Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki

Jakub Cisło
Numer albumu: 275504

Kierunek studiów: Informatyka

Problem rozliczenia wydatków w grupie osób

Praca magisterska
wykonana
pod kierunkiem
prof. dr. hab. Andrzeja Szepietowskiego

Gdańsk, 2022

Pracę dedykuję
żonie Agnieszce
i synowi Antoniemu

Spis treści

1 Wstęp **7**

2 Definicje **9**
2.1 Opis Problemu rozliczenia wydatków 9
2.2 Klasy złożoności 13

3 Dowód $\mathcal{NP}$-zupełności Decyzyjnego problemu rozliczenia wydatków **15**
3.1 Decyzyjny problem rozliczenia wydatków jest w klasie $\mathcal{NP}$ 15
3.2 Redukcja do Decyzyjnego problemu rozliczenia wydatków 15

4 Algorytm dla Problemu rozliczenia wydatków **19**
4.1 Algorytm zachłanny 19
4.2 Algorytm optymalny 22
4.2.1 Zachłanny algorytm wyszukiwania podziału zbilansowanego . . 24
4.3 Algorytm optymalny - wersja szybsza 25

5 Podsumowanie **29**

Rozdział 1

Wstęp

Wyobraźmy sobie grupę osób, która postanawia spędzić wspólnie wakacje. Podczas wyjazdu korzystają oni z wielu atrakcji, za które trzeba zapłacić. Aby usprawnić płatności, każdy rachunek pokrywa jedna osoba (niekoniecznie ta sama za każdym razem). Dodatkowo w każdej atrakcji może uczestniczyć jakiś podzbiór osób, a koszt atrakcji dzielony jest równomiernie pomiędzy korzystających z niej. W ten sposób poszczególne osoby tworzą sobie długi u innych. Przykładową sytuację obrazuje tabela 1.1, długi zaś przedstawione są w tabeli 1.2, gdzie liczba ujemna oznacza, że dana osoba winna jest innej pieniądze, zaś liczba dodatnia oznacza, że to inne osoby muszą danej osobie oddać podaną kwotę.

Atrakcja	Koszt	Kto opłacił	Kto brał udział			
			Ania	Bartek	Czarek	Darek
Ognisko	80	Bartek	✓	✓	✓	✓
Wycieczka	300	Czarek	✓	✓	✓	
Park rozrywki	180	Darek			✓	✓
Basen	100	Czarek		✓	✓	

Tabela 1.1: Przykładowy udział w wydatkach

Atrakcja	Koszt	Koszt na osobę	Bilans			
			Ania	Bartek	Czarek	Darek
Ognisko	80	20	-20	60	-20	-20
Wycieczka	300	100	-100	-100	200	0
Park rozrywki	180	90	0	0	-90	90
Basen	100	50	0	-50	50	0
		Suma	-120	-90	140	70

Tabela 1.2: Bilans dla wydatków z tabeli 1.1

Po powrocie z wakacji znajomi chcą uregulować swoje należności. Zastanawiają się jednak, kto komu powinien oddać jaką kwotę, aby liczba transakcji była możliwie jak najmniejsza. Przykładowe optymalne rozwiązanie znajduje się w tabeli 1.3.

Kto	Komu	Kwota
Ania	Czarek	120
Bartek	Czarek	20
Bartek	Darek	70

Tabela 1.3: Przykładowy optymalny sposób rozliczenia wydatków

Rzeczywiście, jeśli Ania odda Czarkowi 120 zł, a Bartek Czarkowi i Darkowi odpowiednio 20 zł i 70 zł, to zlikwidują swoje zadłużenia. Z drugiej zaś strony, jeśli Czarek otrzyma 120 zł + 20 zł, a Darek 70 zł, to także będą mieli zwrócone poniesione koszty. Łatwo także sprawdzić, że nie da się uzyskać takiego stanu za pomocą tylko 2 transakcji (bo żadna para osób nie ma przeciwnych wartości bilansu).

Głównym tematem niniejszej pracy będzie Problem rozliczenia wydatków w grupie osób (w skrócie PRW). Dane będą bilanse osób (wiersz *Suma* z tabeli 1.2), a rozwiązaniem będzie najkrótsza lista transakcji niwelująca te bilanse do zera (taka jak tabela 1.3). Rozważać będziemy także wariant decyzyjny - DPRW, w którym dana jest również liczba k, a odpowiedzią jest Tak lub Nie w zależności, czy k przelewów wystarczy do rozliczenia.

Autora tej rozprawy do zajęcia się PRW zainspirowała sytuacja z życia - sam potrzebował rozliczyć się wspólnie ze znajomymi. Istnieje wiele aplikacji czy stron internetowych, które w tym pomagają. Przykładami są: Tricount, Splitwise, Kittysplit, jednak nie gwarantują one minimalnej liczby przesunięć pieniędzy. Autor zadał więc sobie pytanie, czy taka gwarancja jest trudna z algorytmicznego punktu widzenia.

W publikacjach [4, 5] można znaleźć podobnie sformułowany problem. David Vávra [4] skupia się jednak na opisie własnej aplikacji, która rozlicza wydatki. Jedynie wspomina, że DPRW jest $\mathcal{NP}$-zupełny, nawiązując do Problemu sumy podzbioru (bez dowodu). Opisuje także 2 algorytmy heurystyczne. Tom Verhoeff [5] również nawiązuje do Problemu sumy podzbioru oraz informuje o $\mathcal{NP}$-zupełności DPRW. Przedstawia redukcję z Problemu trójpodziału oraz szkicuje algorytm, który sprawdza wszystkie możliwe podziały bilansów.

W niniejszej pracy opiszemy PRW i DPRW oraz przypomnimy definicje klas złożoności. Następnie udowodnimy $\mathcal{NP}$-zupełność DPRW pokazując redukcję wielomianową z Problemu sumy podzbioru. W kolejnym rozdziale zaprezentujemy nowy algorytm, który będzie rozwiązywał PRW. Algorytm będzie bazował na szukaniu podziału zbilansowanego o maksymalnej liczności. Udowodnimy, że jest optymalny, a dodatkowo pokażemy dane, dla których znajduje lepsze rozwiązanie niż heurystyka z pracy [4].

Rozdział 2

Definicje

Najpierw zdefiniujmy pojęcia oraz konwencje, z których będziemy korzystać w całej pracy.

Konwencja 2.1. Przez $[\phi]$, gdzie ϕ jest pewnym zdaniem logicznym, będziemy oznaczać wartość logiczną zdania ϕ za pomocą liczb $\{0, 1\}$, tj.:

$$[\phi] = \begin{cases} 1 & \text{gdy } \phi \text{ jest prawdą} \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

Dla przykładu: $[2 > 3] = 0$, [5 jest liczbą pierwszą] $= 1$.

2.1 Opis Problemu rozliczenia wydatków

Zdefiniujmy teraz pojęcia związane ściśle z tytułowym problemem.

Dane jest N osób oraz M atrakcji ($N, M \in \mathbb{Z}_+$). Przez c_j dla $j = 1, ..., M$ oznaczmy cenę j-tej atrakcji, $c_j \in \mathbb{Z}_+$. Niech $p_{ij} =$ [i-ta osoba płaciła za j-tą atrakcję] oraz $q_{ij} =$ [i-ta osoba brała udział w j-tej atrakcji] dla $i = 1, ..., N, j = 1, ..., M$. Zakładamy, że tylko jedna osoba opłaca atrakcję, tj. $\sum_{i=1}^{N} p_{ij} = 1$ dla każdego $j = 1, ..., M$. Oznaczmy jeszcze przez $n_j = \sum_{i=1}^{N} q_{ij}$ dla $j = 1, ..., M$, czyli liczbę osób, które wzięły udział w j-tej atrakcji. Wprowadźmy także oznaczenie $\hat{c}_j = \frac{c_j}{n_j}$ dla $j = 1, ..., M$. Jest to koszt atrakcji na osobę.

Dla sytuacji reprezentowanej przez tabelę 2.1 mamy $N = 4$ oraz $M = 5$, wartości c_j, p_{ij}, q_{ij}, n_j oraz $\hat{c}_j$ są zaś przedstawione w tabeli 2.2.

Atrakcja	Koszt	Kto opłacił	Kto brał udział			
			Ania	Bartek	Czarek	Darek
Ognisko	80	Bartek	✓	✓	✓	✓
Wycieczka	300	Czarek	✓	✓	✓	
Park rozrywki	180	Darek			✓	✓
Basen	100	Czarek		✓	✓	
Muzeum	200	Czarek	✓	✓	✓	✓

Tabela 2.1: Przykładowy udział w wydatkach

j	c_j
1	80
2	300
3	180
4	100
5	200

(a) c_j

j \ i	1	2	3	4
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	0	0	1	0
5	0	0	1	0

(b) p_{ij}

j \ i	1	2	3	4	n_j
1	1	1	1	1	4
2	1	1	1	0	3
3	0	0	1	1	2
4	0	1	1	0	2
5	1	1	1	1	4

(c) q_{ij} oraz n_j

j	$\hat{c}_j$
1	20
2	100
3	90
4	50
5	50

(d) $\hat{c}_j$

Tabela 2.2: Wartości c_j, p_{ij}, q_{ij}, n_j oraz $\hat{c}_j$ dla sytuacji z tabeli 2.1

Możemy teraz zdefiniować bilans osoby.

Definicja 2.2 (Bilans)**.** Bilansem B_i i-tej osoby będziemy nazywać sumę poniesionych przez nią kosztów (sumę pieniędzy, które wydała na atrakcje przez nią opłacane) pomniejszoną o sumę należności za atrakcje, w których brała udział.

$$B_i = \sum_{j=1}^{M} p_{ij} \cdot c_j - \sum_{j=1}^{M} q_{ij} \cdot \hat{c}_j = \sum_{j=1}^{M} (p_{ij} \cdot c_j - q_{ij} \cdot \hat{c}_j)$$

Tabela 2.3 przedstawia bilanse w sytuacji z tabeli 2.1.

i	1	2	3	4
B_i	-170	-140	290	20

Tabela 2.3: Bilanse w sytuacji z tabeli 2.1

Warto w tym momencie poczynić pewne spostrzeżenie.

Obserwacja 2.3. Suma bilansów wszystkich osób jest równa 0.

Dowód.

$$\begin{aligned}
\sum_{i=1}^{N} B_i &= \sum_{i=1}^{N} \left(\sum_{j=1}^{M} p_{ij} \cdot c_j - \sum_{j=1}^{M} q_{ij} \cdot \hat{c}_j \right) \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} p_{ij} \cdot c_j - \sum_{i=1}^{N} \sum_{j=1}^{M} q_{ij} \cdot \frac{c_j}{n_j} \\
&= \sum_{j=1}^{M} \sum_{i=1}^{N} p_{ij} \cdot c_j - \sum_{j=1}^{M} \sum_{i=1}^{N} q_{ij} \cdot \frac{c_j}{n_j} \\
&= \sum_{j=1}^{M} \left(c_j \cdot \sum_{i=1}^{N} p_{ij} \right) - \sum_{j=1}^{M} \left(\frac{c_j}{n_j} \cdot \sum_{i=1}^{N} q_{ij} \right) \\
&= \sum_{j=1}^{M} c_j - \sum_{j=1}^{M} \frac{c_j}{n_j} \cdot n_j \\
&= 0
\end{aligned}$$

□

Osoby, które mają bilans równy 0 będziemy pomijać, gdyż nie będą istotne w rozliczeniach. Pozostałe osoby podzielimy na 2 kategorie.

Definicja 2.4 (Dłużnik)**.** Osobę numer i będziemy nazywać dłużnikiem, jeśli $B_i < 0$.

Definicja 2.5 (Wierzyciel)**.** Osobę numer i będziemy nazywać wierzycielem, jeśli $B_i > 0$.

Niech n oznacza liczbę dłużników, a m - liczbę wierzycieli. Niech $\alpha_1, ..., \alpha_n$ będą numerami osób, które są dłużnikami, a $\beta_1, ..., \beta_m$ - numerami osób, które są wierzycielami. Stwórzmy teraz 2 ciągi: $(a_1, ..., a_n)$ oraz $(b_1, ..., b_m)$ zdefiniowane następująco:

$$\begin{aligned} a_i &= -B_{\alpha_i} \quad \text{dla } i = 1, ..., n \\ b_j &= B_{\beta_j} \quad \text{dla } j = 1, ..., m \end{aligned}$$

Dla bilansów z tabeli 2.3 otrzymujemy:

$$(a_1, a_2) = (170, 140) \qquad (b_1, b_2) = (290, 20)$$

Mamy więc pogrupowane wartości długów oraz wierzytelności. Problem rozliczenia wydatków w grupie osób będzie operował już na tak przygotowanych wartościach.

Problem rozliczenia wydatków w grupie osób (PRW)
WEJŚCIE: Ciągi $(a_1, ..., a_n)$ oraz $(b_1, ..., b_m)$, $a_i, b_j \in \mathbb{Z}_+$ takie, że

$$\sum_{i=1}^{n} a_i = \sum_{j=1}^{m} b_j \tag{2.1}$$

Liczby zapisane są w systemie binarnym.
WYJŚCIE: Macierz $C \in [0, 1]^{n \times m}$ spełniająca warunki

$$\sum_{j=1}^{m} C_{ij} = 1 \qquad \text{dla } i = 1, ..., n \tag{2.2}$$

$$\sum_{i=1}^{n} C_{ij} \cdot a_i = b_j \qquad \text{dla } j = 1, ..., m \tag{2.3}$$

oraz minimalizująca wartość wyrażenia

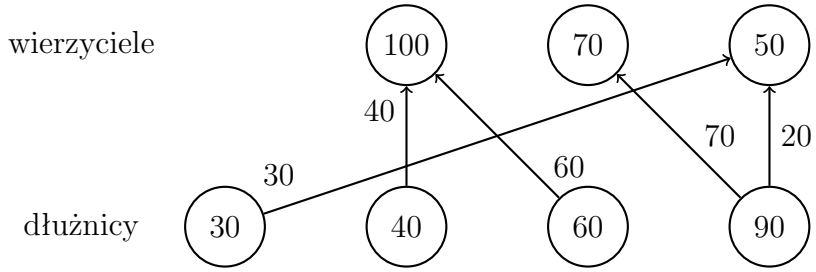
$$\sum_{i=1}^{n} \sum_{j=1}^{m} [C_{ij} \neq 0] \tag{2.4}$$

Założenie (2.1) wynika z obserwacji 2.3. Wartość C_{ij} będziemy interpretować następująco: jaką część swojego długu i-ty dłużnik ma oddać j-temu wierzycielowi. Dlatego też $C_{ij} \in [0, 1]$. To, że każdy dług będzie spłacony oraz że każdy wierzyciel odzyska swoje pieniądze zapewniają odpowiednio warunki (2.2) i (2.3). Wyrażenie (2.4) jest równoznaczne z liczbą transakcji, którą chcemy zminimalizować.

		Wierzyciel		
		100	70	50
Dłużnik	30	0	0	1
	40	1	0	0
	60	1	0	0
	90	0	$\frac{7}{9}$	$\frac{2}{9}$

Tabela 2.4: Przykładowy układ transakcji w postaci macierzy C - poprawne rozwiązanie PRW dla dłużników (30, 40, 60, 90) oraz wierzycieli (100, 70, 50)

Macierz C w naturalny sposób możemy interpretować jako krawędzie pomiędzy dłużnikami a wierzycielami, a całe rozwiązanie reprezentować w postaci grafu dwudzielnego. Dla wygody wagi na krawędziach będą kwotami transakcji zamiast ułamków wartości długów.



Rysunek 2.1: Przykładowy układ transakcji w postaci grafu - poprawne rozwiązanie PRW dla dłużników (30, 40, 60, 90) oraz wierzycieli (100, 70, 50)

W rozważaniach oprócz pełnych rozwiązań (które będziemy nazywać optymalnymi) istotne będą także rozwiązania dopuszczalne.

Definicja 2.6 (Rozwiązanie dopuszczalne)**.** Rozwiązaniem dopuszczalnym dla PRW będziemy nazywać każdą macierz, która spełnia warunki (2.2) oraz (2.3).

Będziemy również rozważać wariant decyzyjny PRW.

Decyzyjny problem rozliczenia wydatków w grupie osób (DPRW)
WEJŚCIE: Liczba $k \in \mathbb{Z}_+$, ciągi $(a_1, ..., a_n)$ oraz $(b_1, ..., b_m)$, $a_i, b_j \in \mathbb{Z}_+$ takie, że

$$\sum_{i=1}^{n} a_i = \sum_{j=1}^{m} b_j$$

Liczby zapisane są w systemie binarnym.
WYJŚCIE: TAK/NIE, w zależności od tego, czy istnieje taka macierz $C \in [0,1]^{n \times m}$, że

$$\sum_{j=1}^{m} C_{ij} = 1 \qquad \text{dla } i = 1, ..., n \tag{2.5}$$

$$\sum_{i=1}^{n} C_{ij} \cdot a_i = b_j \qquad \text{dla } j = 1, ..., m \tag{2.6}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} [C_{ij} \neq 0] \leqslant k \tag{2.7}$$

W tym wariancie na wejściu mamy także liczbę k. Na wyjściu otrzymujemy odpowiedź, czy istnieje sposób rozliczenia wydatków za pomocą co najwyżej k transakcji.

Na koniec zdefiniujemy język związany z DPRW:

Definicja 2.7.

$$L_{\mathsf{DPRW}} = \{D = (k, (a_1, ..., a_n), (b_1, ..., b_m)) \mid \text{odpowiedzią dla } \mathsf{DPRW} \text{ na danych } D \text{ jest } \textsc{Tak}\}$$

2.2 Klasy złożoności

Przytoczmy także kilka definicji, które przydadzą się podczas wyznaczania złożoności DPRW.

Definicja 2.8 (Ograniczenie czasowe maszyny Turinga, [2, Def. 5.1])**.** Powiemy, że maszyna Turinga M działa w czasie ograniczonym przez funkcję $T(n)$ (lub krócej, w czasie $T(n)$), jeżeli dla każdego słowa wejściowego w żadne obliczenie maszyny M na w nie jest dłuższe od $T(|w|)$.

Definicja 2.9 (Klasa $\mathcal{P}$, [2, Def. 5.2])**.** Język $L \in \mathcal{P}$ wtedy i tylko wtedy, gdy istnieje wielomian p oraz deterministyczna maszyna Turinga M z czasem ograniczonym przez p, która akceptuje język L.

Definicja 2.10 (Klasa $\mathcal{NP}$, [2, Def. 5.10])**.** Język $L \in \mathcal{NP}$ wtedy i tylko wtedy, gdy istnieje wielomian p oraz niedeterministyczna maszyna Turinga M z czasem ograniczonym przez p, która akceptuje język L.

Definicja 2.11 (Redukcja wielomianowa, [2, Def. 5.13])**.** Niech $L \subset \Sigma^*$ oraz $K \subset \Delta^*$ będą językami. Powiemy, że L redukuje się w czasie wielomianowym do K (co będziemy oznaczać $L \leqslant_p K$) wtedy i tylko wtedy, gdy istnieje funkcja f obliczalna przez deterministyczną maszynę Turinga w czasie wielomianowym taka, że dla każdego słowa $w \in \Sigma^*$

$$w \in L \iff f(w) \in K$$

Definicja 2.12 (Problem $\mathcal{NP}$-zupełny)**.** Język L jest $\mathcal{NP}$-zupełny wtedy i tylko wtedy, gdy $L \in \mathcal{NP}$ oraz każdy język z klasy $\mathcal{NP}$ redukuje się w czasie wielomianowym do L.

Będziemy także korzystać z następującego twierdzenia:

Twierdzenie 2.13. *Jeżeli L jest $\mathcal{NP}$-zupełny, $K \in \mathcal{NP}$ oraz $L \leqslant_p K$, to K jest $\mathcal{NP}$-zupełny.*

Dowód. Niech H będzie dowolnym problem z klasy $\mathcal{NP}$. Niech M_1 będzie maszyną Turinga działającą w czasie wielomianowym, która redukuje instancje problemu H do instancji problemu L. Niech M_2 będzie maszyną Turinga działającą w czasie wielomianowym, która redukuje instancje problemu L do instancji problemu K. Skonstruujmy maszynę Turinga M, która najpierw działa jak maszyna M_1, a następnie działa jak maszyna M_2 na wyniku z maszyny M_1. Maszyna M redukuje instancje problemu H do instancji problemu K oraz działa w czasie wielomianowym. □

Więcej informacji z teorii złożoności obliczeniowej można znaleźć w publikacjach [2, 3].

Rozdział 3

Dowód $\mathcal{NP}$-zupełności Decyzyjnego problemu rozliczenia wydatków

W tym rozdziale pokażemy, że DPRW jest $\mathcal{NP}$-zupełny korzystając z twierdzenia 2.13. W tym celu udowodnimy, że DPRW jest w klasie $\mathcal{NP}$ oraz pokażemy redukcję z Problemu sumy podzbioru.

3.1 Decyzyjny problem rozliczenia wydatków jest w klasie $\mathcal{NP}$

Twierdzenie 3.1. *DPRW jest w klasie $\mathcal{NP}$.*

Dowód. Niedeterministyczna maszyna Turinga zgaduje macierz C, a następnie sprawdza warunki (2.5), (2.6) oraz (2.7) w czasie wielomianowym. Tak skonstruowana maszyna akceptuje język L_{DPRW}. □

3.2 Redukcja do Decyzyjnego problemu rozliczenia wydatków

Problem sumy podzbioru (PSP)
WEJŚCIE: Skończony zbiór $S \subset \mathbb{Z}_+$ oraz $t \in \mathbb{Z}_+$. Liczby zapisane są w systemie binarnym.
WYJŚCIE: TAK/NIE, w zależności od tego, czy istnieje $S' \subseteq S$ taki, że

$$\sum_{s \in S'} s = t$$

Dla przykładu, niech $S = \{3, 7, 11, 12, 13, 18, 20\}$ oraz $t = 26$. Odpowiedzią jest TAK, bo istnieje żądany podzbiór - jest nim $S' = \{3, 11, 12\}$.

Twierdzenie 3.2 ([1, Tw. 34.15])**.** *PSP jest $\mathcal{NP}$-zupełny.*

Zdefiniujmy teraz algorytm redukcji wielomianowej, który przyjmuje na wejściu instancję PSP oraz zwraca instancję DPRW.

Algorytm REDUKCJA

Wejście: Skończony zbiór $S \subset \mathbb{Z}_+$ oraz $t \in \mathbb{Z}_+$. Liczby zapisane są w systemie binarnym.
Wyjście: Liczba $k \in \mathbb{Z}_+$, ciągi $(a_1, ..., a_n)$ oraz $(b_1, ..., b_m)$, $a_i, b_j \in \mathbb{Z}_+$.

```
1: function REDUKCJA(S, t)
2:     k ← |S|
3:     a ← S
4:     b ← (t, ∑_{s∈S} s − t)
5:     return k, a, b
```

		Wierzyciel	
		26	58
Dłużnik	3	1	0
	7	0	1
	11	1	0
	12	1	0
	13	0	1
	18	0	1
	20	0	1

$k = 7$

Tabela 3.1: Instancja DPRW zredukowana z PSP dla $S = \{3, 7, 11, 12, 13, 18, 20\}$ oraz $t = 26$. W tabeli przedstawiono także macierz C będącą świadectwem odpowiedzi TAK.

Chcemy pokazać 2 własności algorytmu REDUKCJA.

Lemat 3.3. *Algorytm* REDUKCJA *działa w czasie wielomianowym.*

Dowód. Algorytm REDUKCJA wyznacza długość listy z wejścia, sumuje liczby z wejścia, wylicza różnicę wyliczonej sumy i liczby z wejścia, a na koniec kopiuje wartości z wejścia bądź wartości wyliczone. Wszystkie te operacje wykonywane są w czasie wielomianowym, więc cały algorytm także jest wielomianowy. □

Lemat 3.4. *Dana jest dowolna instancja danych dla PSP, tj. skończony zbiór $S \subset \mathbb{Z}_+$ oraz $t \in \mathbb{Z}_+$. Zachodzi następująca równoważność:*

Odpowiedzią dla PSP na danych (S, t) jest TAK $\iff$ *Odpowiedzią dla DPRW na danych zwróconych przez algorytm* REDUKCJA *dla danych (S, t) jest* TAK

W dowodzie lematu 3.4 skorzystamy z pomocniczego lematu 3.5.

Lemat 3.5. *Jeśli macierz* $C \in [0,1]^{n \times m}$ *spełnia warunki*

$$\sum_{j=1}^{m} C_{ij} = 1 \qquad dla\ i = 1, ..., n \tag{3.1}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} [C_{ij} \neq 0] \leqslant n \tag{3.2}$$

to $C_{ij} \in \{0, 1\}$ *dla* $i = 1, ..., n, j = 1, ..., m$.

Dowód. Z (3.1) wiemy, że suma elementów w każdym wierszu 0, zatem w każdym wierszu musi być przynajmniej jedna dodatnia wartość. Co więcej, musi być dokładnie jedna taka wartość - w przeciwnym przypadku otrzymalibyśmy sprzeczność z (3.2). Ponownie z (3.1) otrzymujemy teraz, że niezerowa wartość musi być równa 1. W każdym więc wierszu znajduje się dokładnie jedna liczba 1, a pozostałe wartości to 0. □

Możemy teraz przystąpić do dowodu lematu 3.4.

Dowód lematu 3.4.

($\Longrightarrow$)

Wiemy, że odpowiedzią dla PSP jest Tak, tj. istnieje $S' \subseteq S$ taki, że $\sum_{s \in S'} s = t$. Chcemy pokazać, że odpowiedzią dla DPRW jest Tak, czyli, że istnieje macierz $C \in [0,1]^{|S| \times 2}$ spełniająca odpowiednie warunki.

Przyjmijmy zatem:

$$C_{i1} = [s_i \in S']$$
$$C_{i2} = [s_i \notin S']$$

Sprawdźmy, że wymagane warunki są spełnione.

$$\sum_{j=1}^{m} C_{ij} = \sum_{j=1}^{2} C_{ij} = [s_i \in S'] + [s_i \notin S'] = 1$$
$$\text{dla } i = 1, ..., n$$

$$\sum_{i=1}^{n} C_{ij} \cdot a_i = \sum_{i=1}^{|S|} C_{i1} \cdot s_i = \sum_{i=1}^{|S|} [s_i \in S'] \cdot s_i = \sum_{s \in S'} s = t = b_1 = b_j$$
$$\text{dla } j = 1$$

$$\sum_{i=1}^{n} C_{ij} \cdot a_i = \sum_{i=1}^{|S|} C_{i2} \cdot s_i = \sum_{i=1}^{|S|} [s_i \notin S'] \cdot s_i = \sum_{s \in S, s \notin S'} s = \sum_{s \in S} s - \sum_{s \in S'} s = \sum_{s \in S} s - t = b_2 = b_j$$
$$\text{dla } j = 2$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} [C_{ij} \neq 0] = \sum_{i=1}^{|S|} \sum_{j=1}^{2} [C_{ij} \neq 0] = \sum_{i=1}^{|S|} ([s_i \in S'] + [s_i \notin S']) = \sum_{i=1}^{|S|} 1 = |S| \leqslant |S| = k$$

($\Longleftarrow$)

Wiemy, że odpowiedzią dla DPRW jest Tak, tj. istnieje macierz $C \in [0,1]^{|S|\times 2}$ taka, że:

$$\sum_{j=1}^{2} C_{ij} = 1 \qquad \text{dla } i = 1, ..., |S|$$

$$\sum_{i=1}^{|S|} C_{i1} \cdot s_i = t \qquad \sum_{i=1}^{|S|} C_{i2} \cdot s_i = \sum_{s\in S} s - t$$

$$\sum_{i=1}^{|S|}\sum_{j=1}^{2} [C_{ij} \neq 0] \leqslant |S|$$

Chcemy pokazać, że odpowiedzią dla PSP jest Tak, czyli, że istnieje $S' \subseteq S$ spełniający odpowiednie warunki.

Z lematu 3.5 wiemy, że $C_{ij} \in \{0, 1\}$. Przyjmijmy zatem

$$S' = \{s_i \in S \mid i \in \{1, ..., |S|\} \wedge C_{i1} = 1\}$$

Sprawdźmy, że tak zdefiniowane S' spełnia żądane własności.

$$\sum_{s\in S'} s = \sum_{\substack{i=1,...,|S| \\ C_{i1}=1}} s_i = \sum_{i=1}^{|S|} C_{i1} \cdot s_i = t$$

□

Twierdzenie 3.6. *Algorytm* Redukcja *jest wielomianowym algorytmem redukcji.*

Dowód. Wynika z lematów 3.3 3.4. □

Twierdzenie 3.7. *DPRW jest $\mathcal{NP}$-zupełny.*

Dowód. Wynika z twierdzenia 2.13 oraz twierdzeń 3.1 i 3.6. □

Rozdział 4

Algorytm dla Problemu rozliczenia wydatków

Udowodniliśmy już, że DPRW jest problemem $\mathcal{NP}$-zupełnym. Możemy zatem bez wyrzutów sumienia skonstruować algorytm dla PRW, który będzie działał w czasie wykładniczym. Rozpoczniemy od algorytmu zachłannego, który nie będzie w ogólności optymalny. Posłuży on jednak do konstrukcji algorytmu, który będzie zwracał zawsze najkrótszą listę transakcji. Na koniec usprawnimy algorytm optymalny uzyskując wariant o lepszej złożoności czasowej.

Zaznaczmy, że dla wygody w tym rozdziale będziemy działać na bilansach $(B_1, ..., B_N)$ zamiast na ciągach $(a_1, ..., a_n)$, $(b_1, ..., b_m)$.

4.1 Algorytm zachłanny

Zastanówmy się, jak może działać algorytm, który będzie rozwiązywał PRW. Jednym z pomysłów, który może się nasunąć, jest parowanie dłużnika z wierzycielem wg. jakiegoś kryterium. Przykładowo wybieramy dłużnika z największym długiem oraz wierzyciela z największą wierzytelnością. Niekoniecznie oboje mogą uregulować swoje zobowiązania w całości, ale co najmniej jeden z nich za pomocą transakcji uzyska bilans 0. Osobę (bądź osoby) w pełni rozliczone usuwamy i powtarzamy wybór pary dopóki będą osoby o niezerowym bilansie. Taki algorytm zaprezentował Vávra w pracy [4]. Pseudokod tego podejścia z pewnymi modyfikacjami znajduje się poniżej:

Algorytm Zachłanny

Wejście: Lista bilansów B_i oraz lista indeksów I, dla których bilanse mają zostać rozliczone (domyślnie $I = \{1, ..., |B|\}$, czyli rozliczone mają być wszystkie bilanse). Zakładamy, że $\sum_{i \in I} B_i = 0$ oraz $B_i \neq 0$ dla każdego $i \in I$.
Wyjście: Lista transakcji rozliczająca bilanse B_i dla $i \in I$.

1: **function** Zachłanny$(B, I = \{1, ..., |B|\})$
2: $\quad T \leftarrow \emptyset$

```
3:    while |I| > 0 do
4:        i ← arg max_{ℓ∈I}(−B_ℓ)
5:        j ← arg max_{ℓ∈I}(B_ℓ)
6:        if −B_i < B_j then
7:            x ← −B_i
8:            I ← I − {i}
9:        else if −B_i > B_j then
10:           x ← B_j
11:           I ← I − {j}
12:       else
13:           x ← B_j
14:           I ← I − {i, j}
15:       t ← (i, j, x)
16:       T ← T ∪ {t}
17:       B_i ← B_i + x
18:       B_j ← B_j − x
19:   return T
```

W tym momencie parametr I może wydawać się nadmiarowy, jednak przyda się w dalszych rozważaniach. Zobaczmy zatem symulację działania algorytmu, która znajduje się w tabeli 4.1.

iteracja	B				t
1	$\boxed{-170}$	-140	$\boxed{290}$	20	(1, 3, 170)
2	0	$\boxed{-140}$	$\boxed{120}$	20	(2, 3, 120)
3	0	$\boxed{-20}$	0	$\boxed{20}$	(2, 4, 20)

Tabela 4.1: Przykład działania algorytmu dla $B = (-170, -140, 290, 20)$

Pokażmy teraz kilka własności tego algorytmu.

Lemat 4.1 (Niezmiennik)**.** *Algorytm* Zachłanny *zachowuje niezmiennik* $\sum_{i \in I} B_i = 0$ *oraz* $B_i \neq 0$ *dla każdego* $i \in I$*.*

Dowód. Na samym początku wymagamy, aby dane przekazane do algorytmu spełniały niezmiennik. Pokażmy zatem, że niezmiennik jest spełniony w kroku $\ell > 1$ przy założeniu, że w kroku $\ell - 1$ był zachowany. W kroku ℓ modyfikowane są 2 wartości na liście B: B_i oraz B_j. Zauważmy jednak, że jeśli któraś z tych wartości zostanie wyzerowana to w jednej z gałęzi `if - else if - else` odpowiadający indeks zostanie usunięty z listy I. Nadal więc $B_i \neq 0$ dla każdego $i \in I$.

Jeśli chodzi o sumę to w liniach 17-18 jeden z bilansów jest zwiększany o taką samą wartość jak drugi z bilansów jest zmniejszany. W sumie więc bilans pozostaje niezmieniony. Zmienia się jednak lista I, po której sumujemy. Jednak usuwamy z niej te indeksy, dla których nowa wartość jest równa 0, więc nadal $\sum_{i \in I} B_i = 0$. □

Twierdzenie 4.2. *Algorytm* Zachłanny *znajduje rozwiązanie dopuszczalne dla PRW.*

Dowód. Przeprowadźmy indukcję po $\ell = |I|$. Dla $\ell = 0$, algorytm generuje pustą listę transakcji i jest to rozwiązanie dopuszczalne.

Niech $\ell > 0$. Zakładamy, że dla każdej listy I o długości mniejszej niż ℓ algorytm Zachłanny zwraca poprawne rozwiązanie dopuszczalne. Zgodnie z algorytmem wybieramy dłużnika o największym długu (i) oraz wierzyciela o największej wierzytelności (j). Zauważmy, że zawsze tacy istnieją, co wynika z niezmiennika 4.1 (w I są indeksy, dla których bilanse są niezerowe, a w sumie dają 0). Wybrany więc dłużnik wykonuje przelew o kwocie x dla wybranego wierzyciela. Tym samym co najmniej jeden z nich uzyskuje bilans 0 i jest usuwamy z listy I. W ten sposób uzyskujemy instancję problemu z krótszą listą I i zachowanym niezmiennikiem (lemat 4.1). Możemy zatem skorzystać z założenia indukcyjnego. Rozszerzając rozwiązanie dopuszczalne z mniejszej instancji o transakcję t uzyskujemy rozwiązanie dopuszczalne dla większej instancji. □

Niestety rozwiązanie dopuszczalne nie musi być optymalne. Rozpatrzmy następujący przykład:

$$B = (-30, -40, -60, -90, 100, 70, 50)$$

Optymalnym rozwiązaniem dla powyższych danych jest 5 transakcji:

iteracja	B							t
1	−30	**−40**	−60	−90	100	**70**	50	(2, 6, 40)
2	**−30**	0	−60	−90	100	**30**	50	(1, 6, 30)
3	0	0	−60	**−90**	**100**	0	50	(4, 5, 90)
4	0	0	**−60**	0	10	0	**50**	(3, 7, 50)
5	0	0	**−10**	0	**10**	0	0	(3, 5, 10)

Tabela 4.2: Optymalne rozwiązanie dla $B = (-30, -40, -60, -90, 100, 70, 50)$

a algorytm Zachłanny wyznacza 6 przelewów:

iteracja	B							t
1	−30	−40	−60	**−90**	**100**	70	50	(4, 5, 90)
2	−30	−40	**−60**	0	10	**70**	50	(3, 6, 60)
3	−30	**−40**	0	0	10	10	**50**	(2, 7, 40)
4	**−30**	0	0	0	**10**	10	10	(1, 5, 10)
5	**−20**	0	0	0	0	**10**	10	(1, 6, 10)
6	**−10**	0	0	0	0	0	**10**	(1, 7, 10)

Tabela 4.3: Zachłanne rozwiązanie dla $B = (-30, -40, -60, -90, 100, 70, 50)$

Algorytm Zachłanny nie jest zatem optymalny, jednak posiada jeszcze jedną własność, którą teraz udowodnimy, a z której będziemy korzystać podczas konstrukcji algorytmu optymalnego.

Twierdzenie 4.3. *Algorytm* Zachłanny *zwraca co najwyżej* $|I| - 1$ *transakcji.*

Dowód. Algorytm Zachłanny w każdym obiegu pętli `while` zwiększa rozmiar listy T o jeden jednocześnie zmniejszając listę I o co najmniej jeden element. Ciągle zachowany jest niezmiennik o sumie 0 (lemat 4.1), więc w ostatniej iteracji lista I musi być 2-elementowa. Daje to ograniczenie górne $|T| \leqslant |I| - 1$. □

4.2 Algorytm optymalny

Rozpocznijmy od zdefiniowania, czym jest podział oraz podział zbilansowany. Idea podziałów pochodzi od Vávry [4] i Verhoeffa [5].

Definicja 4.4 (Podział)**.** Rodzinę $\mathcal{F} \subset \mathcal{P}(\{1, ..., N\})$ nazywamy podziałem zbioru $\{1, ..., N\}$ wtedy i tylko wtedy, gdy spełnia następujące warunki:

$$S \neq \emptyset \qquad \forall S \in \mathcal{F} \tag{4.1}$$

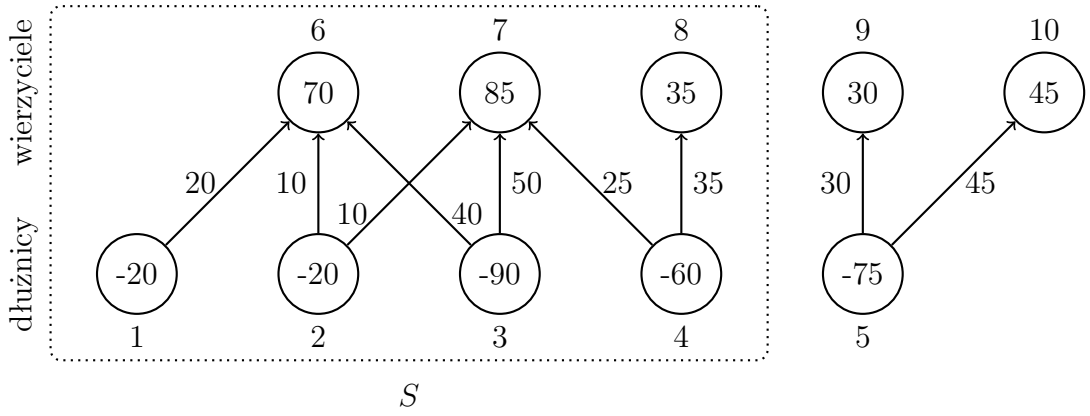
$$\bigcup \mathcal{F} = \{1, ..., N\} \tag{4.2}$$

$$S \cap S' = \emptyset \qquad \forall S, S' \in \mathcal{F}, S \neq S' \tag{4.3}$$

Definicja 4.5 (Podział zbilansowany)**.** Podział $\mathcal{F}$ nazwiemy zbilansowanym wtedy i tylko wtedy, gdy spełnia następującą własność:

$$\sum_{i \in S} B_i = 0 \qquad \forall S \in \mathcal{F} \tag{4.4}$$

W podrozdziale 2.1 wspomnieliśmy, że rozwiązanie dopuszczalne możemy interpretować jako graf, w którym wierzchołkami są dłużnicy i wierzyciela, a krawędziami - transakcje między nimi. W takim grafie może występować kilka spójnych składowych.



Rysunek 4.1: Przykładowy układ transakcji w rozliczeniu z zaznaczoną jedną spójną składową

Zauważmy, że spójne w grafie transakcji wyznaczają podział. Dla przykładu z rysunku 4.1 otrzymujemy podział $\mathcal{F} = \{\{1, 2, 3, 4, 6, 7, 8\}, \{5, 9, 10\}\}$ Okazuje się, że podział ten jest zawsze zbilansowany.

Twierdzenie 4.6. *Dane jest dowolne rozwiązaniu dopuszczalne dla PRW. Jeśli $\mathcal{F}$ jest podziałem wyznaczonym przez spójne składowe w tym rozwiązaniu to $\mathcal{F}$ jest podziałem zbilansowanym.*

Dowód. Przyjrzyjmy się krawędziom w jednej dowolnej spójnej składowej (zob. spójna składowa S na rysunku 4.1). Sumując wartości na krawędziach otrzymujemy z jednej strony sumę długów dla dłużników w tej spójnej składowej, z drugiej zaś strony

sumę wierzytelności dla wierzycieli w tej samej spójnej składowej. Otrzymujemy więc równość:

$$\sum_{i \in S} B_i = 0$$

□

Przypomnijmy teraz zależność pomiędzy liczbą wierzchołków oraz liczbą krawędzi w grafie. Niech $G = (V, E)$ będzie grafem. Jeśli G jest spójny to $|E| \geqslant |V| - 1$. Bardziej ogólnie, jeśli w G jest s spójnych składowych to $|E| \geqslant |V| - s$. Z tej zależności skorzystamy w dowodzie twierdzenia kluczowego dla działania algorytmu optymalnego.

Twierdzenie 4.7 (Ograniczenie dolne)**.** *Jeśli $\mathcal{F}$ jest maksymalnym (w sensie liczności) podziałem zbilansowanym to każde rozwiązanie dopuszczalne dla PRW składa się z co najmniej $N - |\mathcal{F}|$ transakcji.*

Dowód. Na początku pokażmy, że istnieje co najmniej jeden podział zbilansowany. Jest nim $\{\{1, ..., N\}\}$.

Weźmy teraz dowolne rozwiązanie dopuszczalne. Graf transakcji dla tego rozwiązania można podzielić na spójne składowe (być może tylko jedną). Składowe te wyznaczają zbilansowany podział (twierdzenie 4.6), nazwijmy go $\mathcal{F}'$. Z własności grafowych wiemy, że liczba transakcji (czyli liczba krawędzi) w tym rozwiązaniu jest $\geqslant N - |\mathcal{F}'|$. Liczba ta jest także $\geqslant N - |\mathcal{F}|$, bo $\mathcal{F}$ jest maksymalnym w sensie liczności podziałem zbilansowanym. □

Bazując na ograniczeniu dolnym (twierdzenie 4.7) oraz wykorzystując algorytm Zachłanny i jego własność z twierdzenia 4.3 niezależnie dla każdego klastra podziału, skonstruujemy algorytm, który znajduje rozliczenie w dokładnie $N - |\mathcal{F}|$ transakcjach. Dodajmy, że idea wyszukiwania maksymalnego podziału zbilansowanego pojawiła się już wcześniej w pracy Verhoeffa [5], jednak nie była rozwinięta.

Algorytm Optymalny

Wejście: Lista bilansów B_i. Zakładamy, że $\sum_{i=1}^{N} B_i = 0$ oraz $B_i \neq 0$ dla każdego $i = 1, ..., N$.
Wyjście: Lista transakcji rozliczająca bilanse B_i dla $i = 1, ..., N$.

```
1: function Optymalny(B)
2:     F ← OptymalnyPodział(B)
3:     T ← ∅
4:     for S ∈ F do:
5:         T ← T∪Zachłanny(B, S)
6:     return T

7: function OptymalnyPodział(B)
8:     N ← |B|
9:     I ← {1, ..., N}
```

```
10:   F ← ∅
11:   for X ∈ P(P(I)) do
12:       if X jest podziałem zbilansowanym then
13:           if |X| > |F| then
14:               F ← X
15:   return F
```

Algorytm Optymalny składa się z dwóch części. Pierwsza znajduje podział $\mathcal{F}$ z twierdzenia 4.7 o ograniczeniu dolnym. Ta faza realizowana jest w funkcji OptymalnyPodział. Druga część polega na uruchomieniu algorytmu Zachłanny na zbiorach z rodziny $\mathcal{F}$.

Funkcja OptymalnyPodział przegląda wszystkie możliwe rodziny $\mathcal{F} \in \mathcal{P}(\mathcal{P}(\{1, ..., N\}))$ oraz weryfikuje czy $\mathcal{F}$ jest podziałem zbilansowanym (sprawdzając warunki (4.1), (4.2), (4.3), (4.4)). Jeśli tak oraz aktualnie przeglądany podział jest większy od dotychczas znalezionego to następuje aktualizacja wyniku.

Pokażmy zatem, że algorytm Optymalny jest rzeczywiście optymalny.

Twierdzenie 4.8. *Algorytm* Optymalny *jest optymalny, tj. zawsze zwraca* $N - |\mathcal{F}|$ *transakcji, gdzie* $\mathcal{F}$ *jest maksymalnym podziałem zbilansowanym z twierdzenia 4.7 o ograniczeniu dolnym.*

Dowód. Funkcja OptymalnyPodział znajduje $\mathcal{F}$ z ograniczenia dolnego (twierdzenie 4.7). Dla każdego $S \in \mathcal{F}$ jest uruchamiany algorytm Zachłanny, który na podstawie twierdzenia 4.3 zwraca co najwyżej $|S| - 1$ transakcji. Cały algorytm Optymalny zwraca więc co najwyżej:

$$\sum_{S \in \mathcal{F}} (|S| - 1) = \sum_{S \in \mathcal{F}} |S| - |\mathcal{F}| = N - |\mathcal{F}|$$

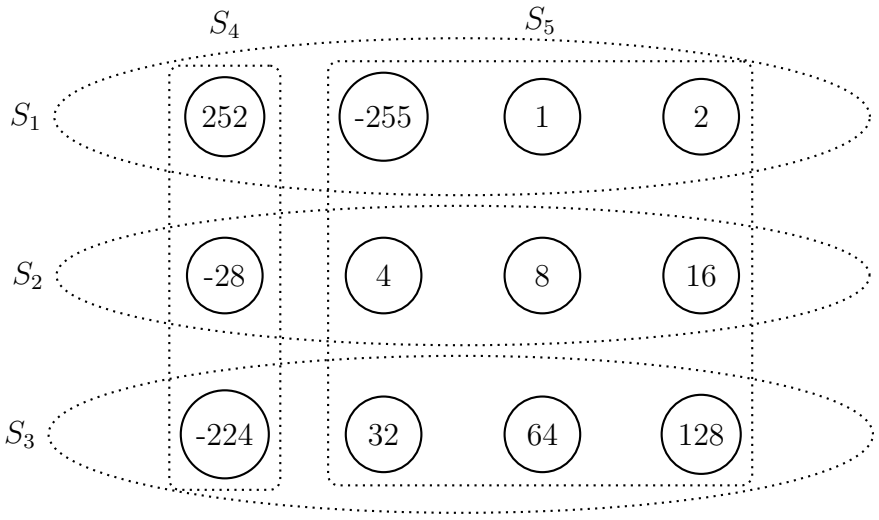
transakcji. Gdyby zwrócił mniej transakcji niż $N - |\mathcal{F}|$ to otrzymalibyśmy sprzeczność z ograniczeniem dolnym (twierdzenie 4.7). Zatem algorytm Optymalny zwraca dokładnie $N - |\mathcal{F}|$ transakcji. □

4.2.1 Zachłanny algorytm wyszukiwania podziału zbilansowanego

Vávra w swojej pracy [4] przedstawia algorytm, w którym konstruuje zbilansowany podział szukając podzbioru najmniejszej mocy o bilansie 0. Podzbiór taki jest dodawany do podziału, a na pozostałej części poszukiwanie jest powtarzane. Vávra sugeruje, że algorytm generuje zawsze maksymalny podział zbilansowany, jednak tak nie jest. Rozpatrzmy następujące bilanse:

$$B = (252, -255, 1, 2, -28, 4, 8, 16, -224, 32, 64, 128)$$

Podzbiory, które sumują się do zera są przedstawione na poniższym rysunku za pomocą linii przerywanych:



Rysunek 4.2: Kontrprzykład dla zachłannego wybierania najmniejszych podzbiorów o sumie 0

Algorytm Vávry wybiera najpierw najmniejszy podzbiór o sumie 0, tj. S_4, a następnie S_5. Generuje tym samym $\mathcal{F} = \{S_4, S_5\}$ o mocy 2. Maksymalna rodzina to jednak $\mathcal{F} = \{S_1, S_2, S_3\}$ o mocy 3.

4.3 Algorytm optymalny - wersja szybsza

Algorytm Optymalny przegląda bardzo dużą liczbę rodzin, z których tylko część jest rozwiązaniami dopuszczalnymi. Aby zredukować liczbę wykonywanych operacji, odwróćmy konstrukcję podziałów zbilansowanych.

Niech $\mathcal{Z} \subset \mathcal{P}(\{1, ..., N\})$ będzie rodziną wszystkich podzbiorów zbilansowanych, tj.

$$\sum_{i \in S} B_i = 0 \qquad \forall S \in \mathcal{Z}$$

Każdy podział zbilansowany zbudowany jest ze zbiorów z rodziny $\mathcal{Z}$. Zauważmy, że prawdziwe jest następujące twierdzenie:

Twierdzenie 4.9. *Jeśli $\mathcal{F}$ jest maksymalnym podziałem zbilansowanym na zbiorze $I \subset \{1, ..., N\}$ oraz $S \in \mathcal{F}$ dla pewnego $S \in \mathcal{Z}$, to $\mathcal{F} - \{S\}$ jest maksymalnym podziałem zbilansowanym na zbiorze $I - S$.*

Dowód. Nie wprost, jeśli istnieje podział zbilansowany $\mathcal{F}'$ większej mocy na zbiorze $I - S$ to $\mathcal{F}' \cup \{S\}$ jest podziałem zbilansowanym o większej liczności niż $\mathcal{F}$, co daje sprzeczność. □

Dzięki powyższej własności możemy skonstruować algorytm, który będzie sprawdzał wszystkie $S \in \mathcal{Z}$, rekurencyjnie znajdował najlepsze rozwiązania podproblemów, powiększał je o zbiór S oraz wybierał maksymalny podział.

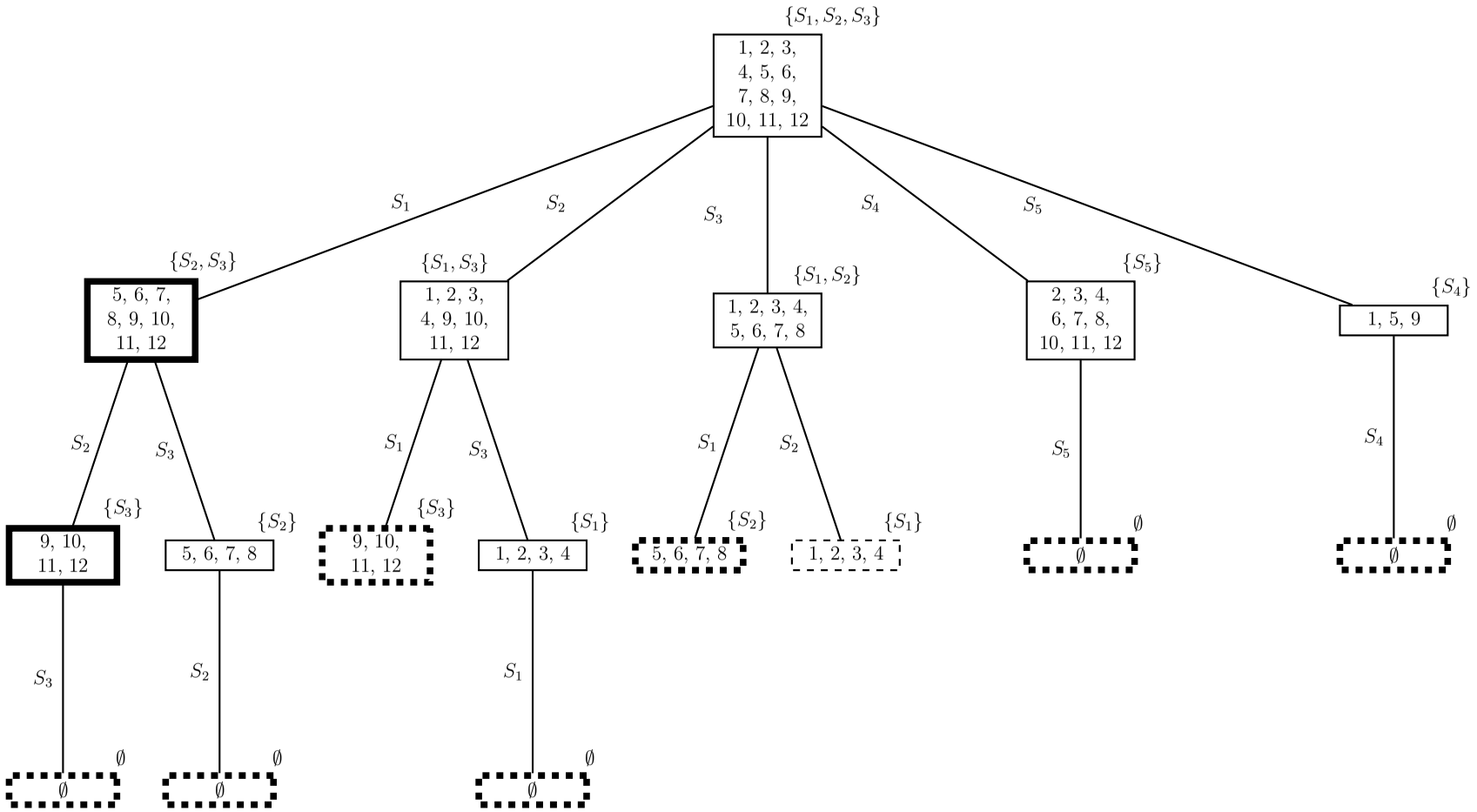
Algorytm SzybkiOptymalny

Wejście: Lista bilansów B_i. Zakładamy, że $\sum_{i=1}^{N} B_i = 0$ oraz $B_i \neq 0$ dla każdego $i = 1, ..., N$.
Wyjście: Lista transakcji rozliczająca bilanse B_i dla $i = 1, ..., N$.

```
1: function SzybkiOptymalny(B)
2:     F ← SzybkiOptymalnyPodział(B)
3:     T ← ∅
4:     for S ∈ F do:
5:         T ← T∪Zachłanny(B, S)
6:     return T

7: function SzybkiOptymalnyPodział(B)
8:     Z ← niepuste podzbiory indeksów o zerowym bilansie na podstawie B
9:     W ← {(∅, ∅)}                    ▷ już wyliczone wyniki - pary (podzbiór
                                         indeksów, maksymalny podział zbilan-
                                         sowany)
10:    function OptymalnyPodziałPodzbioru(I)
11:        if ∃F : (I, F) ∈ W then
12:            return F
13:        F ← ∅
14:        for S ∈ Z do
15:            if S ⊂ I then
16:                X ← OptymalnyPodziałPodzbioru(I − S)
17:                if |X| ⩾ |F| then
18:                    F ← X ∪ {S}
19:        W ← W ∪ {(I, F)}
20:        return F
21:    return OptymalnyPodziałPodzbioru({1, ..., N})
```

Przykład działania algorytmu SzybkiOptymalny przedstawiony jest na rysunku 4.3.



Rysunek 4.3: Wywołania rekurencyjne OptymalnyPodziałPodzbioru w funkcji SzybkiOptymalnyPodział dla $B = (252, -255, 1, 2, -28, 4, 8, 16, -224, 32, 64, 128)$ (zob. także rysunek 4.2). W wierzchołkach przedstawione są indeksy I - parametry wywołania funkcji OptymalnyPodziałPodzbioru, liniami przerywanymi oznaczone są stany, dla których wyniki były już wcześniej policzone i zapamiętane w zmiennej W, pogrubione wierzchołki to stany, których wyniki zostały użyte do wyliczenia wyniku dla swoich ojców, same zaś wyniki dla stanów umieszczone są powyżej wierzchołków.

Twierdzenie 4.10. *Algorytm* SzybkiOptymalny *jest optymalny, tj. zawsze zwraca* $N - |\mathcal{F}|$ *transakcji, gdzie* $\mathcal{F}$ *jest maksymalnym podziałem zbilansowanym z twierdzenia 4.7 o ograniczeniu dolnym.*

Dowód. Algorytm SzybkiOptymalny różni się od algorytmu Optymalny sposobem wyszukiwania podziału $\mathcal{F}$, więc wystarczy pokazać, że funkcja SzybkiOptymalnyPodział rzeczywiście znajduje maksymalny podział zbilansowany.

Najpierw udowodnijmy, że funkcja OptymalnyPodziałPodzbioru zwraca maksymalny podział zbilansowany dla dowolnego $I \subseteq \{1, ..., N\}$. Przeprowadźmy indukcję po $|I|$. Dla $|I| = 0$ wynik jest już w liście W i zostanie zwrócony: $\emptyset$. Odpowiedź jest poprawna.

Załóżmy teraz, że $|I| > 0$, a dla każdego I' takiego, że $|I'| < |I|$ funkcja OptymalnyPodziałPodzbioru zwraca maksymalny podział zbilansowany dla I'. W linii 14 sprawdzane są wszystkie potencjalne elementy S maksymalnego podziału zbilansowanego. Dla każdego przypadku wyliczany jest rekurencyjnie wynik, który jest maksymalnym podziałem zbilansowanym dla $I - S$ na podstawie założenia indukcyjnego. Korzystając zaś z twierdzenia 4.9 jeden z tych podwyników powiększony o zbiór S jest maksymalnym podziałem zbilansowanym dla I.

Pokazaliśmy zatem, że funkcja OptymalnyPodziałPodzbioru zwraca maksymalny podział zbilansowany dla dowolnego $I \subseteq \{1, ..., N\}$. Funkcja SzybkiOptymalnyPodział wywołuje funkcję OptymalnyPodziałPodzbioru z $I = \{1, .., N\}$, więc zwraca maksymalny podział zbilansowany dla wszystkich bilansów. □

Rozdział 5

Podsumowanie

DPRW czyli Decyzyjny problem rozliczenia wydatków w grupie osób okazał się być problemem $\mathcal{NP}$-zupełnym, co udało się dowieść poprzez redukcję z Problemu sumy podzbioru (PSP). Dlatego też skupiliśmy się na poszukiwaniu algorytmu dla PRW, którego złożoność czasowa jest funkcją wykładniczą. Przedstawiony algorytm SzybkiOptymalny zwraca zawsze minimalną liczbę transakcji potrzebną do rozliczenia.

Kilka kwestii związanych z PRW pozostaje jednak wciąż otwartych. Pierwsza z nich to pytanie, czy algorytm SzybkiOptymalny może być jeszcze ulepszony? Podczas odwoływania się do podproblemów, zbiór indeksów $|I|$ zmniejszamy o podzbiór S. Jednak S sam w sobie może mieć nietrywialny podział zbilansowany. Lepiej byłoby zatem znaleźć podzbiory indeksów o zerowym bilansie, które dodatkowo nie mają już nietrywialnego podziału zbilansowanego. Czy jednak można szybko wyznaczyć takie podzbiory?

Możemy także zastanowić się nad różnymi modyfikacjami PRW. Jedną z nich może być wprowadzenie dla każdej osoby jej preferencji odnośnie osób, którym chciałaby oddać swoje długi. Spośród rozwiązań dopuszczalnych wybierane powinny być te, które spełniają jak najwięcej takich preferencji.

Inną modyfikacją może być także umożliwienie korzystania z atrakcji, których ceny są w różnych walutach. Każda z osób będzie wtedy miała długi i wierzytelności osobno w każdej walucie. Jak jednak rozliczyć wydatki, jeśli obowiązywać będą prowizje za przewalutowanie?

Jak można zatem zauważyć, Problem rozliczenia wydatków w grupie osób nie został jeszcze wyczerpany.

Bibliografia

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest i Clifford Stein. *Introduction to Algorithms, 3rd Edition*. Tłum. Krzysztof Diks, Adam Malinowski, Daria Roszkowska i Wojciech Rytter. Wydawnictwo Naukowe PWN, 2012.

[2] Joanna Jędrzejowicz i Andrzej Szepietowski. *Języki, automaty, złożoność obliczeniowa*. Wydawnictwo UG, 2008.

[3] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[4] David Vávra. „Mobile Application for Group Expenses and Its Deployment". Prac. mag. Czech Technical University in Prague, 2012.

[5] Tom Verhoeff. „Settling multiple debts efficiently: an invitation to computing science". W: *Informatics in Education* 3.1 (2004), s. 105–126. ISSN: 1648-5831.