tell the difference in difficulty between $NP-hard$ and NP-complete problems because the class NP includes everything easier than its "toughest" problems--if a problem is not in NP, it is harder than all the problems in NP.

Does P==NP?

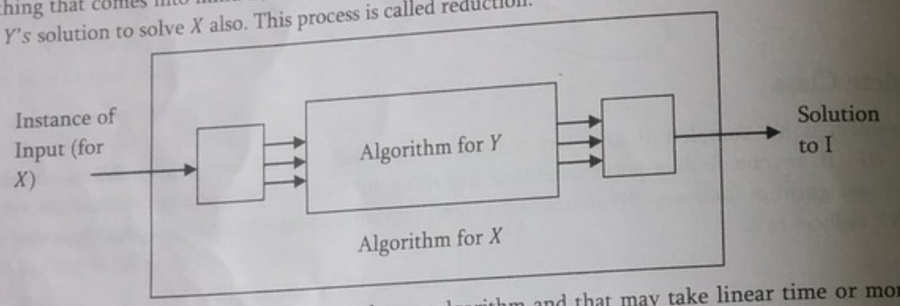
If $P = NP$, it means that every problem that can be checked quickly can be solved quickly (remember the difference between checking if an answer is right and actually solving a problem).

This is a big question (and also nobody knows the answer), because right now there are lots of NP-Complete problems that can't be solved quickly. If $P = NP$, that means there is a way to solve them fast. Remember that "quickly" means not trial-and-error. It could take a billion years, but as long as we didn't use trial and error, it was quick. In future, a computer will come that can change that billion years into a few minutes.

20.7 Reductions

Before discussing reductions, let us consider the following scenario. Assume that, we want to solve a problem X for which we do not know how to solve it and feel it's very complicated. In this case what we do?

The first thing that comes into mind is, if we have similar problem as that of X (let us say Y), then we try to map X to Y and use $Y's$ solution to solve X also. This process is called reduction.



In order to map problem X to problem Y, we need some algorithm and that may take linear time or more. Based on this discussion the cost of solving problem X can be given as:

$$Cost\ of\ solving\ X = Cost\ of\ solving\ Y + Reduction\ time$$

Now, let us consider the other scenario. For solving problem X, sometimes we may need to use $Y's$ algorithm (solution) multiple times. In that case,

$$Cost\ of\ solving\ X = Number\ of\ Times * Cost\ of\ solving\ X + Reduction\ time$$

The main important thing in NP-Complete is reducibility. That means, we reduce (or transform) given NP-Complete problems to other known NP-Complete problem. Since the NP-Complete problems are hard to solve and inorder to prove that given NP-Complete problem is hard, we take one existing hard problem (for which we know how to prove is hard) and try to map given problem to that and finally we prove that the given problem is hard.

Note: It's not compulsory to reduce the given problem to known hard problem to prove its hardness. Sometimes, we reduce the known hard problem to given problem.

Important NP-Complete Problems (Reductions)

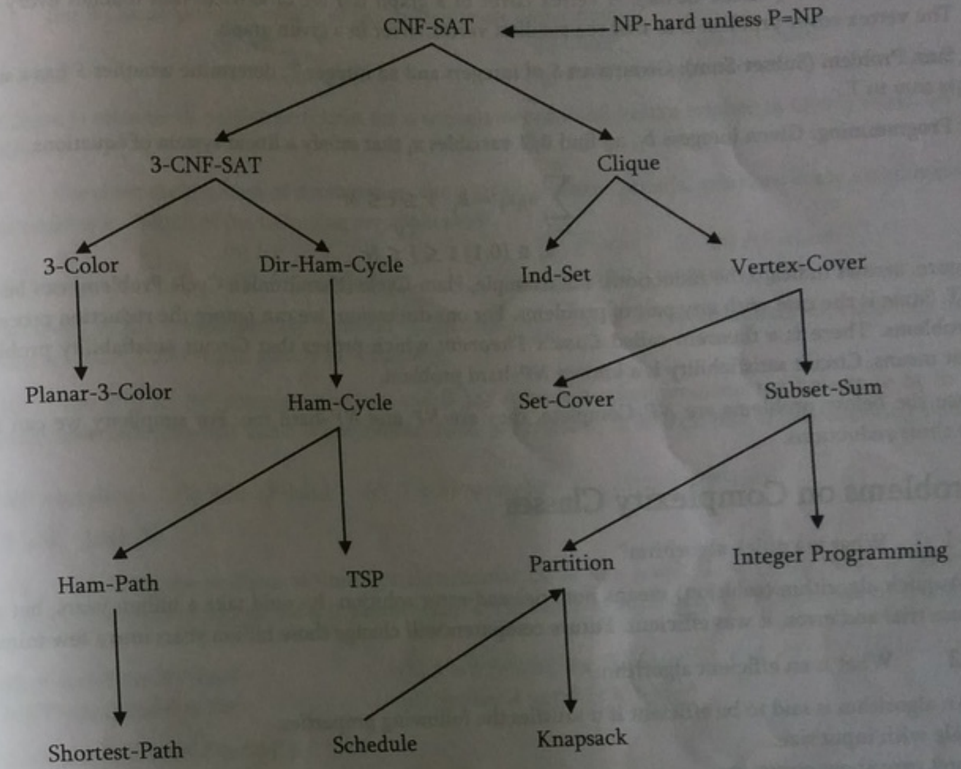
Satisfiability Problem: A boolean formula is in *conjunctive normal form* (CNF) if it is a conjunction (AND) of several clauses, each of which is the disjunction (OR) of several literals, each of which is either a variable or its negation. For example: $(a \vee b \vee c \vee d \vee e) \wedge (b \vee \sim c \vee \sim d) \wedge (\sim a \vee c \vee d) \wedge (a \vee \sim b)$

A 3-CNF formula is a CNF formula with exactly three literals per clause. The previous example is not a 3-CNF formula, since its first clause has five literals and its last clause has only two.

2-SAT Problem: 3-SAT is just SAT restricted to 3-CNF formulas: Given a 3-CNF formula, is there an assignment to the variables so that the formula evaluate to TRUE?

2-SAT Problem: 2-SAT is just SAT restricted to 2-CNF formulas: Given a 2-CNF formula, is there an assignment to the variables so that the formula evaluate to TRUE?

Circuit-Satisfiability Problem: Given a boolean combinational circuit composed of AND, OR and NOT gates, is it satisfiable?. That means, given a boolean circuit consisting of AND, OR and NOT gates properly connected by wires, the Circuit-SAT problem is to decide whether there exists an input assignment for which the output is TRUE.



Hamiltonian Path Problem (Ham-Path): Given an undirected graph, is there a path that visits every vertex exactly once?

Hamiltonian Cycle Problem (Ham-Cycle): Given an undirected graph, is there a cycle (where start and end vertices are same) that visits every vertex exactly once?

Directed Hamiltonian Cycle Problem (Dir-Ham-Cycle): Given a directed graph, is there a cycle (where start and end vertices are same) that visits every vertex exactly once?

Travelling Salesman Problem (TSP): Given a list of cities and their pair-wise distances, the problem is to find a shortest possible tour that visits each city exactly once.

Shortest Path Problem (Shortest-Path): Given a directed graph and two vertices s and t, check whether there is a shortest simple path from s to t.

Graph Coloring: A k-coloring of a graph is to map one of k 'colors' to each vertex, so that every edge has two different colors at its endpoints. The graph coloring problem is to find the smallest possible number of colors in a legal coloring.

3-Color problem: Given a graph, is it possible to color the graph with 3 colors in such a way that every edge has two different colors?

Clique (also called complete graph): Given a graph, the *CLIQUE* problem is to compute the number of nodes in its largest complete subgraph. That means, we need to find the maximum subgraph which is also a complete graph.

Independent Set Problem (Ind_Set): Let G is an arbitrary graph. An independent set in G is a subset of the vertices of G with no edges between them. The maximum independent set problem is the size of the largest independent set in a given graph.

Vertex Cover Problem (Vertex-Cover): A vertex cover of a graph is a set of vertices that touches every edge in the graph. The vertex cover problem is to find the smallest vertex cover in a given graph.

Subset Sum Problem (Subset-Sum): Given a set S of integers and an integer T, determine whether S has a subset whose elements sum to T.

Integer Programming: Given integers b_i, a_{ij} find $0/1$ variables x_i that satisfy a linear system of equations.

$$\sum_{j=1}^{N} a_{ij}x_j = b_i \quad 1 \le i \le M$$
$$x_j \in \{0,1\}\ 1 \le j \le N$$

In the figure, arrows indicate the reductions. For example, Ham-Cycle (Hamiltonian Cycle Problem) can be reduced to CNF-SAT. Same is the case with any pair of problems. For our discussion, we can ignore the reduction process for each of the problems. There is a theorem called *Cook's Theorem* which proves that Circuit satisfiability problem is NP-hard. That means, Circuit satisfiability is a known NP-hard problem.

Note: Since the below problems are NP-Complete, they are NP and NP-hard too. For simplicity we can ignore the proofs for these reductions.

20.8 Problems on Complexity Classes

Problem-1 What is a quick algorithm?

Solution: A quick algorithm (solution) means not trial-and-error solution. It could take a billion years, but as long as we didn't use trial and error, it was efficient. Future computers will change those billion years into a few minutes.

Problem-2 What is an efficient algorithm?

Solution: An algorithm is said to be efficient if it satisfies the following properties:

- Scale with input size.
- Don't care about constants.
- Asymptotic running time: polynomial time.

Problem-3 Can we solve all problems in polynomial time?

Solution: No. The answer is trivial because we have seen lot of problems which took more than polynomial time.

Problem-4 Are there any problems which are NP-hard?

Solution: By definition, NP-hard implies that it is very hard. That means, it is very hard to prove and verify that they are hard. Cook's Theorem proves that Circuit satisfiability problem is NP-hard.

Problem-5 For 2-SAT problem, which of the following are applicable?

(a) P (b) NP (c) $CoNP$ (d) NP-Hard
(e) $CoNP$-Hard (f) NP-Complete (g) $CoNP$-Complete

Solution: 2-SAT is solvable in poly-time. So it is P, NP, and $CoNP$.

Problem-6 For 3-SAT problem, which of the following are applicable?

(a) P (b) NP (c) $CoNP$ (d) NP-Hard
(e) $CoNP$-Hard (f) NP-Complete (g) $CoNP$-Complete

Solution: 3-SAT is NP-complete. So it is NP, NP-Hard, and NP-complete.

Problem-7 For 2-Clique problem, which of the following are applicable?

(a) P (b) NP (c) $CoNP$ (d) NP-Hard
(e) $CoNP$-Hard (f) NP-Complete (g) $CoNP$-Complete

Solution: 2-Clique is solvable in poly-time (check for an edge between all vertex-pairs in $O(n^2)$ time). So it is P, NP, and $CoNP$.

Problem-8 For 3-Clique problem, which of the following are applicable?

(a) P (b) NP (c) $CoNP$ (d) NP-Hard
(e) $CoNP$-Hard (f) NP-Complete (g) $CoNP$-Complete

Solution: 3-Clique is solvable in poly-time (check for a triangle between all vertex-triplets in $O(n^3)$ time). So it is P, NP, and $CoNP$.

Problem-9 Consider the problem of determining, for a given boolean formula, whether every assignment to the variables satisfies it. Which of the following are applicable?

(a) P (b) NP (c) $CoNP$ (d) NP-Hard
(e) CoNP-Hard (f) NP-Complete (g) $CoNP$-Complete

Solution: Tautology is the complimentary problem to Satisfiability, which is NP-complete, so Tautology is $CoNP$-complete. So it is $CoNP$, $CoNP$-hard, and $CoNP$-complete.

Problem-10 Let S be an NP-complete problem and Q and R be two other problems not known to be in NP. Q is polynomial time reducible to S and S is polynomial-time reducible to R. Which one of the following statements is true?

(a) R is NP-complete (b) R is NP-hard (c) Q is NP-complete (d) Q is NP-hard.

Solution: R is NP-hard (b).

Problem-11 Let A be the problem of finding a Hamiltonian cycle in a graph $G = (V, E)$, with $|V|$ divisible by 3 and B be the problem of determining if Hamiltonian cycle exists in such graphs. Which one of the following is true?

(a) Both A and B are NP-hard (b) A is NP-hard, but B is not
(c) A is NP-hard, but B is not (d) Neither A nor B is NP-hard

Solution: Both A and B are NP-hard (a).

Problem-12 Let A be a problem that belongs to the class NP. Then which one of the following is true?

(a) There is no polynomial time algorithm for A.
(b) If A can be solved deterministically in polynomial time, then $P = NP$.
(c) If A is NP-hard, then it is NP-complete.
(d) A may be undecidable.

Solution: If A is NP-hard, then it is NP-complete (c).

Problem-13 Suppose if we assume $Vertex - Cover$ is known to be NP-complete. Based on our reduction, can we say $Independent - Set$ is NP-complete?

Solution: Yes. This follows from the two conditions necessary to be NP-complete:

- Independent Set is in NP, as stated in the problem.
- A reduction from a known NP-complete problem.

Problem-14 Suppose *Independent Set* is known to be NP-complete. Based on our reduction, is *Vertex Cover* NP-complete?

Solution: No. By reduction from Vertex-Cover to Independent-Set, we do not know the difficulty of solving Independent-Set. This is because Independent-Set could still be a much harder problem than Vertex-Cover, we haven't proven that.

Chapter-21

MISCELLANEOUS CONCEPTS

21.1 Introduction

In this chapter we will cover the topics which are useful for interviews and exams.

21.2 Hacks on Bitwise Programming

In C and $C++$ we can work with bits effectively. First let us see the definitions of each bit operation and then move onto different techniques for solving the problems. Basically, there are six operators that C and $C++$ support for bit manipulation:

Symbol	Operation
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive-OR
$\ll$	Bitwise left shift
$\gg$	Bitwise right shift
~	Bitwise complement

21.2.1 Bitwise AND

The bitwise AND tests two binary numbers and returns bit values of 1 for positions where both numbers had a one, and bit values of 0 where both numbers did not have one:

```
  01001011
& 00010101
----------
  00000001
```

21.2.2 Bitwise OR

The bitwise OR tests two binary numbers and returns bit values of 1 for positions where either bit or both bits are one, the result of 0 only happens when both bits are 0:

```
  01001011
| 00010101
----------
  01011111
```

21.2.3 Bitwise Exclusive-OR

The bitwise Exclusive-OR tests two binary numbers and returns bit values of 1 for positions where both bits are different, if they are the same then the result is 0:

```
  01001011
^ 00010101
----------
  01011110
```


21.2.4 Bitwise Left Shift

The bitwise left shift moves all bits in the number to the left and fills vacated bit positions with 0.

```
  01001011
<< 2
--------
  00101100
```

21.2.5 Bitwise Right Shift

The bitwise right shift moves all bits in the number to the right.

```
  01001011
>> 2
--------
  ??010010
```

Note the use of ? for the fill bits. Where the left shift filled the vacated positions with 0, a right shift will do the same only when the value is unsigned. If the value is signed then a right shift will fill the vacated bit positions with the sign bit or 0, which one is implementation-defined. So the best option is to never right shift signed values.

21.2.6 Bitwise Complement

The bitwise complement inverts the bits in a single binary number.

```
  01001011
~
---------
  10110100
```

21.2.7 Checking Whether K-th bit is Set or Not

Let us assume that the given number is n. Then for checking the K^{th} bit we can use the expression: $n \,\&\, (1 \ll K-1)$. If the expression is true then we can say the K^{th} bit is set (that means, set to 1).

Example:

n = 01001011 and $K = 4$	
$1 \ll K-1$	00001000
$n \,\&\, (1 \ll K-1)$	00001000

21.2.8 Setting K-th bit

For a given number n, to set the K^{th} bit we can use the expression: $n \mid 1 \ll (K-1)$

Example:

n = 01001011 and $K = 3$	
$1 \ll K-1$	00000100
$n \mid (1 \ll K-1)$	01001111

21.2.9 Clearing K-th bit

To clear K^{th} bit of a given number n, we can use the expression: $n \,\&\, {\sim}(1 \ll K-1)$

Example:

n = 01001011 and $K = 4$	
$1 \ll K-1$	00001000
${\sim}(1 \ll K-1)$	11110111
$n \,\&\, {\sim}(1 \ll K-1)$	01000011

21.2.10 Toggling K-th bit

For a given number n, for toggling the K^{th} bit we can use the expression: $n \text{ ^} (1 \ll K-1)$

Example:

n =	01001011 and $K = 3$
$1 \ll K-1$	00000100
$n \text{ ^} (1 \ll K-1)$	01001111

21.2.11 Toggling Rightmost One bit

For a given number n, for toggling rightmost one bit we can use the expression: $n \,\&\, n-1$

Example:

n =	01001011
$n-1$	01001010
$n \,\&\, n-1$	01001010

21.2.12 Isolating Rightmost One bit

For a given number n, for isolating rightmost one bit we can use the expression: $n \,\&\, -n$

Example:

n =	01001011
$-n$	10110101
$n \,\&\, -n$	00000001

Note: For computing $-n$, use two's complement representation. That means, toggle all bits and add 1.

21.2.13 Isolating Rightmost Zero bit

For a given number n, for isolating rightmost zero bit we can use the expression: $\sim n \,\&\, n+1$

Example:

n =	01001011
$\sim n$	10110100
$n+1$	01001100
$\sim n \,\&\, n+1$	00000100

21.2.14 Checking Whether Number is Power of 2 or Not

Given number n, to check whether the number is in 2^n form for not, we can use the expression: $if(n \,\&\, n-1 == 0)$

Example:

n =	01001011
$n-1$	01001010
$n \,\&\, n-1$	01001010
$if(n \,\&\, n-1 == 0)$	0

21.2.15 Multiplying Number by Power of 2

For a given number n, to multiply the number with 2^K we can use the expression: $n \ll K$

Example:

n =	00001011 and $K = 2$
$n \ll K$	00101100

21.2.16 Dividing Number by Power of 2

For a given number n, to divide the number with 2^K we can use the expression: $n \gg K$

Example:

n =	00001011 and $K = 2$
$n \gg K$	00010010

21.2.17 Finding Modulo of a Given Number

For a given number n, to find the %8 we can use the expression: n & $0x7$. Similarly, to find %32, use the expression: n & $0x1F$

Note: Similarly, we can write for any modulo value.

21.2.18 Reversing the Binary Number

For a given number n, to reverse the bits (reverse (mirror) of binary number)we can use the following code snippet:

```
unsigned int n, nReverse = n;
int s = sizeof(n);
for (; n; n >>= 1) {
        nReverse <<= 1;
        nReverse |= n & 1;
        s--;
}
nReverse <<= s;
```

Time Complexity: This requires one iteration per bit and the number of iterations depends on the size of the number.

21.2.19 Counting Number of One's in Number

For a given number n, to count the number of 1's in its binary representation we can use any of the following methods.

Method1: Process bit by bit

```
unsigned int n;
unsigned int count=0;
while(n) {
        count += n & 1;
        n >>= 1;
}
```

Time Complexity: This approach requires one iteration per bit and the number of iterations depends on system.

Method2: Using modulo approach

```
unsigned int n;
unsigned int count=0;
while(n) {
        if(n%2 ==1) count++;
        n = n/2;
}
```

Time Complexity: This requires one iteration per bit and the number of iterations depends on system.

Method3: Using toggling approach: $n \,\&\, n-1$

```
unsigned int n;
unsigned int count=0;
while(n) {
        count++;
        n &= n - 1;
}
```

Time Complexity: The number of iterations depends on the number of 1 bits in the number.

Method4: Using preprocessing idea. In this method, we process the bits in groups. For example if we process them in groups of 4 bits at a time, we create a table which indicates the number of one's for each those possibilities (as shown below.)

0000→0	0100→1	1000→1	1100→2
0001→1	0101→2	1001→2	1101→3
0010→1	0110→2	1010→2	1110→3
0011→2	0111→3	1011→3	1111→4

The following code to counts the number of 1 in the number with this approach:

```
int Table = {0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4};
int count = 0;
for(; n; n >>= 4)
        count = count + Table[n & 0xF];
return count;
```

Time Complexity: This approach requires one iteration per 4 bits and the numbers of iterations depend on system.

21.2.20 Creating for Mask for Trailing Zero's

For a given number n, to create a mask for trailing zeros, we can use the expression: $(n \,\&\, -n) - 1$

Example:

$$\begin{array}{rr} n = & 01001011 \\ -n & 10110101 \\ n \,\&\, -n & 00000001 \\ (n \,\&\, -n) - 1 & 00000000 \end{array}$$

Note: In the above case we are getting the mask as all zeros because there are no trailing zeros.

21.2.21 Performing Average without Division

Is there a bit-twiddling algorithm to replace $mid = (low + high) / 2$ (used in Binary Search and Merge Sort) with something much faster?

We can use $mid = (low + high) >> 1$. Note that using $(low + high) / 2$" for midpoint calculations won't work correctly when integer overflow becomes an issue. We can use bit shifting and also overcome a possible overflow issue: $low + ((high - low)/ 2)$ and the bit shifting operation for this is $low + ((high - low) >> 1)$.

21.3 Other Programming Questions

Problem-1 Give an algorithm for printing the matrix elements in spiral order.

Solution: Non-recursive solution involves directions right, left, up, down, and dealing their corresponding indices. Once the first row is printed, direction changes (from right) to down, the row is discarded by incrementing the upper limit. Once the last column is printed, direction changes to left, the column is discarded by decrementing the right hand limit.

```
void Spiral(int **A, int n) {
  int rowStart=0, columnStart=0;
  int rowEnd=n-1, columnEnd=n-1;
  while(rowStart<=rowEnd && columnStart<=columnEnd) {
    int i=rowStart, j=columnStart;
    for(j=columnStart; j<=columnEnd; j++) printf("%d ",A[i][j]);
    for(i=rowStart+1, j--; i<=rowEnd; i++) printf("%d ",A[i][j]);
    for(j=columnEnd-1, i--; j>=columnStart; j--) printf("%d ",A[i][j]);
    for(i=rowEnd-1, j++; i>=rowStart+1; i--) printf("%d ",A[i][j]);
```



```
    rowStart++; columnStart++; rowEnd--; columnEnd--;
  }
}
```

Time Complexity: $O(n^2)$. Space Complexity: $O(1)$.

Problem-2 Give an algorithm for shuffling the desk of cards.

Solution: Assume that we want to shuffle an array of 52 cards, from 0 to 51 with no repeats, such as we might want for a deck of cards. First fill the array with the values in order, then go thru the array and exchange each element with a randomly chosen element in the range from itself to the end. It's possible that an element will be swap with itself, but there is no problem with that.

```
void Shuffle(int cards[], int n){
    srand(time(0));                    // initialize seed randomly

    for (int i=0; i<n; i++)
      cards[i] = i;                    // filling the array with card number

    for (int i=0; i<n; i++) {
      int r = i + (rand() % (52-i));   // Random remaining position.
      int temp = cards[i]; cards[i] = cards[r]; cards[r] = temp;
    }
    printf("Shuffled Cards:" );
    for (int i=0; i<n; i++)
      printf("%d ", cards[i]);
}
```

Time Complexity: $O(n)$. Space Complexity: $O(1)$.

Problem-3 Reversal algorithm for array rotation: Write a function rotate(A[], d, n) that rotates A[] of size n by d elements. For example, the array 1, 2, 3, 4, 5, 6, 7 becomes 3, 4, 5, 6, 7, 1, 2 after 2 rotations.

Solution: Consider the following algorithm.

Algorithm:

```
rotate(Array[], d, n)
reverse(Array[], 1, d) ;
reverse(Array[], d + 1, n);
reverse(Array[], l, n);
```

Let AB are the two parts of the input Arrayay where A = Array[0..d-1] and B = Array[d..n-1]. The idea of the algorithm is: Reverse A to get ArB. /* Ar is reverse of A */
Reverse B to get ArBr. /* Br is reverse of B */
Reverse all to get (ArBr) r = BA.
For example, if Array[] = [1, 2, 3, 4, 5, 6, 7], d =2 and n = 7 then, A = [1, 2] and B = [3, 4, 5, 6, 7]
Reverse A, we get ArB = [2, 1, 3, 4, 5, 6, 7], Reverse B, we get ArBr = [2, 1, 7, 6, 5, 4, 3]
Reverse all, we get (ArBr)r = [3, 4, 5, 6, 7, 1, 2]

Implementation:

```
/* Function to left rotate Array[] of size n by d */
void leftRotate(int Array[], int d, int n) {
        rvereseArrayay(Array, 0, d-1);
        rvereseArrayay(Array, d, n-1);
        rvereseArrayay(Array, 0, n-1);
}
```

```
/*UTILITY FUNCTIONS: function to print an Arrayay */
void printArrayay(int Array[], int size){
        for(int i = 0; i < size; i++)
                printf("%d ", Array[i]);
        printf("%\n ");
}
/*Function to reverse Array[] from index start to end*/
void rvereseArrayay(int Array[], int start, int end) {
        int i;
        int temp;
        while(start < end){
         temp = Array[start];
         Array[start] = Array[end];
         Array[end] = temp;
         start++;
         end--;
        }
}
```

Problem-4 Suppose you are given an array s[1...n] and a procedure reverse (s,i,j) which reverses the order of elements in a between positions i and j (both inclusive). What does the following sequence

do, where 1 < k <= n:

```
reverse (s, 1, k);
reverse (s, k + 1, n);
reverse (s, 1, n);
```

(a) Rotates s left by k positions (b) Leaves s unchanged (c) Reverses all elements of s (d) None of the above

Solution: (b). Effect of the above 3 reversals for any k is equivalent to left rotation of the array of size n by k [refer Problem-3].

Problem-5 Finding Anagrams in Dictionary: ou are given these 2 files: dictionary.txt and jumbles.txt

The jumbles.txt file contains a bunch of scrambled words. Your job is to print out those jumbles words, 1 word to a line. After each jumbled word, print a list of real dictionary words that could be formed by unscrambling the jumbled word. The dictionary words that you have to choose from are in the dictionary.txt file. Sample content of jumbles.txt:

```
nwae: wean anew wane
eslyep: sleepy
rpeoims: semipro imposer promise
ettniner: renitent
ahicryrhe: hierarchy
dica: acid cadi caid
dobol: blood
.....
%
```

Solution: Step-By-Step

Step 1: Initialization

- Open the dictionary.txt file and read the words into an array (before going on - verify the read by echoing out the words back from the array out to the screen).
- Declare a hash table variable.

Step 2: Process the Dictionary foreach dictionary word in the array do the following:

We now have a hash table where eack key is the sorted form of a dictionary word and the value associated to it is a string or array of dictionary words that sort to that same key.

- Remove the newline off the end of each word via chomp($word);
- Make a sorted copy of the word - i.e. rearrange the individual chars in the string to be sorted alphabetically
- Think of the sorted word as the key value and think of the set of all dictionary words that sort to the exact same key word as being the value of the key
- Query the hashtable to see if the sortedWord is already one of the keys
- If it is not already present then insert the sorted word as key and the unsorted original of the word as the value
- Else concat the unsorted word onto the value string already out there (put a space in between)

Step 3: Process the jumbled word file

- Read through the jumbled word file one word at a time. As you read each jumbled word chomp it and make a sorted copy (the sorted copy is your key)
- Print the unsorted jumble word
- Query the hashtable for the sorted copy. If found, print the associated value on same line as key and a then newline.

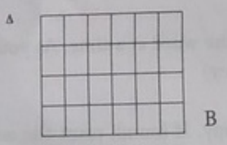
Step 4: Celebrate, we are all done

Sample code in Perl:

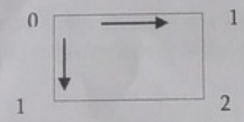
```
#step 1
open("MYFILE",<dictionary.txt>);
while(<MYFILE>){
  $row = $_;
  chomp($row);
  push(@words,$row);
}
my %hashdic = ();

#step 2
foreach $words(@words){
  @not_sorted=split (//, $words);
  @sorted = sort (@not_sorted);
  $name=join("",@sorted);
  if (exists $hashdic{$name})    {
    $hashdic{$name}.=" $words";
  }
  else {
    $hashdic{$name}=$words;
  }
}
$size=keys %hashdic;

#step 3
open("jumbled",<jumbles.txt>);
while(<jumbled>){
  $jum = $_;
  chomp($jum);
  @not_sorted1=split (//, $jum);
  @sorted1 = sort(@not_sorted1);
```

```
    $name1=join("",@sorted1);
    if(length($hashdic{$name1})<1) {
      print "\n$jum : NO MATCHES";
    }
    else {
      @value=split(/ /,$hashdic{$name1});
      print "\n$jum : @values";
    }
}
```

Problem-6 **Pathways:** Given a matrix as shown below, calculate the number of ways for reaching the destination *B* from *A*.



Solution: Before finding the solution, we try to understand the problem with simpler version. The smallest problem that we can consider is the number of possible routes in a 1×1 grid.



From the above figure, it can be seen that:

- From both the bottom-left and the top-right corner there's only one possible route to the destination.
- From the top-left corner there are trivially two possible routes.

Similarly, for 2*x*2 and 3*x*3 grids, we can fill the matrix as:

0	1
1	2

0	1	1
1	2	3
1	3	6

From the above discussion, it is clear that to reach the bottom right corner from left top corner, the paths are overlapping. As unique paths could overlap at certain points (grid cells), we could try to alter the previous algorithm, in a way to avoid following the same path again again. If we start filling 4*x*4 and 5*x*5, we can easily figure out the solution based on our childhood mathematics concepts.

0	1	1	1
1	2	3	4
1	3	6	10
1	4	10	**20**

0	1	1	1	1
1	2	3	4	5
1	3	6	10	15
1	4	10	20	35
1	5	15	35	**70**

Are you able figure out the pattern? It is same as *Pascals* triangle. So, to find the number of ways, we can simply scan through the table and keep couting them while we move from left to right and top to bottom (starting with left-top). We can even solve this problem with mathematical equation of *Pascals* triangle.

References

[1] Akash. Programming Interviews. http://tech-queries.blogspot.com.

[2] Alfred V.Aho,J. E. (1983). Data Structures and Algorithms. Addison-Wesley.

[3] Alfred V.Aho, J. E. (1974). The Design and Analysis of Computer Algorithms. Addison-Wesley.

[4] Algorithms.Retrieved from http://www.cs.princeton.edu/algs4/home

[5] Anderson., S. E. Bit Twiddling Hacks. Retrieved 2010, from Bit Twiddling Hacks: http://www-graphics.stanford.edu /~seander/ bithacks.html

[6] Bentley, J. AT&T Bell Laboratories. Retrieved from AT&T Bell Laboratories.

[7] Bondalapati, K. Interview Question Bank. Retrieved 2010, from Interview Question Bank: http://halcyon.usc.edu/~kiran/msqs.html

[8] Chen. Algorithms http://www2.hawaii.edu/~chenx.

[9] Database, P.Problem Database. Retrieved 2010, from Problem Database: datastructures.net

[10] Drozdek, A. (1996). Data Structures and Algorithms in C++.

[11] Ellis Horowitz, S. S. Fundamentals of Data Structures.

[12] Gilles Brassard, P. B. (1996). Fundamentals of Algorithmics.

[13] Hunter., J. Introduction to Data Structures and Algorithms. Retrieved 2010, from Introduction to Data Structures and Algorithms.:

[14] James F. Korsh, L. J. Data Structures, Algorithms and Program Style Using C.

[15] John Mongan, N. S. (2002). Programming Interviews Exposed. . Wiley-India. .

[16] Judges. Comments on Problems and solutions. http://www.informatik.uni-ulm.de/acm/Locals/2003/html/judge.html.

[17] Kalid. P, NP, and NP-Complete. Retrieved from P, NP, and NP-Complete.: http://www.cs.princeton.edu/~kazad

[18] Knuth., D. E. (1973). Fundamental Algorithms, volume 1 of The Art of Computer Programming. Addison-Wesley.

[19] Knuth., D. E. (1981). Seminumerical Algorithms, volume 2 of The Art of Computer Programming. Addison-Wesley.

[20] Knuth., D. E. (1973). Sorting and Searching, volume 3 of The Art of Computer Programming. Addison-Wesley.

[21] Leon., J. S. Computer Algorithms. . Retrieved 2010, from Computer Algorithms. : http://www.math.uic.edu/~leon/cs-mcs401-s08

[22] Leon., J. S. Computer Algorithms. http://www.math.uic.edu/~leon/cs-mcs401-s08.

[23] OCF. Algorithms. Retrieved 2010, from Algorithms: http://www.ocf.berkeley.edu

[24] Parlante., N. Binary Trees. Retrieved 2010, from cslibrary.stanford.edu: cslibrary.stanford.edu

[25] Patil., V. Fundamentals of data structures. Nirali Prakashan.

[26] Poundstone., W. HOW WOULD YOU MOVE MOUNT FUJI? New York Boston.: Little, Brown and Company. .

[27] Pryor, M. Tech Interview. Retrieved 2010, from Tech Interview: http://techinterview.org

[28] Questions, A. C. A Collection of Technical Interview Questions. Retrieved 2010, from A Collection of Technical Interview Questions: www.spellscroll.com

[29] S. Dasgupta, C. P. Algorithms http://www.cs.berkeley.edu/~vazirani.

[30] Sedgewick., R. (1988). Algorithms. Addison-Wesley.

[31] Sells, C. (2010). Interviewing at Microsoft. Retrieved 2010, from Interviewing at Microsoft: http://www.sellsbrothers.com/fun/msiview

[32] Shene, C.-K. Linked Lists Merge Sort implementation. Retrieved 2010, from Linked Lists Merge Sort implementation: http://www.cs.mtu.edu/~shene

[33] Sinha, P. Linux Journal. Retrieved 2010, from http://www.linuxjournal.com/article/6828.

[34] Structures., d. D. www.math-cs.gordon.edu. Retrieved 2010, from www.math-cs.gordon.edu

[35] T. H. Cormen, C. E. (1997). Introduction to Algorithms. Cambridge: The MIT press.

[36] Tsiombikas, J. Pointers Explained. http://nuclear.sdf-eu.org.

[37] Warren., H. S. (2003). Hackers Delight. Addison-Wesley.

[38] Weiss., M. A. (1992). Data Structures and Algorithm Analysis in C.

[39] wikipedia, T. F. The Free wikipedia. Retrieved from The Free wikipedia: en.wikipedia.org

[40] Zhang., C. programheaven. Retrieved 2010, programheaven.blogspot.com

[41] Mohammed Abualrob, Interview Code Snippets, 2010, interviewcodesnippets.com

[42] Technical Questions. www.ihas1337code.com

Salient Features of Book

- All code written in C/C++
- Data structure puzzles to improve thinking
- Enumeration of possible solutions for each problem
- Covers all topics for competitive exams
- Covers interview questions on data structures and algorithms
- Reference Manual for working people
- Campus Preparation
- Degree/Masters Course Preparation
- Big Job Hunters: Microsoft, Google, Amazon, Yahoo, Oracle, Facebook & many more

About The Author

Narasimha Karumanchi is the Senior Software Developer at Amazon Corporation, India. Most recently he worked for IBM Labs, Hyderabad and prior to that he served for Mentor Graphics and Microsoft, Hyderabad. He received his B-TECH. in Computer Science from JNT University and his M-Tech. in Computer Science from IIT Bombay.

Other Titles

- Coding Interview Questions
- Peeling Design Patterns
- Data Structures And Algorithms Made Easy In Java
- Data Structures And Algorithms Made Easy For GATE

ISBN 978-0-615-45981-3

9 780615 459813

CareerMonk Publications