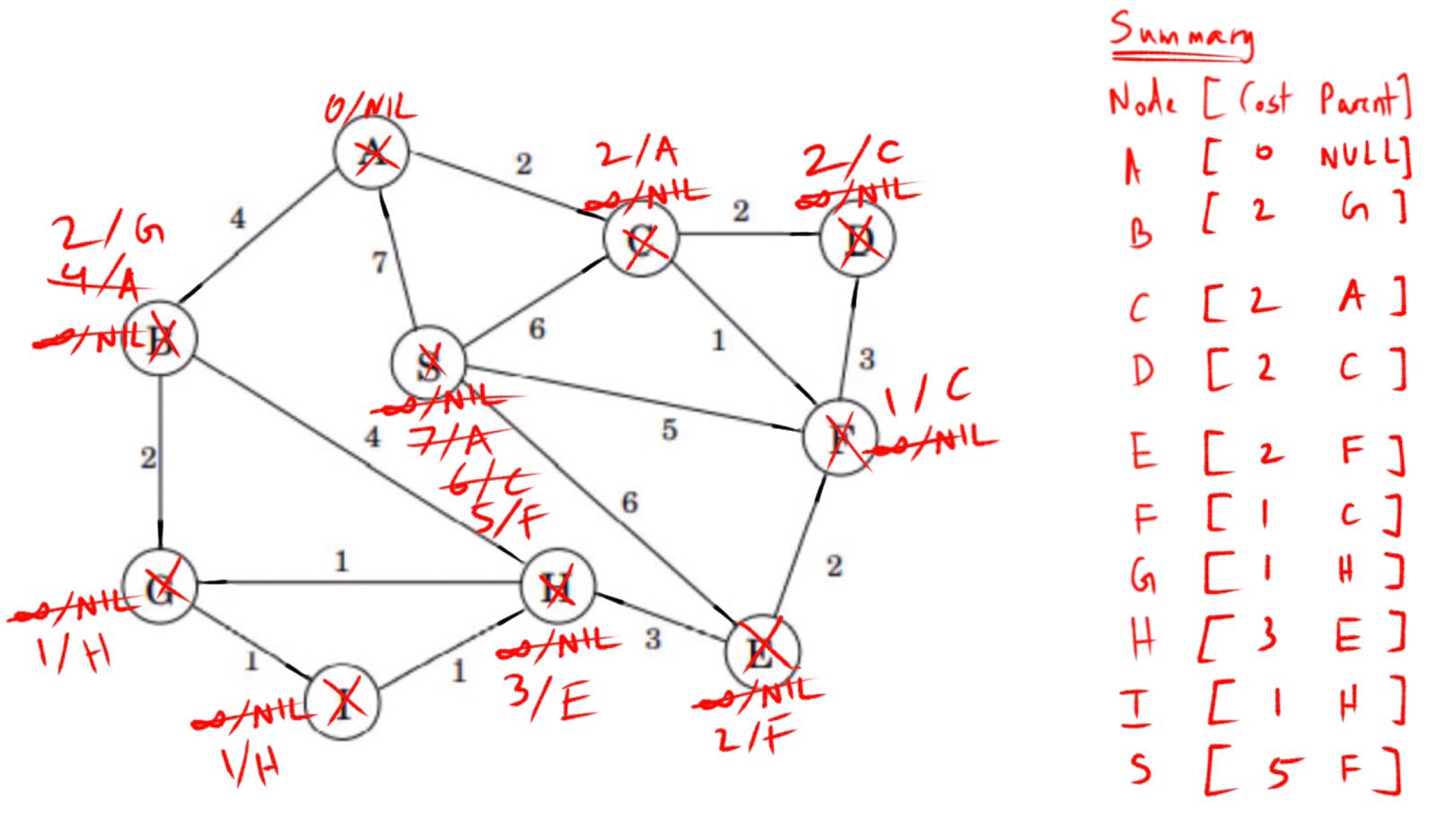
**Muhammad Anas Tanweer 22L-7790 BCS-4C**

**Design & Analysis of Algorithms Assignment 4**

**Problem 1 - Dry Runs**

**Question 1a,** code also available in **Project86.cpp**



Minimum spanning tree cost = Σ(cost) = 0 + 3(1) + 4(2) + 3 + 5 = **19**

**Question 1a END**

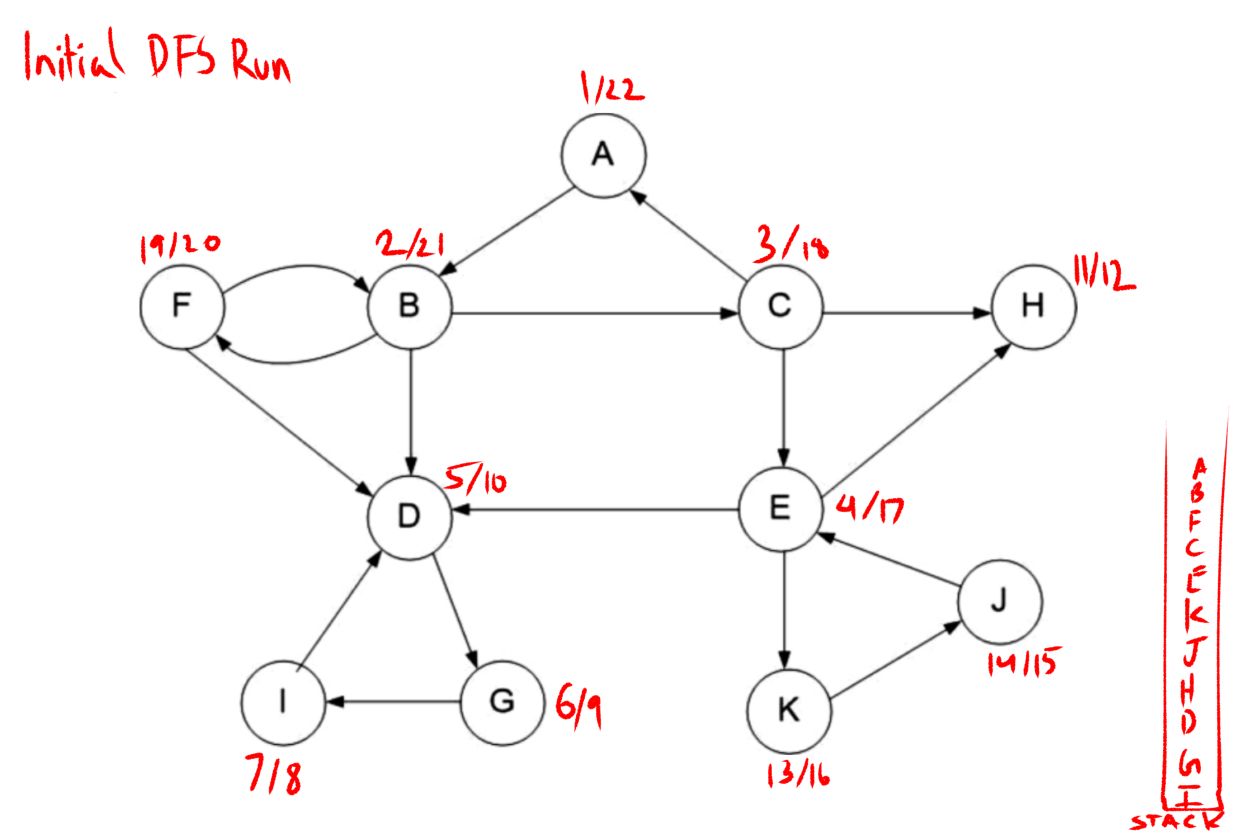
**Question 1b,** A diagram of a diagram with red text

Description automatically generated with medium confidence

Minimum spanning tree cost = Σ(weight) = 3(1) + 4(2) + 3 + 5 = **19**

**Question 1b END**

**Question 2,** code also available in **Project80.cpp**



A diagram of a network

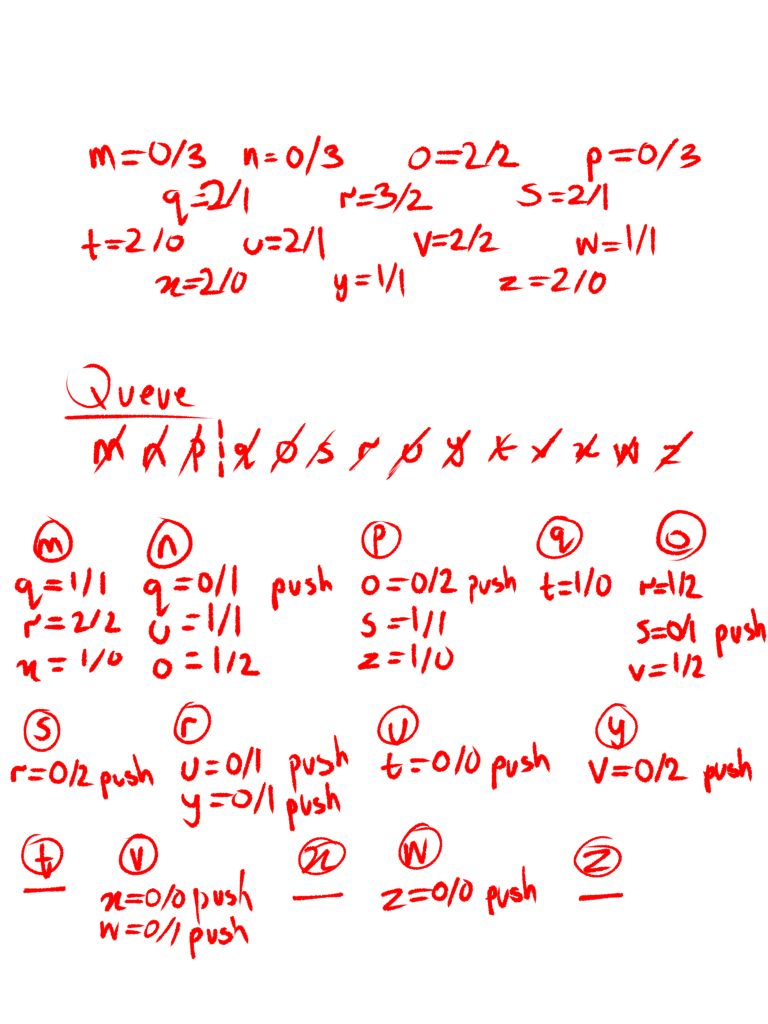
Description automatically generated

A diagram of a diagram

Description automatically generated

**Question 2 END**

**Question 1,**

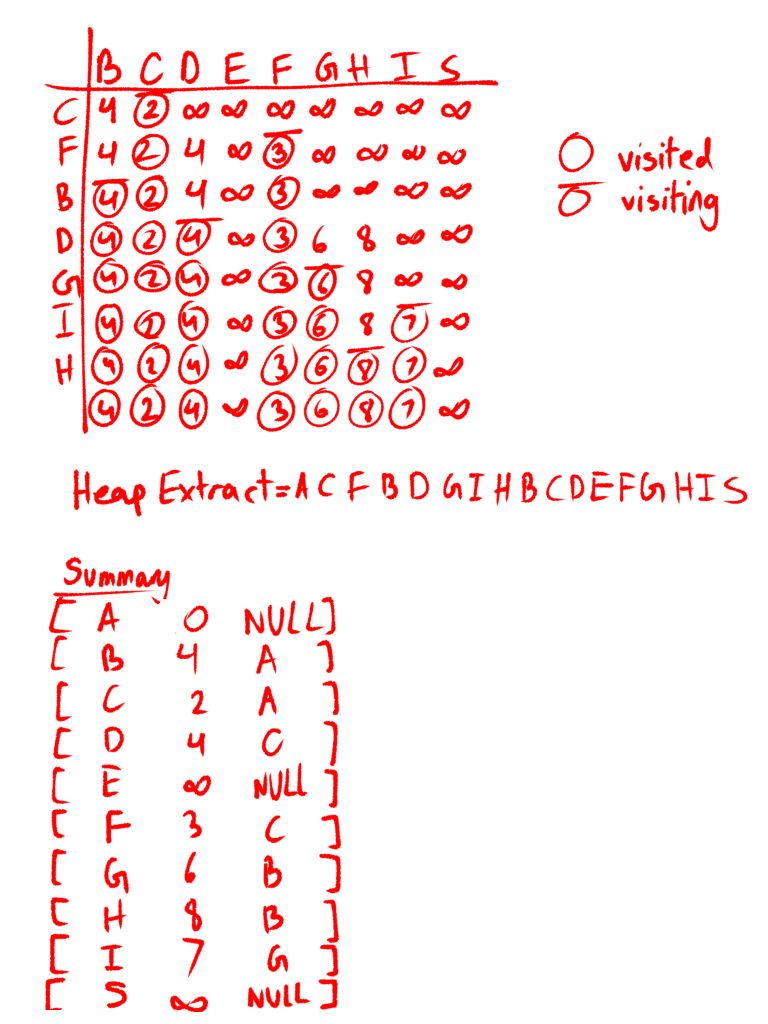


A diagram of a diagram of a diagram

Description automatically generated with medium confidence

**Question 1 END**

**Question 3,** code also available in **Project87.cpp**



**Question 3 END**

**Question 4,** code also available in **Project87.cpp**A white board with red writing

Description automatically generated

A red marker on a white sheet

Description automatically generated with medium confidence

**Question 4 END**

**Problem 2 – Design**

**Question 2a,** code in **Project85.cpp**

|  |
| --- |
| **void DFS(Node\* G)**  {  Node\* u = NULL;  for (int i = 0; i < V; i++)  {  u = start[G->data - 'A'];  if (u->color == "white")  DFSVisit(G, u);  }  }  **void DFSVisit(Node\* G, Node\* u)**  {  start[u->data - 'A']->color = "gray";  Node\* v = adjList[u->data - 'A'];  while (v != NULL)  {  if (start[v->data - 'A']->color == "white")  {  treeEdges.push\_back(Edge(u->data, v->data));  start[v->data - 'A']->parent = start[u->data - 'A'];  DFSVisit(G, v);  }  else if (start[v->data - 'A']->color == "gray")  backEdges.push\_back(Edge(u->data, v->data));  else if (start[v->data - 'A']->parent != start[u->data - 'A'] && start[u->data - 'A']-parent != start[v->data - 'A'])  crossEdges.push\_back(Edge(u->data, v->data));  v = v->next;  }  GLB.push(u->data);  start[u->data - 'A']->color = "black";  } |

|  |
| --- |
| **void PrintDFS()**  {  Node\* u = new Node('A');  DFS(u);  cout << "\*starting from top of stack\*\nStack Order: ";  stack<char>b = GLB;  while (!b.empty())  {  cout << b.top() << " ";  b.pop();  }  cout << endl;  EdgeInfo();  } |

|  |
| --- |
| **void EdgeInfo()**  {  cout << "\033[31mTree Edges:\033[0m" << endl;  for (const auto& edge : treeEdges)  cout << "\t" << edge.from << " -> " << edge.to << endl;  cout << "\033[31mBack Edges:\033[0m" << endl;  for (const auto& edge : backEdges)  cout << "\t" << edge.from << " -> " << edge.to << endl;  cout << "\033[31mCross Edges:\033[0m" << endl;  for (const auto& edge : crossEdges)  cout << "\t" << edge.from << " -> " << edge.to << endl;  } |

**Question 2b,** code in **Project85.cpp**

Only the if statements change

|  |
| --- |
| if (start[v->data - 'A']->color == "white")  {  treeEdges.push\_back(Edge(u->data, v->data));  start[v->data - 'A']->parent = start[u->data - 'A'];  DFSVisit(G, v);  }  else if (start[v->data - 'A']->color == "gray" && start[u->data - 'A']->parent!= start[v->data - 'A'])  backEdges.push\_back(Edge(u->data, v->data));  else if (start[v->data - 'A']->color == "black")  crossEdges.push\_back(Edge(u->data, v->data)); |

**See Project85.cpp for full implementation**

**Question 2 END**

**Question 3,** code in **Project90.cpp**

|  |
| --- |
| **void DFS(Node\* G, char startVertex, char endVertex, int& pathCount)**  {  Node\* u = start[startVertex - 'M'];  if (u->color == "white")  DFSVisit(G, u, endVertex, pathCount);  }  **void DFSVisit(Node\* G, Node\* u, char endVertex, int& pathCount)**  {  Node\* v = adjList[u->data - 'M'];  char Listof = start[u->data - 'M']->data;  char at;  while (v != NULL)  {  at = v->data;  if (v->data == endVertex)  pathCount++;  else  DFSVisit(G, v, endVertex, pathCount);  v = v->next;  }  }  **void CountPaths(char startVertex, char endVertex)**  {  int pathCount = 0;  DFS(start[0], startVertex, endVertex, pathCount);  cout << "\033[31mNumber of Paths From " << startVertex << " to " << endVertex << ":\033[0m " << pathCount << endl;  } |

**See Project90.cpp for full implementation**

**Question 3 END**

**Question 4,** code in **Project80.cpp** & **Project84.cpp**

**Project80.cpp** is the implementation of Kosaraju’s Algo

**Project84.cpp** the implementation of Professor Bacon’s Hypothesis

Both give the same result

**See Project80.cpp** & **Project84.cpp for full implementation**

**Question 4 END**

**Question 5,** code in **Project88.cpp**

The only addition made to Dijkstra Algo was edgeCost. Now the “if” statement also compares and relaxes on the basis of edgeCost (similar logic to minimizing weight cost)

|  |
| --- |
| **void DijkstraRun(char source = 'D')**  {  Node\* a = new Node(source);  minEdgeDijkstra(a);  }  **void minEdgeDijkstra(Node\* r)**  {  start[r->data - 'A']->weight = 0;  start[r->data - 'A']->edgeCost = 0;  priority\_queue<pair<pair<int, int>, char>, vector<pair<pair<int, int>, char>>, greater<pair<pair<int, int>, char>>> minHeap;  for (int i = 0; i < V; i++)  minHeap.push({ {start[i]->weight, start[i]->edgeCost}, start[i]->data });  while (!minHeap.empty())  {  char u\_data = minHeap.top().second;  minHeap.pop();  Node\* u = start[u\_data - 'A'];  start[u->data - 'A']->color = "gray";  Node\* v = adjList[u->data - 'A'];  while (v != NULL)  {  int w = getWeight(u, v);  int p\_w = start[u->data - 'A']->weight;  w += p\_w;  int edgeCount = start[u->data - 'A']->edgeCost + 1;  if (w < start[v->data - 'A']->weight || (w == start[v->data - 'A']->weight && edgeCount < start[v->data - 'A']->edgeCost))  {  v->parent = u;  start[v->data - 'A']->parent = start[u->data - 'A'];  v->weight = w;  start[v->data - 'A']->weight = w;  v->edgeCost = edgeCount;  start[v->data - 'A']->edgeCost = edgeCount;  minHeap.push({ {v->weight, v->edgeCost}, v->data });  }  v = v->next;  }  }  }  **void nodeInfo(char source = 'D')**  {  cout << "\033[36mNode Info\033[0m\n";  cout << "\x1b[31mNode\t[ Cost\tParent\t]\x1b[0m\n";  for (int i = 0; i < V; i++)  {  if (start[i]->parent != NULL)  cout << start[i]->data << "\t[ " << start[i]->weight << "\t" << start[i]->parent->data << "\t]" << endl;  else  cout << start[i]->data << "\t[ " << start[i]->weight << "\tNULL\t]" << endl;  }  pathPrint(source);  }  **void pathPrint(char source)**  {  bool inf = false;  cout << "\033[36mDijkstra Algo Info\033[0m\n";  for (int i = 0; i < V; i++)  {  cout << "\x1b[31m" << source << " -> " << start[i]->data << " Path:\x1b[0m\n\t";  stack<char>s;  Node\* v = start[i];  while (v->data != source)  {  s.push(v->data);  v = v->parent;  if (v == NULL)  {  inf = true;  cout << "INFINITY\n";  break;  }  }  if (!inf)  {  s.push(source);  while (!s.empty())  {  cout << s.top() << " -> ";  s.pop();  }  cout << "\b\b\b \n\tcost = " << start[i]->weight << endl;  cout << "\tedgeCost = " << start[i]->edgeCost << endl;  }  inf = false;  }  } |

**See Project88.cpp for full implementation**

**Question 5 END**

**Question 6,** code in **Project89.cpp**

|  |
| --- |
| **void Dijkstra\_1\_2(Node\* r)**  {  start[r->data - 'A']->weight = 0;  deque<Node\*> q;  q.push\_back(start[r->data - 'A']);  while (!q.empty())  {  Node\* u = q.front();  q.pop\_front();  start[u->data - 'A']->color = "gray";  Node\* v = adjList[u->data - 'A'];  while (v != NULL)  {  int w = getWeight(u, v);  int p\_w = start[u->data - 'A']->weight;  w += p\_w;  if (w < start[v->data - 'A']->weight && start[v->data - 'A']->color == "white")  {  v->parent = u;  start[v->data - 'A']->parent = start[u->data - 'A'];  v->weight = w;  start[v->data - 'A']->weight = w;  if (getWeight(u, v) == 1)  q.push\_front(v);  else  q.push\_back(v);  }  v = v->next;  }  }  } |

**See Project89.cpp for full implementation**

**Question 6 END**