1. příklad

Uvažme "balancovanou"frontu, která má kapacitu k (k je sudé číslo):

- *enqueue*(x) vloží prvek x na konec fronty a má konstantní složitost
- *dequque* odstraní prvek z konce fronty a má konstantní složitost
- *size*() vrací velikost fronty (počet prvků) a má konstantní složitost
- fronta je inicializována náhodnými prvky na velikost $k/2$

Uvažme operace *my_enquque*(x) a *my_dequque*(x) implementované níže.

```
global(int k)
Function my_enqueue(x)
    if size() = k then
        while size() ≥ k/2 do
            dequeue()
    enquque(x)
Function my_dequeue()
    if size() = 0 then
        while size() < k/2 do
            enquque(size())
    else dequeue()
```

Uvažte libovolnou posloupnost n operací *my_enqueue*(x) a *my_dequeue*().

1. Jaká je asymptotická složitost této posloupnosti operací při použití analýzy nejhoršího případu?
2. Jaká je amortizovaná složitost této posloupnosti operací?

Poznámka: Předpokládejte uniformní cenové kritérium, kde složitost každého řádku programu je 1.

2. příklad

Uvažte operaci $inc(< 0..9 > x[\,], int\ size)$, která inkrementuje číslo zapsané pomocí vektoru dekadických čísel a je implementovaná níže. x je pole indexované od 0 do $size - 1$.

```
1 Function inc(< 0,...,9 > x[ ], int size)
2     i := size
3     do
4         i := i − 1
5         x[i] := x[i] + 1 // assume that 9+1 = 0
6     while x[i] = 0 ∧ i > 0
```

Uvažte posloupnost n volání operace $inc(< 0, \dots, 9 > x[\,], int\ size)$ počínaje vektorem $x = [0 \dots 0]$.

1. Jaká je asymptotická složitost této posloupnosti operací při použití analýzy nejhoršího případu?
2. Jaká je amortizovaná složitost této posloupnosti operací?

Poznámka: Předpokládejte uniformní cenové kritérium, kde složitost každého řádku programu je 1 (řádek 3 má složitost 0).

3. příklad (problém obarvení grafu)

Mějme neorientovaný graf $G = (V, E)$, kde $E \subseteq \{\{u, v\} \mid u, v \in V\}$. Ptáme se, zda existuje obarvení uzlů grafu c barvami tak, aby žádné dva sousedící uzly neměly stejnou barvu.

Uvažme tento "brute-force" algoritmus, který využívá operaci *inc* z minulého příkladu. Předkládáme, že $V = \{0, 1, \ldots, |V| - 1\}$.

```
1  Function is_colorable(V, E, c)
2      < 0, ..., c − 1 > color[0, ..., |V| − 1] = [0, ..., 0]
3      while true do
4          res := true
5          foreach {u, v} ∈ E do
6              if color[u] = color[v] then
7                  res := false
8                  break
9          if res = true then
10             return YES
11         if color = [c − 1, ..., c − 1] then
12             return NO
13         else inc(color, |V|)
```

1. Analyzujte asymptotickou složitost algoritmu *is_colorable* v nelepším případě.
2. Analyzujte asymptotickou složitost algoritmu *is_colorable* v nejhorším případě.

Poznámka: Předpokládejte uniformní cenové kritérium, kde složitost každého řádku programu je 1 (včetně řádku 13).

4. příklad (2 obarvitelnost souvislých grafů)

Mějme souvislý neorientovaný graf $G = (V, E)$, kde $E \subseteq \{\{u, v\} \mid u, v \in V\}$. Ptáme se, zda existuje obarvení uzlů grafu dvěma barvami tak, aby žádné dva sousedící uzly neměly stejnou barvu.

"Brute-force"algoritmu z minulého příkladu by vedl na exponenciální složitost. Pro dvě obarvitelnost, ale existuje lepší algoritmus.

4. příklad (2 obarvitelnost souvislých grafů)

Mějme souvislý graf $G = (V, E)$, kde $E \subseteq \{\{u, v\} \mid u, v \in V\}$. Ptáme se, zda existuje obarvení uzlů grafu dvěma barvami tak, aby žádné dva sousedící uzly neměly stejnou barvu.

"Brute-force"algoritmu z minulého příkladu by vedl na exponenciální složitost. Pro dvě obarvitelnost, ale existuje lepší algoritmus.

```
Function is_2colorable(V, E, c)
    <-1, 0, 1> color[0, ..., |V| - 1] = [-1, ..., -1] // -1 uncolored
    color[0] = 0
    stack < int > toProcess
    toProcess.push(0)
    while toProcess.nonempty() do
        u := toProcess.pop()
        foreach v ∈ Adj(u) do
            if color[v] = -1 then
                color[v] := 1 - color[u]
                toProcess.push(v)
            else
                if color[v] = color[u] then
                    return NO
    return YES
```

1. Analyzujte asymptotickou složitost algoritmu *is_2colorable* v nelepším případě.

2. Analyzujte asymptotickou složitost algoritmu *is_2colorable* v nejhorším případě.

Poznámka: Předpokládejte uniformní cenové kritérium, kde složitost každého řádku programu je 1.

5. příklad (problém obarvená KLIKA)

Uvažme konečnou množinu barev B. Mějme neorientovaný graf $G = (V, E)$, kde $E \subseteq \{\{u, v\} \mid u, v \in V\}$ a funkci $c : V \to B$. Množina $V' \subseteq V$ se nazývá *obarvená KLIKA*, pokud V' tvoří úplný podgraf (existuje hrana mezi každýma dvěma vrcholy) a $\exists b \in B : \forall v \in V' : c(v) = b$. Pro dané k se ptáme, zda v G existuje obarvená klika V' o velikosti aspoň k.

Dokažte, že problém obarvená KLIKA je **NP**-úplný.

6. příklad (problém nezávislé množiny)

Mějme neorientovaný graf $G = (V, E)$, kde $E \subseteq \{\{u, v\} \mid u, v \in V\}$. Množina $V' \subseteq V$ se nazývá *nezávislá*, pokud žádné dva uzly z V' netvoří hranu. Pro dané k se ptáme, zda v G existuje nezávislá množina vrcholů V' o velikosti aspoň k.

Dokažte, že problém nezávislé množiny je **NP**-úplný. K důkazu **NP**-těžkosti doporučujeme využít redukce z problému KLIKA.