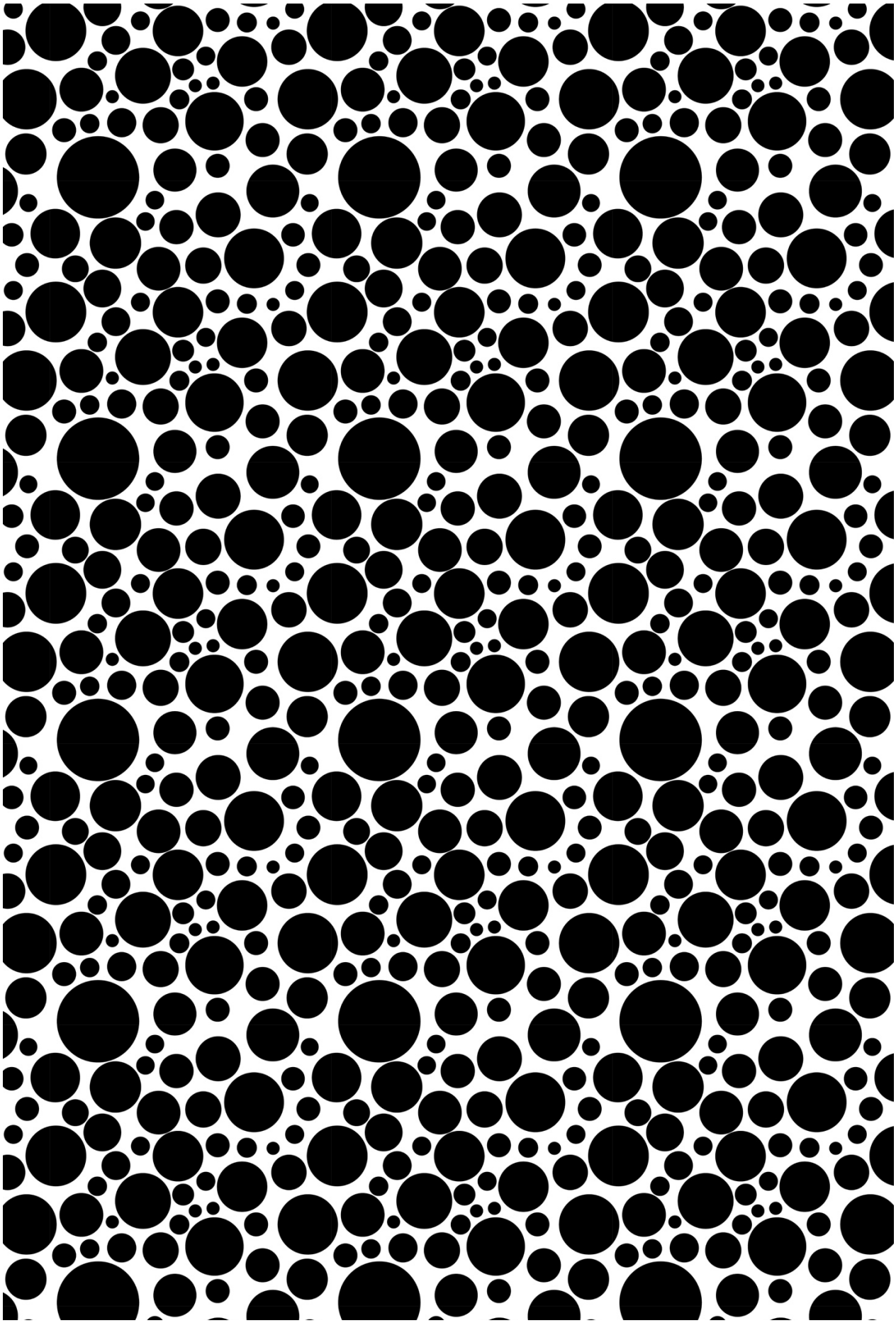
この過去問題は学習の参考にするために提供するものです．問題の形式や大問の数は変更されることがあります．授業中のアナウンスをよく確認し準備をしてください．

2021 年度　アルゴリズム　期末試験

2021 年 7 月 21 日（水）　試験時間：70 分

氏名：　　学籍番号：

- **本冊子上部に，学籍番号，名前を間違いなく明瞭に記入すること．**
- **試験中はマスクを着用すること．**
- 試験開始の合図があるまで，本冊子を開いてはならない．試験開始前に本ページに記載されている注意事項をよく読み，理解すること．
- 本冊子には解答欄が含まれており，指定された部分に解答を記すこと．解答はどのような順番で行ってもよいが，指定された場所に記入すること．誤った場所に書かれた解答は採点されない．
- 本冊子は切り離してはならない．切り離した部分の解答は無効となる．
- 本冊子の最後に余白（2 ページ）がある．この余白は解答のために自由に使ってもよいが，この余白に書かれたことは採点されない．
- 自然言語，擬似コードによる解答では日本語，もしくは英語を使用すること．
- python による解答では必要に応じてコメントを添えること．
- 採点者が誤解なく読むことができるように，解答は丁寧に記載すること．明瞭・丁寧に記載されていない解答は採点されない．
- 教科書，ノート，電子機器，スマートデバイス等の持ち込みは許可されない．
- 電子機器やスマートデバイスは体から取り外し電源を切り，カバン等にしまっておくこと．
- 試験の録画，録音を行ってはならない．
- 監督者の指示に従うこと．
- 監督者に申し出ることがあれば，挙手をして監督者に知らせ，監督者の指示を仰ぐこと．
- 落丁，乱丁，印刷不鮮明があれば，監督者に申し出ること．
- **不正行為が発覚した場合，本試験の採点は行わず，現在までの課題提出状況に関わらず，成績を不可とする．**特に悪質なものはさらなる処分の対象となり得る．特に以下のような行為は発見し次第，即座に不正行為とみなす．
    - 持ち込みが許可されている筆記用具，時計以外の電子機器，参考書，補助ツール等を使用した．
    - 他者と連絡・やり取りをした．
    - 監督者の指示に従わない．
    - 他の受験生の受験を阻害する行為を意図的に行った．
- **内容に関する質問には答えない．問題の指示が曖昧だと考えられる場合，適切な仮定・解釈を自身で設定し，その仮定・解釈を解答内で簡潔に説明すること．**
- オンラインで受験をする者は，別途指示された方法により受験環境を準備すること．必要な準備ができていない場合，受験を認めない．



大問 1

以下の各問の記述が正しいかどうかを答えてください．正しい記述であれば解答欄に「**正**」と書いてください．誤った記述であればどのような誤りかを簡潔に自然言語で説明してください．

問 1-A. 要素数nの二分探索木では，探索にかかる時間計算量を常に$O(\log n)$にできる．
[解答欄]

問 1-B. DFS はキューを使う実装が一般的である．
[解答欄]

問 1-C. ノードの数がnである完全二分木において，ある 1 つのノードを探索する事を考える．根ノードから DFS により探索する場合，平均的な時間計算量は$O(\log n)$である．
[解答欄]

問 1-D. n種類の値を$k(k < n)$種類のハッシュ値に変換する関数を考える時，その関数を適切に設計することで，ハッシュの衝突を避けることができる．
[解答欄]

問 1-E. 挿入ソートはクイックソートよりも，常に時間計算量が大きい．
[解答欄]

問 1-F. 与えられた配列の長さがnである時, マージソートの時間計算量は, 最悪の場合$O(n \log n)$である.
[解答欄]

問 1-G. 与えられた配列の長さがnである時, ヒープソートの空間計算量は, 常に$O(n \log n)$である.
[解答欄]

問 1-H. 照合対象の文字列の長さがn, パターンの文字列の長さがlである時, BM 法の時間計算量は最悪の場合, $O(nl)$である.
[解答欄]

問 1-I. 照合対象の文字列の長さがn, パターンの文字列の長さがlである時, ローリングハッシュを利用した文字列照合（ラビン・カープ法）では, 照合対象の部分文字列のハッシュを逐次計算する. この時, ハッシュを新しく計算するためには, 平均的には$O(l)$要する.
[解答欄]

問 1-J. 要素の数がnの部分和問題は常に多項式時間のオーダーで解くことができることが知られている.
[解答欄]

問 1-K. ノード，辺の総数がそれぞれ$|V|, |E|$である連結グラフが与えられた時，それが有向非巡回グラフであるかを確かめるためには，DFS を用いることで実現でき，その時間計算量は平均的な場合において$O(|V|)$である．
[解答欄]

問 1-L. 連結グラフにおける単一始点最短経路問題は BFS で解くことができる場合がある．
[解答欄]

問 1-M. プリム法は，素集合データ構造（Disjoint Sets）を利用して実装されることが多い．
[解答欄]

問 1-N. エラトステネスの篩の平均的な時間計算量は，与えられる正の整数nに対して$O(n\log n)$である．
[解答欄]

次ページに続く．

大問 2

ある整数の配列 seq（大きさ：N）が与えられた時，その連続する部分列の和の最大値を求める関数 `SubMax(seq)`を python で書いてください．ただし，以下の制約条件を満たすようなプログラムにしてください．この問題では擬似コードでの解答は認められません．

```
def SubMax(seq):

    #この関数の実装を答える

    return [連続する部分列の和の最大値]
```

制約条件

- $1 \leq N \leq 10^7$
- $-10^7 \leq seq[i] \leq 10^7$
- 平均的な場合の時間計算量が，$O(N^2)$より小さい．
  - 時間計算量が$O(N^2)$以上となるプログラムには部分点を与えない．
- 平均的な場合の空間計算量が，$O(1)$.
  - ただし，空間計算量が$O(N)$となるプログラムでも，部分点を与える．

入力例 1

```
SubMax([1,2,3])
```

出力結果

```
6
```

すべての要素を足し合わせたものが最大となります．

入力例 2

```
SubMax([1,2,-4,5])
```

出力結果

```
5
```

最後の要素のみ使ったものが最大です．要素は負の整数になることもあることに注意してください．

入力例 3

```
SubMax([-9,10,-3,-7,15,8,-2,4,-8])
```

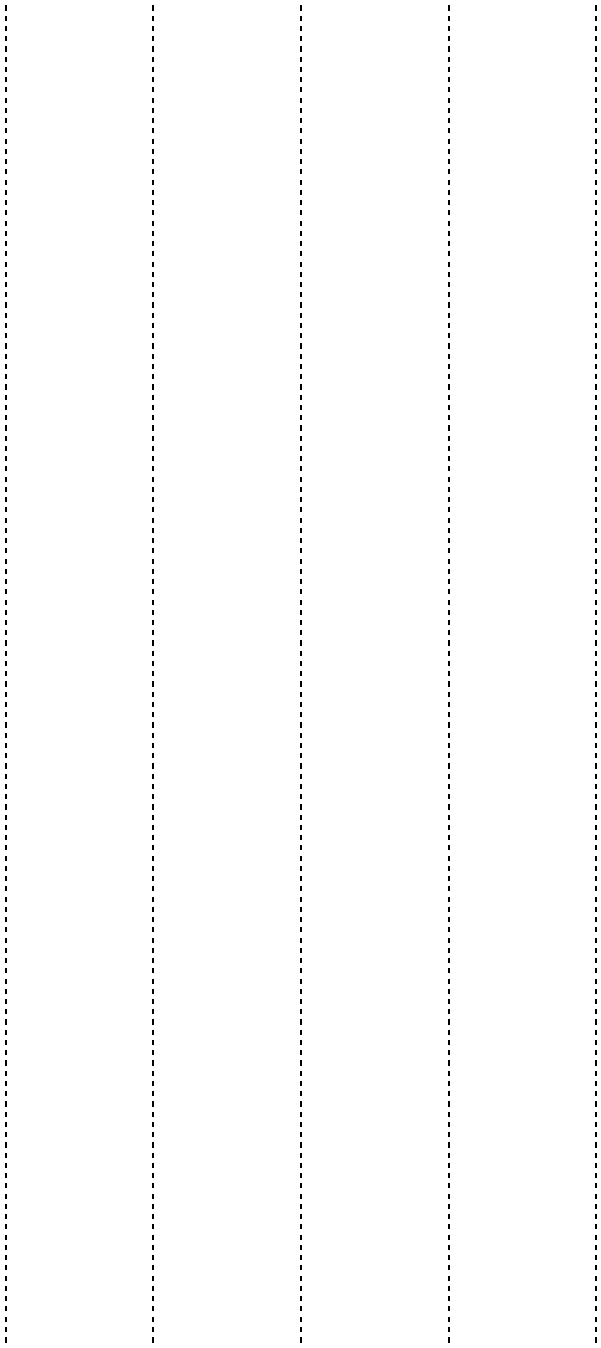
出力結果

```
25
```

[15, 8, -2, 4]の和が最大となります.

[解答欄]　インデントする場合，補助線に揃えること．必要に応じてコメントを添えること．

```
def SubMax(seq):
```



大問 3

皆さんはエンジニアとして，2 つの整数xとnが与えられた時，$k \leq \sqrt[n]{x}$となる最大の整数kを求めるプログラムを書くように指示されました．ただし，このプログラムは非常に特別な環境で実行されるため，以下に与えられる制約条件を全て満たす必要があります．これらを満たすプログラムを python で書いてください．この問題では擬似コードでの解答は認められません．

また，このプログラムの平均的な時間計算量を，その計算量になる理由とともに簡潔に答えてください．

```
def PowerRoot(x,n):

    #この関数の実装を答える

    return [k ≤ ∛xとなる最大の整数k]
```

制約条件

- $1 \leq x \leq 10^9$
- $2 \leq n \leq 10$
- このプログラムの実行環境はとても遅く，1 秒間に10^4回の基本演算・処理しか実行できない．
- どんなx，nに対しても 1 秒以内で結果を返さないといけない．ただし，関数呼び出し等のオーバーヘッドは無視できるものとする．
- 組み込みの `a**b` や `pow(a, b)`は利用できない．
- math ライブラリは使用できない．
- その他の組み込みの関数や他のライブラリも使用できない．

入力例 1

```
PowerRoot(65,3)
```

出力結果

```
4
```

$4 * 4 * 4 = 64$で65に最も近い値となります．

入力例 2

```
PowerRoot(8,3)
```

出力結果

```
2
```

$2 * 2 * 2 = 8$で，$\sqrt[n]{x} = k$となる整数kが存在する場合もあります．

入力例 3

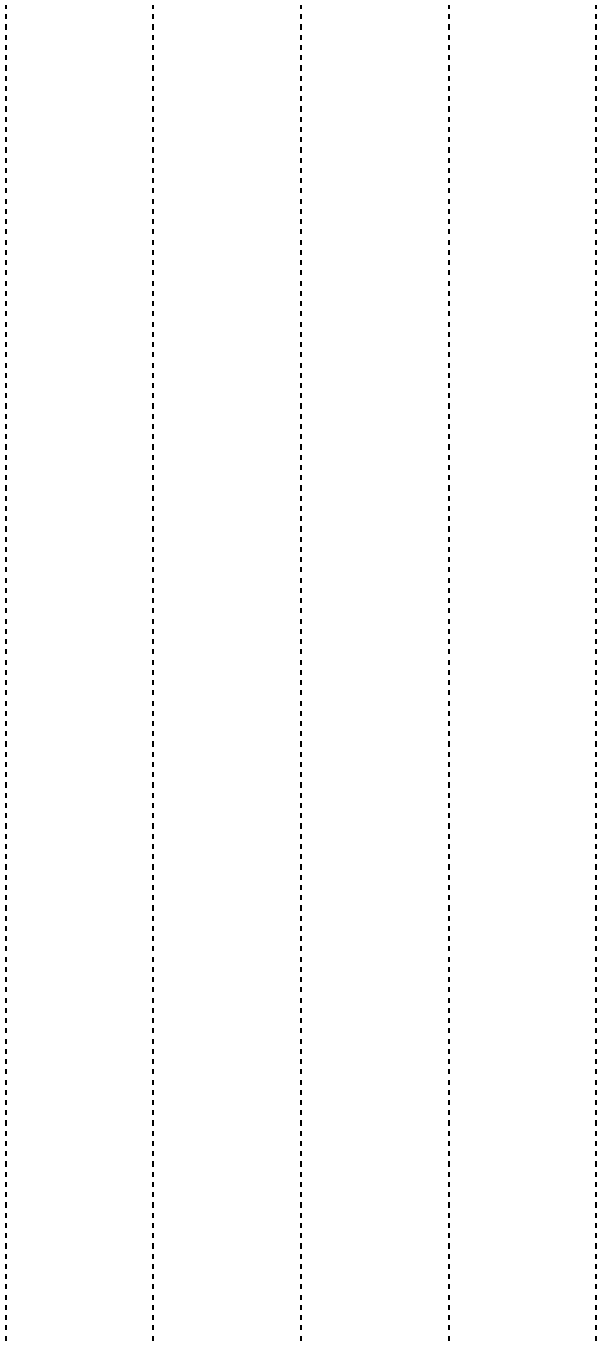
```
PowerRoot(100,4)
```

出力結果

```
3
```

[解答欄]　インデントする場合，補助線に揃えること．必要に応じてコメントを添えること．

```
def PowerRoot(x,n):
```



計算量とその理由：

大問 4

以下のグラフに関する問題に答えてください．

問 4-A. グラフを表すデータ構造として，隣接リストと隣接行列があります．以下の記述に関して，隣接リストのみに当てはまる，隣接行列のみに当てはまる，両方に当てはまる，どちらにも当てはまらない，のうち正しいものを 1 つ選び，丸をつけてください．ただし，与えられるグラフのノードの数を$|V|$，辺の数を$|E|$，１つのノードに接続している辺の平均の数を$\bar{e}$とします．

4-A-1：空間計算量は常に$O(|E|)$となる．

隣接リストのみに当てはまる　　隣接行列のみに当てはまる
両方に当てはまる　　どちらにも当てはまらない

4-A-2：空間計算量は常に$O(|V|)$となる．

隣接リストのみに当てはまる　　隣接行列のみに当てはまる
両方に当てはまる　　どちらにも当てはまらない

4-A-3：完全連結グラフの場合，空間計算量は$O(|V|^2)$となる．

隣接リストのみに当てはまる　　隣接行列のみに当てはまる
両方に当てはまる　　どちらにも当てはまらない

4-A-4：ある 1 つの辺を追加する時，時間計算量は平均的な場合$O(|E|)$となる．

隣接リストのみに当てはまる　　隣接行列のみに当てはまる
両方に当てはまる　　どちらにも当てはまらない

4-A-5：ある 1 つのノードの隣接ノードを全て取得する時，時間計算量は平均的な場合$O(|V|)$となる．

隣接リストのみに当てはまる　　隣接行列のみに当てはまる
両方に当てはまる　　どちらにも当てはまらない

4-A-6：ある 1 つのノードの隣接ノードを全て取得する時，時間計算量は平均的な場合$O(\bar{e})$となる．

隣接リストのみに当てはまる　　隣接行列のみに当てはまる
両方に当てはまる　　どちらにも当てはまらない

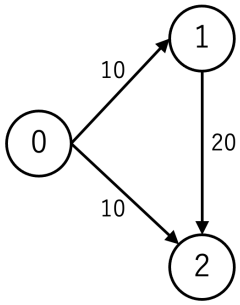
4-A-7：グラフ内の辺をすべて取得する時，時間計算量は常に$O(|E|^2)$となる．

隣接リストのみに当てはまる　　隣接行列のみに当てはまる
両方に当てはまる　　どちらにも当てはまらない

問 4-B. ある無向グラフが与えられた時，python の多次元リストで表現することを考えます．

`[..., [`ノード `i`,ノード `j`,`i` と `j` の間のコスト`], ...]`

ただし，ノード i，ノード j の順で昇順に並んでいるものとします．
例えば，右の有向グラフであれば，

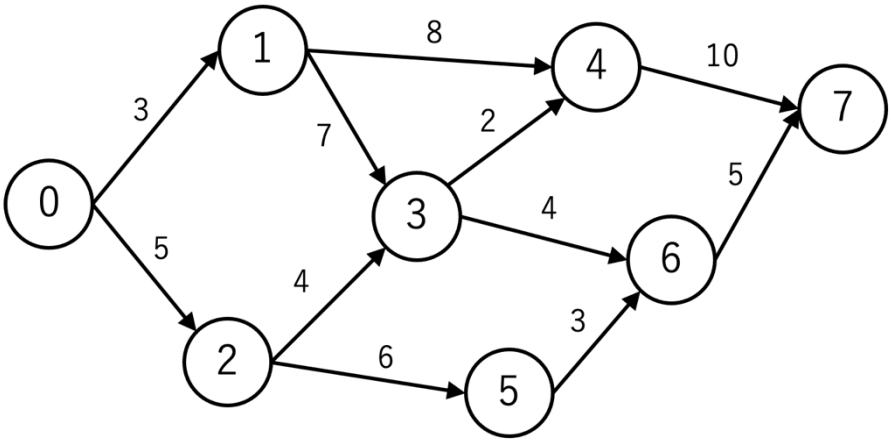



```
[[0,1,10],[0,2,10],[1,2,20]]
```

と表すことになり，

```
[[0,2,10],[0,1,10],[1,2,20]]
[[1,2,20],[0,1,10],[0,2,10]]
```

などは誤った表現として扱います．この時，上記データ構造を用いて以下のグラフを表現してください．



[解答欄]

問 4-C. 問 4-B で表現したグラフにおいて，ノード 0 から各ノードまでの最短距離を，以下のようなリストの形式で答えてください．

`[0, [`ノード `1` までの最短距離`], [`ノード 2 までの最短距離`], ...]`

[解答欄]

問 4-D. コージさんはノード 0 から各ノードまでの最短距離を求めるため，授業で学んだことを思い出し，ベルマンフォード法を実装してみました．コージさんのコードは以下のとおりです．このコードが正しい場合，ベルマンフォード法がどのような処理をして最短の距離を導出するかを自然言語で説明してください．このコードに誤りがある場合，どのように修正すべきかを python，擬似コード，あるいは自然言語（あるいはそれらの組み合わせ）で説明してください．

制約条件

- $2 \leq |V| \leq 10^6$
- 与えられるグラフは 1 つの連結グラフである．
- $c_{i,j}$をノードiからノードjに向かう辺のコストとし，$-10^4 \leq c_{i,j} \leq 10^4$.

```python
# V: ノードの総数（int）
# graph: グラフを表す多次元配列（問4-B と同じもの）
def BellmanFord(V, graph):
    dist = [float('inf')]*V
    dist[0] = 0; flag = True

    while flag:
        flag = False
        for e in graph:
            if dist[e[1]] > e[2] + dist[e[0]]:
                dist[e[1]] = e[2] + dist[e[0]]
                flag = True

    return dist
```

[解答欄]

問 4-E. コージさんはさらに授業で学んだことを振り返り，SPFA（Shortest Path Faster Algorithm）のことを思い出しました. SPFA はベルマン・フォード法を改良したアルゴリズムと理解することができます. SPFA がどのようなアルゴリズムか，ベルマン・フォード法との違いを明確にしながら，自然言語で説明してください.

[解答欄]

問 4-F. SPFA の計算量に関して，知っていることを述べてください.

[解答欄]

大問 5

Koji さんはアナグラムを見つけるのが大好きです．最初は自力でアナグラムを探していましたが，長い文章ではアナグラムを探すのがとても大変なことに気が付き，エンジニアでもある Koji さんはプログラムを書いて処理をすることにしました．具体的には，

> 検索対象となる文字列を text，パターンの文字列を pattern とし，それぞれの長さをL，Pとする．text と pattern の 2 つを引数にとり，text 内に含まれる pattern のアナグラムの総数を返す関数を実装する

ことにしました．Koji さんが試したいくつかのアルゴリズムに関する以下の問に答えてください．

制約条件

- $1 \leq P \leq L \leq 10^8$
- text と pattern を構成する文字列はアルファベットの小文字（a から z の 26 種類）のみ．
- 長さnの文字列 2 つが完全に照合するかどうかを確かめるのには$O(n)$かかるとする．

入力例 1

```
text='kojihajoki'
pattern='koji'
```

出力結果

```
2
```

koji と joki が‘koji’のアナグラムになるので，出力は 2 となります．完全に同一の文字列もアナグラムとみなします．

入力例 2

```
text='abcdefg'
pattern='koji'
```

出力結果

```
0
```

アナグラムが存在しない場合は 0 を返します．

問 5-A. まず最も単純な方法として，**pattern 内に同じ文字が現れない場合に限定することにし**，pattern 内の文字をすべて使って作ることのできる文字列を全部列挙し，それを text の部分文字列と照合する方法を考えました．すなわち，以下のような擬似コードで表現されるプログラムを書きました．

```
def anagram1(text, pattern):
    all_p = [pattern に含まれる文字をすべて使って構成される文字列を全て列挙したもの]
    count = 0
    for [text の先頭から最後まで]:
        [text の現在の位置から pattern と同じ長さの文字列を切り出す]
        [all_p の全てと照合し，完全照合したものがあれば count に 1 足す]

    return count
```

このアルゴリズムを実行したところ，文字列が長くなるとなかなか終了しないことがわかりました．このアルゴリズムの平均時間計算量を，L，Pを用いて表してください．また，この時間計算量になる理由を簡潔に自然言語で説明してください．

[解答欄]

問 5-B. Koji さんはより効率的で，pattern 内に同じ文字が現れても大丈夫な方法を考えることにしました．アルファベットは順序関係が定義できる（例えば，‘a’ < ‘b’ < ‘c’と定義できる）ので，比較に基づくソートアルゴリズムを利用することを思いつきました．これにより，pattern と text の部分文字列をアルファベット順に並べ替えてから照合するコードを書きました．このアルゴリズムの平均時間計算量を，L，Pを用いて表してください．また，この時間計算量になる理由を簡潔に自然言語で説明してください．

[解答欄]

問 5-C. Koji さんはさらなる改良を思いつき，$1 \leq P << L$の時に平均時間計算量が$O(LP)$程度になるコードを書くことができました．このコードがどのようなものか，python，あるいは擬似コードで説明してください．

[解答欄] インデントする場合，補助線に揃えること．必要に応じてコメントを添えること．

```
def anagram3(text, pattern):
```

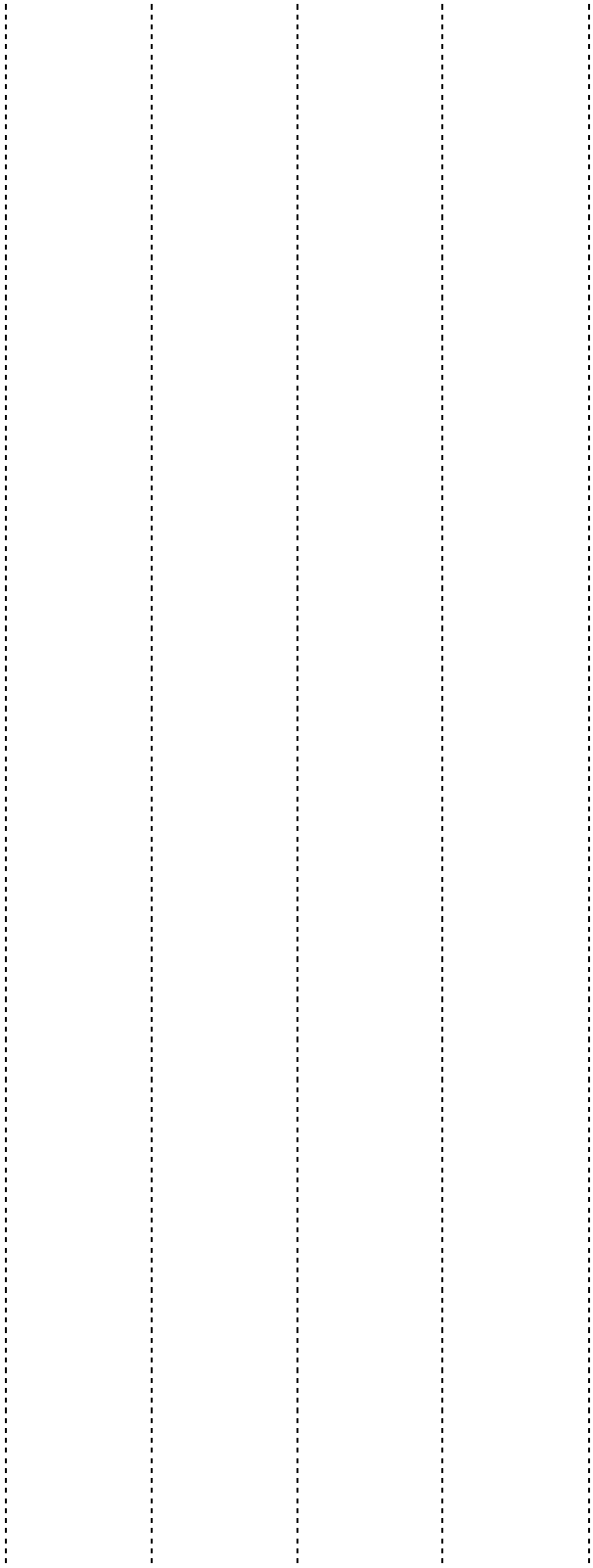
問 5-D. Koji さんは今までに作ったアルゴリズムをいろんなテストケース試していると，パターンの文字列がとても長い場合，特に$1 << P << L$となる時に，計算にとても時間がかかることがわかりました．このような場合でも，今までに作ったアルゴリズムよりも高速に動くと期待できるアルゴリズムを作ることはできるでしょうか？できるならば，そのコードを python，あるいは擬似コードで説明してください．できないならば，その理由を自然言語で説明してください．

[これ以上改良できない場合の解答欄]

（改良できる場合の解答欄は次のページにあります．）

[改良できる場合の解答欄]　インデントする場合，補助線に揃えること．必要に応じてコメントを添えること．

```
def anagram4(text, pattern):
```



以上

（余白：解答の下書きのために使って良い．）

（余白：解答の下書きのために使って良い．）