Sure! Let me explain the code for **Prim's Algorithm** in detail.

**Overview of Prim's Algorithm**

Prim's algorithm is a greedy algorithm used to find the **Minimum Spanning Tree (MST)** of a connected, undirected graph. A Minimum Spanning Tree of a graph is a tree that includes all the vertices of the graph and the sum of the weights of the edges in the tree is minimized.

The algorithm works by starting from an arbitrary vertex and progressively adding the shortest edge (in terms of weight) that connects a vertex in the MST to a vertex outside of the MST until all vertices are included.

**Code Walkthrough**

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

#define MAX\_V 100 // Maximum number of vertices, can be adjusted as needed

* **#include <iostream>**: This header file is for input and output operations.
* **#include <vector>**: This is used for working with the std::vector container, though it's not utilized directly in the code.
* **#include <climits>**: This is used to access INT\_MAX, which represents the largest possible integer value.
* **#define MAX\_V 100**: This defines a constant MAX\_V, which sets the maximum number of vertices in the graph. This can be adjusted based on the expected size of the graph.

**minKey Function**

int minKey(int key[], bool mstSet[], int V) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++) {

if (!mstSet[v] && key[v] < min) {

min = key[v];

min\_index = v;

}

}

return min\_index;

}

* **Purpose**: This function finds the vertex with the smallest key value that has not yet been included in the MST (Minimum Spanning Tree).
* **Parameters**:
  + key[]: An array where key[v] represents the minimum weight edge connecting vertex v to the MST.
  + mstSet[]: A boolean array where mstSet[v] is true if vertex v is included in the MST, and false otherwise.
  + V: The number of vertices in the graph.
* **Logic**:
  + It initializes min to INT\_MAX, which will track the minimum key value.
  + It loops through all vertices and finds the vertex with the smallest key[v] that has not been included in the MST (i.e., mstSet[v] == false).
  + Once the vertex is found, it updates the min\_index and returns the index of the vertex with the smallest key value.

**primMST Function**

void primMST(int graph[MAX\_V][MAX\_V], int V) {

int parent[V]; // Array to store constructed MST

int key[V]; // Key values used to pick minimum weight edge

bool mstSet[V]; // To represent the MST set

* **Purpose**: This function implements Prim's algorithm to find the Minimum Spanning Tree (MST) of the given graph.
* **Parameters**:
  + graph[MAX\_V][MAX\_V]: A 2D array representing the adjacency matrix of the graph where graph[i][j] is the weight of the edge between vertex i and vertex j.
  + V: The number of vertices in the graph.
* **Variables**:
  + parent[]: Stores the parent of each vertex in the MST. This is used to print the edges of the MST after it's constructed.
  + key[]: Stores the minimum weight edge for each vertex. Initially, all values are set to INT\_MAX except for the first vertex, which is set to 0.
  + mstSet[]: A boolean array to track whether a vertex is included in the MST.

// Initialize key values and MST set

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

key[i] = INT\_MAX;

mstSet[i] = false;

}

// Always include the first vertex in the MST

key[0] = 0;

parent[0] = -1;

* **Initialization**:
  + All vertices are initially set to have INT\_MAX as their key value, meaning they are not connected to the MST yet.
  + mstSet[] is set to false for all vertices, indicating that no vertex is part of the MST at the beginning.
  + The first vertex (vertex 0) is selected as the starting point, so key[0] = 0 and parent[0] = -1 (since it has no parent).

// The MST will have V vertices

for (int count = 0; count < V - 1; count++) {

int u = minKey(key, mstSet, V); // Get vertex with minimum key value

mstSet[u] = true; // Add vertex to the MST

* **Main Loop**:
  + The loop runs V-1 times, because a tree with V vertices has V-1 edges.
  + In each iteration, it calls the minKey function to find the vertex u with the minimum key value that is not yet in the MST.
  + The vertex u is then added to the MST by setting mstSet[u] = true.

// Update the key and parent values of adjacent vertices

for (int v = 0; v < V; v++) {

if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {

key[v] = graph[u][v];

parent[v] = u;

}

}

}

* **Updating the Key and Parent Values**:
  + After adding vertex u to the MST, the algorithm updates the key and parent values of its adjacent vertices.
  + If there is an edge between vertex u and v (graph[u][v]), and vertex v is not yet in the MST (!mstSet[v]), and the weight of the edge (graph[u][v]) is smaller than the current key value of vertex v (key[v]), then:
    - Update key[v] to the weight of the edge graph[u][v].
    - Set the parent of vertex v to u, indicating that vertex v is connected to the MST via vertex u.

// Print the MST

cout << "Edge \tWeight\n";

for (int i = 1; i < V; i++) {

cout << parent[i] << " - " << i << " \t" << graph[i][parent[i]] << endl;

}

}

* **Printing the MST**:
  + After constructing the MST, the parent[] array contains the parent vertices of each vertex in the MST.
  + The edges of the MST are printed by iterating through parent[] and printing the edges between parent[i] and i along with their weight graph[i][parent[i]].

**Main Function**

int main() {

int V;

cout << "Enter the number of vertices: ";

cin >> V;

int graph[MAX\_V][MAX\_V];

// Take the adjacency matrix as input

cout << "Enter the adjacency matrix (use 0 for no edge, positive integer for weights):\n";

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

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

cin >> graph[i][j];

}

}

// Call Prim's algorithm to find the MST

primMST(graph, V);

return 0;

}

* **Input**:
  + The program asks the user to input the number of vertices V.
  + Then, it takes an adjacency matrix as input, where graph[i][j] represents the weight of the edge between vertices i and j. If there's no edge between i and j, the weight is 0.
* **Execution**:
  + The primMST() function is called to find the MST, and the MST is printed as a list of edges and their weights.

**Conclusion**

The program implements **Prim's Algorithm** to find the Minimum Spanning Tree of a given graph. The algorithm uses a greedy approach by iteratively adding the minimum weight edge that connects a new vertex to the MST. The code constructs and prints the MST by maintaining a parent-child relationship between vertices and updating the key values for adjacent vertices.

Viva Questions :  
Great! Here's a **complete list of expected viva questions and answers** based on:

* The **problem statement**: Prim's Algorithm for Minimum Spanning Tree (MST)
* The **code** you provided (C++ implementation)
* Related **Greedy Method basics**

## ✅ ****Expected Viva Questions & Answers****

### 🔹 ****Q1. What is Prim's Algorithm?****

**Answer:**  
Prim's algorithm is a greedy algorithm that finds the Minimum Spanning Tree (MST) for a connected, weighted, and undirected graph. It starts with a single vertex and continuously adds the smallest edge that connects a vertex in the tree to a vertex outside the tree until all vertices are included.

### 🔹 ****Q2. What is a Minimum Spanning Tree (MST)?****

**Answer:**  
A Minimum Spanning Tree is a subset of edges in a connected, weighted graph that connects all the vertices together, without any cycles, and with the minimum possible total edge weight.

### 🔹 ****Q3. Why is Prim’s Algorithm called a Greedy Algorithm?****

**Answer:**  
Prim’s algorithm makes a greedy choice at every step: it selects the minimum weight edge that connects the tree to a new vertex. It does this locally at each step without considering future consequences, which is the characteristic of a greedy approach.

### 🔹 ****Q4. What are the advantages and disadvantages of the Greedy Method?****

**Answer:**

✅ **Advantages:**

* Simple and easy to implement.
* Efficient for problems like MST, Huffman coding, etc.
* Often faster than other approaches (like Dynamic Programming) for specific problems.

❌ **Disadvantages:**

* May not always give the optimal solution for all problems.
* Only works for problems where local optimum leads to a global optimum (i.e., has the **greedy-choice property** and **optimal substructure**).

### 🔹 ****Q5. What are the characteristics of a Greedy Algorithm?****

**Answer:**

1. **Greedy Choice Property** – A globally optimal solution can be arrived at by selecting a local optimum.
2. **Optimal Substructure** – The problem can be broken down into subproblems that can be solved independently.
3. **Feasible Solution** – The algorithm always chooses the next best possible option that satisfies the problem’s constraints.

### 🔹 ****Q6. How does Prim’s algorithm work in your code?****

**Answer:**

* It initializes all keys as INT\_MAX, except for the starting vertex.
* It picks the minimum key value vertex not included in MST yet.
* It includes that vertex in MST and updates the keys and parent of its adjacent vertices.
* This process continues until MST includes all vertices.

### 🔹 ****Q7. What is the role of the**** key[] ****array?****

**Answer:**  
key[] stores the minimum weight edge that connects a vertex to the tree. It helps in selecting the next vertex with the minimum edge cost.

### 🔹 ****Q8. What does the**** parent[] ****array do?****

**Answer:**  
parent[] keeps track of the parent vertex for each vertex in the MST. It helps reconstruct and display the MST at the end.

### 🔹 ****Q9. What is the purpose of the**** minKey() ****function?****

**Answer:**  
It returns the index of the vertex that is not yet included in the MST and has the smallest key value. This helps in choosing the next vertex to add to the MST.

### 🔹 ****Q10. What does**** mstSet[] ****represent?****

**Answer:**  
mstSet[] is a boolean array that keeps track of vertices included in the MST. If mstSet[i] = true, it means vertex i is already added to MST.

### 🔹 ****Q11. What is the time complexity of Prim’s algorithm (in this implementation)?****

**Answer:**  
In this array-based implementation, time complexity is **O(V²)**. If a min-priority queue and adjacency list were used (e.g., with a min-heap), it would be **O(E log V)**.

### 🔹 ****Q12. What are the input and output formats of your code?****

**Answer:**

* **Input:** Number of vertices and a weighted adjacency matrix.
* **Output:** Edges included in the MST and their corresponding weights.

### 🔹 ****Q13. Can Prim's algorithm be used on disconnected graphs?****

**Answer:**  
No. Prim’s algorithm assumes that the graph is **connected**. If the graph is disconnected, it cannot produce a spanning tree.

### 🔹 ****Q14. What happens if there is no edge between two vertices (i.e., 0 in the matrix)?****

**Answer:**  
It means there's no direct connection between those two vertices. In the code, such values are ignored during edge selection.

### 🔹 ****Q15. How is this different from Kruskal’s algorithm?****

**Answer:**

| **Feature** | **Prim’s Algorithm** | **Kruskal’s Algorithm** |
| --- | --- | --- |
| Starting Point | Starts from a single vertex | Starts from sorted edges |
| Data Structure | Adjacency matrix/list | Edge list with Union-Find |
| Suitable for | Dense graphs | Sparse graphs |
| Cycle Detection | Not explicitly needed | Uses Union-Find to avoid cycle |

### 🔹 ****Q16. Is Prim’s Algorithm optimal? Why?****

**Answer:**  
Yes, Prim’s algorithm is optimal for finding the minimum spanning tree in a connected graph because it satisfies both greedy choice and optimal substructure properties.

## 🧠 ****Bonus Conceptual Questions****

### 🔹 ****Q17. What is the difference between a tree and a spanning tree?****

**Answer:**  
A tree is an acyclic connected graph. A **spanning tree** is a subgraph of a graph that connects all vertices together without cycles, using only existing edges of the graph.

### 🔹 ****Q18. What is the significance of weights in the graph?****

**Answer:**  
Weights represent the cost, distance, or value associated with traveling from one vertex to another. The goal of MST is to minimize the total of these weights.

### 🔹 ****Q19. Can the MST be unique?****

**Answer:**  
Yes, the MST is unique **if all the edge weights are distinct**. If some weights are equal, multiple MSTs may exist.

### 🔹 ****Q20. Where is Prim's algorithm used in real life?****

**Answer:**

* Network design (telecom, cable TV, electrical grid)
* Road and transport networks
* Cluster analysis in machine learning
* Circuit design (VLSI layout)

Would you like a **PDF/printable cheat sheet** of these questions and answers for your viva prep?