Zadanie 2

Prosty algorytm wykładniczy

Najpierw opiszę prosty algorytm wykładniczy do tego zadania, a potem powiem jak go ulepszyć do PTAS-a. Możemy posortować graf topologicznie (w czasie wielomianowym (a nawet liniowym) od długości wejścia), po czym wykonać programowanie liniowe: dla każdego wierzchołka trzymać D – tablicę $1 + c_{\max}|V|$ liczb, po jednej dla każdego możliwego kosztu ścieżki (każda z krawędzi ma koszt conajwyżej $c_{\max}$, i będzie ich conajwyżej $|V| - 1$ z acykliczności), gdzie i-ta liczba dla wierzchołka v oznacza długość najkrótszej ścieżki z s do v o koszcie i (lub $+\infty$ jeśli takiej ścieżki nie ma).

Oczywiście tablicę tę można wyliczać programowaniem dynamicznym: dla $v = s$ ustawiamy $D_s[0] = 0$, $D_s[i] = +\infty$ dla $i \neq 0$, po czym w kolejności topologicznej ustawiamy $D_v[i] = \min\{D_w[i - c(wv)] + l(wv) : wv \in E, i \geqslant c(wv)\}$ (przy oczywistej konwencji $\min \varnothing = +\infty$).

Kosztem wynikowym będzie $\min\{c : D_t[c] \leqslant L\}$, zaś samą ścieżkę można łatwo odzyskać „cofając" się tym dynamikiem (patrząc, skąd przypisał on aktualną wartość).

0.1 Wielomianowoczasowy schemat aproksymacji

Ustalmy liczbę R oraz $R + 1$ liczb $0 = a_0 < a_1 < \ldots < a_R = (c_{\max} + 42)|V|$ (do dokładnego ustalenia później). Bedę jednak wymagał, żeby $[a_i + l, a_{i+1} + l)$ przecinał conajwyżej 2 przedziały $[a_j, a_{j+1})$ dla każdych i, l, j takich, że napisy mają sens. Warunek ten zajdzie np. wtedy, kiedy przedziały $[a_i, a_{i+1})$ będą miały rosnącą długość.

Teraz zastąpmy tablicę D w algorytmie wykładniczym tablicą D^R, która ma $R+1$ elementów i reprezentuje „zgrubnie" tablicę R (analogicznie jak na ćwiczeniach), gdzie $D^R_v[i] = j$ oznacza, że algorytm twierdzi (być może myląc się – o tym za chwilę), że długość najkrótszej ścieżki z s do v o koszcie z przedziału $[a_i, a_{i+1})$ wynosi j. Jednakże algorytm bardzo boi się uznać ścieżki za tańszą niż jest w rzeczywistości, zatem czasem będzie przeszacowywał jej koszt, tzn. będzie ona uwzględniona nie w $D^R_v[i]$, tylko w jednej z kilku następnych komórek.

Dokładniej: tak jak algorytm wykładniczy odczytywał $D_w[i]$ i na tej postawie rozważał wartość dla $D_v[i + c(wv)]$, tak nasz PTAS, rozważając $D^R_w[i]$ wie jedynie o jakiejś ścieżce o koszcie w przedziale $[a_i, a_{i+1})$, zatem konserwatywnie założy, że ma ona koszt a_{i+1}, zatem powinien ją wpisać w pole $D^R_v[j]$ takie, że $a_j \leqslant a_{i+1} + c(wv) < a_{j+1}$. Jednakże dla absolutnej pewności, że nie niedoszacowuje ścieżki, wpisze ją w $D^R_v[j+1]$.

Po uruchomieniu algorytmu w tablicy D^R_t znajdą się jakieś wartości. Teraz algorytm odpowie, że najtańsza ścieżka o długości conajwyżej L ma koszt a_{k+1}, gdzie k jest minimalne takie, że $D^R_t[k] \leqslant L$.

Oczywiście wartość ta jest niemniejsza niż prawdziwy wynik – algorytm cały czas jedynie przeszacowuje koszty uzyskiwane przez dokładny algorytm wykładniczy. Chcę pokazać, że nie myli się jednak za dużo (dla dobrego doboru ciągu $a_0, \ldots, a_R$). Ustalmy bowiem jakąś ścieżkę z s do t o długości $\leqslant L$ oraz liczbie krawędzi k.

W czasie jej przetwarzania, dla każdej krawędzi na ścieżce, algorytm przeszacował wynik o conajwyżej dwie komórki tablicy D^R – jedną dlatego, że dana na wejściu mogła wynosić np. a_i, a on pesymistycznie przyjął, że a_{i+1} (lecz powoduje to stratę conajwyżej jednej komórki tablicy, na podstawie założenia z początku rozdziału o przecinaniu conajwyżej 2 przedziałów), i następnie celowo pomylił się o jedną komórkę dla pewności.

Na końcu działania, przy podawaniu wyniku, dodatkowo przeszacował wynik o conajwyżej jedną komórkę. Ponieważ $k \leqslant |V| - 1$, widzimy, że przeszacowaliśmy wynik o conajwyżej $2|V| - 1$ komórek. Jeżeli więc tylko $a_{i+2|V|-1}/a_i \leqslant 1 + \varepsilon$ dla każdego i mającego sens, uzyskaliśmy satysfakcjonujący nas schemat aproksymacji.

Teraz pozostaje dobrać liczby a_i. Naturalnym wyborem jest regularny podział na skali logarytmicznej, tj. $a_i = c^i$ dla pewnego c. Wtedy chcemy, żeby $c^{2|V|-1} \leqslant 1 + \varepsilon$, czyli $c \leqslant (1+\varepsilon)^{\frac{1}{2|V|-1}}$. Chcemy $a_R = (c_{\max} + 42)|V|$, a zatem $R = \frac{\log(c_{\max}+42) + \log|V|}{\frac{\log(1+\varepsilon)}{2|V|-1}}$, co jest wielomianowe od rozmiaru problemu.

Algorytm ten zatem będzie działał w czasie $O(ER)$ dla R zadanego powyższym wzorem (zależnym wielomianowo od wejścia).

Oczywiście jeśli odpowiednio dokładnie zaokrąglimy liczby c^i (z zachowaniem dobrego błędu względnego) to powyższy algorytm nadal dobrze aproksymuje.