Lecture 12
CS 170

Sanjam Garg.

DP Approach

1) Define appropriate subproblems
2) Write a recurrence relation
3) Determine order of computation.
   ↳ induces a DAG structure

Subproblem Structure

(i) input $x_1 \dots x_n$ and a subproblem is $x_1 \dots x_i$

$x_1 \dots x_i$ $x_{i+1} \dots x_n$

(ii) input $x_1 \dots x_n$ & $y_1 \dots y_m$ and a subproblem is $x_1 \dots x_i$ & $y_1 \dots y_j$

$x_1 \dots x_i$ $x_{i+1} \dots x_n$

$y_1 \dots y_j$ $y_{j+1} \dots y_m$

(iii) input $x_1 \dots x_n$ and a subproblem is $x_i \dots x_j$

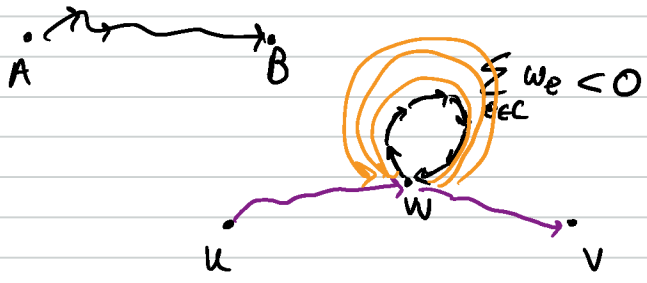
$x_1 \dots$ $x_i \dots x_j$ $\dots x_n$

Shortest paths in a Graph

→ edges with +ve weights

→ edges with -ve weights

- DAG
- arbitrary connections but no -ve cycles.



Dijkstra	$O((n+m)\log n)$	+ve
Bellman-Ford	$O(nm)$	-ve
DAG-SSSP (DP)	$O(n+m)$	

Today: SSSP (Single Source Shortest Path)

Input: -Weighted graph $G=(V,E)$ weights w_e for edge e $(w_e \in R)$

-Source s

Output: For every v, $dist(v)$ = shortest path from s to v $s \rightsquigarrow v$

① $dist(v)$ for $v \in V$

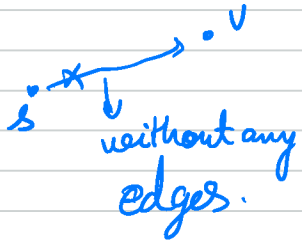
② $dist(v) = \min_{(u,v)\in E} \{ dist(s,u) + w_{uv} \}$

③

Need to redefine the subproblems.

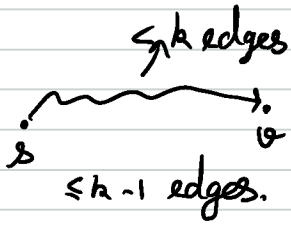
① $dist(v,k)$ = distance of shortest path from $s \rightsquigarrow v$ involving upto k edges.

$dist(v,k) \leftarrow dist(u_1, k-1)$, $dist(u_t, k-1)$



Base: $dist(s,0) = 0$, $dist(v,0) = \infty \quad \forall v \neq s$

②



Case I: $dist(v, k-1)$

Case II: $dist(u, k-1) + w_{uv}$

$dist(v,k) = \min\{ dist(v,k-1), \min_{(u,v)\in E} \{ dist(u,k-1) + w_{uv} \} \}$

③ Nice order to compute $k=1, k=2 \ldots$

Algorithm

```
For v ∈ V   dist(v,0) = ∞

dist(s,0) = 0

For all k = 1 .... n-1                          ← n
    For all v ∈ V                               ← n
        dist(v,k) = dist(v,k-1)
        For all (u,v) ∈ E
            if dist(v,k) > dist(u,k-1) + w(u,v)
            then dist(v,k) = dist(u,k-1) + w(u,v)

Output ∀ v ∈ V  dist(v, n-1)
```

$n \cdot (n+m) \rightarrow O(nm)$

Very similar to BF ↓ worst case $O(nm)$

In BF $dist(v) \in \{ dist(v,0), dist(v,1), \ldots, dist(v,n) \}$

Single Source Reliable Shortest Path.

Input: -Weighted graph $G=(V,E)$ weights w_e for edge e $(w_e \in R)$

-Source s and a bound B

Output: For every v, $dist(v)$ = shortest path from s to v taking almost B edges

$dist(v,B) \rightarrow$ desired output

All pairs shortest path

Input: -Weighted graph $G=(V,E)$ weights w_e for edge e $(w_e \in R)$

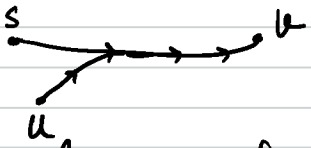
~~-Source s~~

Output: For every u,v, $dist(u,v)$ = shortest path from u to v $u \rightsquigarrow v$

Run BF n-times to get u to v paths for all u.

↳ $O(n^2 m)$

$dist(u,k)$



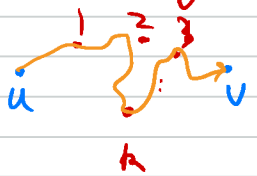
$dist(u,v,k)$ → shortest path from u to v with atmost k edges

↳ $n^2 m$

$u \overset{1 \cdots k-1}{\rightsquigarrow} k$

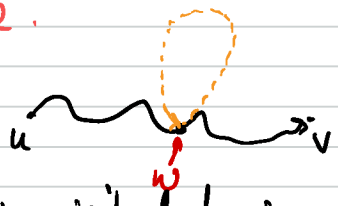
$dist(u,v,k)$ = shortest path from $u \rightsquigarrow v$ that takes vertices in $\{1 \cdots k\}$ only.

$dist(u,v,0) = w_{uv}$



Claim: On the shortest path from u to v no vertex w happens twice.

Proof:

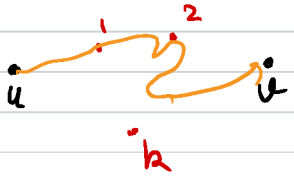


Assume w is visited twice then removing the cycle yield a shorter path.

□

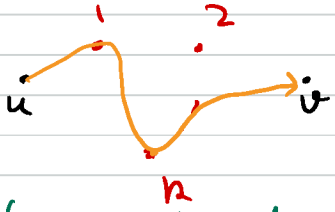
$dist(u, v, k) \rightarrow$

Case I:



$dist(u, v, k-1)$

Case II:

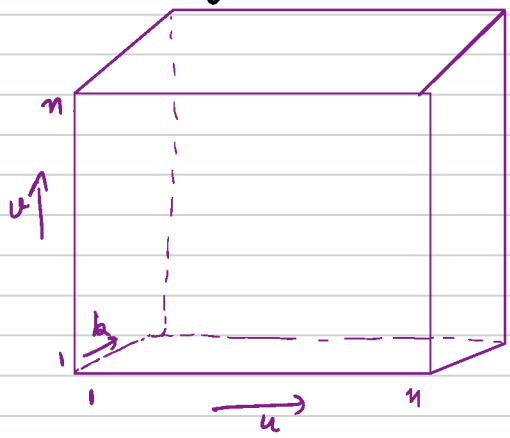


$dist(u, k, k-1) + dist(k, v, k-1)$

$$dist(u, v, k) = \min \{ dist(u, v, k-1),\ dist(u, k, k-1) + dist(k, v, k-1) \}$$

③ Order of computation.

$d(u, v, k)$ only depends on $d(\cdot, \cdot, k-1)$



$O(n^3)$

Algorithm (Floyd-Warshall)

For $i, j = 1 \dots n$ $\quad dist(i, j, 0) = \infty$

For $(i, j) \in E$ $\quad dist(i, j, 0) = w_{ij}$

For $i = 1 \dots n$ $\quad dist(i, i, 0) = 0$

```
For k = 1 .... n
    For i = 1 .... n
        For j = 1 .... n
```

$$dist(i, j, k) = \min\{ dist(i, j, k-1),\ dist(i, k, k-1) + dist(k, j, k-1) \}$$

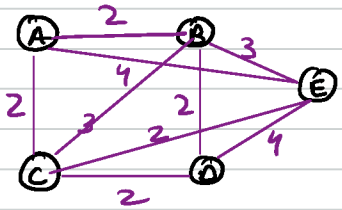
Output $dist(i, j, k)$ for all $i, j \in V$

$O(n^3)$

Travelling Salesman Problem.

Input: n cities & distances d_{ij} $(i \neq j)$

Goal: Find minimum distance path from city 1 back to city 1 visiting each city exactly once.



Brute-force: (i) $1 \to 2 \to 3 \dots \to n$

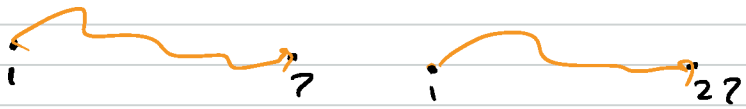
(ii) $1 \to 3 \to 2 \to 4 \dots \to n$

$\vdots$

$n! \approx n^n \to$ costly.

$C(j)$ = cost of minimum path from $1 \to j$

Simplification: TS can end in any of the n cities.



$C(S, j)$ → the least cost path that

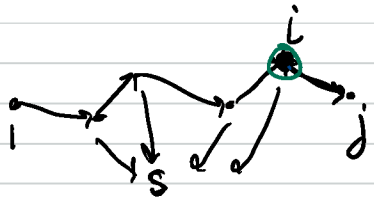
$2^n \times n$

$S \subseteq \{1 \ldots n\}$ s.t $j, 1 \in S$

① starts at 1

② visits all nodes in set S

③ ends in node j.



$i \in S \setminus \{1, j\}$

$$C(S, j) = \min_{i \in S \setminus \{j,1\}} C(S \setminus \{j\}, i) + w_{ij}$$