2045. Second Minimum Time to Reach Destination

A city is represented as a **bi-directional connected** graph with `n` vertices where each vertex is labeled from `1` to `n` (**inclusive**). The edges in the graph are represented as a 2D integer array `edges`, where each `edges[i] = [ui, vi]` denotes a bi-directional edge between vertex `ui` and vertex `vi`. Every vertex pair is connected by **at most one** edge, and no vertex has an edge to itself. The time taken to traverse any edge is `time` minutes.

Each vertex has a traffic signal which changes its color from **green** to **red** and vice versa every `change` minutes. All signals change **at the same time**. You can enter a vertex at **any time**, but can leave a vertex **only when the signal is green**. You **cannot wait** at a vertex if the signal is **green**.

The **second minimum value** is defined as the smallest value **strictly larger** than the minimum value.

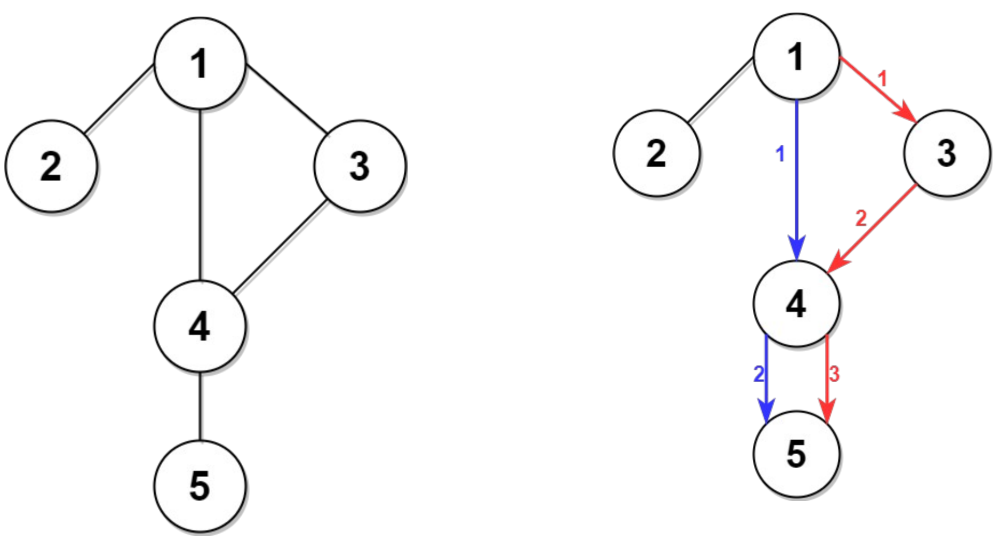
- For example the second minimum value of `[2, 3, 4]` is `3`, and the second minimum value of `[2, 2, 4]` is `4`.

Given `n`, `edges`, `time`, and `change`, return *the **second minimum time** it will take to go from vertex* `1` *to vertex* `n`.

Notes:

- You can go through any vertex **any** number of times, **including** `1` and `n`.
- You can assume that when the journey **starts**, all signals have just turned **green**.

Example 1:



```
Input: n = 5, edges = [[1,2],[1,3],[1,4],[3,4],[4,5]], time = 3, change = 5
Output: 13
Explanation:
The figure on the left shows the given graph.
The blue path in the figure on the right is the minimum time path.
The time taken is:
- Start at 1, time elapsed=0
- 1 -> 4: 3 minutes, time elapsed=3
- 4 -> 5: 3 minutes, time elapsed=6
Hence the minimum time needed is 6 minutes.

The red path shows the path to get the second minimum time.
- Start at 1, time elapsed=0
- 1 -> 3: 3 minutes, time elapsed=3
- 3 -> 4: 3 minutes, time elapsed=6
- Wait at 4 for 4 minutes, time elapsed=10
- 4 -> 5: 3 minutes, time elapsed=13
Hence the second minimum time is 13 minutes.
```

Example 2:


```
Input: n = 2, edges = [[1,2]], time = 3, change = 2
Output: 11
Explanation:
The minimum time path is 1 -> 2 with time = 3 minutes.
The second minimum time path is 1 -> 2 -> 1 -> 2 with time = 11 minutes.
```

Constraints:

- `2 <= n <= 104`
- `n - 1 <= edges.length <= min(2 * 104, n * (n - 1) / 2)`
- `edges[i].length == 2`
- `1 <= ui, vi <= n`
- `ui != vi`
- There are no duplicate edges.
- Each vertex can be reached directly or indirectly from every other vertex.
- `1 <= time, change <= 103`

2045. Second Minimum Time to Reach Destination

```
/*
   Dijkstra
   G(V,E), where E=(n*(n-1)/2),V=n
   Time complexity: O(E+ElogV)
   Extra space complexity: O(EV+V+V+2V)
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;

typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;

class Solution {
   public:
      vvi graph;
   public:
      void build_graph(int n,std::vector<std::vector<int>>& edges){
         graph.resize(n);
         for(auto& edge: edges){
            int u=--edge[0];
            int v=--edge[1];
            graph[u].push_back(v);
            graph[v].push_back(u);
         }
      }
```

```
int dijktra(int n,int change,int time,int start,int end){
        std::vector<int> min_dist(n,INT_MAX),second_min_dist(n,INT_MAX);

        min_dist[start]=0;

        std::priority_queue<ii,vii,std::greater<ii>> q;
        q.push({0,start});
        while(!q.empty()){
            auto [current_time,u]=q.top();
            q.pop();

            if(u==end && second_min_dist[end]!=INT_MAX) return second_min_dist[end];

            int cycle=current_time/change;

            if(cycle%2) current_time=change*(cycle+1);

            for(auto& v: graph[u]){
                int next_time=current_time+time;
                if(min_dist[v]>next_time){
                    second_min_dist[v]=min_dist[v];
                    min_dist[v]=next_time;
                    q.push({min_dist[v],v});
                }
                else if(min_dist[v]<next_time && second_min_dist[v]>next_time){
                    second_min_dist[v]=next_time;
                    q.push({second_min_dist[v], v});
                }
            }
        }

        return -1; // Never reached
    }

    int secondMinimum(int n, std::vector<std::vector<int>>& edges, int time, int change) {
        build_graph(n,edges);

        return dijktra(n,change,time,0,n-1);
    }
};
```

2045. Second Minimum Time to Reach Destination

```
/*
   BFS
   G(V,E), where E=(n*(n-1)/2),V=n
   Time complexity: O(E+V)
   Extra space complexity: O(EV+V+V+2V)
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;

typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;

class Solution {
   public:
      vvi graph;
   public:
      void build_graph(int n,std::vector<std::vector<int>>& edges){
         graph.resize(n);
         for(auto& edge: edges){
            int u=--edge[0];
            int v=--edge[1];
            graph[u].push_back(v);
            graph[v].push_back(u);
         }
      }
```

```
int bfs(int n,int change,int time,int start,int end){
        std::vector<int> min_dist(n,INT_MAX),second_min_dist(n,INT_MAX);

        min_dist[start]=0;

        std::queue<ii> q;
        q.push({0,1}); // {node,frequency}
        while(!q.empty()){
            auto [u,f]=q.front();
            q.pop();

            int current_time=f==1?min_dist[u]:second_min_dist[u];

            int cycle=current_time/change;

            if(cycle%2) current_time=change*(cycle+1);

            if(u==end && second_min_dist[u]!=INT_MAX ) return second_min_dist[u];

            for(auto& v: graph[u]){
                int next_time=current_time+time;
                if(min_dist[v]>next_time){
                    second_min_dist[v]=min_dist[v];
                    min_dist[v]=next_time;
                    q.push({v,1});
                }
                else if(min_dist[v]<next_time && second_min_dist[v]>next_time){
                    second_min_dist[v]=next_time;
                    q.push({v,2});
                }
            }
        }

        return -1; // Never reached
    }

    int secondMinimum(int n, std::vector<std::vector<int>>& edges, int time, int change) {
        build_graph(n,edges);

        return bfs(n,change,time,0,n-1);
    }
};
```