Problem Solving C++ Lab
Assignment - 5

Q1]

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_set>

using namespace std;

bool isPrime(int num) {
    if (num < 2) return false;
    for (int i = 2; i * i <= num; ++i) {
        if (num % i == 0) return false;
    }
    return true;
}

bool isOneDigitDifferent(int num1, int num2) {
    int diffCount = 0;
    while (num1 > 0) {
        if (num1 % 10 != num2 % 10) {
            ++diffCount;
            if (diffCount > 1) return false;
        }
        num1 /= 10;
        num2 /= 10;
    }
    return diffCount == 1;
}
```

```
int shortestPathBetweenPrimes(int start, int end) {
    if (start == end) return 0;

    unordered_set<int> visited;
    visited.insert(start);

    queue<pair<int, int>> q;
    q.push({start, 0});

    while (!q.empty()) {
        int current = q.front().first;
        int steps = q.front().second;
        q.pop();

        for (int i = 1000; i <= 9999; ++i) {
            if (isPrime(i) && isOneDigitDifferent(current, i) && visited.find(i)
== visited.end()) {
                if (i == end) return steps + 1;
                q.push({i, steps + 1});
                visited.insert(i);
            }
        }
    }

    return -1; // No valid path found
}

int main() {
    int start1 = 1033, end1 = 8179;
    int start2 = 1373, end2 = 8017;
    int start3 = 1033, end3 = 1033;
    cout << "Shortest path from " << start1 << " to " << end1 << ": " <<
shortestPathBetweenPrimes(start1, end1) << endl;
    cout << "Shortest path from " << start2 << " to " << end2 << ": " <<
shortestPathBetweenPrimes(start2, end2) << endl;
    cout << "Shortest path from " << start3 << " to " << end3 << ": " <<
shortestPathBetweenPrimes(start3, end3) << endl;

    return 0;
}
```

Q2]

```
#include <iostream>
#include <cmath>

using namespace std;

string constructPalindrome(int n, int k) {
    string result = "1";
    for (int i = 1; i < n; ++i) {
        result += to_string((i % k) + 1);
    }
    return result;
}

int main() {
    int n1 = 5, k1 = 3;
    int n2 = 2, k2 = 8;

    cout << "Palindrome for n = " << n1 << ", k = " << k1 << ": " <<
constructPalindrome(n1, k1) << endl;
```

```
    cout << "Palindrome for n = " << n2 << ", k = " << k2 << ": " << constructPalindrome(n2, k2) << endl;

    return 0;
}
```

Q3]

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_set>

using namespace std;

bool canFinishTasks(int N, vector<pair<int, int>>& prerequisites) {
    vector<int> inDegree(N, 0);
    vector<vector<int>> graph(N, vector<int>());

    for (auto& prereq : prerequisites) {
        graph[prereq.second].push_back(prereq.first);
        ++inDegree[prereq.first];
    }

    queue<int> q;
    for (int i = 0; i < N; ++i) {
        if (inDegree[i] == 0) {
            q.push(i);
        }
    }

    while (!q.empty()) {
        int current = q.front();
        q.pop();

        for (int neighbor : graph[current]) {
            --inDegree[neighbor];
            if (inDegree[neighbor] == 0) {
                q.push(neighbor);
            }
        }
    }

    for (int degree : inDegree) {
        if (degree > 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int N1 = 4, P1 = 3;
    vector<pair<int, int>> prerequisites1 = {{1, 0}, {2, 1}, {3, 2}};

    int N2 = 2, P2 = 2;
    vector<pair<int, int>> prerequisites2 = {{1, 0}, {0, 1}};

    cout << "Finish all tasks for N = " << N1 << ", P = " << P1 << ": " <<
(canFinishTasks(N1, prerequisites1) ? "Yes" : "No") << endl;
    cout << "Finish all tasks for N = " << N2 << ", P = " << P2 << ": " <<
(canFinishTasks(N2, prerequisites2) ? "Yes" : "No") << endl;

    return 0;
```

```
}
```

Q4]

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <unordered_set>
#include <queue>

using namespace std;

string alienOrder(vector<string>& words) {
    unordered_map<char, unordered_set<char>> graph;
    unordered_map<char, int> inDegree;

    // Build graph and inDegree
    for (string word : words) {
        for (char c : word) {
            graph[c] = unordered_set<char>();
            inDegree[c] = 0;
        }
    }

    for (int i = 0; i < words.size() - 1; ++i) {
        string word1 = words[i];
        string word2 = words[i + 1];
        int minLength = min(word1.length(), word2.length());

        for (int j = 0; j < minLength; ++j) {
            if (word1[j] != word2[j]) {
                graph[word1[j]].insert(word2[j]);
                ++inDegree[word2[j]];
                break;
            }
        }
    }

    // Topological Sort using BFS
    string result = "";
    queue<char> q;

    for (auto& entry : inDegree) {
        if (entry.second == 0) {
            q.push(entry.first);
        }
    }

    while (!q.empty()) {
        char current = q.front();
        q.pop();
        result += current;

        for (char neighbor : graph[current]) {
            --inDegree[neighbor];
            if (inDegree[neighbor] == 0) {
                q.push(neighbor);
            }
        }
    }

    // Check for cycle
    for (auto& entry : inDegree) {
        if (entry.second > 0) {
            return ""; // Cycle detected
        }
    }

    return result;
}

int main() {
    vector<string> words1 = {"baa", "abcd", "abca", "cab", "cad"};
    vector<string> words2 = {"caa", "aaa", "aab"};

    cout << "Order of characters for words1: " << alienOrder(words1)
<< endl;
    cout << "Order of characters for words2: " << alienOrder(words2)
<< endl;

    return 0;
}
```

Q5]

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>

using namespace std;

int minDiceThrows(int N, vector<int>& arr) {
    unordered_map<int, int> ladderSnakeMap;

    for (int i = 0; i < 2 * N; i += 2) {
        ladderSnakeMap[arr[i]] = arr[i + 1];
    }
    queue<pair<int, int>> q;
    q.push({1, 0});

    while (!q.empty()) {
        int currentCell = q.front().first;
        int steps = q.front().second;
        q.pop();

        for (int i = 1; i <= 6; ++i) {
            int nextCell = (currentCell + i > 30) ? 30 : currentCell + i;

            if (ladderSnakeMap.find(nextCell) != ladderSnakeMap.end()) {
                nextCell = ladderSnakeMap[nextCell];
            }

            if (nextCell == 30) {
                return steps + 1;
            }

            q.push({nextCell, steps + 1});
        }
    }

    return -1; // No valid path found
}

int main() {
    int N1 = 8;
    vector<int> arr1 = {3, 22, 5, 8, 11, 26, 20, 29, 17, 4, 19, 7, 27, 1, 21,
9};
```

```
    cout << "Minimum dice throws for N = " << N1 << ": " <<
minDiceThrows(N1, arr1) << endl;

    return 0;
}
```

Q6]

```
#include <iostream>
#include <unordered_map>
#include <vector>

using namespace std;

class graphNode {
public:
    int data;
    vector<graphNode*> neighbours;

    graphNode(int value) : data(value) {}
};

unordered_map<graphNode*, graphNode*> cloneMap;

graphNode* cloneGraph(graphNode* node) {
    if (node == nullptr) {
        return nullptr;
    }

    if (cloneMap.find(node) != cloneMap.end()) {
        return cloneMap[node];
    }
    graphNode* cloneNode = new graphNode(node->data);
    cloneMap[node] = cloneNode;

    for (graphNode* neighbor : node->neighbours) {
        cloneNode->neighbours.push_back(cloneGraph(neighbor));
    }

    return cloneNode;
}

int main() {
    // Example Usage
    graphNode* node1 = new graphNode(1);
    graphNode* node2 = new graphNode(2);
    graphNode* node3 = new graphNode(3);

    node1->neighbours.push_back(node2);
    node1->neighbours.push_back(node3);
    node2->neighbours.push_back(node1);
    node2->neighbours.push_back(node3);
    node3->neighbours.push_back(node1);
    node3->neighbours.push_back(node2);

    graphNode* clonedNode = cloneGraph(node1);

    // You can perform operations on the cloned graph as needed.

    return 0;
}
```