Analiza algorytmów – zadanie projektowe AAL-11-LS *kartony*

Semestr 2018Z

Koncepcja rozwiązania

Michał Belniak

1. Treść zadania.

   Ortodyksyjny kolekcjoner tekturowych kartonów zaczyna narzekać na brak miejsca do przechowywania swoich cennych zdobyczy. Postanowił oszczędzić miejsce przez wkładanie kartonów jeden w drugi. W trosce o zachowanie dobrego stanu kartonów, umieszcza tylko jeden karton wewnątrz większego, a wolną przestrzeń wypełnia materiałem ochronnym. Tak zabezpieczony karton może następnie umieścić wewnątrz innego większego kartonu, ale nie może umieścić dwóch kartonów obok siebie w jednym kartonie. Dla danego zbioru kartonów należy znaleźć najlepsze upakowanie kartonów, tzn. takie, które zwalnia najwięcej miejsca.

2. Omówienie zadania wraz z założeniami.

   Zadanie polega na znalezieniu takiego upakowania kartonów, które zwalnia jak najwięcej objętości, co jest równoznaczne minimalizowaniu sumy objętości kartonów niespakowanych, czyli takich, których już nie można nigdzie spakować. Przez spakowanie będziemy rozumieli umieszczenie jednego kartonu w drugi.

   Głównym założeniem w tym zadaniu będzie przyjęcie, że każdy karton jest prostopadłościanem. Ułatwi to określenie jego objętości oraz tego, czy dany karton zmieści się w innym danym kartonie.
   Kolejnym założeniem jest to, że wkładając karton w drugi karton, odpowiadające ściany będą do siebie równoległe. Nie uwzględniamy więc przypadku, gdzie wkładamy karton pod skosem. Ułatwi to sprawdzanie jakie kartony mieszczą się w danym, większym kartonie.

   Oznaczmy zbiór kartonów jako:

   $$K=\{k_1, k_2, k_3, \ldots, k_n\},$$

   gdzie k_i jest trójwymiarowym wektorem postaci (x_i, y_i, z_i), opisującym wymiary i-tego kartonu. Przyjmujemy także, że $x_i \geq y_i \geq z_i$ – ułatwi to sprawdzanie, czy karton mieści się w innym kartonie. Z każdym elementem k_i powiązana jest wartość V_i oznaczająca jego objętość, co pozwala utworzyć zbiór

   $$Vk=\{V_1, V_2, V_3, \ldots, V_n\}, \text{ gdzie } V_i=x_i*y_i*z_i.$$

   Przy tak zdefiniowanych zbiorach, zadanie polega na maksymalizacji funkcji celu:

   $$max\ f_z=V_b,$$

gdzie $V_b=\sum_{i=1}^{n} V_i \cdot c_i,\ c_i = \begin{cases} 1, & \text{jeśli } k_i \text{ spakowany w inny karton} \\ 0, & \text{jeśli } k_i \text{ niespakowany} \end{cases}$ – suma objętości kartonów spakowanych w jakiś inny karton. Innymi słowy, chcemy aby objętość kartonów niespakowanych w żaden inny była jak najmniejsza.

Niech możliwość spakowania będzie zdefiniowana przez wektor (k_i, k_j), $i \neq j$, który oznacza, że karton k_j może być spakowany w karton k_i. Warto zauważyć, że jest to relacja przechodnia, tzn.:

$$(k_i,k_j) \wedge (k_m,k_i) \Rightarrow (k_m,k_j),\ i \neq j \neq m$$

Karton k_j może być spakowany w karton k_i wtedy i tylko wtedy, gdy:

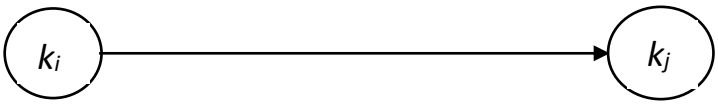
$$x_i > x_j \wedge y_i > y_j \wedge z_i > z_j \qquad (1.1)$$

Wszystkie wektory możliwych zapakowań tworzą zbiór W. Poprawnym rozwiązaniem zadania jest wyznaczony podzbiór *W'* zbioru *W*, w którym dla każdego elementu (k_i, k_j) spełnione są warunki:

1. Jeśli w podzbiorze *W'* istnieje (k_i, k_m), $m \neq j$ to musi istnieć także (k_j, k_m), lub (k_m, k_j). Wynika to z ograniczenia, że nie można umieszczać dwóch kartonów obok siebie.
2. Jeśli w podzbiorze *W'* istnieje (k_k, k_j), $k \neq i$ to musi istnieć także (k_i, k_k), lub (k_k, k_i). Oznacza to, że karton może być spakowany w dwa różne kartony, tylko wtedy, gdy te dwa kartony są wzajemnie spakowane.

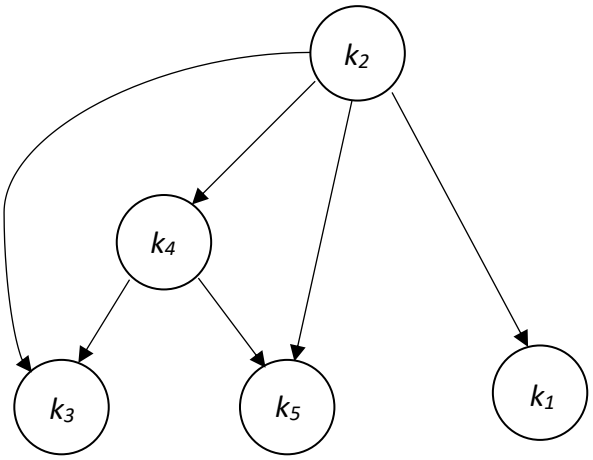
3. Tworzenie odpowiedniej struktury danych.

Wektory oznaczające relację spakowania można reprezentować graficznie. Zbiór możliwych zapakowań można więc reprezentować za pomocą grafu, którego wierzchołkami będą kartony k_i, a krawędziami – wektory (k_i,k_j) ze zbioru W. Pozwoli to uzyskać graf reprezentujący wszystkie możliwe relacje spakowania. Z warunku *(1.1)* wynika, że graf będzie skierowany oraz acykliczny. Poniższy rysunek oznacza, że karton k_j może zostać spakowany w karton k_i.



Dla kilku kartonów przykładowy graf może wyglądać jak na rysunku 1.1.

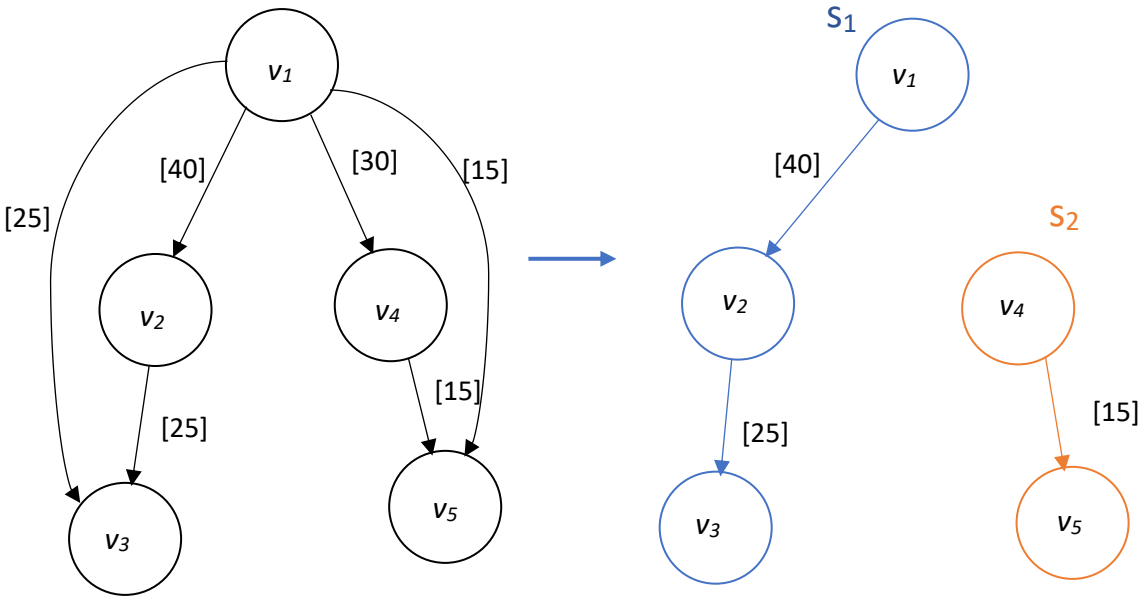
Wyznaczenie podzbioru *W'*, czyli poprawnego rozwiązania polega na wyborze odpowiednich krawędzi grafu, jednak aby znaleźć rozwiązanie optymalne, trzeba najpierw powiązać krawędzie z wartościami objętości kartonów pakowanych. Niech wagą krawędzi (k_i, k_j) będzie objętość kartonu k_j, a więc V_j. Wtedy wybranie krawędzi (k_i, k_j) do podzbioru *W'* będzie oznaczać zaoszczędzenie przestrzeni o objętości V_j, a właśnie sumę zaoszczędzonej objętości należy maksymalizować, aby znaleźć rozwiązanie optymalne.



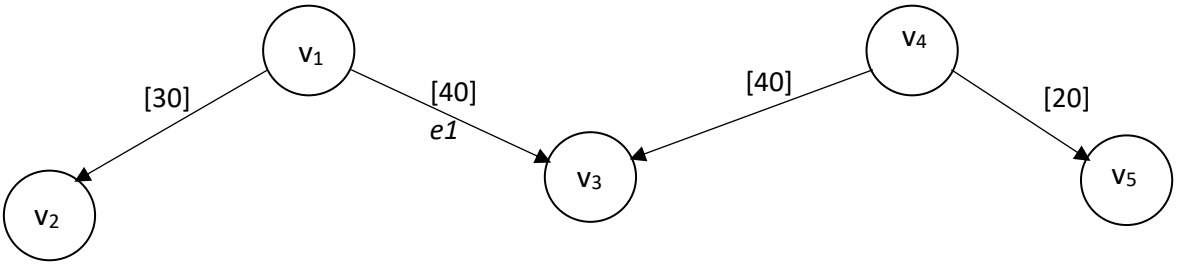
Rysunek 1.1

Analizując postawione warunki można dojść do wniosku, że znalezienie rozwiązania optymalnego polega na wyznaczeniu takich ścieżek, które są wierzchołkami rozłączne oraz razem obejmują wszystkie wierzchołki grafu, a suma wag krawędzi należących do ścieżek jest największa. Co więcej, dzięki temu, że znajdujemy ścieżki w grafie, które są wierzchołami rozłączne, na pewno spełnione zostaną warunki poprawnego rozwiązania z poprzedniej strony.

(V, E) – rozważany graf.
S_o – zbiór ścieżek wyznaczających optymalne rozwiązanie.
$s_i = (v_{i0}, v_{i1}, v_{i2}, \ldots, v_{im})$, $s_i \in S_o$ - i-ta ścieżka.
$s_1 \cap s_2 \cap \ldots \cap s_n = \emptyset$ oraz $s_1 \cup s_2 \cup \ldots \cup s_n = S$



Przeanalizujmy możliwe wybory z perspektywy wierzchołka, z którego wychodzi kilka krawędzi (np. v_1 na rysunku poniżej). Intuicja podpowiada, aby wybrać krawędź z największą wagą – zaoszczędzimy w ten sposób najwięcej objętości. Oznaczmy tę krawędź jako e_1. Trzeba jednak spojrzeć na ten wybór z perspektywy wierzchołka, do którego wchodzi ta krawędź. Nazwijmy ten wierzchołek v_3. Być może do wierzchołka v_3 wchodzi kilka krawędzi. Wtedy wybierając krawędź e_1, uniemożliwiamy wybranie jakiejkolwiek innej krawędzi wchodzącej do v_3, zmuszając w ten sposób wybór alternatywnej ścieżki dla wierzchołków połączonych z v_3. W złośliwym przypadku doprowadzi to do nieprawidłowego rozwiązania. Dlatego należy wprowadzić dodatkowy warunek w przypadku, gdy rozważana krawędź prowadzi do wierzchołka, do którego wchodzi wiele krawędzi.



W powyższym przypadku, wybór krawędzi (v_1, v_3) wymusi wybranie krawędzi (v_4, v_5), co prowadziłoby do rozwiązania nieoptymalnego. Wybrana powinna zostać krawędź (v_1, v_2).

4. Opis algorytmu.

Najpierw należy ustalić kolejność, w jakiej będziemy analizowali wierzchołki i wychodzące z nich krawędzie. Sortując je malejąco po prostu ze względu na ich objętość, otrzymamy listę wierzchołków, którą na pewno rozpoczyna wierzchołek bez wchodzących do niego krawędzi (rodziców). Dla każdego wierzchołka z listy (po kolei) wykonujemy następujące kroki. Tworzymy nową, pustą ścieżkę *s*. Zapisujemy wierzchołek w ścieżce *s* i usuwamy go listy wierzchołków . Następnie znajdujemy najbardziej kosztowną krawędź wychodzącą z tego wierzchołka (krawędź prowadzi do dziecka). Jeśli wierzchołek nie ma żadnego dziecka, to znaczy że jest końcem ścieżki – rozpoczynamy algorytm dla kolejnego wierzchołka z listy.
(a)Dla znalezionego dziecka sprawdzamy, czy ma on więcej niż jednego rodzica. Jeśli nie, dodajemy dziecko do ścieżki *s*, usuwamy je z listy wierzchołków i powtarzamy algorytm dla dziecka. Jeśli tak, to dla każdego rodzica dziecka (nazwijmy v_k) innego niż analizowany obecnie wierzchołek sprawdzamy, jaka jest maksymalna waga wśród dzieci wierzchołka v_k (poza analizowanym dzieckiem*)* i spośród tych wartości zapamiętujemy najmniejszą. Jeśli ta zapamiętana wartość jest mniejsza niż maksymalna wartość pozostałych dzieci analizowanego wierzchołka, usuwamy najbardziej kosztowną krawędź wychodzącą z analizowanego wierzchołka i znajdujemy kolejną największą krawędź, dla której powtarzamy (a). Jeśli zaś ta wartość jest większa lub równa, dodajemy dziecko do ścieżki *s*, usuwamy je z listy wierzchołków i powtarzamy algorytm dla dziecka.
Kończymy algorytm, gdy lista wierzchołków stanie się pusta.

Wszelkie nieścisłości i wątpliwości powinien rozwiać bardziej zwięzły opis:

0. Stwórz listę wierzchołków grafu posortowaną malejąco ze względu na objętość oraz zbiór znalezionych ścieżek.
1. Dla każdego wierzchołka *v* z listy:
    a. Stwórz nową ścieżkę s.
    b. Wykonaj:
        i. Dodaj *v* do ścieżki *s*.
        ii. Znajdź wierzchołek v_k, do którego prowadzi najbardziej kosztowna krawędź z *v*. Jeśli wierzchołek *v* nie ma dzieci, usuń *v* z listy wierzchołków oraz *v* i krawędzie z nim związane z grafu. Powróć z rekurencji lub zakończ punkt b.
        iii. Jeśli v_k ma tylko jednego rodzica, usuń *v* z listy wierzchołków oraz z grafu wraz z krawędziami z nim związanymi i wykonaj pkt. b. dla v_k (rekurencja).
        iv. Jeśli v_k ma więcej niż 1 rodzica, dla każdego jego rodzica v_r znajdź wartość najdroższej krawędzi z v_r nieprowadzącej do v_k (0 jeśli nie ma). Spośród znalezionych wartości wybierz najmniejszą = x_{min}. Znajdź najdroższą krawędź wychodzącą z *v* nieprowadzącą do v_k i zapamiętaj jako x_{alt} (0 jeśli nie ma). Jeśli $x_{alt} > x_{min,}$ usuń krawędź (v, v_k) z grafu i powtórz pkt. ii. W przeciwnym wypadku usuń *v* z listy wierzchołków oraz *v* i krawędzie z nim związane z grafu. Wykonaj pkt. i. dla v_k (rekurencja)
    c. Dodaj ścieżkę *s* do zbioru znalezionych ścieżek.
2. Jeśli lista wierzchołków jest pusta, zakończ algorytm. Zbiór ścieżek wyznacza zbiór *W'*, który jest rozwiązaniem zadania.

5. Implementacja.

Algorytm zostanie zaimplementowany w języku C++. Na wejściu będzie można podać nazwę pliku z wymiarami kolejnych kartonów, bądź uruchomić program z opcją automatycznego generowania określonej liczby losowych kartonów. Wyjściem programu będzie wypisanie znalezionych ścieżek oraz informacja o całkowitej sumie objętości kartonów i objętości zaoszczędzonego miejsca.
Testy wydajnościowe zostaną przeprowadzone dla losowych danych z licznością elementów: 10, 20, 30, 50, 100, 200, 300, 500, 1000, 2000, 4000, 8000 (lub nieco inne, możliwa zmiana). Dodatkowo zostaną przeprowadzone testy przypadków trudnych (takich jak graf dwudzielny pełny) z licznością elementów umożliwiającą sprawdzenie rozwiązania. Dodatkowo, dla testów przewiduję m. in. tworzenie wykresów i tym podobnych zgodnie z wymaganiami projektowymi.