DS-563/CS-543 2025-02-13

Logistics:

- HW 2 out, late deadline extended, no late penalty
  [Question: Anyone using something else than Python or Rust?]
- Tuesday 18: Monday schedule, no lecture
- Will try to schedule additional office hours

Today:

- A few more comments about estimating the number of distinct elements (see previous notes)
- Main topic: Graph connectivity sketches

1 Problem

Input: a stream describing a graph G on $V = [n]$

Question: Is G connected?

We consider two versions of the problem: insertion only (the input stream is a sequence of edges that are never deleted) and insertion–deletion (the input stream is a sequence of updates of the form "`insert` (u, v)" and "`delete` (u, v)" with no edge deleted before it is inserted).

2 Insertion–only streams

We keep a subset F of edges that is a spanning forest of the graph we have seen so far. Initially, $F = \emptyset$. For every edge (u, v) that we see, if u and v are already connected by F, we do nothing. Otherwise, we add this edge to F. At the end of the stream, G is connected if and only if all vertices are connected by F.

Space usage: $O(n)$ words of space, because F consists of at most $n - 1$ edges.

Note: For fast processing, instead of storing explicitly F, you can use the union–find data structure.

3 Insertion–deletions streams

3.1 First attempts

- Sample edges and see what has not been deleted: the graph might become very dense and then have lots of deletions, so this won't work.
- Is a graph that is very sparse at the end of the stream the worst case then? Not really. We have not covered this topic in detail, but if the final graph has k edges, then it can be fully recovered, using $O(k \operatorname{polylog}(n))$ words of space. This can be achieved using a set of techniques known as *sparse recovery* (aka. *compressed sensing*).

3.2 Overview

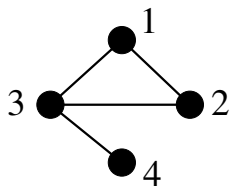
We obtain our solution by combining three ingredients, which we introduce one by one:

1. **An encoding of a graph:** We express the graph as a matrix with $\Theta(n^3)$ $\{-1, 0, 1\}$–entries. Each row of the matrix expresses the connectivity of a single vertex. While this encoding may seem wasteful, it has properties which allow for expressing the connectivity of any set of vertices to its complement by a linear combination of rows of the matrix.
2. **Borůvka's algorithm:** Two popular algorithms for finding the minimum spanning trees are Prim's and Kruskal's algorithms. Borůvka's algorithm is another algorithm for this problem, which guarantees more "parallelism." We will see that it is a crucial property, which will allows for constructing a spanning tree or forest for the input graph in a small number of non-adaptive rounds.
3. **ℓ_0–sampling:** We won't cover exact implementation details of this technique here, but it allows for compressing high–dimensional vectors into a low–dimensional linear sketch that is sufficient for recovering at least one non-zero entry with high probability. We apply this to rows of our graph's encoding to significantly reduce the amount of information we need to store.

3.3 Ingredient 1: The graph's encoding and its properties

We encode adjacency lists of every vertex as a vector of length $\binom{n}{2}$. Every entry corresponds to a single pair of vertices in $V = [n]$ and is indexed by (unordered) pairs $\{j, j'\}$, where $j, j' \in V$ and are different. For a given vertex $i \in V$, we create a vector x_i, such that $(x_i)_{\{j,j'\}}$, the entry indexed by $\{j, j'\}$, is non-zero if and only if $i \in \{j, j'\}$ and $\{j, j'\}$ is present in the graph. In other words, the entry corresponding to a specific edge is non-zero if this edge is incident on i and present in the graph. If this entry is non-zero, it is either -1 or 1. More specifically, $(x_i)_{\{i,j\}} = -1$ if $j < i$ and $(x_i)_{\{i,j\}} = 1$ if $j > i$.

Example:



$$
\begin{array}{cccccccccc}
 & & & \{1,2\} & \{1,3\} & \{1,4\} & \{2,3\} & \{2,4\} & \{3,4\} & \\
x_1 & = & (& 1 & 1 & 0 & 0 & 0 & 0 &) \\
x_2 & = & (& -1 & 0 & 0 & 1 & 0 & 0 &) \\
x_3 & = & (& 0 & -1 & 0 & -1 & 0 & 1 &) \\
x_4 & = & (& 0 & 0 & 0 & 0 & 0 & -1 &)
\end{array}
$$

Note that most entries are 0. If a specific edge is not present in the graph, all entries in the column corresponding to this edge are zero. Otherwise, only two of them are non-zero, i.e., those corresponding to the endpoints of the edge. Moreover, one of them is -1 and the other one is 1.

[To be continued...]