Algorytmy i Struktury danych (2021)

Lista zadań 1

1. Napisz funkcję obliczającą x^n za pomocą $O(\log n)$ mnożeń: (a) z użyciem rekurencji i (b) bez użycia rekurencji. Uwaga: nie możesz używać funkcji `pow, log, exp` itp. a jedynie operator mnożenia. Wskazówka: skorzystaj z faktu, że dla parzystych n zachodzi: $x^n = (x^2)^{n/2}$ lub $x^n = (x^{n/2})^2$.

2. Dana jest funkcja `double f(double)` ciągła, taka że $f(0) < 0 < f(1)$. Napisz program, który metodą bisekcji znajdzie pierwiastek funkcji f (taki x, że $f(x) = 0$). Uwaga: może się zdarzyć, że taki x nie istnieje, więc algorytm powinien znajdować taki x, dla którego $f(x)$ jest najbliżej zera. Warunkiem zakończenia pętli uczyń wykrycie zapętlenia (czyli że końce przedziału przestały się zbliżać do siebie).

3. Shemat Hoernera: (a) Pokaż, że wystarczy dokładnie n mnożeń, aby wyliczyć wartość wielomianu stopnia n, o współczynnikach zawartych w tablicy `a`?

$$W(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

   w ten sposób, że `a[0]`=a_0, `a[1]`=a_1 itd.. Zakładamy, że nie ma dostępu do funkcji `pow(x,n)` obliczającej x^n i potęgowanie należy wykonać za pomocą mnożenia. Nie używamy funkcji z zadania 1.
   (b) Napisz funkcję `double oblicz(double a[], int n, double x)` realizującą twój algorytm. Pamiętaj, że do przechowania wielomian stopnia n ma $n+1$ współczynników więc rozmiar tablicy `a[]` jest o jeden większy od stopnia wielomianu $W(x)$.

4. (a) Jak znaleźć minimum i maksimum n liczb nie używając więcej niż $3n/2$ porównań?
   (b) Napisz funkcję, `void max_min(int a[], int n)`, która zrealizuje ten algorytm i wypisze wartości `max` i `min` na standardowym wyjściu.

5. W jednej linii pliku `a.txt` znajdują się oddzielone spacjami stopień i współczynniki wielomianu $A(x)$ w kolejności: n a_0 a_1 a_2 a_n a w pliku `b.txt` stopień i współczynniki wielomianu $B(x)$. Napisz w C++ program, który odczyta dane z tych plików, obliczy stopień i współczynniki wielomianu $C(x) = A(x)B(x)$, a następnie zapisze je w pliku `c.txt`. jak ilość mnożeń wykonywanych przez twój program zależy od stopni wielomianów $A(x)$ i $B(x)$.

6. Napisz program nie zawierający instrukcji `if` ani `switch` , który policzy ile razy występuje każdy znak `ASCII` w pliku o nazwie podanej jako argument programu (`argv[1]`).

7.
```
struct lnode
{ int key;
  lnode *next;
  lnode(int key0,lnode* next0=nullptr):key(key0),next(next0){}
};
```

   to struktura węzła listy jednokierunkowej, z dodanym konstruktorem. Napisz funkcję:

   (a) `int wypisz(lnode* L)`, która wypisze elementy listy `L` oddzielone spacjami.

   (b) `int suma(lnode* L)`, która obliczy sumę elementów listy `L`.

   (c) `int nty(int n, lnode *L)` której wynikiem jest wartość n-tego elementu listy `L` lub 0, jeśli długość listy jest mniejsza niż n.

   (d) `void insert(lnode *&L, int x)`, która wstawi `x` na początek listy `L`.

   (e) `void insert_after_smaller(lnode *&L, int x)`, która wstawi `x` do listy `L` za wszystkimi elementami od niego mniejszymi.

(f) `void remove(lnode* &L, int x)`, która usunie z listy `L` elementy o wartości `x`.

(g) `void filter(lnode* &L, bool(*cond)(int))`, która usunie z listy `L` klucze `x` dla których `cond(x)==false`.

(h) `void destroy(lnode* &L)`, która usunie wszystkie elementy z listy `L`.

(i) `void reverse(lnode* &L)`, która odwróci kolejność elementów listy `L` zmieniając jedynie zmienną `L` oraz wskaźniki `next` w węzłach.

Algorytmy i Struktury danych (2021)

Lista zadań 2 (tablice, listy, drzewa)

Pomocny przy wykonaniu tej listy jest plik `list.cc` - struktura listy jednokierunkowej, oraz `tree-2020.cc` - drzewa BST - implementacja. Pliki załączone są do maila, ale można też je pobrać z serwisu: `http://panoramx.ift.uni.wroc.pl`. Zapoznaj się z ich zawartością.

1. Ile trzeba porównań, by znaleźć element x w posortowanej tablicy `t` o rozmiarze n. Podaj minimalną wartość gwarantującą sukces i strategię, jak to zrobić. Postaraj się podać wzór ogólny, który pozwoli wyliczyć dokładną wartość dla dowolnego n. Sprawdź go dla $n = 1, ..., 20$.

2. Ile trzeba porównań, by znaleźć element x w nieuporządkowanej tablicy `t` o rozmiarze n. Oblicz wartość średnią i wariancję zakładając, że element x może znajdować się z jednakowym prawdopodobieństwem, pod dowolnym indeksem tablicy.

3. Ile dokładnie potrzeba mnożeń, by obliczyć w standardowy sposób: (a) iloczyn skalarny dwóch wektorów rozmiaru n. (b) Wartość wielomianu stopnia n (schemat Hoernera). (c) Iloczyn dwóch wielomianów stopnia n. (d) Iloczyn dwóch macierzy $n \times n$. (e) Wyznacznik macierzy $n \times n$ (sprowadzenie do postaci trójkątnej przez eliminację Gaussa). Dla każdego przypadku napisz też jakiej klasy $\Theta(n^k)$ są to funkcje. Uwaga: Dzielenie to mnożenie przez odwrotność, więc liczymy je podobnie.

4. Rozważ trzy wersje znajdowania maksimum w tablicy `int maks(int t[],int n)`.
   (a) iteracyjna - `{int x=a[--n];while(n--)if(t[n]<x)x=t[n]; return x;}`
   (b) rekurencyjna - znajdź maksimum `n-1` elementów; porównaj je z ostatnim elementem
   (c) rekurencyjna - podziel tablicę na dwie części; znajdź ich maksima; wybierz większe z nich.
   Ile porównań między elementami wykonuje każda z wersji? Ile pamięci wymaga każda z tych wersji? Uwzględnij fakt, że głębokość rekurencji ma wpływ na zużycie pamięci, ponieważ powstaje wiele kopii zmiennych lokalnych.

5. Napisz procedurę `void reverse(lnode*& L)`, która odwróci listę L modyfikując jedynie pola `next` elementów listy L oraz wskaźnik L.

6. (2pkt) Napisz procedurę `lnode* merge(lnode* L1, lnode* L2)`, która złączy posortowane listy `L1` i `L2` w jedną posortowaną listę, zmieniając jedynie wartości pól `next` i używając tylko dwóch dodatkowych wskaźników. Ilość porównań nie powinna przekroczyć długości wynikowej listy.

7. Zapoznaj się z procedurą wstawiania klucza do drzewa BST. Jakie drzewo powstanie po wstawieniu do pustego drzewa BST liczb od 1 do n w kolejności rosnącej? Jaka potem będzie głębokość drzewa? Ile porównań kluczy wykonano w trakcie tworzenia tego drzewa? Jaka jest złożoność w tego procesu w notacji O? Uwaga: na każdym poziomie drzewa gdy element jest większe od klucza

8. Uzasadnij, że w każdym drzewie BST zawsze ponad połowa wskaźników (pól `left` i `right`) jest równa `NULL`.

9. Ile maksymalnie węzłów może mieć drzewo BST o głębokości h? Wylicz dokładną wartość, przyjmując, że głębokość oznacza ilość poziomów, na których występują węzły (sam korzeń: $h = 1$, korzeń i dzieci: $h = 2$ ...). Wywnioskuj, jaka jest najmniejsza, a jaka największa głębokość drzewa binarnego o n węzłach?

10. W pliku `tree-2020.cc` znajdziesz funkcję `int height(node *t)`, która wyliczy głębokość (ilość poziomów na jakich występują węzły) drzewa BST. Jak zależy czas wykonania tej funkcji od ilości n węzłów drzewa i jego głębokości h?

11. Przeanalizuj operacje dla drzewa BST (`find`, `insert`, `remove`) zawarte w pliku `tree-2020.cc`. Jak ich pesymistyczna złożoność czasowa $T(h)$ zależy od głębokości drzewa h?

12. Implementacja usuwania węzła z drzewa binarnego działa wg następującego schematu:

    (a) jeśli usuwany węzeł nie ma dzieci, to go usuwamy a odpowiedni wskaźnik zmieniamy na `NULL`.

    (b) jeśli ma jedno dziecko, to go usuwamy, a odpowiedni wskaźnik w węźle rodzica zastępujemy wskaźnikiem na to dziecko.

    (c) jeśli ma dwoje dzieci, to nie usuwamy tego węzła, lecz najmniejszy element w jego prawym poddrzewie, a dane i klucz tego elementu wpisujemy do węzła, który miał być usunięty.

    Uzasadnij, dlaczego postępowanie wg punktu (c) nie psuje prawidłowego rosnącego porządku kluczy wypisywanych w porządku `inorder`.

13. Jak zmodyfikować operacje dla drzewa BST (`insert`, `remove`) bez użycia rekurencji, aby działały poprawnie dla drzewa o węzłach gdzie występuje też wskaźnik na ojca. `struct node{int x; node *left; node *right; node *parent;};`

14. Napisz rekurencyjną procedurę `void inorder_do(node *t,void f(node*))`, która wykona funkcję `f` na każdym węźle drzewa `t` w kolejności `in_order`.

15. Wiedząc, że `node` zawiera wskaźniki na rodzica `parent`, napisz nierekurencyjną wersję powyższej funkcji.

16. (2 pkt.) Zakładając, że w każdym węźle drzewa BST jest również wskaźnik na ojca, napisz klasę `iterator` oraz funkcje `iterator begin(node *t)` oraz `iterator end(node *t)`, które pozwolą wypisać wszystkie klucze z drzewa `t` za pomocą instrukcji:

    ```
    for(iterator begin(t); i!=end(t); i++)
        cout<< *i <<endl;
    ```

    Jedyną składową (w części prywatnej) powinien być wskaźnik na bieżący węzeł.

17. (3 pkt.) Zmodyfikuj `iterator` drzewa BST, tak by nie korzystał z pola `parent`. Wskazówka: do części prywatnej iteratora dodaj stos elementów typu `node*` zawierający węzły, powyżej bieżącego.

Algorytmy i Struktury danych (2021)

Lista zadań 3 (rekurencja, drzewa, sortowanie)

1. Ile (dokładnie) porównań wykona algorytm `insertion_sort` w wersji z wartownikiem (liczbą $-\infty$ zapisaną pod adresem `t[-1]`), jeśli dane $(a_1, ..., a_n)$ o rozmiarze `n` zawierają `k` inwersji. Liczba inwersji to liczba takich par (i, j), że $i < j$ i $a_i > a_j$. Jaka jest maksymalna możliwa liczba inwersji dla danych rozmiaru n? Wylicz "średnią" złożoność algorytmu, jaka średnią z maksymalnej i minimalnej ilości porównań jaką wykona. Uwaga: Prawdziwą średnią złożoność oblicza się, jako średnią po wszystkich możliwych permutacjach danych wejściowych.

2. (a) Ile co najwyżej porównań wykona procedura `insertion_sort` działająca na ostatnim etapie `bucket_sort` zakładając, że `bucket_sort` korzysta z `k` pomocniczych kolejek, i że do każdej z nich wpadła taka sama ilość elementów? Zakładamy wersję z wartownikiem na pozycji `t[-1]`.
   (b) Podaj uproszczony wynik dla $k = n/2$, $k = n/4$, $k = n/10$ oraz $k = \sqrt{n}$.
   Następnie każdy z tych wyników zapisz też w notacji asymptotycznej $O(f(n))$.
   (c) Jaki będzie wynik, gdy wszystkie klucze wpadną do tego samego kubełka?

3. Iteracyjna wersja procedury `mergesort` polega na scalaniu posortowanych list. Zaczyna łączenia pojedynczych elementów w posortowane pary, potem par w czwórki itd. aż do połączenia dwóch ostatnich list w jedną.
   (a) Zakładając, że rozmiar tablicy jest potęgą dwójki $n = 2^k$ oraz, że procedura merge wykonuje dokładnie tyle porównań, ile jest elementów po scaleniu, oblicz ile dokładnie porównań wykona cały algorytm.
   (b) Ile razy jest wywołana procedura `merge` w trakcie działania całego algorytmu? Jak zmieni się wynik punktu (a), jeśli założymy, że `merge` zawsze, wykonuje o jedno porównanie mniej, niż zakładaliśmy w punkcie (a)?

4. Napisz procedurę `void insertion_sort(lnode*& L)` – sortowanie przez wstawianie działające na liście jednokierunkowej. Zadbaj o to, by algorytm modyfikował jedynie pola `next` istniejących węzłów (nie używaj `new` ani `delete`). Jeśli list wejściowa jest posortowana, algorytm powinien wykonać tylko $n-1$ porównań. Wskazówka: algorytm wkładać elementy na "nową" listę w kolejności malejącej, a na końcu wywołać `reverse(L)`.

5. W pliku jest $n = 10^9$ liczb całkowitych. Ile potrzeba pamięci i dodawań by sprawdzić, która z sum $k = 10^4$ kolejnych liczb jest największa? Tych sum jest $n - k + 1$ tzn i-ta suma zaczyna się od i-tej liczby. Napisz program obliczający tę wartość tak, aby całkowity rozmiar utworzonych zmiennych wynosił około 20-bajtów (nie licząc obiektów `ifstream`), niezależnie od wartości n i k. Argumentami programu powinny być liczba k oraz nazwa pliku.
   Wskazówka: plik można czytać w dwóch miejscach jednocześnie.

6. Zakładając, że w każdym węźle drzewa BST jest dodatkowe pole `int nL`; pamiętające ilość kluczy w lewym poddrzewie, napisz funkcję `BTSnode* ity(BSTnode *t, int i)`, która zwróci wskaźnik `i`-ty (w kolejności `in order`) węzeł. Przyjmujemy, że numeracja zaczyna się od 0, czyli `ity(t,0)` to węzeł zawierający najmniejszy klucz. Dla wartości $i \geq n$ oraz $i < 0$ wynikiem funkcji powinien być `nullptr`. Pesymistyczna złożoność algorytmu powinna być równa głębokości drzewa. Nie korzystaj z rekurencji.

   Skoryguj procedurę `insert`, i konstruktor `node`, by prawidłowo aktualizowały wartości `nL` w trakcie dodawania elementów.

7. * Skoryguj procedurę `remove`, by prawidłowo aktualizowała wartości `nL` w trakcie usuwania elementów. Napisz funkcję `void remove_ity(BSTnode*&t,int i)`, która usunie `ity(t,i)` węzeł z drzewa.

Ćwiczenia do wykonania jako przygotowanie do kolokwium:

1. Napisz nierekurencyjną procedurę `int poziom(BSTnode * t, int klucz)`, której wynikiem jest poziom w drzewie `t`, na którym występuje `klucz`. Wynik 0 oznacza brak klucza w drzewie, 1 - klucz w korzeniu, 2 - w dziecku korzenia itd.

2. Napisz procedurę `int suma_do_poziomu(BSTnode * t, int poziom)`, której wynikiem jest suma kluczy znajdujących się na głębokości nie większej niż `poziom`. Przyjmujemy, że korzeń drzewa jest na poziomie 1.

3. Napisz funkcję `node* shift_sorted(node*& L)`, która od początku listy `L` odcina listę niemalejącą o największej możliwej długości i ją zwraca. Znaczy to, że cięcie jest przed pierwszym elementem mniejszym od poprzedniego lub na końcu listy.

Zadania dla ambitnych:

1. (3pkt) Napisz procedurę `void merge_sort(node*& n)` - sortowanie przez złączanie działające na liście jednokierunkowej, nie używaj rekurencji. Zadbaj by rozmiar dodatkowej pamięci nie przekroczył $\log_2 n + const$.
   Skorzystaj z procedury `lnode* merge(lnode* L1, lnode* L2)` z poprzedniej listy.

2. (3pkt) Napisz procedurę `merge` działającą na tablicy tak, aby nie wykorzystywać dodatkowego bufora, kosztem zwiększenia złożoności do $O(n \log n)$. Wskazówka: w czasie $O(n)$ można wykonać przejście $(1, 3, 5, 7, 9|2, 4, 6, 8, 10) \rightarrow (1, 2, 5|2, 4|7, 9|6, 8, 10)$, które powoduje, że każdy element lewej części jest mniejszy od każdego w prawej, a następnie rekurencyjnie wywołać `merge` dla każdej części. Jaka będzie wtedy złożoność `mergesort`?

Algorytmy i Struktury danych (2021)

Lista zadań 4 (rekurencja uniwersalna, mergesort, drzewa)

1. Skorzystaj z metody rekurencji uniwersalnej i podaj dokładne asymptotyczne oszacowania dla następujących rekurencji:

   (a) $T(n) = 4T(n/2) + n$,
   (b) $T(n) = 4T(n/2) + n^2$,
   (c) $T(n) = 4T(n/3) + n^3$,

2. Korzystając z twierdzenia o rekurencji uniwersalnej rozwiąż następujące zależności:

   (a) $T(n) = 5T(n/3) + n$,
   (b) $T(n) = 9T(n/3) + n^2$,
   (c) $T(n) = 6T(n/3) + n^2$,
   (d) $T(n) = T(n/2) + 1$,
   (e) $T(n) = 2T(\sqrt{n}) + 1$ (potrzebna zamiana zmiennych).

3. Czas działania algorytmu A opisany jest przez rekurencję $T(n) = 7T(n/2) + n^2$. Algorytm konkurencyjny A' ma czas działania $T'(n) = aT'(n/4) + n^2$. Jaka jest największa liczba całkowita a, przy której A' jest asymptotycznie szybszy niż A?

4. Rozważmy warunek regularności $af(n/n) \leqslant cf(n)$ dla pewnej stałej $c \leqslant 1$, który jest częścią przypadku 3 twierdzenia o rekurencji uniwersalnej. Podaj przykład prostej funkcji $f(n)$, które spełnia wszystkie warunki twierdzenia o rekurencji uniwersalnej z wyjątkiem warunku regularności.

5. Zasymuluj działanie polifazowego mergesorta dla tablicy:
   `{9,22,6,19,21,14,10,17,3,5,60,30,29,1,8,7,6,15,12}`.
   W sortowaniu polifazowym na każdym etapie sortowania scala się sąsiadujące podciągi rosnące, to znaczy: w pierwszym przebiegu `{9,22}` z `{6,19,21}`, `{14}` z `{10,17}` itd..

6. W tablicy `t[n]` umieszczone są w przypadkowej kolejności wszystkie liczby całkowite od `1` do `n+1` za wyjątkiem jednej. Napisz funkcję `int brakujaca(int t[],int n)` zwracającą brakującą liczbę całkowitą, tak aby ilość potrzebnej pamięci nie zależała od n, a czas wykonania był linowy tzn. $\Theta(n)$.

7. (2pkt) Niech $T_2(n)$ oznacza ilość różnych kształtów drzew binarnych o n węzłach. $T(0) = 1$, $T(1) = 1$, $T(2) = 2$, $T(3) = 5$, ...
   (a) Znajdź wzór rekurencyjny wyrażający $T_2(n)$ przez $\{T_2(i) : 0 \leq i < n\}$.
   (b) Napisz w javascript procedurę rekurencyjną, która używa tego wzoru. Zastosuj `BigNum`.
   (b') Przyspiesz twoją procedurę korzystając z symetrii wzoru rekurencyjnego (a).
   (c) Napisz w javascript procedurę nierekurencyjną, która oblicza po kolei wyrazy ciągu $T_2(n)$ i zapisuje je w tablicy. Przy obliczaniu kolejnych wyrazów, korzysta w poprzednio zapisanych wyników.
   (d) Sprawdź ile wartości ciągu $T_2(n)$ możesz uzyskać procedurą (b), (b'), (c) w czasie 2 minut. Przy każdej wartości $T_2(n)$ wypisuj czas jej obliczenia.

8. (2pkt) Wykonaj punkty (a), (c), (d) zadania 7 dla drzew trynarnych, których węzeł wygląda tak: `struct node3{int k; node3* left; node3* middle; node3* right}`.

9. * Dla ambitnych (3pkt) Napisz procedurę `T(i,n)` obliczającą ilość kształtów drzew i-narnych o n węzłach.

Algorytmy i Struktury Danych (2021)

Lista zadań 5 - sortowanie i kopce

Uwaga: Zadania z tej listy przygotowujemy w formie pisemnej. Wyjątkiem jest zadanie 4, które należy zaimplementować w języku C++.

1. a) Czy tablica posortowana malejąco jest kopcem?
   b) Czy ciąg `{23, 17,14,6,13,10,1,5,7,12}` jest kopcem?

2. Zasymuluj działanie `buildheap` dla `t[]={1,2,3,4,5,6,7}`. Zapisz na kartce wygląd tablicy/kopca po każdym wywołaniu procedury `przesiej`.

3. Zilustruj działanie procedury `buildheap` dla ciągu `{5,3,17,10,84,19,6,22,9}`. Narysuj na kartce wygląd tablicy/kopca po każdym wywołaniu procedury `przesiej`.

4. W oparciu o kod procedury `heap_sort` napisz klasę `priority_queue` z metodami `put` i `get_max` oraz `is_empty` i `is_full`. W konstruktorze umieść funkcję `build_heap`.

5. Udowodnij, że procedura `build_heap` działa w czasie liniowym.

6. Udowodnij, że wysokość kopca n-elementowego wynosi $\lfloor \log_2 n \rfloor + 1$.

7. (2 pkt) Podaj ideę algorytmu, jak przy pomocy struktury kopca, złączyć k posortowanych list jednokierunkowych o łącznej ilości elementów n, w jedną posortowaną listę, za używając nie więcej niż $n \log_2 k$ porównań.

Algorytmy i Struktury Danych (2021)

Lista zadań 6 - quick sort, selekcja, sortowanie bez porównań

1. Udowodnij, że jeśli dla pewnego ustalonego q, takiego że $\frac{1}{2} < q < 1$, podczas sortowania szybkiego, procedura `partition`, na każdym poziomie rekurencji podzieli elementy tablicy w stosunku $q : (1 - q)$ to algorytm wykona się w czasie $O(n \log n)$. Wskazówka: udowodnij, że głębokość rekurencji nie przekroczy $-\log n / \log q$ i zaniedbaj błędy zaokrągleń do wartości całkowitych.

2. Napisz wzór na numer kubełka, do którego należy wrzucić liczbę x w sortowaniu kubełkowym, jeśli kubełków jest n, a elementy tablicy mieszczą się przedziale (a, b). Numeracja zaczyna się od 0.

3. Dla jakich danych sortowanie metodą kubełkową ma złożoność $O(n^2)$?

4. Jak obliczyć k-tą od końca cyfrę w liczby x? Jak obliczyć ilość cyfr liczby x? Przyjmujemy układ dziesiętny. Jak wyniki zmienią się w układzie pozycyjnym o gdzie różnych cyfr jest 1000 a ich wartości pokrywają zakres 0-999?

5. (a) Napisz procedurę, `counting_sort(int t[], int n, int c);` która posortuje metodą przez zliczanie liczby w tablicy `t[]` względem cyfry `c`. $c = 0$ oznacza cyfrę jedności, $c = 1$ cyfrę dziesiątek itd...

6. Posortuj metodą sortowania pozycyjnego liczby: 101, 345, 103, 333, 432, 132, 543, 651, 791, 532, 987, 910, 643, 641, 12, 342, 498, 987, 965, 322, 121, 431, 350. W pisemnym rozwiązaniu pokaż, jak wygląda zawartość kolejek, za każdym razem, gdy tablica wyjściowa jest pusta i wszystkie liczby znajdują się w kolejkach, oraz jak wygląda tablica wyjściowa, za każdym razem, gdy sortowanie ze względu na kolejną cyfrę jest już zakończone.

7. (2pkt) Które z procedur sortujących:
   (a) `insertion_sort` (przez wstawianie),
   (b) `quick_sort` (szybkie),
   (c) `heap_sort` (przez kopcowanie),
   (d) `merge_sort` (przez złączanie),
   (e) `counting_sort` (przez zliczanie)
   (f) `radix_sort` (pozycyjne),
   (g) `bucket_sort` (kubełkowe)
   są stabilne? W każdym przypadku uzasadnij stabilność lub znajdź konkretny przykład danych, dla których algorytm nie zachowa się stabilnie.

8. (algorytm Hoare'a) Korzystając funkcji `int partition(int t[], int n)` znanej z algorytmu sortowania szybkiego napisz funkcję `int kty(int t[], int n)`, której wynikiem będzie k-ty co do wielkości element początkowo nieposortowanej tablicy `t`. Średnia złożoność Twojego algorytmu powinna wynieść $O(n)$.

9. * (2pkt.) Napisz program, który znajdzie sposób, w jaki konik szachowy może w 64 poprawnych ruchach odwiedzić wszystkie szachownicy, na każdym będąc dokładnie raz.

10. * (2 pkt.) Napisz program znajdujący wszystkie ustawienia 8 hetmanów na szachownicy, takie że żaden z nich nie szachuje innego. Oszacuj złożoność twojego rozwiązania.

Algorytmy i Struktury Danych (2021)

Lista zadań 7 - drzewa czerwono-czarne

1. Jakie informacje przechowujemy w węźle drzewa czerwono-czarnego? Zadeklaruj strukturę `RBTnode` tak, by dziedziczyła z `BSTnode`. Podaj definicję drzewa czerwono czarnego.

2. (a) Jaka może być minimalna, a jaka maksymalna ilość kluczy w drzewie czerwono-czarnym o ustalonej czarnej wysokości równej h_B?
   (b) Znajdź maksymalną i minimalną wartość stosunku ilości węzłów czerwonych do czarnych w drzewie czerwono-czarnym.

3. Uzasadnij posługując się rysunkiem i opisem, że operacje wykonywane w trakcie wstawiania do drzewa czerwono-czarnego (rotacja i przekolorowanie) nie zmieniają ilości czarnych węzłów, na żadnej ścieżce od korzenia do liścia.

4. (a) Narysuj poprawne drzewo czerwono czarne w którym na lewo od korzenia jest 1 węzeł a na prawo 7 węzłów.
   (b) Czy istnieje poprawne drzewo czerwono czarne, w którym na lewo od korzenia będzie 100 razy mniej węzłów niż na prawo od korzenia?

5. W poniższym drzewie czerwono-czarnym (czarne węzły oznaczono nawiasem kwadratowym):

```
            [5]
       (3)       [11]
   [1]   [4]   (9)
```

   - wstaw do niego `10`.
   - usuń z wyjściowego drzewa `1`.

6. (2 pkt.) Do pustego drzewa czerwono-czarnego wstaw kolejno 20 przypadkowych kluczy. Następnie usuń je w tej samej kolejności w jakiej wstawiałeś. Przypadkowymi kluczami są kolejne litery Twojego nazwiska, imienia i adresu. Zadanie wykonujemy na kartce (lub w pliku).

7. Analizując kod programu `RBT.h` udowodnij, że w trakcie wstawiania do drzewa czerwono-czarnego wykonają się co najwyżej dwie rotacje. Czy tak samo jest w przypadku usuwania?

8. Uzasadnij, że rozmiar stosu ($n = 100$) przyjęty w procedurach `insert` i `remove` w pliku `RBnpnr.h` nigdy nie okaże się za mały.

Algorytmy i Struktury Danych (2021)

Lista zadań 8 (B-drzewa)

1. Jakie informacje przechowujemy w węźle B-drzewa? Podaj definicję B-drzewa.

2. Udowodnij, że żadna z operacji:

   (a) `splitchild`, przesuwająca środkowy klucz (medianę) z węzła o maksymalnej liczbie kluczy do rodzica, a klucze większe od mediany do nowego brata dodanego po prawej stronie dzielonego węzła.

   (b) `unsplit_child` odwrotna do `split_child`, sklejająca dwa sąsiednie węzły o minimalnej liczbie kluczy oraz klucz stojący w rodzicu między nimi w jeden nowy węzeł.

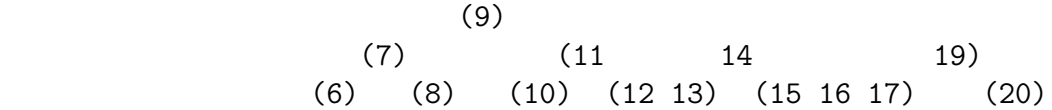
   (c) `borrow_from_sibling`, "rotacja" przenosząca do węzła o minimalnej liczbie kluczy, sąsiedni klucz z rodzica i wpisująca na jego miejsce najbliższy klucz brata (jeśli brat ma co najmniej t kluczy)

   wykonana na drzewie spełniającym wszystkie warunki B-drzewa, nie prowadzi do naruszenia, żadnego z tych warunków.

3. W B-drzewie o $t = 10$:
   - ile kluczy może zawierać korzeń (podaj przedział),
   - ile dzieci może mieć korzeń (podaj przedział),
   - ile kluczy może mieć potomek korzenia (podaj przedział),
   - ile dzieci może mieć potomek korzenia (podaj przedział),
   - ile maksymalnie węzłów może być na k-tym poziomie (przyjmując, że korzeń to poziom 0),
   - ile łącznie kluczy może być na k-tym poziomie (podaj przedział).

4. Jak jest minimalna, a jaka maksymalna liczba kluczy w B-drzewie mającym h poziomów, przy ustalonej wartości parametru t (patrz Cormen).

5. Podano na rysunku B-drzewo o $t = 2$:



   - usuń z tego drzewa `7`.
   - do drzewa widocznego powyżej dodaj `18`.

6. (2 pkt.) Do pustego B-drzewa wstaw kolejno 22 litery swojego imienia i nazwiska oraz adresu pod jakim mieszkasz.
   Następnie usuń je w tej samej kolejności w jakiej były wstawiane.

7. Narysuj B-drzewo o $t = 3$ zawierające dokładnie 17 kluczy na trzech poziomach: korzeń jego dzieci i wnuki. Następnie usuń z tego drzewa korzeń.

Algorytmy i Struktury Danych (2021)

Lista zadań 9 (kopce dwumianowe i Union-Find)

1. Udowodnij, że:

   (a) drzewo dwumianowe rzędu n ma 2^n węzłów.

   (b) na k-tym poziomie drzewa dwumianowego rzędu n znajduje się dokładnie $\binom{n}{k}$ węzłów.

2. Napisz (jak najszybszą) funkcję `int f(int n)` wyliczającą ile jest drzew dwumianowych w kopcu dwumianowym zawierającym n kluczy.

3. (a) Do pustego kopca dwumianowego wstaw (INSERT) kolejno: 1, 12, 3, 14, 5, 16, 7, 20, 25 13, 8

   (b) Dla otrzymanego kopca dwukrotnie wykonaj operację GETMAX.

4. Zaimplementuj klasę `UnionFind`, której argumentem konstruktora jest ilość n wierzchołków grafu (początkowa liczba zbiorów), metoda `int Find(i)` zwraca reprezentanta `i`-tego zbioru, a metoda `void Union(int i ,int j)` scala zbiory zawierające `i` oraz `j`. Zastosuj heurystyki kompresji ścieżek, oraz "podczepiania" zbioru o mniejszej randze pod zbiór o większej randze (patrz Cormen).

5. Zastosuj strukturę z poprzedniego zadania do sprawdzenia czy w tablicy `bool t[n][n]` istnieje ścieżka zawierająca same jedynki (`true`): (a) od pola `t[0][0]` do `t[n-1][n-1]` (b) od pierwszego do ostatniego wiersza (tzn. jakaś komórka z pierwszego wiersza jest połączona ścieżką z jakąś komórką z ostatniego wiersza). Za ścieżkę uważamy ciąg pól tablicy, które stykają się krawędzią (różną się o 1 numerem kolumny albo wiersza).

6. Zastosuj strukturę `UnionFind` do sprawdzenia ile wysp jedynek zawiera tablica `bool t[n][n]`. Za wyspę uważamy zbiór jedynek taki, że z każdej do każdej można przejść ścieżką zawierającą same jedynki poruszając się tylko w poziomie i pionie.

Algorytmy i Struktury Danych (2021)

Lista zadań 10 (programowanie dynamiczne)

1. Znajdź (bez użycia komputera) optymalne nawiasowanie iloczynu macierzy, których wymiary tworzą ciąg [5, 10, 2, 12, 5, 50, 6]. Spamiętywanie (jeśli go użyjesz) wykonuj na kartce.

2. Podane w załączniku (oraz na `panoramixie`) programy `progdyn1.cc` oraz `progdyn2.cc`, rozwiązujące problem optymalnej triangulacji oraz optymalnego nawiasowania z wykorzystaniem spamiętywania, przepisz w taki sposób, by procedury znajdowania minimum nie korzystały z rekurencji, tylko wypełniały tablicę wyników w pętli w takiej kolejności, by w momencie gdy wyliczana jest wartość `F[i][j]` wszystkie potrzebne do jej wyliczenia elementy tablicy były już wypełnione. Wskazówka: należy wyliczać najpierw te pary `F[i][j]` dla których różnica `j-i` jest najmniejsza, czyli najpierw wyliczyć wszystkie koszty mnożenia dwóch sąsiednich macierzy, potem trzech, itd.

3. Jaka jest złożoność procedury wyznaczającej koszt optymalnego mnożenia ciągu n macierzy, zaimplementowana metodą z poprzedniego zadania, czyli wstępującą (bez rekurencji). Ile mnożeń wykona ta procedura? (a) w notacji O. (b) dokładnie.

4. (0.5pkt) Udowodnij, że wszystkich sposobów pocięcia pręta długości n, na kawałki całkowitej długości, jest 2^{n-1}.

5. (2pkt) Dany jest długi pręt stalowy długości n będącej całkowitą liczbą centymetrów, oraz tablica `double cena[n+1];` taka, że `cena[i]` określa, za ile można sprzedać kawałek długości i cm dla każdego $1 \leq i \leq n$ (kawałki długości zero są darmowe :). Napisz regułę rekurencyjną pozwalającą obliczyć jak pociąć na kawałki całkowitej długości pręt, by najwięcej na tym zarobić:
   (a) zakładając że cięcie jest darmowe
   (b) zakładając, że każde przecięcie pręta kosztuje c zł.

6. Zastosuj programowanie dynamiczne, by napisać program, który rozwiązuje problem z poprzedniego zadania: drukuje rosnącą listę długości kawałków, oraz całkowity zysk (dochód ze sprzedaży, pomniejszony w punkcie (b) o koszt cięcia).

   Wskazówka: procedura wypełnia tablice `MaxGain` oraz `LongestBit` (albo `ShortestBit`).

7. (2pkt) Optymalne drzewa poszukiwań binarnych. Istnieje wiele drzew BST zawierających dokładnie ten sam (uporządkowany rosnąco) zestaw kluczy. Zastosuj programowanie dynamiczne do znalezienie optymalnego drzewa poszukiwań binarnych (patrz Cormen). Załóż, że koszt wyszukania klucza to liczba węzłów, jakie trzeba odwiedzić, by go znaleźć (dla klucza w korzeniu koszt 1, piętro niżej – koszt 2, itd). Dane do zadania stanowi tablica `ile`, taka, że `ile[i]` mówi ile razy będzie wyszukiwany i-ty co do wielkości klucz.

Algorytmy i Struktury Danych

Sprawdzian z grafów - przygotowanie

Na podstawie grafu nieskierowanego zadanego jako następująca lista krawędzi:

(1,2):8 (2,3):1 (3,4):15 (2,5):7 (4,5):3 (5,6):12 (1,6):2 (6,7):4 (2,6):20 (5,7):5.

1. Wykonaj rysunek grafu.
2. Znajdź macierz sąsiedztwa.
3. Zapisz tablicę list sąsiedztwa. Wierzchołki na listach sąsiedztwa powinny być są ustawione rosnąco wg numeru wierzchołka. Ta kolejność powinna być stosowana w symulacji algorytmów DFS, BFS i Dijkstry.
4. Zapisz kolejność odwiedzania wierzchołków przez algorytm DFS startujący z wierzchołka 5.
5. Zapisz kolejność odwiedzania wierzchołków w algorytmie BFS startującym z tego samego wierzchołka.
6. (2 pkt) Zasymuluj działanie algorytmu Kruskala i zilustruj rysunkiem:
   - liniami przerywanymi oznacz krawędzie nie należące do drzewa wynikowego,
   - liniami ciągłymi oznacz krawędzie należące do drzewa wynikowego,
   - przy każdej krawędzi w nawiasie okrągłym podaj kolejność w jakiej była ona rozpatrywana.
7. (3 pkt) Zasymuluj działanie algorytmu Dijksty startując z wierchołka 3. Zapisz kroki algorytmu podając w każdym kroku:
   - numer odwiedzanego wierzchołka
   - wykonane w tym kroku operacje `decrease_key` i odpowiednie zmiany w tablicy poprzedników (`prev`)
   - wypisując jaka jest zawartość kolejki priorytetowej po wykonaniu kroku

   Na końcu algorytmu dla każdego wierzchołka zapisz:
   - odległość od wierzchołka startowego
   - numer wierzchołka będącego poprzednikiem

   Algorytm zilustruj grafem, w którym:
   - przy każdym wierzchołku będzie podany w nawiasie okrągłym numer kroku algorytmu, w którym wierzchołek został odwiedzony.
   - strzałkami ciągłymi oznaczone będą krawędzie należące do drzewa wynikowego
   - strzałkami przerywanymi oznaczone będą krawędzie, które w trakcie algorytmu wskazywały na poprzednika, jednak nie należą do drzewa wynikowego.

Algorytmy i Struktury Danych

Lista zadań 12 - kody Huffmana

1. **Regularne** drzewo binarne to takie, które nie ma węzłów z jednym dzieckiem. Udowodnij, że drzewo binarne, które **nie jest regularne**, nie może odpowiadać optymalnemu kodowi prefiksowemu.

2. Czy kody Huffmana są wyznaczone jednoznacznie dla każdego tekstu? Dlaczego?

3. Jaki jest optymalny kod Huffmana, dla zbioru częstości opartego na początkowych $n = 8$ liczbach Fibonacciego: a:1, b:1, c:2, d:3, e:5, f:8, h:21?

   Uogólnij odpowiedź na przypadek dowolnego n.

4. Jaki jest optymalny kod Huffmana, dla zbioru częstości opartego na liczbach: a:11, b:12, c:13, d:14, e:15, f:18, h:19? Porównaj długość zakodowanego tekstu z długością tekstu zakodowanego przy pomocy kodów stałej długości.

   Uogólnij odpowiedź na przypadek $n = 2^k$ liter w alfabecie, gdzie dodatkowo maksymalna ilość wystąpień jest mniejsza od dwukrotności minimalnej ilości wystąpień.

5. Dla podanego tekstu "bababacacadaaasadaca".

   (a) Zasymuluj działanie algorytmu generującego kody Huffmana, narysuj otrzymane drzewo kodów, wypisz kody poszczególnych znaków, oraz zakodowany tekst.

   (b) Oblicz, o ile bitów otrzymana reprezentacja tekstu będzie krótsza od reprezentacji otrzymanej za pomocą kodów o stałej długości.

   (c) Mając dane drzewo kodów i zakodowany tekst wykonaj dekodowanie (zaznaczaj w ciągu bitów kreską gdzie kończą się doby poszczególnych znaków).

   (d) Zobacz, jak deszyfrowany tekst zmieni się, gdy zmienisz zaszyfrowanej wersji na przeciwny.

6. Udowodnij, że długość (ilość bitów) zaszyfrowanego tekstu, jest sumą liczb występujących w wewnętrznych węzłach drzewa kodów (nie liściach).

7. (a) Uogólnij algorytm Huffmana do kodów trójkowych (używających znaków 0,1,2).
   (b) Zmodyfikuj odpowiednio plik `kody-huffmana.cc`.

Algorytmy i Struktury Danych

Lista zadań 13 - DFT, sieci sortujące

1. Dyskretną transformatą Fouriera ciągu $(a_1, ..., a_n)$ nazywamy ciag $(A_1, ..., A_n)$ taki, że:

$$A_k = \sum_{p=0}^{n-1} a_p e^{2\pi ikp/n} \quad \text{dla } k = 0 \ldots n-1. \tag{1}$$

   Odwrotną dyskretną transformatą Fouriera ciągu $(A_1, \ldots, A_n)$ nazywamy ciąg $(b_1, ..., b_n)$ taki, że:

$$b_q = \frac{1}{n} \sum_{k=0}^{n-1} A_k e^{-2\pi iqk/n} \quad \text{dla } q = 0 \ldots n-1 \tag{2}$$

   Wstawiając wzór (1) do prawej strony wzoru (2) udowodnij że $b_q = a_q$ dla $q = 0 \ldots n-1$, czyli, że wynikiem odwrotnej dyskretnej transformaty Fouriera, wykonanej na transformacie ciągu $(a_1, \ldots, a_n)$ jest faktycznie wyjściowy ciąg liczb $(a_1, \ldots, a_n)$.

2. Wyznacz DFT dla ciągów (5, 3), (1, 5, 3, 1) oraz (1,2,3,4,5,6,7,8):
   (a) z definicji,
   (b) symulując działanie algorytmu FFT podanego na wykładzie.

3. Udowodnij zasadę zero-jedynkową dla sieci sortujących: *Jeśli sieć poprawnie sortuje wszystkie możliwe n-elementowe ciągi zer i jedynek, to dobrze sortuje dowolne ciągi liczb rzeczywistych.*

4. * Udowodnij, że `bitonic_half_cleaner(2n)` (patrz Cormen) działa poprawnie dla dowolnego ciągu bitonicznego złożonego z zer i jedynek. To znaczy, że dla dowolnego bitonicznego ciągu zer i jedynek o długości $2n$ danego na wejściu, wynikiem jest: albo ciąg którego lewa połowa to zera a prawa jest bitoniczna, albo ciąg którego lewa połowa jest bitoniczna a prawa to jedynki.

5. Narysuj sieć sortującą n liczb dla $n = 2, 4, 8, 16$. Powinna to być opisana na wykładzie sieć implementująca równoległą wersję algorytmu `mergesort`, działająca w czasie $O((\log n)^2)$. Prześledź działanie sieci o $n = 8$ dla ciągu wejściowego: 8 4 2 3 7 5 6 1, rysując jakie liczby wchodzą i wychodzą z każdego komparatora (na kartkach).