Algorytm pakujący kartony

Kacper Biegajski, AAL 19Z

➔ Problem

Ortodoksyjny kolekcjoner kartonów zaczyna narzekać na brak miejsca. Postanowił oszczędzić miejsce przez wkładanie kartonów jeden w drugi. W trosce o zachowanie dobrego stanu kartonów wkłada tylko jeden karton wewnątrz większego, a wolną przestrzeń wypełnia materiałem ochronnym. Tak zabezpieczony karton umieszcza wewnątrz innego większego kartonu. Nie może schować 2 kartonów obok siebie w innym kartonie. Dla danego zbioru kartonów znaleźć najlepsze upakowanie, czyli zwalniające najwięcej miejsca.

➔ Algorytm

Celem jest minimalizacja sumy objętości kartonów będących na zewnątrz.

Algorytm zachłanny od największego do najmniejszego

- Sortujemy kartony po ich objętości od największego do najmniejszego.
- Bierzemy największy dostępny karton, traktujemy go jako aktywny, wkładamy jako pierwszy element stosu i usuwamy z listy kartonów.
- Przechodzimy na następny w kolejności karton.
- Jeżeli dany karton mieści się w aktywnym kartonie, czyli ma wszystkie krawędzie mniejsze, to wkładamy go jako kolejny element stosu, traktujemy jako aktywny i usuwamy z listy kartonów.
- Powtarzamy punkty 3 i 4 aż dojdziemy do końca tablicy.
- Zamykamy stos.
- Jeżeli lista kartonów nie jest pusta to wracamy do pkt. 2.
- Zapisane stosy tworzą upakowanie kartonów.

Algorytm zachłanny od najmniejszego do największego (reverse)

- Sortujemy kartony po ich objętości od najmniejszego do największego.
- Bierzemy najmniejszy dostępny karton, traktujemy go jako aktywny, wkładamy jako pierwszy element stosu i usuwamy z listy kartonów.
- Przechodzimy na następny w kolejności karton.
- Jeżeli w danym kartonie mieści się aktywny karton, czyli ma wszystkie krawędzie mniejsze, to wkładamy dany karton jako kolejny element stosu, traktujemy jako aktywny i usuwamy z listy kartonów.
- Powtarzamy punkty 3 i 4 aż dojdziemy do końca tablicy.
- Zamykamy stos.
- Jeżeli lista kartonów nie jest pusta to wracamy do pkt. 2.

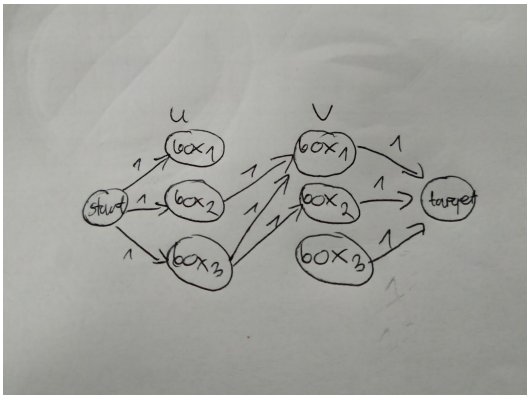
- Zapisane stosy tworzą upakowanie kartonów.

Algorytm znajdujący upakowanie z najmniejszą liczbą kartonów na zewnątrz (z wykorzystaniem metody Forda-Fulkersona) (flow):

- Sortujemy kartony po ich objętości od największego do najmniejszego.
- Tworzymy graf skierowany do obliczenia maksymalnego przepływu (przepływ na danej krawędzi oznacza, że karton mieści się w innym)
    - tworzymy wierzchołki start i target
    - tworzymy wierzchołki kartonów (każdy karton jest reprezentowany przez wierzchołek u i v, będziemy sprawdzać czy kartony u mieszczą się w kartonach v)
    - łączymy krawędziami wierzchołek start z wszystkimi wierzchołkami u, nadajemy im maksymalną przepustowość 1
    - łączymy wszystkie wierzchołki v z wierzchołkiem target, nadajemy im maksymalną przepustowość 1
    - łączymy każdy wierzchołek u z wierzchołkami v, jeżeli karton reprezentowany przez u mieści się w v, nadajemy im maksymalną przepustowość 1
- Znajdujemy maksymalny przepływ w grafie metodą Forda-Fulkersona (do każdego wierzchołka u wchodzi krawędź z przepływem 1, co oznacza że może wejść do środka jednego kartonu v, tak samo od każdego wierzchołka v wychodzi jedna krawędź o przepływie 1, co oznacza że tylko jeden karton mógł wejść do niego)
- Odtwarzamy upakowanie kartonów z grafu (sprawdzamy, które kartony nie wchodziły do innych i znajdujemy sekwencję kartonów w nich)

Maksymalny przepływ oznacza ilość wykonanych upakowań kartonu w karton, im więcej upakowań, tym mniej kartonów na zewnątrz.

Przykładowy graf dla box1, box2, box3, gdzie box1 jest największy, box2 mieści się w box1, a box3 mieści się w obu wcześniejszych kartonach.



➔ Optymalność

- ◆ żaden z algorytmów nie daje gwarancji optymalnego rozwiązania
- ◆ algorytm zachłanny od największego do najmniejszego
  - kontrprzykład: A(5,5,5), B(3,3,6), C(2,2,4), D(4,4,1/2)
  - znalezione rozwiązanie: AC, B, D
  - optymalne rozwiązanie: AD, BC
- ◆ algorytm zachłanny od najmniejszego do największego
  - kontrprzykład: A(4.6, 1.8, 1.3), B(4.9, 2.0, 1.4), C(5.2, 5.2, 5.0), D(4.5, 3.7, 3.4)
  - znalezione rozwiązanie: ABC, D
  - optymalne rozwiązanie: AB, DC
- ◆ algorytm z maksymalnym przepływem
  - kontrprzykład: A(5.6, 5.6, 4.3), B(4.5, 3.9, 1.7), C(4.2, 2.4, 2.0), D(4.1, 1.7, 1.1)
  - znalezione rozwiązanie: ACD, B
  - optymalne rozwiązanie: ABD, C
- ◆ testy algorytmów
  - porównywanie algorytmów do algorytmu od największego do najmniejszego (na tych samych danych)
  - test dla 10000 losowych instancji przy wielkości instancji 7
    - reverse był lepszy 1034 razy (ok. 10%), a niegorszy 4055 (40%) razy
    - flow był lepszy 509 razy (ok. 5%), a niegorszy 6006 (60%) razy
  - test dla 10000 losowych instancji przy wielkości instancji 15 (test trwał o wiele dłużej, ze względu na algorytm flow)
    - reverse był lepszy 1016 razy
    - flow był lepszy 1164 razy
  - test dla 10000 losowych instancji przy wielkości instancji 100
    - reverse był lepszy 29 razy (0.3%) i niegorszy tyle samo razy
    - flow przy takim rozmiarze instancji wykonuje się zbyt długo by przeprowadzić test
- ◆ najlepszy wydaje się być algorytm zachłanny od największego do najmniejszego, przeważnie daje najlepszy wynik i w rozsądnym czasie
- ◆ można rozważyć wykonywanie wszystkich 3 algorytmów dla pojedynczych instancji i wybranie najlepszego upakowania

➔ Szczegóły implementacyjne

- ◆ język: Python wersja 3.7

- użyte struktury danych
  - list (zbiór wszystkich kartonów, stosy upakowanych kartonów)
  - tuple (reprezentacja kartonu (x, y, z, V))
- algorytmy pomocnicze
  - Timsort (wykorzystywany w pythonowej funkcji sort())
  - Forda-Fulkersona (w algorytmie flow)

➔ Złożoność (dla algorytmu od największego do najmniejszego)

- spodziewana $T(n) = O(n^2)$
  Algorytm ma dwie pętle (jedna zagnieżdżona w drugiej), które w pesymistycznym wypadku wykonują po n iteracji (dla każdego kartonu będziemy musieli sprawdzić czy pozostałe mniejsze kartony zmieszczą się do środka)
- wyniki pomiarów czasowych (3 tryb)

$$q(n) = \frac{t(n)\ T(n_{\ mediana})}{t(n_{\ mediana})\ T(n)}$$

n	t(n) [s]	q(n)
1000	0.23313040733337403	1.0145771678283924
2000	0.9089640617370606	0.9889466950746395
3000	2.7007328987121584	1.3059462044720316
4000	3.587327241897583	0.9757468885090217
5000	5.380461263656616	0.9366248209200687
6000	7.8987349510192875	0.9548633015813103
7000	10.618611383438111	0.9431001394533345
8000	14.129860830307006	0.9608245087883721
9000	18.397417736053466	0.9884575272118111
10000 (mediana)	22.978085327148438	1.0
11000	26.560509347915648	0.9552943101789412
12000	31.368974876403808	0.9480341821647921
13000	38.317931985855104	0.986737241732694
14000	50.054562997817996	1.1114086305377455

15000	53.544173908233645	1.0356568133100783
16000	61.122921562194826	1.039083147934126
17000	68.02258052825928	1.0243336100973754
18000	73.70878634452819	0.9900574869960693
19000	88.10103387832642	1.062086687310635

Test był przeprowadzany dla danych trudnych (pesymistycznych). q(n) dla wszystkich n oscyluje w okolicach 1, co oznacza że spodziewana złożoność została dobrze oszacowana.