• Homework 6 due Friday at 2pm
• Homework 7 released tomorrow evening
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 \( \rightarrow \) 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?
Take the original graph, make it directed

mf = 0

Find an augmenting path

Reduce capacity on path

mf = 0

mf = 25
Ford Fulkerson: Example

1. Find an augmenting path and update the flow.
2. Reduce capacity on the path.
3. Repeat steps 1 and 2 until no more augmenting paths can be found.

Example:
- Initial flow: mf = 25
- After reducing capacity: mf = 30
- Find another augmenting path and update flow.
- Reduce capacity on the new path.
- Repeat until no more paths can be found.
- Final flow: mf = 60

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:
      o Between source and applications (26)
      o Between applications and computers (26 * 10)
      o Between computers and sink (10)
      o Total 296
    • O(E * mf), seems very reasonable
• **Edmonds-Karp algorithm**
  • Use BFS to find an augmenting path
  • Runtime: $O(VE^2)$
    • Provable that after $O(VE)$ iterations, all augmenting paths are exhausted
  • Does not run into the same problem as the Ford-Fulkerson degenerate case
    • Recall that capacity $\neq$ weight in the flow graph
    • Each edge is unweighted
Ford-Fulkerson($G, s, t$):
for each edge $(u, v)$ in $E[G]$
  do $f[u, v] \leftarrow 0$, 
  $f[v, u] \leftarrow 0$
while there exists a path $p$ from $s$ to $t$ in the residual network $G_f$
  do $c_f(p) \leftarrow \min[ c_f(u, v) : (u, v) \text{ is in } p ]$
  for each edge $(u, v)$ in $p$
    do $f[u, v] \leftarrow f[u,v] + c_f(p)$
    $f[v, u] \leftarrow -f[u,v]$
void addEdge(int u, int v, int cap, int[][] caps, ArrayList<Integer>[][] adj) {
    if (caps[u][v] == 0 && caps[v][u] == 0) {
        adj[u].add(v); adj[v].add(u);
    }
    caps[u][v] += cap;
}

int maxflow(ArrayList<Integer>[][] adj, int[][] caps, int source, int sink) {
    int ret = 0;
    while (true) {
        int f = augment(adj, caps, source, sink);
        if (f == 0) break;
        ret += f;
    }
    return ret;
}
int augment(ArrayList<Integer>[] adj, int[][] caps, int source, int sink) {
    Queue<Integer> q = new ArrayDeque<Integer>();
    int[] pred = new int[adj.length];
    Arrays.fill(pred, -1);
    int[] f = new int[adj.length];
    pred[source] = source; f[source] = Integer.MAX_VALUE; q.add(source);
    while (!q.isEmpty()) {
        int curr = q.poll(), currf = f[curr];
        if (curr == sink) {
            update(caps, pred, curr, f[curr]);
            return f[curr];
        }
        for (int i = 0; i < adj[curr].size(); ++i) {
            int j = adj[curr].get(i);
            if (pred[j] != -1 || caps[curr][j] == 0) continue;
            pred[j] = curr; f[j] = Math.min(curre, caps[curr][j]); q.add(j);
        }
    }
    return 0;
}
void update(int[][] caps, int[] pred, int curr, int f) {
    int p = pred[curr];
    if (p == curr) return;
    caps[p][curr] -= f;  caps[curr][p] += f;
    update(caps, pred, p, f);
}
• If each edge is also associated with a cost, you may want to find the min-cost max flow
• To do so, replace BFS with Bellman-Ford in Edmonds-Karp
  • Runtime: $O(V^2E^2)$
A consequence of computing the maximum flow is computing the **minimum cut**

Min cut: The smallest cost (or cut set) for removing edges so that the graph is split into two disconnected components
Max-Flow Min-Cut theorem: The maximum flow from $s$ to $t$ is the size of the minimum $s$–$t$ cut. An $s$–$t$ cut is a partition of the nodes into 2 sets $S$, $T$ with $s \in S$ and $t \in T$. The value of the cut is the sum of all capacities on edges from $S$ to $T$.

- Nodes in $S$ are nodes that are reachable from $S$ in the residual graph.
Exercise: Find the min-cut of the following graphs:

a) \[ s \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow t \]

b) 
\[ \begin{array}{c}
10 \\
7 \\
10 \\
\end{array} \quad \begin{array}{c}
1 \\
5 \\
3 \\
\end{array} \quad \begin{array}{c}
1 \\
8 \\
15 \\
\end{array} \quad \begin{array}{c}
4 \\
4 \\
\end{array} \]

c) 
\[ s \rightarrow \quad \rightarrow \quad \rightarrow t \]
(a) The min cut has weight 1.

(b) The min-cut has weight 13.

(c) The min-cut has weight 3. All capacities 1:
Exercise: Suppose you have a problem where vertices have capacities in addition to edges. Explain how to compute the maximum flow.

Exercise: Suppose you are given a graph with edge capacities, and a list of sources and sinks. Each source $s_i$ has an amount of flow $f(s_i)$ that must come out of it. Each sink $k_j$ has an amount of flow $f(k_j)$ that must flow into it. Determine whether these constraints can be satisfied by pushing flow through the network. That is, your answer is boolean.
Exercise:

- Up to 50 cows scattered on a 100x100 field
- Field contains square plots that are of different elevation
- Fields flood every hour to a new level 0-100 over 24 hours
- Cows can move every hour to an adjacent plots whose elevations are higher than level
- Only one cow can be on one plot at the same time
- How many cows can survive?
Exercise: Your statistical model has determined for each pixel the likelihood it should be labeled part of the foreground or background of the image. The pixel $p_i$ has a cost $a_i$ if it is part of the foreground, or a cost $b_i$ if it is part of the background (costs all positive). You have also determined for each pair $(i, j)$ a cost $p_{ij}$ they are placed differently. Give an algorithm to determine an assignment of pixels to the background and foreground that minimizes the cost.
Max-Flow: Competitive Programming 4.6