|  |  |
| --- | --- |
| **UNIT-III**  **Algorithms, Induction and Recursion**  **Algorithm**  An algorithm is a finite set of instructions, those if followed, accomplishes a particular task. It is not language specific, we can use any  language and symbols to represent instructions.  **The criteria of an algorithm**  **Input:** Zero or more inputs are externally supplied to the algorithm.  **Output:** At least one output is produced by an algorithm.  **Definiteness:** Each instruction is clear and unambiguous.  **Finiteness:**In an algorithm, it will be terminated after a finite number of steps for all different cases.  **Effectiveness:** Each instruction must be very basic, so the purpose of those instructions must be very clear to us.  **Analysis of algorithms**  Algorithm analysis is an important part of computational complexities. The complexity theory provides the theoretical estimates for the resources needed by an algorithm to solve any computational task. Analysis of the algorithm is the process of analyzing the problem-solving capability of the algorithm in terms of the time and size required (the size of memory for storage while implementation). However, the main concern of the analysis of the algorithm is the required time or performance.  **Complexities of an Algorithm**  The complexity of an algorithm computes the amount of time and spaces required by an algorithm for an input of size (n). The complexity of an algorithm can be divided into two types. The **time**  **complexity** and the **space complexity**.  Time Complexity of an Algorithm  The time complexity is defined as the process of determining a formula for total time required towards the execution of that algorithm. This calculation is totally independent of implementation and programming language.  Space Complexity of an Algorithm  Space complexity is defining as the process of defining a formula for prediction of how much memory space is required for the successful execution of the algorithm. The memory space is generally considered as the primary memory.  **Mathematical induction**:  **Mathematical induction**, is a technique for proving results or establishing statements for natural numbers. This part illustrates the method through a variety of examples.  Definition  Mathematical Induction is a mathematical technique which is used to prove a statement, a formula or a theorem is true for every natural number.  The technique involves two steps to prove a statement, as stated below −  Step 1(Base step) − It proves that a statement is true for the initial value.  Step 2(Inductive step) − It proves that if the statement is true for the nth iteration (or number *n*), then it is also true for *(n+1)th* iteration ( or number *n+1*).  How to Do It  Step 1 − Consider an initial value for which the statement is true. It is to be shown that the statement is true for n = initial value.  Step 2 − Assume the statement is true for any value of *n = k*. Then prove the statement is true for *n = k+1*. We actually break *n = k+1* into two parts, one part is *n = k* (which is already proved) and try to prove the other part.  **Problem 1**  3n−13n−1 is a multiple of 2 for n = 1, 2, ...  Solution  Step 1 − For n=1,31−1=3−1=2n=1,31−1=3−1=2 which is a multiple of 2  Step 2 − Let us assume 3n−13n−1 is true for n=kn=k, Hence, 3k−13k−1 is true (It is an assumption)  We have to prove that 3k+1−13k+1−1 is also a multiple of 2  3k+1−1=3×3k−1=(2×3k)+(3k−1)3k+1−1=3×3k−1=(2×3k)+(3k−1)  The first part (2×3k)(2×3k) is certain to be a multiple of 2 and the second part (3k−1)(3k−1) is also true as our previous assumption.  Hence, 3k+1–13k+1–1 is a multiple of 2.  So, it is proved that 3n–13n–1 is a multiple of 2.  **Problem 2**  1+3+5+...+(2n−1)=n21+3+5+...+(2n−1)=n2 for n=1,2,…n=1,2,…  Solution  Step 1 − For n=1,1=12n=1,1=12, Hence, step 1 is satisfied.  Step 2 − Let us assume the statement is true for n=kn=k.  Hence, 1+3+5+⋯+(2k−1)=k21+3+5+⋯+(2k−1)=k2 is true (It is an assumption)  We have to prove that 1+3+5+...+(2(k+1)−1)=(k+1)21+3+5+...+(2(k+1)−1)=(k+1)2 also holds  1+3+5+⋯+(2(k+1)−1)1+3+5+⋯+(2(k+1)−1)  =1+3+5+⋯+(2k+2−1)=1+3+5+⋯+(2k+2−1)  =1+3+5+⋯+(2k+1)=1+3+5+⋯+(2k+1)  =1+3+5+⋯+(2k−1)+(2k+1)=1+3+5+⋯+(2k−1)+(2k+1)  =k2+(2k+1)=k2+(2k+1)  =(k+1)2=(k+1)2  So, 1+3+5+⋯+(2(k+1)−1)=(k+1)21+3+5+⋯+(2(k+1)−1)=(k+1)2 hold which satisfies the step 2.  Hence, 1+3+5+⋯+(2n−1)=n21+3+5+⋯+(2n−1)=n2 is proved.  **Problem 3**  Prove that (ab)n=anbn(ab)n=anbn is true for every natural number nn  Solution  Step 1 − For n=1,(ab)1=a1b1=abn=1,(ab)1=a1b1=ab, Hence, step 1 is satisfied.  Step 2 − Let us assume the statement is true for n=kn=k, Hence, (ab)k=akbk(ab)k=akbk is true (It is an assumption).  We have to prove that (ab)k+1=ak+1bk+1(ab)k+1=ak+1bk+1 also hold  Given, (ab)k=akbk(ab)k=akbk  Or, (ab)k(ab)=(akbk)(ab)(ab)k(ab)=(akbk)(ab) [Multiplying both side by 'ab']  Or, (ab)k+1=(aak)(bbk)(ab)k+1=(aak)(bbk)  Or, (ab)k+1=(ak+1bk+1)(ab)k+1=(ak+1bk+1)  Hence, step 2 is proved.  So, (ab)n=anbn(ab)n=anbn is true for every natural number n.  **Strong Induction**  Strong Induction is another form of mathematical induction. Through this induction technique, we can prove that a propositional function, P(n)P(n) is true for all positive integers, nn, using the following steps −  Step 1(Base step) − It proves that the initial proposition P(1)P(1) true.  Step 2(Inductive step) − It proves that the conditional statement [P(1)∧P(2)∧P(3)∧⋯∧P(k)]→P(k+1)[P(1)∧P(2)∧P(3)∧⋯∧P(k)]→P(k+1) is true for positive integers kk | |
|  |  |

|  |  |
| --- | --- |
| Definition. Suppose n and k are nonnegative integers. A recurrence relation of the form c0(n)an  + c1(n)an-1 + …. + ck(n)an-k = f(n) for n ≥ k, where c0(n), c1(n),…., ck(n), and f(n) are functions of n is said to be a linear recurrence relation. If c0(n) and ck(n) are not identically zero, then it is said to be a linear recurrence relation *degree* k. If c0(n), c1(n),…., ck(n) are constants, then the recurrence relation is known as a linear relation with constant coefficients. If f(n) is identically zero, then the recurrence relation is said to be homogeneous; otherwise, it is inhomogeneous.  Thus, all the examples above are linear recurrence relations except (8), (9), and (10); the relation (8), for instance, is not linear because of the squared term.  The relations in (3), (4) , (5), and (7) are linear with constant coefficients.  Relations (1), (2), and (3) have degree 1; (4), (5), and (6) have degree 2; (7) has degree   1. Relations (3) , (4), and (6) are homogeneous.   There are no general techniques that will enable one to solve all recurrence relations. There are, nevertheless, techniques that will enable us to solve linear recurrence relations with constant coefficients.  **SOLVING RECURRENCE RELATIONS BY SUSTITUTION AND GENERATING FUNCTIONS**  We shall consider four methods of solving recurrence relations in this and the next two sections:   * 1. Substitution (also called iteration),   2. Generatingfunctions,   3. Characteristics roots,   In the substitution method the recurrence relation is used repeatedly to solve for a general expression for an in terms of n. We desire that this expression involve no other terms of the sequence except those given by boundary conditions.  The mechanics of this method are best described in terms of examples. We used this method in Example5.3.4. Let us also illustrate the method in the following examples.  **Example**  Solve the recurrence relation an = a n-1 + f(n) for n ³1 by substitution a1= a0 + f(1)  a2 = a1 + f(2) = a0 + f(1) + f(2))  a3 = a2 + f(3)= a0 + f(1) + f(2) + f(3)  .  .  . | |
| difference obtained by setting the right-hand side equal to 0, the ―associated homogeneous equation.< We know how to solve this. Say that  is a solution. Now suppose that ( ) is any particular solution of the inhomogeneous equation. (That is, it solves the equation, but does not necessarily match the initial data.) Then = +( ) is a solution to the inhomogeneous equation, which you can see simply by substituting  into the equation. On the other hand, every solution  of the inhomogeneous equation is of the form  =  +( ) where  is a solution of the homogeneous equation, and ( ) is a particular solution of the inhomogeneous equation. The proof of this is straightforward. If we have two solutions to the inhomogeneous equation, say  1 and 2, then their difference 1− 2=  is a solution to the homogeneous equation, which you can check by substitution. But then 1=  +  2, and we can set  2=( ), since by assumption,  2 is a particular solution. This leads to the following theorem: the general solution to the inhomogeneous equation is the general solution to the associated homogeneous equation, plus any particular solution to the inhomogeneous equation. This gives the following procedure for solving the inhomogeneous equation:   1. Solve the associated homogeneous equation by the method we< ve learned. This will involve variable (or undetermined) coefficients. 2. Guess a particular solution to the inhomogeneous equation. It is because of the guess that I< ve called this a procedure, not an algorithm. For simple right-hand sides , we can say how to compute a particular solution, and in these cases, the procedure merits the name ―algorithm.< 3. The general solution to the inhomogeneous equation is the sum of the answers from the two steps above. 4. Use the initial data to solve for the undetermined coefficients from step 1.   To solve the equation − 6 −1 + 8 −2 = 3. Let< s suppose that we are also given the initial data 0 = 3, 1 = 3. The associated homogeneous equation is − 6 −1 + 8 −2 = 0, so the  characteristic equation is 2 − 6 + 8 = 0, which has roots 1 = 2 and 2 = 4. Thus, the general solution to the associated homogeneous equation is 12 + 24 . When the right-hand side is a polynomial, as in this case, there will always be a particular solution that is a polynomial.  Usually, a polynomial of the same degree will work, so we< ll guess in this case that there is a constant  that solves the homogeneous equation. If that is so, then = −1 = −2 = , and substituting into the equation gives  − 6  + 8  = 3, and we find that  = 1. Now, the general solution to the inhomogeneous equations is 12 + 24 + 1. Reassuringly, this is the answer given in the back of the book. Our initial data lead to the equations 1 + 2 + 1 = 3 and 2 1 + 4 2 + 1 = 3, whose solution is 1 = 3, 2 = −1. Finally, the solution to the inhomogeneous equation, with the initial condition given, is = 3 · 2 − 4 + 1. Sometimes, a polynomial of the same degree as the right-hand side doesn< t work. This happens when the characteristic equation has 1 as a root. If our equation had been − 6 −1 + 5 −2 = 3, when we guessed that the particular solution was a constant , we< d have arrived at the equation  − 6  + 5  = 3, or 0 = 3. The way to deal with this is to increase the degree of the polynomial. Instead of assuming that the solution is constant, we< ll assume that it< s linear. In fact, we< ll guess that it is of the form | |

|  |
| --- |
| = . Then wehave −6 −1 +5 −2 =3, which simplifies to 6 −10 =3 so that  =−34 . Thus,  = −3 4 . This won< t be enough if 1 is a root of multiplicity 2, that is, if  −1 2 is a factor of the characteristic polynomial. Then there is a particular solution of the form  = 2. For second-order equations, you never have to go past this. If the right-hand side is a polynomial of degree greater than 0, then the process works juts the same, except that you start with a polynomial of the same degree, increase the degree by 1, if necessary, and then once more, if need be. For example, if the right-hand side were  =2 −1, we would start by guessing a particular solution  = 1 + 2. If it turned out that 1 was a characteristic root, we would amend our guess to  = 1 2+ 2 + 3. If 1 is a double root, this will fail also, but  = 1 3+ 2 2+ 3 + 4 will work in this case.    Another case where there is a simple way of guessing a particular solution is when the right- hand side is an exponential, say  = . In that case, we guess that a particular solution is just a constant multiple of , say ( )= . Again, we gave trouble when 1 is a characteristic root. We then guess that  = , which will fail only if 1 is a double root. In that case we must use  = 2 , which is as far as we ever have to go in the second-order case. These same ideas extend to higher-order recurrence relations, but we usually solve them numerically, rather than exactly. A third-order linear difference equation with constant coefficients leads to a cubic characteristic polynomial. There is a formula for the roots of a cubic, but it< s very complicated.  For fourth-degree polynomials, there< s also a formula, but it< s even worse. For fifth and higher degrees, no such formula exists. Even for the third-order case, the exact solution of a simple- looking inhomogeneous linear recurrence relation with constant coefficients can take pages to write down. The coefficients will be complicated expressions involving square roots and cube  roots. For most, if not all, purposes, a simpler answer with numerical coefficients is better, even though they must in the nature of things, be approximate.  The procedure I< ve suggested may strike you as silly. After all, we< ve already solved the characteristic equation, so we know whether 1 is a characteristic root, and what it< s multiplicity is. Why not start with a polynomial of the correct degree? This is all well and good, while you< re taking the course, and remember the procedure in detail. However, if you have to use this procedure some years from now, you probably won< t remember all the details. Then the method I< ve suggested will be valuable. Alternatively, you can start with a general polynomial of the maximum possible degree This leads to a lot of extra work if you< re solving by hand, but it< s the approach I prefer for computer solution. |

## Program Correctness

## Important rules:

Below are some of the important rules for effective programming which are consequences of the program correctness theory.

* Defining the problem completely.
* Develop the algorithm and then the program logic.
* Reuse the proved models as much as possible.
* Prove the correctness of algorithms during the design phase.
* Developers should pay attention to the clarity and simplicity of your program.
* Verifying each part of a program as soon as it is developed.