• Homework 6 won't be posted until Friday
• A graph is **Eulerian** if it has an Eulerian circuit
• An **Eulerian circuit** is a walk that uses each edge once and ends where it starts
• A graph is Eulerian iff it is connected and all vertices have even degree
• **Proof:**
  • In an Eulerian circuit, every time you enter a vertex via an edge you must leave via another untraversed edge
  • Consider max length walk that doesn’t reuse edges starting at $v_s$ and ending at $v_e$:
    • Since walk is max length, $v_s = v_e$
    • If not every edge is used along the walk, then there is a vertex outside the walk connected to a vertex in the walk, so we can add it and thus contradict our premise
Consider a rectangle $R$ in the plane with corners $(0, 0)$ and $(a, b)$ with $a$, $b$ positive real numbers. Assume $R$ can be tiled with rectangles (may be all distinct) such that each tile has at least 1 integer side. Prove $R$ has an integer side.
• Create a graph using tile corners as vertices
  • Denote each tile as an H-tile or a V-tile if its horizontal or vertical edge is an integer (if both, just choose one label)
• For each H-tile, draw an edge along its horizontal sides, and do the same for the vertical edges of the V-tiles
  • Each vertex is either touching 2 or 4 rectangles, unless it is an outer corner touching only one.
• The resulting graph has all even degree, except the outer corners, and hence there is a path from one outer corner to another.
• Trees are undirected acyclic connected graphs

• **Definition**: the least common ancestor (LCA) of two nodes is the deepest (w.r.t the root) ancestor of both nodes

• Algorithm for finding LCA:
  1. Build a list of nodes in the graph using depth first traversal
  2. Add a node to the list every time it is visited
  3. For each node, also store its depth **for each list entry**
  4. To find the LCA, find the range minimum on the list of depths between two nodes
     • Each query is $O(\lg n)$ with segment trees
     • Since the list is static, you can precompute $O(n^2)$ and query $O(1)$
### Example

```
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>depth</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>visit</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>first</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>13</td>
<td>X</td>
</tr>
</tbody>
</table>
```
Exercise: give an algorithm for finding single-source shortest path on a tree?

Exercise: give an algorithm for finding all-pairs shortest path on a tree?

Exercise: give an algorithm for finding the length of the shortest path between two nodes on a tree?
A computing center has ten different computers (numbered 0 to 9) on which applications can run. The computers are not multi-tasking, so each machine can run only one application at any time. There are 26 applications, named A to Z. Whether an application can run on a particular computer can be found in a job description (see below).

Every morning, the users bring in their applications for that day. It is possible that two users bring in the same application; in that case two different, independent computers will be allocated for that application.

A clerk collects the applications, and for each different application he makes a list of computers on which the application could run. Then, he assigns each application to a computer. Remember: the computers are not multi-tasking, so each computer must handle at most one application in total. (An application takes a day to complete, so that sequencing i.e. one application after another on the same machine is not possible.)

Output a possible matching between computers and applications, or ! if such a matching is not possible.
• **Problem take-aways**
  • A number of apps are to be run on a number of computers
  • Apps take a day
    • No multitasking on the computer
  • Certain computers are set up to run certain apps
  • Two or more of the same app may be run
  • What is a possible allocation of apps → computers so that all jobs run in one day?
Exercise is a classic max flow problem

Given a “network” graph
  • Connected, weighted, directed
  • Edges act as pipes, weighted by capacity of the pipe
  • Vertices act as splitting points of the pipes
  • Two special nodes: source $s$ and sink $t$

Given a network graph, what is the maximum flow from the source to the sink?
  • How much water can travel through the pipes without bursting?

Two methods we'll talk about today to solve this problem
  • Ford-Fulkerson and Edmonds-Karp
• Ford-Fulkerson
  • Send a flow down a path $p$ whenever there exists an augmenting path $p$ from $s$ to $t$
    • An augmenting path is any path that has capacity
  • Find augmenting paths by using DFS
• Algorithm
  1. $mf \leftarrow 0$
  2. while (exists an augmenting path $p$ from $s$ to $t$)
     1. Send flow along $p = s \rightarrow \ldots \rightarrow i \rightarrow j \rightarrow \ldots \rightarrow t$
     2. Find $f$, the minimum edge weight along $p$
     3. Decrease weight of forward edges $i \rightarrow j$ by $f$
     4. Increase weight of backward edges $j \rightarrow i$ by $f$
     5. $mf \leftarrow mf + f$
  3. Output $mf$
• Decreasing the capacity of the forward edge is obvious.
• Why increase the capacity of the backward edge?
Ford Fulkerson: Example

Take the original graph, make it directed

Find an augmenting path

Reduce capacity on path
Find an augmenting path

Reduce capacity on path

Find an augmenting path

Reduce capacity on path

Done!
• Running time: $O(E \times mf)$
  • So if the max flow is large, the running time could be large!
• Degenerate case:
• How to apply max flow to this problem
  • Treat each application as a vertex
  • Treat each computer as a vertex
  • Draw an edge from each application and computer of capacity 1
  • Create a source node connecting to each application of capacity # batch jobs
  • Create a sink node connecting from all computers of capacity 1
Reconstructing the solution:
- Look for edges in the **residual graph** with weight 0: the two connected vertices are the match

Runtime:
- Ford-Fulkerson
  - Resultant value is at most 10
  - Edges:
    - Between source and applications (26)
    - Between applications and computers (26 * 10)
    - Between computers and sink (10)
    - Total 296
  - $O(E \times mf)$, seems very reasonable
Competitive Programming 4.1-4.6

More solutions for the tiling rectangles problem:
http://www.inference.phy.cam.ac.uk/mackay/rectangles/