Woodland Academy

3.1 データ構造とアルゴリズム

- 基本概念
- 代表的なデータ構造

Shape Your Future

目次

データ構造とは

- **データ構造**[Data Structure]とは、コンピュータ上のデータを保存、整理、管理するための形式です。

- 同じデータを異なる方法で保存できます。データ構造を選択する場合、以下の要素を考慮することが最も重要です：
  - データの**保存方法**は？例えば、データの順番は大事ですか？データの間に何らかの相関性はありますか？
  - どんな**操作**（機能）が必要ですか？例えば、あるデータが保存されていますか？どうかを確認することは可能ですか？データが削除できますか？データの並べ替えは可能ですか？
  - これらの操作の**効率**はどのぐらいですか？すなわち、時間と空間のリソースをそれぞれどの程度消費していますか？

次へ

Woodland Academy

前へ

料理に例えて考えると、、

データ構造は、簡単に言えばデータを整理して格納する方法です。
なので、これを料理に例えると、冷蔵庫やキッチンの棚のようなものです。
異なる食材を適切に整理し、取り出しやすくしておくことで、料理を効率的に進めることができます。

Woodland Academy

前へ

- 例えば、クラスの生徒のファイルを保存するために、次の二つの方法は考えられます：一つ目は、全てのファイルをランダムにテーブルに並べます。二つ目は、ファイルを学籍番号順にフォルダに入れます。

- それぞれの方法について、先程説明した 3 つの要素を考えましょう：
  - 保存方法：一つ目の方法は、個々のファイルの間に関係がありません。二つ目の方法は、ファイルを順番に保存します。
  - 操作：どっちの方法も、ファイルを預ける、ファイルを取り出す、ファイルを検索、などが可能です。
  - 効率：一つ目の方法では、ファイルの預け入れは速いですが、取り出しは遅いです。二つ目の方法では、ファイルの預け入れは遅いですが、取り出しは速いです。

Woodland Academy

抽象データ型

- **抽象データ型**[Abstract Datatype]は、データの格納方法と操作の種類のみを考慮します。

- 抽象型が決まったら、それを様々なデータ構造で実装できます。どのデータ構造を選択するかは、これらのデータ構造が実装した操作の効率によってで決めることができます。

- データ構造を決める一般的なプロセスは：
  1. 必要な機能に基づいて、**どの抽象データ型**を使用するかを決定。

  2. **どんなデータ構造**が、その抽象型を実装できるかを複数特定。

  3. 実際の操作効率の必要性に基づいて、最終的に**どのデータ構造**を使用するかを決定します。

次へ

Woodland Academy

前へ

料理に例えて考えると、、

抽象データ型は、データの操作や挙動を定義する抽象的な概念です。
なので、これを料理に例えると、調理器具や調味料のようなものです。
例えば、スプーンはある特定の機能（かき混ぜる、すくう）を提供しますが、その内部の仕組みや素材は気にせずに利用できます。

スプーンがどんな素材であるかは気にしない！
ご飯が食べられればそれでいいや！

Woodland Academy

- 例えば、ブログサイトにある、ユーザーの全てのブログを保存したい際に、どのデータ構造を使うかを決めましょう：
  1. 抽象データ型：ブログを時間順に保存する必要があります。最新のブログを預け、任意のブログを閲覧する、ブログを削除するなどの操作が必要です。そこで調べてわかったのは、「リスト」タイプではこれらの機能を実現できます。したがって、リストの使用に決定。

  2. データ構造の候補：リストを実装できるのは、「配列」と「連結リスト」の二つのデータ構造があると、調べてわかりました。

  3. データ構造の決定：ユーザーがブログを閲覧する行動をよく使っていることがわかりました。この操作は、連結リストよりも配列の方が効率的なので、最終的に配列を使います。

アルゴリズム

- **アルゴリズム**[Algorithm]は、問題を解決するために使用される一連の操作手順を記述したものです。

- 広義には、レシピや家電の設置説明書など、問題を解決するための**具体的な手順**を記述したものをアルゴリズムと呼ぶことができます。私たちを学ぶアルゴリズムとは、問題を解決するためのコンピュータ・プログラムの手順を意味します。

- アルゴリズムとデータ構造には密接な関係があります：
  - 正しいデータ構造を選択することで、アルゴリズムの効率を向上させることができます。
  - 一方、データ構造自体の運用には、それを実現するためのアルゴリズム設計が必要です。

Woodland Academy

アルゴリズムとプログラム

- アルゴリズムとプログラムの違い：アルゴリズムは、問題解決のステップを抽象的に記述すればよいです。そして、プログラムはこれらの手順を対応する言語のコードで具体的に実装する必要があります。

- そのため、アルゴリズムは特定の言語に依存しないことが可能：同じアルゴリズムを様々な言語で実装できます。

アルゴリズムの評価指標

- 同じ問題を解決するために、様々な方法があります。異なるアルゴリズムのメリット・デメリットをどう比較しますか？

- 情報科学において、一般的にアルゴリズムの評価基準として使われるのは、そのアルゴリズムがどれだけの**リソース**を必要とするかということです。通常、リソースは大きく分けて「**時間リソース**」と「**空間リソース**」の 2 種類が考えられます。

- データ構造もアルゴリズムに従って実装されるため、上記の指標はデータ構造の各操作の評価にも利用できます。

次へ

Woodland Academy

- 時間リソースの評価指標は、**時間計算量**[Time Complexity]といいます。簡単に言えば、時間計算量が高ければ高いほど、アルゴリズムの実行速度は遅くなります。

- 空間リソースの評価指標は、**空間計算量**[Space Complexity]です。空間計算量が高いほど、アルゴリズムが実行するために使用する記憶空間（メモリとか）が多くなります。

- これらの計算量は、**オーダー**[Order]という、数値のレベルによって評定されます。

一般的な計算量のオーダー

- 次の表は、一般的な計算量のオーダー（低いものから高いもので）を示しています。どれが高いのかどれが低いのかを知っていれば十分で、定量的に理解する必要はありません。

名称	記法	よく見る書き方
常数時間（空間）	$O(1)$	O(1)
対数時間	$O(\log n)$	O(logn)、O(log(n))
線形時間	$O(n)$	O(n)
線形対数時間	$O(n \log n)$	O(nlogn)、O(n log(n))
二乗時間	$O(n^2)$	O(n^2)
多項式時間	$O(n^c)$	O(n^c)、O(n^k)
指数時間	$O(2^n)$	O(2^n)、O(c^n)

Woodland Academy

Q&A

Coffee Break

計算量と O-記法（1）

なぜアルゴリズムの評価に計算量を利用するのか？アルゴリズムは特定の言語やマシンに依存しないため、実際の実行時間（空間）や実行したコード量などで直接評価することは困難です。また、ある簡単なテスト状況でアルゴリズムが優れていても、状況がより複雑になった時に、その優位性が保たれることが保証できません。

計算量とは、単にリソースを消費する量ではなく、問題の規模が大きくなるにつれて消費量が変化する速度を評価するものです。言い換えれば、複雑性の低いアルゴリズムは、問題サイズが大きくなっても、リソースにそれほどコストをかけないです。

Coffee Break

計算量と O-記法（2）

アルゴリズムの正確な計算量を計算するのは非常に難しいことが多く、結果が複雑すぎて比較できないです。そのため、一般的には、統計数学の漸近的表記法を用いて、計算量の概算を計算して表します。

この記法は、**ランダウの漸近記法**[Landau Notation]といい、$O(g(x))$ という形で結果の大雑把のレベルを表現します。**O-記法**[Big-O Notation]とも呼ばれます。

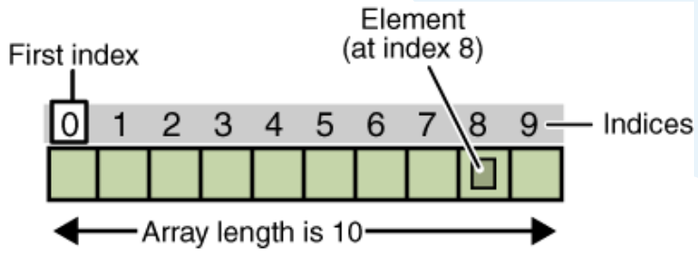
Woodland Academy

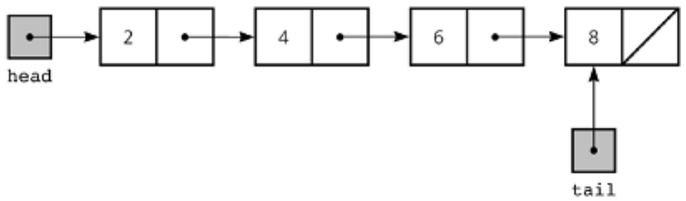
目次

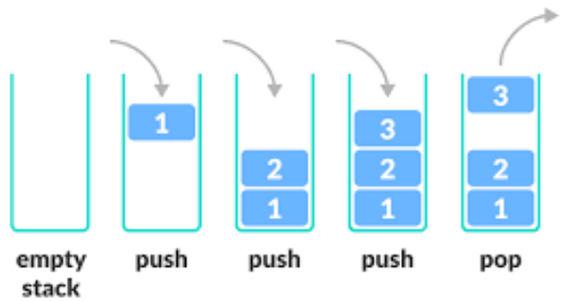
代表的なデータ構造と抽象型

- 実際には、私たちが使用するデータ構造のほとんどは、既にコードとして実装されます。したがって、私たちは、その機能と効率を理解し、システム設計時にどのように選択するかだけを知ればいいです。

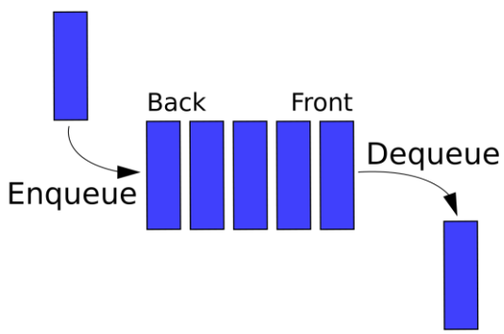
- 一般的なデータ構造は：
  - 配列
  - 連結リスト
  - グラフ、木

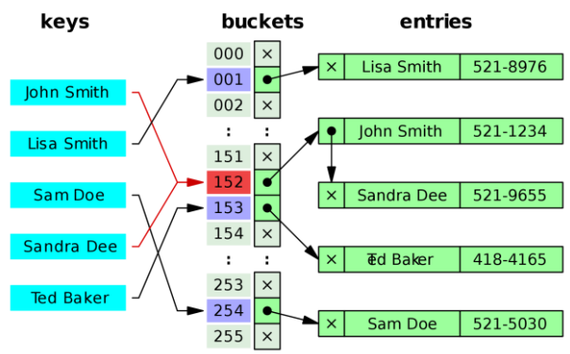






- 一般的な抽象型は：
  - リスト、スタック、キュー
  - セット（集合）
  - 連想配列（辞書）





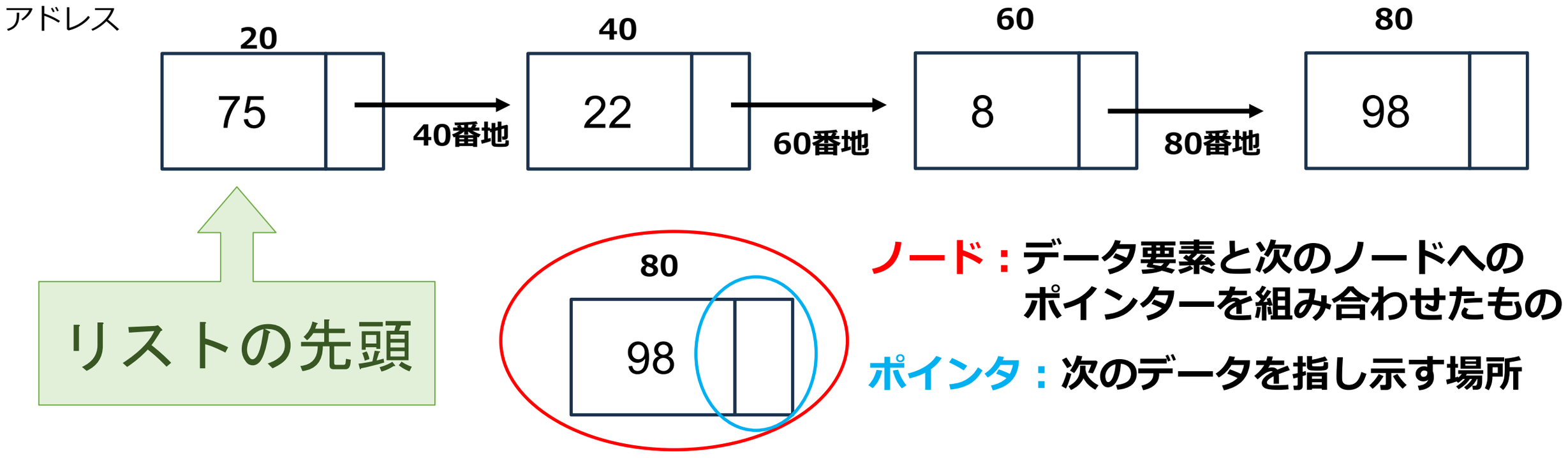
Woodland Academy

配列

- **配列**[Array]は最も基本的なデータ構造の一つです。私たちは既に、Java でその格納方法と機能について熟知したので、その特徴をおさらいしましょう：
  - 格納方法：配列では、全てのデータが**順番に**格納され、各データは整数の**インデックス**（添え字）を持つ。
  - 取得操作：インデックス i によって i 番目のデータの値を取得。
  - 設置操作：インデックス i によって i 番目のデータの値を変更。
- コンピュータ言語では、配列の実装は比較的簡単です：全てのデータをメモリに順次、連続的に格納すればよいです。そのため、ほとんどの言語では既に配列が用意されて、簡単な文法で利用できます。

連結リスト

- **連結リスト**[Linked List]は比較的基本的なデータ構造です。配列と同様、データを順番に格納するために使用されますが、その方法は若干異なります：連結リストのデータは、個々の**ノード**[Node]に存在し、各ノードは、次のノードだけにアクセスできます：



連結リストの格納方法と操作

- 格納方法：連結リストでは、全てのデータが**順番に**格納されます。 ただし、配列とは異なり、データにはインデックスが付かないが、各データは**次のデータ**がどこに存在するかを知ります。
- 先頭の取得：連結リストの最初のノードを取得。
- 繰り返し：あるノードの次のノードを取得。
- データの取得、設置：ノードに格納されているデータを取得、またはデータの値を変更。
- データ挿入：ノードの次に新しいノードを追加し、データを挿入。
- データ削除：連結リストからあるノードを削除。

Woodland Academy

連結リストの実装

- 多くの言語は連結リストを直接サポートしていませんが、基本構文を使って非常に簡単に実装できます。例えば、C/C++ では、「**ポインター**」を使用して次のノードのアドレスを記録し、連結リストの構造が実現できます。

- Java にはポインターはありませんが、「**参照**」の概念がそれに似ています。各ノードにデータと次のノードのオブジェクト（参照）を格納することによって連結リストが実現可能。

Try

LinkedList.java

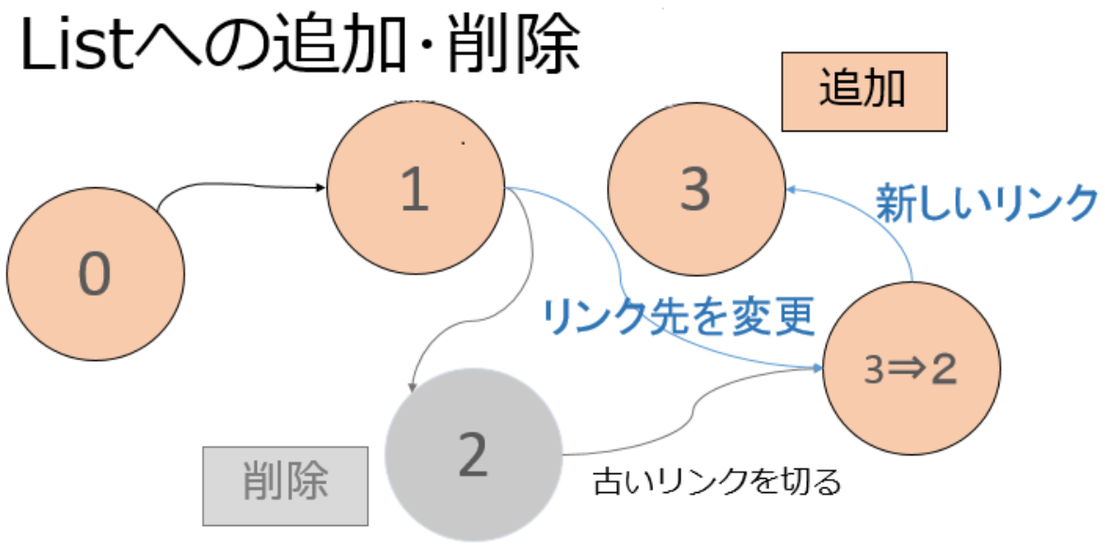
- 実際に使用される連結リストは、双方向リスト、循環リストなどの変種が多いです。

Woodland Academy

配列と（連結）リストの違い

- **データの持ち方**
  - ・リストは、新しい**要素が追加されるたびにメモリの領域を確保**
  - ・配列は、はじめに**メモリ上の連続した領域を確保**します

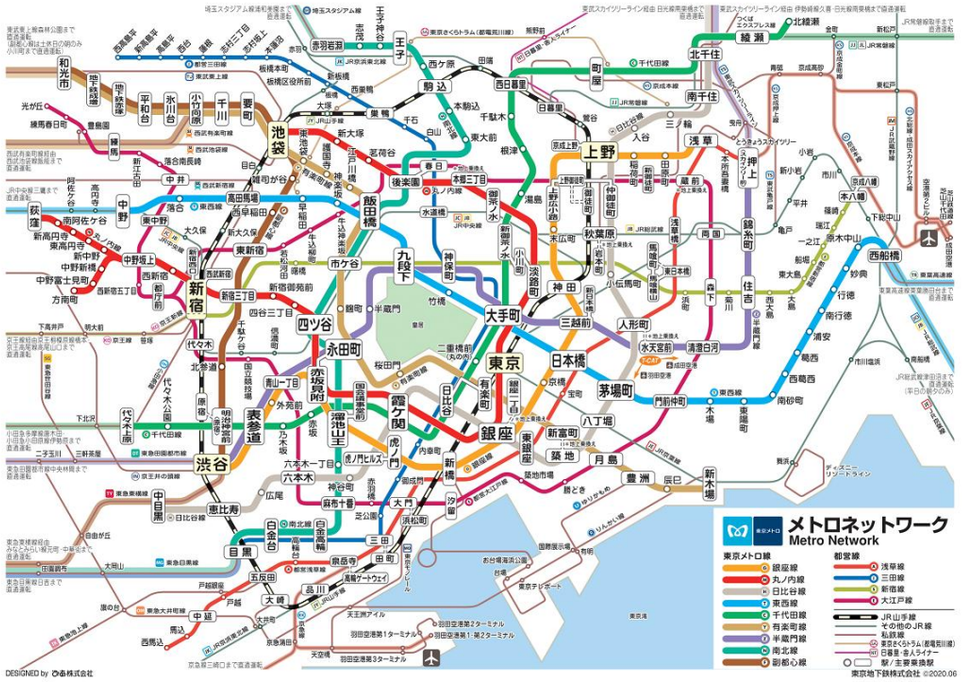
- **要素数の変動**
  - ・リストでは要素数の増減や入れ替えは、**リンク先の変更**を行うだけで済むため容易です。
  - ・配列の場合、はじめに**必要な領域を確保**している為、一度作ってしまうと**要素数の増減を行うことはできません。**



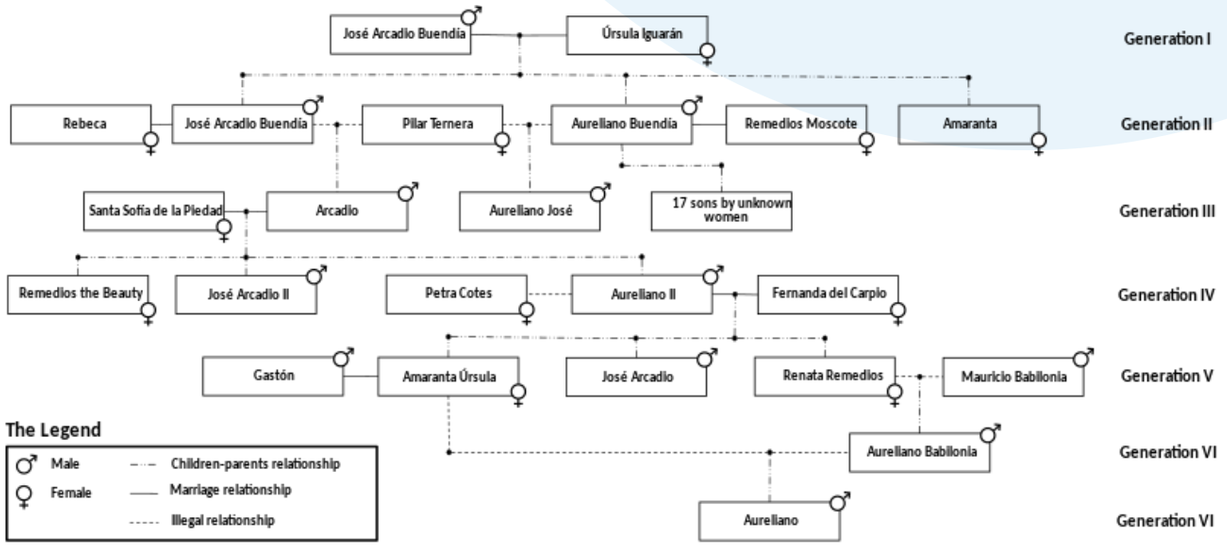
引用文献：
https://www.kenschool.jp/blog/?p=5308

グラフ

- **グラフ**[Graph]はよく使われるデータ構造です。グラフ内のデータも個々の**ノード**に格納されまが、各ノードは**エッジ**[Edge]を介して他の複数のノードと接続しています。日々使われるデータの多くは、グラフで表現できます：



地下鉄路線図



人物関係図

- グラフと図（写真、画像）の区別に注意してください。

Woodland Academy

グラフの格納方法と操作

- 格納方法：グラフでは、データ間に**順次的な関係はない**です。
  データはノードごとに存在し、各ノードは自分に**隣接するノード**（隣接ノード[Neighbour]）にアクセスできます。

- 隣接ノードの取得：ノードの隣接ノードを取得します。

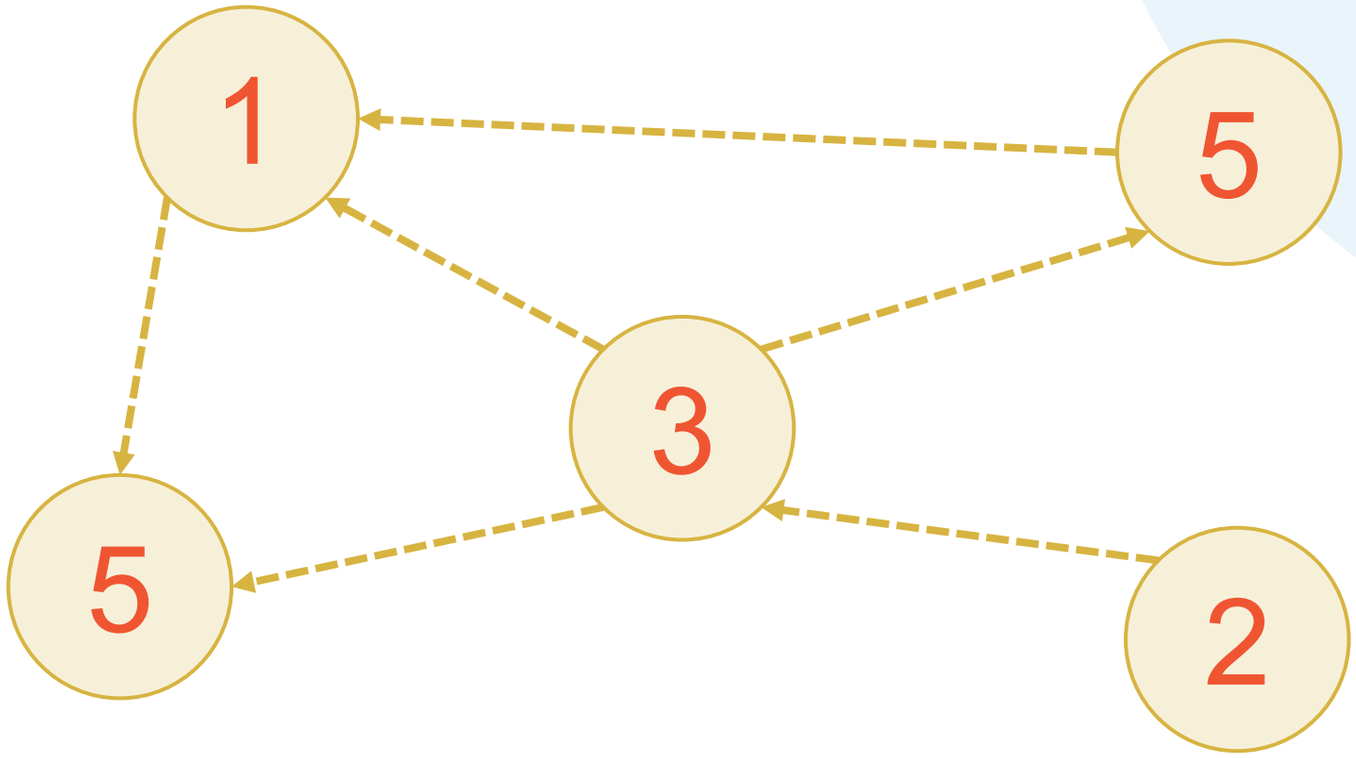
- データの取得、設置：ノードに格納されているデータを取得、またはそのデータの値を変更。

- ノードの追加、削除：ノードをグラフに追加、またはグラフから削除。

- エッジの追加、削除：エッジの追加または削除によって、ノード間の関係を変更。

グラフの実装

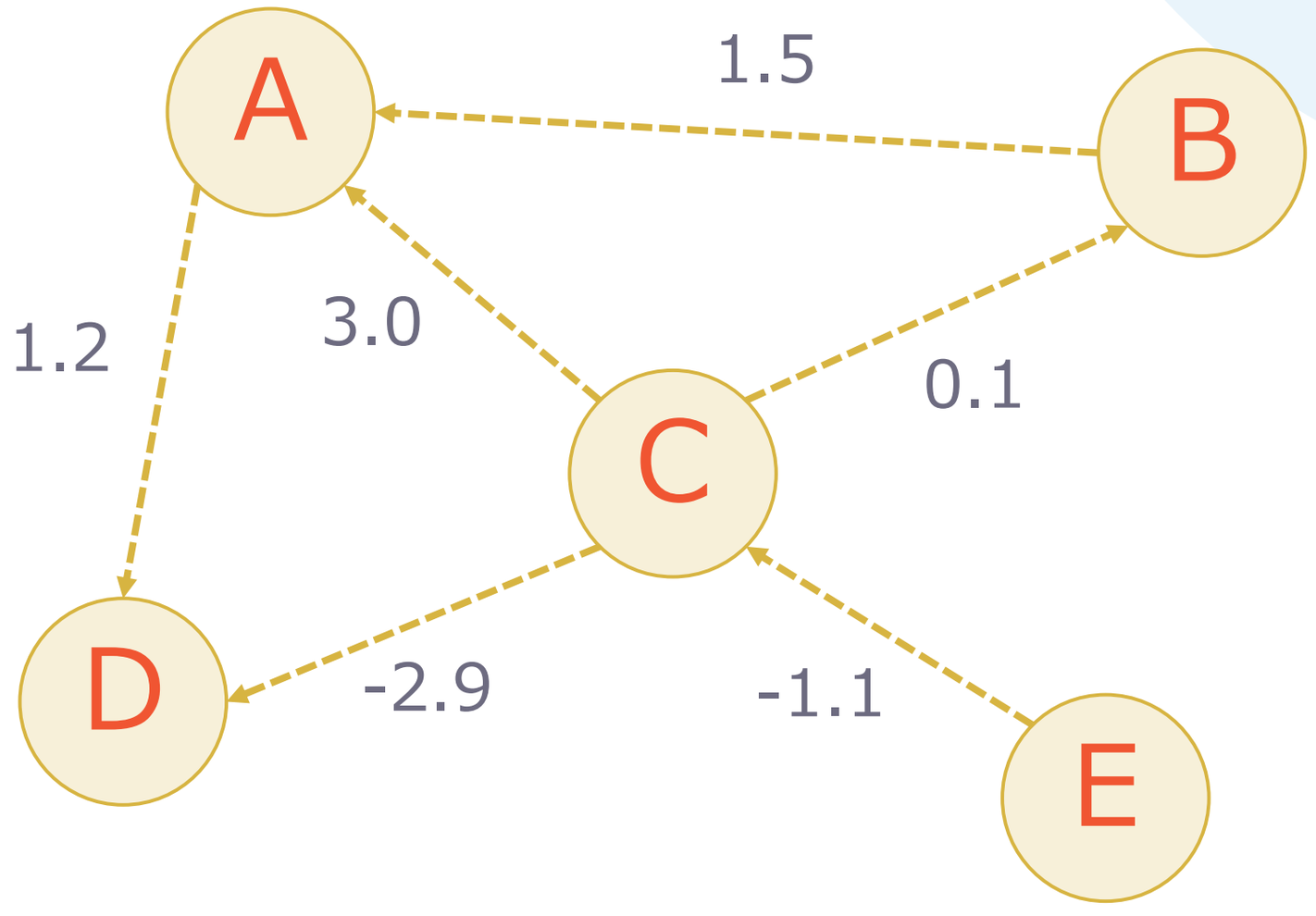
- Java では、グラフの構造もオブジェクトで実装できます：
  各ノードがデータと隣接ノードを保存。



- この形は、隣接リストという形式に近いです。この他、隣接行列や接続行列など、より多いエッジを持つグラフを格納するのに適した形式もあります。

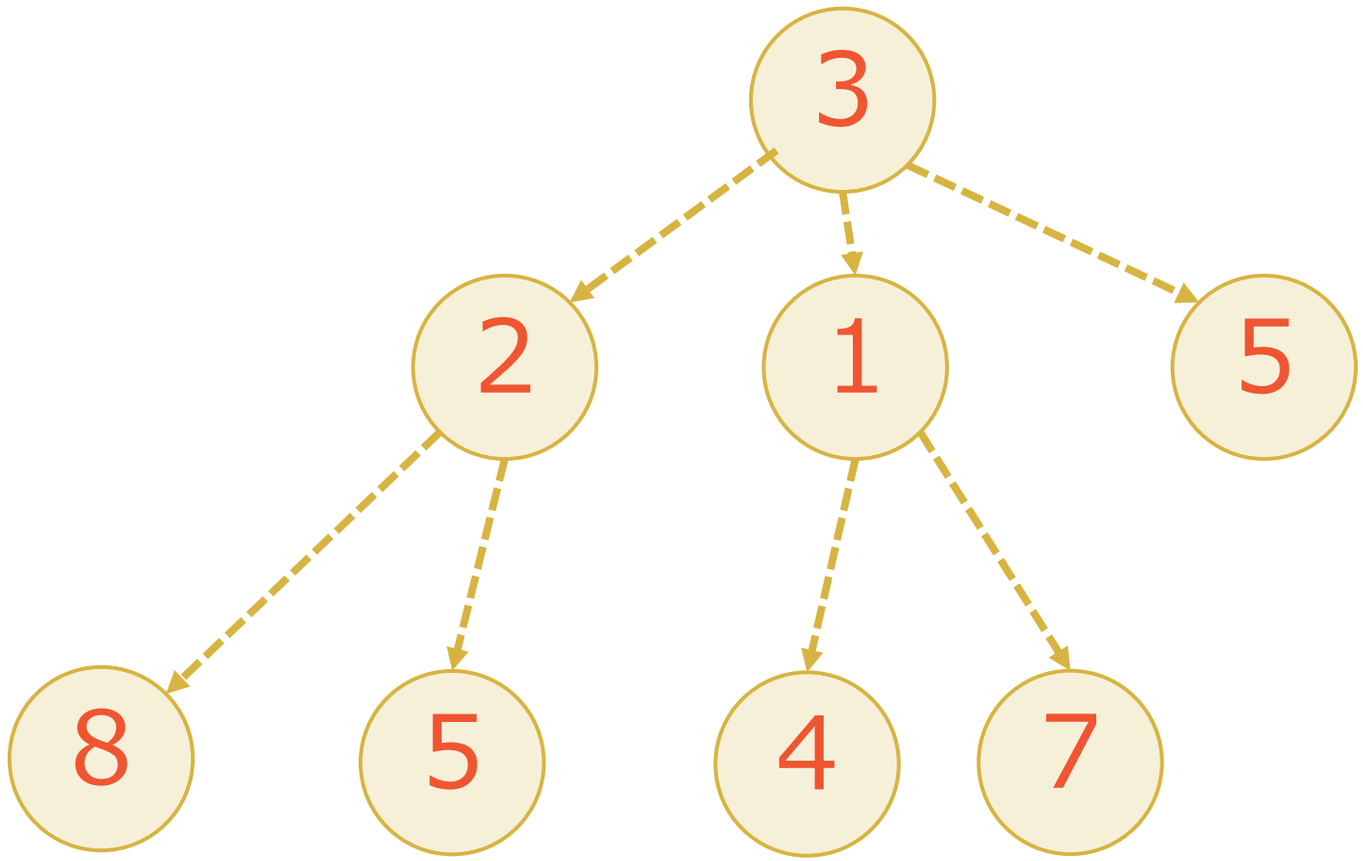
ネットワーク

- また、グラフの各エッジに 1 つのデータを記録することも可能で、このようなグラフは**重み付きグラフ**または**ネットワーク**[Network]と呼ばれます：



木

- **木**[Tree]とは、ループがない特別なグラフです。実際のデータ構造では、一般的に全てのノードがいくつかのレベルに分けられ、各ノードは次のレベルの何個のノードに接続されています：



木に関連する概念

- 一番高いレベルにあるは**根ノード**[Root]と呼ばれ、木の中に **1 つ**だけ存在します。

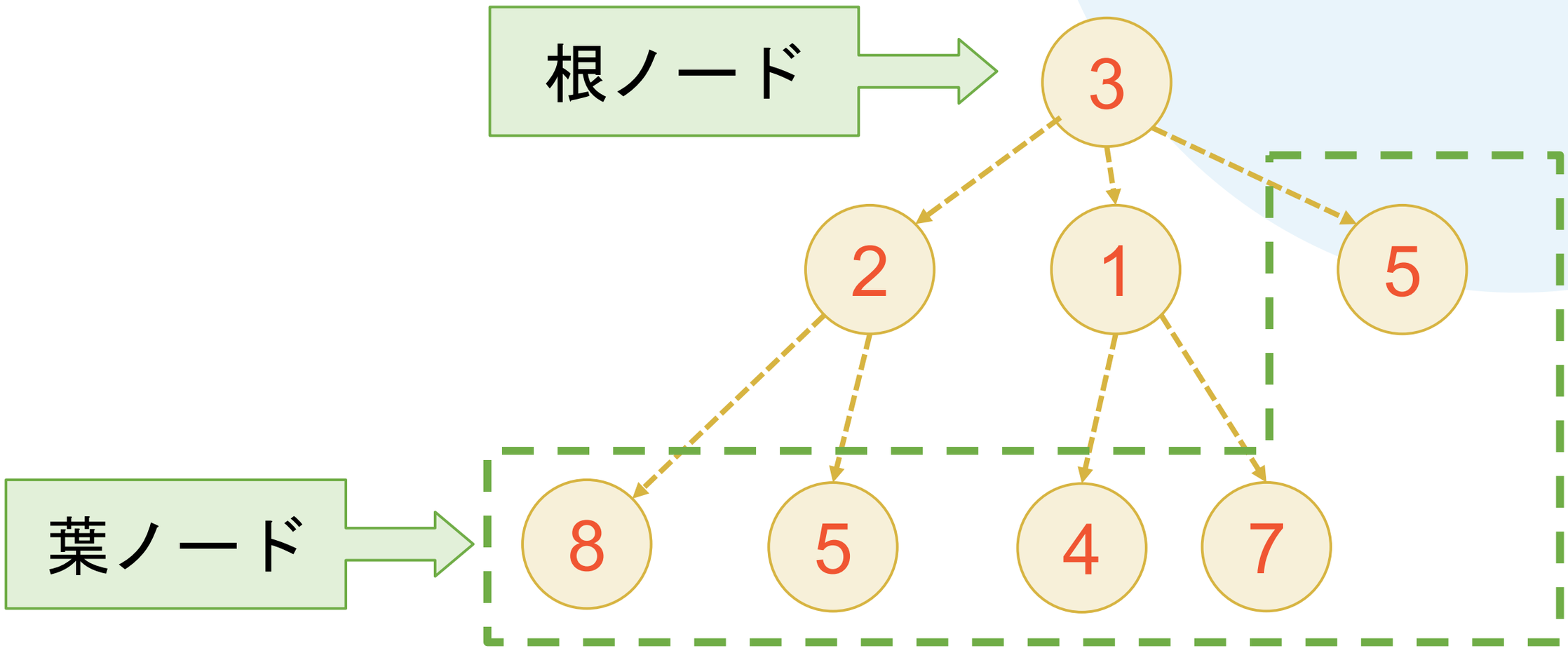
- 各ノードが接続する次のレベルのノードを**子ノード**[Child]と呼びます。各ノードは自分の子ノードの親ノード[Parent]です。各ノードには親ノードが **1 つ**だけ存在します。

- 自分と同じ親を持つノードを**兄弟ノード**[Sibling]といいます。親、または親の親、または親の親の親……などのノードは自分の**祖先ノード**[Ancestor]といい、その逆は**子孫ノード**[Descendant]といいます。

- 子ノードを持たないノードを**葉ノード**[Leaf]と呼びます。

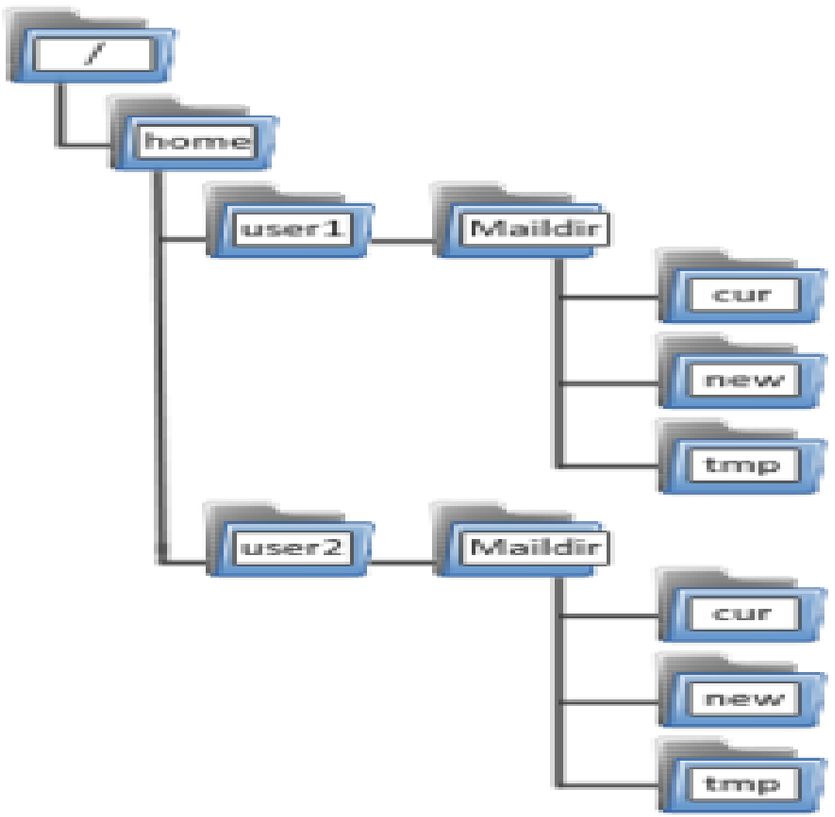
次へ

Woodland Academy



木の応用

- 木をどう実装するかは当分考えなくてもよいですが、実は日常生活で、フォルダシステムなど、既に似たような構造を使っているところが多くあります：



Tips

「ルートディレクトリ」「子フォルダ」等の言葉を聞いたことがありますか？いまなら、これらの名前の意味は分かるでしょうか。

- 木構造の他の使い処を考えましょう。

Woodland Academy

Woodland Academy

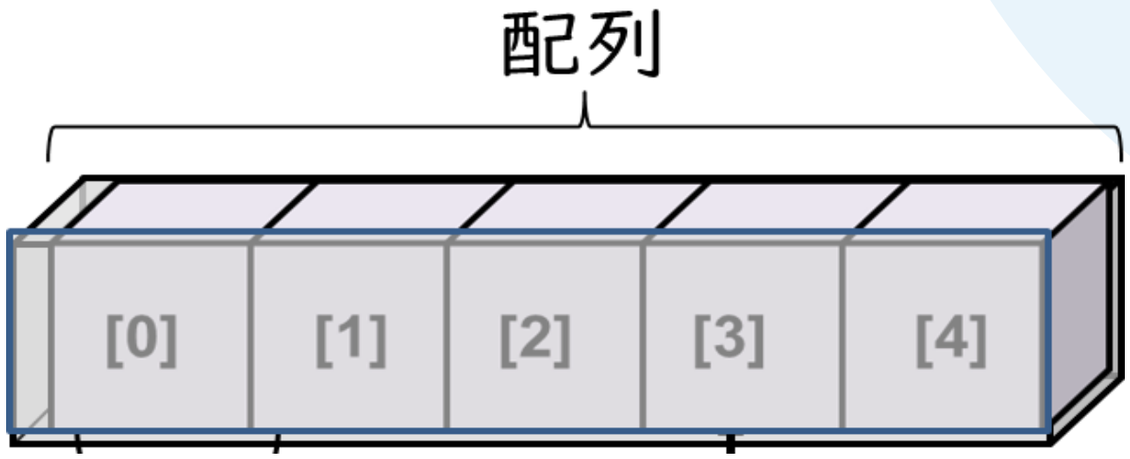
Q&A

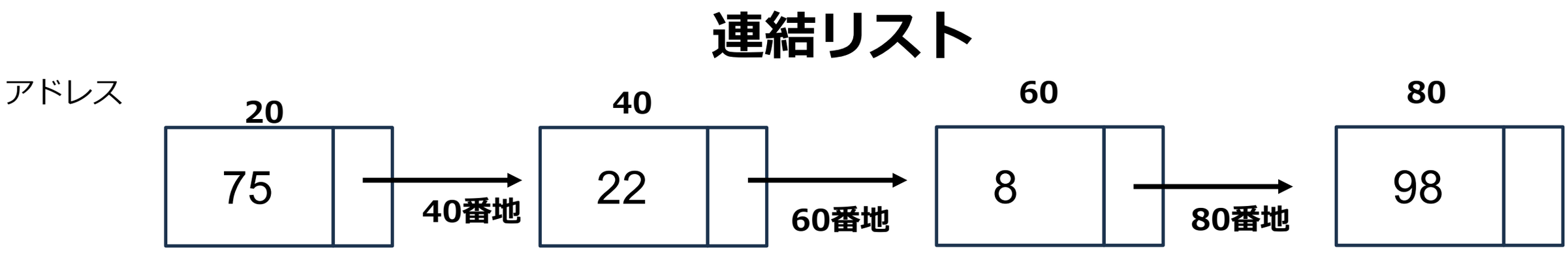
リスト

- **リスト**[List]は最もよく使われる抽象データ型の一つで、単純な**順序データ**を表現します：
  - 格納方法：リスト内のデータは順次に格納。配列のように、各データには、そのデータの数に対応する整数のインデックスが必要。
  - 取得、設置操作：インデックスに基づく位置のデータの取得、またはデータの設定。
  - 任意な挿入、削除操作：リストの任意の位置（インデックスによって）にデータを追加または削除。
  - 特殊な挿入、削除操作：先頭や末尾にデータを追加または削除。
  - 検索操作：リスト内のデータのインデックスを探し、またはリスト内にないことを確認。

リストの実装

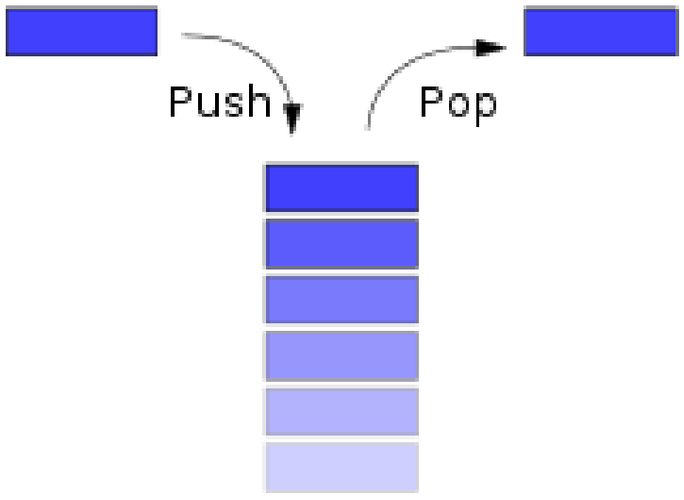
- リストの実装方法が一般的に二つがあります：**配列**を使うか、**連結リスト**使います。選択するために、それぞれどのような操作が効率的なのかだけを覚えておくとよいです：





スタック

- **スタック**[Stack]も、データを**順番に**格納する構造です。しかし、リストとは異なり、スタックは一方の端（先頭[Top]）からしかデータの取得、挿入、削除ができません：



- つまり、スタックの中に保存されるデータは常に、**後入れ先出し**[Last In First Out, LIFO]です。

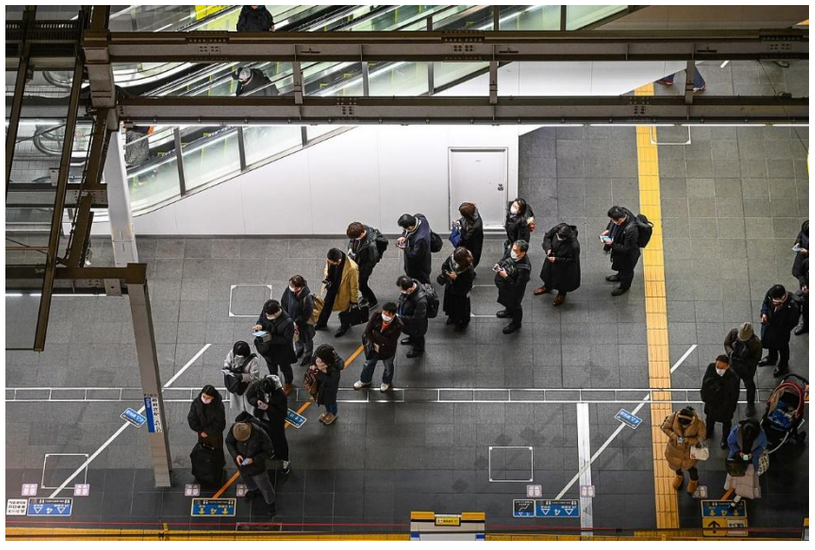
スタックの操作と実装

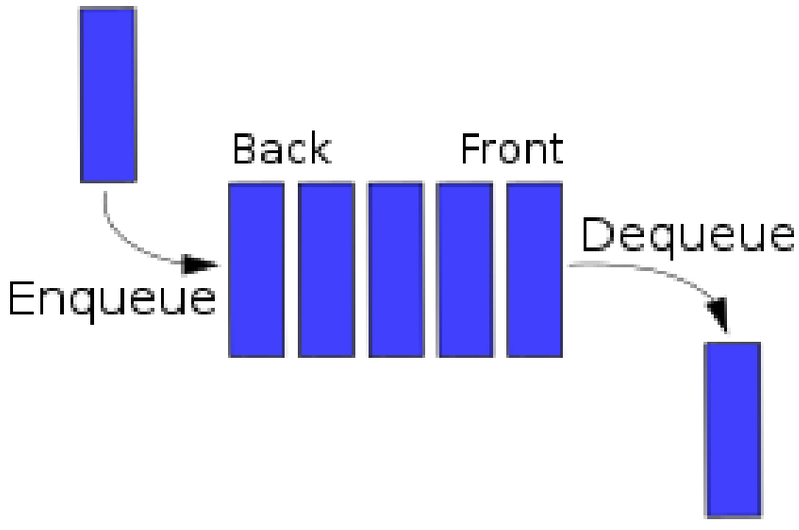
- スタックの操作：
  - 格納方法：リストのデータは順次に格納。
  - **プッシュ**[Push]：スタックの先頭に新しいデータを挿入。
  - **ポップ**[Pop]：スタックの先頭のデータを取得して使用（このデータはスタックから削除）。
  - **ピーク**[Peek]：スタックの先頭のデータを取得（削除せず）。
- スタックは一般に配列や連結リンクを使って簡単に実装できます。どちらの実装でも上記 3 つの操作の計算量は $O(1)$。
- スタックの使いどころ：ウェブブラウザの「戻る」機能や「進む」機能の実装に使用される。

Woodland Academy

キュー

- **キュー**[Queue]も、データを順番に格納する構造です。キューは、一方の端（末尾）またはからしかデータを挿入できず、もう一方の端（先頭）からしかデータの削除や検索ができません：





- スタックとは逆に、キュー内のデータは常に、**先入れ先出し** [First In First Out, FIFO]です。

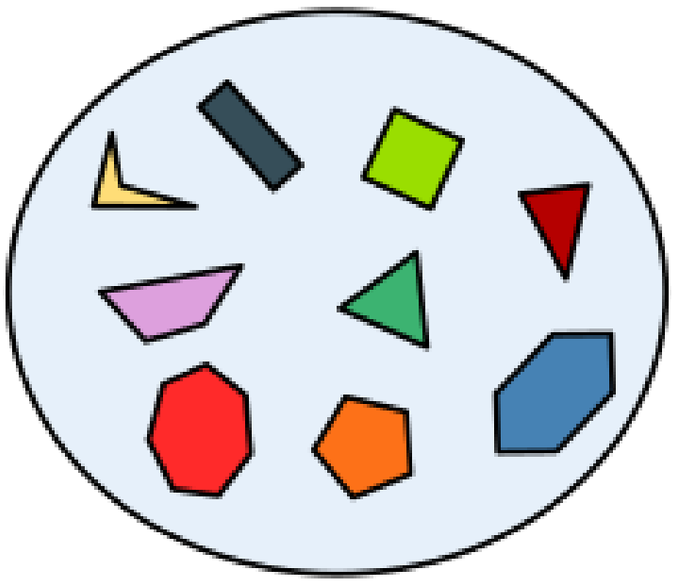
キューの操作と実装

- キューの操作：
  - 格納方法：キュー内のデータは順番に保存。
  - **エンキュー**[Enqueue]：キューの末尾に新しいデータを挿入。
  - **デキュー**[Dequeue]：キューの先頭のデータを取得して使用（このデータはキューから削除）。
  - ピーク：先頭のデータを取得（削除せず）　。
- キューは、配列や連結リストを使って簡単に実装できます。
- 一般的な実装では、上記 3 つの操作の計算量はともに $O(1)$。
- キューの使いどころ：最近使われたアイテムを追跡し、最も古いアイテムを削除するためのキャッシュ実装に使用される。

Woodland Academy

集合

- **集合**[Set]は、データを**順番に並べず**に保持する抽象型です。
  数学の集合と同様に、集合の中に**重複する要素はない**です：



$$\{1, 3, 2, 7\}$$

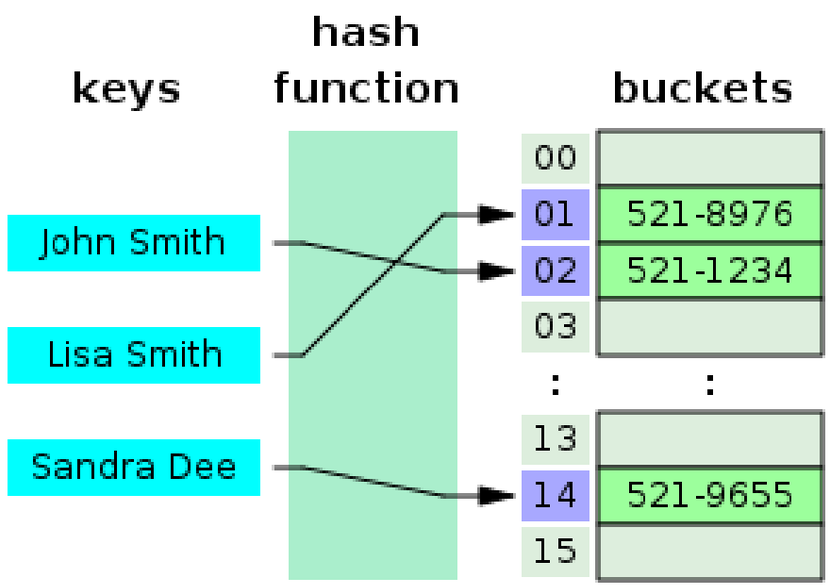
集合の操作と実装

- 集合の操作：
  - 格納方法：集合の中のデータは順序なしに保存。重複するデータはない。
  - 追加：新しいデータを集合に追加。
  - 削除：集合からデータを削除。
  - 検索：あるデータが、集合に含まれているかどうかを確認。
- 集合の一般的な実装として、ハッシュセット[Hash Table]があります。以上の全ての操作の時間計算量は $O(1)$。
- 集合の使いどころ：ECサイトでユーザIDや商品IDなど、一意の要素を保持する場合に使用される。

Woodland Academy

連想配列

- **連想配列**[Associative Array]、または**マップ**[Map]、**辞書**[Dictionary]は、添え字に従ってデータを格納する抽象型です。配列と異なり、連想配列は**数値以外の型**を添え字として扱えます。例えば、電話帳を保存するとき、連絡先の名前（文字列）を添え字として使用できます：



- ここで、添え字として使用されるデータを**キー**[Key]と呼ばれ、実際に保存されるデータを**値**[Value]と呼ばれます。1 つのキーは 1 つの値に対応し、重複するキーは存在しません。

Woodland Academy

連想配列の操作と実装

- 連想配列の操作：
  - 格納方法：連想配列のデータは、**キーと値のペア**で格納します。
  - 取得：キー（添え字）をもとに、対応する値（データ）を取得。
  - 設置：キーに対応する値を変更。
  - 検索：キーが連想配列にあるかどうかをチェック。
  - 削除：連想配列からキーと値のペアを削除します。
- 一般的な実装は**ハッシュテーブル**[Hash Table]で、上記の操作の時間計算量は全て $O(1)$。ハッシュテーブルの使用があんまりも頻繁のため、ハッシュテーブルと呼んだとき、実は連想配列という抽象型を指していることもしばしばあります。

Woodland Academy

Woodland Academy

Q&A

Coffee Break

ハッシュテーブルとハッシュ関数（1）

ハッシュテーブルの基本的な実装は、実はまだ配列を使っています。任意の型のキーを整数に変換し、この整数を、連想配列の値を格納する配列のインデックスとして使用することが必要です。

ここでの中心的な問題は、潜在的に大きな範囲を持つ値（例えば、文字列）を、指定された範囲内の整数に変換する方法です。ハッシュテーブルの実装は、大きなデータから整数への適切なマッピング、すなわち**ハッシュ関数**[Hash Function]を選択することに重点が置かれます。

Coffee Break

ハッシュテーブルとハッシュ関数（2）

そして、ハッシュ関数によって得られた整数をハッシュ値[Hash]と呼ばれます。 ハッシュ値はデータの「指紋」のようなもの：異なるデータは（大きな確率で）異なるハッシュ値を持ちます。

さらに、ハッシュ関数にはもう一つ重要な特徴がある：ハッシュ値から元のデータを推測することができない、という**不可逆性**です。

Coffee Break

ハッシュテーブルとハッシュ関数（3）

この特徴から、ハッシュ関数は情報セキュリティの分野で非常に有用です。 例えば、ウェブサイト利用者のパスワードが直接サーバに保存されている場合、ハッキングにより全ての利用者のパスワードが流出する可能性があります。しかし、ユーザのパスワードのハッシュを保存しておけば、ユーザーがログインする際にパスワードが正しく入力されたかどうかを判断することができる同時、サーバのデータが漏洩しても、ハッカーは元のパスワードが復元できません。

また、ハッシュ関数は、データの暗号化、データの検証、仮想通貨などの分野でも利用されています。

Woodland Academy

Woodland Academy

Q&A

まとめ

Sum Up

1.データ構造とアルゴリズムの基本概念：
① データ構造の選択プロセス。
② データ構造とアルゴリズムの評価基準。

2.基本的なデータ構造：
① リスト、集合、辞書などの主要な抽象構造を覚えましょう。

Woodland Academy

Woodland Academy

Thank you!

From Seeds to Woodland — Shape Your Future.

Shape Your Future