Project Report
on

AN APPLICATION FOR OPENING COFFEE SHOP

DESIGN AND ANALYSIS OF ALGORITHMS

by

G. Dilip Reddy	(2010030041)
K. Vamshi	(2010030326)
M. Ruchitha	(2010030480)
P. Venkata Sai Aditya	(2010030434)

under the supervision of

Ms. P.SREE LAKSHMI

Assistant Professor

Department of Computer Science and Engineering

K L University Hyderabad,

Aziz Nagar, Moinabad Road, Hyderabad – 500 075, Telangana, India.

April, 2022

DECLARATION

The Project Report entitled "**An Application for Opening Coffee Shop**" is a record of bonafide work of **G. Dilip Reddy (2010030041), K. Vamshi (2010030326), M. Ruchitha (2010030480), P. Venkata Sai Aditya (2010030434)**, submitted as a requirement for the completion of the course **Design and Analysis of Algorithms** in the Department of Computer Science and Engineering to the K L University, Hyderabad. The results embodied in this report have not been copied from any other Departments/University/Institute.

G. Dilip Reddy (2010030041)

K. Vamshi (2010030326)

M. Ruchitha (2010030480)

P. Venkata Sai Aditya (2010030434)

CERTIFICATE

This is to certify that the Project Report entitled "**An Application for Opening Coffee Shop**" is being submitted by **G. Dilip Reddy (2010030041), K. Vamshi (2010030326), M. Ruchitha (2010030480), P. Venkata Sai Aditya (2010030434),** as a requirement for the completion of the course **Design and Analysis of Algorithms** in the Department of Computer Science and Engineering, K L University, Hyderabad is a record of bonafide work carried out under our guidance and supervision.

The results embodied in this report have not been copied from any other departments/ University/Institute.

Signature of the Supervisor

Ms P.SREE LAKSHMI

Assistant Professor

Signature of the HOD

Signature of the External Examiner

ACKNOWLEDGEMENT

First and foremost, we thank the lord almighty for all his grace & mercy showered upon us, for completing this project successfully.

We take grateful opportunity to thank our beloved Founder and Chairman who has given constant encouragement during our course and motivated us to do this project. We are grateful to our Principal **Dr. L. Koteswara Rao** who has been constantly bearing the torch for all the curricular activities undertaken by us.

We pay our grateful acknowledgement & sincere thanks to our Head of the Department **Dr. Chiranjeevi Manike** for his exemplary guidance, monitoring and constant encouragement throughout the course of the project. We thank **Ms. P Sree Lakshmi** of our department who has supported throughout this project holding a position of supervisor.

We whole heartedly thank all the teaching and non-teaching staff of our department without whom we won't have made this project a reality. We would like to extend our sincere thanks especially to our parent, our family members and friends who have supported us to make this project a grand success.

ABSTRACT

Our project is implementation of the greedy approach to set cover. The set cover problem is exciting simple to describe: given a collection F of sets, each containing a subset of a universe U of objects, find a smallest subcollection A of F such that every object in U is included in at least one of the sets in A. However, like many such combinatorial problems, set cover is NP-hard, meaning that it is unlikely that an efficient algorithm will be found, and that approximation algorithms must be preferred for non-trivial problem instances. One well-known approximation approach for set cover is to repeatedly add the set with the most uncovered items to the solution, continuing until every element in the universe is covered; this greedy approach has a provable logarithmic approximation ratio, essentially the best feasible ratio.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1. About Set cover problem

Given a set system (U, F), where U contains n elements and F is a family of m subsets of U such that every element of U belongs to at least one subset in F, here each subset in F has a positive weight, a set cover C of U is a subfamily of F such that every element in U is in at least one of the subsets in C. The set cover problem is to find a set cover with the minimal total weight of subsets in the set cover. For this famous NP-hard problem the set cover problem has many applications in practice.

1.2. Explanation with an example

For example, how to route and allocate wavelengths to a broadcast connection so that the total wavelength conversions required is minimized. Under some conditions they formulate this problem as two closely related set cover problems, the minimum wavelength-covering problem and the minimum vertex wavelength-covering problem. But some practical problems may have special configurations and the set cover model may not appropriate for them. In this case, connectedness of a set cover appears to be an important requirement. We therefore in this project consider a natural extension of the set cover problem, the connected set cover problem. Besides the universal set U and a family F of U, we are also given a graph G with vertex-set V (G) = F and edge-set E(G) consisting some edges between some pairs of subsets in F. A set cover C $\subseteq$ F of U is called connected if the induced subgraph G(C) is connected, where G(C) is a subgraph of G that consists of all edges whose two endpoints are both in C. The problem is to find a connected set cover with the minimal number of subsets. It is easy to see that the classic set cover problem is a special case of the connected set cover problem with a completed graph. In this project we will first show that the connected set cover problem is NPhard even if at most one vertex of the given graph has degree greater than two, and it cannot be solved in polynomial-time. We then propose the polynomial time algorithms for the case where every vertex in the graph has degree at most two. For the case where at most one vertex has degree greater than two, we propose an approximation algorithm with performance ratio at most 1 + ln n that is the best possible.

1.3. Complexity Analysis

The complexity of the connected set cover problem for some special graphs. We shall see that the difficulty in solving the connected set cover problem not only lies in the structure of (U, F) system but also related to the property of give graph G. Graph G is called a line graph if two vertices in V (G) have degree one and all others have degree two. Graph G is called a ring graph if it is connected and every vertex in V (G) has degree two. Graph G is called a spider graph if G is a tree and only one vertex has degree greater than two, a spider graph is particularly called a star graph if one vertex has degree greater than one while all others have degree one.

2. LITERATURE SURVEY

A BETTER-THAN-GREEDY APPROXIMATION ALGORITHM FOR THE MINIMUM SET COVER PROBLEM

REFAEL HASSIN AND ASAF LEVIN through their article they convey that**,** In the weighted set-cover problem we are given a set of elements E = {e1, e2,...,en} and a collection F of subsets of E, where each S ∈ F has a positive cost cS. The problem is to compute a subcollection SOL such that S∈SOL Sj = E and its cost S∈SOL cS is minimized. When |S| ≤ k ∀S ∈ F we obtain the weighted k-set cover problem. It is well known that the greedy algorithm is an Hk-approximation algorithm for the weighted k set cover, where Hk = k i=1 1 i is the kth harmonic number, and that this bound is exact for the greedy algorithm for all constant values of k. We give the first improvement on this approximation ratio for all constant values of k.

This result shows that the greedy algorithm is not the best possible for approximating the weighted set cover p. This important progress in the approximability of the unweighted k-set cover does not help to approximate the weighted k-set cover problem within a factor better than Hk. The question whether there are similar results for the weighted problems was left unanswered. A first answer was given by Fujito and Okumura [6], who presented an Hk − 1 12 -approximation algorithm for the k-set cover problem where the cost of each subset is either 1 or 2. Their method is based on extending the algorithm of [7] for this case, but this approach does not extend further to the general case. In this paper we give the first positive answer for a general cost function by developing an approximation algorithm for the weighted k-set cover problem whose performance guarantee is bounded by Hk− k−1 8k9 . This bound is not tight, but it is sufficient to show that the greedy algorithm does not give the best possible performance guarantee for approximating the weighted set cover problem. Our method is based on a modification of the greedy algorithm that allows the algorithm to regret throughout its execution.

IMPROVED LOCAL COMPUTATION ALGORITHM FOR SET COVER VIA SPARSIFICATION

Christoph Grunau, Slobodan Mitrović, Ronitt Rubinfeld, and Ali Vakilian design a Local Computation Algorithm (LCA) for the set cover problem. Given a set system where each set has size at most s and each element is contained in at most t sets, the algorithm reports whether a given set is in some fixed set cover whose expected size is O(log s) times the minimum fractional set cover value. Our algorithm requires $s^{O(\log s)} t^{O(\log s+\log \log t))}$ queries. This result improves upon the application of the reduction of [Parnas and Ron, TCS'07] on the result of [Kuhn et al., SODA'06], which leads to a query complexity of $(st)^{O(\log s \cdot \log t)}$.

To obtain this result, we design a parallel set cover algorithm that admits an efficient simulation in the LCA model by using a sparsification technique introduced in [Ghaffari and Uitto, SODA'19] for the maximal independent set problem. The parallel algorithm adds a random subset of the sets to the solution in a style similar to the PRAM algorithm of [Berger et al., FOCS'89]. However, our algorithm differs in the way that it never revokes its decisions, which results in a fewer number of adaptive rounds. This requires a novel approximation analysis which might be of independent interest.

3. HARDWARE & SOFTWARE REQUIREMENTS

3.1. Hardware Requirements

The hardware requirements that map towards the software are as follows:

RAM	: 4.00 GB
Processor	: Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz 1.70 GHz

3.2. Software Requirements

The major software requirements of the project are as follows:

Language	: HTML, CSS, JAVASCRIPT and SQL
Operating system	: Windows
Tools	: PyCharm

4. FUNCTIONAL AND NON FUNCTIONAL REQUIREMENTS

4.1. Functional Requirements

Latest version of python along with a compiler is required

• python 3.9.10

• VS code/ Pycharm

4.2. Non Functional Requirements

• Input should be given in the proper format.

• All the possible solutions must be highlighted .

• If any location is not considered to be optimal, that should be left.

• In the end after obtaining the optimal location, it should be pointed/marked.

5. PROPOSED SYSTEM

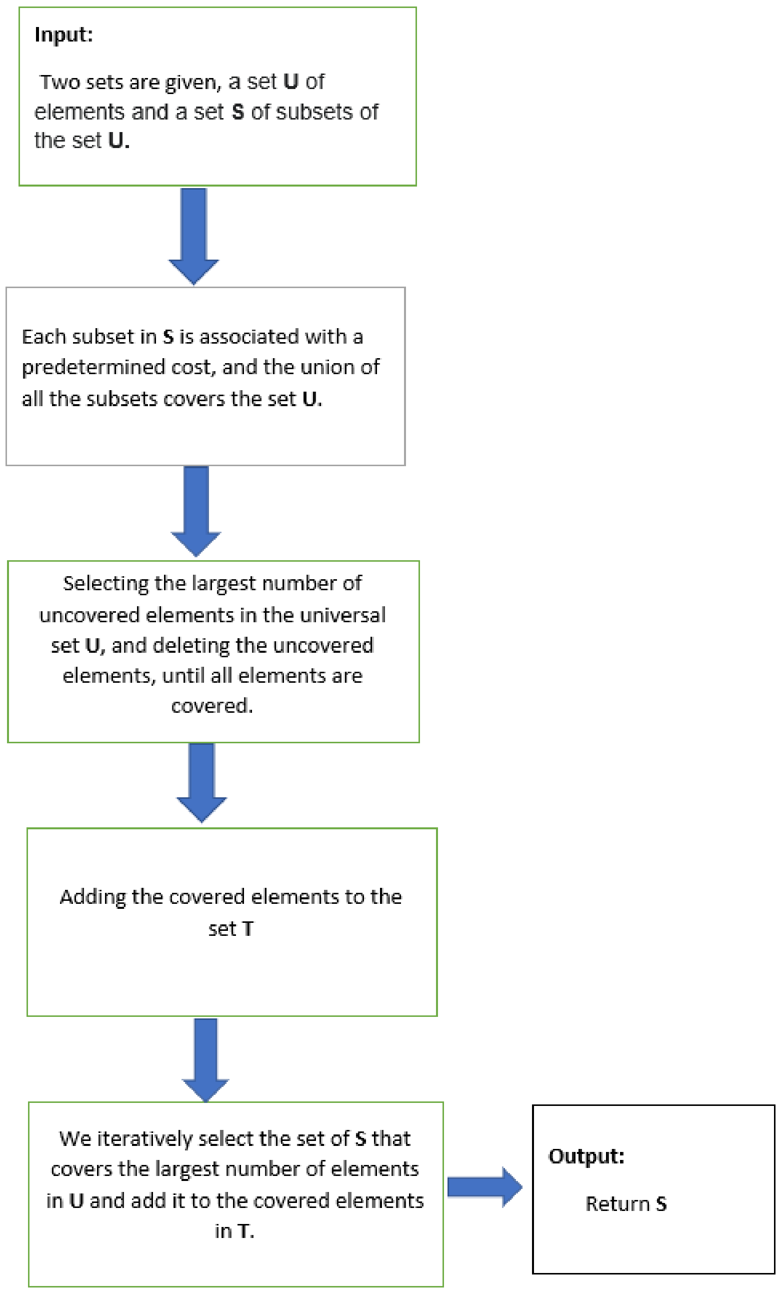

Input:
Two sets are given, a set U of elements and a set S of subsets of the set U.
Each subset in S is associated with a predetermined cost, and the union of all the subsets covers the set U.
Selecting the largest number of uncovered elements in the universal set U, and deleting the uncovered elements, until all elements are covered.
Adding the covered elements to the set T
We iteratively select the set of S that covers the largest number of elements in U and add it to the covered elements in T.
Output:
Return S

Fig 5.1 : Flow Chart

The Fig 5.1 is the best representation which describes how the output is generated and below is the example which is similar to our project.

EXAMPLE:

We have a set U (University areas) that needs to be covered with C (camera locations).

U={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}

C = {C1,C2,C3,C4,C5,C6,C7,C8}

C1={1,3,4,6,7}   C2={4,7,8,12}

C3={2,5,9,11,13}   C4={1,2,14,15}

C5={3,6,10,12,14}   C6={8,14,15}

C7={1,2,6,11}   C8={1,2,4,6,8,12}

The cost of each Camera Location is the same in this case, we just hope to minimize the total number of cameras used, so we can assume the cost of each C to be 1.
Let I represents set of elements included so far. Initialize I to be empty.

First Iteration:

The new element cost for C1=1/5,for C2=1/4,for C3=1/5,for C4=1/4,for C5=1/5,for C6=1/3,for C7=1/4,for C8=1/6
Since,C8 has minimum value,C8 is added, and I becomes {1,2,4,6,8,12}

Second Iteration:

I = {1,2,4,6,8,12}
The new element cost for C1=1/2,for C2=1/1,for C3=1/4,for C4=1/2,for C5=1/3,for C6=1/2,for C7=1/1
Since,C3 has minimum value,C3 is added, and I becomes {1,2,4,5,6,8,9,11,12,13}

Third Iteration:

I = {1,2,4,5,6,8,9,11,12,13}
The new element cost for C1=1/2,for C2 = 1/1,for C4=1/2,for C5=1/3,for C6=1/2,for C7=1/1
Since C5 as minimum value,C5 is added, and I becomes {1,2,3,4,5,6,8,9,10,11,12,13,14}

Fourth iteration:

I = {1,2,3,4,5,6,8,9,10,11,12,13,14}

The new element cost for C1=1/1,for C2=1/1,for C4=1/0,for C6=1/1,for C7=1/0

Since C1,C2,C6 all have meaningful and same values, we can choose either both C1 and C6 or both C2 and C6.As C1 or C2 add 7 to I, and C6 add 15 to I.

I becomes {1,2,4,5,6,7,8,9,10,11,12,13,14,15}

Now,

The solution obtained is:

Option 1: C8 + C3 + C5 + C6 + C1

Option 2: C8 + C3 + C5 + C6 + C2

The greedy algorithm does not provide the optimal solution in this case.

The usual elimination algorithm would give us the minimum number of cameras that we need to install to be4, but the greedy algorithm gives us the minimum number of cameras that we need to install is 5

6. IMPLEMENTATION

6.1. CODE EXPLANATION

Sets: Sets are one of standard python data type which store values. Set is a non-indexed sequence which doesn’t allow duplicate elements.

Used Python Built-in Functions:

lambda (): A lambda function is just like any normal python function, except that it has no name when defining it, and it is contained in one line of code.
len (): It returns the length of an object.
input (): The easiest way to obtain user input from the command-line is input function.

Logical operations performed:

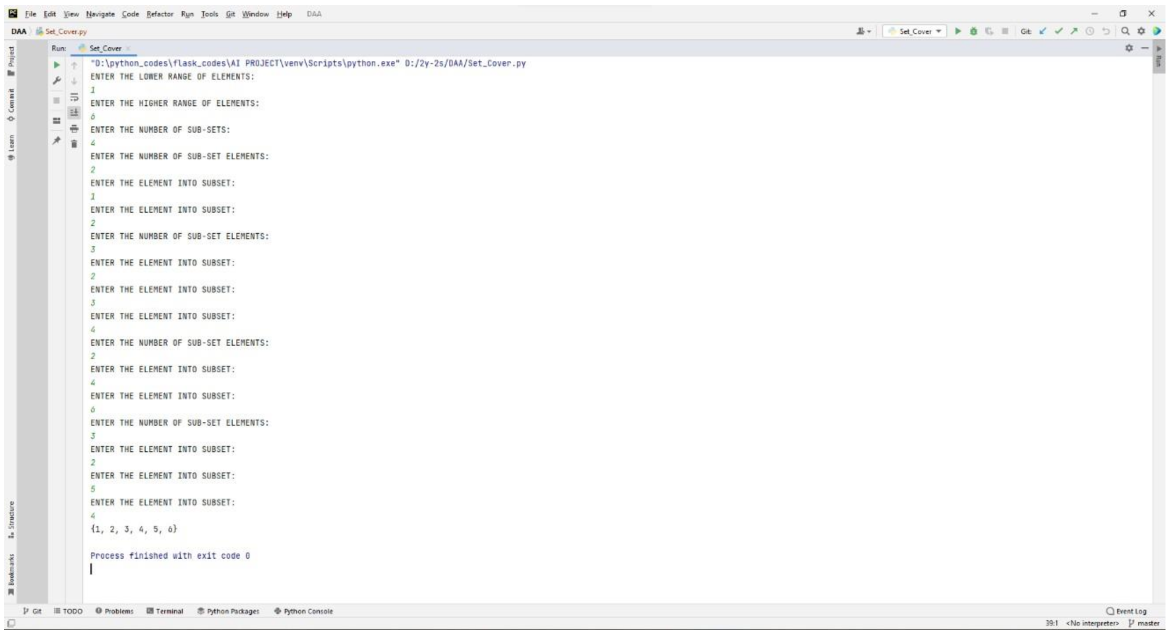
|= operator: The Python |= operator when applied to two Boolean values A and B performs the logical OR operation A | B and assigns the result to the first operand A.

6.2. CONSOLE CODE:

```
def set_cover(universe, subsets):
    elements = set(e for s in subsets for e in s)
    if elements != universe:
        return  None
    covered = set()
    cover = []
    while covered != elements:
        subset = max(subsets, key=lambda s: len(s - covered))
        cover.append(subset)
        covered |= subset
    return cover

def main():
    all_sets = []
    sub_set = set()
    print("ENTER THE LOWER RANGE OF ELEMENTS:")
    a = int(input())
    print("ENTER THE HIGHER RANGE OF ELEMENTS:")
    b = int(input())
    universe = set(range(a, b))
    print("ENTER THE NUMBER OF SUB-SETS:")
    size = int(input())
    for i in range(size):
        print("ENTER THE NUMBER OF SUB-SET ELEMENTS:")
        n = int(input())
        for j in range(n):
            print("ENTER THE ELEMENT INTO SUBSET:")
            element = int(input())
            sub_set.add(element)
            all_sets.append(sub_set)
            continue
    cover = set_cover(universe, all_sets)
    print(all_sets[1])
if__name__== '_main_':
    main()
```

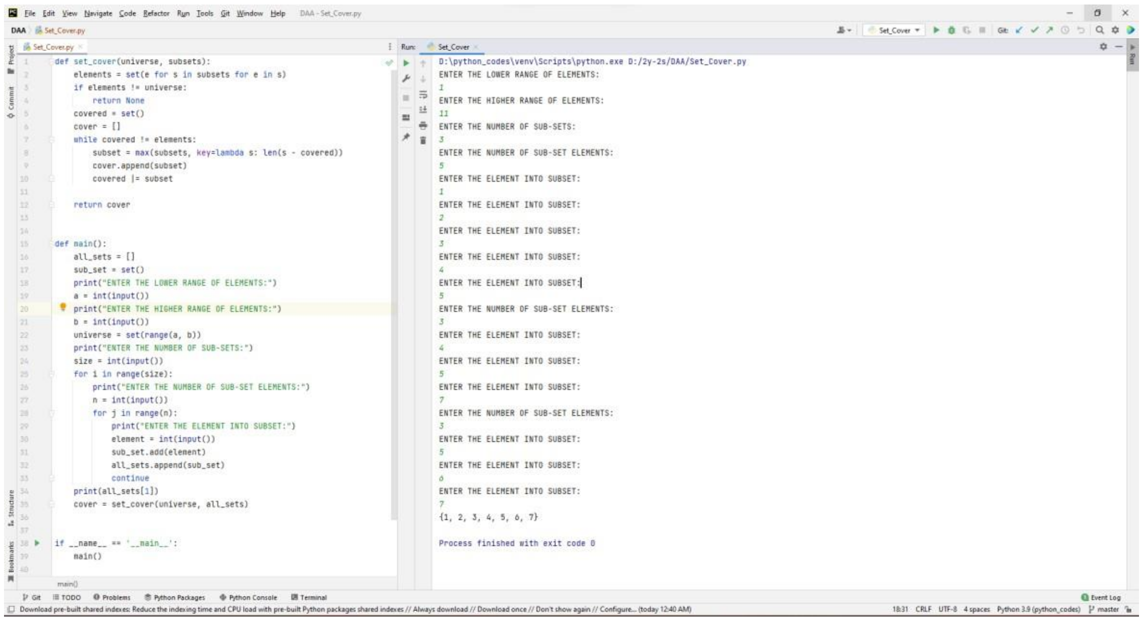
7. RESULTS & DISCUSSION



```
"D:\python_codes\flask_codes\AI PROJECT\venv\Scripts\python.exe" D:/2y-2s/DAA/Set_Cover.py
ENTER THE LOWER RANGE OF ELEMENTS:
1
ENTER THE HIGHER RANGE OF ELEMENTS:
6
ENTER THE NUMBER OF SUB-SETS:
4
ENTER THE NUMBER OF SUB-SET ELEMENTS:
2
ENTER THE ELEMENT INTO SUBSET:
1
ENTER THE ELEMENT INTO SUBSET:
2
ENTER THE NUMBER OF SUB-SET ELEMENTS:
3
ENTER THE ELEMENT INTO SUBSET:
2
ENTER THE ELEMENT INTO SUBSET:
3
ENTER THE ELEMENT INTO SUBSET:
4
ENTER THE NUMBER OF SUB-SET ELEMENTS:
2
ENTER THE ELEMENT INTO SUBSET:
4
ENTER THE ELEMENT INTO SUBSET:
6
ENTER THE NUMBER OF SUB-SET ELEMENTS:
3
ENTER THE ELEMENT INTO SUBSET:
2
ENTER THE ELEMENT INTO SUBSET:
5
ENTER THE ELEMENT INTO SUBSET:
4
{1, 2, 3, 4, 5, 6}

Process finished with exit code 0
```

Fig 7.1 : Console Output



```python
def set_cover(universe, subsets):
    elements = set(e for s in subsets for e in s)
    if elements != universe:
        return None
    covered = set()
    cover = []
    while covered != elements:
        subset = max(subsets, key=lambda s: len(s - covered))
        cover.append(subset)
        covered |= subset

    return cover


def main():
    all_sets = []
    sub_set = set()
    print("ENTER THE LOWER RANGE OF ELEMENTS:")
    a = int(input())
    print("ENTER THE HIGHER RANGE OF ELEMENTS:")
    b = int(input())
    universe = set(range(a, b))
    print("ENTER THE NUMBER OF SUB-SETS:")
    size = int(input())
    for i in range(size):
        print("ENTER THE NUMBER OF SUB-SET ELEMENTS:")
        n = int(input())
        for j in range(n):
            print("ENTER THE ELEMENT INTO SUBSET:")
            element = int(input())
            sub_set.add(element)
            all_sets.append(sub_set)
            continue
    print(all_sets[1])
    cover = set_cover(universe, all_sets)


if __name__ == '__main__':
    main()
```

```
D:\python_codes\venv\Scripts\python.exe D:/2y-2s/DAA/Set_Cover.py
ENTER THE LOWER RANGE OF ELEMENTS:
1
ENTER THE HIGHER RANGE OF ELEMENTS:
11
ENTER THE NUMBER OF SUB-SETS:
3
ENTER THE NUMBER OF SUB-SET ELEMENTS:
5
ENTER THE ELEMENT INTO SUBSET:
1
ENTER THE ELEMENT INTO SUBSET:
2
ENTER THE ELEMENT INTO SUBSET:
3
ENTER THE ELEMENT INTO SUBSET:
4
ENTER THE ELEMENT INTO SUBSET:
5
ENTER THE NUMBER OF SUB-SET ELEMENTS:
3
ENTER THE ELEMENT INTO SUBSET:
4
ENTER THE ELEMENT INTO SUBSET:
5
ENTER THE ELEMENT INTO SUBSET:
7
ENTER THE NUMBER OF SUB-SET ELEMENTS:
3
ENTER THE ELEMENT INTO SUBSET:
5
ENTER THE ELEMENT INTO SUBSET:
6
ENTER THE ELEMENT INTO SUBSET:
7
{1, 2, 3, 4, 5, 6, 7}

Process finished with exit code 0
```

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Download once // Don't show again // Configure... (today 12:40 AM)

Fig 7.2 : Console Output

8. CONCLUSION AND FUTURE WORK

8.1. CONCLUSION

By our system we proposed, an optimal location to open any kind of business can be obtained. Especially in the case of a coffee shop. Not only providing the user with the optimal location but also enhanced user interface to make things easier. This optimal location is provided after verifying all the surrounding areas and extent of people that business could cover. Therefore, we conclude that an optimal point can be obtained with the help of Set Cover Problem.

8.2. FUTURE WORK

For further part we would like to use google API's and help the user to locate a place where it covers all the subset places so that when a coffee shop is opened maximum profit can be obtained at that optimal location on maps. This also enhances the user interface

9. REFERENCES

[1] Sheng, Haiyun, Donglei Du, Yuefang Sun, Jian Sun, and Xiaoyan Zhang. "Approximation Algorithm for Stochastic Set Cover Problem." In International Conference on Algorithmic Applications in Management, pp. 37-48. Springer, Cham, 2020.

[2] Chen, Wenbin, Fufang Li, Ke Qi, Miao Liu, and Maobin Tang. "A Primal-Dual Randomized Algorithm for the Online Weighted Set Multi-Cover Problem." In International Conference on Theory and Applications of Models of Computation, pp. 60-68. Springer, Cham, 2020.

[3] Mitrović, Slobodan, and Ronitt Rubinfeld. "Improved local computation algorithm for set cover via sparsification." (2020).

[4] Kritter, Julien, Mathieu Brévilliers, Julien Lepagnot, and Lhassane Idoumghar. "On the optimal placement of cameras for surveillance and the underlying set cover problem." Applied Soft Computing 74 (2019): 133-153.

[5] Lin, Bingkai. "A simple gap-producing reduction for the parameterized set cover problem." arXiv preprint arXiv:1902.03702 (2019).

[6] Burhan, Hussein M., A. Attea Bara'a, Amenah D. Abbood, Mustafa N. Abbas, and Mayyadah Al-Ani. "Evolutionary multi-objective set cover problem for task allocation in the Internet of Things." Applied Soft Computing 102 (2021):107097.

[7] Gupta, Anupam, and Roie Levin. "The online submodular cover problem." In Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1525-1537. Society for Industrial and Applied Mathematics, 2020.