Zadania domowe. Zestaw 4

Maciej Poleski

22 stycznia 2013

1 DFS

W każdym sensownym spacerze z x do x możemy wyróżnić wierzchołek powiedzmy y. Istnieje droga z x do y i z y do x. Jeżeli w grafie G istnieje droga z x do y, to w grafie z odwróconymi krawędziami istnieje droga z y do x. Przeszukajmy więc graf za pomocą DFS-a zaznaczając w pewnej tablicy wierzchołki odwiedzone. Weźmy graf odwrócony i ponownie przeszukajmy go DFS zaznaczając w innej tablicy odwiedzone wierzchołki (za każdym razem zaczynamy w x. Teraz wystarczy sprawdzić czy istnieje y taki że oba DFS-y osiągnęły go.

```
G - graf (n wierzchołków m krawędzi)
H - G z odwróconymi krawędziami
x - wierzchołek
A[n]
B[n]
dfs(a,b,c) - przeszukuje graf a zaczynając w wierzchołku b i
               zapisując odwiedzone wierzchołki w c tak że
               c[d] = true jeżeli d został odwiedzony
dfs(G,x,A);
dfs(H,x,B);
for v in V(G):
   if A[v] and B[v]:
       TAK
NIE (o ile nie stwierdzono wcześniej TAK)
```

Złożoność to dwa DFS-y + jednokrotne przeglądnięcie wierzchołków. Czyli wymagana.

2

3 splay

Każda ścieżka będzie drzewem splay. Operacja splay zachowuje porządek inorder. Porządek inorder takiego drzewa będzie porządkiem wierzchołków na danej ścieżce. Dodatkowo trzymamy uchwyt do każdego wierzchołka. Można to zrealizować w ten sposób że alokujemy tablicę na n wierzchołków. Każdy element (wierzchołek) w tej tablicy będzie węzłem pewnego drzewa splay. Dla danego wierzchołka możemy uzyskać w czasie stałym odpowiedni węzeł pewnego drzewa (po prostu wiemy gdzie w tej tablicy jest ten wierzchołek)

find(x) - wyciągamy x do korzenia drzewa w którym się znajduje (splay(x)). Teraz schodzimy cały czas w lewo i cały czas w prawo aby odnaleźć oba końce ścieżki (porządek inorder)

unlink(x,y) - wyciągamy powiedzmy x do korzenia (splay(x)). Odcinamy poddrzewo w którym jest y (jest następne lub poprzednie w porządku inorder więc wiadomo gdzie go szukać). W efekcie uzyskujemy dwa drzewa splay o właściwym porządku inorder.

link(x,y) - splay(x), splay(y), sprawdzamy czy x wciąż jest w korzeniu (jeżeli nie to x i y należą do tej samej ścieżki), sprawdzamy czy x oraz y mają tylko jedno dziecko (jeżeli nie to nie są końcami ścieżki). Podpinamy powiedzmy y pod wolny link w x. W efekcie uzyskujemy drzewo splay o odpowiednim porządku inorder.

Każda z powyższych operacji działa w zamortyzowanym czasie logarytmicznym (splay oraz spacer w dół drzewa - taki sam koszt jak splay). W efekcie uzyskujemy wymaganą wydajność.

4 Dijkstra

Realizujemy Dijkstre za pomocą kopca (dla złożoności). Na kopcu przechowujemy wierzchołek, informacje o odwiedzonych wierzchołkach 1-10 (w postaci maski) i obecną odległość (która jest priorytetem na kopcu). Podczas relaxowania wierzchołka:

1. Obliczamy (OR-ujemy) nową maskę (jeżeli jesteśmy w wierzchołku 1-10)

2. Próbujemy relaxować sąsiadów (jeżeli maska pozwala na wejście do kolorowego wierzchołka lub wierzchołek nie ma koloru)

Na podstawie informacji zapisanych w masce wiemy czy "możemy" "przejść" daną krawędzią. Na potrzeby kolejki priorytetowej uznajemy że dwa obiekty są takie same jeżeli

1. Mają ten sam wierzchołek

2. Mają taką samą maske

Czyli na kopcu może się pojawić 2^{10} wariantów każdego wierzchołka. Jak by nie patrzeć 2^{10} to stała. Na początku wrzucamy na kopiec wierzchołek s z pustą maską i zerową odległością. Złożoność to złożoność Dijkstry na kopcu z "sporą" stałą (ale stałą).

5

6

7 F-W

Waluty będą wierzchołkami. Krawędź (x,y) będzie oznaczała możliwość wymiany waluty x na y i etykietowana będzie kosztem tej wymiany. Koszt jest to stosunek ilości wydanych pieniędzy do ilości uzyskanych pieniędzy podczas wymiany (ignorując walute jako jednostke!). Aby uzyskać zysk musimy znaleźć cykl transakcji który ma koszt mniejszy niż 1. Jeżeli z x do y mamy optymalny koszt a oraz z y do z koszt b, to optymalny koszt z x do z jest nie większy niż ab. Tak wygląda nierówność trójkąta w tym grafie. Jako przypadek brzegowy ustalamy wstępny koszt wymiany z dowolnej waluty x do x na 1, a pozostałych na nieskończoność. Uruchamiamy algorytm Floyda-Warshalla.

Jeżeli istnieje wierzchołek y taki że odległość y do y jest mniejsza niż 1 oraz istnieje ścieżka z PLN do y oraz z y do PLN to możemy uzyskać dowolnie wielki zysk. W przeciwnym wypadku nie.

Konstrukcja wejściowej macierzy odległości wymaga n^2 operacji (wpisanie jedynek i nieskończoności) i dodatkowo m operacji (przetworzenie każdej możliwej wymiany i ewentualne poprawienie oferty w macierzy). Sam algorytm n^3. W efekcie złożoność zgodnie z wymaganiami. W rozwiązaniu zakładam że możemy w kantorach wymieniać dowolne kwoty (np. niewymierne).

8 off-line + find-union

Robimy to off-line od końca. Zaczynamy z pustą szachownicą i dodajemy po jednym polu (zgodnie z danymi wejściowymi). Pamiętamy stan szachownicy (które pola jeszcze są wygryzione, a które już nie) oraz przechowujemy strukturę find-union.

Dodając nowe pole do szachownicy (cofając operację wygryzania) tworzymy nowy jedno elementowy zbiór odpowiadający temu polu i sprawdzamy czy istnieją niewygryzione pola na szachownicy będące sąsiadami właśnie dodanego pola. Jeżeli tak to łączymy zbiory tych pól z nowo powstałym zbiorem. Dodatkowo przechowujemy informację o ilości zbiorów przed każdym dodaniem pola (tworzenie nowego zwiększa o jeden, każde union zmniejsza o jeden).

Wypisujemy ustalone informacje od końca. Złożoność - n^2 operacji na

strukturze find-union, każda w zamortyzowanym czasie co najwyżej $\log^*(n)$ do tego stały koszt modyfikowania stanu szachownicy i zapisywania informacji o ilości zbiorów. W efekcie $O(n^2 \log^*(n))$