

CS5234 : Combinatorial and Graph Algorithms
 Homework Set #2
 [Algorithm Paradigm, Priority Queues, Amortized Analysis]

OUT: 28-Aug-2007**DUE:** 11-Sep-2007**Course Web-Site:** <http://www.comp.nus.edu.sg/~cs5234/2007-08/>**IMPORTANT NOTE: Read “Remarks about Homework”.****Solve all the S-Problems in every homework set.**

- Start each of the problems on a separate sheet of paper.
- Make sure your name and matric number is on each sheet.
- To hand the homework in, **staple them together** and hand to me during class or drop them into the CS5234 envelope outside room (COM1 03-41) by the due date.

When asked to “give an algorithm” to solve a problem, your write-up should NOT be just the code. Instead, it should be a short essay. The first paragraph should summarize the problem and what your results are. The body of the essay should consist of the following:

- A description of the algorithm in English and, if helpful, pseudo-code.
- At least one worked example or diagram to show more precisely how your algorithm works.
- A proof (or indication) of the correctness of the algorithm
- An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions which are described clearly. Convoluted and obtuse descriptions will receive low marks.

Routine Practice Problems -- do not turn these in -- but make sure you know how to do them.

R1. [Modelling] You are a guidance counselor in charge of putting high school students into one of two study halls. It doesn't matter how many students are in each study hall; what does matter is that certain pairs of students do not get along well and would cause a major disruption if they were placed in the same study hall. There are n students and you have a list of m pairs of students who shouldn't be placed together. Give an algorithm that determines in time $O(n + m)$ whether it is possible to allocate the students to the two study halls without violating the m constraints. If it is possible to perform such an allocation, your algorithm should produce it. (Note that some students may occur in multiple times in the list of “bad” pairs, but no student would be paired with him/herself.)

R2. [d -heap and Dijkstra's Algorithm]

(a) Prove that, for optimum performance of Dijkstra's shortest path algorithm (on a graph with n nodes and m edges) using a d -heap implementation, we should choose $d = O(m/n)$.

(b) You can verify that this choice is indeed better for certain ranges of m by giving the corresponding speed-up in the algorithm for each m . For example, try

$$m = 4n, \quad m = n \lg n, \quad m = n^{1.2}, \quad m = n^{1.5}$$

R3. Exercises 17.1-1, 17.1-2, 17.1-3 on pp.409-410 of [CLRS01] (Simple Amortization)

Standard-Problems -- solve these problems and turn them in by the due-date.
S1. [Optimal Book Storage]

A book depot wants optimal storage of books that minimizes the storage cost. Suppose that we know the heights and thicknesses of all the books in a collection (assuming that all widths fit on the same shelving, we consider only a two-dimensional problem and ignore book width). Suppose that we have arranged the book heights in ascending order of their n known heights H_1, H_2, \dots, H_n ; that is $H_1 < H_2 < \dots < H_n$. Since we know the thicknesses of the books, we can compute the required length of shelving for each height class. Let L_i denote the length of shelving for books of height H_i . If we order shelves of height H_i for length x_i , we incur cost equal to $F_i + C_i \cdot x_i$, where F_i is a fixed ordering cost (and is independent of the length ordered) and C_i is the cost of the shelf per unit length. Notice that in order to save the fixed cost of ordering, we might not order shelves of every possible height because we can use a shelf of height H_i to store books of smaller heights. We want to determine the length of shelving for each height class that would minimize the total cost of the shelving. Formulate this problem as a shortest path problem.

S2. [Exam-Taking Strategies]

Consider (if you haven't already!) an exam with n questions. For each $i = 1, \dots, n$, question i has integral point value $v_i > 0$ and requires $m_i > 0$ minutes to solve. Suppose further that no partial credit is awarded (unlike my exam).

Your goal is to come up with an algorithm which, given $v_1, v_2, \dots, v_n, m_1, m_2, \dots, m_n$, and V , computes the minimum number of minutes required to earn at least V points on the exam. For example, you can use this algorithm to see how quickly you can get an A on the exam.

(a) Let $M(i, v)$ denote the minimum number of minutes needed to earn v points when you are restricted to selecting from questions 1 through i . Give a recurrence for $M(i, v)$.

(We shall do the base cases for you: for all i , and $v \leq 0$, $M(i, v) = 0$; for $v > 0$, $M(0, v) = \infty$.)

(b) Give pseudocode for an $O(nV)$ -time dynamic programming algorithm to compute the minimum number of minutes required to earn V points on the exam.

(c) Explain how to extend your solution from the previous part to output a list S of questions to solve, such that $V \leq \sum_{i \in S} v_i$ and $\sum_{i \in S} m_i$ is minimized.

(d) Suppose partial credit is given, so that the number of points you receive on a question is proportional to the number of minutes you spend working on it. That is, you earn v_i/m_i points per minute on question i (up to a total of v_i points), and you can work for fractions of minutes. Given an $O(n \lg n)$ -time algorithm to determine which questions to solve (and how much time to devote to them) in order to receive V points the fastest.

S3. [Binary Heaps] Each *delete_min* operation in a binary heap uses $2 \lg n$ comparisons and $\lg n$ data-moves in the worst-case.

(a) Propose a scheme so that the *delete_min* operation uses $\lg n + \lg \lg n + O(1)$ **comparisons** between elements. (**Hint:** Observe that the “*restore path*” is sorted!)

(b) Comment about the implications and/or practicality of this result.

[If you need references on binary heaps, you can read [CLRS]-C6 on priority queues. The difference is they implement a “*max-heap*” instead of our “*min-heap*”.]

S4. Modified from Exercise 17.3-6 of [CLRS] [Queue using two stacks]

This exercise deals with implementing a queue with two ordinary stacks (S_1 and S_2) so that the amortized cost of each ENQUEUE and each DEQUEUE operation is $O(1)$. You can assume that each PUSH(S, x) and POP(S) operation on a stack S costs 1 unit of work.

(a) Give pseudo-code (in style similar to that used in [CLRS]) for ENQUEUE(Q, x) and DEQUEUE(Q) operations. (You can use the PUSH and POP stack operations.)

(b) Suppose that (starting from an empty queue) we do 4 ENQUEUES, then 3 DEQUEUES, then 3 more ENQUEUES, and then 2 more DEQUEUES. What is the total (actual) cost of these 12 operations, and how many elements are in each stack at the end?

(c) Suppose a total of n ENQUEUES and n DEQUEUES are done in some order. What is the worst-case running time of *one* of these operations (give an exact, non-asymptotic answer)? Give a sequence of operations that induces this worst-case behaviour, and indicate which operation has this worst-case running time you have specified.

(d) Suppose we perform an arbitrary sequence of ENQUEUES and DEQUEUES (starting from an empty queue). Give a tight (non-asymptotic) amortized cost of each operation using the accounting method.

(e) Now do the same as part (d) using the potential method.

Advanced Problems -- Try these for challenge and fun. There is no deadline for A-problems.

A2. [Shortest-Path on Special Graphs]

In this problem, we consider special cases of directed graphs with nonnegative edge weights satisfying the triangle inequality, for which the single-source shortest-path problem can be solved faster.

(a) Suppose that all edge weights are integers between 0 and C for some constant C . Give an $O(V+E)$ -time algorithm to solve the single-source shortest-path problem in such a graph.

An interesting property of Dijkstra's algorithm is that it does not require a general priority-queue data structure; instead, it merely requires what is called a monotone priority queue. A monotone priority queue supports the following operations on a dynamic set S :

1. DELETE-MIN(S): Delete and return the current minimum value in S .
2. INSERT(S, x): Insert an element x . However, for the operation to be allowed, x must be larger than or equal to the last value returned by DELETE-MIN.

(b) Design a monotone priority queue that maintains a dynamic set of elements from the universe $\{0, \dots, u-1\}$. INSERT should run in $O(1)$ time. The total time spent by DELETE-MIN over a sequence of k operations should be $O(u + k)$.

(c) Consider a graph in which all shortest paths have integer weights between 0 and $u-1$. Describe how to use your monotone priority queue in Dijkstra's algorithm, and analyze the running time of the resulting algorithm for the single-source shortest-path problem. State your bound in terms of V , E , and u .