Last time on SSSP

- directed graph
- weighted edges
- negative weights ok
- no negative cycles.

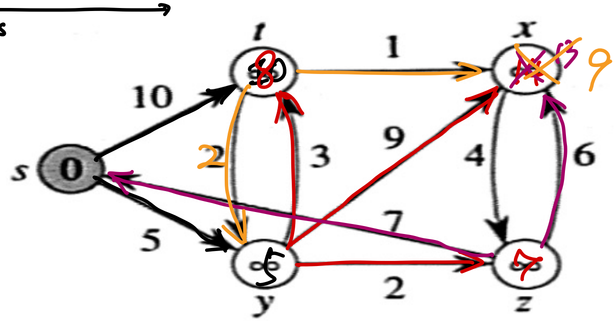
appoarch. relax each edge.

1. Bellman-Ford: Relax the edges in fixed order $|V|-1$ times. (or until no changes)
   - will find a negative cycle either
     - relax the edges one more time ($|V|$th time). still change
     - Any time find a negative val on source, guaranteed to be a negative cycle.
2. Directed Acyclic Graphs (DAG)
   - do a toposort and relax in that order.

<u>This time</u>: Always choose the currently MIN-KEY as the Dijkestra estimation. — the greedy appoarch.

```
Dijkestra (V, E, w, s):
    INIT-SINGLESORCE (V, s).
    S ← ∅
    Q ← V      (a min queue)
    while Q is not empty:
        u ← EXTRACT-MIN(Q)
        S ← S ∪ {u}
        for each vertex v ∈ Adj[u]
            RELAX (u, v, w).
```

Example. $\xrightarrow[\text{Steps}]{1\ \ 2\ \ 3}$



Runtime.

- Every vertex comes in exactly once.
- lg V for MIN-Heap
- worst V

↳ $O(V^2 + V \lg V)$.

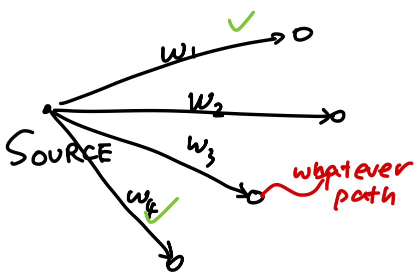
Binom heap $O(\lg n)$

Fib heap $O(1)$

```
Dijkestra (V, E, w, s):
   INIT-SINGLESORCE (v, s).
   S ← ∅
   Q ← V        (a min queue)
   while Q is not empty:
      u ← EXTRACT-MIN(Q)
      S ← S ∪ {u}
      for each vertex v ∈ Adj[u]
         RELAX (u, v, w).
```

Proof of choice.

Assume we have a optimal answer, if it does not have the greedy choice, show you can cut sth out of that and paste to a greedy choice.



ass. $w_1 < w_2 < w_3 < w_4$.

w_3 + whateverpath $> w_1$

for no negative edges

↦ take out $w_3 + \cdots$

put in $w_1 \cdots$

induction on other parts.

This Time: <u>All pair Shortest path</u>: Naïve algo.

- Given a directed graph $G = (V, E)$, weight function $w : E \rightarrow \mathbb{R}$, $|V| = n$.
- Goal: create an $n \times n$ matrix of shortest-path distances from every vertex to every other vertex $\delta(u, v)$.
- Could run BELLMAN-FORD once from each vertex:
  - $O(V^2E)$ which is $O(V^4)$ if the graph is *dense* ($E =\sim V^2$).
- If no negative-weight edges, could run Dijkstra's algorithm once from each vertex:
  - $O(V E \lg V)$ with binary heap—$O(V^3 \lg V)$ if dense
- We'll see how to do in $O(V^3)$ in all cases with dynamic programming (we have already shown the shortest path problem has optimal substructure).

Already shown SPA contains shortest subpaths.

The formal problem statement:

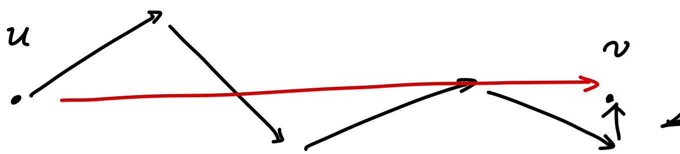
- Assume that G is given as adjacency matrix of weights: $W = (w_{ij})$, with vertices numbered 1 to n.

$$w_{ij} = \begin{cases} 0 & \text{if } i = j\,, \\ \text{weight of } (i, j) & \text{if } i \neq j, (i, j) \in E\,, \\ \infty & \text{if } i \neq j, (i, j) \notin E\,. \end{cases}$$

- Output is the shortest path matrix $D = (d_{ij})$, where $d_{ij} = \delta(i, j)$.

Dynamic Programming Steps

1. Define structure of optimal solution, including what are the largest sub-problems.
2. Recursively define optimal solution
3. Compute solution using table bottom up
4. Construct Optimal solution



Sb tells me this is the small

range of internal paths: $(0, |V|-2)$.

limiting how many edge you'd like to use.

Base Case $\longrightarrow$ if <u>use 0 internal edge</u>: adj[u][v].

...

shortest path from any vert to any vert at most 1 edge paths

Next. prob

u · Most 1^{int} vert allowed on path · v $\quad \min\begin{cases} w_{ij} \\ \forall k,\ w_{ik} \to w_{kj} \end{cases}$

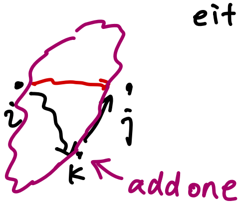
Check which is better

Having got the idea formally write it.

1. Define struct of opt sol

Shortest path btwn any two verts i and j

either



- Shortest path i to j at most $|V-2|$ internal verts.
- Shortest path i to $k + w_{k,j}$

at most $|V-2|$ internal nodes

2. Recursive define opt sol

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i=j \\ \infty & i \neq j \end{cases}$$

shortest path $i \to j$ with 0 int. edges.

$$l_{ij}^{(1)} = \begin{cases} 0 & i=j \\ w_{i,j} & \exists \text{ adj mat} \\ \infty & \text{o.w.} \end{cases}$$

shortest path $i \to j$ with 1 int. edges.

$l_{ij}^{(m)}$ shortest path $i \to j$ with m edges on path. $(m-1$ int verts$)$

$$l_{ij}^{(m)} = \min \left\{ \begin{array}{l} l_{ij}^{(m-1)} \\ \min\limits_{1 \le k \le |V|} \left\{ \underbrace{l_{ik}^{(m-1)} \oplus w_{k,j}}_{\text{Mat. mul elementwize addition}} \right\} \end{array} \right\}$$

add elemwise

← EXTEND.

```
EXTEND(L,W, n)
  create L' an n ×n matrix
  for i ← 1 to n
    for j ← 1 to n
      l'ij ←∞
    for k ← 1 to n
      lij ← min(l'ij , lik + wkj )
  return L'
```

```
SLOW-APSP(W, n)
  L(1) ← W
  for m ← 2 to n -1
    L(m) ←EXTEND(L(m-1),W, n)
  return L(n-1)
```

↑ $O(V^3)$

1° Do it m-1 times.

```
SLOW-APSP(W, n)
  L(1) ← W
  for m ← 2 to n -1
    L(m) ←EXTEND(L(m-1),W, n)
  return L(n-1)
```

↑ $O(V^4)$

2° Use FAST Pow. i.e.

$$M^8 = \left((M M)^2 \right)^2$$

```
FASTER-APSP(W, n)
  L(1) ← W
  m ← 1
  while m < n-1
    L(2m) ←EXTEND(L(m), L(m), n)
    m ← 2m
  return L(m)
```

← $O(V^3 \log V)$.