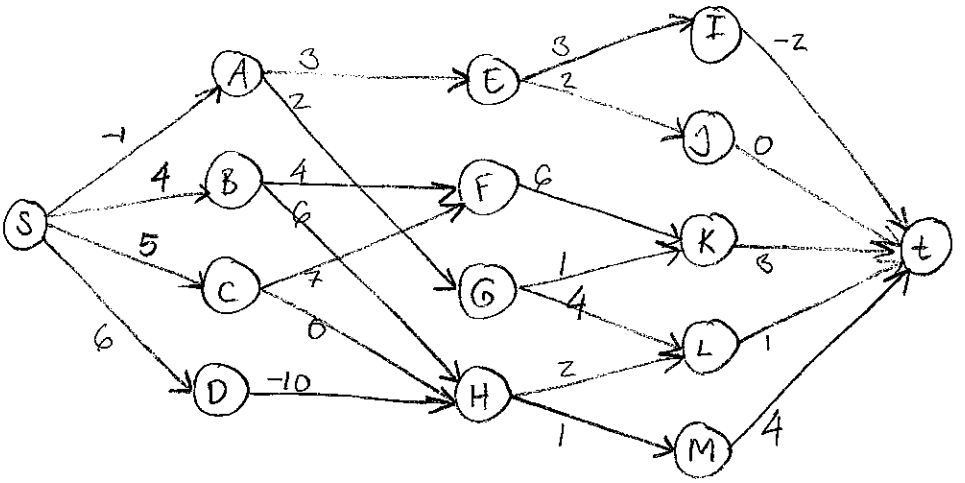
ORIE 3310/5310/5311
Lecture 9

A Motivating Example: The shortest-path problem (revisited)

- Consider the shortest-path problem on the following directed graph:

$G = (N, E)$:



c_{ij} = distance between node i and node j

- Note:
  - The above graph has a special structure: the nodes are "layered"
  - That is: for each node i, there is a number $k_i \geq 0$ such that any path from s to i must involve exactly k_i edges.

    Ex: $i = s$, then $k_i = 0$
    $i = E$, then $k_i = 2$
    $i = t$, then $k_i = 4$
  - We allow negative edge lengths.
  - In any layered graph, there is no cycle. In particular, there are no negative-cycles.

- In the past 4 weeks, we have seen that we can solve the shortest-path problem as a min-cost flow problem using Ford-Fulkerson's algorithm, or as a linear programming problem.

  However, solving it using Dynamic Programming will give us an insight to how we can solve more general classes of problems using Dynamic Programming (DP).

- Consider our graph again, and suppose that while we are trying to solve for a shortest-path from s to t, someone tries to help us by giving us the following information:

Node i	The shortest path from s to i	
$i = I$	$S \to A \to E \to I$,	length = 5
$i = J$	$S \to A \to E \to J$,	length = 4
$i = K$	$S \to A \to G \to K$,	length = 2
$i = L$	$S \to D \to H \to L$,	length = -2
$i = M$	$S \to D \to H \to M$,	length = -3.

Q: Can we now "easily" compute the shortest path from s to t? (How? What is this path and what is its length?)

A: Yes, we can!

Observation: Any path from s to t must go through one of nodes I, J, K, L, M right before arriving at t.

Thus, the shortest s-t path must also go through one of these nodes, then use one more edge to arrive at t. Moreover, the part of this path from s to i must be the shortest s-i path (for i any node along this path).

So, to obtain the shortest s-t path, take the minimum among the following 5 paths:

① the shortest s-I path + the edge (I,t)
② the shortest s-J path + the edge (J,t)
③ the shortest s-K path + the edge (K,t)
④ s-L + the edge (L,t)
⑤ s-M + the edge (M,t)

So, if $f_3^*(i)$:= the shortest path from s to i (using 3 edges), $i = I, J, K, L, M$

then the shortest s-t path (using 4 edges) has length:

$$\min_{i \in \{I,J,K,L,M\}} \left\{ f_3^*(i) + c_{it} \right\}.$$

$$= \min \left\{ f_3^*(I) + c_{It},\ f_3^*(J) + c_{Jt},\ f_3^*(K) + c_{Kt},\ f_3^*(L) + c_{Lt},\ f_3^*(M) + c_{Mt} \right\}$$

$$= \min \{ 5 + (-2),\ 4 + 0,\ 2 + 3,\ -2 + 1,\ -3 + 4 \}$$
$$= \min \{ 3,\ 4,\ 5,\ \textcircled{-1},\ 1 \} = -1, \text{ achieved for } i = L.$$

So, the shortest s-t path has length -1, and the path is:

$s \to D \to H \to L \to t$

($s \to D \to H \to L$: the shortest s-L path; $L \to t$: the edge (L,t).)

- Here, we solve the problem of finding the shortest s-t path using the optimal solutions of five subproblems:
  finding the shortest s-I path,
  the shortest s-J path, ...,
  the shortest s-N path.

- In the example above, we were given the "hints" of what those shortest s-i paths are, for each $i \in \{I, J, K, L, M\}$ by a helpful stranger.

  Q: In reality, how would we find these information ourselves?

  A: Using the same method!

  Ex: To find the shortest s-K path,
  - consider nodes i such that there is an edge from i directly to K:
    $i = F$ and $i = G$.
  - Suppose that
    $f_2^*(F)$ = the length of the shortest s-F path (involving 2 edges)

    $f_2^*(G)$ = the length of the shortest s-G path (involving 2 edges)

  - Then, the shortest s-K path (involving 3 edges):

$$f_3^*(K) = \min_{i \text{ s.t. } (i,K) \in E} \left\{ f_2^*(i) + c_{iK} \right\}.$$

$$= \min \left\{ f_2^*(F) + c_{FK}, f_2^*(G) + c_{GK} \right\}.$$

Note that: $f_2^*(F)$ and $f_2^*(G)$ are also unknown to us at the moment! But they can be computed in terms of $f_1^*(i)$, etc.

- So, in practice, we will start by computing the shortest s-i paths for nodes i that are 1 edge away from s first:

Stage 1: $f_1^*(A) = -1$ using edge (S,A)
$f_1^*(B) = 4$ using edge (S,B)
$f_1^*(C) = 5$ using edge (S,C)
$f_1^*(D) = 6$ using edge (S,D)

Then, we consider nodes that are 2 edges away from s:

Stage 2: $f_2^*(E) = f_1^*(A) + C_{AE} = 2$ using edge (A,E)

$f_2^*(F) = \min\{f_1^*(B) + C_{B,F} + f_1^*(C) + C_{CF}\}$
$= \min\{4+4, 5+7\}$
$= 8$ ∴ using edge (B,F).

$f_2^*(G) = f_1^*(A) + C_{AG} = 2+2 = 4$ using edge (A,G)

$f_2^*(H) = \min\{f_1^*(B) + C_{BH}, f_1^*(C) + C_{CH}, f_1^*(D) + C_{DH}\}$
$= \min\{4+6, 5+0, (6+(-10))\}$
$= -4$ ∴ using edge (D,H).

Then, nodes that are 3 edges away from S:

Stage 3: $f_3^*(I) = f_2^*(E) + C_{EI} = 2+3 = 5$ using (E,I)

$f_3^*(J) = f_2^*(E) + C_{EJ} = 2+2 = 4$ using (E,J)

$f_3^*(K) = \min\{f_2^*(F) + C_{FK}, f_2^*(G) + C_{GK}\}$
$= \min\{8+6, (4+1)\} = 5$ ∴ using (G,K)

$f_3^*(L) = \min\{f_2^*(G) + C_{GL}, f_2^*(H) + C_{HL}\}$
$= \min\{4+4, (-4+2)\} = -2$ ∴ using (H,L)

$f_3^*(M) = f_2^*(H) + C_{HM} = -4+1 = -3$ using (H,M).

And finally, as we saw before:

$f_4^*(t) = \min_{i \text{ s.t. } (i,t) \in E}\{f_3^*(i) + C_{it}\} = -1$ using (L,t).

- So, the length of the shortest s-t path is -1, and the path itself can be traced backwards by looking at the choice that satisfies the min at each step:
  ↳ the edge

  to t uses (L, t).

  to L uses (H, L)

  to H uses (D, H)

  to D uses (S, D)

  So, the shortest s-t path: $S \to D \to H \to L \to t$.

Remarks

- We make the decision in stages.
  In the example above, the stages correspond to the number of edges away from S.

- At each stage, we consider a few different nodes.
  Think of it as follows:
  At stage k, we consider all possible states we could be at after traversing k edges

  So, the states at stage k are nodes that can be reached from s using exactly k edges.

- Let N_k = set of nodes that can be reached from s using k edges.
  At stage k, node j, we compute the shortest s-j path (using k edges) by considering several possibilities (: which node i the path goes through right before reaching node j) and taking the minimum:

$$f_k^*(j) = \min_{i \in N_{k-1} \text{ s.t. } (i,j) \in E} \{ f_{k-1}^*(i) + C_{ij} \}.$$

  - $i \in N_{k-1}$ s.t. $(i,j) \in E$: nodes i in stage $k-1$ such that there is an edge directly from i to j
  - $f_{k-1}^*(i)$: the length of shortest s-i path (using $k-1$ edges)
  - C_{ij}: the length of edge (i,j).

- The equation above implies that for nodes j in stage k, to compute the shortest s-j path, we only need to know the lengths of the shortest s-i paths of nodes i in stage $k-1$ (the previous stage only! Not the stages before $k-1$ or stages after k)

  This is cool because this makes the computation of $f_k^*(j)$ very easy.

We call the equation above the "recursive relation", the "recursion", or "Bellman's Equation".

Also note that in determining $f_k^*(j)$ using $f_{k-1}^*(i)$ (for $i \in N_{k-1}$ s.t. $(i,j) \in E$), we only make one decision, chosen among a specified set of possible options.
→ So decisions are made in stages, for each state in that stage.

When can we use DP?

- Let us look at our recursive relation from the example above:

$$f_k^*(j) = \min_{i \in N_k \text{ s.t. } (i,j) \in E} \{ f_{k-1}^*(i) + c_{ij} \}.$$

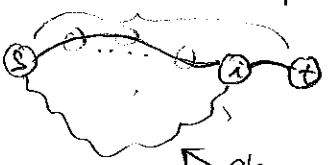
the length of the shortest s-j path = the minimum among all nodes i that has an edge directly from i to j of: {the length of the shortest s-i path + the length of the edge (i,j)}.

- We have the above relation because of the observation:

Observation: If we have a shortest path from s to t that passes through node i first, then uses the edge (i,t), then the portion of the path from s to i must be the shortest $(s$-$i)$ path.

Shortest s-t path, length = $D_{st} = D_{si} + c_{it}$

Proof:



↖ Shortest s-i path, length = $\tilde{D}_{si}$

By contradiction,

Suppose D_{st} is the length of the shortest s-t path.

So, $D_{st} - c_{it}$ = the length of the s to i portion of this path.

Suppose to the contrary, there is a path from s to i that is shorter, with length $\tilde{D}_{si}$ s.t.

$$\tilde{D}_{si} < D_{si} = D_{st} - c_{it}.$$

Then, adding c_{it} to both sides:

$$\tilde{D}_{si} + c_{it} < D_{si} + c_{it} = D_{st}.$$

This produces an s-t path of shorter length, contradiction!

∴ The s to i portion of the shortest s-t path must be a shortest s-i path. □.

- Based on the above observation, we say that <u>the shortest path problem on a layered graph</u> has an <u>optimal substructure</u>

<u>Defn</u> A problem has an <u>optimal substructure</u> if an optimal solution to a problem <u>contains within it</u> optimal solutions to its subproblems.

<u>Ex</u> A shortest (optimal) s-t path <u>contains within it</u> shortest s-i paths for each node i involved in this path.

- This suggests that if a problem has an optimal substructure, then we can first solve for optimal solutions to its subproblems, then use them to construct an optimal solution to the problem.

- This is called the Principle of optimality and it allows us to construct recursive relations and use Dynamic Programming.

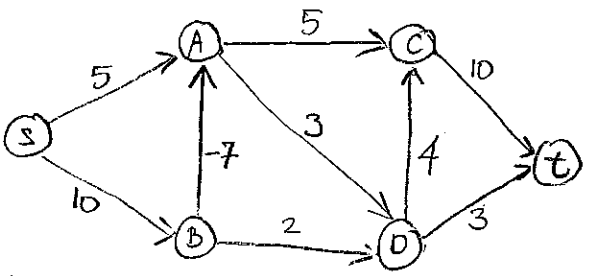
Some more examples of problems that can be solved using DP

Ex 1 Directed Acyclic Graphs: Shortest-path problem

We have seen that if a directed graph is layered, we can use DP to find the shortest s-t path.

We can also use DP for any directed graph that does not contain cycles even though it is not necessarily layered.

For example:



Not layered because node 1 can be reached from s using either 1 or 2 edges; node 3 using 2 or 3 edges, etc.
However it does not contain any directed cycles!

We cannot let Stage correspond to "number of edges away from s" anymore, because the number is not well-defined for all nodes.

However, we can do the following:

- Stage k $\leftrightarrow$ compute the shortest s-j path using only <u>at most</u> k edges.
- States at stage k $\leftrightarrow$ all nodes in the graph that can be reached by <u>at most</u> k edges, denoted by N_k.
- recursion for each state at stage k:

$f_k^*(j)$ = the shortest s-j path using <u>at most</u> k edges.

$$f_k^*(j) = \min_{\substack{i \in N_{k-1} \\ \text{s.t.} (i,j) \in E}} \left\{ f_{k-1}^*(i) + c_{ij} \right\}$$

Solving this problem: HW 5!

Finally, formalizing the DP approach, these are the basic steps of setting up a problem to be solved using DP:

1) Specify the <u>stages</u>
2) Specify the <u>states</u> at each stage
3) Specify the <u>set of possible decisions</u> for <u>each state</u>, at <u>each stage</u>.
4) Describe (in words) the optimization function to be solved at each state i and each stage k.
5) Specify <u>boundary conditions</u> for this function, that can be used as the "base case" (the "smallest subproblem" ← to be solved first) for the recurrence relation
6) Take the word-description in (4) and specify the <u>recurrence relation</u> [this step puts together not just (4) but all the previous steps]
7) Compute optimal solutions of subproblems (starting from the base case in (5) to the "next smallest" problem, etc.) via the recurrence relation, stage-by-stage (i.e. at each stage, <u>compute for all states</u> before moving to the next stage), while also keeping track of the corresponding optimal decision at each stage.
8) Trace back to find the opt solution to the problem.