Walchand College of Engineering, Sangli
Computer Science & Engineering
Third Year

Course: Design and analysis of algorithm Lab

Lab course coordinator:
Mrs A M Chimanna- Batch: - T1, T2,T3,T4

Assignment No.6

Greedy method

PRN : 21510111

Batch : T1

Divide and conquer strategy

Strassen's Matrix Multiplication

1) Implement Naive method multiply two matrices and Justify complexity is o(n3)

```
#include<bits/stdc++.h>
using namespace std;

void multiply(vector<vector<int>>A, vector<vector<int>> B){
    vector<vector<int> > C = { { 0, 0, 0, 0 },
                               { 0, 0, 0, 0 },
                               { 0, 0, 0, 0 },
                               { 0, 0, 0, 0 }};

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            C[i][j] = 0;
            for (int k = 0; k < 4; k++)
            {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }

    for (int i = 0; i <= 3; i++) {
        for (int j = 0; j <= 3; j++) {
            cout << C[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

int main(){
    vector<vector<int> > matrix_A = { { 1, 8, 1, 1 },
```

```
                                        { 2, 2, 0, 2 },
                                        { 3, 3, 3, 3 },
                                        { 2, 2, 2, 2 } };

    vector<vector<int> > matrix_B = { { 1, 1, 1, 1 },
                                      { 8, 2, 2, 2 },
                                      { 3, 3, 3, 3 },
                                      { 2, 2, 2, 2 } };

    multiply(matrix_A, matrix_B);

}
```

2) Implement Strassen's matrix multiplication for 3*3 matrix.
Do analysis of algorithm with respect to time complexity.

```
#include <bits/stdc++.h>
using namespace std;

#define ROW_1 4
#define COL_1 4

#define ROW_2 4
#define COL_2 4

void print(string display, vector<vector<int> > matrix,
        int start_row, int start_column, int end_row,
        int end_column)
{
    cout << endl << display << " =>" << endl;
    for (int i = start_row; i <= end_row; i++) {
        for (int j = start_column; j <= end_column; j++) {
            cout << setw(10);
            cout << matrix[i][j];
        }
        cout << endl;
    }
    cout << endl;
    return;
}
```

```
vector<vector<int> >
add_matrix(vector<vector<int> > matrix_A,
        vector<vector<int> > matrix_B, int split_index,
        int multiplier = 1)
{
    for (auto i = 0; i < split_index; i++)
        for (auto j = 0; j < split_index; j++)
            matrix_A[i][j]
                = matrix_A[i][j]
                + (multiplier * matrix_B[i][j]);
    return matrix_A;
}

vector<vector<int> >
multiply_matrix(vector<vector<int> > matrix_A,
                vector<vector<int> > matrix_B)
{
    int col_1 = matrix_A[0].size();
    int row_1 = matrix_A.size();
    int col_2 = matrix_B[0].size();
    int row_2 = matrix_B.size();

    if (col_1 != row_2) {
        cout << "\nError: The number of columns in Matrix "
                "A must be equal to the number of rows in "
                "Matrix B\n";
        return {};
    }

    vector<int> result_matrix_row(col_2, 0);
    vector<vector<int> > result_matrix(row_1,
                                       result_matrix_row);

    if (col_1 == 1)
```

```
        result_matrix[0][0]
            = matrix_A[0][0] * matrix_B[0][0];
    else {
        int split_index = col_1 / 2;

        vector<int> row_vector(split_index, 0);

        vector<vector<int> > a00(split_index, row_vector);
        vector<vector<int> > a01(split_index, row_vector);
        vector<vector<int> > a10(split_index, row_vector);
        vector<vector<int> > a11(split_index, row_vector);
        vector<vector<int> > b00(split_index, row_vector);
        vector<vector<int> > b01(split_index, row_vector);
        vector<vector<int> > b10(split_index, row_vector);
        vector<vector<int> > b11(split_index, row_vector);

        for (auto i = 0; i < split_index; i++)
            for (auto j = 0; j < split_index; j++) {
                a00[i][j] = matrix_A[i][j];
                a01[i][j] = matrix_A[i][j + split_index];
                a10[i][j] = matrix_A[split_index + i][j];
                a11[i][j] = matrix_A[i + split_index]
                                    [j + split_index];
                b00[i][j] = matrix_B[i][j];
                b01[i][j] = matrix_B[i][j + split_index];
                b10[i][j] = matrix_B[split_index + i][j];
                b11[i][j] = matrix_B[i + split_index]
                                    [j + split_index];
            }

        vector<vector<int> > p(multiply_matrix(
            a00, add_matrix(b01, b11, split_index, -1)));
        vector<vector<int> > q(multiply_matrix(
            add_matrix(a00, a01, split_index), b11));
        vector<vector<int> > r(multiply_matrix(
```

```
        add_matrix(a10, a11, split_index), b00));
    vector<vector<int> > s(multiply_matrix(
        a11, add_matrix(b10, b00, split_index, -1)));
    vector<vector<int> > t(multiply_matrix(
        add_matrix(a00, a11, split_index),
        add_matrix(b00, b11, split_index)));
    vector<vector<int> > u(multiply_matrix(
        add_matrix(a01, a11, split_index, -1),
        add_matrix(b10, b11, split_index)));
    vector<vector<int> > v(multiply_matrix(
        add_matrix(a00, a10, split_index, -1),
        add_matrix(b00, b01, split_index)));

    vector<vector<int> > result_matrix_00(add_matrix(
        add_matrix(add_matrix(t, s, split_index), u,
                split_index),
        q, split_index, -1));
    vector<vector<int> > result_matrix_01(
        add_matrix(p, q, split_index));
    vector<vector<int> > result_matrix_10(
        add_matrix(r, s, split_index));
    vector<vector<int> > result_matrix_11(add_matrix(
        add_matrix(add_matrix(t, p, split_index), r,
                split_index, -1),
        v, split_index, -1));

    for (auto i = 0; i < split_index; i++)
        for (auto j = 0; j < split_index; j++) {
            result_matrix[i][j]
                = result_matrix_00[i][j];
            result_matrix[i][j + split_index]
                = result_matrix_01[i][j];
            result_matrix[split_index + i][j]
                = result_matrix_10[i][j];
            result_matrix[i + split_index]
```

```
                            [j + split_index]
                    = result_matrix_11[i][j];
            }

        a00.clear();
        a01.clear();
        a10.clear();
        a11.clear();
        b00.clear();
        b01.clear();
        b10.clear();
        b11.clear();
        p.clear();
        q.clear();
        r.clear();
        s.clear();
        t.clear();
        u.clear();
        v.clear();
        result_matrix_00.clear();
        result_matrix_01.clear();
        result_matrix_10.clear();
        result_matrix_11.clear();
    }
    return result_matrix;
}

int main()
{
    vector<vector<int> > matrix_A = { { 1, 1, 1, 1 },
                                     { 2, 2, 2, 2 },
                                     { 3, 3, 3, 3 },
                                     { 2, 2, 2, 2 } };

    print("Array A", matrix_A, 0, 0, ROW_1 - 1, COL_1 - 1);
```

```
    vector<vector<int> > matrix_B = { { 1, 1, 1, 1 },
                                      { 2, 2, 2, 2 },
                                      { 3, 3, 3, 3 },
                                      { 2, 2, 2, 2 } };

    print("Array B", matrix_B, 0, 0, ROW_2 - 1, COL_2 - 1);

    vector<vector<int> > result_matrix(
        multiply_matrix(matrix_A, matrix_B));

    print("Result Array", result_matrix, 0, 0, ROW_1 - 1,
        COL_2 - 1);
}
```

To apply Greedy method to solve problems of

1) Job sequencing with deadlines

1.A) Generate table of feasible,processing sequencing , profit .

1.B) What is the solution generated by the function JS when n=7, (*p1,p2,...,p7*) = (3,5,20,18,1,6,30), and (*d1,d2,d3,...,d7*) = (1,3,4,3,2,1,2)?

Solution:

Solution generated by JS function :p6,p7,p4,p3

```
#include <bits/stdc++.h>
using namespace std;
class Job {
    public:
    string id;
    int dead;
    int profit;
};
struct jobProfit {
    bool operator()(Job const& a, Job const& b)
    {
        return (a.profit < b.profit);
    }
};
void printJobScheduling(Job arr[], int n)
{
    vector<Job> result;
    sort(arr, arr + n,
        [](Job a, Job b) { return a.dead < b.dead; });
    priority_queue<Job, vector<Job>, jobProfit> pq;

    for (int i = n - 1; i >= 0; i--) {
```

```
        int slot_available;
        if (i == 0) {
            slot_available = arr[i].dead;
        }
        else {
            slot_available = arr[i].dead - arr[i - 1].dead;
        }
        pq.push(arr[i]);
        while (slot_available > 0 && pq.size() > 0) {
            Job job = pq.top();
            pq.pop();
            slot_available--;
            result.push_back(job);
        }
    }
    sort(result.begin(), result.end(),
        [&](Job a, Job b) { return a.dead < b.dead; });
    for (int i = 0; i < result.size(); i++)
        cout << result[i].id << ' ';
    cout << endl;
}
int main()
{
    vector<int> profit,deadline;
    vector<string> ids;
    int n;
    cout<<"Enter Number of Jobs:";
    cin>>n;
    profit.resize(n);
    deadline.resize(n);
    ids.resize(n);
    cout<<"Enter array Job IDs:"<<endl;
    for(int i=0; i<n; i++)cin>>ids[i];
    cout<<"Enter Array of Profit:"<<endl;
    for(int i = 0; i < n; i++)cin>>profit[i];
    cout<<"Enter Array of Deadline:"<<endl;
    for(int i = 0; i < n; i++)cin>>deadline[i];

    Job arr[n];
    for(int i=0;i<n;i++)
    {
        arr[i]={ids[i],deadline[i],profit[i]};
```

```
    }
    cout << "Following is maximum profit sequence of jobs:"<<endl;
    printJobScheduling(arr, n);
    return 0;
}
```

```
Enter Number of Jobs:7
Enter array Job IDs:
p1 p2 p3 p4 p5 p6 p7
Enter Array of Profit:
3 5 20 18 1 6 30
Enter Array of Deadline:
1 3 4 3 2 1 2
Following is maximum profit sequence of jobs:
p6 p7 p4 p3
```

1.C) **Input**: Five Jobs with following deadlines and profits

JobID	Deadline	Profit
a	2	100
b	1	19
c	2	27
d	1	25
e	3	15

Output: Following is maximum profit sequence of jobs:
a,c,e

```
Enter Number of Jobs:5
Enter array Job IDs:
a b c d e
Enter Array of Profit:
100 19 27 25 15
Enter Array of Deadline:
2 1 2 1 3
Following is maximum profit sequence of jobs:
a c e
```

1.D) Study and implement Disjoint set algorithm to reduce time complexity of JS from O(n^2) to nearly O(n)

```
#include<bits/stdc++.h>
using namespace std;
class Job {
    public:
    string id;
    int dead;
    int profit;
};
class DisjointSet
{
    public:
    int *parent;
    DisjointSet(int n)
    {
        parent = new int[n+1];
        for (int i = 0; i <= n; i++)
            parent[i] = i;
    }
    int find(int s)
    {
        if (s == parent[s])
            return s;
        return parent[s] = find(parent[s]);
    }
    void merge(int u, int v)
    {
```

```
        parent[v] = u;
    }
};
bool cmp(Job a, Job b){
    return (a.profit > b.profit);
}
int findMaxDeadline(struct Job arr[], int n){
    int ans = INT_MIN;
    for (int i = 0; i < n; i++)
        ans = max(ans, arr[i].deadLine);
    return ans;
}
void printJobScheduling(Job arr[], int n)
{
    sort(arr, arr + n, cmp);
    int maxDeadline = findMaxDeadline(arr, n);
    DisjointSet ds(maxDeadline);
    for (int i = 0; i < n; i++)
    {
        int availableSlot = ds.find(arr[i].deadLine);

        if (availableSlot > 0)
        {
            ds.merge(ds.find(availableSlot - 1),
                            availableSlot);

            cout << arr[i].id << " ";
        }
    }
}
int main()
{
    vector<int> profit,deadline;
    vector<string> ids;
    int n;
    cout<<"Enter Number of Jobs:";
    cin>>n;
    profit.resize(n);
    deadline.resize(n);
    ids.resize(n);
    cout<<"Enter array Job IDs:"<<endl;
    for(int i=0; i<n; i++)cin>>ids[i];
```

```
    cout<<"Enter Array of Profit:"<<endl;
    for(int i = 0; i < n; i++)cin>>profit[i];
    cout<<"Enter Array of Deadline:"<<endl;
    for(int i = 0; i < n; i++)cin>>deadline[i];
    Job arr[n];
    for(int i=0;i<n;i++){
        arr[i]={ids[i],deadline[i],profit[i]};
    }
    cout << "Following jobs need to be  executed for maximum profit:"<<endl;
    printJobScheduling(arr, n);
    return 0;
}
```

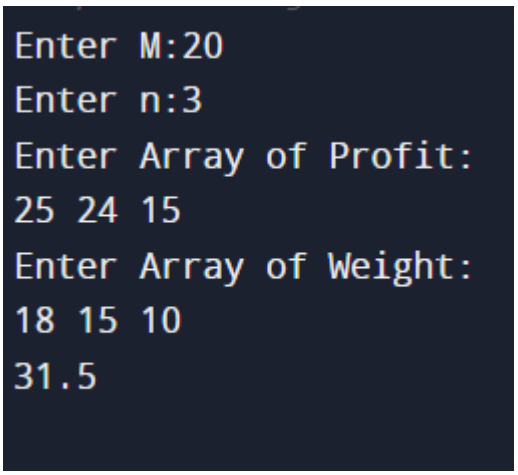
2) To implement Fractional Knapsack problem 3 objects (n=3).

(w1,w2,w3) = (18,15,10)
(p1,p2,p3) = (25,24,15)
M=20
With strategy
a) Largest-profit strategy
b) Smallest-weight strategy
c) Largest profit-weight ratio strategy

```
#include <bits/stdc++.h>

using namespace std;

class Item {

    public:

    int profit, weight;

    Item(int profit, int weight)

    {

        this->profit = profit;

        this->weight = weight;

    }

    Item()

    {}

};

static bool cmp(struct Item a, struct Item b)
```

```
{
    double r1 = (double)a.profit / (double)a.weight;
    double r2 = (double)b.profit / (double)b.weight;
    return r1 > r2;
}
double fractionalKnapsack(int W, struct Item arr[], int N)
{
    sort(arr, arr + N, cmp);
    double finalvalue = 0.0;
    for (int i = 0; i < N; i++) {
        if (arr[i].weight <= W) {
            W -= arr[i].weight;
            finalvalue += arr[i].profit;
        }
        else {
            finalvalue
                += arr[i].profit
                * ((double)W / (double)arr[i].weight);
            break;
        }
    }
    return finalvalue;
}
int main()
{
    int M,n;
```

```
    cout<<"Enter M:";
    cin>>M;
    cout<<"Enter n:";
    cin>>n;
    vector<int> weights,profit;
    weights.resize(n);
    profit.resize(n);
    cout<<"Enter Array of Profit:"<<endl;
    for(int i = 0; i < n; i++)cin>>profit[i];
    cout<<"Enter Array of Weight:"<<endl;
    for(int i = 0; i < n; i++)cin>>weights[i];
    Item arr[n];
    for(int i = 0; i < n;i++)
    arr[i]={profit[i], weights[i]};
    cout << fractionalKnapsack(M, arr, n);
    return 0;
}
```


Enter M:20
Enter n:3
Enter Array of Profit:
25 24 15
Enter Array of Weight:
18 15 10
31.5