CS173 Lecture B, November 3, 2015

Tandy Warnow

November 10, 2015

CS 173, Lecture B
November 3, 2015
Tandy Warnow

Problem 1

Prove (by induction on n, the number of vertices in G) that every simple graph G can be properly vertex-colored using at most $D + 1$ colors, where D is the maximum degree of any vertex in G.
See textbook pages 129-130

Problem 2

Suppose you have an algorithm $\mathcal{A}$ that solves the CLIQUE decision problem. Design an algorithm $\mathcal{B}$ that solves the INDEPENDENT SET decision problem, and which satisfies the following constraint: Given input $G = (V, E)$, graph G on n vertices, and positive integer k, your algorithm $\mathcal{B}$ performs at most a polynomial in n number of steps, where each step is one of the following:

- Applying algorithm $\mathcal{A}$ to some graph with at most n vertices
- Assigning values to variables
- Numeric operations
- Logical operations
- Input/Output operations

Analyze the running time of your algorithm and explain why it correctly solves the problem.

Solution to problem 2

Algorithm $\mathcal{B}$:
Given the pair (G, k) with $G = (V, E)$ and $k \in Z^+$ as input to INDEPENDENT SET, we do the following:

- We compute G^c, the complement graph, which contains the same vertex set as G and whose edge set contains exactly those pairs (x, y) where $(x, y) \notin E$.
- We run $\mathcal{A}$ on (G^c, k) and we return the same answer. Thus, we return YES if and only if $\mathcal{A}$ determines that G^c has a clique of size k.

Solution to problem 2

Note that

- $\mathcal{B}$ takes $O(n^2)$ time to compute the graph G^c (recall G has n vertices)
- G^c contains exactly n vertices
- $\mathcal{B}$ applies $\mathcal{A}$ only once, and does $O(1)$ other operations

To see that $\mathcal{B}$ is correct, we need to show that $\mathcal{B}(G, k) = YES$ if and only if G has an independent set of size k.

Solution to problem 2

$\Rightarrow$: Suppose that $\mathcal{B}(G, k) = YES$. By definition, $\mathcal{A}(G^c, k) = YES$, and so G^c has a clique V_0 of size k. Then note that V_0 is an independent set in G (since edges in G^c correspond to non-edges in G).
Hence, G has an independent set of size k.
$\Leftarrow$: Conversely, suppose that G has an independent set of size k. Then G^c has a clique of size k, and so $\mathcal{A}(G, k) = YES$. Hence by definition $\mathcal{B}(G, k) = YES$.

Problem 3

Consider the algorithm $\mathcal{A}$ from Problem 2. Design an algorithm $\mathcal{C}$ that takes as input a graph $G = (V, E)$ and returns the subset $V_0 \subseteq V$ that is a clique and has the maximum size of all cliques in G.

Given input $G = (V, E)$, a graph on n vertices, your algorithm $\mathcal{C}$ must perform no more than a polynomial in n number of steps, where each step is one of the following:

- Applying algorithm $\mathcal{A}$ to some graph with at most n vertices
- Assigning values to variables
- Numeric operations
- Logical operations
- Input/Output operations

Analyze the running time of your algorithm and explain why it correctly solves the problem.

Solution to problem 3

Algorithm $\mathcal{C}$:
If G has no vertices, return $\emptyset$. Otherwise, assume $V = \{v_1, v_2, \ldots, v_n\}$ and $n \geq 1$.
The algorithm has two phases:

- Phase 1: Find the size of the maximum clique
- Phase 2: Use this information to find the max clique

Solution to problem 3

Phase 1: Find the size k of the maximum clique in G

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

In other words, we start with the largest possible size and we ask if G has a clique of that size.

If so, we return that value, and otherwise we decrease the value.

We keep asking until we find a value that algorithm $\mathcal{A}$ says YES to.

Solution to problem 3

Phase 2: Now find a clique of size k

We use the value k obtained in the first step. Let $V = \{v_1, v_2, \ldots, v_n\}$ be the vertices in G.

- Let G_1 be a copy of G.
- For $i = 1$ up to n DO
  - Let G_i' be the graph obtained by deleting v_i from G_i
  - If $\mathcal{A}(G_i', k) = YES$ then $G_{i+1} := G_i'$
- Return the vertex set V_{n+1} of G_{n+1}.

In other words, we keep deleting vertices that aren't required in order to have a clique of size k. Note that G_i is a subgraph of G_{i-1} for every i, which means that any clique in G_i is a clique in every G_j if $j < i$.

Solution to problem 3

We know that $G = G_1$ has a clique of size k.

Since we never delete a vertex if it would result in a graph that doesn't have a k-clique, G_i has a clique of size k for $i = 1, 2, \ldots, n+1$.

Thus, $G_{n+1} = (V_{n+1}, E_{n+1})$ has a clique of size k.

We just need to show that V_{n+1} has exactly k vertices, and so is a k-clique.

Solution to problem 3

We prove $|V_{n+1}| = k$ by contradiction.
We know G_{n+1} has a k-clique V^*, and so $|V_{n+1}| \geq k$.
Suppose $|V_{n+1}| > k$.
Hence, $V^* \subset V_{n+1}$.
Since V^* is a proper subset, we can pick $v_i \in V_{n+1} \setminus V^*$.

Since we did not delete v_i when we had the chance, $G_i' = G_i \setminus v_i$ does not contain a k-clique.

But $v_i \notin V^*$ and so G_i' contains the k-clique V^*, contradicting our assumptions.

Hence, $|V_{n+1}| = k$, and $\mathcal{B}(G)$ returns a maximum clique in G.

Solution to problem 3

Running time:
Note that $\mathcal{B}$ calls $\mathcal{A}$ at most $2n$ times (that is, at most n times in the first phase, and then n times in the second phase).
The number of other operations is also bounded by a polynomial in n.
Hence the running time is bounded by a polynomial in n, as required.

Problem 4

Recall the all-pairs shortest path problem. Consider the following way of solving it. Let $Q[i, j, k]$ denote the length of the shortest path from v_i to v_j that has **at most** k edges; if no such path exists, then set $Q[i, j, k]$ to ∞.
Give a polynomial time dynamic programming algorithm to compute the matrix of pairwise distances, using this subproblem formulation. Make sure to clearly explain your solution and why it works. Analyze the running time.

Solution to problem 4: Base cases

Let the input graph be $G = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ and let the edge weights be defined by $w : E \rightarrow R^+$, so that $w(e)$ is the weight of edge e.
We let i, j range from 1 to n, and k range from 1 to $n - 1$.
We let $Cost(P)$ denote the cost of path P (which is the sum of the weights of the edges in P).
The base cases are $k = 1$.

- If $i = j$ we set $Q[i, j, 1] = 0$.
- If $i \neq j$ and $(v_i, v_j) \in E$, we set $Q[i, j, 1] = w(v_i, v_j)$.
- If $i \neq j$ and $(v_i, v_j) \notin E$, we set $Q[i, j, 1] = \infty$

Solution to problem 4: Recursion

For $k > 1$ we use the following recursion:

- $Q[i, j, k] = \min\{Q[i, r, k-1] + Q[r, j, 1] : r \in \{1, 2, \ldots, n\}\}$

Recall $Q[i, j, k]$ it is the minimum cost of any path with **at most k edges** between v_i and v_j.
Similarly, $Q[i, j, k-1]$ is the minimum cost of any path with at most k-1 edges between v_i and v_j.
We need to show:

1. $Q[i, j, k] \leq \min\{Q[i, r, k-1] + Q[r, j, 1] : r \in \{1, 2, \ldots, n\}\}$
2. $Q[i, j, k] \geq \min\{Q[i, r, k-1] + Q[r, j, 1] : r \in \{1, 2, \ldots, n\}\}$

Solution to problem 4: Correctness for recursion

We begin by showing
$Q[i,j,k] \leq \min\{Q[i,r,k-1] + Q[r,j,1] : r \in \{1,2,\ldots,n\}\}$.
Let P be a minimum cost path from v_i to v_j using at most k edges.
Hence, $Q[i,j,k] = \mathit{Cost}(P)$.

So, suppose that

$$\min\{Q[i,r,k-1] + Q[r,j,1] : r \in \{1,2,\ldots,n\}\}$$

$$= Q[i,r^*,k-1] + Q[r^*,j,1].$$

Let P_1 be the path in G from v_i to v_{r^*} of length at most $k-1$ achieving cost $Q[i,r^*,k-1]$, and let P_2 be the path in G from v_{r^*} to v_j of length 1 achieving cost $Q[r^*,j,1]$.
Let W be the concatenation of P_1 and P_2.
Note $\mathit{Cost}(W) = \min\{Q[i,r,k-1] + Q[r,j,1] : r \in \{1,2,\ldots,n\}\}$.

Solution to problem 4: Correctness for recursion

Recall that W is the concatenation of P_1 and P_2, and that W is a walk from v_i to v_j satisfying
$Cost(W) = \min\{Q[i, r, k-1] + Q[r, j, 1] : r \in \{1, 2, \ldots, n\}\}$.
If W is a path (no repeated vertices), then we are done.
If W has repeated vertices, the only possibility is that $v_j \in P_1$.
Thus, let P_1' be the path obtained by truncating P_1 at v_j.
Then note:

$$Q[i, j, k] \leq Cost(P_1') < Cost(P_1) < Cost(W)$$

Solution to problem 4: Correctness for recursion

We have shown that $Q[i, j, k]$, the cost of a minimum cost path from v_i to v_j containing at most k edges, is at most $\min\{Q[i, r, k-1] + Q[r, j, 1] : r \in \{1, 2, \ldots, n\}\}$.
In other words, we have proven that
$Q[i, j, k] \leq \min\{Q[i, r, k-1] + Q[r, j, 1] : r \in \{1, 2, \ldots, n\}\}$

Solution to problem 4: Correctness for recursion

We now prove by contradiction that
$Q[i,j,k] \geq \min\{Q[i,r,k-1] + Q[r,j,1] : r \in \{1,2,\ldots,n\}\}$.
Suppose that
$Q[i,j,k] < \min\{Q[i,r,k-1] + Q[r,j,1] : r \in \{1,2,\ldots,n\}\}$.
In other words, suppose that the minimum cost path P from v_i to v_j using at most k edges satisfies

$$Cost(P) < \min\{Q[i,r,k-1] + Q[r,j,1] : r \in \{1,2,\ldots,n\}\}.$$

We will prove this cannot happen, by showing we can pick the vertex v_{r^*} so that

$$Cost(P) = Q[i,j,k] = Q[i,r^*,k-1] + Q[r^*,j,1]$$

$$\geq \min\{Q[i,r,k-1] + Q[r,j,1] : r \in \{1,2,\ldots,n\}\}.$$

Solution to problem 4: Correctness for recursion

We have two cases:

- P uses fewer than k edges
- P uses exactly k edges.

If P uses fewer than k edges, then its cost is equal to $Q[i, j, k-1]$, which is equal to $Q[i, j, k-1] + Q[j, j, 1]$.
Note that setting $r = j$ in the formula $Q[i, r, k-1] + Q[r, j, 1]$ gives $Q[i, j, k-1] + Q[j, j, 1] = Q[i, j, k-1]$.
Hence, for this case,

$$Q[i, j, k] = Q[i, j, k-1] + Q[j, j, 1]$$

$$\geq \min\{Q[i, r, k-1] + Q[r, j, 1] : r \in \{1, 2, \ldots, n\}\}.$$

Solution to problem 4: Correctness for recursion

Now assume the second case, so that P uses exactly k edges.

Since $k > 1$, there is a vertex v_{r^*} immediately preceeding v_j.

Hence, P can be written as the concatenation of two paths P_1 and P_2 where P_1 goes from v_i to v_{r^*} and P_2 goes from v_{r^*} to v_j.

Note that P_1 has exactly $k-1$ edges and P_2 is a single edge.

Furthermore, since P is a minimum cost path, the cost of P_1 is $Q[i, r^*, k-1]$.

Solution to problem 4: Correctness for recursion

Hence,

$$Cost(P) = Q[i, r^*, k-1] + Q[r^*, j, 1]$$

$$\geq \min\{Q[i, r, k-1] + Q[r, j, 1] : r \in \{1, 2, \ldots, n\}\}.$$

Therefore, when the minimum cost path P between v_i and v_j has exactly k edges,

$$Cost(P) \geq \min\{Q[i, r, k-1] + Q[r, j, 1] : r \in \{1, 2, \ldots, n\}\}.$$

Solution to problem 4: Correctness for recursion

We have shown that for all minimum cost paths P from v_i to v_j using at most k edges,

$$Cost(P) \geq \min\{Q[i, r, k-1] + Q[r, j, 1] : r \in \{1, 2, \ldots, n\}\}.$$

and

$$Cost(P) \leq \min\{Q[i, r, k-1] + Q[r, j, 1] : r \in \{1, 2, \ldots, n\}\}.$$

Hence,

$$Cost(P) = \min\{Q[i, r, k-1] + Q[r, j, 1] : r \in \{1, 2, \ldots, n\}\}.$$

Hence, the recursion is correct.

Solution to problem 4: Where is the final output?

How do we use the different $Q[i, j, k]$ values to compute the $n \times n$ matrix D of pairwise distances?

Note that the minimum cost of any path from v_i to v_j uses at most $n - 1$ edges.

Hence, $D[i, j] = Q[i, j, n - 1]$ for all i, j.

Therefore, we return the submatrix $Q[i, j, n - 1]$, letting i, j vary.

Solution to problem 4: Running Time

The algorithm computes all $Q[i, j, 1]$ using the base case rules.

Then it computes all $Q[i, j, k]$ for $k \geq 2$, starting with $k = 2$ and increasing k, until $k = n - 1$.

The running time to compute $Q[i, j, k]$ after the previous elements have been computed is easily seen to be $O(n)$.

There are $O(n^3)$ entries to compute, so this is an $O(n^4)$ algorithm.