All Pairs Shortest Path

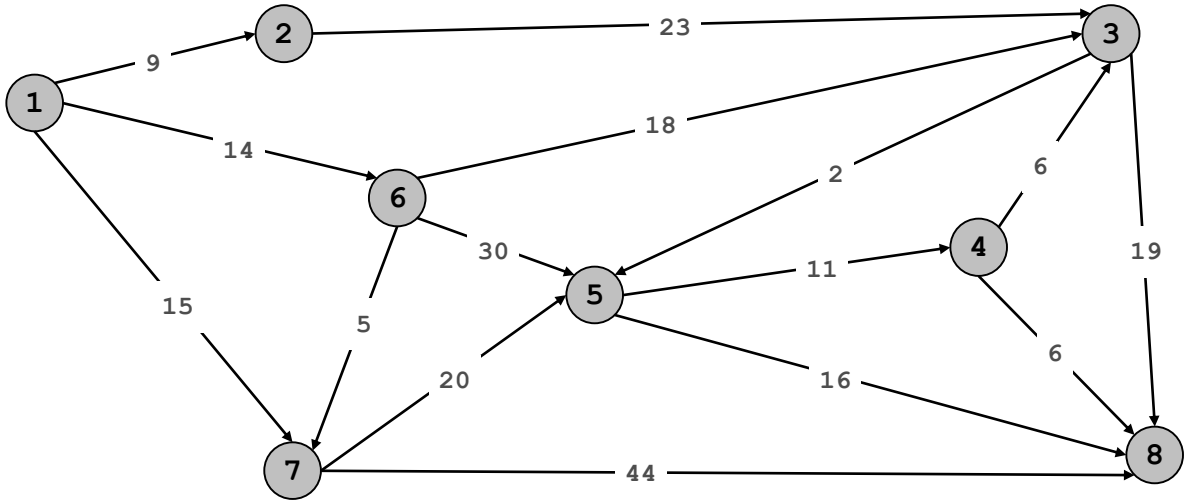
- Lots of Single Sources
- Johnson's Algorithm
- Dynamic Programing
- Divide and Conquer
- Matrix Multiplication
- Floyd-Warshall Algorithm

All Pairs Shortest Path Problem

Shortest path network.

- Directed graph G = (V, E).
- Each edge e is associated with a weight w(e) (positive or negative)
- Assume G contains no negative cycles

All Pairs Shortest Path Problem (APSP): Compute d(u,v), the length of the shortest path from u to v, for any two nodes u, v in V



Lots of Single Sources

Lots of Single Sources

Non-negative weights.

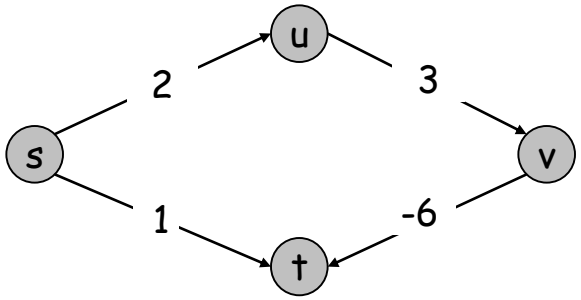
- Apply n times Dijkstra's algorithm: $O(nm+n^2 \log n)$ which is $O(n^3)$ in the worst case

No assumption on weights.

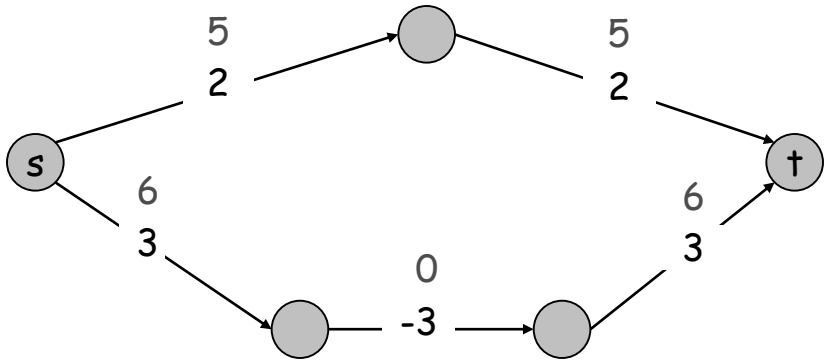
- Apply n times Bellman-Fords algorithm: $O(n^2 m)$ which is $O(n^4)$ in the worst case

Reweighting: Negative Weights

Dijkstra. Can fail if negative weights.



Re-weighting. Adding a constant to every edge weight can fail (the shortest path may change).



Johnson's Algorithm

Reweighting

Reweighting.

- Suppose each vertex v has some associated price c(v), which might be positive, negative, or zero.
- Consider a graph G′ which is equal to G excepts the weights. The weight function w′ of G′ is defined as follows w′(e) = c(u)+w(e)-c(v) for any edge e=(u,v).

Lemma. For any path P: u, x_1,..,x_k, v in G and G′, w′(P)=w(P)+c(u)-c(v)

Pf.

$w'(P) = w'(u, x_1)+w'(x_1, x_2)+\ldots+w'(x_{k-1}, x_k)+w'(x_k, v) =$

$(c(u)+w(u, x_1)-c(x_1))+(c(x_1)+w(x_1, x_2)-c(x_2))+\ldots+(c(x_k)+w(x_k, v)-c(v)) =$

$w(u, x_1)+w(x_1, x_2)+\ldots+w(x_{k-1}, x_k)+w(x_k, v)+c(u)-c(v) = w(P)+c(u)-c(v)$

Corollary. Any shortest path from u to v in G is preserved in G′ as c(u)-c(v) is fixed for all paths between u and v.

Reweighting

Main question. Is it possible to determine all c(u)s such that for any two nodes u and v we have $c(u)+w(u,v)-c(v) \geq 0$?

Lemma. Add an extra vertex s to G and assume w(s,u)=0 for any u. Assume d(u)=the length of the shortest path from s to u. If we set c(u)=d(u), then $c(u)+w(u,v)-c(v) \geq 0$.

Pf.

We have to show $d(u)+w(u,v) \geq d(v)$ which is true due to the properties of shortest paths.

Johnson's Algorithm

```
JohnsonAPSP(G, w)
   add a vetex s to G
   foreach vetex u
       w(s,u) = 0
   d(s,.)=BellmanFord(G,w,s)
   foreach edge (u,v)
          w'(u,v) = d(s,u)+w(u,v)-d(s,v)
   foreach vetex u
       d'(u,.)=Dijkstra(G,w',u)
   foreach vetex u
      foreach vetex v
        d(u,v)= d'(u,v)-d(s,u)+d(s,v)
```

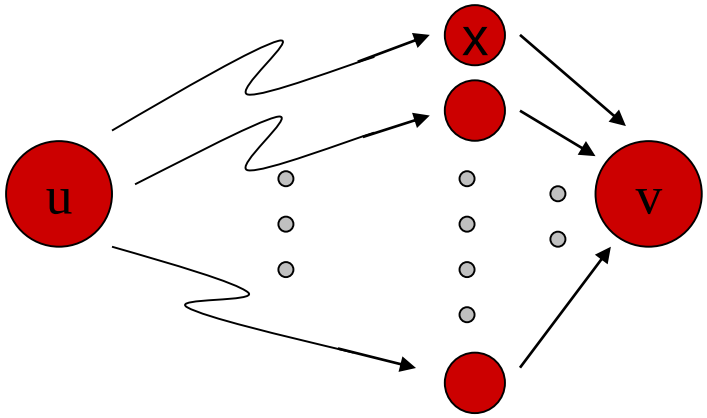
Running time. $O(nm+n^2 \log n)$

Dynamic Programming

Dynamic Programming

$$d(u,v) = \begin{cases} 0 & \text{if } u = v \\ \min_{(x,v)\in E}\{d(u,x) + w(x,v)\} & \text{otherwise} \end{cases}$$

- The above obvious recursive definition only works when G is a dag
- Any directed cycles drive the recurrence into an infinite loop.



Dynamic Programming

$$d(u,v) = \begin{cases} 0 & \text{if } u = v \\ \min_{(x,v)\in E}\{d(u,x) + w(x,v)\} & \text{otherwise} \end{cases}$$

- The above obvious recursive definition only works when G is a dag
- Any directed cycles drive the recurrence into an infinite loop.

We can break this loop by introducing as an additional parameter.

- d(u, v, i) = the length of the shortest path from u to v that uses at most i edges
- The shortest path between any two vertices traverses at most n-1 edges, so the true shortest-path distance is d(u, v, n-1).

Dynamic Programming

$$d(u,v,i) = \begin{cases} 0 & \text{if } u = v \\ \infty & \text{if } i = 0, u \neq v \\ \min_{(x,v)\in E}\{d(u,x,i-1) + w(x,v)\} & \text{otherwise} \end{cases}$$

- d(u, v, i) = the length of the shortest path from u to v that uses at most i edges
- The shortest path between any two vertices traverses at most n-1 edges, so the true shortest-path distance is d(u, v, n-1).

Dynamic Programming

```
DP-APSP(G, w)
   for each vertex u
      for each vertex v
         if u = v then d(u,v,0)=0
         else d(u,v,0) = ∞
   for i = 1 to n-1 do
      for each vertex u
         for each vertex v ≠ u
            d(u,v,i) = d(u,v,i-1)
            for each edge (x,v)
               if d(u,v,i) > d(u,v,i-1)+w(x,v) then
                  d(u,v,i) = d(u,v,i-1)+w(x,v)
```

Running time. $O(n^4)$

Space. $O(n^3)$

Dynamic Programming

```
DP-APSP(G, w)
   for each vertex u
      for each vertex v
         if u = v then d(u,v)=0
         else d(u,v) = ∞
   for i = 1 to n-1 do
      for each vertex u
         for each vertex v ≠ u
            for each edge (x,v)
               if d(u,v) > d(u,v)+w(x,v) then
                  d(u,v) = d(u,v)+w(x,v)
```

Running time. $O(n^4)$

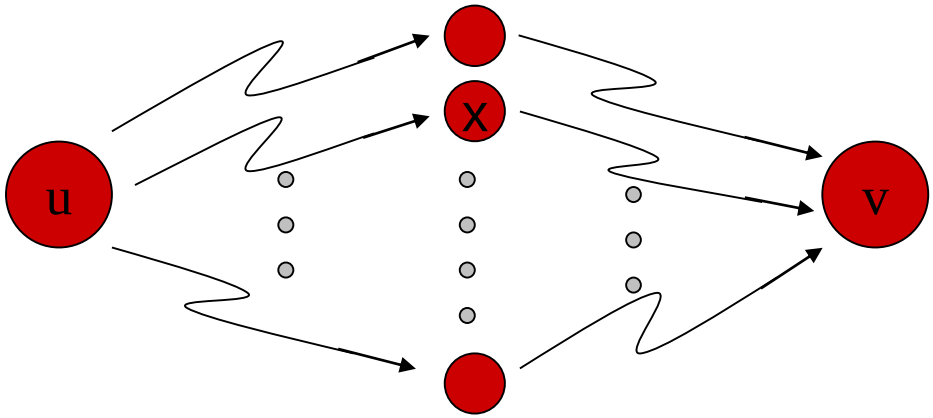
Space. $O(n^2)$

Divide and Conquer

Divide and Conquer

$$d(u,v,i) = \begin{cases} w(u,v) & \text{if } i = 1 \\ \min_{x \in V} \{d(u,x,i/2) + d(x,v,i/2)\} & \text{otherwise} \end{cases}$$

- This recurrence only works when i is a power of 2
- d(u, v, i) is the true shortest-path distance from u to v for all $i \geq n-1$
- In particular, we can use $i = 2^{\lg n} < 2n$



Divide and Conquer

```
DP-APSP(G, w)
   for each vertex u
      for each vertex v
         if u = v then d(u,v,0)=0
         else d(u,v,0) = ∞
   for i = 1 to log n do
      for each vertex u
         for each vertex v ≠ u
            d(u,v,i) = ∞
            for each vertex x
               if d(u,v,i) > d(u,x,i-1)+d(x,v,i-1) then
                  d(u,v,i) = d(u,x,i-1)+d(x,v,i-1)
```

Running time. $O(n^3 \log n)$

Space. $O(n^2 \log n)$

Divide and Conquer

```
DP-APSP(G, w)
   for each vertex u
      for each vertex v
         if u = v then d(u,v,0)=0
         else d(u,v,0) = ∞
   for i = 1 to log n do
      for each vertex u
         for each vertex v ≠ u
            for each vertex x
               if d(u,v) > d(u,x)+d(x,v) then
                  d(u,v) = d(u,x)+d(x,v)
```

Running time. $O(n^3 \log n)$

Space. $O(n^2)$

Matrix Multiplication

Matrix Multiplication

```
MatrixSquare(A)
    for i = 1 to n do
        for j = 1 to n do
            A'[i,j] = 0
            for k = 1 to n do
                A'[i,j]=A'[i,j]+A[i,k].A[k,j]
```

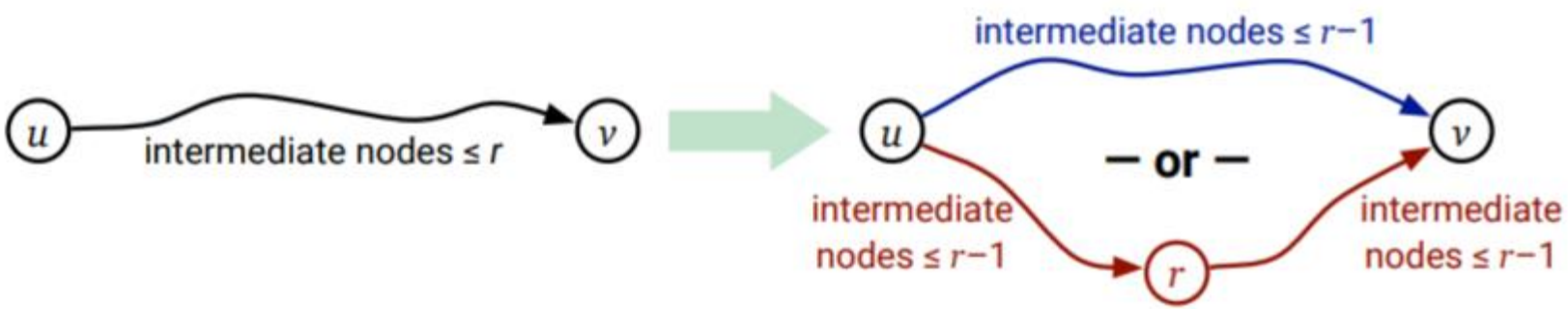
```
APSP-MainLoop(D)
    for each vertex u
        for each vertex v
            D'[u,v] = ∞
            for each vertex x
                D'[u,v]=min(D'[u,v], D[u,x]+D(x,v))
```

- Instead of operators (+, .) we have (min, +)
- We have to compute D, D^2, D^4, …, $D^{\log n}$
- Strassen's algorithm to multiply matrices does not work here, why?

Floyd-Warshall Algorithm

Floyd-Warshall Algorithm

- Number the vertices arbitrarily from 1 to n.
- For every pair of vertices u and v and every integer r, we define a path π(u, v, r) is the shortest path from u to v that passes through only vertices numbered at most r.
- π(u, v, 0) can't pass through any intermediate vertices, so it must be the edge (if any) from u to v.
- For r > 0, π(u, v, r) either (i) passes through r or (ii) is equal to π(u, v, r-1).
- In case (i), π(u, v, r) consists of two paths π(u, r, r-1) and π(r, v, r-1)



Floyd-Warshall Algorithm

$$d(u,v,r) = \begin{cases} w(u,v) & \text{if } r = 0 \\ \min\{d(u,r,r-1) + d(r,v,r-1), d(u,v,r-1)\} & \text{otherwise} \end{cases}$$

```
FloydWarshall(G, w)
    for each vertex u
        for each vertex v
            d(u,v,0) = w(u,v)
    for r = 1 to n do
        for each vertex u
           for each vertex v
               if d(u,v,r-1) < d(u,r,r-1)+d(r,v,r-1) then
                  d(u,v,r) = d(u,v,r-1)
               else
                   d(u,v,r) = d(u,r,r-1)+d(r,v,r-1)
```

Running time. $O(n^3)$

Space. $O(n^3)$

Floyd-Warshall Algorithm

$$d(u,v,r) = \begin{cases} w(u,v) & \text{if } r = 0 \\ \min\{d(u,r,r-1) + d(r,v,r-1), d(u,v,r-1)\} & \text{otherwise} \end{cases}$$

```
FloydWarshall(G, w)
    for each vertex u
        for each vertex v
            d(u,v,0) = w(u,v)
    for r = 1 to n do
        for each vertex u
           for each vertex v
               if d(u,v] > d(u,r)+d(r,v) then
                  d(u,v) = d(u,r)+d(r,v)
```

Running time. $O(n^3)$

Space. $O(n^2)$

References

References

- Chapter 9 of the text book "algorithms" by Jeff Erikson