Cities and Libraries: A Connected Component Problem

Nicholas Fleischhauer

December 5, 2024

Abstract

This paper analyzes an algorithmic problem involving the optimization of library and road construction in a network of cities. By refining the problem statement and exploring connected components within graphs, we develop a cost-minimizing strategy and implement an efficient solution. Custom visualizations and detailed explanations support the methodology and findings presented.

Introduction

In the realm of interview preparation, solving algorithmic problems is a common practice to assess and enhance one's problem-solving skills. This paper presents an analysis of a specific problem sourced from HackerRank *Roads and Libraries — HackerRank* 2024, focusing on optimizing the allocation of libraries and roads within a network of cities.

- **Problem Refinement:** We begin by presenting a refined version of the original problem, addressing and correcting various errors and informational inconsistencies that often accompany such "toy" problems.
- **Thought Process and Optimization:** Following the problem statement, we detail the thought process involved in approaching and solving the problem optimally, highlighting key strategies and considerations.
- **Coded Solution:** Finally, we provide a comprehensive coded solution, illustrating the implementation of the discussed methodologies.

To facilitate a clear understanding of the concepts discussed, custom visualizations of graphs and connected components were generated using the Manim mathematical animation framework Developers 2024.

Problem Statement

As a city planner, your task is to provide the citizens of HackerLand with access to libraries while minimizing the overall infrastructure costs.

Task

Determine the minimum cost to provide library access to all citizens of HackerLand.

Initial Setup

There are n cities numbered from 1 to n. Currently there are no libraries and the cities are not connected. Bidirectional roads may be built between any city pair listed in *city_edges*.

A citizen has access to a library if:

- Their city contains a library or ...
- They can travel by road from their city to another city containing a library.

For example, if a citizen of city 1 doesn't have a library, but city 2 has a library and both cities are connected by a road, then all citizens of city 1 have access to a library and all citizens of city 2 have acces to a library.

Example Problem

The following figure is a sample map of HackerLand, where the dotted lines denote possible roads:

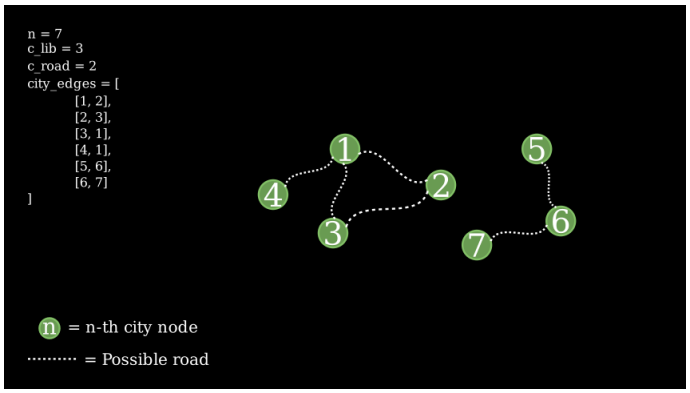

Figure 1: Map of HackerLand illustrating possible road connections

The inputs for this example are as follows:

- $n = 7$, the number of cities
- $c_{\text{lib}} = 3$, the cost of a single library
- $c_{\text{road}} = 2$, the cost of a single road
- *city_edges* $= [[1, 2], [2, 3], [3, 1], [4, 1], [5, 6], [6, 7]]$, a list of *pontential* connections between cities i.e, where a road can be built but hasn't been built yet. (Assume we can't build a road between node 2 and 7 for example)

Example Solution

The minimal cost solution for this example is to build 5 roads and 2 libraries. So the total cost of this is

$$\text{Total Cost} = (5) \times 2 + (2) \times 3 = 16$$

note that *one of the possible roads in the cycle* $1 \to 2 \to 3$ *is unnecessary.*

Coding Setup

Function Description

Complete the function **roads_and_libraries**

```
from typing import List

```

```python
def roads_and_libraries(n: int, c_lib: int, c_road: int, city_edges: List[List[int]]) -> int:
    """
    Determines the minimum cost to provide library access to all citizens of HackerLand.

    Args:
        n (int): The number of cities.
        c_lib (int): The cost to build a single library.
        c_road (int): The cost to build a single road.
        city_edges (List[List[int]]): A list of edges representing possible roads between cities.
            Each sublist contains two integers indicating a bidirectional road
            between the specified cities.

    Returns:
        int: The minimal total cost to ensure all citizens have access to a library.

    Example:
        >>> n = 7
        >>> c_lib = 3
        >>> c_road = 2
        >>> city_edges = [[1, 2], [2, 3], [3, 1], [4, 1], [5, 6], [6, 7]]
        >>> roads_and_libraries(n, c_lib, c_road, city_edges)
        16
    """
    ...
```

Command Line Input Format

The following is the command line input format for testing purposes.

```
q
n m c_lib c_road
u_i v_i
```

where:

- q is the number of queries
- m is the number of potential roads contained within $city_{edges}$
- c_{lib} is the cost to build a library
- c_{road} is the cost to build a road
- u_i and v_i are the i-th pair of cities that can be connected by a potential road

Input Constraints

- $1 \leq q \leq 10$
- $1 \leq n \leq 10^5$
- $0 \leq m \leq min\left(10^5, \frac{n \cdot (n-1)}{2}\right)$

- $1 \leq c_{lib}, c_{\text{road}} \leq 10^5$
- $1 \leq u_i, v_i \leq n$
- each potential road connects two distinct cities

Command Line Input Format Example

An example is provided with two queries. The first query matches the example problem shown in Figure 1, representing the layout of HackerLand, and the second query pertains to another land, Scriptshire, as shown in Figure 2.

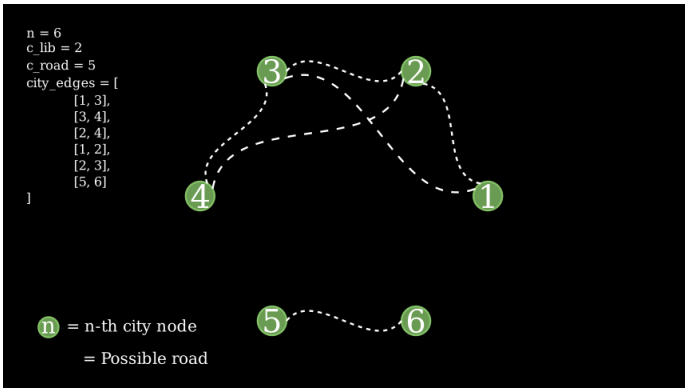

Figure 2: Scriptshire

Command Line Input Format Example

```
STDIN       Function
-----       --------
2           q = 2
7 6 3 2     n = 7, city_edges[] size m = 6, c_lib = 3, c_road = 2
1 2         city_edges = [[1, 2], [2, 3], [3, 1], [4, 1], [5, 6], [6, 7]]
2 3
3 1
4 1
5 6
6 7
6 6 2 5     n = 6, city_edges[] size m = 6, c_lib = 2, c_road = 5
1 3         city_edges = [[1, 3], [3, 4],...]
3 4
2 4
1 2
2 3
5 6
```

Sample Output

The following is the sample output for the inputs used above.

```
16
12
```

Explanation

The cheapest way to make libraries accessible to all **for HackerLand** is to:

- Build a single library within any of the cities in the first component at a cost of 3.
- Build roads between each city within the first component such that each city can reach one another either directly, or via another city by road.
- The same holds for the second component.

This results in a total cost of $(3 \times 2 + 1 \times 3) + (2 \times 2 + 1 \times 3) = 16$, where the first component requires:

- 3 roads at a cost of 2 each
- 1 library at the cost of 3 each

and the second component requires:

- 2 roads at a cost of 2 each
- 1 library at a cost of 3 each

The most economical way to provide library access to all in **Scriptshire** is to build a library in each city because the cost to build a library (2) is less than the cost to build a road (5). Scriptshire contains 6 cities, so the minimal total cost is $6 \times 2 = 12$.

Diving Into the Problem

Our objective is to ensure that all citizens have access to a library while minimizing the overall cost. To achieve this, we need to define an appropriate cost function. Before formulating the cost function, we present some preliminary observations.

Observations and Thoughts

1. A well-known mathematical formula related to triangular numbers Wikipedia 2024a appears in the constraint bounds:
$$0 \leq m \leq \min\left(10^5, \frac{n(n-1)}{2}\right)$$
2. Connecting each city to all other cities would result in $\frac{n(n-1)}{2}$ roads, which is often more than necessary. For example, in HackerLand (Figure 1), seven nodes would require 21 roads for full connectivity, but our optimal solution uses only five roads.
3. The cost function is inherently tied to the concept of connected components.
4. In the case of HackerLand, there are two connected components. Each component requires at least one library.
5. Therefore, the lower bound on the number of libraries is one per connected component.
6. For each connected component, there are two possible strategies:
    - Build one library and construct roads to connect all cities within the component.
    - Build libraries in all cities within the component.

Clarification on Connected Components

A connected component Wikipedia 2024b is a subset of a larger structure (e.g., HackerLand) where every pair of elements within the subset is connected by some defined relationship. In our case, these are the dashed lines indicating where a potential road can be built.

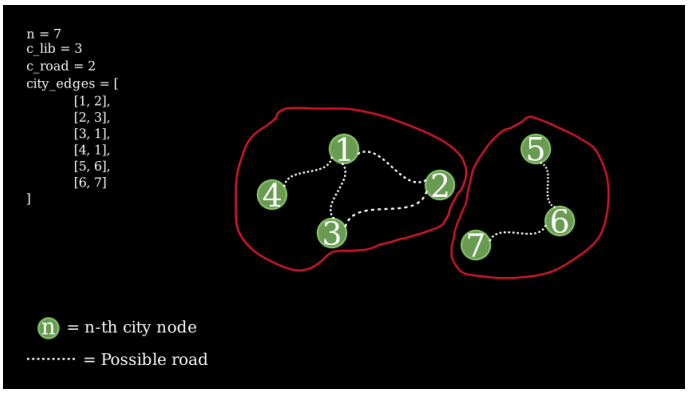

Figure 3: two connected components

Defining the Cost Function

Next, we explore how to define our cost function, starting at the component level.

Game Plan

1. Define a mathematical function that represents cost at the **component level**.
2. Identify edge cases.
3. Determine when minimal cost is achieved, at the **component level**, in terms of options that account for the edge cases and pickrove that minimal **component cost** is achieved.
4. Define a mathematical function that represents the **total cost** in terms of components.
5. Show that **total cost** is minimized as a consequence of all **component costs** being minimized.

Component Cost Function

Within each connected component, we can treat the number of cities k as a fixed constant since it remains unchanged. Our focus is on how many libraries s are needed within it.

Let:

- k be the number of cities.
- s be the number of libraries.
- r be the number of roads.

where $1 \leq s \leq k$ and $0 \leq r \leq k - 1$.

Our options can then be broken down into the following table:

Libraries (s)	**Roads (r)**
1	$k-1$
2	$k-2$
$\vdots$	$\vdots$
$k-1$	1
k	0

Table 1: Relationship between the Number of Libraries (s) and Roads (r) in a Connected Component

The table reveals a pattern that shows a linear relationship between the number of libraries and roads:

$$s + r = k$$

For each additional library built, one fewer road is needed to connect the remaining cities.

We can now define the cost function as:

$$C(s) = s \times c_{\text{lib}} + r \times c_{\text{road}}$$

Since $r = k - s$, we can express the cost function entirely in terms of s and k. Thus, for each connected component with k cities, the cost is:

$$C(s) = s \times c_{\text{lib}} + (k - s) \times c_{\text{road}}$$

Edge Cases

- $c_{\text{lib}} \leq c_{\text{road}}$: Building libraries in all cities is always cheaper or equal in cost to building roads. The code should naturally handle this case because the cost of libraries in all cities will be less than or equal to a single library with all roads.
- When c_{road} is very high: The cost of building roads might make the cost of one library with roads higher than the cost of libraries in all cities. The code should correctly choose to build libraries in all cities in this scenario.

Minimal Cost at the Component Level

To ensure minimal cost, we need to consider the options for each connected component and choose the cheaper one.

Theorem 1. *For a connected component with k cities, the minimal cost is given by:*

$$\min(C(s)) = \min\left(k \times c_{lib},\ c_{lib} + (k-1) \times c_{road}\right)$$

where $C(s)$ is the total cost, c_{lib} is the cost of building a library, and c_{road} is the cost of building a road.

Proof. We analyze the cost function:

$$C(s) = s \times c_{\text{lib}} + (k - s) \times c_{\text{road}}$$

where s is the number of libraries, $r = k - s$ is the number of roads, and $1 \leq s \leq k$.

Case 1: $c_{\text{lib}} > c_{\text{road}}$

- $c_{\text{lib}} - c_{\text{road}} > 0$ - The cost function $C(s)$ increases as s increases. - **Minimum Total Cost**: Achieved when $s = 1$ (one library and $k - 1$ roads).

Case 2: $c_{\text{lib}} < c_{\text{road}}$

- $c_{\text{lib}} - c_{\text{road}} < 0$ - The cost function $C(s)$ decreases as s increases. - **Minimum Total Cost**: Achieved when $s = k$ (libraries in all cities and no roads).

Case 3: $c_{\text{lib}} = c_{\text{road}}$

- $c_{\text{lib}} - c_{\text{road}} = 0$ - The cost function $C(s)$ is constant with respect to s. - Any number of libraries between 1 and k yields the same total cost.

In each case, intermediate values of s do not provide a lower cost than the endpoints $s = 1$ or $s = k$. Therefore, the minimal cost is achieved by choosing the option that results in the lower cost between building one library with roads or building libraries in all cities. □

Based on this analysis, our optimal strategy simplifies to:

- **If** $c_{\text{lib}} \geq c_{\text{road}}$: Build one library and construct roads to connect all cities within the component.
- **If** $c_{\text{lib}} \leq c_{\text{road}}$: Build a library in each city within the component.

Total Cost Function

The total cost of the infrastructure project can be represented as the sum of the cost of each component.

$$\text{Total Cost} = \sum_{\text{all components}} C(s)$$

Since we've minimized the costs on a per component basis, the total cost is also minimized.

Algorithm Solution Overview

The algorithm employs a Breadth-First Search (BFS) to identify connected components within the graph representing the cities and potential roads. For each connected component, we calculate the minimal cost based on the previously defined cost function, choosing between building libraries in all cities or building one library and connecting the remaining cities with roads.

Implementation Code

We have all the pieces solved, now its just a matter of implementing them in code.

Game Plan

1. Create a method for identifying connected components.
2. For each component, apply the minimal cost function.
3. Aggregate all the minimum costs to get the total minimum cost.

To achieve this, we build a graph from the adjacency list (city_edges) and perform a BFS to visit all cities, marking those already visited to avoid cycles.

```python
from typing import List
from collections import defaultdict, deque

def roads_and_libraries(n: int, c_lib: int, c_road: int, city_edges: List[List[int]]) -> int:
    """
    Determines the minimum cost to provide library access to all citizens of HackerLand.

    Args:
        n (int): The number of cities.
        c_lib (int): The cost to build a single library.
        c_road (int): The cost to build a single road.
        city_edges (List[List[int]]): A list of edges representing possible roads between cities.
            Each sublist contains two integers indicating a bidirectional road
            between the specified cities.

    Returns:
        int: The minimal total cost to ensure all citizens have access to a library.

    Example:
        >>> n = 7
        >>> c_lib = 3
        >>> c_road = 2
        >>> city_edges = [[1, 2], [2, 3], [3, 1], [4, 1], [5, 6], [6, 7]]
        >>> roads_and_libraries(n, c_lib, c_road, city_edges)
        16
    """

    # build the graph
    graph = defaultdict(list)
    for u, v in city_edges:
        graph[u].append(v)
        graph[v].append(u)

    visited = [False] * (n + 1)
    total_cost = 0

    for city in range(1, n + 1):
        if not visited[city]:
            # Start BFS from this city
            queue = deque()
            queue.append(city)
            visited[city] = True
            num_cities_in_component = 1

            while queue:
                current_city = queue.popleft()
                for neighbor in graph[current_city]:
                    if not visited[neighbor]:
                        visited[neighbor] = True
                        queue.append(neighbor)
                        num_cities_in_component += 1
```

```
52
53        # Calculate the cost for this connected component
54        cost_libs_in_all_cities = num_cities_in_component * c_lib
55        cost_one_lib_with_roads = c_lib + (num_cities_in_component - 1) * c_road
56        min_component_cost = min(cost_libs_in_all_cities, cost_one_lib_with_roads)
57        total_cost += min_component_cost
58
59    return total_cost
```

Conclusion

In this paper, we successfully formulated and solved the problem of minimizing infrastructure costs for providing library access in a network of cities. By leveraging graph theory concepts and cost analysis, we demonstrated that the optimal strategy depends on the comparative costs of libraries and roads. The implemented algorithm efficiently identifies connected components and calculates the minimal total cost. Future work may explore more complex scenarios, such as dynamic costs or additional constraints.

License

References

Developers, Manim Community (2024). *Manim: Mathematical Animation Framework*. Version v0.14.0. Accessed: 2024-12-03. URL: https://www.manim.community/.

Roads and Libraries — HackerRank (2024). Accessed: 2024-12-03. HackerRank. URL: https://www.hackerrank.com/challenges/torque-and-development/problem (visited on 12/03/2024).

Wikipedia (2024a). *Triangular number*. Accessed: 2024-12-03. Wikipedia. URL: https://en.wikipedia.org/wiki/Triangular_number (visited on 12/03/2024).

— (2024b). *Component (graph theory)*. Accessed: 2024-12-03. Wikipedia. URL: https://en.wikipedia.org/wiki/Component_(graph_theory) (visited on 12/03/2024).