Algorytmy online: lista 10

Zadanie 1. **(2 pkt.)** Pokaż, że konkurencyjność dowolnego algorytmu deterministycznego Det dla problemu przenoszenia pliku wynosi co najmniej 3. Wskazówka: rozważ graf dwuwierzchołkowy i porównaj Det z trzema algorytmami (dwa statyczne i jeden, który ma plik w innym wierzchołku niż Det).

Zadanie 2. **(2 pkt.)** Rozważ następujący algorytm $\textsc{Cnt}_k$ parametryzowany pewną stałą k. Na początku działania $\textsc{Cnt}_k$ losuje wartość licznika C jednostajnie z przedziału $[1, k]$. W każdym kroku, po obsłudze żądania, $\textsc{Cnt}_k$ zmniejsza C o 1 i jeśli $C = 0$, to plik jest przenoszony do miejsca żądania, zaś C jest zmieniane na k.

Pokaż, że dla dowolnego k konkurencyjność algorytmu $\textsc{Cnt}_k$ wynosi co najwyżej

$$\max \left\{ 2 + \frac{2D}{k}, 1 + \frac{k}{2D} \right\} + O(1/D) \ .$$

gdzie D jest rozmiarem pliku. Możesz wykorzystać funkcję potencjału równą $\Phi = (C + D) \cdot d(v_{\mathrm{CNT}}, v_{\mathrm{OPT}})$. Zakładając, że $D \to \infty$ i nie musisz się przejmować całkowitością k, oblicz jakie jest k optymalizujące powyższe maksimum i ile wynosi wtedy konkurencyjność $\textsc{Cnt}_k$.

Zadanie 3. **(2 pkt.)** Rozważmy następujące uogólnienie algorytmu Flip dla problemu przenoszenia pliku: algorytm $\textsc{Flip}(p)$ przenosi plik do żądania z prawdopodobieństwem $p \in (0, 1]$. (Oryginalny algorytm Flip wybierał $p = 1/(2D)$). Ustalmy graf, który składa się z dwóch wierzchołków połączonych krawędzią. Pokaż, że dla dowolnego p konkurencyjność algorytmu $\textsc{Flip}(p)$ na tym grafie wynosi co najmniej 3.

Zadanie 4. Na odwrocie znajdziesz program liniowy, zapisz go do pliku `lower.lp`. Przejrzyj jego zawartość zwracając uwagę na trzy wyróżnione nierówności: dlaczego są one prawdziwe? Uruchom polecenie `lp_solve < lower.lp`.[1]

- **a)** **(2 pkt.)** Na podstawie wyniku narysuj i opisz, jak wygląda faza dla której zachodzi $\textsc{Alg} + \Delta\Phi = 7 \cdot \textsc{Opt}$, gdzie $\Phi = 2 \cdot D \cdot d(\textsc{Alg}, \textsc{Opt})$. Uwaga: prawdopodobnie żądania nie mogą być skupione w jednym miejscu `req`, ten punkt odpowiada tylko ich „środkowi masy”.
- **b)** **(2 pkt.)** Zauważ, że powyższy wynik nie prowadzi łatwo do prawdziwego dolnego ograniczenia, bo odległość $d(\textsc{Alg}, \textsc{Opt})$ jest inna na początku i na końcu fazy. Dodaj odpowiednią równość, żeby to wymusić, wykonaj ponownie program i rozwiąż ponownie poprzedni podpunkt. Jak z obecnej konstrukcji wynika dolne ograniczenie 7 na konkurencyjność algorytmu `Move-To-Min`?

Marcin Bieńkowski

[1] Program `lp_solve` to darmowy solver LP dostępny w większości popularnych systemów operacyjnych. Jeśli w wyniku otrzymasz inne wartości niż `potential_start = 0` i `potential_end = 4`, to wymuś je odkomentowując odpowiednie wiersze.

```
/*
  Generowanie metryki wymuszającej największy współczynnik dla jednej fazy algorytmu Move-To-Min
  dla problemu File migration i konkretnej funkcji potencjału równej 2 * D * dist(ALG, OPT).

  Mamy dane punkty alg0, alg1, opt0 i opt1 (pozycje ALG i OPT na początku i końcu). Zmienne w programie
  to głównie D_x_y, oznaczające odległość między punktami x i y, pomnożoną przez D. Dodatkowo
  D_req_x oznacza sumę odległości od punktu x do wszystkich żądań z danej fazy.

  UWAGA: Wartość ratio jest GÓRNYM ograniczeniem, tj. jeśli program zwróci ratio = 7, to oznacza,
  że faktyczna konkurencyjność to co najwyżej 7. Ale nie na odwrót: może wygenerować układ punktów,
  który NIE JEST MOŻLIWY DO ZAIMPLEMENTOWANIA jako prawdziwa faza.
*/

max: ratio;

ratio = alg_cost + potential_end - potential_start;

alg_cost = D_req_alg0 + D_alg0_alg1;
opt_cost = 1;

opt_cost >= D_req_opt0;                // Lemat ograniczający koszt OPT zastosowany do opt0
opt_cost >= D_req_opt1;                // Lemat ograniczający koszt OPT zastosowany do opt1
opt_cost >= D_opt0_opt1;

potential_start = 2 D_alg0_opt0;
potential_end   = 2 D_alg1_opt1;

// Odkomentuj, jeśli masz za dużo szczęścia:
// potential_start = 0;
// potential_end = 4;

// Jakie jest znaczenie tych trzech nierówności?
D_req_alg1 <= D_req_alg0;
D_req_alg1 <= D_req_opt0;
D_req_alg1 <= D_req_opt1;

// Nierówności trójkąta.
D_req_alg0  <= D_req_alg1  + D_alg0_alg1;
D_req_alg0  <= D_req_opt0  + D_alg0_opt0;
D_req_alg0  <= D_req_opt1  + D_alg0_opt1;
D_req_alg1  <= D_req_alg0  + D_alg0_alg1;
D_req_alg1  <= D_req_opt0  + D_alg1_opt0;
D_req_alg1  <= D_req_opt1  + D_alg1_opt1;
D_req_opt0  <= D_req_alg0  + D_alg0_opt0;
D_req_opt0  <= D_req_alg1  + D_alg1_opt0;
D_req_opt0  <= D_req_opt1  + D_opt0_opt1;
D_req_opt1  <= D_req_alg0  + D_alg0_opt1;
D_req_opt1  <= D_req_alg1  + D_alg1_opt1;
D_req_opt1  <= D_req_opt0  + D_opt0_opt1;
D_alg0_alg1 <= D_req_alg0  + D_req_alg1;
D_alg0_alg1 <= D_alg0_opt0 + D_alg1_opt0;
D_alg0_alg1 <= D_alg0_opt1 + D_alg1_opt1;
D_alg0_opt0 <= D_req_alg0  + D_req_opt0;
D_alg0_opt0 <= D_alg0_alg1 + D_alg1_opt0;
D_alg0_opt0 <= D_alg0_opt1 + D_opt0_opt1;
D_alg0_opt1 <= D_req_alg0  + D_req_opt1;
D_alg0_opt1 <= D_alg0_alg1 + D_alg1_opt1;
D_alg0_opt1 <= D_alg0_opt0 + D_opt0_opt1;
D_alg1_opt0 <= D_req_alg1  + D_req_opt0;
D_alg1_opt0 <= D_alg0_alg1 + D_alg0_opt0;
D_alg1_opt0 <= D_alg1_opt1 + D_opt0_opt1;
D_alg1_opt1 <= D_req_alg1  + D_req_opt1;
D_alg1_opt1 <= D_alg0_alg1 + D_alg0_opt1;
```

```
D_alg1_opt1 <= D_alg1_opt0 + D_opt0_opt1;
D_opt0_opt1 <= D_req_opt0  + D_req_opt1;
D_opt0_opt1 <= D_alg0_opt0 + D_alg0_opt1;
D_opt0_opt1 <= D_alg1_opt0 + D_alg1_opt1;
```