Dokumentacja końcowa
AAL - Analiza algorytmów

Treść:

Dany jest zbiór sal wykładowych oraz zbiór zamówień, określających czas rozpoczęcia i zakończenia wykładu. Ułożyć plan wykorzystania sal, akceptując pewne wykłady i odrzucając inne, tak aby sumaryczny czas wykorzystania sal był jak najdłuższy. Porównać czas obliczeń i wyniki różnych metod.

Struktury danych:

Używane struktury ze standardowej biblioteki:

- std::vector
- std::list
- std::map

Własne struktury:

- *Model* - zawierający wektor sal i zamówień celem zarządzania nimi
- *Classroom* - zawierające ID w postaci ciągu znaków, identyfikujące daną salę wykładową, całkowitą zajętość sali liczoną w trakcie dodawania nowych zajęć, oraz listę przyporządkowanych do niej wykładów
- *Order* - prosta klasa zawierająca ID w postaci ciągu znaków, identyfikująca dane zamówienie, czas rozpoczęcia i czas zakończenia wykładu
- *Result* - klasa przechowująca mapę sal wykładowych z kluczem jako ID danej sali oraz wartością jako jej instancją dla danego rozwiązania (tj. ze znalezioną listą zamówień). Wykorzystywana jest do trzymania całkowitego czasu wykorzystania sal w danym rozwiązaniu oraz prezentacji wyników w postaci planu zajęć jako pliku *.html*
- *PartContainer* - struktura pomocnicza, używana w ostatnim algorytmie, do podziału wektora czasu na nakładające się wykłady. Zawiera początek i koniec okresu nakładających się zajęć oraz maksymalną ilość występujących wykładów w jednej godzinie = ilość potrzebnych sal do ułożenia wszystkich wykładów z tej części.

Algorytmy:

Pomocnicze algorytmy ze standardowej biblioteki użyte w programie:

- std::sort
- std::find
- std::remove

Metody rozwiązania (n – liczba zamówień, k – liczba sal wykładowych):

1. Algorytm programowania dynamicznego:

Opis:

Dla każdej sali przyporządkowanie dla niej optymalnego jej wykorzystania używając wykładów nie umieszczonych w salach już rozważonych. Działanie optymalne dla liczby sal = 1, dla większej ilości sal możliwe przypadki znalezienia nieoptymalnego wyniku dla całego rozważanego problemu.

Działanie:

Zajęcia sortowane są wg czasu zakończenia *~(n * log(n))*. Dla każdej sali *(k * (..)* sprawdzamy optimum czasowe gdybyśmy przypisali wykład *(.. * n + ..)* do rozważanej sali. W tym celu należy porównać dwie liczby a i b, gdzie a jest czasem trwania rozważanego wykładu, b jest sumą czasu trwania rozważanego wykładu i największego wykorzystania sali uzyskanego po dołączeniu wcześniejszych wykładów, który znajdujemy spośród wykładów kończących się nie później niż w chwili rozpoczęcia rozważanego wykładu. Ułożenie zajęć w następnych salach następuje z wyłączeniem już wstawionych zajęć do sal poprzedzających. Następnie dla najlepszego wyniku wykorzystania sali usuwane są *(.. + (n * ..)* użyte wykłady *(.. * n))*.

Przybliżona spodziewana złożoność obliczeniowa:

$$T_{(n)} \approx n \cdot \log n + 2 \cdot k \cdot n^2$$

$$O_{(T_{(n)})} \approx k \cdot n^2$$

2. Algorytm zachłanny:

Opis:

Wykonywany równolegle dla wszystkich sal. Nie gwarantuje to optymalnego układu, aczkolwiek zapewnia szybkie działanie.

Działanie:

Zajęcia posortowane wg czasu zakończenia *~(n * log(n))*. Następnie przechodząc po kolejnych wykładach *(.. + n * ..)* dopasowanie do sali nr. 1, jak nie pasuje (nakłada się) to do sali nr. 2 itd. *(.. * k)*.

Przybliżona spodziewana złożoność obliczeniowa:

$$T_{(n)} \approx n \cdot \log n + n \cdot k$$

$$O_{(T_{(n)})} \approx n \cdot \log n$$

3. Metoda *Brute Force*:

Opis:

Sprawdzenie wszystkich możliwych kombinacji i wybranie najlepszej możliwej. Dla dużych ilości zamówień i sal narzut czasowy obliczeń jest ogromny.

Działanie:

Rekurencyjnie sprawdzamy kolejne możliwości ulokowania zajęć, odrzucając kombinacje z nakładającymi się zajęciami jako nie-akceptowalne rozwiązanie, co zmniejsza ilość możliwości.

Przybliżona spodziewana złożoność obliczeniowa:

$$T_{(n)} \approx k + n + T_{(n-1,k)} + T_{(n,k-1)}$$

$$O_{(T_{(n)})} \approx ? \ (nie\ udało\ mi\ się\ oszacować)$$

4. Połączenie ww. algorytmów:

Opis:

Używane na mniejszych pod-problemach. Algorytm w niektórych przypadkach łańcuchowego nakładania się wykładów, sprowadza się do działania jednego w ww. podejść.

Działanie:

Tworzymy wektor, o długości *24 * liczba dni (5),* odpowiadający kolejnym godzinom. Do każdej godziny *(c * ..)* tworzymy listę do której przyporządkowujemy informację o wykładzie odbywającym się w tym czasie *(.. * n + ..)*. Przechodząc po wektorze *(.. + c * (..)* dzielimy go na części z odrębnie nakładającymi się (lub nie) wykładami *(.. n + ..)* i dodajemy unikalne wykłady do pomocniczej struktury *(.. + n * n) + ..)*. Jeżeli maksymalna długość listy jest mniejsza równa ilości dostępnych sal, to można przyporządkować sale po kolei *(n)*. Jeżeli nie to dla skupisk *(.. + c * ..)* nakładających się sal w miejscach gdzie nakłada się więcej zajęć niż jest sal *(.. * k * n * Talg)* wykonujemy jeden z wyżej wymienionych algorytmów w celu znalezienia odpowiedniego układu:

- Dla liczby nakładających się zajęć < k: - algorytm zachłanny
- Dla n < 10 i k < 5: - metoda *Brute Force*
- w p.p.: - algorytm programowania dynamicznego

Przybliżona spodziewana złożoność obliczeniowa:

$$T_{(n)} \approx c \cdot n + c \cdot (n + n \cdot n?) + c \cdot n \cdot k \cdot Talg_{(n)}$$

$$O_{(T_{(n)})} \approx k \cdot n^2$$