2A – Poborcy podatkowi

autor zadania: Mateusz Radecki

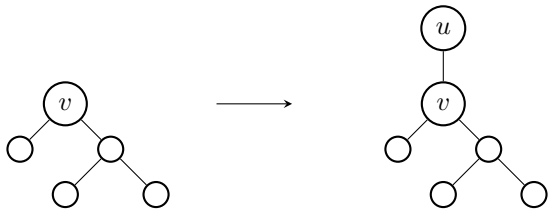
Treść

Dane jest drzewo z wagami na krawędziach. Należy wybrać dowolną liczbę rozłącznych krawędziowo ścieżek. Długość każdej ścieżki musi wynosić dokładnie 4, gdzie przez długość ścieżki rozumiemy liczbę krawędzi w niej. Należy zmaksymalizować sumę wag krawędzi pokrytych przez ścieżki.

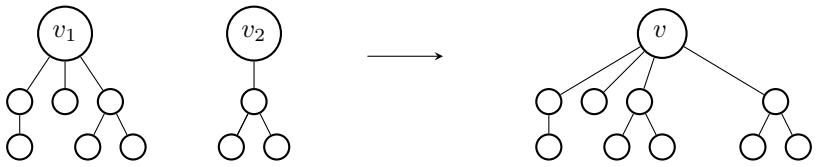
Rozwiązanie

Rozwiążmy zadanie za pomocą programowania dynamicznego. Ukorzeńmy drzewo w dowolnym wierzchołku. Niech $DP[v][i]$ (gdzie $0 \leq i < 4$). Oznacza maksymalną sumę wag krawędzi, jeśli wybieraliśmy ścieżki jedynie w poddrzewie zawieszonym w wierzchołku v, wszystkie wybrane ścieżki mają długość 4, z wyjątkiem jednej z nich, której jeden z końców to v i której długość wynosi i. Jak w standardowym programowaniu dynamicznym na drzewie, musimy umieć zrobić dwie rzeczy.

Dodać korzeniowi drzewa ojca:



Oraz połączyć korzeniami dwa drzewa, gdzie korzeń drugiego ma dokładnie jedno dziecko:



Pierwsze przejście jest prostsze. Znając tablicę $DP[v]$ i znając wagę krawędzi prowadzącej do ojca v (oznaczmy ją w), wystarczy wykonać:

$$DP[u][0] = max(DP[v][0], DP[v][3] + w)$$
$$DP[u][1] = DP[v][0] + w$$
$$DP[u][2] = DP[v][1] + w$$
$$DP[u][3] = DP[v][2] + w$$

Łączenie drzew jest nieco trudniejsze. Zauważmy, że ścieżki długości 1 łączą się ze ścieżkami długości 3, a ścieżki długości 2 łączą się z innymi ścieżkami długości 2. Ścieżki długości 0 (czyli brak ścieżki) nie wpływają na nic. Dla rozważonego prefiksu dzieci v interesuje nas zatem o ile więcej ścieżek długości 1 niż ścieżek długości 3 wybraliśmy, oraz jaka jest parzystość liczby wybranych ścieżek długości 2.

Oznaczmy zatem długość rozważonego prefiksu dzieci przez i, różnicę między liczbą wybranych ścieżek długości 1 i liczbą wybranych ścieżek długości 3 przez j, a parzystość liczby wybranych ścieżek długości 2 przez ℓ. Ten problem możemy również rozwiązać za pomocą programowania dynamicznego. Wprowadźmy tablicę $DP_2[i][j][\ell]$, która odpowiada maksymalnej możliwej sumie wag krawędzi w scenariuszu opisanym wyżej. Łatwo wywnioskować, że zachodzi $DP[v] = \{DP_2[d][0][0], DP_2[d][1][0], DP_2[d][0][1], DP_2[d][-1][0]\}$, gdzie d jest liczbą dzieci v. Z każdego stanu mamy stałą liczbę przejść (dla dziecka v możemy wybrać jedną z czterech możliwych długości ścieżek), stanów mamy jednak kwadratowo wiele, przez co potrzebujemy jakiejś optymalizacji.

Wśród dzieci v istnieje pewne optymalne przyporządkowanie każdemu dziecku długości ścieżki, która ma odchodzić w jego stronę. W tym przyporządkowaniu jest w szczególności podciąg dzieci, które będą miały przyporządkowaną ścieżkę długości 1 lub 3 (a liczności takich dzieci muszą finalnie różnić się co najwyżej o 1). Zauważmy, że gdybyśmy znali taką kolejność dzieci v, w której dzieci, którym należy przyporządkować ścieżki długości 1 oraz 3, występowały na zmianę, to parametr j wcale nie musiałby mieścić się w przedziale $[-n, n]$ – mógłby wtedy mieścić się w przedziale $[-1, 1]$. Niestety, nie znamy takiej kolejności, ale napawa nas to nadzieją – spróbujmy ułożyć dzieci v w losowej kolejności!

Wyobraźmy sobie długi ciąg binarny, w którym zera odpowiadają dzieciom, w które odchodzić będzie ścieżka długości 1, a jedynki odpowiadają dzieciom, w które odchodzić będzie ścieżka długości 3. Wiemy, że nasz ciąg został losowo przemieszany. Zastanówmy się zatem, jaka może być oczekiwana maksymalna różnica między liczbą zer i liczbą jedynek w pewnym prefiksie. Jeśli wszystkie elementy ciągu byłyby niezależnie od siebie losowane ze zbioru $\{0, 1\}$, to interesowałoby nas odchylenie standardowe dość prostej zmiennej, które podpowiadałoby, że szukaną przez nas odpowiedzią jest $\mathcal{O}(\sqrt{n})$. Dodatkowo wiemy, że sumaryczna liczba zer oraz liczba jedynek różnią się co najwyżej o 1, co okazuje się jedynie nam pomagać (jeśli do tej pory spotkaliśmy więcej zer niż jedynek, to następna wartość ma większe szanse być jedynką niż zerem i vice versa).

Warto jednak przekonać się o tym nieco dogłębniej, szczególnie, że musimy ręcznie wybrać jakąś stałą, przez którą ograniczymy możliwe wartości parametru j. Możemy zatem napisać dość prosty program pomocniczy, który (również korzystając z programowania dynamicznego) policzy w złożoności $\mathcal{O}(n \cdot c)$ prawdopodobieństwo, że przy ograniczeniu możliwego przedziału parametru j do $[-c, c]$, nasz algorytm zadziała poprawnie. Przemyślenie szczegółów wspomnianego programu pozostawiamy jako ćwiczenie dla czytelnika. Z napisanego programu łatwo uzyskać informację, że rzeczywiście wartość której szukamy jest rzędu $\mathcal{O}(\sqrt{n})$. Przy jego pomocy możemy również dobrać odpowiednią stałą, zależną od prawdopodobieństwa, którego oczekujemy. Możemy również oczywiście próbować jak najdokładniej dowodzić prawdopodobieństwa sukcesu naszego algorytmu, jednak w konkursach programistycznych, w których nie ma potrzeby dowodzenia poprawności swoich rozwiązań, wspomniany program powinien być w zupełności wystarczający.

Ostateczna złożoność algorytmu wynosi zatem $\mathcal{O}(n \cdot \sqrt{n})$.