Algorytmy optymalizacji dyskretnej 2022/23

LABORATORIUM 4

Maksymalny przepływ (*Max Flow*) – algorytmy bazujące na ścieżkach powiększających

Termin realizacji: **ostatnie zajęcia w semestrze**

Warunek zaliczenia listy: *realizacja zadań 1 i 2 (wraz ze sprawozdaniem).*

Zadanie 1. [7 pkt]
Dla k-elementowego ciągu bitowego x definiujemy wagę Hamminga $H(x)$ jako liczbę jedynek w tym ciągu (waga Hamminga H przyjmuje wartości ze zbioru $\{0, 1, \ldots, k\}$). Niech ponadto $Z(x)$ będzie liczbą zer w ciągu x.

Dla danego $k \in \mathbb{N}$ rozważmy k-wymiarową skierowaną hiperkostkę $H_k = (N_k, A_k)$, tj. graf skierowany o $|N_k| = 2^k$ wierzchołkach, których etykietami są różne ciągi binarne długości k. Krawędzie (łuki) w hiperkostce H_k łączą wierzchołki etykietowane ciągami różniącymi się na dokładnie jednej pozycji i prowadzą od wierzchołka z etykietą o mniejszej wadze Hamminga do wierzchołka z etykietą o większej wadze. Łatwo zauważyć, że każdy wierzchołek $i \in N_k$ jest początkiem lub końcem dokładnie k łuków, a k-bitowe ciągi etykietujące wierzchołki mogą być traktowane jako binarne reprezentacje liczb naturalnych od 0 do $2^k - 1$. Stąd formalnie hiperkostkę skierowaną możemy zdefiniować jako graf o wierzchołkach $N_k = \{0, 1, \ldots, 2^k - 1\}$, gdzie etykieta $e(i)$ wierzchołka i jest k-bitową reprezentacją liczby i, a zbiorem krawędzi (łuków) jest $A_k = \{(i, j) \in N_k \times N_k \colon e(j) - e(i) \geqslant 0 \wedge H(e(j) - e(i)) = 1\}$ (w definicji zbioru krawędzi A_k wyrażenie $e(j) - e(i)$ oznacza różnicę wektorów z $\{0, 1\}^k$ reprezentujących etykiety wierzchołków j oraz i). Dla tak zdefiniowanego grafu H_k mamy $|A_k| = k\, 2^{k-1}$.

Pojemność u_{ij} każdej krawędzi $(i, j) \in A_k$ losujemy niezależnie z rozkładem jednostajnym ze zbioru $\{1, \ldots, 2^l\}$, gdzie $l = \max\{H(e(i)), Z(e(i)), H(e(j)), Z(e(j))\}$.

Napisz program, który zaimplementuje algorytm Edmondsa-Karpa (implementacja metody Forda-Fulkersona polegająca na zwiększaniu przepływu po najkrótszych – w sensie liczby krawędzi – ścieżkach powiększających, tj. ścieżkach ze źródła do ujścia w sieci residualnej; patrz np. rozdział 26.2 w [CLRS09]) i dla podanego parametru wejściowego $k \in \{1, \ldots, 16\}$, oznaczającego wymiar hiperkostki, wygeneruje opisany wyżej graf skierowany H_k oraz obliczy maksymalny przepływ między źródłem $s = 0$ a ujściem $t = 2^k - 1$.

Program powinien przyjmować wartość k jako parametr wejściowy `--size k`.

Drugim (opcjonalnym) parametrem wejściowym powinien być parametr `--printFlow`. W przypadku, gdy zostanie on podany na wejściu, oprócz wartości maksymalnego przepływu program powinien także zwracać wyznaczony przepływ po każdym łuku w sieci, tj. $\bar{x} = (x_{ij})_{(i,j) \in A_k}$.

Wynik (wartość maksymalnego przepływu oraz – opcjonalnie – przepływ po każdym z łuków) powinien być wypisywany na standardowe wyjście, a na standardowym wyjściu błędów powinny być wypisywane w kolejności: czas działania całego programu oraz liczba ścieżek powiększających wyliczanych przez program w trakcie działania algorytmu.

Przeprowadź eksperymenty pozwalające oszacować średnią wielkość przepływu, liczbę ścieżek powiększających i czas działania programu dla wszystkich wartości $k \in \{1, \ldots, 16\}$ (dla każdego k wykonaj odpowiednią liczbę niezależnych powtórzeń). Uzyskane wyniki przedstaw przy pomocy wykresów (wartości badanych statystyk w zależności od k).

Uwaga! Warunkiem koniecznym uzyskania maksymalnej liczby punktów za to zadanie jest wykonanie eksperymentów dla wszystkich podanych wartości k (sieć H_{16} składa się z $2^{16} = 65\,536$ wierzchołków i $2^{19} = 524\,288$ łuków).

Zadanie 2. [4 pkt]
Dla danych $k \in \mathbb{N}$ oraz $i \in \mathbb{N}$ rozważmy nieskierowany dwudzielny graf losowy mający dwa rozłączne podzbiory wierzchołków V_1 oraz V_2, każdy o mocy 2^k. Krawędzie grafu generowane są w taki sposób, że każdy wierzchołek z V_1 ma i sąsiadów z V_2 wybranych niezależnie i jednostajnie losowo.

Napisz program, który dla podanych parametrów wejściowych $k \in \{1, \ldots, 16\}$ oraz $i \leqslant k$ wygeneruje opisany powyżej graf i obliczy wielkość maksymalnego skojarzenia (por. zadanie 6 z Listy 2 na ćwiczenia).

Program powinien przyjmować wartości k oraz i jako parametry wejściowe, odpowiednio, `--size k` oraz `--degree i`.

Trzecim (opcjonalnym) parametrem wejściowym powinien być parametr `--printMatching`. W przypadku, gdy zostanie on podany na wejściu, oprócz wielkości maksymalnego skojarzenia program powinien także zwracać wyznaczone skojarzenie (lista krawędzi).

Wynik powinien być wypisywany na standardowe wyjście, a na standardowym wyjściu błędów powinien być wypisany czas działania całego programu.

Przeprowadź eksperymenty pozwalające oszacować średnią wielkość maksymalnego skojarzenia i czas działania programu dla wszystkich wartości $k \in \{3, \ldots, 10\}$ oraz $i \in \{1, \ldots, k\}$ (wykonaj odpowiednią liczbę niezależnych powtórzeń). Uzyskane wyniki przedstaw przy pomocy wykresów (dla każdego k wygeneruj wykres wielkości maksymalnego skojarzenia w zależności od i, a także dla każdego i wygeneruj wykres czasu działania programu w zależności od k).

Zadanie 3. [4 pkt]
Uzupełnij programy z zadań 1 oraz 2 o generowanie pliku z modelem programowania liniowego (w wybranym języku, np. `GNU MathProg`, `JuMP`, ...) dla problemów maksymalnego przepływu oraz maksymalnego skojarzenia i grafów wygenerowanych w programach. Do parametrów wejściowych programów dodaj opcjonalny parametr `--glpk nazwa`, który utworzy plik o podanej nazwie z modelem dla programu `glpk`.

Plik dla programu `glpk` powinien zawierać komentarze pozwalające zrozumieć zapisany w nim model programowania liniowego.

Rozwiąż wygenerowane modele LP za pomocą solvera `glpk`. Sprawdź, czy `glpk` zwraca takie same wartości maksymalnego przepływu oraz wielkości maksymalnego skojarzenia. Który program liczy rozwiązanie szybciej? Dla małych wartości k (np. $k = 2, 3, 4$) sprawdź także, czy przepływy po wszystkich łukach oraz skojarzenia wyznaczone przez `glpk` i programy z zadań 1 oraz 2 są takie same.

Zadanie 4. [5 pkt]
Uzupełnij zadanie 1 o implementację **jednego** z następujących algorytmów opartych o ścieżki powiększające, działającego w czasie $\mathrm{O}(n^2m)$:

- algorytm SHORTEST AUGMENTING PATH omówiony w rozdziałach 7.2 (oznaczenia i pojęcia) oraz 7.4 (algorytm i analiza) w [AMO93],
- algorytm Dynica (*Dinic's Algoithm*, *Dinitz's Algorithm*, patrz np. rodział 8.2 w [Tar83] lub notatki na portalu Ważniak MIMUW),
- zmodyfikowany algorytm SHORTEST AUGMENTING PATH omówiony w rozdziale 7.5 w [AMO93] (algorytm ten sprowadza się w praktyce do algorytmu Dynica).

Zadbaj o to, aby złożoność Twojej implementacji wybranego algorytmu była rzędu $\mathrm{O}(n^2m)$ (w tym celu dobierz odpowiednie struktury danych, zoptymalizuj wykonywane operacje, itp.).

Przeprowadź analogiczne testy swojej implementacji jak w zadaniu 1 oraz porównaj wyniki eksperymentów (czas działania, liczba wyznaczanych ścieżek powiększających) z rezultatami uzyskanymi w zadaniu 1.

Uzyskane wyniki należy przedstawić w sprawozdaniu (plik pdf). Sprawozdanie powinno zawierać

- **zwięzły** opis algorytmów z zadań 1, 2 i 4 (implementacja, złożoność) oraz **zwięzły** opis modeli LP z zadania 3 (zmienne decyzyjne, ograniczenia, funkcja celu),
- wyniki przeprowadzonych testów i eksperymentów,
- interpretację uzyskanych wyników oraz wnioski.

W przypadku przesyłania rozwiązań prowadzącemu (np. mailowo lub na platformę MS Teams), plik pdf ze sprawozdaniem, pliki z kodem źródłowym oraz plik `README` (opisujący dostarczone pliki oraz zawierający dane autora) powinny być spakowane programem `zip`, a archiwum nazwane numerem indeksu studenta. Archiwum nie powinno zawierać żadnych zbędnych plików.

Użyteczne linki

- Ważniak MIMUW – Zaawansowane algorytmy i struktury danych – Maksymalny przepływ I
- Ważniak MIMUW – Zaawansowane algorytmy i struktury danych – Maksymalny przepływ II
- MIT OpenCouseWare – Introduction to Maximum Flows
- MIT OpenCouseWare – Maximum Flows 2

Literatura

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., USA, 1993.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[Tar83] Robert E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, USA, 1983.