Shortest Path

$G = (V, E)$, Directed Graph (undirected = 2* directed)

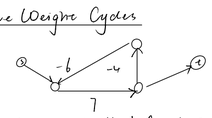
Source "s", destination "t", edge weight $w(e)$ = length of e

⇒ Find shortest path from s to t.

$\delta(s,t)$ = length of shortest path

Single-Source Shortest path : fixed source to all nodes in graph

Negative Weight Cycles



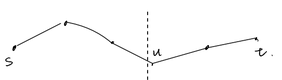
Problem is not well-defined in Negative Weight Cycles

We can always continue in cycle!

⇒ Assume no negative cycles

Subpath optimality

Lemma:



(s,t) is shortest path ⇒ subpath (s,u) & (u,t) are shortest

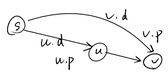
proof by contradiction

if (s,u) is not shortest. then a shorter $(s,u)'$ exists.

then (s,t) is not shortest since we can switch (s,u) to $(s,u)'$

⇒ Contradiction!

Concept of Relaxation



if $v.d > u.d + w(u,v)$ then,

$v.d = u.d + w(u,v)$ and $v.p = u$

Bellman-Ford Algorithm

① Set all $v.d = \infty$, except $s.d = 0$

② relax all edges once.

⋮

relax all edges $V-1$ times.

```
Shortest-Path(G,s):
for each node v ∈ V:
    v.d ← ∞
s.d ← 0.
for i ← 1 to V-1 do:
    for all edge (u,v) ∈ E:
        if v.d > u.d + w(u,v) then:
            v.d = u.d + w(u,v)
            v.p ← u.
```

Bellman-Ford as Dynamic Programming

$$\begin{cases} v.d[i] = \min\limits_{u:(u,v)\in E} \{ u.d[i-1] + w(u,v) \}. \\ v.d[0] = \infty \end{cases}$$

```
Bellman-Ford(G,s):
for each node v ∈ V:
    v.d ← ∞, v.p ← nil.
s.d ← 0.
for i ← 1 to V-1:
    for each u ∈ V:
        if u.d changed in previous iteration:
            for each v ∈ Adj[u]:
                if u.d + w(u,v) < v.d then:
                    v.d ← u.d + w(u,v)
                    v.p ← u.
        if no v.d changed in this iteration then terminate.
```

Shortest Path in DAG

$$\delta(s,v) = \min_{u:(u,v)\in E} \{ \delta(s,u) + w(u,v) \}.$$

```
DAG-Shortest-Path(G,s):
topologically sort vertices in G.
for each vertex v ∈ V
    v.d ← ∞
    v.p ← nil.
s.d ← 0
for each vertex u in topological order.
    for each v ∈ Adj[u].
        if u.d + w(u,v) < v.d then:
            v.d ← u.d + w(u,v)
            v.p ← u.
```

Dijkstra's Algorithm

① Maintain set of exploring nodes S.

② init: $S = \{s\}$. $s.d = 0$, $s.p = nil$., $\forall v \in S$, $v.d = \infty$.

③ Key Lemma:

If all edges leaving S is relaxed. Let $v \in V - S$. $v.d$ is minimum

⇒ $v.d = \delta(s,v)$

Algorithm:

```
Dijkstra(G,s):
for each v ∈ V do:
    v.d ← ∞, v.p ← nil, v.color ← white.
s.d ← 0.
create a priority min-heap Q using d as key.
insert(Q,s)
while Q ≠ ∅:
    u ← Extract_min(Q)                      ← using Lemma
    if u.color = black, continue.
    output (u, u.d, u.p)
    u.color = black.
    for v ∈ Adj[u] do:                      ← relax all edges leaving S
        if v.color = white and u.d + w(u,v) < v.d then:
            v.p ← u.
            v.d ← u.d + w(u,v)
            Insert(Q,v)
```

Running Time: $O(E \log V)$

Correctness (Proof of key lemma)

Lemma: If all edges leaving S is relaxed. Let $v \in V - S$. $v.d$ is minimum

⇒ $v.d = \delta(s,v)$

Proof By contradiction:

Assume $v.d \neq \delta(s,v)$, Since $v.d \geq \delta(s,v)$, $v.d > \delta(s,v)$

Consider the shortest path from S to V

Let x be the last node in S and y be the first node outside S

$x.d = \delta(x,y)$ by definition ($x \to y$ is on the shortest path)

Since $x \to y$ has been relaxed ⇒ $y.d \leq x.d + w(x,y)$

since $x \to y$ is in the shortest path from s to v.

then the subpath from s to y is also shortest (proved previously)

⇒ $\delta(s,y) = x.d + w(x,y)$.

$\delta(s,y) \leq \delta(s,v)$ Assuming no negative weights

$y.d \leq \delta(s,y) \leq \delta(s,v) < v.d$ ⇒ v does not have minimum d! Contradicted!

⇒ Proved

"Dijkstra fails with Negative Weights"

Cannot Be Fixed By "re-weighting" (i.e. adding constant)

A* Search

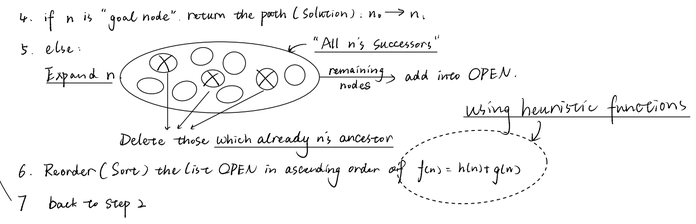
Procedure:

1. Create search tree T, put in start node n_0

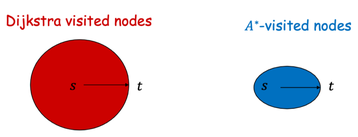
   Put n_0 on a list called OPEN.
2. if OPEN is empty, return failure.
3. OPEN —extract→ first node n
4. if n is "goal node" return the path (solution): $n_0 \to n$.
5. else:

   Expand n → "All n's successors" —remaining nodes→ add into OPEN.

   Delete those which already n's ancestor
6. Reorder (Sort) the list OPEN in ascending order of $f(n) = h(n) + g(n)$ using heuristic functions
7. back to Step 2



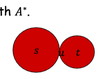
A^* can be used with any function h provided that $h(u,t) \leq \delta(u,t)$. Faster than Dijkstra in practice, but asymptotically the same.



Other fast algorithms $s - t$ shortest path

Bidirectional: start Dijkstra expansions from both s and t in parallel. When you find a common node u in both expansions, stop. The shortest path has distance: $\delta(s,u) + \delta(t,u)$.

Can also be combined with A^*.



Continuous monitoring of shortest path: the previous algorithms return a one-time path, assuming fixed edge weights. Real navigation systems monitor the traffic conditions and continuously update your path when traffic conditions change (e.g., accidents).

Many later algorithms for $s - t$ paths (based on contraction hierarchies, partial materialization, landmarks etc) are much faster than Dijkstra in practice.

All-Pairs Shortest Path

Output: $\delta(u,v)$ for all pairs of (u,v)

Data Structure (for storing all $\delta(u,v)$)

if Storing all pairs explicitly ⇒ Space: $O(V^3)$

Storing each $w(u,v)$:

adjacency matrix = Space: $O(v^2)$ extraction: $O(1)$

Using Previous Algorithm:

Dijkstra's Algorithm: $T = O(VE \log V)$ —if dense→ $O(n^3 \log n)$

if negative weight

Bellman-Ford: $O(V \cdot VE)$ —if dense→ $O(n^4)$

Dynamic Programming:

Solution 1: $d_{ij}^{(m)} = \min\limits_{1 \leq k \leq n} \{ d_{ik}^{(m-1)} + w(k,j) \}$, $d_{ij}^{(1)} = w(i,j)$

where $d_{ij}^{(m)}$: length of shortest path from i to j contains at most m edges.

```
Slow-All-Pairs-Shortest-Paths(G):
d_ij^(1) = w(i,j) for all 1 ≤ i,j ≤ n
for m ← 2 to n − 1
    let D^(m) be a new n × n matrix
    for i ← 1 to n
        for j ← 1 to n
            d_ij^(m) ← ∞
            for k ← 1 to n
                if d_ik^(m−1) + w(k,j) < d_ij^(m) then d_ij^(m) ← d_ik^(m−1) + w(k,j)
return D^(n−1)
```

$O(n^4)$ time, $O(n^3)$ space (can be improved to $O(n^2)$)

Solution 2: $d_{ij}^{(2s)} = \min\limits_{1 \leq k \leq j} \{ d_{ik}^{(s)} + d_{kj}^{(s)} \}$ (cut down size by half).

Algorithm:
- Calculate $D^{(1)}, D^{(2)}, D^{(4)}, D^{(8)}, \ldots$
- Calculating each matrix takes $O(n^3)$ time: total time = $O(n^3 \log n)$.

Solution 3: Floyd-Warshall

$d_{ij}^{(k)}$: length of shortest path from i to j contains vertices in $\{1, 2, \ldots, k\}$

$d_{ij}^{k} = \min \{ d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \}$.

- Case 1: k is not a vertex on the shortest path from i to j
  => then the path uses only vertices in $\{1,2,\ldots,k-1\}$. $d_{ij}^{(k-1)}$
- Case 2: k is an intermediate node on the shortest path from i to j,
  => path can be split into shortest subpath from i to k and a subpath from k to j.
  Both subpaths use only vertices in $\{1,2,\ldots,k-1\}$ $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

```
Floyd-Warshall(G):
d_ij = w(i,j) and intermed[i,j] ← nil for all 1 ≤ i,j ≤ n.
for k ← 1 to n:
    for i ← 1 to n:
        for j ← 1 to n:
            if d_ik + d_kj < d_ij then:
                d_ij ← d_ik + d_kj
                intermed[i,j] ← k.
return D
```

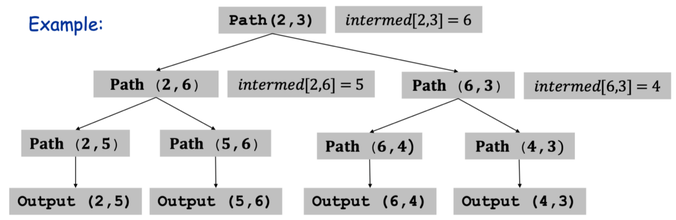
$O(n^3)$ time

$O(n^2)$ space

Extracting Shortest Path:

```
Path(i,j):
if intermed[i,j] = nil then
    output (i,j)
else
    Path(i, intermed[i,j])
    Path(intermed[i,j], j)
```

Example:



Running time: O(length of the shortest path)