

Zad 6

Wierzcholki dowolny graf $G = (V, E)$ i krawędź $e = \{u, v\}$ o wadze $w(e) = w$.

Lemat 1

Dla dowolnego cyklu C w grafie i krawędzi $e_1 \in C$, jeśli waga e_1 jest większa niż waga wszystkich innych krawędzi w cyklu C , to e_1 nie należy do żadnego minimalnego drzewa spajającego w grafie.

Dowód lematu 1

Załóżmy niewprost, że $e_1 \in C$ i e_1 należy do jakiegoś MST_1 .

Jeżeli usuniemy e_1 z MST_1 , to MST_1 rozspojni się na dwa drzewa: T_1, T_2 .

Skoro $e_1 \in C$ to istnieje krawędź $e_2 \in C$ łącząca T_1 z T_2 . Wiemy, że $w(e_1) > w(e_2)$. Wtedy, jeśli połączymy T_1 i T_2 krawędzią e_2 to otrzymamy nowe MST_2 o wadze mniejszej niż MST_1 , a to daje sprzeczność.

Algorytm:

Zapamiętajmy wagę krawędzi e_1 , po czym usunijmy e_1 z grafu.

Teraz możemy za pomocą DFS sprawdzić czy istnieje droga z u do v , przy czym DFS może poruszać się wyłącznie po krawędziach mniejszych lub równych $w(e_1)$.

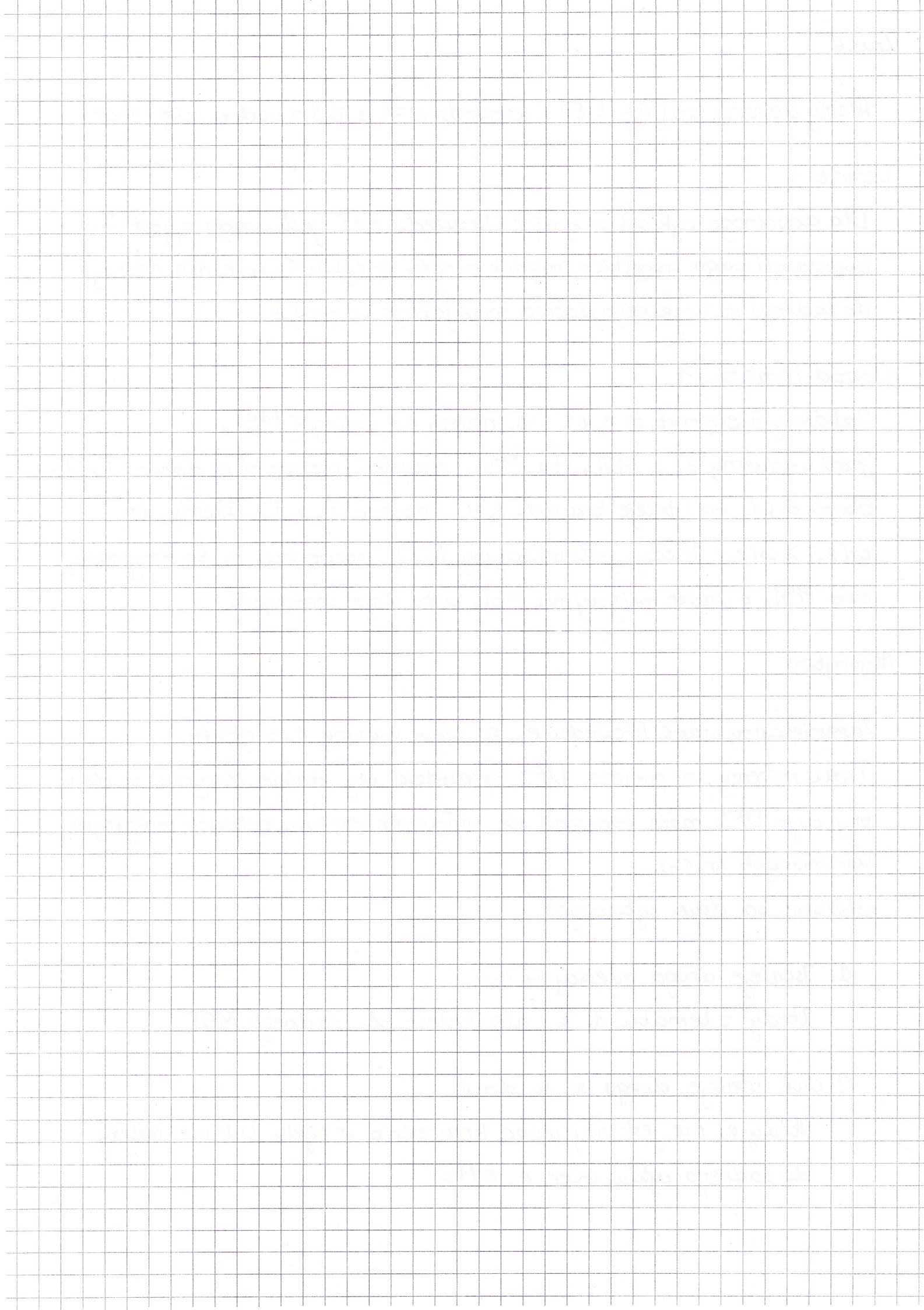
Mogą się dwie sytuacje:

1. Istnieje droga między u i v .

Wtedy z lematu 1, e_1 nie należy do żadnego MST.

2. Nie istnieje droga z u do v .

Wtedy e_1 nie jest najcięższą krawędzią w cyklu lub nie należy do żadnego cyklu, więc $e_1 \in MST$.



Lematy do zad 8

Załóżmy, że $1 < a < b$ (bez utraty ogólności)

Niech $G(x)$ oznacza liczbę monet w reprezentacji liczby x dla nominaliów $\{1, a, b\}$ będącej wynikiem algorytmu zachłanego, zaś $OPT(x)$ – optymalnego rozwiązania. Wtedy $\forall_{x \in \mathbb{N}} OPT(x) \leq G(x)$

Lemat 1

Dla każdego x oraz dowolnej monety $y \in \{1, a, b\}$ zachodzi

$$OPT(x) \leq OPT(x-y) + 1$$

Dowód

Z optymalnej reprezentacji $x-y$ możemy otrzymać reprezentację $OPT(x-y)+1$ dodając jedną monetę y . W najlepszym wypadku otrzymamy w ten sposób optymalną reprezentację x , w innym wypadku reprezentację o większej liczbie monet niż $OPT(x)$.

Lemat 2

$\forall_{y \in \{1, a, b\}} OPT(x) = OPT(x-y) + 1 \Leftrightarrow$ istnieje optymalna reprezentacja x używająca monety y .

Dowód

\Rightarrow Założymy, że $OPT(x) = OPT(x-y) + 1$. Wówczas reprezentacja optymalna x mogła zostać uzyskana poprzez dodanie monety y do optymalnej reprezentacji $x-y$, czyli istnieje optymalna reprezentacja x używająca monety y .

\Leftarrow Założymy, że istnieje optymalna reprezentacja x używająca monety y . Wówczas możemy zabrać jedną monetę y i otrzymać reprezentację $x-y$ o liczbie monet $OPT(x)-1$. Z lematu 1 wynika, że ta reprezentacja $x-y$ jest optymalna, więc $OPT(x) = OPT(x-y) + 1$

Lemat 3

Jeżeli istnieje kontrprzykład x dla nominatorów $\{1, a, b\}$, to najmniejszy taki x spełnia nierówność $b+1 < x < b+a$

Dowód. Założymy, że dla nominatorów $\{1, a, b\}$ strategia zachlanna jest nieoptimalna.

Ograniczenie dolne: Niech x będzie kontrprzykładem. Rozpatrzymy przypadki:

1° $x < b$: Wówczas $OPT(x) = G(x)$, gdyż mamy do wyboru jedynie monety $\{1, a\}$.

2° $x = b$. Wówczas $OPT(x) = 1 = G(x)$. Tu również x nie jest kontrprzykładem.

3° $x = b+1$. Wówczas reprezentacja optymalna daje takie samo rozwiązańe, jak zachlanna. Widzimy więc, że $x > b+1$.

Ograniczenie górnne: Weźmy dowolny $x \geq b+a$. Założymy, że $\forall_{y < x} G(y) = OPT(y)$

Rozpatrzymy przypadki:

1° Moneta b jest w reprezentacji optymalnej. Wtedy:

$$G(x) = G(x-b) + 1 = OPT(x-b) + 1 = OPT(x)$$

2° Moneta a jest w reprezentacji optymalnej:

$$\begin{aligned} G(x) &= G(x-b) + 1 = OPT(x-b) + 1 \stackrel{\text{lema 1}}{\leq} OPT(x-b-a) + 2 \leq G(x-b-a) + 2 = G(x-a) + 1 = \\ &= OPT(x-a) + 1 = OPT(x) \leq G(x), \quad \text{więc } G(x) = OPT(x) \end{aligned}$$

3° Moneta 1 jest w reprezentacji optymalnej:

$$\begin{aligned} G(x) &= G(x-b) + 1 = OPT(x-b) + 1 \leq OPT(x-b-1) + 2 \leq G(x-b-1) + 2 = G(x-1) + 1 = \\ &= OPT(x-1) + 1 = OPT(x) \leq G(x), \quad \text{więc } G(x) = OPT(x) \end{aligned}$$

Stąd dla dowolnego $x \geq b+a$ $G(x) = OPT(x)$, gdy istnieje jeden kontrprzykład mniejszy niż $b+a$.

Stąd też prawdziwa jest nierówność

$$b+1 < x < b+a$$

Zad 8.

Utwórz algorytm, który dla danych liczb naturalnych a i b sprawdza, czy zachlanna strategia wydawania reszty jest poprawna, gdy zbiór nominatów jest równy $X = \{1, a, b\}$

Strategia zachlanna dla problemu wydawania reszty dla $X = \{1, a, b\}$ jest nieoptymalna wtedy, gdy $0 < r < a - k$, gdzie $b = k \cdot a + r$. (czyli $k = \lfloor \frac{b}{a} \rfloor$ oraz $r = b \bmod a$). Założymy, że $1 < a < b$.

Dowód

\Leftarrow Wiedz $b = k \cdot a + r$. Założymy, że $0 < r < a - k$. Wówczas istnieje kontrprzykład $x = b + a - 1$, dla którego reprezentacja zachlanna to $(1, 0, a - 1)$. $x = b + a - 1 = (k + 1)a + (r - 1)$, więc można go również przedstawić jako $(0, k + 1, r - 1)$, co użycia $k + r$ monet, natomiast reprezentacja zachlanna użyczała a monet. Założenia $0 < r < a - k$, więc $0 < r + k < a$ i strategia zachlanna jest niepoprawna.

\Rightarrow Założymy, że strategia zachlanna wydawania reszty jest nieoptymalna dla nominatów $\{1, a, b\}$. Najmniejszy kontrprzykład x spełnia nierówność $b + 1 < x < a + b$ (*), więc jego reprezentacja zachlanna to $(1, 0, i)$, $0 < i < a$. Reprezentacja optymalna to $(0, p, q)$, skoro jest optymalna to $p + q < i + 1$, a skoro $p > 0$, to też $q < i$. W takim razie x jest minimalny, jeżeli $q = 0$ (gdzieby $q \neq 0$ to $x - q$ byłby lepszym kontrprzykładem!)

$$x = b + i = p \cdot a$$

$$b = p \cdot a - i = p \cdot a - i + a - a = (p - 1)a + (a - i)$$

Wtedy $k = p - 1$ oraz $r = a - i$. W nierówności $b + 1 < x < a + b$:

$$0 < (a + b) - x = (a + b) - (b + i) = a - i \quad i \neq 0$$

Mamy:

$$0 < a - i < a$$

Skoro rozwiązażanie optymalne ma mieć mniej monet niż zachłanne, to:

$$p < i + 1$$

$$p - 1 < i$$

Stąd

$$0 < a - i < a - (p - 1), \quad r = a - i, k = p - 1$$

Więc $0 < r < a - k$

Algorytm sprawdzający, czy algorytm zachłanny da rozwiązanie optymalne:

$$r \leftarrow b \bmod a$$

$$k \leftarrow \lfloor \frac{b}{a} \rfloor$$

$$\text{if } (0 < r \text{ and } r < a - k)$$

return False

return True

Zad 9

Chcemy, by wagи $\{w_1, \dots, w_n\}$ były wagami liści w naszym drzewie.
Zauważmy, że możemy budować drzewo „w góre” następującym algorytmem:

Algorytm:

Dla każdego w_i tworzymy wierzchołek v_i i wprowadzamy do kolejki priorytetowej Q .

Dopóki $|Q| > 1$

Niech u, v - dwa pierwsze wierzchołki z Q

Utwórz wierzchołek x o wadze $c(x) = c(u) + c(v)$; ustaw u i v jako jego dzieci.

Wnucić x do kolejki Q

Wstawienie elementu do kolejki priorytetowej działa w czasie $O(\log n)$, a pętla wykona się $O(n)$ razy, więc nasz algorytm działa w czasie $O(n \log n)$.
Złożoność pamięciowa: $O(n)$.

Lemat 1

Niech w_i i w_j będą najmniejszymi wagami w zbiorze $W = \{w_1, \dots, w_n\}$.

Istnieje optymalne rozwiązanie, w którym w_i i w_j są brązowymi.

Dowód

Niech T będzie rozwiązaniem optymalnym. Ustalmy dwa wierzchołki w_k i w_m które są brązowymi i są na najniższym poziomie drzewa T .

Mogliśmy zamienić w_k i w_i oraz w_m i w_j miejscami i otrzymamy drzewo T' .

$$EL(T') \leq EL(T)$$

$$d_T(w_k) \geq d_T(w_i), \quad w_i \leq w_k$$

$$\begin{aligned} EL(T') &= EL(T) - w_i d_T(w_i) - w_k d_T(w_k) + w_i d_T(w_k) + w_k d_T(w_i) = \\ &= EL(T) + w_i(d_T(w_k) - d_T(w_i)) - w_k(d_T(w_k) - d_T(w_i)) = \\ &= EL(T) + (w_i - w_k)(d_T(w_k) - d_T(w_i)) \end{aligned}$$

$$d_T(w_k) - d_T(w_i) \geq 0$$

$$w_i - w_k \leq 0$$

$$EL(T) + (d_T(w_k) - d_T(w_i))(w_i - w_k) \leq EL(T)$$

Lemat 2

Dzewo zwrocone przez algorytm jest optymalne.

Dowód indukcyjny po n:

Baza indukcji: $n=1 \vee n=2$ - istnieje tylko jedno takie dzewo więc jest ono optymalne.

Krok indukcyjny: Założymy, że dla kaidego $k \leq n$ algorytm zwroca optymalne rozwiązańie.

w_i, w_j - najmniejsze wagи:

Rozpatrzmy $W' = W \setminus \{w_i, w_j\} \cup \{w_i + w_j\}$

P - rozwiązańie zwrocone przez algorytm dla W , a P' - dla W' .

Wiek T - dzewo optymalne dla W .

z lematu 1 wiemy, że $w_i \neq w_j$ są brąmi. Usuwamy z T $w_i \neq w_j$, a jako wage ich ojca ustawiamy $w_i + w_j$. Wiek to dzewo nazwana się T' .

$$EL(T') \geq EL(P')$$

$$EL(T) = EL(T') + w_i + w_j$$

$$EL(P) = EL(P') + w_i + w_j$$

$$EL(T) \geq EL(P)$$

Zad 1

Algorytm:

procedure gcd(a, b):

if $\text{abs}(a) == \text{abs}(b)$:

 return a

if $a \% 2 == 0$ and $b \% 2 == 0$:

 return $2 \cdot \text{gcd}(\frac{a}{2}, \frac{b}{2})$

if $a \% 2 == 1$ and $b \% 2 == 0$:

 return $\text{gcd}(a, \frac{b}{2})$

if $a \% 2 == 0$ and $b \% 2 == 1$:

 return $\text{gcd}(\frac{a}{2}, b)$

else return $\text{gcd}((\text{abs}(a) - \text{abs}(b))/2, \text{abs}(b))$

Algorytm realizuje własność 2 treści, więc jest poprawny. Ponadto, $\text{gcd}(a, b) = \text{gcd}(b, a)$ oraz $\text{gcd}(a, b) = a$ gdy $|a| = |b|$.

Zauważmy, że w każdym wywołaniu rekurencyjnym co najmniej jeden z argumentów zmniejsza się co najmniej o połowę. Stąd:

$$T(a, b) = 2(\log a + \log b) = O(\log a + \log b) = O(\log(ab))$$

Standardowy algorytm Euklidesa w każdym kroku daje dwie liczby, których suma jest co najmniej o połowę mniejsza niż liczby w poprzednim kroku.

$$T(a, b) = 2(\log a + \log b) = O(\log(a+b))$$

Porównanie:

$$\log ab \leq^* \log b^2 = 2 \log b = O(\log b)$$

$$\log(a+b) \leq^* \log(b+b) = \log b + \log 2 = O(\log b)$$

* - Bez utraty ogólności

zakładamy, że $a \leq b$

Zad 2

Długość czasu algorytmu: $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2$

Lemat:

Niech $n = 2^i + k$, $k < 2^i$. Wtedy:

$$T(n) = \begin{cases} \frac{3}{2}2^i - 2 + 2k & \text{gdy } k \leq 2^{i-1} \\ 2 \cdot 2^i - 2 + k & \text{gdy } k > 2^{i-1} \end{cases} \quad T(1) = 0$$

Dowód

Podstawa indukcji: $i=1$, $k \in \{0, 1\}$ zachodzi

Krok indukcyjny:

$$1^{\circ} k \leq 2^{i-1}$$

$$\lceil \frac{n}{2} \rceil = 2^{i-1} + \lceil \frac{k}{2} \rceil$$

$$\lceil \frac{k}{2} \rceil \leq 2^{i-2}$$

$$\begin{aligned} T(n) &= T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2 = \frac{3}{2}2^{i-1} - 2 + 2\lfloor \frac{k}{2} \rfloor + \frac{3}{2}2^{i-1} - 2 + 2\lceil \frac{k}{2} \rceil + 2 = \\ &= \frac{3}{2}2^i - 2 + 2k \end{aligned}$$

$$2^{\circ} k > 2^{i-1}$$

$$T(n) = 2 \cdot 2^{i-1} - 2 + \lfloor \frac{k}{2} \rfloor + 2 \cdot 2^{i-1} - 2 + \lceil \frac{k}{2} \rceil + 2 > 2 \cdot 2^i - 2 + k$$

Punkt 1: Należy sprawdzić, kiedy zachodzi $T(n) = OPT(n)$, gdzie

$$OPT(n) = \lceil \frac{3}{2}n - 2 \rceil = 3 \cdot 2^i + \lceil \frac{3}{2}k \rceil - 2$$

$$\cdot r \leq 2^{i-1}$$

$$\begin{aligned} T(n) - OPT(n) &= \frac{3}{2}2^i - 2 + 2k - 3 \cdot 2^{i-1} + 2 - \lceil \frac{3}{2}k \rceil = \frac{3}{2}2^i - 2 + 2k - \frac{3}{2}2^k + 2 - \lceil \frac{3}{2}k \rceil = \\ &= 2k - \lceil \frac{3}{2}k \rceil = \lfloor \frac{k}{2} \rfloor \end{aligned}$$

$$\cdot r > 2^{i-1}$$

$$\begin{aligned} T(n) - OPT(n) &= 2 \cdot 2^i - 2 + k - 3 \cdot 2^{i-1} + 2 - \lceil \frac{3}{2}k \rceil = 2 \cdot 2^i - 2 + k - \frac{3}{2}2^i + 2 - \lceil \frac{3}{2}k \rceil = \\ &= 2^{i-1} - \lceil \frac{k}{2} \rceil \end{aligned}$$

$$T(n) - OPT(n) = 0 \Leftrightarrow (k \leq 2^{i-1} \wedge \lfloor \frac{k}{2} \rfloor = 0) \vee (k > 2^{i-1} \wedge \lceil \frac{k}{2} \rceil = 2^{i-1})$$

$$T(n) - OPT(n) = 0 \Leftrightarrow k \in \{0, 1, \dots, 2^i - 1\} \subseteq (k \leq 2^i - 2 \text{ al.})$$

Punkt 2:

$T(n) - OPT(n) \leftarrow$ oznaczmy przez $N(n)$

$$N(n) = \begin{cases} \lfloor \frac{k}{2} \rfloor & \text{gdy } k \leq 2^{i-1} \\ 2^{i-1} - \lceil \frac{k}{2} \rceil & \text{gdy } k > 2^{i-1} \end{cases}$$

niemalejaca
nizerosnaca

Stąd maksymalna wartość $N(n)$ powinna znajdować się przy $k = 2^{i-1}$

$$N(n)_{\max(k \leq 2^{i-1})} = 2^{i-2}$$

$$N(n)_{\max(k > 2^{i-1})} < 2^{i-1} - \lceil \frac{2^{i-1}}{2} \rceil = 2^{i-1} - 2^{i-2} = 2^{i-2}$$

Stąd $N(n)$ jest najwyżejne dla $k = 2^{i-1}$ i wynosi 2^{i-2} .

Punkt 3:

Procedure MaxMin($S: \text{set}$)

if $|S|=1$ then return $\{a_1, a_1\}$

else if $|S|=2$ then return $(\max(a_1, a_2), \min(a_1, a_2))$

else

if $|S| \% 2 == 0$ and $|S|/2 \% 2 == 1$

podziel S na dwa zbiory $S1, S2$ o mocach $|S1|/2 - 1, |S1|/2 + 1$

else

podziel S na dwa równoliczne (2 dodatk. do 1 elem.) zbiory $S1, S2$.

$(\max1, \min1) \in \text{MaxMin}(S1)$

$(\max2, \min2) \in \text{MaxMin}(S2)$

return $(\max(\max1, \max2), \min(\min1, \min2))$

