**Divide and Conquer Speedup**

* DP must look like dp[i][j] = (max dp[j][k - 1] + C(j, i) for j < i)
* Monotonicity of opt[i][j] for fixed i and 1 <= j <= N

void dnc(int s, int e, int l, int r, int k) {

int m = (s + e) / 2;

for (int i = l; i <= min(m, r); i++) {

// do stuff to determine opt[m][k]

}

ans = max(ans, dp[m][k]);

if (s < m) dnc(s, m - 1, l, opt[m][k], k);

if (m < e) dnc(m + 1, e, opt[m][k], r, k);

}

**Convex Hull Speedup**

DP must look like dp[n] = min/max(m(i) \* x(n) + c(i)) + v(n)

* Look at cost function to see if terms can be arranged to form a linear function
* Use deque to store max convex hull
* Negate gradient, y-intercept and maximum y-value to form a min convex hull

int qry(int x) {

while (D.size() > 1)

if (D[0].first \* x + D[0].second <= D[1].first \* x + D[1].second) D.pop\_front();

else break;

return D[0].first \* x + D[0].second;

}

long double is(int a, int b, int c, int d) { return (long double)(d - b) / (a - c); }

void insert(int m, int c) {

for (int s = D.size(); s > 1; s--) {

if (is(D[s - 1].first, D[s - 1].second, m, c) <= is(D[s - 2].first, D[s - 2].second, m, c)) D.pop\_back(); // cross multiply if precision errors

else break;

}

D.emplace\_back(m, c);

}

// check for parallel lines (ignore the one with lower y-intercept)

**Digit DP**

* Find a way to compress the valid condition into a reasonable state
* State optimizations can be used to change dp[index][last digit][boolean value] to dp[index][boolean value][boolean value] (dunjudge\_unluckyfloors)

**Lexicographical DP (Find kth lexicographically smallest valid sequence)**

* Imagine a sorted list of all the valid sequences
* Create helper DP for counting # of valid sequences with certain prefix
* Find largest character such that lower + dp(chosen string + character) <= k
* Shift up lower (+ dp(chosen string + character)) if lower + dp(chosen string + character) < k
* dunjudge\_enumeratingbrackets, APIO\_2008\_dna

**Lexicographical DP (find lexicographical index of string)**

* Use Digit DP, relabel characters with numbers from 1...character max (from smallest to largest)

**DS Speedup (Fenwick/Segment tree)**

If DP looks like dp[i] = (max dp[j] for j < i, valid(i, j)) and valid is a contiguous range for fixed i, Fenwick tree can be used to store prefix maximums

* dunjudge\_lis\_ex
* Segment tree for range maximums (dunjudge\_orangecollector)

**DS Speedup (Sliding Min/Max Deque)**

Implementation

deque<pair<int, int> > d;

for (int i = 0; i < n; i++) {

while (!d.empty() && d.back().first > a[i]) d.pop\_back();

d.emplace\_back(a[i], i);

while (!d.empty() && d.front().second <= i - k) d.pop\_front();

// d.front().first now stores minimum of [i - k + 1, …, i]

}

* DP must look like dp[i] = min/max(dp[j] + c[j]) + v[i] for left[i] <= j < i
* left[i] <= left[i + 1] for 1 <= i <= N

**Speedup (Sliding Pointer)**

* If accepted range of states to transition to only increases as a prefix, a sliding pointer can be used to keep track of maximum
* dunjudge\_modules, dunjudge\_personalspace