Listy i drzewa

1. (*) Napisz funkcję dublującą elementy danej listy jednokierunkowej/dwukierunkowej.
2. Napisz funkcję, która daną listę jednokierunkową przekształca w listę cykliczną, ale z odwróconym kierunkiem wskaźników. Możesz założyć, że dana lista jest poprawną listą jednokierunkową, to znaczy ma koniec i nie zapętla się.
3. (*) Napisz funkcję, która mając dane dwie listy jednokierunkowe, splata je w jedną listę postaci: pierwszy rekord pierwszej listy, pierwszy rekord drugiej listy, drugi rekord pierwszej listy, drugi rekord drugiej listy, itd. Jeśli jedna z list jest dłuższa, to jej końcówka stanowi końcówkę listy wynikowej.
4. Zaimplementuj typ dwukierunkowej listy cyklicznej. Napisz funkcję tworzącą taką listę na podstawie wektora wartości.
5. (*)[PCh] Dane są dwie listy jednokierunkowe (bez zapętleń), które łączą się i od pewnego miejsca są tą samą listą. Znajdź ten wspólny fragment.
6. Napisz procedurę sprawdzającą, czy dana lista zawiera cykl. Czy potrafisz wyznaczyć jego długość?
7. (*) Zdefiniuj typ reprezentujący drzewa o wierzchołkach dowolnego (skończonego) stopnia. Zdefiniuj garść procedur operujących na takich drzewach (np. głębokość, liczbę elementów, wektor elementów w porządku prefiksowym/postfiksowym).
8. (*)[II OI, zadanie *Obchodzenie drzewa skokami*] Jak obejść wszystkie wierzchołki drzewa ogólnego (odwiedzając każdy wierzchołek dokładnie raz) tak, aby odległość między kolejnymi wierzchołkami nie przekraczała 3. Zimplementuj rozwiązanie (dla drzew binarnych lub ogólnych).
9. Napisz funkcję, która dla danego drzewa binarnego, wyznaczy wektor wartości znajdujących się w liściach.
10. (*) Dana jest definicja typu drzew binarnych:
    ```
    typedef struct node *bin_tree;
    struct node {
        int val;
        bin_tree left, right;
    };
    ```
    Drzewo nazwiemy *ultralewicowym* jeśli głębokości kolejnych liści (od lewej do prawej) tworzą ciąg nierosnący. Napisz funkcję `bool ultraleft(bin_tree t)`, która sprawdza, czy dane drzewo jest ultralewicowe.
11. Dana jest definicja typu drzew binarnych:
    ```
    typedef struct node *bin_tree;
    struct node {
        int val;
        bin_tree left, right;
    };
    ```
    Napisz funkcję `bin_tree bestify(bin_tree t)`, która dla danego drzewa binarnego poprawia je tak, że spełnia ono słabszą wersję warunku BST: dla dowolnego węzła, wartosc w jego lewym synie nie jest większa, a w jego prawym synie nie jest mniejsza, niż w danym węźle. Twoja funkcja nie powinna zmieniać wartości w poszczególnych węzłach, tylko zmieniać sposób w jaki są one połączone. Wynikiem funkcji powinien być wskaźnik na korzeń drzewa po dokonanych zmianach.
12. (*) Dana jest definicja typu drzew binarnych:
    ```
    typedef struct node *bin_tree;
    struct node {
        int val;
        bin_tree left, right;
    };
    ```

?

Napisz funkcję `bin_tree rozcięcie(bin_tree t)`, która znajduje w danym drzewie taką krawędź, po której usunięciu drzewo rozpada się na dwa drzewa o minimalnej różnicy liczby węzłów. Krawędź tę należy usunąć. Wynikiem funkcji powinno być drzewo poniżej usuwanej krawędzi. (Drzewo powyżej tej krawędzi to przekazane drzewo t.) Jeżeli dane drzewo nie ma krawędzi (ma mniej niż dwa wierzchołki), to poprawnym wynikiem jest `NULL`, a dane drzewo nie jest modyfikowane.

13. [PCh] Dana jest definicja typu drzew binarnych:
```
typedef struct node *bin_tree;
struct node {
    int val;
    bin_tree left, right;
};
```
Napisz funkcję `bool symetryczne(bin_tree t)`, która sprawdzi, czy dane drzewo jest symetryczne ze względu na odbicie lewo-prawo.
14. Dana jest definicja typu drzew binarnych:
```
typedef struct node *bin_tree;
struct node {
    int val;
    bin_tree left, right;
};
```
Zaimplementuj typ reprezentujący *kursor*, którym możemy chodzić po węzłach drzewa i oglądać je:
    - `typedef ... walk;` -- typ kursora,
    - `walk make_walk(bin_tree t); /* t != NULL */` -- utworzenie kursora, znajdującego się w korzeniu drzewa,
    - `void go_left(walk &w);` -- przesuwa kursor do lewego poddrzewa (jeśli istnieje),
    - `void go_right(walk &w);` -- przesuwa kursor do prawego poddrzewa (jeśli istnieje),
    - `void go_up(walk &w);` -- przesuwa kursor do ojca poddrzewa (jeśli istnieje),
    - `int lookup(const walk w);` -- zwraca wartość w węźle, na który wskazuje kursor.
15. Dana jest definicja typu drzew binarnych:
```
typedef struct node *bin_tree;
struct node {
    int val;
    bin_tree left, right;
};
```
Napisz funkcję `bin_tree odchudzanie(bin_tree t)`, która mając dane drzewo binarne, usuwa w nim niektóre poddrzewa tak, aby suma wartości w wynikowym drzewie była jak największa. Możesz usuwać tylko całe poddrzewa. Nie zapomnij o zwalnianiu pamięci. Wynikiem funkcji powinien być wskaźnik na korzeń drzewa, lub `NULL` jeśli całe drzewo zostało usunięte.
16. (*) `typedef struct node *bin_tree;`
```
struct node {
    int val;
    bin_tree left, right;
};
```
Napisz funkcję `bin_tree środek(bin_tree t)`, która w danym drzewie binarnym `t` znajduje taki węzeł, dla którego maksymalna spośród jego odległości od liści jest jak najmniejsza. (Mówimy tu o wszystkich liściach drzewa `t`, a nie tylko tych poniżej danego węzła.) Wynikiem powinien być wskaźnik do takiego węzła.
Co się zmieni, jeżeli będziemy chcieli znaleźć wierzchołek, dla którego minimalna spośród jego odległości do liścii jest jak największa?

17. Dana jest deklaracja drzew binarnych, w których węzłach znajdują się dodatkowe wskaźniki do węzłów drzewa, początkowo równe `NULL`: `typedef struct node *bin_tree;`
```
struct node {
```

```
    int val;
    bin_tree left, right, jump;
};
```

1. Drzewo binarne z fastrygą, to drzewo, w którym każdy węzeł posiada dodatkowy wskaźnik na następny węzeł w porządku infiksowym. (Ostatni węzeł powinien zawierać `NULL`.) Napisz procedurę `void fastryguj(bin_tree t)`, która sfastryguje dane drzewo.
2. Napisz funkcję `void w_lewo(bin_tree t)`, która w każdym węźle drzewa ustawia dodatkowy wskaźnik tak, aby wskazywał na węzeł znajdujący się na tej samej głębokości w drzewie i najbliżej w lewo od danego węzła. W przypadku skrajnie lewych węzłów, wskaźnik powinien być `NULL`.
3. Napisz procedurę `void potomek(bin_tree t)`, która w każdym węźle ustawi dodaktowy wskaźnik na najgłębiej położony węzeł będący jego potomkiem.

Last modified: Monday, 12 December 2022, 12:17 PM