Massively Parallel Algorithms for String Matching with Wildcards

MohammadTaghi Hajiaghayi[1], Hamed Saleh[1], Saeed Seddighin[1],
Xiaorui Sun[2] [CoRR’19]

Sublinear Space

Have you ever had a dataset so big that it doesn't fit in the memory?

⬇

Sublinear Space Algorithms!

$$o(n)$$

Sublinear Space

Have you ever had a dataset so big that it doesn't fit in the memory?

Sublinear Space Algorithms!

RAM model

$$o(n)$$

Sublinear Space

Have you ever had a dataset so big that it doesn't fit in the memory?

⬇

Sublinear Space Algorithms!

Alternative models

$$o(n)$$

Alternative Models

of Computation

Massively Parallel Computation

- Modern frameworks for Large-scale parallel/distributed data processing: MapReduce, Hadoop, Spark.

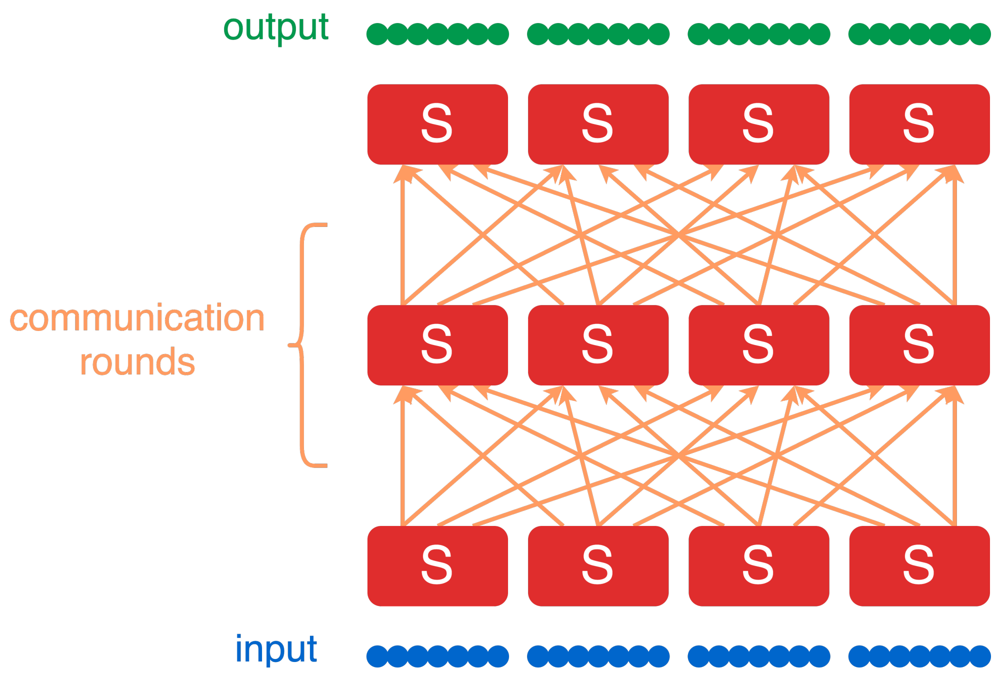
- Key Idea: distribute the workload among several machines.

- The MPC model: A theoretical model to abstract out the computational power of these frameworks.

[Karloff et. al 2010]
[Goodrich et. al 2011]
[Beame et. al 2013]
[Andoni et. al 2014]

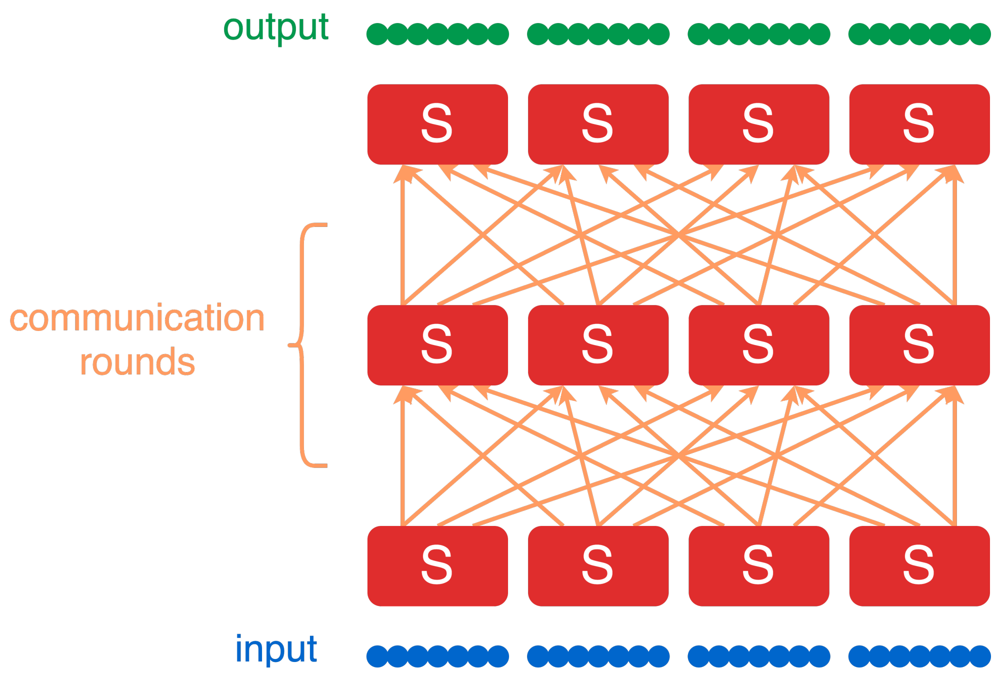
Massively Parallel Computation

- There are M machines each with memory S.
- The input of length N is initially (randomly) distributed among the machines.
  - Sublinear space $S = o(N)$.
  - Usually $N = O(S \cdot M)$.
- The data is processed in several **synchronous rounds**.



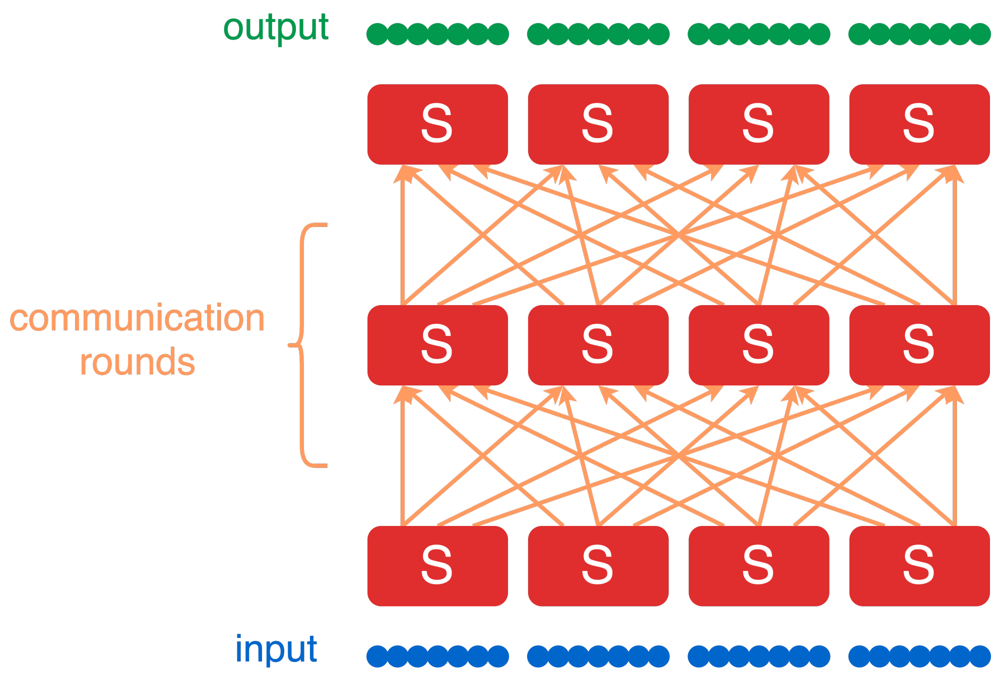
Massively Parallel Computation

- There are M machines each with memory S.
- The data is processed in several **synchronous rounds**. In each round,
  - Machines perform arbitrary computation on their **local** data.
  - Machines **communicate** with each other. Total **incoming/outgoing** messages of each machine is bounded by $O(S)$ words.



Massively Parallel Computation

- There are M machines each with memory S.
- The data is processed in several **synchronous rounds**.
- Main bottleneck: **Communication**. We wish for algorithms with very small number of rounds.
  - Often **sub-logarithmic** rounds.



Related distributed/parallel models

- The PRAM model (shared memory)
  - Any PRAM algorithm running in time $t = t(n)$ can be simulated in $O(t)$ MPC rounds.

[Karloff et. al 2010]

- The LOCAL, CONGEST, and congested-clique models
  - Similar techniques can be used for both MPC and these distributed models.
  - Congested-clique is almost equivalent to MPC (in terms of the number of rounds).

[Behnezhad et. al 2018]

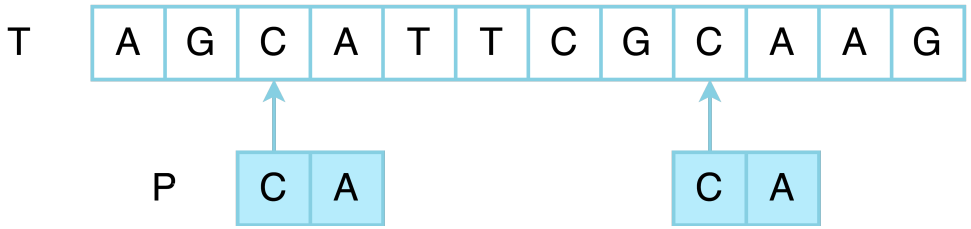
String Matching

Massively Parallel Algorithms for String Matching with Wildcards **[arXiv]**

Hajiaghayi, me, Seddighin, Sun

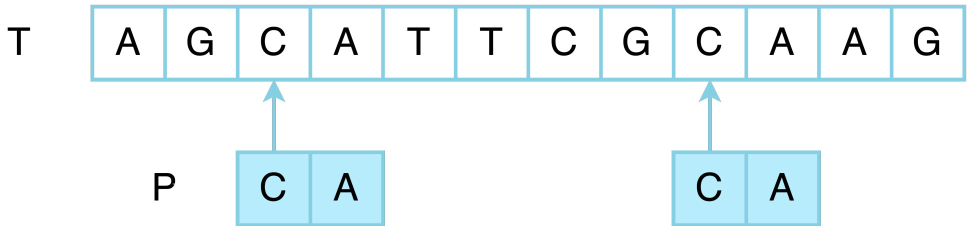
The **String Matching** problem

- An essential problem in bio-informatics and many other areas.
- Given a **text** T and a **pattern** P, we wish to find all substrings of T that match P.



The **String Matching** problem

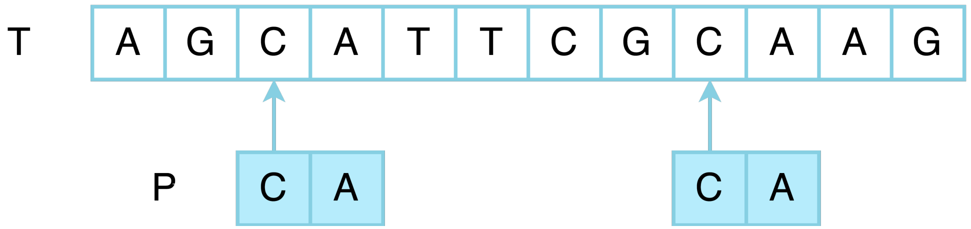
- Given a **text** T and a **pattern** P, we wish to find all substrings of T that match P.
- In the simplest form both T and P are using the same alphabet Σ, and there is no special character.



The **String Matching** problem

- Given a **text** T and a **pattern** P, we wish to find all substrings of T that match P.

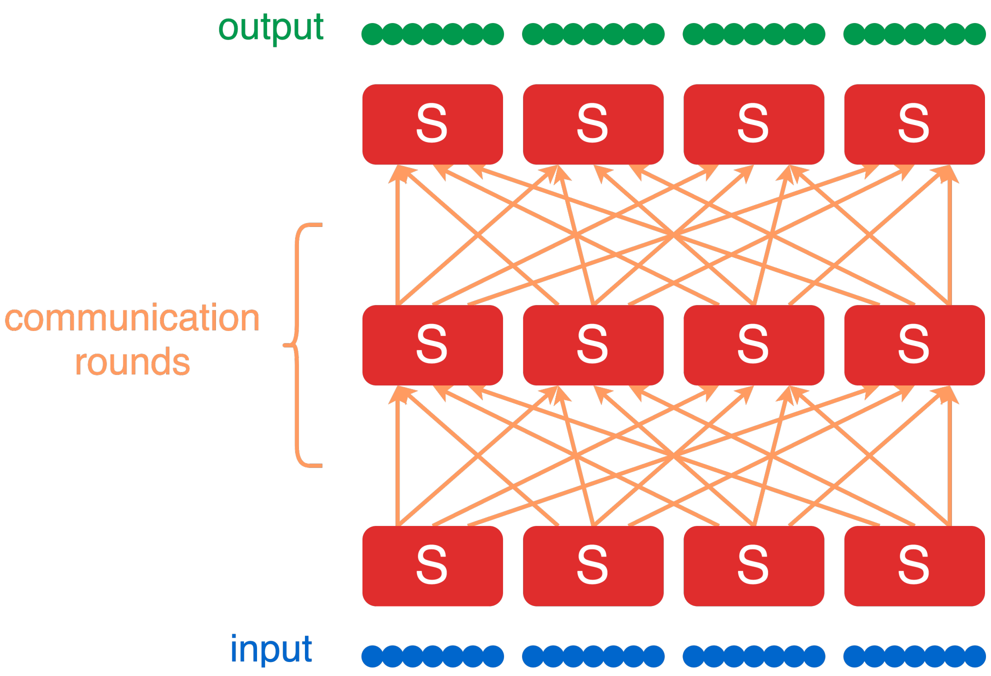
- We also study the case when P can have special characters known as **wildcards**. In particular {‘?’, ‘+’, ‘*’}.



String Matching in MPC

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq \Sigma^m$ are given.
- There is a **constant**-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of P in T.

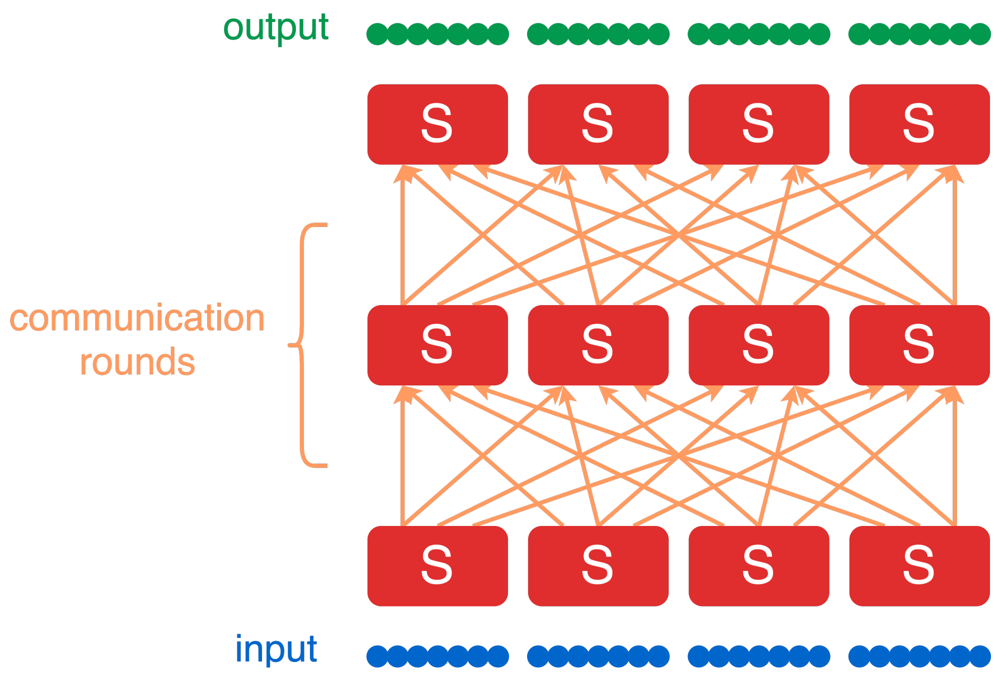
[HSSS 2019]



String Matching in MPC

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq \Sigma^m$ are given.
- There is a **constant**-round MPC algorithm, with $\boldsymbol{S} = O(n^{1-x})$ and $\boldsymbol{M} = O(n^x)$ for any $x < 0.5$, which finds all occurrences of P in T.

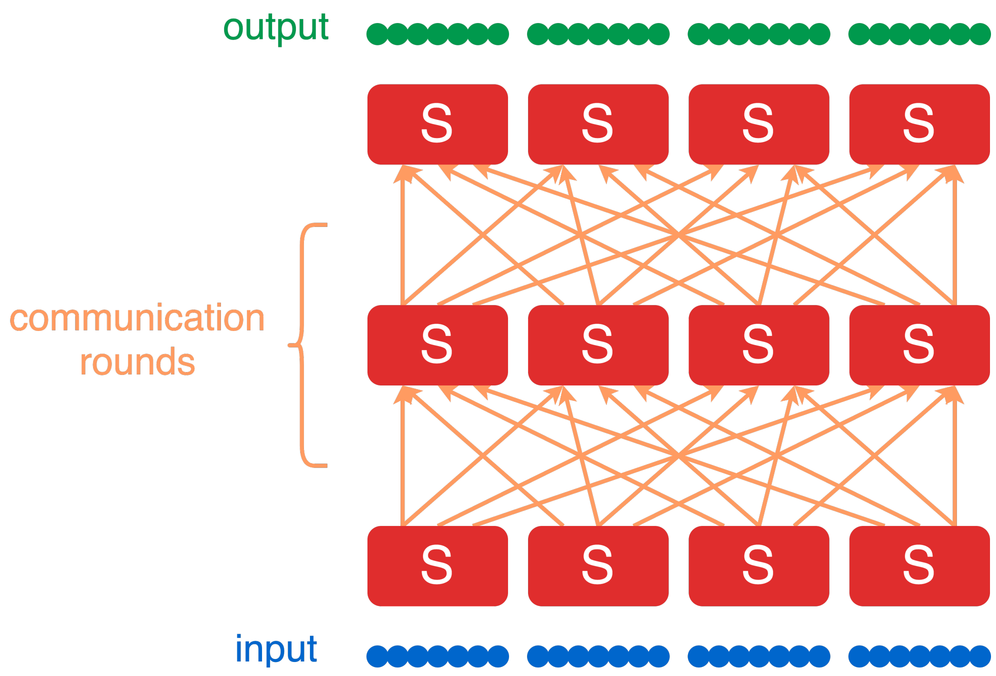
  [HSSS 2019]
- Easy when $\mathrm{m} = O(n^{1-x})$: **Double**-covering.



String Matching in MPC

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq \Sigma^m$ are given.
- There is a **constant**-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of P in T.

  [HSSS 2019]
- Also easy for general m: Partial **hash**ing.

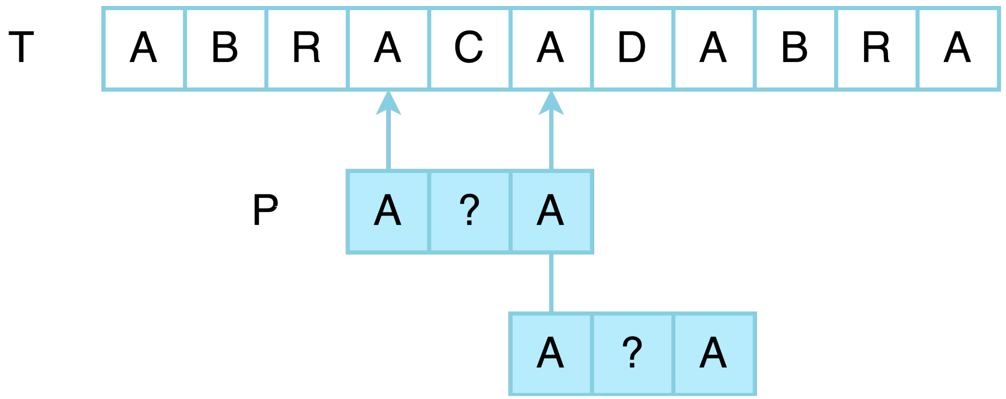


String Matching

with ‘?’ wildcard

String Matching with '?' wildcard

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'?'})^m$ are given.
- The special character '?' can be replaced with any arbitrary character.

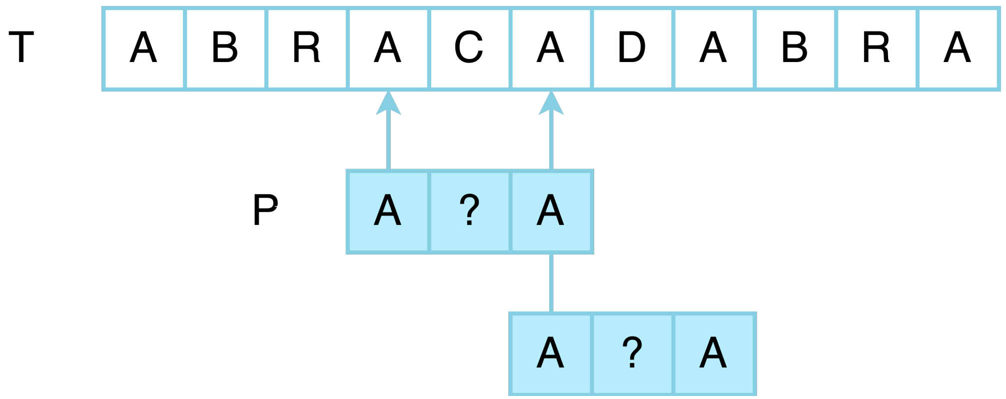


String Matching with '?' wildcard

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'?'})^m$ are given.

- There is a **constant**-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of P in T.

[HSSS 2019]



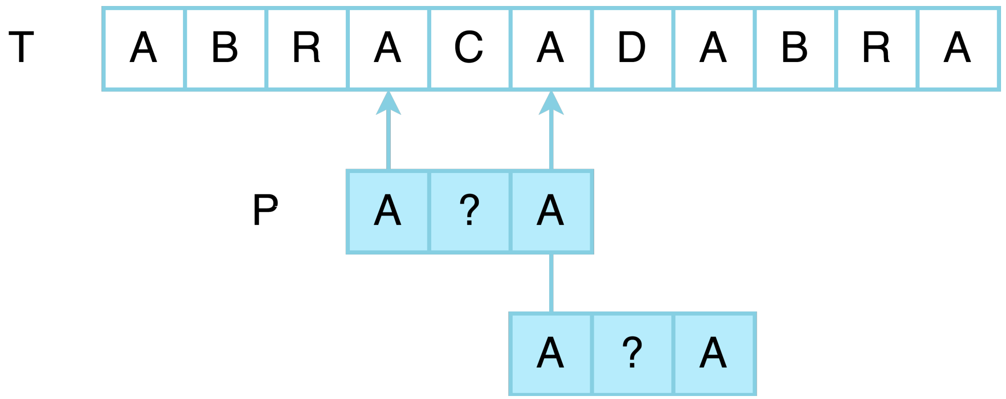
String Matching with '?' wildcard

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'?'})^m$ are given.

- There is a **constant**-round MPC algorithm, with $\boldsymbol{S} = O(n^{1-x})$ and $\boldsymbol{M} = O(n^x)$ for any $x < 0.5$, which finds all occurrences of P in T.

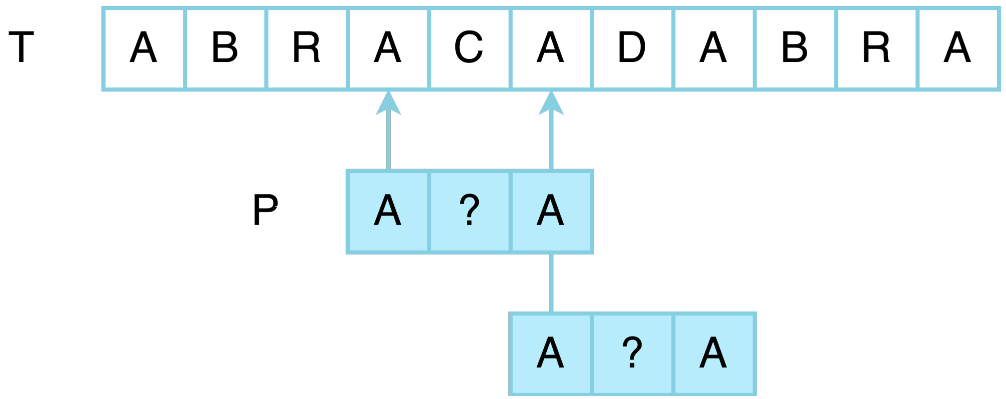
  [HSSS 2019]

- This can be improved to any constant $x < 1$ at the cost of $O\left(\frac{1}{1-x}\right)$ rounds.



'?' wildcard and **convolution**

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'?'})^m$ are given.
- This variant of the string matching problem can be reduced to the convolution of two arrays.
[Fischer et. al 1974]



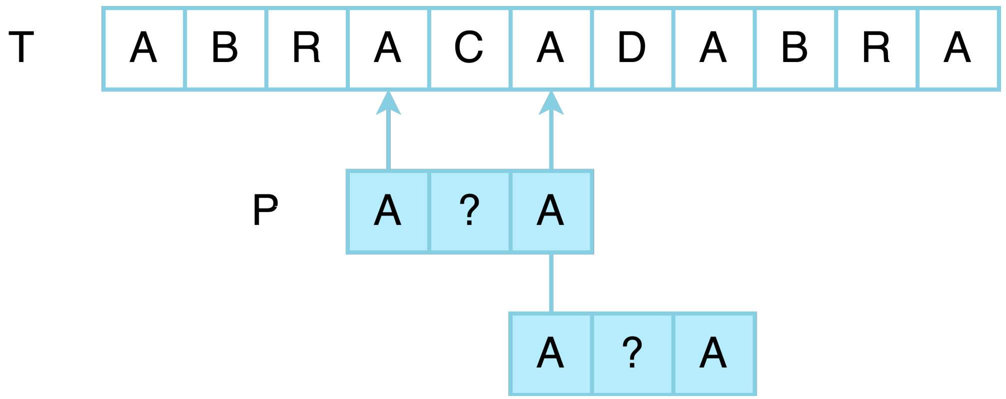
$$T^\dagger = \langle \mathrm{mp}_{T_1}, \mathrm{mp}_{T_1}^{-1}, \mathrm{mp}_{T_2}, \mathrm{mp}_{T_2}^{-1}, \ldots, \mathrm{mp}_{T_n}, \mathrm{mp}_{T_n}^{-1} \rangle$$
$$P^\dagger = \langle \mathrm{mp}_{P_1}, \mathrm{mp}_{P_1}^{-1}, \mathrm{mp}_{P_2}, \mathrm{mp}_{P_2}^{-1}, \ldots, \mathrm{mp}_{P_n}, \mathrm{mp}_{P_n}^{-1} \rangle$$

$$\mathrm{mp}_{\text{'?'}} = \mathrm{mp}_{\text{'?'}}^{-1} = 0, \mathrm{mp}_{\text{'C'}} = 3, \mathrm{mp}_{\text{'C'}}^{-1} = 1/3$$

'?' wildcard and **convolution**

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'?'})^m$ are given.
- This variant of the string matching problem can be reduced to the convolution of two arrays.

[Fischer et. al 1974]



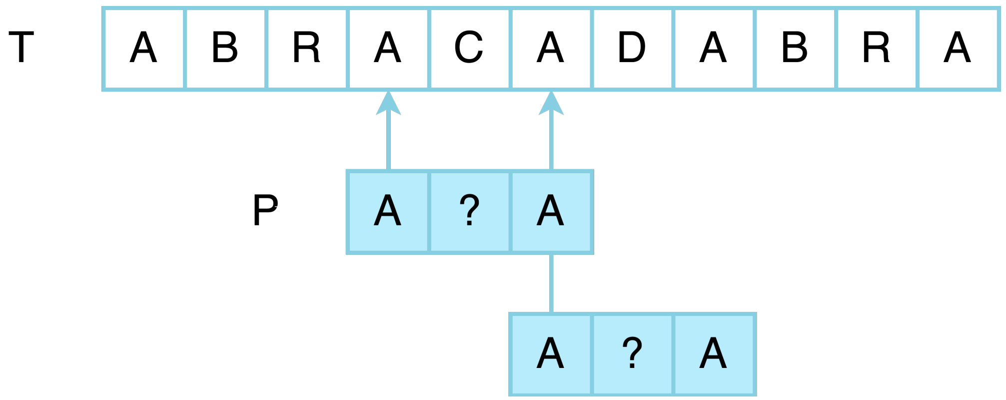
$$T^\dagger = \langle 1, \frac{1}{1}, 2, \frac{1}{2}, 18, \frac{1}{18}, 1, \frac{1}{1}, 3, \frac{1}{3}, 1, \frac{1}{1}, 4, \frac{1}{4}, 1, \frac{1}{1}, 2, \frac{1}{2}, 18, \frac{1}{18}, 1, \frac{1}{1} \rangle$$
$$P^\dagger = \langle 1, \frac{1}{1}, 0, 0, 1, \frac{1}{1} \rangle$$

'?' wildcard and convolution

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'?'})^m$ are given.

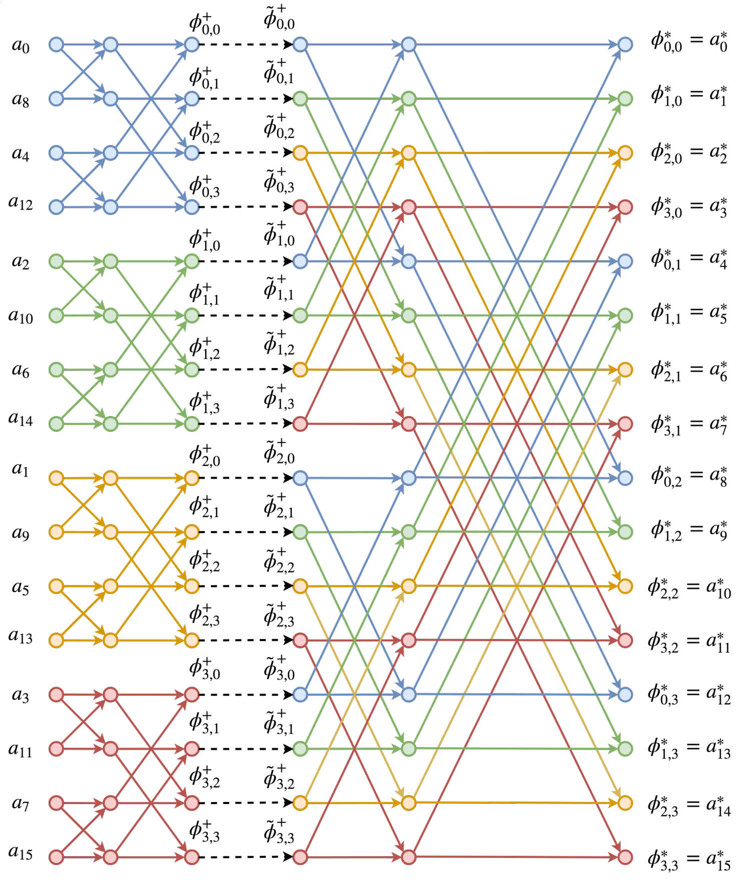
$$T^\dagger = \langle 1, \frac{1}{1}, 2, \frac{1}{2}, 18, \frac{1}{18}, 1, \frac{1}{1}, 3, \frac{1}{3}, 1, \frac{1}{1}, 4, \frac{1}{4}, 1, \frac{1}{1}, 2, \frac{1}{2}, 18, \frac{1}{18}, 1, \frac{1}{1} \rangle$$
$$P^\dagger = \langle 1, \frac{1}{1}, 0, 0, 1, \frac{1}{1} \rangle$$

$$C = T^\dagger * \mathrm{rev}(P^\dagger)$$



FFT in constant rounds

- Performing a **bit-reversal** operation makes the divide and conquer pattern clean.
- It is easy to decompose the **precedence** graph into the **Butterfly** graphs of different sizes.
- Cooley-Tukey with radix $R = n^{1-x}$.
- Further implications such as the **knapsack** problem.



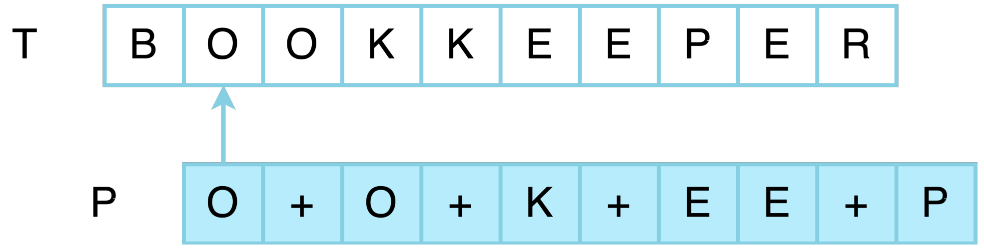
String Matching

with ‘+’ wildcard

String Matching with '+' wildcard

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'+'})^m$ are given.

- The special character '+' means that the preceding character can be repeated arbitrary times.

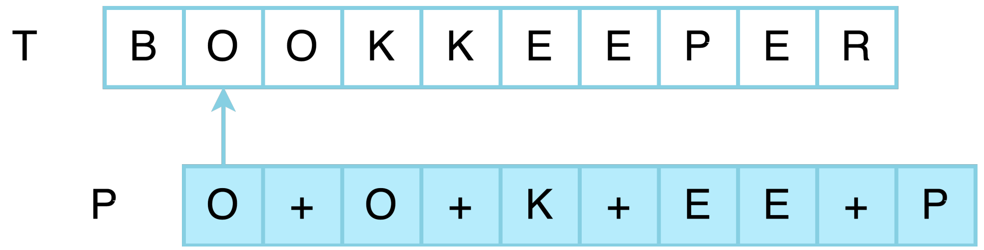


String Matching with ‘+’ wildcard

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{‘+’})^m$ are given.

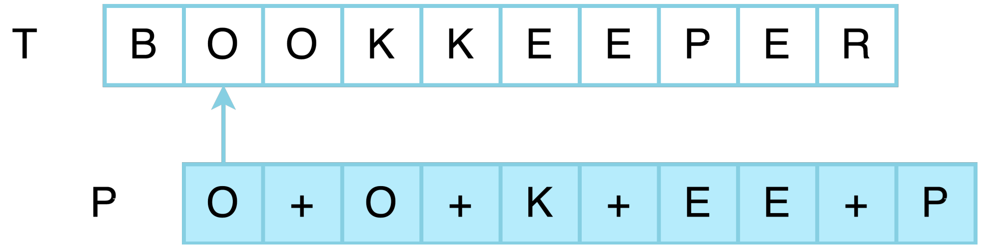
- There is a **constant**-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of P in T.

[HSSS 2019]



Run Length Encoding

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'+'})^m$ are given.



- Perform Run Length Encoding on both strings.

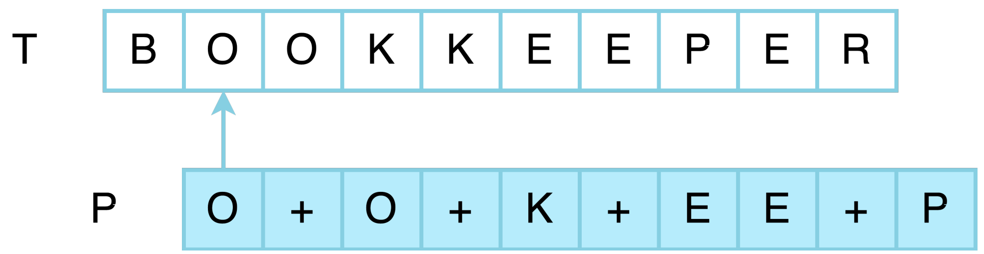
$$T^\circ = \langle\langle \mathsf{b}, 1\rangle, \langle \mathsf{o}, 2\rangle, \langle \mathsf{k}, 2\rangle, \langle \mathsf{e}, 2\rangle, \langle \mathsf{p}, 1\rangle, \langle \mathsf{e}, 1\rangle, \langle \mathsf{r}, 1\rangle\rangle$$
$$P^\circ = \langle\langle \mathsf{o}, 2+\rangle, \langle \mathsf{k}, 1+\rangle, \langle \mathsf{e}, 2+\rangle, \langle \mathsf{p}, 1\rangle\rangle$$

- Reduces to **Greater-than** matching.

Run Length Encoding

- Reduces to **Greater-than** matching.
- A special case of **Subset** matching.
- The **Subset** matching problem can be solved in $O(n \log^2 m)$ by a careful reduction to sparse convolution.

  [Cole et. al 2002]
- It's possible to implement it in $O(1)$ MPC rounds.

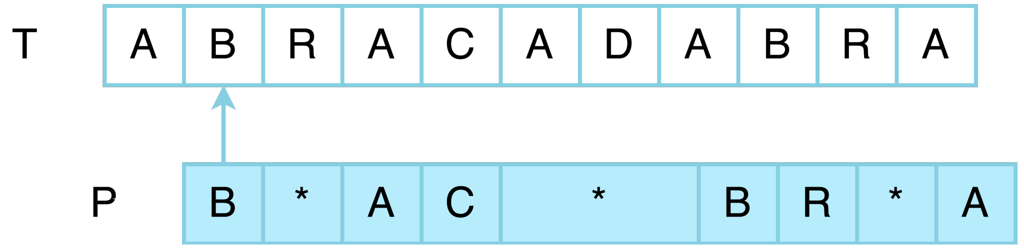


String Matching

with ‘*’ wildcard

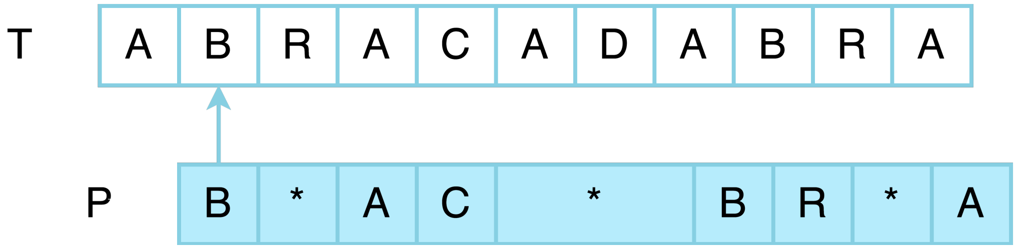
String Matching with '*' wildcard

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'}*\text{'})^m$ are given.
- The special character '$*$' can be replaced with any arbitrary string.



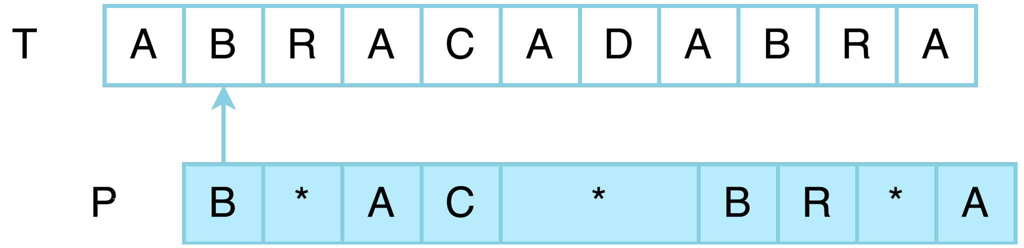
String Matching with '*' wildcard

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'}*\text{'})^m$ are given.
- The special character '$*$' can be replaced with any arbitrary string.
- Unlike '?' and '+', we have no positive result for this wildcard even in $O(\log n)$ rounds…
- There is a conjecture that we can't solve graph **connectivity** in $o(\log n)$ rounds.



String Matching with '*' wildcard

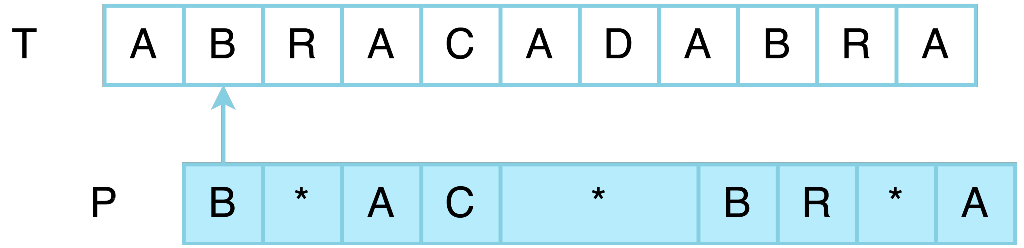
- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'}*\text{'})^m$ are given.
- The special character '*' can be replaced with any arbitrary string.
- Unlike '?' and '+', we have no positive result for this wildcard even in $O(\log n)$ rounds…
- But we can solve it in special cases.



'*' wildcard in **small** patterns

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'}*\text{'})^m$ are given such that $m = O(n^{1-x})$.
- There is a $O(\log n)$-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of P in T.

[HSSS 2019]

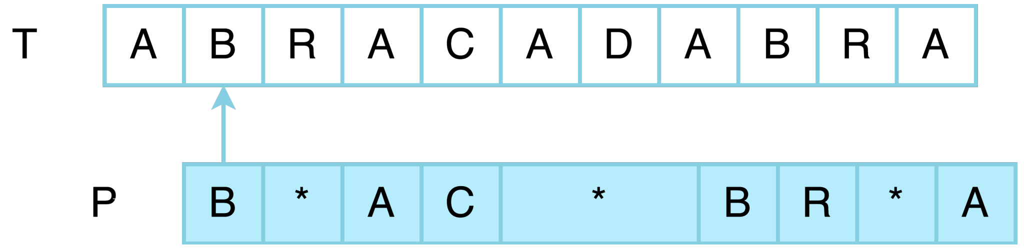


'*' wildcard in no **common prefix** case

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \text{'}*\text{'})^m$ are given such that no two **sub-patterns** share a common **prefix**.

- There is a $O(\log n)$-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of P in T.

[HSSS 2019]



Thanks for watching!

Any questions?