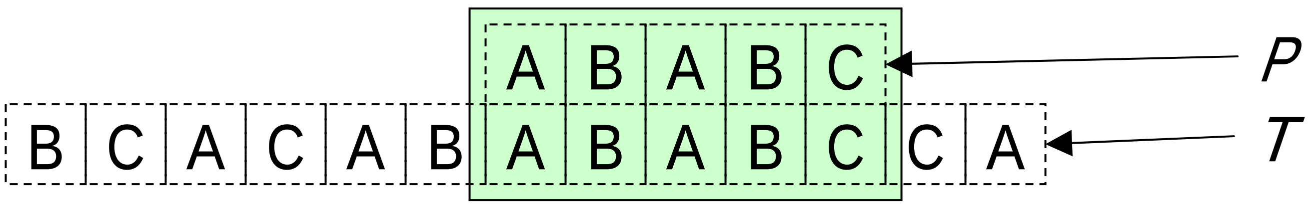
String Matching

Alexandre David
B2-206

The Problem

- Given a text *T* and a patter *P*, find an occurrence of *P* inside *T* or return *no match*.
- *T* is of size *t*, *P* is of size *p*.
- Example:

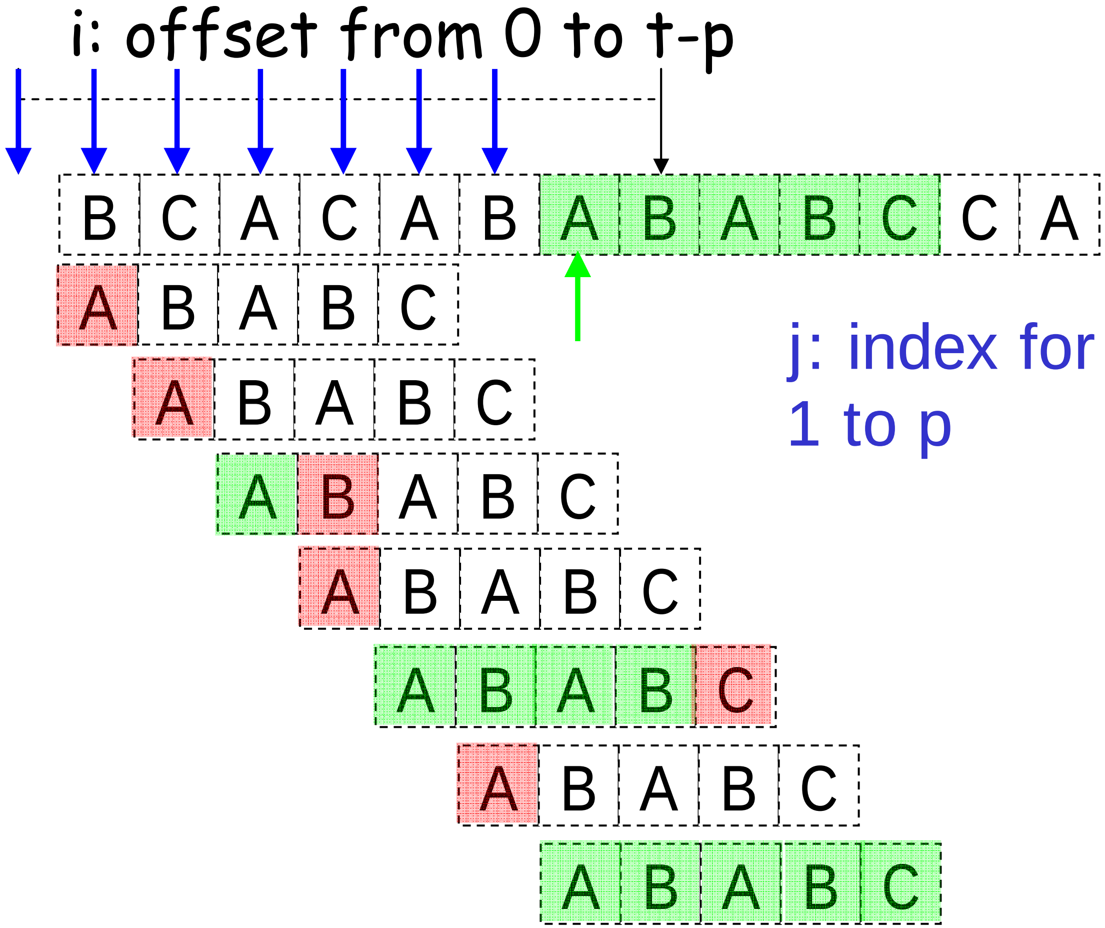


Naïve Solution

- Compare P to T starting at position 1.
  - If mismatch, move P to the right and try again.
  - If match, return current position.
- Worst-case: (t-p+1)*p comparisons, that is O((t-p)*p). If p=O(t) then we have O(t*p).

```
naïve_find(T,P):
p = length(P)
t = length(T)
for i = 0 to t-p do
  ok = true
  for j = 1 to p do
    if P[j] != T[i+j] then
      ok = false
      break
    fi
  done
  if ok then return i+1
done
return -1
```

Example

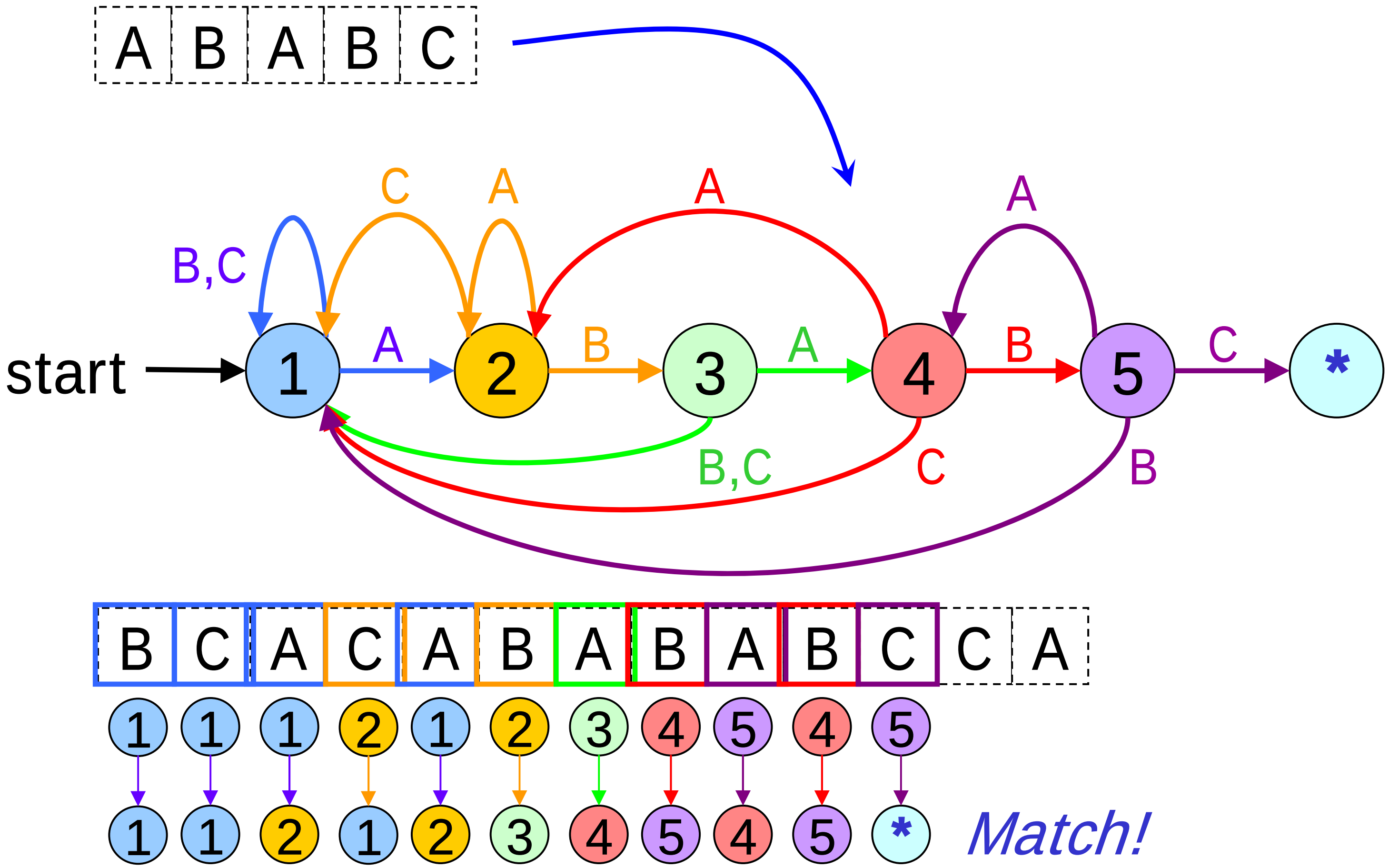


j: index for testing from 1 to p

Solution With Finite Automata

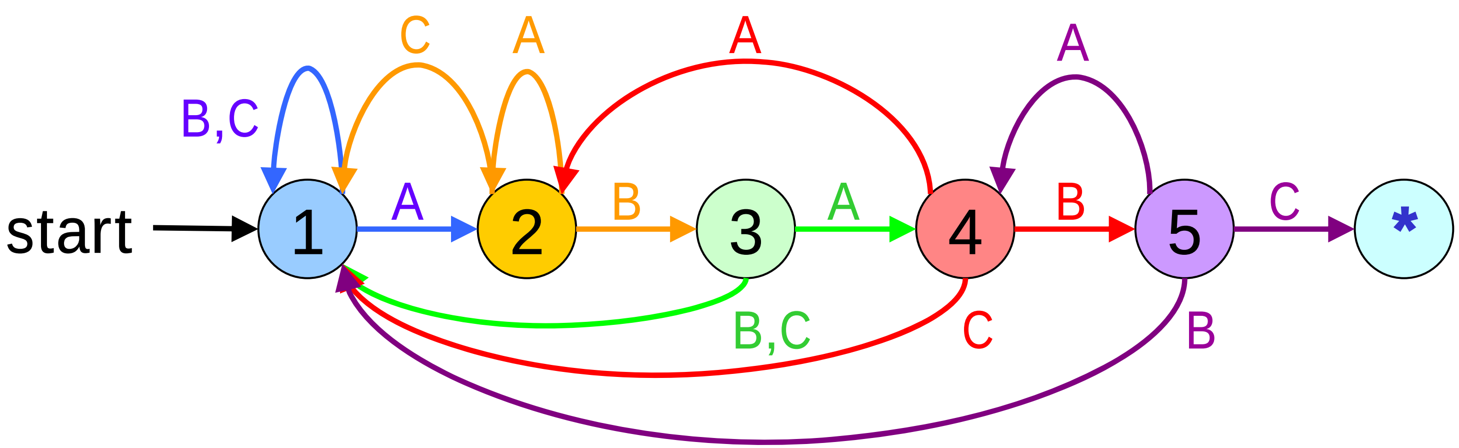
- Given *P*, it is possible to construct a finite automaton that is used to scan *T* in $O(t)$.
- Idea: Remember the last matched sub-string and re-use the information.
- Matching = reach the state *.
  No match = get stuck in the automaton.
- Pre-processing required: Construct the automaton in $O(p*|alphabet|)$.

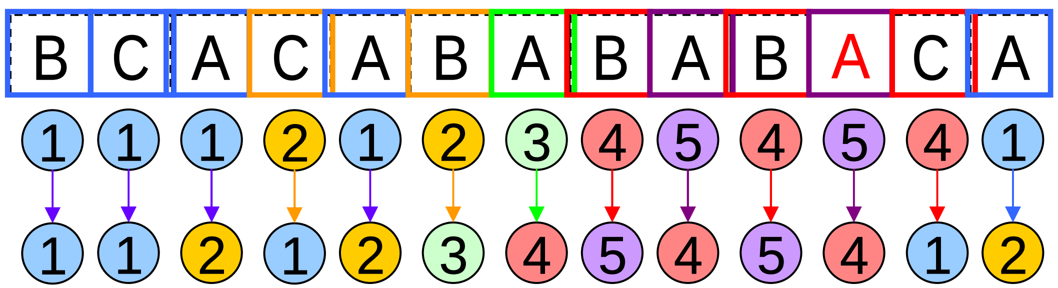
Example With Automaton



Example With Automaton

A	B	A	B	C



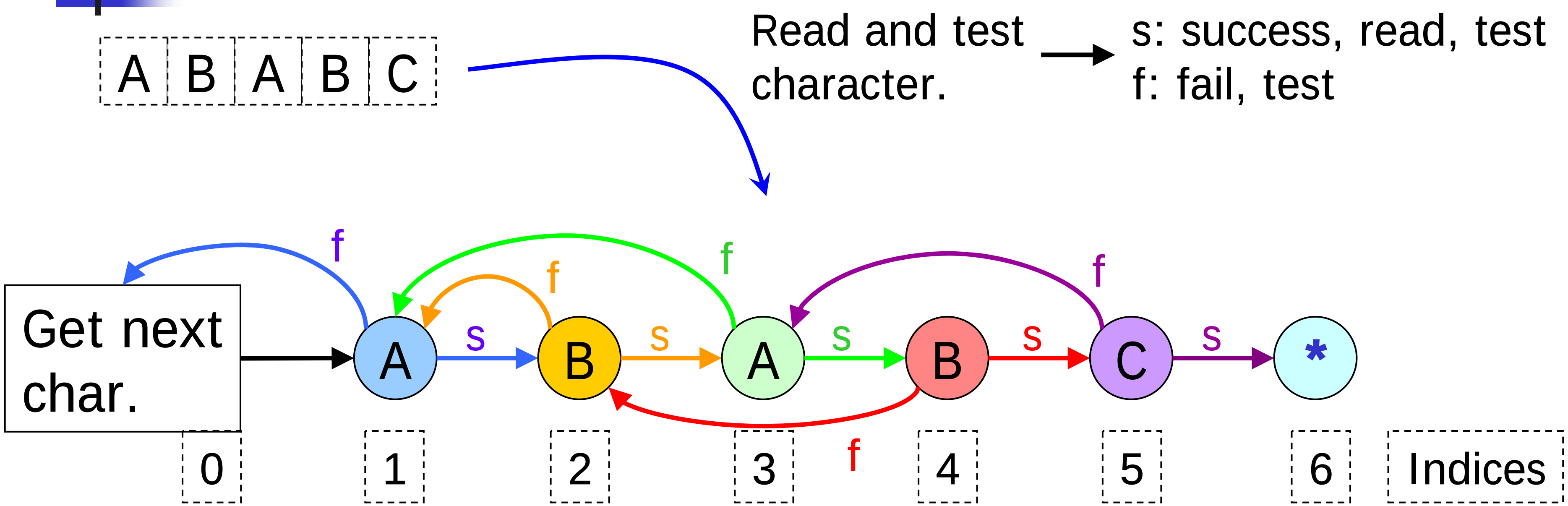


End of string, stuck in the automaton
⇒ No match!

Knuth-Morris-Pratt Flowchart

- Given *P*, it is possible to construct a finite flowchart that is used to scan *T* in $O(t+p)$.
- Idea is to remember the maximum of matchable characters before the i^{th} position.
- Matching = reach the state *****.
  No match = get stuck in the automaton.
- Pre-processing: Construct the flowchart in $O(p^2)$.

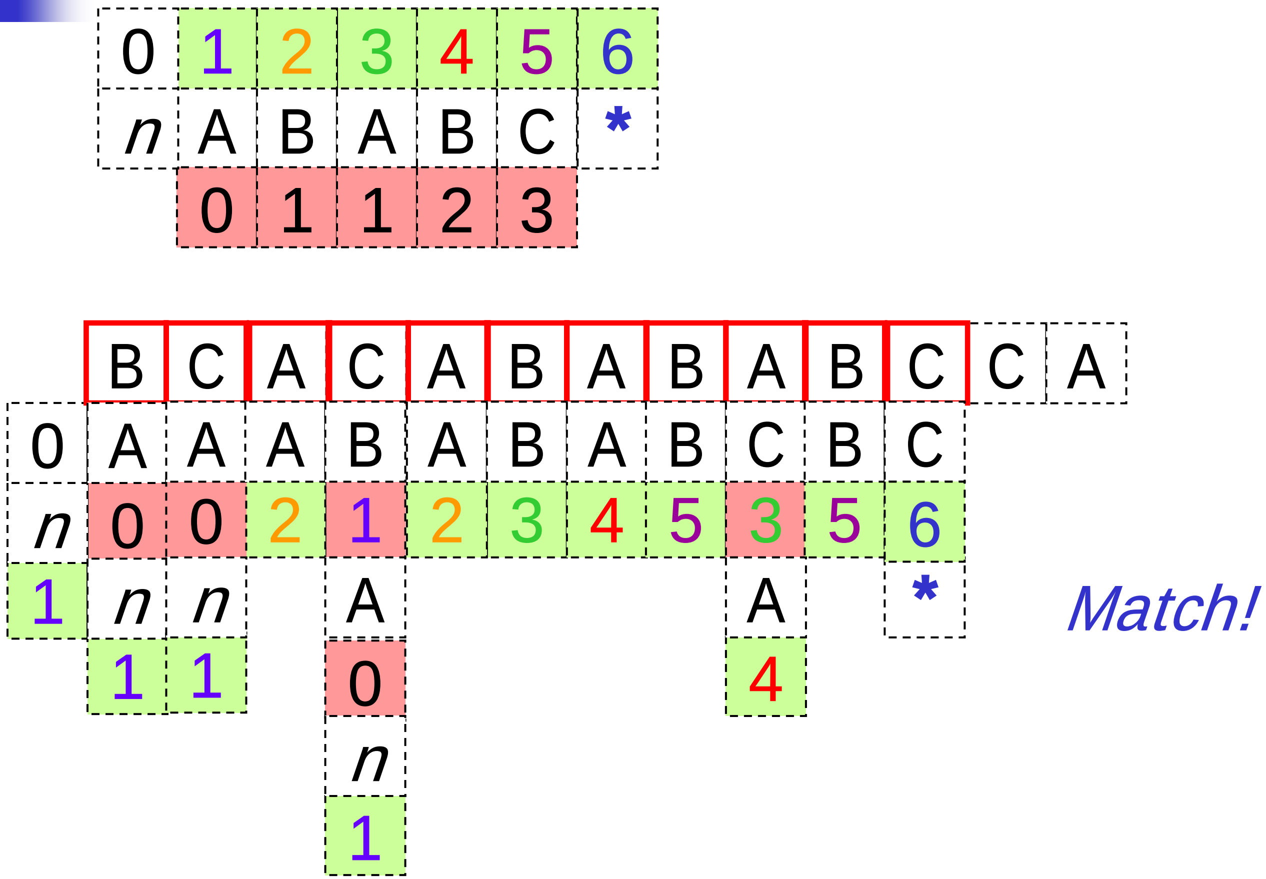
Example With Flowchart



Construct *next* table (f):

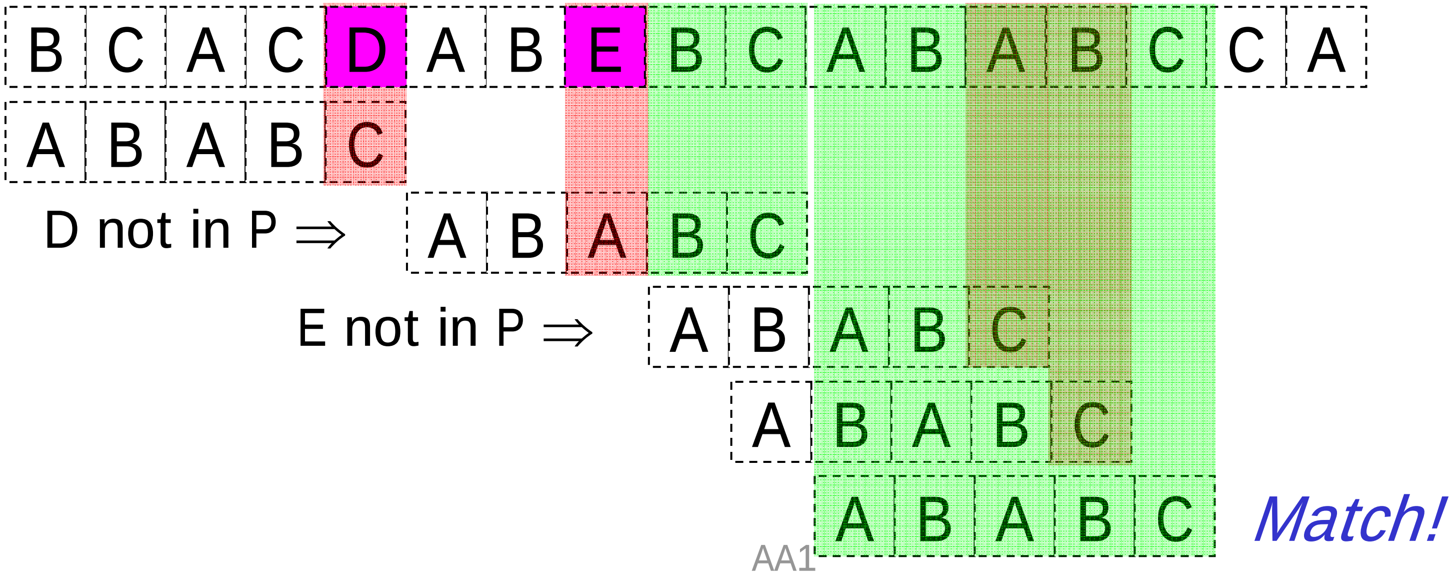
A	B	A	B	C	-
0	1	1	2	3	

Example With Flowchart



Boyer-Moore Algorithm

- Idea is to skip text without checking it. Scan from right to left, use heuristics to decide how far to jump.
- Average running time *O(t/p)*, worst *O(t*p)*.



Rabin-Karp Algorithm

- Use a hash function to identify equal strings! Very powerful for multi-pattern matching.
- Trick: Use a special hash function. Treat the character as a number in some base (usually a big prime) and compute the next hash iteratively.
  Hopefully, we have few collisions.
- Average running time $O(t)$, worst $O(t*p)$.

Rabin-Karp Algorithm

- Hash update = "shift" in the corresponding base.
- In practice, useful to use base 256 for characters and a prime as the hash table size.
  - Very fast and hash performs reasonably well.

Example With Rabin-Karp

A B A B C → $hash_p$ $O(p)$.

B C A C A B A B A B C C A

Initial hash $O(p)$. Test $O(1) \rightarrow$ no.

B C A C A B A B A B C C A

Update hash $O(1)$. Test $O(1) \rightarrow$ no.

B C A C A B A B A B C C A

Updates of hash $O(1)$ + tests $O(1)$... Test $O(1) \rightarrow$ yes.

Test P $O(p) \rightarrow$ yes.