Zadania domowe. Zestaw 3.2

Maciej Poleski

18 grudnia 2012

1

Dla grafu bez wierzchołków potrzeba zero wyjść ewakuacyjnych. Dla grafu z jednym jedno. Dla pozostałych grafów co najmniej dwa (jeżeli przepadnie wierzchołek z wyjściem ewakuacyjnym to na pewno istnieje inny wierzchołek który potrzebuje jakiegoś innego wyjścia ewakuacyjnego). Od tej pory zakładam, że graf ma co najmniej dwa wierzchołki. Każde przyporządkowanie wyjść ewakuacyjnych które uwzględnia możliwość katastrofy jedynie punktów artykulacji jest dobrym przyporządkowaniem dla całego grafu (jeżeli dojdzie do katastrofy nie punktu artykulacji to na pewno istnieje inny punkt który jest wyjściem ewakuacyjnym, a graf nie został rozspójniony). Zastanówmy się jak wygląda sytuacja w przypadku drzewa (liściem nazywamy wierzchołek o stopniu 1 - korzeń może być liściem). Jeżeli dojdzie do katastrofy wierzchołka który jest sąsiadem liścia to potrzebujemy co najmniej tyle wyjść ewakuacyjnych ile ten wierzchołek ma sąsiadów (w szczególności każdy liść który jest sąsiadem tego wierzchołka musi być wyjściem ewakuacyjnym). Rozpatrując wszystkie wierzchołki dochodzimy do wniosku że każdy liść musi być wyjściem ewakuacyjnym. I tyle wystarczy. Jeżeli dojdzie do katastrofy liścia - graf pozostaje spójny oraz istnieje inne wyjście. Jeżeli dojdzie do katastrofy nie liścia - graf rozspójnia się, ale w każdej spójnej składowej jest co najmniej jeden liść (będący jednocześnie liściem drzewa wyjściowego). Czyli potrzebujemy tyle wyjść ile jest liści i istnieje dokładnie jeden sposób ich rozmieszczenia. Teraz przejdźmy do sytuacji ogólnej. Mamy graf (spójny - wszystkie wyjściowe spójne składowe możemy rozważyć osobno i wyniki przemnożyć ponieważ są one od siebie niezależne). Możemy skondensować dwuspójne składowe do wierzchołków. W efekcie redukujemy problem do drzewa. Warto zauważyć, że rozspójniamy takie drzewo dokładnie wtedy gdy usuwamy punkt artykulacji z grafu wyjściowego. Jeżeli w grafie nie ma punktu artykulacji to mamy $\binom{n}{2}$ możliwości wyboru 2 wierzchołków. Jeżeli istnieje jakikolwiek punkt artykulacji to po skondensowaniu uzyskamy drzewo o co najmniej 2 wierzchołkach (czyli zgodne z wcześniejszymi rozważaniami). W takim razie odpowiedzią jest ilość liści w takim drzewie i iloczyn "rozmiarów" liści pomniejszonych o 1 (punkt artukulacji którego wybór nie ma sensu).

```
zakładam że graf jest spójny
n <- liczba wierzchołków
if n < 2:
   print n 1
else:
   wyznacz dla każdej krawędzi dwuspójną skłądową oraz punkty artykulacji
```

```
    if nie istnieje punkt artykulacji:
        print 2 n*(n-1)/2
    else:
        wynik <- 1
        l <- 0
        wyznacz wierzchołki w każdej dwuspójnej składowej
        for each s in zbiór dwuspójnych składowych:
            if istnieje dokładnie jeden punkt artykulacji w s:
                l <- l + 1
                wynik <- wynik * (rozmiar(s) - 1)
        print l wynik
```

2

Do policzenia w trakcie wyszukiwania punktów artykulacji. Po usunięciu punktu artykulacji dzieci drzewowe będą osobnymi składowymi. Przemnożę odpowiednie rozmiary poddrzew (składowych) aby uzyskać liczbę uporządkowanych par wierzchołków które nie są już połączone.

```
n <- ilość wierzchołków
counter <- 0
subtreeSize[1..n]
V[1..n] <- false
pre[1..n]
low[1..n]
ans[1..n] <- 0
 dfs(v,p):
    subtreeSize <- 1
    childSize <- pusta lista
    pre[v] <- low[v] <- counter
    V[v] <- true
    counter <- counter + 1
    art <- false
    childCount <- 0
    for each u ∈ Adj[v]:
        if u = p:
            continue
        if V[u] = false:
            ch <- dfs(u,v)
            subtreeSize <- subtreeSize + ch
```

```
            childCount <- childCount + 1
            low[v] <- min(low[v],low[u])
            if low[u]>=pre[v] and p != 0:
                art <- true
                childSize.append(ch)
        else:
            low[v] <- min(low[v],pre[u])
    if p==0 and childCount>1:
        art <- true
    if art:
        totalSum <- 0
        for each c ∈ childSize:
            totalSum <- totalSum + c
            ans[v] <- ans[v] + (n-c-1)*c
        ans[v] <- ans[v] + (n-1-totalSum)*totalSum
```

Odpowiedź jest zapisana dla każdego wierzchołka w tablicy `ans`. Jest to liczba par uporządkowanych.

3

Zbiór E w połączeniu ze zbiorem wyjściowym wyznacza jednoznacznie graf. Pozostaje określić czy jest możliwe aby dfs przeglądał wierzchołki tego grafu w zadanej kolejności. Narzuca się symulacja. Dla każdej pary wierzchołków będę chciał odpowiedzi na pytanie czy są połączone krawędzią w czasie stałym oraz przeglądać listę sąsiedztwa wierzchołka w czasie proporcjonalnym do ilości sąsiadów. Można to osiągnąć przechowując jednocześnie macierz sąsiedztwa i listy sąsiedztwa.

```
P[0..n-1] <- permutacja podana na wejściu
i <- 1

V[1..n] <- false

 def dfs(v):
    V[v] <- true
    while istnieje nieodwiedzony sąsiad:
        if (v,P[i]) ∈ E:
            i <- i+1
            dfs(P[i-1]) # nie zapętlimy się bo P[0..n-1] jest permutacją
```

```
        else:
            NIE

dfs(P[0])
TAK jeżeli wcześniej nie zadecydowano inaczej
```

W jaki sposób szybko zrealizować test `istnieje nieodwiedzony sąsiad`: sprawdzamy najpierw czy mamy możliwość odwiedzenie wybranego sąsiada. Jeżeli tak - odwiedzamy, jeżeli nie to istnieje szansa że graf jest ok tylko wtedy gdy wszyscy sąsiedzi zostali odwiedzeni. Sprawdzamy to przeglądając jednokrotnie listę sąsiedztwa. Albo natychmiast odpowiemy NIE, albo zakończyliśmy obsługę tego wierzchołka czyli nigdy więcej nie przejdziemy po tej liście sąsiedztwa. W efekcie sumaryczna złożoność $O(|E|)$.