06-19

Friday, June 19, 2020 6:25 PM

Dynamic Programming (DP)

① optimization / counting / feasibility
max min longest

② design recursion (think recursively)

③ run time (is bad very) if impl using recursion naively
same computation

④ - bottom up
- top down w/ memoization

Longest Common Subsequence (LCS)

- input A = [3, 5, 1, 2, 4]
B = [10, 1, 3, 5, 4]

subseq [3, 4] subseq of A

- optimization length (longest)
- (size) of problem / instance

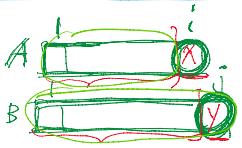
A = [ ]
B = [ ]
(i, j)
len of A len of B

$f(i, j)$:= length of LCS given A[1:i] and B[1:j]
what we want

$f(i-1, j)$ $f(i, j-1)$ $f(i-1, j-1)$
$f(i-3, j)$ $f(i, j-2)$ $f(i-5, j-7)$

"last" of something

common subseq



A[1, 3]
B[1, 5, 3]

case 1: A[i] = B[j]

$f(i, j) = 1 + f(i-1, j-1)$

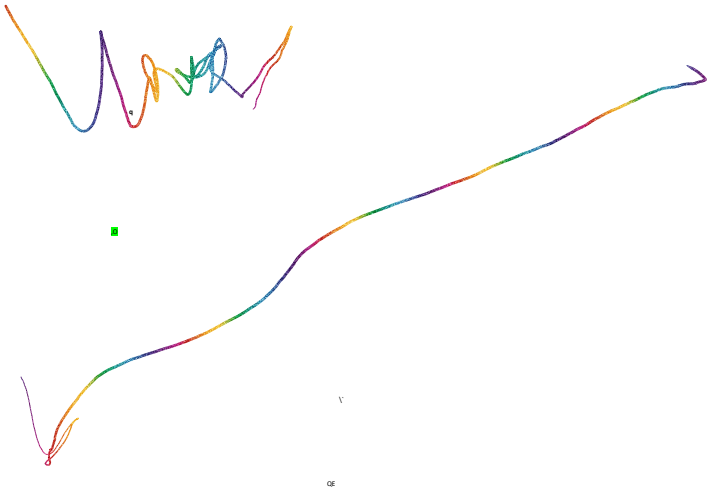
case 2: A[i] ≠ B[j]

$f(i, j) = \max(f(i-1, j), f(i, j-1))$

Base case: i = 0 or j = 0

$f(0, j) = f(i, 0) = 0$

A: [1, 2, 3, 4], B: [1, 2, 3]



A [1,2,3,4] , B [1,2,3]

3 / 2

A [1,2,3], B [1,2,3] | A [2,3,4], B [1,2]

naive recursive impl.

```
LCS(A[1..i], B[1..j]) {
    if (i==0 || j==0) return 0;
    if (A[i] == B[j])
        return 1 + LCS(A[1..i-1], B[1..j-1])
    return max(LCS(A[1..i-1], B[1..j]),
               LCS(A[1..i], B[1..j-1]))
}
```

dp

```
LCS(A[1..i], B[1..j], dp[,]) {
    if(i==0 || j==0) return 0;
    if(dp[i,j] != -1) return dp[i,j];
    if (A[i] == B[j])
        dp[i,j] = 1+LCS(A[1..i-1], B[1..j-1], dp[,])
    else
        dp[i,j] = max(LCS(A[1..i-1], B[1..j], dp[,])
                      LCS(A[1..i], B[1..j-1], dp[,]))
    return dp[i,j];
}
```

T(n, m) m = n

T(n-1, m) T(n, m-1)

T(n-2, m) T(n-1, m-1) T(n-1, m-1) T(n, m-2)

Bottom up

dp: 0 1 2 ... m; 0 1 2 ... n

f(n, m)

f(i, j)

f(i-1, j-1)
f(i-1, j)
f(i, j-1)

```
LCS(A[1...n], B[1...m]){
    for(i=0...n)
        dp[i,0] = 0;
    for(j=0...m)
        dp[0,j] = 0;
    for(i=1....n){
        for(j=1....m){
            if(A[i] == B[j])
                dp[i,j] = 1+dp[i-1,j-1];
            else
                dp[i,j] = max(dp[i-1,j], dp[i,j-1])
        }
    }
    return dp[n,m];
}
```

dp[i,j] = f(i,j)

O(nm)

subseq

length of subseq

A=[1, 3, 5, 7, 9]

B=[3, 7, 1, 2]

i=1, j=4

dp:

	0	1	2	3
0	0	0	0	0
1	0	0	0	1
2	0			
3	0			
4	0			
5	0			

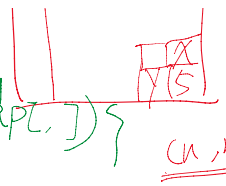
LCS of A & B

length of LCS

dp

X
Y 5

length of




```
find_lcs(A[1..n], B[1..m], dp[,])
    i=n, j=m;
    l=[ ];
    while (i<=0 || j<=0){
        if (A[i]==B[j]){
            l.append(A[i]);
            i = i-1;
            j = j-1;
        } else if (dp[i-1,j] < dp[i,j-1])
            j=j-1;
        } else {
            i=i-1;
        }
    }
    return l.reverse();
}
```

(n,m)

A[n] ≠ B[m]

(n-1, m-1)

O(n+m)

A=[1, 2, 3]

B=[1, 3]

dp

	0	1	2
0	0	0	0
1	0	1	1
2	0	1	1
3	0	1	2

i=0 j=0

l=[3, 1]

[1, 3]

0-1 knapsack

- n items
- each item i value[i] weight[i]
  $ lb
- capacity of knapsack is C
- what is max $

naive enumerate all possible subsets

m

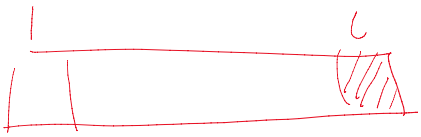
naive enumerate all possible subsets

2^n

$f(i, j)$:= max $ given first i items and knapsack capacity is j

smaller problems

item



less items

smaller knapsack

take ith item

$f(i-1, j - \text{weight}[i]) + \text{value}[i]$

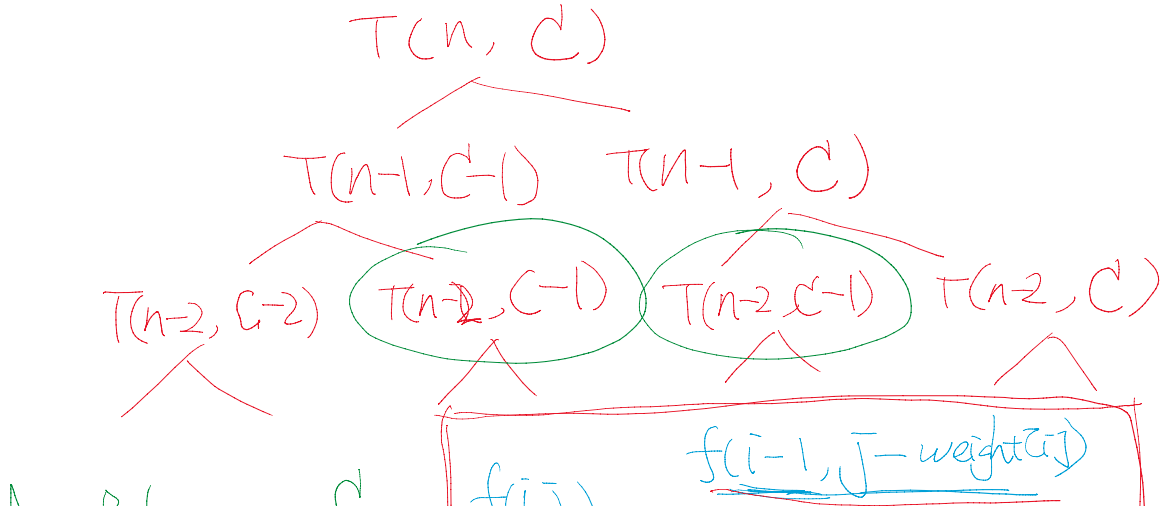
if v

leave ith item alone

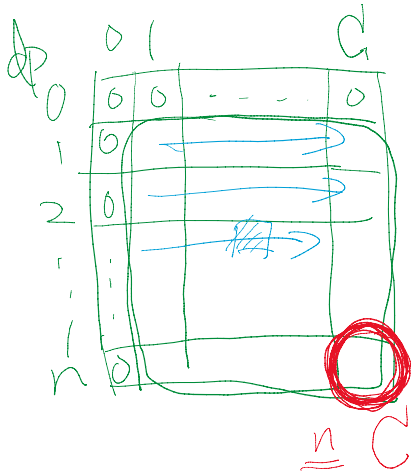
$f(i-1, j)$

$f(i, j) = \max(f(i-1, j - \text{weight}[i]) + \text{value}[i], f(i-1, j))$

base case $f(0, j) = f(i, 0) = 0$



eightA[i][j]



f(i,j)

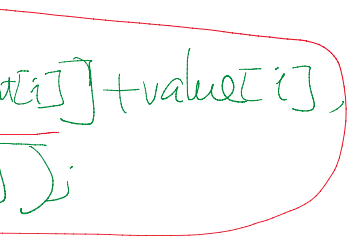
f(i-1, j-weight[i])

f(i-1, j)

```
for (i = 1 ... n) {
  for (j = 1 ... C) {
    if (weight[i] <= j)
      dp[i,j] = max(dp[i-1, j-weigh
                    dp[i-1, j]
    else
      dp[i,j] = dp[i-1, j];
  }
}
return dp[n, C];
```

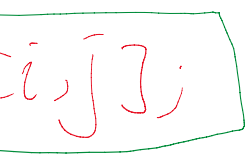
O(nC)

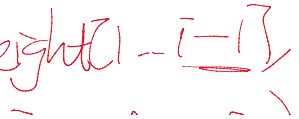
```
knapsack( ... weight[1...i], value[
          j , dp[ ] ) {
  if (i==0 || j==0) return 0
  if (dp[i,j] != -1) return dp[
  dp[i,j] = knapsack( ... , we
```



map (i,j) => $

i





dp[i,j] = knapsack(... , we
value[1..i-1],

if (weight[i] ≤ j) {

dp[i,j] = max (dp[i,j] ,
knapsack (
vale

}

return dp[i,j];

}

bottom up VS

run time

easy to see

space complexity

O(nC)

f(i,j)

f(i-1, j-wi

f(i-1, j)

eight[1...i-1],
], dp[ ])

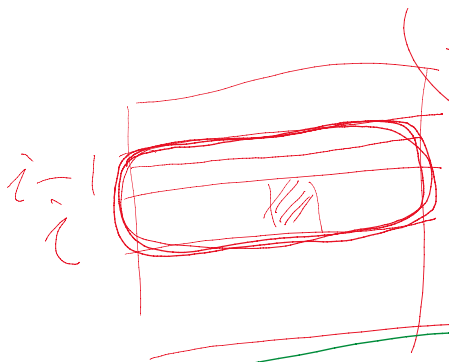
..., weight[1...i-1],
ue[1...i], j-weight[i], dp[ ]) +value[i]))

top down

harder to see

O(nC)

i])



dpi

sequence small ⇒ larger

personal preference

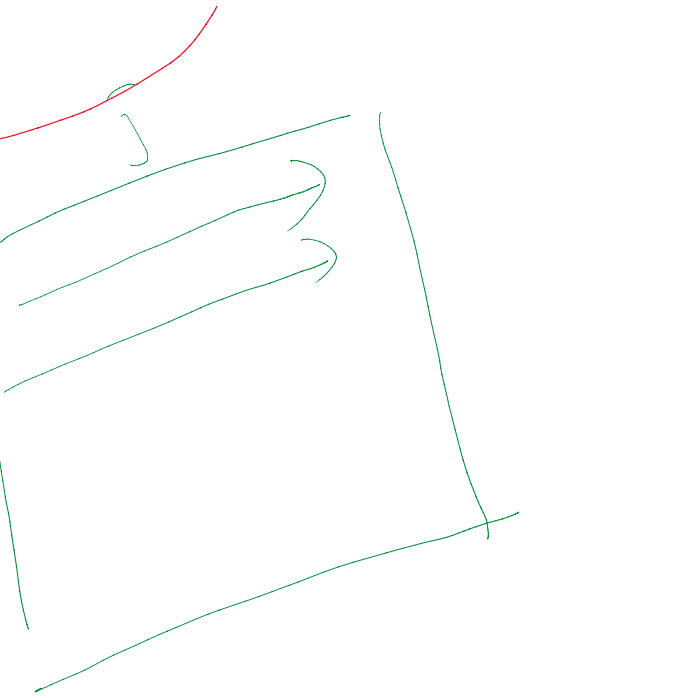
---

n questions in the exa

sore[i] time[i]

I dont give partial

T min max



um.

credit

sub seq

A = [1, 2, 3

[1, 3]

[3, 17

3, 4]

✓

×