CSE 589 -- Lecture 2

The computer is useless
It can only answer questions.
-- Pablo Picasso

Administrivia
New and improved grading scheme:

- **Weekly problem sets 60%**
  - You can work together, but follow Gilligan's Island rule: write it up by yourself, and indicate on your homework who you worked with.
- One-time substitution option for 2 lowest scores on problem sets, to be negotiated with TAs.
  - JAVA applet illustrating how a major algorithm works.
  - Must be gradable without TAs needing to read code.
- Other extra credit options:
  - Transcribing a lecture. Also serves as substitute for associated problem set.
  - Short in-class presentation. Limited options, must let me know by October 15.
- **Final 40%**

Homework Turn-in

Plan for Today

- More dynamic programming?
- More graph algorithms.
  - Minimum spanning tree
    - priority queues
    - union/find
  - Shortest paths
  - Maximum flow

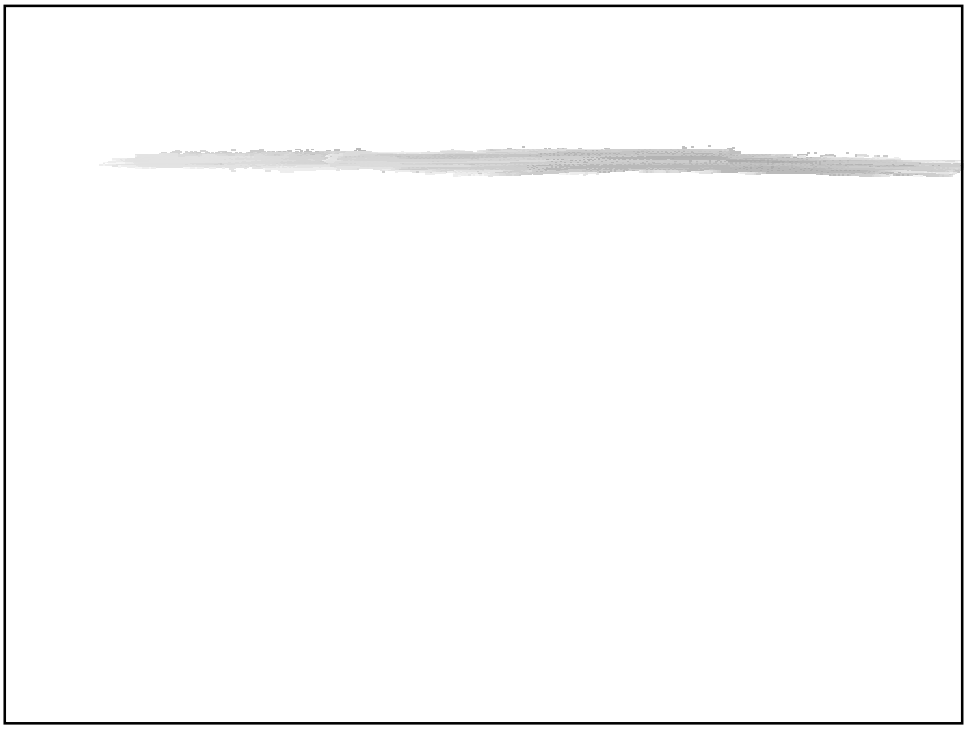
Dynamic Programming

- **characterize structure of an optimal solution**
- **recursively define value of an optimal solution**
- **compute the value of an optimal solution in bottom-up fashion**
- **construct an optimal solution from computed information**

Example DP Problem: Matrix Chain Multiplication

- Given a sequence $A_1, A_2, \ldots, A_n$, where matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product in a way that minimizes the number of scalar operations.

```
Matrix-Multiply(A,B)
  for i = 1 to rows [A]
    for j = 1 to columns [B]
      C[i,j] = 0;
      for k = 1 to columns [A]
        C[i,j] = C[i,j] + A[i,k]*B[k,j]
```



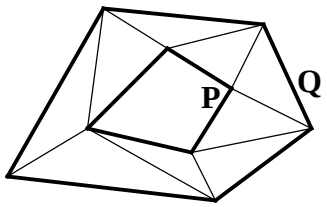
Dynamic Program for Matrix Chain Multiplication

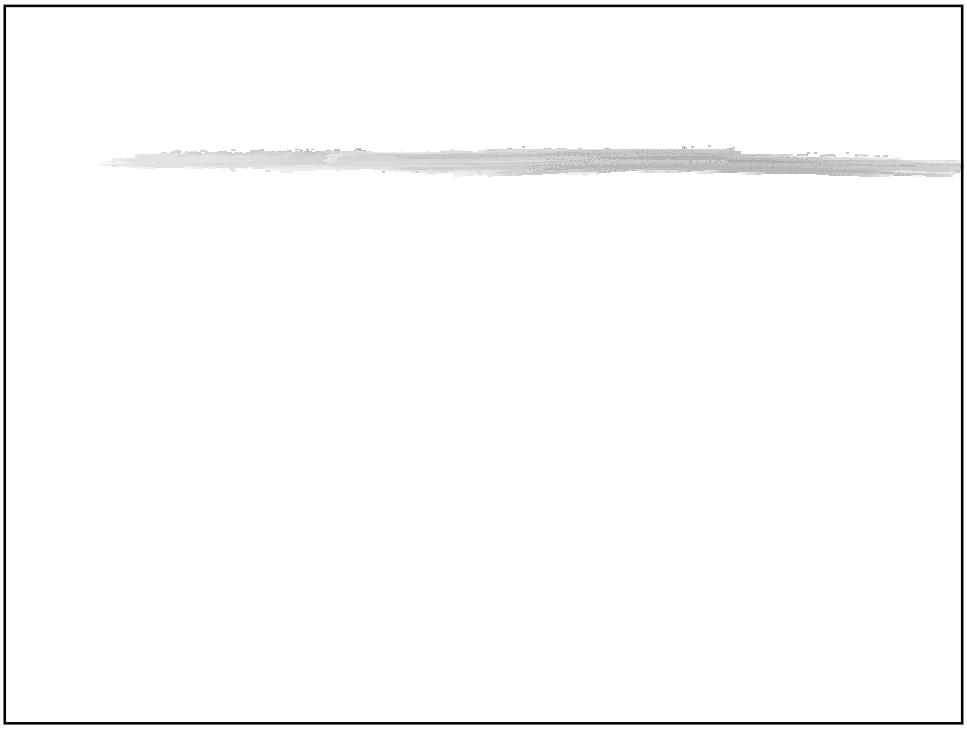
- m[i,j] = minimum number of scalar multiplications needed to compute $A_{i..j}$
- Solution =
- m[i,j] =

Homework problem
Surface Reconstruction

Let P and Q be polygons with m and n vertices, $p_0,\ldots,p_m$ and $q_0,\ldots,q_n$ in 3 dimensions. A tiling of P and Q is a set of triangles, called tiles, whose vertices are of the form (p_i, p_{i+1}, q_j), or (p_i, q_j, q_{j+1}), that is, one side of each tile is a face of one polygon and the remaining tile is on the other polygon. The set of tiles constructed must satisfy two properties: (1) Every face of each polygon is a side of exactly one tile. (2) Every cross edge is a side of exactly two tiles.

Give a dynamic programming algorithm, which given polygons P and Q, constructs a tiling of P and Q of minimum surface area.



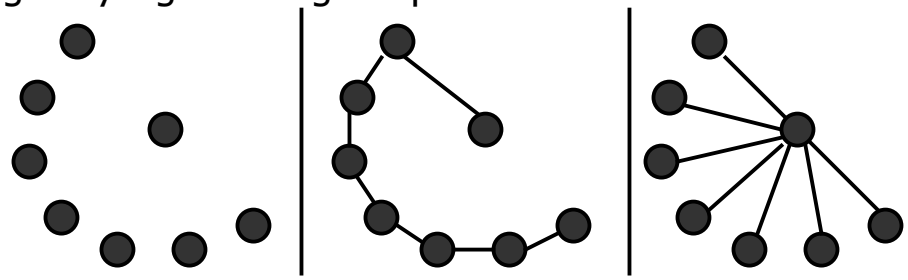


Graph algorithms discussed last time

- Graph traversal
  - Depth-first search
  - Breadth-first search
- Applications of depth-first search
  - Connected components
  - Tree and cycle detection
  - Topological sorting
  - Finding articulation points
  - Strongly connected components
  - Planarity testing

New Topic
Minimum Spanning Trees

- Spanning tree of G -- subgraph of G that has same set of vertices as G and is a tree.
- MST of weighted graph G -- spanning tree of G whose edges sum to minimum weight
- greedy algorithms give optimal solution

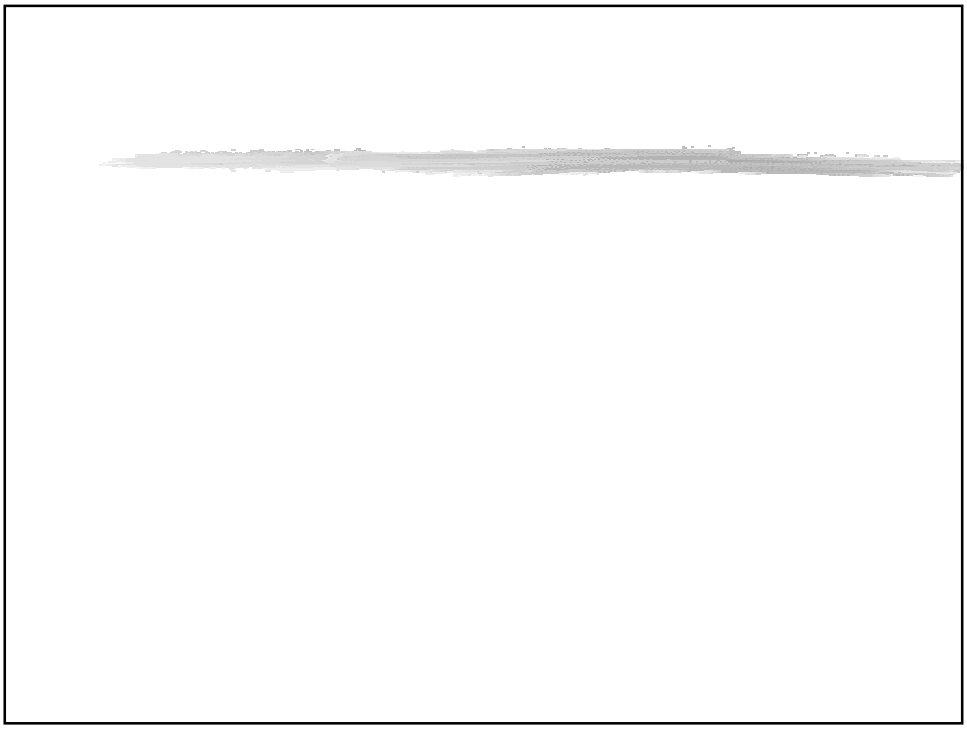


Prim's algorithm

Start with arbitrary vertex

While there are vertices in tree connected to vertices outside
    select minimum weight edge between tree and outside
    add selected edge and vertex to tree

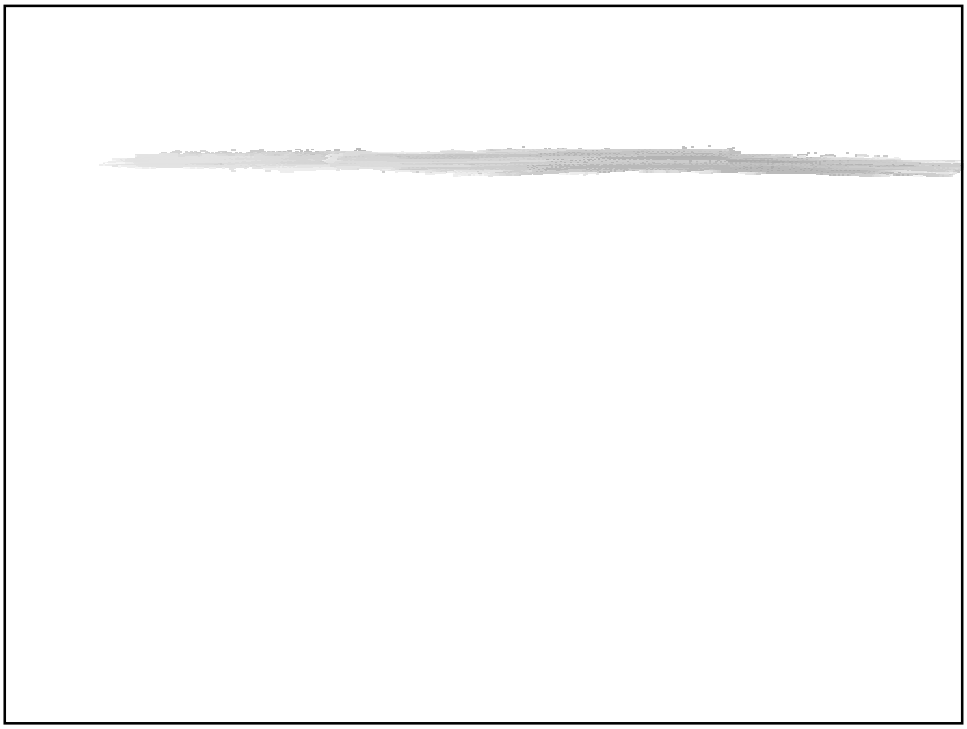
- Correctness ?
- Running Time -- trivial O(nm)
  Better?



Priority Queue

- Data structure for maintaining a set S of elements, each with an associated key
- Ops supported:
  - Insert(S,x)
  - Minimum(S)
  - Extract-Minimum(S)
- One implementation: heap
  - complete binary tree stored in an array A
  - satisfies heap property: A[*parent* [i]] <= A[i]

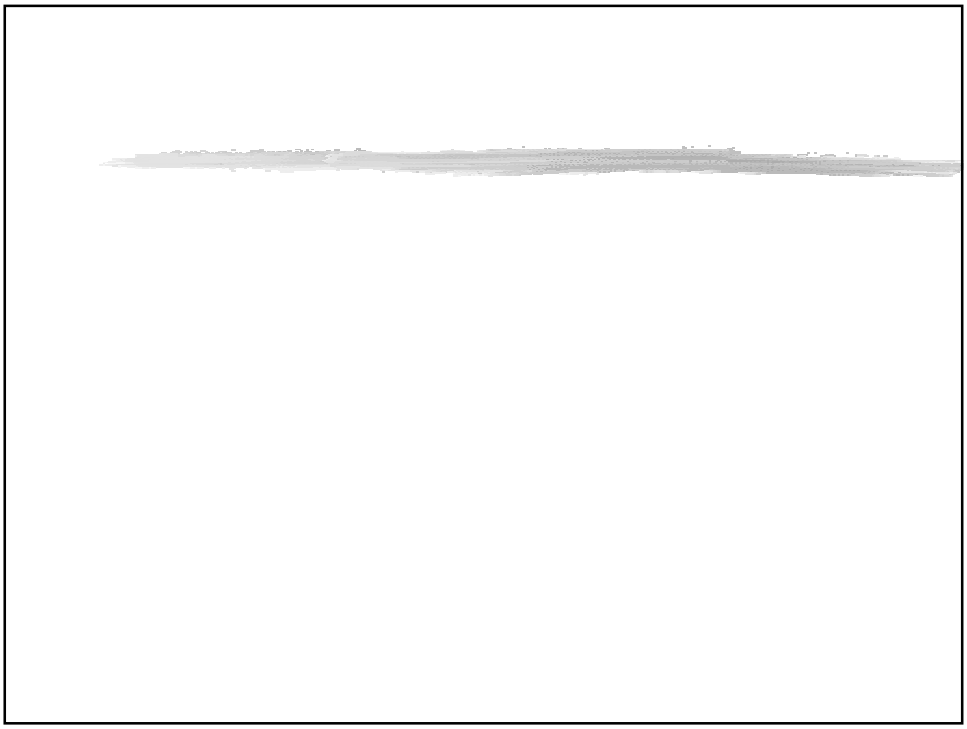
For MST, need Decrease-Key(x) as well.



Kruskal's algorithm

Sort edges in order of increasing weight
count = 0
While (count < n-1)
    get next edge (u,v)
    if u and v are in different components
        add edge to tree, and merge components

- Correctness ??
- Running Time -- trivial O(nm)
  Better?



Disjoint Set Data Structure Union/Find

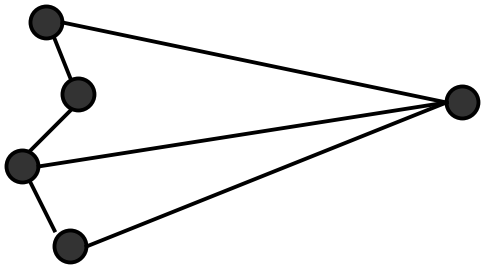
- Data structure for maintaining a set of disjoint dynamic sets, each with representative
- Ops supported:
  - Make-Set(x)
  - Union(x,y)
  - Find-Set(x)
- One implementation:

  with union-by-rank and path compression, running time for m disjoint-set operations on n sets is

  $O(m\ \alpha(m,n))$ $\alpha(m,n)$ = inverse Ackermann f'n

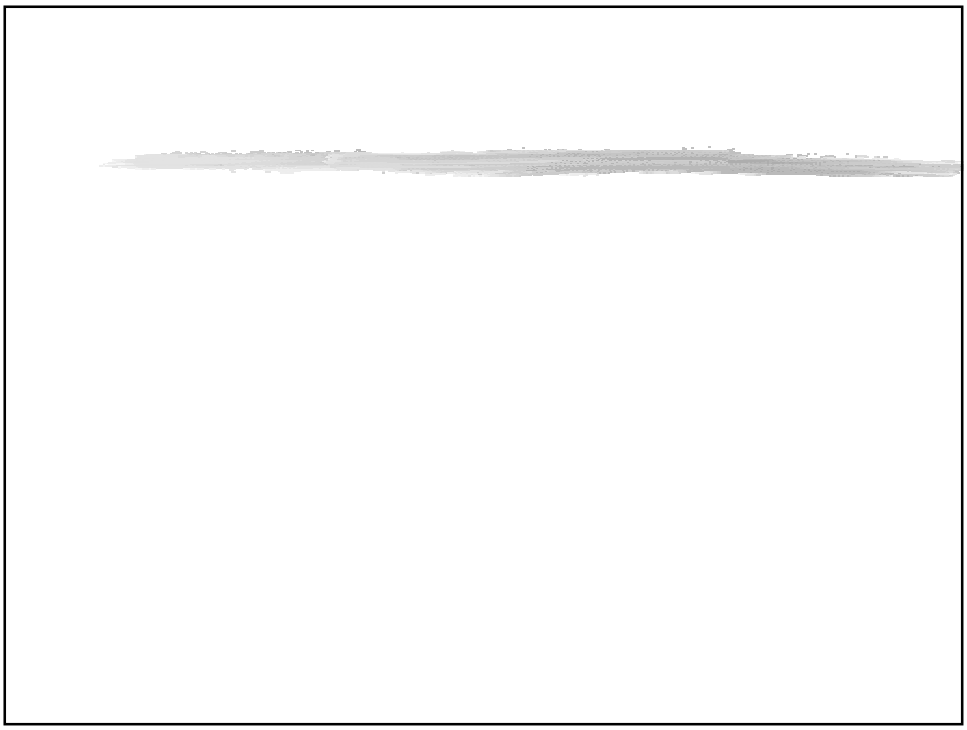
New Topic: Shortest Paths

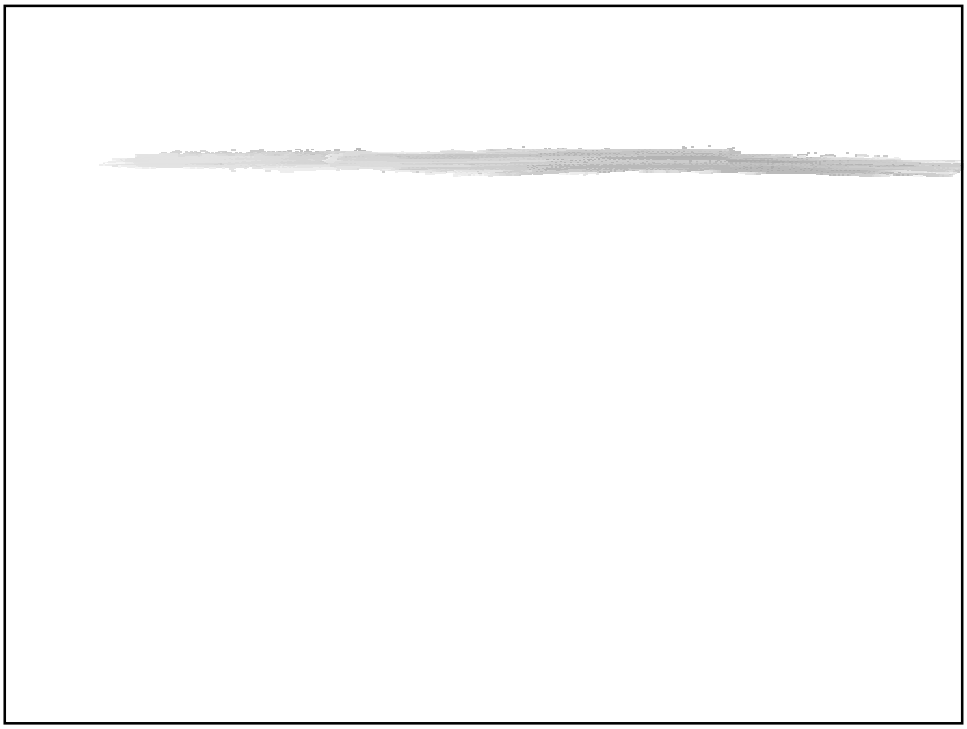
- Given weighted graph, and specified vertex v, find shortest path from v to each other vertex.
- Example: setting up routing tables in Internet.



Dijkstra's algorithm

- For case where all edge weights are non-negative
- Finds shortest path from given vertex v to others
- At the beginning of an iteration, have a set of vertices S to which know the shortest path from v => grow set by one.
- Dynamic programming-like strategy



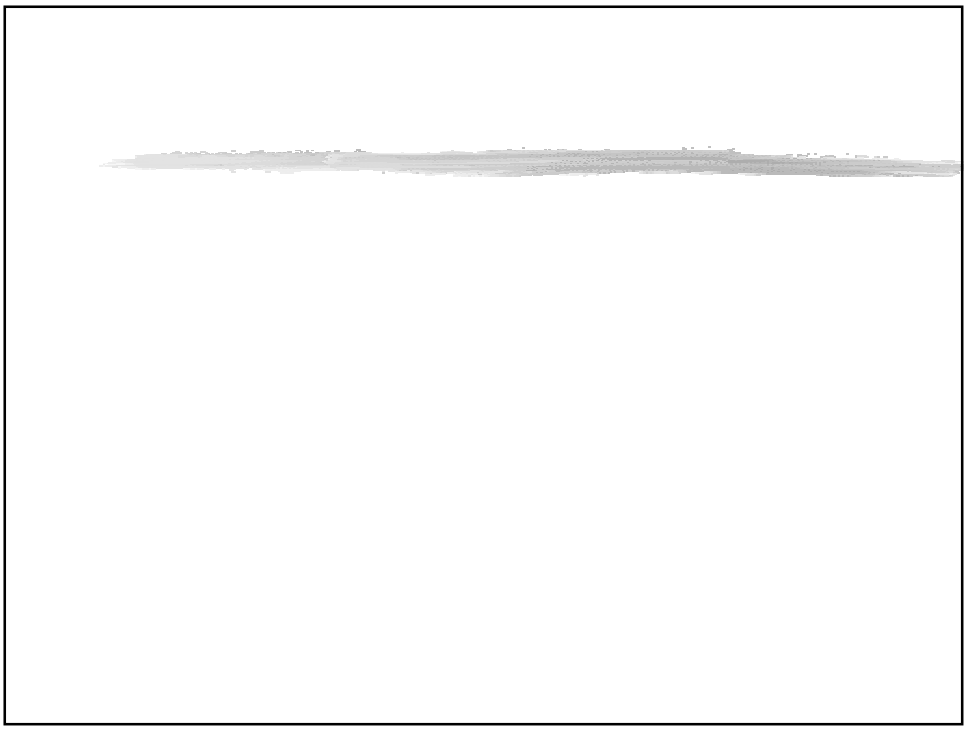


Shortest paths with negative weights

- Bellman-Ford algorithm.
- If algorithm has negative cycle, shortest path undefined.
- Algorithm either returns negative cycle or shortest paths.

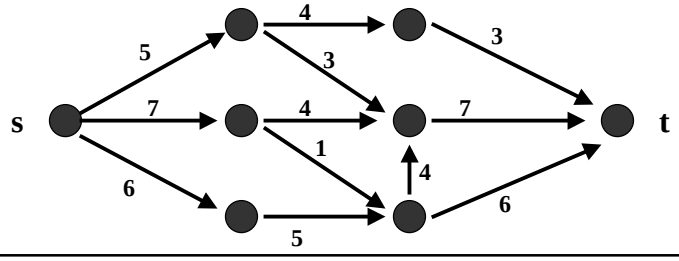
All-pairs shortest paths Floyd-Warshall

- Slick dynamic programming algorithm
- $D[i,j]^k$ = length of shortest path from i to j using only vertices numbered 1,..,k as intermediate vertices



New Topic: Maximum Flow

- Input description: a graph (network) G where each edge (v,w) has associated capacity c(v,w), and a specified source node s and sink node t
- Problem description: What is the maximum flow you can route from s to t while respecting the capacity constraint of each edge?



Max-flow outline:

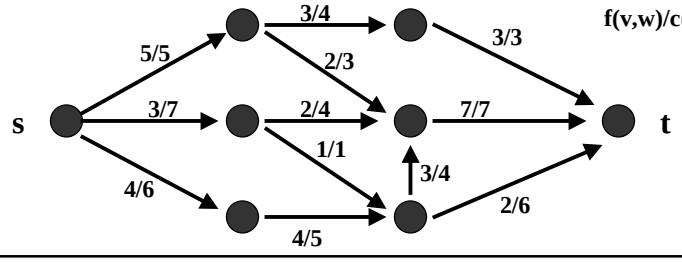
- properties of flow
- augmenting paths
- max-flow min-cut theorem
- Ford-Fulkerson method
- Edmonds-Karp method
- preflow-push methods
- applications

Properties of Flow: f(v,w) -- flow on edge (v,w)

- 0<= f(v,w) <= c(v,w) : the flow through an edge cannot exceed the capacity of an edge
- for all v except s,t Σ_u f(u,v) = Σ_w f(v,w): the total flow entering a vertex is equal to total flow exiting vertex
- total flow leaving s = total flow entering t.

Not a maximum flow!

Notation on edges f(v,w)/c(v,w)

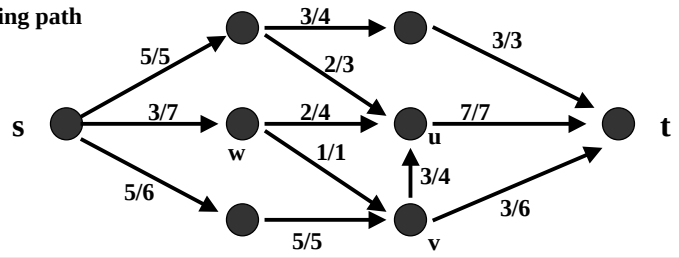


An augmenting path with respect to a given flow f is a

Directed path from s to t which consists of edges from G, but **not necessarily in same direction**. Each edge on path is either:

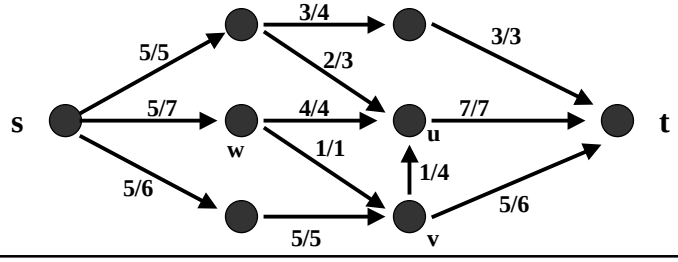
- forward edge: (u,v) in same direction as G and f(u,v) < c(u,v). (c(u,v)-f(u,v) called slack) => has room for more flow.
- backward edge: (u,v) in opposite direction in G (i.e., (v,u) in E) and f(u,v) >0 => can borrow flow from such an edge.

augmenting path s-w-u-v-t



Using an augmenting path to increase flow

- If all edges are forward => move flow through all of them. amount of flow can push = minimum slack
- Otherwise, push flow forward on forward edges, deduct flow from backward edges. Amount of flow can push =



Augmenting Path Theorem:

A flow f is maximum iff it admits no augmenting path

- Already saw that if flow admits an augmenting path, then it is not maximum.
- Suppose f admits no augmenting path. Need to show f maximum.

Cut -- a set of edges that separate s from t.

Capacity of a cut = sum of capacities of edges in cut.

Prove theorem by showing that there is a cut whose capacity = f.

A = vertices s.t. for each v in A, there is augmenting path from s to v. => defines a cut.

Claim: for all edges in cut, f(v,w)=c(v,w)

=> value of flow = capacity of cut defined by A => maximum.

=> Celebrated Max-flow Min-Cut Theorem

- The value of a maximum flow in a network is equal to the minimum capacity of a cut.

- The Integral Flow Theorem

  If the capacities of all edges in the network are integers, then there is a maximum flow whose value is an integer.

Residual Graph w.r.t. flow f

- R=(V,E')

an edge (v,w) in E' if either

- it is forward edge
  - capacity = c(v,w)-f(v,w)
- or it is a backward edge
  - capacity = f(v,w).

- An augmenting path = a regular directed path from s to t in the residual graph.

Ford-Fulkerson Method (G,s,t)

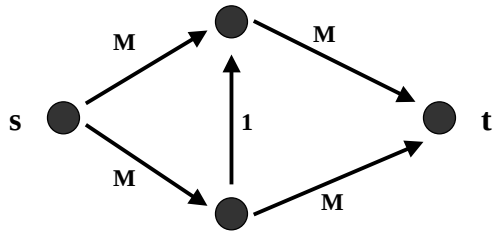
Initialize flow on all edges to 0.

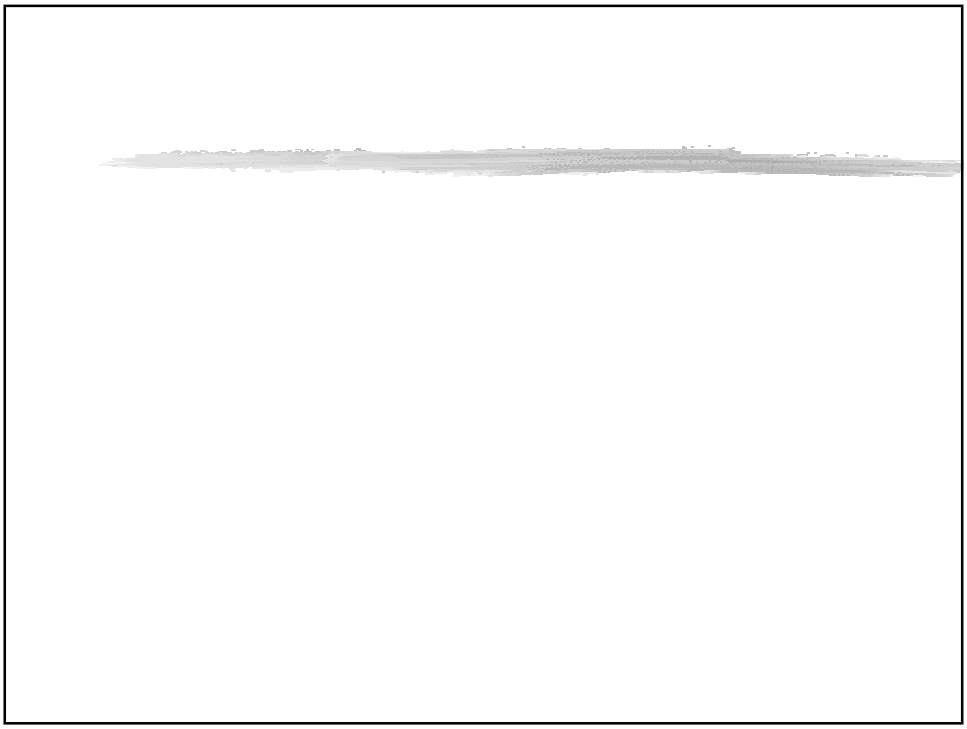
While there is a path p from s to t in residual network R

δ = minimum capacity along p in R

augment δ units of flow along p and update R

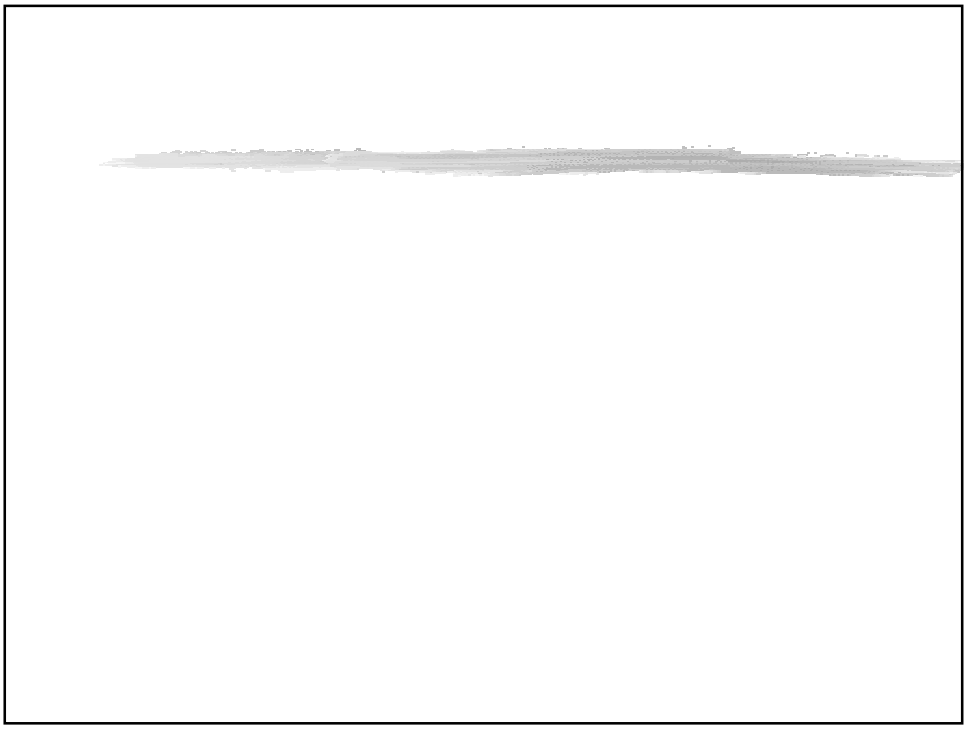
Running Time?





Edmonds-Karp

- If implement computation of augmenting path using breadth-first search, i.e., if augmenting path is a **shortest path** from s to t in residual network, then the number of augmentations O(|V||E|).
  - The shortest-path distance from s to v in R is non-decreasing.
  - The number of times an edge (u,v) can become **critical**, i.e., residual capacity of (u,v) = residual capacity of augmenting path, is O(|V|).

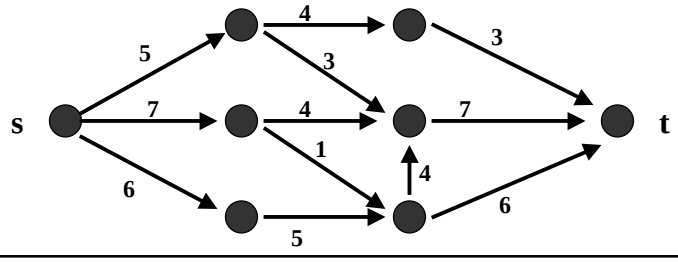


Fastest max-flow algorithms: preflow-push

- work on one vertex at a time, looking only at neighbors in R.
- maintain preflow: flow except that there may be excess flow into u.
- imagine vertices as pipe junctions. each vertex has outflow pipe leading to arbitrarily large reservoir that can accumulate fluid.
- each vertex, its reservoir, and all its pipe connections are on a platform whose height increases as the algorithm progresses.
- can only push flow downhill.
- height of s fixed at |V|, height of t fixed at 0. others start at 0 and increase with time.

Preflow-push algorithms

- if only pipes leaving u that aren't saturated with flow connect to vertices that are on the same level as u or are uphill from u. => **lift** u : increase its height to one more than height of lowest neighbor to which it has unsaturated pipe.
- eventually all flow that can get through has arrived => convert preflow to "legal" flow by sending excess collected in reservoirs of overflowing vertices back to source.

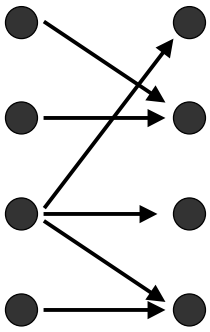


Some applications of max-flow and max-flow min-cut theorem

- Bipartite matching
- Network connectivity
- Scheduling on uniform parallel machines
- Shipping

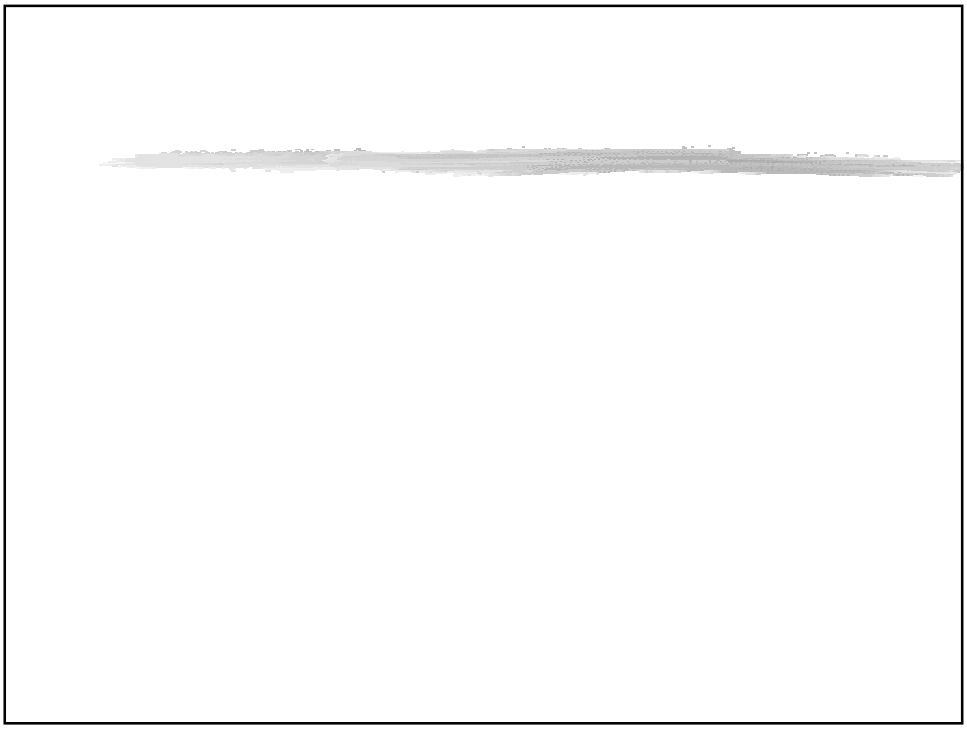
Bipartite Matching

- Input: a bipartite graph G=(V,E)
- Output: Largest-size set of edges M from E such that each vertex in V is incident to at most one edge of S



Network Connectivity

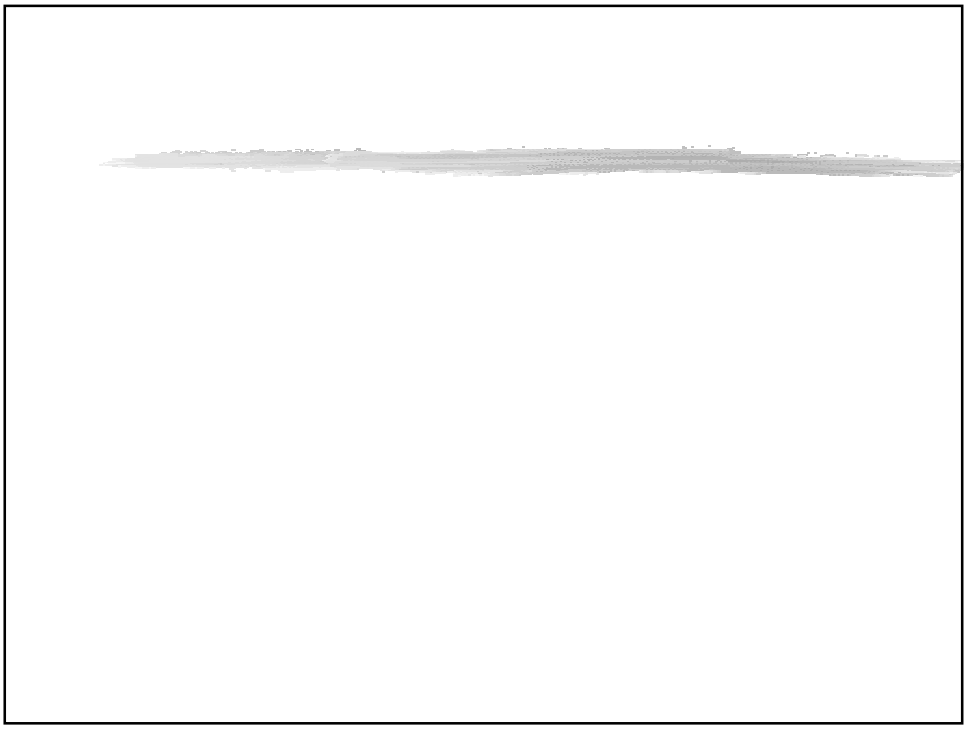
- What is the minimum number of links in the network such that if that many links go down, it is possible for nodes s and t to become disconnected?
- What is the minimum number of nodes in the network such that if that many nodes go down, it is possible for nodes s and t to become disconnected?



Scheduling on uniform parallel machines

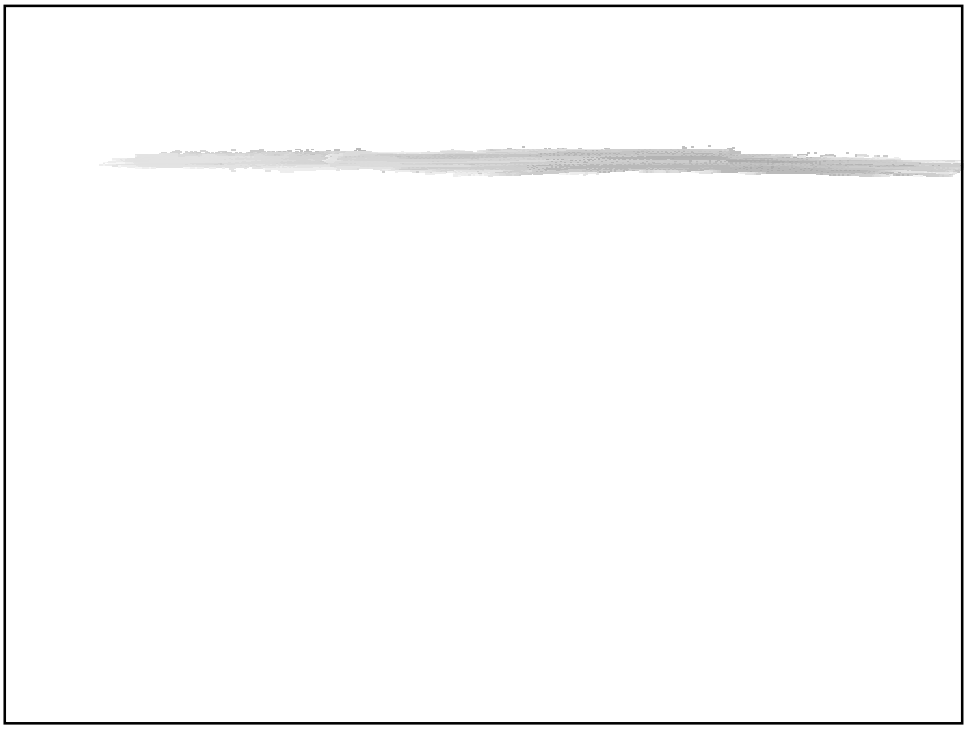
- Have a set J of jobs, M uniform parallel machines.
- Each job j has processing requirement p_j, release date r_j and due date d_j.
- A machine can work on only one job at a time and each job can be processed by at most one machine at a time, but preemptions are allowed.

Scheduling problem: determine a feasible schedule that completes all jobs before their due dates or show that no such schedule exists.



Shipping Problem

- A company has to deliver orders between certain origin-destination pairs. Each order has a delivery date. The company wants to determine the minimum number of shipments needed to to meet the delivery dates for the orders.
- M locations, $T_{i,j}$ = travel time between location i and j
- n orders
  - s_i = source location for order i
  - t_i = destination location for order i
  - d_i = delivery date for order i



Problems

1. Speech recognition problem -- distinguishing between words that sound alike (e.g. to, two, too). You're given a list of words and the likelihood of transitions between them. For each word spoken into the microphone, your program determines the set of words that sound like it. How would you figure out what sentence was spoken?
2. In an optical character recognition system, a method for separating lines of text is needed. Although there is some white space between the lines, problems like noise and the tilt of the page make the separation difficult to find. How would you go about doing line segmentation?

Problems

3. How would you design natural routes for a video-game character to follow through an obstacle-filled room?
4. Video-on-demand servers --You have a set of storage devices, each storing some number of movies. Suppose the movies are replicated to a limited extent across the storage devices, and that each storage device has a certain bandwidth (number of movie requests it can serve per unit time) . Given a set of requests, i.e., a set of movies clients wish to watch, how would you go about figuring out how to satisfy a maximum number of requests at the same time?