Lecture 29: Dynamic Programming
(Shortest Paths)
Given a graph with edge weights, find paths that minimize total weight of edges used

Dijkstra’s shortest path algorithm assumed non-negative edge weights

Another easy case: DAG
What if the graph has both negative weights and cycles?

If the graph has negative cycles, some distances are negative infinity. Otherwise, \( \text{Dist}(C) \equiv \min_{\{v \mid (v,C) \text{ is an edge}\}} \{\text{Dist}(v) + w((v,C))\} \)

But this doesn’t give a consistent recursion, because the recursion graph is not a DAG, so recursion loops around cycles……

How can we make recursion acyclic when the graph is cyclic?
LESSONS

Generalizing problems to allow recursion and hence DP

Keeping recursions simple

Trying multiple recursive algorithms based on different types of case analysis leading to different DP algorithms

Using reductions (again)
WHY WE MIGHT WANT TO DO THIS: ARBITRAGE
What vertex do we visit first (after $s$)?

What vertex do we visit last (before $t$)?

What is the middle vertex along the path?

Do we ever visit vertex $v$?
Attempt 1:

BTSP(s, t, G)

- If $s = t$ return 0
- $M := \infty$
- For each $v \in N(s)$ do:
  - $M := \min(M, (w(s, v) + BTSP(v, t, G)))$
- return $M$
PROBLEM: LOOPING

Cycles in graph = cycles in recursive calls
Attempt 2:

\[ \text{BTSP}(s, t, G) \]

- If \( s = t \) return 0
- \( M := \infty \)
- For each \( v \in N(s) \) do:
  - \( M := \min(M, (w(s, v) + \text{BTSP}(v, t, G - \{s\}))) \)
- return \( M \)
PROBLEM

Changing G in recursive calls: exponentially many sub-graphs.

Usable for longest simple path = TSP = NP-complete, so approach seems inherently exponential.
Let’s put a budget on how deep we’ll recurse, \( T \)

That corresponds to looking at which paths?

How large should \( T \) be?
Let’s put a budget on how deep we’ll recurse, $T$

That corresponds to looking at paths with at most $T$ edges

How large should $T$ be: $T = n$ : search all simple paths. Not all paths we search are simple, but that’s OK
If there is a negative cycle, we can keep going around it. But we can break up any cycle into a set of simple cycles.
If there is a negative cycle, we can keep going around it. But we can break up any cycle into a set of simple cycles.

If the overall cycle is negative, at least one simple cycle is negative. So if there is any negative cycle, there is one of size at most $n$. 
Let’s put a budget on how deep we’ll recurse, $T$

That corresponds to looking at paths of length at most $T$

If there’s no negative cycle, then the shortest paths are simple, don’t repeat vertices.

So the shortest path between any two vertices is of length at most $|V| = n$
Attempt 3:

$BTSP(s, t, G, T)$

If $T=0$ THEN IF $s=t$ return 0 ELSE return infinity;

Min := $BTSP(s,t,G,T-1)$

For each $v \in N(s)$ do:

Min := min (Min, $w((s,v))+BTSP(v,t,G, T-1)$ )

Return Min

Bellman-Ford: Convert above using DP
BF[s, t, T]: shortest weight of a path from s to t with at most T edges

Note: G is also an input to recursion, but G doesn’t change. Only need changing parts as parameters in matrix.
BASE CASES:

Base case: $T = \text{?}$

$BF[s, t,?] = 0$ if $\text{??}$

$\text{??}$ otherwise
Base case: $T = 0$

$BF[s, t, 0] = \begin{cases} 
0 & \text{if } s = t \\
\infty & \text{otherwise}
\end{cases}$
RECURSIVE PART:

Check all first steps along path

$$BF[s, t, T] := \min (BF[s, t, T - 1], \min_{u \in N(s)} BF[u, t, T - 1] + w((s, u)))$$
Sub-problem: $BF(u,v, T) :=$ shortest path from $u$ to $v$ of length at most $T$ (in terms of number of edges)

Recursion: $BF(u,v, T+1) = \min (BF(u,v,T), \min_{\{w \in N(u)\}} BF(w,v,T) + w((u,w))$
BELLMAN-FORD ALGORITHM

- Initialize BF(V,V, 1...|V|) to infinity
- For u ∈ V do: BF(u,u,1):=0
- For (u, v) ∈ E do: BF(u,v,1):= w((u,v))
- For T=2 to n do:
  - For v ∈ V do: For u ∈ V do:
    - BF(u,v,T):= BF(u,v,T-1)
    - For w ∈ N(u) do: IF w(u,w) + BF(w,v,T-1) < BF(u,v,T)
      - THEN BF(u,v,T):=w((u,w))+BF(w,v,T-1)
  - IF any BF(u,u,n) < 0 return "Negative cycle"
  - ELSE return the array BF(u,v,n)
Initialize BF(V, V, 1...|V|) to infinity

For u ∈ V do: BF(u, u, 1) := O(n)
For (u, v) ∈ E do: BF(u, v, 1) := w((u, v))O(m)
For T=2 to n do: O(n) times
   For v ∈ V do: O(n) times
      For u ∈ V do: O(Σ{u∈V} deg(u)) = O(m)
      BF(u, v, T) := BF(u, v, T-1)
      For w ∈ N(u) do: IF w(u, w) + BF(w, v, T-1) < BF(u, v, T) O(deg(u))
         THEN BF(u, v, T) := w((u, w)) + BF(w, v, T-1)
   IF any BF(u, u, n) < 0 return "Negative cycle"
ELSE return the array BF(u, v, n) O(n^2)
Total: O(n^2m)
We never use $BF(u, v, T)$ beyond step $T + 1$.
It's OK if we've made $BF(u, v, T)$ smaller as long as we never make it less than the shortest path length.

So we can just use one matrix $BF(u, v)$ rather than $n$ different matrices, and make both read and write steps from the same Matrix.
Bellman-Ford’s recursion based on "What is the first vertex on the path from $u$ to $v$?"

What other question could we ask about the path?

What is the last vertex on the path? Symmetric
Bellman-Ford’s recursion based on "*What is the first vertex on the path from \( u \) to \( v \)*?"

What other question could we ask about the path?
What is the last vertex? (BF in reverse)

What is the middle vertex of the path? (Min plus matrix multiply method)

Is \( v_n \) ever used in the path? (Floyd-Warshall algorithm)
MIN-PLUS METHOD

- Uses same sub-problems, BF(u,v,T), but only for T a power of 2, so log n possible values

Tries all possible middle elements of path.

\[ BF(u,v,2^k) = \min_{w \in V} BF(u, w, 2^{(k-1)}) + BF(w, v, 2^{(k-1)}) \]
MIN PLUS ALGORITHM

- Initialize MP(V, V, 0... log |V|) to infinity

- For u ∈ V do: MP(u, u, 0):=0
- For (u, v) ∈ E do: MP(u, v, 0):= w((u, v))
- For k=1 to log n do:
  - For v ∈ V do: For u ∈ V do:
    - MP(u, v, k):= MP(u, v, k-1)
    - For w ∈ V do: IF MP(u, w, k-1) + MP(w, v, k-1) < MP(u, v, k)
      - THEN MP(u, v, k):=MP(u, w, k-1)+MP(w, v, k-1)
  - IF any MP(u, u, log n) < 0 return ```Negative cycle''
  - ELSE return the array MP(u, v, log n)
MIN PLUS ALGORITHM

- Initialize MP(V, V, 0... log |V|) to infinity
- For u ∈ V do: MP(u, u, 0):=0
- For (u, v) ∈ E do: MP(u, v, 0):= w((u, v))
- For k=1 to log n do: O(log n)
- For v ∈ V do: For u ∈ V do: O(n^2)
- MP(u, v, k):= MP(u, v, k-1)
- For w ∈ V do: IF MP(u, w, k-1) + MP(w, v, k-1) < MP(u, v, k)
  THEN MP(u, v, k):=MP(u, w, k-1) + MP(w, v, k-1)O(n)
- IF any MP(u, u, log n) < 0 return ``Negative cycle''
- ELSE return the array MP(u, v, log n) Total O(n^3 log n)
WHICH IS BETTER?

- Bellman-Ford $O(n^2m)$
- Min Plus $O(n^3 \log n)$

BF better for very sparse graphs, Min Plus better for dense graphs.

(Floyd-Warshall $O(n^3)$ beats both)

More important, BF easier to implement in dynamic situations, and maintains routing information. So better for optimal routing in network.
Is \( v \) used at all?

Need to eliminate vertices in order, so that remaining vertices are consecutive.

Subtlety one: need to distinguish between endpoints and set of intermediate points

Subtlety two: implicitly assumes no negative cycles, so that shortest paths can be simple.
Assuming the shortest paths are simple, find the shortest path in $G$ from $s$ to $t$ using intermediate nodes $U = \{u_1, \ldots, u_k\}$, where $s$ and $t$ may or may not be in $U$.

\[
\text{BTFW}(s, t, G, U)
\]

- IF $U$ is empty, and $s = t$ return 0
- IF $U$ is empty and $(s, t)$ is an edge return $w(s, t)$
- IF $U$ is empty, return infinity
- Usecase := $\text{BTFW}(s, u_k, G, U - \{u_k\}) + \text{BTFW}(u_k, t, G, U - \{u_k\})$
  
  //Use $u_k$ at most once
- Dontusecase := $\text{BTFW}(s, t, G, U - \{u_k\})$ //Don’t use $u_k$
- Return max(Usecase, Dontusecase)
REACHABLE SUBPROBLEMS

$s$ can be arbitrary, $t$ can be arbitrary, $U = \{v_1, \ldots, v_k\}$ for some $0 \leq k \leq n$

$FW[s, t, k]$: Best path from $s$ to $t$ using only $v_1, \ldots, v_k$ as intermediate vertices

Base case: $U$ is empty, $k = 0$, $FW[s, s, 0] = 0$, $FW(s, t) = w(s, t)$ for edge $(s, t)$, $FW[s, t, 0] = \infty$

Recursion: $FW[s, t, k] = \min ( \quad , \quad )$
REACHABLE SUBPROBLEMS

$s$ can be arbitrary, $t$ can be arbitrary, $U = \{v_1, ..., v_k\}$ for some $0 \leq k \leq n$

$FW[s, t, k] : \text{Best path from } s \text{ to } t \text{ using only } v_1, ..., v_k \text{ as intermediate vertices}$

Base case: $U$ is empty, $k = 0$, $FW[s, s, 0] = 0$, $FW(s, t) = w(s, t)$ for edge $(s, t)$, $FW[s, t, 0] = \infty$

Recursion: $FW[s, t, k] = \min(F[s, v_k, k - 1] + F[v_k, t, k - 1], F[s, t, k - 1])$

Top down: $k$ decreases, bottom-up: $k$ increases
DPFW($G$)
Initialize array FW[$V, V, 1 ... |V|$]
For $s$ in $V$ do: FW[$s, s, 0$] := 0
For edges $(s, t) \in E$ do: FW[$s, t, 0$] = $w(s, t)$
For $s$ in $V$ do: For $t$ in $V - \{s\}$ do: F[$s, t, 0$] := $\infty$
For $k = 1$ to $|V|$ do:
   For $s$ in $V$ do:
      For $t$ in $V$ do:
         FW[$s, t, k$] = $\min(FW[s, v_k, k - 1] + FW[v_k, t, k - 1], FW[s, t, k - 1])$
Return the matrix FW[$s, t, |V|$]

(Note: can save space by reusing single matrix)
WHICH IS BETTER?

- Bellman-Ford $O(n^2m)$
- Min Plus $O(n^3\log n)$

- BF better for very sparse graphs, Min Plus better for dense graphs.

- (Floyd-Warshall $O(n^3)$ beats both)

- More important, BF easier to implement in dynamic situations, and maintains routing information. So better for optimal routing in network.