CSE 347 Lecture 4

Divide-and-conquer and recurrences

1. Analysis of Prim & Kruskal for MST
2. Matrix multiplication — review divide-and-conquer
3. Strassen's algorithm

Last time, we saw two algorithms for the Minimum Spanning Tree problem — we are given a graph (V,E), where V is a set of vertices and E is a set of edges, joining pairs of vertices. We are also given positive costs $c(u,v)$, for each edge $(u,v) \in E$. These might be, for example, the distance between two nodes. We were seeking to find the set of edges of smallest total cost such that every pair of vertices is joined by a path using these edges. We saw that the solution to this problem never contains cycles, and hence we are actually seeking the Minimum Spanning Tree of the graph. We proved that Minimum Spanning Trees satisfy the following Cut Property:

Lemma: Suppose that all edge costs are distinct, and that $S \subsetneq V$ is a nonempty subset of vertices. Then the minimum cost $(u,v) \in E$ with $u \in S$, $v \notin S$ must be included in any minimum spanning tree.

We proved this by showing how such an edge can be exchanged for some other edge crossing out of S, reducing the cost of a tree. Now, let's see how this lemma can be used to analyze our greedy algorithms for finding minimum spanning trees. Let's first recall Prim's algorithm

Initialize $V_1 = \{v_0\}$ for some $v_0 \in V$, $T = \phi$.
For $i = 1, 2, \ldots, |V|-1$
  Choose e_i to be the edge (u_i, v_i) with $u_i \in V_i$ and $v_i \notin V_i$ of smallest cost.
  Put $T = T \cup \{(u_i, v_i)\}$, $V_{i+1} = V_i \cup \{v_i\}$

Theorem if all edge costs are distinct, Prim's Algorithm terminates with T equal to a minimum spanning tree.

Proof? By induction on i, the edges $e_1, \ldots, e_i$ are contained in any minimum spanning tree.

Base: $i = 0$ is an empty set of edges, which is trivially contained in any set.

Induction step: Consider the cut V_i. The Cut Property says that the minimum cost edge with $u \in V_i$ and $v \notin V_i$ must be in any minimum spanning tree. This is e_i. After $|V|-1$ steps, $V_{|V|} = V$, so T is the MST. ▨

It's easy to remove the restriction that the edge costs are distinct— we can "perturb" the

edge costs a little bit — say we have ordered the edges, and suppose that we increase the cost of the jth edge by $\frac{1}{j(j+1)} \left(= \frac{1}{j} - \frac{1}{j+1}\right)$, so that the total cost of any subset of the edges is increased by at most 1. Effectively, we are just breaking ties by preferring edges later in the ordering; the integer part of the cost is still equal to the original cost of any set of edges, but we now indeed have that the edge costs are distinct since they all have a different fractional part.

As for the running time of Prim's algorithm, suppose we implement it as follows: we maintain a priority queue of the edges with (at least) one endpoint in V_i, and a table for V indicating which vertices are in V_i. When we add a vertex to V_{i+1}, we add all of its neighbors to the queue, and to choose the next edge, we pull edges out until we find one with an endpoint outside V_i. Since we only consider adding an edge to the queue once for each endpoint, we only insert/remove $2|E|$ items, so this takes time $O(|E| \log |E|)$.

Now let's consider Kruskal's algorithm:

Initialize $T = \phi$
For $i = 1, 2, \ldots, |V|-1$
  Choose e_i to be the edge of smallest cost such

that $T \cup \{e_i\}$ does not contain a cycle.
Put $T \leftarrow T \cup \{e_i\}$

Theorem If all edge costs are distinct, Kruskal's algorithm terminates with T a minimum spanning tree.

Proof: Consider the edge $e_i = (u,v)$ chosen in some ith iteration of Kruskal's Algorithm. Since this edge does not create a cycle, there must *not* be a path from u to v in the set T so far. Consider $S = \{w \in V$: there is a path from u to w in $T\}$. Note that for any $w \in S$, $w' \notin S$, there is no path from w to w', or else we'd have $w' \in S$ also. Thus, no edge from any $w \in S$ to any $w' \notin S$ would create a cycle; Kruskal's algorithm's choice of e_i therefore means that e_i has the smallest cost among all edges crossing out of S. The Cut Property thus implies that e_i is in the minimum spanning tree. To see that the final T is a spanning tree, we observe that first, we never add an edge that would create a cycle, and second, with each edge we add, we connect two previously disconnected components, reducing their number by one. Since initially there are $|V|$, after $|V|-1$ iterations, there is only one connected component, which thus contains all of V. ☒

Kruskal's algorithm can be implemented as fast as Prim's, but this requires a new data structure. We'll return to it later. For now, note that the same problem may have more than one good algorithm.

Matrix Multiplication

Multiplying two matrices is a fundamental task that turns out to be a computational bottleneck in many applications. Recall that the product C of matrices A ($\ell \times m$) and B ($m \times n$) is an $\ell \times n$ matrix given by $c_{ij} = \sum_{k=1}^{m} A_{ik} \cdot B_{kj}$. This suggests a naive $O(\ell m n)$ algorithm for the task. Perhaps surprisingly, it's possible to do much better.

For simplicity, let's consider the case where $\ell = m = n$, which is a power of 2. First, consider the following recursive algorithm: we break A & B into four minors, each containing half as many rows and columns.

$$\begin{bmatrix} A^{(1,1)} & A^{(1,2)} \\ A^{(2,1)} & A^{(2,2)} \end{bmatrix} \begin{bmatrix} B^{(1,1)} & B^{(1,2)} \\ B^{(2,1)} & B^{(2,2)} \end{bmatrix} = \begin{bmatrix} C^{(1,1)} & C^{(1,2)} \\ C^{(2,1)} & C^{(2,2)} \end{bmatrix}$$

Notice: $C^{(1,1)} = (A^{(1,1)} B^{(1,1)}) + (A^{(1,2)} B^{(2,1)})$, since each entry (i,j) in $A^{(1,1)} B^{(1,1)}$ contains $\sum_{k=1}^{n/2} A_{ik} B_{kj}$, and in $A^{(1,2)} B^{(2,1)}$ contains $\sum_{k=n/2+1}^{n} A_{ik} B_{kj}$. In general,

$$\begin{bmatrix} C^{(1,1)} & C^{(1,2)} \\ C^{(2,1)} & C^{(2,2)} \end{bmatrix} = \begin{bmatrix} A^{(1,1)}B^{(1,1)} + A^{(1,2)}B^{(2,1)} & A^{(1,1)}B^{(1,2)} + A^{(1,2)}B^{(2,2)} \\ A^{(2,1)}B^{(1,1)} + A^{(2,2)}B^{(2,1)} & A^{(2,1)}B^{(1,2)} + A^{(2,2)}B^{(2,2)} \end{bmatrix}$$

So, by computing these 8 $\frac{n}{2} \times \frac{n}{2}$ matrix products and

4 matrix sums (which take time $O(n^2)$ each), we obtain a recursive algorithm with running time given by the recurrence $T(n) = 8 \cdot T(\frac{n}{2}) + O(n^2)$, $T(1) = O(1)$.

What does the tree of recursive calls look like?

level 0 - size n — 1 node — 8 — work per call $O(n^2)$

level i - size $\frac{n}{2^i}$ — 8^i nodes — $O(\frac{n^2}{2^{2i}})$

level $\log_2 n$ - size 1 — $8^{\log_2 n} = n^3$ nodes — $O(1)$

Total time? $\sum_{i=0}^{\log_2 n} c \cdot 2^{3i} \cdot (\frac{n}{2^i})^2 = \sum_{i=0}^{\log_2 n} c \cdot 2^i \cdot n^2 = c \cdot n^2 (2n-1)$

$O(n^3)$ — no improvement so far.

Suppose now that, somehow, we could use only seven recursive multiplication steps, instead of eight.

We'd obtain a total running time of

$$\sum_{i=0}^{\log_2 n} c \cdot 7^i (\frac{n}{2^i})^2 = cn^2 \sum_{i=0}^{\log_2 n} (\frac{7}{4})^i = cn^2 \frac{(7/4)^{\log_2 n + 1} - 1}{7/4 - 1}$$

$$= cn^2 \frac{4}{3}(n^{\log_2 7/4} - 1) \leq O(n^{2.81})$$

Which is even better. Strassen showed that for

$P_1 = A^{(1,1)}(B^{(1,2)} - B^{(2,2)})$, $P_2 = (A^{(1,1)} + A^{(1,2)})B^{(2,2)}$

$P_3 = (A^{(2,1)} + A^{(2,2)})B^{(1,1)}$, $P_4 = A^{(2,2)}(B^{(2,1)} - B^{(1,1)})$

$P_5 = (A^{(1,1)} + A^{(2,2)})(B^{(1,1)} + B^{(2,2)})$, $P_6 = (A^{(1,2)} - A^{(2,2)})(B^{(2,1)} + B^{(2,2)})$

$P_7 = (A^{(1,1)} - A^{(2,1)})(B^{(1,1)} + B^{(1,2)})$, we actually do get

$P_5 + P_4 - P_2 + P_6 = C^{(1,1)}$, $P_1 + P_2 = C^{(1,2)}$

$P_3 + P_4 = C^{(2,1)}$, $P_5 + P_1 - P_3 - P_7 = C^{(2,2)}$ (verify.)

So Strassen obtains a $O(n^{2.81})$ time algorithm for matrix multiplication. A similar algorithm by

Karatsuba earlier showed that the multiplication algorithm you learned in grade school ($O(n^2)$ time for n digit numbers) can also be improved (see the book).

Confirming Strassen — we join terms that cancel out:

$$(A^{(1,1)}+A^{(2,2)})(B^{(1,1)}+B^{(2,2)})+A^{(2,2)}(B^{(2,1)}-B^{(1,1)})-(A^{(1,1)}+A^{(1,2)})B^{(2,2)}$$
$$+(A^{(1,2)}-A^{(2,2)})(B^{(2,1)}+B^{(2,2)})$$
$$= A^{(1,1)}B^{(1,1)}+A^{(1,2)}B^{(2,1)} = C^{(1,1)} \checkmark$$

$$A^{(1,1)}(B^{(1,2)}-B^{(2,2)})$$
$$+(A^{(1,1)}+A^{(1,2)})B^{(2,2)} = A^{(1,1)}B^{(1,2)}+A^{(1,2)}B^{(2,2)} = C^{(1,2)} \checkmark$$

$$(A^{(2,1)}+A^{(2,2)})B^{(1,1)}$$
$$+A^{(2,2)}(B^{(2,1)}-B^{(1,1)}) = A^{(2,1)}B^{(1,1)}+A^{(2,2)}B^{(2,1)} = C^{(2,1)} \checkmark$$

$$(A^{(1,1)}+A^{(2,2)})(B^{(1,1)}+B^{(2,2)})+A^{(1,1)}(B^{(1,2)}-B^{(2,2)})$$
$$-(A^{(2,1)}+A^{(2,2)})B^{(1,1)}-(A^{(1,1)}-A^{(2,1)})(B^{(1,1)}+B^{(1,2)}) = A^{(2,2)}B^{(2,2)}+A^{(2,1)}B^{(1,2)}$$
$$= C^{(2,2)} \checkmark$$