|  |  |
| --- | --- |
| **Optimize water distribution in C++** | |
| #include <iostream>  #include <vector>  #include <queue>  #include <utility>  using namespace std;  class Pair {  public:  int vtx;  int wt;  Pair(int vtx, int wt) {  this->vtx = vtx;  this->wt = wt;  }  bool operator>(const Pair& other) const {  return this->wt > other.wt;  }  };  int minCostToSupplyWater(int n, vector<int>& wells, vector<vector<int>>& pipes) {  vector<vector<Pair>> graph(n + 1);  for (const auto& pipe : pipes) {  int u = pipe[0];  int v = pipe[1];  int wt = pipe[2];  graph[u].emplace\_back(v, wt);  graph[v].emplace\_back(u, wt);  }  for (int i = 1; i <= n; ++i) {  graph[i].emplace\_back(0, wells[i - 1]);  graph[0].emplace\_back(i, wells[i - 1]);  }  int ans = 0;  priority\_queue<Pair, vector<Pair>, greater<Pair>> pq;  pq.emplace(0, 0);  vector<bool> vis(n + 1, false);  while (!pq.empty()) {  Pair rem = pq.top();  pq.pop();  if (vis[rem.vtx]) continue;  ans += rem.wt;  vis[rem.vtx] = true;  for (const Pair& nbr : graph[rem.vtx]) {  if (!vis[nbr.vtx]) {  pq.push(nbr);  }  }  }  return ans;  }  int main() {  int v = 3, e = 2;  vector<int> wells = {1, 2, 2};  vector<vector<int>> pipes = {{1, 2, 1}, {2, 3, 1}};  cout << minCostToSupplyWater(v, wells, pipes) << endl;  return 0;  } | **🧾 Input:**   * **Number of houses (n)** = 3 * **Wells**: [1, 2, 2] → Cost to build wells at house 1, 2, 3 * **Pipes**:   [1, 2, 1]  [2, 3, 1]  **🔧 Graph Construction (Adjacency List):**   | **Node** | **Connections** | | --- | --- | | 0 | (1,1), (2,2), (3,2) | | 1 | (2,1), (0,1) | | 2 | (1,1), (3,1), (0,2) | | 3 | (2,1), (0,2) |   **🧮 Dry Run of Prim's Algorithm:**   | **Step** | **Min Edge Picked (u→v, wt)** | **Added to MST** | **MST Cost** | **Visited Nodes** | **Heap Contents After Push** | | --- | --- | --- | --- | --- | --- | | 1 | (0→0, 0) | 0 | 0 | {0} | (1,1), (2,2), (3,2) | | 2 | (0→1, 1) | 1 | 1 | {0,1} | (2,2), (3,2), (2,1) | | 3 | (1→2, 1) | 2 | 2 | {0,1,2} | (3,2), (2,2), (3,1) | | 4 | (2→3, 1) | 3 | 3 | {0,1,2,3} | Remaining edges ignored (already visited nodes) |   ✅ All nodes visited.  **✅ Final Output:**  3  **📦 Explanation:**   * Use **well at house 1**: cost 1 * Use **pipe 1–2**: cost 1 * Use **pipe 2–3**: cost 1 ➡️ **Total = 3**   🧠 This is cheaper than building all wells (1+2+2=5) |
| Output:- 3 | |