#include <iostream>

#include <vector>

#include <unordered\_map>

#include <unordered\_set>

#include <algorithm>

Class Graph {

Public:

Int nodes;

Std::vector<int> quality;

Std::unordered\_map<int, std::vector<std::pair<int, int>>> adj;

Graph(int n, const std::vector<int>& qualityValues) : nodes(n), quality(qualityValues) {}

Void addEdge(int u, int v, int time) {

Adj[u].emplace\_back(v, time);

Adj[v].emplace\_back(u, time);

}

Int maxPathQuality(int start, int maxTime) {

Int maxQuality = 0;

Std::vector<int> visited(nodes, 0);

Dfs(start, start, 0, 0, maxTime, visited, maxQuality);

Return maxQuality;

}

Private:

Void dfs(int node, int start, int currentQuality, int currentTime, int maxTime,

Std::vector<int>& visited, int& maxQuality) {

If (currentTime > maxTime) return;

If (visited[node] == 0) {

currentQuality += quality[node];

}

If (node == start) {

maxQuality = std::max(maxQuality, currentQuality);

}

Visited[node]++;

For (auto& [neighbor, time] : adj[node]) {

Dfs(neighbor, start, currentQuality, currentTime + time, maxTime, visited, maxQuality);

}

Visited[node]--;

}

};

Int main() {

Int n = 5;

Std::vector<int> qualityValues = {10, 20, 15, 30, 25};

Graph g(n, qualityValues);

g.addEdge(0, 1, 10);

g.addEdge(1, 2, 10);

g.addEdge(0, 3, 20);

g.addEdge(3, 4, 25);

g.addEdge(2, 4, 15);

int startNode = 0;

int maxTime = 60;

int maxQuality = g.maxPathQuality(startNode, maxTime);

std::cout << “The maximum path quality is: “ << maxQuality << std::endl;

return 0;

}