|  |  |
| --- | --- |
| **Hamiltonian Path and Cycle in C++** | |
| #include <iostream>  #include <vector>  #include <unordered\_set>  using namespace std;  // Structure to represent an edge in the graph  struct Edge {  int src;  int nbr;  int wt;  Edge(int src, int nbr, int wt) {  this->src = src;  this->nbr = nbr;  this->wt = wt;  }  };  // Function to add an edge to the graph  void addEdge(vector<Edge>\* graph, int src, int nbr, int wt) {  graph[src].push\_back(Edge(src, nbr, wt));  graph[nbr].push\_back(Edge(nbr, src, wt)); // Assuming undirected graph  }  // Function to perform Hamiltonian path and cycle calculation  void h(vector<Edge>\* graph, int src, unordered\_set<int>& visited, string psf, int originalSrc) {  if (visited.size() == graph->size() - 1) {  cout << psf;  bool containsCycle = false;  for (Edge& e : graph[src]) {  if (e.nbr == originalSrc) {  containsCycle = true;  break;  }  }  if (containsCycle) {  cout << "\*" << endl;  } else {  cout << "." << endl;  }  return;  }  visited.insert(src);  for (Edge& e : graph[src]) {  if (visited.find(e.nbr) == visited.end()) {  h(graph, e.nbr, visited, psf + to\_string(e.nbr), originalSrc);  }  }  visited.erase(src);  }  int main() {  int vtces = 6; // Number of vertices  //int edges = 7; // Number of edges  // Create the graph using adjacency list representation  vector<Edge>\* graph = new vector<Edge>[vtces];  // Add edges to the graph  addEdge(graph, 0, 1, 10);  addEdge(graph, 0, 3, 40);  addEdge(graph, 1, 2, 10);  addEdge(graph, 2, 3, 10);  addEdge(graph, 3, 4, 2);  addEdge(graph, 4, 5, 2);  addEdge(graph, 2, 4, 3);  int src = 0; // Source vertex  // Perform Hamiltonian path and cycle calculation  unordered\_set<int> visited;  h(graph, src, visited, to\_string(src), src);  delete[] graph; // Deallocate memory  return 0;  } | **Goal:**  Explore all **Hamiltonian paths/cycles** starting from node 0.  **🧩 Graph Summary:**   | **Node** | **Neighbors** | | --- | --- | | 0 | 1, 3 | | 1 | 0, 2 | | 2 | 1, 3, 4 | | 3 | 0, 2, 4 | | 4 | 3, 5, 2 | | 5 | 4 |   **✅ Table Format:**   | **Step** | **Current Node** | **Visited Set** | **Path So Far (psf)** | **Action** | | --- | --- | --- | --- | --- | | 1 | 0 | {0} | "0" | Start | | 2 | 1 | {0,1} | "01" | 0 → 1 | | 3 | 2 | {0,1,2} | "012" | 1 → 2 | | 4 | 3 | {0,1,2,3} | "0123" | 2 → 3 | | 5 | 4 | {0,1,2,3,4} | "01234" | 3 → 4 | | 6 | 5 | {0,1,2,3,4,5} | "012345" | 4 → 5 | | 7 | — | — | "012345." | 6 vertices visited, no edge 5→0 |   ➡️ So we **print: 012345.**  **Let’s try another valid path:**   | **Step** | **Current Node** | **Visited Set** | **Path So Far (psf)** | **Action** | | --- | --- | --- | --- | --- | | 1 | 0 | {0} | "0" | Start | | 2 | 3 | {0,3} | "03" | 0 → 3 | | 3 | 2 | {0,3,2} | "032" | 3 → 2 | | 4 | 1 | {0,3,2,1} | "0321" | 2 → 1 | | 5 | 4 | {0,3,2,1,4} | "03214" | 2 → 4 | | 6 | 5 | {0,3,2,1,4,5} | "032145" | 4 → 5 | | 7 | — | — | "032145." | No edge 5→0, just a path |   ➡️ We print: 032145.  **Let's do a cycle example:**   | **Step** | **Current Node** | **Visited Set** | **Path So Far (psf)** | **Action** | | --- | --- | --- | --- | --- | | 1 | 0 | {0} | "0" | Start | | 2 | 3 | {0,3} | "03" | 0 → 3 | | 3 | 4 | {0,3,4} | "034" | 3 → 4 | | 4 | 2 | {0,3,4,2} | "0342" | 4 → 2 | | 5 | 1 | {0,3,4,2,1} | "03421" | 2 → 1 | | 6 | 5 | {0,3,4,2,1,5} | "034215" | 4 → 5 | | 7 | — | — | "034215\*" | Edge exists 5→0 → CYCLE ✅ |   ➡️ We print: 034215\*  **🔚 Summary of Dry Run:**   | **Path** | **Hamiltonian** | **Cycle?** | | --- | --- | --- | | 012345 | ✅ | ❌ | | 032145 | ✅ | ❌ | | 034215 | ✅ | ✅ | |
| Output:- 01\*  03\* | |