

WEEK 8

1. Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm which will find minimum cost required to connect these cities. (use Prim's algorithm)

```
#include <iostream>
#include <vector>
#include <fstream>
#include <set>
using namespace std;

// Prim's algorithm function
int prim(vector<vector<pair<int,int>>>& adj, int N, int source) {
    vector<bool> vis(N, false);
    set<pair<int, int>> s;
    s.insert({0, source});
    int ans = 0;
    while (!s.empty()) {
        auto node = *s.begin();
        s.erase(s.begin());
        int v = node.second;
        int wt = node.first;
        if (vis[v]) continue;
        ans += wt;
        vis[v] = true;
        for (auto child : adj[v]) {
            int child_v = child.second;
            int child_wt = child.first;
            if (!vis[child_v]) {
                s.insert({child_wt, child_v});
            }
        }
    }
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
}  
return ans;  
}  
  
int main() {  
  
                                                                    // Open input and output files  
  
    ifstream fin("input.txt");  
    ofstream fout("output.txt");  
  
                                                                    // Check if files are opened successfully  
  
    if (!fin.is_open() || !fout.is_open()) {  
        cout << "Error occurred while opening files.\n";  
        return 0;  
    }  
  
                                                                    // Input graph  
  
    int vertices, edges;  
    fin >> vertices >> edges;  
    vector<vector<pair<int,int>>> adj(vertices);  
  
    for (int i = 0; i < edges; i++) {  
        int u, v, w;  
        fin >> u >> v >> w;  
        adj[u].push_back({v, w});  
        adj[v].push_back({u, w});  
    }  
    int source = 0;  
    fout << "Minimum spanning tree weight : " << prim(adj, vertices, source);  
                                                                    // Close input and output files  
  
    fin.close();  
    fout.close();  
  
    return 0;  
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

main.cpp	input.txt	:	output.txt	:
1	7 9			
2	5 4 9			
3	1 4 1			
4	5 1 4			
5	4 3 5			
6	4 2 3			
7	1 2 2			
8	3 2 3			
9	3 6 8			
10	2 6 7			
11				

main.cpp	input.txt	:	output.txt	:
1	Minimum spanning tree weight : 17			

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

2 . Implement the previous problem using Kruskal's algorithm.

```
#include <iostream>
#include <vector>
#include <fstream>
#include <queue>
using namespace std;

// Find function for disjoint set
int find(vector<int>& ds, int u) {
    if (u == ds[u]) return u;
    return ds[u] = find(ds, ds[u]);
}

// Union function for disjoint set
void unionof(vector<int>& ds, int u, int v) {
    int p1 = find(ds, u);
    int p2 = find(ds, v);
    if (p1 != p2) {
        ds[p2] = p1;
    }
}

// Kruskal's algorithm to find the weight of the minimum spanning tree
int kruskal(vector<vector<pair<int, int>>>& adj, int V) {
    vector<int> ds(V);
    for (int i = 0; i < V; i++) {
        ds[i] = i;
    }
    priority_queue<pair<int, pair<int, int>>> q;
    for (int i = 0; i < V; i++) {
        for (auto j : adj[i]) {
            if (i < j.first) {
                q.push({-j.second, {i, j.first}});
            }
        }
    }
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
int ans = 0;
while (!q.empty()) {
    int w = q.top().first;
    int u = q.top().second.first;
    int v = q.top().second.second;
    q.pop();

    if (find(ds, u) != find(ds, v)) {
        unionof(ds, u, v);
        ans += (-w);
    }
}
return ans;
}

int main() {

    // Open input and output files

    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully

    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }

    // Input graph

    int vertices, edges;
    fin >> vertices >> edges;
    vector<vector<pair<int, int>>> adj(vertices);
    for (int i = 0; i < edges; i++) {
        int u, v, w;
        fin >> u >> v >> w;
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});
    }
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
fout << "Minimum spanning tree weight : " << kruskal(adj, vertices);
```

```
// Close input and output files
```

```
fin.close();
```

```
fout.close();
```

```
return 0;
```

```
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

main.cpp	input.txt	:	output.txt	:
1	7 9			
2	5 4 9			
3	1 4 1			
4	5 1 4			
5	4 3 5			
6	4 2 3			
7	1 2 2			
8	3 2 3			
9	3 6 8			
10	2 6 7			
11				

main.cpp	input.txt	:	output.txt	:
1	Minimum spanning tree weight : 17			

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

3 . Assume that same road construction project is given to another person. The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project

```
#include <iostream>
#include <vector>
#include <fstream>
#include <queue>
using namespace std;

// Find function for disjoint set

int find(vector<int>& ds, int u) {
    if (u == ds[u]) return u;
    return ds[u] = find(ds, ds[u]);
}

// Union function for disjoint set

void unionof(vector<int>& ds, int u, int v) {
    int p1 = find(ds, u);
    int p2 = find(ds, v);
    if (p1 != p2) {
        ds[p2] = p1;
    }
}

// Kruskal's algorithm to find the weight of the maximum spanning tree

int kruskal(vector<vector<pair<int, int>>>& adj, int V) {
    vector<int> ds(V);
    for (int i = 0; i < V; i++) {
        ds[i] = i;
    }
    priority_queue<pair<int, pair<int, int>>> q;
    for (int i = 0; i < V; i++) {
        for (auto j : adj[i]) {
```


DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
        if (i < j.first) { // Avoid duplicating edges in undirected graph
            q.push({j.second, {i, j.first}});
        }
    }
}

int ans = 0;
while (!q.empty()) {
    int w = q.top().first;
    int u = q.top().second.first;
    int v = q.top().second.second;
    q.pop();

    if (find(ds, u) != find(ds, v)) {
        unionof(ds, u, v);
        ans += w;
    }
}
return ans;
}

int main() {

                                                                    // Open input and output files

    ifstream fin("input.txt");
    ofstream fout("output.txt");

                                                                    // Check if files are opened successfully

    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }

                                                                    // Input graph

    int vertices, edges;
    fin >> vertices >> edges;
    vector<vector<pair<int, int>>> adj(vertices);
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
for (int i = 0; i < edges; i++) {  
    int u, v, w;  
    fin >> u >> v >> w;  
    adj[u].push_back({v, w});  
    adj[v].push_back({u, w});  
}
```

```
fout << "Minimum spanning tree weight : " << kruskal(adj, vertices);
```

```
// Close input and output files
```

```
fin.close();
```

```
fout.close();
```

```
return 0;
```

```
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1  7 9
2  5 4 9
3  1 4 1
4  5 1 4
5  4 3 5
6  4 2 3
7  1 2 2
8  3 2 3
9  3 6 8
10 2 6 7
11 |
```

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1 | Minimum spanning tree weight : 33
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

WEEK 9

1. Given a graph, Design an algorithm and implement it using a program to implement Floyd-Warshall all pair shortest path algorithm. Input Format: The first line of input takes number of vertices in the graph. 2 Input will be the graph in the form of adjacency matrix or adjacency list. If a direct edge is not present between any pair of vertex (u,v), then this entry is shown as AdjM[u,v] = INF.

```
#include <iostream>
#include <vector>
#include <fstream>
#include <climits>
using namespace std;

                                                                    //floyd

void floydWarshall(vector<vector<int>>& dist, int N, ofstream &fout) {
    for (int k = 0; k < N; k++) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX) {
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
                }
            }
        }
    }
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (dist[i][j] == INT_MAX)
                fout << "INF ";
            else
                fout << dist[i][j] << " ";
        }
        fout << endl;
    }
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
int main() {  
  
                                                                    // Open input and output files  
    ifstream fin("input.txt");  
    ofstream fout("output.txt");  
  
                                                                    // Check if files are opened successfully  
    if (!fin.is_open() || !fout.is_open()) {  
        cout << "Error occurred while opening files.\n";  
        return 0;  
    }  
  
                                                                    // Input graph  
    int vertices, edges;  
    fin >> vertices >> edges;  
    vector<vector<int>> adj(vertices, vector<int>(vertices, INT_MAX));  
  
                                                                    // Initialize distances with self-loops  
    for (int i = 0; i < vertices; i++) {  
        adj[i][i] = 0;  
    }  
  
                                                                    // Read the edges  
    for (int i = 0; i < edges; i++) {  
        int u, v, w;  
        fin >> u >> v >> w;  
        u--; v--; // Adjusting for 0-based index  
        adj[u][v] = w;  
    }  
  
                                                                    // Run the Floyd-Warshall algorithm  
    floydWarshall(adj, vertices, fout);  
  
                                                                    // Close input and output files  
    fin.close();  
    fout.close();  
    return 0;  
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

main.cpp	input.txt	:	output.txt	:
1	5		9	
2	1	2	10	
3	1	3	5	
4	1	4	5	
5	2	3	5	
6	2	4	5	
7	2	5	5	
8	3	5	10	
9	4	5	20	
10	5	4	5	

main.cpp	input.txt	:	output.txt	:	
1	0	10	5	5	15
2	INF	0	5	5	5
3	INF	INF	0	15	10
4	INF	INF	INF	0	20
5	INF	INF	INF	5	0
6					
7					

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

2 . Given a knapsack of maximum capacity w . N items are provided, each having its own value and weight. You have to Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight w and has maximum value. You can take fractions of items,i.e. the items can be broken into smaller pieces so that you have to carry only a fraction x_i of item i , where $0 \leq x_i \leq 1$

```
#include <iostream>
#include <vector>
#include <fstream>
#include <climits>
using namespace std;
bool cmp(pair<int, int> a, pair<int, int> b) {
    int val1 = a.second/ a.first;
    int val2 = b.second/ b.first;
    return val1 > val2;
}

int main() {
    // Open input and output files
    ifstream fin("input.txt");
    ofstream fout("output.txt");
    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    vector<pair<int, int>> fk;
    int n; fin >> n;
    for(int i = 0; i < n; i++) {
        int w, v;
        fin >> w >> v;
        fk.push_back({w, v});
    }
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
int weight; fin>>weight;
sort(fk.begin(), fk.end(), cmp);
int currWeight = 0;
double ans = 0.0;
for(int i = 0; i < n; i++) {
    if(currWeight + fk[i].first <= weight) {
        currWeight += fk[i].first;
        ans += fk[i].second;
    }
    else {
        int remain = weight - currWeight;
        ans += (fk[i].second / (double(fk[i].first))) * double(remain);
        break;
    }
}
fout<<ans;

                                                                    // Close input and output files

fin.close();
fout.close();

return 0;
}
```


DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1  6
2  6 6
3  2 10
4  1 3
5  8 5
6  3 1
7  5 3
8  16
```

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1  23.375
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

3 . Given an array of elements. Assume $arr[i]$ represents the size of file i . Write an algorithm and a program to merge all these files into single file with minimum computation. For given two files A and B with sizes m and n , computation cost of merging them is $O(m+n)$. (Hint: use greedy approach)

```
#include <iostream>
#include <vector>
#include <fstream>
#include <queue>
using namespace std;

int main() {
    // Open input and output files
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    int n;
    fin >> n;

    // Min-heap to store the sizes of the files
    priority_queue<int, vector<int>, greater<int>>> q;

    // Read file sizes into the min-heap
    for (int i = 0; i < n; i++) {
        int x;
        fin >> x;
        q.push(x);
    }
    int ans = 0;

    // Continue merging until we have only one file left
    while (q.size() > 1) {
        int x = q.top();
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
q.pop();
int y = q.top();
q.pop();
int mergedSize = x + y;
ans += mergedSize;
q.push(mergedSize);
}

fout << ans;

// Close input and output files

fin.close();
fout.close();

return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

main.cpp	input.txt	:	output.txt	:
1	10			
2	10 5 100 50 20 15 5 20 100 10			
3				

main.cpp	input.txt	:	output.txt	:
1	895			

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

WEEK 10

1. Given a list of activities with their starting time and finishing time. Your goal is to select maximum number of activities that can be performed by a single person such that selected activities must be non-conflicting. Any activity is said to be non-conflicting if starting time of an activity is greater than or equal to the finishing time of the other activity. Assume that a person can only work on a single activity at a time.

```
#include <iostream>
#include <vector>
#include <fstream>
#include <queue>
using namespace std;

int main() {
    // Open input and output files
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    vector<int> ans;
    vector<pair<int, int>> as;
    int n; fin >> n;
    vector<int> v1(n), v2(n);
    for(int i = 0; i < n; i++) fin >> v1[i];
    for(int i = 0; i < n; i++) fin >> v2[i];
    for(int i = 0; i < n; i++) {
        as.push_back({v2[i], v1[i]});
    }
    sort(as.begin(), as.end());
    int curEnd = -1;
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
for(int i = 0; i < n; i++) {
    if(as[i].second > curEnd) {
        ans.push_back(i + 1);
        curEnd = as[i].first;
    }
}
fout<<"No. of non-conflicting activities: "<<ans.size()<<endl;
fout<<"List of selected activities: ";
for(auto it : ans)
    fout<<it<<" ";

                                                                    // Close input and output files

fin.close();
fout.close();

return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

main.cpp	input.txt	output.txt
1	10	
2	1 3 0 5 3 5 8 8 2 12	
3	4 5 6 7 9 9 11 12 14 16	
4		

main.cpp	input.txt	output.txt
1	No. of non-conflicting activities: 4	
2	List of selected activities: 1 4 7 10	

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

2. Given a long list of tasks. Each task takes specific time to accomplish it and each task has a deadline associated with it. You have to design an algorithm and implement it using a program to find maximum number of tasks that can be completed without crossing their deadlines and also find list of selected tasks

```
#include <iostream>
#include <vector>
#include <fstream>
#include <algorithm> // for sort
using namespace std;

int main() {

                                                                    // Open input and output files

    ifstream fin("input.txt");
    ofstream fout("output.txt");

                                                                    // Check if files are opened successfully

    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    int n;
    fin >> n;
    vector<int> time(n);
    for (int i = 0; i < n; i++) {
        fin >> time[i];
    }
    vector<pair<int, int>> arr(n);
    for (int i = 0; i < n; i++) {
        fin >> arr[i].first;
        arr[i].second = i;
    }
    sort(arr.begin(), arr.end());
    int ans = 0, currtime = 0;
```


DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
vector<int> an;
for (int i = 0; i < n; i++) {
    if (currtime + time[arr[i].second] <= arr[i].first) {
        ans++;
        an.push_back(arr[i].second);
        currtime += time[arr[i].second];
    }
}
fout << "Max number of tasks = " << ans << endl;
sort(an.begin(), an.end());
fout << "Selected task numbers: ";
for (int i : an) fout << i + 1 << ' ';
fout << endl;

                                                                    // Close input and output files

fin.close();
fout.close();

return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1  7
2  2 1 3 2 2 2 1
3  2 3 8 6 2 5 3
```

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1  Max number of tasks = 4
2  Selected task numbers: 1 2 3 6
3
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

3. Given an unsorted array of elements, design an algorithm and implement it using a program to find whether majority element exists or not. Also find median of the array. A majority element is an element that appears more than $n/2$ times, where n is the size of array.

```
#include <iostream>
#include <vector>
#include <fstream>
#include <algorithm>
using namespace std;

int main() {

                                                                    // Open input and output files

    ifstream fin("input.txt");
    ofstream fout("output.txt");

                                                                    // Check if files are opened successfully

    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    int n, c = 0, mxC = 0, i;
    fin >> n;
    vector<int> v(n);
    for(int i = 0; i < n; i++)
        fin >> v[i];
    int median = v[n/2];
    sort(v.begin(), v.end());
    for(i = 0; i < n - 1; i++) {
        if(v[i] == v[i + 1])
            c++;
        else {
            mxC = max(mxC, c);
            c = 0;
        }
    }
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
if(v[i] == v[i - 1])
    mxC++;
if(mxC > (n / 2))
    fout<<"Yes";
else
    fout<<"No";
fout<<endl<<median;

// Close input and output files

fin.close();
fout.close();

return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

main.cpp	input.txt	output.txt
1	9	
2	4 4 2 3 2 2 3 2 2	

main.cpp	input.txt	output.txt
1	Yes	
2	2	

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

WEEK 11

1. Given a sequence of matrices, write an algorithm to find most efficient way to multiply these matrices together. To find the optimal solution, you need to find the order in which these matrices should be multiplied.

```
#include <iostream>
#include <vector>
#include <climits>
#include <fstream>
using namespace std;

                                                                    //chain multiply

int matrixChainOrder(const vector<int> &dims, int n) {
    vector<vector<int>> dp(n, vector<int>(n, 0));

    for (int l = 2; l < n; l++) {
        for (int i = 1; i < n - l + 1; i++) {
            int j = i + l - 1;
            dp[i][j] = INT_MAX;
            for (int k = i; k <= j - 1; k++) {
                int q = dp[i][k] + dp[k + 1][j] + dims[i - 1] * dims[k] * dims[j];
                if (q < dp[i][j]) {
                    dp[i][j] = q;
                }
            }
        }
    }
    return dp[1][n - 1];
}

int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");

                                                                    // Check if files are opened successfully

    if (!fin.is_open() || !fout.is_open()) {
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
    cout << "Error occurred while opening files.\n";
    return 0;
}

int n;
fin >> n;

vector<int> dims(n + 1);
for (int i = 0; i <= n; i++) {
    fin >> dims[i];
}

int result = matrixChainOrder(dims, n + 1);
fout << "Minimum number of multiplications required: " << result << endl;

fin.close();
fout.close();

return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

main.cpp	input.txt	:	output.txt	:
1	3			
2	10 30			
3	30 5			
4	5 60			

main.cpp	input.txt	:	output.txt	:
1	Minimum number of multiplications required: 4500			
2				

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

2. Given a set of available types of coins. Let suppose you have infinite supply of each type of coin. For a given value N, you have to Design an algorithm and implement it using a program to find number of ways in which these coins can be added to make sum value equals to N.

```
#include <iostream>
#include <vector>
#include <climits>
#include <fstream>
using namespace std;

//function
void search(vector<vector<int>> &ans, vector<int> &temp, int target, int idx, vector<int> &arr) {
    if(target < 0)
        return;
    if(target == 0) {
        ans.push_back(temp);
        return;
    }
    for(int i = idx; i < arr.size(); i++) {
        temp.push_back(arr[i]);
        search(ans, temp, target - arr[i], i, arr);
        temp.pop_back();
    }
}

int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
int n;
fin>>n;
vector<int> arr(n);
for(int i = 0; i < n; i++)
    fin>>arr[i];
int target;
fin>>target;
vector<vector<int>> ans;
vector<int> temp;
search(ans, temp, target, 0, arr);
fout<<ans.size();

fin.close();
fout.close();

return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

main.cpp	input.txt	output.txt
1	4	
2	2 5 6 3	
3	10	

main.cpp	input.txt	output.txt
1	5	

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

3. Given a set of elements, you have to partition the set into two subsets such that the sum of elements in both subsets is same. Design an algorithm and implement it using a program to solve this problem

```
#include <iostream>
#include <vector>
#include <climits>
#include <fstream>
using namespace std;

//partition

bool canPartition(const vector<int>& nums) {
    int n = nums.size();
    int totalSum = 0;
    for (int num : nums) {
        totalSum += num;
    }
    if (totalSum % 2 != 0) {
        return false;
    }
    int subsetSum = totalSum / 2;
    vector<vector<bool>> dp(n + 1, vector<bool>(subsetSum + 1, false));

    for (int i = 0; i <= n; ++i) {
        dp[i][0] = true;
    }
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= subsetSum; ++j) {
            if (nums[i - 1] <= j) {
                dp[i][j] = dp[i - 1][j] || dp[i - 1][j - nums[i - 1]];
            } else {
                dp[i][j] = dp[i - 1][j];
            }
        }
    }
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
    return dp[n][subsetSum];
}

int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }

    int n;
    fin >> n;

    vector<int> nums(n);
    for (int i = 0; i < n; ++i) {
        fin >> nums[i];
    }

    if (canPartition(nums)) {
        fout << "yes" << endl;
    } else {
        fout << "no" << endl;
    }

    fin.close();
    fout.close();

    return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

main.cpp	input.txt	:	output.txt	:
1	7			
2	1			
3	5			
4	4			
5	11			
6	5			
7	14			
8	10			
9				

main.cpp	input.txt	:	output.txt	:
1	yes			
2				

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

WEEK 12

1. Given two sequences, Design an algorithm and implement it using a program to find the length of longest subsequence present in both of them. A 2 subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

```
#include <iostream>
#include <vector>
#include <climits>
#include <fstream>
using namespace std;
int lcs(string x, string y)
{
    int n = x.size();
    int m = y.size();

    int res[n+1][m+1];
    for(int i = 0; i <= n; i++)
    {
        for(int j = 0; j <= m; j++)
        {
            if(i == 0 || j == 0)
            {
                res[i][j] = 0;
            }
            else if(x[i - 1] == y[j - 1])
            {
                res[i][j] = res[i-1][j-1] + 1;
            }
            else
            {
                res[i][j] = max(res[i - 1][j], res[i][j - 1]);
            }
        }
    }
    cout << "The longest subsequence: ";
```


DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
for(int j = 1; j<= m; j++)
{
    if(res[n][j] != res[n][j - 1])
    {
        cout << x[j - 1];
    }
}
return res[n][m];
}

int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }

    string str1, str2;
    fin>>str1>>str2;

    int x = lcs(str1, str2);
    fout << "\nThe length of the longest common subsequence: " << x << endl;

    fin.close();
    fout.close();

    return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1  AGGTAB
2  GXTXAYB
3
```

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1
2  The length of the longest common subsequence: 4
3
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

2. .Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight $\leq w$ and has maximum value. Here, you cannot break an item i.e. either pick the complete item or don't pick it. (0-1 property).

```
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;
```

// 0/1 Knapsack

```
int KnapDP(vector<int>& P, vector<int>& W, int n, int k) {
    vector<vector<int>> sol(n + 1, vector<int>(k + 1, 0));
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= k; j++) {
            if (W[i - 1] > j) {
                sol[i][j] = sol[i - 1][j];
            } else {
                sol[i][j] = max(sol[i - 1][j], P[i - 1] + sol[i - 1][j - W[i - 1]]);
            }
        }
    }
    return sol[n][k];
}
```

```
int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully

    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
int n;
fin >> n;
vector<int> P(n);
vector<int> W(n);

for (int i = 0; i < n; i++) {
    fin >> P[i] >> W[i];
}

int capacity;
fin >> capacity;

int res = KnapDP(P, W, n, capacity);
fout << "Total Profit: " << res << endl;

fin.close();
fout.close();

return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1  5
2  2 1
3  3 2
4  3 5
5  4 9
6  6 4
7  10
```

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1  Total Profit: 16
2
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

3. Given a string of characters, design an algorithm and implement it using a program to print all possible permutations of the string in lexicographic order.

```
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;

//permutes

void permute(string &s, int l, int r, vector<string> &result) {
    if (l == r) {
        result.push_back(s);
    } else {
        for (int i = l; i <= r; i++) {
            swap(s[l], s[i]);
            permute(s, l + 1, r, result);
            swap(s[l], s[i]);
        }
    }
}

int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }

    string str;
    fin >> str;
    sort(str.begin(), str.end());
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)


```
vector<string> result;
permute(str, 0, str.size() - 1, result);
sort(result.begin(), result.end());

for (const string &perm : result) {
    fout << perm << endl;
}

fin.close();
fout.close();

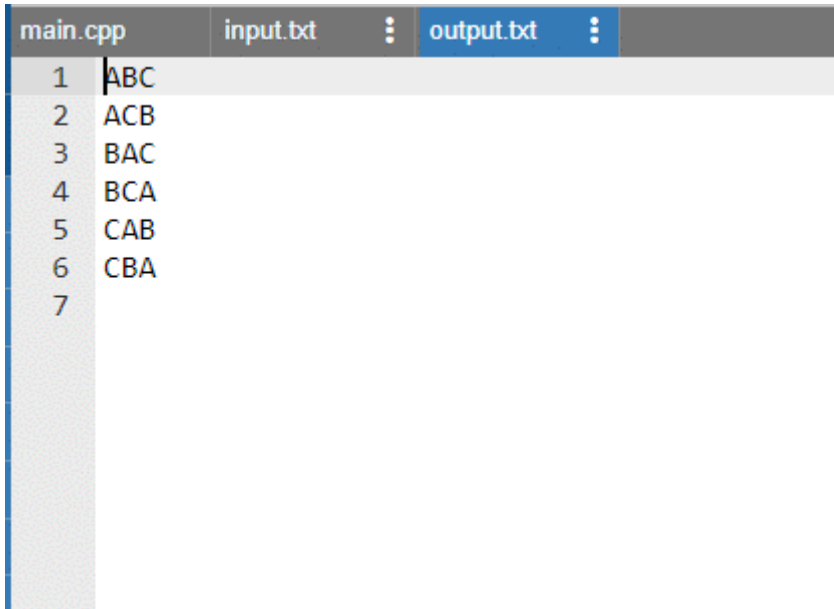
return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)



A screenshot of a code editor window. The title bar shows three tabs: 'main.cpp', 'input.txt', and 'output.txt'. The 'input.txt' tab is active and highlighted in blue. The code area shows a single line of text: '1 CAB'.

```
1 CAB
```



A screenshot of a code editor window. The title bar shows three tabs: 'main.cpp', 'input.txt', and 'output.txt'. The 'output.txt' tab is active and highlighted in blue. The code area shows a list of permutations of the letters A, B, and C, each on a new line, numbered 1 through 7.

```
1 ABC
2 ACB
3 BAC
4 BCA
5 CAB
6 CBA
7
```


DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

WEEK 13

1. Given an array of characters, you have to find distinct characters from this array. Design an algorithm and implement it using a program to solve this problem using hashing. (Time Complexity = $O(n)$)

```
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;
int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    int n; fin >> n;
    vector<char> charArray(n);
    for (int i = 0; i < n; i++) {
        fin >> charArray[i];
    }
    const int MAX_CHAR = 256;
    int freq[MAX_CHAR] = {0};
    for (char ch : charArray) {
        freq[ch]++;
    }
    for (int i = 0; i < MAX_CHAR; i++) {
        if (freq[i] > 0) {
            fout << char(i) << ": " << freq[i] << endl;
        }
    }
    fin.close();
    fout.close();
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1  20
2  a e d e f j t t z a z f t a e e k a e q
```

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1  a: 4
2  d: 1
3  e: 5
4  f: 2
5  j: 1
6  k: 1
7  q: 1
8  t: 3
9  z: 2
10
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

2. Given an array of integers of size n , design an algorithm and write a program to check whether this array contains duplicate within a small window of size $k < n$

```
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;

bool containsNearbyDuplicate(vector<int>& nums, int k) {
    int n = nums.size();
    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n && j <= i + k; ++j) {
            if (nums[i] == nums[j]) {
                return true;
            }
        }
    }
    return false;
}
```

```
int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    int t; fin >> t;
    while(t--){
        int n, k;
        fin >> n;
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
vector<int> nums(n);
for (int i = 0; i < n; ++i) {
    fin >> nums[i];
}
fin>>k;
if (containsNearbyDuplicate(nums, k)) {
    fout << "Yes, there are duplicates within a window of size " << k << endl;
} else {
    fout << "No, there are no duplicates within a window of size " << k << endl;
}

}

fin.close();
fout.close();

return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1  2
2  10
3  1 2 3 4 1 2 3 4 1 2
4  3
5  12
6  1 2 3 1 2 3 1 2 3 1 2 3
7  4
8
```

```
main.cpp  input.txt  ⋮  output.txt  ⋮
1  No, there are no duplicates within a window of size 3
2  Yes, there are duplicates within a window of size 4
3
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

3. Given an array of nonnegative integers, Design an algorithm and implement it using a program to find two pairs (a,b) and (c,d) such that $a*b = c*d$, where a, b, c and d are distinct elements of array. #include <iostream>

```
#include <vector>
```

```
#include <fstream>
```

```
using namespace std;
```

```
vector<pair<pair<int, int>, pair<int, int>>> findPairs(const vector<int>& nums) {
```

```
    vector<pair<pair<int, int>, pair<int, int>>> result;
```

```
    for (int i = 0; i < nums.size(); ++i) {
```

```
        for (int j = i + 1; j < nums.size(); ++j) {
```

```
            for (int k = j + 1; k < nums.size(); ++k) {
```

```
                for (int l = k + 1; l < nums.size(); ++l) {
```

```
                    if (nums[i] * nums[j] == nums[k] * nums[l]) {
```

```
                        result.push_back({ {nums[i], nums[j]}, {nums[k], nums[l]} });
```

```
                    }
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
    return result;
```

```
}
```

```
int main() {
```

```
    ifstream fin("input.txt");
```

```
    ofstream fout("output.txt");
```

```
    // Check if files are opened successfully
```

```
    if (!fin.is_open() || !fout.is_open()) {
```

```
        cout << "Error occurred while opening files.\n";
```

```
        return 0;
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
}

int n;fin>>n;
vector<int> nums(n);
for (int i = 0; i < n; ++i) {
    fin >> nums[i];
}
vector<pair<pair<int, int>, pair<int, int>>> pairs = findPairs(nums);

if (pairs.empty()) {
    fout << "No such pairs found." << endl;
} else {
    cout << "Pairs found:" << endl;
    for (auto& pair : pairs) {
        fout << pair.first.first << pair.first.second << " ";
        fout << pair.second.first << pair.second.second << endl;
    }
}

fin.close();
fout.close();

return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

main.cpp	input.txt	:	output.txt	:
1	10			
2	31 23 4 1 39 2 20 27 8 10			

main.cpp	input.txt	:	output.txt	:
1	4 2			
2	1 8			
3				

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

WEEK 14

1. Given a number n, write an algorithm and a program to find nth ugly number. Ugly numbers are those numbers whose only prime factors are 2, 3 or 5. The sequence 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24,.. is sequence of ugly numbers

```
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;

long long nthUglyNumber(int n) {
    vector<long long> ugly(n);
    ugly[0] = 1;
    int idx2 = 0, idx3 = 0, idx5 = 0;

    for (int i = 1; i < n; ++i) {
        ugly[i] = min({ugly[idx2] * 2, ugly[idx3] * 3, ugly[idx5] * 5});
        if (ugly[i] == ugly[idx2] * 2) ++idx2;
        if (ugly[i] == ugly[idx3] * 3) ++idx3;
        if (ugly[i] == ugly[idx5] * 5) ++idx5;
    }

    return ugly[n - 1];
}

int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
}

int t;fin>>t;

while(t--){
    int n;fin>>n;

    fout<< nthUglyNumber(n) << endl;
}

fin.close();
fout.close();

return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

main.cpp	input.txt	:	output.txt	:
1	3			
2	11			
3	8			
4	15			

main.cpp	input.txt	:	output.txt	:
1	15			
2	9			
3	24			
4				

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

2. Given a directed graph, write an algorithm and a program to find mother vertex in a graph. A mother vertex is a vertex v such that there exists a path from v to all other vertices of the graph.

```
#include <iostream>
#include <vector>
#include <fstream>
#include <stack>
using namespace std;

void dfs(int v, vector<bool>& visited, const vector<vector<int>>& adjMatrix, stack<int>& stk) {
    visited[v] = true;
    for (int u = 0; u < adjMatrix[v].size(); ++u) {
        if (adjMatrix[v][u] && !visited[u]) {
            dfs(u, visited, adjMatrix, stk);
        }
    }
    stk.push(v);
}

int findMotherVertex(const vector<vector<int>>& adjMatrix) {
    int V = adjMatrix.size();
    vector<bool> visited(V, false);
    stack<int> stk;

    for (int i = 0; i < V; ++i) {
        if (!visited[i]) {
            dfs(i, visited, adjMatrix, stk);
        }
    }

    fill(visited.begin(), visited.end(), false);
    int mother = stk.top();
    dfs(mother, visited, adjMatrix, stk);
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
for (bool visit : visited) {
    if (!visit) {
        return -1;
    }
}

return mother;
}

bool isPathExists(int src, int dest, const vector<vector<int>>& adjMatrix) {
    vector<bool> visited(adjMatrix.size(), false);
    stack<int> stk;
    dfs(src, visited, adjMatrix, stk);
    return visited[dest];
}

int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");

    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }

    int V,E;fin>>V>>E;
    vector<vector<int>> adjMatrix(V, vector<int>(V, 0));
    for (int i = 0; i < E; ++i) {
        int u, v;
        fin >> u >> v;
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

```
    adjMatrix[u][v] = 1;
}
int motherVertex = findMotherVertex(adjMatrix);
if (motherVertex != -1) {
    fout << "Mother vertex found: " << motherVertex << endl;
} else {
    fout << "No mother vertex found" << endl;
}

int src, dest;
fin >> src >> dest;

if (isPathExists(src, dest, adjMatrix)) {
    fout << "There exists a path from vertex " << src << " to vertex " << dest << endl;
} else {
    fout << "There is no path from vertex " << src << " to vertex " << dest << endl;
}

fin.close();
fout.close();

return 0;
}
```

DESIGN AND ANALYSIS OF ALGORITHMS LAB (PCS-409)

main.cpp	input.txt	:	output.txt	:
1	5 14			
2	0 1			
3	0 2			
4	1 0			
5	1 2			
6	1 3			
7	1 4			
8	2 0			
9	2 1			
10	2 3			
11	3 1			
12	3 2			
13	3 4			
14	4 1			
15	4 3			
16	0 4			

main.cpp	input.txt	:	output.txt	:
1	Mother vertex found: 0			
2	There exists a path from vertex 0 to vertex 4			
3				