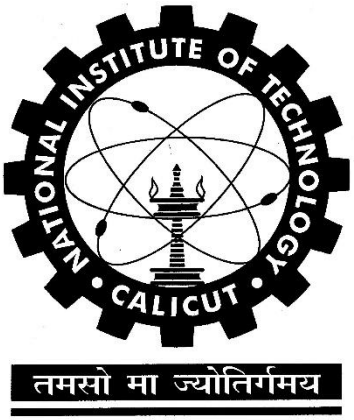


TOPICS IN ALGORITHM

ASSIGNMENT REPORT

SETCOVER (Dynamic Programming Approach)

Instructor: DR.SUBASHINI

Prepared by:

AKARSH JOICE

B160148CS

COURSE:CS4027

DATE:27/04/201

INDEX

1.1 Problem Statement

Input: A universal set S={e1, e2,…,en } and set C={c1,c2,…,cm} where ci $\subset$ S.

Output: Number of sets in C' $\subset$ C such that $\bigcup_{ci \in C'}$ ci=S where C' is the minimum cardinality set.

1.2 Recursive Algorithm(Brute Force Approach)

```
1.SC(m,S,C)
2.if (S == {})
3.          return 0
4.elseif (m == 0)
5.          return +∞
6.else
7.          return min(1+SC(m-1,S-Cm,C),SC(m-1,S,C))
```

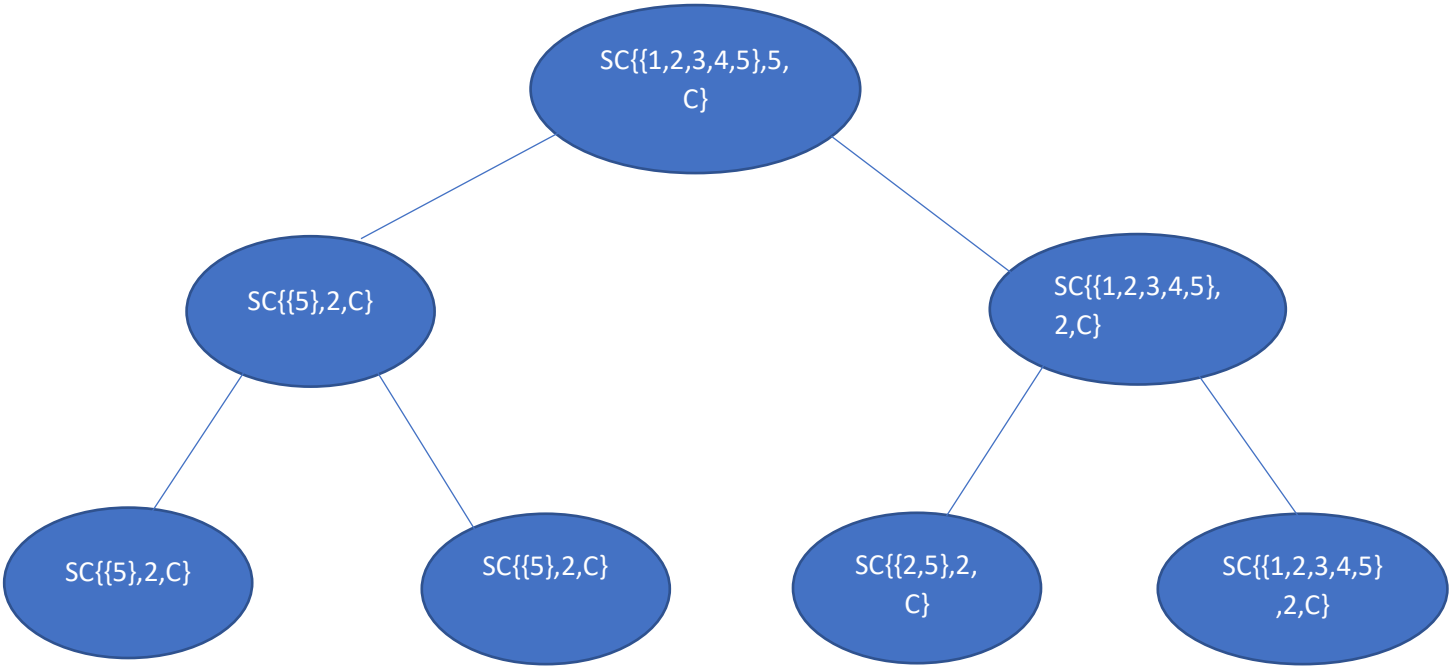
NOTE:min(a,b) returns the min of the two value (a,b).

1.3 Why do we need Dynamic Programming ?

From the recursive algorithm it is clear that there is repeated subproblem.To reduce the computation for these subproblem we use dynamic programming by which values of the repeated subproblem are stored and later reused.

Eg: m=3 ;S={1,2,3,4,5} ; C={{2,3,4},{1,3,4},{1,2,3,4}}

.



Here in the third level we see that the subproblem SC(1,{5}) is repeated.This means that there is repeated computation of this subproblem.To avoid this repeated computation we need a technique where the value of the subproblem is stored for later reuse.Hence we use Dynamic programming.

1.4 Dynamic Programming Algorithm

Setcover(S,C,n,m)

0.Array T[(2^n),m+1] // row i indicates ith subset of set S and column j indicates first j element set of C

1. for each subset i of set S

2. T[i,0]= +∞

3. for(j=0 to m)

4. T[0,j]=0

5. for each subset i of set S

6. for(j=1 to m)

7. T[i,j]=min(T[i,j-1],1+T[i-cj,j-1]

8.print T[2^n,m]

NOTE : min(a , b) returns the min of the two value (a ,b).

1.5 Complexity Discussion

For the Recursive Algorithm ,

T(m)=2T(m-1) + n^2 is the recurrence relation.

On solving the recurrence, we get complexity as O(2^m).In worst case,m=2^n

For the DP approach the complexity is,

Step 1,2: O(2^n)

Step 3,4: O(m)

Step 5,6: O(2^n*m)

Step 7: O(n)

Total complexity=O(2^n + m + m*2^n + n)=O(2^n*m)

We find that the complexity has reduced from O(2^2^n) to O(m*2^n) using Dynamic programming approach.

1.6 Correctness

1.6.1.Description of Setcover(Dynamic programming)

Tabulation method is used here to implement the dynamic programming setcover algorithm.

Input :S={e1,e2,…,en},C={c1,c2,…,cm}

The table corresponding to the input is as follow :rows indicates all possible subsets of set S and jth column indicates set of first j elements of C as shown in fig below.

X/j	{}	{c1}	{c1,c2}	{c1,c2,c3}	. . . . .	C={c1,c2, …,cm}
{}						
{e1}						
{e2}						
. . . .						
.						
S={e1,e2, …,en}						

Fig:T[2^n,m+1]

Each cell T[x,j] represents the minimum no of possible subsets required to cover set x from F={c1∪c2∪... ∪cj}

So the last entry that is T[2^n,m] represent the minimum possible subset required to cover x=S from F={c1∪c2∪... ∪cm) that is C .Therefore last entry gives the output to setcover problem. And this is used as the loop invariant to prove the correctness of the algorithm.

1.6.2 Proof For Correctness

In step0 all the elements in first column except first one gets value +∞.Elements in first column corresponds to a minimum possible subsets required to cover x!={} from the empty set. Since {} has only one subset which is that itself. We cant cover any set!={} with an empty set ,ie corresponds to +∞.Hence true.

In step3 all the elements in 1st row gets value 0.First row correspond to empty set(1st subset of set S containing no elements) that is we don't need any elements to cover this null set . Trivially true.

Loop Invariant

T[x,j] represents the minimum no of possible subsets required to cover set x from C={c1∪c2∪... ∪cj}.

Initialization:

T[{},j]=0 is true for all values of j ranging from 1 to m and is proved above.

T[x,0]= +∞ is true for all subset x of S and is proved above.

Maintenance:

Loop in step5,6 iterates over the array sets elements as T[x,j]=min(1+T[x-cj,j-1],T[x,j-1]).The value is each cell is determined by two cases,

Case 1: x ⊆{c1,c2,…,cj-1}

Since the cell T[x,j-1] contains the minimum number of sets from c0 to cj-1 required to cover set x and given that x ⊆{c1,c2,…,cj-1} it is evident that it is not necessary to include the set cj to the collection of sets required to cover x. Hence we take the minimum possible subsets required to cover x from F={c1,c2,…,cj-1} i.e T[x,j-1].Hence loop invariant is maintained here.

Case 2 : x ⊄ {c1,c2,…,cj-1}

Since x ⊄ {c1,c2,…,cj-1} and x ⊆{c1,c2,…,cj} there is at least one element in cj that is required to cover x,so cj must be included in answer.But we know that there might be sets from c1 to cj-1 that are not necessary if cj is included,since we wanted the optimal solution.So we remove the elements of cj from x and subtract one from j.So we move to the cell T[x-cj,j-1].Since we are including the set cj we add 1 to the value obtained from T[x-cj,j-1] i.e 1 + T[x-cj,j-1] and add it to cell T[x,j].Hence it contains the minimum value and loop invariant is maintained here.

Termination:

As the loop terminate , T[x=S,j=m] corresponds to the minimum possible subset from C to cover S,i.e setcover.

Hence the loop invariant is maintained throughout the loop,proving the correctness of the algorithm.

1.7 Implementation

Programming language used : C language

Compiler : gcc version 8.2.0 (Ubuntu 8.2.0-7ubuntu1)

1.7.1 TEST CASES

1. S = {1,2,3,4,5,6}

   C = {{1,2},{2,3},{4,5},{1,2,3,,4,5},{1,4,6},{3,4,5,6}}

2. S = {1,2,3,4,5,6,7)

   C = {{1},{2},{3},{4},{5},{6},{7},{1,2,3,4,5,6,7}}

3. S = {2,3,4,5,6}

   C = {{2,3},{2,3,4},{2,4,6},{3,6}}

4. S = {1,2,3,4}

   C = {{1},{2},{3},{4}}

5. S = {1,2,3,4,5,6}

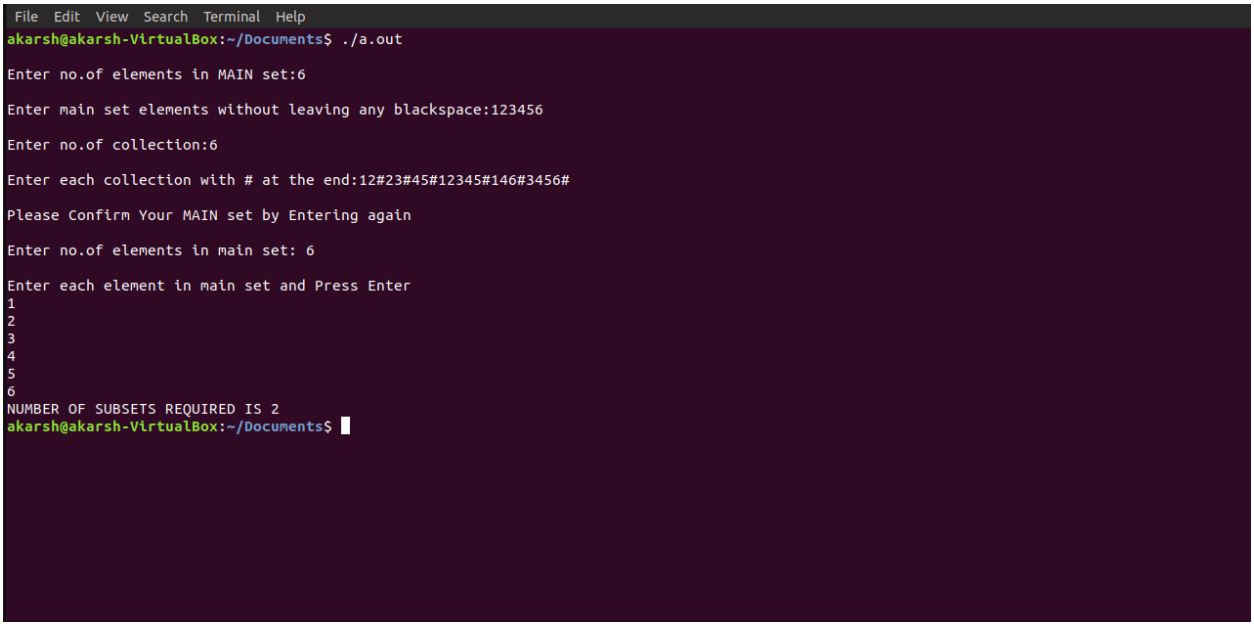
   C = {{1,2,3,4},{5,6},{1,2,3,4,5},{6},{1}}

6.S = {1,3,5,7,9}

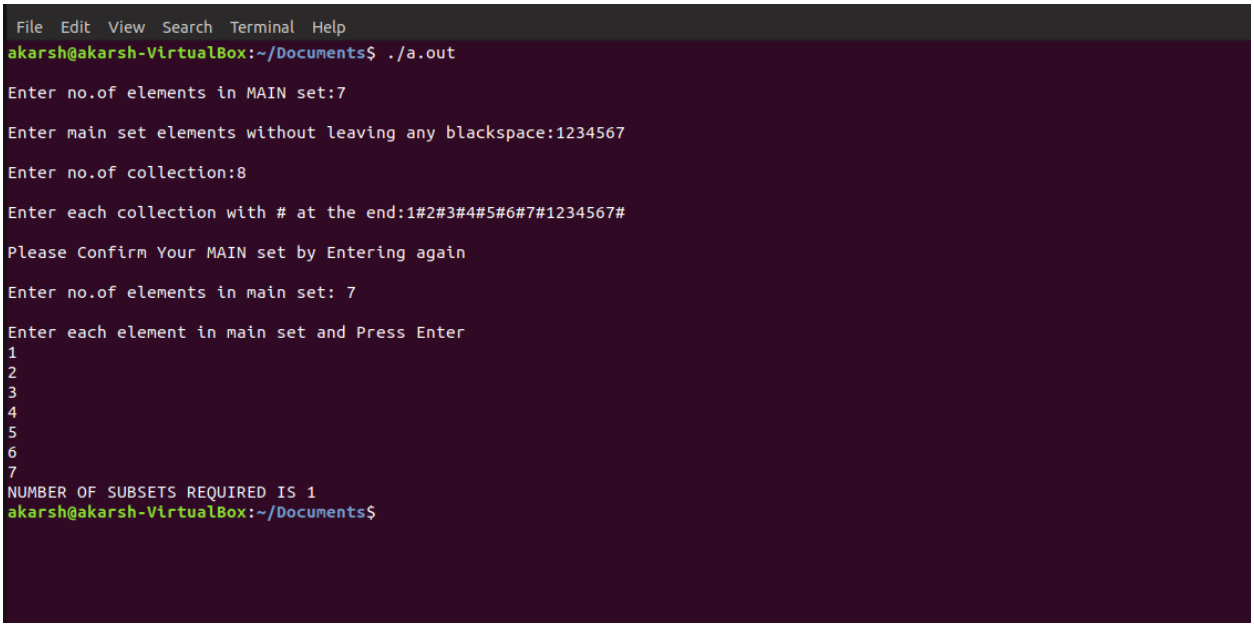
  C = {{1,5,7},{5,7,9},{3,7,9},{1,3},{1,5,9}}

17.2 OUTPUT SCREENSHOT OF TESTCASES

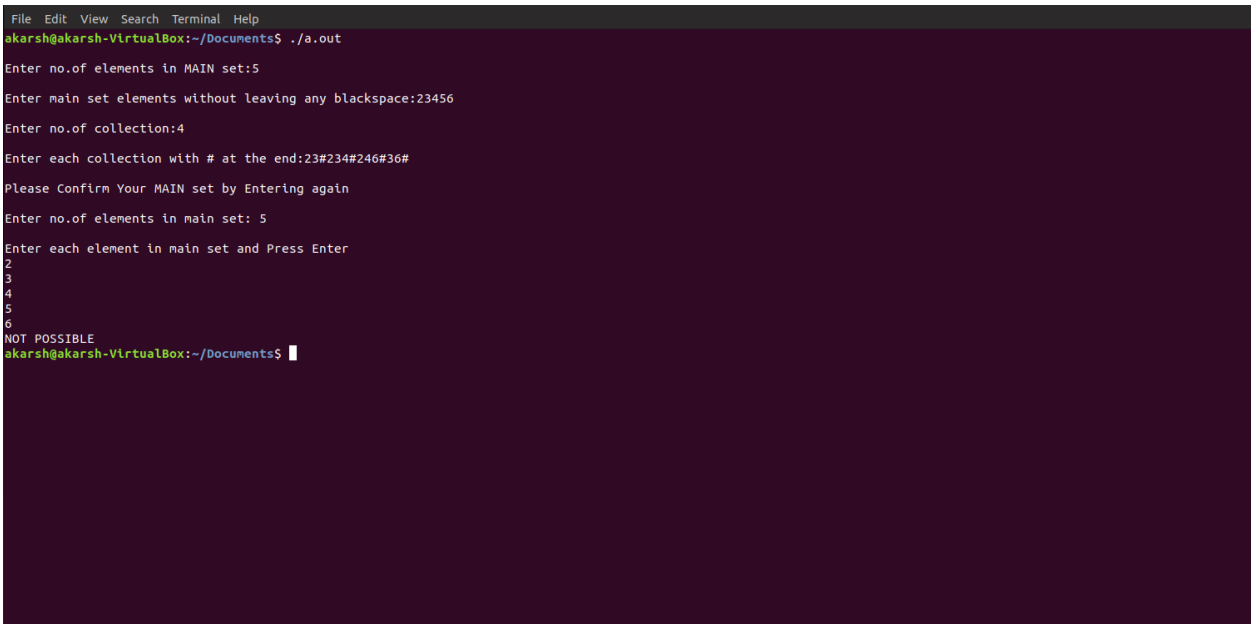
1.



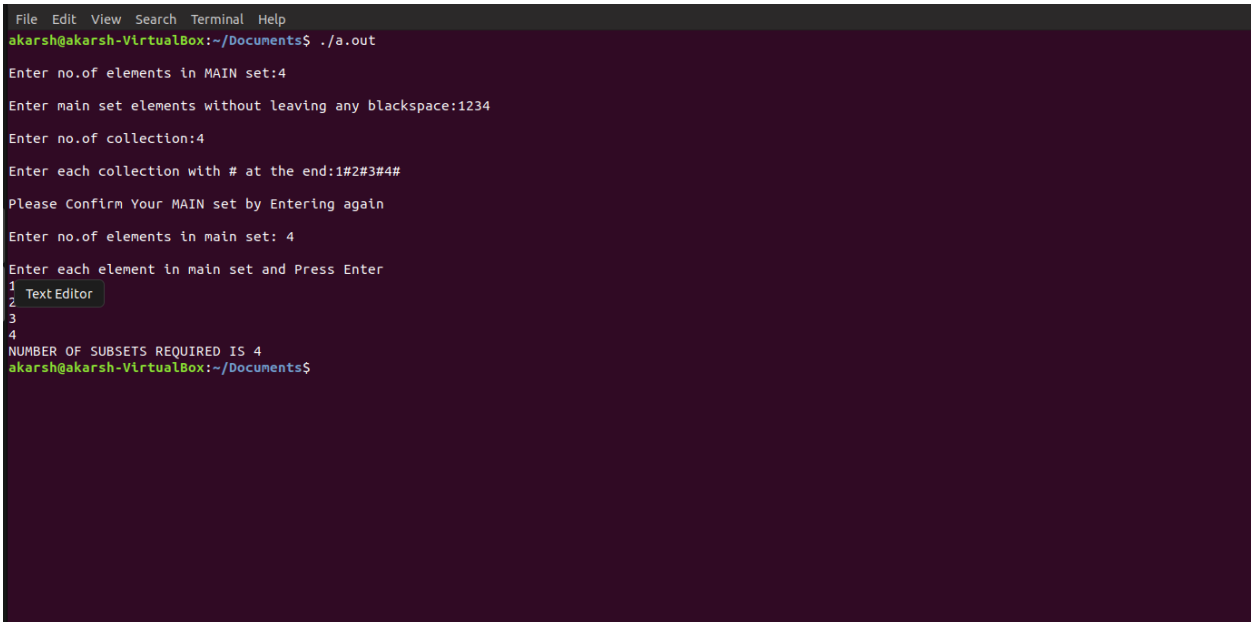
2.



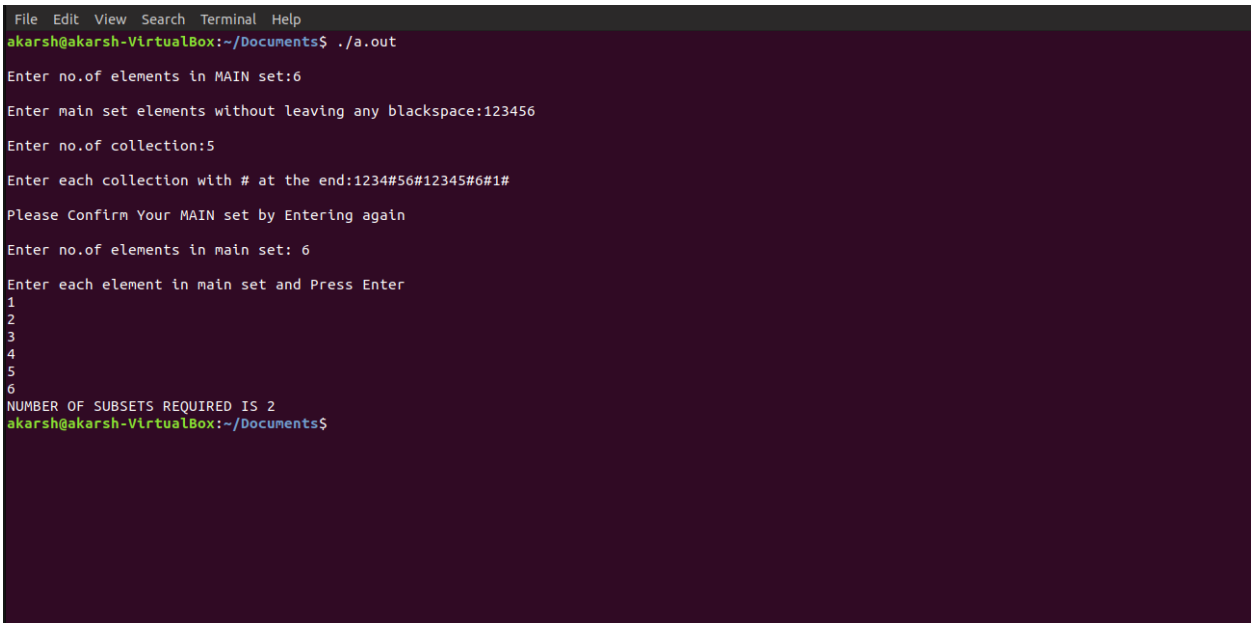
3.


File Edit View Search Terminal Help
akarsh@akarsh-VirtualBox:~/Documents$ ./a.out
Enter no.of elements in MAIN set:5
Enter main set elements without leaving any blackspace:23456
Enter no.of collection:4
Enter each collection with # at the end:23#234#246#36#
Please Confirm Your MAIN set by Entering again
Enter no.of elements in main set: 5
Enter each element in main set and Press Enter
2
3
4
5
6
NOT POSSIBLE
akarsh@akarsh-VirtualBox:~/Documents$

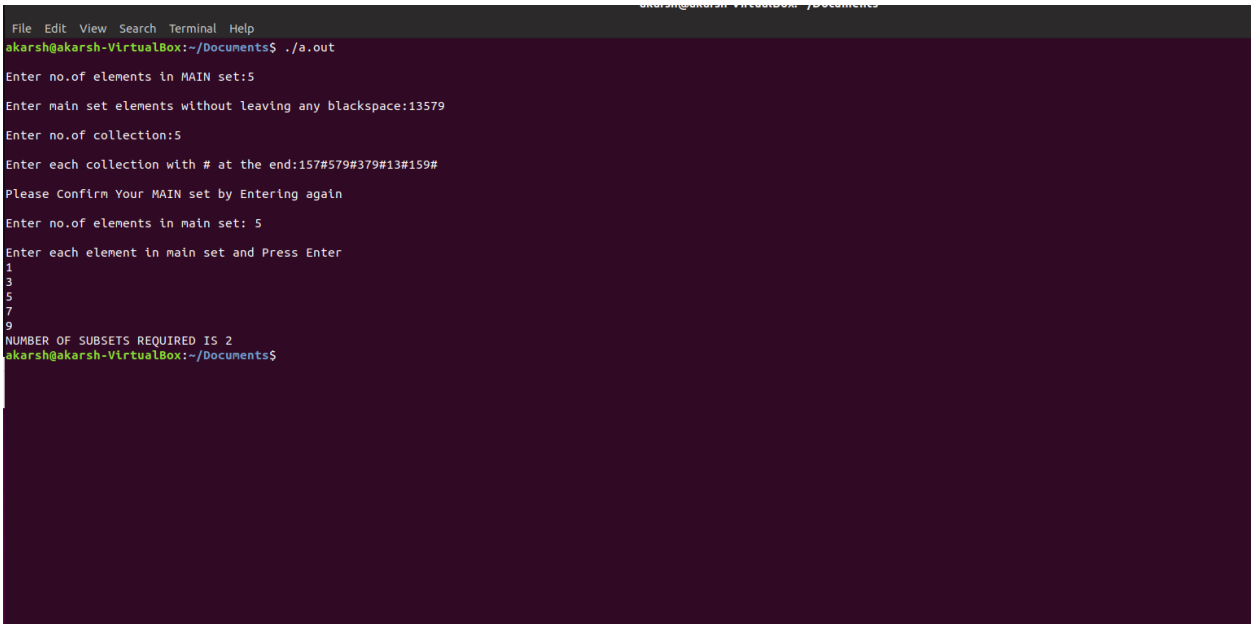
4.


File Edit View Search Terminal Help
akarsh@akarsh-VirtualBox:~/Documents$ ./a.out
Enter no.of elements in MAIN set:4
Enter main set elements without leaving any blackspace:1234
Enter no.of collection:4
Enter each collection with # at the end:1#2#3#4#
Please Confirm Your MAIN set by Entering again
Enter no.of elements in main set: 4
Enter each element in main set and Press Enter
1
2
3
4
NUMBER OF SUBSETS REQUIRED IS 4
akarsh@akarsh-VirtualBox:~/Documents$
Text Editor

5.


File Edit View Search Terminal Help
akarsh@akarsh-VirtualBox:~/Documents$ ./a.out
Enter no.of elements in MAIN set:6
Enter main set elements without leaving any blackspace:123456
Enter no.of collection:5
Enter each collection with # at the end:1234#56#12345#6#1#
Please Confirm Your MAIN set by Entering again
Enter no.of elements in main set: 6
Enter each element in main set and Press Enter
1
2
3
4
5
6
NUMBER OF SUBSETS REQUIRED IS 2
akarsh@akarsh-VirtualBox:~/Documents$

6.


File Edit View Search Terminal Help
akarsh@akarsh-VirtualBox:~/Documents$ ./a.out
Enter no.of elements in MAIN set:5
Enter main set elements without leaving any blackspace:13579
Enter no.of collection:5
Enter each collection with # at the end:157#579#379#13#159#
Please Confirm Your MAIN set by Entering again
Enter no.of elements in main set: 5
Enter each element in main set and Press Enter
1
3
5
7
9
NUMBER OF SUBSETS REQUIRED IS 2
akarsh@akarsh-VirtualBox:~/Documents$