

## Lecture 10: October 10

Lecturer: Micah Adler

Scribe: Piyanuch Silapachote

## 10.1 Administrative

- First midterm will be on October 24 at 5:30pm - 8:30pm
- Second midterm will be on December 12 at 5:30pm - 8:30pm

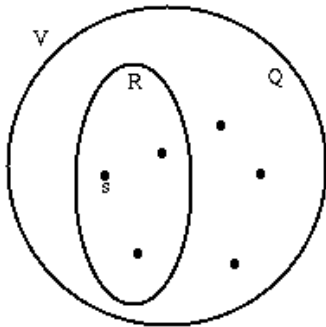
## 10.2 Single-Source Shortest Paths

**Input:** Directed graph  $G = (V, E)$ , weight function  $w : E \rightarrow \Re$ , a source vertex  $s$ .

**Output:**  $\forall v \in V, \delta(s, v)$ , where  $\delta(s, v)$  is the length of the shortest path from  $s$  to  $v$ .

### 10.2.1 Dijkstra's Algorithm

Dijkstra's algorithm solves a special case of the single-source shortest-paths problem in which edges can only have non-negative weights.



Dijkstra's algorithm maintains the following:

- $R$ : set of vertices that have been processed
- $Q = V - R$ : set of vertices that have not been processed
- $d[v]$ : the length of the shortest path from  $s$  to  $v$  using only vertices in  $R$
- invariance:  $\forall v \in R, \delta(s, v) = d[v]$

Figure 10.1: Dijkstra's algorithm

How a vertex is processed?

Find a vertex  $u \in Q$  such that  $\delta(s, u) = d[u]$ . It turns out that  $u = \{u | u \in Q \text{ and } \forall v \in Q : d[u] \leq d[v]\}$ .

Add  $u$  to a set  $R$  and delete  $u$  from a set  $Q$ .

Update the distance vector  $\forall v : d[v] = \min(d[v], d[u] + w(u, v))$ . A more efficient implementation updates distance vector only for vertices adjacent to  $u$  because only these vertices have  $w(u, v)$  not equal to infinity and so only these paths would want to go through  $u$ .



How these two lemma together proved the theorem?

After initialize, the only time  $d[v]$  can change is at an update step and it can only be decreasing.

At some point in the algorithm, a vertex  $v$  is placed in  $R$  and  $d[v] = \delta(s, v)$ , by the second lemma.

Since  $d[v]$  can only decrease and cannot takes the value below its minimum (by the first lemma), once it gets the minimum value  $\delta(s, v)$ , it will stays unchanged throughout the rest of the algorithm. So, at the end, the return value of  $d[v]$  will be  $d[v] = \delta(s, v)$ .

■

#### 10.2.4.1 Proof of lemma 10.2

**Proof:** By contradiction.

Let  $v$  be the first vertex in the running of the algorithm such that  $d[v] < \delta(s, v)$ .

The only step that could have changed  $d[v]$  to this value is at an update step,  $d[v] = d[u] + w(u, v)$ .

Since we assume  $d[v] < \delta(s, v)$ , then  $d[u]$  must also be less than  $\delta(s, u)$ .

But the only way that  $d[u]$  could have gotten that value was in the earlier iteration of an algorithm, which contradicts the assumption that  $v$  was the first such vertex.

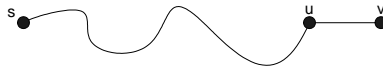
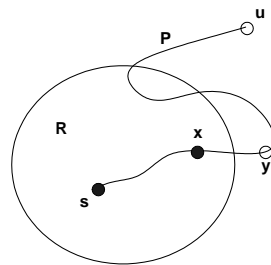


Figure 10.2: Depiction of vertices  $u$  and  $v$  for proof of lemma 10.2

■

#### 10.2.4.2 Proof of lemma 10.3

**Proof:** We will prove, by contradiction, that  $d_u[u] = \delta(s, u)$  where  $d_u[v]$  denote  $d[v]$  when  $u$  is chosen as the minimum. Take a snap shot during the algorithm as shown in figure 10.3 below.



P: Shortest path from source  $s$  to  $u$   
 $y$ : first vertex on  $P$  that is not in  $R$   
 $x$ : predecessor of  $y$  on  $P$

Figure 10.3: Depiction of vertices and path for proof of lemma 10.3

Let  $u$  be the first vertex placed in  $R$  such that  $d_u[u] \neq \delta(s, u)$ .

**Claim 10.4**  $d_u[y] = \delta(s, y)$ .

Since we assume  $u$  to be the first vertex getting the wrong value, other vertices placed in  $R$  before  $u$ , including a vertex  $x$ , must have satisfied the invariant. Hence, we have  $d_x[x] = \delta(s, x)$ .

After  $x$  is placed in  $R$  and after all the corresponding updates are done,  $d[y] \leq d_x[x] + w(x, y)$ .

Since  $P$  is the shortest path passing through both  $x$  and  $y$ , it must be the case that  $d[y] = \delta(s, y)$ .

Since  $y$  is preceding  $u$  and there are no negative weight,  $\delta(s, y) \leq \delta(s, u)$ .

Thus,  $d_u[y] = \delta(s, y) \leq \delta(s, u) \leq d_u[u]$ . But  $d_u[u] \leq d_u[y]$  since we picked  $u$ , not  $y$ .

So  $d_u[u] = d_u[y]$  and  $d_u[u] = \delta(s, u)$ .

This contradicts the assumption that  $d_u[u] \neq \delta(s, u)$ . ■

## 10.3 All Pairs Shortest Paths

**Input:** A directed graph  $G = (V, E)$ , a weight function  $w : E \rightarrow \mathfrak{R}$

**Output:**  $\forall (u, v) \in E, \delta(u, v)$ , where  $\delta(u, v)$  is the length of the shortest path from  $u$  to  $v$ .

From the previous lecture, we know that we can use *dynamic programming* to solve the all-pairs shortest-paths problem and the running time is  $O(|V|^3)$ . Now we are going to give a faster algorithm, but it only works on a special case: an undirected and unweighted graph. This algorithm is called **Seidels Algorithm**[S92], and it uses *matrix multiplication* to compute the all pairs shortest paths.

### 10.3.1 A problem that uses matrix multiplication

**Input:** A graph  $G = (V, E)$ .

**Output:** A graph  $G_2$  which is a graph  $G$  augmented by edges  $(i, j)$  for every pair of vertices  $i$  and  $j$  such that  $G$  has a path of length 2 from  $i$  to  $j$ .

**Example:**

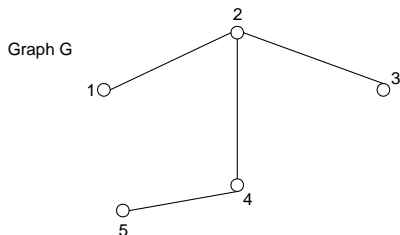


Figure 10.4: A graph  $G$

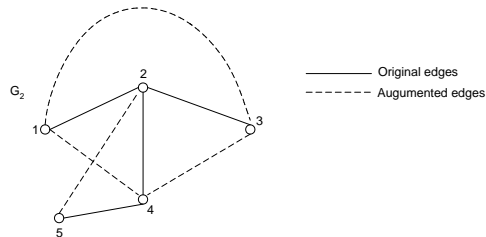


Figure 10.5: A graph  $G_2$

The naive algorithm is to look at all possible pairs of vertices  $(i, j)$  and for every pair, look for an intermediate vertex  $k$ . Since there are  $O(|V|^2)$  possible pairs and  $O(|V|)$  vertices to check for each pair, the total running time of this algorithm is  $O(|V|^3)$ .

It turns out that this running time can be improved by applying matrix multiplication technique.

Define  $M[G]$  as the adjacency matrix of graph  $G$ . An example for the above graph looks like:

$$M[G] = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Consider what we need to do for a vertex-pair  $(i, j)$ :

- Check: For all intermediate vertices  $k$  if both  $(i, k)$  and  $(k, j)$  are equal to 1. This means that there is an edge between  $i$  and  $k$  and an edge between  $k$  and  $j$ .
- Count: The total number of  $k$  that satisfied the check.

This is actually equivalent to matrix multiplication for a resulting entry  $(i, j)$ :

- $M[G]^2 = M[G] \cdot M[G]$ .
- $M[G]^2(i, j) = \sum_{k=1}^{|v|} M[G](i, k) \cdot M[G](k, j)$ .
- Check: The product  $M[G](i, k) \cdot M[G](k, j)$  is equal to 1 only when both components are 1.
- Count: The summation is equivalent to the count.

Thus,  $M[G]^2(i, j) =$  number of paths of length exactly 2 from  $i$  to  $j$ .

An algorithm for computing  $M[G_2]$  given  $M[G]$ :

1. Compute  $M[G]^2 = M[G] \cdot M[G]$
2. Set  $M[G_2](i, j) = \begin{cases} 1 & \text{if } i \neq j \text{ and } (M[G](i, j) = 1 \text{ or } M[G]^2(i, j) > 0) \\ 0 & \text{otherwise} \end{cases}$

The running time for computing  $G_2$  is as fast as the fastest matrix multiplication algorithm.

## References

- CLR90 T. Cormen and C. Leiserson and R. Rivest, *Introduction to Algorithms*, The MIT Press, 1990, pp. 527–531.
- SN00 T. Rath and C. Dansong, *Scribe note for CMPSCI611 Fall 2000 Lecture 8*.
- SN00 J. Lan and H. Chim, *Scribe note for CMPSCI611 Fall 2000 Lecture 9*.