DP Season - Classic Problems and Patterns ✓

Wednesday, 20 July 2022 9:06 PM

Class
Daily Dose] ← 19 to 26
#2 Warchest →
Contest & Doubt Classes → 27 to 31
#3 ← 28, 30

[21-26 — Array
- 21-24: Kadane, LCS/LIS, Knapsack (2 classes)
- 25 - Matrix Expo + Matrix Chain
- 26 - Remaining DP

- 12 patterns — 90-95 % — 2 patterns
- "To include or not to include"
- "Try all possible action/values"

Process

☆ pattern recognition

state
↓ pattern
recurrence | transition
↓
base case
+ [Memoization]

Top Down — recursive

Bottoms up

state
↓ table
base case
↓
transition (fill out table)

table

Classical Problem | Variants

① Coin Change

coins[] = [1, 2, 3], n, sum = 4

infinite supply

no. of ways to get sum out of these coins?

4 = {1+ 1+ 1+ 1 } o/p: 4

4 1's
2 2's
3 1's

2+2
3+1
2+1+1

~~[1,1,2]~~

Clipboard

Top Down

states (variables)

rem_sum → what sum is remaining
n → what are the universe of coins rem- that I can choose from

~~arr~~ → [1, 2, 3] → fixed → state

[1,2]

n, rem

n-1, rem

n-2, rem

[1, 2, 3] } infinite

n = 3, rem_sum = 4

do not include

3 Include

n=2, rem_sum = 4

n=3, rem_sum = 1

* getCount(n, rem_sum) =
~~getCount(n, rem_sum - coins[n-1])~~
+ getCount(n-1, rem_sum)

n=3, rem=4

n=2, rem=4

n=3, rem=1

base case return.

n → 0 [universe] return 0

sum → 0 return 1

sum is never

-ve

sum==0

{ }

* memoization

Clipboard

sum>0

memset * dp[sum][n] → -1

10^5 10^5

```
int    numWays (int coins[], int n, int rem_sum)
{
    if (dp[sum][n] != -1)
        return dp[sum][n];
  → base case
    if (sum == 0)   return 1;     → dp[sum][n] = 1;
    if (n == 0)     return 0;     → dp[sum][n] = 0;
  → do not include
    int res = numWays (coins, n-1, rem_sum);
  → include
    if (coins[n-1] <= rem_sum)
        res += numWays (coins, n, rem_sum - coins[n-1]);
    return res;    → dp[sum][n] = res
}
```

[1 2 3] n=3, rem_sum=4

n-1 → 2, rem_sum → 4

take care of rem_sum ⇒ -ve

---

dp [sum+1] [n+1]

dp [i] [j] = no. of ways to get sum i with j

[1,2,3]

n → rem. coins → coins[0] ... coins[j-1]

sum	0	1	2	3
0	1	1	1	1
1	0	1	1	1
2	0	1	2	2
3	0	1	2	3
4	0	1	3	? 4

sum=1
coins[1]=2

coins[i]>sum
X

2-2

sum=1, n=1

sum=1, n=0

sum=0, n=1

dp [i] [j] → dp [i] [j-1]   // do not include
\+
dp [i - coins[j-1]] [j]   // include

Clipboard

```
int numWays(int coins[], int n, int sum)
{
                0....sum   0...n
    int dp [sum+1] [n+1];
    for (int i=0; i<=n; i++)
          dp [0] [i] = 1;                      base
    for (int i=1 ; i<=sum; i++)                case
          dp[i] [0] = 0;

    for (int i=1; i<= sum; i++){
          for (int j= 1; j<=n; j++){
               dp [i] [j]= dp[i] [j-1]; // do not include
               if (coins [j-1] <= i )
include <--       dp[i][j] += dp [i-coins[j-1]]
                                        [j];
          }
    }
    return dp [sum] [n];

}
```

time: $O(sum * n)$

space: $O(sum * n)$

$O(sum) \Rightarrow$ HW

---

② Edit Distance Problem

two strings

I/p: s1 = "CAT" , s2 = "CUT"

minimum no. of edits reqd to convert s1 into s2?

edits

- ① Delete any char in s1
- ② Add a char at any pos in s1
- ③ Replace a char at any pos, in s1 with another char

$A \rightarrow U \Rightarrow 1$

C(A)UT → CUT ⇒ (2)

saturday

sunday

{ [subproblem] 0 … i, 0 … j } LCS, LIS

1) states ✓

2) transitions

i) if (s1 [i-1] == s2[j-1])

saturday / sunday $\xrightarrow{+0}$ saturda / sunda

editDistance (i, j) = editDistance (i-1, j-1)

ii) Try all possible actions / values
⇓
pattern

satur / sun → +1 delete r → satu / sun

satur / sun → +1 replace r with n → satu / su

satur / sun → +0 add n → satur / su

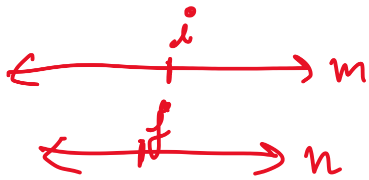
transition: editDistance (i, j)

= min (editDistance (i-1, j) + 1,

editDistance (i, j-1) + 1,

edit Distance (i-1, j-1) + 1)

3) base case



" "

" sun " → length of the non-empty string

base

```
if (i==0)  return j;
if (j==0)  return i;
```

transition

table

		C	A	T
	0	1	2	3
C	1	0	1	2
U	2	1	1	2
T	3	2	2	1

c

c

C

CA

dp[i][j] = edit distance for s1[0... i-1] & s2[0...j-1]

```
int edit (string& s1 , string& s2, int m, int n)
{
    int dp[m+1][n+1];
    for (int i=0; i<=m; i++)
            dp[i][0] = i;
    for (int j=0; j<=n; j++)
          dp[0][j] = j;
    for (int i= 1; i<=m; i++)
      for (int j= 1; j<=n; j++)
      {
          if ( s1[i-1] == s2[j-1])
               dp[i][j] = dp[i-1][j-1];
```

else $dp[i][j] = 1 + \min(dp[i][j-1], dp[i-1][j], dp[i-1][j-1]);$

} }

return $dp[m][n]$;

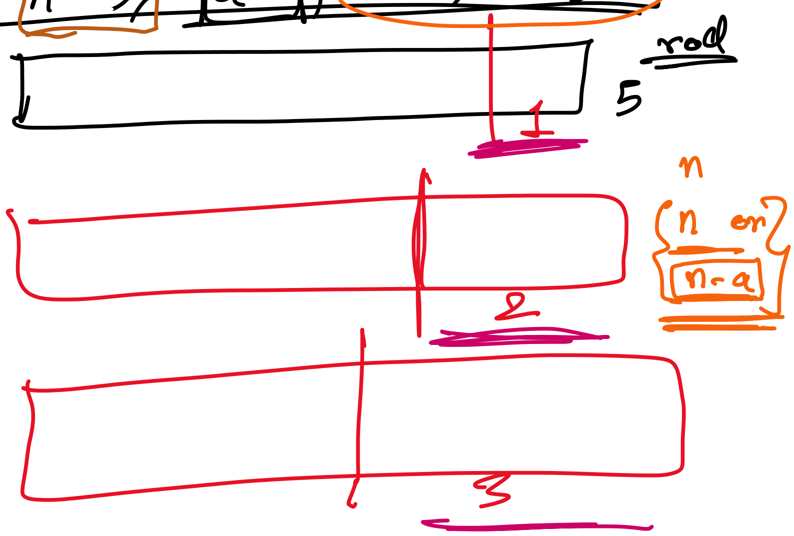
}

time: $m*n$
space: $m*n$

Q3) Maximum Cuts

I/p: $n=5$, $a=1$, $b=2$, $c=3$

Ques: max cuts you can do



3 | 2 ⇒ 2

2 | 2 | 1 ⇒ 3

3 | 1 | 1 ⇒ 3

* 1 | 1 | 1 | 1 | 1 ⇒ 5

op: 5

I/p: $n=3$

$a=2$, $b=4$, $c=2$

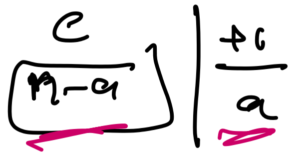
o/p: -1

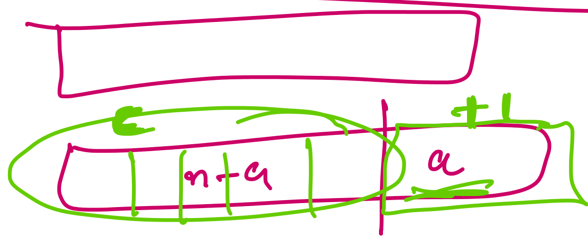
State: n

try all possible

Base case: n==0 → 0, n<0 → return -1

Transition: max (maxCuts(n−a, a, b, c), maxCuts(n−b, a, b, c), maxCuts(n−c, a, b, c)) + 1





Q4 Subset Sum

I/p: [10, 5, 2, 3, 6]

X = 8

Find no. of subsets that sum to X? → reasonable

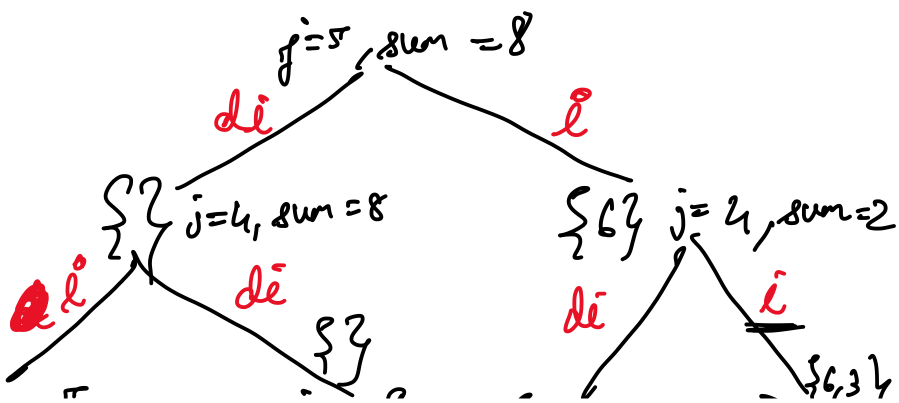
O/p: [5, 3], [2, 6] } 2

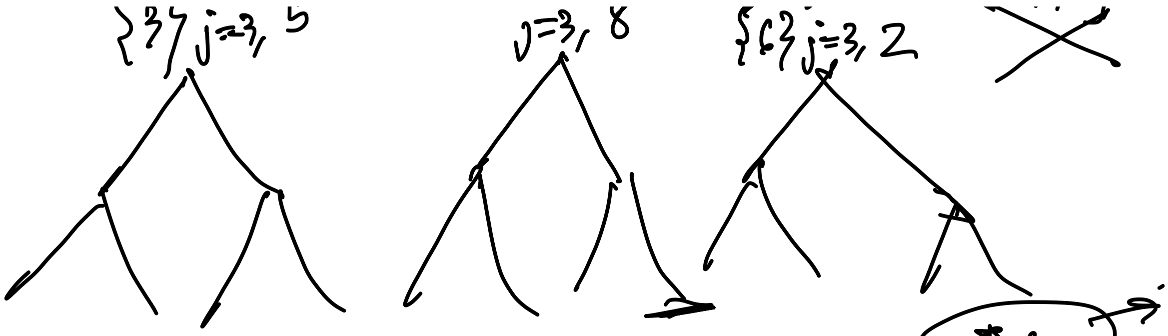
- State → pattern → i or di
- transition
- base case

* rem_sum, rem_subset (j)
* include or not include

10, 5, 2, 3, 6 (j)   sum = 8

HW





$n * sum$

countSubset (arr[], n, sum) =
    countSubsets (arr[], n-1, sum)
  + countSubsets (arr[], n-1, sum - arr[n-1])

n → n-2 → n-3 ....

{ if >= 0 }

n == 0 | sum == 0
sum > 0 | { } return 1
return 0;

→ if (n==0)
    return (sum==0) ? 1 : 0;

Daily Dose
→ ....
→ ....

both methods
↓
{ top down , bottom up }

sum ⇒ -10^3 to 10^3
↓
+ add   [0 ..... $2 \cdot 10^3$]

$\boxed{-10^3}$

$-10^3 - 10^3$ → -10^3
$0 - 2 \cdot 10^3$
↓
$0 \ldots 2 \cdot 10^3$ } indexing

Clipboard