COMPSCI 330: Design and Analysis of Algorithms

Lecture 13: Shortest Path, Minimum Spanning Tree

Lecturer: Rong Ge *Scribe: Mo Zhou*

1 Overview

In this lecture, we will first learn how to compute the shortest path when there are negative edge weights, and then start to talk about Minimum Spanning Trees (MST).

2 Shortest Path with Negative Edge Length

We have learned Dijkstra algorithm, which can compute the shortest path from a source to all other nodes in the graph when the edge weights are non-negative. However, when there are negative edge weights, Dijkstra algorithm will fail. To develop an algorithm that can deal with negative edge lengths, let us first look at a motivating example.

2.1 Motivating Example: Arbitrage

In our example, arbitrage is defined as making profits only through currency exchange. Specifically, let $(u_1, u_2, \cdots, u_k)$ be the order of currency exchange, i.e., you have 1 unit of u_1 initially, then change it to u_2, then u_3, $\cdots$, until you change it to u_k. Let $c(u_i, u_j) > 0$ be the exchange rate, i.e., you can use $c(u_i, u_j)$ unit of u_i can be used to get 1 unit of u_j. Thus, if you change the money according to this path, the final amount of u_k you get will be

$$\frac{1}{c(u_1, u_2)} \cdot \frac{1}{c(u_2, u_3)} \cdots \frac{1}{c(u_{k-1}, u_k)}.$$

If we define $w(u, v) = \log c(u, v)$, then

$$\log \frac{1}{c(u_1, u_2)} \cdot \frac{1}{c(u_2, u_3)} \cdots \frac{1}{c(u_{k-1}, u_k)} = \sum_{i=1}^{k} w(u_i, u_{i+1}).$$

Note that $\sum_{i=1}^{k} w(u_i, u_{i+1})$ is the length of the path $(u_1, u_2, \cdots, u_k)$. Therefore, the shortest path from u_1 to u_k is actually the best way to exchange currency if you want to use u_1 to get u_k. Furthermore, when $u_1 = u_k$, arbitrage is possible if and only if the cycle length is negative. The negative cycles are closely related to shortest paths, and we will see their relationship in the next section.

2.2 Relationship between Shortest Path and Negative Cycles

Definition 1. A cycle $(u_1, u_2, \cdots, u_k)$ is a negative cycle if $\sum_{i=1}^{k-1} w(u_i, u_{i+1}) + w(u_k, u_1) < 0$.

Claim 1. If s can reach a negative cycle $(u_1, u_2, \cdots, u_k)$, then the shortest path from s to any u_i is not defined.

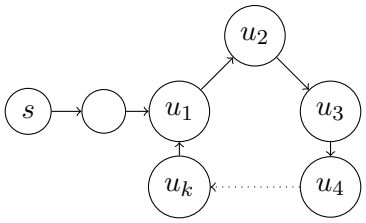
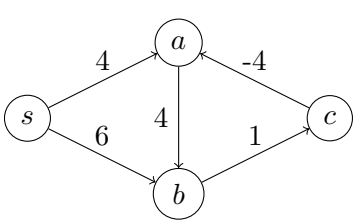


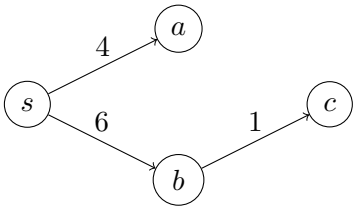
Figure 1: Example Negative Cycle and Shortest Path

Proof. See Figure 1 for an example. First we start from s and reach u_i, and assume the path has length l. Then we follow the negative cycle many times. Assume that the cycle length is $c < 0$, then after t times of following the cycle, the total length from s to u_i is $l + ct$. As t grow to infinity, the length of the path will tends to minus infinity, or equivalently, we say the shortest path from s to any u_i is not defined. □

2.3 Shortest Path with Negative Edge But No Negative Cycle



(a) Example Graph



(b) Shortest Path Tree by Running Dijkstra' Algorithm on the Left Graph

Figure 2: Shortest Path with Negative Edge But No Negative Cycle

When there are negative edges but no negative cycle in a graph, Dijkstra's algorithm might fail. For instance, consider the graph in Figure 2a, running Dijkstra;s algorithm on this graph will get a "shortest path tree" shown in Figure 2b. This tree gives us a "shortest path" from s to a with length 4. However, we know that there is a path $s \to b \to c \to a$, which has length $6 + 1 - 4 = 3$. This indicates that Dijkstra's algorithm fails to return the correct shortest path on this graph.

2.4 Bellman-Ford Algorithm

2.4.1 Algorithm Description

Bellman-Ford Algorithm is a Dynamic Programming algorithm that can deal with the negative edge lengths when computing the shortest paths in a graph. The motivation of this algorithm is: The sub-paths of a shortest path is also a shortest path, otherwise we can replace that sub-path with the actual shortest path and violates the assumption that the original path is the shortest path.

State Definition: Define $d[v, i]$ to be the length of shortest path from s to v using at most i edges.
Transition Function: $d[v, i + 1] = \min\{d[v, i], \min_{(u,v)\in E} w[u, v] + d[u, i]\}$.

Pseudocode:

```
Initialize d[s, 0] = 0, d[u, 0] = +∞ for all other vertices u;
for i=1 to n do
    Initialize d[u, i] = d[u, i − 1];
    for all edges (u, v) do
        if w[u, v] + d[u, i − 1] < d[v, i] then
            d[v,i] = w[u,v]+d[u,i-1];
        end
    end
end
if there is a vertex u such that d[u, n] ≠ d[u, n − 1] then
    There is a negative cycle!
end
```

Algorithm 1: Bellman-Ford Algorithm

2.4.2 Example Run

We run Bellman-Ford Algorithm on the graph in Figure 2a. Then, the dynamic programing table $d(u, i)$ will be the following:

$d(u,i)$	s	a	b	c
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	4	6	$+\infty$
2	0	4	6	7
3	0	3	6	7
4	0	3	6	7

2.4.3 Proof of Correctness

The correctness of Bellman-Ford algorithm directly comes from the following theorem.

Theorem 2. If the graph does not have any negative cycle, then $d[u, n-1]$ will be the shortest path distance from s to u. Also, $d[u, n] = d[u, n-1]$ for any vertex u.

On the other hand, if there is a negative cycle reachable from s, there exists vertex u such that $d[u, n] < d[u, n-1]$.

Proof. The prove can be divided into three parts.
(1) Prove that $d[v, i]$ is indeed the length of shortest path from s to u using at most i edges. Since Bellman-Ford algorithm is a Dynamic Programming algorithm, this part of the proof is exactly the same as the correctness proof of a Dynamic Programming algorithm. Thus, it's omitted here for simplicity.
(2)

Claim 3. If there is no negative cycles in the graph, then the shortest path from s to u has length at most $n-1$.

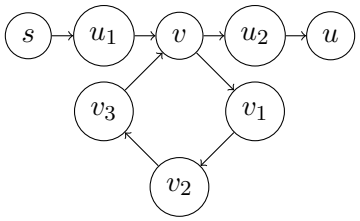


Figure 3: Example Shortest Path

Proof of Claim. Assume toward contradiction that the shortest path from s to u has length at least n. Then the number of nodes on the path is at least $n+1$. By Pigeonhole principle, at least one vertex v was visited at least twice. Assume the shortest path is $p_1 = (s, u_1, \cdots, u_t, v, v_1, \cdots, v_k, v, u_{t+1}, \cdots, u_\ell, u)$, see an example in Figure 3. Compare the length of the path p_1 and $p_2 = (s, u_1, \cdots, u_t, v, u_{t+1}, \cdots, u_\ell, u)$, which are two valid paths from s to u. The difference of the length of p_1 and p_2 is the length of cycle $(v, v_1, \cdots, v_k)$. Since p_1 is the shortest path, p_2 must has a larger of equal length than p_1. If p_2 is longer than p_1, that means cycle $(v, v_1, \cdots, v_k)$ is a negative cycle, which is a contradiction. If the length of those two paths are equal, then we can without loss of generality delete that cycle. In other words, our claim is correct. □

(3) If there is a negative cycle reachable from s, assume that circle is $(u_1, u_2, \cdots, u_k)$. You can still refer to Figure 1 as an example. Since it's a negative cycle, we know that

$$w(u_1, u_2) + w(u_2, u_3) + \cdots + w(u_k, u_1) < 0.$$

Thus,

$$\begin{aligned} d[u_{i+1}, n] &\leq d[u_i, n-1] + w[u_i, u_{i+1}], \forall i \in \{1, 2, \cdots, k-1\} \\ d[u_1, n] &\leq d[u_k, n-1] + w[u_k, u_1]. \end{aligned}$$

Take the sum and we get

$$\sum_{i=1}^{k} d[u_i, n] \leq \sum_{i=1}^{k} d[u_i, n-1] + \textbf{length of cycle} < \sum_{i=1}^{k} d[u_i, n-1],$$

which indicates that there at least exist one i such that $d[u_i, n] < d[u_i, n-1]$. □

2.4.4 Running Time Analysis

The algorithm runs $O(n)$ iterations of update to the values, and each iteration costs $O(m)$ time because it updates m times and each update takes constant time. Thus, the total running time for this algorithm is $O(mn)$.

3 Minimum Spanning Tree

Minimum Spanning Tree (MST) is a spanning tree with the minimum total weight. In this section, we will first learn the definition of a spanning tree and then study some properties for Minimum Spanning Tree, which will be useful in proving the correctness of MST algorithms.

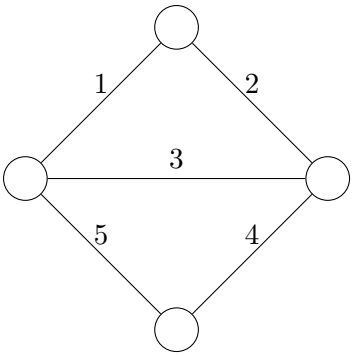


Figure 4: Example Graph

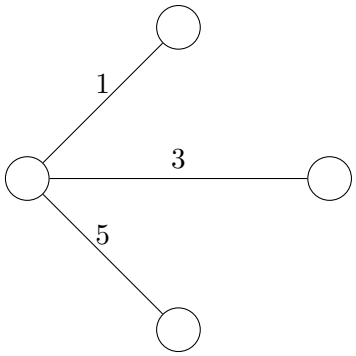


Figure 5: Example Spanning Tree 1

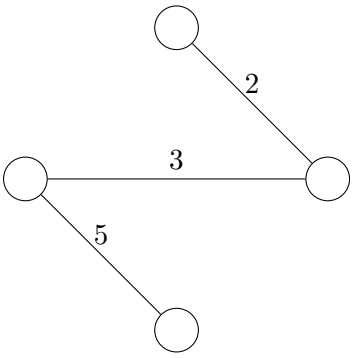


Figure 6: Example Spanning Tree 2

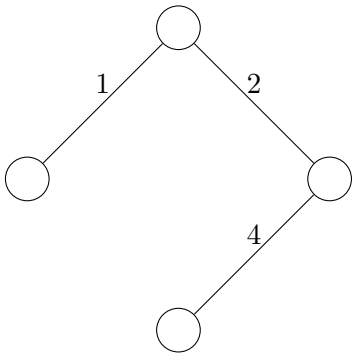


Figure 7: Example Spanning Tree 3

3.1 Spanning Tree

Definition 2. A spanning tree of a graph $G = (V, E)$ is a subset of $n - 1$ edges in E, such that all pairs of vertices are connected by these edges.

An example can be seen in Figure 4. The example spanning trees are Figure 5 through 7. There are usually many possible spanning trees for a graph, e.g., DFS trees and BFS trees. In this particular example, it's easy to see that the MST is Figure 7. That is because the MST must have 3 edges and the best you can hope for is when the edge weights are 1, 2, and 3. However, that is not a valid spanning tree because it is a cycle. Thus, the next smallest will be 1, 2, and 4, which is valid.

3.2 Properties for MST

Claim 4. The subtrees of MST are also MST. See Figure 8 for an example, the nodes with the same color forms a MST of the subgraph containing the same set of vertices.

Proof. The proof of this is similar to the proof of the sub-paths of shortest path are shortest paths. Assume by contradiction that one subtree of MST is not MST, then we replace that subtree with the true MST for that subgraph and get a smaller spanning tree for the original graph, which is a contradiction. □

However, that does not mean this problem can be solved by dynamic programming. The main difficulty is that we are not able to know how to partition the nodes into subgraphs, and the possible

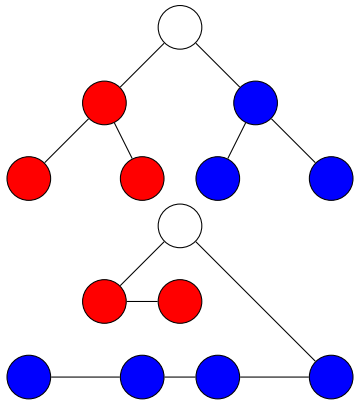


Figure 8: Subtrees of MST

number of partitions are exponential. Instead, the following property makes sure that this problem can be solved using Greedy algorithm.

3.3 Cuts in Graphs

Definition 3. A cut is a subset of edges that separates the vertices into two parts. It is specified with two set of vertices: a set of vertices S and the remaining vertices $\bar{S}$, and a cut is defined as

$$c(S, \bar{S}) = \{(u, v) \in E | u \in S, v \in \bar{S}\}.$$

If $(u, v) \in E$, $u \in S$, $v \in \bar{S}$, we say edge (u, v) "crosses" the cut $(S, \bar{S})$.

Claim 5. For any spanning tree T, any cut $(S, \bar{S})$, there must be at least one edge (u, v) in both T and $(S, \bar{S})$.

Proof. Assume by contradiction that the spanning tree does not contain any edge in the cut, then in that spanning tree, the nodes in S are disconnected with the nodes in $\bar{S}$, which contradicts the definition of a spanning tree. □

An important property of trees is: Adding a non-tree edge to a tree will form a cycle, and removing any edge from that newly-formed cycle will result in a new tree. This is the "swap" operation that we use to prove the correctness of a greedy algorithm. Thus, the MST can probably be constructed using a greedy algorithm. Then we have the following lemma:

Lemma 6. Suppose F is a set of edges that is inside some MST T, if $(S, \bar{S})$ is a cut that does not contain any edge in F and e is the minimum cost edge in the cut, then it is safe to add e to F, i.e., $F \cup \{e\} \subset T'$ for some MST T'.

We will not prove the lemma formally here, but the main intuition is: If a non-minimum edge was added to the tree and the minimum was not, then we can always "swap" them. That gives us the following greedy algorithm for MST:

repeat
  Find a cut that does not have any edge in the current tree;
  Add the min cost edge of the cut to the tree;
until *the tree has* $n - 1$ *edges*;

Algorithm 2: Greedy Algorithm for MST

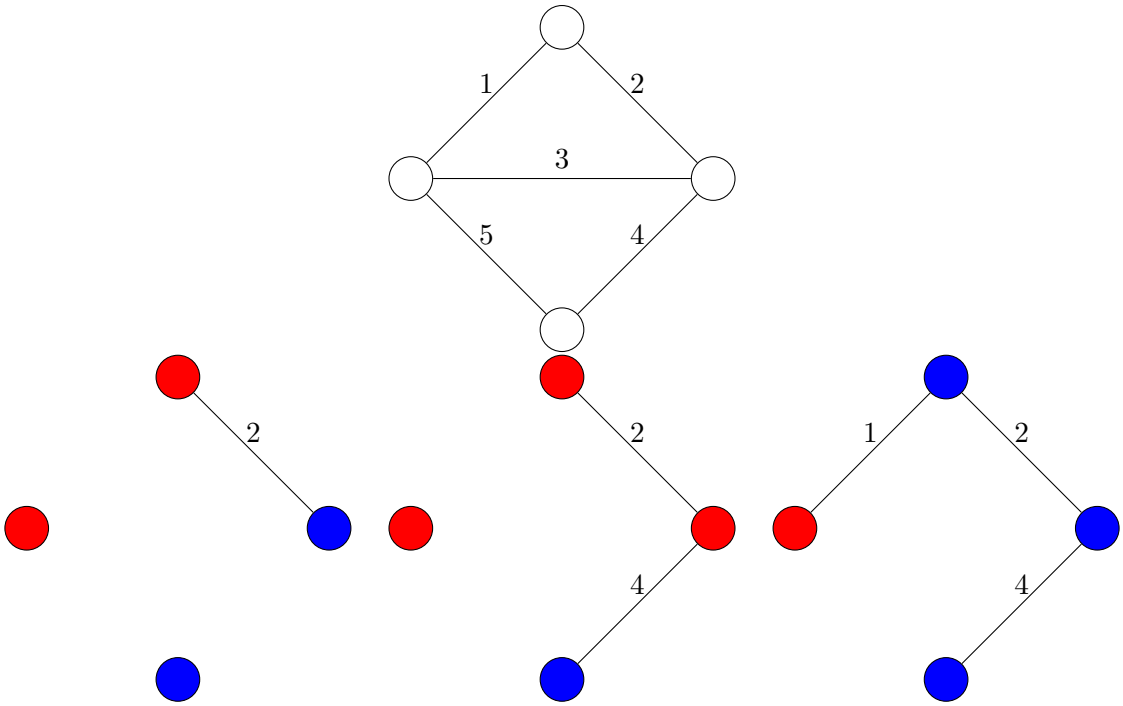


Figure 9: Example Run of Greedy Algorithm for MST

An example run is performed in Figure 9. The upper-left corner is the original graph. For the other three graphs, the red vertices are those in S and the blue vertices are in $\bar{S}$. The newly-added edge in those graphs are the minimum edges in the cut. Finally, we got the correct MST.