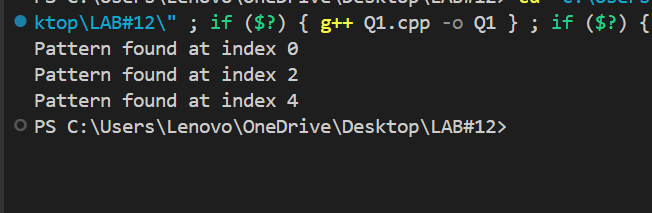
**K23-0607 DS LAB # 12 DEC 8 ,2024**

**QUESTION # 1**



#include <iostream>

#include <cstring>

using namespace std;

void searchPattern(const char\* text, const char\* pattern) {

    int textLength = strlen(text);

    int patternLength = strlen(pattern);

    for (int i = 0; i <= textLength - patternLength; ++i) {

        int j = 0;

        while (j < patternLength && text[i + j] == pattern[j])

            ++j;

        if (j == patternLength)

            cout << "Pattern found at index " << i << endl;

    }

}

int main() {

    const char\* text = "ABABABA";

    const char\* pattern = "ABA";

    searchPattern(text, pattern);

    return 0;

}

**QUESTION # 2**

A computer screen with text and symbols

Description automatically generated

#include <iostream>

using namespace std;

int directions[8][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}, {-1, -1}, {-1, 1}, {1, -1}, {1, 1}};

void dfs(int\*\* grid, int n, int m, int x, int y, bool\*\* visited) {

    visited[x][y] = true;

    for (int i = 0; i < 8; ++i) {

        int newX = x + directions[i][0];

        int newY = y + directions[i][1];

        if (newX >= 0 && newX < n && newY >= 0 && newY < m && grid[newX][newY] == 1 && !visited[newX][newY]) {

            dfs(grid, n, m, newX, newY, visited);

        }

    }

}

int countIslands(int\*\* grid, int n, int m) {

    bool\*\* visited = new bool\*[n];

    for (int i = 0; i < n; ++i) {

        visited[i] = new bool[m];

        for (int j = 0; j < m; ++j)

            visited[i][j] = false;

    }

    int count = 0;

    for (int i = 0; i < n; ++i) {

        for (int j = 0; j < m; ++j) {

            if (grid[i][j] == 1 && !visited[i][j]) {

                dfs(grid, n, m, i, j, visited);

                count++;

            }

        }

    }

    for (int i = 0; i < n; ++i)

        delete[] visited[i];

    delete[] visited;

    return count;

}

int main() {

    int n = 4, m = 2;

    int\*\* grid = new int\*[n];

    grid[0] = new int[m]{0, 1};

    grid[1] = new int[m]{1, 0};

    grid[2] = new int[m]{1, 1};

    grid[3] = new int[m]{1, 0};

    cout << countIslands(grid, n, m) << endl;

    for (int i = 0; i < n; ++i)

        delete[] grid[i];

    delete[] grid;

    return 0;

}

**QUESTION # 3**

#include <iostream>

using namespace std;

void createAdjacencyList(int V, int E, int edges[][2]) {

    int\*\* adjList = new int\*[V];

    int\* sizes = new int[V];

    for (int i = 0; i < V; ++i) {

        adjList[i] = new int[V];

        sizes[i] = 0;

    }

    for (int i = 0; i < E; ++i) {

        int u = edges[i][0];

        int v = edges[i][1];

        adjList[u][sizes[u]++] = v;

        adjList[v][sizes[v]++] = u;

    }

    for (int i = 0; i < V; ++i) {

        cout << i << ": ";

        for (int j = 0; j < sizes[i]; ++j) {

            cout << adjList[i][j] << " ";

        }

        cout << endl;

    }

    for (int i = 0; i < V; ++i)

        delete[] adjList[i];

    delete[] adjList;

    delete[] sizes;

}

int main() {

    int V = 5, E = 7;

    int edges[7][2] = {{0, 1}, {0, 4}, {4, 1}, {4, 3}, {1, 3}, {1, 2}, {3, 2}};

    createAdjacencyList(V, E, edges);

    return 0;

}

A computer screen shot of a computer code

Description automatically generated

**QUESTION # 4**

#include <iostream>

#include <cstring>

using namespace std;

char toLowerCase(char c) {

    if (c >= 'A' && c <= 'Z')

        return c + ('a' - 'A');

    return c;

}

void computeLPSArray(const char\* pattern, int patternLength, int\* lps) {

    int len = 0;

    lps[0] = 0;

    int i = 1;

    while (i < patternLength) {

        if (toLowerCase(pattern[i]) == toLowerCase(pattern[len])) {

            len++;

            lps[i] = len;

            i++;

        } else {

            if (len != 0) {

                len = lps[len - 1];

            } else {

                lps[i] = 0;

                i++;

            }

        }

    }

}

void KMPSearch(const char\* text, const char\* pattern) {

    int textLength = strlen(text);

    int patternLength = strlen(pattern);

    int\* lps = new int[patternLength];

    computeLPSArray(pattern, patternLength, lps);

    int i = 0, j = 0;

    while (i < textLength) {

        if (toLowerCase(pattern[j]) == toLowerCase(text[i])) {

            i++;

            j++;

        }

        if (j == patternLength) {

            cout << "Pattern found at index " << i - j << endl;

            j = lps[j - 1];

        } else if (i < textLength && toLowerCase(pattern[j]) != toLowerCase(text[i])) {

            if (j != 0) {

                j = lps[j - 1];

            } else {

                i++;

            }

        }

    }

    delete[] lps;

}

int main() {

    const char\* text = "Data Structures";

    const char\* pattern = "data";

    KMPSearch(text, pattern);

    return 0;

}

A computer screen with white text

Description automatically generated

**QUESTION # 5**

A black screen with white text

Description automatically generated

#include <iostream>

#include <cstring>

using namespace std;

struct Pattern {

    char\* pattern;

    int length;

    int badCharTable[256];

};

void preprocessBadChar(Pattern& pat) {

    for (int i = 0; i < 256; ++i)

        pat.badCharTable[i] = pat.length;

    for (int i = 0; i < pat.length - 1; ++i)

        pat.badCharTable[(unsigned char)pat.pattern[i]] = pat.length - i - 1;

}

int searchPattern(const char\* text, int textLength, const Pattern& pat) {

    int shift = 0;

    while (shift <= textLength - pat.length) {

        int j = pat.length - 1;

        while (j >= 0 && pat.pattern[j] == text[shift + j])

            --j;

        if (j < 0)

            return shift;

        shift += pat.badCharTable[(unsigned char)text[shift + pat.length - 1]];

    }

    return -1;

}

void searchMultiplePatterns(const char\* text, int textLength, char\*\* patterns, int\* patternLengths, int numPatterns) {

    Pattern\* pats = new Pattern[numPatterns];

    for (int i = 0; i < numPatterns; ++i) {

        pats[i].pattern = patterns[i];

        pats[i].length = patternLengths[i];

        preprocessBadChar(pats[i]);

    }

    for (int i = 0; i < textLength; ++i) {

        for (int j = 0; j < numPatterns; ++j) {

            int foundIndex = searchPattern(text + i, textLength - i, pats[j]);

            if (foundIndex == 0) {

                cout << "Pattern found: " << pats[j].pattern << " at index " << i << endl;

            }

        }

    }

    delete[] pats;

}

int main() {

    const char\* text = "ABCDEFG";

    int textLength = strlen(text);

    char\* patterns[] = { "ABC", "EFG" };

    int patternLengths[] = { 3, 3 };

    int numPatterns = 2;

    searchMultiplePatterns(text, textLength, patterns, patternLengths, numPatterns);

    return 0;

}

**QUESTION # 6**

#include <iostream>

#include <cstring>

using namespace std;

bool isPalindrome(const char\* str, int start, int end) {

    while (start < end) {

        if (str[start] != str[end])

            return false;

        ++start;

        --end;

    }

    return true;

}

void findPalindromes(const char\* text) {

    int n = strlen(text);

    int base = 31, mod = 1000000007;

    int\* hashPrefix = new int[n + 1];

    int\* hashSuffix = new int[n + 1];

    int\* power = new int[n + 1];

    hashPrefix[0] = 0;

    hashSuffix[n] = 0;

    power[0] = 1;

    for (int i = 1; i <= n; ++i) {

        hashPrefix[i] = (1LL \* hashPrefix[i - 1] \* base + (text[i - 1] - 'A' + 1)) % mod;

        power[i] = (1LL \* power[i - 1] \* base) % mod;

    }

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

        hashSuffix[i] = (1LL \* hashSuffix[i + 1] \* base + (text[i] - 'A' + 1)) % mod;

    }

    for (int len = 1; len <= n; ++len) {

        for (int i = 0; i + len - 1 < n; ++i) {

            int j = i + len - 1;

            int prefixHash = (hashPrefix[j + 1] - 1LL \* hashPrefix[i] \* power[len] % mod + mod) % mod;

            int suffixHash = (hashSuffix[i] - 1LL \* hashSuffix[j + 1] \* power[len] % mod + mod) % mod;

            if (prefixHash == suffixHash && isPalindrome(text, i, j)) {

                for (int k = i; k <= j; ++k)

                    cout << text[k];

                cout << endl;

            }

        }

    }

    delete[] hashPrefix;

    delete[] hashSuffix;

    delete[] power;

}

int main() {

    const char\* text = "ABCBAB";

    findPalindromes(text);

    return 0;

}

A screen shot of a computer

Description automatically generated