Inclusion-Exclusion and Dynamic Programming

Siyang Chen
Simple example:

Suppose we are given a list $a_0, a_1, \ldots, a_n$ and an integer $k$, and we want to compute for all $i$:

$$ b_i = \sum_{j=i-k+1}^{i} a_j $$

Naïve solution:

```c
for( int i = 0; i < n; i++ )
  for( int j = max( i - k - 1, 0 ); j <= i; j++ )
    b[i] += a[j];
```

Running time?

$O(nk)$
Simple example:

Suppose we are given a list $a_0, a_1, \ldots, a_n$ and an integer $k$, and we want to compute for all $i$:

$$b_i = \sum_{j=i-k+1}^{i} a_j$$

Naïve solution:

```cpp
for( int i = 0; i < n; i++ )
    for( int j = max( i - k - 1, 0 ); j <= i; j++ )
        b[i] += a[j];
```
Simple example:

Suppose we are given a list $a_0, a_1, \ldots, a_n$ and an integer $k$, and we want to compute for all $i$:

$$b_i = \sum_{j=i-k+1}^{i} a_j$$

Naïve solution:

```c
for( int i = 0; i < n; i++ )
    for( int j = max( i - k - 1, 0 ); j <= i; j++ )
        b[i] += a[j];
```

Running time?
Simple example:

Suppose we are given a list \( a_0, a_1, \ldots, a_n \) and an integer \( k \), and we want to compute for all \( i \):

\[
    b_i = \sum_{j=i-k+1}^{i} a_j
\]

Naïve solution:

```cpp
for( int i = 0; i < n; i++ )
    for( int j = max( i - k - 1, 0 ); j <= i; j++ )
        b[i] += a[j];
```

Running time? \( O(nk) \)
Simple example:

What if $k$ is large? Is there a better way?
Simple example:

What if $k$ is large? Is there a better way?

Alternative solution:

```c
for( int i = 0; i < n; i++ )
{
    b[i] = a[j];
    if( i > 0 ) b[i] += b[i-1];
    if( i >= k ) b[i] -= a[i-k];
}
```
Simple example:

What if $k$ is large? Is there a better way?

Alternative solution:

```c
for( int i = 0; i < n; i++ )
{
    b[i] = a[j];
    if( i > 0 ) b[i] += b[i-1];
    if( i >= k ) b[i] -= a[i-k];
}
```

Running time?
Simple example:

What if $k$ is large? Is there a better way?

Alternative solution:

```c
for( int i = 0; i < n; i++ )
{
    b[i] = a[j];
    if( i > 0 ) b[i] += b[i-1];
    if( i >= k ) b[i] -= a[i-k];
}
```

Running time? $O(n)$
Slightly harder example:

Now suppose we are given an $n \times m$ matrix

$$A = \begin{pmatrix}
a_{00} & a_{01} & \cdots & a_{0m} \\
a_{10} & a_{11} & \cdots & a_{1m} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n0} & a_{n1} & \cdots & a_{nm}
\end{pmatrix}$$

For some integers $x$ and $y$, define $B = \{b_{ij}\}$ as

$$b_{i,j} = \sum_{i'=i-x+1}^{i} \sum_{j'=j-y+1}^{j} a_{ij}$$
Slightly harder example:

Now suppose we are given an $n \times m$ matrix

$$A = \begin{pmatrix} a_{00} & a_{01} & \ldots & a_{0m} \\ a_{10} & a_{11} & \ldots & a_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n0} & a_{n1} & \ldots & a_{nm} \end{pmatrix}$$

For some integers $x$ and $y$, define $B = \{b_{ij}\}$ as

$$b_{i,j} = \sum_{i'=i-x+1}^{i} \sum_{j'=j-y+1}^{j} a_{ij}$$

How to compute $B$ efficiently?
Slightly harder example:

One solution is to use a similar method as above:

```
for( int i = 0; i < n; i++ )
for( int j = 0; j < m; j++ )
{
    c[i][j] = a[i][j];
    if( j > 0 ) c[i][j] += b[i][j-1];
    if( j >= y ) c[i][j] -= a[i][j-y];
}
```

```
for( int i = 0; i < n; i++ )
for( int j = 0; j < m; j++ )
{
    b[i][j] = c[i][j];
    if( i > 0 ) b[i][j] += b[i-1][j];
    if( i >= x ) b[i][j] -= c[i-x][j];
}
```

Question: What does $c[i][j]$ represent?
Slightly harder example:

One solution is to use a similar method as above:

```c
for( int i = 0; i < n; i++ )
for( int j = 0; j < m; j++ )
{
    c[i][j] = a[i][j];
    if( j > 0 ) c[i][j] += b[i][j-1];
    if( j >= y ) c[i][j] -= a[i][j-y];
}
```

```c
for( int i = 0; i < n; i++ )
for( int j = 0; j < m; j++ )
{
    b[i][j] = c[i][j];
    if( i > 0 ) b[i][j] += b[i-1][j];
    if( i >= x ) b[i][j] -= c[i-x][j];
}
```
One solution is to use a similar method as above:

```c
for( int i = 0; i < n; i++ )
  for( int j = 0; j < m; j++ )
  {
    c[i][j] = a[i][j];
    if( j > 0 ) c[i][j] += b[i][j-1];
    if( j >= y ) c[i][j] -= a[i][j-y];
  }

for( int i = 0; i < n; i++ )
  for( int j = 0; j < m; j++ )
  {
    b[i][j] = c[i][j];
    if( i > 0 ) b[i][j] += b[i-1][j];
    if( i >= x ) b[i][j] -= c[i-x][j];
  }
```

Question: What does $c[i][j]$ represent?
Slightly harder example, redux:

An alternative solution, popular in programming contests:
Slightly harder example, redux:

An alternative solution, popular in programming contests:

```c
for( int i = 0; i < n; i++ )
for( int j = 0; j < m; j++ )
{
    sum[i][j] = a[i][j];
    if( i > 0 ) sum[i][j] += sum[i-1][j];
    if( j > 0 ) sum[i][j] += sum[i][j-1];
    if( i > 0 && j > 0 ) sum[i][j] -= sum[i-1][j-1];
}

for( int i = 0; i < n; i++ )
for( int j = 0; j < m; j++ )
{
    b[i][j] = sum[i][j];
    if( i >= x ) b[i][j] -= sum[i-x][j];
    if( j >= y ) b[i][j] -= sum[i][j-y];
    if( i >= x && j >= y ) b[i][j] += sum[i-x][j-y];
}
```
Slightly harder example, redux:

Questions:
Slightly harder example, redux:

Questions:

1. What does $\text{sum}[i][j]$ represent?
Slightly harder example, redux:

Questions:

1. What does $\text{sum}[i][j]$ represent?

2. How is $\text{sum}[i][j]$ used to compute $b[i][j]$? Why is this correct?
Principle of Inclusion-Exclusion:

\[ |A \cup B| = |A| + |B| - |A \cap B| \]
Principle of Inclusion-Exclusion:

\[ |A \cup B| = |A| + |B| - |A \cap B| \]

- Very useful for computing expressions involving summations involving subsets in a dynamic program.
Principle of Inclusion-Exclusion:

\[ |A \cup B| = |A| + |B| - |A \cap B| \]

- Very useful for computing expressions involving summations involving subsets in a dynamic program.
- Caveat: Naïvely applying PIE to high-dimensional sums can lead to exponentially many terms. I.e., for 3 dimensions:

\[
\begin{align*}
\text{sum}[i][j][k] &= a[i][j][k] \\
&- \text{sum}[i-1][j][k] - \text{sum}[i][j-1][k] - \text{sum}[i][j][k-1] \\
&+ \text{sum}[i-1][j-1][k] + \text{sum}[i][j-1][k-1] + \text{sum}[i-1][j][k-1] \\
&- \text{sum}[i-1][j-1][k-1]
\end{align*}
\]
Principle of Inclusion-Exclusion:

\[ |A \cup B| = |A| + |B| - |A \cap B| \]

- Very useful for computing expressions involving summations involving subsets in a dynamic program.
- Caveat: Naïvely applying PIE to high-dimensional sums can lead to exponentially many terms. I.e., for 3 dimensions:

\[
\text{sum}[i][j][k] = a[i][j][k] \\
- \text{sum}[i-1][j][k] - \text{sum}[i][j-1][k] - \text{sum}[i][j][k-1] \\
+ \text{sum}[i-1][j-1][k] + \text{sum}[i][j-1][k-1] + \text{sum}[i-1][j][k-1] \\
- \text{sum}[i-1][j-1][k-1]
\]

- In these cases it is better to use the previous approach.