Checkpoint 2

- 2 pages of notes
- 2/15 through 4/2 (will not include network flow)
- Will make some practice problems soon

Checkpoint 2 topics

- Greedy algorithms
  - Proving correctness
  - Developing algorithms
  - Comparing vs. dynamic programming
- Hash tables
  - Collision resolution by chaining
  - Open addressing
  - Hash functions
- Dynamic programming

Admin

Assignment 9

Checkpoint 2 (DP through graphs... will not include flow networks)

Mentor hour update:
- No more Saturday hours for now
- Additional hours Friday: 5:30-7:30pm
Checkpoints 2 topics

graphs, different types of graphs
- terminology
- representing graphs (adjacency list/matrix)

graph algorithms
- Traversal: BFS, DFS
- MST: Prim’s, Kruskal’s
- Topological sort
- Detecting cycles
- Shortest path algorithms: Floyd-Warshall, Dijkstra’s
- Connectedness: Detecting cycles
- Single source shortest paths: Dijkstra’s, Bellman-Ford
- All pairs shortest paths: Floyd-Warshall, Johnson’s

Notes
- min-cut property (proving correctness of MST algorithms)

Student networking

You decide to create your own computer network:
- You get three of your friends and string some network cables
- Because of capacity (due to cable type, distance, computer, etc.) you can only send a certain amount of data to each person
- If edges denote capacity, what is the maximum throughput you can send from S to T?

You decide to create your own campus network:
- You get three of your friends and string some network cables
- Because of capacity (due to cable type, distance, computer, etc.) you can only send a certain amount of data to each person
- If edges denote capacity, what is the maximum throughput you can send from S to T?

Another flow problem

How much water flow can we continually send from s to t?
Another flow problem

Flow graph/networks

Flow constraints

Max flow problem
Applications?

- network flow
  - water, electricity, sewage, cellular…
  - traffic/transportation capacity
- bipartite matching
- sports elimination

Max flow origins

Rail networks of the Soviet Union in the 1950’s.

The US wanted to know how quickly the Soviet Union could get supplies through its rail network to its satellite states in Eastern Europe.

In addition, the US wanted to know which rails it could destroy most easily to cut off the satellite states from the rest of the Soviet Union.

These two problems are closely related: solving the max flow problem also solves the min cut problem of figuring out the cheapest way to cut off the Soviet Union from its satellites.

Source: Backstrom, The Importance of Algorithms, at www.topcoder.com

Algorithm idea

send some flow down a path
Algorithm idea

Send some flow down a path

Now what?

Algorithm idea

Reroute some of the flow

Total flow?

Algorithm idea

Reroute some of the flow

Algorithm idea
send some flow down a path

reroute some of the flow
Algorithm idea

Are we done?
Is this the best we can do?

Cuts

A cut is a partitioning of the vertices into two sets $S_s$ and $S_t = V - S_s$

Flow across cuts

In flow graphs, we're interested in cuts that separate $s$ from $t$, that is $s \in S_s$ and $t \in S_t$

Flow across cuts

The flow "across" a cut is the total flow from nodes in $S_s$ to nodes in $S_t$ minus the total flow from nodes in $S_s$ to $S_t$
The flow "across" a cut is the total flow from nodes in \( S_s \) to nodes in \( S_t \) minus the total from nodes in \( S_t \) to \( S_s \).

\[
10 + 10 - 6 = 14
\]

Consider any cut where \( s \in S_s \) and \( t \in S_t \), i.e., the cut partitions the source from the sink.

What do we know about the flow across the any such cut?

The flow across ANY such cut is the same and is the current flow in the network.

\[
4 + 10 = 14
\]
Consider any cut where \( s \in S \) and \( t \in S \), i.e., the cut partitions the source from the sink.

The flow across ANY such cut is the same and is the current flow in the network.

**Why? Can you prove it?**

- Inductively:
  - every vertex is on a path from \( s \) to \( t \)
  - in-flow = out-flow for every vertex (except \( s, t \))
  - flow along an edge cannot exceed the edge capacity
  - flows are positive
Flow across cuts

The flow across ANY such cut is the same and is the current flow in the network.

Base case: \( S_s = S_t \)
- Flow is total from \( s \) to \( t \); therefore the total flow out of \( s \) should be the flow.
- All flow from \( s \) gets to \( t \).
  - Every vertex is on a path from \( s \) to \( t \).
  - \( \text{in-flow} = \text{out-flow} \)

Inductive case: Consider moving a node \( x \) from \( S_s \) to \( S_s \).

Is the flow across the different partitions the same?

**Inductive case:** Consider moving a node \( x \) from \( S_s \) to \( S_s \).

\[
\text{cut} = \text{left-inflow}(x) + \text{right-outflow}(x) = \text{left-inflow}(x) - \text{left-outflow}(x) = \text{right-outflow}(x) - \text{right-inflow}(x)
\]

Consider any cut where \( s \in S_s \) and \( t \in S_t \), i.e., the cut partitions the source from the sink.

The flow across ANY such cut is the same and is the current flow in the network.
The "capacity of a cut" is the maximum flow that we could send from nodes in $S$ to nodes in $S$ (i.e. across the cut).

**How do we calculate the capacity?**

Capacity is the sum of the edges from $S$ to $S$:

$10 + 9 = 19$

**Why?**

- Any more and we would violate the edge capacity constraint.
- Any less and it would not be maximal, since we could simply increase the flow.
Max Power

https://www.youtube.com/watch?v=BSVms6c79m

47

Maximum flow

For any cut where $s \in S$ and $t \in S$
- the flow across the cut is the same
- the maximum capacity (i.e., flow) across the cut is the sum of the capacities for edges from $S$ to $S$

49

Are we done?
Is this the best we can do?

Maximum flow

For any cut where $s \in S$ and $t \in S$
- the flow across the cut is the same
- the maximum capacity (i.e., flow) across the cut is the sum of the capacities for edges from $S$ to $S$

50

What is the minimum capacity cut for this graph?

Capacity $= 10 + 4$

51

Is this the best we can do?

We can do no better than the minimum capacity cut
What is the minimum capacity cut for this graph?

**Capacity** = 10 + 4

flow = minimum capacity, so we can do no better

send some flow down a path

How do we determine the path to send flow down?

Send some flow down a path

Search for a path with remaining capacity from s to t

Reroute some of the flow

How do we handle “rerouting” flow?
During the search, if an edge has some flow, we consider "reversing" some of that flow.

The residual graph $G_f$ is constructed from $G$:

For each edge $e$ in the original graph $G$:
- If $\text{flow}(e) < \text{capacity}(e)$:
  - Introduce an edge in $G_f$ with capacity $= \text{capacity}(e) - \text{flow}(e)$
  - This represents the remaining flow we can still push.
- If $\text{flow}(e) > 0$:
  - Introduce an edge in $G_f$ in the opposite direction with capacity $= \text{flow}(e)$
  - This represents the flow that we can reroute/reverse.

Find a path from $s$ to $t$ in $G_f$. 

Find a path from $s$ to $t$ in $G_f$.

None exist... done!
Algorithm idea

Find a path from \( s \) to \( t \) in \( G_f \)
Algorithm idea

Ford-Fulkerson

Ford-Fulkerson(G, s, t)
flow = 0 for all edges
G_f = residualGraph(G)
while a simple path exists from s to t in G_f
send as much flow along the path as possible
G_f = residualGraph(G)
return flow

Ford-Fulkerson: runtime?

Ford-Fulkerson(G, s, t)
flow = 0 for all edges
G_f = residualGraph(G)
while a simple path exists from s to t in G_f
send as much flow along the path as possible
G_f = residualGraph(G)
return flow

Can we simplify this expression?
Ford-Fulkerson: runtime?

Ford-Fulkerson(G, s, t)
flow = 0 for all edges

Gr = residualGraph(G)
while a simple path exists from s to t in Gr:
send as much flow along path as possible
Gr = residualGraph(G)
return flow

max-flow
- increases every iteration
- integer capacities, so integer increases

Can we bound the number of times the loop will execute?

Overall runtime? $O(\text{max-flow} \times E)$

- BFS or DFS
- $O(V + E) = O(E)$
Can you construct a graph that could get this running time?

**Hint:**
Can you construct a graph that could get this running time?

What is the problem here? Could we do better?
Faster variants

- Edmonds-Karp
  - Select the shortest path (in number of edges) from s to t in Gf
  - How can we do this?
  - Use BFS for search
  - Running time: $O(VE)$
  - Avoids issues like the one we just saw
  - See the book for the proof
  - or

- Preflow-push (aka push-relabel) algorithms
  - $O(V^3)$

Other variations...

- Preflow-push (aka push-relabel) algorithms
  - $O(V^3)$

Network flow properties

- If one of these is true then all are true (i.e., each implies the others):
  - $f$ is a maximum flow
  - $G_f$ (residual graph) has no paths from s to t
  - $|f|$ = minimum capacity cut

Handout

88
89
90
91
How much water flow can we continually send from s to t?