データ構造とアルゴリズム チートシート

第1部: 探索アルゴリズムと計算量

重要公式

- **m-ブロック法の最悪比較回数:** $h = m - 1 + \lceil n/m \rceil$
- **相加相乗平均によるmの最適化:** $m + n/m \geq 2\sqrt{m \cdot \frac{n}{m}} = 2\sqrt{n}$。$m = \sqrt{n}$ のとき最小値をとる。

主要アルゴリズム

- **逐次探索法 (線形探索):**
  - **概要:** 配列の先頭から順に目的の値と一致するか比較していく 。
  - **計算量:**
    - 最悪計算時間: $O(n)$
    - 最良計算時間: $O(1)$
    - 平均計算時間: $O(n)$ （仮定によるが、一般的に）
- **m-ブロック法:**
  - **概要:** ソート済みの配列をm個のブロックに分割する。まず、探す値xが含まれるブロックを特定し、そのブロック内を逐次探索する。
  - **計算量:** 最悪計算時間はブロック数$m = \sqrt{n}$のとき$O(\sqrt{n})$となる。
- **2分探索法 (Binary Search):**
  - **概要:** ソート済みの配列の中央値と目的の値を比較し、探索範囲を半分に絞り込んでいく。
  - **計算量:** 最悪計算時間は$O(\log n)$。
- **ハッシュ法 (オープンアドレス法):**
  - **概要:** ハッシュ関数を用いてデータ格納先のインデックス（ハッシュ値）を計算する。もし衝突（同じハッシュ値）が発生した場合、次の空き場所を探して格納する 。
  - **計算量:**
    - 最悪計算時間: $O(n)$
    - 平均計算時間: 定数時間 $O(1)$ (占有率αに依存)

基本用語

- **アルゴリズム:** 問題を解くための手順や手法 。
- **オーダ記法 ($O(f(n))$):** 関数の増加の速さを評価する記法で、アルゴリズムの計算量（特に実行時間）の漸近的な上限を示す 。$g(n)$が$O(f(n))$であるとは、ある正の定数cとn_0が存在し、全ての$n > n_0$に対して$g(n) \leq cf(n)$が成り立つことを意味する 。
- **Ω記法 ($\Omega(f(n))$):** 関数の漸近的な下限を示す記法 。$cf(n) \leq g(n)$が成り立つこと 。
- **Θ記法 ($\Theta(f(n))$):** 関数の増加の速さを正確に（上下から）評価する記法。$O(f(n))$と$\Omega(f(n))$が両方成り立つ状態 。

- **タイトな評価:** アルゴリズムの計算量を、それ以上増加速度の小さい関数では評価できない、最も的確なオーダで評価すること 。
- **最悪時計算時間:** 最も時間がかかる入力が与えられた場合の計算時間 。
- **最良時計算時間:** 最も時間がかからない入力が与えられた場合の計算時間 。
- **静的探索問題:** 探索中にデータ集合が変化しない問題 。
- **動的探索問題:** 探索中にデータ集合が変化する問題 。

第2部: 基本データ構造

主要アルゴリズム

- **スタック操作:**
  - `Push(S, d)`: スタックSにデータdを追加する。
  - `Pop(S)`: スタックSから最後に追加されたデータを取り出す。
- **キュー操作:**
  - `Enqueue(Q, d)`: キューQにデータdを追加する。
  - `Dequeue(Q)`: キューQから最初に追加されたデータを取り出す。

基本用語

- **配列 (Array):** メモリ上の連続した領域にデータを格納するデータ構造。インデックスを指定して$O(1)$時間で任意の位置のデータにアクセス可能（ランダムアクセス）。データの挿入・削除は、要素の再配置が必要なため時間がかかることがある 。
- **連結リスト (Linked List):** データと次の要素へのポインタ（参照）を組にしたノードを数珠つなぎにしたデータ構造 。i番目の要素へのアクセスには$O(i)$時間かかる（シーケンシャルアクセス）。データの挿入・削除は、ポインタの付け替えのみで行えるため高速 。
- **スタック (Stack):** 最後に入れたデータを最初に取り出す (LIFO: Last-In, First-Out) という制約を持つデータ構造 。
- **キュー (Queue):** 最初に入れたデータを最初に取り出す (FIFO: First-In, First-Out) という制約を持つデータ構造 。
- **抽象データ型 (ADT):** データの具体的な実現方法とは無関係に、データに対する操作の仕様によって定義されるデータ型 。スタックやキューはその一例 。

第3部: 木構造とソート

【追加】木の走査 (Traversal)

木の全てのノードを体系的に訪問する方法。

- **先行順 (Pre-order):** （根 → 左部分木 → 右部分木）の順で訪問する。
- **中間順 (In-order):** （左部分木 → 根 → 右部分木）の順で訪問する。二分探索木をこの順で走査すると、キーがソートされた順序で得られる。
- **後行順 (Post-order):** （左部分木 → 右部分木 → 根）の順で訪問する。

主要アルゴリズム

- **2分探索木 (BST) の操作:**
    - **探索:** 根から比較を開始し、キーの値に応じて左か右の部分木へ探索を進める。計算時間は木の高さhに比例し、$O(h)$。
    - **挿入:** 探索と同様の手順で挿入位置を決定し、新しいノードとして追加する。
    - **削除:** 削除対象ノードの子の数（0, 1, 2個）に応じて、3つのケースに分けて処理を行う。
        1. **子が0個の場合:** そのままノードを削除する。
        2. **子が1個の場合:** そのノードを削除し、子ノードを元の位置に付け替える。
        3. **子が2個の場合:** 削除対象ノードの右部分木の最小値（または左部分木の最大値）を持つノードで値を置き換え、その後その最小値（最大値）ノードを削除する。
- **バブルソート (Bubble Sort):**
    - **概要:** 隣り合う要素を比較・交換しながら、最大値を配列の末尾に移動させる。これを繰り返す。
    - **計算量:** $O(n^2)$
- **選択ソート (Selection Sort):**
    - **概要:** 未ソート部分から最大値（または最小値）を探し、未ソート部分の末尾の要素と交換する。これを繰り返す。
    - **計算量:** $O(n^2)$
- **挿入ソート (Insertion Sort):**
    - **概要:** ソート済み部分に、未ソート部分の要素を適切な位置に挿入していく。これを繰り返す。
    - **計算量:** 最悪$O(n^2)$、最良$O(n)$
- **ヒープソート (Heap Sort):**
    - **概要:** 配列からヒープを構成し、根（最大/最小値）を配列末尾と交換しながらソートする。
    - **計算量:** $O(n \log n)$
- **クイックソート (Quick Sort):**
    - **概要:** 基準値（ピボット）を選び、大小で分割する処理を再帰的に行う。
    - **計算量:** 平均$O(n \log n)$、最悪$O(n^2)$
- **マージソート (Merge Sort):**
    - **概要:** 配列を半分に分割し続け、要素が1つになったら、ソートしながら統合（マージ）していく。
    - **計算量:** $O(n \log n)$
- **基数ソート (Radix Sort):**
    - **概要:** 比較に基づかないソート。最下位の桁から順番に、各桁の値に基づいて安定ソートを繰り返す。
    - **内部アルゴリズム:** 各桁のソートには、計数ソート（Counting Sort）がよく用いられる。計数ソートは、各値の出現回数を数え上げ、その累積和を利用してソート後の位置を決定する$O(n + k)$のアルゴリズム（kは値の範囲）。
    - **計算量:** データがd桁で、各桁が取りうる値の種類がkの場合、$O(d(n + k))$。

基本用語

- **2分探索木 (Binary Search Tree, BST):** 任意のノードにおいて、「（左の子孫のキー）$\leq$（自身のキー）$\leq$（右の子孫のキー）」という条件を満たす2分木。
- **平衡2分探索木 (Balanced BST):** 木の高さが$O(\log n)$に保たれるように工夫されたBST。AVL木や赤黒木などがある。探索・挿入・削除の最悪計算量が$O(\log n)$となる。
- **ヒープ (Heap):** 「親のキーは子のキーより大きい（または小さい）」という条件を満たす完全2分木。配列で効率的に表現できる。優先度付きキューの実装によく使われる。
- **安定ソート (Stable Sort):** 同じ値を持つ要素の元の順序関係が、ソート後も保たれるソートアルゴリズム。

第4部: アルゴリズム設計技法

主要アルゴリズム

- **ユークリッドの互除法 (再帰):**
  - **概要:** 2つの整数n, mの最大公約数$\gcd(n, m)$を求める。$\gcd(n, m) = \gcd(m, n \bmod m)$という性質を再帰的に利用する。
  - **計算量:** $O(\log m)$
- **最近接ペア問題 (分割統治法):**
  - **概要:** 平面上の点集合をx座標で半分に分割し、左右の領域で再帰的に最近接ペアを求める。その後、分割線をまたぐペアを効率的に探索し、全体の最近接ペアを見つける。
  - **計算量:** $O(n \log n)$
- **0-1ナップサック問題 (動的計画法):**
  - **概要:** アイテムを1からi番目まで、容量をjまでとしたときの価値の最大値を表に記録していく。$D[i][j] = \max(D[i-1][j], D[i-1][j-w_i] + v_i)$ の漸化式を解く。
  - **計算量:** $O(nC)$（n:アイテム数, C:容量）

基本用語

- **再帰 (Recursion):** ある関数（手続き）の中で、自分自身を呼び出すこと。問題をより小さい同じ構造の問題に帰着させて解く 。
- **分割統治法 (Divide and Conquer):** 問題を複数の部分問題に分割し、部分問題を再帰的に解き、それらの解を統合して元の問題を解く手法 。
- **動的計画法 (Dynamic Programming, DP):** 小さな部分問題から順に解き、その結果を表に記録して再利用することで、計算の重複を避ける手法 。

第5部: グラフアルゴリズム

主要アルゴリズム

- **幅優先探索 (Breadth-First Search, BFS):**
  - **概要:** 始点から近い順に頂点を探索する。キューを用いて実装される。
  - **応用:** 重みなしグラフの最短経路問題。
  - **計算量:** $O(n+m)$（n:頂点数, m:辺数）
- **深さ優先探索 (Depth-First Search, DFS):**

    - **概要:** 始点から行けるところまで深く探索し、行き止まったら戻って別の経路を探索する。スタック（または再帰）を用いて実装される。
    - **応用:** トポロジカルソート、連結成分分解など。
    - **計算量:** $O(n+m)$
- **ダイクストラ法 (Dijkstra's Algorithm):**
    - **概要:** 負の重みがないグラフにおける単一始点最短経路問題を解く。始点から距離が確定した頂点の集合を広げていく。優先度付きキュー（ヒープ）を用いると効率的。
    - **計算量:** ヒープ（優先度付きキュー）を使うと$O((n+m)\log n)$。
    - **応用（経路復元）:** 各頂点について、最短経路上の直前の頂点を配列$\mathrm{prev}[v]$に記録しておく。ゴールからprev配列を逆にたどることで、最短経路を復元できる。
- **クラスカル法 (Kruskal's Algorithm):**
    - **概要:** 最小全域木（MST）を求めるグリーディアルゴリズム。辺を重みの小さい順にソートし、閉路を作らないように辺を追加していく 。閉路判定にはUnion-Findデータ構造が有効 。
    - **計算量:** $O(m \log n)$
- **Ford-Fulkerson法 (最大フロー):**
    - **概要:** ネットワークの最大フローを求める。残余グラフ上で始点から終点への増加道（フローを増やせる経路）を見つけ、フローを更新する。これを増加道がなくなるまで繰り返す 。
    - **計算量:** 増加道の選び方によるが、BFSを用いる(Edmonds-Karp)と$O(nm^2)$ 。

基本用語

- **グラフの表現:**
    - **隣接行列:** $n \times n$の行列で、辺(i, j)が存在すれば$A[i][j] = 1$（または重み）とする。メモリは$O(n^2)$ 。
    - **隣接リスト:** 各頂点から出ている辺の接続先頂点をリストで保持する。メモリは$O(n+m)$ 。
- **最短経路問題:** グラフ上の2頂点間の重みが最小となる経路を求める問題 。
- **最小全域木 (MST):** 連結な重み付きグラフの全ての頂点を含む木のうち、辺の重みの総和が最小になるもの 。
- **ネットワークフロー:** 容量制限のある有向グラフ（ネットワーク）において、始点（ソース）から終点（シンク）へ流せる「もの」の流れ 。

第6部: 計算困難性

主要アルゴリズム

- **2-Opt法 (TSPの近似解法):**
    - **概要:** 巡回セールスマン問題(TSP)に対する局所探索法。巡回路から2辺を選び、交差しないように繋ぎ変えることで、より短い経路を探す 。
- **焼きなまし法 (Simulated Annealing):**
    - **概要:** 局所探索法の改良版。解が悪化する方向へも確率的に遷移することで、局所最適解に陥るのを避ける発見的手法 。
- **分枝限定法 (Branch and Bound):**
    - **概要:** 最適解を厳密に求める手法。解の探索を木構造で行い、途中で最適解になり得ないと判断された枝（部分解）の探索を打ち切る（枝刈り）ことで効率化する 。

基本用語

- **巡回セールスマン問題 (TSP):** 全ての都市を一度ずつ訪れて出発点に戻ってくる最短の巡回路を求める問題 。
- **クラスP:** 多項式時間で解を求めることができる問題のクラス 。
- **クラスNP:** 解が与えられたとき、その正しさを多項式時間で検証（答え合わせ）できる問題のクラス 。
- **NP完全問題 (NP-Complete):** クラスNPに属する問題の中で「最も難しい」問題のクラス。NPに属する任意の問題から多項式時間で帰着できる 。P≠NPという予想のもとでは、多項式時間で解けないと考えられている 。
- **多項式時間帰着:** ある問題Aのインスタンスを、別の問題Bのインスタンスに多項式時間で変換し、Bを解くことでAの解が得られるようにすること。BがAより難しくないことを示す 。
- **近似アルゴリズム:** 最適解を保証する代わりに、高速に「そこそこ良い」解（近似解）を求めるアルゴリズム 。