Kolokwium II, grupa 2

1. Porównaj złożoność metody sortowania przez wstawianie realizowane w tablicy z metodą porządkowania przez wstawianie kolejnych wczytywanych danych w odpowiednich miejscach dynamicznej listy jednokierunkowej. Opisz w szczególności czy (a jeśli tak, to w jakim stopniu) zastosowanie listy może zmniejszyć liczbę porównań i/lub podstawień niezbędnych do posortowania danych.

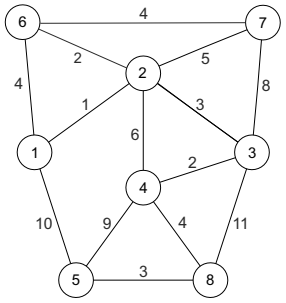
2. Narysuj kolejne stadia tworzenia kopca zgodnie z poznanym algorytmem (po 1 rysunku na każdy poziom) z następujących danych: 2, 15, 8, 4, 10, 7, 3, 5, 11, 6, 20, 9, 1, 5.

3. Zakladamy, że mamy do dyspozycji funkcję `void dodaj(wel *P, int y)`, która do uporządkowanej listy opisanej pierwszym parametrem wstawia nowy element `y` we właściwej pozycji. Niech `N` będzie liczbą danych `y` (czytanych kolejno z wejścia), `a` i `b` będą zadanymi parametrami ograniczającymi zbiór wartości `y`, to jest $\texttt{a} \le \texttt{y} \le \texttt{b}$, oraz $\texttt{M} < \texttt{N}$ będzie projektowaną liczbą kubełków, implementowanych jako uporządkowane listy obsługiwane funkcją `dodaj`, dostępnych poprzez tablicę wskaźników na poszczególne kubełki, `wel T[M]`. Napisz kod programu realizującego sortowanie kubełkowe dla tych danych. Pierwsza część programu czyta dane `y` z wejścia i umieszcza je w odpowiednich kubełkach, druga wyświetla dane w porządku rosnącym, odczytując je z kolejnych kubełków.

4. Graf G, o wierzchołkach $V = \{1, 2, ..., 8\}$ opisany jest przez listy sąsiedztwa `S[v]` oraz macierz odległości `W[u][v]` zgodnie z rysunkiem poniżej.

```
S[1]  =  {2,5,6}        S[5]  =  {4,8,1}
S[2]  =  {4,7,3,6,1}    S[6]  =  {7,1,2}
S[3]  =  {8,4,2,7}      S[7]  =  {3,6,2}
S[4]  =  {3,5,8,2}      S[8]  =  {4,3,5}
```

Narysuj drzewo najkrótszych ścieżek utworzone przez algorytm Dijkstry, gdy startujemy z wierzchołka 5. Podaj kolejność, w jakiej wierzchołki grafu kwalifikowane są przez algorytm jako "odwiedzone".



5. *Ekscentrycznością* $\mathcal{E}$ wierzchołka v w nieskierowanym grafie G nazywamy długość najdłuższej wśród najkrótszych ścieżek z v do wszystkich innych wierzchołków,

$$\mathcal{E}(v) \;=\; \max_{x \in V(G)} d_{\min}(v, x)\,, \qquad d_{\min}(v, x) = \text{długość najkrótszej ścieżki z } v \text{ do } x\,.$$

Centrum $\mathcal{C}$ grafu G jest to zbiór wierzchołków G o minimalnej ekscentryczności. Wśród zastosowań można wymienić np. algorytm wyboru wsi na siedzibę gminnej przychodni zdrowia. Wybór wsi z centrum grafu połączeń drogowych jest najbardziej sprawiedliwy dla mieszkańców całej gminy, także tych z jej obrzeży.

Załóżmy, że mamy funkcję `float dmin(int u, int v)`, zwracającą długość najkrótszej ścieżki z u do v w grafie. Napisz program, który wyznaczy ekscentryczności `E[v]` wszystkich wierzchołków, a następnie centrum danego grafu. Centrum może zawierać więcej niż jeden wierzchołek, jeśli kilka z nich ma identyczną ekscentryczność.

Kolokwium II, grupa 3

1. Jedna z modyfikacji algorytmu sortowania bąbelkowego polega na "wahadłowej" zmianie kierunku przeglądania tablicy w kolejnych przebiegach: od $i = 1$ do $n - 1$, następnie od $i = n - 2$ do 1, dalej od $i = 2$ do $n - 2$ itd. W ten sposób w pierwszym przebiegu największy element trafia na koniec tablicy, w drugim — najmniejszy na jej początek itd. Czy taka modyfikacja wpływa na:

a) liczbę zamian sortowanych elementów,
b) liczbę wykonanych porównań elementów,
c) obydwie,
d) żadną z nich?

Odpowiedź uzasadnij.

2. Sprawdź czy następujące dane tworzą kopiec: 17, 9, 11, 8, 3, 10, 9, 4, 7, 5, 3, 15, 13. Jeśli to konieczne, popraw jego strukturę, a następnie zasymuluj proces sortowania tych danych przez rozbiór kopca. Rozwiązanie przedstaw w postaci kilku rysunków pokazujących kolejne kroki działania algorytmu.

3. Zakladamy, że mamy do dyspozycji funkcję `void dodaj(wel *P, float y)`, która do uporządkowanej listy opisanej pierwszym parametrem wstawia nowy element `y` we właściwej pozycji. Niech `N` będzie liczbą danych `y` (czytanych kolejno z wejścia), `a` i `b` typu `float` będą zadanymi parametrami ograniczającymi zbiór wartości `y`, to jest $\texttt{a} \leq \texttt{y} \leq \texttt{b}$, oraz $\texttt{M} < \texttt{N}$ będzie projektowaną liczbą kubełków, implementowanych jako uporządkowane listy obsługiwane funkcją `dodaj`, dostępnych poprzez tablicę wskaźników na poszczególne kubełki, `wel T[M]`. Napisz kod programu realizującego sortowanie kubełkowe dla tych danych. Pierwsza część programu czyta dane `y` z wejścia i umieszcza je w odpowiednich kubełkach, druga wyświetla dane w porządku rosnącym, odczytując je z kolejnych kubełków.

4. Graf G o wierzchołkach `a, b, c, ...` opisany jest przez następujące listy sąsiedztwa:

```
S[a]  =  {c,g}        S[b]  =  {d,c,h}
S[c]  =  {g,b,f,a}    S[d]  =  {b,e,h}
S[e]  =  {d}          S[f]  =  {c,g}
S[g]  =  {c,a,f}      S[h]  =  {d,b}
```

Narysuj graf G oraz drzewa utworzone odpowiednio podczas przeszukania G algorytmami DFS i BFS, gdy startujemy z wierzchołka `g`. W każdym przypadku podaj kolejność, w jakiej odwiedzane będą wierzchołki grafu.

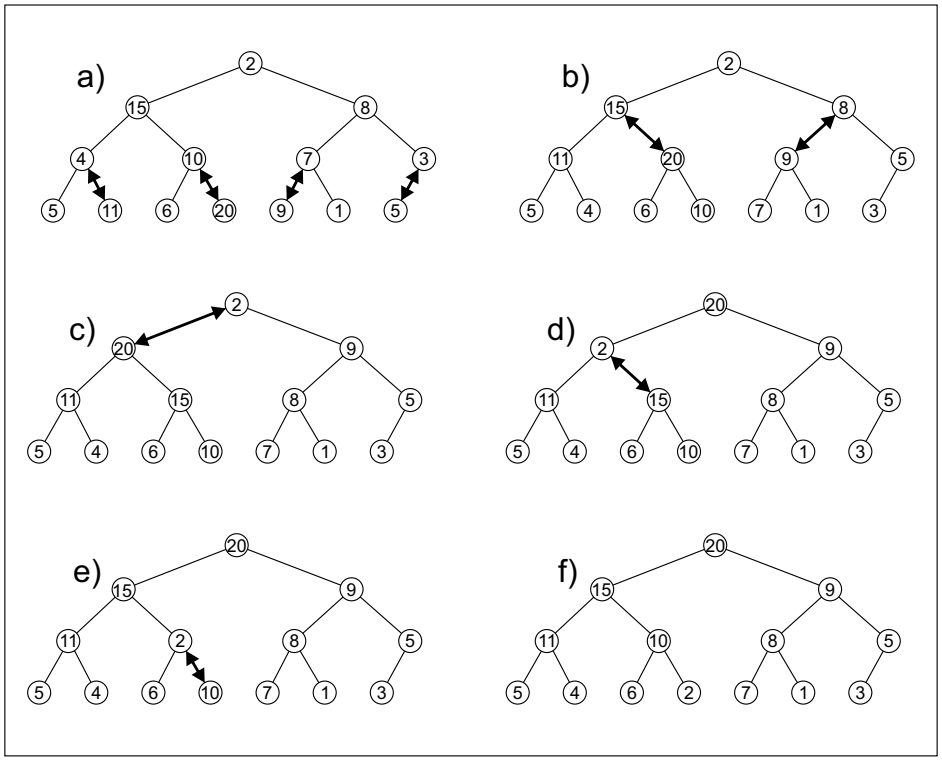
5. Gwarancją poprawnego działania algorytmu Dijkstry wyznaczania najkrótszych ścieżek w grafie jest założenie, że długości krawędzi grafu są nieujemne, $D[u][v] \geq 0$. Załóżmy, że macierz sąsiedztwa D zawiera ujemne elementy. Nasuwająca się metoda zaradzenia temu problemowi polegć może na dodaniu do wszystkich elementów D ustalonej liczby $C > 0$ tak, aby stały się one dodatnie. Po zastosowaniu algorytmu do zmodyfikowanej w powyższy sposób macierzy D, w wyznaczonej najkrótszej ścieżce między wierzchołkami s i t należałoby odjąć liczbę C od długości każdej zawartej w niej krawędzi, by uzyskać rozwiązanie wyjściowego problemu. Uzasadnij czy zaproponowana metoda jest poprawna.

Rozwiązania – grupa 2

Gr2-1. Najgorszy przypadek dla porządkowania w tablicy: ciąg wejściowy ułożony malejąco, $n, n-1, \ldots, 2, 1$. Liczba porównań w kolejnych przebiegach pętli wynosi $1+2+\cdots+n-1 = n(n-1)/2$. Liczba podstawień to $3+4+\cdots+n+n+1 = (n+1)(n+2)/2-3$.

Najgorszy wariant dla porządkowania listy: ciąg danych ułożony rosnąco $1, 2, \ldots n$, kolejne elementy trafiają zawsze na koniec listy. Liczba porównań wynosi $1+2+\cdots+n-1 = n(n-1)/2$, a więc tyle samo co dla sortowania tablicy. Liczba podstawień (dla każdej danej tworzony jest nowy element na końcu listy) $1+1+\cdots+1 = n$ (lub $4n$ jeśli liczyć także podstawienia wskaźników `nast`). Zatem liczba podstawień jest rzędu $O(n)$, a nie jak dla tablicy $O(n^2)$.

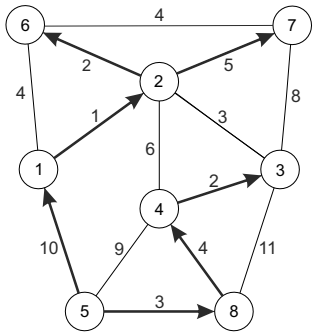
Gr2-2. Stadia tworzenia kopca:



Gr2-3.

```
wel T[M] = {0};                 // tablica wskaźników na kubełki, T[k]=NULL;
float D = (b-a)/M;              // wielkość pojedynczego kubełka

for (i = 0; i<N; i++){          // wczytanie danych
    scanf("%f", &y);
    k = (int) (y-a)/D;          // wyznaczenie numeru kubełka dla wartości y
    dodaj(&T[k], y);
}
for (k = 0; k<M; k++)           // wyświetlanie zawartości kolejnych kubełków
    for (p = T[k]; p!=NULL; p=p->nast) printf("%f\n", p->x);
```

Gr2-4. Kolejność zakwalifikowania wierzchołków jako "odwiedzone": 5, 8, 4, 3, 1, 2, 6, 7.



Gr2-5.

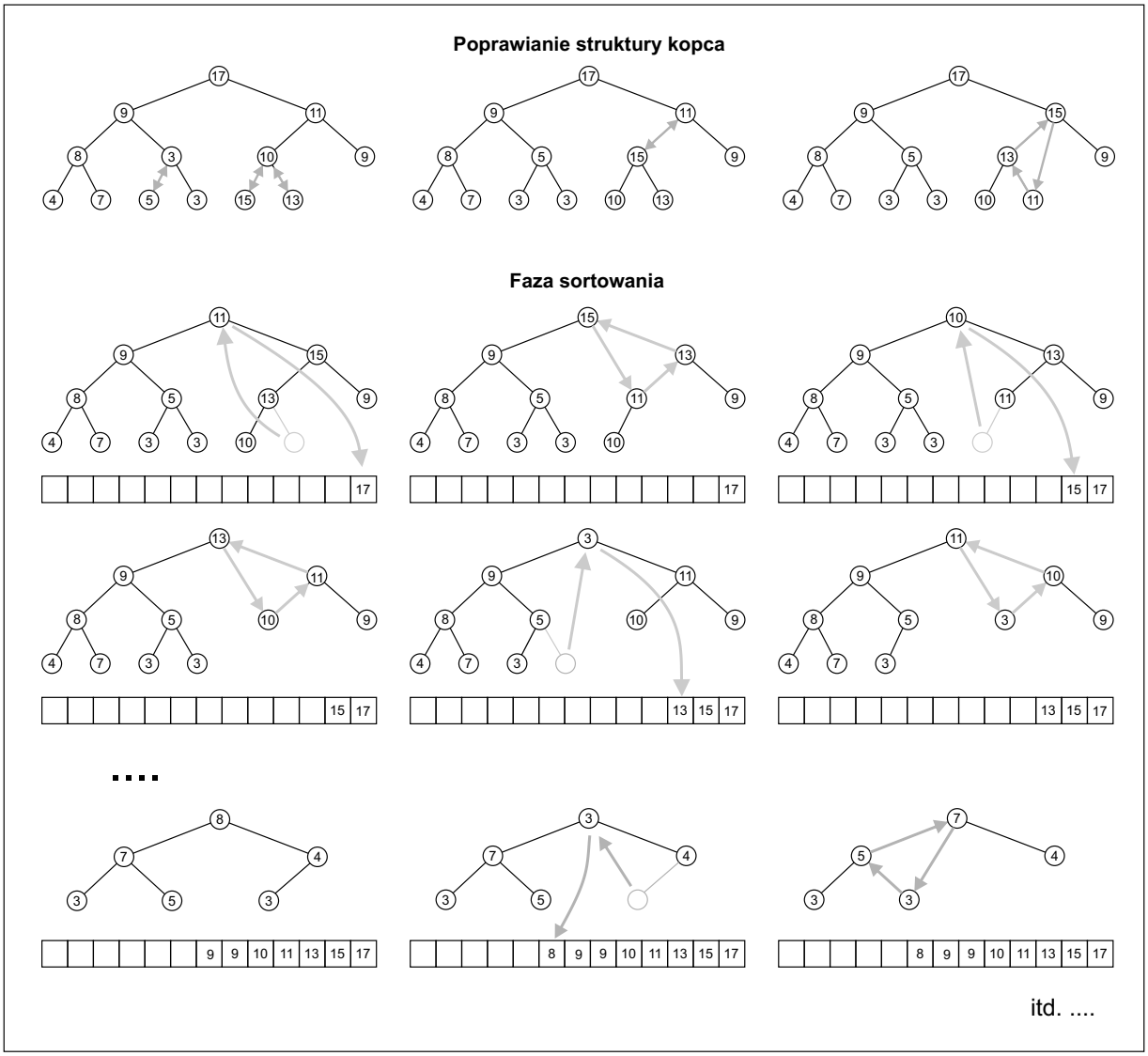
```
#define n ...
float E[n]={0};
float d, Emin=9999999.0;

for (u=1; u<=n; u++){
   for (v=u+1; v<=n; v++){
      d = dmin(u,v);
      if (d > E[u]) E[u] = d;
      if (d > E[v]) E[v] = d;
   }
   if (Emin > E[u]) Emin = E[u];
}

printf("Centrum grafu: ");
for (u=1; u<=n; u++)
   if (E[u] == Emin) printf("%i ", u);
```

Rozwiązania – grupa 3

Gr3-1. Liczba porównań jest taka sama jak w przypadku zwykłego sortowania bąbelkowego, zmiana kierunku przeglądania tablicy nic w tym zakresie nie zmienia. Pierwszy przebieg w prawo wykona $n-1$ porównań kolejno dla x_0 z x_1, x_1 z x_2, aż do x_{n-2} z x_{n-1}. Drugi przebieg postepujący w lewo wykona $n-2$ porównania, od x_{n-2} z x_{n-3} do x_1 z x_0. Kolejny w prawo od x_1 z x_2 do x_{n-3} z x_{n-2}, czyli $n-3$ porównania, itd. Łacznie jest ich $\frac{n(n-1)}{2}$, tak jak poprzednio.

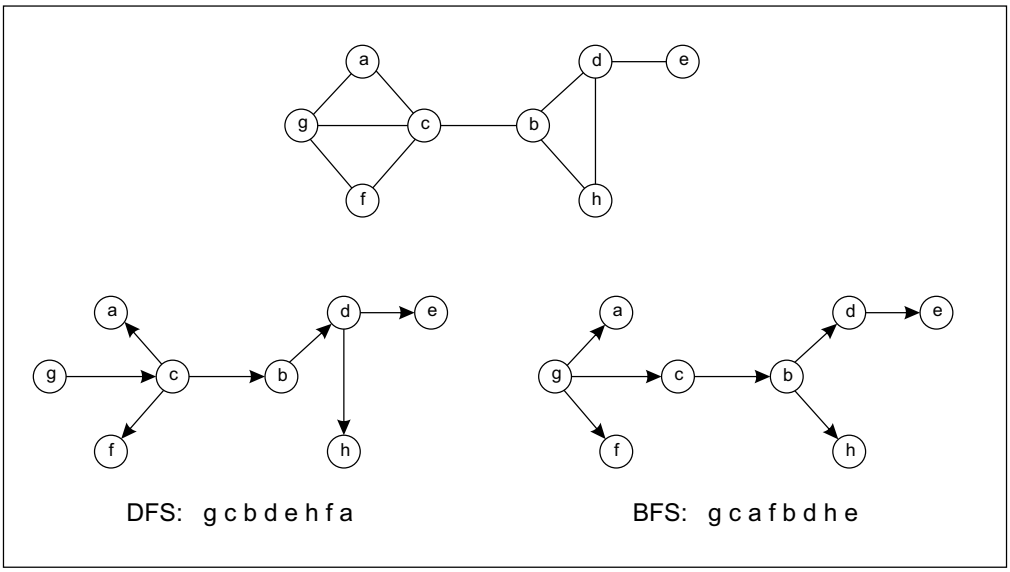
Liczba przestawień także nie ulega zmianie: ponieważ wszystkie wersje sortowania bąbelkowego wykonują jedynie zamiany sąsiednich elementów, przesuwając zawsze większe elementy w prawo, a mniejsze w lewo, ta liczba jest po prostu zadana przez wyjściową permutację sortowanych wartości $x_0, \ldots, x_{n-1}$. Wersja wahadłowa zmienia jedynie kolejność w jakiej dokonywane są zamiany sąsiednich par wartości.

Gr3-2. Sortowanie z użyciem kopca:

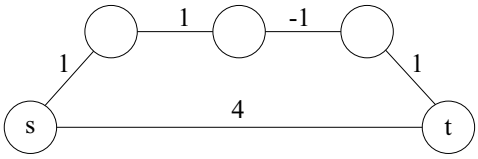


Gr3-3. P. wyżej zad. **Gr2-3**.

Gr3-4.



Gr3-5. Proponowana metoda nie jest poprawna. Pokazuje to następujący przykład, w którym $C = 1$.



Górna ścieżka z `s` do `t` o łącznej długości 2 jest krótsza, jednak po dodaniu $C = 1$ do wszystkich krawędzi, długość górnej ścieżki staje się równa 6, natomiast długość dolnej wynosi 5 i to ona zostanie wytypowana jako najkrótsza. Po odjęciu C otrzymujemy fałszywy wynik 4 jako minimalną odległość z s do t. Błąd w rozumowaniu polega na tym, że skala tymczasowej modyfikacji długości ścieżek zależy od liczby krawędzi w nich zawartych, a przecież "krótsze" w sensie wagi ścieżki mogą się składać z wielu krawędzi.