Implement w/ 2xn array C:
C[1][...n]: best time through S1,...,n
C[2][...n]: " " " S2,...,n

Make Car(a1,a2,t1,t2,n). // a1,a2: lists of station times
// t1,t2: " " transfer
// Base Cases
// n: number of stations.
C[1][0] = a1, // time through S1,
C[2][0] = a2, // " " S2,

4) How to fill matrix

for (j = 2 to n)
    // times through S1,j
    from1to1 = C[1][j-1] + a1j; // coming from line 1
    from2to1 = C[2][j-1] + t1j-1 + a1j; // transferring
    // from line 2

    if (from1to1 < from2to1)
        C[1][j] = from1to1
        prevline[1][j] = 1 // best prev station on line 1
    else
        C[1][j] = from2to1
        prevline[1][j] = 2 // best prev station on line 2

    // times through S2,j
    from1to2 = C[1][j-1] + t1j-1 + a2j; // transferring
    // from line 1
    from2to2 = C[2][j-1] + a2j; // from line 2
if (from 2 to 2 < from 1 to 2)

\[ C_{2ij} = \text{from 2 to 2} \]
\[ \text{prevline}[2][j] = 2 \quad // \text{best prev station on line 2} \]

else

\[ C_{2ij} = \text{from 1 to 2} \]
\[ \text{prevline}[2][j] = 1 \quad // \text{best prev station on line 1} \]

end for

(5) How to find final answer:

// Final min cost?
if \( C_{1i} < C_{2i} \)

\[ c^* = C_{1i} \]
\[ \text{lastline} = 1 \quad // \text{last station on line 1} \]

else

\[ c^* = C_{2i} \]
\[ \text{lastline} = 2 \quad // \text{last station on line 2} \]

end if
Try for example:

(from line 1, from line 2)

\[ \min \]

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 7 & (16, 19) & (19, 17) & (21, 25) & (29, 32) \\
2 & 8 & (14, 13) & (25, 19) & (22, 23) & (29, ?)
\end{array}
\]

Look at previous entries of table to fill next entry.

\[ C^* = \min (29, ?7) = 27 \]

Now, how to find the sequence of lines to visit?

* To Do:* Update code to do this.

First, last line to visit?

- line that yields \( C^* \)

<add to code>

Now, \((b)\) find sequence of lines? Can see in \( C \) array
How to keep track of this while filling $c$?

Another array: prevline

prevline $[j,j]$ $(j=2,n)$ previous best line before $S_{nj}$

prevline $[2][j]$ $S_{nj}$

(add to code) look at $C$ to find line previous to station 5.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>-1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

last line = 2

Now trace prevline starting from lastline to -1.

lastline = 2 $\Rightarrow$ 2 $\leftarrow$ 2 $\leftarrow$ 1 $\leftarrow$ 2 $\leftarrow$ 2 $\leftarrow$ -1

(station 5) lastline


:. best line at station 4 was 2.
(7) Run Time:

Run Time: n Stations?

to fill C[J][J]: for loop from 1 to n : O(n)
" " prevLine[J][J] : O(n)
" " Scan : O(n)

Total : O(n)
Longest Common Subsequence

Compare DNA of 2 organisms

$S_1 = A C C G G T C A C T \cdots$
$S_2 = T T C C G A T C T G \cdots$

Many ways to measure similarity:
- # identical characters in the same index
- # substrings in common
- longest common subsequence

First, what is a subsequence?

Informally: string $Z$ is a subsequence of string $X$ if $Z$ is made up of characters from $X$ in the same order they appear in $X$.

Formally: For a string $X = x_1, x_2, \ldots, x_m$, $Z$ is a length-$k$ subsequence of $X$ if there exists "an increasing sequence of indices" $i_1 < i_2 < \ldots < i_k$ s.t.

$Z = x_{i_1} x_{i_2} \cdots x_{i_k}$

Subsequences of $S_1$? A GCC CTG GA

✓ ✓ X ✓
Longest common subsequence (LCS) of 2 strings \(X, Y\) is longest subsequence that both \(X, Y\) contain.

LCS problem - Given 2 strings \(X = x_1, x_2, \ldots, x_m\) and \(Y = y_1, y_2, \ldots, y_n\), find the LCS of \(X\) and \(Y\). Denote \(\text{LCS}(X, Y)\).

"We will start with finding the length of the LCS. Denote \(|\text{LCS}(X, Y)|\)"

**Brute Force?**

1. Enumerate all subsequences of \(X\) and all subsequences of \(Y\).
   - \(2^m\) subsequences of \(X\) for each character: 2 choices - "\(\)" and either include it in the subsequence or don't.
2. Hash one list into table. Hash the other while storing collisions. (Collisions indicate common subsequences. \(\Rightarrow 0(2^m + 2^n)\))
3. Scan collisions to find the one of max. length. Total: \(0(2^m + 2^n)\).

**Greedy?** Unclear what a greedy approach would be.

**Dynamic Programming**

Does problem exhibit optimal substructure property? Yes \(\checkmark\).

\(\Rightarrow\) LCS of \(X, Y\) depends on LCS of substrings of \(X, Y\).
Which substrings should we consider?

Suppose: \( S_1 = \overline{A} \overline{C} \overline{C} \overline{C} \ldots \overline{C} \overline{C} \) \( \overset{?}{\Rightarrow} \) \( \text{LCS}(x, y) = A \).
\[ S_2 = \overline{B} \overline{B} \overline{B} \overline{B} \ldots \overline{B} \overline{B} \overline{A} \]

- or - (vice versa)

\[ S_1 = \overline{B} \overline{B} \overline{B} \ldots \overline{B} \overline{B} \overline{A} \]
\[ S_2 = \overline{A} \overline{C} \overline{C} \overline{C} \ldots \overline{C} \overline{C} \]

"Need to consider all of \( S_1 \) and all of \( S_2 \) starting from the beginning"

prefix - any leading (starting from beginning)
contiguous (adjacent) substring

ex: \( X = \overline{B} \overline{A} \overline{C} \overline{D} \overline{B} \) prefixes: \( B, BA, BAC, BACD, \ldots \)
\[ Y = \overline{B} \overline{D} \overline{E} \overline{B} \ldots \overline{B}, BD, BDE, \ldots \]

Denote:
\[ \overline{x}_i = \text{length-}i \text{ prefix of } X \]
\[ \overline{y}_i = \text{" } \overline{y} \text{" } \]

ex: \( \overline{x}_2 = \overline{B} \overline{A} \), \( \overline{y}_3 = \overline{B} \overline{D} \overline{E} \), \( \overline{x}_1 = \overline{B} \), \( \overline{y}_1 = \overline{B} \), etc...
Use solutions for shorter prefixes of X,Y to find solutions for longer prefixes.

But how?

Initial idea: compare same-length prefixes of X,Y

ex: X = BACDB
    Y = BDEB
    ↑↑↑↑↑

LCS so far: B B B B → but LCS = BDB not B!

Consider this example:

S₁ = C D D D
S₂ = D D D C

to find LCS(S₁, S₂) = D D D...
must compare every prefix of S₁ with every prefix of S₂

S₁ C C C ... C ... C D C D ... C D D D
S₂ D D D D ... D D D D C ... D D D D C
CS so far - - C D D D

X B B B B ... B ... BA ... BACD ... BACDB
Y B BD BDE ... BDEB B BD ... BDEB
Now...

\[ X = \text{BACDB} \]
\[ Y = \text{BDDEB} \]

Suppose you knew \( |\text{LCS}| \) of these prefixes

\[ |\text{LCS}(\overline{X}_4, \overline{Y}_3)| = 2 \quad (BD) \]

How to use next chars in \( X, Y \) to find \( \text{LCS}(\overline{X}_5, \overline{Y}_4) \)?

2 cases based on next chars \( X_5, Y_4 \)

**Case 1: If** \( x_i = y_j \)

\[ X = \overline{X}_{i-1} B \]
\[ Y = \overline{Y}_{j-1} B \]

Can match \( x_i + y_j \) to get a longer LCS.

\[ |\text{LCS}(\overline{X}_i, \overline{Y}_j)| = |\text{LCS}(\overline{X}_{i-1}, \overline{Y}_{j-1})| + 1 \]

**Question:** What if \( x_i \) also matches with a previous char in \( Y \)?

**Ex:** \( X = \text{BACDN} \) Better to match \( x_i \) (N) with previous \( N \) in \( Y \) only if

\[ Y = \text{BNEN} \]

Would match \( x_i \) (N) with previous char in \( Y \) only if there is a future character in \( X \) that can be matched with another character in \( Y \).