Dynamic Programming 2

나정휘
https://justiceHui.github.io/

목차

Divide and Conquer Optimization

Divide and Conquer Optimization

- Monotone Matrix
  - 아래 조건을 만족하는 행렬을 “monotone”하다고 부름
    - opt(i) = i번째 행에서 가장 좋은(최대/최소/…) 원소의 열 번호, 여러 개라면 가장 왼쪽에 있는 열
    - 모든 $1 \le i < N$에 대해 $opt(i) \le opt(i+1)$을 만족하면 monotone

  - N * M 크기의 monotone matrix의 모든 row optima를 O(NM)보다 빠르게 구할 수 있을까?

Divide and Conquer Optimization

- Divide and Conquer Optimization
  - 크기가 N * M인 monotone matrix A에서 모든 row optima를 O(N + M log N)에 찾는 방법
  - 분할 정복을 사용

  - f(s, e, l, r) = s..e번째 행의 row optima를 구함, 이때 row optima의 열 번호는 l 이상 r 이하
    - s > e 이면 종료
    - m = (s + e) / 2번째 행의 row optima 위치 opt(m)을 O(r-l) 시간에 구함
    - f(s, m-1, l, opt(m))
    - f(m+1, e, opt(m), r)
  - f(1, N, 1, M) 호출하면 됨

Divide and Conquer Optimization

- Divide and Conquer Optimization
  - f(s, e, l, r) = s..e번째 행의 row optima를 구함, 이때 row optima의 열 번호는 l 이상 r 이하
    - s > e 이면 종료
    - m = (s + e) / 2번째 행의 row optima 위치 opt(m)을 O(r-l) 시간에 구함
    - f(s, m-1, l, opt(m))
    - f(m+1, e, opt(m), r)

  - 시간 복잡도
    - 재귀 호출 횟수 O(N)
    - 재귀 깊이 최대 O(log N)
    - 각 깊이마다 row optima를 찾는데 필요한 총 시간 O(M)
    - 따라서 전체 시간 복잡도는 O(N + M log N)

Divide and Conquer Optimization

- BOJ 11001 김치
  - i번째 날의 온도를 T[i], 김치의 i번째 날에서의 가치를 V[i]라고 하자. 이때 T[i] ≥ T[i+1]
  - i번째 날 장독대에 넣어서 j번째 날에 꺼낸 김치의 맛은 (j – i) * T[j] + V[i]. 이때 V[i] > 0
  - j – i ≤ D를 만족해야 할 때, 가능한 김치의 맛의 최댓값을 구하는 문제

  - 행렬 A[i, j] = (j – i) * T[j] + V[i] 를 정의하자.
  - A는 monotone matrix
    - (귀류법) opt(i) = p, opt(i+1) = q 라고 할 때 p > q라고 가정하자.
    - A[i, q] ≤ A[i, p] 이므로 (q – i) * T[q] ≤ (p – i) * T[p]
    - A[i+1, q] ≥ A[i+1, p] 이므로 (q – i – 1) * T[q] ≥ (p – i – 1) * T[p]
    - 두 부등식이 모두 만족하기 위해서는 T[q] < T[p]가 되어야 함
    - 하지만 T의 정의에 따라 T[q] ≥ T[p]가 되어야 하므로 모순

Divide and Conquer Optimization

- BOJ 11001 김치
  - $A[i, j] = (j - i) * T[j] + V[i]$, $T[j] \geq T[j+1]$, $V[i] > 0$
  - A는 monotone matrix

  - $j - i \leq D$인 원소만 고려해도 별로 문제가 없음
    - A의 submatrix도 monotone matrix
    - $i < N$ 이면 $opt(i) \neq i$ 이므로 $opt(i)$번째 열은 $i+1$번째 행에서도 살아 있음
      - $j = i$ 이면 $j - i = 0$이라 무조건 손해
    - 따라서 $j - i \leq D$인 원소만 고려해도 monotone matrix

  - divide and conquer Optimization을 사용하면 $O(N \log N)$에 해결 가능

Divide and Conquer Optimization

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int N, D, T[101010], V[101010]; ll R;
inline ll Cost(int i, int j){ return 1LL * (j - i) * T[j] + V[i]; }

void Solve(int s, int e, int l, int r){
    if(s > e) return;
    int m = (s + e) / 2;
    ll mx = -1, opt = -1;
    for(int i=max(m,l); i<=min(r,m+D); i++){
        ll now = Cost(m, i);
        if(now > mx) mx = now, opt = i;
    }
    R = max(R, mx);
    Solve(s, m-1, l, opt);
    Solve(m+1, e, opt, r);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> D;
    for(int i=1; i<=N; i++) cin >> T[i];
    for(int i=1; i<=N; i++) cin >> V[i];
    Solve(1, N, 1, N);
    cout << R;
}
```

질문?

Divide and Conquer Optimization

- DP에서의 활용
  - 아래 두 조건을 만족하면 $1 \le k \le K$, $1 \le i \le N$일 때 $O(KN^2)$를 $O(KN \log N)$으로 최적화 가능
    - $D(k, i) = \min\{ D(k-1, j) + C(j, i) \}$ 꼴의 점화식
      - N개의 원소를 연속한 K개의 구간으로 나눠야 하는 문제 등
    - $D(k, i)$가 최적이 되는 j를 $opt(k, i)$라고 했을 때 $opt(k, i) \le opt(k, i+1)$을 만족해야 함

  - $f(k, s, e, l, r) = D(k, s..e)$를 구함, 이때 $l \le opt(k, s..e) \le r$을 만족해야 함
  - 각각의 k마다 $O(N \log N)$이므로 총 $O(KN \log N)$

  - $opt(k, i) \le opt(k, i+1)$ 여부를 더 쉽게 판단할 수 있을까?

Divide and Conquer Optimization

- DP에서의 활용
  - opt(k, i) ≤ opt(k, i+1) 여부를 더 쉽게 판단할 수 있을까?
    - D(k, i) = min{ D(k-1, j) + C(j, i) } 를 계산하는 것은
    - 행렬 M_k(i, j) = D(k-1, j) + C(j, i)의 row optima를 계산하는 것과 동일

    - 따라서 M_k가 monotone인 것과 opt(k, i) ≤ opt(k, i+1)인 것은 동치
    - D(k-1, j)는 C(j, i)의 각각의 행마다 같은 값을 더하는 것이므로 row optima의 위치는 변하지 않음
    - 따라서 C^T가 monotone인 것과 M_k도 monotone인 것은 동치
    - DnC Opt가 가능 = C^T가 monotone

Divide and Conquer Optimization

- DP에서의 활용
  - $opt(k, i) \le opt(k, i+1)$ 여부를 더 쉽게 판단할 수 있을까?
    - monotone보다 조건이 더 강한 monge matrix인지 확인하는 것도 유용함

    - 아래 성질을 만족하는 N * M 행렬 A를 monge matrix라고 부름
      - $1 \le a < b \le N$, $1 \le c < d \le M$을 만족하는 a, b, c, d에 대해 $A[a, c] + A[b, d] \le A[a, d] + A[b, c]$
      - monge matrix가 monotone인 것은 쉽게 보일 수 있음

    - monge matrix의 transpose도 monge matrix이므로
    - C^T가 transpose인 것을 보이거나 C가 monge인 것을 보이면 됨

Divide and Conquer Optimization

- monge matrix의 성질
  - monge matrix는 좋은 성질을 많이 갖고 있기 때문에 알아두면 편리함
  - Remark. A[a, c] + A[b, d] ≤ A[a, d] + A[b, c]

  - A[i, j]를 구간 [i, j]의 비용으로 생각하면, 두 구간이 겹쳐 있으면 풀어주는 것이 이득이라는 뜻
    - 교집합과 합집합의 관점에서 생각해 볼 수 있음
  - monge matrix의 행과 열 일부만 선택해서 순서를 유지한 채로 만든 행렬도 monge matrix
  - monge matrix 2개를 더해도 monge matrix
    - 따라서 음이 아닌 정수 x, y와 monge matrix A, B에 대해 xA + yB는 monge matrix
  - 모든 i, j에 대해 A[i, j] + A[i+1, j+1] ≤ A[i, j+1] + A[i+1, j]인 것과 monge인 것은 동치
    - 따라서 2 * 2 크기의 submatrix만 확인해도 됨

Divide and Conquer Optimization

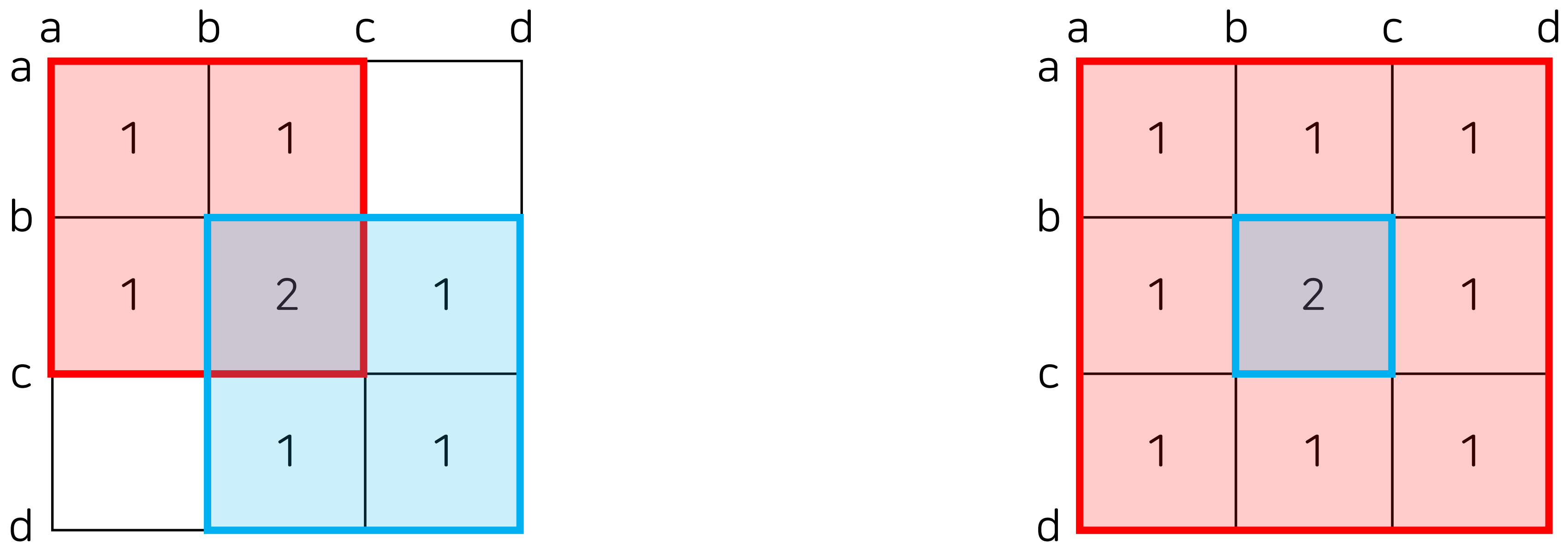
- BOJ 14177 티떱랜드
  - N개의 원소를 연속한 K개의 구간으로 나눠야 함
  - 모든 원소는 정확히 한 구간에 들어가야 하고, 정확히 K개의 비어있지 않은 구간이 되어야 함
  - i번째 원소와 j번째 원소를 같은 구간에 넣는데 필요한 비용은 U[i][j], U[i][j] ≥ 0
  - 비용을 최소화하는 문제

  - C[i][j] = i번째 사람부터 j번째 사람을 한 구간으로 만드는 비용이라고 정의하자.
    - C[i][j] = $\text{sum}_{x=i..j-1}\text{sum}_{y=x+1..j}$ U[x][y]

Divide and Conquer Optimization

- BOJ 14177 티떱랜드
  - C[i][j] = i번째 사람부터 j번째 사람을 한 구간으로 만드는 비용이라고 정의하자.
    - $C[i][j] = \mathrm{sum}_{x=i..j-1}\mathrm{sum}_{y=x+1..j}\ U[x][y]$
  - C는 monge matrix

$$C[a, c] + C[b, d] \le C[a, d] + C[b, c]$$



Divide and Conquer Optimization

```cpp
#include <bits/stdc++.h>
using namespace std;

int N, K, A[4040][4040], C[4040][4040], D[888][4040];

void DnC(int k, int s, int e, int l, int r){
    if(s > e) return;
    int m = (s + e) / 2, opt = l;
    D[k][m] = 0x3f3f3f3f;
    for(int i=l; i<=min(r, m-1); i++){
        int now = D[k-1][i] + C[i+1][m];
        if(D[k][m] > now) D[k][m] = now, opt = i;
    }
    DnC(k, s, m-1, l, opt);
    DnC(k, m+1, e, opt, r);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K;
    for(int i=1; i<=N; i++) for(int j=1; j<=N; j++) cin >> A[i][j];
    for(int i=1; i<=N; i++) partial_sum(A[i]+1, A[i]+N+1, A[i]+1);
    for(int i=1; i<=N; i++) for(int j=i+1; j<=N; j++) C[i][j] = C[i][j-1] + A[j][j] - A[j][i-1];

    memset(D, 0x3f, sizeof D);
    D[0][0] = 0;
    for(int i=1; i<=N; i++) D[1][i] = C[1][i];
    for(int i=2; i<=K; i++) DnC(i, i, N, i-1, N-1);
    cout << D[K][N];
}
```

질문?

Monotone Queue Optimization

Monotone Queue Optimization

- Linear Time CHT
  - 다음과 같은 문제는 스택을 이용해서 O(N+Q)에 해결할 수 있음
    - 일차 함수를 추가하는 연산 N번, 이때 추가되는 일차 함수의 기울기는 단조 감소
    - 특정 x좌표에서 일차 함수들의 최솟값을 구하는 쿼리 Q번, 이때 최솟값을 구하는 x좌표는 단조 증가
  - 구하는 방법
    - 정답이 될 가능성이 있는 직선을 스택으로 관리
    - 직선 삽입 연산
      - 어떤 일차 함수 f를 추가할 때, 스택의 맨 뒤에 있는 함수가 항상 f보다 크다면 제거
      - f보다 작은 구간이 존재하지 않는 직선을 모두 제거한 뒤 f 삽입
    - 최솟값 연산
      - 최솟값을 갖고 있는 함수가 있는 인덱스를 관리
      - 인덱스는 단조 증가하므로 amortized O(1)에 처리 가능
      - 인덱스를 증가시키는 것은 pop front와 동일하게 생각할 수 있음

Monotone Queue Optimization

- Monotone Queue Optimization
  - 기울기와 쿼리에 단조성이 있는 CHT의 일반화된 버전

  - $D(i) = \min_{0 \le j < i}\{ D(j) + C(j, i) \}$
  - 모든 $p < q$에 대해 다음을 만족하는 지점 cross(p, q)가 존재
    - $k <$ cross(p, q) 이면 $D(p) + C(p, k) < D(q) + C(q, k)$
    - cross(p, q) $\le k$ 이면 $D(p) + C(p, k) > D(q) + C(q, k)$

  - naïve하게 계산하면 $O(N^2)$이지만 monotone queue opt를 사용하면 $O(N \log N)$

Monotone Queue Optimization

- Monotone Queue Optimization
  - 모든 $p < q$에 대해 다음을 만족하는 지점 cross(p, q)가 존재
    - $k < \text{cross}(p, q)$ 이면 $D(p) + C(p, k) < D(q) + C(q, k)$
    - $\text{cross}(p, q) \le k$ 이면 $D(p) + C(p, k) > D(q) + C(q, k)$

  - $f_p(k) = D(p) + C(p, k)$와 $f_q(k) = D(q) + C(q, k)$의 교점이 1개 이하라는 뜻
    - ex. $C(j, i) = A(i) * B(j)$ 꼴이면 $f_*(k)$가 일차 함수 꼴이므로 자명하게 교점은 1개 이하
    - 기울기와 쿼리 위치에 단조성이 있는 CHT와 비슷하게 생각할 수 있음

  - 정답이 될 수 있는 함수들의 후보를 저장하는 Deque를 관리
    - CHT에서 스택과 현재의 최솟값을 나타내는 인덱스를 저장한 것과 동일

Monotone Queue Optimization

- Monotone Queue Optimization
  - 정답이 될 수 있는 함수들의 후보를 저장하는 Deque를 관리

  - D(i) = min{ D(j) + C(j, i) }를 계산해야 하는 상황
    - deque Q에서는 D(i..n)에서 답이 될 수 있는 j를 순서대로 관리
      - Q의 모든 원소가 cross(Q_i, Q_{i+1}) < cross(Q_{i+1}, Q_{i+2})를 만족하도록 유지
      - cross(Q_0, Q_1) ≥ i를 만족하도록 유지
    - D(i) = D(Q_0) + C(Q_0, i) 계산하고 i를 Q에 삽입
      - cross($Q_{|Q|-2}$, $Q_{|Q|-1}$) ≥ cross($Q_{|Q|-1}$, i) 이면 pop back
      - cross(Q_0, Q_1) = i 이면 pop front

Monotone Queue Optimization

- Monotone Queue Optimization
  - D(i) = min{ D(j) + C(j, i) }를 계산해야 하는 상황
    - deque Q에서는 D(i..n)에서 답이 될 수 있는 j를 순서대로 관리
    - D(i) = D(Q_0) + C(Q_0, i) 계산하고 i를 Q에 삽입
      - cross($Q_{|Q|-2}$, $Q_{|Q|-1}$) ≥ cross($Q_{|Q|-1}$, i) 이면 pop back
      - cross(Q_0, Q_1) = i 이면 pop front

  - 시간 복잡도
    - 각 원소는 Q에 최대 1번 삽입/삭제
    - 이 과정에서 cross를 O(N)번 호출
    - 이분 탐색을 이용해 cross 함수를 계산하면 전체 시간 복잡도는 O(N log N)

Monotone Queue Optimization

```cpp
template<class T, bool GET_MAX = false> // D[i] = func_{0 <= j < i} D[j] + cost(j, i)
pair<vector<T>, vector<int>> monotone_queue_dp(int n, const vector<T> &init, auto cost){
  assert((int)init.size() == n + 1); // cost function -> auto, do not use std::function
  vector<T> dp = init; vector<int> prv(n+1);
  auto compare = [](T a, T b){ return GET_MAX ? a < b : a > b; };
  auto cross = [&](int i, int j){
    int l = j, r = n + 1;
    while(l < r){
      int m = (l + r + 1) / 2;
      if(compare(dp[i] + cost(i, m), dp[j] + cost(j, m))) r = m - 1; else l = m;
    }
    return l;
  };
  deque<int> q{0};
  for(int i=1; i<=n; i++){
    while(q.size() > 1 && compare(dp[q[0]] + cost(q[0], i), dp[q[1]] + cost(q[1], i))) q.pop_front();
    dp[i] = dp[q[0]] + cost(q[0], i); prv[i] = q[0];
    while(q.size() > 1 && cross(q[q.size()-2], q.back()) >= cross(q.back(), i)) q.pop_back();
    q.push_back(i);
  }
  return {dp, prv};
}
```

질문?

Monotone Queue Optimization

- C가 monge matrix인 경우
  - $i < j < k$ 에 대해 아래 두 가지 명제가 참
  - $D(i) + C(i, k+1) \leq D(j) + C(j, k+1)$ 이면 $D(i) + C(i, k) \leq D(j) + C(j, k)$
    - $D(i) - D(j) \leq C(j, k+1) - C(i, k+1) \leq C(j, k) - C(i, k)$
    - 따라서 $D(i) + C(i, k) \leq D(j) + C(j, k)$
  - $D(i) + C(i, k) \geq D(j) + C(j, k)$ 이면 $D(i) + C(i, k+1) \geq D(j) + C(j, k+1)$
    - $D(j) - D(i) \leq C(i, k) - C(j, k) \leq C(i, k+1) - C(j, k+1)$
    - 따라서 $D(i) + C(i, k+1) \geq D(j) + C(j, k+1)$

  - 그러므로 monotone queue optimization을 사용할 수 있음

Monotone Queue Optimization

- 다른 DP 최적화와의 호환성
  - 기울기와 쿼리 위치에 단조성이 있는 CHT
    - D(i) = min{ D(j) + M(j) * X(i) }
    - M(j)가 단조 감소, X(i)가 단조 증가하면 C(j, i) = M(j) * X(i)는 monge matrix
  - Divide and Conquer Optimization
    - D(k, i) = min{ D(k-1, j) + C(j, i) }
    - D(k-1, j)와 D(k, i)를 각각 D(j), D(i)로 바꾸면 monotone queue optimization 형태
    - C가 monge 등 dp 최적화에서 요구하는 조건을 만족하면 사용 가능
  - Li Chao Tree
    - Li Chao Tree도 교점이 1개인 함수의 최대/최소를 관리하는 자료구조
    - monotone queue optimization 구현하기 귀찮으면 리차오 트리 사용해도 됨

질문?