JOI春季セミナー

上級1日目　動的計画法

目標

次のタイプの動的計画法の問題が解けるようになること

- 基本的なDP
- 編集距離に関するDP
- 区間に関するDP

説明は程々に、演習する時間を多くとりたいと思います。

動的計画法とは

> 与えられた問題全体を一連の部分問題に上手に分解し，各部分問題に対する解をメモ化しながら，小さな部分問題からより大きな部分問題へと順に解を求めていく手法

（テキストより）

基本的なDP

EDPC A - Frog 1 (https://atcoder.jp/contests/dp/tasks/dp_a)

N 個の足場があります。足場には $1, 2, \ldots, N$ と番号が振られています。各 $i\ (1 \leq i \leq N)$ について、足場 i の高さは h_i です。

最初、足場 1 にカエルがいます。カエルは次の行動を何回か繰り返し、足場 N まで辿り着こうとしています。

- 足場 i にいるとき、足場 $i+1$ または $i+2$ へジャンプする。このとき、ジャンプ先の足場を j とすると、コスト $|h_i - h_j|$ を支払う。

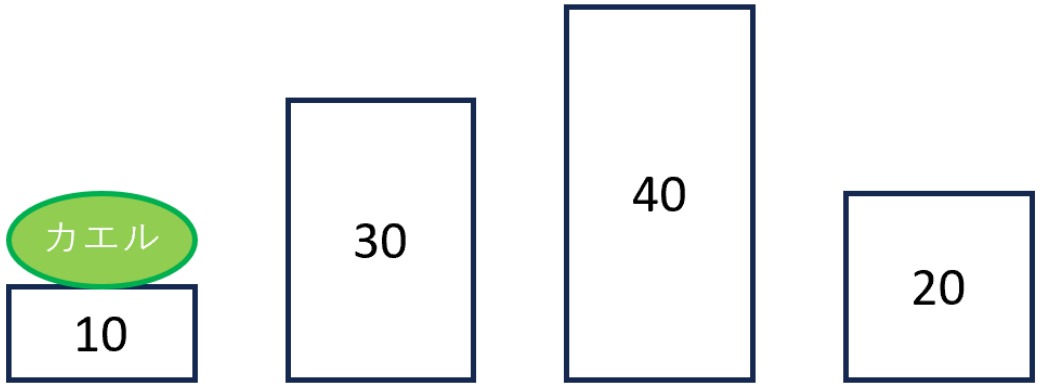
カエルが足場 N に辿り着くまでに支払うコストの総和の最小値を求めてください。

制約：$2 \leq N \leq 10^5, \quad 1 \leq h_i \leq 10^4$

基本的なDP

サンプル1

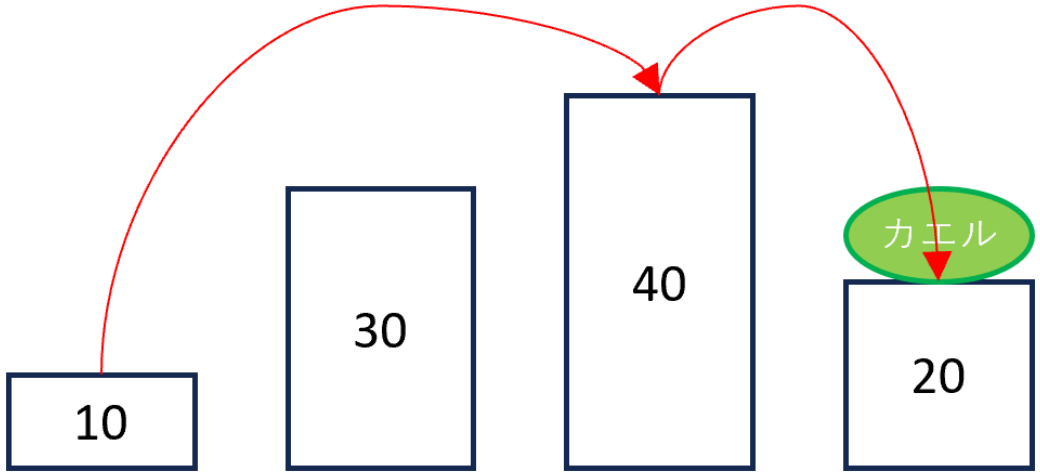
$N = 4, h_i = (10, 30, 40, 20)$



基本的なDP

サンプル1

$N = 4, h_i = (10, 30, 40, 20)$

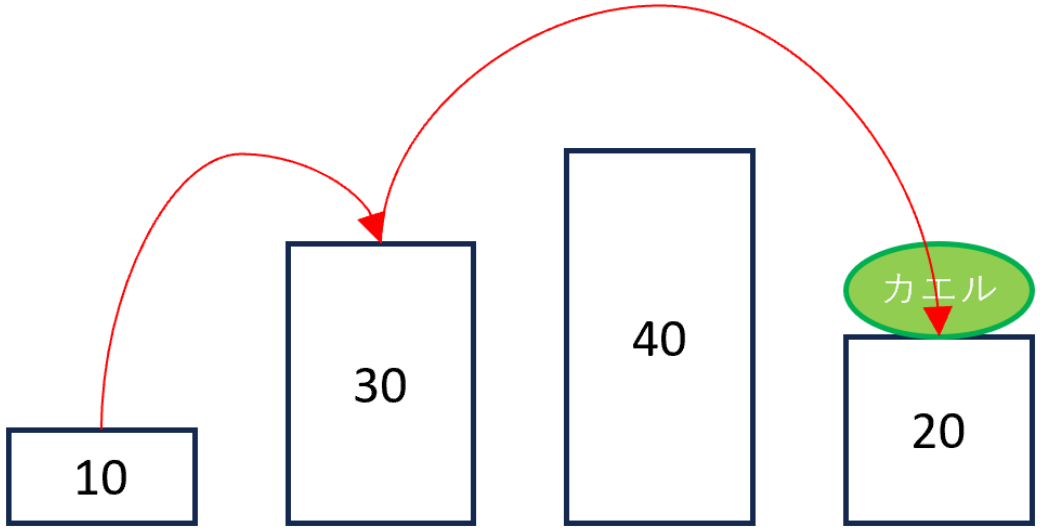


足場 $1 \to 3 \to 4$ と移動すると、コストは $|10 - 40| + |40 - 20| = 50$

基本的なDP

サンプル1

$N = 4, h_i = (10, 30, 40, 20)$



足場 $1 \to 2 \to 4$ と移動すると、コストは $|10 - 30| + |30 - 20| = 30$ （これが最小）

基本的なDP

式で表す

$\mathrm{dp}[i]$ を、「足場 i に辿り着くために必要な最小コスト」と定義してみると…？

- $\mathrm{dp}[1] = 0$
- $\mathrm{dp}[2] = \mathrm{dp}[1] + |h_1 - h_2|$
- $\mathrm{dp}[3] = \min\left(\mathrm{dp}[1] + |h_1 - h_3|, \mathrm{dp}[2] + |h_2 - h_3|\right)$
- $\mathrm{dp}[4] = \min\left(\mathrm{dp}[2] + |h_2 - h_4|, \mathrm{dp}[3] + |h_3 - h_4|\right)$
- $\vdots$
- $\mathrm{dp}[i] = \min\left(\mathrm{dp}[i-2] + |h_{i-2} - h_i|, \mathrm{dp}[i-1] + |h_{i-1} - h_i|\right) \quad (3 \leq i \leq N)$

基本的なDP

漸化式が出てきます

- $\mathrm{dp}[1] = 0$
- $\mathrm{dp}[2] = \mathrm{dp}[1] + |h_1 - h_2|$
- $\mathrm{dp}[i] = \min\left(\mathrm{dp}[i-2] + |h_{i-2} - h_i|, \mathrm{dp}[i-1] + |h_{i-1} - h_i|\right) \quad (3 \leq i \leq N)$

特に、 $\mathrm{dp}[3]$ 以降は 1 つ前の結果と 2 つ前の結果が分かっていれば計算可能

⇒「$\mathrm{dp}[N]$ を計算したければ $\mathrm{dp}[N-1]$ と $\mathrm{dp}[N-2]$ を計算すればいい」と言える

⇒ **部分問題に分解する**

基本的なDP

1-index は書きにくいので 0-index に直しておきます

- $\mathrm{dp}[0] = 0$
- $\mathrm{dp}[1] = \mathrm{dp}[1] + |h_0 - h_1|$
- $\mathrm{dp}[i] = \min\left(\mathrm{dp}[i-2] + |h_{i-2} - h_i|, \mathrm{dp}[i-1] + |h_{i-1} - h_i|\right) \quad (2 \leq i \leq N-1)$

i の昇順に $\mathrm{dp}[i]$ の値を求めていきましょう

基本的なDP

ソースコード (C++)

```
int main(){
  int N; cin >> N;
  vector<int> h(N);
  for(int i = 0; i < N; ++i) cin >> h[i];

  vector<int> dp(N, 0);
  dp[1] = abs(h[0] - h[1]);
  for(int i = 2; i < N; ++i){
    dp[i] = min(dp[i - 2] + abs(h[i - 2] - h[i]),
                dp[i - 1] + abs(h[i - 1] - h[i]));
  }
  cout << dp[N - 1] << endl;
}
```

基本的なDP

ソースコード (Python)

```
N = int(input())
h = list(map(int, input().split()))

dp = [0] * N
dp[1] = abs(h[0] - h[1])
for i in range(2, N):
    dp[i] = min(dp[i - 2] + abs(h[i - 2] - h[i]),
                dp[i - 1] + abs(h[i - 1] - h[i]))

print(dp[N - 1])
```

基本的なDP

緩和について

$x = \min(x, y)$ みたいな処理を**緩和**と言います

頻出なので、予めテンプレート化しておくと良いです

```
template<typename T>
void chmin(T &x, T y){x = min(x, y);}
```

緩和を用いた Frog 1 の実装

```
int main(){
  int N; cin >> N;
  vector<int> h(N);
  for(int i = 0; i < N; ++i) cin >> h[i];

  vector<int> dp(N, 0);
  dp[1] = abs(h[0] - h[1]);
  for(int i = 2; i < N; ++i){
    chmin(dp[i], dp[i - 2] + abs(h[i - 2] - h[i]));
    chmin(dp[i], dp[i - 1] + abs(h[i - 1] - h[i]));
  }
  cout << dp[N - 1] << endl;
}
```

ナップサック問題

ナップサック問題

N 個の品物があります。品物には $1, 2, \ldots, N$ と番号が振られています。各 i $(1 \leq i \leq N)$ について、品物 i の重さは w_i で、価値は v_i です。

N 個の品物のうちいくつかを選び、ナップサックに入れて持ち帰ることにしました。ナップサックの容量は W であり、持ち帰る品物の重さの総和は W 以下でなければいけません。

持ち帰る品物の価値の総和の最大値を求めてください。

制約 : $1 \leq N \leq 100, \quad 1 \leq W \leq 10^5, \quad 1 \leq w_i \leq W, \quad 1 \leq v_i \leq 10^9$

ナップサック問題

サンプル

$N = 3, W = 8, w_i = (3, 4, 5), v_i = (30, 50, 60)$

このとき $i = 1, 3$ 番目の品物を入れたとき、価値が $30 + 60 = 90$ で最大になる

ナップサック問題

$\mathrm{dp}[i][j]$ を「i 番目までの荷物を入れて、重さを j 以下とするときの価値の最大値」と定義する

$\mathrm{dp}[0][0] = 0$ と初期化する

今のナップサックの容量が j のとき、i 番目の品物を入れることができるかの判定
$\Rightarrow j + w_i \leq W$ であること

入れられるとき、入れた結果の価値が更新できるなら更新（最大値を更新していく）
$\Rightarrow \mathrm{dp}[i][j] = \max\left(\mathrm{dp}[i-1][j], \mathrm{dp}[i-1][j-w_i] + v_i\right) \quad (j - w_i \geq 0)$

編集距離に関するDP

編集距離？

2 つの文字列 S, T の**類似度**を測るものを指します

編集距離に関するDP

編集距離

2 つの文字列 S, T が与えられ、S に次の 3 通りの操作を繰り返し施すことで T に変換したいとき、一連の操作の操作回数の最小値

- 変更：S 中の文字を 1 つ選んで任意の文字に変更する
  - 例）`JOI` $\rightarrow$ `IOI`
- 削除：S 中の文字を 1 つ選んで削除する
  - 例）`JOI` $\rightarrow$ `OI`
- 挿入：S の好きな箇所に好きな文字を 1 文字挿入する
  - 例）`JOI` $\rightarrow$ `JOIG`

編集距離に関するDP

編集距離の例

$S = \texttt{kodansha}, T = \texttt{danshari}$ の場合、次の 4 回の操作で S を T に変化できます

1. 削除：$\texttt{kodansha} \to \texttt{odansha}$
2. 削除：$\texttt{odansha} \to \texttt{dansha}$
3. 削除：$\texttt{dansha} \to \texttt{danshar}$
4. 削除：$\texttt{dansha} \to \texttt{danshari}$

編集距離に関するDP

編集距離の例

$S = \texttt{logistic}, T = \texttt{algorithm}$ の場合は？

ちょっと難しいので、 $S' = \texttt{log}, T' = \texttt{alg}$ で考えます
これは 2 回で可能です

- 挿入：$\texttt{log} \to \texttt{alog}$
- 削除：$\texttt{alog} \to \texttt{alg}$

編集距離に関するDP

編集距離の例

では $S'' = \texttt{logi}, T' = \texttt{alg}$ のときは？
これは 3 回で可能です

- 挿入：$\texttt{logi} \to \texttt{alogi}$
- 削除：$\texttt{alogi} \to \texttt{algi}$
- 削除：$\texttt{algi} \to \texttt{alg}$ （この操作が増える）

編集距離に関するDP

編集距離の例

では $S' = \texttt{log}, T'' = \texttt{algo}$ のときは？
これは 3 回で可能です

- 挿入：$\texttt{log} \to \texttt{alog}$
- 削除：$\texttt{alog} \to \texttt{alg}$
- 挿入：$\texttt{alg} \to \texttt{algo}$ （この操作が増える）

編集距離に関するDP

あれ？

$S'' = \texttt{logi}, T' = \texttt{alg}$ や $S' = \texttt{log}, T'' = \texttt{algo}$ のとき、
どちらも $S' = \texttt{log}, T' = \texttt{alg}$ から操作回数を 1 回増やせば求められた

「S の先頭 i 文字と T の先頭 j 文字の編集距離」は次の 2 つが求まれば求められる？

- 「S の先頭 $i-1$ 文字と T の先頭 j 文字の編集距離」
- 「S の先頭 i 文字と T の先頭 $j-1$ 文字の編集距離」

⇒ **部分問題に分解**できる

編集距離に関するDP

式で表す

$\mathrm{dp}[i][j]$ を「S の先頭 i 文字と T の先頭 j 文字の編集距離」と定義します。

$\mathrm{dp}[0][0] := 0$ とし（先頭 0 文字 $=$ 空文字列）、
その他は ∞ （十分大きな定数）で初期化します。
また、文字列 X の i 文字目を X_i と表記します

編集距離に関するDP

式で表す

1. 変更操作
   - $S_i = T_j$ のとき、編集距離を増やすことなく一致させられます
     - $\mathrm{dp}[i][j] = \min(\mathrm{dp}[i][j], \mathrm{dp}[i-1][j-1])$
   - $S_i \neq T_j$ のとき、S_i を T_j に変更することで一致させられます
     - $\mathrm{dp}[i][j] = \min(\mathrm{dp}[i][j], \mathrm{dp}[i-1][j-1]+1)$
2. 削除操作
   - $\mathrm{dp}[i-1][j]$ の状態から S_i を削除すれば一致させられます
     - $\mathrm{dp}[i][j] = \min(\mathrm{dp}[i][j], \mathrm{dp}[i-1][j]+1)$
3. 挿入操作
   - $\mathrm{dp}[i][j-1]$ の状態から T_j を挿入すれば一致させられます
     - $\mathrm{dp}[i][j] = \min(\mathrm{dp}[i][j], \mathrm{dp}[i][j-1]+1)$

編集距離に関するDP

実例

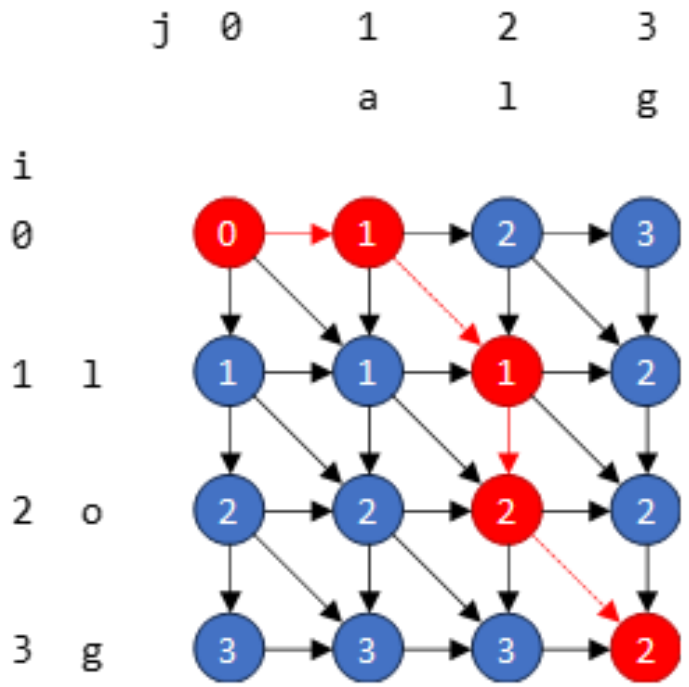
$S = \texttt{log}, T = \texttt{alg}$ の場合に、$\mathrm{dp}[3][3]$ を求める

- $\mathrm{dp}[1][0]$ は $S' = \mathrm{l}$ から 1 文字削除することで $T' = \phi$ に一致させられる
- $\mathrm{dp}[0][1]$ は $S' = \phi$ に 1 文字挿入することで $T' = \texttt{a}$ に一致させられる
- $\mathrm{dp}[1][1]$ は $S' = \phi$ から $\texttt{l}$ を $\texttt{a}$ に変更することで $T' = \texttt{a}$ に一致させられる

編集距離に関するDP

実例

結局 $\mathrm{dp}[i][j]$ は以下のようになり、答えは 2 と求められる



編集距離に関するDP

実際にソースコードを書いてみましょう

AtCoder上の問題 ⇒ https://atcoder.jp/contests/pastbook2022/tasks/pastbook2022_b

編集距離に関するDP

```
int main(){
  int M, N; cin >> M >> N;
  string S, T; cin >> S >> T;
  vector<vector<int>> dp(M + 1, vector<int>(N + 1, 1 << 30));
  dp[0][0] = 0;
  for(int i = 0; i <= M; ++i){
    for(int j = 0; j <= N; ++j){
      if(i != 0 && j != 0){
        if(S[i - 1] == T[j - 1]) chmin(dp[i][j], dp[i - 1][j - 1]);
        else chmin(dp[i][j], dp[i - 1][j - 1] + 1);
      }
      if(i != 0) chmin(dp[i][j], dp[i - 1][j] + 1);
      if(j != 0) chmin(dp[i][j], dp[i][j - 1] + 1);
    }
  }
  cout << dp[M][N] << endl;
}
```

区間に関するDP

> あるデータの列を、複数の区間に分割することを考える
>
> このとき、与えられたコストの最小値／スコアの最大値はどのようになるか？

みたいな問題を指します
俗に「区間DP」とも

区間に関するDP

例題 (https://atcoder.jp/contests/pastbook2022/tasks/pastbook2022_a)

N 個の要素が 1 列に並んでいて、これをいくつかの区間に分割したいものとします。各区間 $[l, r)$ にはスコア $c_{l,r}$ が定められているものとします。

K を N 以下の正の整数として、$K+1$ 個の整数 $t_0, t_1, \ldots, t_K$ を $0 = t_0 < t_1 < \cdots < t_K = N$ を満たすようにとったとき、区間分割 $[t_0, t_1), [t_1, t_2), \ldots, [t_{K-1}, t_K)$ のスコアを次のように定めます。

- $c_{t_0,t_1} + c_{t_1,t_2} + \cdots + c_{t_{K-1},t_K}$

区間分割の仕方をすべて考えたときの、考えられるスコアの最小値を求めてください。

制約：$1 \leq N \leq 1000, \quad 0 \leq c_{l,r} \leq 100$

区間に関するDP

ちょっと待て

- $[l, r)$ ← これはなんですか

半開区間と呼ばれる区間の表現の仕方です。

例えば、$A = (3, 1, 4, 1, 5, 9)$ としたとき、$[2, 4)$ は A の 2 番目から $4 - 1 = 3$ 番目の区間、すなわち $(1, 4)$ を指しているということを表しています。4 番目は含みません。

（つまり、左側を**含み**、右側を**含まない**区間ということです）

他にも、例えば $[1, 5)$ は $(3, 1, 4, 1)$ を、$[4, 7)$ は $(1, 5, 9)$ を表します。

区間に関するDP

サンプル

$N = 3$ で、$c_{i,j}$ $(i < j)$ の値は以下の通り

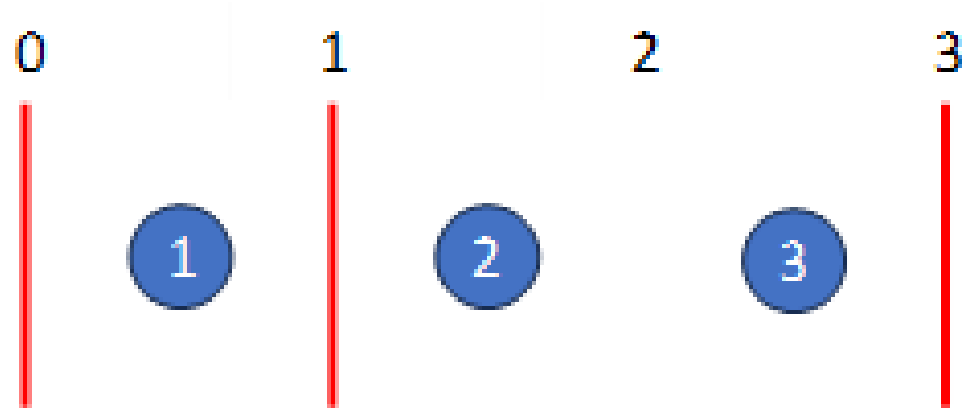
（使いませんが、入力が簡単のため $c_{j,i} = c_{i,j}, c_{i,i} = 0$ と定義します）

i＼j	0	1	2	3
0	0	7	1	6
1	7	0	5	3
2	1	5	0	4
3	6	3	4	0

区間に関するDP

サンプル

- 例えば、$K = 2$ として $t_i = (0, 1, 3)$ とすると以下の通り
- スコアは $c_{0,1} + c_{1,3} = 7 + 3 = 10$ となる

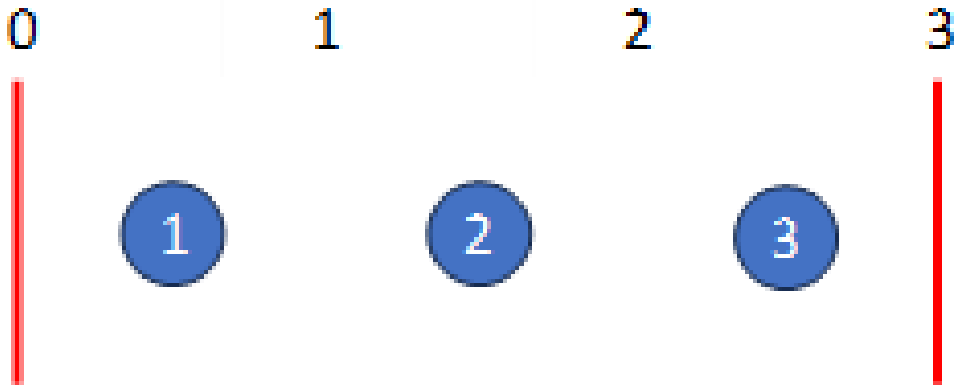


i＼j	0	1	2	3
0	0	7	1	6
1	7	0	5	3
2	1	5	0	4
3	6	3	4	0

区間に関するDP

サンプル

- $K = 1$ として $t_i = (0, 3)$ とすると以下の通り
- スコアは $c_{0,3} = 6$ となる

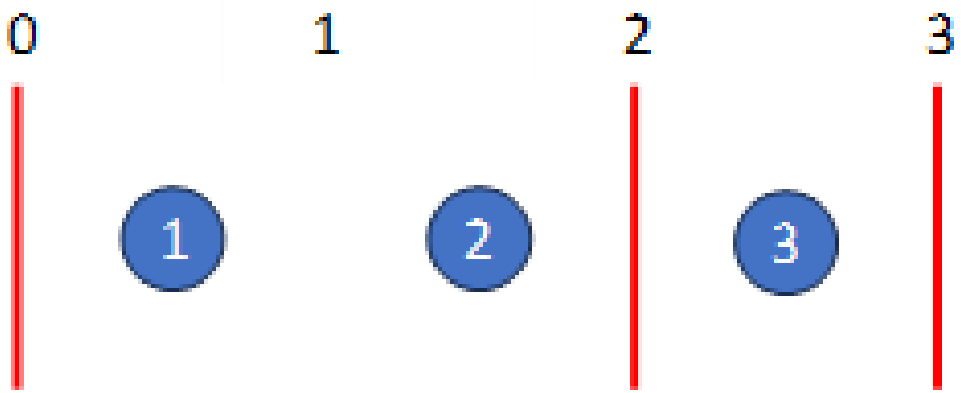


i＼j	0	1	2	3
0	0	7	1	6
1	7	0	5	3
2	1	5	0	4
3	6	3	4	0

区間に関するDP

サンプル

- $K = 2$ として $t_i = (0, 2, 3)$ とすると以下の通り
- スコアは $c_{0,2} + c_{2,3} = 1 + 4 = 5$ となる
- この例ではこれが最小値になる

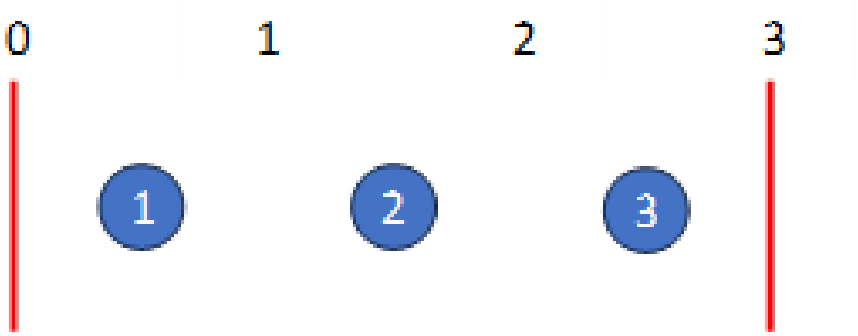


i＼j	0	1	2	3
0	0	7	1	6
1	7	0	5	3
2	1	5	0	4
3	6	3	4	0

区間に関するDP

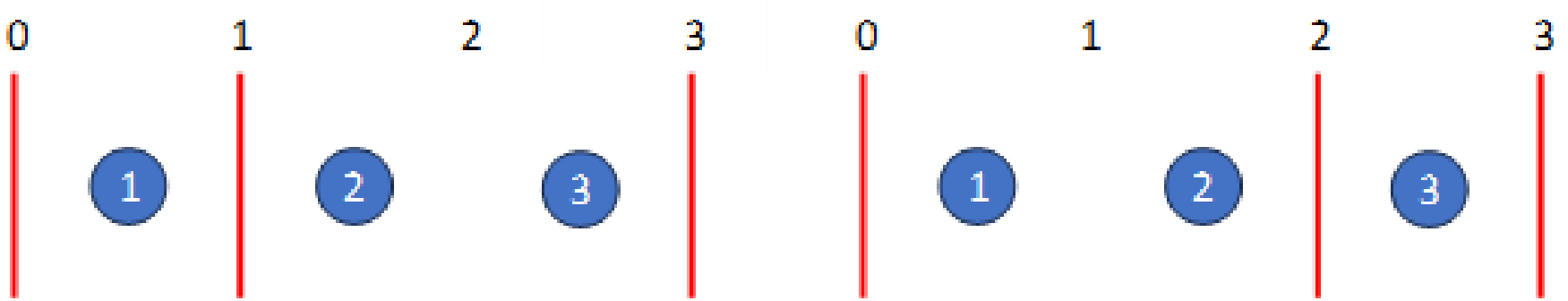
どう考えるか

とりあえず $K = 2$ として、2 つの区間に分けることを考えてみる



区間に関するDP

- 1 の位置に挿入したときのスコアは？
  - $[0, 1)$ のスコアに $c_{1,3}$ を加えたもの
- 2 の位置に挿入したときのスコアは？
  - $[0, 2)$ のスコアに $c_{2,3}$ を加えたもの



区間に関するDP

- 1 の位置に挿入したときのスコアは？
  - $[0, 1)$ のスコアに $c_{1,3}$ を加えたもの
  - $[0, 1)$ のスコアが分かっていれば計算できる
- 2 の位置に挿入したときのスコアは？
  - $[0, 2)$ のスコアに $c_{2,3}$ を加えたもの
  - $[0, 2)$ のスコアが分かっていれば計算できる

区間に関するDP

$\mathrm{dp}[i]$ を「区間 $[0, i)$ のスコアの最小値」と定義すると…？

- 1 の位置に挿入したときのスコアは？
  - $\mathrm{dp}[1] + c_{1,3}$
- 2 の位置に挿入したときのスコアは？
  - $\mathrm{dp}[2] + c_{2,3}$

つまり、 $\mathrm{dp}[3] = \min\left(\mathrm{dp}[1] + c_{1,3}, \mathrm{dp}[2] + c_{2,3}\right)$ と表せる

区間に関するDP

一般化

$\mathrm{dp}[i]$ を「区間 $[0, i)$ のスコアの最小値」と定義する
$\mathrm{dp}[0] = 0, \mathrm{dp}[i] = \infty \ (1 \leq i \leq N)$ と初期化

遷移式は $\mathrm{dp}[i] = \min_{0 \leq j < i} (\mathrm{dp}[j] + c_{j,i})$

答えは $\mathrm{dp}[N]$

※ $\min_{0 \leq j < i}$ は、「$0 \leq j < i$ を満たす j についての括弧の中身の最小値」という意味です

これを実装すると、計算量 $O(N^2)$ で答えを求めることができます

区間に関するDP

```
int main(){
  int N; cin >> N;
  vector<vector<int>> c(N + 1, vector<int>(N + 1));
  for(int i = 0; i <= N; ++i){
    for(int j = 0; j <= N; ++j){
      cin >> c[i][j];
    }
  }

  vector<int> dp(N + 1, 1 << 30);
  dp[0] = 0;
  for(int i = 1; i <= N; ++i){
    for(int j = 0; j < i; ++j){
      chmin(dp[i], dp[j] + c[j][i]);
    }
  }
  cout << dp[N] << endl;
}
```

練習問題

編集距離に関するDP

- EDPC F - LCS
- ABC185 E - Sequence Matching

区間に関するDP

- EDPC N - Slimes
- 鉄則本 B21 - Longest Subpalindrome
- ABC217 F - Make Pair
- ABC325 G - offence （難）

おわり

おつかれさまでした！