Dynamic Programming (III)

Alex Tung

14 April 2018

Today’s highlights

- Ugly state DP
- DP Optimization

- Note: Some materials from DP (III) in 2015 are not covered in this lesson. You are encouraged to study them yourself.

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?

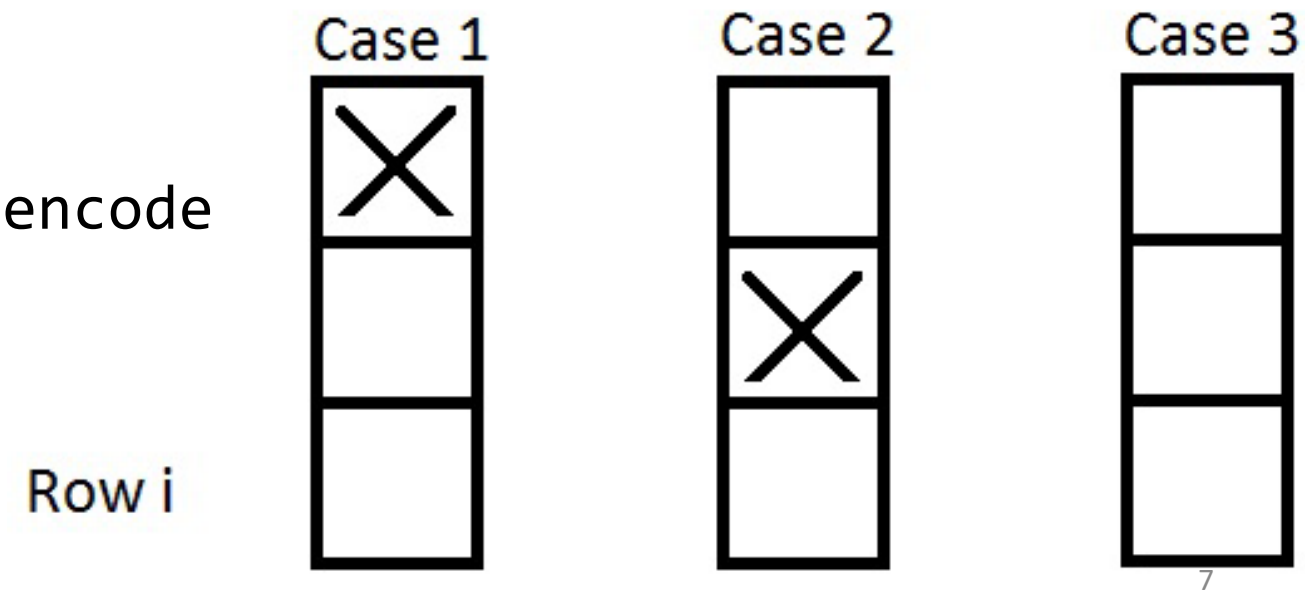
Analysis

- Proceed row by row
- Assume the squads on row 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 * (# 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, of course)

Further optimization

- Skip the state if dp[i-1][state] = NINF
- Not many states are reachable due to the restriction on same row placement
- e.g. 221210_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
  - <u>Optimize DP transition</u> → What we will explore today

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 = ?

Bowling for Numbers ++

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

Bowling for Numbers ++

- 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

Bowling for Numbers ++

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

DP State definition

- Let ps[i] = s[1] + s[2] + ... + s[i]
  - Partial sum
- Let dp[i][j] be the maximum achievable score with i hits and the <u>rightmost</u> 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 <u>rightmost</u> hit pin being j
- We assume the hits are from left to right

- Two cases:
  - Case 1. The ith hit does not overlap with the $(i-1)^{th}$ hit
  - Case 2. The ith 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) works.
- Recall the criteria:
  - $dp[i] = \max_{L(i)<=j<i}(f(i) + g(j))$
  - $L(i)$ is increasing
- We maintain a queue Q[] of indices such that
  - $Q[j] < Q[j+1]$
  - $g(Q[j]) >= g(Q[j+1])$

  for all j.

Monotone Queue: Step by step

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

```
while (Q not empty) and (Q[l] < L(i))
  l++;
```

(Here, we use an array and two pointers, l and r, to represent the deque.)

Monotone Queue: Step by step

- Step 2: Update answer using Q[l]

```
if (Q not empty)
  dp[i] = f(i) + g(Q[l]);
```

Monotone Queue: Step by step

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

```
while (Q not empty) and (g(Q[r]) < g(i))
  r--;
```

Monotone Queue: Step by step

- Step 4: Insert i at the back

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

Monotone Queue: Step by step

```
1. while (Q not empty) and (Q[l] < L(i))
    l++;
2. if (Q not empty)
    dp[i] = f(i) + g(Q[l]);
3. while (Q not empty) and (g(Q[r]) < g(i))
    r--;
4. 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 <u>an order of N</u>

- 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 get 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
- 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 R^2

- We may apply CHT if g is <u>monotone</u>
- 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 $p_1, \ldots, p_k$ such that $p_1 = 1$, $p_k = N$, $p_i < p_{i+1}$ for all i, and $\text{sum}(a[p_{i+1}] * b[p_i])$ is minimal. Output that minimal sum.

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

CF189C Kalila and Dimna in the Logging Industry

- 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)

CF189C Kalila and Dimna in the Logging Industry

- 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])

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])

(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 $i+1, \ldots, N$).

- 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 (q’s size >= 2) and (m(q[L], q[L+1]) > -a[i])
  L++;
```

CHT: Step by step

- Step 2: Update answer using q[L]

```
if (Q not empty)
  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 (q’s size >= 2) and (m(q[R-2], q[R-1]) < m(q[R-1], i))
  R--;
```

CHT: Step by step

- Step 4: Insert i at the back

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

CHT: Step by step

```
1. while (q’s size >= 2) and (m(q[L], q[L+1]) > -a[i])
    L++;
2. if (q not empty)
    dp[i] = dp[q[L]] + a[i] * b[q[L]];
3. while (q’s size >= 2) and (m(q[R-2], q[R-1]) < m(q[R-1], i))
    R--;
4. 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 b-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

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] | i and j are between L and R)
- Add them up to get the total *cost* of this partition
- Find the minimal cost

CF321E Ciel and Gondolas

- 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)

CF321E Ciel and Gondolas

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

- 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;
```

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);
```

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

- Bowling for Numbers ++
  - http://ideone.com/tY1Kk3
- Kalila and Dimna in the Logging Industry
  - http://ideone.com/y47MiU
- Ciel and Gondolas
  - http://ideone.com/7Mpq2o

The End

- PM Session: Mini-Competition III
- **Group** mini-comp, ACM style! :D

Appendix: Why $C[i][j] \leq C[i][j+1]$ in CF321E

Alex Tung

April 14, 2018

In today's DP (III) lecture, I promised to write a proof that D&C optimization can be used for solving Ciel and Gondolas. Here it is.

Recall that $dp[i][j] :=$ minimal cost of partitioning $[1, j]$ into i groups, and we have the simple transition formula

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

where $f(l, r)$ equals the sum of elements in the subarray $s[l..r][l..r]$.

1 The First Step

To simplify notations a bit, assume we are transitioning from $old[0..n]$ to $dp[0..n]$ (instead of from $dp[i-1][0..n]$ to $dp[i][0..n]$, and write $C[j]$ for $C[i][j]$. (All this is done to get rid of the somewhat irrelevant index i.)

So we want to prove $C[j] \leq C[j+1]$, where $C[j]$ is the smallest index k minimizing $old[k] + f(k+1, j)$. It is usually a good idea to prove *by contradiction*: suppose $C[j] > C[j+1]$, then there must be something wrong.

2 Inequalities

Here are two simple inequalities.

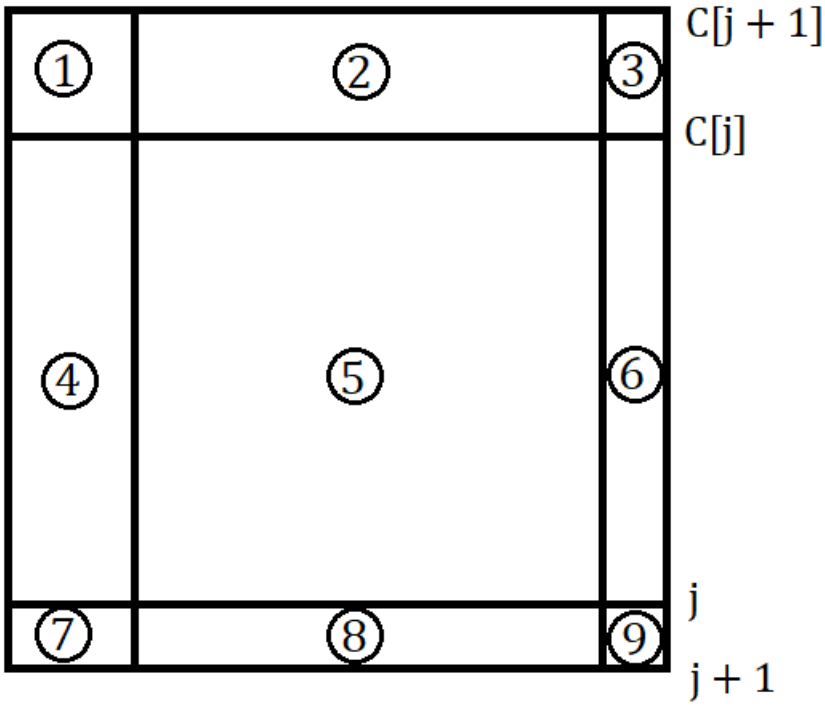
- $old[C[j]] + f(C[j]+1, j) < old[C[j+1]] + f(C[j+1]+1, j)$
  ($\because C[j]$ is the best index for $dp[j]$.)
- $old[C[j+1]] + f(C[j+1]+1, j+1) \leq old[C[j]] + f(C[j]+1, j+1)$
  ($\because C[j+1]$ is the best index for $dp[j+1]$.)

Add them up to get

$$f(C[j]+1, j) + f(C[j+1]+1, j+1) < f(C[j]+1, j+1) + f(C[j+1]+1, j).$$

3 Why is this absurd?

This is (almost) a proof without words:



So our assumption $C[j] > C[j+1]$ has led us to the inequality $f(C[j]+1, j) + f(C[j+1]+1, j+1) < f(C[j]+1, j+1) + f(C[j+1]+1, j)$. But...

$$LHS = 5 + (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9)$$

$$RHS = (1 + 2 + 4 + 5) + (5 + 6 + 8 + 9)$$

$$\therefore LHS - RHS = 3 + 7 \geq 0. \text{ Contradiction!}$$