2024 Columbia Training Camp

Day 8: DP Optimization

Christian Yongwhan Lim

Wednesday, August 28, 2024

DP Optimizations

- There are several cases where we can make dynamic programming run (much) faster; we need applying some tricks in case-by-case basis. Without the tricks, they will TLE.
- The problem difficulty is usually **CF 2400+**.
- Some well known tricks are:
  - **Divide and Conquer Optimization**
  - **Knuth's Optimization**
  - **Convex Hull Trick**
- Let's take a brief look at what they are and jump immediately to practice problems.

Divide and Conquer Optimization

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

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

- Let `opt(i,j)`, "optimal splitting point", be the value of `k` that minimizes the above expression.

Divide and Conquer Optimization

- Assuming that the cost function satisfies the quadrangle inequality:

$$C(a,c) + C(b,d) \leq C(a,d) + C(b,c) \text{ for all } a \leq b \leq c \leq d.$$

- We can show that `opt(i,j) ≤ 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.

Divide and Conquer Optimization

- 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') ≤ opt(i,j)`.
  - This means when computing `opt(i,j')`, we do not have to consider as many splitting points!

Divide and Conquer Optimization

- 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(m n log n)` runtime.
  - Each possible value of `opt(i,j)` only appears in `log n` different nodes.

Divide and Conquer Optimization: Important Note

- 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.
- 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!

Knuth's Optimization

- Knuth's optimization, also known as the **Knuth-Yao Speedup**, is a special case of dynamic programming on ranges, that can optimize the time complexity of solutions by a linear factor, from **$O(n^3)$** for standard range DP to **$O(n^2)$**.

Knuth's Optimization

- The Speedup is applied for transitions of the form:

$$dp(i,j) = \min_{i \le k < j} [dp(i,k) + dp(k+1,j) + C(i,j)].$$

- Let `opt(i,j)`, "optimal splitting point", be the value of k that minimizes the expression in the transition.

Knuth's Optimization

- The optimization requires that the following holds:

$$opt(i, j-1) \leq opt(i, j) \leq opt(i+1, j)$$

- We can show that it is true when the cost function C satisfies the following conditions for a ≤ b ≤ c ≤ d:
  - C(b,c) ≤ C(a,d);
  - C(a,c)+C(b,d) ≤ C(a,d)+C(b,c) [**Quadrangle Inequality**]

Knuth's Optimization

- Let's process the dp states in such a way that we calculate `dp(i,j-1)` and `dp(i+1,j)` before `dp(i,j)`, and in doing so we also calculate `opt(i,j-1)` and `opt(i+1,j)`.
- Then, for calculating `opt(i,j)`, instead of testing values of k from i to j-1, we only need to test from `opt(i,j-1)` to `opt(i+1,j)`.
- To process (i,j) pairs in this order, it is sufficient to use nested for loops in which i goes from the maximum value to the minimum one and j goes from i+1 to the maximum value.

Knuth's Optimization: Complexity

$$\sum_{i=1}^{N} \sum_{j=i+1}^{N} [opt(i+1,j) - opt(i,j-1)]$$

$$\sum_{i=1}^{N} \sum_{j=i}^{N-1} [opt(i+1,j+1) - opt(i,j)]$$

$$\sum_{k=1}^{N} [opt(k,N) - opt(1,k)] = O(n^2)$$

Convex Hull Trick and Li Chao Tree: Motivation

- There are n cities.
- You want to travel from city 1 to city n by car.
- To do this, you have to buy some gasoline.
- It is known that a liter of gasoline costs $cost_k$ in the k^{th} city.
- Initially your fuel tank is empty and you spend one liter of gasoline per kilometer.
- Cities are located on the same line in ascending order with k^{th} city having coordinate x_k.
- Also you have to pay $toll_k$ to enter k^{th} city.
- Your task is to make the trip with minimum possible cost.

Convex Hull Trick and Li Chao Tree: Motivation

- Need to solve:

$$dp_i = toll_i + \min_{j<i}(cost_j \cdot (x_i - x_j) + dp_j)$$

- Naive approach will give you $O(n^2)$ complexity.

Convex Hull Trick and Li Chao Tree: Motivation

- This can be improved to **O(n log n)**.
  - The problem can be reduced to:
    - Adding linear functions kx + b to the set
    - Finding minimum value of the functions in some particular point x.

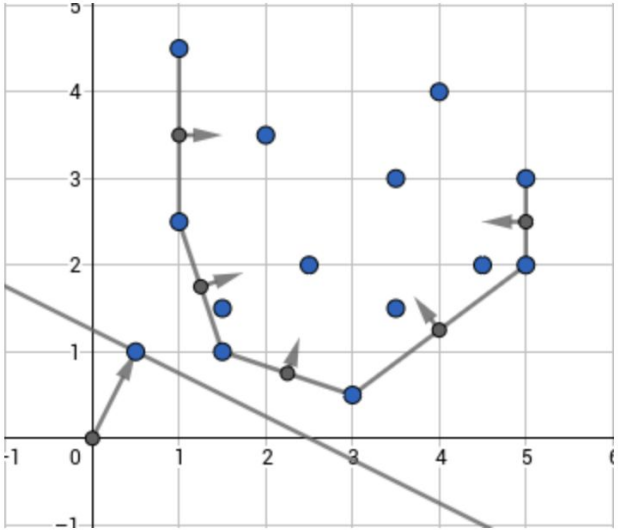
- There are two main approaches one can use here:
  - **Convex Hull Trick**
  - **Li Chao Tree**

Convex Hull Trick: Main Idea

- **Maintain a lower convex hull of linear functions.**
- It would be a bit more convenient to consider them not as linear functions, but as points $(k; b)$ on the plane such that we will have to find the point which has **the least dot product** with a given point $(x; 1)$; that is, for this point, $kx+b$ is minimized. This is the same as initial problem.

Convex Hull Trick: Main Idea

- Such minimum will necessarily be on lower convex envelope of these points as can be seen below:



Convex Hull Trick: Main Idea

- **Keep points on the convex hull and normal vectors of the hull's edges.**
- When you have a $(x;1)$ query, you'll have to find the normal vector closest to it in terms of angles between them, then the optimum linear function will correspond to one of its **endpoints**.
  - Points having a constant dot product with $(x;1)$ lie on a line which is orthogonal to $(x;1)$
  - So, the optimum linear function will be the one in which tangent to convex hull which is collinear with normal to $(x;1)$ touches the hull.

Li Chao Tree: Main Idea

- Assume you are given a set of functions such that each two can intersect at most once.
- Let's **keep in each vertex of a segment tree** *some function* in such way, that if we go from root to the leaf it will be guaranteed that one of the functions we met on the path will be the one giving the minimum value in that leaf.
- Let’s construct this segment tree!

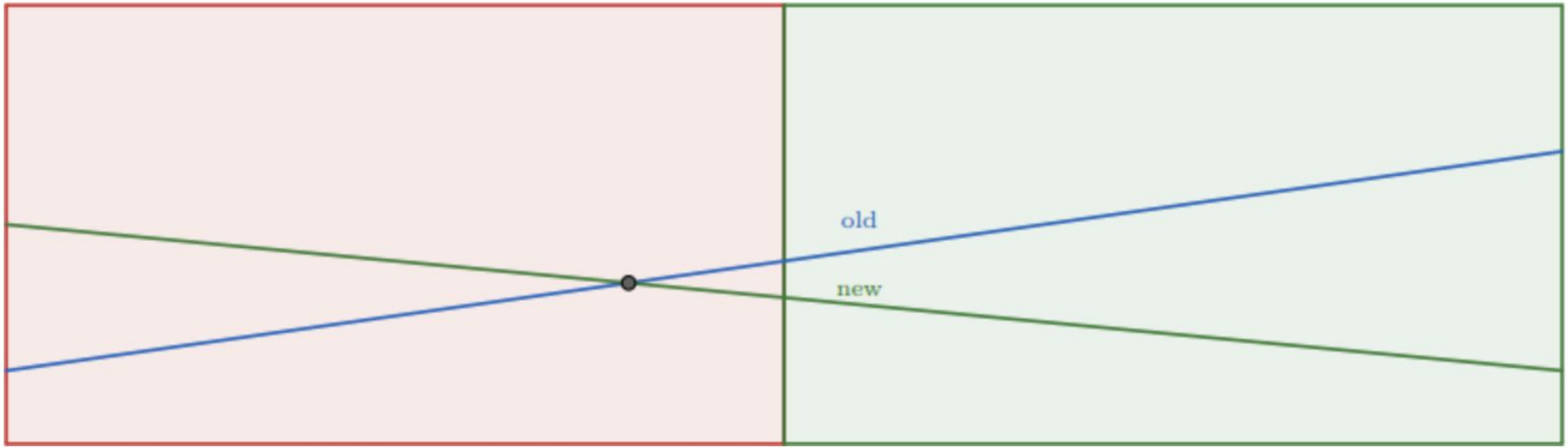
Li Chao Tree: Main Idea

- Assume we're in some vertex corresponding to half-segment **[l,r)** and the function $\mathbf{f_{old}}$ is kept there and we add the function $\mathbf{f_{new}}$.
- Then the **intersection point** will be **either in [l;m) or in [m;r)** where m is the midpoint of l and m.
- We can efficiently find that out by comparing the values of the functions in points l and m.

Li Chao Tree: Main Idea

- If the **dominating function changes**, then it is in [l;m) otherwise it is in [m;r).
- For the half of the segment with no intersection, we will pick the lower function and write it in the current vertex.
- You can see that it will always be the **one which is lower in point m.**
- After that, we **recursively** go to the other half of the segment with the function which was the upper one.

Li Chao Tree: Illustration



Summary

Name	Recurrence	Condition	Reduction
Divide & Conquer	$dp[i][j] = \min_{k<j} (dp[i-1][k] + C[k][j])$	for $a \le b \le c \le d$, $C[a][c]+C[b][d] \le C[a][d]+C[b][c]$	$kn^2 \Longrightarrow kn \log n$
Knuth	$dp[i][j] = \min_{i<k<j} (dp[i][k] + dp[k+1][j] + C[i][j])$	for $a \le b \le c \le d$, $C[b][c] \le C[a][d]$ $C[a][c]+C[b][d] \le C[a][d]+C[b][c]$	$n^3 \Longrightarrow n^2$
Convex Hull 1	$dp[i] = \min_{j<i} (dp[j] + b[j]\ a[i])$	$b[j] \ge b[j+1]$ $a[i] \le a[i+1]$	$n^2 \Longrightarrow n$
Convex Hull 2	$dp[i][j] = \min_{k<j} (dp[i-1][k] + b[k]\ a[j])$	$b[k] \ge b[k+1]$ $a[j] \le a[j+1]$	$kn^2 \Longrightarrow kn$

Practice #1: Ciel and Gondolas (321E)

Practice #2: Partition Game (1527E)

Practice #3: Trucks and Cities (1101F)

Practice #4: Kalila and Dimna... (319C)

Practice #5: Product Sum (631E)

Additional Practice Problems

- https://codeforces.com/problemset/problem/673/E
- https://codeforces.com/problemset/problem/834/D
- https://codeforces.com/contest/868/problem/F
- https://onlinejudge.org/external/100/10003.pdf
- https://onlinejudge.org/external/120/12057.pdf
- https://onlinejudge.org/external/103/10304.pdf
- https://codeforces.com/contest/932/problem/F
- https://codeforces.com/problemset/problem/660/F

Contact Information

- **Email**: yongwhan.lim@columbia.edu
- **Personal Website**: https://www.yongwhan.io/
- **LinkedIn Profile**: https://www.linkedin.com/in/yongwhan/
  - Feel free to send me a connection request!
  - Always happy to make connections with awesome people like yourself! 🙂


THANK YOU