**Report Section: UT Network**

1. Minimum Cost to Connect to UT (Dijkstra)

|  |
| --- |
| 1. Function findMinimumStudentCost (start, dest):    1. For each student s in students:       1. S.minCost = infinity    2. Start.minCost = 0    3. H = buildHeap(students)    4. While H is not empty:       1. If current == dest:          1. Return current.minCost       2. For each neighbor v of current with edge cost c:          1. If current.minCost + c < v.mincost:             1. V.minCost = current.minCost + c             2. changeKey(H, v, v.minCost)    5. return -1 since destination is unreachable   RunTime: V = students E = wires O((V+E) log V) |

1. Minimum Cost to Connect Entire Class to UT (Prim’s)

|  |
| --- |
| 1. Function findMinimumClassCost():    1. For each student s in students:       1. S.minCost = infinity    2. start = UT looking for the last student in the list    3. start.minCost = 0    4. visited = array[v] intiailized to false    5. totalcost = 0    6. heap = buildHeap(students)    7. visitedCount = 0    8. while heap is not empty:       1. current = extractMin(heap)       2. if visited[current.name]: continue       3. visited[current.name] = true       4. visitedCount += 1       5. total Cost += current.minCost       6. for each neighbor v of current with edge cost c:          1. if not visited[v.name] and c < v.minCost:             1. v.minCost = c             2. changeKey(heap, v, c)    9. if visitedCount < v:       1. return -1 disconnected graph    10. return total cost   RunTime: V = students E = wires O((V+E) log V) |

1. Notes on Heap Usage

|  |
| --- |
| 1. buildHeap = O(V) 2. extractMin = O(log V) 3. changeKey = O(log V)  * Dijkstra finds the minCost to connect to UT * Prim’s MST finds the min total cost to connect the entire class * Both algorithms rely on a minheap to effectively select the next cheapest student. Tie breakers are done using the student ID in minCost is equal * Both run in O((V+E) log V) |