Some more probability

**Lemma**

\[ E_1, \ldots, E_n: n \text{ events (not necessarily independent). Then,} \]

\[ \Pr \left[ \bigcap_{i=1}^n E_i \right] = \Pr[E_1] \ast \Pr[E_2 \mid E_1] \ast \Pr[E_3 \mid E_1 \cap E_2] \ast \ldots \ast \Pr[E_n \mid E_1 \cap \ldots \cap E_{n-1}] . \]

Proof...

**Lemma**

\[ \Pr[X_i = 1] = 1/i. \]

**Proof.**

1. Fix elements appearing in \( \text{set}(P_i) = \{s_1, \ldots, s_i\} \).
2. \[ \Pr \left[ p_i = \min(P_i) \bigg| \text{set}(P_i) = S \right] = 1/i. \]

\[
\begin{align*}
\Pr \left[ p_i = \min(P_i) \right] &= \sum_{S \subseteq P, |S| = i} \Pr \left[ p_i = \min(P_i) \bigg| \text{set}(P_i) = S \right] \Pr[S] \\
&= \sum_{S \subseteq P, |S| = i} \frac{1}{i} \Pr[S] = \frac{1}{i}.
\end{align*}
\]
Theorem
In a random permutation of \( n \) distinct numbers, the minimum of the prefix changes in expectation is \( \ln n + 1 \) times.

Proof.
1. \( Y = \sum_{i=1}^{n} X_i \).
2. \( E[Y] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} 1/i \leq \ln n + 1. \)

Proof continued...
1. For any indices \( 1 \leq i_1 < i_2 < \ldots < i_k \leq n \), and observe that \( \Pr[\mathcal{E}_{i_k} | \mathcal{E}_{i_1} \cap \ldots \cap \mathcal{E}_{i_{k-1}}] = \Pr[\mathcal{E}_{i_k}] \),
2. because \( \mathcal{E}_{i_k} \) determined after all \( \mathcal{E}_{i_1}, \ldots, \mathcal{E}_{i_{k-1}} \).
3. By induction: \( \Pr[\mathcal{E}_{i_2} \cap \ldots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_2} | \mathcal{E}_{i_1} \cap \ldots \cap \mathcal{E}_{i_k}] \Pr[\mathcal{E}_{i_3} \cap \ldots \cap \mathcal{E}_{i_k}] = \ldots = \prod_{j=1}^{k} \Pr[\mathcal{E}_{i_j}] = \prod_{j=1}^{k} \frac{1}{i_j} . \)
4. \( \Rightarrow \) variables \( X_1, \ldots, X_n \) are independent.
5. Result readily follows from Chernoff’s inequality. □
Grids...

1. **r**: Side length of grid cell.
2. Grid cell IDed by pair of integers.
3. Constant time to determine a point \( p \)'s grid cell id:
   \[
   \text{id}(p) = (\lfloor \frac{p_x}{r} \rfloor, \lfloor \frac{p_y}{r} \rfloor)
   \]
4. Limited use of the floor function (but no word packing tricks).
5. Use hashing on (grid) points.
6. Store points in grid...
   ...in linear time.

Storing point set in grid/hash-table...

**Hashing:**
1. Non-empty grid cells
2. For non-empty grid cell:
   List of points in it.
3. For a grid cell:
   Its neighboring cells.

Closet pair in a square

**Lemma**
Let \( P \) be a set of points contained inside a square \( \square \), such that the sidelength of \( \square \) is \( \alpha = \text{CP}(P) \). Then \( |P| \leq 4 \).

**Proof.**
Partition \( \square \) into four equal squares \( \square_1, \ldots, \square_4 \).
Each square diameter \( \sqrt{2\alpha}/2 < \alpha \).
... contain at most one point of \( P \); that is, the disk of radius \( \alpha \) centered at a point \( p \in P \) completely covers the subsquare containing it; see the figure on the right.
\( P \) can have four points if it is the four corners of \( \square \).

Verify closet pair

**Lemma**
P: set of \( n \) points in the plane. \( \alpha \): distance. Verify in linear time whether \( \text{CP}(P) < \alpha \), \( \text{CP}(P) = \alpha \), or \( \text{CP}(P) > \alpha \).

**Proof.**
Indeed, store the points of \( P \) in the grid \( G_\alpha \). For every non-empty grid cell, we maintain a linked list of the points inside it. Thus, adding a new point \( p \) takes constant time. Specifically, compute \( \text{id}(p) \), check if \( \text{id}(p) \) already appears in the hash table, if not, create a new linked list for the cell with this ID number, and store \( p \) in it. If a linked list already exists for \( \text{id}(p) \), just add \( p \) to it. This takes \( O(n) \) time overall.
Now, if any grid cell in \( G_\alpha(P) \) contains more than, say, 4 points of \( P \), then it must be that the \( \text{CP}(P) < \alpha \), by previous lemma.
1. When insert a point $p$: fetch all the points of $P$ in cluster of $P$.
2. Takes constant time.
3. If there is a point closer to $p$ than $\alpha$ that was already inserted, then it must be stored in one of these 9 cells.
4. Now, each one of those cells must contain at most 4 points of $P$ by prev lemma.
5. Otherwise, already stopped since $CP(\cdot) < \alpha$.

---

1. $S$ set of all points in cluster.
2. $|S| \leq 9 \cdot 4 = O(1)$.
3. Compute closest point to $p$ in $S$. $O(1)$ time.
4. If $d(p, S) < \alpha$, we stop; otherwise, continue to next point.
5. Correctness: `$CP(P) < \alpha$' returned only if such pair found.

---

1. Assume $p$ and $q$: realizing closest pair.
2. $\|p - q\| = CP(P) < \alpha$.
3. When later point (say $p$) inserted, the set $S$ would contain $q$.
4. Algorithm would stop and return `$CP(P) < \alpha$'.
5. $\blacksquare$
Weak analysis...

Lemma
Let $t$ be the number of different values in the sequence $\alpha_2, \alpha_3, \ldots, \alpha_n$. Then $E[t] = O(\log n)$. As such, in expectation, the above algorithm rebuilds the grid $O(\log n)$ times.

proof
1. $X_i = 1 \iff \alpha_i < \alpha_{i-1}$.
2. $E[X_i] = Pr[X_i = 1]$ and $t = \sum_{i=3}^{n} X_i$.
3. $Pr[X_i = 1] = Pr[\alpha_i < \alpha_{i-1}]$.
4. Backward analysis. Fix $P_i$.
5. $q \in P_i$ is critical if $CP(P_i \setminus \{q\}) > CP(P_i)$.
6. No critical points, then $\alpha_{i-1} = \alpha_i$ and then $Pr[X_i = 1] = 0$.

Proof continued...
1. If one critical point, then $Pr[X_i = 1] = 1/i$.
2. Assume two critical points and let $p, q$ be this unique pair of points of $P_i$ realizing $CP(P_i)$.
3. $\alpha_i < \alpha_{i-1} \iff p$ or $q$ is $p_i$.
4. $Pr[X_i = 1] = 2/i$.
5. Cannot be more than two critical points.
6. Linearity of expectations: $E[t] = E[\sum_{i=3}^{n} X_i] = \sum_{i=3}^{n} E[X_i] \leq \sum_{i=3}^{n} 2/i = O(\log n)$.

Expected linear time analysis...

Theorem
$\mathcal{P}$: set of $n$ points in the plane. Compute the closest pair of $\mathcal{P}$ in expected linear time.

Proof.
1. $X_i = 1 \iff \alpha_i \neq \alpha_{i-1}$.
2. Running time is proportional to $R = 1 + \sum_{i=3}^{n} (1 + X_i \cdot i)$.
3. $E[R] = E[1 + \sum_{i=3}^{n} (1 + X_i \cdot i)] \leq n + \sum_{i=3}^{n} E[X_i] \cdot i \leq n + \sum_{i=3}^{n} i \cdot \frac{2}{i} \leq 3n$, by linearity of expectation and since $E[X_i] = Pr[X_i = 1] \leq 2/i$.
4. Expected running time of the algorithm is $O(E[R]) = O(n)$.

Nets
The Main Tool

$r$-net $\mathcal{N} \subseteq \mathcal{P}$ is an $r$-net if

- Every point in $\mathcal{P}$ has distance $< r$ to a point in $\mathcal{N}$
- For any two $p, q \in \mathcal{N}$, we have $d(p, q) \geq r$. 

![Net Diagram]
Computing an r-net

Application of Grids: Computing nets
...in linear time

Repeatedly:
1. Pick any unmarked point.
2. Mark all neighbors in distance in an r-grid.

(A) Neighbors in distance are in neighboring cells.
(B) Neighboring Cells found in $O(1)$ time.
(C) Cells contain lists of points.

Input
1. $G = (V, E)$: edge-weighted.
2. $n$ vertices and $m$ edges.
3. Task: compute an ordering $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ of vertices.
4. $\forall v \in V \quad L_v$:
   $\pi_i \in L_v$ $\iff \pi_i$ closest vertex to $v$ in prefix $\langle \pi_1, \ldots, \pi_i \rangle$.
5. Example: Streaming scenario - install servers in a network. 
   ... every client in t network needs to know its closest server.
6. ... client needs to maintain its current closest server.
7. How min total size of lists? $L = \sum_{v \in V} |L_v|$.

Algorithm
1. $\pi_1, \ldots, \pi_n$ : random permutation of $V$ of $G$.
2. $\forall v \in V \quad \delta(v) = +\infty$.
3. For $i = 1$ to $n$ do:
   3.1 $\delta(\pi_i)$ to 0.
   3.2 start Dijkstra from the $i$th vertex $\pi_i$.
   3.3 Dijkstra propagates to $u$ only if improves current distance.
   3.4 Update $\delta(u)$ to $d_G(\pi_i, u) \iff d_G(\pi_i, u) < \delta'(u)$
      $\delta'(u)$: value before this iteration started.
   3.5 If $\delta(u)$ updated: add $\pi_i$ to $L_u$. 
Performance

Lemma
Algorithm computes a permutation \( \pi \), such that:

1. \( E[|L|] = O(n \log n) \).
2. Expected running time \( O\left((n \log n + m) \log n\right) \).
3. \( n = |V(G)| \) and \( m = |E(G)| \).
4. Both bounds also hold with high probability.

Proof
1. Fix a vertex \( v \in V = \{v_1, \ldots, v_n\} \).
2. \( U = \{d_G(v, v_1), \ldots, d_G(v, v_n)\} \): random permutation of \( U \).
3. By lemma seen \( \pi \) min changes \( O(\log n) \) times in expectations + high prob.
4. \( |L_v| = O(\log n) \) in expectation + high probability.
5. Running time:
   6.1 For \( uv \in E(G) \): \( \delta(u) \) or \( \delta(v) \) changes \( O(\log n) \) times.
   6.2 \( uv \) gets visited \( O(\log n) \) times by all "Disjkstras",
   6.3 Overall running time \( O(n \log^2 n + m \log n) \):
      6.3.1 \( O(n \log n) \) changes in \( \delta(\cdot) \).
      6.3.2 \( n \): delete-min operations
      6.3.3 Edge triggers \( O(\log n) \) decrease-key operations.
      6.3.4 \( \text{time} (\text{decrease-key}) = O(1) \)
         \( \text{time} (\text{delete-min}) = O(\log n) \).
      (Fibonacci heaps).

Computing \( r \)-net in sparse graphs.

1. \( G = (V, E) \) be a weighted graph with \( n \) vertices and \( m \) edges, let \( r > 0 \).
2. \( \pi_i \): \( i \)-th vertex in a random permutation of \( V \).
3. \( \forall v \in V : \delta(v) := +\infty \).
4. Test whether \( \delta(\pi_i) \geq r \), if so:
   4.1 Add \( \pi_i \) to the resulting net \( \mathcal{N} \).
   4.2 Set \( \delta(\pi_i) \) to zero.
   4.3 Perform Dijkstra's algorithm starting from \( \pi_i \),
      Occupy a vertex \( u \) only if improve distance: \( \delta(u) \).
   4.4 If a vertex \( u \) is expanded: \( \delta(u) \): computed distance
      from \( \pi_i \), and relax the edges adjacent to \( u \) in the graph.

Correctness

Lemma
The set \( \mathcal{N} \) is an \( r \)-net in \( G \).

Proof.
1. End: \( \forall v \in V : \delta(v) < r \).
2. By induction: if \( \ell = \delta(v) \), for some vertex \( v \), then the
   distance of \( v \) to the set \( \mathcal{N} \) is at most \( \ell \).
3. Every two points in \( \mathcal{N} \) have distance \( \geq r \). Indeed, when
   the algorithm handles vertex \( v \in \mathcal{N} \), its distance from all
   the vertices currently in \( \mathcal{N} \) is \( \geq r \).
**Lemma**

Consider an execution of the algorithm, and any vertex \( v \in V \). The expected number of times the algorithm updates the value of \( \delta(v) \) during its execution is \( O(\log n) \), and more strongly the number of updates is \( O(\log n) \) with high probability.

**Proof...**

1. Assume all distances in \( G \) are distinct.
2. \( S_i \): set of all vertices \( x \in V \), such that:
   - (A) \( d(x, v) < d(v, \Pi_i) \), where \( \Pi_i = \{\pi_1, \ldots, \pi_i\} \).
   - (B) If \( \pi_{i+1} = x \) then \( \delta(v) \) would change in the \((i + 1)\)th iteration.
3. \( s_i = |S_i| \). Observe \( S_1 \supseteq S_2 \supseteq \cdots \supseteq S_n \), and \( |S_n| = 0 \).
4. \( E_{i+1} \): event that \( \delta(v) \) changed in iteration \((i + 1)\) (active iteration).
5. \((i + 1)\) iteration active: \( \pi_{i+1} \in S_i \).
6. \( \pi_{i+1} \): uniform distribution over the vertices of \( S_i \).

**Proof continued...**

1. \( E_{i+1} \) happens then \( s_{i+1} \leq s_i / 2 \), with probability \( \geq 1/2 \).
2. iteration is **lucky**.
3. After \( O(\log n) \) lucky iterations set \( S_i \) empty: Done.
4. \( E_1, \ldots, E_n \): Independent.
5. By Chernoff inequality, after \( c \log n \) active iterations, at least \( \lceil \log_2 n \rceil \) iterations lucky. with high probability.  

**Correctness continued...**

**Lemma**

Given a graph \( G = (V, E) \), with \( n \) vertices and \( m \) edges, the above algorithm computes an \( r \)-net of \( G \) in \( O((n + m) \log n) \) expected time.

**Proof.**

By above lemma, the two \( \delta \) values associated with the endpoints of an edge get updated \( O(\log n) \) times, in expectation, during the algorithm’s execution. As such, a single edge creates \( O(\log n) \) decrease-key operations in the heap maintained by the algorithm. Each such operation takes constant time if we use Fibonacci heaps to implement the algorithm.