CS120: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Lecture 16: Graphs, Ethics, Logic

Harvard SEAS - Fall 2021 *Oct. 28, 2021*

1 Announcements

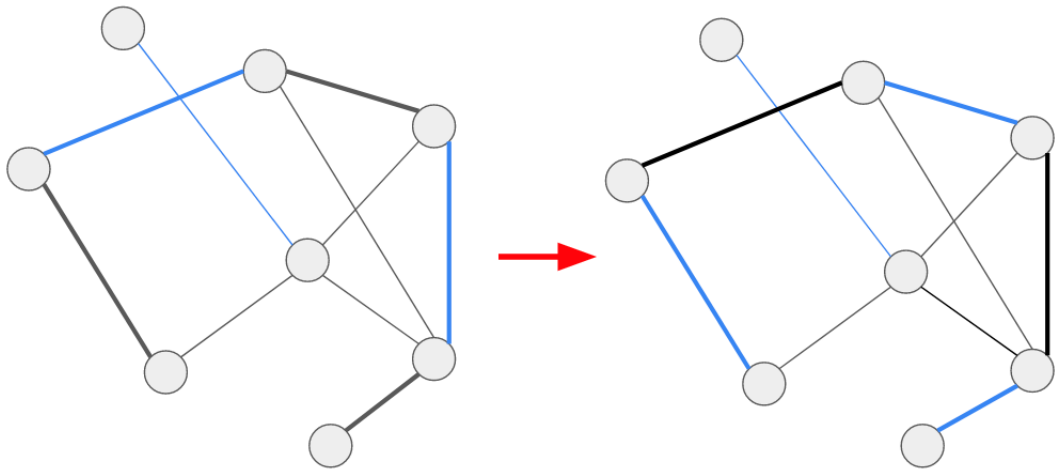
Recommended Reading:

- Lewis–Zax Ch. 9–10.
- Please fill out the mid-semester feedback form! Up on Ed
- Sections will resume on Saturday

2 Matching Wrap-Up

Last time we saw:

Lemma 2.1. *Given a graph $G = (V, E)$, a matching M, and an augmenting path P with respect to M, we can construct a matching M' with $|M'| = |M| + 1$ in time $O(n)$.*



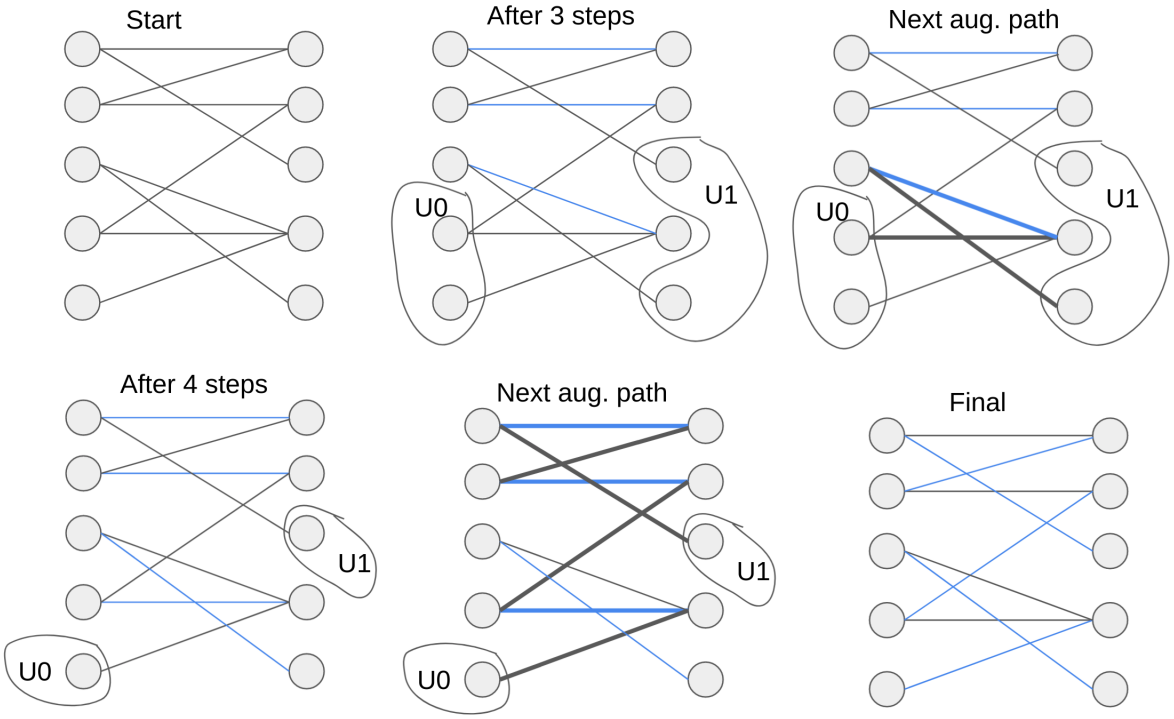
Lemma 2.2. *Let $G = (V_0 \cup V_1, E)$ be bipartite and let M be a matching in G that is not of maximum size. Let U be the vertices that are not matched by M, and $U_0 = V_0 \cap U$ and $U_1 = V_1 \cap U$. Then:*

1. *G has an alternating path that starts in U_0 and ends in U_1.*
2. *Every shortest alternating path from U_0 to U_1 is an augmenting path.*

With these lemmas, we obtain a matching algorithm: repeatedly search for a shortest alternating path from U_0 to U_1, and use it to grow our matching.

```
MaxMatchingAltPaths(G)
  Input  : A bipartite graph G = (V, E)
  Output : A maximum-size matching M ⊆ E
Remove isolated vertices from G;
Let V_0, V_1 be the bipartition (i.e. 2-coloring) of V;
M = ∅;
repeat
    Let U be the vertices unmatched by M, U_0 = V_0 ∩ U, U_1 = V_1 ∩ U;
    Try to find a shortest alternating path P from U_0 to U_1;
    if P ≠ ⊥ then augment M using P via Lemma 2.1;
until P = ⊥;
return M
```

Example:



The correctness of Algorithm 10 follows from Lemma 2.2, so we only need to analyze the run time:

Theorem 2.3. *Maximum Matching can be solved in time $O(mn)$ on bipartite graphs with m edges and n vertices.*

Proof.
We can find a shortest alternating path from U_0 to U_1 in time $O(n+m)$ via BFS. Then to bound the overall runtime, note that a matching contains at most $n/2$ edges, so we will augment our matching $O(n)$ times, giving a total runtime of $O(n(n+m))$. □

3 Graphs and Modelling

- Edges = reachability/proximity: ShortestPaths, Connected Components, Clustering
- Edges = conflicts/incompatibility: Coloring, Independent Sets
- Edges = compatibility for an exclusive assignment: Matching, Edge Coloring

4 Cautions about Deploying Algorithms

When we are designing and using algorithms to solve well-defined mathematical problems, we may get the feeling that we are solving just doing objective science and engineering. However, when we apply these algorithms to real-world situations, especially (but not only) those involving humans, there can be real, sometimes unintended, consequences that should be evaluated ethically. We won't have time to explore these in depth this semester, but future computer science courses you take will.

Some examples from last two lectures (independent set, interval scheduling,matching):

Mathematically precise $\neq$ devoid of moral implications. For instance, a dating app finding a maximum matching can be precisely described as an algorithm, but there are still choices inherent to the implementation: why do we use that algorithm? How does it choose between competing solutions? How do we handle user preferences?

- Choice of objective function: What is the problem we are solving, and how do we measure the quality of our solution? For example, should all patients and donors be treated equally in kidney exchange, or should some (e.g. ones with more severe disease) be prioritized?
- Input data: what data are we using? Is it biased in some way, does it reflect pre-existing disparities? For example, if road network or traffic data is poor-quality in some neighborhoods, fewer rideshare drivers might get matched there (cf. PS7).
- Impact of the decisions on users and thus on future data (vicious cycles): For example, if a social network tries to form communities among its users using an independent set or coloring algorithm based on similar profiles, it might create echo chambers and increase homogeneity within the groups. Also, once the algorithm is in place, some users may try to influence the outcome by changing their data. For example, in kidney exchange, there is a concern that hospitals will not reveal all of their donors in order to maximize the number of their patients that get served.
- Privacy: Do the users want us to be using their data in this way? When we produce an output, might that reveal sensitive information abbout the users?

5 Propositional Logic

Motivation: Logic is a fundamental building block of computation (e.g. digital circuits), and is also a very expressive language for encoding computational problems we want to solve.

Definition 5.1 (informal)**.** A *boolean formula* φ is a formula built up from a finite set of variables, say $x_0, \ldots, x_{n-1}$, using the logical operators $\wedge$ (AND), $\vee$ (OR), and $\neg$ (NOT), and parentheses.

Every boolean formula φ on n variables defines a boolean function, which we'll abuse notation and also denote by $\varphi : \{0,1\}^n \to \{0,1\}$, where we interpret 0 as false and 1 as true, and give $\wedge, \vee, \neg$ their usual semantics (meaning).

See the Lewis–Zax text for formal, inductive definitions of boolean formulas and the corresponding boolean functions.

Example:

$$\varphi_{maj}(x_0, x_1, x_2) = (x_0 \wedge x_1) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_0)$$

Evaluates to 1 if at least two of its inputs are 1, so $\phi_{maj}(1,1,0) = 1$ and $\phi_{maj}(1,0,0) = 0$.

$$\varphi_{pal}(x_0, x_1, x_2, x_3) = ((x_0 \wedge x_3) \vee (\neg x_0 \wedge \neg x_3)) \wedge ((x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2))$$

Evaluates to 1 if the input is a palindrome.

Definition 5.2. A *literal* is a variable (e.g. x_i) or its negation ($\neg x_i$).

A boolean formula is in *disjunctive normal form (DNF)* if it is the OR of a set of *terms*, each of which is the AND of a set of literals.

A boolean formula is in *conjunctive normal form (CNF)* if it is the AND of a set of *clauses*, each of which is the OR of a set of literals.

Note that we implicitly assume here (by use of the word "set") that terms and clauses do not contain any duplicate literals. We can also remove any clause or term with both a variable x and its negation $\neg x$, as that clause or term will be always true (in the case of a clause) or always false (in the case of a term). Also by convention, an empty term is always true and an empty clause is always false.

Lemma 5.3. *For every boolean function $f : \{0,1\}^n \to \{0,1\}$, there are boolean formulas φ and ψ in DNF and CNF, respectively, such that $f \equiv \varphi$ and $f \equiv \psi$, where we use $\equiv$ to indicate equivalence as functions.*

Proof. For a function $f : \{0,1\}^n \to \{0,1\}$, we can define the DNF

$$\varphi(x_0, \ldots, x_{n-1}) = \bigvee_{\alpha \in \{0,1\}^n : f(\alpha) = 1} ((x_0 = \alpha_0) \wedge (x_1 = \alpha_1) \wedge \cdots \wedge (x_{n-1} = \alpha_{n-1}))\,.$$

Note that $x_i = \alpha_i$ can be rewritten as either x_i (with $\alpha_i = 1$) or $\neg x_i$ (with $\alpha_i = 0$). For example, applying to the palindrome function on 4 bits, we get

$$\varphi(x_0, x_1, x_2, x_3) = (x_0 \wedge x_1 \wedge x_2 \wedge x_3) \vee (x_0 \wedge \neg x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_0 \wedge x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_0 \wedge \neg x_1 \wedge \neg x_2 \wedge \neg x_3),$$

where the terms correspond to the satisfying assignments $(1,1,1,1)$, $(1,0,0,1)$, $(0,1,1,0)$, and $(1,1,1,1)$.

For the CNF, we define

$$\varphi(x_0, \ldots, x_{n-1}) = \bigwedge_{\alpha \in \{0,1\}^n : f(\alpha) = 0} ((x_0 \neq \alpha_0) \vee (x_1 \neq \alpha_1) \vee \cdots \vee (x_{n-1} \neq \alpha_{n-1}))\,.$$

For example, the majority function on 3 bits can be written as:

$$\psi(x_0, x_1, x_2) = (x_0 \vee x_1 \vee x_2) \wedge (\neg x_0 \vee x_1 \vee x_2) \wedge (x_0 \vee \neg x_1 \vee x_2) \wedge (x_0 \vee x_1 \vee \neg x_2),$$

with the clauses corresponding to the 4 non-satisfying assignments $(0, 0, 0), (1, 0, 0), (0, 1, 0), (1, 1, 1)$. This example shows that the DNF and CNF given by the general construction are not necessarily the smallest ones possible for a given function, as the majority function can also be expressed by the following simpler CNF formula:

$$(x_0 \vee x_1) \wedge (x_0 \vee x_2) \wedge (x_1 \vee x_2).$$

□