DOWNLOAD PDF DAA NOTES

Chapter 1 : Design and Analysis of Computer Algorithms

Design and Analysis of Algorithm Notes pdf - DAA notes pdf file Design and Analysis of Algorithm Notes pdf - DAA pdf notes - DAA notes pdf file to download are listed below please check it - Latest Material Links.

Recursive algorithms An algorithm is said to be recursive re if the same algorithm is invokedd iin the body direct recursive. Algorithm A is said aid to be indirect recursive if it calls another er algorithm which in turn calls A. Factorial computa tation n! Binomial coefficie cient computation Example 3: Tower of Hanoii problem pr Example 4: Analysis Framework General framework for analy lyzing the efficiency of algorithms is discuss ssed here. There are two kinds of efficiency: Time efficie iciency indicates how fast an algorithm in questionn runs; space efficiency deals with the extra space s the algorithm requires. In the early days of electronic ic computing, both resources time and space were w at a premium. Now the amount of extra spac ace required by an algorithm is typically nott oof as much concern, In addition, the research exper perience has shown that for most problems, we can achieve much more spectacular progress in speed than in space. Therefore, followingg a well-established tradition of algorithm textbook oks, we primarily concentrate on time efficienc ency. For Fo example, it takes longer to sort larger arrays,, multiply m larger matrices, and so on. There are situations, where the choice of a parameter indicating an input ut size does matter. The choice of an appropriate te size metric can be influenced by operationss of the algorithm in question. If the algorithm examines individual characters of its inpu put, then we should measure the size by the numb mber of characters; if it works by processing ng words, we should count their number in the inpu put. We should make a special note no about measuring the size of inputs for algorithms alg involving properties of numbers e. This metric usually gives a better ideaa aabout the efficiency of algorithms in question. The thing to do is to identify the most impportant operation of the algorithm, called the bas asic operation, the operation contributing the most to the total running time, and compute the number of times the basic operation is execu ecuted. For example, most sorting algorithms al work by comparing elements keys ke of a list being sorted with each other; for suc uch algorithms, the basic operation is a key com omparison. As another example, algorith rithms for matrix multiplication and polyn lynomial evaluation require two arithmetic operatio tions: Prerpared by Harivinod N www. Then we can estimate the runningg time T n of a program implementing this is algorithm on that computer by the formula: Values of several functions important for analysis of algorithms Algorithms that require an exponential ex number of operations are practica tical for solving only problems of very small sizes. The worst-casee efficiency e of an algorithm is its efficiencyy for the worst-case input of size n, for which thee algorithm a runs the longest among all possible le inputs of that size. Consider the algorithm for seq equential search. Prerpared by Harivinod N Page 1. Introdu troduction The running time of above algorithm al can be quite different for the same me list size n. In the worst case, when there are no matching elements or the first matching element el happens to be the last one on the list,, the algorithm makes the largest number of key comparisons among all possible inputs off size si n: Thu hus it guarantees that for any instance of size n, n the running time will not exceed Cworst n , itss running ru time on the worst-case inputs. The best-case efficiency eff of an algorithm is its efficiency for or the best-case input of size n, for which the algorit rithm runs the fastest among all possible inputs uts of that size. We determine the kind of inputs inp for which the count C n will be thee ssmallest among all possible inputs of size n. The analysis of the best-case se efficiency is not nearly as important as tha hat of the worst-case efficiency. This information is provided by ave verage-case efficiency. Let us consider again sequenti ntial search. We can findd tthe average number of key comparisons Cavg n as follows. In the case of an unsuccessful searc arch, the number of comparisons is n with the prob robability of such a search being 1- p. Therefo efore, Investigation of the average-ccase efficiency is considerably more difficult ult than investigation of the worst-case and best-ca case efficiencies. But there are many import ortant algorithms for which the average case effic ficiency is much better than the overly pess essimistic worst-case efficiency would lead us to believe. Space Sp efficiency is measured by counting the th number of extra memory units consumed by b the algorithm. The efficiencies of somee algorithms a may differ significantly for input uts of the same size. For such algorithms, wee need n to

distinguish between the worst-case, se, average-case, and best-case efficiencies. Performance Analysis lysis 2. This inclu cludes memory space for codes, variables, constants co and so on. A variable part that depends de on the input, output and recursion sta stack. Consider followin wing algorithm abc Here fixed component depend nds on the size of a, b and c. Let us considerr the th algorithm to find sum of array. For the algorithm given here re the problem instances are characterized by n, the number of elements to be summed. Thee space s needed by a[ ] depends on n. This is the sum of the time ta taken to execute all instructions in the program. Exact estimation runtime iss a complex task, as the number of instru truction executed is dependent on the input data. Also A different instructions will take differentt ttime to execute. So for the estimation of the timee complexity c we count only the number of program pr steps. A program step is loosely defined de as syntactically or semantically meani aning segment of the program that has and execut cution time that is independent of instance characteristics. We can determine the stepss needed n by a program to solve a particularr pproblem instance in two ways. In the first method we introd oduce a new variable count to the program whhich is initialized to zero. We also introduce state atements to increment count by an appropriat iate amount into the program. So when each timee original program executes, the count alsoo incremented i by the step count. Consider the algo lgorithm sum. After the introduction of the he count the program will be as follows. The second method to determ rmine the step count of an algorithm is to bui uild a table in which we list the total number of step teps contributed by each statement. An example ple is shown below. Prerpared by Harivinod N.

Chapter 2 : DAA Notes Module 1 - PDF Free Download

Design And Analysis Of Algorithm, DAA Notes For exam preparations, pdf free download Classroom notes, Engineering exam notes, previous year questions for Engineering, PDF free download.

Since multiplications are more ex- pensive than additions, let us count the cost of multiplications only. Here, we have to keep track of the lengths of the entries of the matrix. So the running time of computing Fn using Method 3 is dependent on the multipli- cation algorithm. Well, multiplication is multiplication â€" what can we do about it? Before that let us summarize what we know about it. Multiplying two n digit numbers using the add-and-shift method takes O n2 steps where each step involves multiplying two single digits bits in the case of binary representation , and generat- ing and managing carries. For binary representation this takes O n for multiplying with each bit and finally n shifted summands are added â€" the whole process takes O n2 steps. Using such a method of multiplication implies that we cannot do better than n2 steps to compute Fn. For any significant asymptotically better improvement, we must find a way to multiply faster. What is maximum size of the numbers involved in subtraction? The running time is roughly O n1. It is possible to multiply much faster using a generalization of the above method in O n log n log log n by a method of Schonage and Strassen. However it is quite in- volved as it uses Discrete Fourier Transform computation over modulo integer rings and has fairly large constants that neutralize the advantage of the asymptotic im- provement unless the numbers are a few thousand bits long. Despite architectural variations, the assembly level language sup- port is very similar â€" the major difference being in the number of registers and the word length of the machine. But these parameters are also in a restricted range of a factor of two, and hence asymptotically in the same ball park. In summary, think about any computer as a machine that supports a basic instruction set consisting of arithmetic and logical operations and memory accesses including indirect addressing. We will avoid cumbersome details of the exact instruction set and assume realistically that any instruction of one machine can be simulated using a constant number of available instruction of another machine. Since analysis of algorithms involves counting the number of operations and not the exact timings which could differ by an order of magnitude , the above simplification is justified. The careful reader would have noticed that during our detailed analysis of Method 3 in the previous sections, we were not simply counting the number of arithmetic operations but actually the number of bit-level operations. Therefore the cost of a multiplication or addition was not unity but proportional to the length of the input. Had we only counted the number of multiplications for computing xn, that would only be O log n. This would indeed be the analysis in a uniform cost model where only the number of arithmetic also logical operations are counted and does not depend on the length of the operands. A very common us of this model is for comparison-based problems like sorting, selection, merging, and many data-structure operations. For these problems, we often count only the number of comparisons not even other arithmetic operations without bothering about the length of the operands involved. In other words, we implicitly assume O 1 cost for any comparison. This is not considered unreasonable since the size of the numbers involved in sorting do not increase during the course of the algorithm for majority of the commonly known sorting problems. On the other hand consider the following problem of repeated squaring n times starting with 2. The resultant is a number 22n which requires 2n bits to be represented. It will be very unreasonable to assume that a number that is exponentially long can be written out or even stored in O n time. Therefore the uniform cost model will not reflect any realistic setting for this problem. On the other extreme is the logarithmic cost model where the cost of an operation is proportional to length of the operands. This is very consistent with the physical world and also has close relation with the Turing Machine model which is a favorite of complexity theorists. Our analysis in the previous sections is actually done with this model in mind. It is not only the arithmetic operations but also the cost of memory access is proportional to the length of the address and the operand. We assume that for an input of size n, any operation involving operands of size log n 2 takes O 1 steps. This is justified as follows. All microprocessor chips have specialized hardware circuits for arithmetic operations like multiplication, addition, division etc. The reason that log n is a natural choice for a word is that, even to address an input size n, you require log n

bits of address space. We will also use this model, popularly known as Random Access Machine or RAM in short except for problems that deal with numbers as inputs like multiplication in the previous section where we will invoke the log cost model. In the beginning, it is desirable that for any algorithm, you get an estimate of the maximum size of the numbers to ensure that operands do not exceed log n so that it is safe to use the RAM model. There is clear trade-off between the simplicity and the fidelity achieved by an abstract model. One of the obvious and sometimes serious drawback of the RAM model is the assumption of unbounded number of registers since the memory access cost is uniform. In reality, there is a memory hierarchy comprising of registers, several levels of cache, main memory and finally the disks. We incur a higher access cost as we go from registers towards the disk and for techological reason, the size of the faster memory is limited. There could be a disparity of between the fastest andthe slowest memory which makes the RAM model somewhat suspect for larger input sizes. This has been redressed by the external memory model. Given the rather high cost of a disk access compared to any CPU operation, this model actually ignores all other costs and counts only the number of disk accesses. The disk is accessed as contiguous memory locations called blocks. The blocks have a fixed size B and the simplest model is parameterized by B and the size of the faster memory M. In this two level model, the algorithms are only charged for transferring a block between the internal and external memory and all other computation is free. As the model becomes more complicated, designing algorithms also becomes more challenging and often more laborious. At the most intuitive level it symbolises what can be achieved by cooperation among individuals in terms of expediting an activity. It is not in terms of division of labor or specialisation , but actually assuming similar capabil- ities. Putting more labourers clearly speeds up the construction and similarly using more than one processor is likely to speed up computation. Ideally, by using p pro- cessors we would like to obtain a p-fold speed up over the conventional algorithms; however the principle of decreasing marginal utility shows up. One of the intuititive reasons for this that with more processors as with more individuals , the commu- nication requirements tend to dominate after a while. But more surprisingly, there are algorithmic constraints that pose serious limitations to our objective of obtaining proportional speed-up. Here p processors are connected to a shared memory and the communication happens through reading and writing ina globally shared memory. It is left to the algorithm designer to avoid read and write conflicts. It is further assumed that all operations are synchorized globally and there is no cost of synchronization. In this model, there is no extra overhead for communi- cation as it charged in the same way as a local memory access. Even in this model, it has been shown that it is not always possible to obtain ideal speed up. As an example consider the elementary problem of finding the minimum of n elements. It has been proved that with n processors, the time parallel time is at least log log n. For certain problems, like depth first search of graphs, it is known that even if we use any polynomial number of processors, we cannot obtain polylogarithmic time! So, clearly not all problems can be parallelized effectively. A more realistic parallel model is the interconnection network model that has an underlying communication network, usually a regular topology like a two-dimensional mesh, hypercube etc. These can be embedded into VLSI chips and can be scaled according to our needs. To implement any any parallel algorithm, we have to design efficient schemes for data routing. A very common model of parallel computation is a hardware circuit comprising of basic logic gates. The signals are transmitted in parallel through different paths and the output is a function of the input. The size of the circuit is the number of gates and the parallel time is usually measured in terms of the maximum path length 12 from any input gate to the output gate each gate contributes to a unit delay. Those familiar with circuits for addition, comparison can analyse them in this framework. The carry-save adder is a low-depth circuit that adds two n-bit numbers in about O log n steps which is much faster than a sequential circuit that adds one bit at a time taking n steps. One of the most fascinating developments is the Quantum Model which is in- herently parallel but it is also fundamentally different from the previous models. A breakthrough result in recent years is a polynomial time algorithm for factorization which forms the basis of many cryptographic protocals in the conventional model. Biological Computing models is a very active area of research where scientists are trying to assemble a machine out of DNA strands. It has potentially many advantages over silicon based devices and is inherently parallel.

DOWNLOAD PDF DAA NOTES

Chapter 3 : DAA Notes - PDF Free Download

CS DAA Notes. Anna University Regulation Computer Science Engineering (CSE) CS DAA Notes for all 5 units are provided below. Download link for CSE 4th SEM CS DESIGN AND ANALYSIS OF ALGORITHMS Lecture Notes are listed down for students to make perfect utilization and score maximum marks with our study materials.

Chapter 4 : DESIGN AND ANALYSIS OF ALGORITHMS | VTU CSE NOTES

Unit Priority Queue, Heap and Heap sort,Heap Sort,Priority Queue implementation using heap tree,Binary Search trees,Balanced Trees,Dictionary,Disjoint Set Operations,Recurrence Relations - Iterative Substitution Method.

Chapter 5 : CS DAA Notes, DESIGN AND ANALYSIS OF ALGORITHMS Lecture Notes â€“ CSE 4th SEM

Analysis and Design of Algorithms, DAA Notes For exam preparations, pdf free download Classroom notes, Engineering exam notes, previous year questions for Engineering, PDF free download.

Chapter 6 : Analysis and Design of Algorithms - DAA Study Materials | PDF FREE DOWNLOAD

DAA Notes In this context you will find out a list of DAA notes. Some notes are Handwritten notes and some are the documents that are collected.

Chapter 7 : DAA â€“ LECTURE NOTES | Harsh Divya

This DAA Study Material and DAA Notes & Book has covered every single topic which is essential for calendrierdelascience.com BE Students. Design and Analysis of Algorithms Study Materials provided here is specifically prepared for JNTUH JNTUK JNTUA R13, R10, R09 Students but all other University students can also download it as it has covered every single.

Chapter 8 : B-TechStuff: DAA Complete notes

DAA Notes. Third Year of Computer Engineering (Course) Design and Analysis of Algorithms(DAA) Unit-I. DAA Unit-I. Unit-II. Final DAA Unit-II. Unit-III.

Chapter 9 : DAA Notes | Akshay Jain

Note that if a set is represented by a list, depending on the application ap at hand, it might be worth maint intaining the list in a sorted order. Dictionary: In computing, the th operations we need to perform for a set et or a multiset most often are searching for a given giv item, adding a new item, and deleting ng an item from the collection.