DAOA Practical

1)Coin Change DP

```
#include<stdio.h>
void main()
{
    int den[]={1,4,6};
    int N=8;
    int n=sizeof(den)/sizeof(den[0]);
    int i,j,sn;
    int k=0;
    int C[n+1][N+1];
    for(i=0;i<=n;i++)
    {
        for(j=0;j<=N;j++)
        {
            if(i==0 || j==0)
            {
                C[i][j]=0;
            }
            else if(i==1)
            {
                C[i][j]=1+C[i][j-den[i-1]];
            }
            else if(j<den[i-1])
            {
                C[i][j]=C[i-1][j];
            }
            else
            {
                if(C[i-1][j]<=1+C[i][j-den[i-1]])
                {
                    C[i][j]=C[i-1][j];
                }
                else
                {
                    C[i][j]=1+C[i][j-den[i-1]];
                }
            }
```

```
        }
    }
    for(i=0;i<=n;i++)
    {
        for(j=0;j<=N;j++)
        {
            printf("%d ",C[i][j]);
        }
        printf("\n");
    }
    sn=C[n][N];
    printf("\nNo of coins: %d\n",sn);
    int sol[sn];
    for(i=n;i>=0;i--)
    {
        for(j=N;j>=0;j--)
        {
            if(C[i][j]==C[i-1][j])
            {
                i--;
            }
            else
            {
                j=j-den[i-1];
            }
            sol[k++]=den[i-1];
        }
    }
    for(i=0;i<sn;i++)
    {
        printf("%d ",sol[i]);
    }
}
```

2)Coin Change Greedy

```
#include<stdio.h>
void main()
{
    int arr[]={ 1, 2, 5, 10, 20, 50, 100, 200, 2000};
```

```
    int value=93;
    int n=sizeof(arr)/sizeof(arr[0]);
    int sol [n];
    int i;
    //Initialize sol as 0 array
    for (int i = 0; i < n; i++)
    {
        sol[i] = 0;
    }
    //Start at highest denomination and traverse till lowest denomination
    for(i=n-1;i>=0;i--)
    {
        while(arr[i]<=value) //Check how many coins of this denomination
can be there
        {
            sol[i]+=1;
            value=value-arr[i]; //Subtract Value with the value of the
coin added
        }
    }
    //Print Solution
    for(i=0;i<n;i++)
    {
        printf("%d ",sol[i]);
    }
}
```

3)Dijkstra Greedy

```
#include<stdio.h>
#include<stdbool.h>
#define V 9
#define inf 9999

int minDist(int dist[],bool visited[])
{
    int min_idx;
    int min=inf;
    for(int v=0;v<V;v++)
    {
```

```
        if(visited[v]==false && dist[v]<min) //Check if not visited and if
distance is lesser than min
        {
            min=dist[v];
            min_idx=v;
        }
    }
    return min_idx;
}

void dijkstra(int graph[V][V],int src)
{
    bool visited[V];
    int dist[V];
    int  i;
    //Initialize Visited array with false and distance array with inf
    for(i=0;i<V;i++)
    {
        dist[i]=inf;
        visited[i]=false;
    }
    //Make visited of src index to true and distance of src index to 0
    dist[src]=0;
    visited[src]=true;

    for(i=0;i<V;i++)
    {
        int u=minDist(dist,visited);
        visited[u]=true;
        for(int v=0;v<V;v++)
        {
            if(!visited[v] && //If node is not visited
                graph[u][v] && //If there is path between u -> v
                dist[u]!=inf && //If distance is not inf
                graph[u][v] + dist[u]<dist[v]) //If distance from u -> v
is less than directly to v
            {
                dist[v]=graph[u][v] + dist[u];
            }
        }
```

```
    }
    //Display Solution
    for (i = 0; i < V; i++)
    {
        printf("%d ",dist[i]);
    }
}
void main()
{
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
    dijkstra(graph, 0);
}
```

4)Job Schedule Greedy

```
#include<stdio.h>
void main()
{
    int i,j,temp;
    int job[][3]={ { 'a', 2, 100 },
                   { 'b', 1, 19 },
                   { 'c', 2, 27 },
                   { 'd', 1, 25 },
                   { 'e', 3, 15 } };
    int n=sizeof(job)/sizeof(job[0]);
    int profit=0;
    for(i=0;i<n;i++)
    {
        for(j=i;j<n;j++)
        {
```

```
            if(job[i][2]<job[j][2])
            {
                temp=job[i][2];
                job[i][2]=job[j][2];
                job[j][2]=temp;

                temp=job[i][1];
                job[i][1]=job[j][1];
                job[j][1]=temp;

                temp=job[i][0];
                job[i][0]=job[j][0];
                job[j][0]=temp;

            }
        }
    }
    int a[n];
    for(i=0;i<n;i++)
    {
        a[i]=0;
    }
    for(i=0;i<n;i++)
    {
        for(j=n-1;j>=0;j--)
        {
            if(a[j]==0 && job[i][1]>=j+1)
            {
                a[j]=job[i][0];
                profit=profit+job[i][2];
                break;
            }
        }
    }
    printf("Profit: %d\n",profit);
    for(i=0;i<n;i++)
    {
        printf("%c ",a[i]);
    }
}
```

5)Knapsack Greedy

```
#include<stdio.h>
//Swapping arrays
void swap(double a[],double b[])
{
    double temp[4];
    memcpy(temp,a,sizeof(double)*3);
    memcpy(a,b,sizeof(double)*3);
    memcpy(b,temp,sizeof(double)*3);
}
void main()
{
    //Array is initialized as {Value,Weight,0,Index} 0 for ratio
    double arr[][4]={{100, 20,0.0,0},{60,10,0.0,1},{120, 30,0.0,2}};
    int weight=50;
    int n=sizeof(arr)/sizeof(arr[0]);
    float sol[n];
    char solc[n];
    int i,j,temp;
    float profit=0;
    float frac=0.0;
    //Initialize sol and solc array
    for (int i = 0; i < n; i++)
    {
        solc[i] = 'x';
        sol[i] = 1.0;
    }
    //Adding ratios to arr
    for(i=0;i<n;i++)
    {
        arr[i][2]=arr[i][0]/arr[i][1];
    }
    //Sorting wrt ratio in descending order
    for(i=0;i<n;i++)
    {
        for(j=i;j<n;j++)
        {
            if(arr[i][2]<=arr[j][2])
            {
```

```
                swap(arr[i],arr[j]);
            }
        }
    }
    for(i=0;i<n;i++)
    {
        if(arr[i][1]<=weight) //Checking if Weight less than bag
        {
            sol[i]=1.0; //Adding ratio 1
            solc[i]=(char)((int)arr[i][3]+65); //Adding Character of the
item added
            profit=profit+arr[i][0]; //Adding Value to Profit
            weight=weight-arr[i][1]; //Subtracting Weight of item from
weight of bag
        }
        else
        {
            frac=weight/arr[i][1]; //Calculating Fraction
            sol[i]=frac; //Adding fraction as ratio
            solc[i]=(char)((int)arr[i][3]+65); //Adding Character of the
item added
            profit=profit+arr[i][0]*frac; //Adding Value to Profit
            break; //Stoping as weight of bag will become 0
        }
    }
    //Printing Solution
    for(i=0;i<n;i++)
    {
        printf("[%c %.2f] ",solc[i],sol[i]);
    }
    printf("\nProfit: %.2f",profit);
}
```

6) Matrix Multiplication DP

```
#include<stdio.h>
#define inf 9999
void main()
{
```

```
    int seq[]={2,3,4,2};
    int i,j,k,min;
    int n=sizeof(seq)/sizeof(seq[0]);
    int C[n][n];
    int P[n][n];
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            P[i][j]=0;
            C[i][j]=0;
        }
    }
    for(i=1;i<n;i++)
    {
        for(j=0;j<n && j<n-i;j++)
        {
            min=inf;
            for(k=j;k<j+i;k++)
            {
                if(min>C[j][k]+C[k+1][j+i]+seq[j-1]*seq[k]*seq[j+i])
                {
                    min=C[j][k]+C[k+1][j+i]+seq[j-1]*seq[k]*seq[j+i];
                    P[j][j+i]=k;
                }
            }
            C[j][j+i]=min;
        }
    }
    printf("\nCost Matrix: \n");
    for(i=1;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            if(j>=i)
            {
                printf("%d\t",C[i][j]);
            }
            else
            {
```

```
                printf("\t");
            }
        }
        printf("\n");
    }
    printf("\nOptimal Parenthesis Matrix: \n");
    for(i=1;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            if(j>=i)
            {
                printf("%d\t",P[i][j]);
            }
            else
            {
                printf("\t");
            }
        }
        printf("\n");
    }

}
```

7) LCS

```
#include<stdio.h>
#include<string.h>

void main()
{
    char X[] = "STONE";
    char Y[] = "LONGEST";
    int x=strlen(X);
    int y=strlen(Y);
    int i,j,s;
    int L[x+1][y+1];
    for(i=0;i<=x;i++)
    {
        for(j=0;j<=y;j++)
```

```
        {
            if(i==0 || j==0)
            {
                L[i][j]=0;
            }
            else if(X[i-1]==Y[j-1])
            {
                L[i][j]=1+L[i-1][j-1];
            }
            else
            {
                L[i][j]=(L[i-1][j] > L[i][j])?L[i-1][j]:L[i][j-1];
            }
        }
    }
    for(i=0;i<=x;i++)
    {
        for(j=0;j<=y;j++)
        {
            printf("%d ",L[i][j]);
        }
        printf("\n");
    }
    s=L[x][y];
    printf("\nLongest Substring: %d",s);
}
```

8) KMP

```
#include<stdio.h>
#include<string.h>
void main()
{
    char txt[]="hello";
    char pattern[]="ll";
    int i=0;
    int j=0;
    int ptr=-1;
    while(i<strlen(txt) && j<strlen(pattern))
    {
```

```
        if(txt[i]==pattern[j])
        {
            if(j==0)
            {
                ptr=i;
            }
            j++;
            i++;
        }
        else
        {
            if(ptr!=-1)
            {
                i=ptr+1;
            }
            else
            {
                i++;
            }
            j=0;
            ptr=-1;
        }
    }
    if(ptr==strlen(pattern))
    {
        printf("%d",ptr);
    }
    else
    {
        printf("Not Found");
    }
}
```

9) TSP DP

```
#include<stdio.h>
void main()
{
    int n=4;
    int dist[4][4]={{0, 22, 26, 30},
```

```
                    {30, 0, 45, 35},
                    {25, 45, 0, 60},
                    {30, 35, 40, 0}};
    int c=tsp(n,dist,1,0);
    printf("Cost: %d",c);
}
int tsp(int n,int dist[n][n],int visited,int current)
{
    if(visited==(1<<n)-1)
    {
        return dist[current][0];
    }
    int min=9999;
    for(int i=0;i<n;i++)
    {
        if(!(visited & (1<<i)))
        {
            int newc=dist[current][i]+tsp(n,dist,visited | (1<<i),i);
            if(newc<min)
            {
                min=newc;
            }
        }
    }
    return min;
}
```

10) N-Queen

```
#include<stdio.h>
#include<math.h>

int a[10];
int count=0;
int place(int pos)
{
    for(int i=1;i<pos;i++)
    {
        if(a[i]==a[pos] || abs(a[i]-a[pos])==abs(i-pos))
        {
```

```
            return 0;
        }
    }
    return 1;
}
void printsol(int n)
{
    count++;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            if(a[i]==j)
            {
                printf("Q\t");
            }
            else
            {
                printf("*\t");
            }
        }
        printf("\n");
    }
    printf("\n");
}
void queen(int n,int k)
{
    if(k>n)
    {
        printsol(n);
    }
    else
    {
        for(a[k]=1;a[k]<=n;a[k]++)
        {
            if(place(k))
            {
                queen(n,k+1);
            }
        }
```

```
    }
}
void main()
{
    int n=8;
    queen(n,1);
    printf("Count: %d",count);
}
```

11) Sum of Subset

```
#include<stdio.h>
#include<stdbool.h>

int set[]={3,4,5,6};
int target=9;
int n=sizeof(set)/sizeof(set[0]);
bool chosen[10];

void main()
{
    subset(0,0);
}

void printsol(int n)
{
    for(int i=0;i<n;i++)
    {
        if(chosen[i])
        {
            printf("%d ",set[i]);
        }
    }
    printf("\n");
}

void subset(int index,int sum)
{
    if(index==n)
    {
        if(sum==target)
```

```
        {
            printsol(n);
        }
        return;
    }
    chosen[index]=true;
    subset(index+1,sum+set[index]);
    chosen[index]=false;
    subset(index+1,sum);
}
```

12) Prims

```
#include<stdio.h>
#include<stdbool.h>
#define V 5
#define inf 9999

void main()
{
    int graph[V][V] = {
    {0, 9, 75, 0, 0},
    {9, 0, 95, 19, 42},
    {75, 95, 0, 51, 66},
    {0, 19, 51, 0, 31},
    {0, 42, 66, 31, 0}};
    int src=0;
    int n=0;
    int x,y,i,j,min;
    bool visited[V];
    //Initialize visited with false
    for(i=0;i<n;i++)
    {
        visited[i]=false;
    }
    //Visited of source becomes true
    visited[src]=true;
    while(n<V-1)
    {
        x=0;
        y=0;
```

```
        min=inf;
        for(i=0;i<V;i++)
        {
            if(visited[i]) //If already visited
            {
                for(j=0;j<V;j++)
                {
                    if(!visited[j] && graph[i][j] && graph[i][j]<min) //If
next node not visited and path exits and path less than min
                    {
                        min=graph[i][j];
                        x=i;
                        y=j;
                    }
                }
            }
        }
        printf("%d -> %d: %d\n",x,y,graph[x][y]);
        visited[y]=true;
        n++;
    }
}
```

13) Kruskals

```
#include<stdio.h>
#define V 5
#define INF 9999
int parent[V];
int find(int i)
{
    while(parent[i]!=i)
    {
        i=parent[i];
    }
    return i;
}
void unionset(int i,int j)
{
    int a=find(i);
    int b=find(j);
```

```
    parent[a]=b;
}
void main()
{
    int graph[V][V] = {{INF, 2, INF, 6, INF},
                       {2, INF, 3, 8, 5},
                       {INF, 3, INF, INF, 7},
                       {6, 8, INF, INF, 9},
                       {INF, 5, 7, 9, INF}};
    int mincost = 0;
    int n=0;
    int i,j,x,y,min;
    for (int i = 0; i < V; i++)
    {
        parent[i] = i;
    }
    while(n<V-1)
    {
        x=0;
        y=0;
        min=INF;
        for(i=0;i<V;i++)
        {
            for(j=0;j<V;j++)
            {
                if(find(i)!=find(j) && graph[i][j]<min)
                {
                    min=graph[i][j];
                    x=i;
                    y=j;
                }
            }
        }
        unionset(x,y);
        printf("%d -> %d: %d\n",x,y,min);
        n++;
        mincost+=min;
    }
    printf("Cost: %d",mincost);
}
```

14) Strassens

```
#include<stdio.h>
void main()
{
    int n = 4;
    int C[n][n];
    int A[4][4]={{1,2,3,4},
                 {1,2,3,4},
                 {1,2,3,4},
                 {1,2,3,4}};
    int B[4][4]={{1,0,0,0},
                 {0,1,0,0},
                 {0,0,1,0},
                 {0,0,0,1}};
    strassen(n, A, B, C);
    printf("\nResult of matrix multiplication:\n");
    printMatrix(n, C);
}
void add(int n,int A[n][n],int B[n][n],int C[n][n])
{
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            C[i][j]=A[i][j]+B[i][j];
        }
    }
}
void subtract(int n,int A[n][n],int B[n][n],int C[n][n])
{
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            C[i][j]=A[i][j]-B[i][j];
        }
    }
}
void printMatrix(int n, int mat[][n])
```

```
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("%d ", mat[i][j]);
        }
        printf("\n");
    }
}
void strassen(int n,int A[n][n],int B[n][n],int C[n][n])
{
    if(n==1)
    {
        C[0][0]=A[0][0]*B[0][0];
        return;
    }
    n=n/2;
    printf("%d",n);
    int
A11[n][n],A12[n][n],A21[n][n],A22[n][n],B11[n][n],B12[n][n],B21[n][n],B22[
n][n];
    int C11[n][n],C12[n][n],C21[n][n],C22[n][n];
    int P1[n][n],P2[n][n],P3[n][n],P4[n][n],P5[n][n],P6[n][n],P7[n][n];
    int
S1[n][n],S2[n][n],S3[n][n],S4[n][n],S5[n][n],S6[n][n],S7[n][n],S8[n][n],S9
[n][n],S10[n][n];
    int temp1[n][n],temp2[n][n],temp3[n][n],temp4[n][n];
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            A11[i][j]=A[i][j];
            A12[i][j]=A[i][j+n];
            A21[i][j]=A[i+n][j];
            A22[i][j]=A[i+n][j+n];
            B11[i][j]=B[i][j];
            B12[i][j]=B[i][j+n];
            B21[i][j]=B[i+n][j];
            B22[i][j]=B[i+n][j+n];
```

```
        }
    }
    add(n, A11, A22, S1);           //S1=A11+A22
    add(n, B11, B22, S2);           //S2=B11+B22
    add(n, A21, A22, S3);           //S3=A21+A22
    subtract(n, B12, B22, S4);      //S4=B12-B22
    subtract(n, B21, B11, S5);      //S5=B21-B11
    add(n, A11, A12, S6);           //S6=A11+A22
    subtract(n, A21, A11, S7);      //S7=B11-B12
    add(n, B11, B12, S8);           //S8=B11+B12
    subtract(n, A12, A22, S9);      //S9=A12-A22
    add(n, B21, B22, S10);          //S10=B21+B22

    strassen(n, S1, S2, P1);        //P1=S1.S2
    strassen(n, S3, B11, P2);       //P2=S3.B11
    strassen(n, A11, S4, P3);       //P3=A11.S4
    strassen(n, A22, S5, P4);       //P4=A22.S5
    strassen(n, S6, B22, P5);       //P5=S6.B22
    strassen(n, S7, S8, P6);        //P6=S7.S8
    strassen(n, S9, S10, P7);       //P7=S9.S10

    add(n,P1,P7,temp1);
    subtract(n,P4,P5,temp2);
    add(n,temp1,temp2,C11);         //C11=P1+P4-P5+P7
    add(n,P3,P5,C12);               //C12=P3+P5
    add(n,P2,P4,C21);               //C21=P2+P4
    add(n,P1,P6,temp3);             //C22=P1+P3-P2+P6
    subtract(n,P3,P2,temp4);
    add(n,temp3,temp4,C22);

    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            C[i][j]=C11[i][j];
            C[i][j+n]=C12[i][j];
            C[i+n][j]=C21[i][j];
            C[i+n][j+n]=C22[i][j];
        }
    }
```

```
}
```

15) 15 Puzzle

```
#include <stdio.h>
#include <stdlib.h>

#define N 4

int getInvCount(int *arr)
{
    int inv_count = 0;
    for (int i = 0; i < N * N - 1; i++)
    {
        for (int j = i + 1; j < N * N; j++)
        {
            if (arr[j] && arr[i] && arr[i] > arr[j])
                inv_count++;
        }
    }
    return inv_count;
}

int findXPosition(int puzzle[N][N])
{
    for (int i = N - 1; i >= 0; i--)
        for (int j = N - 1; j >= 0; j--)
            if (puzzle[i][j] == 0)
                return N - i;
}

int isSolvable(int puzzle[N][N])
{
    int invCount = getInvCount((int *)puzzle);
    if (N & 1)
        return !(invCount & 1);
    else
    {
        int pos = findXPosition(puzzle);
```

```
        if (pos & 1)
            return !(invCount & 1);
        else
            return invCount & 1;
    }
}

int getManhattanDistance(int value, int row, int col)
{
    if (value == 0)
        return 0;
    int goalRow = (value - 1) / N;
    int goalCol = (value - 1) % N;
    return abs(row - goalRow) + abs(col - goalCol);
}

int calculateTotalCost(int puzzle[N][N])
{
    int totalCost = 0;
    totalCost += getInvCount((int *)puzzle);
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            totalCost += getManhattanDistance(puzzle[i][j], i, j);
    return totalCost;
}

int main()
{
    int puzzle[N][N] = {
        {1, 2, 3, 4},
        {5, 6, 0, 8},
        {9, 10, 7, 11},
        {13, 14, 15, 12},
    };

    if (!isSolvable(puzzle))
    {
        printf("Not Solvable\n");
        return 0;
    }
```

```
    int totalCost = calculateTotalCost(puzzle);
    printf("Total Cost: %d\n", totalCost);
    return 0;
}
```