Travel Fast

This problem was worth 1500 points.
The author of this problem is Aryan V S.

Note: GitHub does not support LaTex in Markdown. If you want a more readable version of the problem, download the PDF file instead.

Statement

Aryan and Dhruv were recently employed at Evil-Corp. Until now, all the work was to be done in online work-from-home mode but since restrictions due to the ongoing pandemic are being eased, Evil-Corp requires all employees to be present in-person for offline mode. An announcement related to the same was released in a hurry.

Aryan and Dhruv are now worried about the time it would take them to reach the head office of Evil-Corp from their current city.

The current city in which they are in is x. They need to reach the head office located in the city y.

The cities can be described as a connected graph of n nodes and m bidirectional edges. The graph has no self loops or multiple edges.

The travel time from city u_i to city v_i is t_i. Additionally, since they are using the futuristic Hyperloop mode of travel, there is another constraint. The Hyperloops only depart at multiples k_{ij} $(1 \le j \le 5)$ starting at time 0 from the city u_i.

Entering, exiting or changing a Hyperloop transport at a city takes negligible time and can be considered 0 seconds.

Since Aryan and Dhruv are too busy worrying, they ask you to help them find the minimum time to reach the head office.

Input Format

The first line contains four integers n, m, x and y.

The following m lines contain eight integers each.

Each line is described as: u_i, v_i, t_i, k_{i1}, k_{i2}, k_{i3}, k_{i4}, k_{i5}.

Output Format

The output should contain a single integer - the minimum time it would take to reach the head office.

Constraints

$1 \le n \le 2 \cdot 10^5$

$n - 1 \leq m \leq 4 \cdot 10^5$

$1 \leq x, y, u_i, v_i \leq n$

$1 \leq t_i \leq 10^5$

$1 \leq k_{ij} \leq 1000$

Sample Tests

Sample Test 1

Input

```
5 5 1 5
1 2 20 5 7 9 11 13
1 3 14 5 7 9 11 13
3 4 1 5 7 9 11 13
3 5 28 5 7 9 11 13
4 2 17 5 7 9 11 13
```

Output

```
42
```

Explanation

The optimal path is $1 \rightarrow 3 \rightarrow 5$.

- As all Hyperloops start at 0 seconds, there is no wait time at city 1
- It takes 14 seconds to reach city 3 from city 1. From city 3 to city 5, Hyperloops run at multiples of $5, 7, 9, 11, 13$. As the total time right now is 14 seconds, there is no wait time at city 3 either because at the 14^{th} second, a Hyperloop running at multiples of 7 is about to start
- It takes 28 seconds to reach the city 5 from station 3.

The total time taken is $14 + 28 = 42$ seconds.

Sample Test 2

Input

```
5 5 1 5
1 2 4 10 11 13 6 19
1 3 100 10 11 13 6 19
3 4 7 10 11 13 6 19
3 5 5 10 11 13 6 19
4 2 9 10 11 13 6 19
```

Output

```
31
```

Explanation

Sample Test 3

Input

```
5 4 2 4
2 1 100 1000 999 998 997 996
1 3 100 100 200 100 200 300
5 3 100 144 42 1 69 57
4 5 100 3 5 7 9 11
```

Output

```
400
```

Solution

Time Complexity: $O(n \cdot log(n))$

Memory Complexity: $O(n)$

This problem is a simple Dijkstra's Algorithm Shortest Path problem, but with a slight twist. In the normal Dijkstra problem, you are given the edge weights directly but here, the edge weights are dynamic (not exactly, but the edge weights are determined by some function that takes two parameters) and need to be calculated based on two parameters:

- the time spent to reach current city
- the time we have to wait until the next HyperLoop arrives

We already know the first parameter for every city (it is exactly the same as done in normal Dijkstra implementation i.e. $dist_u$). To account for the second "wait" parameter, where we must wait for the HyperLoop to arrive, we can try finding the smallest multiple of the wait times greater than or equal to the time it took us to reach the current city. This ("wait" time) at a city can be calculated as:

$$\lceil \frac{dist_u}{k_{uj}} \rceil \cdot k_{uj} - dist_u$$

for the current city u. We can try all the five values of k_{uj} and see which one produces the minimum wait time. Everything else is the exact same copy-pasted version of Dijkstra.

Solution in C++:

```cpp
/* Arrow */

#ifdef LOST_IN_SPACE
#  if   __cplusplus > 201703LL
#    include "lost_pch1.h" // C++20
#  elif __cplusplus > 201402LL
#    include "lost_pch2.h" // C++17
#  else
#    include "lost_pch3.h" // C++14
#  endif
#else
#  include <bits/stdc++.h>
#endif
```

```cpp

constexpr bool test_cases = false;

void solve () {
  int n, m, x, y;
  std::cin >> n >> m >> x >> y;
  --x;
  --y;

  struct node {
    int v;
    int64_t t;
    std::array <int64_t, 5> k;
  };

  std::vector <std::vector <node>> graph (n);
  std::array <int64_t, 5> buffer;
  for (int i = 0; i < m; ++i) {
    int u, v, t;
    std::cin >> u >> v >> t;
    --u; --v;

    for (int j = 0; j < 5; ++j)
      std::cin >> buffer[j];

    graph[u].push_back({v, t, buffer});
    graph[v].push_back({u, t, buffer});
  }

  const int64_t inf = 1e18 + 42;
  std::vector <int64_t> dist (n, inf);
  std::priority_queue <std::pair <int64_t, int>> pq;

  dist[x] = 0;
  pq.push({0, x});

  while (!pq.empty()) {
    auto [c, u] = pq.top();
    pq.pop();

    if (-c != dist[u])
      continue;

    for (auto& [v, t, k]: graph[u]) {
      int64_t min_wait = inf;
      for (int i = 0; i < 5; ++i) {
        int64_t wait = (dist[u] + k[i] - 1) / k[i] * k[i] - dist[u];
        min_wait = std::min(min_wait, wait);
      }
      assert(min_wait >= 0);

      if (dist[u] + min_wait + t < dist[v]) {
        dist[v] = dist[u] + min_wait + t;
        pq.push({-dist[v], v});
      }
    }
  }

  std::cout << dist[y] << '\n';
}

int main () {
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);
  std::cout.precision(10);
  std::cerr.precision(10);
  std::cout << std::fixed << std::boolalpha;
  std::cerr << std::fixed << std::boolalpha;

  int cases = 1;
  if (test_cases)
    std::cin >> cases;
  while (cases--)
    solve();

  return 0;
}
```

Solution in Python:

I've never implemented Dijkstra in Python as it isn't my primary programming language. If any participant would like to implement the Dijkstra solution for this problem and send it over, I'll be glad to include it here.