algorithms and data structures (shortest path)

amir yehudayoff

shortest paths

finding shortest paths is an important task

- maps
- make decisions
- explore domain

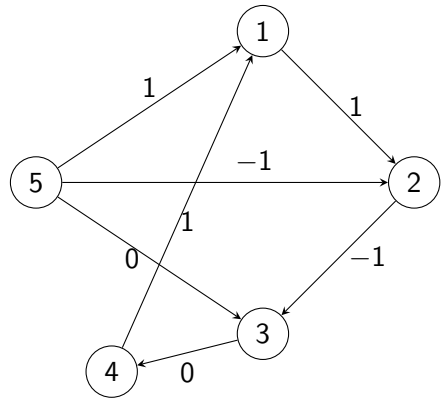
efficient algorithms for distances in graph (Dijkstra)

what if not a distance?

weighted graphs

let $G = (V, E)$ be a directed graph
let $w : E \to \mathbb{R}$ be a weight function

remark: possibly negative

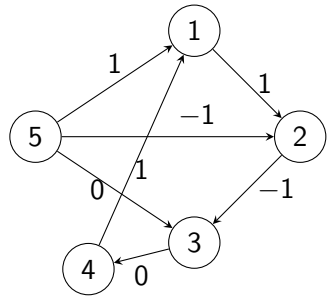


path weights

the weight of a path $\gamma = (v_0, v_1, \ldots, v_m)$ is

$$w(\gamma) = \sum_{i=1}^{m} w(v_i, v_{i-1})$$

example: weight of $5, 2, 3, 4$?



"distances"

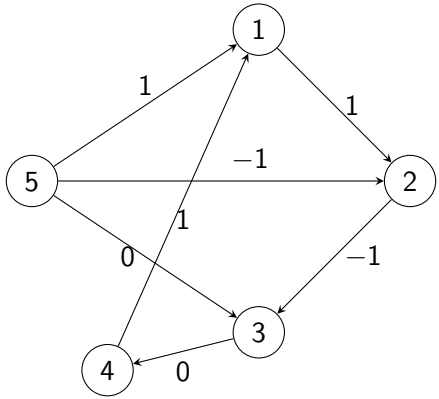
the weight of a path $\gamma = (v_0, v_1, \ldots, v_m)$ is $w(\gamma) = \sum_{i=1}^{m} w(v_i, v_{i-1})$

the "distance"

$$d(u, v) = \min\{w(\gamma) : u \xrightarrow{\gamma} v\}$$

remarks:
not a distance!
could be $-\infty$

$d(5,4) = ?$



input
digraph $G = (V, E)$
weights $w : E \to \mathbb{R}$
source $s \in V$

output
$v \mapsto d(s, v)$ or "negative cycle"

Bellman-Ford

for $v \in V$ set $d_0(v) = \infty$

for $k = 0, 1, \ldots, n-1$ set $d_k(s) = 0$

for $k = 0, 1, \ldots, n-2$
  for $(u, v) \in E$
    set $d_{k+1}(v) = \min\{d_k(v),\ d_k(u) + w(u, v)\}$

for $(u, v) \in E$
  if $d_{n-1}(v) > d_{n-1}(u) + w(u, v)$ return "negative cycles"

return d_{n-1}

initialization
for $v \in V$ set $d_0(v) = \infty$

know
for $k = 0, 1, \ldots, n-1$ set $d_k(s) = 0$

update
for $k = 0, 1, \ldots, n-2$
  for $(u, v) \in E$
    set $d_{k+1}(v) = \min\{d_k(v),\ d_k(u) + w(u, v)\}$

sanity check
for $(u, v) \in E$
  if $d_{n-1}(v) > d_{n-1}(u) + w(u, v)$ return "negative cycles"

return d_{n-1}

compute d_1, d_2

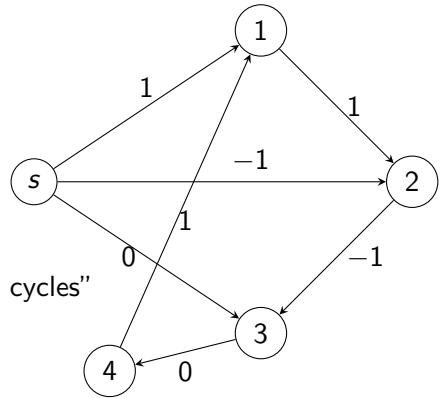
for $v \in V$ set $d_0(v) = \infty$

for $k = 0, 1, \ldots, n-1$ set $d_k(s) = 0$

for $k = 0, 1, \ldots, n-2$
  for $(u, v) \in E$
    set $d_{k+1}(v) = \min\{d_k(v),\ d_k(u) + w(u, v)\}$

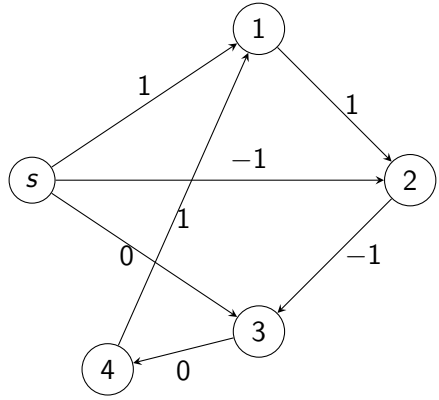
for $(u, v) \in E$
  if $d_{n-1}(v) > d_{n-1}(u) + w(u, v)$ return "negative cycles"

return d_{n-1}



compute d_1, d_2

	d_0	d_1	d_2
s	0	0	0
1	∞	1	1
2	∞	-1	-1
3	∞	1	-2
4	∞	∞	0



observation
the running time of Bellman-Ford is $O(|V| \cdot |E|)$

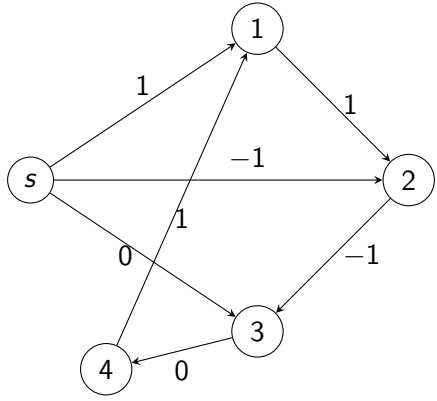
define
for integer k

$$\delta_k(v) \text{ is the minimum weight of a path } s \to v \text{ of length } \leq k$$

(if no path ∞)

remarks
finite "proxy" for d
not $-\infty$

$d(s,3) = -2 \quad \delta_0(3) = \infty \quad \delta_1(3) = 0 \quad \delta_2(3) = -2$



claim
if there are no negative cycles in G, w then for all v

$$\delta_{n-1}(v) = d(s, v)$$

proof
with no negative cycles minimum weight path is simple
length of simple path $\leq n - 1$

claim
for every $k \geq 0$ and for all v

$$d_k(v) = \delta_k(v)$$

proof?

claim $d_k(v) = \delta_k(v)$

proof
base: $k = 0$ is true
step: assume it is true for k ...

claim $d_k = \delta_k$ assuming $d_{k-1} = \delta_{k-1}$

proof

for vertex v there are two options

$\delta_k(v) = \infty$

$\delta_k(v) < \infty$

claim $d_k = \delta_k$ assuming $d_{k-1} = \delta_{k-1}$

proof for vertex v so that $\delta_k(v) = \infty$

$\delta_{k-1}(v) = \infty$

for all u so that $(u, v) \in E$ $\delta_{k-1}(u) = \infty$

algorithm is $d_k(v) = \min\{d_{k-1}(v),\ d_{k-1}(u) + w(u, v)\}$

so $d_k(v) = \infty$

claim $d_k = \delta_k$ assuming $d_{k-1} = \delta_{k-1}$

proof for vertex v so that $\delta_k(v) < \infty$

let γ be a minimum weight path $s \to v$ of length $\leq k$

let u be so that γ is $s \to \cdots \to u \to v$

claim $d_k = \delta_k$ assuming $d_{k-1} = \delta_{k-1}$

proof

γ is minimum weight $s \to v$ of length $\leq k$ and u before v in γ

by minimality and induction

$$\delta_k(v) = \delta_{k-1}(u) + w(u, v) = d_{k-1}(u) + w(u, v)$$

by algorithm

$$d_k(v) \leq d_{k-1}(u) + w(u, v)$$

so $d_k(v) \leq \delta_k(v)$

claim $d_k = \delta_k$ assuming $d_{k-1} = \delta_{k-1}$

proof
proved $d_k(v) \leq \delta_k(v)$ and other direction remains

by minimality for all r such that $(r, v) \in E$

$$\delta_k(v) \leq \delta_{k-1}(r) + w(r, v)$$

by induction

$$\delta_k(v) \leq d_{k-1}(r) + w(r, v)$$

$$\delta_k(v) \leq \delta_{k-1}(v) = d_{k-1}(v)$$

by algorithm $\delta_k(v) \leq d_k(v)$

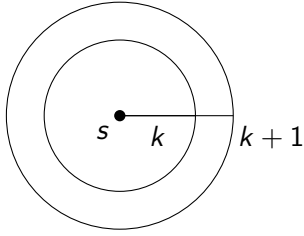
claim

for every $k \geq 0$ and for all v

$$d_k(v) = \delta_k(v)$$

idea

at time k we fully explored the "k-ball around s"



corollary
if there are no negative cycles in G, w then

$$d_{n-1}(v) = \delta_{n-1}(v) = d(s, v)$$

proof
two equalities from previous claims

claim

if the algorithm did not report "negative cycle" then there are no negative cycles (reachable from s)

claim
if the algorithm did not report "negative cycle" then there are no negative cycles (reachable from s)

proof let $v_0 \to v_1 \to \cdots \to v_m = v_0$ be a cycle reachable from s

by corollary and assumption $d_{n-1}(v_{i+1}) \leq d_{n-1}(v_i) + w(v_i, v_{i+1})$

summing

$$\sum_{i=0}^{m-1} d_{n-1}(v_i) = \sum_{i=0}^{m-1} d_{n-1}(v_{i+1}) \leq \left(\sum_{i=0}^{m-1} d_{n-1}(v_i) \right) + w(\text{cycle})$$

claim

if the algorithm reported “negative cycle” then there is a negative cycle

claim
if the algorithm reported "negative cycle" then there is a negative cycle

proof (contra positive) assume there are no negative cycles

so for all v

$$d_{n-1}(v) = \delta_{n-1}(v) = d(s, v)$$

the algorithm does not report "negative cycle" because for every $(u, v) \in E$

$$d(s, v) \leq d(s, u) + w(u, v)$$

theorem

if there are no negative cycles in G, w then output $v \mapsto d(s, v)$

if there are negative cycles reachable from s then output "negative cycle"

the running time is $O(|V| \cdot |E|)$

goal: compute $d(s, v)$ on a weighted graph G

process: iteratively produces better and better approximations

$$d_0, d_1, d_2, \ldots$$

where d_0 contains no info and d_{k+1} is built from d_k

after $n - 1$ iterations we reach the goal (or abort)

optimization

we have a collection of objets P

usually P is "huge"

each object has a weight $w : P \to \mathbb{R}$

wish to compute $\min\{w(p) : p \in P\}$ or find a minimizer

example: P is paths in digraph weighted by $w : E \to \mathbb{R}$

optimization

wish to compute $\min\{w(p) : p \in P\}$

when can we hope to do this efficiently?

what are the algorithmic principles?

optimization

wish to compute $\min\{w(p) : p \in P\}$

when can we hope to do this efficiently?

what are the algorithmic principles?

- greedy algorithm
- divide and conquer
- dynamic programming

wish to compute $\min\{w(p) : p \in P\}$

example: the cube

$$P = \{0,1\}^n$$

and

$$w(p_1, \ldots, p_n) = \sum_i w_i p_i$$

for some known $w_1, \ldots, w_n$

wish to compute $\min\{w(p) : p \in P\}$

example: intervals

$$P = \{[a, b] : 1 \leq a \leq b \leq n\}$$

and

$$w([a, b]) = \sum_{a \leq i \leq b} w_i$$

for some known $w_1, \ldots, w_n$

wish to compute $\min\{w(p) : p \in P\}$

example: distances

$$P = \{\text{paths in digraph}\}$$

and

$$w : E \to \mathbb{R}$$

wish to compute $\min\{w(p) : p \in P\}$

DP is used when

- objects in P can be broken into local pieces
- the total cost can be derived from the local costs
- a local optimal solution is used in many other solutions
- global optimality implies local optimality

(this is not a mathematical statement)

dynamic programming (DP)

wish to compute $\min\{w(p) : p \in P\}$

if P is paths in digraph weighted by $w : E \to \mathbb{R}$ then

- weight of a path is sum of its edges
- a subpath of an optimal path is an optimal path
- an optimal path can be used in many "larger paths"

summary

discrete optimization

greedy
divide and conquer
dynamic programming

all pairs

to compute all pairwise distances using Bellman-Ford the time is

$$O(n \cdot n^3) = O(n^4)$$

all pairs

to compute all pairwise distances using Bellman-Ford the time is

$$O(n \cdot n^3) = O(n^4)$$

we can do better...

think of weights as a matrix $W = (W_{u,v})$

define matrix product

$$(A \star B)_{u,v} = \min\{A_{u,r} + B_{r,v} : r \in V\}$$

[1] when there are no negative cycles

all pairs

think of weights as a matrix $W = (W_{u,v})$

define matrix product

$$(A \star B)_{u,v} = \min\{A_{u,r} + B_{r,v} : r \in V\}$$

claim $W^{\star n}$ contains all pairwise distances[1]

[1] when there are no negative cycles

think of weights as a matrix $W = (W_{u,v})$

define matrix product

$$(A \star B)_{u,v} = \min\{A_{u,r} + B_{r,v} : r \in V\}$$

claim $W^{\star n}$ contains all pairwise distances[1]

running time repeated squaring

$$M_1 = W \quad \text{and} \quad M_{k+1} = M_k \star M_k$$

takes time $O(n^3 \log n)$

[1] when there are no negative cycles

summary

efficient algorithm for shortest path for general weighted digraphs

dynamic programming

iteratively improving a solution to optimization problem

iterated matrix product