GRUPA I

1. Napisz kod funkcji `int segment(wel L)` operującej na jednokierunkowej liście nieuporządkowanej `L`, która zwraca długość najdłuższego segmentu w `L`, tj. ciągu jej elementów utworzonego przez kolejne liczby `int`. Np. dla listy

$$1 \to 2 \to 3 \to 5 \to 3 \to 2 \to 3 \to 4 \to 5 \to 3 \to \texttt{NULL}$$

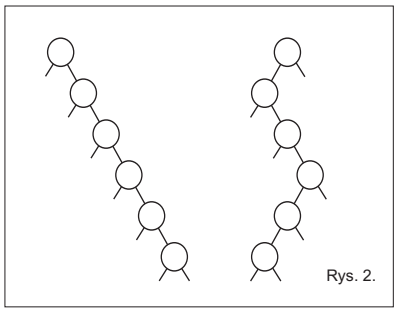
funkcja powinna zwrócić wartość 4, bo najdłuższy segment to $2 \to 3 \to 4 \to 5$.

2. Napisz funkcję logiczną `int powt(wel L)`, która zwraca wartość 1, gdy jednokierunkowa nieuporządkowana lista cykliczna wskazywana przez `L` zawiera powtarzające się elementy, lub 0 w przeciwnym razie.

3. Napisz funkcję `void odwróć(wel2 *L)`, króra dokonuje odwrócenia listy dwukierunkowej `*L`. Zakładamy tu, że lista nie jest zamknięta w cykl, a `L` jest adresem wskaźnika na jej początek. UWAGA: odwracanie listy dwukierunkowej polega na zamianie wartości pól `q->poprz` i `q->nast` w kolejnych węzłach listy. W rezultacie wskaźnik `*L` powinien odsyłać do dawnego ostatniego elementu listy, który teraz jest pierwszy.

4. Napisz kod rekursywenej funkcji `int suma_liści(weld T)`, która jako wartość zwraca sumę liczb zapisanych w liściach drzewa `T`. Gdy drzewo jest puste, funkcja zwraca wartość 0.

5. Napisz funkcję `int degen(weld T)`, która zwraca wartość 1 jeśli drzewo `T` ma zdegenerowaną strukturę, efektywnie identyczną ze zwykłą listą jednokierunkową (tak jak na rysunku).



GRUPA II

1. Skonstruuj funkcję `int krotność(wel L, int y)`, która zwraca liczbę wystąpień wartości `y` na nieuporządkowanej prostej liście `L`. Jeśli np. `krotność(L,5)==0`, oznacza to, że liczba 5 nie występuje wśród elementów `L`.

2. Napisz funkcję `void mv2end(wel2 *P, int y)`, która odnajduje w cyklicznej liście 2-kierunkowej `*P` element o wartości `y` i przenosi go na koniec listy. `P` jest adresem wskaźnika na pierwszy element listy. Wartość `*P` może więc ulec zmianie, gdy `y` znajduje się na początku listy. Jeśli `y` nie ma w `*P`, pozostaje ona niezmieniona.

3. Napisz funkcję `void invc(wel C)`, która odwraca kierunek wskaźników w cyklicznej liście jednokierunkowej `C` bez tworzenia nowych elementów. Wskaźnik `C` na głowę listy nie zmienia swojej wartości. Np. jeśli lista zawiera kolejno liczby 1, 2, 3, 4, odwrócona lista też zaczyna się od 1, ale wskaźniki na następne elementy pokazują kolejno 4, 3, 2. Ostatni element 2 wskazuje na głowę 1.

4. Napisz funkcję `int min_poziom_l(weld T)`, która zwraca minimalny numer poziomu w T, na którym znajduje się liść. Gdy `T==NULL`, funkcja zwraca 0. Korzeń drzewa ma poziom 1.

5. Napisz kod funkcji `int uporz(weld T)`, która zwraca wartość 1 jeśli jej argument `T` jest drzewem uporządkowanym lub 0, gdy `T` nie jest uporządkowane.

GRUPA III

1. Napisz funkcję `usun_powt(wel C)`, która usuwa powtarzające się elementy z nieuporządkowanej jednokierunkowej listy cyklicznej `C`. Jeśli np. lista zawiera elementy $2 \to 5 \to 1 \to 2 \to 2 \to 4 \to 1$, po wykonaniu funkcji lista `C` powinna mieć postać $2 \to 5 \to 1 \to 4$.

2. Napisz procedurę `void min2front(wel *L)`, która odnajduje minimalny element nieuporządkowanej listy jednokierunkowej `L` i przenosi go na jej początek. Wszystkie inne elementy listy pozostają na swoim miejscu.

3. Wykorzystaj funkcję `min2front` z poprzedniego zadania do posortowania nieuporządkowanej listy. Jaka jest pesymistyczna ocena złożoności (wyrażona liczbą porównań i przestawień elementów listy względem jej długości n) takiego sposobu sortowania? Odpowiedź uzasadnij.

4. Napisz funkcję `void usun_l(weld *T)`, która usuwa wszystkie liście z drzewa `*T`. Jeśli korzeń jest liściem, czyli jedynym elementem drzewa, wynikiem jest drzewo puste.

5. Należy porównać 2 *uporządkowane* drzewa pod kątem występowania w nich wspólnych elementów. Napisz funkcję logiczną `int wspólne(weld T1, weld T2, int k)`, która zwraca wartość 1 jeśli `T1` i `T2` mają przynajmniej `k` wspólnych elemenów. Interesuje mnie rozwiązanie optymalne, które przegląda każde z drzew nie więcej niż 1 raz. Można wykorzystać wywołania funkcji `weld następnik(weld T)`, takiej jak w zadaniu 7 z listy II (bez pisania tej kodu funkcji).

PRZYKŁADOWE ROZWIĄZANIA

GRUPA I

1.
```
int segment (wel L){
   int s=1, ms=0, y;
   if (L!=NULL){
      y = L->x;
      L = L->nast;
      while (L!=NULL){
         if (L->x==y+1){ s++;}
         else {
            if (s>ms) ms=s;
            s=1;
         }
         y=L->x;
         L=L->nast;
      }
   }
   return (s>ms) ? s : ms;
}
```

2.
```
int powt(wel L){
   wel p, q;
   if (L != NULL){
      p = L;
      do {
         for (q=p->nast; q!=L; q=q->nast)
            if (q->x == p->x) return 1;
         p = p->nast;
      } while (p != L)
   }
   return 0;
}
```

3.
```
void odwróć(wel2 *L){
   wel2 p=*L;
   while (p != NULL) {
      *L = p;
      p = p->nast;
      (*L)->nast = (*L)->poprz;
      (*L)->poprz = p;
   }
}
```

4.
```
int suma_liści(weld T){
      if (T==NULL) return 0;
      if ((T->l==NULL) && (T->p==NULL)) return T->x;
      return suma_liści(T->l) + suma_liści(T->p);
}
```

5.
```
int degen(weld T){
      if (T==NULL) return 1;
      return ((T->l==NULL) && degen(T->p)) || ((T->p==NULL) && degen(T->l));
```

GRUPA II

1.
```
int krotność(wel L, int y){
   int k=0;
   while (L!=NULL){
      if (L->x == y) k++;
      L = L->nast;
   }
   return k;
}
```

2.
```
void mv2end(wel2 *P, int y){
   wel2 q=*P;
   if (q!=NULL){
      while((q->x!=y) && (q->nast!=*P)) q=q->nast;
      if (q==*P){*P = (*P)->nast;}
      else{
         if (q->nast!=*P){
            q->nast->poprz = q->poprz;
            q->poprz->nast = q->nast;
            q->poprz = (*P)->poprz;
            q->nast = *P;
            (*P)->poprz->nast = q;
            (*P)->poprz = q;
         }
      }
   }
}
```

3.
```
void invc(wel C){
   welp p = C, q, r;
   if (C != NULL){
      q = C->nast;
      r = q->nast;
      do {
         q->nast = p;
         p = q;
         q = r;
         r = r->nast;
      } while (q != C);
      q->nast = p;
   }
}
```

4.
```
int min_poziom(weld T){
   if (T==NULL) return 0;
   if (T->l == NULL) return 1 + min_poziom(T->p);
   if (T->p == NULL) return 1 + min_poziom(T->l);
   return 1 + min( min_poziom(T->l), min_poziom(T->p) );
}
```

5.
```
int uporz(weld T){
   int upl=1, upp=1;
   if (T!=NULL){
      if (T->l!=NULL) upl = uporz(T->l) && (T->x >= maxT(T->l));
      if (T->p!=NULL) upp = uporz(T->p) && (T->x <= minT(T->p));
   }
   return upl && upt;
}
```

```
int maxT(weld T){
   if (T!=NULL){
      while (T->p!=NULL) T = T->p;
      return T->x;
   }
   return INT_MIN;
}
```

```
int minT(weld T){
   if (T!=NULL){
      while (T->l!=NULL) T = T->l;
      return T->x;
   }
   return INT_MAX;
}
```

GRUPA III

1.
```
int usuń_powt(wel C){
   wel p = C, *q;
   if (p!=NULL)
      while (p->nast!=C){
         q = &(p->nast);
         do{
            if ((*q)->x==p->x){
               r = *q;
               *q = r->nast;
               free(r);
            }
            else {q = &((*q)->nast;}
         } while (*q!=C);
         if (p->nast!=C) p = p->nast;
      }
   }
}
```

2.
```
void min2front(wel *L){
    wel *p, *q=L, r;
    if (*L != NULL){
       p = &((*L)->nast);
       while (*p != NULL) {
          if ((*p)->x < (*q)->x) q = p;
          p = &((*p)->nast);
       }
       if (q != L){
          r = *L;
          *L = *q;
          *q = *q->nast;
          (*L)->nast = r;
       }
    }
}
```

3.
```
void sort(wel *L){
    if (*L != NULL){
        while ((*L)->nast != NULL) {
            min2front(L);
            L=&((*L)->nast);
        }
    }
}
```

Oszacowanie złożoności: Funkcja `min2front` wykonuje tyle kroków w poszukiwaniu minimum, ile wynosi długość listy. W pierwszym przebiegu pętli `while` w funkcji `sort` jest to n kroków, w drugim $n-1$ ... itd., a w ostatnim 1. Łączna liczba kroków algorytmu wynosi więc

$$n + n-1 + \cdots + 1 = \frac{n(n+1)}{2} = O(n^2).$$

4.
```
void usun_l(weld *T){
      if (*T!=NULL) {
         if (((*T)->l == NULL) && ((*T)->p == NULL)) { free(*T); *T=NULL; }
         else {
            usun_l(&((*T)->l));
            usun_l(&((*T)->p));
         }
      }
}
```

5.
```
int wspólne(weld T1, weld T2, int k){
   weld p1 = minD(T1), p2 = minD(T2);
   while ((k>0) && (p1!=NULL) && (p2!=NULL)){
      while ((p1!=NULL) && (p1->x < p2->x)) p1 = następnik(p1);
      while ((p2!=NULL) && (p2->x < p1->x)) p2 = następnik(p2);
      if ((p1!=NULL) && (p2!=NULL)){
         if (p1->x == p2->x) k--;
         p1 = następnik(p1);
         p2 = następnik(p2);
      }
   }
   return (k==0);
}

weld minD(weld T){
   if (T!=NULL){
      while (T->l!=NULL) T = T->l;
   }
   return T;
}

weld następnik(weld T){
   weld s;
   if (T!=NULL){
      if (T->p!=NULL) return minD(T->p));
      do{
         s = T;
         T = T->rodz;
      } while (T!=NULL && (T->p == s));
   }
   return T;
}
```