Carlos Mc Gregor c.mcgregormuro@mail.utoronto.ca

CSC373S: Algorithm Design, Analysis & Complexity

Lecture 15

Wednesday February 8, 2017
based on notes by Denis Pankratov

Graph Algorithms (*continued*)

Single-source Shortest Path

Input: $G = (V, E)$ (directed or undirected)
$c : E \to \mathbb{R}$ so that there are no negative weight cycles
$S \in V$ - source

Output: $dist[]$ so that $\forall u \in V, dist[u] = d[s, u]$

Special Case:

$$c > 0$$

Dijkstra solves this case in $\mathcal{O}((|V| + |E|) \log |V|)$ using *binary heap* If $\exists e \in E, c(e) < 0 \to$ Dijkstra fails

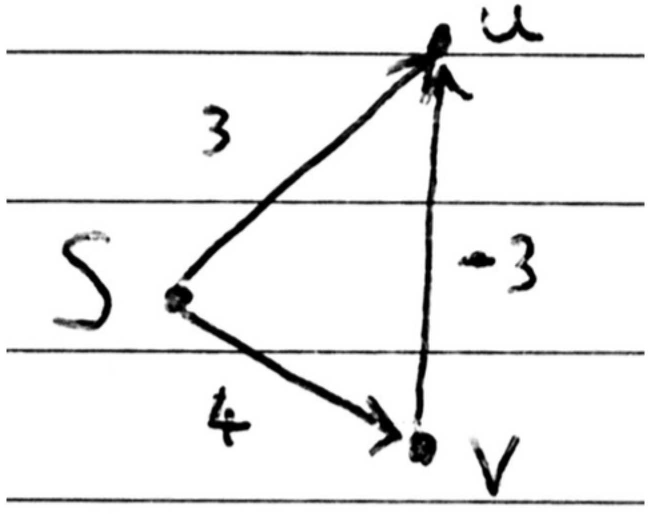

Figure 1: $dist[u] = 3 \neq d(s, v) = 4 - 3 = 1$

Key operation in Dijkstra:

```
Update(e = (u, v))
  if dist[v] > dist[u] + c(u, v)
    dist[v] = dist[u] + c(u, v)
    prev[v] = u
```

Let us assume that the following invariant holds $dist[u] \geq d(s,u)$.

1. This operation is safe, *i.e.* we can perform this operation as many times as we want w/o violatin the invariant. Moreover, if $dist[u] = d(s,u)$, then $dist[]$ won't be updated.

2. If $s = v_0, v_1, \ldots, v_k = u$ is a shortest path from s to u then the sequence of updates $update(v_{i-1}, v_i)$ for $i \in \{1, .., k\}$ results in $dist[u] = d(s,u)$.

Problem: we don't know intermediate vertices.
We know that shortest path can be taken to be simple $\Rightarrow k \leq |V| - 1$
$\Rightarrow$ update <u>all</u> edges $|V|$ times
This *algorithm* is <u>Bellman-Ford</u>.

```
def Bellman-Ford(G,c,s):
  init arrays dist[], prev[], both of size |V|

  for v in V:
    dist[v] = float('inf')
    prev[v] = None

  dist[s] = 0

  for k=1 to |V|:
    for e in E:
      update(e)

  return dist[]
```

<u>Runtime:</u> $\mathcal{O}(1)$ per update, number of updates is $|V||E| \Rightarrow$ overall $\mathcal{O}(|V||E|)$

Can be modified to detect *negative weight cycles* reachable from s.

1. Run *Bellman-Ford* as described

2. Run one more iteration of the outer *for loop* (k)

<u>Claim:</u> Any $dist[]$ value changes in this last iteration $\Leftrightarrow \exists$ *negative weight cycle* reachable from s

Proof: $\Rightarrow$ Clear, follows from correctness.
$\Leftarrow$ Assume that $v_0, v_1, \ldots, v_k = v_0$ is a *negative weight cycle*

Assume from contradiction that none $dist[]$ have changed
$\Rightarrow dist[v_i] \leq dist[v_{i-1}] + c(v_{i-1}, v_1)$ (update for (v_{i-1}, v_i) failed)
$\forall i \in \{1, .., k\}$

A) $\sum_{i=1}^{k} dist[v_i] \leq$ B) $\sum_{i=1}^{k} dist[v_{i-1}] +$ Cost of the cycle) $\sum_{i=1}^{k} c(v_{i-1}, v_i)$

$A = B$ because $v_k = v_0$
Cancel A with B gets cost of cycle ≥ 0

Exercise: Restate *Bellman-Ford* as a *dynamic programming algorithm*. Semantic array, computational array, etc.

All-pairs shortest paths

Input: $G = (V, E)$ (directed or undirected)
$C : E \to \mathbb{R}$ assume no *negative weight cycles*

Output: $dist[][]$ so that $\forall u, v \in V, dist[u][v] = d(u, v)$

Approaches

1. Run *Bellman-Ford* $|V|$ times, runtime $\mathcal{O}(|V|^2|E|)$
2. Use ideas similar to matrix multiplication + add repeated squaring $\Rightarrow \mathcal{O}(|V|^3 \log |V|)$ (*see CLRS*)

Floyd-Warshall Dynamic Programming Algorithm

Let $V = \{v_1, v_2, \ldots, v_n\}(n = |V|)$

Semantic Array
$C[u, v, k] =$ length of shortest path from u to v using only nodes from $\{v_1, \ldots, v_k\}$ as intermediate nodes (*note:* $V \to u, V \to v, [n] \to k$).

Computational Array
Base case:

$$k = 0, C[u, v, 0] = \begin{cases} c(u, v), & \text{if } (u, v) \in E \\ \infty, & \text{otherwise} \end{cases}$$

$C[u,v,k] = \min\{C[u,v,k-1], C[u,v_k,k-1] + C[v_k,v,k-1]\}$

Equivalence: Shortest path from u to v using intermediate nodes $\{v_1,\dots,v_k\}$ uses v_k & then u to v_k and v_k to v are shortest paths using only intermediate nodes $\{v_1,\dots,v_{k-1}\}$ doesn't use v_k

Algorithm:

```
def Floyd-Warshall(G,C):
  # (|V| + 1) is 0-based
  init C - array of size |V| * |V| * (|V| + 1)

  for u in V:
    for v in V:
      if (u, v) in E:
        C[u, v, 0] = c(u, v)
      else:
        C[u, v, 0] = float('inf')

  for k = 1 to |V|:
    for u in V:
      for v in V:
      C[u, v, k] = min(C[u, v, k-1], C[u,v_k, k-1]
                                  + C[v_k, v, k -1])
  # return slice of array
  return C[.,.,|V|]
```

Runtime: $\mathcal{O}(|V|^3)$
Memory Use: $\Theta(|V|^3)$

Excercise: modify the algoritm to use $\mathcal{O}(|V|^2)$ memory