

6

Overlapping Subproblems

Introduction to DP

- ① Tabular Method \rightarrow Bottom Up^o
 - ② Memoization \rightarrow Top to Bottom

~~# Memo~~ 'iotion'

0 = 50396

$$l = 115 \text{ g}$$

Fibonacci

$f(n) \leq 3^n$ (by induction)

$$\left\{ \begin{array}{l} \text{if } n \neq 1 \\ \text{if } n = 1 \end{array} \right\} \text{if } n \neq 1$$

$$\text{reduces } f(n-1) + f(n-2);$$

W. H. T.

Converg

• 20150 44

$$b_i = \text{arg} \min_{b_i} \{ \text{err}_i \} = \{ -1 \}$$

$f(m)$

$$\begin{cases}
 & \text{if } (n \leq 1) \text{ true} \\
 & T = O(N) \\
 & S = O(N) + O(N)
 \end{cases}$$

1979 + 1979 = 1979 Refutez

every - if (dplng := -)

Wetted = water
Wetted = water

~~def~~ i return $dp[n]$;

-eh

return $n + f(n-1) + f(n-2)$;

②

Widgit Diagrams

Q1 A. Tabular form

Widgit diagram: $\text{dp}(n, -1)$
initialization: $\text{dp}(0, -1) = 0$
recurrence: $\text{dp}(i, -1) = \text{dp}(i-1, -1) + \text{dp}(i-2, -1)$
for (int $i = 2$ to n)

T₀₀

$$\text{dp}[0] = 0$$

$$\text{dp}[1] = 1$$

for (int $i = 2$ to n)

initial $T \rightarrow O(N)$

$S \rightarrow O(N)$

$$\text{dp}[i] = \text{dp}[i-1] + \text{dp}[i-2];$$

cout << dp[n] << endl;

Space Optimized

inf. prev2 = 0
inf. prev = 1

$T \rightarrow O(N)$

$S \rightarrow O(1)$

(i) \rightarrow for (int $i = 2$; $i \leq n$; $i++$)

int prev

$$\text{inf. curr} = \text{prev} + \text{prev2};$$

prev2 = prev;

prev = curr;

cout << prev << endl;

③

→ (Greedy won't work here because values are not uniform)

0/1 Knapsack

Max: $3 + 4 + 5 = 12$ } \rightarrow Maximize

By trying all combinations, with recursion

Rules: (E, D) (E, \bar{D})

~~constraint~~ \Rightarrow Express in every term in index
(~~constraint~~) \quad (1,0) \quad (0,1) \quad (0,0) (ind, value)
→ Explore all possibilities; (pick, !pick)

→ Mark all possibilities \Rightarrow AT the
optimum $= (\text{opt})$ + op $= \text{opt}$

$$\textcircled{5} \quad \{ \omega = G(\omega) \} \cap \omega = \frac{G(\omega)}{\omega + \text{Im } \omega}$$

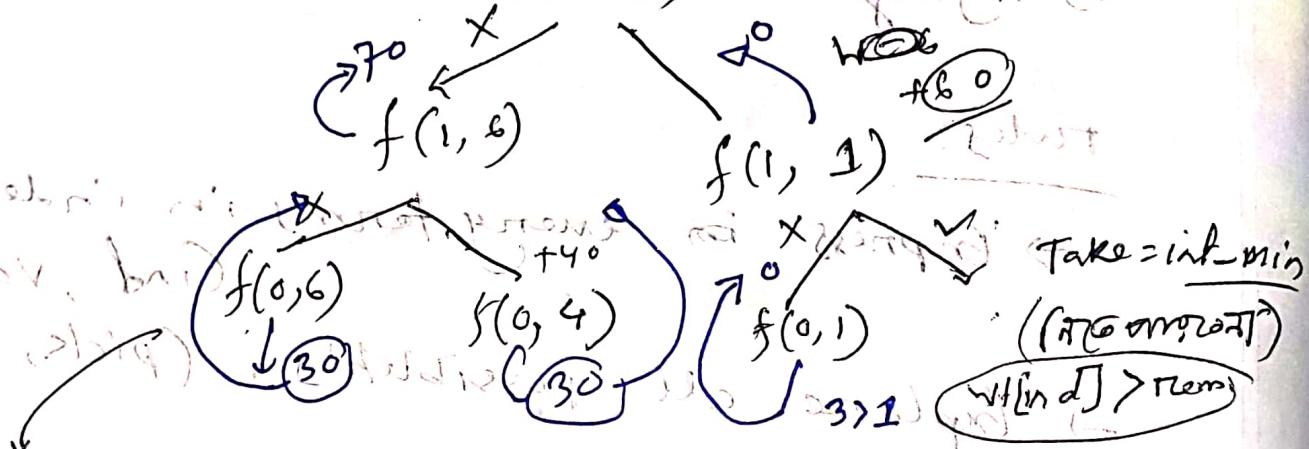
$$\begin{cases} f = f_1(t) + 0 \\ g = (0, 1)t + 0 \end{cases} \quad \text{not enough info}$$

$$\# \text{ wt} \rightarrow \begin{matrix} \text{ind} = \\ 0 & 1 & 2 \\ 3 & 2 & 5 \end{matrix} \quad \{ \quad \} \quad \{ \quad \}$$

$$V \rightarrow S \{ \begin{matrix} 72^\circ & 48^\circ & 60^\circ & 30^\circ & 30^\circ \end{matrix} \} \quad \text{W} \leftarrow 6$$

finds, w

What are conditions (R, G) the script of



$$\text{Not Take} = 0 + f(0, 6) = 30 \quad \left. \begin{array}{l} \text{Taking 6 is better than 5} \\ \text{So, } f(0, 6) = 30 \end{array} \right\} 70$$

for $f(1, 1)$

$$\text{Not taken} = 0 + f(0,1) = 0 \quad \text{⑥}$$

Take \approx Int-simil

for $f(e, b)$

$$\text{Net fake} = 0 + f(1, 6) = 70$$

$$\text{Take } = 60 + f(1,1) = 60$$

70 ✓

③

Knapsack Problem

Algo. (Recursive) T(n)

$\rightarrow O(2^n)$

$f(\text{ind}, w) \rightarrow O(N)$

{

if ($\text{ind} == 0$) {

if ($wt[0] \leq w$) return $val[0]$;

else return 0;

else {

$\text{notTake} = 0 + f(\text{ind} - 1, w)$

$\text{Take} = \frac{\text{Int} - \text{MIN}}{2}$

if ($wt[\text{ind}] \leq w$)

$\text{take} = val[\text{ind}] + f(\text{ind} - 1, w - wt[\text{ind}])$

return $\text{Max}(\text{notTake}, \text{Take});$

$\rightarrow \text{Dp}(\text{ind}, w)$

$\text{dp}(\text{ind}, w) = \text{dp}(\text{ind} - 1, w) + \text{dp}(\text{ind} - 1, w - wt[\text{ind}])$

$\text{dp}(\text{ind}, w) = \text{dp}(\text{ind} - 1, w)$

left for later) \rightarrow insert at

left for later) \rightarrow insert at

$\text{dp}(\text{ind}, w) = \text{dp}(\text{ind} - 1, w)$

left for later) \rightarrow insert at

$\text{dp}(\text{ind}, w) = \text{dp}(\text{ind} - 1, w)$

$\text{dp}(\text{ind}, w) = \text{dp}(\text{ind} - 1, w)$

⑥

$$\begin{cases} T \rightarrow O(N \times W) \\ S \rightarrow O(N \times W) + O(N) \end{cases}$$

memoization

dp[N][W+1]

rec ↓
stack space

$f(\text{ind}, \text{w})$

{ if (ind == 0)

{ if (wt[ind] <= w) ret val[ind];

else return 0;

if (dp[ind][w].n == -1)

rec f(ind-1, w);

else return dp[ind][w];

not take = 0 + f(ind-1, w)

(Take, not take) INIT min

if (wt[ind] <= w)

{ take = val[ind] + f(ind-1, w-wt[ind]);

dp[ind][w] = take;

return max(take, not take);

// driver

int knapsack(vi wt, vi val, int n, int W)

vector<vector<int>> dp (n, vector<int>

(W+1, -1));

return f(n-1, W, wt, val, dp);

Q)

Tabular Form

all the weight ~~less than~~ ^{less than} ~~w~~ will be considered

1. ~~Base Case~~ ^{ind dp(1, n-1)}
2. ~~insert wt (0, w)~~
3. ~~copy the recurrence~~ ^(bottom up)

ind $dp[0:N][m+1] = \{0\}$
for $i = \text{wt}[0] \rightarrow \text{wt}[N]$
 $dp[0][i] = \text{val}[i]$

(Every w greater than $\text{wt}[0]$, can be
on condition taken)

else (length w is $m+1$ and)

if $w = \text{val}[i]$ then

$(w \rightarrow \text{val}[i]) \in$

else if $w > \text{val}[i]$ then

inserting

if $w = \text{val}[i] + \text{val}[i+1]$ then

$(w \rightarrow \text{val}[i], \text{val}[i+1]) \in$

else if $w > \text{val}[i] + \text{val}[i+1]$ then



⑧

Tabular Form:

Knapsack (Vifont, vi. qdval, int. int. MexW)

9. `vector<vector<int>> dp(n, vector<int> (MaxW+1, 0));`

for ($\inf w = w[0]; w < \max w; w++$)

IP (JEW) ENGLISH (JEW) 500

for (int index = 0; index < n; index++)

{ for (int w = 0, w < = MaxW; w++)

```
int netTake = 0 + dpl[nd-1][w];
```

int fake = INT_MIN;

if ($wd[\text{ind}] \leftarrow w$)

Take = $\text{val}[\text{ind}] \rightarrow \text{dp}[\text{ind}-1]$

3 - Mr. W. C. and

→ $\Delta p[\text{ind}]^T w] = \max_{\overline{w}} (\text{takes, Not takes});$

refers up to $\{n\}(\max W)$

⑨

Tabular for (Space optimized)

knapsack (int wt[], int val[], int n, int MaxW)
(MaxW) $\leftarrow \{0, 1, \dots, \text{MaxW}\}$
prev (MaxW+1, 0), cur (MaxW+1, 0)
and prev (MaxW+1, 0) = 0
and cur (MaxW+1, 0) = 0
and for (int w = wt[0]; w <= MaxW; w++)
 prev[w] = val[0];
 for (int ind = 1; ind < n; ind++)
 for (int w = 0; w = < max(w, wt[ind]))
 if not take = 0 + prev[w];
 int notTake = 2nd - min;
 if (wt[ind] <= w)
 take = val[ind] + prev[w - wt[ind]]
 cur[w] = max (take, notTake);
 prev = cur;
 return prev[MaxW];

Table:

Weights: $\{3, 4, 6, 5\}$

Size = 8

Profits: $\{2, 3, 1, 4\}$

Max ($dp[i-1][j]$, $dp[i-1][j-wt[i]] + pt[i]$)

$$0 + 2 = 2$$

| i based index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-----------------------|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 2 | 3 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 | 0 | 0 | 2 | 3 | 4 | 4 | 5 | 6 |

Longest Common Subsequence

Subsequence: Order maintains.

$a b c d e$
Subsequence: ade, bce, acd, ...

(LCS) $f(x, y)$ \rightarrow $x + y$

longest common subsequence:

$S_1 = a b c d e f g h i j$ $S_2 = g i e d g i d h$
longest common subsequence: cdgi

Rules:

(for the current) $f(x, y)$

1. Express in index

2. Explore all the possibilities

3. Take the best among them

index $\rightarrow f(z) \rightarrow$ string [0 ... ?]

4. do comparison characterwise

cases: residual frappé

Case - 1: If 2 characters are same:

$$\text{LCS}(\boxed{1234}, \boxed{123})$$

$$= 1 + \cos(\text{f13}, y), \text{f12,3}$$

$$\therefore 2 + \text{LCS}(\{3, 4\}, \{3\}) \text{ (Final)}$$

General \rightarrow $1 + 2 \cos(\text{ind}_1 - 1, \text{ind}_2 - 1)$

Case-2: 2 characters not same

$\{cs(yx_A, AY_2x)\}$

$$LCS(YXA, AYZ)$$

lcs (indx1, indx2)

Leg(ind1, ind2-1)

General:

0 + ~~less~~ (Man)

0 + M an (les(indx-1, indy))

lcs (ind1, ind2-1)

~~2.500 mts. ascd.~~

code: (recursive)

string, index
lcs(x, y, m, n)

{ if (m = n = 0) return 0;

if (x[m] == y[n])

return 1 + lcs(x, y, m-1, n-1);

return ~~lcs(x, y, m-1, n)~~ + ~~lcs(x, y, m, n-1)~~

code (convert to DP)

int lcs(x, y, m, n)

if (n == 0 || m == 0) return 0;

if (dp[m-1][n-1] != -1)

return dp[m-1][n-1];

if (x[m-1] == y[n-1])

return dp[m-1][n-1] = 1 + lcs(x, y, m-1, n-1);

return dp[m-1][n-1] = max(lcs(x, y, m-1, n), lcs(x, y, m, n-1))

}

Iterative: (Tabulation)

ACACDB

CBDA

if (match)

$S(i+1, j+1) + 1$

else $\max(i+1, j), (i, j+1)$

| | | C | B | D | A | |
|---|---|---|---|---|---|-----------|
| | | 0 | 0 | 0 | 0 | 0 |
| | | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 0 | 0 | 0 | 0 | 1 (Match) |
| C | 0 | 1 | 1 | 1 | 1 | 1 |
| A | 0 | 1 | 1 | 1 | 1 | 2 (Match) |
| D | 0 | 1 | 1 | 2 | 2 | 2 |
| B | 0 | 1 | 1 | 2 | 2 | 2 |
| | 0 | | | | | |

code:

Technique:

1. copy the base case

$(+i, +j) \rightarrow (i, j)$ \leftarrow shift

base case

$(+i, +j) \rightarrow (i, j)$ 3. copy the recurrence

$\rightarrow \text{vector} \ll \text{vector} \langle \text{int} \rangle \rangle \text{ dp}(n+1, \text{vector} \langle \text{int} \rangle \langle m+1, 0 \rangle);$
~~($i \in [0, n]$)~~
for (int $i=1$; $i \leq n$; $i++$)

~~for (int $j=1$; $j \leq m$; $j++$)~~

{ if ($s[i-1] == s1[j-1]$)

$\text{dp}[i][j] = 1 + \text{dp}[i-1][j-1];$

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

}

out $\ll \text{dp}[n][m]$

Space Optimization:

vector<int> prev(n+1, 0), cur(n+1, 0);

```
for (int i = 1; i <= n; i++)
```

```
for (int j=1; j <= m; j++)
```

$\{ \quad \text{if } (s[i:i]) = s[i:j]\}$

(+1) $\text{cur}(j) = 1 + \text{prev}(j-1)$

Griffiths, Mrs

$$\text{cur}(j) = \max(\text{prev}(j), \text{cur}(j-1)).$$

$$p_{\text{new}} = \sin(\pi)$$

conf << prov[*mg*];

Centrifugal pump

Knuth-Morris-Pratt (KMP)

prefix: abc d

a, ab, abc, abc d.

suffix: abc d

do ~~do~~, d, cd, bd, abc d.

printing LCS

{ int $m = dp[m][n]$;
string ans = " " ; ans += "\$" ; // dummy value

for (int index = m - 1 ;

int i = n, j = m ;

while (i > 0 && j > 0)

{ if ($s[i-1] = s[j-1]$)

{ ans [index] = s [i-1] ;

index-- ;

i-- ;

j-- ;

else if ($dp[i-1][j] > dp[i][j-1]$)

i-- ;

else j-- ;

}

String Matching Algorithm

Knuth - Morris - Pratt Algorithm (KMP)

Another ~~method~~ for string matching

vector<int> lpsF(string pattern)

{
vector<int> lps (pattern.length());
vector<int> lps (pattern.length());

int index = 0;

for (int i = 1; i < pattern.length(); —)

{ if (pattern [index] == pattern [i])

{ lps [i] = index + 1;

{ index++; i++;

else if (index) index = lps [index - 1]

else index = 0;

else lps [i] = 0; i++;

return lps;

}

and search the string with pattern

(Ques) Implement KMP - Return - Max

Ans: KMP (string text, string pattern)

Vector $\langle \text{int} \rangle$ lps of (pattern).

if ($i = 0$), $j = 0$
while ($i < \text{text.length}$) do

(if $\text{text}[i] == \text{pattern}[j]$) $i = i + 1$ and

$\text{counting} = \{ i++ ; j++ \}$

else $i = i - j + 1$

else { if ($j != 0$) $j = \text{lps}[j-1]$ }

($i = \text{text.length}$) $i = i + 1$

if ($i < \text{text.length}$) $i = i + 1$

cout << "found" << endl;

}

return count;

}

and search the string with pattern