طراحی الگوریتم

۱۰ آذر ۱۳۹۸

ملکی مجد

Topic	Reference
Recursion and Backtracking	Ch.1 and Ch.2 JeffE
Dynamic Programming	Ch.3 JeffE and Ch.15 CLRS
Greedy Algorithms	Ch.4 JeffE and Ch.16 CLRS
Amortized Analysis	Ch.17 CLRS
Elementary Graph algorithms	Ch.6 JeffE and Ch.22 CLRS
Minimum Spanning Trees	Ch.7 JeffE and Ch.23 CLRS
Single-Source Shortest Paths	Ch.8 JeffE and Ch.24 CLRS
All-Pairs Shortest Paths	Ch.9 JeffE and Ch.25 CLRS
Maximum Flow	Ch.10 JeffE and Ch.26 CLRS
String Matching	Ch.32 CLRS
NP-Completeness	Ch.12 JeffE and Ch.34 CLRS

All-Pairs Shortest Paths

the problem of finding shortest paths between all pairs of vertices in a graph.

Johnson's algorithm for sparse graphs

$$O\left(V^2 \lg V + VE\right)$$

uses the technique of ***reweighting***

Set new weight

- The new set of edge weights must satisfy two important properties.

1. For all pairs of vertices $u, v \in V$, a path p is a **shortest path** from u to v using old weight function **if and only** if p is also a **shortest path** from u to v using the new weight function.
2. For all edges, the new weight is **nonnegative**

Reweighting does not change shortest paths

Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbf{R}$, let $h : V \to \mathbf{R}$ be any function mapping vertices to real numbers. For each edge $(u, v) \in E$, define

$$\widehat{w}(u, v) = w(u, v) + h(u) - h(v) .$$

Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be any path from vertex v_0 to vertex v_k. Then p is a shortest path from v_0 to v_k with weight function w if and only if it is a shortest path with weight function $\widehat{w}$. That is, $w(p) = \delta(v_0, v_k)$ if and only if $\widehat{w}(p) = \widehat{\delta}(v_0, v_k)$. Also, G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function $\widehat{w}$.

Producing nonnegative weights by reweighting
first make a new graph

- Make a new graph
- Add a new vertex s
- Connect s to all other vertices with weight zero

- Note that because *s* has no edges that enter it, **no shortest paths in *new graph***, other than those with source *s*, contain *s*.
- Moreover, *new graph* has **no negative-weight cycles if and only if** *the initial graph* has no negative-weight cycles.

New weight function
second calculate hand then $\hat{\boldsymbol{w}}$

- Define $\boldsymbol{h}(\boldsymbol{v}) = \boldsymbol{\delta}(\boldsymbol{s}, \boldsymbol{v})$ for all $v \in V$.

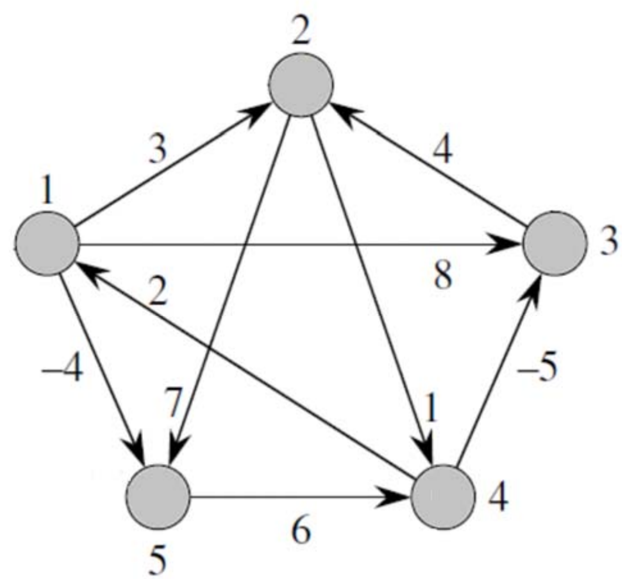
By the triangle inequality we have $h(v) \leq h(u) + w(u, v)$ for all edges $(u, v) \in E'$.

$$\hat{\boldsymbol{w}}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{w}(\boldsymbol{u}, \boldsymbol{v}) + \boldsymbol{h}(\boldsymbol{u}) - \boldsymbol{h}(\boldsymbol{v}) \geq 0,$$

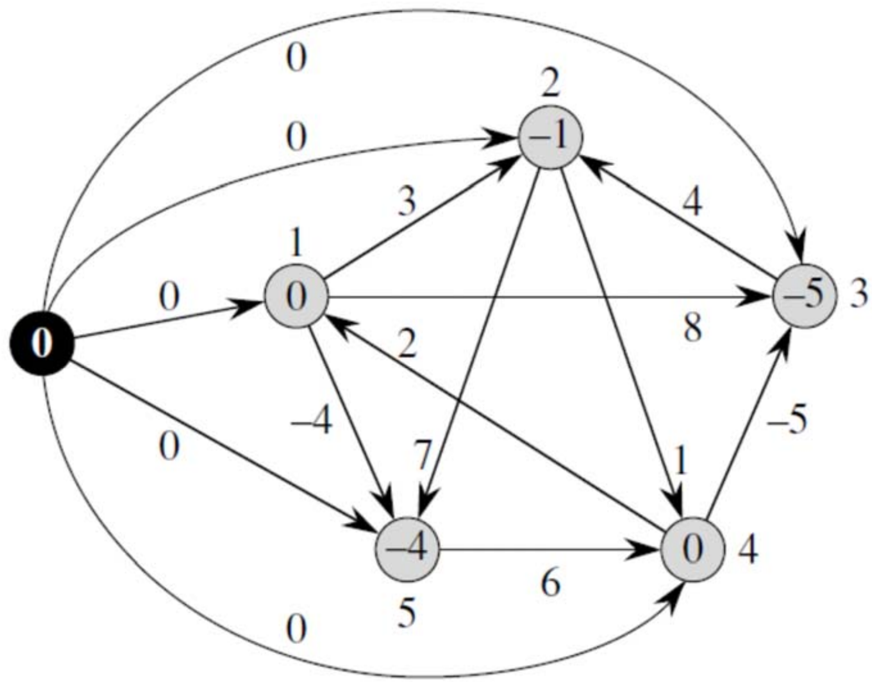
What is the relationship between the weight functions *w* and $\hat{w}$

if w(u, v) ≥ 0 for all edges *(u, v)* ∈ *E*?

A sample graph
-initially

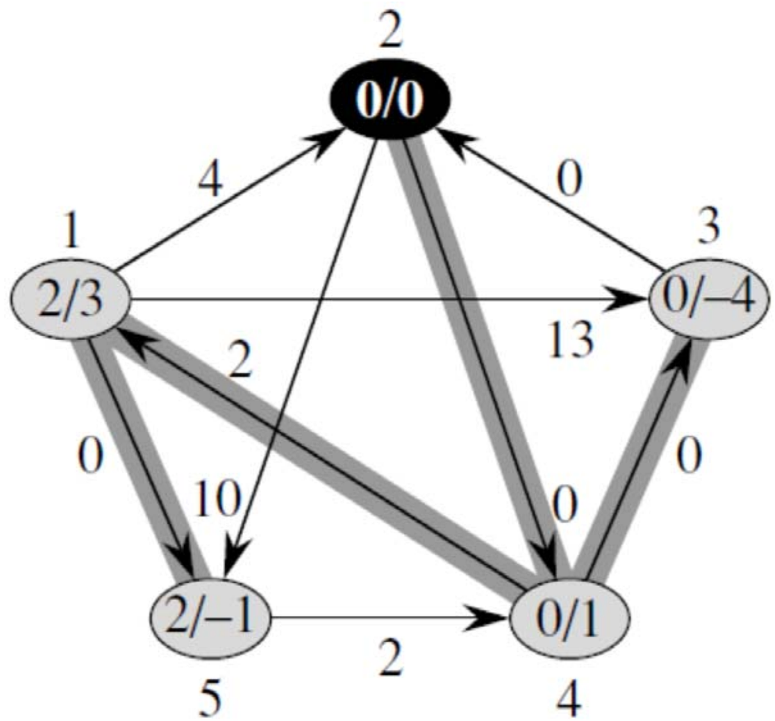


A sample graph
-add a new vertex and compute shortest path from it



A sample graph

-shortest path from a vertex by considering new weights



Your friend claims that there is a simpler way to reweight edges than the method used in Johnson's algorithm.

Letting $w^{*} = \min(u, v) \in E\ \{w(u, v)\}$,

Define $\hat{\boldsymbol{w}}(u, v) = w(u, v) - w^{*}$ for all edges *(u, v)* ∈ *E*.

What is wrong with the proposed method of reweighting?

Computing all-pairs shortest paths

$\text{JOHNSON}(G)$

compute G', where $V[G'] = V[G] \cup \{s\}$,
  $E[G'] = E[G] \cup \{(s, v) : v \in V[G]\}$, and
  $w(s, v) = 0$ for all $v \in V[G]$
if $\text{BELLMAN-FORD}(G', w, s) = \text{FALSE}$
  then print "the input graph contains a negative-weight cycle"
  else **for** each vertex $v \in V[G']$
      do set $h(v)$ to the value of $\delta(s, v)$
        computed by the Bellman-Ford algorithm
    for each edge $(u, v) \in E[G']$
      do $\widehat{w}(u, v) \leftarrow w(u, v) + h(u) - h(v)$
    for each vertex $u \in V[G]$
      do run $\text{DIJKSTRA}(G, \widehat{w}, u)$ to compute $\widehat{\delta}(u, v)$ for all $v \in V[G]$
        for each vertex $v \in V[G]$
          do $d_{uv} \leftarrow \widehat{\delta}(u, v) + h(v) - h(u)$
    return D

- If the min-priority queue in Dijkstra's algorithm is implemented by a **Fibonacci** heap,

  the running time of Johnson's algorithm is $\boldsymbol{O(V^2\ lg\,V\ +\ V\ E)}$.

The simpler binary min-heap implementation yields a running time of $\boldsymbol{O(V\ E\ lg\,V)}$, which is still asymptotically faster than the Floyd-Warshall algorithm if the graph is sparse.

Example

Show the values of h and $\widehat{w}$ computed by the algorithm