MATH 566: Lecture 10 (09/19/2024)

Today: * Shortest path problem: – assumptions, analogy
– application
– Dijkstra's algo

We finish the discussion on flow decomposition with a theorem:

Flow Decomposition Theorem Any feasible flow $\bar{x}$ ($x_{ij}\ \forall (i,j)\in A$) can be decomposed into

(a) the sum of flows in directed paths from supply to demand nodes, and

(b) the sum of flows around directed cycles.

The decomposition will have <u>at most</u> $m+n$ directed paths and cycles, out of which at most m are cycles.

<u>IDEA</u> Each step of the flow decomposition algorithm identifies a path P or a cycle W. Pushing $\Delta(P)$ or $\Delta(W)$ (capacity of P or W) will necessarily zero out at least one $b(i)$ or one y_{ij}.

Corollary Any circulation $\bar{x}$, i.e., flow $\bar{x}$ with $b(i)=0\ \forall i$, can be decomposed into the sum of flows around at most m directed cycles.

Notice that we could use the bounds of $m+n$ on the number of cycles and paths for doing the complexity analysis of the flow decomposition algorithm.

We will use the flow decomposition theorem in the proofs of certain results on the shortest path problem, and also later on for other results.

Shortest Path (SP) Problem

Recall: length of a path P is sum of lengths of arcs in P.

We study the following generalization of the SP problem: → as previously introduced

Given: $G=(N,A)$ with costs c_{ij} for $(i,j) \in A$, and a source node s.
Goal: Find shortest length directed paths from s to all $j \in N/\{s\}$.

Assumptions

1. c_{ij} are integers.
   For ease of analysis of complexity of algorithms.

2. There is a directed path from s to each $i \in N/\{s\}$.

   If there is no such path to a particular node j, we add an extra arc (s,j) with $c_{sj} = +\infty$. If such an arc is part of the SP tree, we know that node is in fact not reachable from s via a directed path.

3. There is no negative cost cycle in G.

   If there is such a cycle, going around it infinitely many times will decrease the total cost without limits! We will have to then impose extra restrictions that we use each arc at most once. But such restrictions make the problem hard to solve. Under this assumption, each arc would be used at most once in the SP tree.

In general, we do permit some C_{ij}'s to be negative, as long as there are no negative cost cycles.

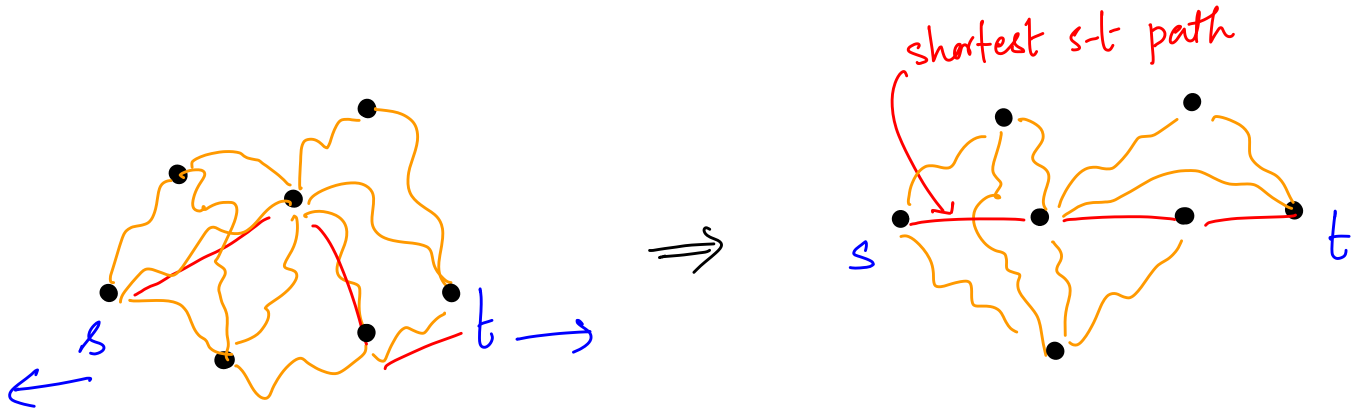
We first consider the case where all $C_{ij} \geq 0$. This case will be easier than the general case that allows $C_{ij} < 0$. As it turns out, we would use similar ideas to solve both cases.

An Analogy: The string model

Assume $C_{ij} \geq 0$.

Make a model of G with beads and strings. Each node gets a bead, and each arc gets an inelastic string of length C_{ij}.

Consider the s-t version of the shortest path problem. Imagine pulling apart the beads corresponding to the nodes s and t.



The set of strings that become taut (or tight) first represents an s-t shortest path.

Applications of the Shortest Path Problem

Recall: TeX paragraph spacing problem (AMO Problem 1.7).
Section 4.3 in AMO describes many more applications modeled as SP problems. We consider one case here.

The knapsack Problem

A hiker wants to decide which items to include in her knapsack. She has to choose from n items. The knapsack can carry up to W lbs. Item i has weight w_i and utility u_i. The goal is to maximize total utility while not exceeding total weight W.
We will model this problem as a shortest path problem.

Example $n=4, W=6$

i	1	2	3	4
w_i	4	2	3	1
u_i	40	15	20	10

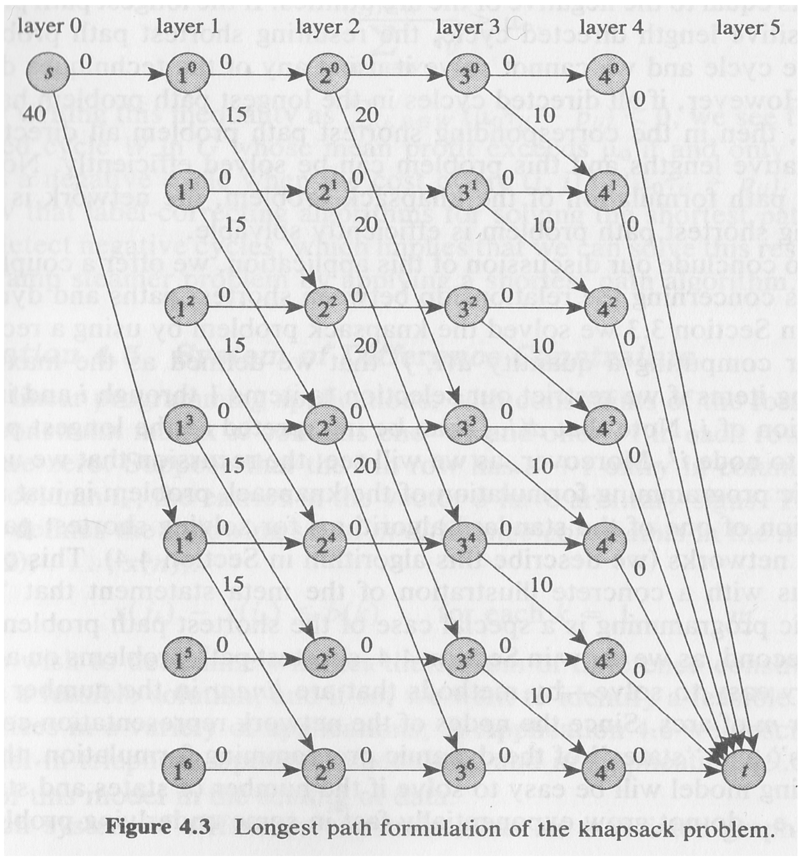


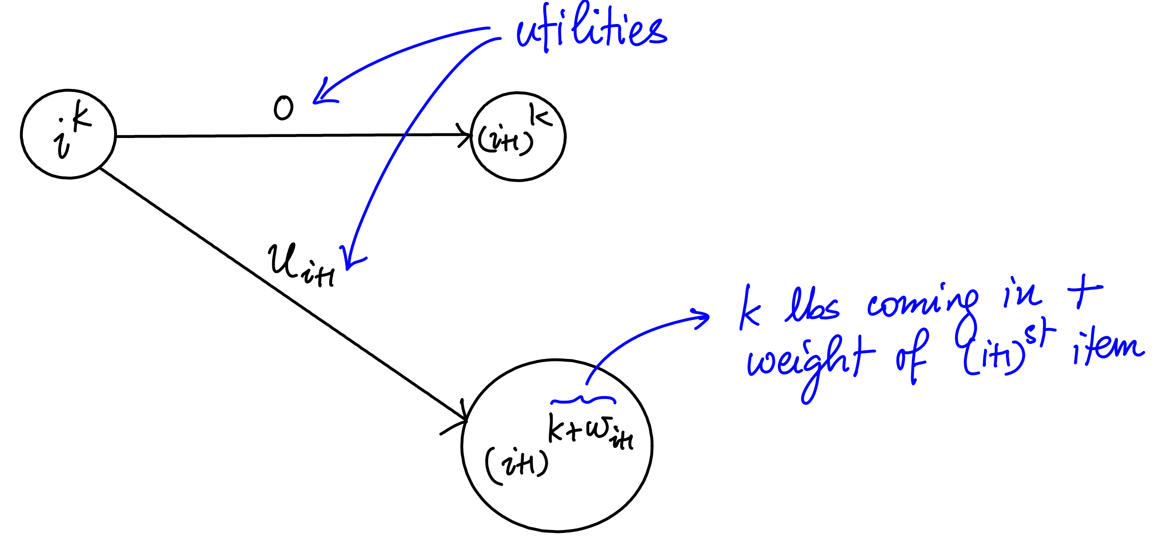
Figure 4.3 Longest path formulation of the knapsack problem.

Notation

$\textcircled{i^k}$: items $1, 2, \ldots, i$ use k lbs (of total weight)

The figure illustrates a longest path formulation of this problem.

From node $\boxed{i^k}$, we have two outarcs, corresponding to the hiker including item $i+1$ in the knapsack, or leaving it out.



From s, we add $(s, 1^0)$ with utility 0, and $(s, 1^{w_1})$ with utility u_1. Finally, we connect all nodes from Layer n (last layer), i.e., nodes n^w for $w = 1, 2, \ldots, W$, to t with zero utility arcs.

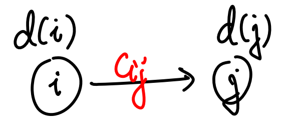
Find the s-t longest path. When $u_i \geq 0 \ \forall i$, we can negate all utilities and solve the s-t SP problem.

The knapsack problem has many applications, including in cryptography, multiprocessor scheduling, fair division. etc. The model illustrated here (with layers corresponding to each item and levels corresponding to the total weight) is a typical instance of dynamic programming. Hence, under mild assumptions, many dynamic programming problems could be cast as shortest path problems.

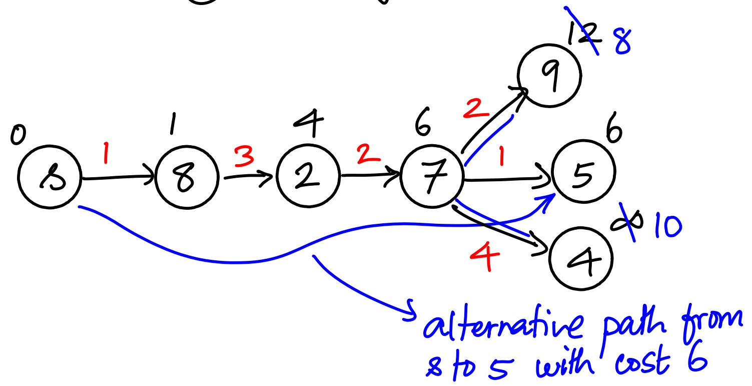
Algorithms for Shortest Path Problem (Recall, we assume $c_{ij} \geq 0 \; \forall (i,j)$ for now)

Let $d(\cdot)$ denote a vector of temporary distance labels for the nodes. Thus, $d(i)$ = length of **some** path from s to i. Hence, $d(i)$ is an upper bound on the SP length from s to i.

A key Step in all SP algorithms



```
Procedure UPDATE(i)
  for each $(i,j) \in A(i)$ do
    if $d(j) > d(i) + c_{ij}$ then
        $d(j) := d(i) + c_{ij}$;
        pred(j) := i;
    endif
  endfor
```



UPDATE(7): We update d(4) and d(9), but do not change d(5).

Note the similarity to the process of looking for admissibility of arcs from node i, and marking the heads of admissible arcs.

When the $d(i)$'s become permanent, they represent SP distances from s to i.

We now present the first algorithm for SP problems where $c_{ij} \geq 0$ for all (i,j).

Dijkstra's algorithm (pronounced 'Daikstraa')

We maintain and update two (sub) sets of nodes.

S : set of "permanent" nodes ($d(i)$ is permanent)

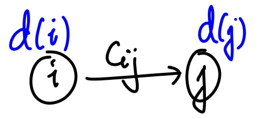
$\bar{S} = N \setminus S$: set of nodes with temporary $d(i)$'s.

```
algorithm Dijkstra;
begin
    $S := \emptyset$; $\bar{S} := N$;                              } initialization
    $d(i) := \infty$ for each node $i \in N$;
    $d(s) := 0$ and pred($s$) := 0;
    while $|S| < n$ do  ⟶ run till every d(i) is made permanent
    begin
        let $i \in \bar{S}$ be a node for which $d(i) = \min\{d(j) : j \in \bar{S}\}$;   } node selection
make i     { $S := S \cup \{i\}$;
"permanent"{ $\bar{S} := \bar{S} - \{i\}$;
        for each $(i, j) \in A(i)$ do
            if $d(j) > d(i) + c_{ij}$ then $d(j) := d(i) + c_{ij}$ and pred($j$) := $i$;   } UPDATE(i)
    end;
end;
```

Figure 4.6 Dijkstra's algorithm.

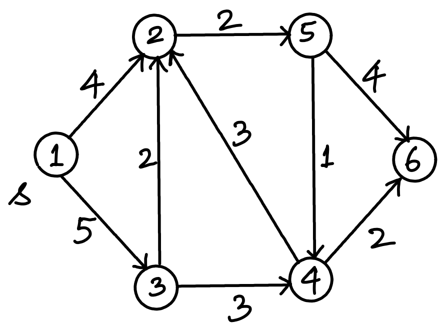
In words, pick node i with smallest $d(\cdot)$ from $\bar{S}$, make it permanent, i.e., move it to S. Then run UPDATE(i) by exploring the out arcs of node i.

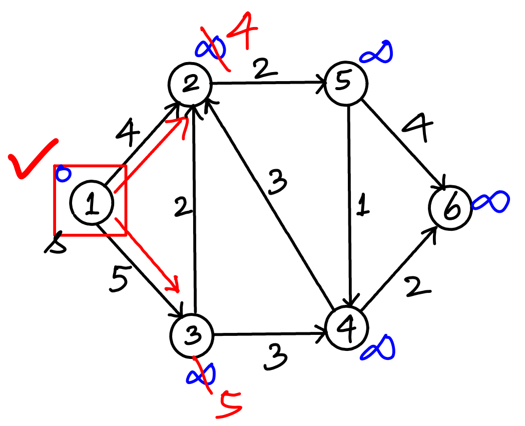
Illustration



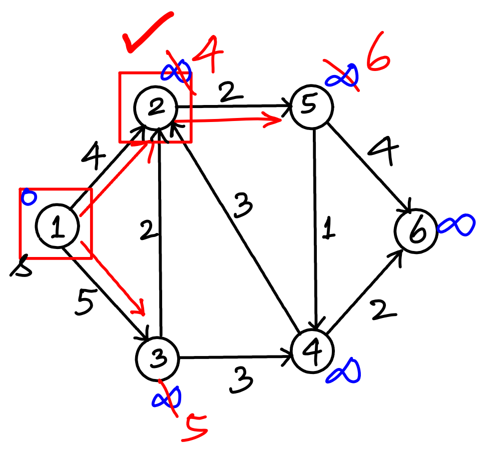
✓ : latest node that's made permanent

$\boxed{i}$: node i is made permanent

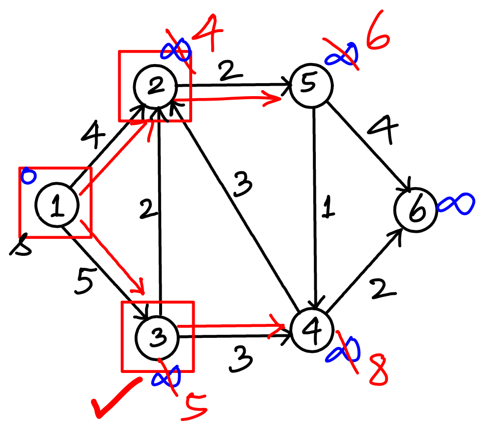




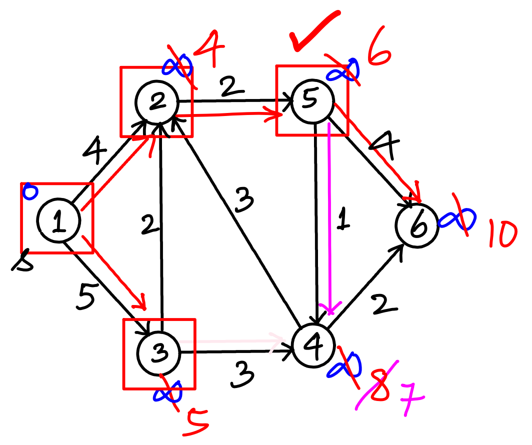
→



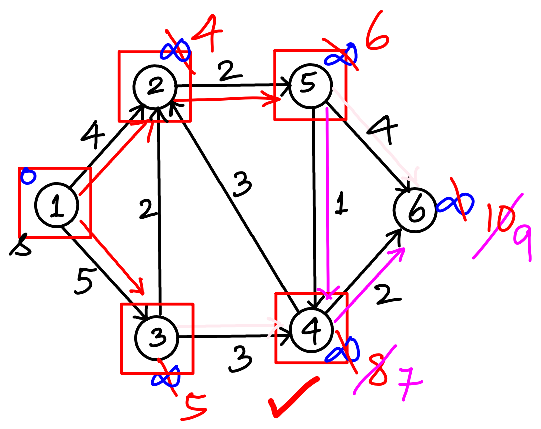
→



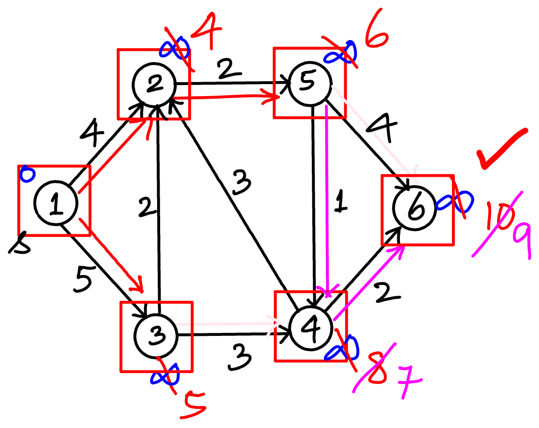
→



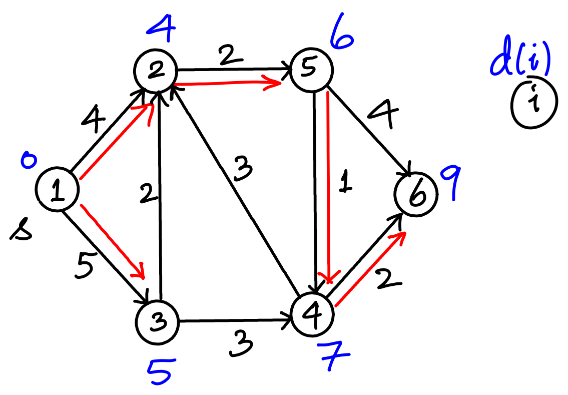
→



→



Here is the shortest path tree:



Notice that the SP tree could be different from the BFS tree.

~~For~~ instance, we could get to 4 from 1 via 3 (1-3-4) in BFS, but we go 1-2-5-4 in the SP tree.