Chapter 3

Lecture 3: Graphs and flows

Graphs: a useful combinatorial structure.

Definitions: graph, directed and undirected graph, edge as ordered pair, path, cycle, connected graph, strongly connected directed graph, tree, root, parent, child, ancestor, descendant, leaf.

Theorem 16. *An n-node three has n-1 edges.*

Theorem 17. *For $G = (V, E)$ undirected graph on n nodes any two of these statements implies the third:*
a) G is connected.
b) G does not contain cycle.
c) G has $n - 1$ edges.

Graph representations. There are two typical data structures to represent graphs:

- Adjacency matrix
- Adjacency list: n length array, each record containing the list of all adjacent nodes.

3.1 Graph connectivity and traversal, undirected and directed graphs, bipartite graphs

Clearly, a graph is connected iff there is a path between any two nodes.

Problem. Connectivity problem: is graph G connected?

We answer the question by traversing the graph from an arbitrary node. If all nodes can be reached, then the graph is connected.

Algorithm. Breadth-first search (BFS): adds nodes layer by layer, adding all neighbors of each already discovered node. Layer L_{j+1} consists of all nodes that do not belong to earlier layers, but have an edge to a node in layer L_j.

Theorem 18. *Let T be a BFS tree. Let nodes x and y belong to layers L_i and L_j, and (x, y) be an edge of G. Then $|i - j| \leq 1$.*

Proof. Proof by contradiction. Assume $i < j - 1$. Consider the point when the edges incident to x are examined. Nodes discovered from x belong to layer L_{i+1} or earlier. This is contradiction. □

Algorithm. Depth-first search (DFS): adds nodes along a path, then backtracks to not fully discovered nodes. Recursive implementation is straightforward.

Theorem 19. *Let T be a DFS tree. Let (x, y) be an edge of G, but not in T. Then one of x or y is the ancestor of the other.*

Proof. Suppose x is reached first by the algorithm. If (x, y) is not added, it means that y is already explored, which means, y was explored between the invocation and the end of the recursive call DFS(x), which means y is a descendant of x. □

Theorem 20. *BFS and DFS can be implemented in $O(m + n)$ with adjacency list data structure, and with queue respective stack for the dynamic data handling.*

See slides and book for the algorithms.

Problem. Bipartiteness: Given graph $G = (V, E)$, is he graph bipartite?

Lemma 13. *If a graph is bipartite, then it cannot contain an odd cycle.*

Theorem 21. *Let G be a connected graph, and L_i-s the layers produced by BFS. Then, one of these holds:*

1. *There is no edge joining nodes of the same layer. Then G is bipartite.*
2. *There is an edge connecting nodes of the same layer. Then G has an odd cycle, and is not bipartite.*

Proof. Based on the theorem on the structure of BFS trees. □

The following is not included in the book, but it is useful to know.

Definition. Euler tours: Given graph $G = (V, E)$, an Euler trail is a trail that contains each edge of G (exactly ones). If the trail is closed, then we have an Euler tour.

Theorem 22. ***Euler theorem:*** *Given a connected graph $G = (V, E)$, the the following statements are equivalent:*
(a) G is Eulerian (contains an Euler tour).
(b) Each vertex of G has even degree.
(c) The edge set of G can be partitioned into cycles.

Based on the theorem it is easy to find an Euler tour using the algorithm and data structures proposed by Hierholzer, in $O(m)$ time.

Now let us consider **directed graphs**.

Lemma 14. *If u and v are mutually reachable and v and w are mutually reachable, then s and w are mutually reachable.*

Theorem 23. *Test if G is **strongly connected**: run BFS from any node s on G and G^{rev}. G is strongly connected iff BFS reaches all nodes in both cases.*

Definition. Directed acyclic graph (DAG): graph without directed cycle.

DAGs have several application areas, in network protocols as well as for dependent job scheduling.

Definition. Topological ordering: of directed graph G is an ordering of its nodes as $v_1, v_2...v_n$, such as for every edge (v_i, v_j) we have $i < j$.

Lemma 15. *In every DAG there is a node with no incoming edge.*

Proof. Pick a node, and follow the edges backward. Since all nodes has an incoming edge, sooner or later you reach the first node, or some other, already visited nodes. □

Theorem 24. *G is a DAG iff G has topological ordering.*

Proof. Backward: By contradiction. Assume, G has topological ordering, but also a cycle. Let v_i be the lowest indexed node in C, and v_j the node before v_i in the cycle. Since $j > i$ it contradicts the existence of DAG.

Forward: by constructing a topological ordering, based on the lemma above. □

See the algorithm in the slide. Algorithm complexity: $O(n + m)$, if we keep track nodes that become leave nodes, then we need to address all nodes, and all edges once.

3.2 Maximum-flow in directed graphs

Definition. Flow, f in $G = (V, E)$, c_e:

- capacity condition: $0 \leq f(e) \leq c_e$,
- flow conservation: for each node $v \in V$, $v \neq s, v \neq t$, $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$.

The value of a flow f is $\nu(f) = \sum_{e \text{ out of } s} f(e)$.

Problem. Given directed graph $G = (V, E)$, with edge capacities $c_e \in \mathbb{N}$, a source node s and sink node t. Find an **s-t flow of maximum possible value**.

Definition. Given a graph G and a flow f on G, the residual graph G_f is defined as:

- The node set of G_f is the same as that of G.
- For edge $e = (u, v) \in G$, $f(e) < c_e$, there are $c_e - f(e)$ leftover units of capacity. These are the forward edges in G_f.
- For edge $e = (u, v) \in G$, $f(e) > 0$, we add an edge $e' = (v, u)$ in G_f, $c_{e'} = f(e)$. These are the backward edges in G_f.

Algorithm. The Ford-Fulkerson Algorithm

See the book.

Note, the assumption of integer capacities is important, see a non-terminating example for non-integer capacities in Wikipedia.

If you need to consider undirected graphs, you have to transform it to directed graphs, but adding edges in opposite directions.

We need to prove that:

- The augmentation leads to flows. We need to check that the new f' is a flow: respects the capacities and the flow conservation law.
- The algorithm terminates.
- The algorithm finds the maximum flow. (This will end up to be the max-flow min-cut theorem.)

Theorem 25. *The F-F algorithm can be implemented to run in $O(mC)$ time, where $C = \sum_{e \text{ out of } s} c_e$.*

Proof. The while loop of the algorithm increases the flow by at least 1 in each iteration, that is, the maximum number of *while* iterations is C. The augmentation needs to find a path to t, this can be done with BFS or DFS in $O(m + n) = O(m)$ time. □

Now we will prove that the F-F algorithm finds the maximum flow.

Definition. A cut of G is a partition (A, B) of V, such that $s \in A$ and $t \in B$. The capacity of the cut is $\sum_{e \text{ out of } A} c_e$.

Lemma 16. *For flow f and cut (A, B), the value of the flow $\nu(f) = f^{out}(A) - f^{in}(A)$.*

Proof.

$$f^{\text{in}}(s) = 0, f^{\text{out}}(t) = 0$$
$$\nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$$
$$f^{\text{out}}(v) - f^{\text{in}}(v) = 0, \forall v \in V \setminus \{s, t\}$$
$$\nu(f) = \sum_{A} (f^{\text{out}}(v) - f^{\text{in}}(v))$$

Considering, that there are three type of edges: inside set A, leaving A or coming to A, and edges inside A bring the same amount of flow as they take away in the above equation, we have

$$\nu(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = f^{\text{out}}(A) - f^{\text{in}}(A).$$

□

Corollary 3.

$$\nu(f) \leq c(A, B).$$

This is an important consequence, since it means that the value of a flow is surely not higher than any of the cuts.

Now let us consider f be the flow generated by the F-F algorithm. Note, it means that there is no path form s to t in the residual graph G_f. Then, the following theorem holds.

Theorem 26. *For f, there is an $s-t$ cut (A^*, B^*) in G for which $\nu(f) = c(A^*, B^*)$. Consequently, f has the maximum value of any flow.*

Proof. A way to prove the theorem is to identify such a cut. We can do it as follows.

Let A^* be the set of nodes that are reachable from s in G_f. We will show that A^* and $B^* = V - A^*$ is a cut we are looking for.

- (A^*, B^*) is an $s - t$ cut, since t is definitely in B^*.
- For all edges e out of A^* $f(e) = c_e$, since otherwise the head of the edge would be reachable in G_f.
- For all edges e into A^* $f(e) = 0$, since otherwise there would be an edge out of A^* in G_f.

Consequently,

$$\nu(f) = f^{\text{out}}(A*) - f^{\text{in}}(A^*) = \sum_{e \text{ out of} A^*} c_e - 0 = c(A^*, B^*).$$

□

Now, we know that all flows have to be smaller than all the cuts, and we showed that is is possible to find a flow that reaches the smallest cut capacity. Consequently, we can state the following theorem.

Theorem 27. ***Max-flow Min-cut theorem:*** *In every flow network, the maximum value of an $s - t$ flow is equal to minimum capacity of an $s - t$ cut.*

Corollary 4. *A minimum cut can be found by the F-F algorithm in $O(Cm)$ time.*

Now lets see a simple application of the F-F algorithm.

Problem. Edge-Disjoint Paths in directed graphs: Given $G = (V, E)$, find the maximum number edge disjoint paths from $s \in V$ to $t \in V$.

We transform the problem to a max-flow problem, by assigning $c_e = 1$ unit capacity to each edge.

Theorem 28. *There are k edge-disjoint paths in G iff the value of the maximum flow is at least k.*

Proof. The two directions of the proof:
k edge disjoint path $\Rightarrow$ the value of the max flow in G is at least k: the k paths can define the flow.
the value of the max flow in G is at least $k \Rightarrow$ k edge disjoint path: we can show that the flow can be separated into edge-disjoint paths. □

The trivial consequence of the theorem that F-F can construct disjoint paths in $O(mn)$ time.

Extensions: undirected graph (add directed edges), and node-disjoint path (substitute each node with two nodes connected by and edge).

3.3 Maximum matching in bipartite graphs

Problem. Bipartite matching: Given $G = (V, E)$, $V = X \cup Y$, find a matching of maximum size.

Algorithm. F-F based matching: Extend G to G', with a source and a sink node (see figure), consider $c_e = 1$, find maximum flow in G'.

Theorem 29. *The size of the maximum matching in G is equal to the max flow in G'; and the edges in the matching are the X-Y edges in the flow.*

Proof. Let be M' the set of X-Y edges carrying flow in G', and the maximum flow k. We want to show that M is a matching and it carries k units.

- $|M'| = k$: Consider a cut (A, B), $A = s \cup X$. The value of the flow $f = f_{\text{out of } A} - f_{\text{into } A}$. Since no edges go into A k edges through the cut need to carry the flow.
- Each node in X is the tail of at most one edge in M': since each can be reached by one unit of flow only.
- Each node in Y is the head of at most one edge in M': as above.

Consequently, M' is a matching of size k. □

The consequence is that the F-F algorithm finds maximum matching in $O(nm)$ time.

It is possible to formulate the algorithm without extending the graph, following the *method of alternating paths*. See the book.

Theorem 30. ***Hall's theorem**: Assume that the bipartite graph G has to sides $|X| = |Y|$. The graph either has a perfect matching, or $\exists A \subseteq X$,such that $\Gamma(A) < |A|$, where $\Gamma(A)$ is the sets of nodes in Y that are adjacent to nodes in A. The perfect matching or a set A can be found in $O(mn)$ time.*

The proof is based on the max-flow - min-cut theorem. See the book.

Problem. Minimum-cost perfect matching: Given bipartite graph $G = (V, E)$, $|X| = |Y| = n$ and costs c_e, find a perfect matching with minimum cost.

Minimum cost perfect matching can be found based on the alternating path approach, always looking for cheapest alternating path. The running time is the time required for n shortest-path computations.

See algorithm and proofs in the book. Often referred to as the Hungarian method, and has a graph and a matrix based version.

3.4 What have we learned?

- Graph traversal with BFS and DFS
- Identification of bipartite graphs
- Identification of Euler graphs
- Similar results of directed graphs: strong connectivity, DAGs, topological ordering
- Flows and cuts
- Ford-Fulkerson, and the Max-flow min cut theorem.
- Disjoint paths
- Matching, perfect matching, minimum cost matching.