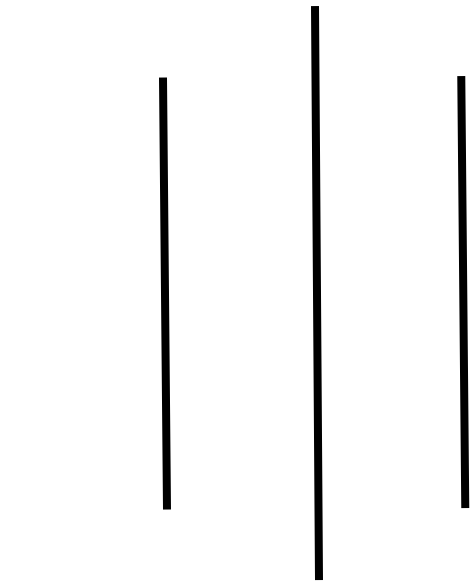


LAB REPORT ON

DESIGN AND ANALYSIS OF ALGORITHMS

SUBMITTED BY : SAMYAK MANANDHAR SUBMITTED TO : PRADYUMNA BHATTARAI

FACULTY : BSC.CSIT 5TH SEMESTER

INDEX

S.N.	TITLE	DATE	SIGNATURE
1.	Basic Algorithms & their complexities. 1.1 Algorithm for GCD 1.2 Algorithm for Fibonacci Series.	31ST JAN, 2025	
2.	Iterative Algorithms 2.1 Sequential Search and its analysis 2.2 Sorting Algorithms 2.2.1 Bubble Sort 2.2.2 Selection Sort 2.2.3 Insertion Sort	7TH FEB, 2025	
3.	Divide and Conquer Algorithm 3.1 Quick Sort 3.2 Heap Sort 3.3 Merge Sort 3.4 Order Statistics	14TH FEB, 2025	
4.	Greedy algorithms 4.1 Job sequencing algorithm 4.2 Fractional Knapsack 4.3 Huffmann Coding 4.4 Prim's and Kruskal's algorithm 4.5 Dijkstra's Algorithm	21ST FEB, 2025	
5.	Dynamic Programming 5.1 0/1 Knapsack Problem 5.2 Floyd Warshall Problem 5.3 Travelling Salesman 5.4 Matrix Chain Multiplication (Parenthesis Order)	7TH MAR, 2025	
6.	Backtracking Algorithm 6.1 Sum of subset Problem 6.2 N-Queen Problem	21ST MAR, 2025	

LAB-3.4 Order Statistics

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void selectionSort(int arr[], int n) {
  int i, j, min_idx, temp;
  for (i = 0; i < n - 1; i++) {
    min_idx = i;
    for (j = i + 1; j < n; j++) {
      if (arr[j] < arr[min_idx]) {
        min_idx = j;
      }
    }
    if (min_idx != i) {
      temp = arr[i];
      arr[i] = arr[min_idx];
      arr[min_idx] = temp;
    }
  }
}
int generateRandom(int lower, int upper) {
  return (rand() % (upper - lower + 1)) + lower;
}
int main() {
  int arr[20], k;
  char choice;
  srand(time(0));
  do {
    printf("Original Array: ");
    for (int i = 0; i < 20; i++) {
      arr[i] = generateRandom(1, 100);
      printf("%d ", arr[i]);
    }
    printf("\n");
    printf("Enter the value of k (1 to 20): ");
    scanf("%d", &k);
    if (k < 1 || k > 20) {
      printf("Invalid k! Please enter a value between 1 and 20.\n");
    } else {
```

```
        selectionSort(arr, 20);
        printf("Sorted Array: ");
        for (int i = 0; i < 20; i++) {
            printf("%d ", arr[i]);
        }
        printf("\nThe %d-th smallest element is: %d\n", k, arr[k - 1]);
    }
    printf("Do you want to find order statistic again? (y/n): ");
    scanf(" %c", &choice);
  } while (choice == 'y' || choice == 'Y');

  printf("Exiting the program!\n");
  return 0;
}
```

Output:

```
C:\Users\PC\Documents\prime college\Fifth Semester\samyak manandhar\DAA\10.exe
Original Array: 42 76 84 1 40 12 16 9 40 50 73 46 57 21 18 35 77 78 98 20
Enter the value of k (1 to 20): 7
Sorted Array: 1 9 12 16 18 20 21 35 40 40 42 46 50 57 73 76 77 78 84 98
The 7-th smallest element is: 21
Do you want to find order statistic again? (y/n): n
Exiting the program!

--------------------------------
Process exited after 13.22 seconds with return value 0
Press any key to continue . . .
```

Greedy algorithms

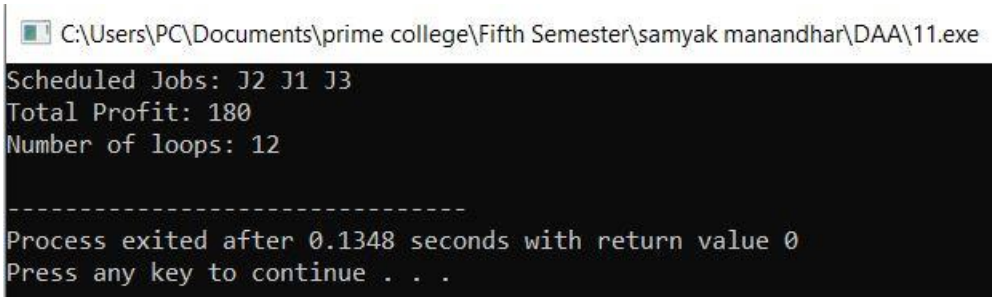
LAB-4.1 Job sequencing algorithm

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct {
  char id[5];
  int deadline;
  int profit;
} Job;
int min(int a, int b) {
  return (a < b) ? a : b;
}
int compare(const void* a, const void* b) {
  return ((Job*)b)->profit - ((Job*)a)->profit;
}
int main() {
  Job jobs[] = { {"J1", 2, 60}, {"J2", 1, 100}, {"J3", 3, 20}, {"J4", 2, 40}, {"J5", 1, 20} };
  int n = sizeof(jobs) / sizeof(jobs[0]);
  int result[n];
  int slot[n];
  memset(slot, 0, sizeof(slot));
  int totalProfit = 0;
  int loopCount = 0;
  qsort(jobs, n, sizeof(Job), compare);
  loopCount++;
  for (int i = 0; i < n; i++) {
    loopCount++;
    for (int j = min(n, jobs[i].deadline) - 1; j >= 0; j--) {
      loopCount++;
      if (!slot[j]) {
        result[j] = i;
        slot[j] = 1;
        totalProfit += jobs[i].profit;
        break;
      }
    }
  }
  printf("Scheduled Jobs: ");
```

```
    for (int i = 0; i < n; i++) {
        if (slot[i])
            printf("%s ", jobs[result[i]].id);
    }
    printf("\nTotal Profit: %d\n", totalProfit);
    printf("Number of loops: %d\n", loopCount);
    return 0;
}
```

Output:

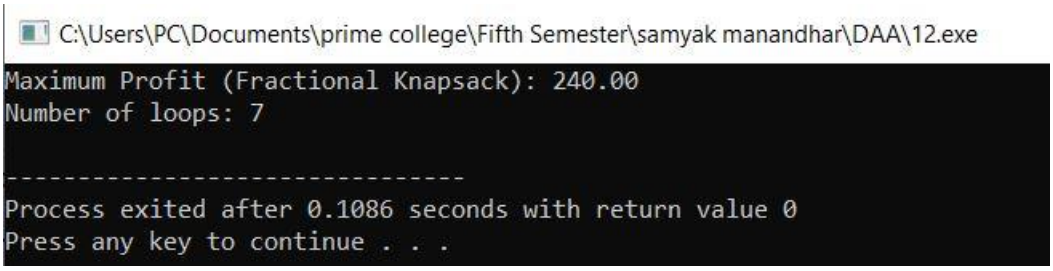


LAB-4.2 Fractional Knapsack

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
   int weight, profit;
   float ratio;
} Item;
int compare(const void* a, const void* b) {
   return ((Item*)b)->ratio > ((Item*)a)->ratio ? 1 : -1;}
int main() {
   int capacity = 50;
   Item items[] = { {10, 60}, {20, 100}, {30, 120} };
   int n = sizeof(items) / sizeof(items[0]);
   float totalProfit = 0.0;
   int loopCount = 0;
   for (int i = 0; i < n; i++) {
      loopCount++;
      items[i].ratio = (float)items[i].profit / items[i].weight;    }
   qsort(items, n, sizeof(Item), compare);
   loopCount++;
   for (int i = 0; i < n; i++) {
      loopCount++;
      if (capacity >= items[i].weight) {
         totalProfit += items[i].profit;
         capacity -= items[i].weight;
      } else {
         totalProfit += items[i].ratio * capacity;
         break;      }   }
   printf("Maximum Profit (Fractional Knapsack): %.2f\n", totalProfit);
   printf("Number of loops: %d\n", loopCount);
   return 0;
}
```

Output:



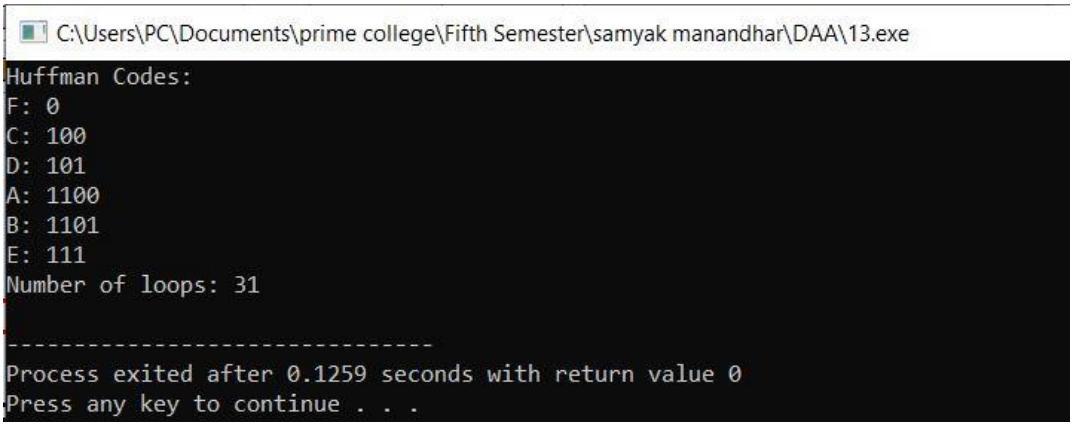
LAB-4.3 Huffmann Coding

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 100
typedef struct Node {
  char data;
  int freq;
  struct Node *left, *right;
} Node;
Node* createNode(char data, int freq) {
  Node* n = (Node*)malloc(sizeof(Node));
  n->data = data;
  n->freq = freq;
  n->left = n->right = NULL;
  return n;}
void printCodes(Node* root, int arr[], int top, int* loopCount) {
  if (root->left) {
    (*loopCount)++;
    arr[top] = 0;
    printCodes(root->left, arr, top + 1, loopCount);   }
  if (root->right) {
    (*loopCount)++;
    arr[top] = 1;
    printCodes(root->right, arr, top + 1, loopCount);   }
  if (!root->left && !root->right) {
    printf("%c: ", root->data);
    for (int i = 0; i < top; i++) printf("%d", arr[i]);
    printf("\n");   }}
Node* buildHuffmanTree(char data[], int freq[], int n, int* loopCount) {
  Node* heap[SIZE];
  int size = n;
  for (int i = 0; i < n; i++) {
    heap[i] = createNode(data[i], freq[i]);
    (*loopCount)++;   }
  while (size > 1) {
    int min1 = 0, min2 = 1;
    (*loopCount)++;
    if (heap[min2]->freq < heap[min1]->freq) {
      int temp = min1; min1 = min2; min2 = temp;      }
    for (int i = 2; i < size; i++) {
```

```
        (*loopCount)++;
        if (heap[i]->freq < heap[min1]->freq) {
          min2 = min1;
          min1 = i;
        } else if (heap[i]->freq < heap[min2]->freq) {
          min2 = i;
        } } Node* left = heap[min1], *right = heap[min2];
      Node* newNode = createNode('-', left->freq + right->freq);
      newNode->left = left;
      newNode->right = right;
      if (min1 < min2) {
        heap[min1] = newNode;
        heap[min2] = heap[size - 1];
      } else {
        heap[min2] = newNode;
        heap[min1] = heap[size - 1];      }
      size--;   }
   return heap[0];}
int main() {
   char data[] = { 'A', 'B', 'C', 'D', 'E', 'F' };
   int freq[] = { 5, 9, 12, 13, 16, 45 };
   int n = sizeof(data) / sizeof(data[0]);
   int arr[100], top = 0;
   int loopCount = 0;
   Node* root = buildHuffmanTree(data, freq, n, &loopCount);
   printf("Huffman Codes:\n");
   printCodes(root, arr, top, &loopCount);
   printf("Number of loops: %d\n", loopCount);
   return 0;
}
```

Output:



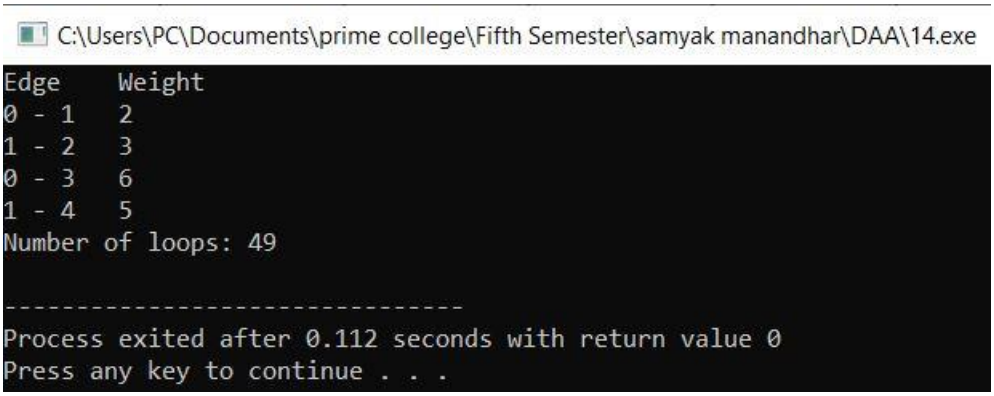
LAB-4.4 Prim's and Kruskal's algorithm

Code:

```
#include <stdio.h>
#include <limits.h>
#define V 5
int minKey(int key[], int mstSet[], int* loopCount) {
  int min = INT_MAX, min_index;
  for (int v = 0; v < V; v++) {
    (*loopCount)++;
    if (!mstSet[v] && key[v] < min) {
      min = key[v];
      min_index = v;
    }
  }
  return min_index;
}
void primMST(int graph[V][V], int* loopCount) {
  int parent[V], key[V], mstSet[V];
  for (int i = 0; i < V; i++) {
    key[i] = INT_MAX;
    mstSet[i] = 0;
    (*loopCount)++;
  }
  key[0] = 0;
  parent[0] = -1;
  for (int count = 0; count < V - 1; count++) {
    (*loopCount)++;
    int u = minKey(key, mstSet, loopCount);
    mstSet[u] = 1;
    for (int v = 0; v < V; v++) {
      (*loopCount)++;
      if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {
        parent[v] = u;
        key[v] = graph[u][v];
      }
    }
  }
  printf("Edge \tWeight\n");
  for (int i = 1; i < V; i++) {
    printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);
  }
```

```
}
int main() {
   int graph[V][V] = {
      { 0, 2, 0, 6, 0 },
      { 2, 0, 3, 8, 5 },
      { 0, 3, 0, 0, 7 },
      { 6, 8, 0, 0, 9 },
      { 0, 5, 7, 9, 0 },
   };
   int loopCount = 0;
   primMST(graph, &loopCount);
   printf("Number of loops: %d\n", loopCount);
   return 0;
}
```

Output:

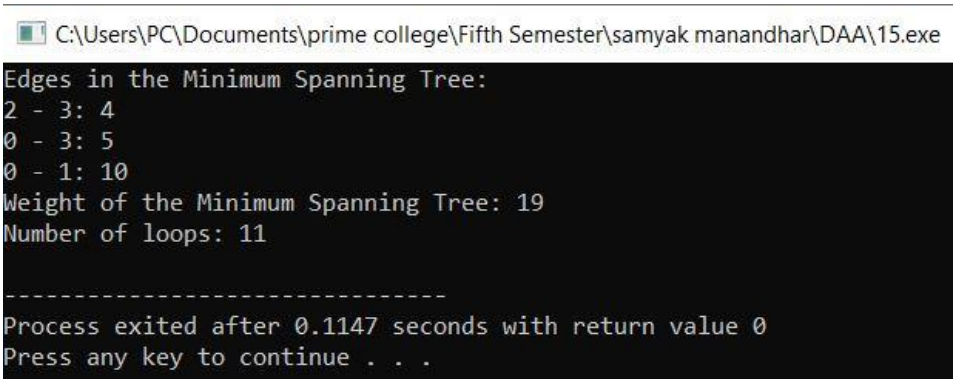


Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct {
   int u, v, weight;
} Edge;
int findParent(int parent[], int i) {
   if (parent[i] == -1) return i;
   return findParent(parent, parent[i]);}
void unionSets(int parent[], int rank[], int x, int y) {
   int xroot = findParent(parent, x);
   int yroot = findParent(parent, y);
   if (rank[xroot] < rank[yroot]) parent[xroot] = yroot;
   else if (rank[xroot] > rank[yroot]) parent[yroot] = xroot;
   else {
```

```
    parent[yroot] = xroot;
    rank[xroot]++;   }}
int compare(const void* a, const void* b) {
  return ((Edge*)a)->weight - ((Edge*)b)->weight;}
void kruskal(int n, Edge edges[], int* loopCount) {
  int parent[MAX], rank[MAX];
  for (int i = 0; i < n; i++) {
    parent[i] = -1;
    rank[i] = 0;
    (*loopCount)++;   }
  qsort(edges, n, sizeof(Edge), compare);
  (*loopCount)++;
  int mstWeight = 0;
  printf("Edges in the Minimum Spanning Tree:\n");
  for (int i = 0; i < n; i++) {
    (*loopCount)++;
    int x = findParent(parent, edges[i].u);
    int y = findParent(parent, edges[i].v);
    if (x != y) {
      unionSets(parent, rank, x, y);
      mstWeight += edges[i].weight;
      printf("%d - %d: %d\n", edges[i].u, edges[i].v, edges[i].weight);     }   }
  printf("Weight of the Minimum Spanning Tree: %d\n", mstWeight);}
int main() {
  Edge edges[] = {
    {0, 1, 10}, {0, 2, 6}, {0, 3, 5}, {1, 3, 15}, {2, 3, 4}   };
  int n = sizeof(edges) / sizeof(edges[0]);
  int loopCount = 0;
  kruskal(n, edges, &loopCount);
  printf("Number of loops: %d\n", loopCount);
  return 0;}
```

Output:



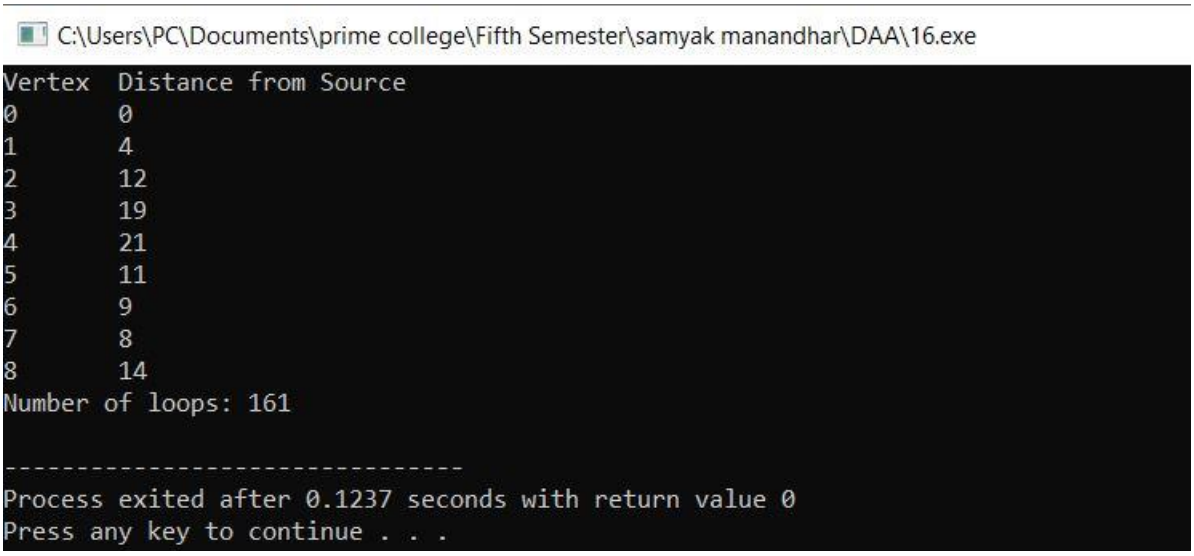
LAB-4.5 Dijkstra's Algorithm

Code:

```
#include <stdio.h>
#include <limits.h>
#define V 9
int minDistance(int dist[], int sptSet[], int* loopCount) {
  int min = INT_MAX, min_index;
  for (int v = 0; v < V; v++) {
    (*loopCount)++;
    if (!sptSet[v] && dist[v] <= min) {
      min = dist[v];
      min_index = v;
    }
  }
  return min_index;
}
void dijkstra(int graph[V][V], int src, int* loopCount) {
  int dist[V], sptSet[V];
  for (int i = 0; i < V; i++) {
    dist[i] = INT_MAX;
    sptSet[i] = 0;
    (*loopCount)++;
  }
  dist[src] = 0;
  for (int count = 0; count < V - 1; count++) {
    (*loopCount)++;
    int u = minDistance(dist, sptSet, loopCount);
    sptSet[u] = 1;
    for (int v = 0; v < V; v++) {
      (*loopCount)++;
      if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v]) {
        dist[v] = dist[u] + graph[u][v];
      }
    }
  }
  printf("Vertex \tDistance from Source\n");
  for (int i = 0; i < V; i++) {
    printf("%d \t%d\n", i, dist[i]);
  }
}
int 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}
    };
    int loopCount = 0;
    dijkstra(graph, 0, &loopCount);
    printf("Number of loops: %d\n", loopCount);
    return 0;
}
```

Output:



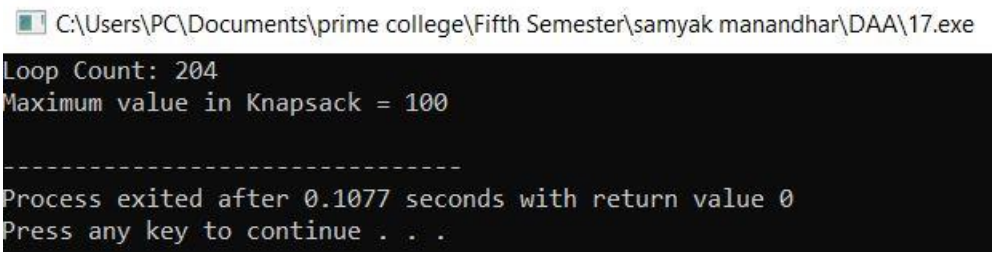
Dynamic Programming

LAB-5.1 0/1 Knapsack Problem

Code:

```
#include <stdio.h>
int knapsack(int W, int wt[], int val[], int n) {
   int dp[n+1][W+1];
   int loopCount = 0;
   for (int i = 0; i <= n; i++) {
      for (int w = 0; w <= W; w++) {
         loopCount++;
         if (i == 0 || w == 0)
            dp[i][w] = 0;
         else if (wt[i-1] <= w)
            dp[i][w] = (val[i-1] + dp[i-1][w-wt[i-1]] > dp[i-1][w]) ? (val[i-1] + dp[i-1][w-wt[i-1]]) : dp[i-1][w];
         else
            dp[i][w] = dp[i-1][w];
      }
   }
   printf("Loop Count: %d\n", loopCount);
   return dp[n][W];
}
int main() {
   int val[] = {30, 40, 60};
   int wt[] = {10, 20, 30};
   int W = 50;
   int n = sizeof(val) / sizeof(val[0]);
   printf("Maximum value in Knapsack = %d\n", knapsack(W, wt, val, n));
   return 0;
}
```

Output:

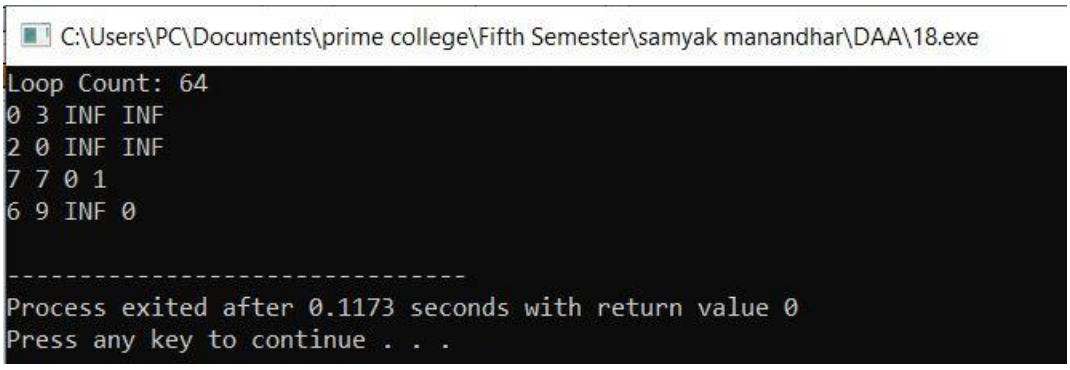


LAB-5.2 Floyd Warshall Problem

Code:

```
#include <stdio.h>
#define INF 99999
void floydWarshall(int graph[][4], int V) {
  int dist[V][V];
  int loopCount = 0;
  for (int i = 0; i < V; i++)
    for (int j = 0; j < V; j++)
      dist[i][j] = graph[i][j];
  for (int k = 0; k < V; k++) {
    for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++) {
        loopCount++;
        if (dist[i][j] > dist[i][k] + dist[k][j])
          dist[i][j] = dist[i][k] + dist[k][j];
      }    }  }
  printf("Loop Count: %d\n", loopCount);
  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
      if (dist[i][j] == INF)
        printf("INF ");
      else
        printf("%d ", dist[i][j]);      }
    printf("\n");   }}
int main() {
  int graph[4][4] = { {0, 3, INF, INF}, {2, 0, INF, INF}, {INF, 7, 0, 1}, {6, INF, INF, 0} };
  int V = 4;
  floydWarshall(graph, V);
  return 0;
}
```

Output:

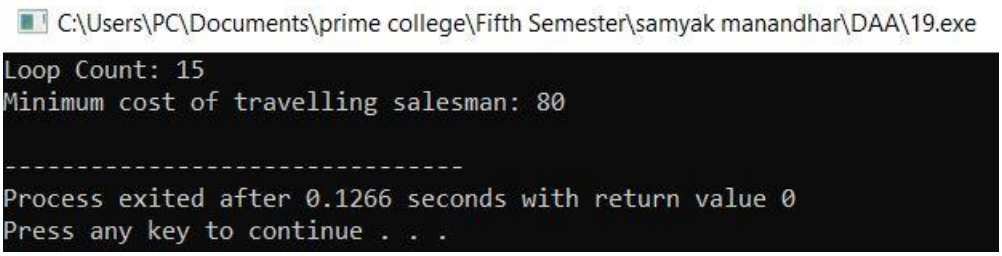


LAB-5.3 Travelling Salesman

Code:

```
#include <stdio.h>
#include <limits.h>
#define V 4
#define INF INT_MAX
int tsp(int dist[][V], int visited[], int currPos, int count, int cost, int ans) {
  static int loopCount = 0;
  if (count == V && dist[currPos][0]) {
    ans = (cost + dist[currPos][0] < ans) ? (cost + dist[currPos][0]) : ans;
    return ans;
  }
  for (int i = 0; i < V; i++) {
    if (!visited[i] && dist[currPos][i]) {
      loopCount++;
      visited[i] = 1;
      ans = tsp(dist, visited, i, count + 1, cost + dist[currPos][i], ans);
      visited[i] = 0;
    }
  }
  if (currPos == 0 && count == 1) {
    printf("Loop Count: %d\n", loopCount);
  }  return ans;
}
int main() {
  int dist[][V] = { {0, 10, 15, 20}, {10, 0, 35, 25}, {15, 35, 0, 30}, {20, 25, 30, 0} };
  int visited[V] = {0};
  visited[0] = 1;
  int ans = INF;
  printf("Minimum cost of travelling salesman: %d\n", tsp(dist, visited, 0, 1, 0, ans));
  return 0;
}
```

Output:



LAB-5.4 Matrix Chain Multiplication (Parenthesis Order)

Code:

```
#include <stdio.h>
#include <limits.h>
int matrixChainOrder(int p[], int n) {
  int m[n][n];
  int s[n][n];
  int loopCount = 0;
  for (int i = 1; i < n; i++)
    m[i][i] = 0;
  for (int len = 2; len < n; len++) {
    for (int i = 1; i < n - len + 1; i++) {
      int j = i + len - 1;
      m[i][j] = INT_MAX;
      for (int k = i; k < j; k++) {
        loopCount++;
        int q = m[i][k] + m[k+1][j] + p[i-1] * p[k] * p[j];
        if (q < m[i][j]) {
          m[i][j] = q;
          s[i][j] = k;
        }       }     }
  }
  printf("Loop Count: %d\n", loopCount);
  return m[1][n-1];
}
int main() {
  int arr[] = {1, 2, 3, 4};
  int n = sizeof(arr) / sizeof(arr[0]);
  printf("Minimum number of multiplications is %d\n", matrixChainOrder(arr, n));
  return 0;
}
```

Output:

```
C:\Users\PC\Documents\prime college\Fifth Semester\samyak manandhar\DAA\20.exe
Loop Count: 4
Minimum number of multiplications is 18

--------------------------------
Process exited after 0.1092 seconds with return value 0
Press any key to continue . . .
```

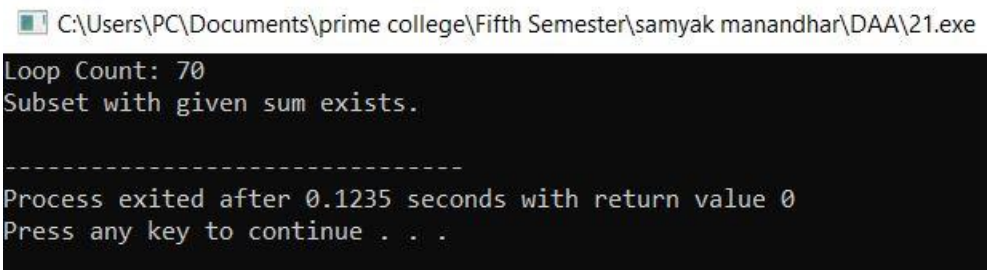
Backtracking Algorithm

LAB-6.1 Sum of subset Problem

Code:

```
#include <stdio.h>
int subsetSum(int set[], int n, int sum) {
   int dp[n+1][sum+1];
   int loopCount = 0;
   for (int i = 0; i <= n; i++) {
      for (int j = 0; j <= sum; j++) {
         loopCount++;
         if (j == 0)
            dp[i][j] = 1;
         else if (i == 0)
            dp[i][j] = 0;
         else if (set[i-1] <= j)
            dp[i][j] = dp[i-1][j] || dp[i-1][j-set[i-1]];
         else
            dp[i][j] = dp[i-1][j];
      }  }
   printf("Loop Count: %d\n", loopCount);
   return dp[n][sum];
}
int main() {
   int set[] = {3, 34, 4, 12, 5, 2};
   int sum = 9;
   int n = sizeof(set) / sizeof(set[0]);
   if (subsetSum(set, n, sum))
      printf("Subset with given sum exists.\n");
   else
      printf("No subset with given sum exists.\n");
   return 0;
}
```

Output:

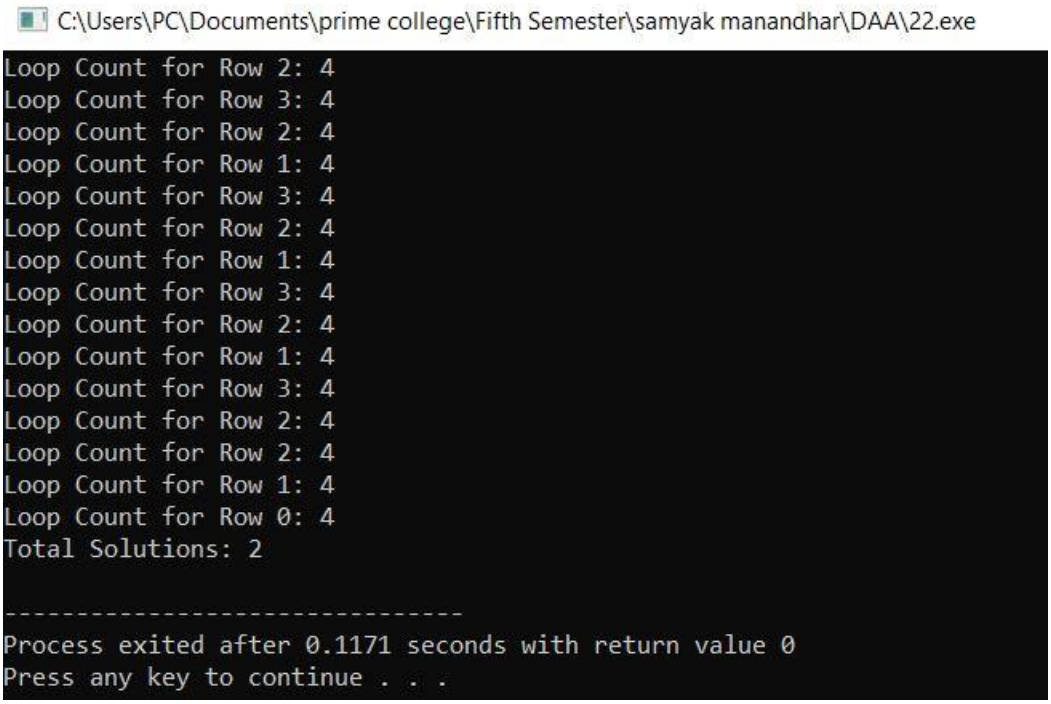


LAB-6.2 N-Queen Problem

Code:

```
#include <stdio.h>
#include <stdbool.h>
int count = 0;
bool isSafe(int board[][10], int row, int col, int N) {
  int i, j;
  for (i = 0; i < row; i++)
    if (board[i][col])
      return false;
  for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
    if (board[i][j])
      return false;
  for (i = row, j = col; i >= 0 && j < N; i--, j++)
    if (board[i][j])
      return false;
  return true;
}
void solveNQueen(int board[][10], int row, int N) {
  int loopCount = 0;
  if (row == N) {
    count++;
    return;
  }
  for (int col = 0; col < N; col++) {
    loopCount++;
    if (isSafe(board, row, col, N)) {
      board[row][col] = 1;
      solveNQueen(board, row + 1, N);
      board[row][col] = 0;
    }
  }
  printf("Loop Count for Row %d: %d\n", row, loopCount);
}
int main() {
  int N = 4;
  int board[10][10] = {0};
  solveNQueen(board, 0, N);
  printf("Total Solutions: %d\n", count);
  return 0;
}
```

Output:


C:\Users\PC\Documents\prime college\Fifth Semester\samyak manandhar\DAA\22.exe
Loop Count for Row 2: 4
Loop Count for Row 3: 4
Loop Count for Row 2: 4
Loop Count for Row 1: 4
Loop Count for Row 3: 4
Loop Count for Row 2: 4
Loop Count for Row 1: 4
Loop Count for Row 3: 4
Loop Count for Row 2: 4
Loop Count for Row 1: 4
Loop Count for Row 3: 4
Loop Count for Row 2: 4
Loop Count for Row 2: 4
Loop Count for Row 1: 4
Loop Count for Row 0: 4
Total Solutions: 2

Process exited after 0.1171 seconds with return value 0
Press any key to continue . . .