資料結構與程式設計

(Data Structure and Programming)

104 學年上學期複選必修課程 901 31900

Homework #7 (Due: 9:00pm, Tuesday, Dec 22, 2015)

0. Objectives

1. Implement a task manager using heap and hash.

2. Get more insights about how different data structures affect the performance of "insert", "delete", and "find" operations.

1. Problem Description

In this homework, we are going to implement a task manager with the following usage:

taskMgr [**-File** <*dofile*>]

where the **bold words** indicate the command name or required entries, square brackets "[ ]" indicate optional arguments, and angle brackets "< >" indicate required arguments. Do not type the square or angle brackets.

This taskMgr program should provide the following functionalities:

1. A task manager (a global variable "*`taskMgr`*" of `class TaskMgr`) to supervise various task nodes (class `TaskNode`). A task node stores the information of the machine that executes the task, which includes the machine name and its accumulated workload. *All the task nodes in the task manager should have distinct machine names.*

2. A hash (`class HashSet<TaskNode>`) to look up the task node by machine name, and a heap (`class MinHeap<TaskNode>`) to record and dynamically update the minimum task node (i.e. the one with the minimum accumulated load). *Please note that the task nodes stored in hash and in heap should be in sync all the time.*

3. To properly allocate the number of buckets for the hash and reserve the memory for the heap, there should be a command (TASKInit) to initialize

the task manager. In addition, there should be commands to insert new task nodes to and remove existing task node from the task manager (TASKNew and TASKRemove, respectively). Note that both TASKNew and TASKRemove commands can only be evoked after the task manager is initialized (TASKInit).

4. There should be a command (TASKQuery) to query and print the task node(s).

5. There should be a command (TASKAssign) to assign new task(s) with specified workload to the task node(s) of the minimum accumulated load(s).

2. Supported Commands

In this homework, you should support these new commands:

```
TASKAssign: Assign load to the minimum task node(s)
TASKInit:   Initialize task manager
TASKNew:    Add new task nodes
TASKQuery:  Query task manager
TASKRemove: Remove existing task nodes
```

Please refer to Homework #3 and #4 for the lexicographic notations.

2.1 Command "TASKInit"

Usage: **TASKInit <(size_t numMachines)>**

Description: Initialize the task manager "*taskMgr*". The option "*numMachines*" is the approximated number of task machines (i.e. task nodes) that you expect to add later. It will initialize the hash with the *getHashSize(nMachines)* (note: defined in "*util/util.cpp*") number of buckets and reserve the memory for the heap (i.e. capacity = *numMachines*). Note that no task nodes will be created or added. They should be added by the TASKNew command. If the global task manager "*taskMgr*" is initially NOT NULL, it will be deleted and reconstructed. A warning message "`Warning: Deleting task manager...` " will be issued in such case.

Example:

```
task> taski 100      // initialize with number of approximated task nodes = 100
task> taski          // Error: missing number of task nodes
task> taski 100 200  // Error: extra argument (200)
```

2.2 Command "TASKNew"

Usage: **TASKNew <-Random (size_t numMachines) |
-Name (string name) (size_t
load)>**

Description: Insert new task nodes to the task manager and print out "Task node inserted: (name, load)" for each newly added one. It can be randomly generated with specified number of machines or manually created with assigned machine name and workload. For random generation, exactly "*numMachines*" number of task nodes should be created. That is, if a node with duplicated name is generated, it should be discarded and regenerated. On the other hand, the "-Name" option manually creates ONE machine at a time. If the machine name collides with some task node in the task manager, an error message "`Error: Task node (`*`name`*`) already exists.`" will be issued and no task node will be added.

Example:

task> taskn –r 10 // randomly generate 10 distinct task nodes

task> taskn –n abcde 123 // manually insert the task node ("abcde", 123)

task> taskn –n abc 123 // manually insert the task node ("abc", 123)

2.3 Command "TASKRemove"

Usage: **TASKRemove <-Random (size_t numMachines)
|
-Name (string name)>**

Description: Remove task node(s) from the task manager and print out "Task node removed: (name, load)" for each removed one. It can be randomly removed with specified number of machines or manually removed with assigned machine name. For each of the random removal, a random number i between 0 and s-1 (s is the number of total task nodes) is first generated and the i^th^ element of the heap will be removed (by MinHeap::delData(i)). The corresponding task node in hash should then be removed, too. Note that after each removal, the heap structure should be properly maintained. For the manual removal, task node with the specified name in both heap and hash will be removed. If there is no task node with such name, an error message "`Error: Task node (`*`name`*`) does not exist.`" will be issued.

Example:

task> taskr –r 10 // randomly remove 10 task nodes

task> taskr –n abcde // manually remove the task node with name "abcde"

2.4 Command "TASKQuery"

Usage: **TASKQuery <(string name) | -HAsh | -HEap | -MINimum >**

Description: Query and print the task node in the task manager. If the machine name is specified, query the node with that name and print it. If there is no such node, print out an error message "`Query fails!`". If the option "-HAsh" or "-HEap" is specified, print all task nodes with the order as stored in the hash and heap, respectively. Lastly, the option "-MINimum" will print out the one with the minimum workload.

Example:

task> taskq abcdefg // query the task node with name = "abcdefg"

task> taskq –hash // print out all task nodes with the order as stored in hash

task> taskq –min // print out the node with minimum workload

2.5 Command "TASKAssign"

Usage: **TASKAssign <(size_t load)> [-Repeat (size_t repeats)]**

Description: Assign the workload to the task node with the minimum accumulated workload. The option "-Repeat (size_t repeats)" specifies how many times the same workload will be repeatedly assigned. Note that every time the workload is assigned, the accumulated workload of the minimum task node will be incremented by "*load*" and the minimum task node (in the heap) will be updated accordingly. Therefore, it is likely that the repeated workload will be assigned to different task nodes.

Example:

task> taska –r 100 1000 // assign workload 1000 to the min node for 100 times

3. What you should do?

You are encouraged to follow the steps below for this homework assignment:

1. Read the specification carefully and make sure you understand the requirements.

2. Think first how you are going to write the program, assuming you don't have the reference code.

3. Type "make 32", "make 64" or "make mac" to switch to a proper platform. The default is 64-bit platform.

4. Play with the reference programs in the “ref” directory to make sure you understand the spec.

5. Implement the TODOs in “*myHashSet.h*” and “*myMinHeap.h*” in the “*util*” package. Write some test programs to test them first.

6. Finish the TODOs in “taskMgr.cpp” for the task manager commands.

7. Test your implementation with the provided testcases. You are also encouraged to create more testcases on your own.

4. Grading

We will test your submitted programs with various testcases and compare the outputs with our reference program. Minor differences due to printing alignment, spacing, error message, etc can be tolerated. However, to assist TAs for easier grading work, please try to match your output with ours.