

「プログラム設計とアルゴリズム」レポート課題

2022/01/27

アンダーランド ジェイク

1A193008-2

期末答案

1.

N 個の整数 $a_i (i = 0, 1, \dots, N - 1)$ 、 N 個の整数 $b_i (i = 0, 1, \dots, N - 1)$ 、および整数 K が与えられている。この時、与えられた2個の整数列から、1つずつ整数を選んでその和を計算するものとする。その和の値のうち、 K 以上の範囲での最小値を $O(N \log N)$ の計算量で求めるアルゴリズムを設計し、そのアルゴリズムの概略を日本語で説明せよ。また、解答したアルゴリズムの計算量が $O(N \log N)$ となる理由の概要を日本語で説明せよ。

--- 解 ---

まず、 b_i をソートする。これは $O(N \log N)$ の計算量でできる。

次に、一つずつ a_i をループし、 a_i ごとに $b_j \geq K - a_i$ となる b_j の最小値を求める。この時、 b_i はソートされているので、二分探索法を用いれば、 $O(\log N)$ で目的の b_i を求めることができる。二分探索法は探索を繰り返し、そのたびに選択肢を半分に絞るので、 2^k の選択肢を k 回で一つに絞ることができ、 $N = 2^k$ とおくと N を $k = \log N$ 回で絞ることができるのでその計算量は $O(\log N)$ となる。総じて、この段階での操作は $O(\log N)$ を N 回繰り返すので $O(N \log N)$ である。

組み合わせると、 $O(N \log N) + O(N \log N) = 2O(N \log N) = O(N \log N)$ である。

```
In [ ]: # コード例
import math
import bisect

def pair_sum_min(a, b, K):
    min_sum = math.inf
    b = sorted(b) # O(N log N)
    n = len(b)
    for ai in a: # O(N)
        i = bisect.bisect_left(b, K - ai) # lowerbound function, O(log N)
        #print(ai, bi)
        if i < n:
            min_sum = min(min_sum, ai + b[i])
    return min_sum

print(pair_sum_min([3, 4, 5, 6, 5, 7], [10, 20, 30, 19, 50, 34], 21))
```

2.

N 個の整数 $s_i (i = 0, 1, \dots, N - 1)$ 及び N 個の整数 $t_i (i = 0, 1, \dots, N - 1)$ が与えられている。ここで、 s_i 及び t_i はそれぞれ、仕事 i の開始時刻と終了時刻を意味するものとする。ある時刻において同時に出来る仕事は高々 1 つであり、各仕事は途中から参加したり途中でやめたりすることは出来ないものとする。ただし、仕事 a の終了時刻と仕事 b の開始時刻が同一であった場合には、 a と b の仕事は両方行う事が出来るものとする。この時、行う事が出来る仕事の最大数を求めるアルゴリズムを設計し、そのアルゴリズムの概略を日本語で説明せよ。

また、解答したアルゴリズムの計算量を記し、その計算量となる理由の概要を日本語で説明せよ。

--- 解 ---

この問題は、常に終了時刻が最も早い仕事を選ぶ greedy アルゴリズムにより仕事数を最大化できる。これは、終了時刻が最も早い仕事が、残りの仕事を入れられる時間を最大化するからである。現時点でも最も終了時刻の早い仕事の終了時刻が t_x であるとする。この仕事を選択すると、他の仕事を入れられる区間は $[t_x:]$ である。もし終了時刻が t_y の他の仕事を選べば、活用できる残り時間は $[t_y:]$ となり、 $t_x \leq t_y$ であるので、 $[t_y:]$ が $[t_x:]$ より多く仕事を持つことは不可能である。

よって、まずは $O(N \log N)$ でソートを行う。その後、終了時刻が最も早い仕事を見つけ、その仕事の終了時刻以後に開始時刻がくる仕事のうちで終了時刻が最小のものを選択する、という過程を繰り返す。このうち、ソート後に終了時刻が最小になるものを探すのは $s_y \geq t_x$ の条件を確認しながら仕事の区間リストをループしていくべきから、最悪計算時間は $O(N)$ である。

したがって、全体の最悪計算量は $O(N \log N)$ である。

```
In [ ]: S = [1, 3, 6, 7, 9]
T = [3, 8, 7, 10, 11]
def scheduling_problem(S, T):
    cnt = 0
    selected_t = -1 # 現在選択されている仕事の終了時刻
    jobs = sorted(list(zip(S, T)), key=lambda x: x[1]) # 終了時刻でソート, O(NlogN)
    for s, t in jobs:
        if s >= selected_t:
            selected_t = t
            cnt += 1
    return cnt

print(scheduling_problem(S,T))
```

3.

サイズが 5 のハッシュテーブルに、`{apple, banana, orange, peach}` という 4 つのデータをこの順に挿入することを考える。ハッシュ関数 `h(x)` には「`x` の文字数を 5 で割った余り」を用いる。また、ハッシュ値の衝突対策としてはオープンアドレス法(クローズドハッシュ法)を用い、再ハッシュの方法には線形探索法「 $(h(x)+i)\%5$ (i は再ハッシュの回数)」を用いるものとする。

1. 4 つのデータを全て挿入した時点におけるハッシュテーブルの状態を図示せよ。
2. 1 で示したハッシュテーブルから、`orange` を削除した後のハッシュテーブルの状態を図示せよ。

--- 解 ---

1.

location	value
0	apple
1	banana
2	orange
3	peach
4	nil

2.

location	value
0	apple
1	banana
2	deleted
3	peach
4	nil

4.

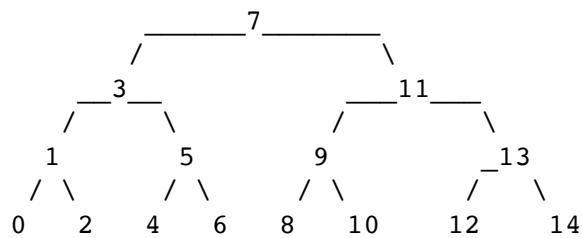
1. 二分探索木とはどのようなデータ構造であるか、例を図示しながら簡潔に説明せよ。構造のみが説明されていればよく、検索・挿入・削除などの操作について説明する必要はない。
2. 二分探索木において、子が二つある要素を削除する場合、どのような処理が行われるか。例を図示しながら簡潔に説明せよ。
3. 二分探索木に格納されているデータの個数が N である時、要素を削除する操作の最悪計算量を記せ。

--- 解 ---

1. 二分探索木は、それぞれの頂点が最高2つの子を持つ二分木で、次の特徴を満たすものを言う：
全ての頂点 v において、 v の左部分木に含まれるどの頂点 v_l に対しても $v.key() \geq v_l.key()$ であり、 v の右部分木に含まれるどの頂点 v_r にたいしても $v.key() \leq v_r.key()$ が成り立つ。
以下に、その例を記す。全てのノードで上の特徴が成り立つことが観察できる。

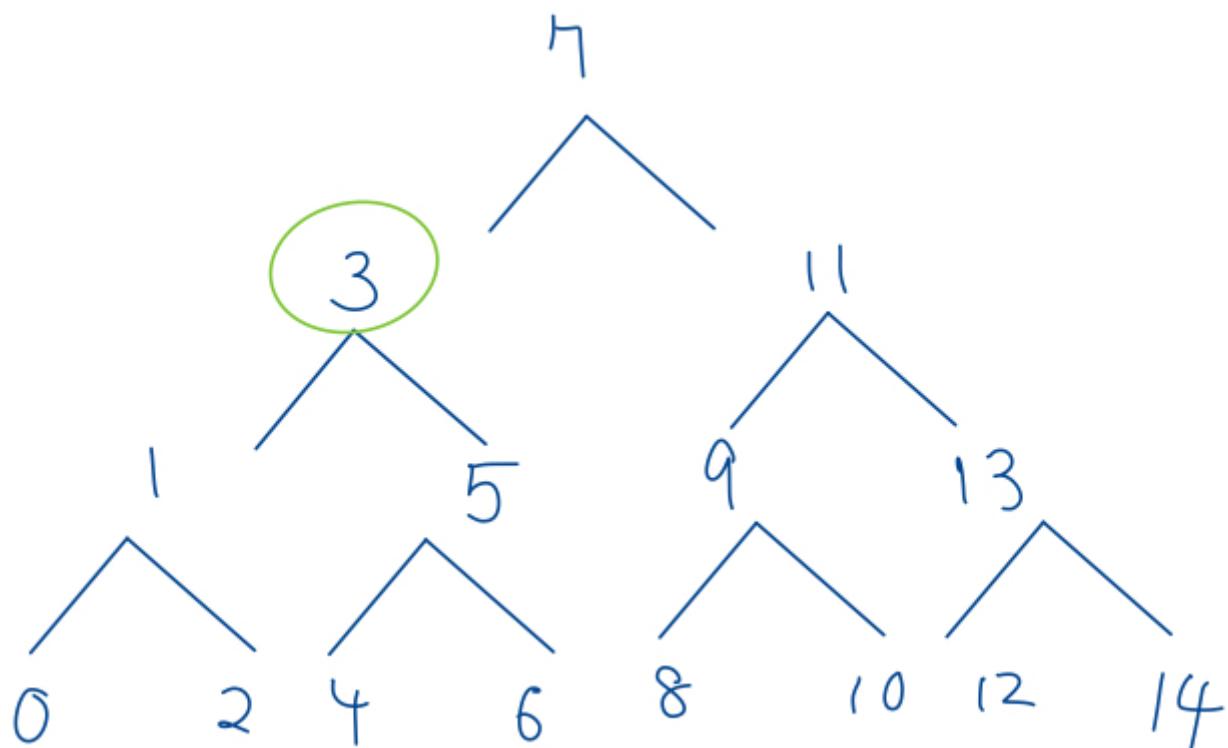
```
In [ ]: from binarytree import bst
```

```
my_bst = bst(height=3, is_perfect=True)  
print(my_bst)
```

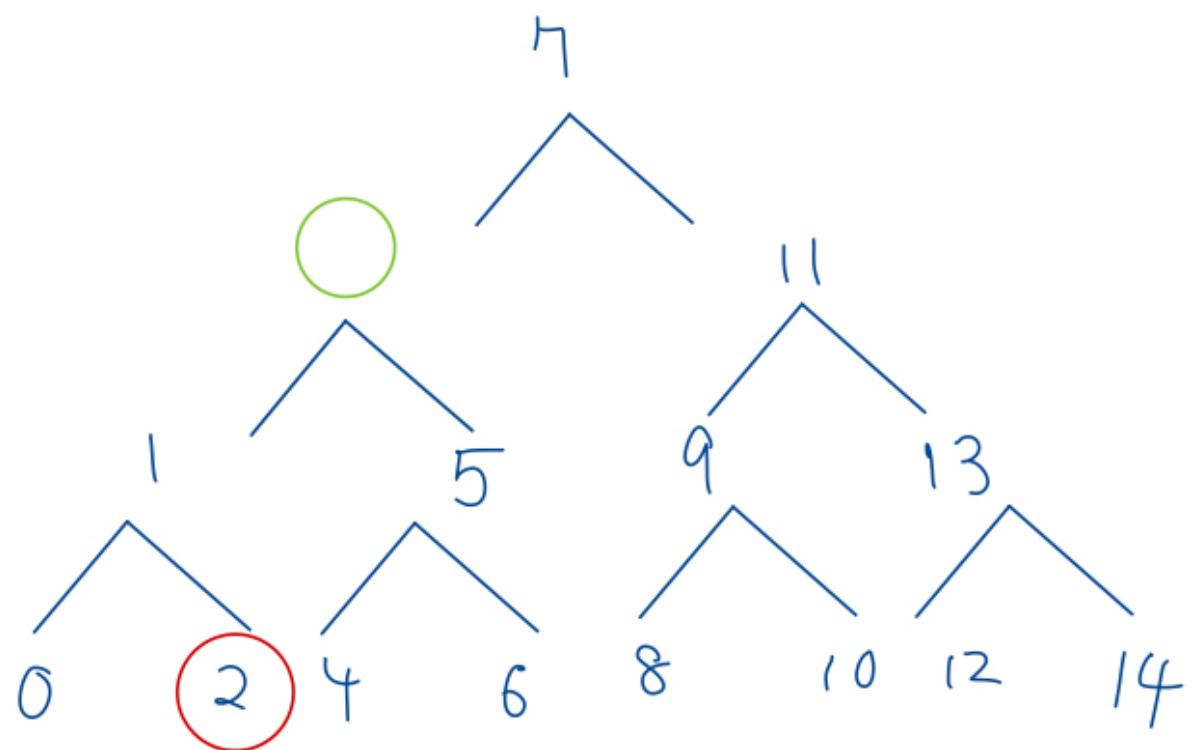


2. 子が2つある頂点を削除する場合、全ての左部分木の頂点の値以上の値を持ち、全ての右部分木の頂点の値以下の値を持った子頂点を、削除した頂点があったところに持って来なくてはならない。このような子頂点は削除されたもとの頂点の左部分木の最大値あるいは右部分木の最小値をもった頂点である。下の図は、前者の工程を示している。

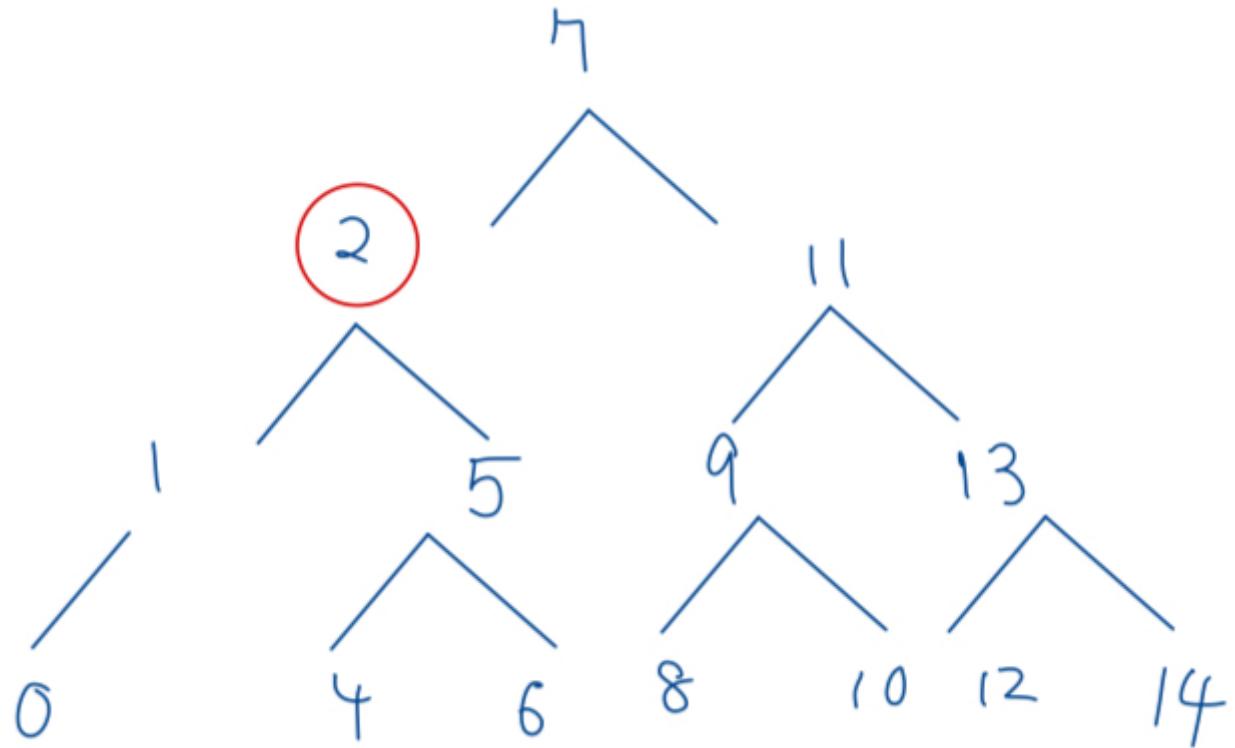
図の二分探索木の3の要素を削除することを考える



まず3を削除し、3の左部分木の最大の要素を見つける



3がもともとあった場所に2を挿入する



3. 要素の削除は、削除する要素を見つけ、また場合によっては代入する要素を見つける操作である。この時、二分探索技で探索できる限界はその木の葉（末端）までなので、この操作は、木の高さを h とすると、高々 $O(h)$ である。この時、各頂点には子供が最大で2なので、おおよそ $2^h = N \implies h = \log N$ である。従って、この操作の最悪計算量は $O(\log N)$ である。

5.

1. 整数の集合{202, 151, 457, 931, 594, 504, 353, 493}が与えられた時、これを基数ソートによりソートする過程を図示せよ。アルゴリズムの簡潔な説明も加えること。
2. 要素数を N 、桁数を L 、各桁で取りうる値の種類数を A とする(1. の例で言えば、 $N = 8, L = 3, A = 10$ となる)。この時の基数ソートの計算量を記せ。

--- 解 ---

1. 基数ソートアルゴリズムは、 n 行の整数の集合を仮定し、1行から n 行まで順に、その桁の数字を参照元にして整数をバケットソートしていくアルゴリズムである。バケットソートは、全ての要素 x が $0 \leq x \leq A$ であるという前提の下、各要素の出現回数を $\text{count}[]$ に記録し、 $0 \dots x \dots A$ の要素を順に $\text{count}[x]$ 個並べていくアルゴリズムである。以下に実装例と、基数ソートにより問題で与えられた整数の集合{202, 151, 457, 931, 594, 504, 353, 493}をソートする基数ソートの過程を表す図をコードのアウトプットとして示す。

```
In [ ]: def countingSortRadix(inputLst, placeValue):
    occurences = [0] * 10 # 各位で整数の上限は9
    n = len(inputLst)

    # 指定された位の数字を抽出する
    for i in range(n):
        placeElement = (inputLst[i] // placeValue) % 10
        occurences[placeElement] += 1

    for i in range(1, 10):
        occurences[i] += occurences[i-1]

    # Reconstructing the output array
    sortedLst = [0] * n
    for i in range(n-1, -1, -1):
        currentEl = inputLst[i]
        placeElement = (inputLst[i] // placeValue) % 10
        occurences[placeElement] -= 1
        newPosition = occurences[placeElement]
        sortedLst[newPosition] = currentEl

    return sortedLst

def radixSort(inputLst, maxDigits):
    sortingProcess = [inputLst]
    sortedLst = inputLst
    for i in range(maxDigits):
        s = ""
        placeVal = 10 ** i
        sortedLst = countingSortRadix(sortedLst, placeVal)
        sortingProcess.append(sortedLst)

    return sortingProcess

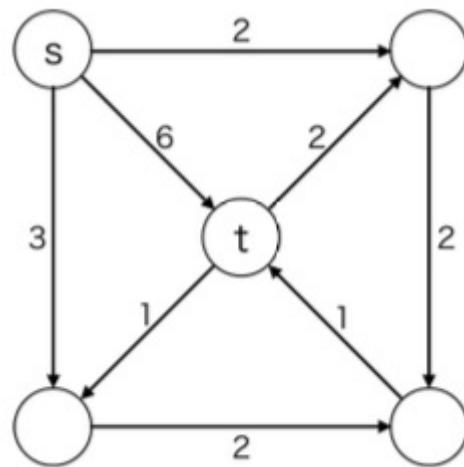
inputLst = [202, 151, 457, 931, 594, 504, 353, 493]
process = radixSort(inputLst, 3)
for i, lst in enumerate(process):
    if not i:
        print(lst, "<-インプットとなるリスト")
    else:
        print(lst, "<-{}の位をもとにソートしたリスト".format(10 ** (i-1)))
```

[202, 151, 457, 931, 594, 504, 353, 493] <-インプットとなるリスト
[151, 931, 202, 353, 493, 594, 504, 457] <-1の位をもとにソートしたリスト
[202, 504, 931, 151, 353, 457, 493, 594] <-10の位をもとにソートしたリスト
[151, 202, 353, 457, 493, 504, 594, 931] <-100の位をもとにソートしたリスト

2. バケットソートの計算量が $O(N + A)$ であり、そのバケットソートを桁数だけ繰り返すので、基数ソートの計算量は $O(L(N + A))$

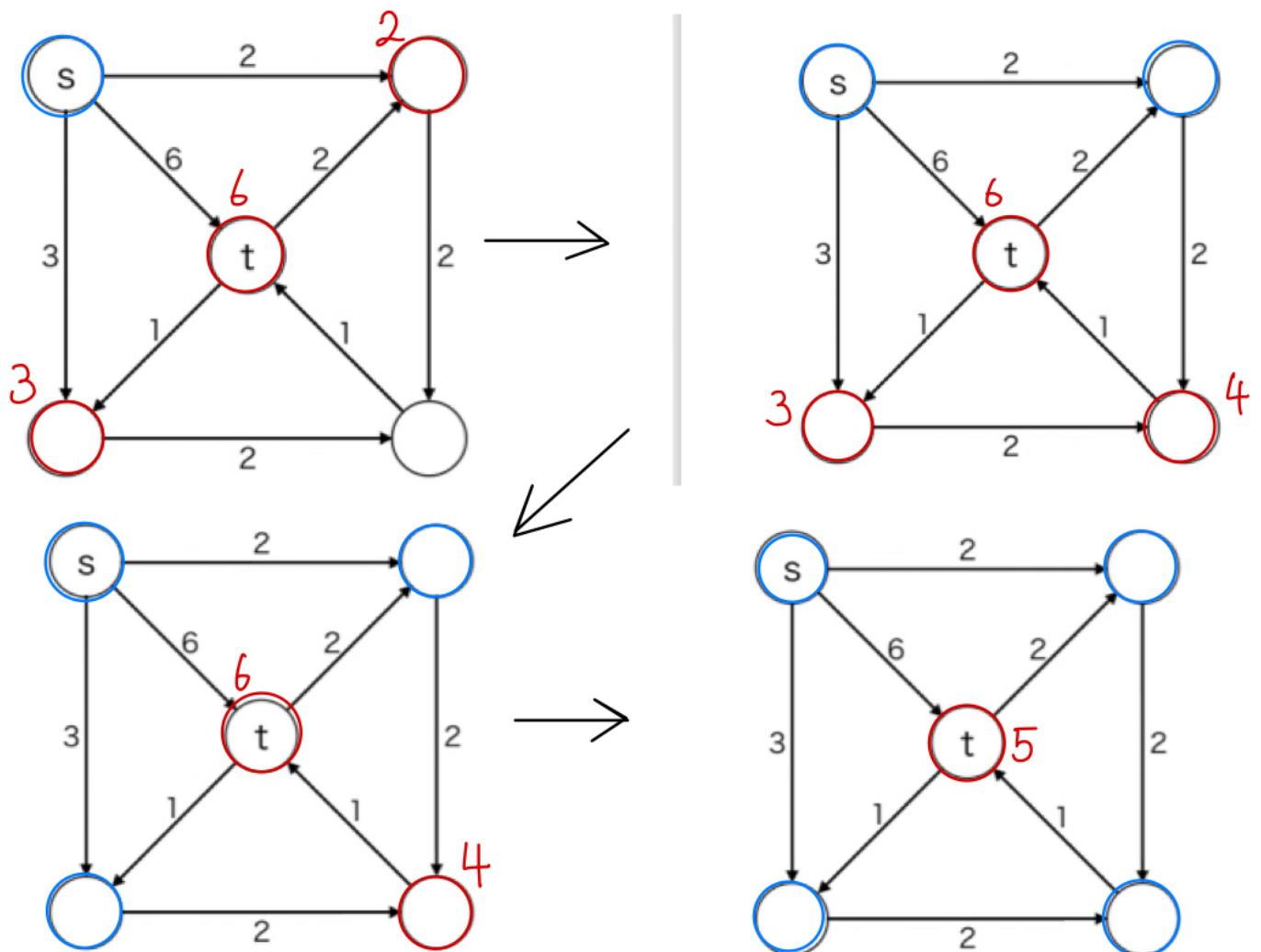
6.

- 下記の有向グラフにおいて、頂点 s から頂点 t への最短路長を求めよ。ただし、アルゴリズムとしてはダイクストラ法を利用し、途中経過を適宜図示しつつアルゴリズムの簡潔な説明を加えること。



- ダイクストラ法の計算量は、その実装によって $O(|E| \log |V|)$ または $O(|V|^2)$ となる。前者が後者より有用である場合とは、入力がどのようなグラフである場合と考えられるか、その理由も含めて簡潔に答えよ。
- ダイクストラ法の計算量を $O(|E| \log |V|)$ とする実装方法の概要を日本語で説明せよ。
- ダイクストラ法を用いて最短路を求めることができない有向グラフとは、どのようなグラフであるか答えよ。

1.



上の図が示すのは、ダイクストラ法で求める最短路長5の導き方である。

ダイクストラ法は、頂点集合全体を V とし、すでに最短路が確定された頂点の集合を S とします。現時点では始点からの経路が判明している頂点のうち、経路が最小なもの v を S に加える。この時、まだ S に入っておらず、 v と接する頂点の経路を、 v への最短距離を利用して計算しなおします。こうして、目的の頂点の最短路長が決定するまで繰り返します。上のグラフでは、青が最短距離が判明した(S に属する)頂点を指し、赤が経路が判明している頂点を指します。

2. $O(|E| \log |V|)$ の実装が $O(|V|^2)$ より有用となるのは、 $|E| = O(V)$ であるとき、つまりグラフが疎である時だ。

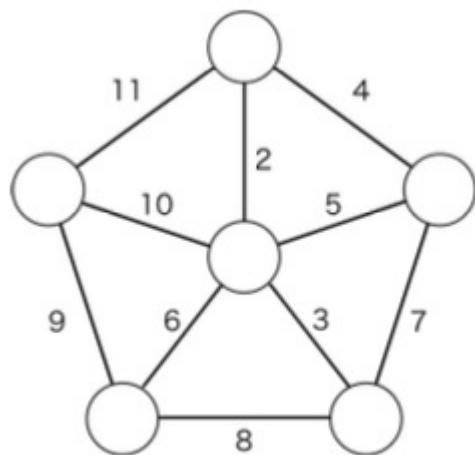
$|E| = O(V)$ だと、 $O(|E| \log |V|) = O(|V| \log |V|) \leq O(|V|^2)$ 。グラフが密で、 $|E| = O(|V|^2)$ の時は、 $O(|E| \log |V|) = O(|V|^2 \log |V|) \geq O(|V|^2)$ となり、後者の方が前者より有用になってしまう。

3. 単純なダイクストラ法では、現時点では始点からの経路が判明している頂点のなかで始点からの経路が最小になるものを線形探索法で求めていたが、 $O(|E| \log |V|)$ の実装では、線形探索法の代わりに二分ヒープを使うことで最小値を $O(1)$ で出せる。また、ヒープを整える操作の計算量は $O(\log |V|)$ であるので、線形の場合の $O(|V|)$ を下回る。

4. 経路のウェイトに負の値を含む有向グラフでは、ダイクストラ法の正当性が担保されず、使用できない。

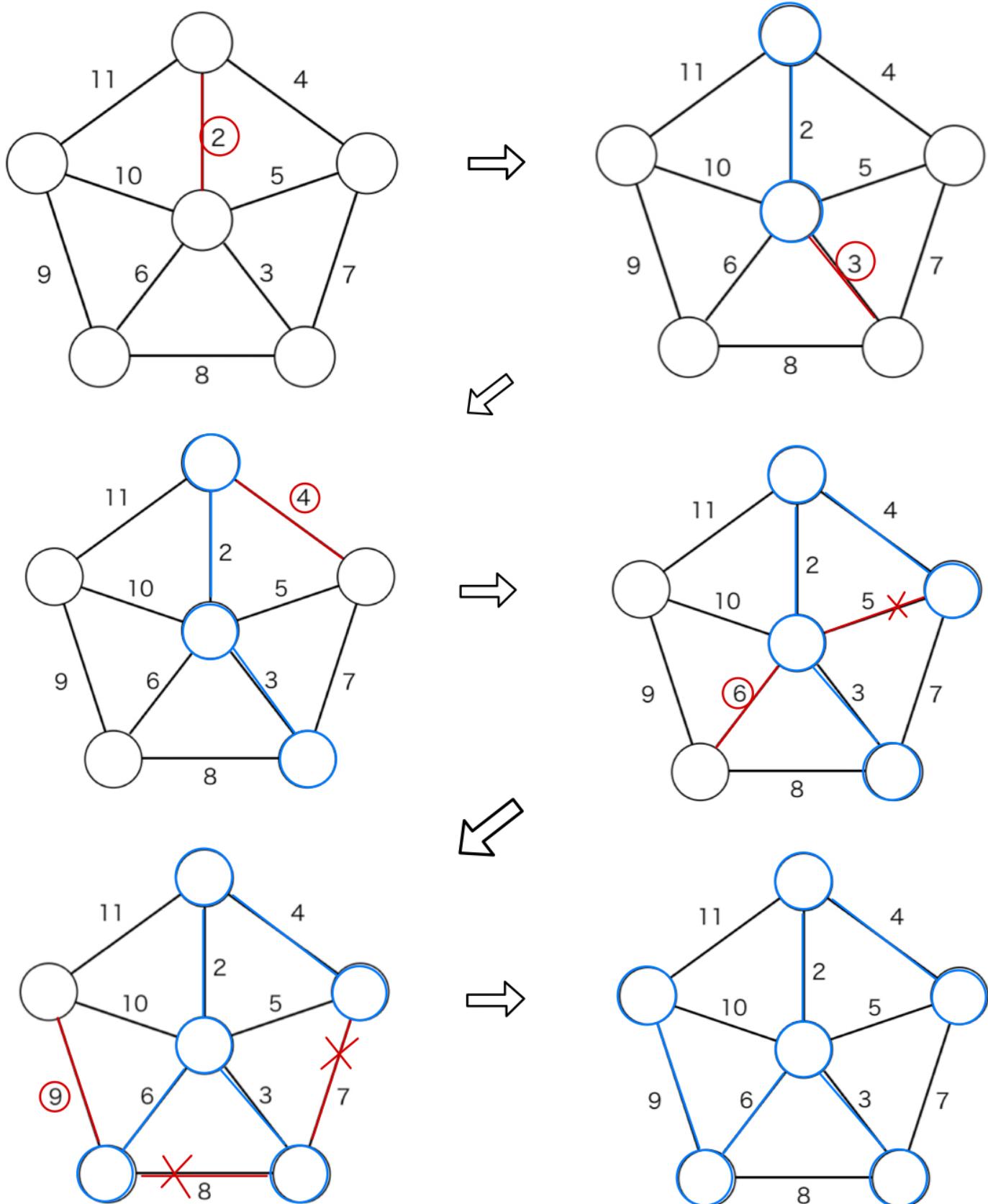
7.

1. 最小全域木とは何か、簡潔に説明せよ。
2. 下記の無向グラフの最小全域木をクラスカル法によって求めた結果を図示せよ。途中経過も適宜図示しつつ、アルゴリズムの簡潔な説明を加えること。



--- 解 ---

1. 最小全域木は、重み付き無向グラフ $G = (V, E)$ の部分グラフであって木であり、 G の全ての頂点を含みながらその辺の重みの総和が最小なものをいう。
2. クラスカル法とは、単純な貪欲法です。辺の重みが小さいものから順に木に追加していき、辺がサイクルを形成してしまう場合は、その辺は破棄して次に進みます。全ての頂点が含まれたら、終了します。



8.

1. 「局所最適解」と「大域最適解」の言葉の意味を、2つの言葉の関係性がわかるように簡潔に説明せよ。
2. 「クラス P」と「クラス NP」の言葉の意味を、2つの言葉の関係性がわかるように簡潔に説明せよ。

--- 解 ---

1. 局所最適解は、ある近傍内で最もスコアの良い解であるのに対し、大域最適解とは全ての領域の中のあらゆる解の中でも最もスコアの良い解を言う。大域最適解は常にそれを含む近傍の局所最適解である裏腹、局所最適解は大域最適解であるとは限らない。
 2. クラスPの問題とは、多項式時間アルゴリズムで解ける判定問題の集合である。対して、クラスNPの問題とは、解が与えられれば、その解の検証を多項式時間でできる判定問題の集合である。クラスP問題は必然的にクラスNPでもあるが、逆の包含関係が成立しているかはいまだにわかっていない。
-

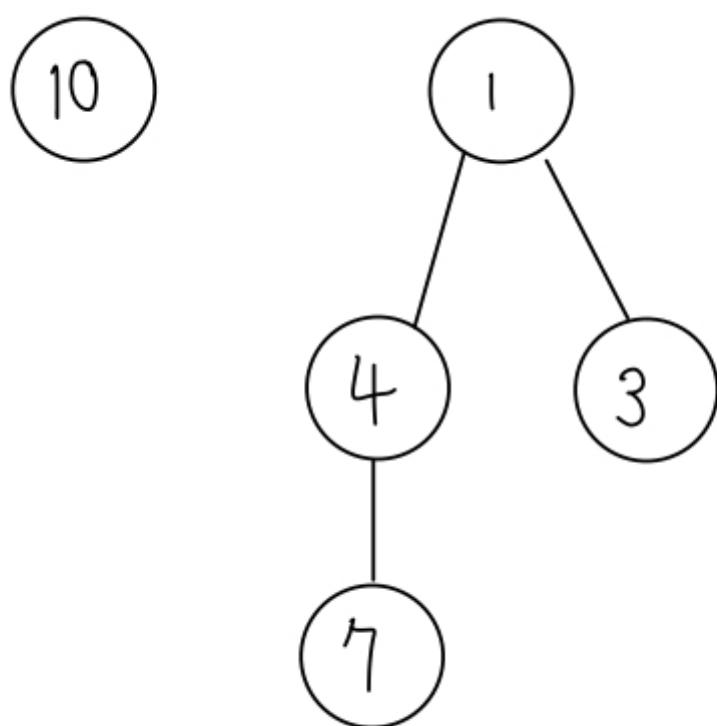
9.

二項ヒープについて、以下の問いに答えよ。

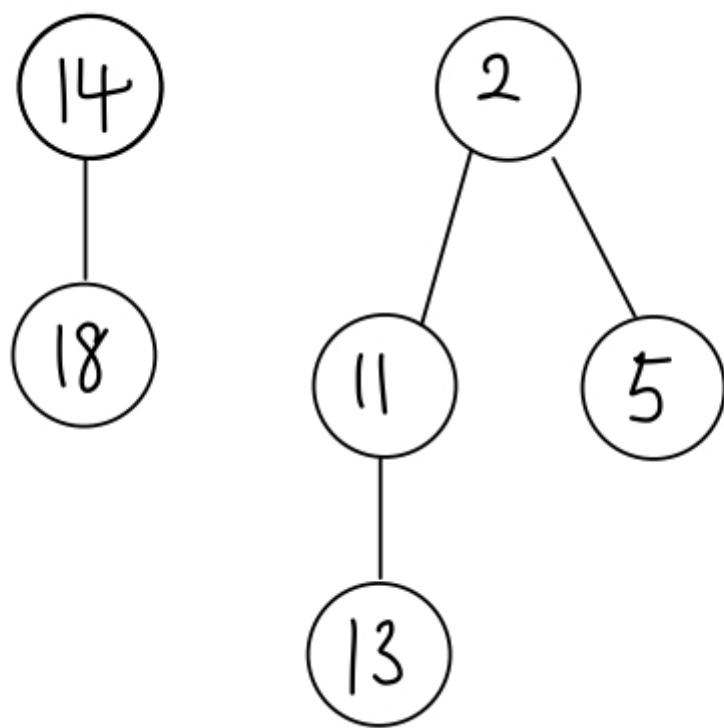
1. 整数の集合{1, 3, 4, 7, 10}を二項ヒープに格納した場合、どのような構造となるか。結果を図示せよ。
2. 整数の集合{2, 5, 11, 13, 14, 18}を二項ヒープに格納した場合、どのような構造となるか。結果を図示せよ。
3. 1.で作成した二項ヒープと2.で作成した二項ヒープの併合処理を行った場合、どのような構造の二項ヒープが得られるか。結果を図示せよ。
4. 要素数 N の集合を格納した二項ヒープと、要素数 M の集合を格納した二項ヒープの併合処理を行う。この処理の計算量を記せ。

--- 解 ---

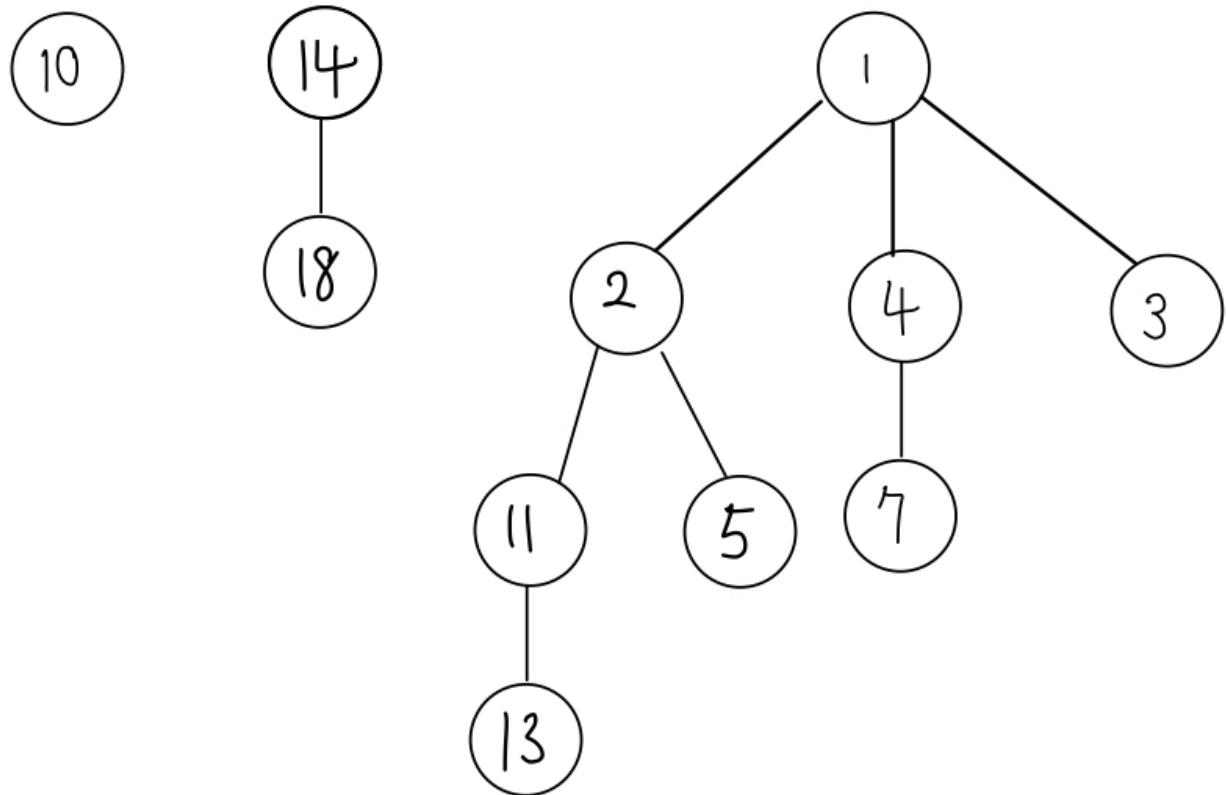
1.



2.



3.



4. $O(\log(M + N))$