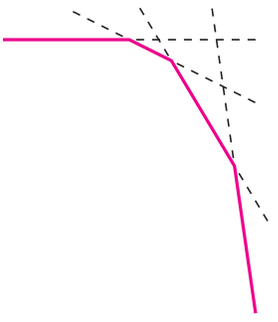
Informatyczny kącik olimpijski (26): Posłańcy

Kolejny kącik poświęcamy zadaniu *Harbingers* (*Posłańcy*) z Olimpiady Informatycznej Środkowej Europy 2009.



Rozważmy pewne państwo z wyróżnioną stolicą, w którym sieć dróg tworzy drzewo (tj. nieskierowany graf spójny bez cykli). W każdym mieście (wierzchołku grafu) i znajduje się posłaniec o parametrach: s_i – czas w minutach potrzebny na przygotowanie do podróży, v_i – liczba minut, w ciągu których przebywa kilometr trasy. Znamy też długości poszczególnych dróg. Aby przesłać wiadomość z pewnego miasta x do stolicy, należy wysłać stamtąd posłańca w jej kierunku (w drzewie jest on wyznaczony jednoznacznie). Na swej trasie, w dowolnym mieście może on albo postanowić kontynuować podróż w kierunku stolicy, albo skończyć podróż i przekazać wiadomość kolejnemu posłańcowi, który będzie musiał przygotować się do podróży i kontynuować dzieło poprzednika z podobnymi opcjami w każdym kolejno napotkanym mieście. Zmian posłańców na drodze do celu może być dowolnie wiele, ale każdy następny potrzebuje czasu na przygotowanie się. Dla każdego miasta chcielibyśmy obliczyć, w ile minut można najszybciej przesłać stamtąd wiadomość do stolicy.

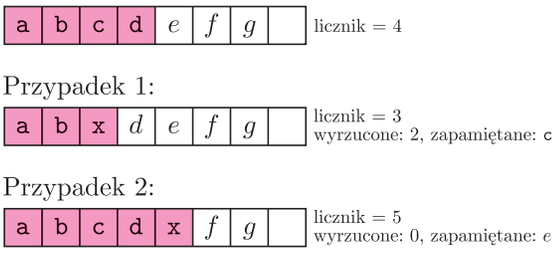
Jako pierwsze nasuwa się rozwiązanie o złożoności $O(n^2)$. Obliczajmy wyniki w_i dla kolejnych wierzchołków, poczynając od tych najbliższych stolicy. Zaczniemy od $w_s = 0$ dla samej stolicy. Dla każdego kolejnego wierzchołka i musimy zdecydować, do którego miasta powinien dojść posłaniec wysłany z i przed pierwszą zmianą. Niech będzie to miasto j, oczywiście na drodze z i do stolicy. Wtedy $w_i = \min_j \{s_i + v_i \cdot odl(i,j) + w_j\}$, przy czym $odl(i,j)$ jest odległością pomiędzy miastami i oraz j. Jeżeli na samym początku obliczymy odległości d_i od stolicy do wszystkich miast, to możemy przekształcić nasz wzór tak: $w_i = \min_j \{s_i + v_i \cdot (d_i - d_j) + w_j\}$; jego bezpośrednie zastosowanie daje rozwiązanie o złożoności $O(n^2)$.

Okazuje się, że rozwiązanie to można usprawnić. Zacznijmy od dalszego przekształcenia naszego wzoru: $w_i = s_i + v_i \cdot d_i + \min_j \{w_j - d_j \cdot v_i\}$. Wśród prostych o równaniach $y = w_j - d_j \cdot x$ (wciąż dla j ze ścieżki łączącej i ze stolicą) będziemy szukać tej, która leży najniżej dla $x = v_i$. Współczynniki kierunkowe tych prostych dla kolejnych j są uporządkowane malejąco, bo odległość od stolicy jest coraz większa. Aby efektywnie znajdować najniższą prostą, pamiętamy dla kolejnych przedziałów, która prosta jest najniżej na danym przedziale (patrz rysunek powyżej).

Będziemy przechodzić nasze drzewo w głąb (ze stolicy), obliczając kolejne w_i i utrzymując taką strukturę prostych dla aktualnej ścieżki ze stolicy do wierzchołka, w którym jesteśmy. Będziemy wykonywać trzy rodzaje operacji:

1. wyszukanie w strukturze prostej o najmniejszej wartości y dla $x = v_i$, aby móc obliczyć w_i;
2. wstawienie nowej prostej i aktualizacja struktury;
3. wyrzucenie takiej prostej i cofnięcie zmian w strukturze (przy cofaniu się w przeszukiwaniu grafu).

Pierwszą operację wykonujemy za pomocą prostego wyszukiwania binarnego po końcach przedziałów zawartych w strukturze. Druga wymaga usunięcia pewnej liczby prostych z prawej strony (potencjalnie żadnej) i umieszczenia na ich miejscu nowej. Nowo wstawiona prosta jest zawsze położona najbardziej na prawo, bo ma najmniejszy współczynnik kierunkowy. Aby móc wykonywać trzecią operację, usuwane w drugiej operacji proste musimy jakoś przechowywać w pamięci. W tym celu do implementacji naszej struktury użyjemy tablicy, którą będziemy wypełniać, począwszy od skrajnie lewych komórek, zaopatrzonej w licznik aktualnie używanych komórek. Przy wstawianiu nowej prostej zapamiętujemy gdzieś na boku, co znajdowało się w komórce tablicy, w której tę prostą umieściliśmy, oraz ile prostych usunęliśmy ze struktury (a więc stałą ilość danych dla każdej operacji). Poniżej zilustrowano przebieg wstawiania prostej x do struktury w dwóch różnych przypadkach. Dane na białym tle nie są w aktualnej strukturze, ale znajdują się w tablicy i mogą kiedyś posłużyć do odtwarzania wcześniejszej zawartości struktury. Widać, że w obu przypadkach można później, przy cofaniu się w przeszukiwaniu grafu, zrekonstruować wyjściową zawartość struktury.



Zaznaczmy, że w drugiej operacji należy binarnie wyszukać, ile prostych z końca trzeba usunąć. Tak więc, operacje pierwszą i drugą wykonujemy w czasie logarytmicznym, a trzecią w czasie stałym. Otrzymujemy algorytm działający w czasie $O(n \log n)$ i pamięci $O(n)$.

Dodajmy na koniec, że w tym zadaniu w połowie testów stopień każdego z wierzchołków był równy co najwyżej 2. Polecamy Czytelnikowi zastanowienie się, na ile upraszcza to omawiany algorytm i czy w tym przypadku istnieje szybsze asymptotycznie rozwiązanie.

Tomasz KULCZYŃSKI