bfs.cpp

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

using namespace std;

void bfs(vector<vector<int>>& grafo, int inicio) {
    int n = grafo.size();
    vector<bool> visitado(n, false);
    queue<int> fila;

    visitado[inicio] = true;
    fila.push(inicio);

    while (!fila.empty()) {
        int atual = fila.front();
        fila.pop();
        cout << "Visitando: " << atual << endl;

        for (int vizinho : grafo[atual]) {
            if (!visitado[vizinho]) {
                visitado[vizinho] = true;
                fila.push(vizinho);
            }
        }
    }
}
```

bipartido.cpp

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

using namespace std;

bool eh_bipartido(vector<vector<int>>& grafo) {
    int n = grafo.size();
    vector<int> cor(n, -1);

    for (int inicio = 0; inicio < n; inicio++) {
        if (cor[inicio] == -1) {
            queue<int> fila;
            fila.push(inicio);
            cor[inicio] = 0;

            while (!fila.empty()) {
                int u = fila.front(); fila.pop();

                for (int v : grafo[u]) {
                    if (cor[v] == -1) {
                        cor[v] = 1 - cor[u];
                        fila.push(v);
```

```
                } else if (cor[v] == cor[u]) {
                    return false;
                }
            }
        }
    }

    return true;
}
```

conexo.cpp

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

using namespace std;

void dfs_conexo(vector<vector<int>>& grafo, vector<bool>& visitado, int v) {
    visitado[v] = true;
    for (int vizinho : grafo[v])
        if (!visitado[vizinho])
            dfs_conexo(grafo, visitado, vizinho);
}

bool eh_conexo(vector<vector<int>>& grafo) {
    int n = grafo.size();
    vector<bool> visitado(n, false);
    dfs_conexo(grafo, visitado, 0);
    for (bool v : visitado)
        if (!v) return false;
    return true;
}
```

dfs.cpp

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

using namespace std;

void dfs_util(vector<vector<int>>& grafo, vector<bool>& visitado, int v) {
    visitado[v] = true;
    cout << "Visitando: " << v << endl;

    for (int vizinho : grafo[v]) {
        if (!visitado[vizinho]) {
            dfs_util(grafo, visitado, vizinho);
        }
    }
}

void dfs(vector<vector<int>>& grafo, int inicio) {
    int n = grafo.size();
    vector<bool> visitado(n, false);
    dfs_util(grafo, visitado, inicio);
```

```
}
```

dijkstra.cpp

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

using namespace std;

void dijkstra(vector<vector<pair<int, int>>>& grafo, int origem) {
    int n = grafo.size();
    vector<int> dist(n, INT_MAX);
    dist[origem] = 0;

    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> fila;
    fila.push({0, origem});

    while (!fila.empty()) {
        int u = fila.top().second;
        int d = fila.top().first;
        fila.pop();

        if (d > dist[u]) continue;

        for (auto [v, peso] : grafo[u]) {
            if (dist[u] + peso < dist[v]) {
                dist[v] = dist[u] + peso;
                fila.push({dist[v], v});
            }
        }
    }

    for (int i = 0; i < n; i++)
        cout << "Distância até " << i << ": " << dist[i] << endl;
}
```

subgrafo.cpp

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

using namespace std;

bool eh_subgrafo(vector<vector<int>>& grafo, vector<vector<int>>& subgrafo) {
    for (int u = 0; u < subgrafo.size(); u++) {
        for (int v : subgrafo[u]) {
            if (find(grafo[u].begin(), grafo[u].end(), v) == grafo[u].end())
                return false;
        }
    }
    return true;
}
```