

Design & Analysis of Algorithms

Asim Manna

Roll No. CrS1908

1. Efficient algorithm of LCM :

For finding lcm of two integer ,I have to find atfirst gcd.

Let a,b two integers.

```
GCD(x,y)
while(y!=0)
    temp = y;
    y = x % y;
    x = temp;
return  x
```

```
LCM(x,y)
if x = 0 or y = 0
    return 0;
else :
    return (x * y)/GCD(x,y)
```

(b).Complexity :

Let,a and b are two integers of n-bit.

Then for multiplications we have to calculate $n \times n$ multiplications of single digit,.

So, for multiplication, the time complexity is $O(n^2)$.

For division, time complexity is $O(n^2)$.

For finding gcd ,time complexity is $O(n^3)$.

Total time complexity is $O(n^2) + O(n^2) + O(n^3) = O(n^3)$

(c).The above algorithm may not work properly for more than 32 bit integers .In the following image atfirst we take two integer 456824 and 452548 ,and the lcm is 144089336 which is wrong as the multiplication of two integers exceeds 32 bit.But in the next example ,gives a right lcm.I am attaching corresponding screenshot in the nextpage.

C-code of lcm:

```
#include <stdio.h>

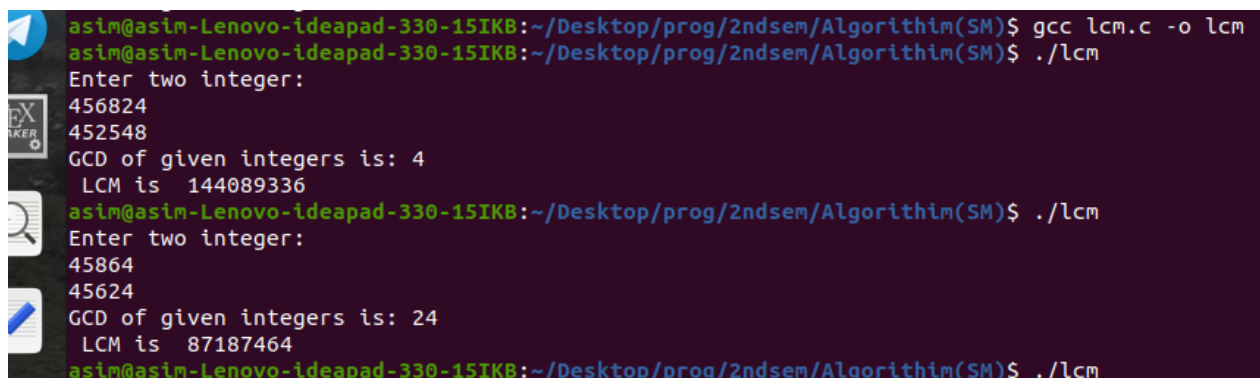
int main()
{
    int x, y, temp, gcd,m,lcm;

    printf("Enter two integer: \n");
    scanf("%d %d", &x, &y);
    int n1=x,n2=y;

    while (y != 0)
    {
        temp = y;
        y = x % y;
        x = temp;
    }

    gcd = x;
    printf("GCD of given integers is: %d\n", gcd);
    lcm=(n1*n2)/gcd;
    printf(" LCM is  %d \n",lcm);

    return 0;
}
```

Output:

```
asim@asim-Lenovo-ideapad-330-15IKB:~/Desktop/prog/2ndsem/Algorithim(SM)$ gcc lcm.c -o lcm
asim@asim-Lenovo-ideapad-330-15IKB:~/Desktop/prog/2ndsem/Algorithim(SM)$ ./lcm
Enter two integer:
456824
452548
GCD of given integers is: 4
LCM is 144089336
asim@asim-Lenovo-ideapad-330-15IKB:~/Desktop/prog/2ndsem/Algorithim(SM)$ ./lcm
Enter two integer:
45864
45624
GCD of given integers is: 24
LCM is 87187464
asim@asim-Lenovo-ideapad-330-15IKB:~/Desktop/prog/2ndsem/Algorithim(SM)$ ./lcm
```

3. Mergesort in C code :

```
#include<stdio.h>

void mergesort(int a[],int low,int high)
{
    int mid;
    if(low<high)
    {
        mid=(low+high)/2;
        mergesort(a,low,mid);
        mergesort(a,mid+1,high);
        merge(a,low,mid,mid+1,high);
    }
}

void merge(int a[],int low1,int high1,int low2,int high2)
{
    int temp[1000];
    int i=low1,j=low2,k=0;
    while(i<=high1 && j<=high2)
    {
        if(a[i]<a[j])
            temp[k++]=a[i++];
        else
            temp[k++]=a[j++];
    }
    while(i<=high1)
        temp[k++]=a[i++];

    while(j<=high2)
        temp[k++]=a[j++];

    for(i=low1,j=0;i<=high2;i++,j++)
        a[i]=temp[j];
}

int main()
{
    int a[1000],n,i;
    printf("Enter no of elements:");
    scanf("%d",&n);
    printf("Enter the elements:");

    for(i=0;i<n;i++)
        scanf("%d",&a[i]);

    mergesort(a,0,n-1);

    printf("\nThe sorted array is :");
    for(i=0;i<n;i++)
        printf("%d ",a[i]);

}
```

Output:

```

asim@asim-Lenovo-ideapad-330-15IKB:~/Desktop/prog/2ndsem/Algorithm(SM)$ ./merge1
Enter no of elements:5
Enter the elements:6
-8
9
-17
8
The sorted array is :-17 -8 6 8 9 asim@asim-Lenovo-ideapad-330-15IKB:~/Desktop/prog/2nds

```

4.

5. (a). **Algorithm of All pair shortest path :**

The Floyd-Warshall Algorithm solves the All Pairs Shortest Path (APSP) problem: given a graph $G = (V, E)$, find the shortest path distances $d(s, t)$ for all $s, t \in V$.

Floyd-Warshall Algorithm (G)

$d_k(u, u) = 0, \forall u \in V, k \in \{0, 1, \dots, n\}$
 $d_k(u, v) = \infty, \forall u, v \in V, u \neq v, k \in \{1, \dots, n\}$
 $d_0(u, v) = c(u, v), \forall (u, v) \in E$
 $d_0(u, v) = \infty, \forall (u, v) \notin E$
 for $k = 1, \dots, n$ do
 for $(u, v) \in V$ do
 $d_k(u, v) = \min\{d_{k-1}(u, v), d_k(u, k) + d_k(k, v)\}$
 return $d_n(u, v), \forall u, v \in V$

Complexity : Consider the graph $G=(V,E)$.where $| V |= n$.

Here we have used three for loop (for k and choosing two vertices),for that the required time is $o(n^3)$

. For addition it will take $o(n^3)$ *(constant time).

so The total runtime of the Floyd-Warshall algorithm is is $O(n^3)$.

(c). I am taking complete graph K_8 ,with weight cost 1 for every edge. Then in every iteration the matrix will be same beacuse the entries of matrix does not change and every time $d_k(u, v) = d_{k-1}(u, v) = 1$, where $u \neq v$ and the 8×8 matrix will be ,

$$\begin{pmatrix} 0 & 1 & \dots & 1 \\ 1 & 0 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 0 \end{pmatrix}$$

6. (a). **Longest Common Subsequence Problem :**

Let us consider two string :

S1 :abcde and S2 : ab

Now ,here common sequences are $\{a\}, \{b\}, \{a, b\}$.

and so the longest common subsequence is $\{a, b\}$

(b). **An algorithm in dynamic programming approach:**

```

LCS(S1, S2)
m ← S1.length
n ← S2.length
C[i, 0] = C[0, j] = 0  ∀ i = 1, ..., m, j = 1, ..., n.
for i = 1, ..., m
  for j = 1, ..., n
    if S1[i - 1] = S2[j - 1] :
      C[i, j] = C[i - 1, j - 1] + 1
    else :
      C[i, j] = max{C[i, j - 1], C[i - 1, j]}

```

(c). **Complexity:**

Here we use two string of length m and n that means a 2D array of dimension $m \times n$. For that we need $O(mn)$ time. For finding longest common subsequence we have to go one row with one column. it takes $O(m+n)$ time.

So for this algorithm the time complexity to $O(n \times m)$ where n and m are the lengths of the strings.

7. (a). **Basic idea of Greedy algorithm:**

Basically, Greedy algorithm is used in the case of optimization problem. This algorithm takes the optimal choice at each step as it targets to find the overall optimal way to solve the entire problem.

(b). **The minimum spanning tree given an undirected weighted graph:**

Kruskal's Algorithm:

Let $G = (V, E)$ be a undirected graph . We are finding the minimum spanning tree by using Kruskal's algorithm . Kruskal's algorithm follows greedy approach as in each iteration it finds an edge which has least weight and add it to the growing spanning tree. The steps of this algorithm are following:

- (i). Atfirst we collect all the weights of graph and sorting the graph edges with respect to their weights in non-decreasing order.
- (ii). Then we are adding edges to the minimum spanning tree from the edge with the smallest weight to the edges of the largest weight until to the edges.
- (iii). Only add those edges which doesn't form a cycle and repeat Step (ii) until there are $(V-1)$ edges in the spanning tree.

By the above process we can find the minimum spanning tree in a undirected graph.

(c). **Proof:**

claim: If I run Kruskal's algorithm on a connected undirected weighted graph G , its output T is a minimum weight spanning tree. For that I have to prove T is a spanning tree with minimum weight.

Now, T is spanning tree as T is forest (there is no cycle), spanning and connected.

Now I will prove T is spanning tree with minimum weight. for that let T_1 minimum weight spanning tree.

Then if $T = T_1$ than we are done.

if not, then there exist an edge of minimum weight which is in T_1 but not in T. Then $T \cup \{e\}$ is contain a cycle C , say. Then,

- (i). $wt(\text{of every edge in } C) \leq wt(e)$.
- (ii). There is some another edge say, s not in T_1 .

Consider the tree $T_2 = T \setminus \{e\} \cup \{s\}$.

Then T_2 is spanning tree and $wt(T) \leq wt(T_2) \leq wt(T_1)$

Then above process we have $wt(T) \leq wt(T_2) \leq wt(T_3) \leq \dots \leq wt(T_1)$

We can see that T is minimum weight spanning tree.

so. we are done.

(d).Complexity;

Let $G=(V,E)$ be a connected graph . Then $|V| - 1 \leq |E|$.

Then sorting the weights of edges of the graph it takes $O(E \log E)$ time. Then initially for choosing each vertex into its own disjoint set, which takes $O(V)$ times.

Since, $|V| - 1 \leq |E|$, the total time required $O((E + V) \log E) \leq O(E \log E)$.

Now, since $|V|^2 > |E|$ implies $2 \log |V| > \log |E|$. implies $O(E \log E) < O(E \log V)$.

In Kruskal's algorithm, We need $O(E \log V)$ time .

8.

9. (a). 3-SAT Problem:

Input: Consider the set of variables $X = \{x_1, x_2, \dots, x_n\}$ and a collection of clauses $C = \{c_1, c_2, \dots, c_m\}$ over X such that $|c_i| = 3, 1 \leq i \leq m$.

Output: There a truth assignment for X that satisfies all clauses in C .

Vertex Cover Problem:

Input: A graph $G = (V, E)$ and an integer $k \leq |V|$

Output: There a subset S of at most k vertices such that every $e \in E$ has at least one vertex in S .

(b).Proof of Vertex cover problem is NP-complete:

Given that 3-SAT is NP-complete and we know that Vertex-Cover is NP. Now We want reduces from 3-SAT to Vertex-Cover.

For that, atfirst we construct the variables of the 3-SAT problem. For each Boolean variable x_i , we create two vertices x_i and \bar{x}_i connected by an edge. To cover these edges. Next, we want construct the clauses of the 3-SAT problem. We create a triangle depend upon every clause. Let c be clause, and consider three vertices, one for each literal in each clause. Then connects by edges, the vertices forms a triangle. At least two vertices of them per triangle must be included in any vertex cover of these triangles.

Now we will show that in 3-SAT formula, if it has a vertex-cover iff the 3SAT formula has a satisfying assignment. Let in 3-SAT, there are l literals and c clauses. Then, we will prove the truth assignment is satisfiable iff have a vertex cover of sizes $l+2c$.

Given any vertex cover C of size $l+2c$, Then we need exactly l of the vertices for covering the pair literal and other $2c$ vertices for covering the triangle. We choose the literal nodes in the vertex cover C to be our assignment of the formula. Therefore, if C gives a vertex cover the corresponding truth assignment has to satisfy the formula. Conversely, Given a satisfying truth assignment for the clauses of 3-SAT formula. we want construct a vertex cover of $l+2c$ no. of vertices. Chose l nodes for our l variables. For covering one triangle we need at most 2 vertices i.e if we have c clauses (triangle) so, we need total $2c$ no. of vertices. Then also in every clause will be true for one literals. So, we get a vertex cover of $l+2c$ no. of vertices.

The above process i.e construction a graph with a vertex cover of $l+2m$ no. of vertices, it will be done in polynomial time. So, We reduces from 3-SAT problem to vertex-cover problem. Then vertex-cover is NP-hard problem. Since 3-SAT is NP-complete, then we can say Vertex-cover problem is NP-complete.

Date-01.08.2020

Roll No-CrS1908

Click Here: <https://crs1908.wordpress.com>

Mob no- 6289664072

Email: asimman17@gmail.com