Zadanie 3, lista 4

Jakub Marcinkowski

5 maja 2020

1 Algorytm

Nasz algorytm będzie rekurencyjny.
Dla każdego wierzchołka *v* będziemy pamiętać dwie liczby (x_1, x_2). Pierwsza liczba będzie oznaczać wielkość najliczniejszego zbioru niezależnego poddrzewa v, do którego v należy, zaś druga, najliczniejszego zbioru niezależnego v, takiego, że v do niego nie należy.
Jak można zauważyć, dla wierzchołka v będącego liściem, para taka będzie równa $(1, 0)$.
Niech wierzchołek u nie będzie liściem, wówczas niech wierzchołki $u_1, u_2, ..., u_k$ będą jego dziećmi, a pary $(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)$ wyliczonymi dla dzieci wartościami najliczniejszych zbiorów niezależnych ich poddrzew.
wówczas, wartości (x, y) dla poddrzewa wierzchołka u będą równe
$(y_1 + y_2 + ... + y_k, max(x_1, y_1) + max(x_2, y_2) + ... + max(x_k, y_k))$. Jest tak, ponieważ wartość x opisuje liczność zbioru niezależnego, gdy u do niego należy. Wówczas żaden z synów u nie może należeć do tego zbioru.
Wartość y opisuje liczność zbioru niezależnego, gdy u do niego nie należy. Ponieważ synowie u w żaden sposób od siebie nie zależą, wówczas mogą dowolnie do najliczniejszego zbioru niezależnego należeć, lub nie. Dlatego wybieramy opcje maksymalizujące wynik.
Algorytm będzie wyglądał mniej więcej tak:

```
int licznosc_zbiorow[n][2];
bool odwiedzone[n];

void dfs(int v) {
  odwiedzone[v] = true;
  if (sasiedzi[v].size() <= 1){
    licznosc_zbiorow[v][0] = 1;
    licznosc_zbiorow[v][1] = 0;
    return;
  }
  for(int a = 0; a < sasiedzi[v].size(); a++) {
    u = sasiedzi[v][a];
    if (!odwiedzone[u]) {
      dfs(u);
      licznosc_zbiorow[v][0] += licznosc_zbiorow[u][1];
      licznosc_zbiorow[v][1] += max(licznosc_zbiorow[u][0],
                                    licznosc_zbiorow[u][1]);
    }
  }
}

dfs(korzen);
```

Budowanie najliczniejszego zbioru niezależnego możemy zrealizować drugim DFS-em.
Zacznie on od korzenia i będzie działał następująco:

- Jeżeli ojciec wierzchołka v należy do najliczniejszego zbioru niezależnego, v do niego nie należy.
- Jeżeli ojciec wierzchołka v nie należy do najliczniejszego zbioru niezależnego, v do niego należy, jeżeli v należy do najliczniejszego zbioru niezależnego swojego poddrzewa (krócej, jeżeli $x_v > y_v$).

Niech funkcja add(v, A) dodaje wierzchołek v do zbioru wierzchołków A, zaś S będzie najliczniejszym zbiorem niezależnym.

```
void budowanie(int v, bool ojc_nalezy) {
  odwiedzone[v] = true;
  bool v_nalezy = false;
  if (!ojc_nalezy && licznosc_zbiorow[v][0] > licznosc_zbiorow[v][1])
{
    add(v, S);
    v_nalezy = true;
  }
  for(int a = 0; a < sasiedzi[v].size(); a++) {
    u = sasiedzi[v][a];
    if (!odwiedzone[u]) {
      dfs(u, v_nalezy);
    }
  }
}
```

2 Uzasadnienie poprawności

Dowód poprawności można poprowadzić przez indukcję.
Pokażemy, że dla każdego drzewa T o korzeniu w v, obie liczby (x_v, y_v), które zwróci nasz algorytm, są optymalne, to jest, są wielkościami najliczniejszych zbiorów niezależnych, w którym v jest (x_v) i w którym v nie ma(y_v).

2.1 Podstawa indukcji

Dla n = 1 łatwo. Pojedyńczy wierzchołek jest liściem, a najliczniejszy zbiór niezależny zawiera tylko jego. Nasz algorytm zwróci taki wynik.

2.2 Krok indukcyjny

Załóżmy, że dla każdego drzewa mającego mniej niż n wierzchołków nasz algorytm zwróci poprawny wynik.
Rozważmy drzewo T wielkości n o korzeniu w v.
Niech wierzechołki $v_1, v_2, ..., v_k$ to dzieci v.
Wiemy, że wszystkie poddrzewa ukorzenione w $v_1, ..., v_k$ mają mniej niż n wierzchołków, więc dla nich nasz algorytm zwraca poprawne wyniki $(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)$.

Rozważmy dwa przypadki:

1. v należy do najliczniejszego zbioru niezależnego drzewa T.

2. v nie należy do najliczniejszego zbioru niezależnego drzewa T.

Najpierw rozważmy przypadek pierwszy.
W tym przypadku żaden z wierzchołków $v_1, v_2, ..., v_k$ nie należy do najliczniejszego zbioru niezależnego drzewa T. Ponieważ również wartości $y_1, y_2, ..., y_k$ są poprawne, a poddrzewa o korzeniach w wierzchołkach $v_1, v_2, ..., v_k$, są niezależne, wartość x wyliczona dla wierzchołka v również będzie poprawna. Nasz algorytm zwróci więc, w tym przypadku, poprawny zbiór niezależny.
Rozważmy teraz przypadek drugi:
Zauważmy, że jeżeli maksymalny zbiór niezależny T nie zawiera v jest on sumą maksymalnych zbiorów niezależnych poddrzew ukorzenionych w $v_1, v_2, ..., v_k$, dla których nasz algorytm zwraca dobre wyniki. Stąd, dla v też zwróci dobry wynik.

3 Złożoność

Dla obu funkcji nasz algorytm rozpatrzy każdy z wierzchołków raz, więc ma on złożoność liniową $O(n)$.