Longest Palindromic Substring - Manacher's Algorithm: - $O(|S|)$.

Input: String S $S[1..n]$ Output: Length of a longest substring of S that is a palindrome.

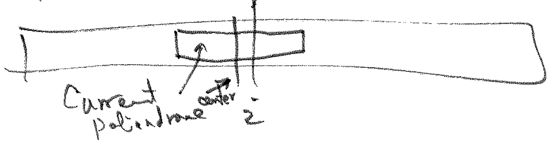
- This problem can be solved with suffix trees, but solution is complicated.

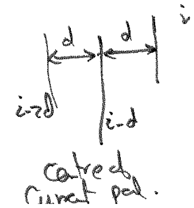
$|S| = K$

For positions $i = 1 \ldots K$, $C[2i]$ = Length of longest odd-length palindrome centred at $S[i]$

For gaps $0 \ldots K$: $C[2 \cdot i+1]$ = Length of longest even-length substring centred at gap i.
Gap i is between $S[i]$ and $S[i+1]$

- $C[1 .. 2K+1]$ is calculated by scanning S from left to right.
- We keep track of the most recent palindrome seen that stretches to i or beyond.





- If longest palindrome centred at $i-2d$ is contained within longest palindrome centred at $i-d$, then we can copy their value to $C[i]$ without any comparisons.

- If l.p. centred at $i-2d$ crosses the boundary of l.p. centred at $i-d$, then palindrome centred at i will extend beyond the right end of palindrome centred at $i-d$.

- In all cases, comparisons of elements starts at the right boundary of the current palindrome.

- Read rest of the story in blogs.

Pop: Back to D.P.

DP in strings:

1. shuffle: Given $A = cat$ $B = ball$, can the string $C = cbatall$ be generated by shuffling A and B?

characters of A must appear in same order
" " B " " "

but A and B can be interleaved.

- Greedy algorithm does not work.

DAC $\longrightarrow$ DP approach.

$A[1..m]$ $B = [1..n]$ $C = [1..m+n]$

Define $f(i,j) = \begin{cases} \text{true if } A[1..i] \text{ and } B[1..j] \text{ can be shuffled into } C[1..i+j]; \\ \text{false otherwise.} \end{cases}$

Recursion for f:

Base: $\begin{cases} f(0,0) = \text{true} \\ f(i,0) = \text{true if } A[1..i] = C[1..i] \\ f(0,j) = \text{true if } B[1..j] = C[1..j] \end{cases}$

$f(i,j) =$ step: if $A[i] \neq B[j]$ then $\begin{cases} = f(i-1,j) \text{ if } A[i] = C[i+j] \\ f(i,j-1) \text{ if } B[j] = C[i+j] \\ \text{false if } A[i] \neq C[i+j] \text{ and } B[j] \neq C[i+j] \end{cases}$

if $A[i] = B[j]$ then $\begin{cases} f(i-1,j) \text{ or } f(i,j-1) & \text{if } A[i] = B[j] = C[i+j] \\ \text{false if } A[i] \neq C[i+j] \end{cases}$

Recursive algorithm's RT is $O(2^{m+n})$ in the worst case.

DP: Solve problems in increasing order of i, j and store $f(i,j)$ in $F[i,j]$.

$F[0,0] \leftarrow$ true for $i \leftarrow 1$ to m do $F[i,0] \leftarrow$ false
for $i \leftarrow 1$ to m do
  if $A[i] = C[i]$ then $F[i,0] \leftarrow$ true
  else break

```
        for j ← 1 to n do  F[0,j] ← false
//Base  for j ← 1 to n do
            if B[j] = C[j] then
                F[0,j] ← true
            else break.

//step  for i ← 1 to m do
          for j ← 1 to n do
            if A[i] ≠ B[j] then
               if A[i] = C[i+j] then
                  F[i,j] ← F[i-1,j]
               else if B[j] = C[i+j] then
                  F[i,j] ← F[i,j-1]
               else F[i,j] ← false

            else  // A[i] = B[j]
               if A[i] = C[i+j] then
                  F[i,j] ← F[i-1,j] or F[i,j-1]
               else
                  F[i,j] ← false.

        Return F   //Answer is in F[m,n].
```

$\boxed{RT = O(m \cdot n)}$

Other string problems solved by DP:

- Break s into least number of palindromes

- Given a string S and a dictionary of words, break S into least number of words from dictionary.

  cat champion

  Not best:

  $f(i,j) = k$, $S[i..j]$ can be broken into k words, smallest k.

  - $O(n^3)$

  Better: $f(i) = k$ if $s[1..i]$ can be broken into k words (min k).

  - $O(n^2)$?