Marcin Kubica
Treść zadania, Opracowanie

Marcin Kubica, Marcin Mucha
Programy

Sznurki

Przedsiębiorstwo String-Toys S.A. zwróciło się do Ciebie o pomoc w rozwiązaniu następującego problemu. String-Toys S.A. chce produkować ze sznurka modele spójnych grafów acyklicznych.

Każdy graf składa się z wierzchołków i pewnej liczby krawędzi łączących różne wierzchołki. Ten sam wierzchołek może być połączony z wieloma innymi wierzchołkami. Graf jest spójny i acykliczny, gdy od każdego wierzchołka można przejść do każdego innego wierzchołka, wędrując po krawędziach i co więcej, bez zawracania można to zrobić dokładnie na jeden sposób.

Wierzchołki grafów mają być modelowane przez węzły zawiązane na kawałkach sznurka, a krawędzie przez odcinki sznurka pomiędzy węzłami. Każdy węzeł może być wynikiem zasupłania kawałka sznurka lub związania w tym samym miejscu wielu kawałków. Ze względów technicznych koszty produkcji zależą od liczby kawałków sznurka użytych do wykonania modelu oraz od długości najdłuższego z kawałków. (Każda krawędź ma długość 1. Długość sznurka użytego do zrobienia węzłów jest pomijalna.)

Zadanie

Zadanie polega na napisaniu programu, który:

- *wczyta ze standardowego wejścia opis spójnego grafu acyklicznego, który ma być modelowany,*
- *obliczy:*
  - *minimalną liczbę kawałków sznurka potrzebnych do wykonania modelu,*
  - *zakładając, że liczba kawałków sznurka jest minimalna, minimalną długość najdłuższego kawałka sznurka potrzebnego do wykonania modelu,*
- *wypisze dwie obliczone liczby na standardowe wyjście.*

Wejście

Pierwszy wiersz zawiera jedną dodatnią liczbę całkowitą n — liczbę wierzchołków w grafie, $2 \leqslant n \leqslant 10\,000$. Wierzchołki są ponumerowane od 1 do n. Kolejne $n-1$ wierszy zawiera opisy krawędzi, po jednej w wierszu. Każdy z tych wierszy zawiera parę liczb całkowitych a i b oddzielonych pojedynczym odstępem, $1 \leqslant a,b \leqslant n$, $a \neq b$. Para taka reprezentuje krawędź łączącą wierzchołki a i b.

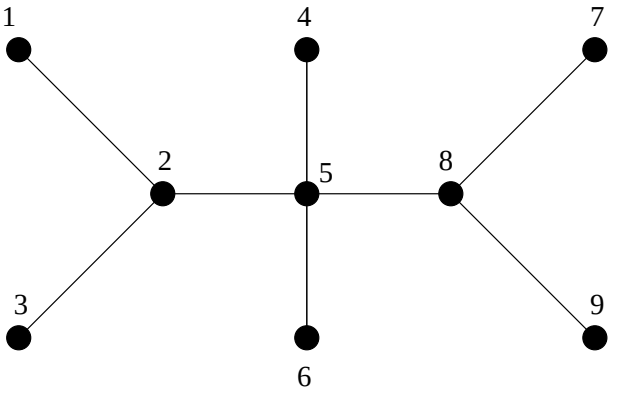
Wyjście

Twój program powinien wypisać w pierwszym wierszu wyjścia dwie liczby całkowite k i l oddzielone pojedynczym odstępem: k powinno być minimalną liczbą kawałków sznurka potrzebnych do wykonania modelu grafu, a l powinno być minimalną długością najdłuższego kawałka sznurka (zakładając, że zużyliśmy k kawałków sznurka).

Przykład

Dla danych wejściowych:

```
9
7 8
4 5
5 6
1 2
3 2
9 8
2 5
5 8
```



poprawnym wynikiem jest:

```
4 2
```

Rozwiązanie

W zadaniu należy obliczyć dwie liczby: minimalną liczbę sznurków oraz najmniejsze możliwe ograniczenie na długości sznurków, przy założeniu, że ich liczba jest minimalna. Zajmiemy się najpierw pierwszą z tych liczb, a potem drugą.

Minimalna liczba sznurków

Minimalną liczbę sznurków można łatwo obliczyć uogólniając twierdzenie Eulera ([14] p. 7.4, [23] p. 2.7, [32] par. 6). W zadaniu zajmujemy się spójnymi grafami acyklicznymi, czyli drzewami. Twierdzenie Eulera możemy zapisać następująco, używając terminologii z zadania:

Twierdzenie 1 *Jeżeli mamy dany spójny graf, w którym z każdego wierzchołka, z wyjątkiem co najwyżej dwóch, wychodzi parzysta liczba krawędzi, to model takiego grafu można wykonać z jednego kawałka sznurka. Jeżeli mamy dwa wierzchołki, z których wychodzi nieparzysta liczba krawędzi, to w wierzchołkach tych są końce sznurka. Jeżeli z każdego wierzchołka wychodzi parzysta liczba krawędzi, to sznurek tworzy pętlę (cykl).*

Zauważmy prosty fakt:

Fakt 1 Liczba wierzchołków, z których wychodzi nieparzysta liczba krawędzi jest parzysta.

Dowód Każda krawędź ma dwa końce, a łączna liczba krawędzi jest całkowita. ■

Fakt ten jest znany jako lemat o podawaniu rąk ([17] zad. 5.4-1, [32] par. 2). W szczególności nie jest możliwe, abyśmy mieli tylko jeden wierzchołek, z którego wychodzi nieparzysta liczba krawędzi.

Twierdzenie Eulera można uogólnić:

Fakt 2 Jeżeli w spójnym grafie mamy $2k$ wierzchołków, z których wychodzą nieparzyste liczby krawędzi, to model takiego grafu możemy wykonać z k sznurków.

Dowód Dodajmy do grafu na chwilę k dodatkowych krawędzi, w taki sposób, że z wszystkich wierzchołków wychodzą parzyste liczby krawędzi. Model takiego grafu możemy wykonać z jednego kawałka sznurka i to w formie pętli. Usuńmy teraz k dodanych krawędzi, rozcinając pętlę na k kawałków sznurka. Otrzymujemy model oryginalnego grafu. ■

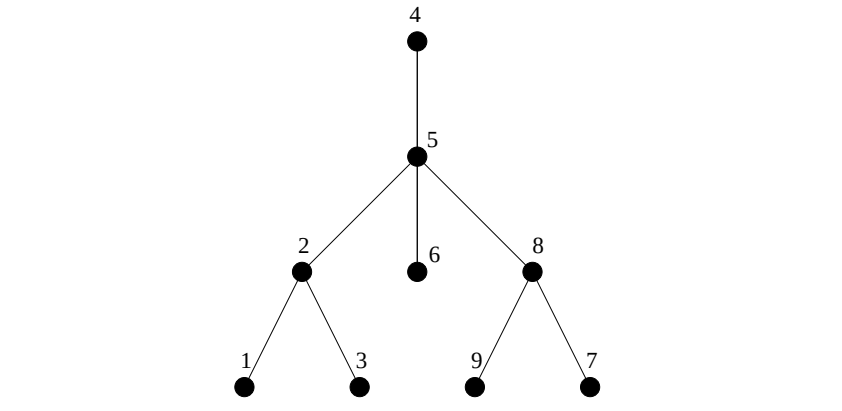
Tak więc, aby obliczyć minimalną liczbę sznurków, wystarczy policzyć, ile jest takich wierzchołków, z których wychodzi nieparzysta liczba krawędzi i podzielić tę liczbę przez 2.

W zadaniu zajmujemy się drzewami, czyli spójnymi grafami acyklicznymi. W przypadku drzew można prosto opisać, jak może wyglądać model drzewa. W każdym wierzchołku, z którego wychodzi nieparzysta liczba krawędzi umieszczamy jeden koniec sznurka i wyprowadzamy go wzdłuż dowolnej krawędzi. Dla każdego wierzchołka pozostaje parzysta liczba krawędzi, które modelujemy sznurkiem przechodzącym przez wierzchołek. Dla każdego wierzchołka, krawędzie takie łączymy w dowolny sposób w pary i przeprowadzamy sznurki zgodnie z podziałem na pary. Każda taka konstrukcja daje model drzewa wykonany z minimalnej liczby kawałków sznurka.

Ograniczenie na długość kawałków sznurka

Wiemy już, jak mogą wyglądać modele drzew wykonane z minimalnej liczby kawałków sznurka. Zastanówmy się, jak poprowadzić sznurki tak, aby długość najdłuższego sznurka była jak najmniejsza.

Dla uproszczenia rozważań ukorzenimy nasze drzewo. Na korzeń wybieramy dowolny liść (wierzchołek, z którego wychodzi tylko jedna krawędź). Ukorzenienie możemy sobie wyobrazić jako chwycenie modelu grafu za korzeń i podniesienie do góry tak, że cały model swobodnie zwisa. Wierzchołek znajdujący się bezpośrednio powyżej danego wierzchołka nazywamy jego przodkiem, a wierzchołki znajdujące się bezpośrednio poniżej nazywamy potomkami. Poniższy rysunek przedstawia ukorzenione drzewo z przykładu z treści zadania, gdzie jako korzeń wybrano wierzchołek nr 4.



Ukorzenione drzewo.

Poniżej przedstawiamy procedurę, która wczytuje dane wejściowe, oblicza minimalną liczbę kawałków sznurka potrzebnych do wykonania modelu oraz konstruuje ukorzenione drzewo[1]. Wykorzystuje ona algorytm przeszukiwania grafów w głąb (DFS).

```
(* Typ reprezentujący dowolne drzewo ukorzenione. *)
type drzewo = Wezel of drzewo list;;
(* Wczytanie danych. Wynikiem jest drzewo i liczba sznurków. *)
let wczytaj_dane () =
  (* Liczba węzłów. *)
  let n = scanf "%d " id
  in
    let sasiedzi = make (n+1) []
    and odwiedzone = make (n+1) false
    and konce = ref 0
    in
      (* Wczytanie krawędzi. *)
      (* Wynikiem jest korzeń - jeden z liści. *)
      let rec wczytaj () =
        let i = ref 1
        in
          begin
            (* Wczytanie krawędzi. *)
            for j = 1 to n-1 do
              let a = scanf "%d " id
              and b = scanf "%d " id
              in begin
                sasiedzi.(a) <- b::sasiedzi.(a);
                sasiedzi.(b) <- a::sasiedzi.(b)
              end
            done;
            (* Końce sznurków. *)
            for j = 1 to n do
              if length sasiedzi.(j) mod 2 = 1 then
                konce := !konce + 1
            done;
            (* Wybór korzenia - jeden z liści. *)
            while length sasiedzi.(!i) > 1 do i:= !i + 1 done;
            !i
          end
      (* Konstrukcja drzewa - DFS. *)
      and dfs v =
        begin
          odwiedzone.(v) <- true;
```

[1]Procedura ta pochodzi z pliku `szn.ml`, który można znaleźć na załączonym dysku CD-ROM. Rozwiązanie to jest napisane w języku Ocaml. (Opis języka Ocaml, dystrybucję kompilatora oraz inne informacje na ten temat można znaleźć na stronie `http://caml.inria.fr/ocaml/`.) Na załączonym dysku można również znaleźć rozwiązanie napisane w języku C++: `szn.cpp`.

```
40:           Wezel
41:             (fold_left
42:               (fun a w -> if odwiedzone.(w) then a else (dfs w)::a)
43:               [] sasiedzi.(v))
44:         end
45:       in
46:         let korzen = wczytaj ()
47:         in (dfs korzen, !konce / 2);;
```

Modele drzewa mogą mieć następującą postać:

1. Jeżeli dany wierzchołek ma nieparzystą liczbę potomków i nie jest to korzeń, to sznurek wychodzący z jednego z jego potomków prowadzi do jego przodka. Sznurki wychodzące z pozostałych potomków są połączone ze sobą parami.
2. Jeżeli dany wierzchołek ma parzystą liczbę potomków, to jeden ze sznurków ma w nim koniec. Możliwe są dwa przypadki:
   (a) Sznurki wychodzące z potomków są połączone ze sobą parami. Do przodka prowadzi sznurek o końcu w danym wierzchołku.
   (b) Do przodka prowadzi sznurek wychodzący od jednego z potomków. Sznurek od innego z potomków kończy się w danym wierzchołku. Sznurki wychodzące z pozostałych potomków są połączone ze sobą parami.
3. Korzeń wybraliśmy tak, że ma tylko jednego potomka. Sznurek wychodzący z niego kończy się w korzeniu.

Problem wyznaczenia minimalnej długości najdłuższego kawałka sznurka można rozwiązać stosując programowanie dynamiczne. Dla każdego wierzchołka v możemy rozważyć poddrzewo o korzeniu w tym wierzchołku. Z każdego takiego poddrzewa (z wyjątkiem całego drzewa) jeden sznurek wychodzi do góry, a dodatkowo pewne sznurki mogą być w nim zawarte w całości. Niech d będzie ograniczeniem na długość sznurków całkowicie zawartych w rozważanym poddrzewie. Przez $S(v,d)$ oznaczymy minimalną długość sznurka wychodzącego z poddrzewa o korzeniu w v do góry (a dokładniej tej jego części, która jest zawarta w rozważanym poddrzewie). Jeżeli nie jest możliwe, żeby sznurki zawarte w całości w poddrzewie nie były dłuższe niż d, to przyjmujemy $S(v,d) = \infty$. Zauważmy, ze funkcja S jest niemalejąca, tzn. im ostrzejsze ograniczenie nakładamy na sznurki zawarte w całości w poddrzewie, tym dłuższy może być sznurek wychodzący z poddrzewa.

Niech $\varphi(v)$ będzie długością najdłuższej ścieżki w poddrzewie o korzeniu v (tzw. średnica poddrzewa). Interesować nas będą wartości funkcji $S(v,d)$ dla $d = 0,1,\ldots,\varphi(v)$. Niech r będzie korzeniem drzewa. Wówczas szukane ograniczenie na długość kawałków sznurka to:

$$\min_{d=0,1,\ldots,\varphi(r)} (\max(d, S(r,d)))$$

Niech v będzie wierzchołkiem w grafie, dla którego wiemy już jak przebiega sznurek wychodzący z v do góry. Oznaczmy przez $p(v)$ liczbę potomków v. Pozostaje nam parzysta liczba sznurków wychodzących z potomków v: $w_1, w_2, \ldots, w_{p(v)}$, które należy połączyć w pary. Zastanówmy się, jak sprawdzić, czy da się tak połączyć te sznurki, aby długość żadnego sznurka zawartego w poddrzewie nie przekraczała zadanego ograniczenia d? Pokażemy, że można je łączyć na zasadzie największy z najmniejszym.

Fakt 3 Niech $p(v)$ będzie parzyste i niech $l_1 \leqslant l_2 \leqslant \ldots \leqslant l_{p(v)}$ będą minimalnymi długościami sznurków wychodzących z w_1, w_2, …, $w_{p(v)}$, przy założeniu, że żaden sznurek całkowicie zawarty w poddrzewach o korzeniach w_1, w_2, …, $w_{p(v)}$ nie jest dłuższy niż d. Wówczas sznurki wychodzące z w_1, w_2, …, $w_{p(v)}$ można połączyć w pary tak, aby żaden z nich nie był dłuższy niż d wtedy i tylko wtedy, gdy $l_1 + l_{p(v)} \leqslant d$, $l_2 + l_{p(v)-1} \leqslant d$, … oraz $l_{\frac{p(v)}{2}} + l_{\frac{p(v)}{2}+1} \leqslant d$.

Dowód Załóżmy, że można sznurki połączyć w pary tak, aby ich długości nie przekraczały d. Jeżeli sznurek wychodzący z w_1 nie jest połączony ze sznurkiem wychodzącym z $w_{p(v)}$, to niech i i j będą takie, że sznurek wychodzący z w_1 jest połączony ze sznurkiem wychodzącym z w_i, a sznurek wychodzący z $w_{p(v)}$ będzie połączony ze sznurkiem wychodzącym z w_j, $i \neq j$.

Zauważmy, że zachowując ograniczenie d na długość sznurków możemy zamienić połączenia i połączyć sznurek wychodzący z w_1 ze sznurkiem wychodzącym z $w_{p(v)}$, oraz sznurek wychodzący z w_i ze sznurkiem wychodzącym z w_j. Skoro $l_1 + l_i \leqslant d$, $l_{p(v)} + l_j \leqslant d$ oraz $l_1 \leqslant l_i, l_j \leqslant l_{p(v)}$, to $l_i + l_j \leqslant d$ oraz $l_1 + l_{p(v)} \leqslant d$. Analogicznie zamieniamy połączenia pozostałych sznurków. ■

Fakt ten pozwala nam zaimplementować zachłanny algorytm sprawdzający, czy sznurki o danych długościach można połączyć w pary tak, aby ich długości nie przekroczyły danego ograniczenia. Algorytm ten jest ujęty w postaci procedury `pary`. Procedura ta ma dwa parametry `l` i `d` — `l` to lista długości sznurków uporządkowana nierosnąco, `d` to ograniczenie na długości sznurków. Procedura ta korzysta z dwóch procedur pomocniczych: `ile_par` i `cut`. Procedura `ile_par` ma trzy parametry: dwie listy liczb oraz ograniczenie `d`. Bada ona, ile kolejnych elementów z danych list można połączyć w pary o sumach nie przekraczających `d`. Procedura `cut` ma charakter pomocniczy i wybiera z podanej listy `l` elementy od `i`-tego do `j`-tego. Procedury `ile_par` i `cut` zostały wydzielone, gdyż będziemy z nich jeszcze korzystać dalej.

```
1:  (* Wybiera z listy fragment od i-tego do j-tego elementu. *)
2:  let cut i j l =
3:    let rec cut_iter i j l a =
4:      if i > 1 then
5:        cut_iter (i-1) (j-1) (tl l) a
6:      else if j > 0 then
7:        cut_iter i (j-1) (tl l) (hd l :: a)
8:      else
9:        rev a
10:   in
11:     cut_iter i j l [];;
12: (* Sprawdza ile pierwszych elementów z obu list *)
13: (* można połączyć w pary o sumach <= d.         *)
14: let ile_par l1 l2 d =
15:   let rec ile_par_iter l1 l2 a =
16:     match (l1, l2) with
17:       ([], []) -> a |
18:       (h1::t1, h2::t2) ->
19:         if h1 + h2 + 2 <= d then
```

```
20:           ile_par_iter t1 t2 (a+1)
21:         else a |
22:       _ -> a
23:   in
24:     ile_par_iter l1 l2 0;;
25: (* Sprawdź czy elementy listy posortowanej *)
26: (* nierosnąco można połączyć w pary <= d   *)
27: let pary l d =
28:   let dl = length l
29:   in
30:     if dl mod 2 = 0 then
31:       ile_par (cut 1 (dl / 2) l)
32:               (rev (cut (dl / 2 + 1) dl l)) d = dl / 2
33:     else false;;
```

Zastanówmy się, jak można obliczyć wartości funkcji $S(v,d)$. Rozważmy najpierw przypadek, gdy v ma nieparzystą liczbę potomków. Niech $l_1 \leqslant l_2 \leqslant \ldots \leqslant l_{p(v)}$ będą minimalnymi długościami sznurków wychodzących z potomków v. Zauważmy, że jeżeli nie jesteśmy w stanie połączyć w pary $l_1, l_2, \ldots, l_{p(v)}$ tak, aby ich sumy nie przekraczały d (przypadek, gdy do góry przechodzi najdłuższy ze sznurków), to $S(v,d) = \infty$. Jeżeli można to zrobić, to możemy dalej sprawdzać, jaki jest najkrótszy wychodzący z poddrzew sznurek, który może przechodzić do góry, tzn. taki, że pozostałe sznurki możemy połączyć w pary o długościach nie większych niż d. Sprawdzamy kolejno coraz krótsze sznurki. Powiedzmy, że sprawdziliśmy, iż do góry mogą wychodzić sznurki o długościach l_{i+1}, ..., $l_{p(v)}$ i chcemy sprawdzić, czy może wychodzić do góry sznurek o długości l_i. Korzystając z faktu 3, wystarczy sprawdzić jeden z dwóch warunków:

- $l_{i+1} + l_{p(v)-i} + 2 \leqslant d$, dla $i \geqslant \frac{p(v)+1}{2}$,
- $l_{i+1} + l_{p(v)+1-i} + 2 \leqslant d$, dla $i < \frac{p(v)+1}{2}$.

Możemy więc napisać następującą procedurę obliczającą $S(v,d)$ w przypadku, gdy v ma nieparzystą liczbę potomków.

```
1: (* S(v,d) dla nieparzystej liczby potomków v,   *)
2: (* na podstawie d oraz [S(wp,d); ...; S(w1,d)]. *)
3: let nieparzyste ls d =
4:   let dl = length ls
5:   in
6:     if pary (tl ls) d then
7:       (* Najdłuższy sznurek może wychodzić do góry. *)
8:       (* Szukamy krótszego.                         *)
9:       let i = ile_par (cut 1 (dl/2) ls)
10:                       (rev (cut (dl/2 + 2) dl ls)) d
11:      in
12:        if i = dl/2 then
13:          (* Środkowy (co do długości) sznurek może *)
14:          (* wychodzić do góry. Szukamy krótszego.  *)
```

```
15:           let j = ile_par (cut (dl/2 + 1) (dl-1) ls)
16:                           (rev (cut 1 (dl/2) ls)) d
17:           in
18:             nth ls (j + dl/2) + 1
19:         else nth ls i + 1
20:     else
21:       (* Nie da się. *)
22:       inf
```

Załóżmy teraz, że wierzchołek v ma parzystą liczbę potomków. Jeżeli sznurki wychodzące z tych potomków można połączyć w pary tak, aby ich długości nie przekraczały d, to sznurek wychodzący z v do góry może mieć początek w v. Jeżeli tak nie jest, to jeden ze sznurków wychodzących z potomków v musi się kończyć w v, a jeden musi iść dalej do góry. Pozostałe sznurki są połączone w pary ze sobą.

Jeżeli $l_{p(v)} + 1 \leqslant d$, to najkorzystniej jest dokonać wyboru tak, aby w v kończył się najdłuższy sznurek, wychodzący z $w_{p(v)}$. Wówczas problem redukuje się do problemu, gdy v ma nieparzystą liczbę potomków.

Jeżeli $l_{p(v)} + 1 > d$, to sznurek wychodzący z $w_{p(v)}$ musi przechodzić przez v do góry. Podobnie, problem redukuje się wówczas do przypadku, gdy v ma nieparzystą liczbę potomków — musimy sprawdzić, iż pozostałe sznurki można połączyć w pary (z wyjątkiem jednego, który ma koniec w v) tak, że ich długości nie przekraczają d. W przeciwnym przypadku $S(v,d) = \infty$.

Powyższe obserwacje przekładają się na następującą procedurę:

```
1: (* S(v,d) dla parzystej liczby potomków v,      *)
2: (* na podstawie d oraz [S(wp,d); ...; S(w1,d)]. *)
3: let parzyste ls d =
4:   if pary ls d then
5:     (* Z v wychodzi nowy sznurek. *)
6:     0
7:   else
8:     if hd ls + 1 <= d then
9:       (* Najdłuższy z wychodzących sznurków kończymy, *)
10:      (* a pozostałych mamy nieparzystą liczbę.       *)
11:      nieparzyste (tl ls) d
12:    else
13:      if nieparzyste (tl ls) d <= d then
14:        (* Najdłuższy z wychodzących sznurków przechodzi  *)
15:        (* do góry, o ile pozostałe nie są dłuższe niż d. *)
16:        hd ls + 1
17:      else
18:        (* Nie da się. *)
19:        inf;;
```

Poniższa procedura łączy oba przypadki: dla $p(v)$ parzystego i nieparzystego.

```
1: (* Obliczenie S(v,d) na podstawie d oraz [S(w1,d); ...; S(wp,d)]. *)
2: let s d l =
```

```
3:   (* Posortowana nierosnąco lista  [S(w1,d); ...; S(wp,d)]. *)
4:   let l_sort =
5:     let por x y = if x < y then 1 else if x > y then -1 else 0
6:     in sort por l
7:   in
8:     if length l mod 2 = 1 then
9:       (* Nieparzysta liczba potomków. *)
10:      nieparzyste l_sort d
11:    else
12:      (* Parzysta liczba potomków. *)
13:      parzyste l_sort d;;
```

Ze względu na konieczność zastosowania sortowania, procedura ta ma złożoność czasową rzędu $O(p(v) \cdot \log p(v))$. Jej złożoność pamięciowa jest rzędu $O(p(v))$. Obliczenie wszystkich wartości $S(v,0), S(v,1), \ldots, S(v,\varphi(v))$ wymaga czasu rzędu $O(p(v) \cdot \log p(v) \cdot \varphi(v))$. Jak jednak obliczać $\varphi(v)$?

Zastanówmy się gdzie może znajdować się najdłuższa ścieżka w poddrzewie o korzeniu v? Pierwsza możliwość jest taka, że ścieżka ta nie przechodzi przez v — mieści się ona całkowicie w jednym z poddrzew, których korzeniami są potomkowie v. Druga możliwość jest taka, że ścieżka ta przechodzi przez v. Wówczas, o ile v ma przynajmniej dwóch potomków, to ścieżka ta łączy dwa najgłębiej położone liście w dwóch najwyższych poddrzewach v. Tak więc, żeby obliczyć $\varphi(v)$ musimy znać wysokości potomków v oraz średnice poddrzew, których korzeniami są potomkowie v: $\varphi(w_1), \varphi(w_2), \ldots, \varphi(w_{p(v)})$.

Oto procedura, która rekurencyjnie przegląda wczytane drzewo, dla każdego wierzchołka v oblicza wysokość poddrzewa o korzeniu v, jego średnicę, oraz wartości $S(v,0), S(v,1), \ldots, S(v,\varphi(v))$.

```
1:  (* Analiza danego drzewa.           *)
2:  (* Obliczane są:                   *)
3:  (* - wysokość drzewa h,            *)
4:  (* - średnica drzewa sr,           *)
5:  (* - sl = [S(v,0); ... ; S(v, sr)]. *)
6:  let rec analiza (Wezel l) =
7:    (* Poddaj analizie poddrzewa. *)
8:    let w = map analiza l
9:    in
10:     (* Wysokość i średnica drzewa. *)
11:     let (h, sr)  =
12:       (* Dwie największe wysokości poddrzew. *)
13:       let (h1, h2) =
14:         fold_left (fun (a1, a2) (h, _, _) ->
15:           if h >= a1 then (h, a1) else
16:           if h >= a2 then (a1, h) else (a1, a2))
17:           (-1, -1) w
18:       in
19:         (h1 + 1,
20:          fold_left (fun a (_,s,_) -> max a s) (h1 + h2 + 2) w)
```

```
21:     in
22:       (* [S(v,0); ... ; S(v, sr)] *)
23:       let sl =
24:         (* Iteracja obliczająca S(v, d) dla kolejnych d. *)
25:         let rec s_iter d ll ak =
26:           (* Ogon listy, przy czym jeśli pusty, to *)
27:           (* dublowana jest głowa.                 *)
28:           let tail l =
29:             match l with
30:               [x]  -> [x] |
31:               h::t -> t |
32:               _ -> failwith "tail: l = []"
33:           in
34:             if d > sr then rev ak else
35:               s_iter (d+1)
36:                 (map tail ll)
37:                 ((s d (map hd ll)) :: ak)
38:         in
39:           s_iter 0 (map (fun (_, _, l) -> l) w)[]
40:       in (h, sr, sl);;
```

Jaka jest złożoność tej procedury? Obliczenie wysokości i średnicy poddrzewa o korzeniu v na podstawie wysokości i średnic poddrzew o korzeniach w potomkach v wymaga czasu liniowego ze względu na $p(v)$. Tak więc dominujący jest czas potrzebny na obliczenie wartości $S(v,d)$, czyli:

$$\sum_{v=1,2,\ldots,n} (p(v)\cdot\log p(v)\cdot\varphi(v)) \leqslant n\cdot\sum_{v=1,2,\ldots,n} (p(v)\cdot\log p(v)) = O(n^2\log n).$$

Złożoność pamięciowa tej procedury jest rzędu $\Theta(n)$. Z jednej strony konstrukcja drzewa wymaga takiej pamięci. Z drugiej strony, łączna liczba wartości funkcji S, jakie musimy równocześnie pamiętać, nie jest większa niż liczba przeanalizowanych już wierzchołków drzewa.

Procedura obliczająca końcowy wynik ma postać:

```
1: (* Minimalne ograniczenie na długość sznurków. *)
2: let ograniczenie t =
3:   let (_, sr, sl) = analiza t
4:   in
5:     snd
6:       (fold_left
7:         (fun (d, m) s -> (d+1, min (max d s) m))
8:         (0, inf) sl);;
```

Złożoność całego algorytmu pozostaje taka sama, jak złożoność procedury `analiza`, czyli złożoność czasowa jest rzędu $O(n^2\log n)$, a pamięciowa rzędu $O(n)$.

Testy

Rozwiązania zawodników były testowane na dziesięciu testach: jednym małym teście poprawnościowym oraz dziewięciu, coraz większych testach losowych. Poniższa tabela przedstawia wielkości tych testów.

nr testu	n
1	19
2	143
3	397
4	1 431
5	2 101
6	2 837
7	4 041
8	7 230
9	9 115
10	10 000