1. A don’t care character # is a special character that matches any single character. For example, the pattern #oke# matches sokeri, pokeri and tokeni.

(a) Modify the Shift-And algorithm to handle don’t care characters.

**Solution:** Compute the symbol bitvectors so that, at the position of a don’t care character, there is a 1-bit in every bit vector:

- **Preprocess:**
  1. For \( c \in \Sigma \cup \{\#\} \) do \( B[c] \leftarrow 0 \)
  2. For \( i \leftarrow 0 \) to \( m - 1 \) do \( B[P[i]] \leftarrow B[P[i]] + 2^i \)
  3. For \( c \in \Sigma \) do \( B[c] \leftarrow B[c] | B[\#] \)
  4. \( B[\#] \leftarrow 2^m - 1 \)

The last line adds support for don’t care characters in the text.

(b) It may appear that the Morris–Pratt algorithm can handle don’t care characters almost without change: Just make sure that the character comparisons are performed correctly when don’t care characters are involved. However, such an algorithm would be incorrect. Give an example demonstrating this.

**Solution:** Let \( P = \#oke\#i \) and \( T = jokekeri \). Now \( \text{fail}[5] = 2 \) because \( \#o \) (a prefix of \( P[0..5] \)) matches \( e\# \) (a suffix of \( P[0..5] \)). Then the text search proceeds as follows:

```
jokerkeri
    #oke#i
  #oke#i
```

producing an incorrect match at position 3. On the other hand, setting \( \text{fail}[5] \) to a smaller value (a larger value doesn’t make sense) would be incorrect too, because then the algorithm would miss the occurrence eokeri in the text \( T = jokeokeri \).

The root of the problem is that the Morris–Pratt algorithm is based on the assumption that the matching relation is transitive but don’t care characters break the transitivity. In the example, the algorithm finds that \( r \) matches \( \# \) and \( \# \) matches \( o \), and assumes that this implies that \( r \) matches \( o \).

2. Let \( P_k = \{P_1, \ldots, P_{2k}\} \) be a set of patterns such that

- for \( i \in [1..k] \), \( P_i = a^i \) and
- for \( i \in [k + 1..2k] \), \( P_i = P'_i a^k \) such that \( |P'_i| = k \) and each \( P'_i \) is different.

(a) Show that the total size of the sets \( \text{patterns}(\cdot) \) in the Aho–Corasick automaton for \( P_k \) is asymptotically larger than \( |P_k| \).

**Solution:**

For \( j \in [k + 1..2k] \) and \( d \in [1..k] \), let \( u_{jd} \) be the node of the AC automaton representing \( P'_jd^d \) (a prefix of \( P_j \)). Then \( \{1, \ldots, d\} \subseteq \text{patterns}(u_{jd}) \) and \( |\text{patterns}(u_{jd})| \geq d \). Thus

\[
\sum_{j \in [k+1..2k], d \in [1..k]} |\text{patterns}(u_{jd})| \geq k \frac{k(k + 1)}{2} = \Omega(k^3)
\]

while \( |P_k| = 2k^2 + \frac{k(k+1)}{2} = O(k^2) \).
(b) Describe how to represent the sets \( \text{patterns}(\cdot) \) so that

- the total space complexity is never more than \( O(|\mathcal{P}|) \) for any \( \mathcal{P} \)
- each set \( \text{patterns}(\cdot) \) can be listed in linear time in its size.

**Solution:**

Let \( \text{pat}(v) \) be as in the trie for \( \mathcal{P} \):

\[
\text{pat}(v) = i \text{ if } S_v = P_i \text{ and } \text{pat}(v) = \perp \text{ if there is no such } i.
\]

Let \( \text{next}(v) = u \) such that \( S_u \) is the longest proper suffix of \( S_v \) with \( \text{pat}(u) \neq \perp \) and \( \text{next}(v) = \perp \) if there is no such \( u \).

The following code computes \( \text{patterns}(v) \) in linear time of its size.

1. \( R \leftarrow \emptyset \)
2. if \( \text{pat}(v) \neq \perp \) then \( R \leftarrow \{ \text{pat}(v) \} \)
3. \( v \leftarrow \text{next}(v) \)
4. while \( v \neq \perp \) do
5. \( R \leftarrow R \cup \{ \text{pat}(v) \} \)
6. \( v \leftarrow \text{next}(v) \)
7. return \( R \)

The space requirement is constant for each node.

3. Show that edit distance is a **metric**, i.e., that it satisfies the metric axioms:

- \( ed(A, B) \geq 0 \)
- \( ed(A, B) = 0 \) if and only if \( A = B \)
- \( ed(A, B) = ed(B, A) \) (symmetry)
- \( ed(A, C) \leq ed(A, B) + ed(B, C) \) (triangle inequality)

**Solution:**

(a) The number of edit operations cannot be negative.
(b) No edit operations are required if and only if \( A = B \).
(c) For every edit operation, the inverse edit operation exists (and has the same cost). For every set of edit operations changing \( A \) into \( B \), the set of inverse operations changes \( B \) into \( A \) (and has the same cost). In particular, this holds for the minimal set of edit operations with the cost \( ed(A, B) \), which shows that \( ed(B, A) \leq ed(A, B) \). By symmetry, \( ed(A, B) \leq ed(B, A) \), and thus \( ed(A, B) = ed(B, A) \).
(d) Suppose we change \( A \) into \( C \) in two stages by first changing \( A \) to \( B \) and then \( B \) to \( C \). If we use the minimal set of edit operations in each stage, the total number of edits is \( ed(A, B) + ed(B, C) \). Thus \( ed(A, C) \leq ed(A, B) + ed(B, C) \).

4. Let \( \Sigma = \{a, b, c\} \). Define the function \( \gamma : \Sigma \times \Sigma \rightarrow \mathbb{R}_{\geq 0} \) as follows

\[
\gamma(a, a) = \gamma(b, b) = \gamma(c, c) = 0 \\
\gamma(a, b) = \gamma(b, c) = \gamma(c, a) = 0.5 \\
\gamma(b, a) = \gamma(c, b) = \gamma(a, c) = 1.5
\]

Let \( ed_\gamma \) be a **weighted edit distance**, where the cost of substituting a character \( x \) with a character \( y \) is \( \gamma(x, y) \). The cost of insertions and deletions is 1.
(a) It might seem that we can compute $ed_\gamma(A, B)$ using the recurrence for the standard edit distance (page 117 on the lecture nodes) except $\delta$ is replaced by $\gamma$. Show that this is not the case by providing an example for which the recurrence produces an incorrect distance.

**Solution:**

Let $A = a$ and $B = c$. Then $ed_\gamma(A, B) \leq 1$ (we can do two substitutions $a \to b \to c$) while the algorithm from the lectures will return the answer 1.5.

The algorithm for the standard edit distance $ed$ does not correctly compute $ed_\gamma$ because the above $\gamma$ does not satisfy the triangle equality: $1.5 = \gamma(a, c) \not\leq \gamma(a, b) + \gamma(b, c) = 1$. As a result, it might be cheaper to perform several substitutions to change one letter into another, than just do a single replacement.

The algorithm from the lecture could, however, be used to compute $ed_\gamma$ if, instead of replacing $\delta$ with $\gamma$, we replace $\delta$ with $\gamma'$, where $\gamma'(a, b)$, for any $a, b \in \Sigma$, is the minimal cost of changing $a$ into $b$ by performing a sequence of substitutions. The values of $\gamma'$ can be obtained by computing the shortest paths between all pairs of vertices in a complete digraph $G$, where $V(G) = \Sigma$ and $\gamma$ is the edge weight function.

(b) Is $ed_\gamma$ a metric?

**Solution:**

$ed_\gamma$ is not a metric, because it does not satisfy the symmetry axiom. Let $A = a$, $B = c$.

**Claim.** $ed_\gamma(A, B) \neq ed_\gamma(B, A)$.

**Proof.** We can transform $B$ into $A$ by replacing $c$ with $a$ (cost 0.5), thus $ed_\gamma(B, A) \leq 0.5$.

We will now show that $ed_\gamma(A, B) \geq 1$. Any sequence of edit operations consisting of only substitutions either contains just one nontrivial (other that $x \to x$ for some $x \in \Sigma$) replacement (in which case it must be $a \to c$ having cost 1.5) or more. In the latter case, the total cost is $\geq 1$ since the cheapest nontrivial replacement has cost 0.5. On the other hand, if we use at least one indel operation the total cost will also add up to $\geq 1$.

5. Let $P = evete$ and $T = neeteneeveteen$. Use Ukkonen’s cut-off algorithm to find the occurrences of $P$ in $T$ for $k = 1$.

**Solution:**

<table>
<thead>
<tr>
<th>neeteneeveteen</th>
<th>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>1 1 0 0 1 0 1 0 0 1 0 1 0 0 1</td>
</tr>
<tr>
<td>v</td>
<td>2 2 1 1 1 1 1 1 1 0 1 1 1 1 1</td>
</tr>
<tr>
<td>e</td>
<td>1 2 1 2 1 1 1 0 1 1 1 1 2</td>
</tr>
<tr>
<td>t</td>
<td>2 1 2 2 2 2 1 0 1 2 2</td>
</tr>
<tr>
<td>e</td>
<td>1 2</td>
</tr>
</tbody>
</table>