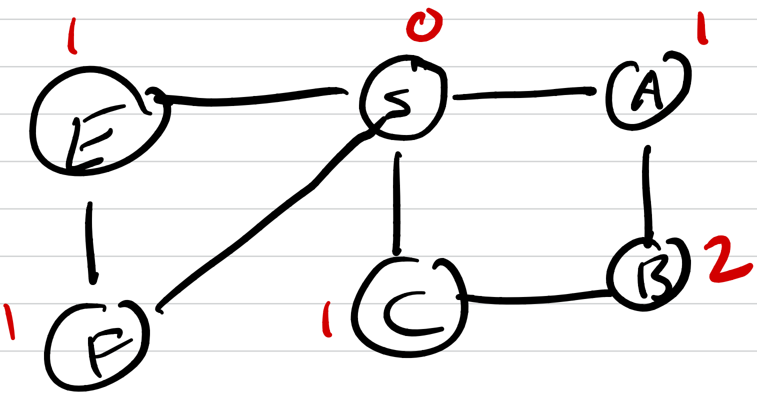
Paths in

Graphs

Single-source shortest paths (SSSP)

Input: Graph G, "source" vertex $s \in V$

Output: $\forall\, v \in V$, $d(s,v)$ = length of shortest path from s to v



Unweighted: all edges length 1

Breadth-first search

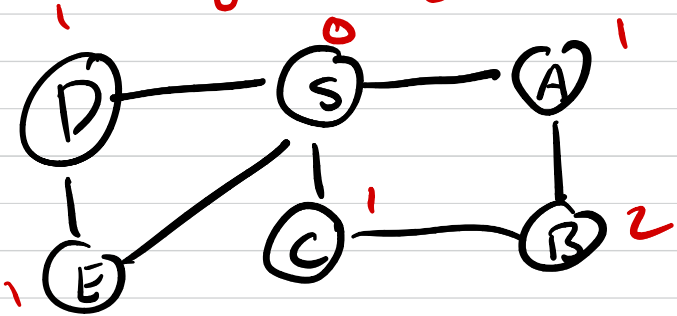
Positive lengths: $\ell : E \to \{1, 2, 3, \ldots\}$

Dijkstra

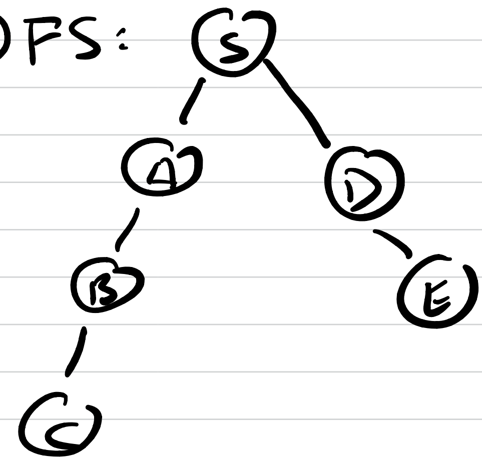
Arbitrary length edges

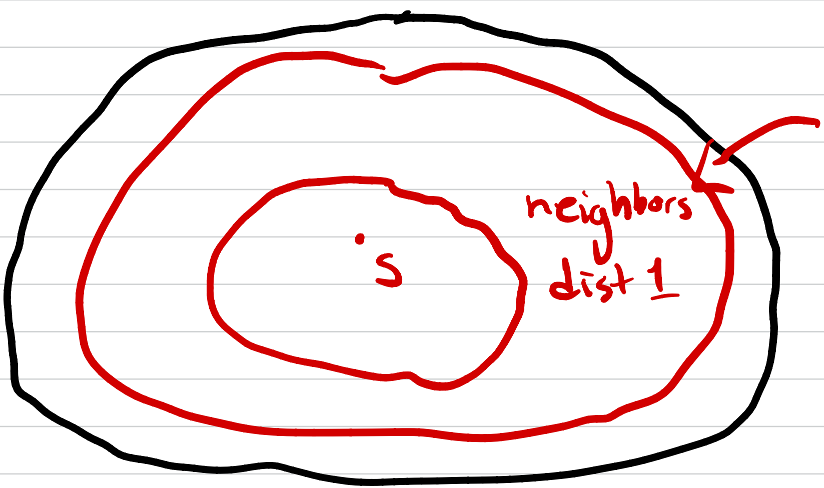
Bellman-Ford

Unweighted graphs



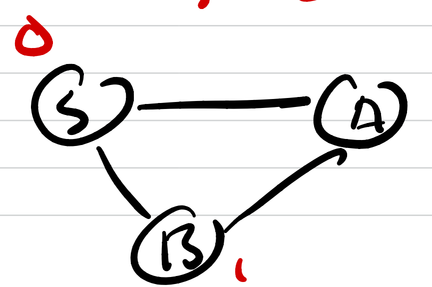
DFS:





G

neighbors of neighbors
(have not yet seen)
dist 2



Breadth-first search

bfs (G, s)

$dist[s] = 0$
$\forall u \neq s,\ dist[u] = \infty$

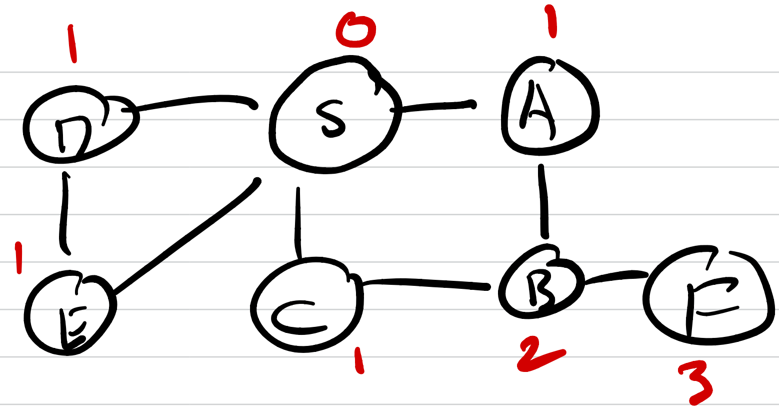
$Q = \{s\}$ (queue containing s)

while Q is not empty

  $u = dequeue(Q)$

  for all v s.t. $(u,v) \in E$
    if $dist[v] = \infty$
      $enqueue(Q, v)$

      $dist[v] = dist[u] + 1$

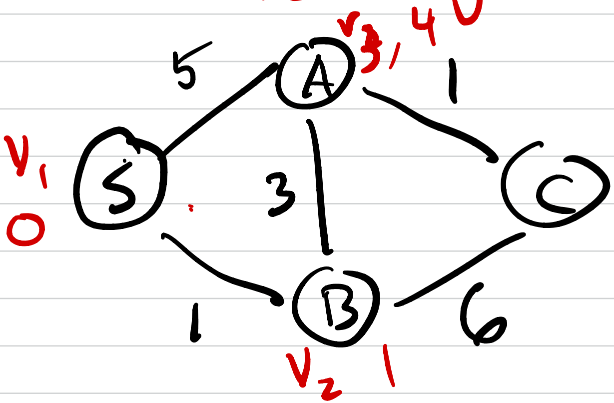


Q : $\cancel{S}\ \cancel{A}\ \cancel{C}\ \cancel{D}\ \cancel{E}\ \cancel{B}\ \cancel{F}$

Runtime: $O(n+m)$ time
linear, same as DFS

DFS is just BFS w/ stack

Positive lengths



Dijkstra's algorithm

1. Compute v_1 = closest vertex to s
   and $d(s, v_1)$

2. Compute $v_2 = 2^{nd}$ closest to s
   and $d(s, v_2)$

3. " v_3 3^{rd} " "

⋮

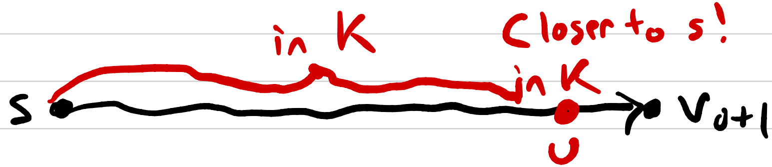
K = the set of "known" vertices
at some step

$= \{v_1, \dots, v_i\}$

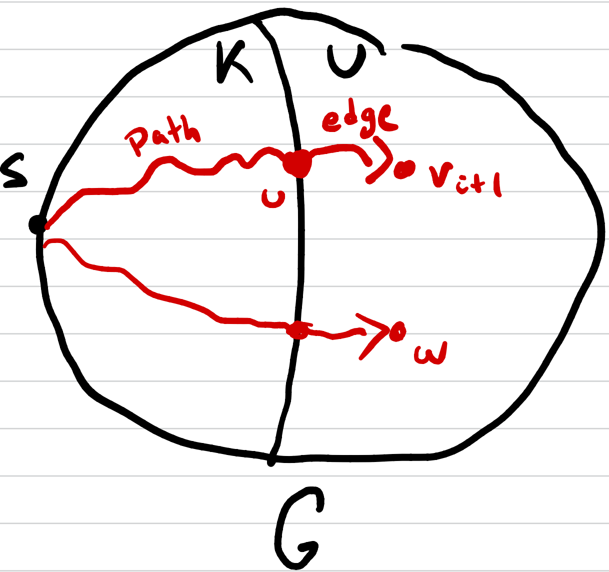
U = "unknown" vertices $\quad U = V / K$

Q: Given $K = \{v_1, \dots, v_i\}$.

How to compute v_{i+1}? = closest vertex not in K in U



($s \to v_{i+1}$ shortest path)



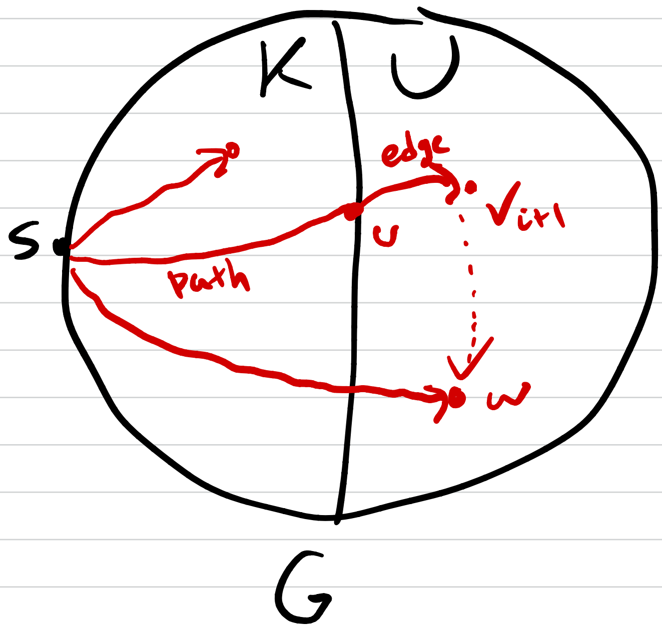
v_{i+1} = endpoint of the shortest path of this form

$$dist[v_{i+1}] = dist[u] + \ell(u, v_{i+1})$$

u minimizes $dist[u] + \ell(u, v_{i+1})$

Dijkstra has dist[v] for each $v \in V$

$$\text{dist}[v] = \begin{cases} d(s,v) & \text{if } v \in K \\ \min\limits_{u \in K} \text{dist}[u] + \ell(u,v) \end{cases}$$



After adding v_{i+1} to K

If $(v_{i+1}, w) \in E$

$$\text{dist}[w] = \min\{\text{dist}[w],\ \text{dist}[v_{i+1}] + \ell(v_{i+1}, w)\}$$

update(v_{i+1}, w)

dijkstra(G, ℓ, s)

$dist[s] = 0$

$\forall u \neq s$, $dist[u] = \infty$

$U = V$ (insert $(u, dist[u])$ $\forall u$)

while U is not empty

Choose $u \in U$ with minimum $dist[u]$

Remove u from U.

(u = DeleteMin())

for each v s.t. $(u,v) \in E$

$dist[v] = \min(dist[v], dist[u] + \ell(u,v))$

DecreaseKey$(v, dist[v])$

Priority queue

Contains a set of (element, key) pairs

integer

- Insert (elem, key)
- DecreaseKey (elem, key)

(Replaces elem's old key w/ new key)

- DeleteMin()

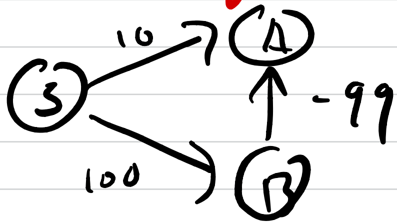
Dijkstra's runtime

Insert n times
DeleteMin n times
DecreaseKey m times

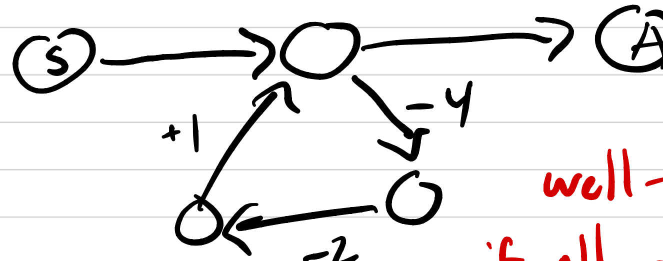
Implementation	Insert	DelMin	Decrease	Total
array	$O(1)$	$O(n)$	$O(1)$	$O(n^2+m) = O(n^2)$
binary heap	$\log(n)$	$\log(n)$	$\log(n)$	$O((n+m)\log(n))$
Fibonacci heap	$O(1)$	$O(\log(n))$	$O(1)$	$O(n\log(n) + m)$

Mikkel Thorup 2004: $O(n \log\log n + m)$

Negative weights



Does it make sense?



well-defined if all cycles have positive lengths

update(u,v)

$$\text{dist}[v] = \min\{\text{dist}[v],\ \text{dist}[u] + \ell(u,v)\}$$

① Update is "safe": $\text{dist}[v] \geq d(s,v)$

② Suppose shortest $s \to v$ path is $s \to \to \to \to \to u \to v$ and $d[u] = d(s,u)$. Then after update, $\text{dist}[v] = d(s,v)$

$s \to u_1 \to u_2 \to u_3 \to \cdots \to u_t \to v$

→ update(s, u_1)

→ update(u_1, u_2)

⋮

→ update(u_{t-1}, u_t)

→ update(u_t, v)

Bellman-Ford(G, ℓ, s)

For $i = 1 .. N-1$

Update all the edges

(for all $(u,v) \in E$,

update(u,v))

Runtime: $O(nm)$

(best known for arbitrary weights)