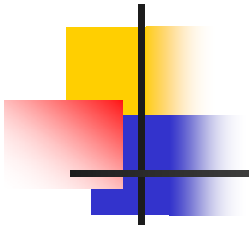
Song Classification for Dancing

Manolis Cristodoukalis, Costas Iliopoulos, M. Sohel Rahman, W.F. Smyth



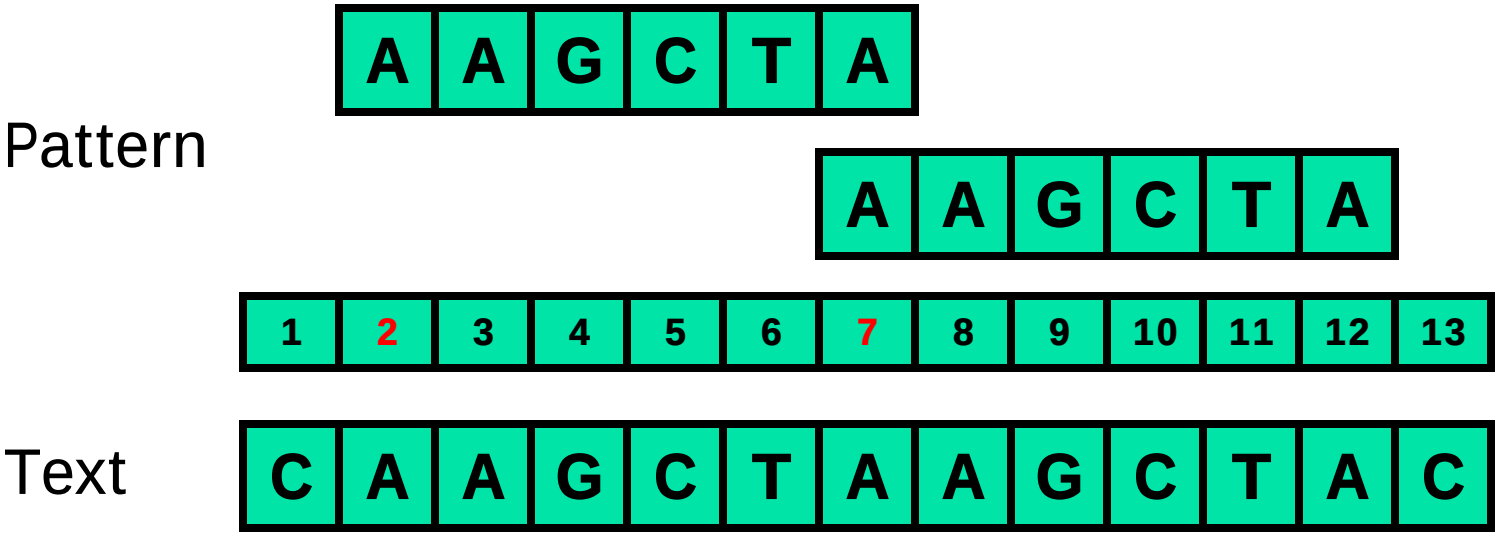
Classical Pattern Matching

- Input: A text T = T[1..n], A Pattern P = P[1..m], both over the alphabet Σ.
- Output:
  - Whether P occurs in T
  - If yes, then the set occ(P) = {i | T[i..i+m-1] = P}

Example – Pattern matching

Pattern P = AAGCTA

Text T = CAAGCTAAGCTAC



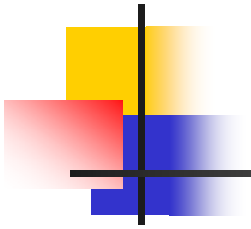
Occ(P) = {2, 7}

Our Case

- Input:
  - Text → A musical Sequence
  - Pattern → Rhythm
- Output:
  - Slightly different from Classical PM
  - A ‘extended/modified’ notion of PM

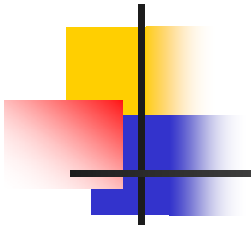
Musical Sequence

- A string t = t[1]t[2]...t[n]
  - t[i] $\in$ N^+
- Example:
  - [0,50,100,200,250,300,350,400,500,550]
  - Significance: Sequence of events!



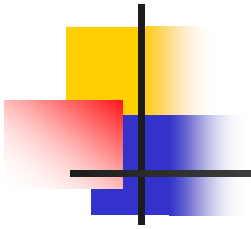
Musical Sequence

- A string t = t[1]t[2]...t[n]
  - t[i] $\in$ N^+
- Example:
  - [0,50,100,200,250,300,350,400,500,550]
  - Some (musical) event occurs at 0 ms



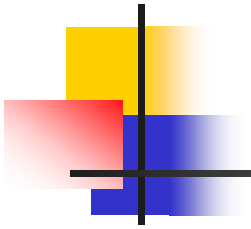
Musical Sequence

- A string t = t[1]t[2]...t[n]
  - t[i] $\in$ N^+
- Example:
  - [0,50,100,200,250,300,350,400,500,550]
  - Some (musical) event occurs at 0 ms
  - Some (musical) event occurs at 50 ms



Musical Sequence

- A string t = t[1]t[2]...t[n]
  - t[i] $\in$ N^+
- Example:
  - [0,50,100,200,250,300,350,400,500,550]
  - Some (musical) event occurs at 0 ms
  - Some (musical) event occurs at 50 ms
  - Some (musical) event occurs at 100 ms
  - And so on...

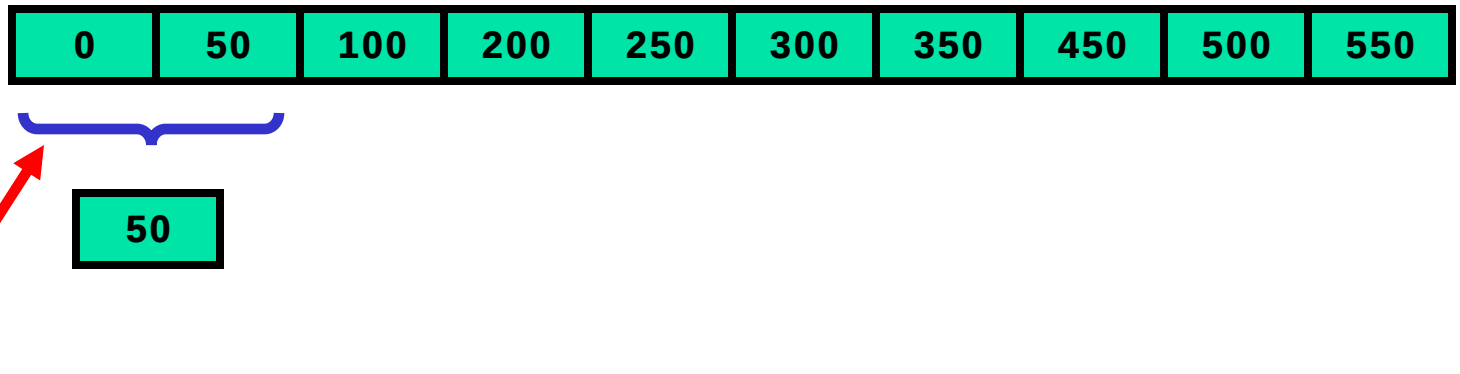


Musical Sequence

- [0,50,100,200,250,300,350,400,500,550]
- Alternative Representation (We use)
  - [50,50,100,50,50,50,50,100,50]
  - Significance: Sequence of “duration” of the events!

Musical Sequence

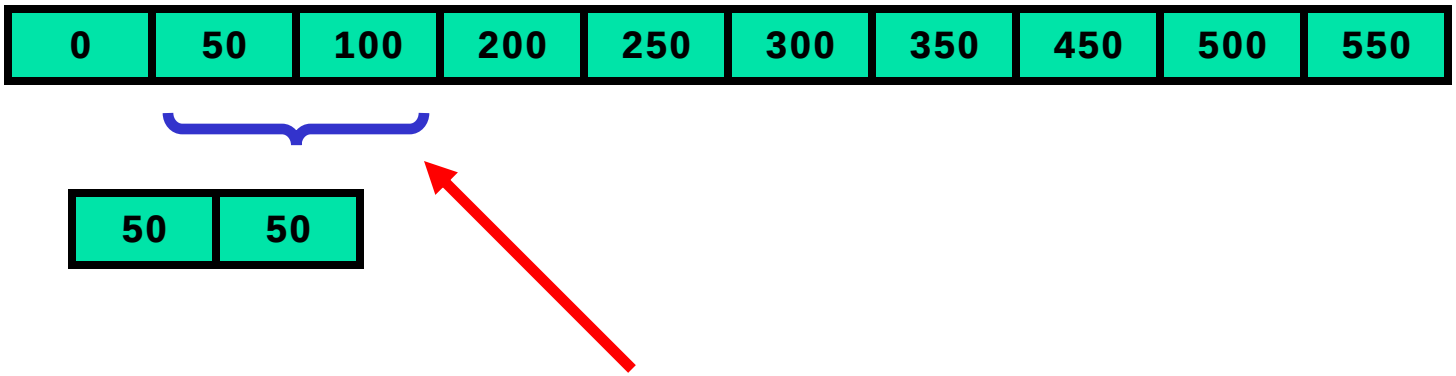
First Representation



Duration of First Event = 50 – 0 = 50

Musical Sequence

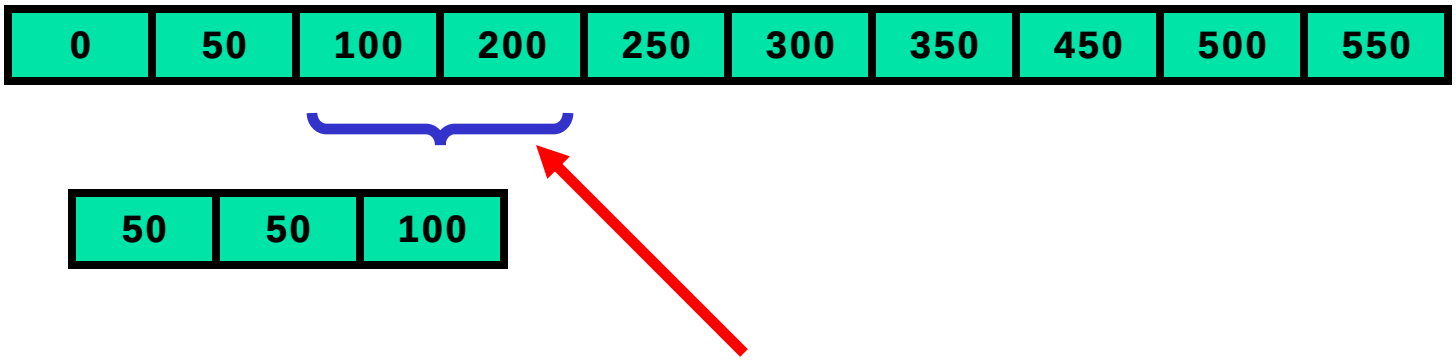
First Representation



Duration of Second Event = 100 – 50 = 50

Musical Sequence

First Representation

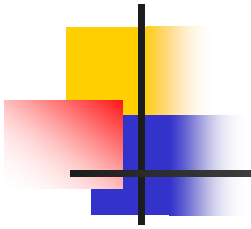


Duration of Second Event = 200 – 100 = 100

Musical Sequence

First Representation

Alternative Representation



Rhythm

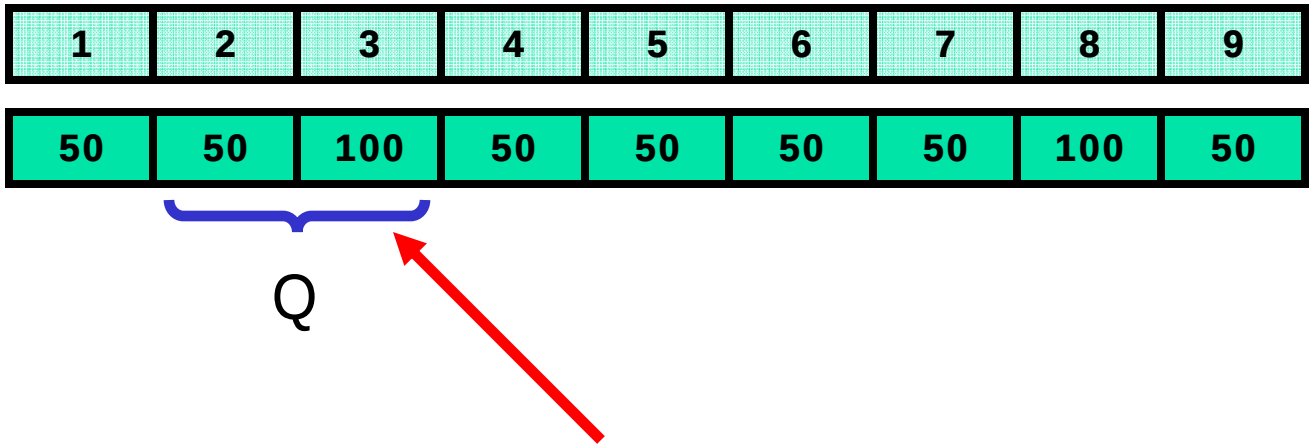
- A string r = r[1]r[2]...r[m]
  - r[j] ∈ {Q, S}
  - Q means a quick(er) event
  - S means a slow(er) event
  - S is double the length (duration) of Q
  - Exact length of Q or S is not a priori known

New Notion of Match

- Let $Q \equiv q \in N^{+}$
- Then $S \equiv 2 \times q$
- Q matches t[i..i'] iff
  - $q = t[i] + t[i+1] + \ldots + t[i']$
    - $1 \leq i \leq i' \leq n$
  - If i = i' then the match is SOLID

New Notion of Match

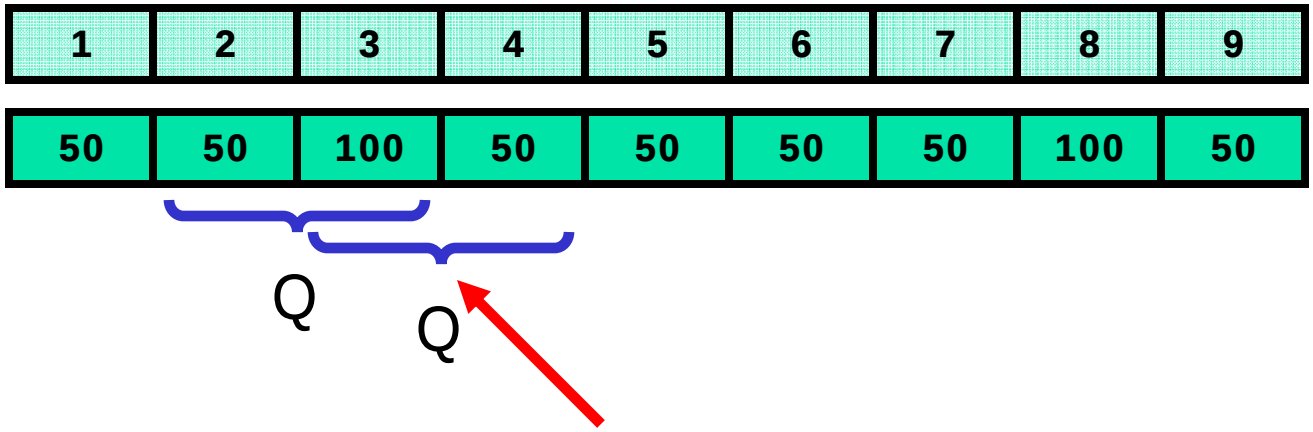
Let q = 150



50+100 = 150 = q

New Notion of Match

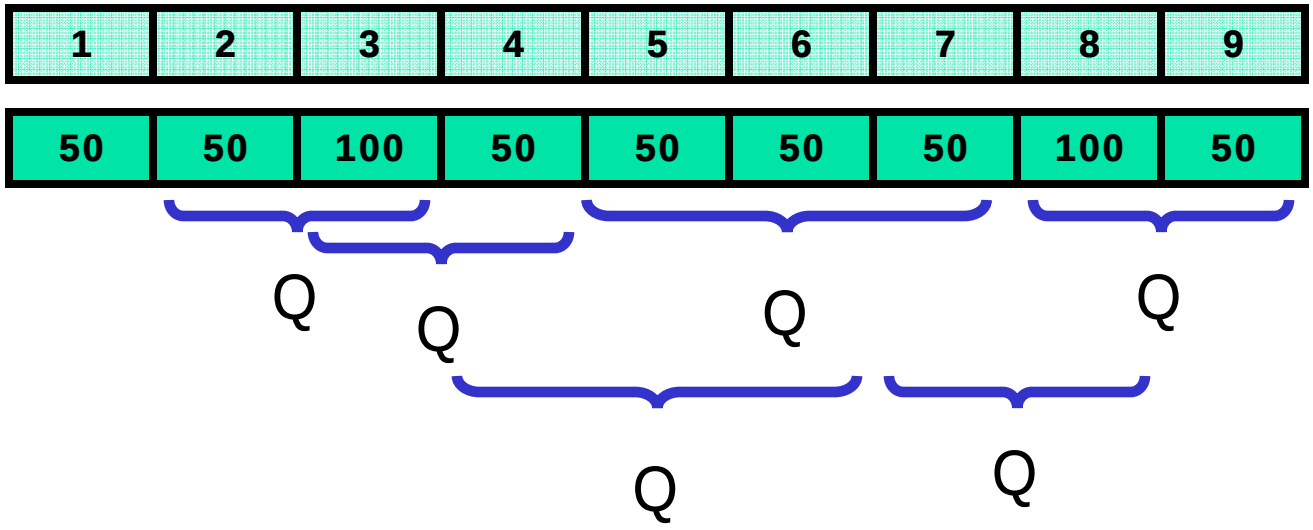
Let q = 150



100 + 50 = 150 = q

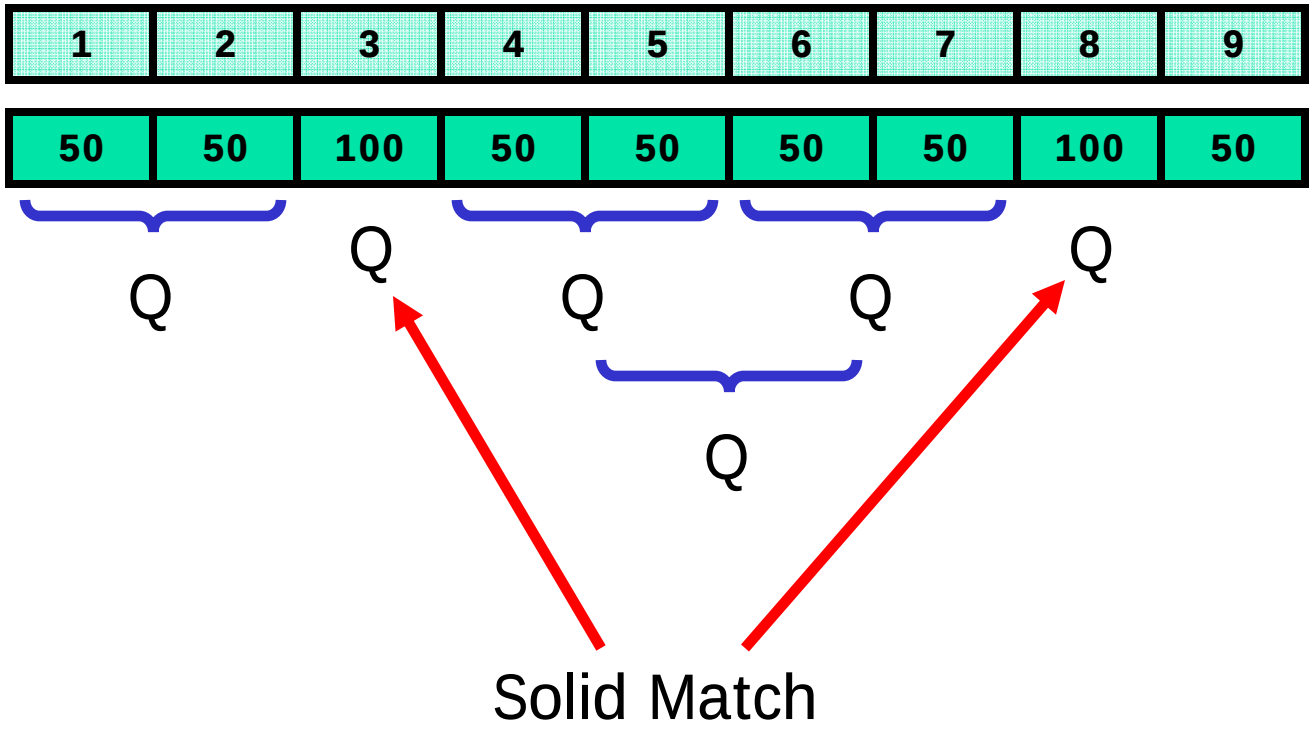
New Notion of Match

Let q = 150



New Notion of Match

Let q = 100

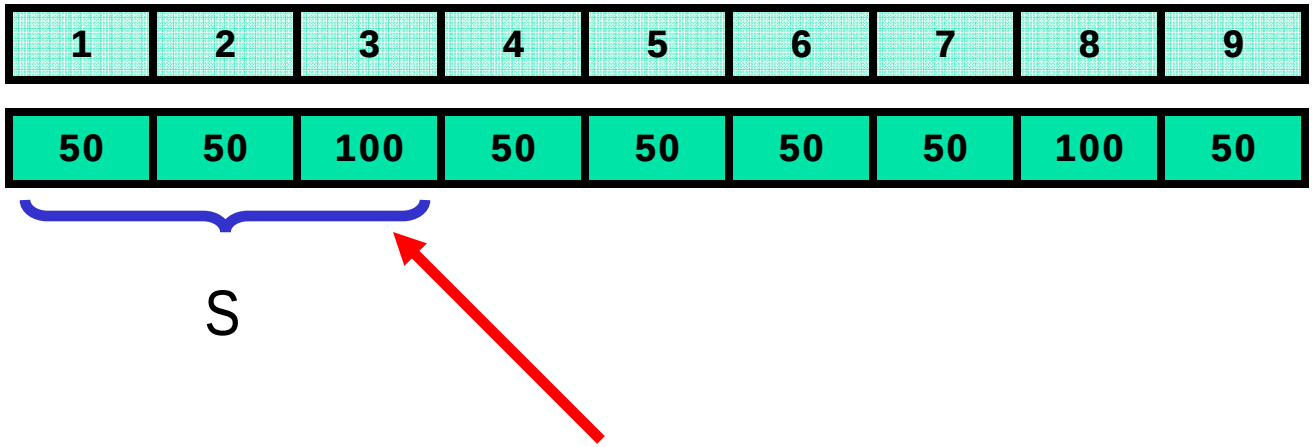


New Notion of Match

- $S \equiv 2 \times q$
- S matches t[i..i'] iff either of following is true:
  - i = i' and t[i] = 2q (SOLID)
  - $i \neq i'$ and there exists $i \leq i_1 \leq i'$ such that
    - $q = t[i] + t[i+1] + \ldots + t[i_1] = t[i_1+1] + t[i+1] + \ldots + t[i']$
    - $1 \leq i \leq i' \leq n$
- So, S is either solid or a tile of 2 consec utive Q's

New Notion of Match

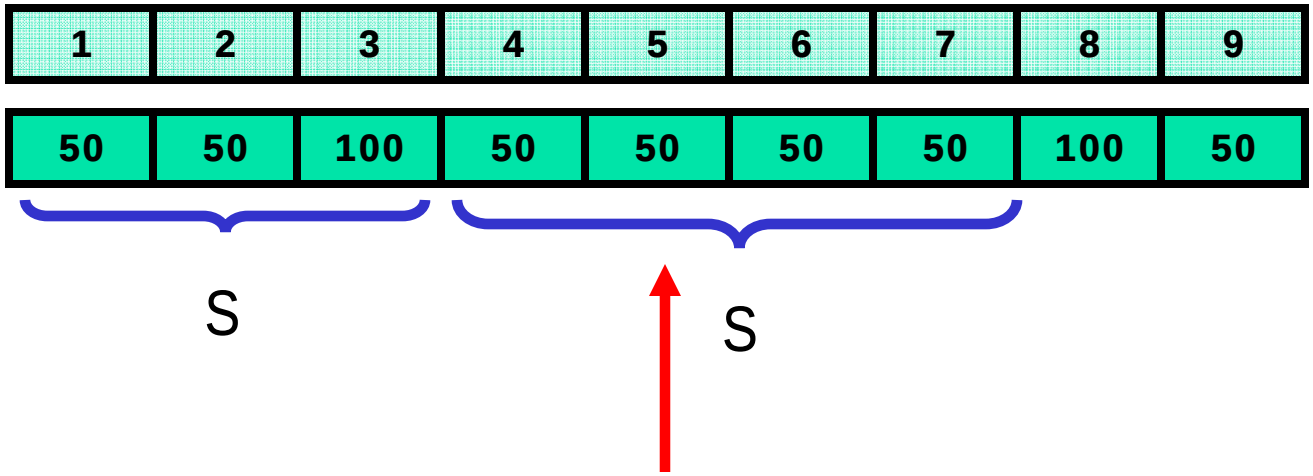
Let q = 100, then S ≡ 2q = 200



t[1] + t[2] = 100 = t[3]

New Notion of Match

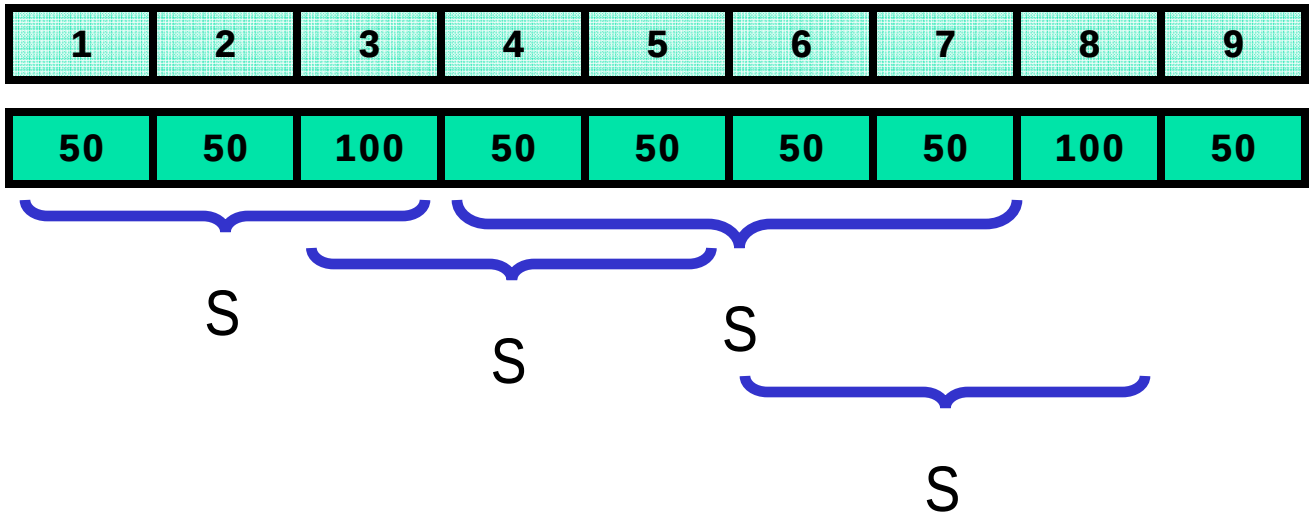
Let q = 100, then S ≡ 2q = 200



t[4] + t[5] = 100 = t[6] + t[7]

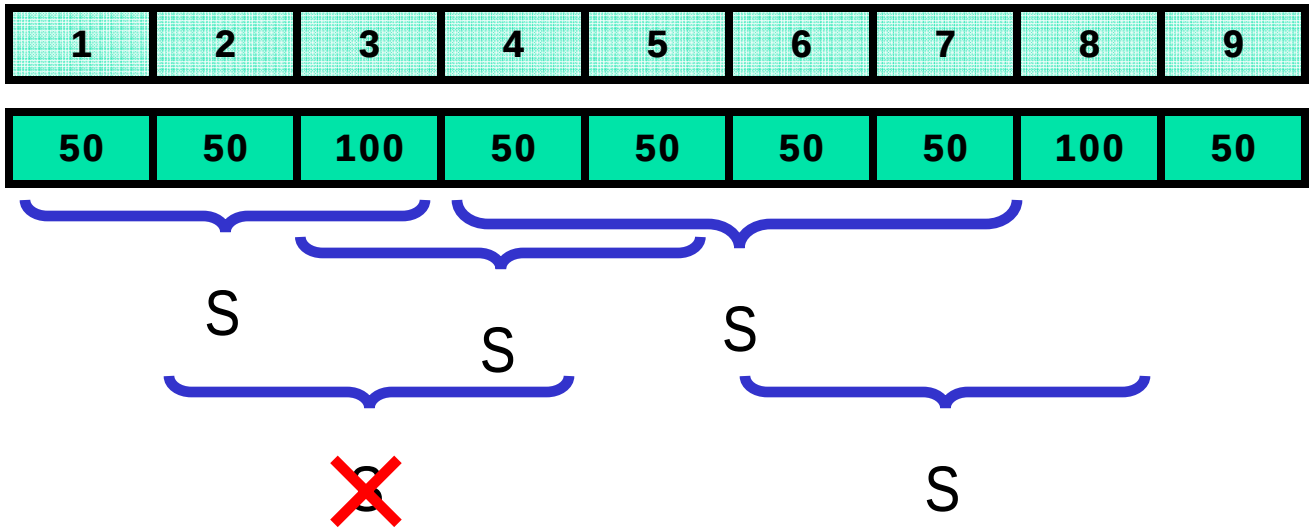
New Notion of Match

Let q = 100, then S ≡ 2q = 200



New Notion of Match

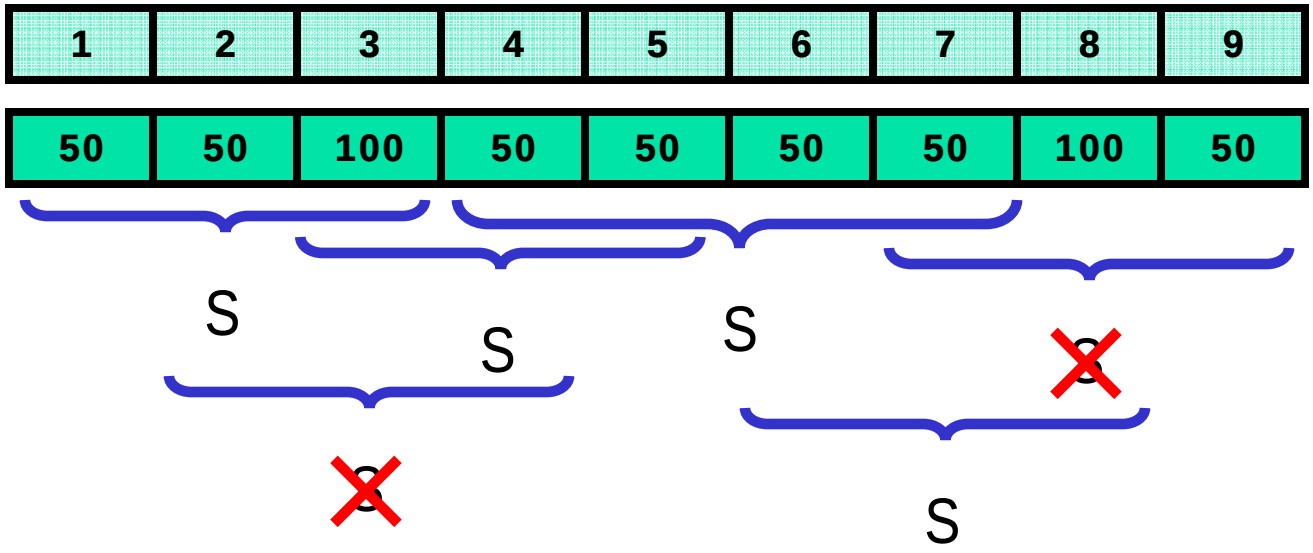
Let q = 100, then S ≡ 2q = 200



Although, t[2] + t[3] + t[4] = 200 = 2q

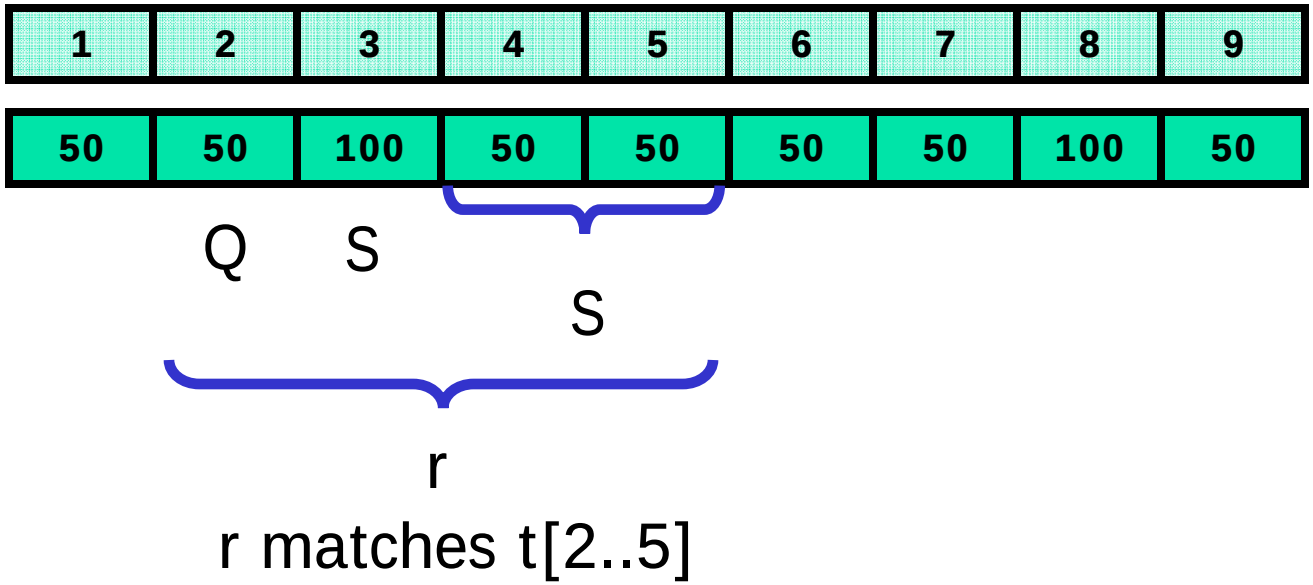
New Notion of Match

Let q = 100, then S ≡ 2q = 200



New Notion of Match

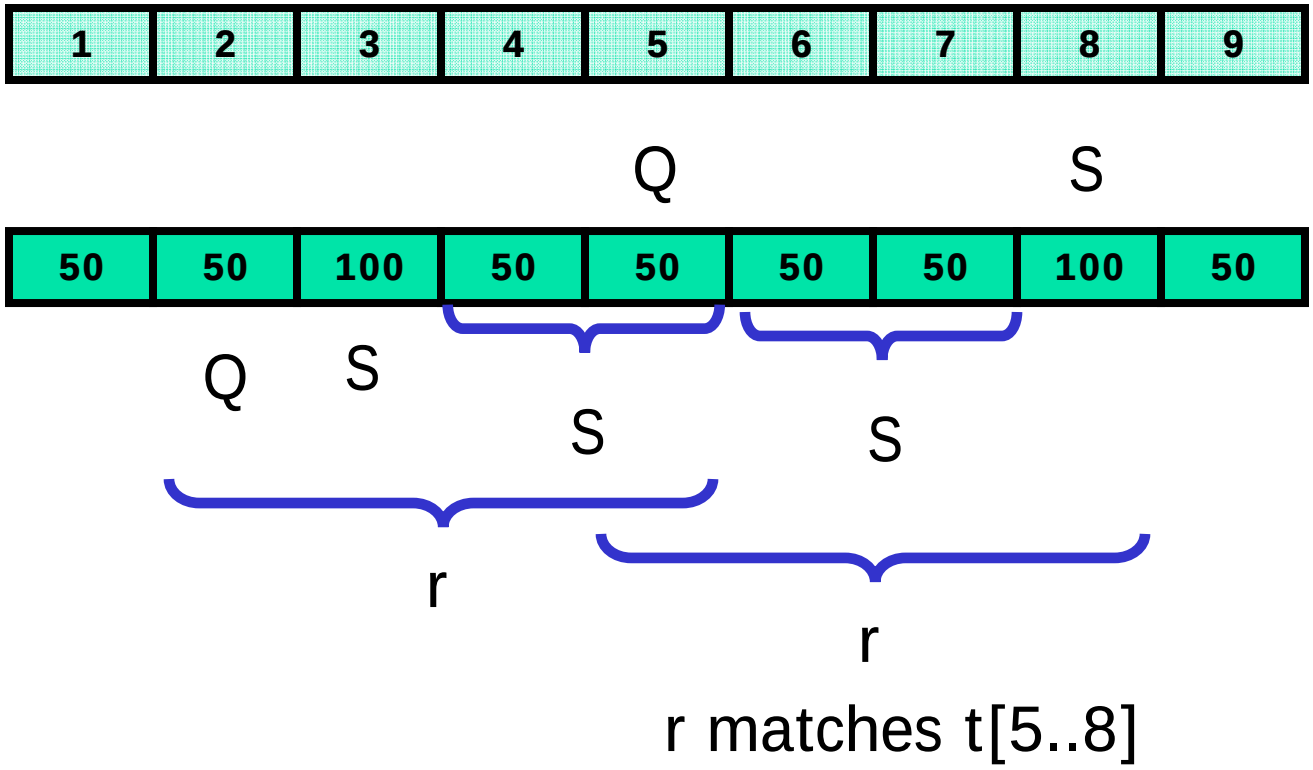
Let r = QSS and Let q = 50



r matches t[2..5]

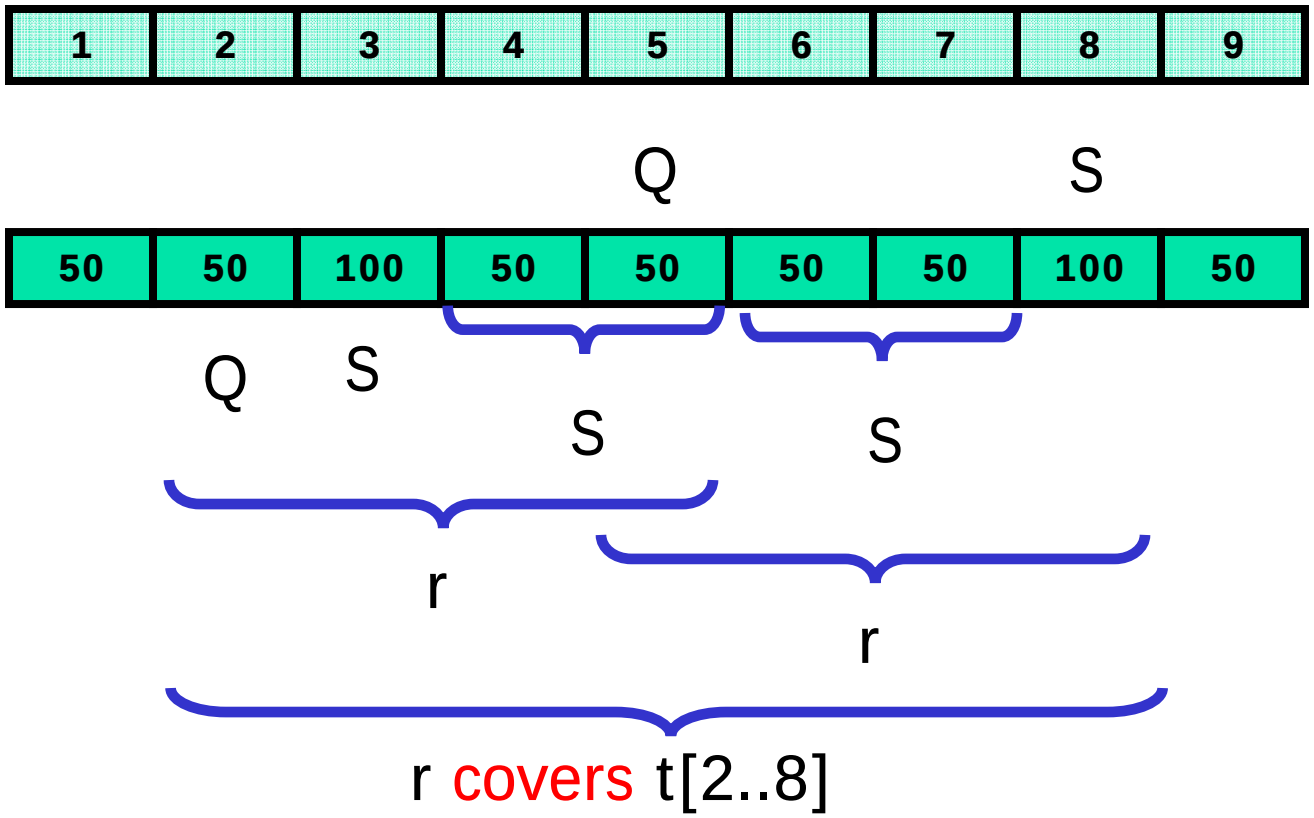
New Notion of Match

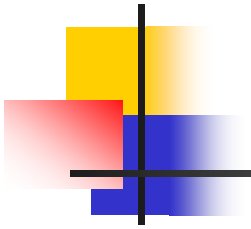
Let r = QSS and Let q = 50



New Notion of Match

Let r = QSS and Let q = 50





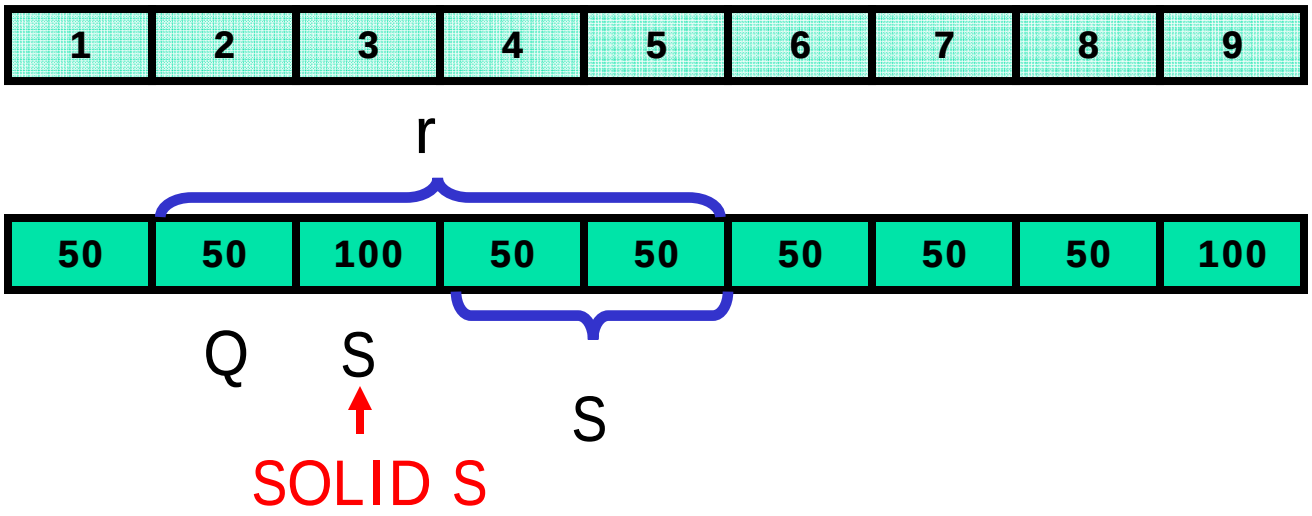
Our Problem

- Input:
  - A musical Sequence t
  - A rhythm r
- Output:
  - The longest substring t[i..i’] that is covered by r

Restriction

- For each match of r, at least one S mus t be solid

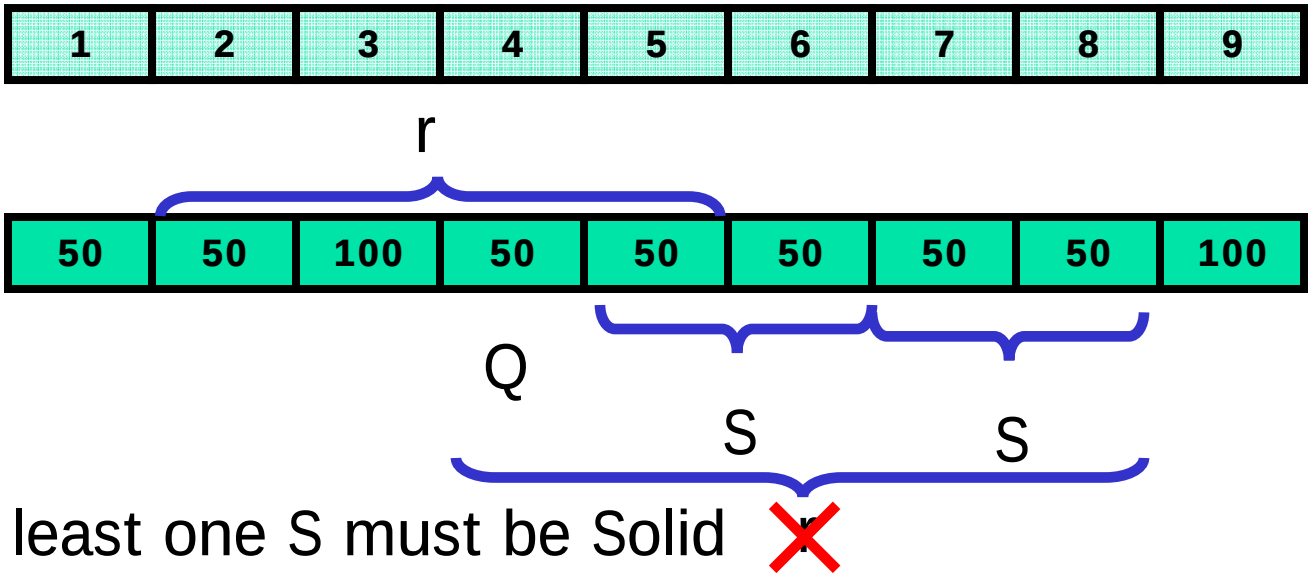
  Let r = QSS and Let q = 50



Restriction

- For each match of r, at least one S mus t be solid

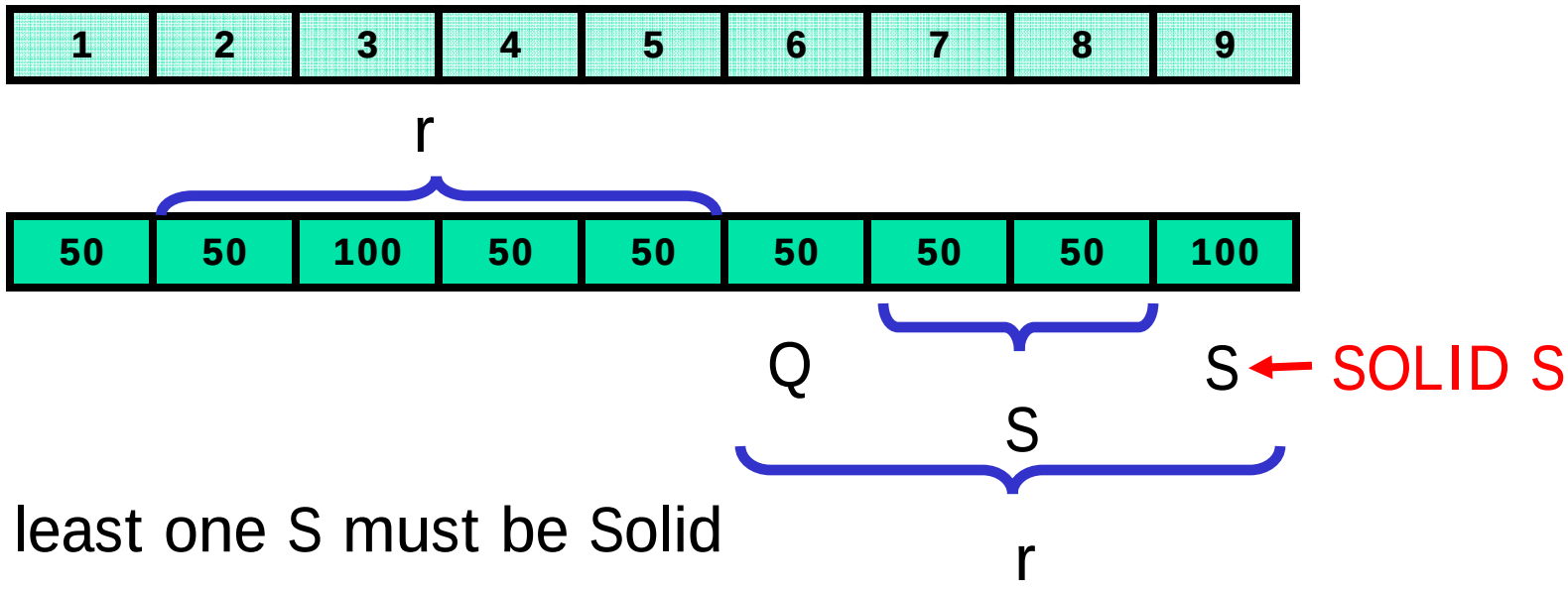
  Let r = QSS and Let q = 50



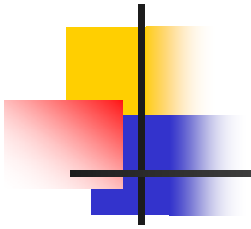
Restriction

- For each match of r, at least one S mus t be solid

  Let r = QSS and Let q = 50

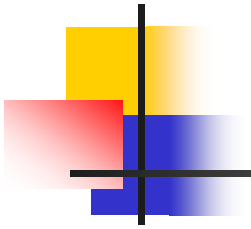


At least one S must be Solid



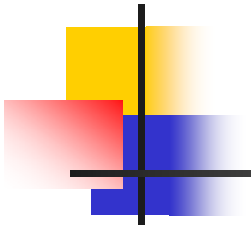
Motivation

- Our aim is to classify music according to dancing rhythm
- Music seq can be considered as a series of events corresponding to music signals
  - Drum beats
  - Guiter picks
  - Horn hits



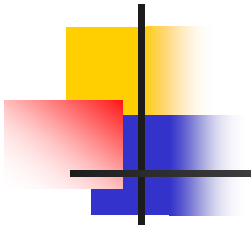
Motivation

- The intervals between these events cha racterize how the song is danced
- Basically two dancing rhythms: Quick & Slow
- Example:
  - cha-cha ≡ SSQQSSSQQS
  - foxtrot ≡ SSQQSSQQ
  - jive ≡ SSQQSQQS



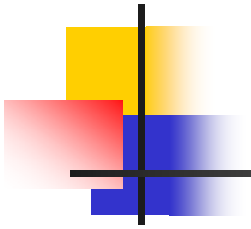
Motivation

- So solution to our problem can classify songs according to dancing rhythms!



Algorithm

- Stage 1: Find all occurrences of S for a chosen value $\sigma \in \Sigma$ such that $\sigma/2 \in \Sigma$.
- Stage 2: Transform areas around each S into sequences of Q.
- Stage 3: Find the matches of r and cons equently the cover.



Algorithm: Stage 1

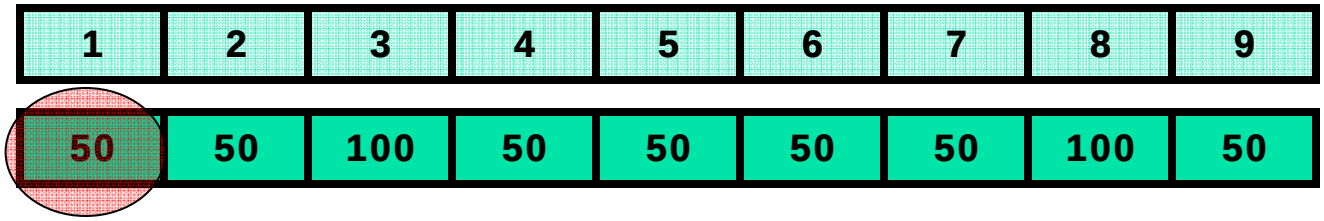
- We construct two arrays:
  - first[1..|Σ|]: first[σ] = i iff the first occurrence of symbol σ appears at position i.
  - next[1..n]: next[i] = j iff the next occurrence of symbol at t[i] appears at t[j].

Algorithm: Stage 1

1	2	3	4	5	6	7	8	9
50	50	100	50	50	50	50	100	50

Σ = {50, 100}, assume indexed alphabet

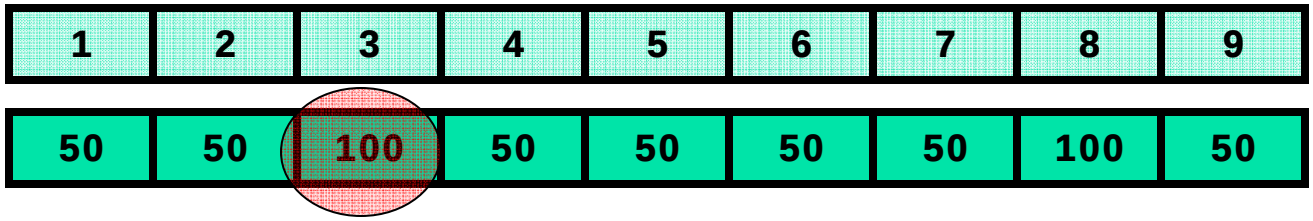
Algorithm: Stage 1



Σ = {50, 100}, assume indexed alphabet

first[1] = 1 First occurrence of 50

Algorithm: Stage 1

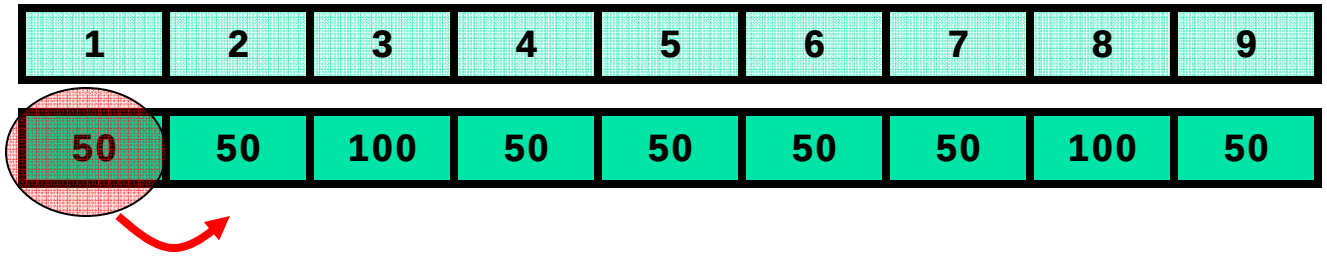


Σ = {50, 100}, assume indexed alphabet

first[1] = 1

first[2] = 3 First occurrence of 100

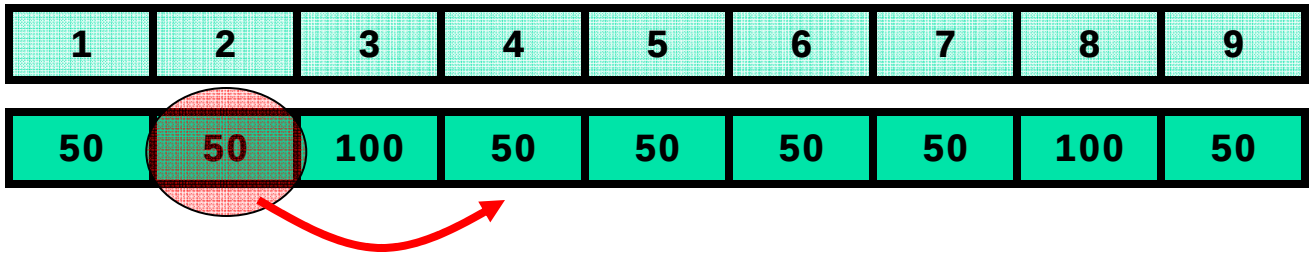
Algorithm: Stage 1



next[1] = 2

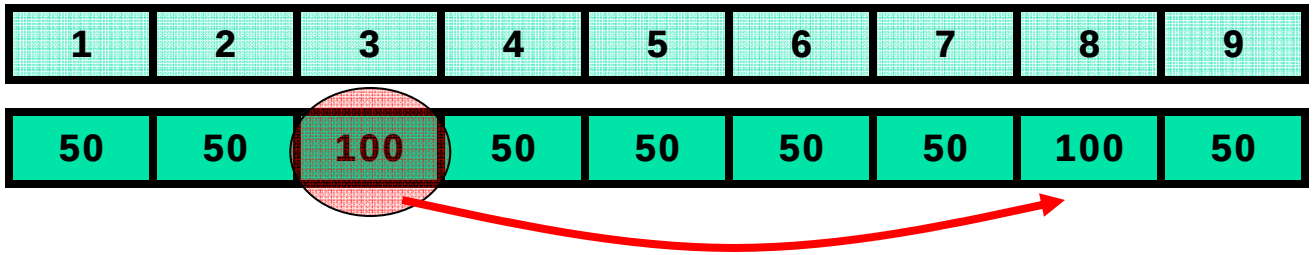
Next occurrence of t[1] = 50

Algorithm: Stage 1



next[1] = 2 next[2] = 4 Next occurrence of t[2] = 50

Algorithm: Stage 1

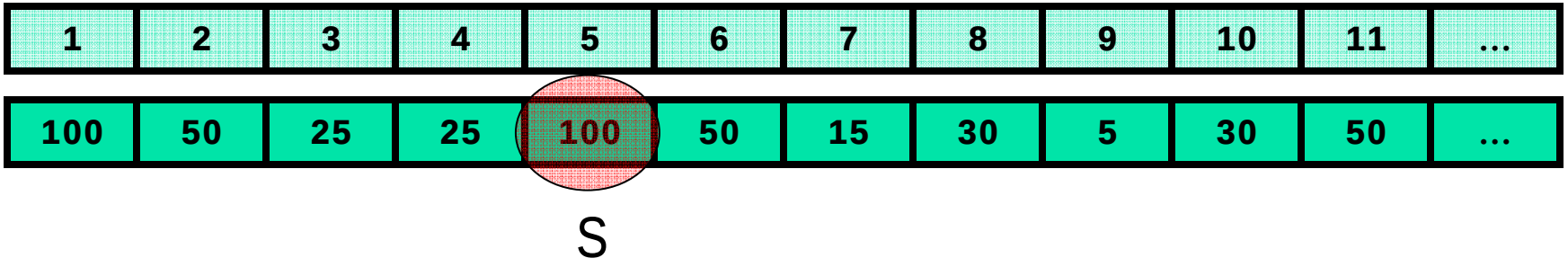


next[1] = 2 next[2] = 4

next[3] = 8 Next occurrence of t[3] = 100

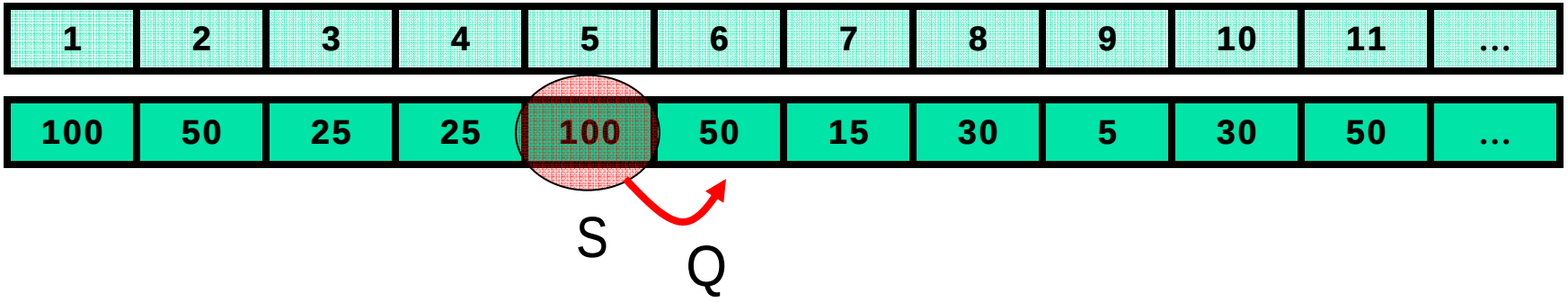
And so on…

Algorithm: Stage 2

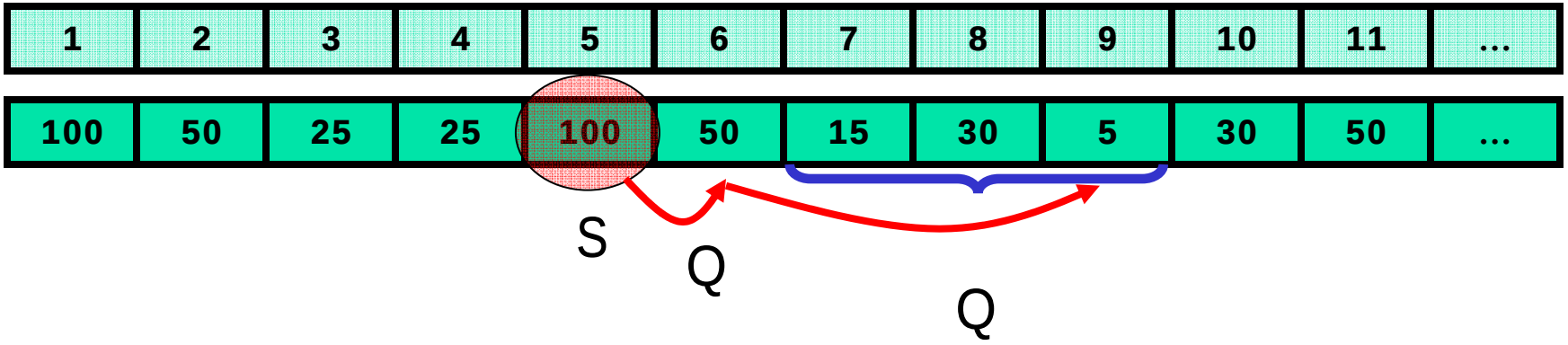


Consider an S at t[5].

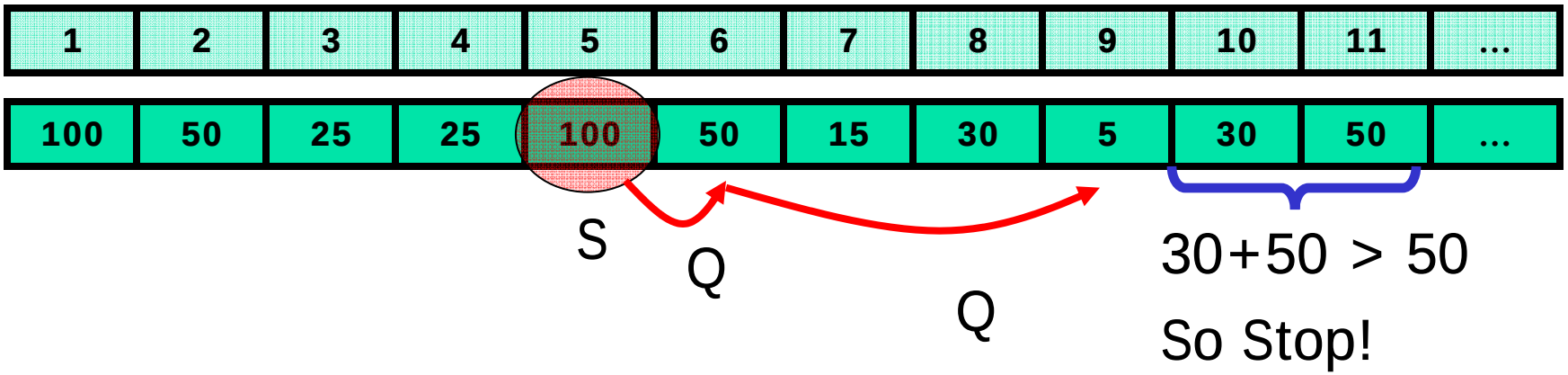
Algorithm: Stage 2



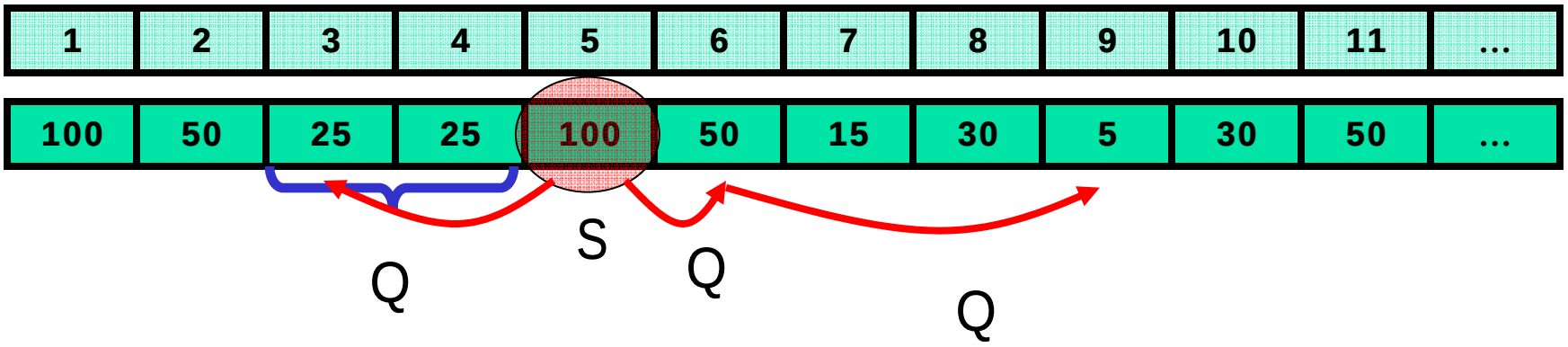
Algorithm: Stage 2



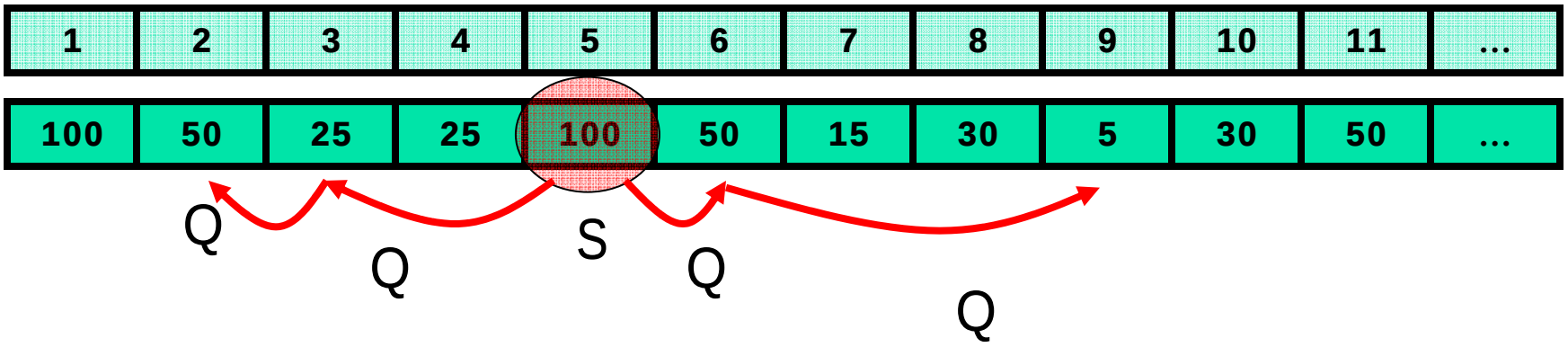
Algorithm: Stage 2



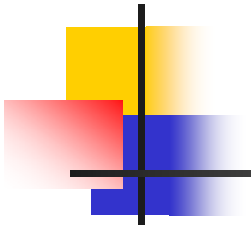
Algorithm: Stage 2



Algorithm: Stage 2



So we get a new sequence t′ consisting of Q and S

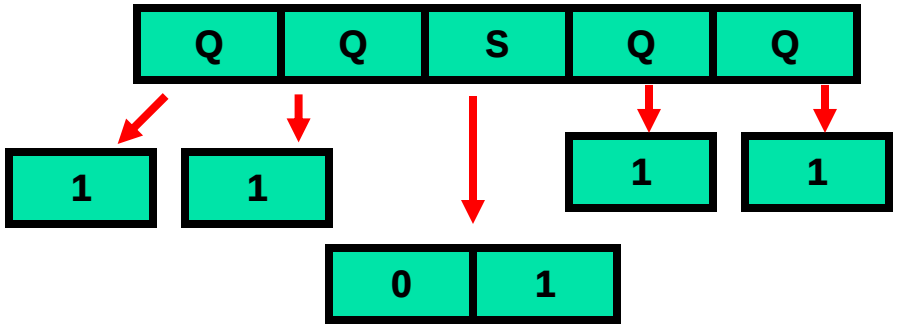


Algorithm: Stage 3

- Here we get t' consisting of Q and S.
- We first want to find the matches of r in t'.
- Define $S_{t'} \equiv S$ in t', $Q_{t'} \equiv Q$ in t'
- Define $S_r \equiv S$ in r, $Q_r \equiv Q$ in r

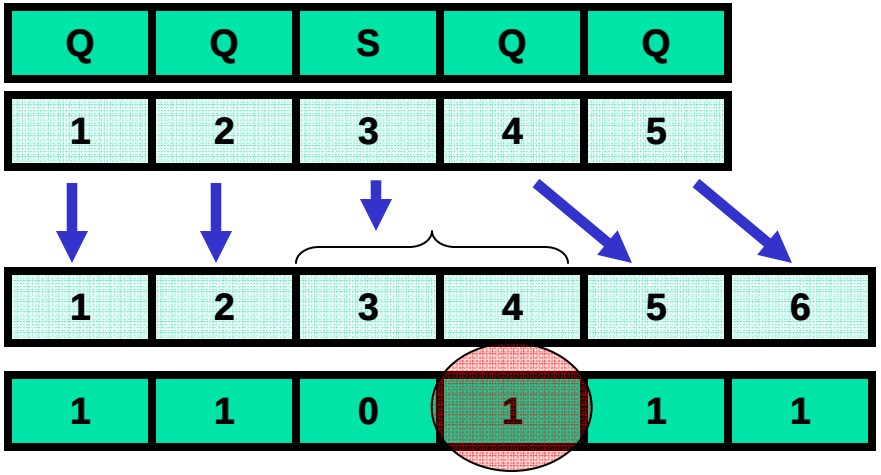
Algorithm: Stage 3

- Construct t′′ as follows:
  - $S_{t'} = 01$, $Q_{t'} = 1$
- Construct Invalid set I where I contains the position of 1 due to $S_{t'}$.



Algorithm: Stage 3

- Construct t′′ as follows:
  - $S_{t'} = 01$, $Q_{t'} = 1$
- Construct Invalid set I where I contains the position of 1 due to $S_{t'}$.

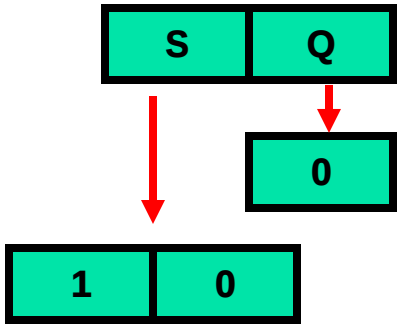


No match can occur at position 4 because it is not a ‘real’ position in t′

I = {4}

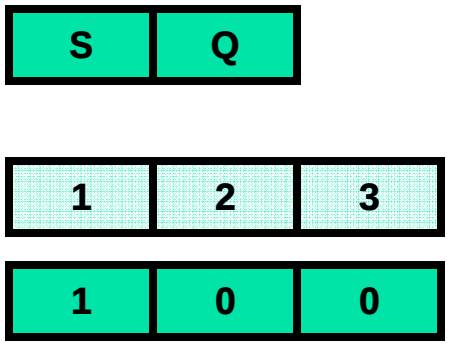
Algorithm: Stage 3

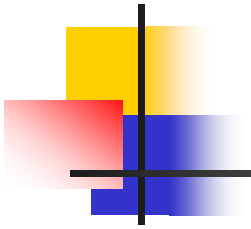
- Construct r′ as follows:
  - $S_r = 10$, $Q_r = 0$



Algorithm: Stage 3

- Construct r′ as follows:
  - $S_r = 10$, $Q_r = 0$



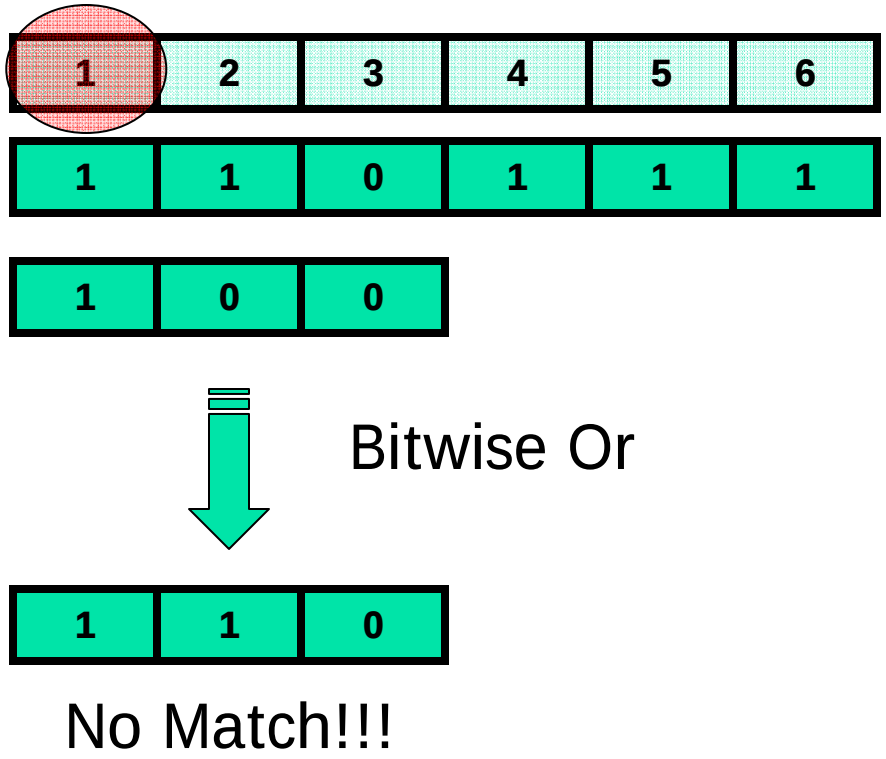


Algorithm: Stage 3

- Find matches of r’ in t’’.
  - Perform bitwise or at each position.
  - If the result is all 1 and the position is not i n invalid set then a match
  - Otherwise no match

Algorithm: Stage 3

- Find matches of r’ in t’’.

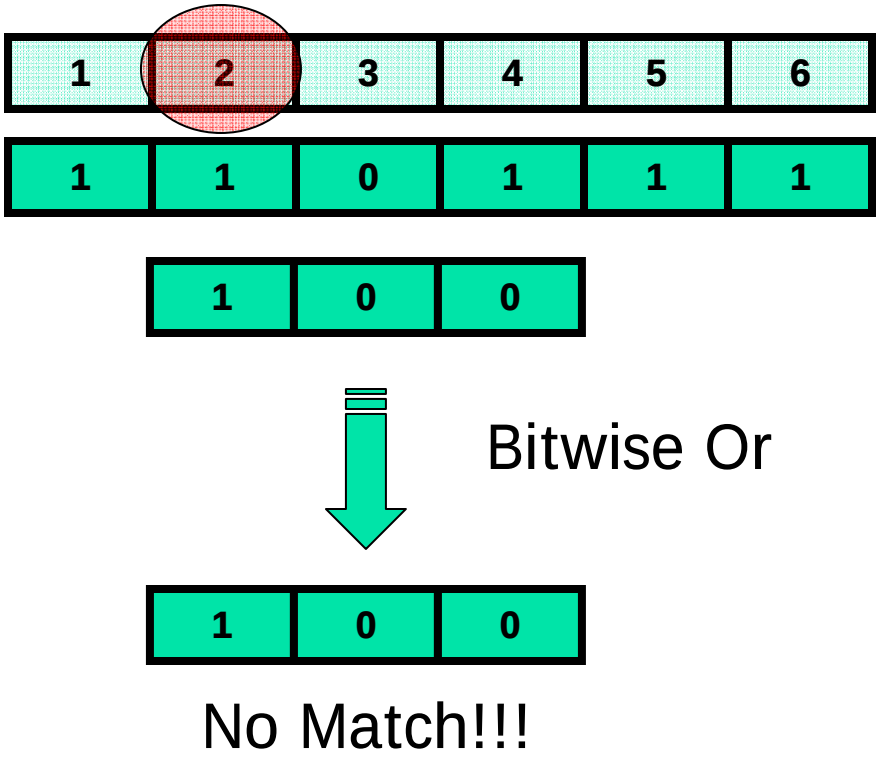


I = {4}

No Match!!!

Algorithm: Stage 3

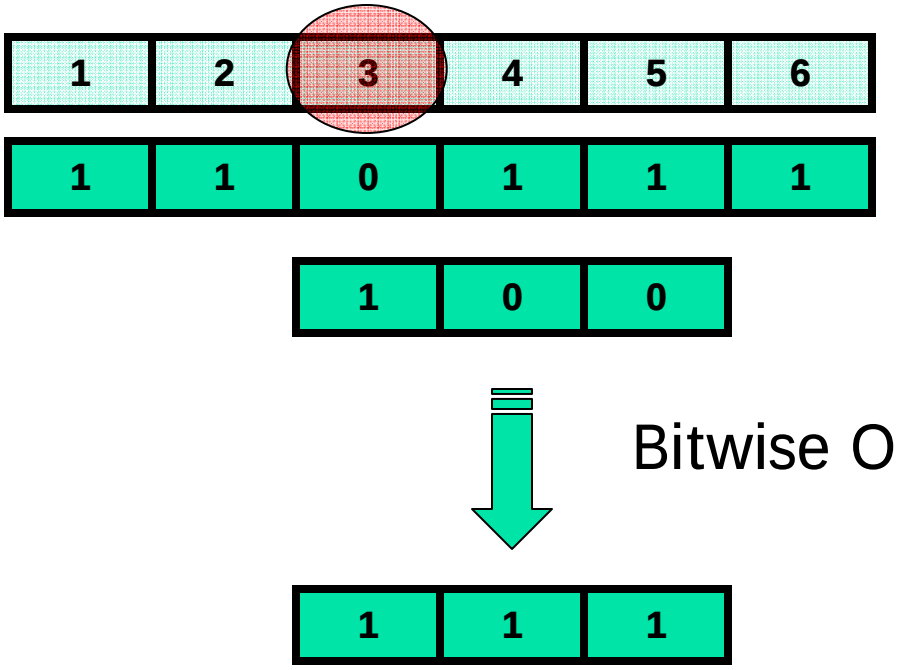
- Find matches of r’ in t’’.



I = {4}

Algorithm: Stage 3

- Find matches of r’ in t’’.

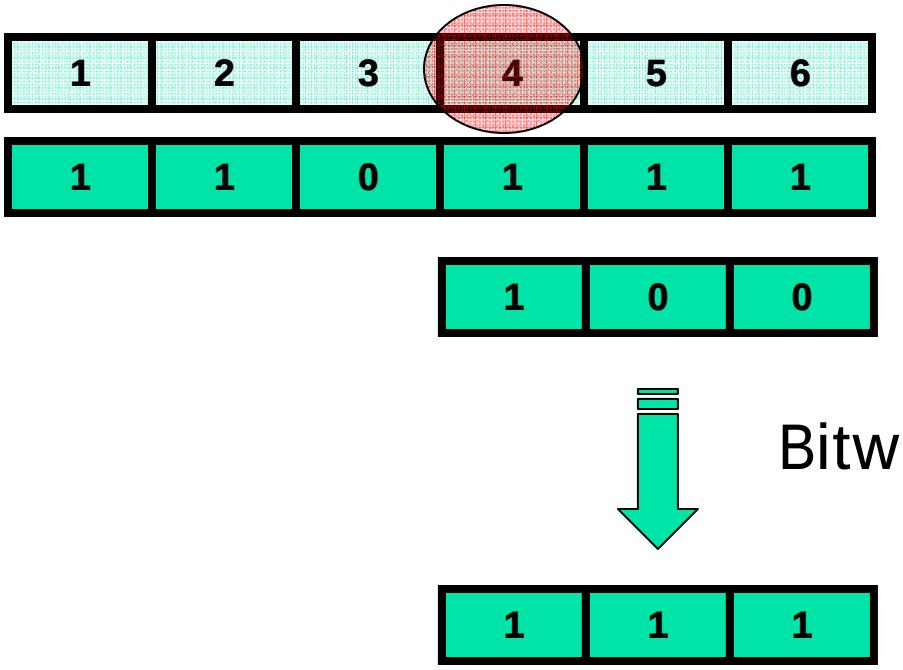


I = {4}

All 1’s!!! And 3 $\notin$ I. So a MATCH!!!

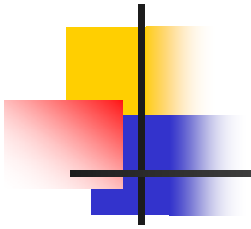
Algorithm: Stage 3

- Find matches of r’ in t’’.



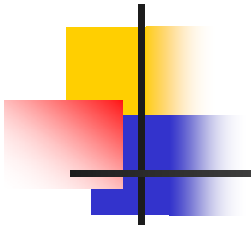
I = {4}

All 1’s!!! But, 4 ∈ I. So NO MATCH!!!



Algortihm

- In this way we can find the matches & then we can easily compute the cover



Conclusion

- In practical cases size of the rhuthm is 10~13. So we have assumed m to be constant.
- It would be interesting, however, to remove t he dependency on m
- It would be interesting to remove the restricti on of one S being Solid
- Applying another restriction on the number of additions in the numeric text may turn out to be useful