Persistent Data Structures (Version Control)

Ephemeral
- V0
  - update
  - version list
- V1
- V2
- V3
- V4
- V5
- V6
  - query

Partial persistence
- V0
  - version tree
- V1
- V2
- V3
- V4
- V5
- V6
  - query

Full persistence
- V0
  - version tree
- V1
- V2
- V3
- V4
- V5
- V6
  - query

Confluently persistence
- V0
  - version DAG
- V1
- V2
- V3
- V4
- V5
  - update/merge/query all versions

Purely functional
- car cdr
  - never modify
  - only create new pairs
  - only DAGs

Retroactive
- V0
  - update & query all versions
- V1
  - updates in the past propagate
Retroactive Data Structures


\[ \begin{align*}
\text{m} & \quad \text{Total number of updates/versions} \\
\text{r} & \quad \text{Distance from current time} \\
\text{n} & \quad \text{Maximal data structure size at any time}
\end{align*} \]

**Partial retroactive**  Update all versions & **query current**

**Full retroactive**  Update & query all versions
Rollback $\rightarrow$ Full Retroactivity

**Theorem 3.1.** Given any data structure that performs a collection of operations each in worst case $T(n)$ time, there is a corresponding retroactive data structure that supports the same operations in $O(T(n))$ time, and supports retroactive versions of those operations in $O(rT(n))$ time.

+ Generic, can always be applied, space efficient
- Slow retroactive operations
Lower bounds for Retroactivity

**Theorem 3.2.** There exists a data structure in the straight-line-program model, supporting $O(1)$ time update operations, but the (partially) retroactive insertions of those operations require $\Omega(r)$ time, worst case or amortized.

\[
a_0 + a_1x + a_2x^2 + \cdots + a_nx^n
\]

( requires $\Omega(n)$ multiplications given $x$ by Motzkin’s theorem )

**Theorem 3.3.** In the cell-probe model, there exists a data structure supporting partially retroactive updates in $O(1)$ time, but fully retroactive queries of the past require $\Omega(\log n / \log \log n)$ time.

\[
\begin{array}{ccccccc}
x_0 & x_1 & x_2 & \cdots & x_i & \cdots & x_n
\end{array}
\]

( prefix sum queries require $\Omega(\log n)$ )

Theorem 3.4. Any partially retroactive data structure in the pointer-machine model with constant indegree, supporting $T(m)$-time retroactive updates and $Q(m)$-time queries about the present can be transformed into a fully retroactive data structure with amortized $O(\sqrt{m}T(m))$-time retroactive updates and $O(\sqrt{m}T(m) + Q(m))$-time fully retroactive queries using $O(mT(m))$ space.
Partial Retroactive Commutative Data Structures

**Lemma 4.1.** Any data structure supporting a commutative set of operations allows the retroactive insertion of operations in the past (and queries in the present) at no additional asymptotic cost.

**Lemma 4.2.** Any data structure supporting a commutative and invertible set of operations can be made partially retroactive at no additional asymptotic cost.

**Lemma 4.3.** Any data structure for a searching problem can be made partially retroactive at no additional asymptotic cost.

\textit{commutative} = state independent of order of operations
Theorem 4.1. Any data structure for a decomposable searching problem supporting insertions, deletions, and queries in time $T(n)$ and space $S(n)$ can be transformed into a fully retroactive data structure with all operations taking time $O(T(m))$ if $T(m) = \Omega(n^e)$ for some $e > 0$, or $O(T(n) \log m)$ otherwise. The space used is $O(S(m) \log m)$. 
## Specific Retroactive Data Structures

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Partially Retroactive</th>
<th>Fully Retroactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary (exact)</td>
<td>$O(\log m)$</td>
<td>$O(\log m)$</td>
</tr>
<tr>
<td>Dictionary (successor)</td>
<td>$O(\log m)$</td>
<td>$O(\log^2 m)$</td>
</tr>
<tr>
<td>Queue</td>
<td>$O(1)$</td>
<td>$O(\log m)$</td>
</tr>
<tr>
<td>Stack</td>
<td>$O(\log m)$</td>
<td>$O(\log m)$</td>
</tr>
<tr>
<td>DEQUE</td>
<td>$O(\log m)$</td>
<td>$O(\log m)$</td>
</tr>
<tr>
<td>Union/Find *</td>
<td>$O(\log m)$</td>
<td>$O(\log m)$</td>
</tr>
<tr>
<td>Priority Queue</td>
<td>$O(\log m)$</td>
<td>$O(\sqrt{m \log m})$</td>
</tr>
</tbody>
</table>