CS 173
Introduction to Graph Algorithms
Tandy Warnow
Today’s material

- Exhaustive search strategies
- Greedy search
- Decision, Optimization, and Construction Problems
- Proving theorems about graphs
Suppose you want to solve MAX CLIQUE. Given input graph $G = (V, E)$:

- Enumerate all subsets of $V$
- For each one, determine if it is a clique; if so, record size
- Return size of largest clique found.

Obviously correct, but too expensive. (How expensive?)

This is an example of **Exhaustive Search**
Solving MAX CLIQUE using greedy search

Given input graph $G = (V, E)$:

- Order the vertices $v_1, v_2, \ldots, v_n$
- $A := \{v_1\}$
- For $i = 2$ up to $n$ DO:
  - If $A \cup \{v_i\}$ is a clique, then $A := A \cup \{v_i\}$

Return $A$

Obviously $A$ is a clique, but it may not be maximum.
This is a fast algorithm, but it may not find an optimal solution.
(Class: show such a graph.)

This is an example of a greedy algorithm.
How can we solve MAX CLIQUE?

- The exhaustive search strategy is not polynomial time.
- The greedy algorithm is fast but not guaranteed to find an optimal solution.
- What should we do?

The problem is MAX CLIQUE is NP-hard!
In Class Problem

Suppose you have an algorithm $\mathcal{A}$ that solves the decision problem for MATCHING:

- **Input:** Graph $G = (V, E)$ and positive integer $k$
- **Question:** $\exists E_0 \subseteq E$ such that $|E_0| = k$ and $E_0$ is a matching?

Can we make a polynomial number of calls to $\mathcal{A}$ (and a polynomial amount of other operations) to
- find the size of the maximum matching in $G$?
- find the largest matching in $G$?
Decision, Optimization, and construction problems

A **decision** problem answers Yes or NO (typically, asks whether something exists).
An **optimization** problem seeks the best possible score (typically, the size of the smallest object, or the largest object).
A **construction** problem seeks the object with the best possible score.

Example: CLIQUE.

- The decision problem takes as input a graph $G$ and an integer $k$ and asks whether the graph $G$ has a clique of size $k$.
- The optimization problem takes as input a graph $G$ and asks for the largest $k$ so that $G$ has a clique of size $k$.
- The decision problem takes as input a graph $G$ and asks for the largest clique.
Decision, Optimization, and construction problems

A decision problem answers Yes or NO (typically, asks whether something exists).
An optimization problem seeks the best possible score (typically, the size of the smallest object, or the largest object).
A construction problem seeks the object with the best possible score.

Example: VERTEX COLORING.

- The decision problem takes as input a graph $G$ and an integer $k$ and asks whether the graph $G$ can be properly vertex-colored using $k$ colors.
- The optimization problem takes as input a graph $G$ and asks for the smallest $k$ so that $G$ can be properly vertex-colored using $k$ colors.
- The decision problem takes as input a graph $G$ and asks for a proper vertex coloring using the smallest number of colors.
Relationship between decision, optimization, and construction problems

To solve the optimization problem, we define Algorithm $\mathcal{B}$ as follows.
The input is graph $G = (V, E)$. If $E = \emptyset$, we return 0. Else, we do the following:

- For $k = |E|$ down to 1, DO
  - If $\mathcal{A}(G, k) = \text{YES}$, then Return($k$)

It is easy to see that
- $\mathcal{B}$ is correct,
- that $\mathcal{B}$ calls $\mathcal{A}$ at most $m$ times
- that $\mathcal{B}$ does at most $O(m)$ additional steps.

Hence $\mathcal{B}$ satisfies the desired properties.
Relationship between decision, optimization, and construction problems

We define Algorithm $C$ to find a maximum matching, as follows. The input is graph $G = (V, E)$. If $E = \emptyset$, we return $\emptyset$. Otherwise, let $E = \{e_1, e_2, \ldots, e_m\}$, and let $k = B(G)$.

- Let $G^*$ be a copy of $G$
- For $i = 1$ up to $m$ DO
  - Let $G'$ be the graph obtained by deleting edge $e_i$ (but not the endpoints of $e_i$) from $G^*$.
  - If $A(G', k) = YES$, then set $G^* := G'$.
- Return the edge set $E(G^*)$ of $G^*$.

It is easy to see that $C$ calls $B$ once, calls $A$ at most $m$ times, and does at most $O(m)$ other operations. Hence the running time satisfies the required bounds.
What about accuracy?
Finding the largest matching in a graph

- Let $G^*$ be a copy of $G$
- For $i = 1$ up to $m$ DO
  - Let $G'$ be the graph obtained by deleting edge $e_i$ (but not the endpoints of $e_i$) from $G^*$.
  - If $\mathcal{A}(G', k) = YES$, then set $G^* = G'$.

Return the edge set of $G^*$.

Notes:
- The edge set returned at the end is a matching (we’ll look at this carefully in the next slide).
- We never reduce the size of the maximum matching when we delete edges. Hence, $\mathcal{B}(G^*) = \mathcal{B}(G)$.
- Therefore this algorithm returns a maximum matching.
Finding the largest matching in a graph

Recall $k$ is the size of a maximum matching in input graph $G$, with edge set $\{e_1, e_2, \ldots, e_m\}$, $m \geq 1$.

- Let $G^*$ be a copy of $G$
- For $i = 1$ up to $m$ DO
  - Let $G'$ be the graph obtained by deleting edge $e_i$ (but not the endpoints of $e_i$) from $G^*$.
  - If $A(G', k) = \text{YES}$, then set $G^* = G'$.

Return the edge set of $G^*$.

**Theorem:** The edge set $E^*$ of $G^*$ is a matching in $G$.

**Proof (by contradiction):** If not, then $E^*$ has at least two edges $e_i$ and $e_j$ (both from $E$) that share an endpoint. Let $E_0$ be a maximum matching in $G^*$; hence $E_0$ is a maximum matching for $G$. Note that $E_0$ cannot include both $e_i$ and $e_j$. Suppose (w.l.o.g.) $e_i \not\in E_0$. During the algorithm, we checked whether a graph $G'$ that did not contain $e_i$ had a matching of size $k$. Since we did not delete $e_i$, this means the answer was NO. But the edge set of that $G'$ contains the matching $E_0$, which means $G'$ has a matching of size $k$, yielding a contradiction.
Reductions

- We used an algorithm \( A \) for decision problem \( \pi \) to solve an optimization or construction problem \( \pi' \) on the same input. We also required that we call \( A \) at most a polynomial number of times, and that we do at most a polynomial number of other operations.

- This means that if \( A \) runs in polynomial time, then we have a polynomial time algorithm for both \( \pi \) and \( \pi' \). Note that we use two things here: \( A \) is polynomial, and the input did not change in size.

- What we did isn’t really a Karp reduction, because Karp reductions are only for decision problems... but the ideas are very related.

- If you can understand why this works, you will understand why Karp reductions have to satisfy what they satisfy.

Just try to understand the ideas. This is not about memorization.
Complements of graphs

Let $G = (V, E)$ be a graph.

The graph $G^c$ contains the same vertex set, but only contains the missing edges (though not the self-loops), and is referred to as the complement of $G$.

- Suppose $V_0$ is a clique in $G$. What can you say about $V_0$ in $G^c$?
- Suppose $V_0$ is an independent set in $G$. What can you say about $V_0$ in $G^c$?
Complements of sets

Suppose $V_0$ is an independent set in $G$. What can you say about $G \setminus V_0$?

Suppose $V_0$ is a clique in $G$. What can you say about $V \setminus V_0$?

Suppose $V_0$ is a vertex cover in $G$. What can you say about $V \setminus V_0$?
**Adding vertices to graphs:**
Suppose you add a vertex \( v \) to \( G \) and make \( v \) adjacent to every vertex in \( V \).

Let the new graph be called \( G' \).

How do these values change between \( G \) and \( G' \)?
- the size of the maximum clique,
- the size of the maximum independent set,
- the chromatic number
Things to think about

▶ Suppose $G$ is a simple graph that has a maximum matching of size $k$ and a maximum vertex cover of size $k'$. Prove that $k' \geq k$.

▶ Prove that every tree can be 2-colored.

▶ Prove that every tree with at least two three vertices has a sibling pair of leaves (where two leaves are siblings if they share a neighbor).

▶ Come up with a simple algorithm to find a maximal matching (i.e., a matching that cannot be enlarged by adding another edge) in a graph, and analyze its running time.

▶ Show how having an algorithm to compute the chromatic number in a graph can be used to find an optimal vertex coloring for a graph, with only a polynomial number of calls to the algorithm.