Dynamic Programming Variations

Recap of DP

Write the problem in a way such that you can compute the answer efficiently if you have solved smaller versions of the same problem.

Start by computing the smaller versions, only compute each one once, and store the result.

The types of values we store we call the *state* (e.g. “my dp state for https://dmoj.ca/problem/dpa is dp[i] is the cheapest way to get to from the starting position to position i”).

The function we use to calculate the value of a state based on previous states is called the *transition* function (e.g. “the transition function is dp[i] = min(dp[i-1] + abs(arr[i] - arr[i-1]), dp[i-2] + abs(arr[i] - arr[i-2]))”).

Often trivial, but you will need to have a *base case*, a dp state whose value does not depends on other dp states (e.g. “the base case is dp[0] = 0”).

Purpose of today’s lesson

Introduce specific types of DP states that come up often

Suggest common strategies and optimizations that can be used to solve them

Interval DP

Example problem

You are given an array of length N ($2 \leq N \leq 400$), where the ith element has value a_i ($1 \leq a_i \leq 10^9$).

In one operation, you can choose two adjacent elements and merge them into a new element. If the two elements had values x and y, the new element will have value x + y and this incurs a cost of x + y.

Find the minimum cost required to merge all elements in the array into one element.

https://dmoj.ca/problem/dpn

Example problem solution

State: dp[l][r] is min cost to merge all elements in the subarray (interval) [l, r).

We don’t need to know anything else about a subarray to use it for future calculations, since regardless of the merge order, the value of the new element is always the same.

Transition: $dp[l][r] = \min_{l<i<r}(dp[l][i] + dp[i][r]) + sum[l][r]$ where sum[l][r] is the sum of the subarray [l, r).

Base case: $dp[i][i+1] = 0$

https://github.com/bci-csclub/comp-prog-sols/blob/main/dmoj/dpn/dpn.cpp

Interval DP definition

‘Interval DP’ most commonly refers to a DP algorithm where your state is some function computed over a subarray.

Complexity will usually be at least N^2 or N^3.

Useful to iterate through intervals in increasing order of length.

Problems

https://dmoj.ca/problem/dpn
https://dmoj.ca/problem/ccc16s4 (need to be careful with transition to get N^3 instead of N^4)
https://dmoj.ca/problem/ioi09p4 (2D)
https://dmoj.ca/problem/dpl (some basic game theory)
https://dmoj.ca/problem/noi09p3 (harder to recognize that this is interval dp, but problem shouldn't be too bad once you know that it's interval dp)

Tree DP

Example problem

Given a tree with N ($1 \leq N \leq 10^5$) nodes, count the number of ways to paint the vertices black and white such that no two adjacent nodes are colored.
Output the answer modulo $10^9 + 7$.

https://dmoj.ca/problem/dpp

Example problem solution

State: dp[i][0] is number of valid ways of painting nodes in the subtree of i such that i is painted white. dp[i][1] is number of ways if node i is black.

Transition: dp[i][0] = prod(dp[j][0] + dp[j][1]) % 1e9+7 for j which are the children of i

dp[i][1] = prod(dp[j][0]) % 1e9+7 for j which are the children of i

Base case: dp[i][0] = dp[i][1] = 1 if i has no children

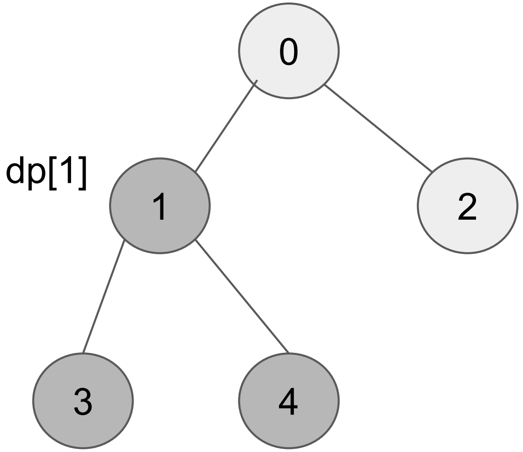
It is most convenient to use a DFS to calculate the dp values.

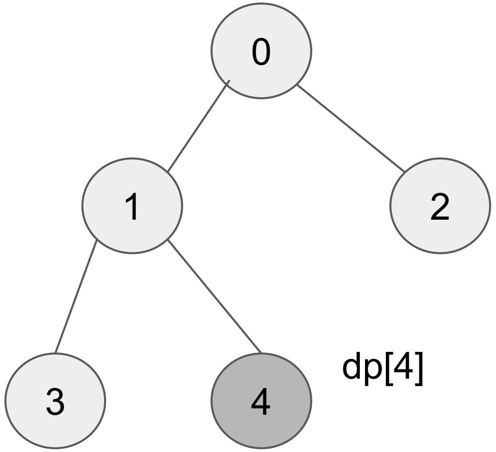
Note that we can take mod of intermediate values. This is the main reason problems might require you to output the answer modulo a large number (often prime); small integers are easier to work with.

(a + b) % m = (a % m + b % m) % m

(a * b) % m = (a % m) * (b % m) % m

https://github.com/bci-csclub/comp-prog-sols/blob/main/dmoj/dpp/dpp.cpp


0
dp[1]
1
2
3
4


0
1
2
3
4
dp[4]

Tree DP definition

‘Tree DP’ most commonly refers to a DP algorithm used for a problem which has a tree (graph).

The state often corresponds to a subtree.

DFS is often used.

Sometimes you will have more dp states, ones which correspond to a portion of the tree excluding each subtree. In this case you do a dfs upwards first to calculate the dp values for subtrees, then go downwards (with a second dfs) to calculate the dp state above a node. See dmopc14c4p6 and dpv for examples of such problems.

Problems

https://dmoj.ca/problem/dpp
https://dmoj.ca/problem/dmopc19c3p4
https://dmoj.ca/problem/dmopc14c4p6 (up-down tree dp)
https://dmoj.ca/problem/dpv (up-down tree dp, you do **not** need any more knowledge about modular arithmetic than those on the previous slides, in fact knowing more might actually make the problem harder)
https://dmoj.ca/problem/ioi05p6 (dp state might need many values at a node)

Bitmask DP

Example problem

Given an edge-weighted directed graph with N ($1 \le N \le 18$) nodes, find the longest simple path (a path which does not visit the same node twice) from node 0 to node N-1.

https://dmoj.ca/problem/cco15p2

Example problem solution

State: dp[s][i] is the longest path starting from node 0 and ending at node i such that s is the set of visited nodes.

Transition: dp[s][i] = max(dp[s \ i][j] + adjMat[j][i]) where j is an element of s and there is an edge from j to i, \ is the set difference operator, and adjMat[j][i] is the length of the edge from j to i.

Base case: dp[{0}][0] = 0

https://github.com/bci-csclub/comp-prog-sols/tree/main/dmoj/cco15p2

Integers to represent sets

i^{th} bit represents whether element i is in the set.

Takes significantly less memory than an actual set, indexes arrays as well.

Bitwise Operators

`a >> b` - bitwise right shift - shifts every bit in `a` to the right by `b` positions
`a << b` - bitwise left shift - shifts every bit in `a` to the left by `b` positions
`a & b` - bitwise AND - performs the logical AND operator on each bit of `a` and `b`
`a | b` - bitwise OR - performs the logical OR operator on each bit of `a` and `b`
`a ^ b` - bitwise XOR - performs the logical XOR operator on each bit of `a` and `b`
`~a` - bitwise not - flips all the bits in the number
Treating integers as bools - in c++/python, any integer other than 0 is true and 0 is false. Java doesn't support casting between int and boolean.

Examples

0101 << 1 is 1010

0101 & 0110 is 0100

0110 ^ 1010 is 1100

Common operations

Compute 2^n: `1 << n`

Set the ith bit to 1: `a |= 1 << i`

Retrieve the ith bit: `((a >> i) & 1)`

Create a number with the rightmost i bits set to 1:
`((1 << i) - 1)`

Get the least significant set bit of a: `a & -a`

Note that the order of operations of bitwise operators is often not what you want, so I would recommend using parentheses when writing complicated expressions.

More bitwise functions

C++ specific (and all starting with __ compiler specific):

__builtin_popcount(x) and __builtin_popcountll(x) return number of set bits in x. std::popcount is standard in C++20.

__builtin_ctz(x) and __builtin_ctzll(x) return number of trailing zeros in x.

__builtin_clz(x) and __builtin_clzll(x) return number of leading zeros in x.

__lg(x) returns floor($\log_2$(x)). You can also use the clz functions for this.

Sum over subsets example problem

You have N rabbits ($1 \le N \le 16$) and want to partition them into groups. If the i^{th} and j^{th} rabbit are in the same group, your score increases by $a_{i,j}$ ($-10^9 \le a_{i,j} \le 10^9$). Find max possible score.

https://dmoj.ca/problem/dpu

Sum over subsets example problem solution

First compute score[s] = score received from putting all rabbits in s in the same group (s is a set, or an integer < 2^N, same thing).

State: dp[s] is max score you can get if you've determined the groups for all rabbits in s (although they might not all be in the same group) and they are in separate groups from the remaining rabbits (those not in s).

Transition: dp[s] = max(dp[t] + score[s \ t]) where t is a subset of s.

Base case: dp[0] = 0

Naive implementation: check all s, t pairs and check whether t is s subset of s. Complexity: O(4^N).

Better implementation: for each s, loop through only the t that are actually its subsets. Complexity: O(3^N).

How? `for(int t = (s-1) & s; t != s; t = (t-1) & s);` Note that this excludes t = s.

https://github.com/bci-csclub/comp-prog-sols/blob/main/dmoj/dpu/dpu.cpp

Problems

https://dmoj.ca/problem/cco15p2
https://dmoj.ca/problem/dpo
https://dmoj.ca/problem/coci06c1p5
https://dmoj.ca/problem/wac4p3 (requires minor amounts of math/geometry)
https://dmoj.ca/problem/qccp1 (also some basic game theory)
https://dmoj.ca/problem/dpu (sum over subsets)