Kosen Club CP 勉強会

DYNAMIC PROGRAMMING

Dynamic Programming

対象となる問題を複数の部分問題に分割し、部分問題の計算結果を記録しながら解いていく手法を総称してこう呼ぶ。

細かくアルゴリズムが定義されているわけではなく、下記2条件を満たすアルゴリズムの総称である。

１．分割統治法：部分問題を解き、その結果を利用して、問題全体を解く

２．メモ化：部分問題の計算結果を再利用する

Example 1: Fibonacci number

A[1] = 1, A[2] = 1, A[n] = A[n-1] + A[n-2]

Bottom Up:

```
int dp[N] = {0};

int main() {
  dp[1] = dp[2] = 1;
  For(int i=3; i<=n; i++) {
      dp[i] = dp[i-1] + dp[i-2];
  }
  return dp[n];

}
```

Top Down:

```
int dp[N] = {0};

int calc(int n) {
      If(dp[n]) return dp[n];
      return dp[n] = calc(n-1) + calc(n-2);

}

Int main() {

   return calc(n);

}
```

Example 2: Linear DP

[Climbing Stairs – LeetCode]

You are climbing a staircase. It takes `n` steps to reach the top.

Each time you can either climb `1` or `2` steps. In how many distinct ways can you climb to the top?

1.State: dp[n], 1 ees n hurtelh yalgaatai zamiin too

2.Transition: dp[n] = dp[n-1] + dp[n-2], suuliin alham ni 1 baih, suuliing alham ni 2 baih

3.Answer: dp[n]

Example 3: Knapsack DP

D - Knapsack 1 (atcoder.jp)

N 個の品物があります。品物には $1, 2, \ldots, N$ と番号が振られています。各 i $(1 \leq i \leq N)$ について、品物 i の重さは w_i で、価値は v_i です。

太郎君は、N 個の品物のうちいくつかを選び、ナップサックに入れて持ち帰ることにしました。ナップサックの容量は W であり、持ち帰る品物の重さの総和は W 以下でなければなりません。

太郎君が持ち帰る品物の価値の総和の最大値を求めてください。

1.State: dp[i][j], ehnii i shinamonogoos hund ni j baih ueiin bolomjit hamgiin ih kachi

2.Transition: dp[i][j] = max{dp[i-1][j], dp[i-1][j-w[i]] + v[i]}, i-iig awahgui, i-iig awna 2-iin max

3.Answer: dp[n][W]

Example 4: Interval DP

[Palindromic Substrings - LeetCode]

Given a string `s`, return *the number of* ***palindromic substrings*** *in it.*

A string is a **palindrome** when it reads the same backward as forward.

A **substring** is a contiguous sequence of characters within the string.

1.State: dp[i][j], Is string[i:j] palindrome?

2.Transition: dp[i][j] = (s[i]==s[j]) && dp[i+1][j-1], 2 zah ni tentsuu buguud goliin string ni pali

3.Answer: Sum of dp[i][j], (i<=j)

Example 5: bit DP

DPL_2_A < Problems | Aizu Online Judge (u-aizu.ac.jp)

For a given weighted directed graph $G(V, E)$, find the distance of the shortest route that meets the following criteria:

- It is a closed cycle where it ends at the same point it starts.
- It visits each vertex exactly once.

1.State: dp[s][i], bitmask s hotoor neg udaa zochlood i deer zogsson baih min dist

2.Transition: dp[s][i] = min{dp[s-i][j] + d[j][i] | all j included in (s-i)}

3.Answer: min{dp[(1<<n)-1][i] | all i}

Example 6: Cadane's Algorithm

Maximum Subarray - LeetCode

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return *its sum*.

A **subarray** is a **contiguous** part of an array.

1.State: dp[i], i-dahi element deer tugsuh subarray dotroos max sum ni

2.Transition: dp[i] = max{a[i], d[i-1] + a[i]}, zuwhun ganst element, urdah-iin max nemeh

3.Answer: max{dp[i] | all i}

Example 7: Longest Common Subsequence

[Longest Common Subsequence - LeetCode]

Given two strings `text1` and `text2`, return *the length of their longest* ***common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings.

1.State: dp[i][j], text1-iin ehnii i string, text2-iin ehnii j string dotroos LCS

2.Transition: dp[i][j] = max{dp[i-1][j-1] + (text1[i]==text2[j]), d[i-1][j], dp[i][j-1]},

3.Answer: dp[n][m]

Example 8: Edit Distance

[Edit Distance - LeetCode]

Given two strings `word1` and `word2`, return *the minimum number of operations required to convert* `word1` *to* `word2`.

You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

1.State: dp[i][j], word1-iin ehnii i string ees word2-iin ehnii j string uusgeh min operations

2.Transition: dp[i][j] = max{dp[i-1][j-1] + (word1[i]!=word2[j]), d[i-1][j]+1, dp[i][j-1]+1},

3.Answer: dp[n][m]

Example 9: Longest Increasing Subsequence

[Longest Increasing Subsequence - LeetCode]

Given an integer array `nums`, return the length of the longest strictly increasing subsequence.

A **subsequence** is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements. For example, `[3,6,2,7]` is a subsequence of the array `[0,3,1,6,2,2,7]`.

1.State: dp[i], i urttai LIS –iin hamgiin suuliin element-iin min

2.Transition: *lower_bound(dp, dp + n, a[i]) = a[i];

3.Answer: max i such that dp[i]<INF

Example 10: DP on Trees

https://atcoder.jp/contests/dp/tasks/dp_p

N 頂点の木があります。頂点には $1, 2, \ldots, N$ と番号が振られています。各 i $(1 \leq i \leq N-1)$ について、i 番目の辺は頂点 x_i と y_i を結んでいます。

太郎君は、各頂点を白または黒で塗ることにしました。ただし、隣り合う頂点どうしをともに黒で塗ってはいけません。

頂点の色の組合せは何通りでしょうか？ $10^9 + 7$ で割った余りを求めてください。

1.State: dp[i][0], subtree i –iin root ni black baih bolomjit budalt
dp[i][1], subtree i –iin root ni white baih bolomjit budalt

2.Transition: dp[i][0] = multi{dp[j][1] | j child of i}
dp[i][1] = multi{dp[j][1]+dp[j][0] | j child of i}

3.Answer: dp[0][1] + dp[0][0]