Approximate String Matching

Now we are ready to tackle the main problem of this part: approximate string matching.

**Problem 2.7:** Given a text $T[1..n]$, a pattern $P[1..m]$ and an integer $k \geq 0$, report all positions $j \in [1..m]$ such that $ed(P, T(j - \ell...j)) \leq k$ for some $\ell \geq 0$.

The factor $T(j - \ell...j)$ is called an approximate occurrence of $P$.

There can be multiple occurrences of different lengths ending at the same position $j$, but usually it is enough to report just the end positions. We ask for the end position rather than the start position because that is more natural for the algorithms.
Define the values $g_{ij}$ with the recurrence:

\[
g_{0j} = 0, \quad 0 \leq j \leq n, \\
g_{i0} = i, \quad 1 \leq i \leq m, \quad \text{and} \\
g_{ij} = \min \left\{ \begin{array}{ll}
g_{i-1,j-1} + \delta(P[i],T[j]) \\
g_{i-1,j} + 1 \\
g_{i,j-1} + 1 \end{array} \right. \\
1 \leq i \leq m, 1 \leq j \leq n. 
\]

**Theorem 2.8:** For all $0 \leq i \leq m$, $0 \leq j \leq n$:

\[
g_{ij} = \min \{ ed(P[1..i],T(j-\ell...j)) \mid 0 \leq \ell \leq j \}.
\]

In particular, $j$ is an ending position of an approximate occurrence if and only if $g_{mj} \leq k$. 
**Proof.** We use induction with respect to \( i + j \).

**Basis:**
\[
\begin{align*}
g_{00} &= 0 = ed(\epsilon, \epsilon) \\
g_{0j} &= 0 = ed(\epsilon, \epsilon) = ed(\epsilon, T(j - 0..j)) \quad \text{(min at } \ell = 0) \\
g_{i0} &= i = ed(P[1..i], \epsilon) = ed(P[1..i], T(0 - 0..0)) \quad (0 \leq \ell \leq j = 0)
\end{align*}
\]

**Induction step:** Essentially the same as in the proof of Theorem 2.2.
Example 2.9: \( P = \text{match}, \ T = \text{remachine}, \ k = 1 \)

<table>
<thead>
<tr>
<th>g</th>
<th>r e m a c h i n e</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>m</td>
<td>1 1 1 0 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>a</td>
<td>2 2 2 1 0 1 2 2 2 2 2 2</td>
</tr>
<tr>
<td>t</td>
<td>3 3 3 2 1 1 2 3 3 3 3</td>
</tr>
<tr>
<td>c</td>
<td>4 4 4 3 2 1 2 3 4 4</td>
</tr>
<tr>
<td>h</td>
<td>5 5 5 4 3 2 1 2 3 4</td>
</tr>
</tbody>
</table>

One occurrence ending at position 6.
Algorithm 2.10: Approximate string matching

Input: text $T[1..n]$, pattern $P[1..m]$, and integer $k$

Output: end positions of all approximate occurrences of $P$

1. for $i \leftarrow 0$ to $m$ do $g_{i0} \leftarrow i$
2. for $j \leftarrow 1$ to $n$ do $g_{0j} \leftarrow 0$
3. for $j \leftarrow 1$ to $n$ do
4.     for $i \leftarrow 1$ to $m$ do
5.         $g_{ij} \leftarrow \min\{g_{i-1,j-1} + \delta(A[i], B[j]), g_{i-1,j} + 1, g_{i,j-1} + 1\}$
6.     if $q_{mj} \leq k$ then output $j$

- Time and space complexity is $O(mn)$.
- The space complexity can be reduced to $O(m)$ by storing only one column as in Algorithm 2.5.
Ukkonen’s Cut-off Heuristic

We can speed up the algorithm using the diagonal monotonicity of the matrix \((g_{ij})\):

A diagonal \(d, -m \leq d \leq n\), consists of the cells \(g_{ij}\) with \(j - i = d\). Every diagonal in \((g_{ij})\) is monotonically increasing.

Example 2.11: Diagonals -3 and 2.

<table>
<thead>
<tr>
<th>g</th>
<th>r e m a c h i n e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>m</td>
<td>1 1 1 0 1 1 1 1 1 1</td>
</tr>
<tr>
<td>a</td>
<td>2 2 2 1 0 1 2 2 2 2</td>
</tr>
<tr>
<td>t</td>
<td>3 3 3 2 1 2 3 3 3 3</td>
</tr>
<tr>
<td>c</td>
<td>4 4 4 3 2 1 2 3 4 4</td>
</tr>
<tr>
<td>h</td>
<td>5 5 5 4 3 2 1 2 3 4</td>
</tr>
</tbody>
</table>
More specifically, we have the following property.

**Lemma 2.12:** For every $i \in [1..m]$ and every $j \in [1..n]$, $g_{ij} = g_{i-1,j-1}$ or $g_{ij} = g_{i-1,j-1} + 1$.

**Proof.** By definition, $g_{ij} \leq g_{i-1,j-1} + \delta(P[i], T[j]) \leq g_{i-1,j-1} + 1$. We show that $g_{ij} \geq g_{i-1,j-1}$ by induction on $i + j$.

The induction assumption is that $g_{pq} \geq g_{p-1,q-1}$ when $p \in [1..m]$, $q \in [1..n]$ and $p + q < i + j$. At least one of the following holds:

1. $g_{ij} = g_{i-1,j-1} + \delta(P[i], T[j])$. Then $g_{ij} \geq g_{i-1,j-1}$.

2. $g_{ij} = g_{i-1,j} + 1$ and $i > 1$. Then

$$g_{ij} = g_{i-1,j} + 1 \geq g_{i-2,j-1} + 1 \geq g_{i-1,j-1}$$

3. $g_{ij} = g_{i,j-1} + 1$ and $j > 1$. Then

$$g_{ij} = g_{i,j-1} + 1 \geq g_{i-1,j-2} + 1 \geq g_{i-1,j-1}$$

4. $i = 1$. Then $g_{ij} \geq 0 = g_{1,j-1}$.

$g_{ij} = g_{i,j-1} + 1$ and $j = 1$ is not possible because $g_{i,1} \leq g_{i-1,0} + 1 < g_{i,0} + 1$. □
We can reduce computation using diagonal monotonicity:

- Whenever the value on a column $d$ grows larger than $k$, we can discard $d$ from consideration, because we are only interested in values at most $k$ on the row $m$.

- We keep track of the smallest undiscarded diagonal $d$. Each column is computed only up to diagonal $d$.

**Example 2.13:** $P = \text{match}$, $T = \text{remachine}$, $k = 1$

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>e</th>
<th>m</th>
<th>a</th>
<th>c</th>
<th>h</th>
<th>i</th>
<th>n</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$m$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$m$</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$m$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$m$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$m$</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

58
The position of the smallest undiscarded diagonal on the current column is kept in a variable $top$.

**Algorithm 2.14:** Ukkonen’s cut-off algorithm

*Input:* text $T[1..n]$, pattern $P[1..m]$, and integer $k$

*Output:* end positions of all approximate occurrences of $P$

1. for $i ← 0$ to $m$ do $g_{i0} ← i$
2. for $j ← 1$ to $n$ do $g_{0j} ← 0$
3. $top ← \min(k + 1, m)$
4. for $j ← 1$ to $n$
5.   for $i ← 1$ to $top$ do
6.     $g_{ij} ← \min\{g_{i-1,j-1} + \delta(A[i], B[j]), g_{i-1,j} + 1, g_{i,j-1} + 1\}$
7.   while $g_{top,j} > k$ do $top ← top - 1$
8.   if $top = m$ then output $j$
9.   else $top ← top + 1$
The time complexity is proportional to the computed area in the matrix \((g_{ij})\).

- The worst case time complexity is still \(O(mn)\).
- The average case time complexity is \(O(kn)\). The proof is not trivial.

There are many other algorithms based on diagonal monotonicity. Some of them achieve \(O(kn)\) worst case time complexity.
Myers’ Bitparallel Algorithm

Another way to speed up the computation is bitparallelism.

Instead of the matrix \((g_{ij})\), we store differences between adjacent cells:

**Vertical delta:** \(\Delta v_{ij} = g_{ij} - g_{i-1,j}\)

**Horizontal delta:** \(\Delta h_{ij} = g_{ij} - g_{i,j-1}\)

**Diagonal delta:** \(\Delta d_{ij} = g_{ij} - g_{i-1,j}\)

Because \(g_{i0} = i\) ja \(g_{0j} = 0\),

\[
g_{ij} = \Delta v_{1j} + \Delta v_{2j} + \cdots + \Delta v_{ij}
= i + \Delta h_{i1} + \Delta h_{i2} + \cdots + \Delta h_{ij}
\]

Because of diagonal monotonicity, \(\Delta d_{ij} \in \{0,1\}\) and it can be stored in one bit. By the following result, \(\Delta h_{ij}\) and \(\Delta v_{ij}\) can be stored in two bits.

**Lemma 2.15:** \(\Delta h_{ij}, \Delta v_{ij} \in \{-1,0,1\}\) for every \(i,j\) that they are defined for.

The proof is left as an exercise.
Example 2.16: ‘−’ means −1, ‘=’ means 0 and ‘+’ means +1

<table>
<thead>
<tr>
<th></th>
<th>r e m a c h i n e</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0</td>
</tr>
<tr>
<td>a</td>
<td>+ + + + + = = + + + + + + + + + + + + +</td>
</tr>
<tr>
<td>t</td>
<td>1 = 1 = 1 − 0 + 1 = 1 = 1 = 1 = 1 = 1 = 1</td>
</tr>
<tr>
<td>c</td>
<td>+ + + + + = = − = = = + + + + + + + + +</td>
</tr>
<tr>
<td>h</td>
<td>2 = 2 = 2 − 1 − 0 + 1 + 2 = 2 = 2 = 2 = 2</td>
</tr>
<tr>
<td>t</td>
<td>3 = 3 = 3 − 2 − 1 = 1 + 2 + 3 = 3 = 3</td>
</tr>
<tr>
<td>c</td>
<td>+ + + + + = = + = + = = + = + = + = + = +</td>
</tr>
<tr>
<td>h</td>
<td>4 = 4 = 4 − 3 − 2 − 1 + 2 + 3 + 4 = 4</td>
</tr>
<tr>
<td>m</td>
<td>+ + + + + = = + = + = + = − = − = − = − = − =</td>
</tr>
<tr>
<td>a</td>
<td>5 = 5 = 5 − 4 − 3 − 2 − 1 + 2 + 3 + 4 = 4</td>
</tr>
</tbody>
</table>
In the standard computation of a cell:

- Input is $g_{i-1,j}, g_{i-1,j-1}, g_{i,j-1}$ and $\delta(P[i], T[j])$.
- Output is $g_{ij}$.

In the corresponding bitparallel computation:

- Input is $\Delta v^{\text{in}} = \Delta v_{i,j-1}$, $\Delta h^{\text{in}} = \Delta h_{i,j-1}$ and $Eq_{ij} = 1 - \delta(P[i], T[j])$.
- Output is $\Delta v^{\text{out}} = \Delta v_{i,j}$ and $\Delta h^{\text{out}} = \Delta h_{i,j}$.
The computation rule is defined by the following result.

**Lemma 2.17:** If $Eq = 1$ or $\Delta v^\text{in} = -1$ or $\Delta h^\text{in} = -1$, then $\Delta d = 0$, $\Delta v^\text{out} = -\Delta h^\text{in}$ and $\Delta h^\text{out} = -\Delta v^\text{in}$.
Otherwise $\Delta d = 1$, $\Delta v^\text{out} = 1 - \Delta h^\text{in}$ and $\Delta h^\text{out} = 1 - \Delta v^\text{in}$.

**Proof.** We can write the recurrence for $g_{ij}$ as

$$g_{ij} = \min\{g_{i-1,j-1} + \delta(P[i], T[j]), g_{i,j-1} + 1, g_{i-1,j} + 1\}$$

$$= g_{i-1,j-1} + \min\{1 - Eq, \Delta v^\text{in} + 1, \Delta h^\text{in} + 1\}.$$

Then $\Delta d = g_{ij} - g_{i-1,j-1} = \min\{1 - Eq, \Delta v^\text{in} + 1, \Delta h^\text{in} + 1\}$
which is 0 if $Eq = 1$ or $\Delta v^\text{in} = -1$ or $\Delta h^\text{in} = -1$ and 1 otherwise.

Clearly $\Delta d = \Delta v^\text{in} + \Delta h^\text{out} = \Delta h^\text{in} + \Delta v^\text{out}$.
Thus $\Delta v^\text{out} = \Delta d - \Delta h^\text{in}$ and $\Delta h^\text{out} = \Delta d - \Delta v^\text{in}$. □
To enable bitparallel operation, we need two changes:

- The $\Delta v$ and $\Delta h$ values are “trits” not bits. We encode each of them with two bits as follows:

  $Pv = \begin{cases} 
  1 & \text{if } \Delta v = +1 \\
  0 & \text{otherwise}
  \end{cases}$

  $Mv = \begin{cases} 
  1 & \text{if } \Delta v = -1 \\
  0 & \text{otherwise}
  \end{cases}$

  $Ph = \begin{cases} 
  1 & \text{if } \Delta h = +1 \\
  0 & \text{otherwise}
  \end{cases}$

  $Mh = \begin{cases} 
  1 & \text{if } \Delta h = -1 \\
  0 & \text{otherwise}
  \end{cases}$

  Then

  $\Delta v = Pv - Mv$

  $\Delta h = Ph - Mh$

- We replace arithmetic operations ($+,-,\min$) with logical operations ($\land,\lor,\neg$).
Now the computation rules can be expressed as follows.

**Lemma 2.18:**

\[
\begin{align*}
P_v^{\text{out}} &= M_h^{\text{in}} \lor \neg (X_v \lor P_h^{\text{in}}) \\
M_v^{\text{out}} &= P_h^{\text{in}} \land X_v \\
P_h^{\text{out}} &= M_v^{\text{in}} \lor \neg (X_h \lor P_v^{\text{in}}) \\
M_h^{\text{out}} &= P_v^{\text{in}} \land X_h
\end{align*}
\]

where \(X_v = Eq \lor M_v^{\text{in}}\) and \(X_h = Eq \lor M_h^{\text{in}}\).

**Proof.** We show the claim for \(P_v\) and \(M_v\) only. \(P_h\) and \(M_h\) are symmetrical.

By Lemma 2.17,

\[
\begin{align*}
P_v^{\text{out}} &= (\neg \Delta d \land M_h^{\text{in}}) \lor (\Delta d \land \neg P_h^{\text{in}}) \\
M_v^{\text{out}} &= (\neg \Delta d \land P_h^{\text{in}}) \lor (\Delta d \land 0) = \neg \Delta d \land P_h^{\text{in}}
\end{align*}
\]

Because \(\Delta d = \neg (Eq \lor M_v^{\text{in}} \land M_h^{\text{in}}) = \neg (X_v \lor M_h^{\text{in}}) = \neg X_v \land \neg M_h^{\text{in}},\)

\[
\begin{align*}
P_v^{\text{out}} &= ((X_v \lor M_h^{\text{in}}) \land M_h^{\text{in}}) \lor (\neg X_v \land \neg M_h^{\text{in}} \land \neg P_h^{\text{in}}) \\
&= M_h^{\text{in}} \lor \neg (X_v \lor M_h^{\text{in}} \lor P_h^{\text{in}}) \\
&= M_h^{\text{in}} \lor \neg (X_v \lor P_h^{\text{in}}) \\
M_v^{\text{out}} &= (X_v \lor M_h^{\text{in}}) \land P_h^{\text{in}} = X_v \land P_h^{\text{in}}
\end{align*}
\]

In the last step, we used the fact that \(M_h^{\text{in}}\) and \(P_h^{\text{in}}\) cannot be 1 simultaneously. □
According to Lemma 2.18, the bit representation of the matrix can be computed as follows.

\[
\begin{align*}
&\text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
&P_{v_0} \leftarrow 1; \ M_{v_0} \leftarrow 0 \\
&\text{for } j \leftarrow 1 \text{ to } n \text{ do} \\
&P_{h_0j} \leftarrow 0; \ M_{h_0j} \leftarrow 0 \\
&\text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
&X_{h_{ij}} \leftarrow E_{q_{ij}} \lor M_{h_{i-1,j}} \\
&P_{h_{ij}} \leftarrow M_{v_{i,j-1}} \lor \neg(X_{h_{ij}} \lor P_{v_{i,j-1}}) \\
&M_{h_{ij}} \leftarrow P_{v_{i,j-1}} \land X_{h_{ij}} \\
&\text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
&X_{v_{ij}} \leftarrow E_{q_{ij}} \lor M_{v_{i,j-1}} \\
&P_{v_{ij}} \leftarrow M_{h_{i-1,j}} \lor \neg(X_{v_{ij}} \lor P_{h_{i-1,j}}) \\
&M_{v_{ij}} \leftarrow P_{h_{i-1,j}} \land X_{v_{ij}} \\
\end{align*}
\]

This is not yet bitparallel though.
To obtain a bitparallel algorithm, the columns $P_{v,j}$, $M_{v,j}$, $X_{v,j}$, $P_{h,j}$, $M_{h,j}$, $X_{h,j}$ and $E_{q,j}$ are stored in bitvectors.

Now the second inner loop can be replaced with the code

$$
X_{v,j} \leftarrow E_{q,j} \lor M_{v,j-1}
$$
$$
P_{v,j} \leftarrow (M_{h,j} \ll 1) \lor \neg(X_{v,j} \lor (P_{h,j} \ll 1))
$$
$$
M_{v,j} \leftarrow (P_{h,j} \ll 1) \land X_{v,j}
$$

A similar attempt with the for first inner loop leads to a problem:

$$
X_{h,j} \leftarrow E_{q,j} \lor (M_{h,j} \ll 1)
$$
$$
P_{h,j} \leftarrow M_{v,j-1} \lor \neg(X_{h,j} \lor P_{v,j-1})
$$
$$
M_{h,j} \leftarrow P_{v,j-1} \land X_{h,j}
$$

Now the vector $M_{h,j}$ is used in computing $X_{h,j}$ before $M_{h,j}$ itself is computed! Changing the order does not help, because $X_{h,j}$ is needed to compute $M_{h,j}$.

To get out of this dependency loop, we compute $X_{h,j}$ without $M_{h,j}$ using only $E_{q,j}$ and $P_{v,j-1}$ which are already available when we compute $X_{h,j}$.
Lemma 2.19: \( Xh_{ij} = \exists \ell \in [1, i] : Eq_{\ell j} \land (\forall x \in [\ell, i - 1] : Pv_{x,j-1}) \).

Proof. We use induction on \( i \).

Basis \( i = 1 \): The right-hand side reduces to \( Eq_{1j} \), because \( \ell = 1 \). By Lemma 2.18, \( Xh_{1j} = Eq_{1j} \lor Mh_{0j} \), which is \( Eq_{1j} \) because \( Mh_{0j} = 0 \) for all \( j \).

Induction step: The induction assumption is that \( Xh_{i-1,j} \) is as claimed. Now we have

\[
\exists \ell \in [1, i] : Eq_{\ell j} \land (\forall x \in [\ell, i - 1] : Pv_{x,j-1}) \\
= Eq_{ij} \lor \exists \ell \in [1, i - 1] : Eq_{\ell j} \land (\forall x \in [\ell, i - 1] : Pv_{x,j-1}) \\
= Eq_{ij} \lor (Pv_{i-1,j-1} \land \exists \ell \in [1, i - 1] : Eq_{\ell j} \land (\forall x \in [\ell, i - 2] : Pv_{x,j-1})) \\
= Eq_{ij} \lor (Pv_{i-1,j-1} \land Xh_{i-1,j}) \quad \text{(ind. assump.)} \\
= Eq_{ij} \lor Mh_{i-1,j} \quad \text{(Lemma 2.18)} \\
= Xh_{ij} \quad \text{(Lemma 2.18)}
\]

\( \square \)
At first sight, we cannot use Lemma 2.19 to compute even a single bit in constant time, not to mention a whole vector $Xh \ast j$. However, it can be done, but we need more bit operations:

- Let $\oplus$ denote the xor-operation: $0 \oplus 1 = 1$ and $0 \oplus 0 = 1 \oplus 1 = 0$.

- A bitvector is interpreted as an integer and we use addition as a bit operation. The carry mechanism in addition plays a key role. For example $0001 + 0111 = 1000$.

In the following, for a bitvector $B$, we will write

$$B = B[1..m] = B[m]B[m-1]...B[1]$$

The reverse order of the bits reflects the interpretation as an integer.
Lemma 2.20: Denote $X = Xh_{*j}$, $E = Eq_{*j}$, $P = Pv_{*,j-1}$ ja olkoon $Y = (((E \land P) + P) \lor P) \lor E$. Then $X = Y$.

Proof. By Lemma 2.19, $X[i] = 1$ iff and only if

a) $E[i] = 1$ or

b) $\exists \ell \in [1, i] : E[\ell \ldots i] = 00 \cdots 01 \land P[\ell \ldots i - 1] = 11 \cdots 1$.

and $X[i] = 0$ iff and only if

c) $E_{1\ldots i} = 00 \cdots 0$ or

d) $\exists \ell \in [1, i] : E[\ell \ldots i] = 00 \cdots 01 \land P[\ell \ldots i - 1] \neq 11 \cdots 1$.

We prove that $Y[i] = X[i]$ in all of these cases:

a) The definition of $Y$ ends with "$\lor E$" which ensures that $Y[i] = 1$ in this case.
b) The following calculation shows that \( Y[i] = 1 \) in this case:

\[
\begin{array}{c}
\text{ } \\
i & \ell \\
E[\ell \ldots i] &= 00 \ldots 01 \\
P[\ell \ldots i] &= b1 \ldots 11 \\
(E \land P)[\ell \ldots i] &= 00 \ldots 01 \\
((E \land P) + P)[\ell \ldots i] &= \overline{b}0 \ldots 0c \\
(((E \land P) + P) \lor P)[\ell \ldots i] &= 11 \ldots 1\overline{c} \\
Y = (((((E \land P) + P) \lor P) \lor E)[\ell \ldots i] &= 11 \ldots 11
\end{array}
\]

where \( b \) is the unknown bit \( P[i] \), \( c \) is the possible carry bit coming from the summation of bits 1 \( \ldots, \ell - 1 \), and \( \overline{b} \) and \( \overline{c} \) are their negations.

c) Because for all bitvectors \( B \), \( 0 \land B = 0 \) ja \( 0 + B = B \), we get

\[
Y = (((0 \land P) + P) \lor P) \lor 0 = (P \lor P) \lor 0 = 0.
\]

d) Consider the calculation in case b). A key point there is that the carry bit in the summation travels from position \( \ell \) to \( i \) and produces \( \overline{b} \) to position \( i \). The difference in this case is that at least one bit \( P[k] \), \( \ell \leq k < i \), is zero, which stops the carry at position \( k \). Thus

\[
((E \land P) + P)[i] = b \quad \text{and} \quad Y[i] = 0.
\]

\( \square \)
As a final detail, we compute the bottom row values \( g_{mj} \) using the equalities
\[
g_{m0} = m \quad \text{ja} \quad g_{mj} = g_{m,j-1} + \Delta h_{mj}.
\]

**Algorithm 2.21:** Myers’ bitparallel algorithm

Input: text \( T[1..n] \), pattern \( P[1..m] \), and integer \( k \)

Output: end positions of all approximate occurrences of \( P \)

1. for \( c \in \Sigma \) do \( B[c] \leftarrow 0^m \)
2. for \( i \leftarrow 1 \) to \( m \) do \( B[P[i]][i] = 1 \)
3. \( P_v \leftarrow 1^m; \ M_v \leftarrow 0; \ g \leftarrow m \)
4. for \( j \leftarrow 1 \) to \( n \) do
5. \( \text{Eq} \leftarrow B[T[j]] \)
6. \( X_h \leftarrow (((\text{Eq} \land P_v) + P_v) \lor P_v) \lor \text{Eq}; \)
7. \( P_h \leftarrow M_v \lor \neg(X_h \lor P_v) \)
8. \( M_h \leftarrow P_v \land X_h; \)
9. \( X_v \leftarrow \text{Eq} \lor M_v \)
10. \( P_v \leftarrow (M_h << 1) \lor \neg(X_v \lor (P_h << 1)) \)
11. \( M_v \leftarrow (P_h << 1) \land X_v \)
12. \( g \leftarrow g + P_h[m] - M_h[m] \)
13. if \( g \leq k \) then output \( j \)