Name : Sonal Dilip Gholap

UID : 2021600020

Branch : CSE-AIML

Batch : B

Experiment : 9

Aim : Experiment based on Approximation Algorithm (Set Cover Problem).

- **Objectives :**

1. To study approximation algorithms like set covering problem.
2. To implement set covering problem using greedy approximation method.
3. To derive and analyze the time complexity of set covering problem.

- **Theory :**

1. NP-completeness is a concept in computational complexity theory that refers to the difficulty of solving certain computational problems.
2. A problem is said to be NP-complete if it belongs to the complexity class NP (nondeterministic polynomial time) and any problem in NP can be reduced to it in polynomial time.
3. In other words, if a problem is NP-complete, it is one of the hardest problems in the class NP, and there is no known algorithm that can solve it efficiently in the worst case.
4. An approximation algorithm is a way of dealing with NP-completeness for an optimization problem. This technique does not guarantee the best solution.
5. The goal of the approximation algorithm is to come as close as possible to the optimal solution in polynomial time. Such algorithms are called approximation algorithms or heuristic algorithms.
6. An approximation algorithm guarantees to run in polynomial time though it does not guarantee the most effective solution.
7. The performance ratio of an approximation algorithm is defined as the ratio of the solution produced by the algorithm to the optimal solution. A good approximation algorithm has a performance ratio that is as close to 1 as possible.

8. Many approximation algorithms use randomness to improve their performance. Randomization can be used to speed up the algorithm, to reduce the approximation error, or both.
9. Many approximation algorithms use a greedy approach, which means that they make locally optimal choices at each step in the hope of obtaining a globally optimal solution.

- **Set covering problem**

Pseudocode for set covering problem

setCover(U, S, cost):

```
while U is not empty:
    pick a subset Si with the smallest alpha from S
    for each uncovered element e in U:
        if e is in Si:
            set the price of e to alpha
            remove e from U
        if elements were covered:
            add Si to the solution set C
            add the cost of Si to the total cost

return C and the total cost
```

Derivation of Time Complexity

	Time
	n
while U is not empty :	$n(m \log m)$
pick a subset S_i with smallest alpha	$n(nm)$
for each uncovered element e in U	$n(1)$
set price$(e) = \alpha$	$n(n)$
remove e from U	$n(1)$
if elements were covered :	$n(1)$
add S_i to C	$n(1)$
total += cost of S_i	

Sorting subsets by alpha takes $O(|S| \log |S|)$ time.

Max no. of times while loop runs = n where n is the no. of elements in U.

Max. no. of times for loop runs = mn where m is the no. of subsets.

$\therefore$ Time complexity = $O(n^2 m)$

Algorithm has a polynomial time complexity

$T(n) = n + nm \log m + n^2 m + n + n^2 + n + n + n$

$T(n) = n^2 m + nm \log m + n^2 + 5n$

$\therefore T(n) = O(n^2 m)$ which is polynomial in nature

Solution :

* Approximation algorithms

$U = \{1, 2, 3, 4, 5\}$ $\quad$ $S = \{S1, S2, S3\}$

$S1 = \{4, 1, 3\}$ $\quad$ $cost(S1) = 5$
$S2 = \{2, 5\}$ $\quad$ $cost(S2) = 10$
$S3 = \{1, 4, 3, 2\}$ $\quad$ $cost(S3) = 3$

$$\alpha_{S1} = \frac{c(S1)}{|S1 \cap U|} = \frac{5}{3}$$

$$\alpha_{S2} = \frac{c(S2)}{|S2 \cap U|} = \frac{10}{2} = 5$$

$$\alpha_{S3} = \frac{c(S3)}{|S3 \cap U|} = \frac{3}{4}$$

$$\alpha_{S3} < \alpha_{S1} < \alpha_{S2}$$

Pick S3 as it has the least α
Elements in $S3 = \{1, 4, 3, 2\}$

$cost(1) = \alpha_{S3} = 3/4$
$cost(4) = \alpha_{S3} = 3/4$
$cost(3) = \alpha_{S3} = 3/4$
$cost(2) = \alpha_{S3} = 3/4$

$\therefore$ cost of $S3 = \frac{3 \times 4}{4} = 3$

Erase $\{1, 4, 3, 2\}$ from U

Now, $U = \{5\}$

Next pick S1 because of its small α.
Elements in S1 = {4, 1, 3}
These elements are already covered & not present in U
so move on to the next option

$S2 = \{2, 5\}$

$cost(2) = \alpha_{S2} = 10/2 = 5$
$cost(5) = \alpha_{S2} = 5$

cost of S2 = 5×2
$= 10$

Erase {2,5} from U.
Now, U is empty.

Minimum cost = cost of S3 + cost of S2
= 3 + 10
Minimum cost = 13

Code :

```
#include<stdio.h>
#include<vector>
#include<bits/stdc++.h>
#include<iterator>
#include<iostream>

using namespace std;

//function to sort a vector of pairs based on the second element
bool sortBySecond(const pair <vector<int>,int>  &a, const pair <vector<int>,int>  &b){
    return (a.second < b.second);
}

int main(){

    vector<int> U = {1,2,3,4,5};

    cout<<"Universal Set: ";
```

```
    for(int i = 0; i < U.size(); i++){
        cout<<U[i]<<" ";
    }

    vector<vector<int>> S;

    vector<int> s1 = {4,1,3};
    vector<int> s2 = {2,5};
    vector<int> s3 = {1,4,3,2};

    S.push_back(s1);
    S.push_back(s2);
    S.push_back(s3);

    vector<int> cost = {5,10,3};
    vector<int> C;
    vector<float> alpha;

    for(int i = 0; i < 3; i++){
        float val = (float)cost[i]/(float)S[i].size();
        alpha.push_back(val);
    }

    vector<pair<vector<int>,float> > v;       //pair of {set, alpha}

    for(int i = 0; i < 3; i++){
        v.push_back(make_pair(S[i],alpha[i]));
    }

    sort(v.begin(),v.end(),sortBySecond);

    int total = 0;
    vector<int> setIndices;

        while (!U.empty()) {

            vector<int> Si = v.front().first;   //get the subset with the smallest alpha
            float alp = v.front().second;
            v.erase(v.begin());

            bool covered = false;

            for (int j = 0; j < Si.size(); j++) {   //check if element is already covered
                if (find(U.begin(), U.end(), Si[j]) != U.end()) {
                    //total += alp*Si.size();
                    covered = true;
                    U.erase(find(U.begin(), U.end(), Si[j]));  //remove the covered element
from U
                    //break;
                }
```

```cpp
            }

            if(covered){

                total += alp*Si.size();  //add the set's price to total cost

                auto it = find(S.begin(),S.end(),Si);
                setIndices.push_back(distance(S.begin(),it)+1);
            }

        }


    cout<<"\nGiven subsets: ";
    for(int i = 0; i < S.size(); i++){
        cout<<endl;
        cout<<"S"<<i+1<<" = ";

        vector<int> p = S[i];
        for(int j = 0; j < S[i].size(); j++){
            cout<<p[j]<<" ";
        }
    }

    cout<<"\nCost of each subset: ";
    for(int i = 0; i < cost.size(); i++){
        cout<<cost[i]<<" ";
    }

    cout<<"\n\nThe sets taken into consideration are: ";
    for (int i = 0; i < setIndices.size(); i++) {
        cout<<"S"<<setIndices[i]<<" ";
    }
    cout<<"\nThe minimum set cost is: "<<total<< endl;

    return 0;
}
```

Output :

```
PS C:\C++ Learning Course>  & 'c:\Users\Sonal Dilip Gholap\.vscode\extensions\ms-vscode.cpptoo
ls-1.15.2-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In
-4jezcuwb.vhr' '--stdout=Microsoft-MIEngine-Out-wf5l3oem.lom' '--stderr=Microsoft-MIEngine-Err
or-x1w54pnl.iap' '--pid=Microsoft-MIEngine-Pid-izmwzhc1.ca2' '--dbgExe=C:\Program Files (x86)\
mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\mingw32\bin\gdb.exe' '--interpreter=mi'
Universal Set: 1 2 3 4 5
Given subsets:
S1 = 4 1 3
S2 = 2 5
S3 = 1 4 3 2
Cost of each subset: 5 10 3

The sets taken into consideration are: S3 S2
The minimum set cost is: 13
PS C:\C++ Learning Course>
```

Result :

1. Time complexity of set cover problem – **O(n^2m)**

Conclusion :

1. Approximation algorithms are a powerful tool for solving hard optimization problems.
2. Approximation algorithms guarantee to run in polynomial time though they do not guarantee the most effective solution.
3. Approximation algorithms are used to get an answer near the optimal solution of an optimization problem in polynomial time.