

Day_7: DSA

Q4.7 hackcode 2594 Minimum Time To Repair Cars

given: monks: [4, 2, 3, 1] Highup monk
cars = 10
↓ slow process

min	Cars
4	1
16	2
32	3

4 * 10
8 * 10 = time = 400

Search Space
[low, high]
1, 400


Code:


Class Solution:

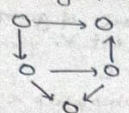
```
def repairCars(self, monks: List[int], cars: int) -> int:
    def count-required(time):
        count = 0
        for m in monks:
            count += int(sqrt(time / m))
        return count

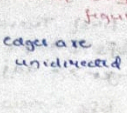
    l, r = 1, monks[0] * cars * cars
    res = -1
    while l <= r:
        m = (l + r) // 2
        repaired = count-required(m)
        if repaired >= cars:
            res = m
            r = m - 1
        else:
            l = m + 1
    return res
```

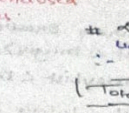
Introduction to graphs: Not only enclosed figures.

Nodes / vertex: 

Edge: 

Undirected graph: 

Directed graph: 

Acyclic: 

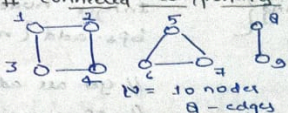
Path: Contains a lot of nodes and each of them are reachable.
 cont. not appear twice Node
 for ex: 1 2 3 5 → path
 1 2 3 2 1 → Not path.

Graph Representation / Java:
 Input: N nodes, M edges
 no. of nodes → 5
 no. of edges → 6

Store:
 → Matrix {Adjacent}
 → List (adjacency)
 Method is not used.

Adjacency List: `ArrayList<ArrayList<>> adj`

Space: $O(2M)$ *
 Edges:
 0 →
 1 → {2, 3}
 2 → {1, 4, 5}
 3 → {1, 4}
 4 → {2, 3, 5}
 5 → {2, 4}

Connected Components:

 There are component of graph
 Vis =

0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

 Visited: 0 1 2 3 4 5 6 7 8 9
 for (i = 1 → 10)
 { if (!visited[i])
 traversal(i);
 }
 In one traversal it only visits the connected part rest is untouched.
 We need again to call traversal.

BFS of a graph: → Traversal Technique
Breadth first search
 Level wise:
 → 1 2 6 3 4 7 8 5
 only one node can be at level = 0
 if starting node = 6
 we are storing graph in Adjacency list:
 → data structure

Starting Node = 1
 → 1 2
 Who are your Neighbours?
 Take out and print
 Queue
 FIFO
 If once visited we need not to do anything.
 Space: $O(3N)$ *
 Time: $O(N) + O(2E)$
 while:
 Vis array:

0	1	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

 already in Queue

Code: BFS

// function to return breadth first traversal of given graph

```

public ArrayList<Integer> bfsOfGraph(int V,
    ArrayList<ArrayList<Integer>> adj) {
    ArrayList<Integer> bfs = new ArrayList<>();
    boolean vis[] = new boolean[V];
    Queue<Integer> q = new LinkedList<>();

    adding in queue {
        q.add(0);
        vis[0] = true;
    }

    Traversing and returning {
        while (!q.isEmpty()) {
            Integer node = q.poll();
            bfs.add(node);

            // get all adjacent vertex of the dequeued vertex
            // If a adjacent has not visited then mark it
            // and enqueue it.
            for (Integer it : adj.get(node)) {
                getting neighbours of each and adding in queue {
                    if (vis[it] == false) {
                        vis[it] = true;
                        q.add(it);
                    }
                }
            }
        }
    }

    return bfs;
}

```

