CS 5381 Analysis of Algorithms

Rattikorn Hewett

Assignment 3
Due date: November 3, 2016

A priority queue for a simplified agenda manager in a rule-based expert system shell

The two basic elements of a rule-based expert system shell are a *knowledge base* and an *inference engine*. The knowledge base contains the domain knowledge in the form of *rules*. The inference engine uses these rules to produce a solution to a given problem.

The inference engine operates in cycles. In each cycle, it first determines which rules can be activated for firing (called *activated* rules), then selects **one** rule for execution and finally executes the selected rule. Each activated rule has a corresponding priority. An agenda contains a list of activated rules, with their priorities, maintained by an agenda manager. In each cycle, an agenda manager updates list of activated rules by adding new activated rules and deleting rules that are no longer activated and the rule that was executed in a previous cycle.

Your assignment is to implement a simplified agenda manager. In each cycle, the agenda manager first deletes from the agenda the activated rule that was executed in a previous cycle. A list of new activated rules is given to the agenda manager to add to the agenda. Once a rule is added to the agenda, it remains there until it is executed. An agenda manager then determines the rule with highest priority for the inference engine to execute in this cycle. Only one rule can be executed in each cycle. A rule that has been executed and removed from the agenda may be put on the agenda again in the next cycle. The system halts when the agenda is empty or after it runs 30 cycles.

Your implementation must use a priority queue implemented by using a heap. For simplicity, assume that the agenda can contain at most 300 rules (although a system may have more than 300 rules). Each rule is represented by a character string of length 5 and a priority of range between 1 and 100.

The agenda manager takes a file named **input.txt** as input. Each line of the file is a list of ordered pairs (x, y) where x is a rule and y is a priority. The agenda manager builds a priority queue (heap) from the rules and priorities in the first line of the file. It then outputs the rule with the highest priority and removes it from the queue. Each subsequent line of the input file is used as the list of rules to be added in another cycle.

Example of input file:

(arule, 12), (brule, 21), (zrule, 70), (drule, 25), (erule, 10)
(frule, 3)
(grule, 20), (srule, 100)

The actions of the agenda manager for this file are:
Cycle 1: Build a priority queue (heap) containing the 5 rules with corresponding priorities as in line 1. Output zrule and remove it from the queue.
Cycle 2: Add frule with priority 3 to queue. Output drule and remove it from the queue.
Cycle 3: Add grule and srule with respective priorities 20 and 100 to the queue. Output srule and remove it from the queue.
Cycles 4-8: Output and remove from the queue rules brule, grule, arule, erule, and frule.

Your agenda manager must include the following operations:

BuildQueue: builds a priority queue (heap) from an unsorted array of rules with priorities
Heapify: used by BuildQueue to maintain the heap properly
ExtractMax: returns the rule with the highest priority
Insert: adds a new rule to the queue
Delete: removes a rule from the queue

What to submit:
Part 1: Results obtained from the runs against each of the given tests. The results should show content of the agenda in each reasoning cycle and clearly state which cycle the inference terminates.
Part 2: Experiment to see the run time performance of your simplified agenda manager for different problem sizes. Show the graph of the above result and relate it to the asymptotic behavior of the worst case run time.
Part 3: Intelligent and concise documentation.

Grading:
Your program will be evaluated based on results of the program execution when tested with given input files. The testing files will be made available in the submission week. If your program terminates normally with correct outputs for all tests, your code and documentation will be examined and scored. Otherwise, your code and documentation will not be graded and you may improve your score by making correction and resubmitting your work one more time within a week after the assignment is returned.

Plagiarism and academic irregularities: (see syllabus) will not be tolerated and could result in failing the course. No code sharing in any degree.