고급 7회차
Divide and Conquer Optimization

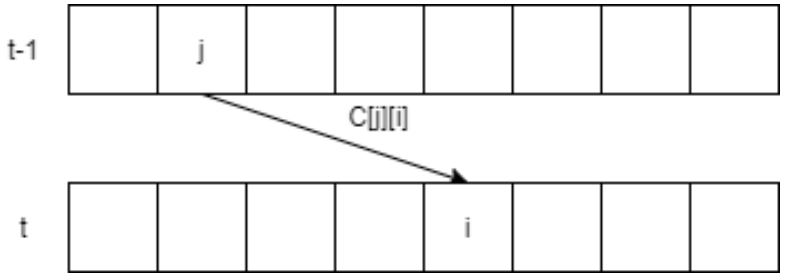
서강대학교 전해성(seastar105)

Target DP Recurrence Equation

$$dp[t][i] = \min_{j<i}(dp[t-1][j] + C[j][i])$$

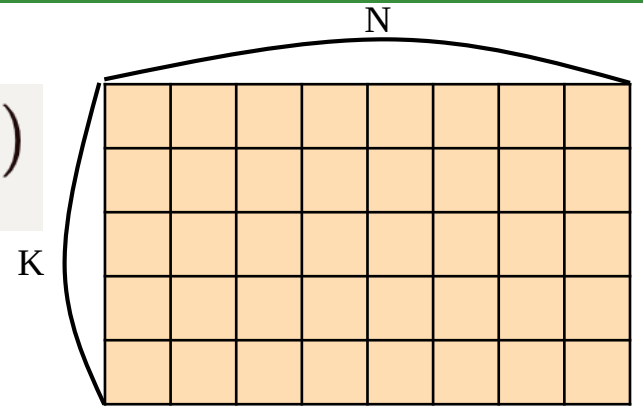
위 식은 아래와 같은 DP테이블을 채워나가는데 바로 이전 행의 정보를 이용해서 현재 행을 채워나갑니다.

그리고 이전 행의 j열을 이용해서 i열을 채운다면 C[j][i]만큼 추가로 비용이 든다는 것입니다.



Target DP Recurrence Equation Time Complexity

$$dp[t][i] = \min_{j<i}(dp[t-1][j] + C[j][i])$$



채워야 할 총 테이블의 크기가 K by N 이라고 합시다.

일반적으로 한 행을 채우는 데에 걸리는 시간복잡도는 $O(N^2)$이고 행의 개수가 K이므로 테이블을 전부 채우는 데에는 $O(KN^2)$이 걸립니다.

DnC opt는 특정 조건을 만족할 때 시간복잡도를 줄여주는 방법입니다.

Optimality

$$dp[t][i] = \min_{j<i}(dp[t-1][j] + C[j][i])$$

위 점화식에서 dp[t][i]를 최소로 만들어주는 j를 opt_i라고 하겠습니다. 그러면, 아래와 같은 조건이 성립할 때, 우리는 최적화를 진행할 수 있습니다.

$$opt_{i'} \leq opt_i \ \ for \ \ all \ \ i' < i$$

Optimality

$$opt_{i'} \leq opt_i \ \ for \ \ all \ \ i' < i$$

위 조건이 성립하면, 한 행을 채우는 시간복잡도를 O(NlogN)으로 줄일 수 있으며, 모든 행에 대해서 이 조건이 성립하면 O(KNlogN)에 모든 테이블을 채우는 것이 가능합니다.

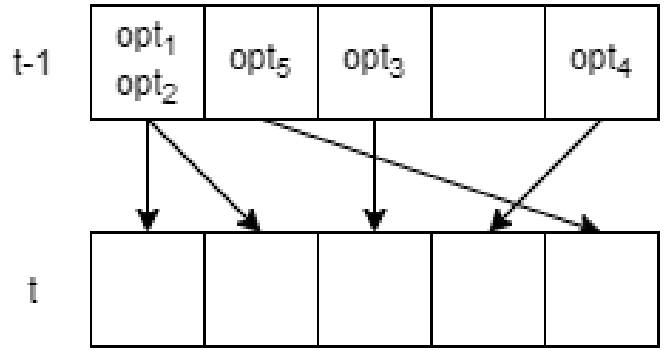
위 조건을 최적해의 단조성이라고 부르겠습니다.

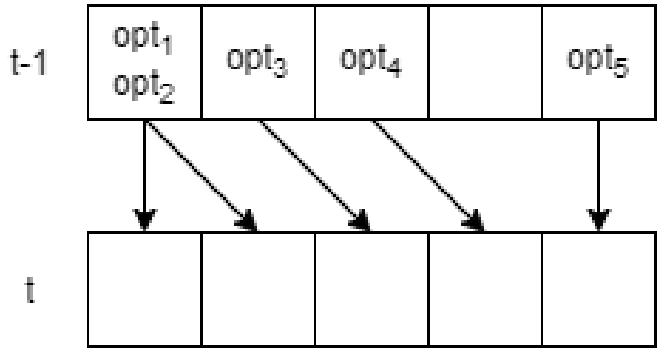
Monotonicity of Optimality

$$opt_{i'} \leq opt_i \quad for\ all\ i' < i$$

위 조건을 그림으로 나타내자면 아래와 같습니다.

왼쪽이 조건을 만족하지 못하는 상태이며, 오른쪽이 조건을 만족하는 상태입니다.





How Optimization Works

$$opt_{i'} \leq opt_i \ \ for\ all\ i' < i$$

최적해가 단조성을 가질 때, 실제로 어떻게 최적화가 이루어지는지 살펴보겠습니다.

구간 [s,e]에 대해서 DP테이블을 채운다고 합시다.

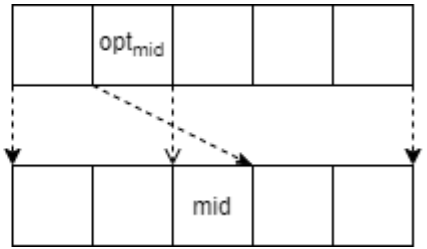
최적해가 단조성을 가지기 때문에 $opt_s <= opt_e$입니다.

How Optimization Works

$$opt_{i'} \leq opt_i \quad for\ all\ i' < i$$

[s,e]의 중간점인 mid=$\frac{s+e}{2}$에 대해 opt_{mid}를 구합니다. [opt_s,opt_e]를 전부 살펴보기 때문에 O(opt_e-opt_s)만큼 걸립니다.

그러면 [s,mid-1]에 대해서 opt_s <= opt_{mid-1} <= opt_{mid} 이며, [mid+1, e]에 대해서 opt_{mid} <= opt_{mid+1} <= opt_e가 성립합니다.

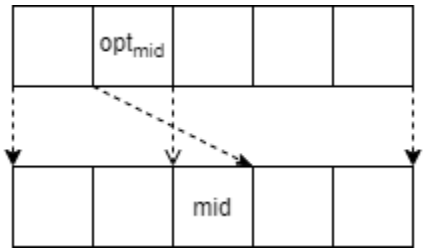


How Optimization Works

$$opt_{i'} \leq opt_i \quad for\ all\ i' < i$$

mid의 값을 구했으니 [s,e]를 [s, mid-1]과 [mid+1, e]로 쪼갭시다.

[s,e]에서 최적해의 탐색 구간은 [opt_s,opt_e]였습니다. [s, mid-1]에 대해서 최적해의 탐색구간이 [opt_s, opt_{mid}]로 바뀌고, [mid+1, e]에 대해서 최적해의 탐색구간이 [opt_{mid}, opt_e]로 바뀝니다.



Time Complexity

설명한 과정을 반복해 나감으로 [s,e]에서 모든 값을 구할 수 있습니다.

$[opt_s, opt_e]$의 길이를 M, [s,e]의 길이를 N이라고 한다면 한 행의 값을 채우는 데는 O(NlogM)만큼 걸린다.

왜 그렇게 되는지 잘 이해가 안되면 퀵소트를 떠올려보면 좋을 것 같습니다.

Quadrangle Inequality

최적화가 언제 가능하고 어떻게 하는 것인지 보았습니다. 이제 중요한 것을 생각해야 합니다.

최적해의 단조성은 어떻게 판단해야 할까?

아래의 조건은 최적해의 단조성과 필요충분조건은 아니지만, 충분조건입니다.

$$C[a][c] + C[b][d] \leq C[a][d] + C[b][c], for\, all\, a \leq b \leq c \leq d$$

여기서 C는 DP 점화식에서 코스트로 사용했던 그 C입니다.

위 조건을 만족하는 2차원 배열 C를 **Monge Array**라고도 부릅니다. 물론 실제로 이런 배열을 만들 필요는 없고 C[i][j]를 계산할 수 있으면 충분합니다.

Quadrangle Inequality

$$C[a][c] + C[b][d] \leq C[a][d] + C[b][c], for\ all\ a \leq b \leq c \leq d$$

위 사각부등식을 만족할 때, 최적해가 단조성을 가지면 충분조건임을 증명할 수 있습니다.

C가 위 부등식을 만족하며 a < b에 대해서 $opt_a > opt_b$라고 합시다. 편의상 opt_a를 A, opt_b를 B라고 하겠습니다. (B < A < a < b)

$$dp[t][i] = \min_{j<i}(dp[t-1][j] + C[j][i])$$

dp 점화식에 대입하면 아래와 같은 식을 얻을 수 있습니다.

$$dp[t][a] = dp[t-1][A] + C[A][a] \leq dp[t-1][B] + C[B][a]$$
$$dp[t][b] = dp[t-1][B] + C[B][b] \leq dp[t-1][A] + C[A][b]$$

Quadrangle Inequality

$$dp[t][a] = dp[t-1][A] + C[A][a] \leq dp[t-1][B] + C[B][a]$$
$$dp[t][b] = dp[t-1][B] + C[B][b] \leq dp[t-1][A] + C[A][b]$$

두 식을 더해주고 정리하면 아래와 같은 부등식이 나옵니다.

$$C[B][a] + C[A][b] \geq C[B][b] + C[A][a], (B < A < a < b)$$

이는 아래 사각부등식을 만족한다는 처음 가정과 모순됩니다. 따라서 $opt_a <= opt_b$임을 증명할 수 있습니다.

$$C[a][c] + C[b][d] \leq C[a][d] + C[b][c], for\ all\ a \leq b \leq c \leq d$$

DnC Optimization

$$dp[t][i] = \min_{j<i}(dp[t-1][j] + C[j][i])$$

위와 같은 dp식으로 K by N 테이블을 채우려면 $O(KN^2)$이 걸립니다. 다만, 최적해가 단조성을 가진다면 이를 O(KNlogN)으로 낮출 수 있습니다.

C 배열이 아래의 부등식을 만족하는 Monge Array라면 최적해가 단조성을 가지게 됩니다.

$$C[a][c] + C[b][d] \leq C[a][d] + C[b][c], for\ all\ a \leq b \leq c \leq d$$

Monge is Not Necessary

$$C[a][c] + C[b][d] \leq C[a][d] + C[b][c], for\ all\ a \leq b \leq c \leq d$$

C 배열이 Monge Array라는 것은 DnC opt를 활용하기 위한 충분조건이지 필요조건이 아닙니다.

최적해가 단조성을 보이는 것이 DnC opt를 활용하기 위한 필요충분조건입니다.

단순히 DP테이블을 채우는 문제뿐 아니라 이 단조성 자체를 발견해서 시간을 줄여 푸는 문제도 많습니다.

General Implementation

```
int C(int i, int j) {}; // cost function

long long dp_cur[MAXN], dp_prev[MAXN];
void solve(int s, int e, int opts, int opte) {
    if(s > e) return ;
    int mid = (s+e) >> 1;
    long long &ans = dp[mid];
    ans = INF;
    int mid_opt = opts;
    for(int i=opts;i<min(mid,opte);++i) {
        if(ans > dp_prev[i] + C(i,mid)) {
            ans = dp_prev[i] + C(i,mid);
            mid_opt = i;
        }
    }
    solve(s,mid-1,opts,mid_opt);
    solve(mid+1,e,mid_opt,opte);
}
```

$$dp[t][i] = \min_{j<i}(dp[t-1][j] + C[j][i])$$

위 점화식을 만족하면서 최적해가 단조성을 보일 때, 이전 행의 정보를 가지고 현재 행을 O(NlogN)만에 계산하는 코드입니다.

K = 1일 경우도 있습니다.

“Any Question?”