Algorytmy i Struktury Danych
Lista zadań laboratoryjnych
2024/2025 r.

Norbert Jankowski, Miłosz Michalski

http://www.is.umk.pl/~norbert/asd/lab-zadania.pdf

Zadania 1–16 pomagają w opanowaniu podstawowych technik programowania z użyciem dynamicznych struktur danych, takich jak listy liniowe i cykliczne jedno- i dwukierunkowe, kolejki oraz drzewa. Realizując zadania dla np. jednokierunkowej listy liniowej należy stworzyć jeden program, który w bloku głównym zawiera dialogowe menu pozwalające użytkownikowi na cykliczny wybór kolejnej operacji do wykonania na liście:

```
Nastepna operacja: _
1 - dodaj element na poczatku listy
2 - dodaj element na koncu listy
3 - usun pierwszy element listy
4 - usun ostatni element listy
5 - odszukaj zadany element
6 - dodaj nowy element przed lub za wskazanym
7 - usun wskazany element
8 - wczytaj zawartosc listy z pliku
9 - zapisz zawartosc listy do pliku
10 - wyswietl zawartosc listy
0 - zakoncz dzialanie programu
```

Poszczególne operacje implementowane są jako funkcje biblioteczne i wywoływane z pętli dialogowej w bloku głównym. Inne specyficzne operacje, np. porównanie lub połączenie dwóch list (p. zad. 7 i 8), powinny być zaimplementowane także jako funkcje i włączone jako nowe operacje w menu. Funkcja odczytu zawartości listy z pliku jest istotna (!), ponieważ pozwoli ona nam na sprawne testowanie działania programu na specjalnie przygotowanych danych. Liczby zapisywane do pliku mają być oddzielone spacjami. Wczytywanie zakłada, że liczby będą oddzielone znakami białymi (spacja, tabulacja, znak nowego wiersza). Realizacja dalszych zadań, np. dla list dwukierunkowych lub cyklicznych, polegać będzie po prostu na odpowiedniej modyfikacji kodów funkcji napisanych wcześniej.

1. Opracować bibliotekę podstawowych funkcji operowania na jednokierunkowej liście prostej dodawanie na początek i na koniec, usuwanie pierwszego i ostatniego elementu, szukanie elementu, dodawanie nowego elementu przed i za znalezionym, usuwanie znalezionego, wyświetlanie listy od początku i od końca. Zbudować program wg schematu opisanego wyżej. Lista jako dane zawiera liczby całkowite i jest nieposortowana.

   Biblioteka powinna mieć podział na pliki `.c` i `.h`. To dotyczy także kolejnych zadań.

2. Uzupełnić program o funkcję usuwającą z listy wszystkie wystąpienia wskazanej wartości x w wersji nierekurencyjnej i rekurencyjnej.

3. Uzupełnić program o funkcję wyznaczającą wartość najczęściej występującą na liście.

4. Zbudować funkcję `usunniepodz(l, k)`, która w zadanej argumentem `l` liście usunie wszystkie liczby niepodzielne przez wartość `k`.

5. Uzupełnić bibliotekę z zadania 1 o procedurę odwracania listy bez generowania nowych elementów (tj. wyłącznie przez manipulacje na wskaźnikach). Zaproponować oddzielnie rozwiązania iteracyjne i rekursywne.

6. Korzystając z tych samych struktur danych co w zadaniu 1, tworzymy teraz listę posortowaną rosnąco. Repertuar operacji na takiej liście obejmuje dodawanie nowego elementu (wstawiany jest do listy zgodnie z porządkiem), pobieranie pierwszego i ostatniego elementu, wyszukiwanie zadanego elementu i usuwanie wyszukanego. Operacje muszą uwzględniać, że lista jest posortowana. Zapis i odczyt z pliku oraz wyświetlanie listy bez zmian. Zrealizować wersję listy z wartownikiem i bez wartownika.

7. Korzystając z procedur zad. 6 utworzyć dwie oddzielne listy uporządkowane z dwóch serii danych, a następnie napisać funkcję, która dokona połączenia dwóch list w jedną listę uporządkowaną i zwróci wskaźnik na tę połączoną listę. Uwaga: "dobre" rozwiązanie polega na przenoszeniu całych serii elementów list, gdy to tylko możliwe.

8. Podobnie jak w zadaniu 7, lecz tym razem procedura ma porównać dwie listy uporządkowane. Powinna ona zwrócić dwie listy elementów, które mają składać się odpowiednio z elementów pierwszej listy nie należących do drugiej i odwrotnie. W przypadku, gdy listy są równe, wówczas listy wynikowe będą puste.

9. Powtórzyć zadanie 1 dla liniowej listy dwukierunkowej nieposortowanej oraz zadanie 6 dla takiej listy posortowanej rosnąco.

10. Powtórzyć zadanie 1 dla nieposortowanych list cyklicznych a) jednokierunkowej i b) dwukierunkowej.

11. Dla listy dwukierunkowej cyklicznej startując z wybranego elementu (pierwszego dodanego elementu) usuwaj co k-ty element, aż na liście zostanie jedna wartość, która będzie stanowić wynik zadania.

12. Napisz funkcję, która dla dwóch nieposortowanych jednokierunkowych list cyklicznych usunie z pierwszej z nich wszystkie wartości występujące w drugiej.

13. Zaimplementuj kolejkę priorytetową. Potrzebne będą następujące funkcje: dodaj nowy element; pobierz z listy element o największym PRIORYTECIE (usuwając go jednocześnie z listy); zmień wartość PRIORYTETU wskazanego elementu.

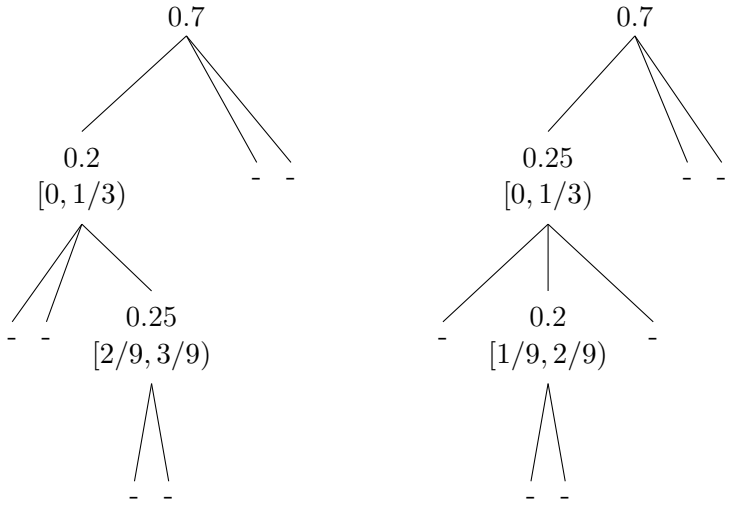
14. Zaimplementuj operacje na drzewie binarnym łańcuchów znakowych z porządkiem: dodawanie, usuwanie, szukanie, wyznaczanie minimum i maksimum, wyznaczanie poprzednika i następnika wskazanego elementu (można korzystać ze wskaźnika na węzeł rodzica). Uzupełnij program o procedurę kontrolnego drukowania zawartości utworzonego drzewa z ukazaniem jego struktury.

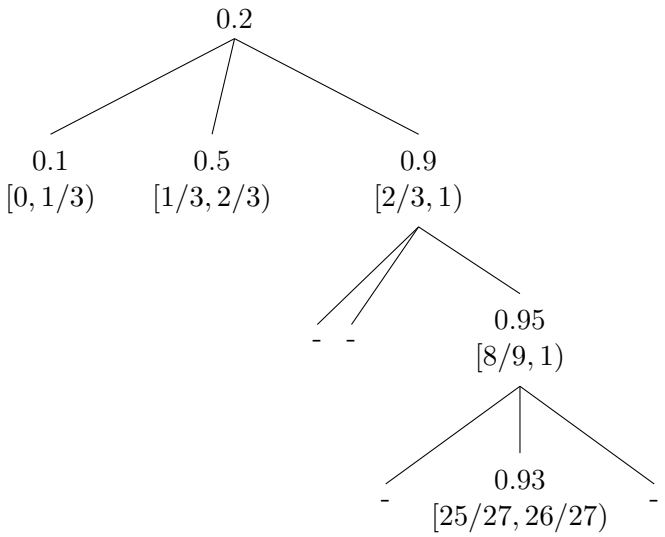
15. Uzupełnij poprzednie zadanie o procedurę zapisu utworzonego drzewa z porządkiem do pliku i funkcję jego odczytu. Odczytanie pliku ma automatycznie odtworzyć strukturę drzewa bez konieczności ponownego porównywania wczytywanych elementów.

16. Zaimplementuj funkcję, która sprawdzi czy dwa drzewa binarne z porządkiem składają się z dokładnie takich samych wartości (z takich samych ciągów liczb). Postaraj się aby funkcja działała jak najszybciej. UWAGA: drzewa mogą mieć zupełnie różne struktury, mimo że zawierają te same elementy.

17. Stwórz funkcje do obsługi drzewa przypominającego działanie katalogów w systemach operacyjnych. Węzły–katalogi przyjmują nazwy (łańcuchy znakowe). W węźle można mieć inne węzły (katalogi, pliki), tak więc potrzebne jest: dodawanie, usuwanie, szukanie, wyświetlanie zawartości węzła, ...

18. 3-drzewo to drzewo, które ma przechowywać punkty na odcinku $[0, 1)$. Każdy węzeł drzewa może mieć 3 poddrzewa. Każdy węzeł przechowuje jedną wartość, która musi odpowiadać przedziałowi węzła. Węzeł korzeń odpowiada za cały przedział $[0, 1)$, więc może przechowywać dowolną jedną wartość z tego przedziału. Gdy do drzewa zechcemy włożyć drugą wartość, trzeba będzie dodać odpowiedni nowy węzeł pod węzłem korzenia. Każdy podwęzeł odpowiada za $1/3$ przedziału rodzica, tak więc dzieci korzenia odpowiadają kolejno za przedziały: $[0, 1/3)$, $[1/3, 2/3)$, $[2/3, 1)$ (podobnie dla wnuków, etc.). Załóżmy, że do drzewa dodajemy wartości: 0.7, 0.2, 0.25. Wtedy 0.7 znajduje się w korzeniu. 0.2 w lewym podwęźle korzenia a 0.25 w prawym podwęźle dziecka korzenia (por. rys. poniżej, po lewej). Natomiast, gdy do drzewa włożymy wartości 0.7, 0.25, 0.2, to powstanie nieco inne drzewo (por. rys. poniżej, drzewo środkowe). Kolejny przykład: 0.2, .5, 0.1, 0.9, 0.95, 0.93 (drzewo po prawej stronie poniższego rysunku). Zaimplementuj funkcję dodającą wartości do drzewa i funkcję szukającą zadanych wartości w drzewie.





19. Zaimplementować z pomocą stosu kalkulator wyznaczający wartość wyrażeń zapisanych w odwrotnej notacji polskiej (ONP). Zakładamy, że wyrażenia mogą składać się z liczb całkowitych, operatorów $+$, $-$, $*$, $/$, a także operacji `neg` (minus jednoargumentowy).

    Przykłady wyrażeń:
    ```
    3 4 + 7 *
    4 neg 5 2 + neg *
    ```

20. Wczytać wyrażenie ONP i zbudować binarne drzewo tego wyrażenia. Wyznaczyć jego wartość za pomocą rekursywnej funkcji działającej na tym drzewie (składnia wyrażeń jak w zadaniu 19).

21. Porównanie wydajności algorytmów sortowania. Napisać osobne procedury sortowania metodami: bąbelkową, przez wstawianie, metodą Shella (na bazie sortowania bąbelkowego) oraz quick sort. W każdej z nich należy stosownie osadzić instrukcje zliczania wykonanych podstawień i porównań sortowanych danych. Program generuje losowo długą, testową serię liczb ($N \sim 10\,000$) i kolejno poddaje je (te same losowe dane!) sortowaniu wszystkimi wymienionymi metodami, rejestrując dla każdej z nich liczbę wykonanych porównań i podstawień kluczy. Cykl ten powtarzany jest $M \sim 1000$ razy, dla nowych losowych danych. Dla każdej z wymienionych metod program powinien wyświetlić w tabeli następujące dane: maksymalna i minimalna w serii M prób liczba wykonanych podstawień i porównań sortowanych kluczy oraz średnia liczba podstawień i porównań.

22. Zaimplementuj sortowanie przez kopce dla łańcuchów znakowych (zakładamy porządek leksykograficzny).

23. Zaimplementuj sortowanie przez zliczanie dla wartości naturalnych od 1 do k. k jest argumentem funkcji sortującej, podobnie jak dane do posortowania. Korzystając z zadania 21 porównaj szybkość algorytmu z algorytmem quicksort i shell.

24. Zaimplementuj drzewa zbalansowane AVL. Zadanie sprowadzić do zmiany funkcji dodających i usuwających element z zadania 14.

25. Zaimplementuj algorytm wyznaczania najkrótszych ścieżek — Dijkstry lub Bellmana–Forda. Implementacja ma być oparta na rozwiązaniach z poprzednich zadań dotyczących list (należy

stworzyć tablice list sąsiedztwa dla reprezentacji grafu).

Progi zaliczeniowe (minimalna liczba zadań na daną ocenę):

Ocena	Min liczba zadań
3.0	12
3.5	14
4.0	16
4.5	18
5.0	20

Poprawne wykonanie zadań 1, 2, 5, 6, 9, 10, 12, 14, 16, 21, 22, 25 jest wymagane do zaliczenia laboratorium. Pozostałe zadania nie są obowiązkowe, ale są źródłem dodatkowych punktów na wyższą ocenę. Szczegóły u prowadzącego laboratorium.

UWAGA: Zadania 7, 8 oraz 17–21 mogą okazać się (nieco) czasochłonne.