We can reduce computation using diagonal monotonicity:

- Whenever the value on a diagonal \(d\) 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 4.13:** \(P = \text{match}, T = \text{remachine}, k = 1\)

\[
\begin{array}{cccccccccccc}
g & r & e & m & a & c & h & i & n & e \\
\hline
\text{g} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{m} & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\text{m} & 1 & 1 & 1 & 0 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\
\text{m} & 1 & 1 & 2 & 3 \\
\text{m} & 1 & 2 & 3 \\
\text{m} & 1 & 2 \\
\end{array}
\]
The position of the smallest undiscarded diagonal on the current column is kept in a variable $top$.

**Algorithm 4.14:** 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 $i \leftarrow 0$ to $m$ do $g_{i0} \leftarrow i$
2. for $j \leftarrow 1$ to $n$ do $g_{0j} \leftarrow 0$
3. $top \leftarrow \min(k + 1, m)$
4. for $j \leftarrow 1$ to $n$ do
5. for $i \leftarrow 1$ to $top$ do
6. $g_{ij} \leftarrow \min\{g_{i-1,j-1} + \delta(A[i], B[j]), g_{i-1,j} + 1, g_{i,j-1} + 1\}$
7. while $g_{top,j} > k$ do $top \leftarrow top - 1$
8. if $top = m$ then output $j$
9. else $top \leftarrow top + 1$
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 4.15:** \(\Delta h_{ij}, \Delta v_{ij} \in \{-1, 0, 1\}\) for every \(i, j\) that they are defined for.

The proof is left as an exercise.
Example 4.16: ‘−’ 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>+ + + + + = + + + + + + + + + +</td>
</tr>
<tr>
<td></td>
<td>1 = 1 = 1 - 0 + 1 = 1 = 1 = 1 = 1</td>
</tr>
<tr>
<td>a</td>
<td>+ + + + + = + = = = = + + + + + + + +</td>
</tr>
<tr>
<td></td>
<td>2 = 2 = 2 - 1 - 0 + 1 + 2 = 2 = 2 = 2</td>
</tr>
<tr>
<td>t</td>
<td>+ + + + + = + = + = + = + = + + + + +</td>
</tr>
<tr>
<td></td>
<td>3 = 3 = 3 - 2 - 1 = 1 + 2 + 3 = 3 = 3</td>
</tr>
<tr>
<td>c</td>
<td>+ + + + + = + = + = = + = + = + = + + +</td>
</tr>
<tr>
<td></td>
<td>4 = 4 = 4 - 3 - 2 - 1 + 2 + 3 + 4 = 4</td>
</tr>
<tr>
<td>h</td>
<td>+ + + + + = + = + = + = = = - = - = - =</td>
</tr>
<tr>
<td></td>
<td>5 = 5 = 5 - 4 - 3 - 2 - 1 + 2 + 3 + 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 \( \Delta v^{in} = \Delta v_{i,j-1}, \Delta h^{in} = \Delta h_{i,j-1} \) and \( E_{ij} = 1 - \delta(P[i], T[j]) \).
- **Output** is \( \Delta v^{out} = \Delta v_{i,j} \) and \( \Delta h^{out} = \Delta h_{i,j} \).
The computation rule is defined by the following result.

**Lemma 4.17:** If $Eq = 1$ or $\Delta v_{in} = -1$ or $\Delta h_{in} = -1$, then $\Delta d = 0$, $\Delta v_{out} = -\Delta h_{in}$ and $\Delta h_{out} = -\Delta v_{in}$. Otherwise $\Delta d = 1$, $\Delta v_{out} = 1 - \Delta h_{in}$ and $\Delta h_{out} = 1 - \Delta v_{in}$.

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

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

$$= g_{i-1,j-1} + \min\{1 - Eq, \Delta v_{in} + 1, \Delta h_{in} + 1\}.$$

Then $\Delta d = g_{ij} - g_{i-1,j-1} = \min\{1 - Eq, \Delta v_{in} + 1, \Delta h_{in} + 1\}$

which is 0 if $Eq = 1$ or $\Delta v_{in} = -1$ or $\Delta h_{in} = -1$ and 1 otherwise.

Clearly $\Delta d = \Delta v_{in} + \Delta h_{out} = \Delta h_{in} + \Delta v_{out}$.

Thus $\Delta v_{out} = \Delta d - \Delta h_{in}$ and $\Delta h_{out} = \Delta d - \Delta v_{in}$.

□
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:

  \[
  P_v = \begin{cases} 
  1 & \text{if } \Delta v = +1 \\
  0 & \text{otherwise}
  \end{cases} \quad M_v = \begin{cases} 
  1 & \text{if } \Delta v = -1 \\
  0 & \text{otherwise}
  \end{cases} \\
  P_h = \begin{cases} 
  1 & \text{if } \Delta h = +1 \\
  0 & \text{otherwise}
  \end{cases} \quad M_h = \begin{cases} 
  1 & \text{if } \Delta h = -1 \\
  0 & \text{otherwise}
  \end{cases}
  \]

  Then

  \[
  \Delta v = P_v - M_v \\
  \Delta h = P_h - M_h
  \]

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

**Lemma 4.18:**

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

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 4.17,

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

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

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

In the last step, we used the fact that \( M_h^{\text{in}} \) and \( P_h^{\text{in}} \) cannot be 1 simultaneously. \( \square \)
According to Lemma 4.18, the bit representation of the matrix can be computed as follows.

\[
\begin{align*}
\text{for } i &\leftarrow 1 \text{ to } m \text{ do} \\
Pv_{i0} &\leftarrow 1; \hspace{1em} Mv_{i0} \leftarrow 0 \\
\text{for } j &\leftarrow 1 \text{ to } n \text{ do} \\
Ph_{0j} &\leftarrow 0; \hspace{1em} Mh_{0j} \leftarrow 0 \\
\text{for } i &\leftarrow 1 \text{ to } m \text{ do} \\
Xh_{ij} &\leftarrow Eq_{ij} \lor Mh_{i-1,j} \\
Ph_{ij} &\leftarrow Mv_{i,j-1} \lor \neg (Xh_{ij} \lor Pv_{i,j-1}) \\
Mh_{ij} &\leftarrow Pv_{i,j-1} \land Xh_{ij} \\
\text{for } i &\leftarrow 1 \text{ to } m \text{ do} \\
Xv_{ij} &\leftarrow Eq_{ij} \lor Mv_{i,j-1} \\
Pv_{ij} &\leftarrow Mh_{i-1,j} \lor \neg (Xv_{ij} \lor Ph_{i-1,j}) \\
Mv_{ij} &\leftarrow Ph_{i-1,j} \land Xv_{ij}
\end{align*}
\]

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

\[
\begin{align*}
X_{v,j} & \leftarrow E_{q,j} \lor M_{v,j-1} \\
P_{v,j} & \leftarrow (M_{h,j} << 1) \lor \neg(X_{v,j} \lor (P_{h,j} << 1)) \\
M_{v,j} & \leftarrow (P_{h,j} << 1) \land X_{v,j}
\end{align*}
\]

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

\[
\begin{align*}
X_{h,j} & \leftarrow E_{q,j} \lor (M_{h,j} << 1) \\
P_{h,j} & \leftarrow M_{v,j-1} \lor \neg(X_{h,j} \lor P_{v,j-1}) \\
M_{h,j} & \leftarrow P_{v,j-1} \land X_{h,j}
\end{align*}
\]

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 $X_{h,j}$. 

119
Lemma 4.19: \[ X_{h_{ij}} = \exists \ell \in [1, i] : Eq_{\ell j} \land (\forall x \in [\ell, i - 1] : Pv_{x,j-1}) \]

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

Basis \( i = 1 \): The right-hand side reduces to \( Eq_{1j} \), because \( \ell = 1 \). By Lemma 4.18, \( X_{h_{1j}} = Eq_{1j} \lor Mh_{0j} \), which is \( Eq_{1j} \) because \( Mh_{0j} = 0 \) for all \( j \).

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

\[
\exists \ell \in [1, i] : Eq_{\ell j} \land (\forall x \in [\ell, i - 1] : Pv_{x,j-1})
\]

\[= Eq_{ij} \lor \exists \ell \in [1, i - 1] : Eq_{\ell j} \land (\forall x \in [\ell, i - 1] : Pv_{x,j-1})\]

\[= Eq_{ij} \lor (Pv_{i-1,j-1} \land \exists \ell \in [1, i - 1] : Eq_{\ell j} \land (\forall x \in [\ell, i - 2] : Pv_{x,j-1}))\]

\[= Eq_{ij} \lor (Pv_{i-1,j-1} \land X_{h_{i-1,j}}) \quad \text{(ind. assump.)}\]

\[= Eq_{ij} \lor Mh_{i-1,j} \quad \text{(Lemma 4.18)}\]

\[= X_{h_{ij}} \quad \text{(Lemma 4.18)}\]

\[\square\]
At first sight, we cannot use Lemma 4.19 to compute even a single bit in constant time, not to mention a whole vector $Xh_{*j}$. 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

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

The reverse order of the bits reflects the interpretation as an integer.
**Lemma 4.20:** 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 4.19, $X[i] = 1$ iff and only if

a) $E[i] = 1$ or

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

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

c) $E_{1...i} = 00...0$ or

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

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:

\[
\begin{array}{cccc}
& i & \ell \\
E[\ell...i] &=& 00...01 \\
P[\ell...i] &=& b1...11 \\
(E \land P)[\ell...i] &=& 00...01 \\
((E \land P) + P)[\ell...i] &=& b0...0c \\
(((E \land P) + P) \lor P)[\ell...i] &=& 11...1\bar{c} \\
Y &=& (((((E \land P) + P) \lor P) \lor E)[\ell...i] = 11...11
\end{array}
\]

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 $\bar{b}$ and $\bar{c}$ 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 $\bar{b}$ 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

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

□
As a final detail, we compute the bottom row values $g_{mj}$ using the equalities $g_{m0} = m$ ja $g_{mj} = g_{m,j-1} + \Delta h_{mj}$.

**Algorithm 4.21:** 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) $Pv \leftarrow 1^m$; $Mv \leftarrow 0$; $g \leftarrow m$  
(4) for $j \leftarrow 1$ to $n$ do  

(5) $Eq \leftarrow B[T[j]]$  
(6) $Xh \leftarrow (((Eq \land Pv) + Pv) \lor Pv) \lor Eq$  
(7) $Ph \leftarrow Mv \lor \neg(Xh \lor Pv)$  
(8) $Mh \leftarrow Pv \land Xh$  
(9) $Xv \leftarrow Eq \lor Mv$  
(10) $Pv \leftarrow (Mh << 1) \lor \neg(Xv \lor (Ph << 1))$  
(11) $Mv \leftarrow (Ph << 1) \land Xv$  
(12) $g \leftarrow g + Ph[m] - Mh[m]$  
(13) if $g \leq k$ then output $j$