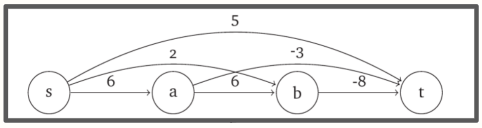
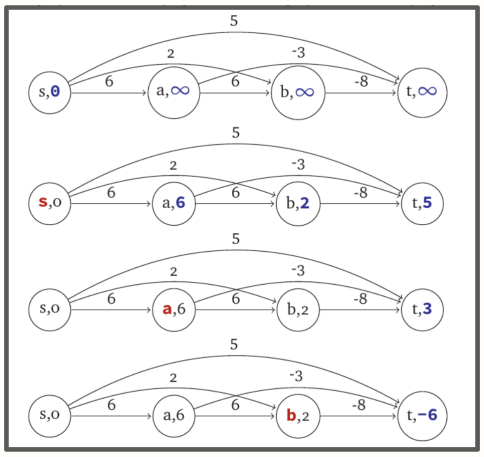
1. Given a directed-acyclic-graph ($G = (V, E)$) with integer (positive or negative) edge weights:

   (a) Give an algorithm to find the **shortest** path from a node s to a node t.

Solution: Because there are negative edge weights we cannot use a greedy method like Dijkstra's algorithm; however because the graph is directed and acyclic we can use a topological sort. Topologically sorting the graph means that every vertex can only reach vertices below them in the sort and cannot reach vertices above them in the sort.



This means after topologically sorting the graph we start at node s and compute the shortest path from node s to each of the nodes below it in sequential order. To do this we initialize the distance from s to all the other nodes as ∞ then we update the distance from node s to be the weight of the edges from s to the neighbors of s. Then we move on to the next sequential node say u, and look at its neighbor, say v. If the distance from s to u plus the edge weight from u to v is less than the current distance from s to v then we update the value. This repeats until we reach node t.

fig 

Let s and t be the nth and mth node in the topological sort and let $N(u)$ be the neighbor set of u. Then the algorithm is

```
DAGSP(V, E, s, t):
  (Vs, Es) ← TopSort(V, E)
  n ← index(s)
  m ← index(t)
  D[n] ← 0
  for k ← n + 1 to m
      D[k] ← ∞
  for i ← n to m − 1
      for v in N(v_i)
          j ← index(v)
          D[j] ← min{D[j], D[i] + Es[i][j]}
  return D[m]
```

A topological sort takes $\boldsymbol{O(V+E)}$ time and the for loops in the algorithm takes $\boldsymbol{O(V+E)}$. So the total running time is $\boldsymbol{O(V+E)}$. ■

(b) Give an algorithm to find the **longest** path from a node s to a node t.

Solution (Direct): To find the longest path from node s to node t we can do the same process as part a) but instead we initialize the values to $-\infty$ then take the maximum value.

```
DAGLP(V, E, s, t):
  (Vs, Es) ← TopSort(V, E)
  n ← index(s)
  m ← index(t)
  D[n] ← 0
  for k ← n + 1 to m
      D[k] ← −∞
  for i ← n to m − 1
      for v in N(v_i)
          j ← index(v)
          D[j] ← max{D[j], D[i] + Es[i][j]}
return D[m]
```

This has a running time of $\boldsymbol{O(V+E)}$.

■

Solution (Reduction): This problem can be reduced to the part a). Multiplying all of the edge weights by -1 then finding the shortest path then multiplying that value by -1 yields the longest path.

```
DAGLP(V, E, s, t):
  Et ← −1 ∗ E
  x ← DAGSP(V, Et, s, t)
  return −1 ∗ x
```

This has a running time of $\boldsymbol{O(V+E)}$.

■

2. Given a directed graph $G = (V, E)$ with non-negative edge lengths $\ell(e), e \in E$ and a node $s \in V$, describe an algorithm to find the length of a shortest cycle containing the node s.

Solution: A shortest cycle containing s must be composed of some path from s to a vertex v, followed by an edge (v, s). Run Dijkstra's algorithm to find the shortest distances $d(s, v)$ from s to each vertex v. Then the shortest cycle containing s is found by $\min_{v \in V} \{d(s, v) + \ell(v, s)\}$. The time taken is just the time to run Dijkstra's algorithm $O(m + n \log n)$ plus the time to take the minimum; to compute $d(s, v) + \ell(v, s)$ we scan the adjacency list of v, the total work for computing all these values and then taking the minimum is $O(m + n)$. Thus the total time is $O(m + n \log n)$. ■

3. Suppose we have a collection of cities and different airlines offer flights between various pairs of cities. Some airlines only fly between some pairs of cities. Some pairs of cities are served by many airlines. Each airline charges perhaps different amounts for their one-way tickets.

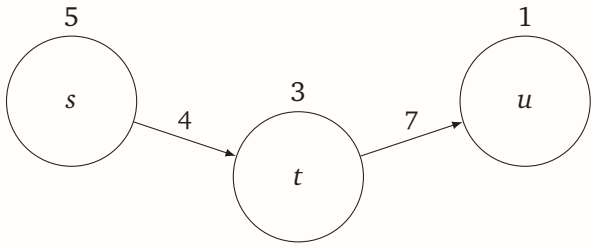
   - Suppose you'd like to get from City A to City B at the least total cost. Describe an efficient solution. (Your solution may change planes to a different airline as needed.)
   - It turns out that airports charge usage taxes. Different airports may charge different amounts in tax. Your cost of traveling from A to B now includes all of the flight costs, plus all of the taxes of the airports that you stopover along the way from A to B. Model this as a graph problem and give an efficient solution to find the least cost way to get from A to B.

Solution: Part (a) is immediately seen to be an application of shortest path, where the graph $G = (V, E)$ is given by $V =$ the set of airports, and $E = \{(u, v) \mid$ some airline flies directly from city u to city $v\}$. But what is the length $\ell(u, v)$ of edge (u, v)? We could include multi edges in which for each airline flying direclty from u to v we include an edge with length the cost of the airfare, or simpler, we use a single edge with length $\ell(u, v)$ defined to be the minimum fare offered by any airline that flies directly from u to v.

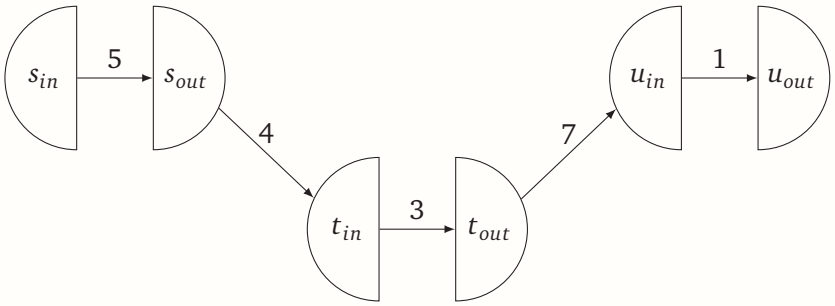
Finding a shortest path from city A to city B in G thus gives us the minimum airfare from A to B in the stated problem.

Part (b) is a bit trickier because of the airport taxes. We employ a trick typically used when a graph has costs associated with vertices as well as costs on edges: we split each vertex into two pieces, and insert an edge between them with weight equal to the cost of the vertex. Each vertex will have an "in" version, and an "out" version, with all incoming edges directed to the "in" version, and all outgoing edges emanating from the "out" version. Refer to the figures below.

Original graph with vertex weights:



New graph with only edge weights:



The solution is then to create the graph as in part (a), but then split nodes as described and insert edges with lengths corresponding to the airpor taxes. The minimum cost path from A_{in} to B_{out} will give the least cost of traveling from A to B, paying all airport taxes, including those at airports A and B. (What if taxes at airports A and B were not required to be paid?)

Formally, the graph we create is $G = (V, E)$, where V and E and the edge weights are defined as follows:

- $V = V_{in} \cup V_{out}$, where $V_{in} = \{v_{in} \mid v \text{ is a city}\}$, and $V_{out} = \{v_{out} \mid v \text{ is a city}\}$.
- $E = E_1 \cup E_2$, where $E_1 = \{(u_{out}, v_{in}) \mid \text{there is an airline that flies from city } u \text{ to city } v\}$, and $E_2 = \{(v_{in}, v_{out}) \mid v \text{ is a city}\}$.
- If $e = (u_{out}, v_{in}) \in E_1$, then $\ell(e) =$ the least cost airfare for flying directly from city u to city v.
- If $e = (v_{in}, v_{out}) \in E_2$, then $\ell(e) =$ the airport tax for the airport in city v.

The time to create the graph G is linear in the number of cities and number of flights specified, and the running time of the algorithm is thus dominated by running the shortest path algorithm which takes time $O(m + n \log n)$, where m is the number of flights, and n the number of cities. ■

4. There are n galaxies connected by m intergalactic teleport-ways. Each teleport-way joins two galaxies and can be traversed in both directions. However, the company that runs the teleport-ways has established an extremely lucrative cost structure: Anyone can teleport *further* from their home galaxy at no cost whatsoever, but teleporting *toward* their home galaxy is prohibitively expensive.

   Judy has decided to take a sabbatical tour of the universe by visiting as many galaxies as possible, starting at her home galaxy. To save on travel expenses, she wants to teleport away from her home galaxy at every step, except for the very last teleport home.

   Describe and analyze an algorithm to compute the maximum number of galaxies that Judy can visit. Your input consists of an undirected graph G with n vertices and m edges describing the teleport-way network, an integer $1 \le s \le n$ identifying Judy's home galaxy, and an array $D[1..n]$ containing the distances of each galaxy from s.

Solution: We reduce this problem to finding the length of the longest path in a dag $G = (V, E)$ as follows:

- V is the set of n galaxies, plus an artificial target node t. Let s denote Judy's home galaxy.
- E contains a directed edge $u \to v$ if either of the following conditions is satisfied
  - There is a teleport-way between galaxy u and galaxy v, and v is farther from s than u.
  - $v = t$ and there is a teleportway between galaxy u and s.
- We need to compute the length of the longest path in G from s to t.
- We can compute this length using dynamic programming as described in Tuesday's lecture (and in the lecture notes).
- The algorithm runs in $O(V + E) = \boldsymbol{O(n + m)}$ ***time.*** ■

To think about later: You probably heard of the phrase "six degrees of separation" and the "small world" phenomenon; see https://en.wikipedia.org/wiki/Six_degrees_of_separation. The idea is that in many interesting networks people or objects are within a small distance of each other. At the same time we believe in "locality" in that each person may only a small number of people compared to the total population. The next two problems explore the tradeoffs between diameter and degree in a graph to explore this in a more quantitative fashion.

5. Suppose G is a graph with maximum degree d. The diameter of the graph is $\max_{u,v} \text{dist}(u, v)$. Prove that the diameter of the graph is $\Omega(\log_d n)$ where n is the number of nodes. It is easier to consider $d = 5$ or some other small constant for simplicity. *Hint:* Consider the BFS layers starting at any vertex v.

   The point of the problem is to show that if all degrees are small then the diameter must grow with the number of nodes.

   Solution: Suppose the maximum degree is d. Let the diameter be $D = d(s, t)$ for some pair of nodes s and t. Then the number of nodes that can be in layer 1 of a BFS search starting at s (distance 1 away from s) is, by definition of degree, at most d. The number of nodes that can then be at layer 2 is at most d^2, since each node at layer 1 is connected to at most d other nodes. In general, the number of nodes at layer d is at most d^i. Since the diameter is D, the total number of nodes summing across all of the D layers in a BFS starting from s is at most $\sum_{i=0}^{D} d^i \leq d^{D+1}$. But if there are n nodes, this means $n \leq d^{D+1}$, and taking logs base d, we have $\log_d n \leq D + 1$, so diameter $D = \Omega(\log_d n)$ as was to be shown. ■

6. Suppose the diameter of an undirected simple graph is d. Prove that there is a node with degree at most $3n/d$. *Hint:* Consider the BFS layers for the pair defining the diameter. It is easier to prove a bound such as $9n/d$.

This problem is to show you that if the diameter is small then there must be a large degree node.

Solution: Note that the graph is connected if it has finite diameter. If the diameter is d then there is a pair of nodes s, t such that $d(s,t) = d$. Consider the BFS layers starting from s. Hence we have $L_0 = \{s\}$, $L_1, L_2, \ldots, L_d$ where $t \in L_d$. Consider any node u in a layer L_i where $1 \leq i \leq d-1$. From the properties of the BFS layers u can be connected only to nodes in L_{i-1}, L_i, L_{i+1}. Thus $deg(u) \leq |L_{i-1}| + |L_i| + |L_{i+1}|$. For $i = 0$ we can say that $deg(s) \leq |L_1|$. For $i = d$ we can say that $deg(t) \leq |L_{d-1}| + |L_d|$. Suppose the claim is false and hence $deg(v) > 3n/d$ for every $v \in V$. For $1 \leq i \leq d-1$ pick an arbitrary node u_i from L_i (each layer is non-empty so we can do this). We have $deg(u_i) \leq |L_{i-1}| + |L_i| + |L_{i+1}|$ and since we assumed that the claim is false, $3n/d < |L_{i-1}| + |L_i| + |L_{i+1}|$ for each $1 \leq i \leq d-1$. For $i = 0$ and $i = d$ we have $|L_1| > 3n/d$ and $|L_{d-1}| + |L_d| > 3n/d$. Summing these inequalities from $i = 0$ to d we obtain

$$|L_0| + 3(|L_1| + |L_2| + \ldots + |L_{d-1}|) + 2|L_d| > (d+1) \cdot 3n/d > 3n.$$

The above implies that

$$|L_0| + |L_1| + \ldots + |L_d| > n$$

which is a contradiction since the every node is in exactly one of the layers. Thus there must be a node with degree at most $3n/d$.

■

A more elegant solution.

Solution: Let $P = v_0, v_1, \ldots, v_d$ be the diameter-defining path of the graph. Here, unless $|i-j| = 1$, there cannot be an edge joining v_i and v_j, for otherwise the distance between v_0 and v_d would be smaller than d. In addition, any vertex u not in P can have at most three neighbors in P for the same reason.

Therefore, the sum of degrees of the vertices in P is

$$\sum_{i=0}^{d} \deg(v_i) \leq 2d + 3(n-d-1) = 3n-d-3$$

By the Pigeonhole Principle, there is a node in P with degree at most

$$\frac{1}{d+1} \sum_{i=0}^{d} \deg(v_i) \leq \frac{3n-d-3}{d+1} \leq \frac{3n}{d}$$

■