HKOI Training 2018/19
Dynamic Programming (III)

Alex Tung

13 April 2019

Today’s Topics

- Ugly state DP
- DP Optimization

Ugly state DP

- We have seen “nicer” DP problems
  - For example, count number of paths from (1,1) to (N,M)

```
if (i,j) is blocked
   dp[i][j] = 0;
else
   dp[i][j] = dp[i-1][j] + dp[i][j-1]
```

- But the state representation is not always nice!
  - For example, bitmask DP

NOI 2001 炮兵陣地

- Given a N*M grid (N <= 100, M <= 10)
- Some cells are hills (‘H’), some plains (‘P’)
- You can place artillery (炮兵) only in ‘P’ cells
- Artillery squads cannot attack each other
- Their attack range:

```
..x..
..x..
xxPxx
..x..
..x..
```

NOI 2001 炮兵陣地

- Your task is to find the maximal number of artillery squads that may be placed
- Notice that M is small
- How to DP?

Analysis

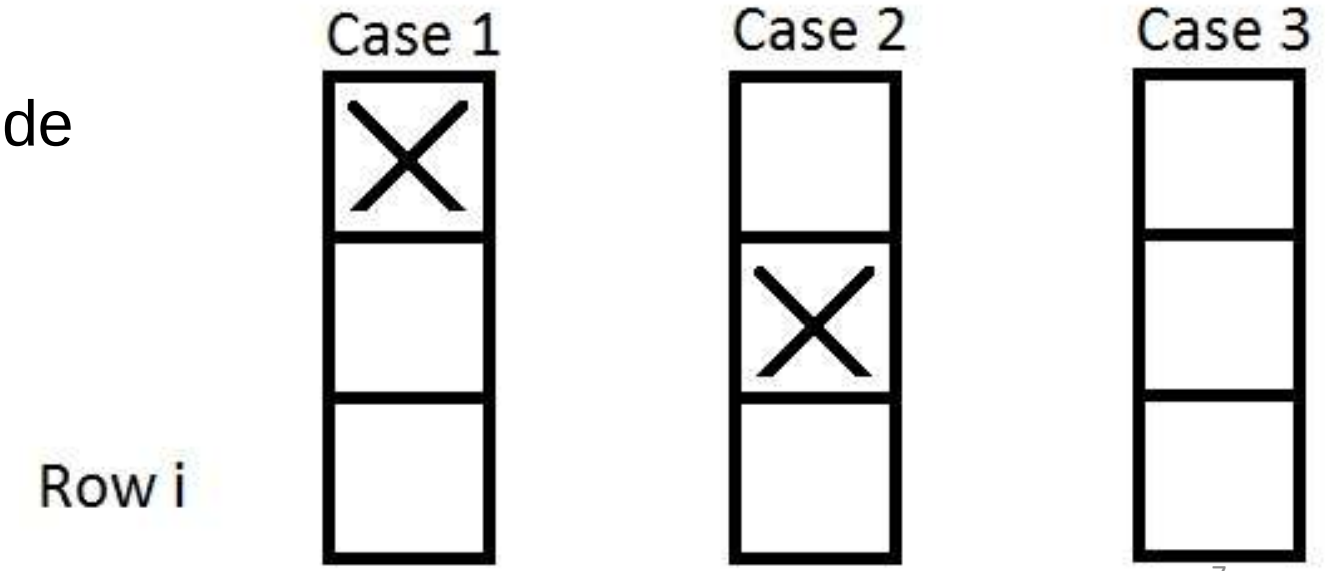
- Proceed row by row
- Assume the squads on rows 1..(i-1) are fixed

- One can place a squad in cell (i, j) iff:
  - [Previous rows] Cells (i-1, j) and (i-2, j) are empty
  - [Current row] Cells (i, j-2), (i, j-1), (i, j+1) and (i, j+2) are empty
  - [Current cell] Cell (i, j) is ‘P’

Representing the states

[Previous rows]

- We see that squad placement on rows 1 ... (i-3) does not affect row i
- Only need to know the “state” of the previous two rows
- Three possibilities
  - Use base-3 integers to encode
  - 0 = Case 1
  - 1 = Case 2
  - 2 = Case 3



State transition

[Current row]

- When we do state transition, we select a subset of {1, 2, ..., M} to put squads on
- To prevent attacking each other, make sure adjacent values differ by at least 3
- {1, 4, 8} is ok, {} is ok, while {1, 5, 7} is not

Algorithm

```
for i from 1 to N
 for j from 0 to 3^M – 1
  for all valid row patterns  //e.g. {1, 4, 8}
     if (all corresponding positions are ‘P’) and
       (none attacked by squads on previous rows)
      j’ = new state
      c = size of row pattern  //{1, 4, 8} → c = 3
      dp[i][j’] = max(dp[i][j’], dp[i-1][j] + c)
```

Summary

- Time complexity: $O(3^M * N * (\#\text{ row patterns}))$
- Implementation details may be complicated
- If you want to submit:
  - http://wcipeg.com/problem/noi01p2
  - http://poj.org/problem?id=1185

  (both require registration)

Further optimization

- Skip the state if dp[i-1][state] = -INF

- Not many states are reachable due to the restriction on same row placement
- e.g. 201210_3 is not reachable

With Optimization

```
for i from 1 to N
 for j from 0 to 3^M – 1
  if dp[i-1][j] = -INF
    continue
  for all valid row patterns  //e.g. {1, 4, 8}
     if (all corresponding positions are ‘P’) and
       (none attacked by squads on previous rows)
      j’ := new state
      c := size of row pattern  //{1, 4, 8} → c = 3
      dp[i][j’] := max(dp[i][j’], dp[i-1][j] + c)
```

Practice Problems (Ugly State DP)

- CF342D Xenia and Dominoes
- HKOJ T112 Tetrisudoku

Next: DP Optimization

- Monotone Queue
- Convex Hull Trick
- Divide and Conquer

Why Optimize?

- Suppose you have a correct DP formula (state definition, state transition)
- Sometimes, the DP may not run in time!
- Four main ways to solve:
  - Explore non-DP solutions
  - Write auxiliary DPs (DP2[][], DP3[][], ...) to speed up
  - Come up with alternative DP formulas
  - Optimize DP transition → What we will explore today

Main Idea

Monotonicity

DP Optimization

- Monotone Queue

Optimize using Monotone Queue

- The basic form of DP formula:

  $dp[i] = max_{L(i)<=j<i} (dp[j]) + f(i)$

- L(i) is <u>increasing</u>
  - Side note: If L(i) is not increasing, use segment tree to store, retrieve, and update dp values

- May replace dp[j] by any function depending on j
  - e.g. g(j) = dp[j] * 2 – j

Optimize using Monotone Queue

- Naïve implementation: $O(N^2)$

```
for i from 1 to N
    dp[i] = -INF
    for j from L(i) to i - 1
      dp[i] = max(dp[i], f(i) + g(j))
```

- We can do better!

Bowling for Numbers ++

- CCC 2007 Stage 2 Problem

- You have N (N<=10000) bowling pins and K (K<=500) bowling balls, each ball has width w (w<=100)
- Each pin has a score s[i] from -10000 to 10000
- You are allowed to miss

- Maximum achievable score = ?

Sample I/O

Sample (N = 9, K = 4, w = 3)

```
2 8 -5 3 5 8 4 8 -6
X X -5 3 5 8 4 8 -6  (ball 1, score = 10)
_ _ -5 X X X 4 8 -6  (ball 2, score = 26)
_ _ -5 _ _ X X X -6  (ball 3, score = 38)
_ _ -5 _ _ _ _ _ -6  (ball 4, score = 38)
```

- Answer = 38

A Simpler Version

- What if all pins have non-negative values?
  - Then, it is always better to hit more pins than to miss

  - dp[i][j]: max. value if we use i balls to hit pins 1...j
  - dp[i][j] = max(dp[i][j-1], dp[i-1][j-W] + sum(s[j-W+1]...s[j])) //roughly

  - O(NK)
  - CCC 2007 Stage 1 Senior Q5

But...

- Unfortunately, such is not the case in the current task
- We may sometimes want to “avoid” negative values

DP Formulation

- Let ps[i] = s[1] + s[2] + ... + s[i] (Partial sum of pin values)
- Let dp[i][j] be the maximum achievable score with i hits and the rightmost hit pin being j
- dp[0][0] = 0
- dp[0][i] = -INF for i > 0
  - Otherwise, error --- for example, a negative pin may be “skipped” without cost

Two cases

- Let dp[i][j] be the maximum achievable score with i hits and the rightmost hit pin being j
- We assume the hits are from left to right

- Two cases:
  - Case 1. The i^{th} hit does not overlap with the $(i-1)^{th}$ hit
  - Case 2. The i^{th} hit overlaps with the $(i-1)^{th}$ hit

Transition formula

To calculate dp[i][j],

- Let M1 = $\max_{0<=k<j-w}$(dp[i-1][k] + ps[j] - ps[j-w])
  - Case 1
  - Use dp2[i-1][k] := max(dp[i-1][1], ..., dp[i-1][k])
- Let M2 = $\max_{j-w<=k<j}$(dp[i-1][k] + ps[j] - ps[k])
  - Case 2
  - Use monotone queue to optimize!

Then dp[i][j] = max(M1, M2)

Monotone Queue: Step by step

- Here is how the monotone queue (actually a deque, or doubly-ended queue) works.
- Recall the criteria:
  - $dp[i] = \max_{L(i)<=j<i}(f(i) + g(j))$
  - L(i) is increasing
- We maintain a deque Q[] of indices such that
  - Q[j] < Q[j+1]
  - g(Q[j]) >= g(Q[j+1])

for all j.

Convention

- We use an array Q[] and two pointers l and r to represent the deque
- Q[l] is the head of the deque
- Q[r] is the tail of the deque

- Deque is empty iff l = r + 1
- Initially, l := 1, r := 0 (i.e. deque is empty)

Monotone Queue: Step by step

- Step 1: Pop elements in the front that are “out of bounds”

```
while (l <= r) and (Q[l] < L(i))
  l++;
```

Monotone Queue: Step by step

- Step 2: Update answer using Q[l]

```
if (l <= r)
  dp[i] = f(i) + g(Q[l]);
```

Monotone Queue: Step by step

- Step 3: Pop elements at the back that have small values

```
while (l <= r) and (g(Q[r]) < g(i))
  r--;
```

Monotone Queue: Step by step

- Step 4: Insert i at the back

```
r++;
Q[r] = i;
```

Monotone Queue: Step by step

1. `while (l <= r) and (Q[l] < L(i))`
   `l++;`
2. `if (l <= r)`
   `dp[i] = f(i) + g(Q[l]);`
3. `while (l <= r) and (g(Q[r]) < g(i))`
   `r--;`
4. `r++;`
   `Q[r] = i;`

Summary

- Original time complexity: $O(N^2K)$ / $O(NKw)$
- Optimized time complexity: $O(NK)$

- Monotone queue optimization reduces DP time complexity by an order of N

- We can use rolling array to reduce memory complexity to O(N)

Final step before AC

N = 2, K = 1, w = 2

-1 1

- Answer = 1, but cannot be obtained from our dp...
- Solution: add (w – 1) copies of 0s at the end

Break;

DP Optimization

- Monotone Queue
- Convex Hull Trick

Convex Hull Trick (CHT)

- The earliest CHT task that I know is “Batch Scheduling” in IOI 2002
  - 11 contestants got full scores :o
- Other occurrences in big competitions:
  - APIO 2010 Commando
  - APIO 2014 Split the Sequence
  - IOI 2016 Aliens (60 points)
- Something a good IOI contestant should know :D

Convex Hull Trick (CHT)

- The basic form of DP formula:

  $dp[i] = max_{j<i} (dp[j] + f[i] * g[j])$

- Intuitively looks like y = mx + c, a line on the plane

- We may apply CHT if g is monotone
- If f is also monotone, then it will be even easier

CF189C Kalila and Dimna in the Logging Industry

- Problem reduces to:
- Given N, a[i], b[i], find indices p_1, ..., p_k such that $p_1 = 1$, $p_k = N$, $p_i < p_{i+1}$ for all i, and sum($a[p_{i+1}] * b[p_i]$) is minimal. Output that minimal sum.

- Important constraints:
  - a[] is (strictly) increasing
  - b[] is (strictly) decreasing

Sample I/O

e.g. N = 6

a[] = {1, 2, 3, 10, 20, 30}

b[] = {6, 5, 4, 3, 2, 0}

- Choose indices 1, 2, 4, 6

  → sum = a[2]*b[1] + a[4]*b[2] + a[6]*b[4] = 152

- Choose indices 1, 3, 6

  → sum = a[3]*b[1] + a[6]*b[3] = 138,

  which is better (and, in fact, the best)

DP Formulation

- Let dp[i] = the best result obtainable by picking indices ending with i
- dp[1] = 0
- $dp[i] = \min_{j<i}(dp[j] + a[i] * b[j])$
  - Not max this time

- Naïve DP transition: $O(N^2)$
- Target: O(N) !!

Key idea of CHT

- Let us pick two indices j and k (1 <= **j < k** < i)
- When will we choose j instead of k? Or vice versa?

Key idea of CHT

- When will we choose j instead of k? Or vice versa?

k better than j

<=> dp[j] + a[i] * b[j] > dp[k] + a[i] * b[k]

<=> (dp[j] – dp[k]) / (b[j] – b[k]) > -a[i]

Does it look like $(y_2 – y_1) / (x_2 – x_1)$?

Let m(j, k) = (dp[j] – dp[k]) / (b[j] – b[k]) (“Slope” function)

Key idea of CHT

k better than j

<=> (dp[j] – dp[k]) / (b[j] – b[k]) > -a[i]

Does it look like $(y_2 – y_1) / (x_2 – x_1)$?

Let m(j, k) = (dp[j] – dp[k]) / (b[j] – b[k]) (“Slope” function)

(We assume j < k every time we write m(j, k).)

Key idea of CHT

k better than j <=> m(j, k) > -a[i]

Property 1: If m(j, k) < m(k, l), then there is no need to consider k.

- Reason: there are no cases where k must be chosen

- If m(j, k) > -a[i], then surely m(k, l) > -a[i]. Then l is even better than k
- If m(j, k) <= -a[i], then j is no worse than k

Key idea of CHT

k better than j <=> m(j, k) > -a[i]

Property 2: If m(j, k) > -a[i], there is no need to consider j in subsequent steps (steps i+1, ..., N).

- Reason: a is (strictly) increasing, so

  (-a[i]) > (-a[i’]) for any i’ > i

- Then m(j, k) > -a[i] > -a[i’]

Key idea of CHT

- **Property 1**: If j < k < l and m(j, k) < m(k, l), then there is no need to consider k.
- **Property 2**: If m(j, k) > -a[i], there is no need to consider j in subsequent steps (steps i+1, ..., N).

- What do these two properties give us?

Key idea of CHT

- **Property 1**: If j < k < l and m(j, k) < m(k, l), then there is no need to consider k.
- **Property 2**: If m(j, k) > -a[i], there is no need to consider j in subsequent steps (steps i+1, ..., N).

- Property 1 means that we only need to maintain a deque Q[L..R] such that m(Q[i], Q[i+1]) >= m(Q[i+1], Q[i+2])

Key idea of CHT

- **Property 1**: If j < k < l and m(j, k) < m(k, l), then there is no need to consider k.
- **Property 2**: If m(j, k) > -a[i], there is no need to consider j in subsequent steps (steps i+1, ..., N).

- Property 2 means that we can pop q[L] (front) from the deque, until m(Q[L], Q[L+1]) <= -a[i], to get the optimal answer

CHT: Step by step

- Step 1: Pop elements in the front that we will never use again [Property 2]

```
while (R-L >= 1) and (m(Q[L], Q[L+1]) > -a[i])
  L++;
```

CHT: Step by step

- Step 2: Update answer using Q[L]

```
if (L <= R)
  dp[i] = dp[Q[L]] + a[i] * b[Q[L]];
```

CHT: Step by step

- Step 3: Pop elements at the back that will never be considered [Property 1]

```
while (R-L >= 1) and (m(Q[R-1], Q[R]) < m(Q[R], i))
  R--;
```

CHT: Step by step

- Step 4: Insert i at the back

```
R++;
Q[R] = i;
```

CHT: Step by step

1. `while (R-L >= 1) and (m(Q[L], Q[L+1]) > -a[i])`
   `L++;`
2. `if (L <= R)`
   `dp[i] = dp[Q[L]] + a[i] * b[Q[L]];`
3. `while (R-L >= 1) and (m(Q[R-1], Q[R]) < m(Q[R], i))`
   `R--;`
4. `R++;`
   `Q[R] = i;`

Summary

- Notice that CHT (at least in this problem) is simply a variant of monotone queue optimization
- The monotonicity does not lie in the values itself, but in the “slope function”

- Time complexity: O(N)

CHT: Further complications

- $dp[i] = max_{j<i} (dp[j] + f[i] * g[j])$

- f/g = i/i, i/d, d/i, d/d, n/i, n/d
  - i: increasing, d: decreasing, n: neither
  - i/i and d/d are not interesting
  - For n/i and n/d, property 2 does not hold; **need binary search**
- Transition formula: max, min
- You have to write down the condition for “k better than j” and do the algebra correctly

CHT: Further complications

- When g is not strictly monotone (i.e. may have same values), direct computation of slope formula will give division by 0

- Also, using double for slope calculation may sometimes result in precision error. Use integer whenever possible.

DP Optimization

- Monotone Queue
- Convex Hull Trick
- Divide and Conquer

Divide and Conquer (D&C) Optimization

- The basic form of DP formula:

  $dp[i][j] = max_{k<j} (dp[i-1][k] + f(k, j))$

- Let C[i][j] be the smallest index k' such that

  dp[i][j] = dp[i-1][k'] + f(k', j)
- In other words, the transition from (i-1, k') to (i, j) is optimal among all choices of k
- We can apply D&C Optimization if C[i][j] <= C[i][j+1] for all j

D&C Optimization

- C[i][j] <= C[i][j+1] for all j

- Another form of *monotonicity*!

CF321E Ciel and Gondolas

- Given N, G, and an NxN symmetric matrix s[][] containing values from 0 to 9
- s[i][i] = 0 for all i
- Divide [1, N] into G disjoint groups $[1, a_1]$, $[a_1 + 1, a_2]$, ..., $[a_{G-1} + 1, a_G]$ ($a_G = N$)
- For each group [L, R], calculate sum(s[i][j] | L <= i, j <= R)
- Add them up to get the total *cost* of this partition
- Find the minimal cost

Sample I/O

5 2 (N = 5, G = 2)

0 0 1 1 1

0 0 1 1 1

1 1 0 0 0

1 1 0 0 0

1 1 0 0 0

- Answer = 0 ($a_1 = 2$, $a_2 = 5$)

DP Formulation

- Let dp[i][j] = minimal cost of partitioning [1, j] into i groups
- Let f(L, R) = sum(s[i][j] | L <= i, j <= R)

- $dp[i][j] = \min_{k<j} (dp[i-1][k] + f(k+1, j))$

- Answer = dp[G][N]

CF321E Ciel and Gondolas

- $dp[i][j] = \min_{k<j} (dp[i-1][k] + f(k+1, j))$
- $f(k+1, j)$ can be calculated in $O(1)$ by 2D partial sum
- Naïve solution has time complexity $O(GN^2)$
- Target: $O(GN \log N)$

CF321E Ciel and Gondolas

- Recall the condition for applying D&C Optimization:

- Let C[i][j] be the smallest index k’ such that dp[i][j] = dp[i-1][k’] + f(k’, j).
- **Check that C[i][j] <= C[i][j+1] for all j**

(See appendix)

D&C Optimization: Step by step

- The key idea is to write a recursive function to perform the DP transition

```
void solve(int i, int L, int R, int optL, int optR);
```

- The above function calculates dp[i][L..R], knowing that C[i][j] is between optL and optR for L <= j <= R

Idea:

- Let M = (L+R)/2
- Find dp[i][M] and C[i][M]. Then call solve() for the left and the right parts.

D&C Optimization: Step by step

- Step 1: Base case

```
if (L > R) return;
```

D&C Optimization: Step by step

- Step 2: Find dp[i][M] and C[i][M]

```
int opt = optL;       //opt represents C[i][M]
for(p = optL + 1; p <= optR; p++)
  if(dp[i-1][p] + f(p+1, M) < dp[i-1][opt] + f(opt+1, M))
    opt = p;
```

(For maximization problems, change “<" to “>")

D&C Optimization: Step by step

- Step 3: Update dp[i][M]

```
dp[i][M] = dp[i-1][opt] + f(opt+1, M);
```

D&C Optimization: Step by step

- Step 4: Recursively solve the left and right parts

```
solve(i, L, M – 1, optL, opt);
solve(i, M + 1, R, opt, optR);
```

Here, The condition C[i][j] <= C[i][j + 1] is used to narrow the range of candidate transitions from [optL, optR] to [optL, opt] and [opt, optR] respectively.

D&C Optimization: Step by step

```
void solve(int i, int L, int R, int optL, int optR){
1. if (L > R) return;
2. int opt = optL;       //opt represents C[i][M]
   for(p = optL + 1; p <= optR; p++)
     if(dp[i-1][p] + f(p+1, M) < dp[i-1][opt] + f(opt+1, M))
       opt = p;
3. dp[i][M] = dp[i-1][opt] + f(opt+1, M);
4. solve(i, L, M – 1, optL, opt);
   solve(i, M + 1, R, opt, optR);
}
```

Back to the task...

- Set dp[0][0] = 0 and dp[0][i] = INF for i > 0
- Call solve(i, 1, N, 1, N) for i = 1, ..., G

- It can be shown that each solve() runs in time complexity O(N log N)

- Overall time complexity: O(GN log N)

Minor details

- Use rolling array for DP calculation
- Huge input (4000^2 numbers), need fast I/O methods to get AC

Practice Problems (DP Opt)

- CF311B Cats Transport
- CF660F Bear and Bowling 4
- Hackerrank Guardians of the Lunatics
- APIO 2010 Commando
- APIO 2014 Splitting the Sequence

...and more in the CF blog mentioned in reference

Other Interesting DP Problems

- CF 590D Top Secret Task
- CF 489E Hiking
- HKOJ M1331 Resources Conflict
- HKOJ M1724 Guess the Number
- HKOJ M1741 Fill in the Bag

References

- Tasks from HKOJ, Codeforces, CCC, NOI
- http://codeforces.com/blog/entry/8219
  - A summary of different types of DP Optimization

Model Solutions (Updated)

- Bowling for Numbers ++
  - https://ideone.com/D2LQmj
- Kalila and Dimna in the Logging Industry
  - https://ideone.com/Y65oHV
- Ciel and Gondolas
  - https://ideone.com/ZQ7pmY

The End

- PM Session: Mini-Competition IV