CS173 Lecture B, November 3, 2015

Tandy Warnow

November 3, 2015

CS 173, Lecture B
November 3, 2015
Tandy Warnow

Announcements

- Examlet 7 is a take-home exam, and is due November 10, 11:05 AM, in class.
- Homework #11 is due Sunday November 8, 10 PM. (Early in other words, because that way you can get feedback on what you got right and wrong, and go to office hours on Monday for additional help.)
- I will be out of town next Tuesday (exam will be proctored by course staff).
- I will not hold office hours next week.
- Office hours this week: Today (Tuesday) 1-3, and Thursday, November 5, from 1-2 PM.

Take-home Examlet

- This examlet is on the honor code (see front page). Please, don't discuss the examlet with anyone.
- Course staff should only answer questions that have to do with understanding the examlet problems, not solving them.
- The examlet is due in class on Tuesday November 10, by 11:05 AM. Late submissions will not be graded. We'll go over the examlet at 11:05!
- If you won't be able to get to class on time, return the examlet to Elaine Wilson *before* it is due in class.

Examlet questions

1. Proving that a graph G with maximum vertex degree D can be properly vertex-colored with $D + 1$ colors. Do this proof by induction on the number of vertices in G. (Hint: draw pictures of graphs)
2. Show that if you can solve the decision problem for CLIQUE you can solve the decision problem for INDEPENDENT SET using a polynomial number of operations, with a polynomial number of calls to the algorithm for CLIQUE.
3. Show that if you can solve the decision problem for CLIQUE then you can construct the largest clique in an input graph using a polynomial number of operations, with a polynomial number of calls to the algorithm for CLIQUE.
4. Design a polynomial time dynamic programming algorithm for all-pairs shortest path, but with a different subproblem formulation.

Assigned Reading this week

The assigned reading this week includes material about

- The *All-Pairs Shortest Path* problem, and a solution by Floyd and Warshall
- A 2-approximation algorithm for Minimum Vertex Cover

Today I'll present a way of solving All-Pairs Shortest Path that is essentially the same as what you've read, but *simpler* in two ways:

- We assume the graph is undirected and all edges have positive weights.
- We don't concern ourselves with finding the shortest paths, just their costs.

All Pairs Shortest Path

The input is an undirected graph $G = (V, E)$ with positive edge weights on the edges.

Given a path P between two vertices v_i and v_j, the **cost** (or **length**) of P is the sum of the weights on the edges in the path. We write this as $Cost(P)$.

A shortest path between two vertices is one with minimum cost.

We want to find the length of the shortest path between every pair of vertices, and store this in an $n \times n$ matrix (where $n = |V|$).

All Pairs Shortest Path

- Input: graph $G = (V, E)$, $V = \{v_1, v_2, \ldots, v_n\}$, and edge weights given by $w : E \rightarrow R^+$. (Hence $w(e)$ is the weight of edge e.)
- Output: D, an $n \times n$ matrix, so that $D[i, j]$ is the length of the shortest path from v_i to v_j. Note we set $D[i, i] = 0$ for all $i = 1, 2, \ldots, n$.

For $i \neq j$, and given a path P from v_i to v_j, we look at the *internal nodes* of the path (i.e., everything except the endpoints), and let $MAX(P)$ denote the maximum index of any internal node. If the path is a single edge, then $MAX(P) = 0$.
Thus

- For $P = v_3, v_1, v_2, v_5, v_7$, $MAX(P) = 5$.
- For $P = v_5, v_1, v_2, v_7$, $MAX(P) = 2$.
- For $P = v_1, v_2$, $MAX(P) = 0$ (no internal nodes).

Floyd-Warshall Algorithm

Let $M[i, j, k]$ be the length of the shortest path P from v_i to v_j such that $MAX(P) \leq k$.
If $i = j$, we set $M[i, j, k] = 0$.
If $i \neq j$ and there is no path between v_i and v_j satisfying $MAX(P) \leq k$, then we set $M[i, j, k] = \infty$.
We let $k = 0, 1, 2, \ldots, n$, and $1 \leq i, j \leq n$.
Questions:

- What is $M[1, 2, 0]$?
- What is $M[1, 2, 2]$?
- What is $M[1, 2, 3]$?
- What is $M[1, 2, 5]$?

Floyd-Warshall Algorithm

Remember $M[i, j, k]$ is the length of the shortest path P from v_i to v_j such that $MAX(P) \leq k$.
Suppose we were able to compute (correctly) all $M[i, j, k]$, for $1 \leq i, j \leq n$ and $0 \leq k \leq n$.

- Question: How could we compute the length of the shortest path from v_i to v_j?
- Answer: it is the same as $M[i, j, n]$.

So once we have computed all entries in the 3-dimensional matrix M, we return the 2-dimensional matrix obtained by setting $k = n$.

Floyd-Warshall Algorithm, k=0

When $k = 0$ we are asking about the lengths of paths that have no internal nodes.

Case: $i = j$**:** Set $M[i, i, 0] = 0$

Case: $i \neq j$**:** $M[i, j, 0]$ is the length of the shortest path P from v_i to v_j with $MAX(P) = 0$, or ∞ if no such path exists.

If the path P exists, it is a single edge e, and its weight is $w(e)$.

Hence, $M[i, j, 0] = w(v_i, v_j)$ if $(v_i, v_j) \in E$, and otherwise $M[i, j, 0] = \infty$.

Floyd-Warshall Algorithm

After we compute all entries of the matrix M with $k = 0$, can we compute the entries with $k = 1$?

If $i = j$, we set $M[i, j, 1] = 0$.

For $i \neq j$, consider a shortest path P from v_i to v_j with $MAX(P) \leq 1$. What does it look like?

Floyd-Warshall Algorithm

Consider a shortest path P from v_i to v_j with $MAX(P) \leq 1$.
Cases:

- P is a single edge e, and so $Cost(P) = w(e) = M[i, j, 0]$.
- P has an internal vertex, which must be v_1.
  Hence P has two edges, (v_i, v_1) and (v_1, v_j).
  Then $Cost(P) = w(v_i, v_1) + w(v_1, v_j)$.
  Note that $w(v_i, v_1) = M[i, 1, 0]$ and $w(v_1, v_j) = M[1, j, 0]$.

Hence $M[i, j, 1] = \min\{M[i, j, 0], M[i, 1, 0] + M[1, j, 0]\}$.

Floyd-Warshall Algorithm

After we compute all entries of the matrix M with $k = 0, 1, \ldots, K-1$, can we compute the entries with $k = K$?

Consider a shortest path P from v_i to v_j with $MAX(P) \leq K$.
Cases:

- P satisfies $MAX(P) \leq K-1$. Then $Cost(P) = M[i, j, K-1]$.
- P satisfies $MAX(P) = K$.
  Hence $v_K \in P$. Analyzing this is a bit more complicated, but we will show the path P satisfies
  $Cost(P) = M[i, K, K-1] + M[K, j, K-1]$.

Floyd-Warshall Algorithm

Consider a shortest path P from v_i to v_j with $MAX(P) = K$.
Hence $v_K \in P$.
Write P as the concatenation of two paths, P_1 from v_i to v_K and P_2 from v_K to v_j.
Do you agree that $Cost(P) = Cost(P_1) + Cost(P_2)$?
Questions:

- What is $MAX(P_1)$?
- What is $MAX(P_2)$?

Floyd-Warshall Algorithm

P is a shortest path P from v_i to v_j with $MAX(P) = K$.
We write P as the concatenation of P_1 from v_i to v_K and P_2 from v_K to v_j.
Note that $MAX(P_1) \leq K - 1$ and $MAX(P_2) \leq K - 1$.
Hence,

- $Cost(P_1) = M[i, K, K - 1]$
- $Cost(P_2) = M[K, j, K - 1]$
- $Cost(P) = M[i, K, K - 1] + M[K, j, K - 1]$.

Floyd-Warshall Algorithm

$$M[i,j,K] = \min\{M[i,j,K-1], M[i,K,K-1] + M[K,j,K-1]\}$$

Floyd-Warshall Algorithm

Algorithm:

- %Set base cases:
- $M[i,i,0] := 0$
- For $i \neq j$, if $(v_i, v_j) \in E$ then $M[i,j,0] := w(v_i, v_j)$, else $M[i,j,0] := \infty$
- For $k = 1$ up to n DO:
  - For all $i = 1, 2, \ldots, n$, $M[i,i,k] := 0$
  - For all pairs i, j with $i \neq j$, $M[i,j,k] := \min\{M[i,j,K-1], M[i,K,K-1] + M[K,j,K-1]\}$

Easy to see this is $O(n^3)$ (in fact, $\Theta(n^3)$).

Floyd-Warshall Algorithm

- As presented, this algorithm only returns the length (i.e., cost) of the shortest path between every pair of nodes. It does not return the path itself, or directly help you find the path. The assigned reading shows you how to can store information to help you reconstruct the shortest path.
- This algorithm, as described, was only for undirected graph with positive edge weights. It's a trivial modification to deal with directed graphs or with weights that can be negative. (It just gets strange to think about shortest paths when there are negative cycles...)

Dynamic Programming Algorithms

When you are asked to create a DP algorithm, remember to:

- Define what your variables mean
- Show the base cases and how you'll compute them
- Show how you'll compute the other variables, using previously computed variables (this is the "recursive definition")
- State the order in which you compute the variables (remember - bottom-up)
- Show where your "answer" will be (or how you'll compute your answer from your variables)
- Explain why your algorithm is correct
- Analyze the running time

Other problems on the take-home examlet

Let's work on something similar to what's in the examlet.
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?
Hence, given a graph $G = (V, E)$ and a positive integer k,
$\mathcal{A}(G, k) = YES$ if and only if G has a matching of size k.

Using algorithms for MATCHING

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

We want to use 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

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 $\mathcal{A}(G, k) = YES$, 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 $\mathcal{C}$ 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 $k = \mathcal{B}(G)$.

- 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 $\mathcal{A}(G', k) = YES$, then set $G^* := G'$.
- Return the edge set $E(G^*)$ of G^*.

It is easy to see that $\mathcal{C}$ calls $\mathcal{B}$ once, calls $\mathcal{A}$ at most m times, and does at most $O(m)$ other operations. Hence the running time satisifes 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 $\mathcal{A}(G', k) = YES$, 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.*) $e_i \notin E_0$. 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 $\mathcal{A}$ for decision problem π to solve an optimization or construction problem π' on the same input. We also required that we call $\mathcal{A}$ at most a polynomial number of times, and that we do at most a polynomial number of other operations.
- This means that if $\mathcal{A}$ runs in polynomial time, then we have a polynomial time algorithm for both π and π'. Note that we use two things here: $\mathcal{A}$ 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.

Graph properties

Complements of graphs Let $G = (V, E)$ be a graph. Its complement G^c contains the same vertex set, but only contains the missing edges.

- 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?

Graphs

Complementary sets

- Suppose V_0 is an independent set in G. What can you say about $G - V_0$?
- Suppose V_0 is a clique in G. What happens if you delete V_0 from G?
- Suppose V_0 is a vertex cover in G. What happens if you delete V_0 from G?

Manipulating Graphs

Adding vertices to graphs:

Suppose you add a vertex to G and make it adjacent to every vertex in V. What happens to

- the size of the maximum clique,
- the size of the maximum independent set,
- the chromatic number

Preparing for Thursday

Try to prove the following theorems:

- If $G = (V, E)$ is a graph and $M \subseteq E$ is a matching, then every vertex cover for G has at least $|M|$ vertices.
- If $G = (V, E)$ is a graph and $M \subseteq E$ is a *maximal* matching, then if we delete all the vertices that are endpoints in any edge in M from G, we either get the empty graph, or we get an independent set.
- Let M be a maximal matching for graph G, and let V_0 be endpoints of edges in M. Let V_1 be the smallest vertex cover for G. How large can $\frac{|V_0|}{|V_1|}$ be?

Note the definition of maximal matching: you can't add any edges to a maximal matching and have it still be a matching. For an example of a maximal but not maximum matching, consider the graph P_4 with four vertices, a, b, c, d, and three edges $(a, b), (b, c), (c, d)$. The middle edge (b, c) is a maximal matching but not a maximum matching.

How hard do you think it is to find a maximal matching for a graph?