Recap: Longest Common Subsequence (LCS)

① Subproblems: $lcs[i,j]$ is LCS of $x_{[1:i]}$ and $y_{[1:j]}$

② Guess: If $x_i \neq y_j$, guess which one is NOT in LCS?
(can be both)

③ Recurrence:

$$lcs[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ lcs[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(lcs[i-1,j], lcs[i,j-1]) & \text{if } i,j>0 \text{ and } x_i \neq y_j \end{cases}$$

④ Memoization / Bottom-up: $\Rightarrow$ $O(m \cdot n)$ runtime

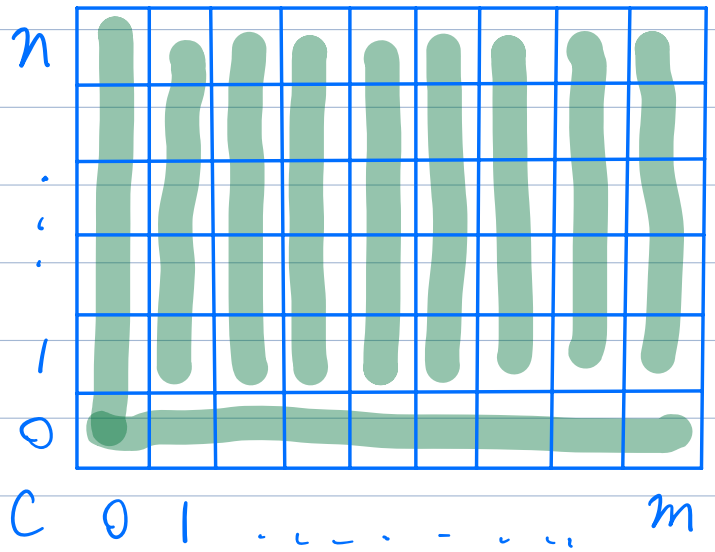
⑤ ☺

1. LCS (Cont'd)

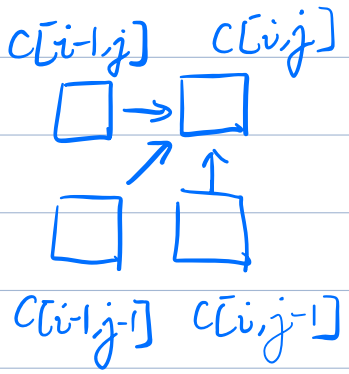
Pseudo-Code for LCS (for giving the actual subsequence)

```
LCS(x[1,...,m], y[1,...,n]):
1.   For i=0 to m:
2.       c[i,0]=0; s[i,0]=""
3.   For j=1 to n:
4.       c[0,j]=0; s[0,j]=""
5.   For i=1 to m:
6.       For j=1 to n:
7.           If x[i]==y[j]:
8.               c[i,j]=c[i-1,j-1]+1
9.               s[i,j]=s[i-1,j-1]+x[i]
10.          Else If c[i-1,j]≥c[i,j-1]:
11.              c[i,j]=c[i-1,j]
12.              s[i,j]=s[i-1,j]
13.          Else:
14.              c[i,j]=c[i,j-1]
15.              s[i,j]=s[i,j-1]
16.  Return c[m,n]
```

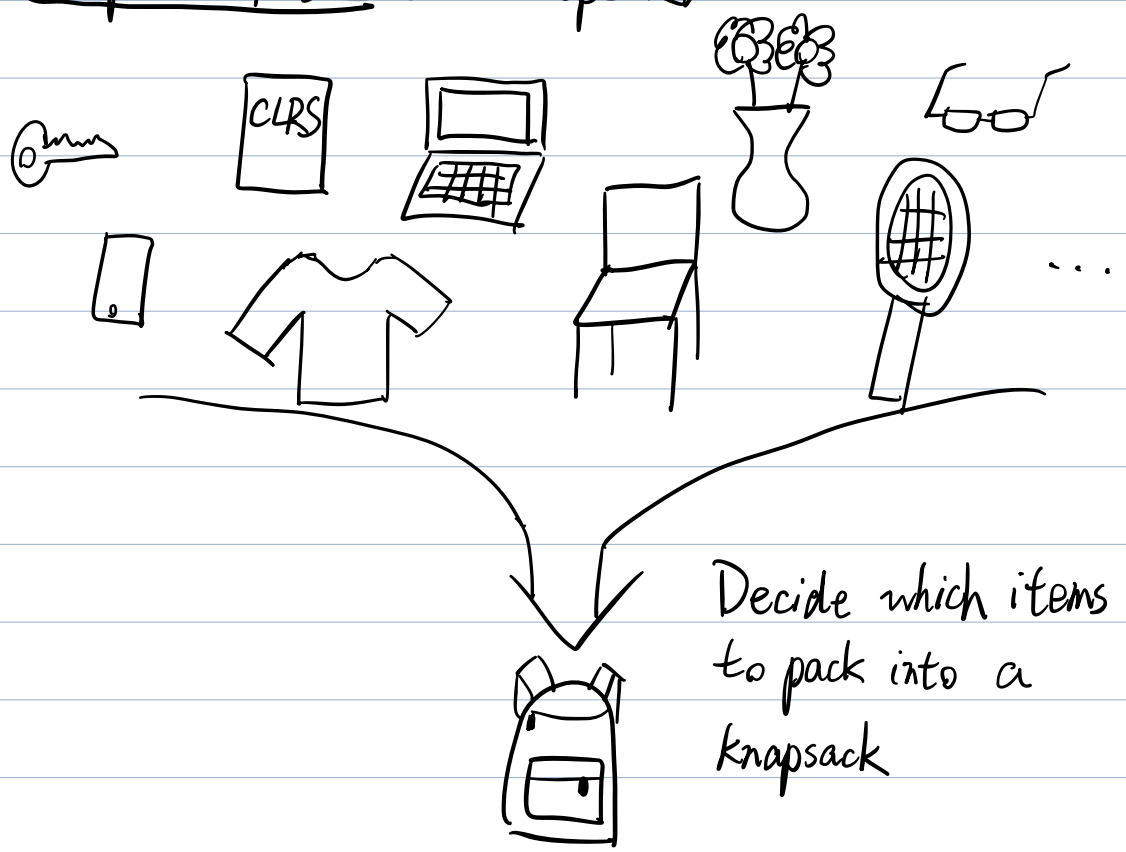
How do we fill up c to get c[m,n]?



← Solution @ c[m,n]



2. Knapsack Problem (0-1 Knapsack)



Decide which items to pack into a knapsack

- List of n items each with size S_i (integer), and value/utility v_i.
- Knapsack of size S
- What is max total value of items w/ total size $\leq S$?

e.g.

Item	Size	Value
1	1	1
2	2	6
3	5	18
4	6	22
5	7	28

$S = 11$

Greedy? pick the max $\frac{\text{value}}{\text{size}}$ item.

$\Rightarrow \{5, 2, 1\}$ total value $= 35$

But if we pick $\{3, 4\}$, we get value 40!

Greedy doesn't work here...

DP to the rescue!

① Subproblems: items $1, \ldots, i$ & remaining space $X \leq S$

② Guess: do we bring item i or not?

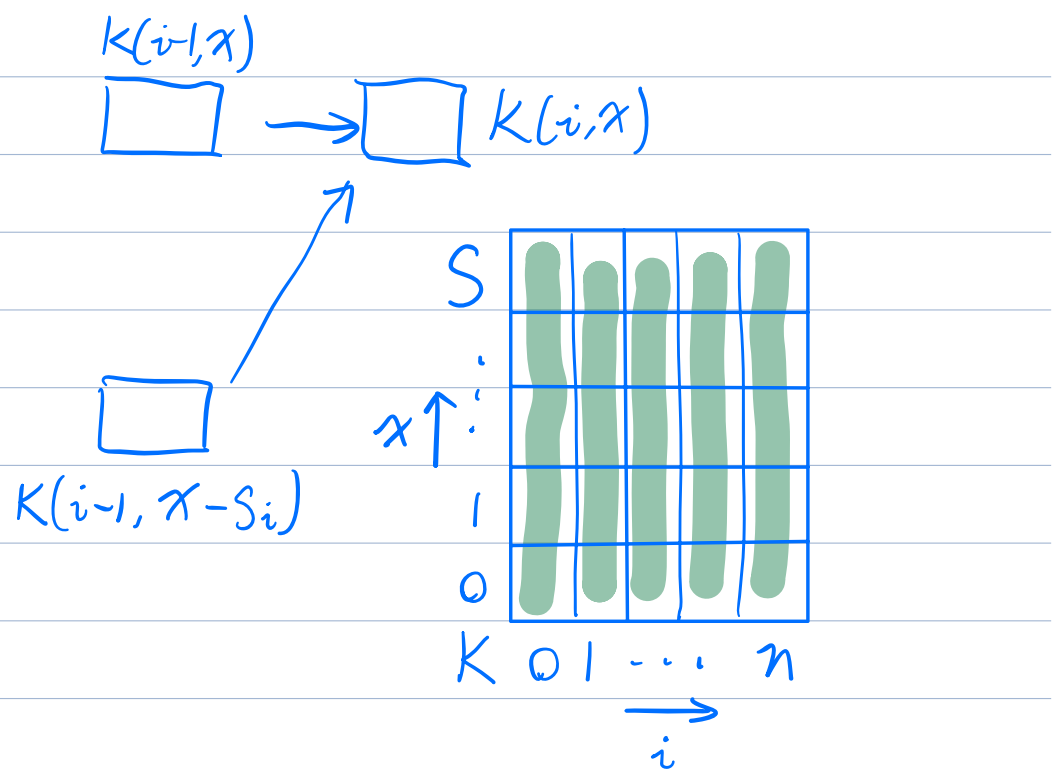
③ Recurrence: $K(i,x) = \max(K(i-1,x), K(i-1, x-s_i)+v_i)$

Base Cases? No items left ($i=0$)

Slight issue here: $x-s_i < 0$ will go out of bounds

Fix:

$$K(i,x) = \begin{cases} 0 & \text{If } i=0 \\ K(i-1,x) & \text{If } i>0 \text{ and } x<s_i \\ \max(K(i-1,x), K(i-1,x-s_i)+v_i) & \text{Otherwise} \end{cases}$$



④ Memoization / Bottom-up:

subproblems: at most $(n+1)\cdot(S+1) = O(n \cdot S)$

time / subproblem: $O(1)$

$\Rightarrow$ Runtime is $O(n \cdot S)$

⑤ 

(not really polynomial, because input S only needs $\log S$ bits)
"pseudopolynomial"

Extensions:

captured by 0-1 knapsack:

- Bounded Knapsack Problem:
  Each item can have multiple copies
  e.g. 2 phones, 4 keys, 5 CLRS textbooks
- Unbounded Knapsack Problem:
  Unlimited # of copies
  e.g. ∞ phones, keys, CLRS textbooks

- Fractional Knapsack Problem:
  Can take a fraction of an item w/ proportional value
  e.g. candies, bread, pizza, cheese, ham
  Greedy actually works here!

Example of MCQ DP Problem:

10. The Levenshtein distance is a metric for measuring the difference between two strings. It is defined as the minimum number of single-character edits (insertions, deletions or substitutions) required to change one string into the other. For instance, the Levenshtein distance between "midterm" and "mayhems" is 5 by making the following edits:

midterm → madterm → mayterm → mayherm → mayhem → mayhems

A DP algorithm for finding the Levenshtein distance between two strings $X = x_1x_2 \dots x_m$ and $Y = y_1y_2 \dots y_n$ uses a 2-dimensional array D, with m rows and n columns. The entry $D[i, j]$ for $0 \le i \le m, 0 \le j \le n$ is the Levenshtein distance between substrings $x_1x_2 \dots x_i$ and $y_1y_2 \dots y_j$. We compute $D[i, j]$ as

$$D[i,j] = \begin{cases} j & \text{if } i = 0 \\ i & \text{if } j = 0 \\ \underline{\quad ① \quad} & \text{if } i, j \ge 1 \text{ and } x_i = y_j \\ \underline{\quad ② \quad} & \text{if } i, j \ge 1 \text{ and } x_i \ne y_j \end{cases}$$

What are the missing terms in the blanks? ..... (A)

(A) ① : $D[i-1, j-1]$ ② : $1 + \min(D[i-1, j], D[i, j-1], D[i-1, j-1])$
(B) ① : $D[i-1, j-1]$ ② : $1 + \min(D[i-1, j], D[i, j-1])$
(C) ① : $D[i-1, j-1] + 1$ ② : $\min(D[i-1, j], D[i, j-1])$
(D) ① : $D[i-1, j-1] + 1$ ② : $\min(D[i-1, j], D[i, j-1], D[i-1, j-1])$

We can easily find the answer for such MCQs!
To check if ① should have "+1" or not, we can try examples X="ac" and Y="bc". That rules out C and D. Then, consider X="abc" and y="abd", that rules out B. So correct answer is A.

Ex: Why is A the correct optimal substructure?