1 From Last Lecture

We created an efficient \((2k - 1)\)-distance oracle that uses \(\tilde{O}(k \cdot n^{1+1/k})\) space and answers a query in \(O(k)\) time. Recall that we constructed sets \(A_i\) for \(0 \leq i \leq k - 1\) by setting \(A_0 = V\) and sampling \(A_i\) from \(A_{i-1}\). Then, for any vertex \(v \in V\), we defined \(p_i(v)\) to be the closest node in \(A_i\) to \(v\), with the additional requirement that if \(d(v, p_i(v)) = d(v, p_{i+1}(v))\) we must set \(p_i(v) = p_{i+1}(v)\). Finally, we defined \(B_i(v) = \{x \in A_i | d(v, x) < d(v, p_{i+1}(v))\}\) for \(0 \leq i \leq k - 2\) and \(B(v) = A_{k-1} \cup \left( \bigcup_{i=0}^{k-2} B_i(v) \right)\). We proved that \(|B(v)| = \tilde{O}(k \cdot n^{1/k})\) and, for all \(i\), \(p_i(v) \in B(v)\). For the query algorithm, we store \(d(v, x)\) for all \(v \in V\) and \(x \in B(v)\).

Algorithm 1: Query\((u, v)\)

\[
\begin{align*}
&w \leftarrow p_0(v) = v; \\
&\text{for } i = 1 \rightarrow k \text{ do} \\
&\quad //w = p_{i-1}(v) \in B(v); \\
&\quad \text{if } w \in B(u) \text{ then} \\
&\quad \quad \text{return } d(u, w) + d(w, v); \\
&\quad \text{else} \\
&\quad \quad w \leftarrow p_i(u); \\
&\quad \quad \text{swap } u \text{ and } v;
\end{align*}
\]

In the last lecture, we showed that Query\((u, v)\) returns a \((2k - 1)\)-approximation of the distance.

2 A \((4k - 3)\)-approximation

Now suppose we do not swap \(u\) and \(v\) at the end of each iteration, and set \(w \leftarrow p_i(v)\) at each iteration. In other words, we are trying the find the minimum \(i^*\) such that \(p_{i^*}(u) \in B(u)\), and return \(d(u, p_{i^*}(v)) + d(p_{i^*}(v), v)\) as the distance estimate. We call this algorithm Query\(^*\)((u, v)\). This is useful in contexts where we only have local information (as we will see in Compact Routing).

We show that Query\(^*\)((u, v)\) actually returns a \((4k - 3)\)-approximation.

**Theorem 2.1.** Let \(i^*\) be the minimum value such that \(p_{i^*}(v) \in B(u)\). Then \(d(u, p_{i^*}(v)) + d(p_{i^*}(v), v) \leq (4k - 3) \cdot d(u, v)\).

**Proof.** We start by showing that \(d(v, p_i(v)) \leq 2i \cdot d(u, v)\) for all \(i \leq i^*\), by an inductive argument. The inequality is trivially true for \(i = 0\) as \(p_0(v) = v\) and thus \(d(v, p_0(v)) = 0 \leq 2 \cdot 0 \cdot d(u, v)\).

Suppose it is true for \(i < i^*\). Then for \(i + 1\):

\[
\begin{align*}
d(v, p_{i+1}(v)) &\leq d(v, p_{i+1}(u)) \\
&\leq d(v, u) + d(u, p_{i+1}(u)) \\
&\leq d(v, u) + d(u, p_i(v)) \\
&\leq d(v, u) + d(u, v) + d(v, p_i(v)) \\
&\leq 2(i + 1) \cdot d(v, u).
\end{align*}
\]
The first line follows from the definition of \( p_{i+1}(v) \) and the fact that \( p_{i+1}(u) \in A_{i+1} \). The second and fourth lines follow from triangle inequality. The third line follows from the definition \( i^* \): we know that for all \( i < i^* \), \( p_i(v) \notin B(u) \), which implies \( d(u, p_i(v)) \geq d(u, p_{i+1}(u)) \). Finally, the last line follows from the inductive hypothesis.

Therefore, we have

\[
\begin{align*}
\quad & d(u, p_{i^*}(v)) + d(p_{i^*}(v), v) \\
\leq & (d(u, v) + d(v, p_{i^*}(v)) + d(p_{i^*}(v), v) \\
\leq & (4i^* + 1)d(u, v),
\end{align*}
\]

where the first line follows from triangle inequality, and the second line follows by plugging the result of the inductive argument. Since \( i^* \leq k - 1 \), we get that \( d(u, p_{i^*}(v)) + d(p_{i^*}(v), v) \leq (4k - 3)d(u, v) \).

\[\square\]

3 Compact Routing

The ideas for the \((4k - 3)\)-approximation distance oracle can be used for compact routing, which we will define below. We have a graph \( G = (V, E) \) and every node \( v \in V \) has a routing table \( R(v) \). Each node receives packets that arrive with a header of information, including \( L(u) \) - the label of the destination node \( u \). The node then looks at its routing table \( R(v) \) and decides which neighbor to send the packet to.

We want to design a method that stores small \( R(v) \) and \( L(u) \) for each node, while achieving short (i.e. close to optimal) paths for each packet.

We will eventually show the following theorem.

**Theorem 3.1.** For any constant integer \( k \geq 2 \), there is a compact routing protocol with \( O(\log n^2) \) bit labels, \( \tilde{O}(n^{1/k}) \) bit routing tables, and gives a \((4k - 3)\)-approximation of the shortest path.

The compact routing scheme in Theorem 3.1 takes polynomial preprocessing time, but we won’t cover it in this lecture.

Let’s consider a first attempt. We will compute the distance oracle as before. For each vertex \( v \in V \), \( R(v) \) will store \( p_i(v) \) for all \( i \) and, for all \( x \in B(v) \), the next node in the shortest path from \( v \) to \( x \). And for each vertex \( u \), the label will be \( L(u) = \{u, p_0(u), \ldots, p_{k-1}(u)\} \). We thus have \(|R(v)| \sim |B(v)| = \tilde{O}(kn^{1/k})\) and the bit complexity of \( L(u) \) is \( O(k \log n) \), both of which are pretty good!

We now discuss how to decide which node to route an incoming packet to. Suppose node \( u \) gets a packet sending to \( v \). We run the algorithm without swapping as described in the previous section (i.e. we try each \( p_i(v) \) and check if they are in \( B(u) \)). More formally:

**Algorithm 2:** NextNode\(_u(v)\)

\[
\text{for } i = 0 \to k - 1 \text{ do }
\]

\[
\quad \text{if } p_i(v) \in B(u) \text{ then }
\]

\[
\quad \quad \text{Send packet to next node on shortest path from } u \text{ to } p_i(v);
\]

This algorithm has several issues in its implementation.

1. Suppose we route from \( u \) to \( u' \) where \( u' \) is the next node after \( u \) on the shortest path to \( p_{i^*}(v) \). What happens if \( p_{i^*}(v) \notin B(u') \)?

2. Suppose the current node \( x \) is \( p_{i^*}(v) \). How do we route down to \( v \)?

As it turns out, problem 1 will never happen due to the following lemma.

**Lemma 3.1.** If \( p_{i^*}(v) \in B(u) \) and \( u' \) is a node on the shortest path from \( u \) to \( p_{i^*}(v) \), then \( p_{i^*}(v) \in B(u') \).
Proof. Note \( p_\ast(v) \in B(u) \) implies \( d(u, p_\ast(v)) < d(u, p_{j+1}(u)) \). But

\[
d(u, u') + d(u', p_\ast(v)) = d(u, p_\ast(v)) < d(u, p_{j+1}(u)) \leq d(u, p_{j+1}(u')) \leq d(u, u') + d(u', p_{j+1}(u')),
\]

where the first equality follows by the definition of \( u' \), the first \( \leq \) inequality follows by the definition of \( p_{j+1}(u) \) and the fact that \( p_{j+1}(u') \in A_{j+1} \), and the last inequality follows from triangle inequality. Therefore,

\[
d(u', p_\ast(v)) < p(u', p_{j+1}(u')) \implies p_\ast(v) \in B(u').
\]

Now we consider problem 2. Firstly, notice that the route from \( p_\ast(v) \) to \( v \) is along the edges of the shortest path tree rooted at \( p_\ast(v) \), so we will try to make use of compact routing protocols on trees.

**Theorem 3.2.** Suppose there is a compact routing protocol for routing on an \( n \)-node tree \( T \) with labels \( L_T(v) \) of size \( \ell(n) \) and route tables \( R_T(v) \) of size \( r(n) \), then there is a compact routing protocol for general graphs with \( O(n^{1/k_T(n)}) \) size route tables and \( O(\ell(n)) \) size labels, and it routes on a \((4k-3)\)-approximation shortest path.

**Proof.** For every node \( x \) in the graph, let’s compute a shortest path tree \( T_x \) rooted at \( x \). Suppose that there is some scheme that given a tree \( T \), creates labels \( L_T(u) \) for each node \( u \in T \) and routing tables \( R_T(u) \) for each node \( u \in T \), so that given \( L_T(v) \) and \( R_T(x) \), \( x \) can route to the next node \( x' \) on the tree path from \( x \) to \( v \) in \( T \).

Then, we can augment our routing scheme for general graphs as follows:

**The labels** \( L(u) \). The label \( L(v) \) of \( v \) contains \( p_0(v), \ldots, p_{k-1}(v) \) and for each \( j \in \{0, \ldots, k-1\} \) the label \( L_{T_{p_j(v)}}(v) \) for the node \( v \) in the shortest paths tree rooted at \( p_j(v) \).

**The routing tables** \( R(u) \). The routing table \( R(u) \) of \( u \) contains \( B(u) \) and contains for each \( x \in B(u) \), the routing table \( R_{T_x}(u) \).

**Routing.** Suppose node \( u \) wants to send a packet to \( v \). First it looks at \( L(v) \) and finds the minimum \( i^* \) such that \( p_{i^*}(v) \in B(u) \). It can do this since \( B(u) \) is in \( R(u) \). Now let \( y = p_{i^*}(v) \) and let \( T = T_y \) be the shortest paths tree rooted at \( y \). First, \( u \) writes \( y \) and \( L(v) \) in the header of the message. Then \( u \) accesses \( L_T(v) \) from \( L(u) \) and \( R_T(u) \) from \( R(u) \) and uses these to route to the next node \( u' \) on the shortest path from \( u \) to \( v \) in \( T \). From then on, each node that gets the message learns \( L(v) \) and \( y \), and can access \( R_{T_y}(x) \) since by Lemma 3.1 above and Lemma 3.2 below, \( y \in B(x) \). Thus every node on the path \( u \rightarrow p_{i^*}(v) = y \rightarrow v \) can route to the next node of the path until \( v \) is reached. By the results from the previous section, we are guaranteed to route along a \((4k-3)\)-approximate path.

**Lemma 3.2.** Suppose that \( x \) lies on the shortest path between \( p_{i^*}(v) \) and \( v \). Then \( p_{i^*}(v) \in B(x) \).

**Proof.** Let \( J \) be the largest index such that \( p_{i^*}(v) \in A_J \). Then in particular \( p_J(v) = p_{i^*}(v) \) and \( p_{J+1}(v) \neq p_{i^*}(v) \). Suppose that \( p_J(v) = p_{i^*}(v) \notin B(x) \). Then, \( d(x, p_J(v)) \geq d(x, p_{J+1}(x)) \) by the definition of \( B(x) \) and \( J \). However, then we have

\[
d(v, p_{J+1}(v)) \leq d(v, p_{J+1}(x)) \leq d(v, x) + d(x, p_{J+1}(x)) \leq d(v, x) + d(x, p_J(v)) = d(v, p_J(v)).
\]

On the other hand, since \( J + 1 \geq J \), we must have \( d(v, p_{J+1}(v)) \geq d(v, p_J(v)) \). Therefore, \( d(v, p_{J+1}(v)) = d(v, p_J(v)) \). However, the definition of \( p_J(v) \) requires us to set \( p_J(v) = p_{J+1}(v) \) when \( d(v, p_{J+1}(v)) = d(v, p_J(v)) \), so we must have \( p_J(v) = p_{J+1}(v) \), a contradiction.

\[\square\]

\[\square\]
4 Routing on trees

Here we show that one can route along a tree \( T \) with routing tables of size \( O(\log n) \) bits and labels of size \( O(\log^2 n) \) bits. Together with Theorem 3.2, it proves Theorem 3.1.

Let \( T \) be a tree rooted at a node \( r \).

**First Attempt.** Let us perform a DFS traversal of \( T \) and label the nodes in DFS traversal order. Identify each node with its DFS number. Observe that after doing so, the nodes in any subtree of \( T \) can be represented as a consecutive interval of integers as follows. For each \( x \in T \), let \( f(x) \) denote the descendant of \( x \) with largest DFS label. Then \( x \) is on the shortest path from the root to any node in \( [x, f(x)] \). Hence, one potential protocol is as follows. Suppose we are at \( x \) and we want to route to \( y \). Then, search for a child \( c \) of \( x \) such that \( v \in [c, f(c)] \). Once we find such a child \( c \), we will route to \( c \).

However, since a node can have \( \Omega(n) \) children, that can make some of our routing tables huge. Therefore, we need to find some way to reduce the number of edges to children that we store for each node.

**Second Attempt** For \( x \in T \) and for each child \( x' \) of \( x \), denote \( x' \) a “light” child of \( x \) if the subtree rooted at \( x' \) contains at most half the nodes in the subtree rooted at \( x \). Otherwise, denote \( x' \) as a “heavy” child. Observe that each node \( x \) has at most 1 heavy child. Also, any path along \( T \) from a node \( x \) to some descendant \( v \) can contain at most \( O(\log n) \) light children since at each light child the number of descendents halves. We will make use of these two properties.

For each \( x \in T \), we store \( x, f(x), h(x), f(h(x)) \) and its parent \( p(x) \) in \( R_T(x) \). Now, when we route a packet to \( v \) we first check if \( v \in [x, f(x)] \) and if not, we route to the parent \( p(x) \) of \( x \) that we store in \( R_T(x) \). If \( v \in [x, f(x)] \), we check if \( v \in [h(x), f(h(x))] \). If so, then we can route to the heavy child \( h(x) \) that we access from \( R_T(x) \). Otherwise, we conclude that the next node on the path to \( v \) is a light child of \( x \). We will get this light child from \( L_T(v) \).

Now we describe what we store in \( L_T(v) \). Let \( e_1, e_2, \ldots, e_t \) be all the light children along the path from the root \( r \) to \( v \). We will store the pairs \( (p(e_i), e_i) \) in our header \( L_T(v) \). Now suppose that as above we are at \( x \) and have established that the path to \( v \) goes through a light child of \( x \). Then we search through \( L_T(v) \) for a pair \( (p(e_i), e_i) \) where \( x = p(e_i) \). If \( v \) is a descendent of \( x \) but not of \( h(x) \) then this pair will be in \( L_T(v) \). Then we route to \( e_i \).

By the two properties that we described, the size of \( R_T(x) \) for each \( x \) is just 4 integers, \( x, f(x), h(x), p(x) \) each having \( O(\log n) \) bits, thus taking \( O(\log n) \) space. The size of \( L_T(x) \) is \( O(\log n) \) integer pairs, which takes \( O(\log^2 n) \) space as promised.

Below, we give the steps to the entire tree routing algorithm.

**Algorithm 3:** Route\((x, L(v))\)

```plaintext
if v is not contained in [x, f(x)] then
    Route(p(x), L(v));
    Report done;
else
    if x = v then
        Report done;
    if x has heavy child then
        Let h(x) be heavy child of x;
        if v is contained in [h(x), f(h(x))] then
            Route(h(x), L(v));
            Report done;
        Find pair (p(e_i), e_i) in L(v) such that p(e_i) = x;
        Route(e_i, L(v));
        Report done;
```