

2021 年度 卒業論文

# 理解と活用を重視した アルゴリズム学習教材の 作成

指導教員 須田 宇宙 准教授

千葉工業大学 情報ネットワーク学科  
須田研究室

1832138 氏名 松丸 颯汰

提出日 2022年1月31日

# 目次

<b>1</b>	<b>はじめに</b>	<b>2</b>
<b>2</b>	<b>関連研究</b>	<b>3</b>
2.1	アルゴリズムの学習支援 . . . . .	3
2.1.1	ソースコードを用いる学習支援 . . . . .	3
2.1.2	ソースコードを用いない学習支援 . . . . .	3
2.2	オンラインジャッジシステムの活用 . . . . .	4
2.2.1	オンラインジャッジシステムとは . . . . .	4
2.2.2	講義における活用 . . . . .	4
2.3	問題点 . . . . .	4
<b>3</b>	<b>実際のオンラインジャッジシステムの活用例</b>	<b>5</b>
3.1	AtCoder とは . . . . .	5
3.2	競技プログラミングとは . . . . .	5
3.3	AtCoder のコンテスト . . . . .	5
3.4	AtCoder におけるオンラインジャッジシステムの利用 . . . . .	6
3.5	オンラインジャッジシステムによる実行例 . . . . .	7
<b>4</b>	<b>アルゴリズム学習教材について</b>	<b>12</b>
4.1	プログラミングを含まない学習教材 . . . . .	12
4.2	プログラミングを含む学習教材 . . . . .	12
<b>5</b>	<b>作成した教材について</b>	<b>13</b>
5.1	対象とする学習者 . . . . .	13
5.2	目的とする利用方法 . . . . .	13
5.3	到達目標 . . . . .	13
5.4	題材としたアルゴリズム . . . . .	14
5.4.1	動的計画法とは . . . . .	14
5.5	利用環境 . . . . .	14
5.5.1	TechFUL とは . . . . .	14
5.6	利用する言語 . . . . .	15
5.7	教材の構成 . . . . .	16
5.7.1	導入 動的計画法とは . . . . .	16
5.7.2	1 章 動的計画法の考え方 . . . . .	17
5.7.3	2 章 数式をプログラミングする . . . . .	18
5.8	実際の運用形式 . . . . .	18

5.8.1	C 言語版セッション . . . . .	19
5.8.2	他言語版セッション . . . . .	20
<b>6</b>	<b>おわりに</b>	<b>22</b>
	<b>謝辞</b>	<b>23</b>
	<b>付録 A 作成した教材</b>	<b>25</b>
A.1	動的計画法とは . . . . .	25
A.2	動的計画法の考え方 . . . . .	26
A.2.1	導入 例題 (線形探索問題) の紹介 . . . . .	26
A.2.2	部分問題に分ける . . . . .	28
A.2.3	2 数の比較 . . . . .	29
A.2.4	問題の解説 . . . . .	30
A.2.5	Gist の解答例 . . . . .	31
A.2.6	計算結果をメモする . . . . .	32
A.2.7	問題を解く . . . . .	34
A.2.8	数列内の最大の数の出力 . . . . .	36
A.2.9	問題の解説 . . . . .	37
A.2.10	Gist の解答例 . . . . .	39
A.2.11	計算結果をメモする意味 . . . . .	41
A.3	数式をプログラミングする . . . . .	43
A.3.1	導入 例題 (フィボナッチ数列) の紹介 . . . . .	43
A.3.2	計算量を考える . . . . .	44
A.3.3	漸化式をプログラミングする . . . . .	47
A.3.4	フィボナッチ数列 . . . . .	48
A.3.5	問題の解説 . . . . .	49
A.3.6	Gist の解答例 . . . . .	51
A.3.7	トリボナッチ数列 . . . . .	53
A.3.8	問題の解説 . . . . .	54
A.3.9	Gist の解答例 . . . . .	55
	<b>付録 B C 言語版セッションで追加した項目</b>	<b>56</b>
B.1	はじめに . . . . .	56
B.2	C 言語入出力チートシート . . . . .	59
B.3	大きい数 初期プログラム . . . . .	62
B.4	最大の数 初期プログラム . . . . .	62
B.5	フィボナッチ数列 初期プログラム . . . . .	63
B.6	トリボナッチ数列 初期プログラム . . . . .	63

B.7	さいごに . . . . .	64
<b>付録 C</b>	<b>TechFUL で教材を表示した場合の例</b>	<b>66</b>
C.1	セクションを利用した際のイメージ . . . . .	66
C.2	教材と問題のイメージ . . . . .	67
C.3	コードテストを行った時のイメージ . . . . .	69
C.4	正誤判定を行った際のイメージ . . . . .	72

# 1 はじめに

近年、情報化に伴い大量のデータを扱う機会は増加している。大量のデータを処理するには、全てのパターンを計算することは困難であることが多いため、計算量を削減するために様々なアルゴリズムが活用されている。

データを扱うプログラムを書く際に役立つアルゴリズムは数多くある。これらのアルゴリズムを活用するためには、アルゴリズム名だけではなくそのアルゴリズムを利用して行える処理や計算量、実装方法など様々な知識を必要とする。

アルゴリズムの学習をする際には、理論のみを学習するのではなく実践も併せて行うことにより、知識が定着しやすい。実際に、ウェブ上にあるアルゴリズム学習教材は、コンテストサイト等を活用して問題を解くような形式となっているものが多い。

プログラミングを含むアルゴリズム学習教材にはアルゴリズムの実装を学習することがメインの教材が多い。また、本などの媒体の場合は具体的な問題等があってもプログラミング環境の構築や正誤判定などの面で実践が難しい場合がある。

大学の講義でもアルゴリズムを扱った講義はあるが、それらの講義ではアルゴリズムの知識の習得が主であり実際の活用やプログラミングなどを扱うことは少ない。また、講義内でプログラミングを扱う際にも、利用する言語が講義により異なっており、利用したことのない言語の場合にはアルゴリズムの学習に加えて言語の文法等の学習コストがかかる可能性がある。そのため、初学者が学習する際にはアルゴリズムの具体的な知識の習得と実践に特化した学習を行う機会は少なくなっている。

そこで本研究では、講義の課題や教材として利用できる、アルゴリズムの知識を深めた上でプログラミングによる実践を行うプロセスに特化した初学者向けの教材の作成を目的とする。

## 2 関連研究

関連研究は、アルゴリズムの学習支援に関する研究、オンラインジャッジシステムの活用に関する研究の 2 種類に分けることができる。

### 2.1 アルゴリズムの学習支援

コンピュータを用いた初学者に対するアルゴリズム学習支援は、ソースコードを用いるものと用いないものの 2 種類に分けることができる。

#### 2.1.1 ソースコードを用いる学習支援

新開ら [1] は問題を細かい問題に分解していくことによりアルゴリズムの構造を考え、それを基にプログラミングを行う手法を提案している。この手法を用いた実験の結果、制御構造の理解を深めるのに役立つという評価が得られた。

大城ら [2] はソースコードを基にアニメーションを作成する手法を提案している。ソースコードを実行すると、その処理がアニメーションとして実行される。アニメーション実行中はソースコード内の対応する行がハイライト表示になり、どこを実行しているのかが分かりやすくなっている。

#### 2.1.2 ソースコードを用いない学習支援

佐藤ら [3] はフローチャートを用いた自習システムを開発している。このシステムでは、問題に対して行う処理をフローチャートで記述し、その正しさを判定したり C 言語のソースプログラムに変換することが可能となっている。

阿部 [4] や大森ら [5] のようにアニメーションを利用した教材も多く作られている。また、Algorithm Visualizer[6] や VISUALGO[7] のようにアルゴリズムの動作をアニメーションで確認できる Web サイトも存在する。

## 2.2 オンラインジャッジシステムの活用

### 2.2.1 オンラインジャッジシステムとは

オンラインジャッジシステム (以下 OJS) は, 問題に対して予め用意された入力値と出力値の組み合わせを利用して, 提出されたコードをオンライン上で実行し, 自動で正誤判定を行うシステムのことである. OJS は AtCoder[8] や TopCoder[9] といった競技プログラミングサイトや paiza ラーニング [10] などのプログラミングやアルゴリズムの学習サイト等で利用されている.

また, OnlineJudge2.0[11] や Arrow Judge[12] など, オープンソースの OJS も公開されており, 個人でシステムを構築することも容易となっている.

### 2.2.2 講義における活用

長尾ら [13] は採点作業の効率化を目的としてプログラミング演習の講義で OJS を利用している. 従来の紙媒体を利用した場合と比較した結果, 採点作業の効率化は行えたが学習効果には大きな差は見られなかった.

松永 [14] はプログラミング演習の再履修講義で OJS を利用している. 元のプログラミング演習の講義では, 課題に対してプログラミングを行い, そのプログラムを教員が目視で採点する形式をとっている. 再履修講義では正確なプログラミング能力を徐々に獲得していくことを目的としており, 正確なプログラムを要求する OJS と相性はいいと考えられる. 結果は, OJS によりプログラミング能力を向上させることのできる学生は一部に留まり, OJS の利用のみでは効果を得ることは難しかった.

## 2.3 問題点

プログラミング教育における OJS の利用は検討されているが, アルゴリズム学習に特化した研究は行われていない. また, 従来の講義や課題のスタイルでは OJS の導入による学習効率の向上は見られておらず, OJS を利用した際に学習効率のいい教材の開発が必要となっている.

### 3 実際のオンラインジャッジシステムの活用例

OJS の活用例として、競技プログラミングサイト AtCoder[8] を紹介する。

#### 3.1 AtCoder とは

AtCoder は、「競技プログラミング」と呼ばれるコンテストを行うサービスである。日本国内の競技プログラミングコンテストサイトとしては最大規模であり、日本人登録者が 16 万人、海外を含め 30 万人以上の登録者がいる。また、毎週コンテストが開催されており、8,000 人以上が参加している。

プログラミング言語は C 言語, C++, Java, Python3, Rust を始めとした 40 以上の言語に対応しており、参加者は好きな言語を利用してコンテストに参加することができる。

#### 3.2 競技プログラミングとは

競技プログラミングは、様々なアルゴリズムや数学的知識を利用し、与えられた問題に適するプログラムを提出する速度を競うコンテストである。

AtCoder においては、コンテストで与えられる問題にそれぞれ得点が決まっており、問題に正解することで得点を得ることができる。そして、開催時間内に得た得点及びその提出時間により順位が決定される。誤ったプログラムを提出すると、1 回につき提出時間に 5 分のペナルティが追加されるため、より速く、正確にプログラミングを行うことが重要となる。

#### 3.3 AtCoder のコンテスト

AtCoder における定期開催コンテストは 4 種類あり、表 1 のようになっている。ABC, ARC, AGC では入出力が予め決まっており、プログラムを提出することにより正誤判定が行われる。AHC のみヒューリスティックと呼ばれる形式であり、最適解を求めることが難しい問題に対しより最適解に近い値を求めることを目的とするコンテストである。

コンテストではそれぞれの問題に対してサンプルとして 1 つから 3 つのテストケースが公開されており、ユーザーはこのテストケースを利用し、AtCoder 上のコードテストページや自分の開発環境でテストを行うことができる。

表 1 AtCoder の定期開催コンテスト

コンテスト名	難易度	時間	問題数
AtCoder Beginner Contest(ABC)	初心者～中級者向け	100 分	8 問
AtCoder Regular Contest(ARC)	初心者～上級者向け	120 分	6 問
AtCoder Grand Contest(AGC)	中級者～上級者向け	150～180 分	6 問
AtCoder Heuristic Contest(AHC)	初心者～上級者向け	240～480 分	1 問



### 3.4 AtCoder におけるオンラインジャッジシステムの利用

AtCoder では, 提出されたプログラムは OJS を利用し, 計算時間, メモリ使用量のそれぞれの制限内で処理が終わっているか, および出力結果が正しいかを判断したうえで正誤判定が行われる. 誤ったプログラムを提出した場合, CE(Compilation Error, コンパイルエラー), RE(Runtime Error, 実行時エラー), TLE(Time Limit Exceeded, 制限時間超過), WA (Wrong Answer, 出力結果が誤っている) など, どのような理由で間違っているのかが表示される.

### 3.5 オンラインジャッジシステムによる実行例

AtCoder ので 2022 年 1 月 8 日に行われたコンテスト, AtCoder Beginner Contest 234(ABC234) の過去問から, B 問題を解いた場合の状況を例に挙げる. ABC234 では, A~G, Ex の 8 問の問題が出題された. 問題の難易度は A 問題が一番易しく, Ex 問題が一番難しくなっており, B 問題は 2 番目に易しい問題である.

ABC234 の B 問題は以下のような問題となる.

B - Longest Segment

#### 問題文

二次元平面上に  $N$  個の点があります.  $i$  個目の点の座標は  $(x_i, y_i)$  です.

この中から 2 個の点を選ぶとき, それらを結ぶ線分の長さの最大値を求めてください.

#### 制約

- ・ 実行時間制限:2sec
- ・ メモリ制限:1024MB
- ・  $2 \leq N \leq 100$
- ・  $-1000 \leq x_i, y_i \leq 1000$
- ・  $(x_i, y_i) \neq (x_j, y_j) \ (i \neq j)$
- ・ 入力はすべて整数

#### 入力

入力は以下の形式で標準入力から与えられる.

$N$

$x_1 y_1$

$x_2 y_2$

:

$x_N y_N$

#### 出力

2 点を結ぶ線分の長さの最大値を出力せよ.

想定解との絶対誤差または相対誤差が  $10^{-6}$  以下であれば正解とみなされる.



例として、問題に対して提出したコードを時系列順に図 1, 図 3, 図 5 に示す。また、それぞれの提出結果を図 2, 図 4, 図 6 に示す。

図 1 のコードには 2 点の問題があり、配列の参照エラーと計算ミスがある。このコードを提出した結果、図 2 のように実行時エラーが発生した。

```
1  #!/usr/bin/env python3
2
3  N = int(input())
4  XY = [list(map(int, input().split())) for _ in range(N)]
5
6  ans = 0
7
8  for i in range(N):
9      // 配列外の数値を参照している
10     for j in range(i + 1, N + 1):
11         lx = XY[i][0]; ly = XY[i][1]
12         rx = XY[j][0]; ry = XY[j][1]
13         ans = max(ans, (rx - lx) ** 2 + (ry - ly) ** 2)
14
15 print(ans)
```

図 1 配列参照を誤った提出コード

## 提出情報

提出日時	2022-01-27 01:55:38
問題	B - Longest Segment
ユーザ	hanayuki7793 
言語	Python (3.8.2)
得点	0
コード長	317 Byte
結果	
実行時間	24 ms
メモリ	8912 KB

## ジャッジ結果



セット名	Sample	All
得点 / 配点	0 / 0	0 / 200
結果	 × 2	 × 13

図 2 配列参照を誤った提出コードの提出結果

ソースコードを確認すると、10 行目で要素数が N の配列 XY に対し、添字 N でアクセスすることにより配列の参照エラーが発生していた。そこで、8 行目の N を N-1 に、10 行目の N+1 を N に変更し図 3 のコードを提出した。提出した結果、図 4 のように計算結果が誤っていることが分かった。


```

1  #!/usr/bin/env python3
2
3  N = int(input())
4  XY = [list(map(int, input().split())) for _ in range(N)]
5
6  ans = 0
7
8  for i in range(N - 1):
9      for j in range(i + 1, N):
10         lx = XY[i][0]; ly = XY[i][1]
11         rx = XY[j][0]; ry = XY[j][1]
12         ans = max(ans, (rx - lx) ** 2 + (ry - ly) ** 2)
13
14  // 平方根を求め忘れている
15  print(ans)

```

図 3 計算を誤った提出コード

#### 提出情報

提出日時	2022-01-27 01:56:08
問題	B - Longest Segment
ユーザ	hanayuki7793 
言語	Python (3.8.2)
得点	0
コード長	317 Byte
結果	WA
実行時間	28 ms
メモリ	9012 KB

#### ジャッジ結果

セット名	Sample	All
得点 / 配点	0 / 0	0 / 200
結果	WA × 2	AC × 1 WA × 12

図 4 計算を誤った提出コードの提出結果

問題を確認すると、2 点を結ぶ線分の長さを求める問題となっており、答えは平方根を取る必要があることが分かった。そこで、15 行目の計算の修正を行い図 5 のコードを提出した。


```

1  #!/usr/bin/env python3
2
3  import math
4
5  N = int(input())
6  XY = [list(map(int, input().split())) for _ in range(N)]
7
8  ans = 0
9
10 for i in range(N - 1):
11     for j in range(i + 1, N):
12         lx = XY[i][0]; ly = XY[i][1]
13         rx = XY[j][0]; ry = XY[j][1]
14         ans = max(ans, (rx - lx) ** 2 + (ry - ly) ** 2)
15
16 print(math.sqrt(ans))

```

図 5 正しい提出コード

#### 提出情報

提出日時	2022-01-27 01:56:40
問題	B - Longest Segment
ユーザ	hanayuki7793 
言語	Python (3.8.2)
得点	200
コード長	343 Byte
結果	AC
実行時間	29 ms
メモリ	8920 KB

#### ジャッジ結果

セット名	Sample	All
得点 / 配点	0 / 0	200 / 200
結果	AC × 2	AC × 13

図 6 正しい提出コードの提出結果

以上のように、提出結果を確認することによりどのようなエラーが発生しているかが分かり、それを手掛かりにデバッグを行うことができる。また、それまでの提出結果は図7のように全て参照することが出来るようになっており、過去に提出したコードも確認することができる。

提出日時	問題	ユーザ	言語	得点	コード長	結果	実行時間	メモリ	
2022-01-27 01:56:40	B - Longest Segment	hanayuki7793 Q	Python (3.8.2)	200	343 Byte	AC	29 ms	8920 KB	詳細
2022-01-27 01:56:08	B - Longest Segment	hanayuki7793 Q	Python (3.8.2)	0	317 Byte	WA	28 ms	9012 KB	詳細
2022-01-27 01:55:38	B - Longest Segment	hanayuki7793 Q	Python (3.8.2)	0	317 Byte	RE	24 ms	8912 KB	詳細
2022-01-08 22:07:37	E - Arithmetic Number	hanayuki7793 Q	Python (3.8.2)	500	1277 Byte	AC	27 ms	9144 KB	詳細
2022-01-08 22:02:54	E - Arithmetic Number	hanayuki7793 Q	Python (3.8.2)	0	1018 Byte	WA	30 ms	9124 KB	詳細
2022-01-08 21:29:02	D - Prefix K-th Max	hanayuki7793 Q	Python (3.8.2)	400	791 Byte	AC	446 ms	69104 KB	詳細
2022-01-08 21:07:00	C - Happy New Year!	hanayuki7793 Q	Python (3.8.2)	300	665 Byte	AC	25 ms	8984 KB	詳細
2022-01-08 21:04:53	B - Longest Segment	hanayuki7793 Q	Python (3.8.2)	200	698 Byte	AC	28 ms	9192 KB	詳細
2022-01-08 21:02:30	A - Weird Function	hanayuki7793 Q	Python (3.8.2)	100	621 Byte	AC	26 ms	9168 KB	詳細

図7 提出履歴

## 4 アルゴリズム学習教材について

一般に利用されているアルゴリズム学習教材は、プログラミングを含むものとそうでないものに大別できる。

### 4.1 プログラミングを含まない学習教材

プログラミングを含まないものには、知識の習得をメインとする教科書や本などを用いた学習や、コンピュータサイエンスアンプラグド (以下 CS アンプラグド) と呼ばれるカードなどの教具を用いて直感的にアルゴリズムを理解することを目的とするものなどがある。

CS アンプラグドに関しては、小学校におけるプログラミング学習の必修化などに伴い、主に小学生や中学生を対象とする低年齢向け教材の研究が活発に行われている。また、低年齢向けであることから高度なアルゴリズムを対象としたものは少なく、主に二進数やソートアルゴリズムなどの基本となるアルゴリズムの学習を目的とした教材が多い。

### 4.2 プログラミングを含む学習教材

プログラミングを含むものには、本などの教材に問題が載っていてそれを解く形式のものや、オンライン上の教材で OJS による実行環境があり、その場で正誤判定を行えるものなどがある。また、競技プログラマ向けの本であるプログラミングコンテストチャレンジブック [15] などでは、実際に Web 上の OJS に掲載されている問題を取り扱うなど、OJS を利用して正誤判定ができる紙媒体の教材もある。

アルゴ式 [16] や paiza ラーニング [10] などのウェブ上の教材では OJS を利用する形式となっている。これは自分の書いたコードの正誤判定を実際の数値を用いて判定することができるため、人の目による判定よりもより確実に速いことが理由と考えられる。

## 5 作成した教材について

### 5.1 対象とする学習者

本学において、発展的なアルゴリズムの講義が行われるのは2年次後期である4Sからである。本学においては、2年次の前期までにC言語を利用したプログラミング演習の講義が行われており、アルゴリズムの学習者はプログラミングにおける基礎的な構文である標準入出力やループ、条件分岐などについてある程度理解していると考えられる。

よって、本教材の対象とする学習者は、プログラミング言語に対する最低限の知識を有し、アルゴリズムに関する知識をあまり持たない者とした。

### 5.2 目的とする利用方法

本教材は、講義においてプログラミングを行う機会が少ない点を解消することを目的としている。よって、講義においてダイクストラ法などの類似アルゴリズムや動的計画法を扱う場合や、講義内でプログラミングを行う機会がない場合の自習や課題での利用を目的としている。

### 5.3 到達目標

本教材の利用による到達目標として、アルゴリズムの考え方や実装方法がある程度わかり、問題を見た際に学習したアルゴリズムが利用できるかどうかを考えることができるようことを目標としている。

アルゴリズムを利用した実装を行う際に必要となるのは、問題に合わせたアルゴリズムを考えることであり、データ構造と利用するアルゴリズムが分かれば検索等を利用することによりアルゴリズムの実装は容易であることが多い。そのため、どのような場合に利用できるのか、という点への理解を重視している。



## 5.4 題材としたアルゴリズム

初学者を対象とした教材であることから、題材とするアルゴリズムは処理の流れが理解しやすく、今後学習する様々なアルゴリズムを理解する上で土台となるようなものが望ましい。

動的計画法はデータ処理において効率化を行う際に多く用いられるアルゴリズムであり、ダイクストラ法などの大学の講義において学ぶアルゴリズムとも大きく関係する。また、1次元の動的計画法は配列を利用すれば変化を考えやすく、初学者が学ぶのに適したアルゴリズムであると考えられる。

そこで、本教材では動的計画法を題材とした。

### 5.4.1 動的計画法とは

動的計画法 (以下 DP) は、求めたい問題を複数の部分問題に分け、部分問題を計算・記録しながら解いていくことにより元の問題の答えを導く手法のことである。DP の特徴として、部分問題への分割、計算結果の記録という 2 点がある。部分問題への分割により同じ問題が現れた場合、以前の計算結果を再利用することが可能となり、これにより計算量を減らすことができる。

動的計画法にはトップダウン方式とボトムアップ方式の 2 種類の実装方法がある。

トップダウン方式では、再帰関数のように元の問題から順により細かい部分問題を解いていく。再帰関数に動的計画法を適用した場合、メモ化再帰と呼ばれることもある。

ボトムアップ方式では、トップダウン方式とは逆に最も細かい部分問題から順に解いていく形式となっている。

## 5.5 利用環境

本教材では、構成の都合上テキスト、問題の両方を参照でき、OJS を利用できる環境が必要となる。2.2.1 で述べたように、OJS は自分で環境を作成することもできるが、自習や課題を目的としている点から、講義内等の限定的な時間の利用ではなく 24 時間の運用が必要となり、維持コストがかかる。以上の点から、テキストと問題を利用でき、OJS による正誤判定が行える TechFUL[17] を利用することとした。

### 5.5.1 TechFUL とは

TechFUL は、Web サイト上に用意されたプログラミング問題を解くことにより、IT エンジニアのスキルとプログラミング能力の測定や評価、学習などが行えるプラットフォームである。プログラムの判定には OJS が利用されている。

教育機関向けの契約を行うことにより授業、テスト、課題、コンテストが自由に開催できるようになる。また、教材や問題の登録も行えるようになる。教材や問題は、テキストか PDF を利用し作成することができる。テキストでは Markdown と一部の TeX 記法が利用できる。

本学では TechFUL の教育機関向けシステムが利用可能であったことから、本教材において利用することとした。

## 5.6 利用する言語

TechFUL で利用できる言語の一覧を表 2 に示す. 本教材のテキスト中で例として利用する言語は, 大学のプログラミング演習の講義においてほとんどの学生が利用したことがあると考えられるため C 言語とした.

問題の解答や解説に関しては, 学習者の中にも C 言語以外を利用する者がいる可能性を考慮し, C++, Python3, Java についても作成した. また, それ以外の言語に関しても模範解答はないが利用できるようにした.

表 2 TechFUL で利用できる言語

言語名	バージョン
PHP7.3	PHP 7.3.26
Java11	OpenJDK 11.0.5
C11	GCC 9.3.0
C++17	GCC 9.3.0
Python2.7	2.7.16
Python3	3.7.4
JavaScript	Node.js14.2.0
Go 1.14	1.14.2
Scala	2.12.8
Rust1.39	1.39.0
Ruby2.6	2.6.6
C # 7	Mono 6.12.0.107
Objective-C	Clang 9.0.1
Perl5	5.30.3
Swift	5.2.3
Kotlin	1.3.40
Racket / Scheme	7.8
R	3.6.2
SQLite3	

## 5.7 教材の構成

教材の構成を表 3 に示す。教材はテキストと問題から構成されており、基本的には前から順に参照する形となっている。

表 3 教材の構成

種類	章	内容
テキスト		動的計画法とは
	1 章	動的計画法の考え方
テキスト	1 節	導入 例題 (線形探索問題) の紹介
テキスト	2 節	部分問題に分ける
問題		2 数の比較
テキスト		問題の解説
テキスト	3 節	計算結果をメモする
テキスト	4 節	問題を解く
問題		数列内の最大の数の出力
テキスト		問題の解説
テキスト	Ex	計算結果をメモする意味
	2 章	数式をプログラミングする
テキスト	1 節	導入 例題 (フィボナッチ数列) の紹介
テキスト	2 節	計算量を考える
テキスト	3 節	漸化式をプログラミングする
問題		フィボナッチ数列
テキスト		問題の解説
問題		トリボナッチ数列
テキスト		問題の解説

### 5.7.1 導入 動的計画法とは

この章では、アルゴリズムとは何か、アルゴリズムを知ることの意味、DP の説明の 3 つについて説明を行った。

本教材における DP は、以下の 3 つのステップを利用し考えていく形式とした。

1. (部分問題に) 分ける
2. (計算結果を) メモる
3. (メモを元に) 求める

### 5.7.2 1章 動的計画法の考え方

この章では、DP を利用して例題を解くことにより、各ステップにおける考え方を順を追って説明した。

利用した例題は以下の問題である。線形探索問題と呼ばれる問題だが、前から順に最大値を更新していく手順は動的計画法の考え方と同じであるため、例題として利用した。

#### 例題 1

##### 問題文

長さ  $N$  の整数列  $A = A_0, A_1, \dots, A_{N-1}$  が与えられます。  
 $A$  に含まれる最大の数求めなさい。

##### 制約

- $2 \leq N \leq 10^5$
- $-10^6 \leq A_i \leq 10^6$

##### 入力

入力は以下の形式で標準入力から与えられる。

$N$

$A_0 A_1 \dots A_{N-1}$

##### 出力

$A$  に含まれる最大の数 を 1 行で出力せよ。

教材における流れは、はじめにこの問題を部分問題に分割した。この問題では数列内の最大の数を求めるが、コンピュータにおいて大量の数から 1 度に 1 つの数を見つけ出すことはできない。そこで、2 つの数と比較することを繰り返し行い、最終的に全ての要素について比較を行う、という手順を部分問題として紹介した。

次に、動的計画法の主要な考え方である計算結果のメモ化について説明した。本教材では計算の流れがわかりやすいように配列にメモを残す形式を利用し、メモを残す配列の宣言やどのような手順でメモが更新されるのかについて紹介した。

最後に、具体的な数値を用いてどのようにメモ配列が更新されていくのかを確認した。また、練習問題として 2 数の比較を行うこと、配列内の最大値を求めることを扱った。

### 5.7.3 2章 数式をプログラミングする

この章では、漸化式のプログラミングを例に DP の利用について説明した。

例題として、フィボナッチ数列を取り扱った。フィボナッチ数列は以下のような漸化式で表される数列である。

- $F_0 = 0$
- $F_1 = 1$
- $F_{n+2} = F_n + F_{n+1} (n \geq 0)$

はじめに再帰関数による実装を紹介し、その計算量を確認した。その後、メモ化を行うことによる計算量の効率化について紹介した。本教材では、ボトムアップ方式による DP の方がより初心者が理解しやすいと考え、再帰関数による実装及びそのメモ化については紹介程度に留めた。

次に、漸化式を変形することにより部分問題として見ることを確認した。そして、その部分問題を基に問題を解く形式とした。練習問題として、フィボナッチ数列とトリボナッチ数列を取り扱った。

## 5.8 実際の運用形式

TechFUL において、教材を運用する際の実際の構成について作成した。TechFUL では、言語を指定した穴埋め形式の問題を作成できる。そのため、C 言語に関しては入出力など最低限の部分を実装したファイルに対し穴埋め形式の問題を作成することにより、よりアルゴリズム学習のみに集中できるようにした。

### 5.8.1 C 言語版セクション

C 言語版セクションの構成を表 4 に示す。教材の内容に追加して、教材の説明、入出力方法の説明、アルゴリズム学習に役立つ本や Web サイト等の紹介した。また、問題は全て穴埋め形式とし、主に入力部分のみ予め実装された形式とした。TechFUL では穴埋め形式問題について、特定の行を変更禁止にできる機能があるが、本教材では全ての行について変更を行えるようにした。そのため、学習者が変数名を変更したり、main 関数の前に別の関数を追加したりすることも可能である。

例として、問題「大きい数」で利用したコードを図 8 に示す。この問題では 2 つの変数  $X, Y$  が与えられるため、予めその部分の入力のみを実装した。

表 4 C 言語版セクションにおける教材の構成

種類	タイトル	内容
テキスト	はじめに	教材の説明
テキスト	C 言語入出力チートシート	主要な入出力の方法一覧
テキスト	導入	動的計画法とは
テキスト	1. 動的計画法の考え方 1	導入 例題 (線形探索問題) の紹介
テキスト	1. 動的計画法の考え方 2	部分問題に分ける
問題	大きい数	2 数の比較
テキスト	大きい数 解説	問題の解説
テキスト	1. 動的計画法の考え方 3	計算結果をメモする
テキスト	1. 動的計画法の考え方 4	問題を解く
問題	最大の数	数列内の最大の数の出力
テキスト	最大の数 解説	問題の解説
テキスト	1. 動的計画法の考え方 Ex	計算結果をメモする意味
テキスト	2. 数式をプログラミングする 1	導入 例題 (フィボナッチ数列) の紹介
テキスト	2. 数式をプログラミングする 2	計算量を考える
テキスト	2. 数式をプログラミングする 3	漸化式をプログラミングする
問題	フィボナッチ数列	フィボナッチ数列
テキスト	フィボナッチ数列 解説	問題の解説
問題	トリボナッチ数列	トリボナッチ数列
テキスト	トリボナッチ数列 解説	問題の解説
テキスト	さいごに	他のアルゴリズム学習サイト等の紹介

```

1  /* CやC++などシェルに実行結果コード返却を明示する言語を利用する場合 基本的に0
   /* を返却してください.  */
2  #include <stdio.h>
3
4  int main(void)
5  {
6      /* 入力 */
7      int x, y;
8      scanf( "%d %d ", &x, &y );
9
10
11     return 0;
12 }

```

図8 問題「大きい数」におけるC言語のコード

### 5.8.2 他言語版セクション

他言語版セクションの構成を表5に示す。C言語版セクションとは異なり、利用する言語が不明であることから入出力の説明を行っていない。また、問題の形式も通常のものとなっている。それ以外の部分はC言語と共通のものであり、教材中の例として利用しているコードはC言語である。

表 5 他言語版セクションにおける教材の構成

種類	タイトル	内容
テキスト	はじめに	教材の説明
テキスト	導入	動的計画法とは
テキスト	1. 動的計画法の考え方 1	導入 例題 (線形探索問題) の紹介
テキスト	1. 動的計画法の考え方 2	部分問題に分ける
問題	大きい数	2 数の比較
テキスト	大きい数 解説	問題の解説
テキスト	1. 動的計画法の考え方 3	計算結果をメモする
テキスト	1. 動的計画法の考え方 4	問題を解く
問題	最大の数	数列内の最大の数の出力
テキスト	最大の数 解説	問題の解説
テキスト	1. 動的計画法の考え方 Ex	計算結果をメモする意味
テキスト	2. 数式をプログラミングする 1	導入 例題 (フィボナッチ数列) の紹介
テキスト	2. 数式をプログラミングする 2	計算量を考える
テキスト	2. 数式をプログラミングする 3	漸化式をプログラミングする
問題	フィボナッチ数列	フィボナッチ数列
テキスト	フィボナッチ数列 解説	問題の解説
問題	トリボナッチ数列	トリボナッチ数列
テキスト	トリボナッチ数列 解説	問題の解説
テキスト	さいごに	他のアルゴリズム学習サイト等の紹介



## 6 おわりに

本研究では、動的計画法を使用できる状況や考え方、その応用までの過程について、基礎となる知識や実装を細かいステップで学習できる教材を作成した。

どの程度のステップで教材を作成すると初学者にとってモチベーションと理解度のバランスがいか、問題数や難易度のバランス、ページ数によるモチベーションの変化等についての研究が今後の課題となる。

## 謝辞

本論文の執筆にあたりご指導くださった須田先生に感謝申し上げます。また、研究室のメンバーには研究についての多くの指摘やアドバイスを頂きました。本当にありがとうございました。

## 参考文献

- [1] 新開 純子, 炭谷 真也, ‘プロセスを重視したプログラミング教育支援システムの開発’, 日本教育工学会論文誌 31 巻 (2007)Suppl. 号, pp.45-48, 2007
- [2] 大城 正典, 永井 保夫, ‘初学者向けプログラミング学習のための初等アルゴリズム視覚化システム’, 情報シンポジウム 2018 年 8 月, pp.104-111, 2018
- [3] 佐藤 寛修, 阿部 清彦, 大山 実, 大井 尚一, ‘コンピュータプログラミング学習のためのアルゴリズム自習システム’, 工学教育 60 巻 (2012)4 号, pp.91-96, 2012
- [4] 阿部 哲也, ‘アルゴリズム教育のための教材の開発’, 日本教育工学雑誌 27 巻 (2003) suppl 号 pp.45-48, 2003
- [5] 大森 康正, 上野 晴樹, ‘アルゴリズムアニメーションを用いたプログラミング教育システム’, 情報処理学会全国大会講演論文集 第 53 回 (コンピュータと人間社会), pp.287-288, 1996
- [6] ‘Algorithm Visualizer’, <https://algorithm-visualizer.org/>
- [7] ‘VISUALGO’, <https://visualgo.net/>
- [8] ‘AtCoder’, <https://atcoder.jp/>
- [9] ‘TopCoder’, <https://www.topcoder.com/>
- [10] ‘paiza’, <https://paiza.jp/>
- [11] ‘OnlineJudge2.0’, <https://github.com/QingdaoU/OnlineJudge>
- [12] ‘Arrow Judge’, <https://github.com/hiromu/arrow-judge>
- [13] ‘オンラインジャッジシステムのプログラミング演習への導入と評価’, 情報処理学会 第 78 回全国大会講演論文集, pp.537-538, 2016
- [14] ‘導入プログラミング教育におけるオンラインジャッジシステムの活用の試み’, 専修大学情報科学研究所 情報科学研究 31 号, pp.25-41, 2011
- [15] 秋葉 拓哉, 岩田 陽一, 北川 宜稔, 2012, ‘プログラミングコンテストチャレンジブック [第 2 版] ～問題解決のアルゴリズム活用力とコーディングテクニックを鍛える～’, マイナビ出版
- [16] ‘アルゴ式’, <https://algo-method.com/>
- [17] ‘TechFUL’, <https://techful-programming.com/>

## 付録 A 作成した教材

### A.1 動的計画法とは

#### そもそもアルゴリズムって？

---

アルゴリズムとは、何かの処理を行う際の手順を表すものです。これはよく料理のレシピに例えられます。

料理をする時、多くの人はレシピを参考にします。料理のレシピには材料や分量、調理の手順などが載っていて、料理をするときにはそのレシピの通りに調理をしてやれば美味しい料理を作ることができます。材料や分量が同じでも、調味料や手順を間違えると、カレーを作ろうとしたのに肉じゃがになったりシチューになったり、はたまた食べられないダークマターのようなものになったりしてしまいます。

プログラミングではこの材料や分量にあたる部分がデータ、調理の手順がアルゴリズムと呼ばれています。

#### アルゴリズムを知る、ということ

---

僕たちは普段から様々な料理と親しんでいるので美味しい食べ方のできるレシピを何となくでイメージすることも調べることもできます。たとえば「魚を食べたい!」と思ったら、刺身、湯がく、茹でる、煮る、焼く、揚げる、蒸す、燻す…と色々な調理法が思い浮かびます。

アルゴリズムの場合も同じように、様々なアルゴリズムを「知って」いれば何となく使えそうなアルゴリズムをイメージしたり調べたりすることができるようになります。

本教材では動的計画法というアルゴリズムのみを扱いますが、もし興味があれば是非色々なアルゴリズムを「知って」「使って」みてください。

#### 動的計画法とは

---

動的計画法 (Dynamic Programming : 以下 DP) は、求めたい問題を複数の部分問題に分け、部分問題を計算・記録しながら解いていくことにより元の問題の答えを導く手法のことです。

…と言ってもイマイチ分かりづらいので、本教材での DP は

- ① (部分問題に) 分ける
- ② (計算結果を) メモる
- ③ (メモを元に) 求める

の 3 ステップで考えていきたいと思います。

## A.2 動的計画法の考え方

### A.2.1 導入 例題 (線形探索問題) の紹介

まずは動的計画法の 3 ステップ

①分ける

②メモる

③求める

について見ていきます。

この章では次の問題について考えます。

---

#### 例題 1

長さ  $N$  の整数列  $A = \{A_0, A_1, \dots, A_{N-1}\}$  が与えられます。

$A$  に含まれる最大の数求めなさい。

•  $1 \leq N \leq 10^5$

•  $-10^6 \leq A_i \leq 10^6$

入力

$N$

$A_0 A_1 \dots A_{N-1}$

---

いわゆる線形探索法と呼ばれる問題です。

一応、以下に線形探索における解答コードを載せておきます。

読むとヒントになるかもしれませんが、別に読まなくても大丈夫です。

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     /* n:要素数 a:入力される整数列 */
6     int n, a[100000];
7     /* res:答えを格納する変数 */
8     /* 最大値を求める問題なので初期値を十分に小さい数にする */
9     int res = - 2 << 30;
10
11     /* 入力 */
12     scanf( "%d ", &n);
13
14     for(int i = 0; i < n; i++){
15         scanf( "%d ", &a[i]);
16     }
17     /* 入力ここまで */
18
19     /* aの各要素とresを比較し、a[i]の方が大きい場合resを更新する */
20     for(int i = 0; i < n; i++){
21         if(res < a[i]){
22             res = a[i];
23         }
24     }
25
26     printf( "%d ", res);
27
28     return 0;
29 }

```

線形探索による解法では、求める数に対して十分に小さい値を初期値とし、配列の全ての要素と順番に比較していくことにより求められます。

今回はこの問題を DP の考え方をもとに解いていきます。

### A.2.2 部分問題に分ける

---

#### 例題 1

長さ  $N$  の整数列  $A = \{A_0, A_1, \dots, A_{N-1}\}$  が与えられます。

$A$  に含まれる最大の数を求めなさい。

•  $1 \leq N \leq 10^5$

•  $-10^6 \leq A_i \leq 10^6$

入力

$N$

$A_0 A_1 \dots A_{N-1}$

---

まずは最初のステップとして、①分けるについて考えます。

今回の問題の場合、複数の数字を比較して最大のものを選ぶ必要があります。

僕たちの場合は適当に数字が並んでたらその中から一番大きいものを何となくで選べますが、プログラムでは2つのものしか比較ができません。

そこで、この問題を以下のような部分問題に分割してみます。

- $A_0$  と  $A_1$  のうち大きいものを求める
- $A_1$  までの最大値と  $A_2$  のうち大きいものを求める
- $A_2$  までの最大値と  $A_3$  のうち大きいものを求める
- :
- $A_{N-2}$  までの最大値と  $A_{N-1}$  のうち大きいものを求める

実は、これをそのまま実装すると線形探索のコードと同じものが出来上がるんですけど今回はもうちょっと工程を増やします。

次の節では②メモるについて見ていきますが、まずは C 言語の文法の復習がてら次の練習問題を解いてみてください。

### A.2.3 2数の比較

## 問題文

---

整数  $X, Y$  が与えられます。

2つの整数のうち、大きいほうの整数を教えてください。

## 制約

---

- $-100 \leq X, Y \leq 100$
- $X \neq Y$

## 入力

---

入力は以下の形式で標準入力から与えられる。

$XY$

## 出力

---

1行で、大きいほうの整数を出力せよ。



#### A.2.4 問題の解説

プログラムで2つの値を比較する場合にはif文による分岐が利用できます。  
よって、以下のいずれかの方針で実装することができます。

- $X$ の方が大きければ  $X$  を、それ以外の場合は  $Y$  を出力する
- $X$ の方が小さければ  $Y$  を、それ以外の場合は  $X$  を出力する
- $Y$ の方が大きければ  $Y$  を、それ以外の場合は  $X$  を出力する
- $Y$ の方が小さければ  $X$  を、それ以外の場合は  $Y$  を出力する

解答例は以下のようになります。

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     /* 入力 */
6     int x, y;
7     scanf( "%d %d ", &x, &y);
8
9     if(x > y){
10         printf( "%d ", x);
11     }else{
12         printf( "%d ", y);
13     }
14
15     return 0;
16 }
```

また、max() などの最大値を返す関数を別に実装すると main 関数内での記述がわかりやすくなります。

```
1 #include <stdio.h>
2
3 int max(int x, int y) {
4     return x > y ? x : y;
5 }
6
7 int main(void) {
8     /* 入力 */
9     int x, y;
10    scanf( "%d %d ", &x, &y);
11
12    printf( "%d ", max(x, y));
13
14    return 0;
15 }
```

Java/Python/C++ の解答例は以下を確認してください。

解答例 (Gist)[[リンク](#)]

### A.2.5 Gist の解答例

基本的にすべての言語に max 関数が用意されているためそれを利用することができます。  
max 関数を利用しない場合は C 言語と同様の実装をしてください。

#### C++ による解答例

```
1 // C++の場合
2 // 実行環境ではbits/stdc++.hのインクルードが可能ですが、
3 // VisualC++等の環境で開発を行っている場合は手元でコンパイルできない可能性が
4 // ヘッダファイルをそれぞれインクルードしてください
5
6 #include <bits/stdc++.h>
7 // #include <algorithm>
8 // #include <iostream>
9
10 using namespace std;
11
12 int main(void) {
13     int x, y;
14     cin >> x >> y;
15
16     cout << max(x, y) << endl;
17
18     return 0;
19 }
```

#### Java による解答例

```
1 // Javaの場合
2 // クラス名をMainとする必要があります
3
4 import java.util.*;
5
6 public class Main{
7     public static void main(String[] args) {
8         int x, y;
9         Scanner sc = new Scanner(System.in);
10         x = Integer.parseInt(sc.next());
11         y = Integer.parseInt(sc.next());
12
13         System.out.println(Math.max(x, y));
14     }
15 }
```

#### Python3 による解答例

```
1 # Pythonの場合
2 # x if x > y else y等の記述でも可能です
3
4 x, y = map(int, input().split())
5
6 print(max(x, y))
```

### A.2.6 計算結果をメモする

---

#### 例題 1

長さ  $N$  の整数列  $A = \{A_0, A_1, \dots, A_{N-1}\}$  が与えられます。

$A$  に含まれる最大の数を求めなさい。

・  $1 \leq N \leq 10^5$

・  $-10^6 \leq A_i \leq 10^6$

入力

$N$

$A_0 A_1 \dots A_{N-1}$

---

①分けるにより、今回の問題は以下のような部分問題にすることが出来ました。

- ・  $A_0$  と  $A_1$  のうち大きいものを求める
- ・  $A_1$  までの最大値と  $A_2$  のうち大きいものを求める
- ・  $A_2$  までの最大値と  $A_3$  のうち大きいものを求める
- ・
- ・  $A_{N-2}$  までの最大値と  $A_{N-1}$  のうち大きいものを求める

次のステップとして、②メモるについて考えていきます。

そもそも、ここで言うメモとは何か？ という話なのですが、これは「計算結果を記録しておく」ことを表しています。

普段の生活でするメモと同じように、とりあえず忘れないように残しとこ～って感じの使い方ですね。

じゃあ DP では何をメモするのか、というと「部分問題を解いた時に出てきた解」をメモしておきます。

今回の場合は  $A_1$  までの最大値、 $A_2$  までの最大値、…がメモに残る感じですね。

部分問題を解きながらメモをしていくと、大体の場合は元の問題の答えが勝手に求められます。

今回の場合は  $A_{N-1}$  までの最大値がそのまま答えになっています。

そのため、③求めるパートでは答えを出力するだけになることが多いです。

次に、メモの方法について説明します。

といっても慣例的なものなのでこれじゃなきゃダメ！ といったものではないのですが、色々なコード等を読む上で使ってる人が多い方法なので今回はこの方法を使ってみてください。

メモを残すときには基本的には配列を利用します。

配列名は慣例的に dp、もしくは DP が利用されることが多いです。(memo や MEMO などを使う人もいます)

```
1 /* どちらでもいい */
2 int dp[100];
3 int DP[100];
```

配列の要素数は人により考え方が異なりますが

```
1 /* 要素数Nの上限が決まっている場合 */
2 int n_max = 100;
3
4 /* ① 必要数ピッタリで作る */
5 int dp[100];
6
7 /* ② 必要数より少し多めに作る */
8 int dp[110];
```

といった手法を取ることが多いです。

それぞれのメリットとしては

- ①…メモリ使用量が減る
- ②…配列参照のエラーが起こりづらくなる

といった点があります。

どちらを使っても問題ないのですが、C 言語の場合①のメリットが非常に薄いので②を使うのがいいかな、と思います。

他言語を使う場合は①にも大きなメリットがある場合があるので好みによって使い分けてください。

以上の内容を元に、次の節では具体例を見ていきます。

### A.2.7 問題を解く

---

#### 例題 1

長さ  $N$  の整数列  $A = \{A_0, A_1, \dots, A_{N-1}\}$  が与えられます。

$A$  に含まれる最大の数を求めなさい。

•  $1 \leq N \leq 10^5$

•  $-10^6 \leq A_i \leq 10^6$

入力

$N$

$A_0 A_1 \dots A_{N-1}$

---

①分けるにより、今回の問題は以下のような部分問題にすることが出来ました。

- $A_0$  と  $A_1$  のうち大きいものを求める
- $A_1$  までの最大値と  $A_2$  のうち大きいものを求める
- $A_2$  までの最大値と  $A_3$  のうち大きいものを求める
- $\vdots$
- $A_{N-2}$  までの最大値と  $A_{N-1}$  のうち大きいものを求める

また、dp 配列を用いたメモを使うといい、という話がありました。

ここからは具体的な例を元に解き方を考えていきます。

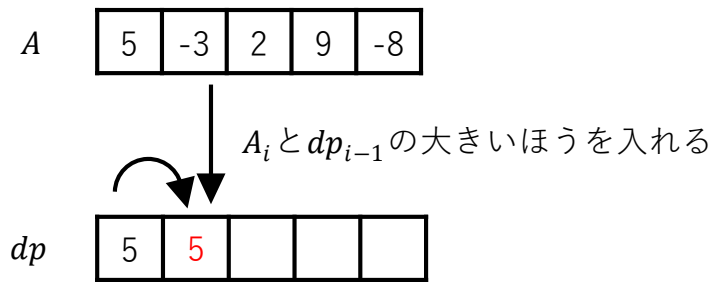
例として、 $N = 5$ ,  $A = \{5, -3, 2, 9, -8\}$  というサンプルについて考えてみます。

これに対して dp 配列を用意するのですが、ちょっと工夫してこんな形にしてみます。

- $dp[i] (0 \leq i < N)$  は  $A[i]$  までの最大値

つまり、①分けるで考えたパターンに加えて、 $dp[0]$  は  $A[0]$  までの最大値、というイメージを追加している感じです。

このようにすると、以下の図のように配列 dp と配列 A の添字を揃えて考えることができるようになるので管理がしやすくなります。



実は dp なんていないのは最初に説明しちゃったんですが、折角なのでこれらの内容を利用して、例題を「dp 配列を利用して」解いてみてください。

### A.2.8 数列内の最大の数の出力

## 問題文

---

長さ  $N$  の整数列  $A = \{A_0, A_1, \dots, A_{N-1}\}$  が与えられます。  
 $A$  に含まれる最大の数を求めなさい。

## 制約

---

- $2 \leq N \leq 10^5$
- $-10^6 \leq A_i \leq 10^6$

## 入力

---

入力は以下の形式で標準入力から与えられる。

$N$

$A_1 A_2 \dots A_N$

## 出力

---

1 行で、 $A$  に含まれる最大の数を出力せよ。

### A.2.9 問題の解説

基本的には途中計算をメモしておくための配列 (今回は  $dp$ ) を用意したうえで、

- $dp[i](0 \leq i < N)$  は  $A[i]$  までの最大値

という式をそのまま実装する形になります。

今回  $i$  の値は  $0$  から  $N - 1$  まで増加していくため、`for` 文を利用してループしてやれば  $dp[0]$  から順に計算していくことができます。

注意点として、 $dp[-1]$  は配列の添字として利用できないため、 $i = 0$  の時のみ処理を分けるか  $dp[0] = A[0]$  という形にして  $i = 1$  からループを開始するとよいです。

以下に解答例を示します。

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int n, a[100000], dp[100000];
6
7     /* 入力 */
8     scanf("%d", &n);
9     for (int i = 0; i < n; i++) {
10         scanf("%d", &a[i]);
11     }
12
13     dp[0] = a[0];
14
15     for (int i = 1; i < n; i++) {
16         if (a[i] > dp[i - 1]) {
17             dp[i] = a[i];
18         } else {
19             dp[i] = dp[i - 1];
20         }
21     }
22
23     printf("%d", dp[n - 1]);
24
25     return 0;
26 }
```



また、大きい数と同様に `max()` 関数などの大きいほうの数を返す関数を実装すると分かりやすくなります。

以下に解答例を示します。

```
1 #include <stdio.h>
2
3 int max(int x, int y)
4 {
5     return x > y ? x : y;
6 }
7
8 int main()
9 {
10     /* 入力 */
11     int n, a[100000], dp[100000];
12     scanf(" %d", &n);
13
14     for (int i = 0; i < n; i++) {
15         scanf(" %d", &a[i]);
16     }
17
18     dp[0] = a[0];
19
20     for (int i = 1; i < n; i++) {
21         dp[i] = max(dp[i - 1], a[i]);
22     }
23
24     printf(" %d", dp[n - 1]);
25
26     return 0;
27 }
```

Java/Python/C++ の解答例は以下を確認してください。

解答例 (Gist)[[リンク](#)]

### A.2.10 Gist の解答例

基本的には C 言語と同様の実装が可能です。

なお、C++ の `max_element` 関数、Python の `max()` 関数などを使うと配列内の最大値を求めることもできます。今回は一応 DP を使おう！ という問題なので解答例は DP を利用した解答を示します。

#### C++ による解答例

```
1 #include <bits/stdc++.h>
2 // #include <iostream>
3
4 using namespace std;
5
6 int main(void) {
7     int n, a[100000], dp[100000];
8
9     cin >> n;
10    for (int i = 0; i < n; i++){
11        cin >> a[i];
12    }
13
14    dp[0] = a[0];
15
16    for (int i = 1; i <= n; i++) {
17        dp[i] = max(dp[i - 1], a[i]);
18    }
19
20    cout << dp[n - 1] << endl;
21
22    return 0;
23 }
```

#### Java による解答例

```
1 import java.util.*;
2
3 public class Main{
4     public static void main(String[] args) {
5         int n;
6         Scanner sc = new Scanner(System.in);
7         n = Integer.parseInt(sc.next());
8
9         int a[] = new int[100000];
10        for (int i = 0; i < n; i++) {
11            a[i] = Integer.parseInt(sc.next());
12        }
13
14        int dp[] = new int[100000];
15        dp[0] = a[0];
16
17        for(int i = 1; i < n; i++){
18            dp[i] = Math.max(dp[i - 1], a[i]);
19        }
20
21        System.out.println(dp[n - 1]);
22    }
23 }
```

## Python3 による解答例

```
1 n = int(input())
2 a = list(map(int, input().split()))
3
4 dp = [0] * n
5
6 dp[0] = a[0]
7
8 for i in range(1, n):
9     dp[i] = max(dp[i - 1], a[i])
10
11 print(dp[-1])
```

### A.2.11 計算結果をメモする意味

注) この節は DP の理解に繋がるといいなのコーナーとなっているため読み飛ばしても大丈夫です。

たとえば、算数でこんな問題が出たとします。

- ①  $1 + 2 = ?$
- ②  $1 + 2 + 3 = ?$
- ③  $1 + 2 + 3 + 4 = ?$
- ④  $1 + 2 + 3 + 4 + 5 = ?$
- ⑤  $1 + 2 + 3 + 4 + 5 + 6 = ?$
- ⑥  $1 + 2 + 3 + 4 + 5 + 6 + 7 = ?$
- ⑦  $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = ?$
- ⑧  $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = ?$
- ⑨  $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = ?$

実際にこんな問題が出るかは知りませんが。

多分、多くの人は

- ①  $1 + 2 = 3$
- ②  $3 + 3 = 6$
- ③  $6 + 4 = 10$
- ④  $10 + 5 = 15$

みたいな感じで計算すると思います。一つ上の答えがそのまま使えますしね。

これが、DP におけるメモをすることの意味です。

かっこよく言うなら計算の再利用が可能になります。

DP とは、計算の再利用を上手く利用することにより、実は同じ計算をしていた無駄な部分を削って計算量を減らすアルゴリズムとなっています。

ホントに減ってるの？ と気になったら上の問題を実際に何回足し算してるか数えてみてください。

今回は 9 までしかやっていないので差が小さいですが、これが 100,000 や 1,000,000 といった数字になっていくと明確な差が表れます。

興味があればプログラムを書いて数えてみてもいいかもしれません。

(パッと見て気づく人は多いかもしれませんが最後に足す数を  $N$  と置いた場合、計算回数はそれぞれ  $(\sum_{k=1}^{n-1} k)$  と  $(N - 1)$  になっています。手計算でも求められますね。)

## A.3 数式をプログラミングする

### A.3.1 導入 例題 (フィボナッチ数列) の紹介

DP の考え方が何となく見えてきたところで、今度は数式を計算していきます。

DP は前の状況をメモしておく、という仕組み上漸化式をプログラミングする場合に非常に相性がいいです。

今回は例として、DP の説明で必ずと言っていいほど用いられるフィボナッチ数列の計算量効率化について考えていきます。

## フィボナッチ数列とは？

フィボナッチ数列は、0, 1, 1, 2, 3, 5, 8, 13, ... のようにどの数字も前の 2 つを足した数になっている数列のことです。

漸化式で表すと以下ようになります。

- $F_0 = 0$
- $F_1 = 1$
- $F_{n+2} = F_n + F_{n+1} (n \geq 0)$

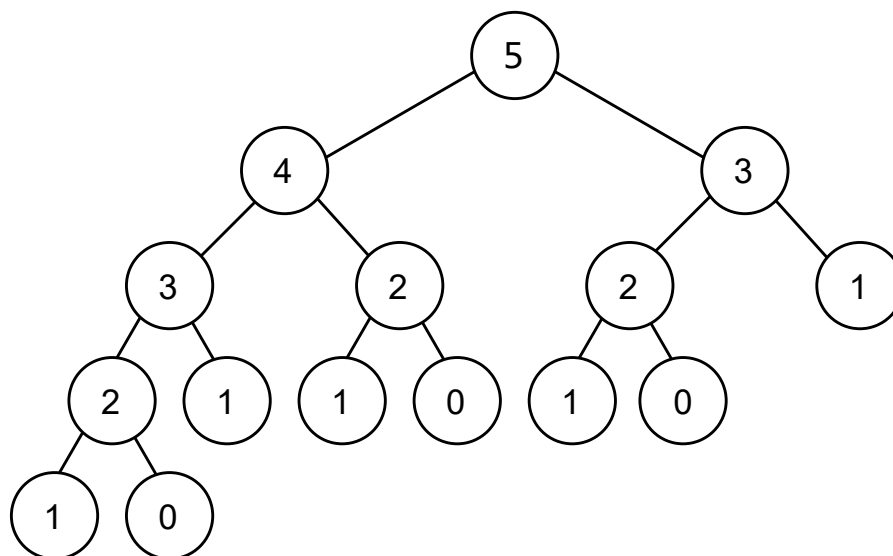
フィボナッチ数列をプログラムで計算する簡単な実装として、再帰関数による実装があります。以下にコードを示します。

```
1 #include <stdio.h>
2
3 int fib(int n)
4 {
5     /* F(0) = 0 */
6     if(n == 0) return 0;
7     /* F(1) = 1 */
8     if(n == 1) return 1;
9     /* n ≥ 2 の時は再帰する */
10    return fib(n - 2) + fib(n - 1);
11 }
12
13 int main(void)
14 {
15     /* 入力 */
16     int n;
17     scanf("%d", &n);
18     /* 入力ここまで */
19
20     /* 関数を呼び出して戻り値を出力する */
21     printf("%d", fib(n));
22
23     return 0;
24 }
```

この方法だと実装は簡単ですが、 $N$  の値が大きくなると計算量が膨大になる、という問題点があります。

### A.3.2 計算量を考える

再帰関数による計算では以下のような計算を行っています。



図の棒で繋がれている部分は計算に利用することを表しています。

図は  $F_5$  の例ですが、 $F_5$  を求めるために  $F_4$  と  $F_3$  を求める、

$F_4$ を求めるために  $F_3$  と  $F_2$  を…という形になっています。

そのため、 $F_5$ を求めるためには画像の線の部分 (関数呼び出し) と  $F_5$  を求めるための呼び出しを合わせて 15 回も計算を行う必要があります。

このような再帰関数による実装では、数字が 1 上がるだけで大体 2 倍の計算量が必要になるため  $N=40$  でも 3 億回以上の関数呼び出しが発生してしまいます。

最近のパソコンで 1 秒間に処理できる計算量は  $10^9$  回程度とされているので、 $N=43$  程度で 1 秒を超えてしまうことになります。

そこで注目したいのが図の中の同じ数字の部分で、再帰関数では毎回同じ計算を繰り返していることがわかります。

…ところで、同じ計算を効率化するアルゴリズム、ありましたよね。

DP の考え方②メモるを使うと、このような場合に大幅な効率化ができるようになります。

メモを利用したコードを以下に示します。

```
1 #include <stdio.h>
2
3 /* メモ用配列の宣言 */
4 int dp[100];
5
6 int fib(int n)
7 {
8     /* メモがある場合はメモの値を返す */
9     if (dp[n] != -1) return dp[n];
10
11     /* メモがない時は計算結果をメモしながら再帰する */
12     return dp[n] = fib(n - 2) + fib(n - 1);
13 }
14
15 int main(void)
16 {
17     int n;
18
19     /* 入力 */
20     scanf("%d", &n);
21     /* 入力ここまで */
22
23     /* dp配列を-1で初期化する */
24     for (int i = 0; i < 100; i++) dp[i] = -1;
25
26     /* fib(0) = 0, fib(1) = 1 */
27     dp[0] = 0;
28     dp[1] = 1;
29
30     /* 関数を呼び出して戻り値を出力する */
31     printf("%d", fib(n));
32
33     return 0;
34 }
```

一般にこのような方法をメモ化再帰と呼びます。

このメモ化再帰、もちろん DP の一種ではありますが前の章で使ったような前から順番に計算していく DP と比較するとメリットとデメリットがあります。

まずメリットとして、

- 漸化式をそのまま実装できる

という点があります。これはかなり強力なメリットで、参照の順番等を考えなくても必要な順に勝手に再帰してくれるため雑な実装をする際にはかなり重宝します。

デメリットとしては、

- 再帰関数を見慣れていないと挙動がわかりづらい
- メモリ使用量が多い
- 言語によっては再帰上限がある場合がある

などの点があります。



このうち 3 つ目の再帰上限がかなり厄介で、C 言語でもスタックに利用されるメモリ領域を使い切るとスタックオーバーフローが発生してしまいます。

そこで、今回はメモ化再帰ではなく、応用の効きやすい前から順に計算していくような DP について考えていきます。

### A.3.3 漸化式をプログラミングする

ここで、フィボナッチ数列の漸化式を思い出してほしいんですが、

- $F_0 = 0$
- $F_1 = 1$
- $F_{n+2} = F_n + F_{n+1} (n \geq 0)$

こんな形になっていました。

この3つ目の式を少し変形すると、

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-2} + F_{n-1} (n \geq 2)$

このような形に直すことができます。

この形の式であれば、 $F_0$  と  $F_1$  があれば  $F_2$  が求められる、 $F_1$  と  $F_2$  があれば  $F_3$  が求められる、…といったように前から順に求めていくことができそうです。

ということで、以上の内容や1章の内容を利用して次の問題を解いてみてください。

一応前から計算する DP で解くことを想定していますが、メモ化再帰を利用しても問題ありません。

### A.3.4 フィボナッチ数列

## 問題文

---

整数  $N$  が与えられるので、フィボナッチ数列の  $N$  番目の数字を出力してください。  
フィボナッチ数列は以下の式で表される数列です。

- $F_0 = 0$
- $F_1 = 1$
- $F_{n+2} = F_n + F_{n+1} (n \geq 0)$

## 制約

---

- $2 \leq N \leq 45$

## 入力

---

入力は以下の形式で標準入力から与えられる。  
 $N$

## 出力

---

1 行で、フィボナッチ数列の第  $N$  項を出力せよ。

### A.3.5 問題の解説

制約の  $N \leq 45$  より、単純な再帰関数では時間内に計算できないため DP を利用して解く必要があります。

DP による実装の場合

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-2} + F_{n-1} (n \geq 2)$

という形式で実装が行えるためこの通りにプログラムを書けばいいです。

具体的には、途中計算結果を保存する配列を用意して  $F_2$  から  $F_N$  まで順に計算していくことにより求めることができます。

以下に解答例を示します。

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     /* 入力 */
6     int n, dp[50];
7
8     scanf( "%d ", &n);
9
10    dp[0] = 0;
11    dp[1] = 1;
12
13    for (int i = 2; i <= n; i++) {
14        dp[i] = dp[i - 2] + dp[i - 1];
15    }
16
17    printf( "%d ", dp[n]);
18
19    return 0;
20 }
```

また、今回は推奨していませんがメモ化再帰による実装も可能です。  
一応、以下に解答例を示します。

```
1 #include <stdio.h>
2
3 int dp[50];
4
5 int fib(int num)
6 {
7     if (dp[num] >= 0) return dp[num];
8
9     return dp[num] = fib(num - 2) + fib(num - 1);
10 }
11
12 int main(void)
13 {
14     /* 入力 */
15     int n;
16     scanf(" %d", &n);
17
18     /* dp配列の初期化 */
19     for (int i = 0; i < 50; i++){
20         dp[i] = -1;
21     }
22     dp[0] = 0;
23     dp[1] = 1;
24
25     printf(" %d", fib(n));
26
27     return 0;
28 }
```

Java/Python/C++ の解答例は以下を確認してください。

解答例 (Gist)[[リンク](#)]

### A.3.6 Gist の解答例

基本的には C 言語と同じように

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-2} + F_{n-1} (n \geq 2)$

を実装すればいいです。

#### C++ による解答例

```
1 // C++ の場合
2
3 #include <bits/stdc++.h>
4 // #include <iostream>
5
6 using namespace std;
7
8 int main(void) {
9     int n, dp[50];
10    cin >> n;
11
12    dp[0] = 0;
13    dp[1] = 1;
14
15    for (int i = 2; i <= n; i++){
16        dp[i] = dp[i - 1] + dp[i - 2];
17    }
18
19    cout << dp[n] << endl;
20
21    return 0;
22 }
```

#### Java による解答例

```
1 // Java の場合
2
3 import java.util.*;
4
5 public class Main {
6     public static void main(String[] args) {
7         int n;
8         Scanner sc = new Scanner(System.in);
9         n = Integer.parseInt(sc.next());
10
11         int dp[] = new int[50];
12         dp[1] = 1;
13
14         for (int i = 2; i <= n; i++) {
15             dp[i] = dp[i - 1] + dp[i - 2];
16         }
17
18         System.out.println(dp[n]);
19     }
20 }
```

## Python3 による解答例

```
1 # Python の場合
2
3 n = int(input())
4
5 dp = [0] * 50
6
7 dp[1] = 1
8
9 for i in range(2, n + 1):
10     dp[i] = dp[i - 1] + dp[i - 2]
11
12 print(dp[n])
```

### A.3.7 トリボナッチ数列

## 問題文

---

整数  $N$  が与えられるので、トリボナッチ数列の  $N$  番目の数字を出力してください。  
トリボナッチ数列は以下の式で表される数列です。

- $F_0 = 0$
- $F_1 = 0$
- $F_2 = 1$
- $F_{n+3} = F_n + F_{n+1} + F_{n+2} (n \geq 0)$

## 制約

---

- $2 \leq N \leq 38$

## 入力

---

入力は以下の形式で標準入力から与えられる。

$N$

## 出力

---

1 行で、トリボナッチ数列の第  $N$  項を出力せよ。



### A.3.8 問題の解説

フィボナッチ数列と同様に DP を利用して実装することができます。

トリボナッチ数列を表す漸化式は

- $F_0 = 0$
- $F_1 = 0$
- $F_2 = 1$
- $F_{n+3} = F_n + F_{n+1} + F_{n+2} (n \geq 0)$

であることから、フィボナッチ数列と同様に変形を行い

- $F_0 = 0$
- $F_1 = 0$
- $F_2 = 1$
- $F_n = F_{n-3} + F_{n-2} + F_{n-1} (n \geq 3)$

という形式で実装が行えるためこの通りにプログラムを書けばいいです。

以下に解答例を示します。

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     /* 入力 */
6     int n, dp[50];
7
8     scanf( "%d ", &n);
9
10    dp[0] = 0;
11    dp[1] = 0;
12    dp[2] = 0;
13
14    for (int i = 3; i <= n; i++) {
15        dp[i] = dp[i - 3] + dp[i - 2] + dp[i - 1];
16    }
17
18    printf( "%d ", dp[n]);
19
20    return 0;
21 }
```

Java/Python/C++ の解答例は以下を確認してください。

解答例 (Gist)[[リンク](#)]

### A.3.9 Gist の解答例

C 言語と同様に実装が可能です。

#### C++ による解答例

```
1 #include <bits/stdc++.h>
2 // #include <iostream>
3
4 using namespace std;
5
6 int main(void) {
7     int n, dp[50];
8     cin >> n;
9
10    dp[0] = 0;
11    dp[1] = 0;
12    dp[2] = 1;
13
14    for (int i = 3; i <= n; i++){
15        dp[i] = dp[i - 1] + dp[i - 2] + dp[i - 3];
16    }
17
18    cout << dp[n] << endl;
19
20    return 0;
21 }
```

#### Java による解答例

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         int n;
6         Scanner sc = new Scanner(System.in);
7         n = Integer.parseInt(sc.next());
8
9         int dp[] = new int[50];
10        dp[2] = 1;
11
12        for (int i = 3; i <= n; i++) {
13            dp[i] = dp[i - 1] + dp[i - 2] + dp[i - 3];
14        }
15
16        System.out.println(dp[n]);
17    }
18 }
```

#### Python3 による解答例

```
1 n = int(input())
2
3 dp = [0] * 50
4
5 dp[2] = 1
6
7 for i in range(3, n + 1):
8     dp[i] = dp[i - 1] + dp[i - 2] + dp[i - 3]
9
10 print(dp[n])
```

## 付録 B C 言語版セッションで追加した項目

### B.1 はじめに

#### 言語について

本教材では、基本的に C 言語を利用しています。

他言語の利用については以下のようになっています。

利用できる言語一覧についてはこちらから確認してください。

言語	教材中のコード例	解答への利用	解答例
C 言語	○	○	○
Java Python3 C++	×	○ (*1)	○
その他言語	×	○ (*1)	×

\*1:C 言語以外を利用するには他言語版セッションに参加する必要があります。

#### 教材中の表現について

##### 計算量

教材中では「計算量」という表現が多く出てきます。

これは、厳密には時間計算量と呼ばれるもので答えを出すまでにかかる計算回数を表しています。

本教材中での認識としては計算量を減らすとかかる時間が短くなる、程度の認識で問題ありません。

より深く知りたい場合は以下のようなキーワードを調べてみるといいかもしれません。

- オーダ記法 (オーダー記法、O 記法)
- ランダウの記号
- 時間計算量

## 問題について

### 問題の形式

本教材で利用する問題は、複数パターンの入力例に対してそれぞれ答えを出力する形式となっています。

よって、解答となるプログラムは以下のような流れになっている必要があります。

1. 入力を受け取る
2. 何らかの処理をする (問題により変化する)
3. 出力を行う

サンプルのみを確認して入力値を定数としないように注意してください。

## 入出力

C 言語での入出力一覧は次の教材に纏めてあります。

必要に応じて参照してください。

## 問題文について

問題文は以下の形式になっています。

## 問題文

---

問題文の本文です。

ここにある問題をプログラムを用いて解きます。

## 制約

---

入力値の範囲です。

$0 \leq N \leq 100$  であれば  $N$  が 0 から 100 の範囲のいずれかを取ることを示しています。

## 入力

---

入力の形式です。

ここに示される形式で入力値が与えられます。

## 出力

---

出力の形式です。

ここに示される形式で出力を行う必要があります。

## サンプルケース

---

入力のサンプルケースです。

実際の採点ではここにあるケースに加え複数のケースで判定を行います。

## B.2 C 言語入出力チートシート

ここでは、C 言語における基本的な入出力の方法を確認できます。

なお、C 言語の標準入力では空白区切り、行区切り共に同じように取得することが可能です。

### 入力編

---

#### 1つの整数の入力

入力

$N$

$N$  は整数

```
1 int n;  
2 scanf( "%d ", &n);
```

#### 2つの整数の入力

入力

$N M$

$N, M$  は整数

```
1 int n, m;  
2 scanf( "%d ", &n);  
3 scanf( "%d ", &m);
```

#### 整数列の入力

入力

$N$

$A_1, A_2, \dots, A_N$

$N, A_i$  は整数

```
1 int n, a[101];  
2 scanf( "%d ", &n);  
3 for(int i = 0; i < n; i++){  
4     scanf( "%d ", &a[i]);  
5 }
```

## 1 行 1 文字の入力

入力

$C$

$C$  は半角文字

```
1 char c;  
2 scanf( "%c ", &c );
```

## 1 行の文字列の入力

入力

$S$

$S$  は文字列

```
1 char s[101];  
2 scanf( "%s ", s );
```

## 出力編

---

### 1 つの整数の出力

出力

$N$

$N$  は整数

```
1 int n;  
2 printf( "%d \n ", n );
```

### 整数列の出力 (行区切り)

出力

$A_1$

$A_2$

:

$A_N$

$N, A_i$  は整数

```
1 int n, a[101];  
2 for( int i = 0; i < n; i++){  
3     printf( "%d \n ", a[i]);  
4 }
```

## 整数列の出力 (空白区切り)

出力

$A_1 A_2 \dots A_N$

$N, A_i$  は整数

```
1 int n, a[101];
2 for(int i = 0; i < n; i++){
3     if(i != 0) putchar( ' ');
4     printf( "%d ", a[i]);
5 }
6 printf( " \n ");
```

## 1 行の文字列の出力

入力

$S$

$S$  は文字列

```
1 char s[101];
2 printf( "%s \n ", s);
```

## オマケ

---

### 十分に大きい数 / 小さい数

C 言語で大きい数や小さい数を使う場合、シフト演算子を利用して数値を代入することが多いで  
す。

シフト演算子は bit 表記 (2 進数) の状態で桁をシフトする演算で、big の場合は  $10(2)$  を左に 29  
ビット算術シフトした値=1073741824 となります。

これで足りない場合には limit.h を include すれば INT\_MAX により 2147483647 が取得できま  
すが…そこまで必要になることはあまりないと思います。

```
1 int big = 2 << 29;
2 int small = -2 << 30;
```



### B.3 大きい数 初期プログラム

```
1  /* CやC++などシェルに実行結果コード返却を明示する言語を利用する場合 基本的に0
   2  を返却してください。 */
3
4  #include <stdio.h>
5
6  int main(void)
7  {
8      /* 入力 */
9      int x, y;
10     scanf( "%d %d ", &x, &y);
11
12     return 0;
```

### B.4 最大の数 初期プログラム

```
1  /* CやC++などシェルに実行結果コード返却を明示する言語を利用する場合 基本的に0
   2  を返却してください。 */
3
4  #include <stdio.h>
5
6  int main()
7  {
8      /* 入力 */
9      int n, a[100000];
10     scanf( "%d ", &n);
11
12     for(int i = 0; i < n; i++){
13         scanf( "%d ", &a[i]);
14     }
15
16     return 0;
```

## B.5 フィボナッチ数列 初期プログラム

```
1  /* CやC++などシェルに実行結果コード返却を明示する言語を利用する場合 基本的に0
   2  を返却してください。 */
3
4  #include <stdio.h>
5
6  int main(void)
7  {
8      /* 入力 */
9      int n;
10     scanf( "%d ", &n);
11
12     return 0;
```

## B.6 トリボナッチ数列 初期プログラム

```
1  /* CやC++などシェルに実行結果コード返却を明示する言語を利用する場合 基本的に0
   2  を返却してください。 */
3
4  #include <stdio.h>
5
6  int main(void)
7  {
8      /* 入力 */
9      int n;
10     scanf( "%d ", &n);
11
12     return 0;
```

## B.7 さいごに

DP がどんな時に使えるのか、何となく理解できたでしょうか。

今回紹介した例は少ないですが、DP が利用できる計算にはこれ以外にも様々なパターンがあります。

有名なものとナップザック問題と呼ばれるものや巡回セールスマン問題 (厳密解を求めるもので、要素数が多いと計算できません) など DP を利用して解くことが可能です。

また、BFS(幅優先探索) や DFS(深さ優先探索)、ダイクストラ法 (グラフ上の最短経路を求めるアルゴリズム) など DP を利用した手法だったりします。

興味があれば是非色々と調べてみてください。

オマケとして、DP に限らず様々なアルゴリズムを学習できるコンテンツを紹介したいと思います。

### 超初心者向け

AtCoder Programming Guide for beginners (APG4b)

競技プログラミングサイト AtCoder の C++ 初心者向け講座です。

C 言語とかめんどくさくて使ってもらえないけど他の言語の文法とかわからん！ という人や C++ に興味があるけどなにで勉強すればいいのかわからない、という人におすすめです。

### 初心者向け

アルゴ式

様々なアルゴリズムが基礎から学べるサイトです。

レベル的にはプログラミングの文法はわかるけどアルゴリズムはわからない人向けです。

### 中級者向け

Educational DP Contest

名前の通り DP を使う問題が沢山載っています。

難易度的には比較的簡単に解ける問題 (A 問題など) から全くもって初心者向けではない問題 (J 問題以降) など様々な難易度・ジャンルの DP 問題があります。

Project Euler

ガチ勢向け数学コンテストです。

英語のみのサイトのため参加のハードルは高めですが英語に慣れていれば良質な問題が多いためオススメです。

## AtCoder

いわゆる競技プログラミングと呼ばれるもののコンテストサイトです。

ほぼ毎週コンテストが行われており、DP を利用した問題もたくさん出ています。

興味があれば過去問を解いてみるといいかもしれません。

## TechFUL

一応ね、一応同じプラットフォームなので紹介だけ。色々な問題を解くことができますが採点システムや諸々の導線に若干の難があるので個人的な評価は低め。問題文がわかりづらい。

一応コードを書いていると就活に使えたりします。

## paiza スキルチェック

こちらも同様に色々な問題がありますがどんなアルゴリズムを使うかや解答例は非公開なので難易度は高めコードを書いていると就活に使えたりします。ランクが高いと面接の交通費支給があったりと便利です。

その他にも、興味があれば競技プログラミングなどを調べてみると面白いかもしれません。

AtCoder などでは同じ大学の人と競い合ったりもできます。興味があれば、是非。

また、オススメの本は以下の通りです。

どちらも大学図書館の書架にあるため気になったら是非借りてみてください。

- 問題解決力を鍛える!アルゴリズムとデータ構造 (大槻兼資)
- プログラミングコンテストチャレンジブック (秋葉拓哉)

## 付録 C TechFUL で教材を表示した場合の例

### C.1 セクションを利用した際のイメージ

The screenshot shows the TechFUL website interface. The header includes the TechFUL logo and navigation links: プログラミング問題, イベント, 求人, ポートフォリオ, 選考管理, チャット, お知らせ, and a language selector set to 日本語. The main banner for the 'C言語版' event displays the title 'イベントタイトル: アルゴリズム学習教材', the theme 'イベントテーマ: 動的計画法', the dates '2021-12-01(水) 11:10 ~ 2022-01-30(日) 00:00', the number of questions '問題数: 20問', and the venue '開催場所: everywhere' with an 'オンライン' button. Below the banner are tabs for 'イベント概要', 'セッション', 'スレッド', '解答状況', and 'コード提出履歴'. A '問題FAQ' link is also present. The 'セッション' tab is active, showing a list of sections with their topics, difficulty levels, and associated text materials.

セッション	テキスト教材
はじめに アルゴリズム 初級 難易度 1	テキスト教材
C言語入出力チートシート プログラミング基礎 初級 難易度 1	テキスト教材
導入 アルゴリズム 初級 難易度 1	テキスト教材
1.動的計画法の考え方 1 アルゴリズム 初級 難易度 1	テキスト教材
	テキスト教材

## C.2 教材と問題のイメージ

<<- 戻る

≡ 1.動的計画法の考え方 1

👤 作成者: 松丸颯汰 作成日: 2021年12月08日

アルゴリズム/ レベル: 初級

# 1.動的計画法の考え方 1

## 1.1 導入

まずは動的計画法の3ステップ

- ①分ける
- ②メモる
- ③求める

について見ていきます。

この章では次の問題について考えます。

### 例題1

長さ  $N$  の整数列  $A=[A_0, A_1, \dots, A_{N-1}]$  が与えられます。

$A$  に含まれる最大の数を求めなさい。

- ・  $1 \leq N \leq 10^5$
- ・  $-10^6 \leq A_i \leq 10^6$

入力

$N$

$A_0 A_1 \dots A_{N-1}$

いわゆる線形探索法と呼ばれる問題です。

## 「最大の数」 / 難易度: 1

問題タイプ: 穴埋め問題    目標タイム: 30分    アルゴリズム / 実装問題

### 問題文

長さ  $N$  の整数列  $A = [A_1, A_2, \dots, A_N]$  が与えられます。  
 $A$  に含まれる最大の数を求めなさい。

### 制約

- $2 \leq N \leq 10^5$
- $-10^6 \leq A_i \leq 10^6$

### 入力

入力は以下の形式で標準入力から与えられる。

$N$

$A_1 \ A_2 \ \dots \ A_N$

### 出力

1行で、 $A$  に含まれる最大の数を出力せよ。

#### サンプルケース1

入力値 行数: 2

```
5
5 -3 2 9 8
```

出力値 行数: 1

```
9
```

#### サンプルケース2

入力値 行数: 2

```
5
72 72 72 72 72
```

出力値 行数: 1

```
72
```

テストする

## C.3 コードテストを行った時のイメージ

[< 戻る](#)[> サンプルコード](#)[> 問題FAQ](#)

### 「最大の数」 / 難易度 : 1

問題タイプ:穴埋め問題 目標タイム:30分 アルゴリズム/ 実装問題

#### 問題文

長さ  $N$  の整数列  $A = [A_1, A_2, \dots, A_N]$  が与えられます。  
 $A$  に含まれる最大の数求めなさい。

#### 制約

- $2 \leq N \leq 10^5$
- $-10^6 \leq A_i \leq 10^6$

#### 入力

入力は以下の形式で標準入力から与えられる。

$N$   
 $A_1 \ A_2 \ \dots \ A_N$

#### 出力

1行で、 $A$  に含まれる最大の数を出せよ。

#### サンプルケース1

入力値 行数:

```
5
5 -3 2 9 8
```

出力値 行数:

```
9
```

テスト結果 × **COMPILE ERROR**

```
main.c: In function 'max':
main.c:5:23: error: expected
5 |         if(x > y) return
```

#### サンプルケース2

入力値 行数:

```
5
72 72 72 72 72
```

出力値 行数:

```
72
```



## 「最大の数」 / 難易度 : 1

問題タイプ:穴埋め問題 目標タイム:30分 アルゴリズム/ 実装問題

### 問題文

長さ  $N$  の整数列  $A=[A_1, A_2, \dots, A_N]$  が与えられます。  
 $A$  に含まれる最大の数を求めなさい。

### 制約

- $2 \leq N \leq 10^5$
- $-10^6 \leq A_i \leq 10^6$

### 入力

入力は以下の形式で標準入力から与えられる。

 $N$  $A_1 A_2 \dots A_N$ 

### 出力

1行で、 $A$  に含まれる最大の数を出力せよ。

#### サンプルケース1

入力値 行数:

```
5
5 -3 2 9 8
```

出力値 行数:

```
9
```

テスト結果 × **WRONG ANSWER**

```
10
```

#### サンプルケース2

入力値 行数:

```
5
72 72 72 72 72
```

出力値 行数:

```
72
```

テスト結果 × **WRONG ANSWER**

## 「最大の数」 / 難易度 : 1

問題タイプ: 穴埋め問題 目標タイム: 30分 アルゴリズム / 実装問題

### 問題文

長さ  $N$  の整数列  $A = [A_1, A_2, \dots, A_N]$  が与えられます。  
 $A$  に含まれる最大の数を求めなさい。

### 制約

- $2 \leq N \leq 10^5$
- $-10^6 \leq A_i \leq 10^6$

### 入力

入力は以下の形式で標準入力から与えられる。

 $N$  $A_1 \ A_2 \ \dots \ A_N$ 

### 出力

1行で、 $A$  に含まれる最大の数を出力せよ。

#### サンプルケース1

入力値 行数:

```
5
5 -3 2 9 8
```

出力値 行数:

```
9
```

テスト結果 ✓ PASSED

```
9
```

#### サンプルケース2

入力値 行数:

```
5
72 72 72 72 72
```

出力値 行数:

```
72
```

テスト結果 ✓ PASSED

## C.4 正誤判定を行った際のイメージ

0 問 / 7 問中 テストケース正解

× random\_1 WA

× random\_2 WA

× max\_1 WA

× max\_2 WA

× min\_1 WA

× min\_2 WA

× random\_maxlen\_1 WA

🏠 獲得合計ポイント  
(内訳)

テスト結果

**WRONG ANSWER**

実行速度

**0.003 s**

問題一覧へ戻る

ダッシュボードへ戻る

次の問題へ進む

## 7問 / 7問中 全テストケース正解

✓ random\_1 AC

✓ random\_2 AC

✓ max\_1 AC

✓ max\_2 AC

✓ min\_1 AC

✓ min\_2 AC

✓ random\_maxlen\_1 AC

🏠 獲得合計ポイント  
(内訳)

テスト結果

PASSED

実行速度

0.003 s

問題一覧へ戻る

ダッシュボードへ戻る

次の問題へ進む