Dynamic programming optimizations

Maxim Akhmedov
Moscow State University, Yandex
January 27th, 2017

This text contains the brief description of several dynamic programming optimizations techniques that often appear on programming competitions.

1 Optimum monotonicity / binary search / two pointers

**Problem**: professor lives in an $n$ floor building and has $k$ transistors. He knows that there exists some floor $i$ ($1 \leq i \leq n - 1$) that if he throws a transistor from floor $i$ or lower it won’t be broken, and if he throws it from floor $i + 1$ or higher, it will definitely be broken. In which smallest number of throws professor can determine that critical $i$?

**DP**: $D[n][k]$ is a minimum number of throws needed to professor to find out the critical $i$ if he knows that floor $l$ is still OK, floor $r$ is not OK, $r - l = n$ and he has $k$ transistors left.

Then:

- $D[1][k] = 0$ since we already found the critical $i$;
- $D[n][k] = \min_{1 \leq j \leq n-1} \max(D[j][k-1], D[n-j][k])$;

This is an $O(n^2k)$ DP.

**Optimization 1**: $f(j) = D[j][k-1]$ is decreasing, $g(j) = D[n-j][k]$ is increasing, hence $\max(D[j][k-1], D[n-j][k])$ decreases till some moment (while $D[j][k-1] \geq D[n-j][k]$) and then increases (in this statement terms “increasing/decreasing” allow equality, i.e. they are not strict). Hence, we may find an optimum point $optj[n][k]$ as a root of a function $f(j) - g(j) = D[j][k-1] - D[n-j][k]$ using the binary search. This leads to an $O(nk \log n)$ solution.

**Optimization 2**: when we move from $n$ to $n + 1$, the function $f(j) = D[j][k-1]$ stays the same and $g(j) = D[n-j][k]$ is replaced with $g^*(j) = D[n+1-j][k]$. Note that $g^*(j) \geq g(j)$, hence $optj[n+1][k] \geq optj[n][k]$. In order to calculate $optj[n+1][k]$, assign it to $optj[n][k]$ and increase it until $f(optj[n+1][k])$ becomes smaller than $g(optj[n+1][k])$. This leads to an $O(nk)$ solution.

2 Convex hull trick (linear version)

**Problem**: You are given $n$ numbers $x_1 < x_2 < \ldots < x_n$ and a constant $C$. Choose some subsequence of them $y_1, \ldots, y_k$ such that $y_1 = x_1$, $y_k = x_k$ and the value $\sum_{i=1}^{k-1} (y_{i+1} - y_i)^2 + Ck$ is as small as possible.

**DP**: $D[i]$ is the smallest possible $\sum_{i=1}^{j-1} (y_{j+1} - y_j)^2 + Cj$ if $y_j = x_i$ for some $j$. Then:
\[ D[1] = -C; \]
\[ D[i] = \min_{1 \leq j \leq i-1} (D[j] + (x_i - x_j)^2 + C); \]

This is an \( O(n^2) \) solution.

**Optimization 1:**
\[
D[i] = \min_{1 \leq j \leq i-1} (D[j] + (x_i - x_j)^2 + C) = x_i^2 + C + \min_{1 \leq j \leq i-1} (D[j] + x_j^2 - 2x_ix_j) = x_i^2 + C + \min_{1 \leq j \leq i-1} (x_j, D[j] + x_j^2) \cdot (-x_i, 1) \]
\[
D[i] = x_i^2 + C + \max_{1 \leq j \leq i-1} (x_j, D[j] + x_j^2) \cdot (x_i, -1) \]

Let \( \vec{P}_j = (x_j, D[j] + x_j^2) \). Keep the lower hull of \( \vec{P}_j \). The \( j \) such that \( \vec{P}_j \cdot \vec{v}_i \to \max \) is always some point of a convex hull of \{\( P_j \}\}; namely, the lower hull of those points because the \( y \)-component of a vector \( \vec{v}_i \) in our case is negative.

Lower hull may be kept in the stack. New points are added to the right of the old ones (since \( x_j \) increases), so the stack may be recalculated in amortized \( O(1) \) (similar to the Andrew monotone chain algorithm).

Optimum \( j \) may be find via the binary search over the convex hull since \((\vec{P}_j, \vec{v}_i)\) increases up to some moment and then decreasing over all \( j \) belonging to the lower hull.

The complexity is \( O(n \log n) \).

**Optimization 2:** note that vector \( \vec{v}_i \) also moves to the right (its \( x \)-component increases).

It means that the pointer on the optimum point on lower hull also moves only to the right. Keep the optimum pointer \( \text{opt}[i] \) and try to move it to the right while it is profitable when moving from \( i \) to \( i+1 \).

The complexity is \( O(n) \).

## 3 Divide and Conquer optimization

**Problem:** You are given \( n \) integers \( x_1, x_2, \ldots, x_n \). Divide them into \( k \) consecutive groups such that \( \sum_{i=1}^{k} w_i \log w_i \to \min \) where \( w_i \) is the sum in the \( k \) group.

**DP:** \( DP[i][j] \) is the minimum penalty for dividing first \( j \) numbers into \( i \) groups. Then:

- \( DP[0][0] = 0; \)
- \( DP[i][j] = \min_{0 \leq z \leq j-1} (DP[i-1][z] + (S[j] - S[z]) \log(S[j] - S[z])) \) where \( S_j = x_1 + x_2 + \ldots + x_j \);

This is an \( O(n^2k) \) solution.

**Optimization:** notice the important property of optimal point monotonicity. Denote as \( \text{opt}_z[i][j] \) the value of \( z \) that is the optimum for the expression above.

**Lemma:** \( \text{opt}_z[i][j] \leq \text{opt}_z[i][j+1]. \)

**Lemma proof:** use induction and Karamata’s inequality.

Let’s calculate the \( i \)-th layer of DP using the following recursive procedure:
• **void calc(i, l, r)**

• Pre-requisite: \( \text{opt}[i][l-1] \) and \( \text{opt}[i][r+1] \) are already calculated (let \( \text{opt}[i][0] = 1 \) and \( \text{opt}[i][n+1] = n \));

• If \( l > r \), return;

• Let \( m = \lfloor (l + r)/2 \rfloor \), calculate \( \text{opt}[i][m] \) by iterating with \( z \) between \( \text{opt}[i][l] \) and \( \text{opt}[i][r] \);

• Make a recursive call of \( \text{calc}(i, l, m - 1) \) and \( \text{calc}(i, m + 1, r) \).

In total, each level of recursion works in \( O(n) \) and there are \( \log n \) recursion levels. Hence, everything works in \( O(nk \log n) \).

## 4 Knuth optimization

**Problem:** You are given values \( x_1, x_2, \ldots, x_n \). Organize them into a binary tree (without reordering) so that the sum of the values multiplied by their depths in the tree is as small as possible.

**DP:** \( D[l][r] \) is the cost of the best tree that may be built over the elements from \( l \)-th to \( r \)-th.

- \( D[l][l-1] = x_l \);
- \( D[l][r] = \min_{l \leq i \leq r} \left( D[l][i-1] + D[i+1][r] + (x_l + x_{l+1} + \ldots + x_r) \right) = \min_{l \leq i \leq r} (D[l][i-1] + D[i+1][r] + (S[r] - S[l-1])) \) where \( S[r] = x_1 + x_2 + \ldots + x_r \).

This is an \( O(n^3) \) DP.

**Optimization:** Consider \( \text{opti}[l][r] \) to be the optimum value of \( i \) for the formula above.

**Lemma:** \( \text{opti}[l][r-1] \leq \text{opti}[l][r] \leq \text{opti}[l+1][r] \).

**Lemma proof:** prove it by yourself. Prove the \( \text{opti}[l][r-1] \leq \text{opti}[l][r] \) by contradiction, consider the right path inside the optimum binary search tree, and find the contradiction.

Now, calculate DP in order of increasing \( r-l \). Iterate with \( i \) only in range \( \text{opti}[l][r-1], \text{opti}[l+1][r] \). Thus, the running time for a fixed \( r-l = d \) will be proportional to \( \text{opti}[d+1][2] - \text{opti}[d][1] + \text{opti}[d+2][3] - \text{opti}[d+1][2] + \ldots + \text{opti}[n][n-d+1] - \text{opti}[n-1][n-d] = \text{opti}[n][n-d+1] - \text{opti}[d][1] = O(n) \). So, the overall running time is \( O(n^2) \).

## 5 Lagrange optimization

**Problem:** IOI2016 Aliens [http://ioinformatics.org/locations/ioi16/contest/day2/aliens.pdf]

**DP and optimization:** Refer to the analysis of the contest [http://ioinformatics.org/locations/ioi16/contest/IOI2016_analysis.pdf]