ANSWER FOR ADT PRATICAL QUESTIONS

1. Naïve String Matching Algorithm

```
#include <iostream>

#include <string>
using namespace std;

void naiveSearch(string text, string pattern) {
    int n = text.length();
    int m = pattern.length();

    for (int i = 0; i <= n - m; i++) {
        int j;
        for (j = 0; j < m; j++) {
            if (text[i + j] != pattern[j]) break;
        }
        if (j == m) {
            cout << "Pattern found at index " << i << endl;
        }
    }
}

int main() {
    string text = "AABAACAADAABAABA";
    string pattern = "AABA";
    naiveSearch(text, pattern);
    return 0;
}
```

---

2. Rabin-Karp Algorithm

```
#include <iostream>
#include <string>
using namespace std;

#define d 256
void rabinKarp(string text, string pattern, int q) {
    int n = text.length();
    int m = pattern.length();
    int i, j, p = 0, t = 0, h = 1;

    for (i = 0; i < m - 1; i++) h = (h * d) % q;

    for (i = 0; i < m; i++) {
        p = (d * p + pattern[i]) % q;
        t = (d * t + text[i]) % q;
    }

    for (i = 0; i <= n - m; i++) {
        if (p == t) {
            for (j = 0; j < m; j++) {
                if (text[i + j] != pattern[j]) break;
            }
            if (j == m) cout << "Pattern found at index " << i << endl;
        }
        if (i < n - m) {
            t = (d * (t - text[i] * h) + text[i + m]) % q;
```

```
            if (t < 0) t = (t + q);
        }
    }
}

int main() {
    string text = "AABAACAADAABAABA";
    string pattern = "AABA";
    int q = 101; // A prime number
    rabinKarp(text, pattern, q);
    return 0;
}
```

---

3. Knuth-Morris-Pratt (KMP) Algorithm

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

void computeLPSArray(string pattern, vector<int> &lps) {
    int length = 0, i = 1;
    lps[0] = 0;

    while (i < pattern.length()) {
        if (pattern[i] == pattern[length]) {
            length++;
            lps[i] = length;
            i++;
        } else {
```

```
            if (length != 0) length = lps[length - 1];
            else {
                lps[i] = 0;
                i++;
            }
        }
    }
}

void KMP(string text, string pattern) {
    int n = text.length();
    int m = pattern.length();
    vector<int> lps(m);

    computeLPSArray(pattern, lps);
    int i = 0, j = 0;

    while (i < n) {
        if (pattern[j] == text[i]) {
            i++;
            j++;
        }
        if (j == m) {
            cout << "Pattern found at index " << i - j << endl;
            j = lps[j - 1];
        } else if (i < n && pattern[j] != text[i]) {
            if (j != 0) j = lps[j - 1];
            else i++;
        }
```

```
    }
}

int main() {
    string text = "aabaacaadaabaaba";
    string pattern = "aaba";
    KMP(text, pattern);
    return 0;
}
```

---

4. Manacher's Algorithm

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

string preprocessString(string s) {
    string t = "^";
    for (char c : s) {
        t += "#" + string(1, c);
    }
    t += "#$";
    return t;
}

void manacher(string s) {
    string t = preprocessString(s);
    int n = t.length();
    vector<int> p(n, 0);
```

```
    int c = 0, r = 0;

    for (int i = 1; i < n - 1; i++) {
        int mirr = 2 * c - i;
        if (i < r) p[i] = min(r - i, p[mirr]);

        while (t[i + 1 + p[i]] == t[i - 1 - p[i]]) p[i]++;

        if (i + p[i] > r) {
            c = i;
            r = i + p[i];
        }
    }

    int maxLen = 0, center = 0;
    for (int i = 1; i < n - 1; i++) {
        if (p[i] > maxLen) {
            maxLen = p[i];
            center = i;
        }
    }

    cout << "Longest Palindromic Substring: "
         << s.substr((center - maxLen) / 2, maxLen) << endl;
}

int main() {
    string s = "abaaba";
    manacher(s);
```

```
    return 0;
}
```

---

5. Boyer-Moore Algorithm

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

void badCharHeuristic(string str, vector<int> &badChar) {
    for (int i = 0; i < 256; i++) badChar[i] = -1;
    for (int i = 0; i < str.length(); i++) badChar[(int)str[i]] = i;
}

void boyerMoore(string text, string pattern) {
    int n = text.length();
    int m = pattern.length();
    vector<int> badChar(256);
    badCharHeuristic(pattern, badChar);

    int s = 0;
    while (s <= n - m) {
        int j = m - 1;
        while (j >= 0 && pattern[j] == text[s + j]) j--;

        if (j < 0) {
            cout << "Pattern found at index " << s << endl;
            s += (s + m < n) ? m - badChar[text[s + m]] : 1;
        } else {
```

```
            s += max(1, j - badChar[text[s + j]]);
        }
    }
}

int main() {
    string text = "AABAACAADAABAABA";
    string pattern = "AABA";
    boyerMoore(text, pattern);
    return 0;
}
```

6. Knapsack Algorithm (0/1 Knapsack Problem)

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

int knapsack(vector<int> weight, vector<int> profit, int capacity) {
    int n = weight.size();
    vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));

    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= capacity; w++) {
            if (weight[i - 1] <= w) {
                dp[i][w] = max(dp[i - 1][w], profit[i - 1] + dp[i - 1][w - weight[i -
1]]);
            } else {
                dp[i][w] = dp[i - 1][w];
```

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

int main() {
    vector<int> weight = {1, 2, 3};
    vector<int> profit = {10, 15, 40};
    int capacity = 6;

    cout << "Maximum Profit: " << knapsack(weight, profit, capacity) << endl;
    return 0;
}
```

---

7. Assignment Problem

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

#define INF INT_MAX

int findMinCost(vector<vector<int>> &cost, vector<bool> &visited, int worker,
int n) {
    if (worker == n) return 0;

    int minCost = INF;
```

```
    for (int job = 0; job < n; job++) {
        if (!visited[job]) {
            visited[job] = true;
            minCost = min(minCost, cost[worker][job] + findMinCost(cost, visited,
worker + 1, n));
            visited[job] = false;
        }
    }
    return minCost;
}

int main() {
    vector<vector<int>> cost = {{9, 2, 7}, {6, 4, 3}, {5, 8, 1}};
    int n = cost.size();
    vector<bool> visited(n, false);

    cout << "Minimum Assignment Cost: " << findMinCost(cost, visited, 0, n)
<< endl;
    return 0;
}
```

---

8. Floyd-Warshall Algorithm

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

#define INF 1e9
```

```
void floydWarshall(vector<vector<int>> &graph) {
    int n = graph.size();
    vector<vector<int>> dist = graph;

    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] < INF && dist[k][j] < INF) {
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
                }
            }
        }
    }

    cout << "Shortest distances between every pair of vertices:" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dist[i][j] == INF) cout << "INF ";
            else cout << dist[i][j] << " ";
        }
        cout << endl;
    }
}

int main() {
    vector<vector<int>> graph = {
        {0, 3, INF, INF},
        {2, 0, INF, 7},
        {INF, INF, 0, 1},
```

```
        {6, INF, INF, 0}};
    floydWarshall(graph);
    return 0;
}
```

---

9. Coin Exchange Problem

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

int coinChange(vector<int> coins, int sum) {
    vector<int> dp(sum + 1, INT_MAX);
    dp[0] = 0;

    for (int i = 1; i <= sum; i++) {
        for (int coin : coins) {
            if (i - coin >= 0 && dp[i - coin] != INT_MAX) {
                dp[i] = min(dp[i], dp[i - coin] + 1);
            }
        }
    }

    return dp[sum] == INT_MAX ? -1 : dp[sum];
}

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

```
    int sum = 4;

    int result = coinChange(coins, sum);
    if (result != -1)
        cout << "Minimum coins required: " << result << endl;
    else
        cout << "No solution exists." << endl;

    return 0;
}
```

---

10. Longest Common Subsequence

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

int lcs(string s1, string s2) {
    int n = s1.length(), m = s2.length();
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (s1[i - 1] == s2[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            } else {
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
            }
        }
```

```
    }

    return dp[n][m];
}

int main() {
    string s1 = "AGGTAB";
    string s2 = "GXTXAYB";

    cout << "Length of Longest Common Subsequence: " << lcs(s1, s2) << endl;
    return 0;
}
```

11. Longest Palindromic Subsequence

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

int longestPalindromeSubseq(string s) {
    int n = s.length();
    vector<vector<int>> dp(n, vector<int>(n, 0));

    for (int i = n - 1; i >= 0; i--) {
        dp[i][i] = 1;
        for (int j = i + 1; j < n; j++) {
            if (s[i] == s[j]) {
                dp[i][j] = dp[i + 1][j - 1] + 2;
            } else {
                dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
```

```
            }
        }
    }

    return dp[0][n - 1];
}

int main() {
    string s = "abc";
    cout << "Length of Longest Palindromic Subsequence: " <<
longestPalindromeSubseq(s) << endl;
    return 0;
}
```

---

12. Activity Selection Problem

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

void activitySelection(vector<int> start, vector<int> finish) {
    int n = start.size();
    vector<pair<int, int>> activities;

    for (int i = 0; i < n; i++) {
        activities.push_back({finish[i], start[i]});
    }
```

```
    sort(activities.begin(), activities.end());

    cout << "Selected activities are:" << endl;
    int lastFinish = -1;
    for (auto &activity : activities) {
        if (activity.second >= lastFinish) {
            cout << "Activity: (" << activity.second << ", " << activity.first << ")"
<< endl;
            lastFinish = activity.first;
        }
    }
}

int main() {
    vector<int> start = {10, 12, 20};
    vector<int> finish = {20, 25, 30};
    activitySelection(start, finish);
    return 0;
}
```

---

13. Graph Coloring Problem

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

bool isSafe(int node, vector<vector<int>> &graph, vector<int> &colors, int
color, int V) {
    for (int i = 0; i < V; i++) {
```

```
        if (graph[node][i] && colors[i] == color) {
            return false;
        }
    }
    return true;
}

bool graphColoringUtil(vector<vector<int>> &graph, int m, vector<int>
&colors, int node, int V) {
    if (node == V) return true;

    for (int color = 1; color <= m; color++) {
        if (isSafe(node, graph, colors, color, V)) {
            colors[node] = color;

            if (graphColoringUtil(graph, m, colors, node + 1, V)) {
                return true;
            }

            colors[node] = 0;
        }
    }
    return false;
}

void graphColoring(vector<vector<int>> &graph, int m) {
    int V = graph.size();
    vector<int> colors(V, 0);
```

```
    if (graphColoringUtil(graph, m, colors, 0, V)) {
        cout << "Solution exists with the following coloring:" << endl;
        for (int i = 0; i < V; i++) {
            cout << "Vertex " << i + 1 << " -> Color " << colors[i] << endl;
        }
    } else {
        cout << "No solution exists." << endl;
    }
}

int main() {
    vector<vector<int>> graph = {
        {0, 1, 1, 1},
        {1, 0, 1, 0},
        {1, 1, 0, 1},
        {1, 0, 1, 0}};
    int m = 3; // Number of colors
    graphColoring(graph, m);
    return 0;
}
```

---

14. Huffman Coding Compression Algorithm

```
#include <iostream>
#include <queue>
#include <unordered_map>
using namespace std;

struct Node {
```

```
    char ch;
    int freq;
    Node *left, *right;

    Node(char c, int f) : ch(c), freq(f), left(nullptr), right(nullptr) {}
};

struct Compare {
    bool operator()(Node *l, Node *r) {
        return l->freq > r->freq;
    }
};

void printCodes(Node *root, string str) {
    if (!root) return;

    if (root->ch != '$') cout << root->ch << ": " << str << endl;

    printCodes(root->left, str + "0");
    printCodes(root->right, str + "1");
}

void huffmanCoding(string text) {
    unordered_map<char, int> freq;
    for (char ch : text) freq[ch]++;

    priority_queue<Node *, vector<Node *>, Compare> pq;

    for (auto pair : freq) {
```

```
        pq.push(new Node(pair.first, pair.second));
    }

    while (pq.size() > 1) {
        Node *left = pq.top(); pq.pop();
        Node *right = pq.top(); pq.pop();

        Node *top = new Node('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        pq.push(top);
    }

    printCodes(pq.top(), "");
}

int main() {
    string text = "huffman coding algorithm";
    huffmanCoding(text);
    return 0;
}
```

---

15. Minimum Spanning Tree (Prim's Algorithm)

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

```
int findMinKey(vector<int> &key, vector<bool> &mstSet, int V) {
    int minKey = INT_MAX, minIndex;

    for (int v = 0; v < V; v++) {
        if (!mstSet[v] && key[v] < minKey) {
            minKey = key[v];
            minIndex = v;
        }
    }
    return minIndex;
}

void primMST(vector<vector<int>> &graph) {
    int V = graph.size();
    vector<int> key(V, INT_MAX);
    vector<bool> mstSet(V, false);
    vector<int> parent(V, -1);

    key[0] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = findMinKey(key, mstSet, V);
        mstSet[u] = true;

        for (int v = 0; v < V; v++) {
            if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
```

```
        }
    }

    cout << "Edge   Weight" << endl;
    for (int i = 1; i < V; i++) {
        cout << parent[i] << " - " << i << "    " << graph[i][parent[i]] << endl;
    }
}

int main() {
    vector<vector<int>> graph = {
        {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}};
    primMST(graph);
    return 0;
}
```

16. Sieve of Sundaram Algorithm

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

void sieveOfSundaram(int n) {
    int m = (n - 1) / 2;
    vector<bool> marked(m + 1, false);
```

```
    for (int i = 1; i <= m; i++) {
        for (int j = i; (i + j + 2 * i * j) <= m; j++) {
            marked[i + j + 2 * i * j] = true;
        }
    }

    if (n > 2) cout << 2 << " ";

    for (int i = 1; i <= m; i++) {
        if (!marked[i]) cout << 2 * i + 1 << " ";
    }
    cout << endl;
}

int main() {
    int n = 20;
    cout << "Prime numbers up to " << n << " are: ";
    sieveOfSundaram(n);
    return 0;
}
```

---

17. N Queens Problem

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

bool isSafe(vector<vector<int>> &board, int row, int col, int n) {
    for (int i = 0; i < col; i++) {
```

```
        if (board[row][i]) return false;
    }

    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j]) return false;
    }

    for (int i = row, j = col; i < n && j >= 0; i++, j--) {
        if (board[i][j]) return false;
    }

    return true;
}

bool solveNQueensUtil(vector<vector<int>> &board, int col, int n) {
    if (col >= n) return true;

    for (int i = 0; i < n; i++) {
        if (isSafe(board, i, col, n)) {
            board[i][col] = 1;
            if (solveNQueensUtil(board, col + 1, n)) return true;
            board[i][col] = 0;
        }
    }
    return false;
}

void solveNQueens(int n) {
    vector<vector<int>> board(n, vector<int>(n, 0));
```

```
    if (!solveNQueensUtil(board, 0, n)) {
        cout << "No solution exists!" << endl;
        return;
    }

    cout << "Solution for " << n << " queens problem:" << endl;
    for (const auto &row : board) {
        for (int cell : row) {
            cout << (cell ? "Q " : ". ");
        }
        cout << endl;
    }
}

int main() {
    int n = 4;
    solveNQueens(n);
    return 0;
}
```

---

18. Hamiltonian Circuit Problem

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

bool isSafe(int v, vector<vector<int>> &graph, vector<int> &path, int pos) {
    if (graph[path[pos - 1]][v] == 0) return false;
```

```
    for (int i = 0; i < pos; i++) {
        if (path[i] == v) return false;
    }

    return true;
}

bool hamiltonianCycleUtil(vector<vector<int>> &graph, vector<int> &path, int
pos) {
    int n = graph.size();

    if (pos == n) {
        return graph[path[pos - 1]][path[0]] == 1;
    }

    for (int v = 1; v < n; v++) {
        if (isSafe(v, graph, path, pos)) {
            path[pos] = v;
            if (hamiltonianCycleUtil(graph, path, pos + 1)) return true;
            path[pos] = -1;
        }
    }

    return false;
}

void hamiltonianCycle(vector<vector<int>> &graph) {
    int n = graph.size();
```

```
    vector<int> path(n, -1);
    path[0] = 0;

    if (!hamiltonianCycleUtil(graph, path, 1)) {
        cout << "No Hamiltonian Cycle found!" << endl;
        return;
    }

    cout << "Hamiltonian Cycle exists: ";
    for (int v : path) cout << v << " ";
    cout << path[0] << endl;
}

int main() {
    vector<vector<int>> graph = {
        {0, 1, 1, 0},
        {1, 0, 1, 1},
        {1, 1, 0, 1},
        {0, 1, 1, 0}};
    hamiltonianCycle(graph);
    return 0;
}
```

---

19. Subset Sum Problem

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

```
bool subsetSum(vector<int> &arr, int n, int sum) {
    vector<vector<bool>> dp(n + 1, vector<bool>(sum + 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 <= sum; j++) {
            if (arr[i - 1] <= j) {
                dp[i][j] = dp[i - 1][j] || dp[i - 1][j - arr[i - 1]];
            } else {
                dp[i][j] = dp[i - 1][j];
            }
        }
    }

    return dp[n][sum];
}

int main() {
    vector<int> arr = {3, 34, 4, 12, 5, 2};
    int sum = 9;

    if (subsetSum(arr, arr.size(), sum)) {
        cout << "Subset with the given sum exists!" << endl;
    } else {
        cout << "No subset with the given sum exists!" << endl;
    }

    return 0;
```

```
}
```

---

20. Knight's Tour Problem

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

bool isSafe(int x, int y, int n, vector<vector<int>> &board) {
    return (x >= 0 && y >= 0 && x < n && y < n && board[x][y] == -1);
}

bool knightTourUtil(int x, int y, int movei, vector<vector<int>> &board,
vector<int> &moveX, vector<int> &moveY, int n) {
    if (movei == n * n) return true;

    for (int k = 0; k < 8; k++) {
        int nextX = x + moveX[k];
        int nextY = y + moveY[k];

        if (isSafe(nextX, nextY, n, board)) {
            board[nextX][nextY] = movei;
            if (knightTourUtil(nextX, nextY, movei + 1, board, moveX, moveY, n))
return true;
            board[nextX][nextY] = -1;
        }
    }

    return false;
```

```
}

void knightTour(int n) {
    vector<vector<int>> board(n, vector<int>(n, -1));
    vector<int> moveX = {2, 1, -1, -2, -2, -1, 1, 2};
    vector<int> moveY = {1, 2, 2, 1, -1, -2, -2, -1};

    board[0][0] = 0;

    if (!knightTourUtil(0, 0, 1, board, moveX, moveY, n)) {
        cout << "Solution does not exist!" << endl;
        return;
    }

    cout << "Knight's Tour solution:" << endl;
    for (const auto &row : board) {
        for (int cell : row) {
            cout << cell << " ";
        }
        cout << endl;
    }
}

int main() {
    int n = 8;
    knightTour(n);
    return 0;
}
```

---

21. Sudoku Solver Problem

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

#define N 9

bool isSafe(int grid[N][N], int row, int col, int num) {
    for (int x = 0; x < N; x++) {
        if (grid[row][x] == num || grid[x][col] == num) return false;
    }

    int startRow = row - row % 3, startCol = col - col % 3;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (grid[i + startRow][j + startCol] == num) return false;
        }
    }
    return true;
}

bool solveSudoku(int grid[N][N]) {
    int row = -1, col = -1;
    bool isEmpty = true;

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (grid[i][j] == 0) {
```

```
                row = i;
                col = j;
                isEmpty = false;
                break;
            }
        }
        if (!isEmpty) break;
    }

    if (isEmpty) return true;

    for (int num = 1; num <= 9; num++) {
        if (isSafe(grid, row, col, num)) {
            grid[row][col] = num;
            if (solveSudoku(grid)) return true;
            grid[row][col] = 0;
        }
    }
    return false;
}

void printGrid(int grid[N][N]) {
    for (int r = 0; r < N; r++) {
        for (int d = 0; d < N; d++) {
            cout << grid[r][d] << " ";
        }
        cout << endl;
    }
}
```

```
int main() {
    int grid[N][N] = {
        {5, 3, 0, 0, 7, 0, 0, 0, 0},
        {6, 0, 0, 1, 9, 5, 0, 0, 0},
        {0, 9, 8, 0, 0, 0, 0, 6, 0},
        {8, 0, 0, 0, 6, 0, 0, 0, 3},
        {4, 0, 0, 8, 0, 3, 0, 0, 1},
        {7, 0, 0, 0, 2, 0, 0, 0, 6},
        {0, 6, 0, 0, 0, 0, 2, 8, 0},
        {0, 0, 0, 4, 1, 9, 0, 0, 5},
        {0, 0, 0, 0, 8, 0, 0, 7, 9}};

    if (solveSudoku(grid)) {
        cout << "Solved Sudoku Grid:" << endl;
        printGrid(grid);
    } else {
        cout << "No solution exists!" << endl;
    }

    return 0;
}
```

---

22. Subset Sum Problem (Duplicate Entry)

This problem has already been provided. Refer to **Program 19**.

---

23. Knight’s Tour Problem (Duplicate Entry)

This problem has already been provided. Refer to **Program 20**.