Last update: March 5, 2025 Original

Divide and Conquer DP

Divide and Conquer is a dynamic programming optimization.

Preconditions

Some dynamic programming problems have a recurrence of this form:

$$dp(i, j) = \min_{0 \leq k \leq j} dp(i - 1, k - 1) + C(k, j)$$

where $C(k, j)$ is a cost function and $dp(i, j) = 0$ when $j < 0$.

Say $0 \leq i < m$ and $0 \leq j < n$, and evaluating C takes $O(1)$ time. Then the straightforward evaluation of the above recurrence is $O(mn^2)$. There are $m \times n$ states, and n transitions for each state.

Let $opt(i, j)$ be the value of k that minimizes the above expression. Assuming that the cost function satisfies the quadrangle inequality, we can show that $opt(i, j) \leq opt(i, j + 1)$ for all i, j. This is known as the *monotonicity condition*. Then, we can apply divide and conquer DP. The optimal "splitting point" for a fixed i increases as j increases.

This lets us solve for all states more efficiently. Say we compute $opt(i, j)$ for some fixed i and j. Then for any $j' < j$ we know that $opt(i, j') \leq opt(i, j)$. This means when computing $opt(i, j')$, we don't have to consider as many splitting points!

To minimize the runtime, we apply the idea behind divide and conquer. First, compute $opt(i, n/2)$. Then, compute $opt(i, n/4)$, knowing that it is less than or equal to $opt(i, n/2)$ and $opt(i, 3n/4)$ knowing that it is greater than or equal to $opt(i, n/2)$. By recursively keeping track of the lower and upper bounds on opt, we reach a $O(mn \log n)$ runtime. Each possible value of $opt(i, j)$ only appears in $\log n$ different nodes.

Note that it doesn't matter how "balanced" $opt(i, j)$ is. Across a fixed level, each value of k is used at most twice, and there are at most $\log n$ levels.

Generic implementation

Even though implementation varies based on problem, here's a fairly generic template. The function `compute` computes one row i of states `dp_cur`, given the previous row $i - 1$ of states `dp_before`. It has to be called with `compute(0, n-1, 0, n-1)`. The function `solve` computes `m` rows and returns the result.

```
int m, n;
vector<long long> dp_before, dp_cur;

long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};

    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid), k});
    }

    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

long long solve() {
    dp_before.assign(n,0);
    dp_cur.assign(n,0);

    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);

    for (int i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }

    return dp_before[n - 1];
}
```

Things to look out for

The greatest difficulty with Divide and Conquer DP problems is proving the monotonicity of opt. One special case where this is true is when the cost function satisfies the quadrangle inequality, i.e., $C(a,c) + C(b,d) \le C(a,d) + C(b,c)$ for all $a \le b \le c \le d$. Many Divide and Conquer DP problems can also be solved with the Convex Hull trick or vice-versa. It is useful to know and understand both!

Practice Problems

- AtCoder - Yakiniku Restaurants

- CodeForces - Ciel and Gondolas (Be careful with I/O!)
- CodeForces - Levels And Regions
- CodeForces - Partition Game
- CodeForces - The Bakery
- CodeForces - Yet Another Minimization Problem
- Codechef - CHEFAOR
- CodeForces - GUARDS (This is the exact problem in this article.)
- Hackerrank - Guardians of the Lunatics
- Hackerrank - Mining
- Kattis - Money (ACM ICPC World Finals 2017)
- SPOJ - ADAMOLD
- SPOJ - LARMY
- SPOJ - NKLEAVES
- Timus - Bicolored Horses
- USACO - Circular Barn
- UVA - Arranging Heaps
- UVA - Naming Babies

References

- Quora Answer by Michael Levin
- Video Tutorial by "Sothe" the Algorithm Wolf

Contributors:

arknave (45.97%) sgtlaugh (31.45%) dmiao623 (5.65%) mhayter (4.03%) adamant-pwn (4.03%) omkarbajaj073 (4.03%) CDTheGod (0.81%) jakobkogler (0.81%) simonyanli (0.81%) 18o3 (0.81%) Alpacabla (0.81%) roll-no-1 (0.81%)