Divide and Conquer DP Optimization

Firman Hadi P. – TKTPL Genap 2018/2019

Fakultas Ilmu Komputer, Universitas Indonesia

28 Februari 2019

Outline

SPOJ LARMY - Lannister Army

Diberikan sebuah barisan N ($1 \leq N \leq 5000$) orang tentara. Tentara ke-i mempunyai tinggi H_i ($1 \leq H_i \leq 10^5$). Anda harus memecah barisan tersebut menjadi K buah subbarisan ($1 \leq K \leq N$) yang *non-overlapping* sehingga:

- Setiap barisan harus berisi setidaknya satu orang tentara.
- Harga dari suatu subbarisan $[l, r]$ adalah banyaknya pasangan (i, j) sehingga $l \leq i < j \leq r$ dan $H_i > H_j$.

Harga dari suatu konfigurasi yang valid adalah jumlah dari seluruh harga K subbarisan tersebut. Hitunglah harga minimum yang dapat dicapai oleh suatu konfigurasi!

Solusi

Misalkan $f(i, j)$ adalah harga minimum yang bisa dicapai apabila kita ingin memecah tentara ke-1 (*1-based indexing*) sampai ke-j menjadi i buah subbarisan.

Solusi

Misalkan $f(i, j)$ adalah harga minimum yang bisa dicapai apabila kita ingin memecah tentara ke-1 (*1-based indexing*) sampai ke-j menjadi i buah subbarisan.

$$f(i,j) = \begin{cases} \infty & \text{jika } i > 0 \text{ dan } j = 0 \\ \infty & \text{jika } i = 0 \text{ dan } j > 0 \\ 0 & \text{jika } i = 0 \text{ dan } j = 0 \\ \min\limits_{k<j}\{f(i-1,k) + C(k+1,j)\} & \text{jika } i > 0 \text{ dan } j > 0 \end{cases}$$

Dengan $C(i, j)$ merupakan harga dari sebuah barisan yang berisi tentara ke-i sampai ke-j.

Solusi

Misalkan $f(i,j)$ adalah harga minimum yang bisa dicapai apabila kita ingin memecah tentara ke-1 (*1-based indexing*) sampai ke-j menjadi i buah subbarisan.

$$f(i,j) = \begin{cases} \infty & \text{jika } i > 0 \text{ dan } j = 0 \\ \infty & \text{jika } i = 0 \text{ dan } j > 0 \\ 0 & \text{jika } i = 0 \text{ dan } j = 0 \\ \min\limits_{k<j}\{f(i-1,k) + C(k+1,j)\} & \text{jika } i > 0 \text{ dan } j > 0 \end{cases}$$

Dengan $C(i,j)$ merupakan harga dari sebuah barisan yang berisi tentara ke-i sampai ke-j.

Bagian rekursi dari $f(i,j)$ mempunyai maksud: "*pisahkan barisan dari orang ke-*$(k+1)$ *sampai orang ke-*j*!*"

Kita dapat *precompute* nilai $C(i, j)$ untuk setiap (i, j) dalam waktu $O(N^2)$.

Jawaban terdapat pada $f(K, N)$.

Kita dapat *precompute* nilai $C(i, j)$ untuk setiap (i, j) dalam waktu $O(N^2)$.

Jawaban terdapat pada $f(K, N)$.

Kompleksitas memori: $O(KN)$
Kompleksitas waktu: $O(N^2 + KN^2)$?

Kita dapat *precompute* nilai $C(i,j)$ untuk setiap (i,j) dalam waktu $O(N^2)$.

Jawaban terdapat pada $f(K,N)$.

Kompleksitas memori: $O(KN)$
Kompleksitas waktu: $O(N^2 + KN^2)$? TLE!

Mempercepat perhitungan DP...

Observasi

Misalkan $opt(i, j) =$ nilai k sehingga $f(i, j)$ minimum. Perhatikan bahwa $opt(i, j) \leq opt(i, j + 1)$.

Intuisi: 'titik potong optimal' barisan yang lebih pendek berada setidaknya lebih kiri atau tepat di titik potong optimal pada barisan yang lebih panjang.

Mempercepat perhitungan DP...

Observasi

Misalkan $opt(i,j) =$ nilai k sehingga $f(i,j)$ minimum. Perhatikan bahwa $opt(i,j) \leq opt(i,j+1)$.

Intuisi: 'titik potong optimal' barisan yang lebih pendek berada setidaknya lebih kiri atau tepat di titik potong optimal pada barisan yang lebih panjang.

Bagaimana cara memanfaatkan observasi ini?

Mempercepat perhitungan DP...

Bayangkan kita sedang mengisi tabel DP $dp[i][j] = f(i, j)$ secara *bottom-up* untuk setiap baris dan setiap kolom.

Setelah kita menghitung $dp[i][j]$, kemudian kita ingin menghitung $dp[i][j+1]$, apakah kita harus menghitung untuk setiap k?

Mempercepat perhitungan DP...

Bayangkan kita sedang mengisi tabel DP $dp[i][j] = f(i, j)$ secara *bottom-up* untuk setiap baris dan setiap kolom.

Setelah kita menghitung $dp[i][j]$, kemudian kita ingin menghitung $dp[i][j+1]$, apakah kita harus menghitung untuk setiap k? Tidak!

Mempercepat perhitungan DP...

Bayangkan kita sedang mengisi tabel DP $dp[i][j] = f(i, j)$ secara *bottom-up* untuk setiap baris dan setiap kolom.

Setelah kita menghitung $dp[i][j]$, kemudian kita ingin menghitung $dp[i][j+1]$, apakah kita harus menghitung untuk setiap k? Tidak!

Kita cukup memulai k dari $opt(i, j)$ saja!

Mempercepat perhitungan DP...

Pengisian tabel DP dari kiri ke kanan untuk setiap barisnya akan mempunyai kompleksitas waktu terburuk $O(N^2)$ bahkan dengan menggunakan observasi sebelumnya.

Mempercepat perhitungan DP...

Pengisian tabel DP dari kiri ke kanan untuk setiap barisnya akan mempunyai kompleksitas waktu terburuk $O(N^2)$ bahkan dengan menggunakan observasi sebelumnya.

Pengisian dari kanan ke kiri juga memberikan waktu yang sama.

Mempercepat perhitungan DP...

Pengisian tabel DP dari kiri ke kanan untuk setiap barisnya akan mempunyai kompleksitas waktu terburuk $O(N^2)$ bahkan dengan menggunakan observasi sebelumnya.

Pengisian dari kanan ke kiri juga memberikan waktu yang sama.

Tampaknya, pengisian dari tengah memberikan kompleksitas waktu yang lebih baik, yaitu $O(N \log N)$ untuk setiap baris!

Mempercepat perhitungan DP...

Algorithm $\textsc{FillTableRow}(dp, i, l, r, opt_l, opt_r)$

Input: dp = DP table, i = row number, (l, r) = column interval, (opt_l, opt_r) = possible optimal points interval

1: **if** $l > r$ **then**
2:     **exit procedure**
3: **end if**
4: $mid \leftarrow \left\lfloor \frac{L+R}{2} \right\rfloor$
5: $opt_{mid} \leftarrow$ optimal k for $f(i, mid)$ ▷ Try k in $[opt_l, opt_r]$
6: $dp[i][mid] = f(i, mid)$ ▷ Use $k = opt_{mid}$
7: **do** $\textsc{FillTableRow}(dp, i, l, mid - 1, opt_l, opt_{mid})$
8: **do** $\textsc{FillTableRow}(dp, i, mid + 1, r, opt_{mid}, opt_r)$
9: **exit procedure**

Mempercepat perhitungan DP...

Pengisian setiap baris tabel akan memberikan kompleksitas waktu $O(KN \log N)$ pada kasus terburuk.

Kompleksitas waktu akhir menjadi $O(N^2 + KN \log N)$ yang cukup untuk mendapatkan *verdict* ***accepted***.

Solusi akhir

Algorithm LARMY, time complexity: $O(N^2 + KN \log N)$

Input: N, K, N warriors with height $H[i]$

Output: Minimum possible total cost (unhappiness)

$dp \leftarrow K \times N$ array
$C \leftarrow N \times N$ array ▷ used for storing $C(i, j)$ values
precompute all $C(i, j)$ and store to C array ▷ see code for details
fill dp with basecase values
for each i in $\{1, 2, ...K\}$ **do**
    run $\textsc{FillTableRow}(dp, i, 1, N, 0, N-1)$
end for
return $dp[K][N]$

Yay!

23285961	2019-02-23 19:07:46	Firman Hadi P.	Lannister Army	**accepted** edit ideone it	2.93	110M	CPP14-CLANG

Outline

Bentuk rekurens

Biasanya, DP yang mempunyai transisi kurang lebih seperti berikut:

$$dp[i][j] = \min_{k \leq j}\{dp[i-1][k] + C(k,j)\},\ 1 \leq i \leq K,\ 1 \leq j \leq N$$

dengan $opt(i,j) \leq opt(i,j+1)$, atau lebih umumnya nilai fungsi $opt(i,j)$ monoton pada nilai i yang tetap, dapat dipercepat menggunakan optimisasi *divide and conquer*.

Bentuk rekurens

Biasanya, DP yang mempunyai transisi kurang lebih seperti berikut:

$$dp[i][j] = \min_{k \leq j}\{dp[i-1][k] + C(k,j)\},\ 1 \leq i \leq K,\ 1 \leq j \leq N$$

dengan $opt(i,j) \leq opt(i,j+1)$, atau lebih umumnya nilai fungsi $opt(i,j)$ monoton pada nilai i yang tetap, dapat dipercepat menggunakan optimisasi *divide and conquer*.

Optimisasi tersebut dapat mempercepat kompleksitas waktu dari $O(KN^2)$ menjadi $O(KN \log N)$, dengan asumsi setiap perhitungan $C(k,j)$ menggunakan waktu $O(1)$.

Teknik DnC

Menggunakan sifat monoton dari fungsi opt, kita dapat mempercepat pengisian pengisian tabel DP secara *bottom-up* dengan teknik *divide and conquer*.

Teknik yang dilakukan secara umum sama dengan algoritma $\textsc{FillTableRow}$ yang sudah dijelaskan sebelumnya.

Teknik DnC

Algorithm $\textsc{FillTableRow}(dp, i, l, r, opt_l, opt_r)$

Input: dp = DP table, i = row number, (l, r) = column interval, (opt_l, opt_r) = possible optimal points interval

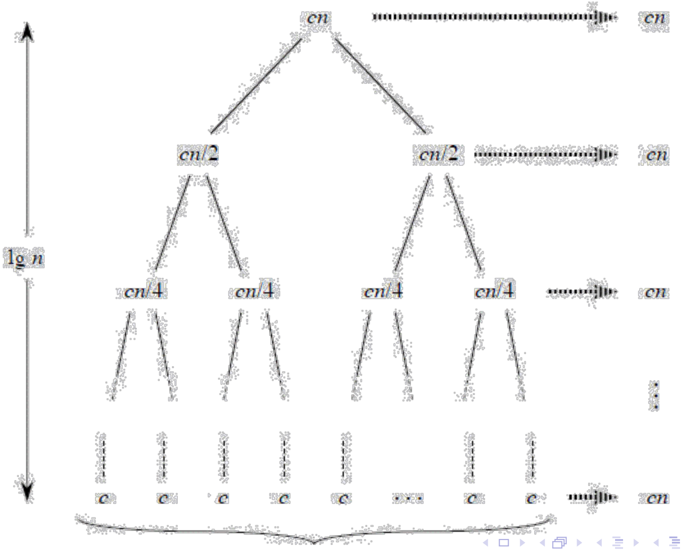
```
if l > r then
    exit procedure
end if
mid ← ⌊(L+R)/2⌋
opt_mid ← optimal k for f(i, mid)            ▷ Try k in [opt_l, opt_r]
dp[i][mid] = f(i, mid)                        ▷ Use k = opt_mid
do FillTableRow(dp, i, l, mid − 1, opt_l, opt_mid)
do FillTableRow(dp, i, mid + 1, r, opt_mid, opt_r)
exit procedure
```

Kompleksitas waktu

Untuk setiap pengisian baris tabel, kasus terburuknya adalah kita mendapatkan $opt(i, j) = j$ untuk setiap (i, j) sehingga untuk rekursi harus dilakukan pencarian nilai $opt(i, mid)$ sepanjang interval (l, r).

Tetapi, perhatikan bahwa kedalaman rekursi akan terbatas pada $\log N$, sehingga kompleksitas waktu akan tetap $O(N \log N)$.

Kompleksitas waktu (visualisasi)



Kompleksitas waktu (visualisasi)

$c =$ *cost* untuk komputasi $C(k, j)$, yaitu $O(1)$.

Jumlahan cn sebanyak $\log n = cn \log n = n \log n$.

Outline

Contoh yang lain

ACM-ICPC 2017 Asia Ho Chi Minh City - Famous Pagoda

Klik judul untuk melihat deskripsi soal.

Contoh yang lain

Observasi

- $1 \leq k \leq 2 \rightarrow$ pecah dua kasus untuk perhitungan *cost*
- $N \leq 2000 \rightarrow$ *precompute cost*
- Misalkan $f(i, j) =$ harga minimum untuk membangun i tangga untuk posisi ke-1 sampai ke-j...

Contoh yang lain

Observasi

- $1 \leq k \leq 2 \rightarrow$ pecah dua kasus untuk perhitungan *cost*
- $N \leq 2000 \rightarrow$ *precompute cost*
- Misalkan $f(i, j) =$ harga minimum untuk membangun i tangga untuk posisi ke-1 sampai ke-j...
  $opt(i, j) \leq opt(i, j + 1)$!

Contoh yang lain

Observasi

- $1 \leq k \leq 2 \rightarrow$ pecah dua kasus untuk perhitungan *cost*
- $N \leq 2000 \rightarrow$ *precompute cost*
- Misalkan $f(i, j) =$ harga minimum untuk membangun i tangga untuk posisi ke-1 sampai ke-j...
  $opt(i, j) \leq opt(i, j + 1)$!

Permasalahan ini dapat diselesaikan dengan DP ditambah optimisasi *divide and conquer*. Implementasi diserahkan kepada pembaca sebagai latihan :D

Kesimpulan

Beberapa transisi DP dapat dipercepat menggunakan teknik *divide and conquer*.

Kesimpulan

Beberapa transisi DP dapat dipercepat menggunakan teknik *divide and conquer*.

Dengan sedikit modifikasi, Anda dapat juga menggunakan teknik ini untuk mencari nilai maksimum.

Referensi

- cp-algorithms.com - Divide and Conquer DP
- Codeforces Blog - Dynamic Programming Optimizations
- StackOverflow - algorithms: how do divide-and-conquer and time complexity O(nlogn) relate?