positive integers in all steps of the calculation*!

I recommend using a long\_integer to hold the intermediate results of the hash calculation. Remember a “slice” in Ada allows the user to treat a substring as a single unit. Given str has the value “ABCD” str(2..3) refers to the characters in positions 2 through 3 as a group from the string str, i.e., ”BC.” “str(2..2) would be the second character in the string. I am assuming you have instantiated appropriate functions (coercion) which converts character strings(8 bits per ASCII character) to a long integer (32 bits) so overflow will not occur in the sum.

**Each entry placed in the hash table must be a record with 3 fields. The first field is the key, the second the initial hash address, and the third field is the number of probes to insert the key in the table counting the initial hash address as the first probe.**

A) Create a hash table in main memory and fill it 50% full. Use the linear probe technique developed in class to handle collisions. Now look up the first 30 entries placed in the table. Print the minimum, maximum, and average number of probes required to locate the first 30 keys placed in the table. Now search for the last 30 keys placed in the table. Print the minimum, maximum, and average number of probes required to locate the last 30 keys placed in the table. Print the contents of the hash table clearly indicating open entries (this should allow you to see the primary/secondary clustering effect). Calculate and print the theoretical expected number of probes to locate a random item in the table. Explain your empirical results in light of the theoretical results. **Your grade points will be highly dependent on your explanation!**

B) Create a hash table in main memory and fill it 90% full. Use the linear probe technique developed in class to handle collisions. Now look up the first 30 entries placed in the table. Print the minimum, maximum, and average number of probes required to locate the first 30 keys placed in the table. Now search for the last 30 keys placed in the table. YOU MUST PHYSICALLY SEARCH FOR EACH KEY TO CALCULATE THE STATISTICS! Print the minimum, maximum, and average number of probes required to locate the last 30 keys placed in the table. You must start filling the table with the same keys in the same order as used when only filling the table 40% full. Print the contents of the table clearly indicating open entries (this should allow you to see the primary clustering effect). Calculate and print the theoretical expected number of probes to locate a random item in the table. Discuss your empirical results in light of the theoretical results. **Your grade points will be highly dependent on your explanation!**

C) Repeat A and B above but use the random probe for handling collisions as developed in class. When you print the contents of the table you should be able to visually see the secondary clustering effect. Calculate and print the theoretical expected number of probes to locate a random item in the table. Discuss your empirical results in light of the theoretical results. **Your grade points will be highly dependent on your explanation!**

D) ***Present all your results in a neat tabular format (typed naturally). Compare your results to the theoretical results. If there are differences, please explain why***. **I expect you to physically calculate the theoretical values for comparison to your empirical results for both the linear and random probes**! You may not ask a friend, you may not look them up in some table! Providing someone else with the results will not only cause them to fail the lab but you as well. You may however teach them to use a calculator, spreadsheet, log tables, or other techniques.

E) My required hash function has multiple weaknesses. Criticize the hash function on a technical basis. **To receive full credit, you must state clearly and explicitly state why my hash function should fail!** Based on your criticism, write a better hash function. **You must explain explicitly why your hash function should be better from both a theoretical and empirical standpoint. You will not receive credit for simply writing a hash function that performs better**. Implement the hash function and generate the same results as required for parts A through D presented in tabular format for comparison. Formally evaluate the results as part of your lab (typed evaluation). ***The results for all parts of the lab should appear in a single table***. Shame on you if your hash function’s performance does not exceed the theoretical values for both the linear and random probe collision handling techniques. Script Kiddies may have trouble evaluating the failures of my hash function and creating a better function.

**"B" Option:**

You do not need to complete the "C" option using a main memory table. Rather, complete the "C" option **using a random access (relative) file for the hash table**. The **hash table must never appear in main memory**! Be sure to address all parts of the “C” option.

A relative file is preferred for this option. You may however use a commercial database product accessed from your “C,” C++, C#, Java or Ada program if desired rather than a relative file. If a database is utilized, the primary key must be the hash address calculated as specified above.

**"A" Option:**

Do both the "C" and "B" options. **Professionally compare and explain the results.**

**Words:**

1234567890123456

Aguirrie

Alcantara

Bhandari

Carmona

Casper

Cook

Daniels

Nienberg

Paschal

Red

Salkowski

Zulfiqar

Qamruddin

Acevedo

Ajose

Arauza

Buck

Clark

Crouch

Davies

Dugger

Egbe

Ellington

Farral

Garza

Gurung

Joseph

Kelly

Corey

Adam

Clayton

Dustin

Robert

Kyle

Scott

Octavio

Judy

Derek

Jeffrey

Jordon

Vinnela

Lisa

Todd

Veronica

Matthew

Michael

Akhila

John

Charles

James

Chris

Wade

Christopher

Fernando

Batbold

Joel

Fabulous

Misogamist

Maiden

Eye

Constriction

Necromancer

Syncopate

Yolk

Afterwards

Person

Northwest

Irreversible

Fabricate

Honor

Staple

Under

Jutty

Finagle

Cook

Rush

Wine

Screen

Perfect

mole

parasympathetic

poison

brutalize

cap

ratiocination

cauldron

prepossess

wince

orthodontist

live

magnetic

inlet

constrain

marsupial

rationalize

scat

toluene

wet

sparse

quandary

dactyl

nosegay

option

forgetful

privilege

sponsor

exhilarate

guard

noggin

prologue

seal

seat

tiller

ichthyosaur

lazy

malfeasance

compass

diastase

emperor

history

keep

gangway

labial

sacrament

taint

withal

oak

preordain

haberdasher

crimp

luculent

pennywort

prevalent

monolith

suffrage

wiper

zygote

academy

amputate

brash

consummation

epicure

indulgent

minute

job

hypodermic

meridian

sinister

tag

uterus

perennial

humidity

gynecology

forego

interchangeable

needle

reach

urban

nonunion

jingle

feculent

deciduous

channel

decimate

alarm

perimeter

unlucky

wonderful

script

particularity

mousse

issue

graduation

neutrality

proxy

swine

witticism

parish

sickly

way

periodic

mail

close

bash

conclusion

drive

foreclose

habituation

liberty

pall

recite

empress

impetuous

ruminate

fork

horizon

prerequisite

romance

sensual

story

tuck

imperialist

great

excite

east

conclave

beast

calico

beggar

article

enemy

chuck

critical

ghost

dispense

figure

algorithm

bile

handsome

cat

ivy

impersonal

pennant

elephant

glycerin

ink

squid

pumpkin

lexicon

mediocrity

meet

alabaster

morning

calculator

speaker

pacify

print

condition

popcorn

spank

moon

quadrant

earthquake

revive

sculpture

misbehave

skeleton

ligament

statute

divisible

document

infant

divot

conspiracy

bunt

notice

The hash address must be calculated as follows using Wallgol:

HA = {abs[ slice(4..5) ] + abs[slice(13..14) ] } / 65,535 + abs[slice(10..10)]

For example, it the string contains the value “ABCDEFGHIJKLMNO” then the hash address would be

HA = {abs[ “DE” ] + abs[ “MN” ] } / 65536 + abs[“J”].

= {abs[6869] + abs[7778]}/65536 + abs[75]

= 14647/65536 + 75

= 75