Temat: Algorytmy przybliżone (aproksymacyjne). Pojęcia: błąd przybliżenia, heurystyka. Przykłady prostych algorytmów przybliżonych. Schemat aproksymacji.

1. Algorytmy przybliżone (aproksymacyjne)

Mając do rozwiązania trudny obliczeniowo problem optymalizacyjny, możemy wybrać jedno z dwóch następujących podejść:

- Zastosować **algorytm dokładny** oparty na technice powrotów, której wykorzystanie znacznie redukuje nieefektywność pełnego przeglądu przypadków. Taka ulepszona technika przeszukiwania przestrzeni rozwiązań gwarantuje, iż otrzymujemy rozwiązanie dokładne (optymalne), ale w najmniej korzystnym przypadku danych algorytmy takie mogą wykonywać liczbę operacji elementarnych rosnącą wykładniczo. Stają się wówczas nieużyteczne dla rzeczywistych danych.

- Zastosować **algorytm przybliżony (aproksymacyjny)**, działający zawsze szybko (czyli w czasie wielomianowym), który dla niektórych przypadków danych generuje rozwiązania przybliżone (nieoptymalne), które leżą dość blisko rozwiązań optymalnych.

2. Oszacowanie jakości algorytmów aproksymacyjnych. Pojęcie błędu przybliżenia i heurystyki

Algorytmy dokładne poddawane są analizie złożoności obliczeniowej (czasowej i pamięciowej), natomiast algorytmy przybliżone są badane również pod względem jakości aproksymacji mierzonej wielkością błędu przybliżenia.

a) Ograniczenie względne błędu przybliżenia algorytmu aproksymacyjnego

Załóżmy, że mamy do czynienia z problemem optymalizacyjnym, w którym każde potencjalne rozwiązanie ma dodatni koszt i chcemy znaleźć rozwiązanie prawie optymalne. Zależnie od problemu rozwiązanie optymalne może być zdefiniowane jako to o maksymalnym możliwym koszcie lub o minimalnym. Zadanie optymalizacyjne polega więc na maksymalizacji albo na minimalizacji.

Mówimy, że błąd przybliżenia algorytmu aproksymacyjnego dla danego problemu ma **ograniczenie względne** $\rho(n)$, jeśli dla dowolnych danych wejściowych rozmiaru *n* koszt *C* otrzymany jako rozwiązanie konstruowane przez ów algorytm, szacuje się z dokładnością do czynnika $\rho(n)$ przez koszt C^* rozwiązania optymalnego:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

- Dla problemu maksymalizacji $0 < C \leq C^*$, a współczynnik C^* / C określa, ile razy koszt rozwiązania optymalnego jest większy od kosztu rozwiązania przybliżonego.

- Dla problemu minimalizacji $0 < C^* \leq C$, a współczynnik C / C^* określa, ile razy koszt rozwiązania przybliżonego jest większy od kosztu rozwiązania optymalnego.

Ograniczenie względne algorytmu aproksymacyjnego nigdy nie jest mniejsze niż 1, ponieważ nierówność $C / C^* < 1$ implikuje $C^* / C > 1$. Ograniczeniem względnym algorytmu optymalnego jest 1, a algorytm o dużym ograniczeniu względnym może dać rozwiązanie znacznie gorsze niż optymalne.

b) Ograniczenie błędu względnego algorytmu aproksymacyjnego

Czasami wygodniej jest operować pojęciem błędu względnego przy określaniu jakości aproksymacji. Dla dowolnych danych wejściowych **błąd względny** definiuje się jako

$$\frac{\left|C - C^*\right|}{C^*}$$

Błąd względny jest zawsze nieujemny. Dla algorytmu aproksymacyjnego $\varepsilon(n)$ jest **ograniczeniem błędu względnego,** jeśli

$$\frac{\left|C - C^*\right|}{C^*} \leq \varepsilon(n)$$

Jeżeli na przykład wynik zwrócony przez algorytm dokładny w problemie minimalizacji wynosi 100, a ograniczenie błędu względnego dla pewnego algorytmu aproksymacyjnego rozwiązującego ten problem jest stałe i wynosi 0.2, to znaczy że wynik zwrócony przez algorytm przybliżony może być równy co najwyżej 120.

Dla wielu problemów skonstruowano algorytmy aproksymacyjne o stałym, niezależnym od *n* ograniczeniu względnym. Dla innych problemów naukowcom nie udało się zaprojektować żadnego wielomianowego algorytmu aproksymacyjnego o stałym ograniczeniu względnym. Ograniczenie względne jest wówczas funkcją rosnącą wraz z rozmiarem danych wejściowych *n*.

Algorytmy przybliżone są oparte o zastosowanie różnych heurystyk. **Heurystyka** to zbiór specyficznych dla danego zadania reguł, które mogą dopomóc w odkryciu jak najlepszego rozwiązania. Tradycyjne sposoby przeszukiwania zbioru rozwiązań zależą jedynie od informacji dostarczanej przez dotychczas zbadane elementy tego zbioru. Można jednak stworzyć inny rodzaj strategii, która dzięki odpowiedniej heurystyce pozwala uwzględnić informacje o niezbadanej jeszcze części przestrzeni rozwiązań. Heurystyki są to więc wszelkie metody pozwalające algorytmowi poszukującemu

rozwiązania pójść "na skróty". Skuteczności kroków heurystycznych nie można w pełni udowodnić teoretycznie, można jedynie pokazać doświadczalnie ich trafność.

3. Przykłady prostych algorytmów przybliżonych

a) Algorytm aproksymacyjny dla problemu komiwojażera oparty na technice włączania

WP: Graf $G = \langle V, E, w \rangle$ - pełny, zorientowany z dodatnimi wagami; V - zbiór wierzchołków, E - zbiór krawędzi, w - funkcja wag grafu.
WK: Najtańszy w sensie łącznej sumy wag cykl Hamiltona w grafie.

Idea algorytmu
Wybieramy wierzchołek początkowy cyklu *s*. Następny wierzchołek cyklu wybieramy spośród *n*-1 pozostałych wierzchołków, zgodnie z następującym kryterium:
(*) Wybieramy wierzchołek nie odwiedzony, położony najdalej od cyklu (tzn. od wszystkich wierzchołków, które już są w cyklu). Oznaczmy wybrany wierzchołek przez *p*.
Otrzymujemy w ten sposób cykl złożony z dwóch wierzchołków: (*s*, *p*, *s*). Następnie jest wybierany kolejny wierzchołek, spełniający kryterium (*). Do bieżącego rozwiązania włączamy albo cykl (*s*, *p*, *q*, *s*) albo (*s*, *q*, *p*, *s*) w zależności od tego, który jest tańszy. Kontynuujemy krok wybierania wierzchołka i włączania go do cyklu częściowego aż do momentu uzyskania pełnego cyklu.

Oznaczmy przez V_T zbiór wierzchołków należących do bieżącego cyklu, a E_T zbiór krawędzi tego cyklu.

I krok: Wybieranie wierzchołka

Aby efektywnie wyznaczyć wierzchołek nieodwiedzony, leżący najdalej od wierzchołków bieżącego cyklu, zastosujemy tablicę jednowymiarową odległości *dist*, taką, że *dist*[*v*] zapamiętuje najmniejszą odległość wierzchołków cyklu od *v*. Jeżeli wierzchołek *f*

jest włączany do cyklu jako następny, to ma największą wartość w tablicy *dist*. Po włączeniu *f* do cyklu tablica *dist* jest uaktualniana i każdy jej element staje się równy minimum spośród bieżącej wartości w tablicy *dist* i odpowiedniej wartości w wierszu *f* macierzy wag *W*.

II krok: Włączanie wierzchołka do cyklu

Załóżmy, że bieżący cykl zawiera *k* wierzchołków i następnym do włączenia wierzchołkiem jest *f*. Badamy każdy łuk (*i*, *j*) cyklu i określamy koszt włączenia *f* między *i* oraz *j*, który jest równy:

$$c_{ij} = w_{if} + w_{fj} - w_{ij}$$

Wśród *k* łuków cyklu wybieramy łuk (*t*, *h*) o najmniejszym koszcie c_{th}. Wierzchołek *f* włączamy do cyklu między *t* i *h*, uaktualniamy długość cyklu i wartości elementów w tablicy *dist*.

Pseudokod algorytmu włączania

// Inicjalizacja
V_T={*s*};
E_T={(*s*, *s*)};
w_{ss}=0;
koszt =0;
for *u*∈ $V \setminus \{s\}$ *dist*[*u*]=w_{su};
// Iteracja
while ($|V_T| < n$)
{
f= wierzchołek ze zbioru $V \setminus V_T$ o największej wartości *dist*;
for $(i, j) \in E_T$ $c_{ij} = w_{if} + w_{fj} - w_{ij}$;
(*t*, *h*)=łuk z E_T o najmniejszym koszcie c_{th};
$E_T = E_T \cup \{(t, f), (f, h)\} - \{(t, h)\}$;
$V_T = V_T \cup \{f\}$;
koszt=*koszt* + c_{th};
for *x*∈ $V \setminus V_T$ *dist*[*x*]=*min*{*dist*[*x*], w_{fx}};
}

Wierzchołki V_T i łuki E_T, tworzące bieżący cykl, są reprezentowane w tablicy *cycle*, w której element *cycle*[*i*] przyjmuje wartość *j*, gdy krawędź (*i*, *j*) należy do bieżącego cyklu i 0 w przeciwnym przypadku.

Przykład

$$W = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{bmatrix} 0 & 3 & 93 & 13 & 33 & 9 \\ 4 & 0 & 77 & 42 & 21 & 16 \\ 45 & 17 & 0 & 36 & 16 & 28 \\ 39 & 90 & 80 & 0 & 56 & 7 \\ 28 & 46 & 88 & 33 & 0 & 25 \\ 3 & 88 & 18 & 46 & 92 & 0 \end{bmatrix} \\ \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

Inicjalizacja:
$s = 1$ *dist*: - , 3, 93, 13, 33, 9, *cycle* : 1, 0, 0, 0, 0, 0
koszt: 0

Iteracja:
I krok
$f = 3$ - wierzchołek o maksymalnej wartości *dist*
koszt:= *koszt* + c_{11}=*koszt* + w_{13}+w_{31}=0 + 93 + 45 =138
dist: -, 3, -, 13, 16, 9, *cycle*: 3, 0, 1, 0, 0, 0, cykl: 1-3-1

II krok
$f = 5$
koszt:= *koszt* + *min*{c_{13}, c_{31}}=
= *koszt* +*min*{w_{15}+w_{53} - w_{13}, w_{35}+w_{51} - w_{31}}=
= 138 + *min*{33+88 - 93, 16 + 28 - 45}=
= 138 + *min*{28, -1}=137
dist: -, 3, -, 13, - , 9, *cycle*: 3, 0, 5, 0, 1,0, cykl: 1-3-5-1

$$W = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 3 & 93 & 13 & 33 & 9 \\ 4 & 0 & 77 & 42 & 21 & 16 \\ 45 & 17 & 0 & 36 & 16 & 28 \\ 39 & 90 & 80 & 0 & 56 & 7 \\ 28 & 46 & 88 & 33 & 0 & 25 \\ 3 & 88 & 18 & 46 & 92 & 0 \end{bmatrix} \end{array}$$

III krok

$f = 4$

$koszt := koszt + min\{c_{13}, c_{35}, c_{51}\} =$

$= koszt + min\{w_{14}+w_{43} - w_{13}, w_{34}+w_{45} - w_{35}, w_{54}+w_{41} - w_{51}\} =$

$= 137 + min\{0, 76, 44\} = 137 + 0 = 137$

dist: -, 3, -, -, -, 7 *cycle*: 4, 0, 5, 3, 1, 0, cykl: 1-4-3-5-1

IV krok

$f = 6$

$koszt := koszt + min\{c_{14}, c_{43}, c_{35}, c_{51}\} =$

$= 137 + min\{42, -55, 104, 0\} = 137 - 55 = 82$

dist: -, 3, -, -, -, - *cycle*: 4, 0, 5, 6, 1, 3, cykl: 1-4-6-3-5-1

V krok

$f = 2$

$koszt := koszt + min\{c_{14}, c_{46}, c_{63}, c_{35}, c_{51}\} =$

$= 82 + min\{32, 99, 147, 22, 22\} = 82 + 22 = 104$

Mamy tym razem dwie możliwości włączenia z najmniejszym kosztem. Wybieramy c_{35} i otrzymujemy ostatecznie rozwiązanie *cycle*: 1, 4, 6, 3, 2, 5, 1 o całkowitym koszcie 104.

Otrzymane rozwiązanie jest jedną z dwóch optymalnych tras komiwojażera dla tego przykładu. W ogólnym przypadku nie można jednak oczekiwać, że zawsze będziemy mieli tyle szczęścia, aby algorytm przybliżony generował rozwiązanie optymalne.

Zaleca się, aby algorytm przybliżony zastosować n razy dla danego przypadku danych i wybrać rozwiązanie o najniższym koszcie całkowitym.

Złożoność czasowa algorytmu przybliżonego
Przy jednym uruchomieniu : $O(n^2)$.
Przy n uruchomieniach: $O(n^3)$

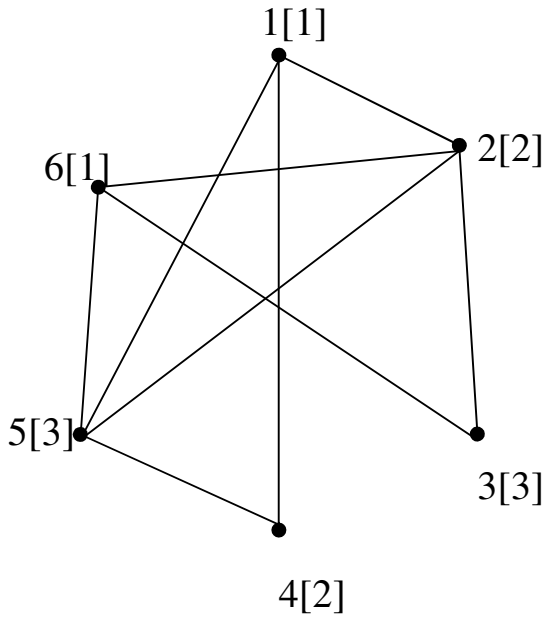
Twierdzenie
Algorytm aproksymacyjny dla problemu komiwojażera zrealizowany metodą przez włączanie jest algorytmem aproksymacyjnym z ograniczeniem względnym równym 2 dla grafów ze spełnioną nierównością trójkąta (Graf ma spełniony warunek nierówności trójkąta, gdy dla każdej trójki krawędzi grafu: (x,z), (x,y) oraz (y,z) spełniony jest warunek: $w_{xz} \leq w_{xy}+w_{yz}$).
Czyli $\frac{C}{C^*} \leq 2$, gdzie C^* jest optymalnym cyklem dla danego grafu, a C jest cyklem wyznaczonym przez algorytm przybliżony.

Niestety nie istnieje algorytm o stałym ograniczeniu względem dla grafów, które nie spełniają nierówności trójkąta.

b) Algorytmy aproksymacyjne dla problemu kolorowania grafu oparty na generowaniu maksymalnych zbiorów niezależnych

Przypisanie kolorów wierzchołkom grafu G, po jednym kolorze dla każdego wierzchołka, tak, aby sąsiednie wierzchołki otrzymały różne kolory nazywamy **pokolorowaniem grafu** G.

Przykład



Liczby w nawiasach kwadratowych oznaczają numer koloru przyporządkowanego wierzchołkowi w procesie kolorowania.

Problem kolorowania grafu

WP: *G* - graf nieskierowany, bez wag.

WK: Liczba chromatyczna grafu – najmniejsza liczba kolorów, którą można pokolorować graf *G*.

Problem kolorowania grafu jest równoważny problemowi podziału zbioru wierzchołków grafu na minimalną liczbę podzbiorów niezależnych:

Takie dwa podzbiory, w których żadne dwa wierzchołki nie sąsiadują ze sobą nazywamy **podzbiorami niezależnymi**.

Podzbiór wierzchołków grafu *W* nazywamy **maksymalnym zbiorem niezależnym**, jeśli nie istnieje zbiór zawierający *W* i różny od *W*, który jest niezależny.

Największy zbiór niezależny to taki maksymalny zbiór niezależny, który ma najwięcej wierzchołków spośród wszystkich zbiorów maksymalnych.

Oznaczenia:

$G_{/W}$ - **podgraf grafu G indukowany przez zbiór $W \subseteq V$**, tzn. $G_{/W} = <W, E'>$, gdzie E' zawiera te krawędzie grafu G, których oba końce należą do W.

Problem maksymalnych zbiorów niezależnych

WP: Graf nieskierowany, bez wag

WK: Wyznaczenie takiej minimalnej liczby k, dla której istnieje podział zbioru V wierzchołków grafu na k zbiorów niezależnych $V_1, V_2, ..., V_k$ takich, że : $V_i \cap V_j = \varnothing$ dla $i \neq j,\ i, j = 1,2,...,k$ oraz $\bigcup_{i=1}^{k} V_i = V$.

Idea algorytmu

Tworzony jest ciąg klas barwliwości (podzbiory wierzchołków, które mogą być pomalowane tym samym kolorem), które są największymi zbiorami niezależnymi w podgrafach, wyznaczonymi przez wierzchołki niepokolorowane.

Algorytm największych zbiorów niezależnych

```
U =V;
k=0;
while (U≠∅)
  {
    k=k +1;
 (*)MAXIND(U, W);// znaleźć najliczniejszy zbiór niezależny
                          // W w pografie G/U
    for u∈ W   f[u]=k;  // f - funkcja kolorująca graf
    U=U \ W;
  }
```

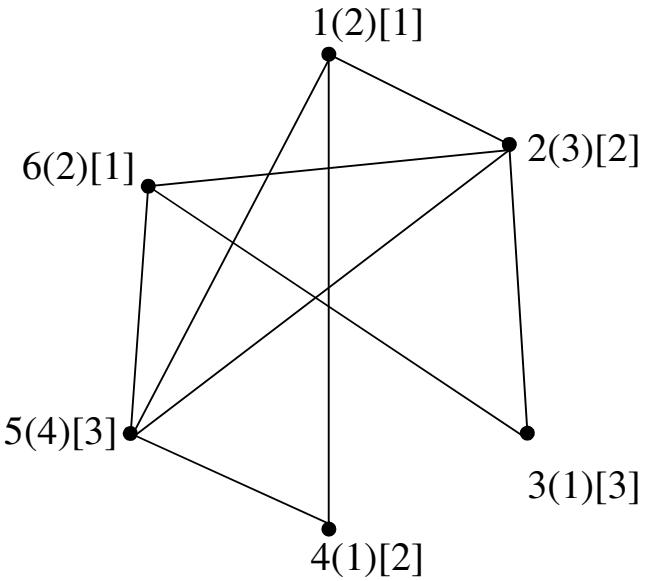
(*) Znajdowanie przybliżonego najliczniejszego podzbioru niezależnego w grafie G

MAXIND(U, W)

$U_1 = U$;
$W = \emptyset$;
while ($U_1 \neq \emptyset$)
{
znaleźć wierzchołek u o minimalnym stopniu w $G_{/U1}$;
$W = W \cup \{u\}$;
$U_1 = U_1 \setminus \{u\} \setminus \{v \in U_1 : (v, u) \in E\}$;
}

Poczynając od $W=\emptyset$, zbiór W jest powiększany w każdym kroku o wierzchołek mający najmniejszy stopień w podgrafie utworzonym przez wierzchołki nie sąsiadujące z elementami zbioru W.

Przykład



Algorytm dokładny daje wynik $k = 3$.
Podział:{1, 6}, {2, 4}, {3, 5} (pokolorowanie zaznaczone w[])
Algorytm przybliżony daje wynik $k = 4$.
Podział:{3, 4}, {1, 6}, {2}, {5} (pokolorowanie zaznaczone w ( ))

Złożoność czasowa algorytmu największych zbiorów niezależnych, wykorzystującego przybliżone generowanie najliczniejszego podzbioru niezależnego.

Można pokazać, że algorytm ten ma złożoność $O(n^3)$.

Twierdzenie
Algorytm aproksymacyjny największych zbiorów niezależnych dla problemu kolorowaniu grafu jest algorytmem aproksymacyjnym z ograniczeniem względnym $O(n/\log n)$.

$$\frac{C}{C^*} \leq O\left(n / \log n\right)$$

c) Prosty algorytm sekwencyjny dla problemu kolorowania grafu (algorytm Welsha-Powella)

W algorytmie tym będziemy zakładali, że wierzchołki zostały ponumerowane w kolejności nierosnących stopni.

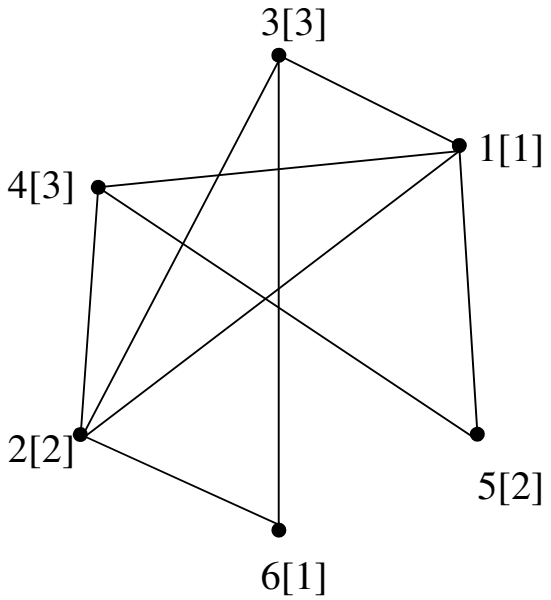
Idea algorytmu
Wierzchołek v_i jest dodawany do podgrafu indukowanego przez już pomalowane wierzchołki v_1, v_2, ..., v_{i-1}, po czym określa się nowe pokolorowanie wierzchołków v_1, v_2, ..., v_{i-1}, v_i. Krok ten jest powtarzany dla i=1, 2, ..., n, przy czym dla i=1 podgraf jest pusty. W każdym kroku staramy się użyć możliwie najmniej kolorów. W algorytmie tym wierzchołek v_i otrzymuje kolor o najniższym numerze.

Algorytm

$f[v_1]$=1;
for (i= 2; i<=n; i++)
  $f[v_i]$=min{k: k≥1 i $f[v_j]$≠k dla każdego v_j (1≤j≤i)
    sąsiadującego z v_i };

Zauważmy, że w prostym algorytmie sekwencyjnym, w momencie kolorowania wierzchołka v_i, wierzchołki poprzedzające v_i zachowują swoje kolory.

Przykład



Wierzchołki grafu są ponumerowane w kolejności nierosnących stopni. Prosty algorytm sekwencyjny daje 3-pokolorowanie (zaznaczone w []):

Złożoność czasowa prostego algorytmu sekwencyjnego
$O(n^2)$.

Twierdzenie
Prosty algorytm sekwencyjny z założenie uporządkowania wierzchołków w kolejności nierosnących stopni jest algorytmem aproksymacyjnym z ograniczeniem względnym nie przekraczającym wartości $1+\max_{1\le i\le n}\{\deg(v_i)\}$.

Czyli $\frac{C}{C^*} \le 1+\max_{1\le i\le n}\{\deg(v_i)\}$ ($\deg(v_i)$ - stopień wierzchołka v_i)

4. Schemat aproksymacji

Dla niektórych problemów trudnych obliczeniowo istnieją algorytmy aproksymacyjne, za pomocą których można uzyskiwać coraz mniejsze ograniczenie błędu względnego kosztem dłuższego czasu obliczeń.

Schemat aproksymacji (*TAS – z ang. Time Aproximation Schema*) to algorytm aproksymacyjny, który na wejściu otrzymuje dane problemu oraz wartość $\varepsilon > 0$ i wyznacza rozwiązanie o ograniczeniu błędu względnego równym ε.

d) Schemat aproksymacji dla problemu sumy podzbioru

WP: $s_1, s_2, ..., s_n, B$ - liczby całkowite nieujemne

WK: Podzbiór indeksów $J \in \{1, ..., n\}$ taki, że suma $\sum_{j \in J} s_j \leq B$ i jest maksymalna.

Przykład

Dane: $S = \{104, 102, 201, 101\}$
$B = 308$
Wynik: 307=104+102+101

a) Algorytm dokładny

Oznaczenia:

$A+x=\{a+x: a \in A\}$
$S = \{ s_i: 1 \leq i \leq n\}$

EXACT_SUBSET_SUM(*S*, *B*)

```
n = |S|;
L = 0;
for (i = 1; i <=n; i++)
    {
      L=MERGE_LISTS(L, L+s_i);
     "usuń z L wszystkie elementy większe niż B";
     }
```

return „największy element z listy L”;

Przykład

Dane: $S = \{104, 102, 201, 101\}$

$B = 308$

Zawartość listy L po kolejnych iteracjach:

L_0: 0
L_1: 0, 104
L_2: 0, 102, 104, 206
L_3: 0, 102, 104, 201, 206, 303, 305, 407
L_4: 0, 101, 102, 104, 201, 203, 205, 206, 302, 307, 404, 406

Kolorem niebieskim zostały zaznaczone elementy usuwane z listy, a kolorem czerwonym zaznaczony został wynik.

Łatwo zauważyć, że w kolejnych iteracjach pętli for lista L będzie zawierać wszystkie możliwe wartości sum podzbiorów spośród elementów s_1, ... , s_i, gdyż operacja MERGE_LISTS scala posortowane listy (w czasie liniowym ze względu na ich długość). Jedna z tych list zawiera wszystkie sumy z poprzedniej iteracji, a druga wszystkie sumy, w których występuje nowy element. Poprawność zwracanej wartości jest oczywista.

Czas działania algorytmu jest rzędu $O(n)$ ze względu na rozmiar zbioru S, ale wykładniczy ze względu na pesymistyczną długość listy L.

O tego typu problemach, dla których algorytm ma koszt wielomianowy ze względu na ograniczenie rozmiaru danych, ale ponadwielomianowy ze względu na faktyczny rozmiar struktur danych użytych do rozwiązania problemu nazywamy **pseudowielomianowymi**.

Aby uzyskać schemat aproksymacji stosujemy modyfikację algorytmu dokładnego polegającą na skracaniu listy L w czasie działania algorytmu. Ogólnie usuniemy z niej pewne elementy, ale tak, żeby

pozostałe dobrze reprezentowały wszystkie występujące na początku. Przy skracaniu użyjemy parametru $0 < \delta < 1$.

Skrócenie listy L przez δ polega na usunięciu z niej możliwie największej liczby elementów tak, żeby dla każdego elementu usuwanego d pozostał element z taki, że :

$$(1- \delta)d \le z \le d$$

Element z jest jakby reprezentantem elementu d. Każdy element skróconej listy jest oczywiście elementem pierwotnej wersji. Oto kod funkcji TRIM skracającej listę postaci $L = (x_1, ..., x_m)$:

TRIM(L, δ)
$M = |L|$;
$L'=(x_1)$;
$last = x_1$;
for (i=2; i<=m; i++)
    if ($last$<(1-δ) x_i)
        {
         "dołącz x_i na koniec listy L' ";
          $last = x_i$;
         }
return L';

Algorytm przybliżony polega na zastosowaniu operacji TRIM po każdym kroku scalania listy L z listą $L+s_i$.

APROX_SUBSET_SUM(S, B, ε)
1. $n = |S|$;
2. $L = 0$;
3. for (i=1; i <=n; i++)
    {
4. L=MERGE_LISTS(L, $L+s_i$);
5. L= TRIM(L, ε/n);
6. "usuń z L wszystkie elementy większe niż B";
    }

7. return „największy element z listy L”;

Przykład

$S = \{104, 102, 201, 101\}$
$B = 308$
$\varepsilon = 0.2$
Parametr skracania $\delta = \varepsilon/4 = 0.05$

Zawartość listy L po kolejnych iteracjach
2. L_0 : 0
4: L_1 : 0, 104
5: L_1 : 0, 104
6: L_1 : 0, 104

4: L_2 : 0, 102, 104, 206 102>(1-0.05)*104
5: L_2 : 0, 102, 206
6: L_2 : 0, 102, 206

4: L_3 : 0, 102, 201, 206, 303, 407 201>(1-0.05)*206
5: L_3 : 0, 102, 201, 303, 407
6: L_3 : 0, 102, 201, 303

4: L_4 : 0, 101, 102, 201, 203, 302, 303, 404 101>(1-0.05)*102,
5: L_4 : 0, 101, 201, 302, 404 201>(1-0.05)*203,
6: L_4 : 0, 101, 102, 201, 302 302>(1-0.05)*303

Wynik wynosi 302 i mieści się w zakresie dokładności $\varepsilon = 0.2$,
ponieważ optymalne rozwiązanie wynosi 307 = 104 + 102 + 101

Czas działania algorytmu jest wielomianowy ze względu na długość list L_i występujących w iteracjach. Można pokazać, że długość listy L_i jest rzędu $O(n\log B/\varepsilon)$.