Useful facts

Order Notation Summary:

- \( f(n) \in O(g(n)) \) if \( \exists c > 0 \) and \( n_0 \geq 0 \) such that \( |f(n)| \leq c |g(n)| \) \( \forall n \geq n_0 \).
- \( f(n) \in \Omega(g(n)) \) if \( \exists c > 0 \) and \( n_0 \geq 0 \) such that \( |f(n)| \geq c |g(n)| \) \( \forall n \geq n_0 \).
- \( f(n) \in \Theta(g(n)) \) if \( \exists c_1, c_2 > 0 \) and \( n_0 \geq 0 \) such that \( c_1 |g(n)| \leq |f(n)| \leq c_2 |g(n)| \) \( \forall n \geq n_0 \).
- \( f(n) \in o(g(n)) \) if \( \forall c > 0 \) \( \exists n_0 \geq 0 \) such that \( |f(n)| \leq c |g(n)| \) \( \forall n \geq n_0 \).
- \( f(n) \not\in \Omega(g(n)) \) if \( \forall c > 0 \) \( \exists n_0 \geq 0 \) such that \( |f(n)| \geq c |g(n)| \) \( \forall n \geq n_0 \).

Some useful sums:

\[
\begin{align*}
\sum_{i=1}^{n} i &= \frac{n(n+1)}{2} \\
\sum_{i=1}^{n} i^2 &= \frac{n(n+1)(2n+1)}{6} \\
\sum_{i=1}^{n} i^3 &= \frac{n^2(n+1)^2}{4} \\
\sum_{i=1}^{n} \frac{1}{i} &= \ln n + \gamma + o(1) \in \Theta(\log n)
\end{align*}
\]

Some useful facts:

\[
\begin{align*}
\sum_{i=0}^{\infty} \frac{1}{2^i} &= 2 \\
\sum_{i=0}^{\infty} \frac{i}{2^i} &= 2 \\
\sum_{i=0}^{n} 2^i &= 2^{n+1} - 1 \\
\sum_{i=1}^{\infty} \frac{1}{i^2} &= \frac{\pi^2}{6}
\end{align*}
\]

Some well-known sequences:

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power of 2, ( 2^n )</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td>Factorial ( n! )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>24</td>
<td>120</td>
<td>720</td>
<td>5040</td>
<td>40320</td>
<td>362880</td>
</tr>
<tr>
<td>Fibonacci number ( F(n) )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>13</td>
<td>21</td>
<td>34</td>
</tr>
<tr>
<td>Catalan-number ( C(n) )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>14</td>
<td>42</td>
<td>132</td>
<td>429</td>
<td>1430</td>
<td>4862</td>
</tr>
</tbody>
</table>

Randomization, probability and moments:

- \( \text{random}(\text{int } n) \) returns an integer in \{0, \ldots, n-1\}, chosen uniformly.
- \( E[aX] = aE[X] \), \( E[X+Y] = E[X] + E[Y] \) (linearity of expectation)
- \( V[X] = V[a+X] \)
- Chebyshev’s inequality: \( P(|X - E[X]| \geq t) \leq \frac{V(X)}{t^2} \)

Some recursions that we have seen:

<table>
<thead>
<tr>
<th>Recursion</th>
<th>resolves to</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) = T(n/2) + \Theta(1) )</td>
<td>( T(n) \in \Theta(\log n) )</td>
</tr>
<tr>
<td>( T(n) = 2T(n/2) + \Theta(n) )</td>
<td>( T(n) \in \Theta(n \log n) )</td>
</tr>
<tr>
<td>( T(n) = 2T(n/2) + \Theta(\log n) )</td>
<td>( T(n) \in \Theta(n \log n) )</td>
</tr>
<tr>
<td>( T(n) = T(cn) + \Theta(n) ) for some ( 0 &lt; c &lt; 1 )</td>
<td>( T(n) \in \Theta(n) )</td>
</tr>
<tr>
<td>( T(n) = \frac{1}{2} T(\frac{n}{2}) + \frac{1}{2} T(n-1) + \Theta(1) )</td>
<td>( T(n) \in \Theta(\log n) )</td>
</tr>
<tr>
<td>( T(n) = \frac{1}{n} \sum_{i=0}^{n-1} \max{T(i), T(n-i-1)} + \Theta(n) )</td>
<td>( T(n) \in \Theta(n) )</td>
</tr>
<tr>
<td>( T(n) = \frac{2}{n} \sum_{i=0}^{n-1} T(i) + \Theta(n) )</td>
<td>( T(n) \in \Theta(n \log n) )</td>
</tr>
<tr>
<td>( T(n) = \frac{1}{n} \sum_{i=2}^{n-1} T(i) )</td>
<td>( T(n) \in \Theta(n^3) )</td>
</tr>
<tr>
<td>( T(n) = T(\sqrt{n}) + \Theta(\sqrt{n}) )</td>
<td>( T(n) \in \Theta(\sqrt{n}) )</td>
</tr>
<tr>
<td>( T(n) = T(\sqrt{n}) + \Theta(1) )</td>
<td>( T(n) \in \Theta(\log \log n) )</td>
</tr>
</tbody>
</table>
Some definitions that we have seen:

- $F[j] = \text{length } \ell \text{ of the longest prefix of } P \text{ that is a suffix of } P[1..j]$
- $F^+[j] = \begin{cases} \text{length } \ell \text{ of the longest prefix of } P \text{ that is a suffix of } P[1..j], \text{ and } P[j+1] \neq P[\ell]. \\ 0 \text{ if there is no such } \ell. \end{cases}$
- $L[c] = \begin{cases} \text{maximal index } j \text{ with } P[j] = c \\ -1 \text{ if there is no such } j. \end{cases}$
- $S[j] = \text{max } \{\text{index } \ell : P[j+1..m-1] \text{ is a prefix of } P^*[\ell+1..m-2]\}.$
- Elias-Gamma code $E(k) = 0^{\lceil \log k \rceil} + (k)_2$

Some slides:

### Range search data structures summary
- **Quadtrees**
  - simple (also for dynamic set of points)
  - work well only if points evenly distributed
  - wastes space for higher dimensions
- **kd-trees**
  - linear space
  - range search time $O(\sqrt{n} + s)$
  - inserts/deletes destroy balance and range search time (no simple fix)
- **range-trees**
  - range search time $O(\log^2 n + s)$
  - wastes some space
  - inserts/deletes destroy balance (can fix this with occasional rebuild)

**Conventional:** Points on split lines belong to right/top side.

### Compression summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Run-length encoding</th>
<th>Lempel-Ziv-Welch</th>
<th>bzip2 (uses Burrows-Wheeler)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huffman</td>
<td>variable-length</td>
<td>fixed-length</td>
<td>multi-step</td>
</tr>
<tr>
<td>single-character</td>
<td>multi-character</td>
<td>multi-character</td>
<td>multi-step</td>
</tr>
<tr>
<td>2-pass, must send dictionary</td>
<td>1-pass</td>
<td>1-pass</td>
<td>not streamable</td>
</tr>
<tr>
<td>60% compression on English text</td>
<td>bad on text</td>
<td>45% compression on English text</td>
<td>70% compression on English text</td>
</tr>
<tr>
<td>optimal 01-prefix-code</td>
<td>good on long runs (e.g., pictures)</td>
<td>good on English text</td>
<td>better on English text</td>
</tr>
<tr>
<td>requires uneven frequencies</td>
<td>requires runs</td>
<td>requires repeated substrings</td>
<td>requires repeated substrings</td>
</tr>
<tr>
<td>rarely used directly</td>
<td>rarely used directly</td>
<td>frequently used</td>
<td>used but slow</td>
</tr>
<tr>
<td>part of pkzip, JPEG, MP3</td>
<td>GIF, some variants of PDF, compress</td>
<td>bzip2 and variants</td>
<td></td>
</tr>
</tbody>
</table>

### String Matching Conclusion

- Our algorithms stopped once they have found one occurrence.
- Most of them can be adapted to find all occurrences within the same worst-case run-time.

### Extendible hashing

We can save links (hence space in internal memory) with two tricks:
- Expand the trie so that all leaves have the same **global depth** $d_T$.
- Store only the leaves, and in an array $D$ of size $2^{d_T}$.

---

**Convention:** Points on split lines belong to right/top side.

---

**Some definitions that we have seen:**

- $F[j] = \text{length } \ell \text{ of the longest prefix of } P \text{ that is a suffix of } P[1..j]$.
- $F^+[j] = \begin{cases} \text{length } \ell \text{ of the longest prefix of } P \text{ that is a suffix of } P[1..j], \text{ and } P[j+1] \neq P[\ell]. \\ 0 \text{ if there is no such } \ell. \end{cases}$
- $L[c] = \begin{cases} \text{maximal index } j \text{ with } P[j] = c \\ -1 \text{ if there is no such } j. \end{cases}$
- $S[j] = \text{max } \{\text{index } \ell : P[j+1..m-1] \text{ is a prefix of } P^*[\ell+1..m-2]\}.$
- Elias-Gamma code $E(k) = 0^{\lceil \log k \rceil} + (k)_2$

**Some slides:**

### Range search data structures summary

- **Quadtrees**
  - simple (also for dynamic set of points)
  - work well only if points evenly distributed
  - wastes space for higher dimensions
- **kd-trees**
  - linear space
  - range search time $O(\sqrt{n} + s)$
  - inserts/deletes destroy balance and range search time (no simple fix)
- **range-trees**
  - range search time $O(\log^2 n + s)$
  - wastes some space
  - inserts/deletes destroy balance (can fix this with occasional rebuild)

**Convention:** Points on split lines belong to right/top side.

### Compression summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Run-length encoding</th>
<th>Lempel-Ziv-Welch</th>
<th>bzip2 (uses Burrows-Wheeler)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huffman</td>
<td>variable-length</td>
<td>fixed-length</td>
<td>multi-step</td>
</tr>
<tr>
<td>single-character</td>
<td>multi-character</td>
<td>multi-character</td>
<td>multi-step</td>
</tr>
<tr>
<td>2-pass, must send dictionary</td>
<td>1-pass</td>
<td>1-pass</td>
<td>not streamable</td>
</tr>
<tr>
<td>60% compression on English text</td>
<td>bad on text</td>
<td>45% compression on English text</td>
<td>70% compression on English text</td>
</tr>
<tr>
<td>optimal 01-prefix-code</td>
<td>good on long runs (e.g., pictures)</td>
<td>good on English text</td>
<td>better on English text</td>
</tr>
<tr>
<td>requires uneven frequencies</td>
<td>requires runs</td>
<td>requires repeated substrings</td>
<td>requires repeated substrings</td>
</tr>
<tr>
<td>rarely used directly</td>
<td>rarely used directly</td>
<td>frequently used</td>
<td>used but slow</td>
</tr>
<tr>
<td>part of pkzip, JPEG, MP3</td>
<td>GIF, some variants of PDF, compress</td>
<td>bzip2 and variants</td>
<td></td>
</tr>
</tbody>
</table>

### String Matching Conclusion

- Our algorithms stopped once they have found one occurrence.
- Most of them can be adapted to find all occurrences within the same worst-case run-time.

### Extendible hashing

We can save links (hence space in internal memory) with two tricks:
- Expand the trie so that all leaves have the same **global depth** $d_T$.
- Store only the leaves, and in an array $D$ of size $2^{d_T}$.