Mikołaj Korobczak

Algorytmy i struktury danych

Zadanie 6 lista 2

Wrocław, 9 kwietnia 2020

Prowadzący: mgr Adam Kunysz

1. Treść zadania

Ułóż algorytm, który dla danego spójnego grafu G oraz krawędzi e sprawdza w czasie $O(n+m)$, czy krawędź e należy do jakiegoś minimalnego drzewa spinającego grafu G. Możesz założyć, że wszystkie wagi krawędzi są różne.

2. Własności MST

Aby pokazać szukany algorytm i zrozumieć, że działa on poprawnie należy powiedzieć o kilku własnościach MST.

— Jeżeli założymy, że nasz graf G ma krawędzie o parami różnych wagach to istnieje tylko jedno MST.

— Jeżeli e jest mostem w grafie G to e należy do MST.

Ponadto warto udowodnić pewien przydatny lemat.

Lemat 1. *Dla dowolnej krawędzi e w grafie G, e jest w MST grafu G wtedy i tylko wtedy, gdy dla dowolnego cyklu na którym leży krawędź e, krawędź ta nie jest krawędzią o największej wadze spośród wszystkim na tym cyklu.*

Dowód.

$\Rightarrow$ Załóżmy, że e należy do MST. Weźmy cykl C na którym leży e i załóżmy nie wprost, że e jest na nim krawędzią o największej wadze. Wiemy, że istnieje krawędź f na cyklu C, której nie ma w MST. Jeżeli usuniemy z MST krawędź e i dodamy krawędź f to drzewo MST nie przstanie być spójne, ponieważ nadal będziemy patrzeć na cykl bez jednej krawędzi, a taka struktóra jest grafem spójnym. Nasze nowe drzewo MST ma mniejszą wagę niż poprzednio więc sprzeczność. $\lightning$

$\Leftarrow$ Załóżmy, że e leży na cyklu C i nie jest na nim krawędzią o największej wadze (i jeżeli leży na innych cyklach, to na nich też nie jest krawędzią o największej wadze). Rozważmy algorytm Kruskala (zakładam, że jest on poprawny). Algorytm ten będzie dodawał kolejne krawędzi z cyklu C to MST dopóki cały cykl będzie spójny, ale nie będzie cyklem. Ponieważ taka sytuacja zajdzie dopiero, gdy zostanie jedna krawędź z tego cyklu poza MST i ponieważ algorytm ten dodaje krawędzie rosnąco po wagach, to krawędź e zostanie dodana. Więc $e \in$ MST.

$\square$

3. Algorytm

Algorytm dostaje graf G oraz krawędź e, jak również tablice wag $w[]$.
Nasz algorytm będzie zmodyfikowanym algorytmem DFS.

```
we := w[e]
REMOVE e FROM G
```

Niech $e = \{v, u\}$. W tym momencie będziemy chodzić po grafie zgodnie z algorytmem DFS zaczynając od wierzchołka v z poniższą modyfikacją. Niech a to rozważany wierzchołek.

```
    visited[a] := 1
    f := //krawędź, która łączy a oraz następny wierzchołek do rozstrzygnięcia//
    weight := w[f]
    IF we < weight THEN
        //zignoruj tę krawędź//
    ENDIF
ENDWHILE
IF visited[u] == 1
    RETURN false
ELSE
    RETURN true
ENDIF
```

Algorytm jak łatwo się domyślić zwraca *true* jeżeli e należy do MST i *false* wpp.

4. Dowód poprawności

Dowód. Rozważmy dwa przypadki:

- e nie jest w MST.
  Wtedy leży na jakimś cyklu, na którym ma ma największą wagę. Wtedy nasz algorytm przejdzie po tym cyklu, bo wszystkie krawędzie mają mniejszą wagę od wagi e i dojdziemy do drugiego wierzchołka e i algorytm zwróci *false*.
- e jest w MST.
  To oznacza, że albo e jest mostem, albo dla każdego cyklu, na którym leży, jest krawędź o większej wade. Jeżeli jest mostem to po usunięciu tej krawędzi nie da się dojść do z jego wierzchołka tej krawędzi do drugiego więc algorytm zwróci *true*. Jeżeli zaś zachodzi drugi przypadek to algorytm chcąc przejść dowolnym z tych cyklów do drugiego końca krawędzi e natknie się na krawędź, która ma większą wagę od e i nie będzie mógł dojść, więc algorytm zwróci *true*.

$\square$