DP with indices

2020 Winter 중급

20141284 이기현

기본적인 다이나믹 프로그래밍

5 #2748 피보나치 수 2

3 #1463 1로 만들기

1 #11048 이동하기

기본적인 다이나믹 프로그래밍

5 #2748 피보나치 수 2

dp[n] = dp[n-1] + dp[n-2]

3 #1463 1로 만들기

dp[n] = min{ dp[n/3] , dp[n/2] , dp[n-1] } + 1

1 #11048 이동하기

dp[n][m] = min{ dp[n-1][m] , dp[n][m-1] , dp[n-1][m-1] } + cost[n][m]

기본적인 다이나믹 프로그래밍

- n번째 ~의 값, n행 m열의 ~의 값 (또는 최적해) 등에 초점

- 인덱스에 i번째 외에도 다른 정보를 이용할 수 있을까?

기본적인 다이나믹 프로그래밍

- n번째 ~의 값, n행 m열의 ~의 값 (또는 최적해) 등에 초점
- 인덱스에 i번째 외에도 다른 정보를 이용할 수 있을까?

⇒ 크기 / 구간 / 위치 (등등)

배낭 문제 (Knapsack Problem)

- 말 그대로 '배낭' 에 관련됨.
- 배낭의 무게 한도 / 각 물건의 무게 및 가치 / 최대한 가치가 높도록

배낭 문제 (Knapsack Problem)

- 말 그대로 '배낭' 에 관련됨.
- 배낭의 무게 한도 / 각 물건의 무게 및 가치 / 최대한 가치가 높도록

- 물건을 쪼개서 넣을 수 있을 때 (Fractional Knapsack Problem)

- 물건을 넣거나 안 넣는 것만 가능할 때 (0-1 Knapsack Problem)

배낭 문제 (Knapsack Problem)

- 말 그대로 '배낭' 에 관련됨.
- 배낭의 무게 한도 / 각 물건의 무게 및 가치 / 최대한 가치가 높도록

- 물건을 쪼개서 넣을 수 있을 때 (Fractional Knapsack Problem)

  Greedy Algorithm

- 물건을 넣거나 안 넣는 것만 가능할 때 (0-1 Knapsack Problem)

  Dynamic Programming

#12865 평범한 배낭 5

- n개의 물건이 있고, 각 물건은 가치 v_i 와 무게 w_i 를 가지고 있다.
- 최대 k무게까지 버틸 수 있는 배낭이 있다.
- 배낭에 넣을 수 있는 물건의 가치의 합의 최대는?
- $1 \le n \le 100$, $1 \le k \le 100{,}000$

#12865 평범한 배낭 5

- n = 3 , k = 9
- A → $v_a = 5, w_a = 2$
- B → $v_b = 4, w_b = 3$
- C → $v_c = 2, w_c = 5$

#12865 평범한 배낭 5

- n = 3 , k = 9
- A → $v_a = 5 , w_a = 2$
- B → $v_b = 4 , w_b = 3$
- C → $v_c = 2 , w_c = 5$

• {A}	W = 2 V = 5
• {B}	W = 3 V = 4
• {C}	W = 5 V = 2
• {A, B}	W = 5 V = 9
• {A, C}	W = 7 V = 7
• {B, C}	W = 8 V = 6

#12865 평범한 배낭 5

- 모든 조합을 고려하자!

#12865 평범한 배낭 5

- 모든 조합을 고려하자!

- 각 물건에 대해 경우의 수 2가지

- 총 n개의 물건 → 2^n 가지의 경우의 수

#12865 평범한 배낭 5

- 모든 조합을 고려하자!
- 각 물건에 대해 경우의 수 2가지
- 총 n개의 물건 → 2^n 가지의 경우의 수
- 모든 조합을 체크하는 시간 → $O(2^n)$ → TLE

#12865 평범한 배낭 5

- index에 ‘크기’ 개념 넣기

- dp[n][w] = n번째 물건까지 고려하여 w무게를 담았을 때, 최대 가치

#12865 평범한 배낭 5

- index에 '크기' 개념 넣기

- dp[n][w] = n번째 물건까지 고려하여 w무게를 담았을 때, 최대 가치

$$dp[0][i] = 0\,, dp[0][w_i] = v_i$$

$$dp[n][w] = \max\{dp[n-1][w-w_i] + v_i\,, dp[n-1][w]\}$$

#12865 평범한 배낭 5

- n = 4 , k = 7
- A → $v_a = 13 , w_a = 5$
- B → $v_b = 8 , w_b = 3$
- C → $v_c = 6 , w_c = 2$
- D → $v_d = 12 , w_d = 4$

	0	1	2	3	4	5	6	7
A	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0

#12865 평범한 배낭 5

- n = 4 , k = 7
- A → $v_a = 13 , w_a = 5$
- B → $v_b = 8 , w_b = 3$
- C → $v_c = 6 , w_c = 2$
- D → $v_d = 12 , w_d = 4$

	0	1	2	3	4	5	6	7
A	0	0	0	0	0	13	0	0
B	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0

#12865 평범한 배낭 5

- n = 4 , k = 7
- A → $v_a = 13\,, w_a = 5$
- B → $v_b = 8\,, w_b = 3$
- C → $v_c = 6\,, w_c = 2$
- D → $v_d = 12\,, w_d = 4$

	0	1	2	3	4	5	6	7
A	0	0	0	0	0	13	0	0
B	0	0	0	8	0	13	0	0
C	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0

#12865 평범한 배낭 5

- n = 4 , k = 7
- A → $v_a = 13, w_a = 5$
- B → $v_b = 8, w_b = 3$
- C → $v_c = 6, w_c = 2$
- D → $v_d = 12, w_d = 4$

	0	1	2	3	4	5	6	7
A	0	0	0	0	0	13	0	0
B	0	0	0	8	0	13	0	0
C	0	0	6	8	0	14	0	19
D	0	0	0	0	0	0	0	0

#12865 평범한 배낭 5

- n = 4 , k = 7
- A → $v_a = 13 , w_a = 5$
- B → $v_b = 8 , w_b = 3$
- C → $v_c = 6 , w_c = 2$
- D → $v_d = 12 , w_d = 4$

	0	1	2	3	4	5	6	7
A	0	0	0	0	0	13	0	0
B	0	0	0	8	0	13	0	0
C	0	0	6	8	0	14	0	19
D	0	0	6	8	12	14	18	20

#12865 평범한 배낭 5

- 21 line

  첫번째 물건 처리

- 24 line

  i번째 물건을 못 넣을 경우

- 25 line

  i번째 물건을

  넣을지 안넣을지 결정

```cpp
17  vector<int> w(n), v(n);
18  for (int i = 0; i < n; ++i) cin >> w[i] >> v[i];
19
20  vector<vector<int> > dp(n, vector<int>(k + 1));
21  dp[0][w[0]] = v[0];
22  for (int i = 1; i < n; ++i)
23      for (int j = 0; j <= k; ++j)
24          if (j < w[i]) dp[i][j] = dp[i - 1][j];
25          else dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);
26
27  int ans = -1;
28  for (int i = 1; i <= k; ++i) ans = max(ans, dp[n - 1][i]);
```

#12865 평범한 배낭 5

- 21 line

  첫번째 물건 처리

- 24 line

  i번째 물건을 못 넣을 경우

- 25 line

  i번째 물건을

  넣을지 안넣을지 결정

```
17  vector<int> w(n), v(n);
18  for (int i = 0; i < n; ++i) cin >> w[i] >> v[i];
19
20  vector<vector<int> > dp(n, vector<int>(k + 1));
21  dp[0][w[0]] = v[0];
22  for (int i = 1; i < n; ++i)
23      for (int j = 0; j <= k; ++j)
24          if (j < w[i]) dp[i][j] = dp[i - 1][j];
25          else dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);
26
27  int ans = -1;
28  for (int i = 1; i <= k; ++i) ans = max(ans, dp[n - 1][i]);
```

시간 복잡도 : $O(NK)$ / 공간 복잡도 : $O(NK + 2N)$

배낭 문제 (Knapsack Problem)

- 시간 복잡도가 $O(NK)$로 K가 매우 커질 경우, 문제 해결이 힘들 수 있다.
- K가 커질 경우, 메모리제한이 아슬아슬 또는 터지는 경우가 있다.
  (→ Sliding Window 개념을 이용하여 해결 가능)
- 아직까지는 다항시간 풀이가 나오지 않았다.

배낭 문제 (Knapsack Problem) – 메모리 최적화 (1)

- dp[i][j] 를 구할 때,
  i-1 행의 값에만 접근
- 행이 2개인 dp table 이용
- 두 행을 번갈아 가며 사용

```cpp
vector<int> w(n), v(n);
for (int i = 0; i < n; ++i) cin >> w[i] >> v[i];

vector<vector<int> > dp(n, vector<int>(k + 1));
dp[0][w[0]] = v[0];
for (int i = 1; i < n; ++i)
    for (int j = 0; j <= k; ++j)
        if (j < w[i]) dp[i][j] = dp[i - 1][j];
        else dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);

int ans = -1;
for (int i = 1; i <= k; ++i) ans = max(ans, dp[n - 1][i]);
```

배낭 문제 (Knapsack Problem) – 메모리 최적화 (1)

- dp[i][j] 를 구할 때,
  i-1 행의 값에만 접근

- 행이 2개인 dp table 이용

- 두 행을 번갈아 가며 사용

```cpp
vector<int> w(n), v(n);
for (int i = 0; i < n; ++i) cin >> w[i] >> v[i];

vector<vector<int> > dp(2, vector<int>(k + 1));
dp[0][w[0]] = v[0];
for (int i = 1; i < n; ++i)
    for (int j = 0; j <= k; ++j) {
        int cur = i % 2, prev = !(i % 2);
        if (j < w[i]) dp[cur][j] = dp[prev][j];
        else dp[cur][j] = max(dp[prev][j], dp[prev][j - w[i]] + v[i]);
    }

int ans = -1;
for (int i = 1; i <= k; ++i) ans = max(ans, dp[(n - 1) % 2][i]);
```

공간 복잡도 : $O(2K + 2N)$

배낭 문제 (Knapsack Problem) – 메모리 최적화 (2)

- w[i]보다 작은 j에 대해서는 갱신이 되지 않고, j – w[i] 로부터 전파
- 행이 1개인 dp table을 이용
- j를 k부터 w[i]까지만 갱신

```
vector<int> w(n), v(n);
for (int i = 0; i < n; ++i) cin >> w[i] >> v[i];

vector<vector<int> > dp(2, vector<int>(k + 1));
dp[0][w[0]] = v[0];
for (int i = 1; i < n; ++i)
    for (int j = 0; j <= k; ++j) {
        int cur = i % 2, prev = !(i % 2);
        if (j < w[i]) dp[cur][j] = dp[prev][j];
        else dp[cur][j] = max(dp[prev][j], dp[prev][j - w[i]] + v[i]);
    }

int ans = -1;
for (int i = 1; i <= k; ++i) ans = max(ans, dp[(n - 1) % 2][i]);
```

배낭 문제 (Knapsack Problem) – 메모리 최적화 (2)

- w[i]보다 작은 j에 대해서는 갱신이 되지 않고, j – w[i] 로부터 전파

- 행이 1개인 dp table을 이용

- j를 k부터 w[i]까지만 갱신

```cpp
vector<int> w(n), v(n);
for (int i = 0; i < n; ++i) cin >> w[i] >> v[i];

vector<int> dp(k + 1);
dp[w[0]] = v[0];
for (int i = 1; i < n; ++i)
    for (int j = k; j >= w[i]; --j)
        dp[j] = max(dp[j], dp[j - w[i]] + v[i]);

int ans = -1;
for (int i = 1; i <= k; ++i) ans = max(ans, dp[i]);
```

공간 복잡도 : $O(K + 2N)$

#11066 파일 합치기 3

- n개의 파일이 있고 하나의 파일로 만들려고 한다.
- 그 과정에서 파일은 두 개씩 합치고 연속된 파일을 합치도록 한다.
- 파일을 합치는 비용은 두 파일의 크기의 합이다.
- 모든 파일을 합칠 때 필요한 최소비용은?
- $1 \leq n \leq 500$

#11066 파일 합치기 3

- n = 4
- file = 40 30 30 50

File1	File2	File3	File4	Ans
40	30	30	50	0

#11066 파일 합치기 3

- n = 4
- file = 40 30 30 50

File1	File2	File3	File4	Ans
40	30	30	50	0
70		30	50	70

#11066 파일 합치기 3

- n = 4
- file = 40 30 30 50

<table>
<tr><th>File1</th><th>File2</th><th>File3</th><th>File4</th><th>Ans</th></tr>
<tr><td>40</td><td>30</td><td>30</td><td>50</td><td>0</td></tr>
<tr><td colspan="2">70</td><td>30</td><td>50</td><td>70</td></tr>
<tr><td colspan="2">70</td><td colspan="2">80</td><td>150</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
</table>

#11066 파일 합치기

- n = 4
- file = 40 30 30 50

<table>
<tr><th>File1</th><th>File2</th><th>File3</th><th>File4</th><th>Ans</th></tr>
<tr><td>40</td><td>30</td><td>30</td><td>50</td><td>0</td></tr>
<tr><td colspan="2">70</td><td>30</td><td>50</td><td>70</td></tr>
<tr><td colspan="2">70</td><td colspan="2">80</td><td>150</td></tr>
<tr><td colspan="4">150</td><td>300</td></tr>
</table>

#11066 파일 합치기

- 위 4개의 파일이 합쳐지기 위해선?

#11066 파일 합치기 3

- 위의 4개의 파일이 합쳐진 파일을 만들기 위해선?

#11066 파일 합치기 3

- 위의 4개의 파일이 합쳐진 파일을 만들기 위해선?

- 구간에 대한 dp 점화식

$$dp[l][r] = \min_{\{l \le k < r\}} \{\, dp[l][k] + dp[k+1][r] \,\} + \text{file[l]} \sim \text{file[r]}$$

#11066 파일 합치기 3

- 구간에 대한 점화식

$$dp[l][r] = \min_{\{l \le k < r\}} \{ dp[l][k] + dp[k+1][r] \} + \text{file[l]}\sim\text{file[r]}$$

- 시간 복잡도

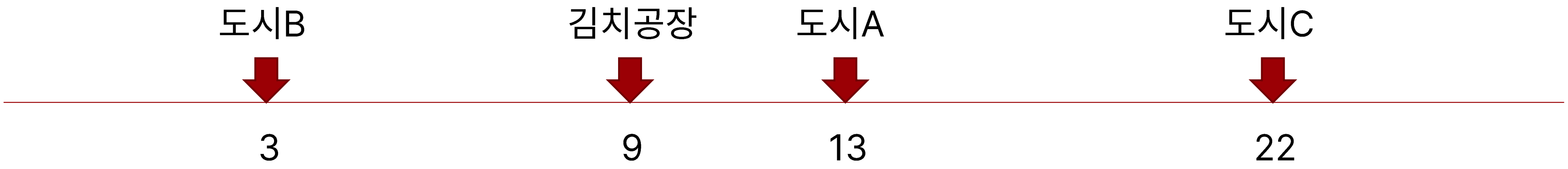
구간 - $O(n^2)$ / 구간분할 - $O(n) \rightarrow O(n^3)$

(knuth optimization 이라는 최적화 기법을 이용하여 $O(n^2)$으로 줄일 수 있다.)

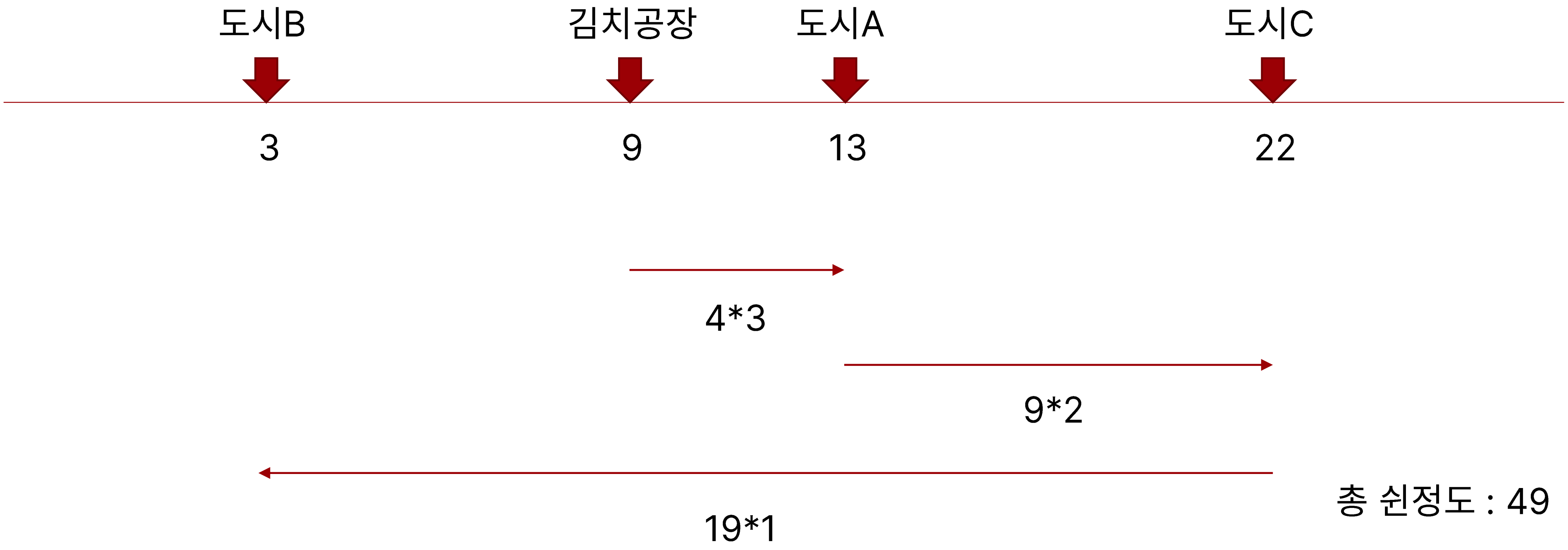
#2184 김치 배달 3

- 김치 공장으로부터 1차원 직선 상의 n개의 도시로 김치 1포기씩 배달을 하려고 한다.
- 1초에 1만큼 이동을 하며 김치는 1초에 1만큼 쉰다.
- 모든 도시에 김치를 배달 했을 때, 김치의 쉰 정도의 합의 최솟값은?
- $1 \leq n \leq 1000$

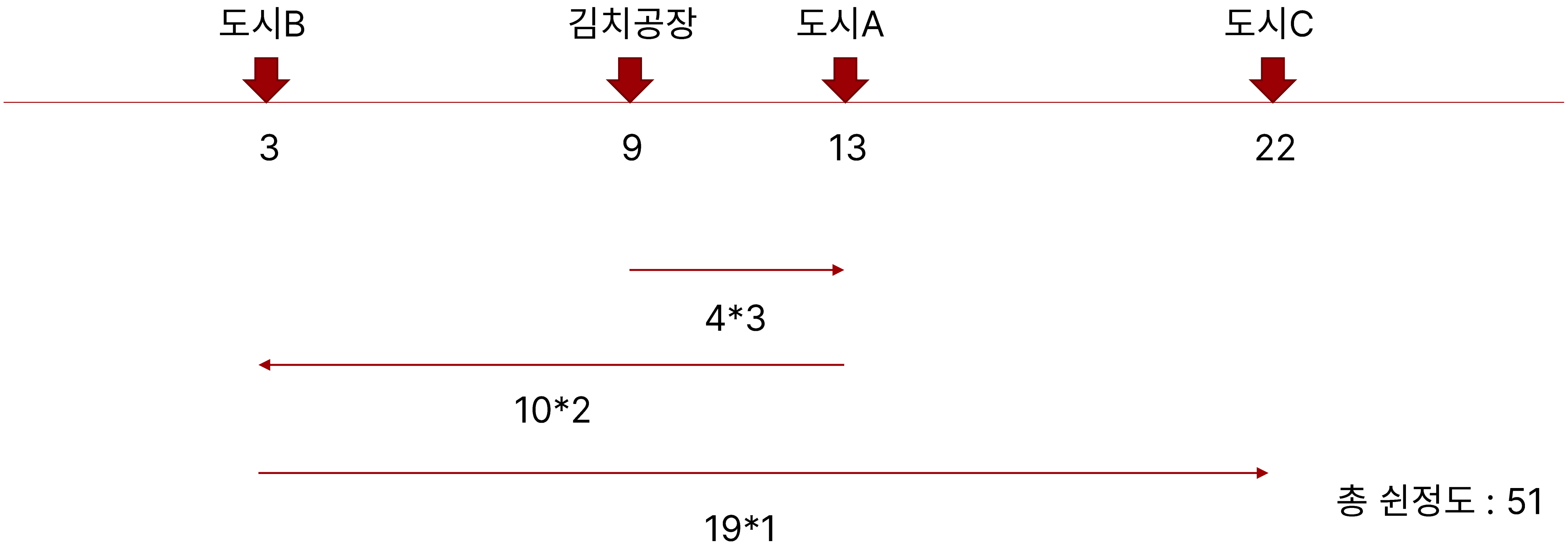
#2184 김치 배달



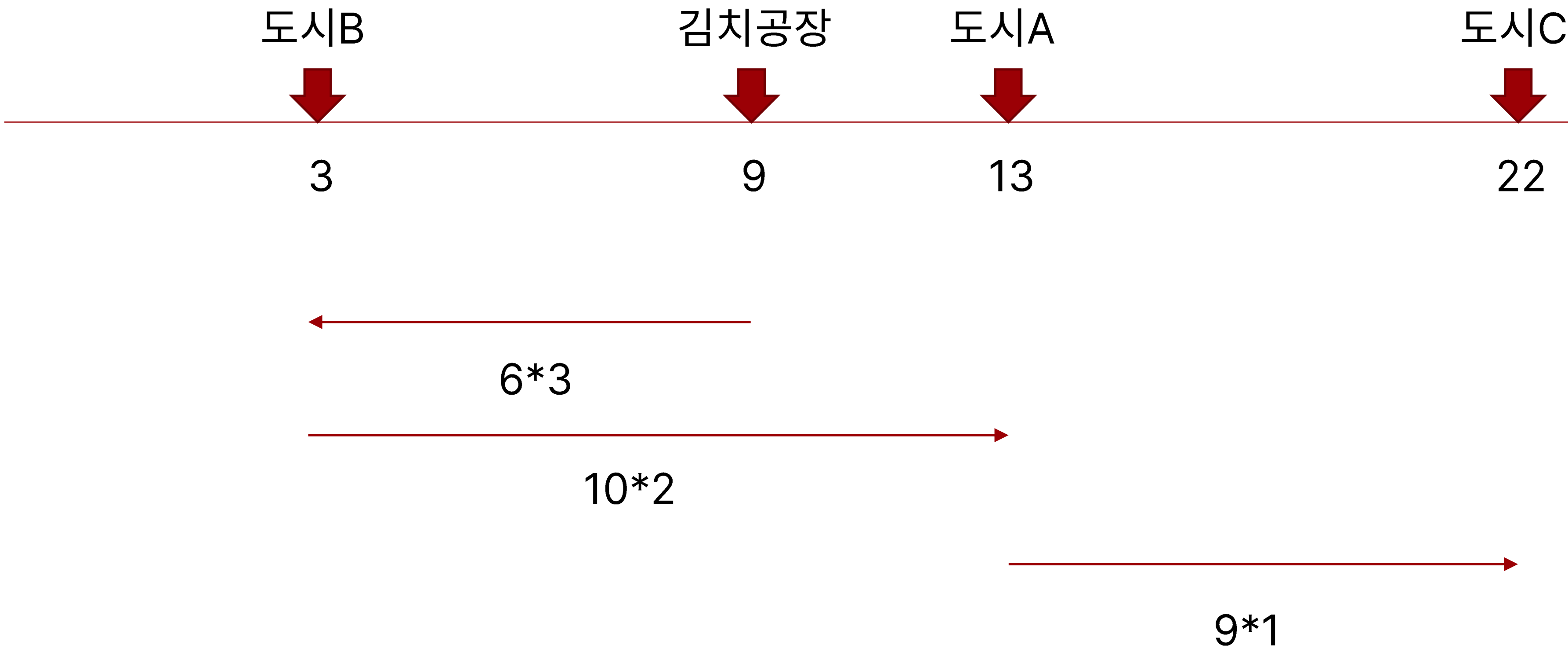
#2184 김치 배달



#2184 김치 배달 3



#2184 김치 배달



총 쉰정도 : 47

#2184 김치 배달 3

- 공장에서 시작해서 구간이 왼쪽 또는 오른쪽으로 1칸씩 확장되고 있음

#2184 김치 배달 3

- 공장에서 시작해서 구간이 왼쪽 또는 오른쪽으로 1칸씩 확장되고 있음

- 구간에 대한 점화식

$dp[l][r]$ = l~r 구간의 도시에 모두 배달 했을 때, 쉰 정도의 최솟값

$$dp[l][r] = \min\{ dp[l+1][r] + \sim, dp[l][r-1] + \sim \}$$

#2184 김치 배달 3

- 공장에서 시작해서 구간이 왼쪽 또는 오른쪽으로 1칸씩 확장되고 있음

- 구간에 대한 점화식

$dp[l][r]$ = l~r 구간의 도시에 모두 배달 했을 때, 쉰 정도의 최솟값

$$dp[l][r] = \min\{\, dp[l+1][r] + \sim, dp[l][r-1] + \sim \}$$

- l~r 구간을 배달하고 다음 배달을 갈 때,
  l에 있는 것과 r에 있는 건 ~ 부분 값이 달라짐

⇒ 어디에서 오는지가 중요

#2184 김치 배달 3

- l~r 구간에 배달했을 때, 있을 수 있는 곳은 l 또는 r

- 구간과 위치에 대한 점화식

$dp[l][r][where]$ = l~r 구간의 도시에 모두 배달했고,

where(l 또는 r)에 마지막 배달했을 때, 쉰 정도의 최솟값

$$dp[l][r][cur] = \min\{\begin{matrix} dp[l+1][r][prev] + dist(cur, prev) * (\sim) \\ dp[l][r-1][prev] + dist(cur, prev) * (\sim) \end{matrix}\}$$

cur, prev = 각 구간의 좌측 끝(0) or 우측 끝(1)

#2184 김치 배달 3

- 도시, 공장의 좌표 넣어주고
- 공장의 인덱스 찾기

```
for (int i = 1; i <= n; ++i) {
    int x;
    cin >> x;
    loca.push_back(x);
}
loca.push_back(l);
sort(loca.begin(), loca.end());
s = lower_bound(loca.begin(), loca.end(), l) - loca.begin();
```

#2184 김치 배달

- 20,25 line
  l은 항상 시작점 왼쪽
  r은 항상 시작점 오른쪽
- 22~23 line
  l+1~r 구간으로부터
  왼쪽으로 확장
- 26~27 line
  l~r-1 구간으로부터
  오른쪽으로 확장

```
15 void solve(int l, int r, int rem) {
16     if (dp[l][r][0] != -1) return;
17 
18     dp[l][r][0] = dp[l][r][1] = INF;
19 
20     if (l < s) {
21         solve(l + 1, r, rem + 1);
22         dp[l][r][0] = min(dp[l + 1][r][0] + (loca[l + 1] - loca[l]) * rem,
23                           dp[l + 1][r][1] + (loca[r] - loca[l]) * rem);
24     }
25     if (s < r) {
26         solve(l, r - 1, rem + 1);
27         dp[l][r][1] = min(dp[l][r - 1][0] + (loca[r] - loca[l]) * rem,
28                           dp[l][r - 1][1] + (loca[r] - loca[r - 1]) * rem);
29     }
30 }
```

```
47     memset(dp, -1, sizeof(dp));
48     dp[s][s][0] = dp[s][s][1] = 0;
49     solve(0, n, 1);
50 
51     cout << min(dp[0][n][0], dp[0][n][1]);
```

#2184 김치 배달 3

- 20,25 line
  - l은 항상 시작점 왼쪽
  - r은 항상 시작점 오른쪽
- 22~23 line
  - l+1~r 구간으로부터
  - 왼쪽으로 확장
- 26~27 line
  - l~r-1 구간으로부터
  - 오른쪽으로 확장

```
15 void solve(int l, int r, int rem) {
16     if (dp[l][r][0] != -1) return;
17 
18     dp[l][r][0] = dp[l][r][1] = INF;
19 
20     if (l < s) {
21         solve(l + 1, r, rem + 1);
22         dp[l][r][0] = min(dp[l + 1][r][0] + (loca[l + 1] - loca[l]) * rem,
23                           dp[l + 1][r][1] + (loca[r] - loca[l]) * rem);
24     }
25     if (s < r) {
26         solve(l, r - 1, rem + 1);
27         dp[l][r][1] = min(dp[l][r - 1][0] + (loca[r] - loca[l]) * rem,
28                           dp[l][r - 1][1] + (loca[r] - loca[r - 1]) * rem);
29     }
30 }
```

```
47     memset(dp, -1, sizeof(dp));
48     dp[s][s][0] = dp[s][s][1] = 0;
49     solve(0, n, 1);
50 
51     cout << min(dp[0][n][0], dp[0][n][1]);
```

시간 복잡도 : $O(n^2)$

문제 추천

- 2 4781 사탕가게
- 3 11066 파일 합치기
- 5 12865 평범한 배낭
- 4 2449 전구
- 5 5557 1학년
- 1 1509 팰린드롬 분할
- 1 12920 평범한 배낭 2
- 3 2184 김치 배달
- 4 5626 제단
- 1 2419 사수아탕