CMPSCI 611 Advanced Algorithms **Fall 2000**

Lecture 9: October 10

Lecturer: Micah Adler *Scribe: Jiang Lan and Haumei Chim*

9.1 Single Source Shortest Paths

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

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

Next we'll present Dijkstra's Algorithm that solves the single-source shortest-paths problem on a weighted, directed graph for the case in which all edge weights are nonnegative.

9.1.1 Dijkstra's Algorithm

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

Output: $\forall v \in V$, $\delta(s, v)$.

In this algorithm, we need to maintain two things:

R**:** set of vertices with known minimum path length

$d[v]$**:** minimum path length to vertex v using only R

1. $d[s] = 0$
2. $\forall v \neq s, d[v] = w(s, v)$
3. $R = \{s\}, Q = V - \{s\}$
4. **while** $|Q| \geq 1$
5. $\quad u = \min(d[v])$, where $v \in Q$ get the smallest path to a vertex not in R (hence, in Q)
6. $\quad Q = Q - \{u\}$
7. $\quad R = R \cup \{u\}$
8. $\quad$ **for each** $v \in Adj[u]$ ($Adj[u]$ denotes the set of adjacent nodes of u)
9. $\quad\quad d[v] = \min(d[v], d[u] + w(u, v))$
10. **return** d

9.1.2 Running time of Dijkstra's algorithm

- Simple implementation, i.e., use a binary heap at steps 5 and 9. Running time = $O(|E| log|V|)$.
- Complicated implementation, i.e., use Fibonacci heap. Running time = $O(|E| + |V| log|V|)$.

Note that Dijkstra's algorithm assumes nonnegative edge weights. Another algorithm, the Bellman-Ford algorithm, solves the single-source shortest-paths problem in the more general case in which weights can be negative.

9.1.3 Proof of Dijkstra's algorithm

Prove the correctness of Dijkstra's algorithm is equivalent to proving the following theorem.

Theorem 9.1 *If* $\forall e \in E, w(e) \geq 0$ *then at the end of Dijkstra's algorithm,* $\forall v \in V, d[v] = \delta(s, v)$.

Proof: Note that $d[v]$ can only decrease during the algorithm (at Step 9). The proof of this theorem consists of two parts, the first part is a claim and the second part is a lemma. We are going to prove each part seperately.

Claim 9.2 $\forall v \in V, d[v] \geq \delta(s, v)$.

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 step 9, $d[v] = d[u] + w(u, v)$.

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 algorithm, which contradicts the assumption that v was the first such vertex, (c.f. Figure 9.1.).

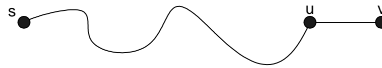

Figure 9.1: Depiction of nodes u, v

■

Lemma 9.3 $\forall u \in V$, *when* u *is placed in* R, $d[u] = \delta(s, u)$.

Proof:

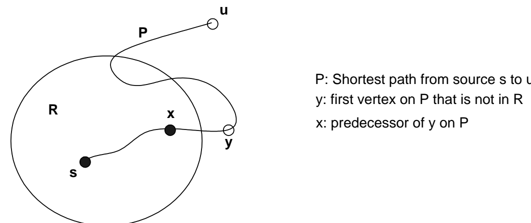

Figure 9.2: Proof of Lemma 9.3

Let u be the first vertex placed in R such that $d[u] \neq \delta(s, u)$ (see Fig 9.2). When we choose a u we have some v in R such that $w(u, v)$ is minimal. p is the shortest path from s to v. y is the final vertex on p not6 in R when u is chosen; x is the immediate predecessor on p of y. Note that y may be u, and that x may be s, but x is not y.

Let $d_{v_1}[v_2]$ denote $d[v_2]$ when v_1 is chosen as the minimum. We need to prove $d_u[u] = \delta(s,u)$.

In order to prove the lemma, we have another claim:

Claim 9.4 $d_u[y] = \delta(s,y)$.

Since we assume u to be the first vertex to get the wrong value and x is before u in p (see Figure 9.2) $d_x[x] = \delta(s,x)$.

After x is picked for R, $d[y] \le d[x] + w(x,y)$ (from completion of step 9).

If p is the shortest path then the $d(s,y) = d(s,x) + w(x,y)$. But we just said it must be less than or equal to this. Therefore $d_u[y] = \delta(s,y)$.

Thus $d_u[y] = \delta(s,y)$ and $\delta(s,y) \le \delta(s,u)$ (since we chose u before y), and $\delta(s,u) \le d_u[u]$.

But $d_u[u] \le d_u[y]$ (from step 5)

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

■

Combine the proof of the claim and the lemma, the proof of Dijkstra's algorithm is complete. ■

9.2 All Pairs Shortest Paths

Input: A Directed Graph $G = (V,E)$, where V is a set of vertices and E is a set of edges.

A weight function $w : E \to \Re$

Output: $\forall (u,v) \in E, \delta(u,v)$, the total weight 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.

Let's look at a problem that can use matrix multiplication.

Given a graph $G = (V,E)$, we want to compute G_2 which is 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:

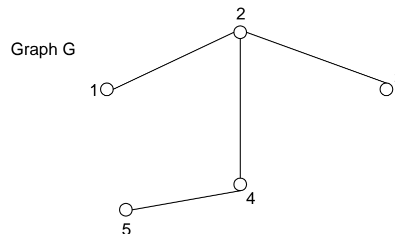

Figure 9.3: A graph G

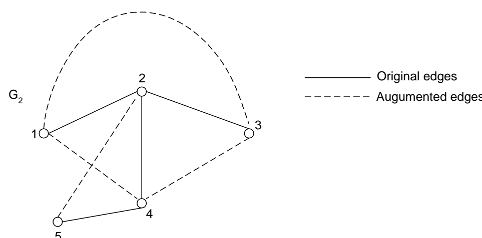

Figure 9.4: A graph G_2

The obvious algorithm to compute G_2 is to look at each node k and determine whether (i,k) and (k,j) both exist. The run time for this algorithm is $O(|V|^3)$. It turns out that this obvious algorithm is almost like a matrix multiplication, so we can actually use matrix multiplication to compute G_2.

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

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

However, we cannot compute G_2 directly, we need to define an intermediate matrix $M[G]^2 = M[G] \cdot M[G]$.

$M[G]^2(i,j) = \sum_{k=1}^{|v|} M[G](i,k) \cdot M[G](k,j)$

The product $M[G](i,k) \cdot M[G](k,j)$ is only a 1 when each component is a 1. That means there is both an edge between i and k and an edge between k and j.

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

Thus to compute $M[G_2]$, we need:

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. Let's use $\mu(n)$ denotes the time required for $n \times n$ matrix multiplication. So, the run time for computing G_2 is $\mu(|V|)$ and the run time for filling in the enries of G_2 is $O(|V|^2)$.

In order to compute the all pairs shortest paths, we need another intermediate matrix called $P[G]$, the parity matrix, which is a binary matrix where

$$P[G](i,j) = \begin{cases} 1 & \text{if shortest path from } i \text{ to } j \text{ has odd length} \\ 0 & \text{otherwise} \end{cases}$$

Algorithm for the All Pairs Shortest Paths (APSP)

Now we are ready to define a recursive algorithm to compute APSP. We begin with the graph G, compute the all pairs shortest paths of G_2 and then use it to compute the all pairs shortest path of G.

The following algorithm assumes our given graph is connected and we have an adjacency matrix for it.

$APSP(M[G])$

1. **Compute** $M[G_2]$
2. **If** $\forall i \neq j, M[G_2](i,j) = 1$
3. **then return** $D[G](i,j) = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } M[G](i,j) = 1 \\ 2 & \text{otherwise} \end{cases}$ the all pairs shortest paths matrix.
4. **else** $D'[G] = APSP(M[G_2])$
5. **compute** $P[G]$
6. **return** $D[G] = 2D'[G] - P[G]$

The Proof of correctness for the All Pairs Shortest Paths will appear in Lecture 10.

References

S92 R. Seidel, On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs, *Journal of Computer and System Sciences* **51**, 1992, pp. 400–403.