# Editorial: Bicycles - 1915G

Avighna

June 2024

You start off in city 1 with a bike with a slowness factor of $s_1$. Let's try traversing all edges in a sort of brute-force manner. We reach new cities. If there's a better bike available here, buy it. Now repeat: from each of the new cities, traverse every edge to visit every other city you can.

However, just doing this will cause you to visit city 1 again, possibly with no improvements made to your bike, so you'll be at the same state that you were in the beginning. We don't want this, however, sometimes it'll be worth going back to a city if you have a better bike. To prevent this, let's create a two-dimensional array $vis[a][b]$ ($vis[city][bike]$) that keeps track of whether we've visited city $a$ with a bike with a slowness factor of $b$. Use this to visit each city-bike pair not more than once.

Let's also make another two-dimensional array called $dist[a][b]$ ($dist[city][bike]$) that will store the best distance we've found from city 1 to city $a$ at a stage where the best bike we have has a slowness factor of $b$. We're including the bike at city $a$ for this. If we're at city $c_1$ and we're going to city $c_2$ with an edge with a weight of $w$ on a bike with a slowness factor of $f$, then if $d + wf < dist[c_2][min(f, s_{c_2})]$, where $d$ is our current path's distance from city 1 to city $c_1$, we've found a shorter path to $c_2$ with the best bike we have access to (again, including the bike at $c_2$).

At this point, you may notice that the algorithm I've described is very similar to Dijkstra's algorithm (not a coincidence!). The final answer is $\min\{dist[n][i]\}\,\forall\, i \in [1, 1000]$.

The code for this is as follows:

```
void solve() {
  vector<vector<pair<ll, ll>>> adj;
  vector<ll> s;

  vector<vector<bool>> vis;
  vector<vector<ll>> dist;

  ll n, m;
  cin >> n >> m;
  adj.resize(n + 1);
  for (ll i = 0; i < m; ++i) {
    ll u, v, w;
    cin >> u >> v >> w;
    adj[u].emplace_back(v, w);
    adj[v].emplace_back(u, w);
  }
  s.resize(n + 1);
  for (ll i = 1; i <= n; ++i) {
    cin >> s[i];
  }

  // vis[city][bike]
  vis = vector<vector<bool>>(n + 1, vector<bool>(1001));
  dist = vector<vector<ll>>(n + 1, vector<ll>(1001, inf));
  dist[1][s[1]] = 0;
```

```
  priority_queue<pair<ll, pair<ll, ll>>> pq;
  pq.push({0, {1, s[1]}});
  while (!pq.empty()) {
    auto p = pq.top();
    pq.pop();
    ll ct = p.second.first, sf = p.second.second, d = -p.first;
    if (vis[ct][sf]) {
      continue;
    }
    vis[ct][sf] = true;
    for (auto &[nb_ct, w] : adj[ct]) {
      ll best_sf = min(sf, s[nb_ct]);
      if (dist[nb_ct][best_sf] > d + w * sf) {
        dist[nb_ct][best_sf] = d + w * sf;
        pq.push({-dist[nb_ct][best_sf], {nb_ct, best_sf}});
      }
    }
  }
  cout << *min_element(dist[n].begin(), dist[n].end()) << "\n";
}
```