GDZIE MOŻNA SPOTKAĆ DRZEWA ROZPINAJĄCE

Błażej Osiński
Uniwersytet Warszawski
blazej.osinski@gmail.com

1. Zagadnienia programowe

Ten scenariusz zajęć jest poświęcony algorytmom grafowym, które służą do znajdowania drzewa rozpinającego w grafie z obciążonymi krawędziami. Te zagadnienia wykraczają poza podstawę programową informatyki, również w zakresie rozszerzonym.

2. Temat zajęć:

Gdzie można spotkać drzewa rozpinające

Zajęcia mają na celu zaprezentowanie uczniom kilku ciekawych problemów algorytmicznych, których wspólnym mianownikiem jest problem znajdowania minimalnego drzewa rozpinającego w grafie.

3. Cele zajęć

W wyniku tych zajęć uczeń powinien umieć:

- zaimplementować algorytmy Kruskala i Prima;
- zastosować standardowe algorytmy wyznaczania drzew rozpinających w sytuacjach nietypowych;
- rozróżnić rozwiązanie aproksymacyjne od heurystycznego.

Nie mniej ważnym celem zajęć jest uświadomienie uczniom, że do rozwiązywania złożonych problemów nie wystarcza znajomość sposobu działania standardowych algorytmów, ale niezbędne jest dogłębne zrozumienie ich istoty.

4. Przygotowanie uczniów

Uczniowie przystępujący do tych zajęć powinni:

- posiadać podstawową wiedzę na temat grafów, znać takie pojęcia, jak: graf nieskierowany, spójność grafów, drzewo, przeszukiwanie grafów, most, cykl, cykl Eulera;
- umieć rozwiązywać podstawowe problemy grafowe, wraz z implementacją w wybranym języku programowania;
- umieć konstruować algorytmy zachłanne, a także uzasadniać ich poprawność;
- znać podstawowe struktury danych, w szczególności kopce i strukturę zbiorów rozłącznych.

5. Metody pracy

W zajęciach są stosowane następujące metody pracy:

- generalnie, rozwiązywanie każdego rozważanego zagadnienia (problemu) składa się z sześciu etapów, które składają się na metodę rozwiązywania problemów z pomocą komputera; te etapy, to:
  - opis, dyskusja i zrozumienie sytuacji problemowej,
  - podanie specyfikacji problemów do rozwiązania,
  - zaprojektowanie rozwiązanie (algorytmu),
  - implementacja (realizacja) rozwiązania w postaci programu komputerowego,
  - testowanie i ewaluacja rozwiązania komputerowego,
  - prezentacja sposobu otrzymania rozwiązania i samego rozwiązania;
- grupowa praca, burza mózgów sterowana podpowiedziami nauczyciela, w celu znalezienia rozwiązania problemów;
- implementacja omawianych rozwiązań w wybranym języku programowania;
- zastosowanie poznanych metod do rzeczywistych danych i porównanie wyników z realną sytuacją.

6. Formy pracy

Założone cele są realizowane za pomocą następujących form pracy:

- podczas burzy mózgów prowadzonej przez całą klasę lub w grupach uczniów – ma to doprowadzić do sformułowania algorytmu dla przedstawionego problemu;
- implementacja w wybranym języku programowania odbywa się samodzielnie, lub w parach (dobranych tak, by uczniowie byli na podobnym poziomie swoich informatycznych umiejętności);
- testowanie programów może odbywać się w grupach uczniów, jednak na końcu programy będą poddane automatycznej ewaluacji.

7. Materiały pomocnicze

Nauczyciel i uczniowie korzystają z tablicy lub z tablicy z kartkami papieru, gdzie zapisują specyfikacje problemów, opisy algorytmów, schematy działania algorytmów, fragmenty programów. Przydatne mogą być także nośniki informacji do przenoszenia danych między komputerami. Do prezentacji wyników pracy (na przykład lekcji 2.) przyda się rzutnik multimedialny..

8. Środki dydaktyczne

Uczniowie wykorzystują w czasie tych zajęć:

- materiały dotyczące algorytmiki, opracowane w projekcie Informatyka +;
- dane, a także narzędzia do wizualizacji dostępne w sieci Internet;
- komputer i jego podstawowe oprogramowanie, w tym kompilatory algorytmicznych języków programowania.

9. Przebieg zajęć (kolejnych lekcji)

Zajęcia, których celem jest omówienie ciekawych problemów związanych z wyszukiwaniem drzew rozpinających, mogą być rozłożone na kilka lekcji, niekoniecznie kolejnych. W niniejszej propozycji przyjęliśmy następujący tryb realizacji omawianego tematu:

- na początku prezentowane, lub przypominane są algorytmy Kruskala i Prima (Lekcja 1);
- następnie uczniowie zastosują poznane algorytmy do rzeczywistych danych – porównają planowaną w Polsce sieć autostrad z minimalnym drzewem rozpinającym (Lekcja 2);
- w dalszej części uczniowie są konfrontowani z problemami o rosnącym stopniu trudności, których rozwiązanie wymaga zmodyfikowania poznanych algorytmów (Lekcja 3);
- na koniec uczniowie zajmują się problemem aproksymacji minimalnego drzewa łączącego podzbiór wierzchołków. Szczególnie ważne jest uświadomienie różnicy między rozwiązaniem aproksymacyjnym a heurystycznym (Lekcja 4).

Lekcja 1. Porównanie algorytmów znajdowania minimalnego drzewa rozpinającego. Czas: 60 min.

Lekcja rozpoczyna się od postawienia przez nauczyciela problemu znalezienia minimalnego drzewa rozpinającego, zilustrowanego na przykładzie budowy najtańszej sieci autostrad (do tego przykładu wracamy na drugiej lekcji). Można też wspomnieć, że pierwsze poprawne rozwiązanie tego problemu zostało opublikowane w 1926 przez Czecha Otakara Borůvka, który starał się znaleźć optymalną sieć elektryczną dla swojego kraju. Po wprowadzaniu bardziej formalnego opisu problemu pracując z grupą należy opisać poprawne rozwiązania: algorytmy Kruskala i Prima. Dokładny ich opis można znaleźć m.in. w książce *Wprowadzanie do algorytmów*.

Z dużym prawdopodobieństwem część uczniów zna rozwiązanie tego problemu i chętnie będą oni o nim rozmawiać. Na początku zajmiemy się tym algorytmem, który pierwszy pojawi się w pracy z uczniami. Należy omówić go dokładnie, w szczególności zwrócić uwagę na kwestię pełnego uzasadnienia poprawności zastosowanej metody zachłannej (nie jest to zadanie trudne, a jednak często uczniowie mają z nim problemy).

Przy omawianiu algorytmu Prima należy porównać go z algorytmem Dijkstry: na przykład niech uczniowie skonstruują graf, dla którego te algorytmy wybierają **różne** krawędzie. Warto też omówić sposoby prostej implementacji kopca w języku C++ z wykorzystaniem biblioteki STL (za pomocą kontenerów *set* lub *priority_queue*).

Przy algorytmie Kruskala należy natomiast poświęcić trochę czasu strukturze zbiorów rozłącznych: uczniowie powinni uzasadnić, że zamortyzowany koszt pojedynczej operacji wynosi $O(\log n)$. Jako ciekawostkę można dodać, że koszt ten to tylko $O(\log^* n)$ i podać przykłady, jak wolno rośnie funkcja log*.

Po omówieniu pierwszego z poprawnych algorytmów uczniowie przystępują do jego implementacji (pojedynczo lub w parach), a wynikowy program poddają automatycznej ocenie na zestawie testów przygotowanym przez nauczyciela. Jeżeli uczestnicy będą się męczyli z tym bardzo długo, należy dokończenie implementacji pozostawić jako pracę domową, by został czas na drugą część zajęć.

Teraz należy zaprezentować uczniom drugi z poprawnych algorytmów, który nie pojawił się w dyskusji z uczniami, a następnie porównać go z poprzednim. Zestawienie można przeprowadzać

w postacie pytań, lub krótkich zadań dla uczestników. Przykładowe problemy, na które uczniowie powinni udzielić odpowiedzi, to:

- Dokładne obliczenie złożoności czasowej w zależności od liczb *n* i *m*, liczby wierzchołków i liczby krawędzi w grafie (oznaczenia *n* i *m* stosujemy również w dalszej części).
- Który algorytm będzie prostszy w implementacji? Tu nie ma jednoznacznej odpowiedzi, ale zastanowienie się nad tym pytaniem przyczyni się do lepszego zrozumienia obu algorytmów.
- Jak zmodyfikować oba algorytmy, by znaleźć minimalny las rozpinający w grafie niespójnym?
- Czy oba algorytmy można zmodyfikować tak, by znalazły maksymalne drzewo rozpinające?

Te pytania nawiązują do jednego z celów lekcji, którym jest przekonanie uczniów, by nie patrzyli na algorytmy jak na „czarne skrzynki", ale raczej jak na metody, które należy rozumieć i dostosowywać w zależności od potrzeb.

Jako zadanie domowe należy zadać implementację drugiego algorytmu znajdowania minimalnego drzewa rozpinającego. Ważne jest, by uczniowie samodzielnie zakodowali oba algorytmy, gdyż będą z nich korzystać podczas kolejnych lekcji. W szczególności będą modyfikować i dostosowywać swoje programy, dlatego bardzo istotne jest, by były to programy przez nich napisane i które bardzo dobrze znają i rozumieją.

Lekcja 2. Sieć autostrad jako minimalne drzewo rozpinające. Czas: 45 min.

Celem tej lekcji jest przekonanie uczniów, że algorytmy są często sposobami rozwiązania rzeczywistych problemów, a nie jedynie teoretyczną rozrywką. Jako przykładem posłużymy się wspomnianym już problemem budowy sieci autostrad. Jest to na tyle ważny i obecny w mediach temat, że powinien zainteresować wielu uczniów.

Państwo planując budowę dróg staje przed pewnym dylematem. Z jednej strony chciałoby, żeby sieć autostrad umożliwiała przedostanie się między dowolnymi dwoma ważnymi punktami (np. dużymi miastami, przejściami granicznymi). Z drugiej zaś strony – budowanie dróg jest drogie, więc nie można sobie pozwolić na marnowanie pieniędzy na zbędne połączenia. Wydaje się, że rozsądnym rozwiązaniem może być takie zaplanowanie autostrad, by stanowiły drzewo rozpinające grafu, którego wierzchołkami będą ważne punkty w państwie.

Celem uczniów jest sprawdzenie tej hipotezy na przykładzie sieci autostrad w Polsce. Pracować będą w parach lub w małych grupach. Kolejne zadania do wykonania przez uczestników to:

1. Odnalezienie odległości (dla uproszczenia w linii prostej) między punktami połączonymi przez aktualny system autostrad (między dużymi miastami i przejściami granicznymi): Gdańsk, Łódź, Warszawa, Kraków, Wrocław, Świecko, Olszyna, Jędrzychowice, Gorzyczki, Korczowa, Kukuryki.
   Potrzebne informacje można odnaleźć na stronie: http://www.odleglosci.pl/odleglosci-kody.php Nauczyciel może też dla uproszczenia przygotować dla uczniów tabelkę z tymi odległościami.
2. Przekształcenie danych do formatu, którego używa program z poprzednich zajęć (lub dostosowanie programu do formatu danych).

3. Obliczenie minimalnego drzewa rozpinającego (nie wystarczy jego wielkość, potrzebne będą też użyte krawędzie).
4. Wizualne przedstawienie otrzymanego rozwiązania. Uczniowie mogą nanieść wierzchołki i krawędzie grafu na mapę użytkownika (moje mapy) na stronie internetowej: http://maps.google.com/.
5. Porównanie zarówno kształtu, jak i wielkości swojego rozwiązania z rzeczywistym systemem autostrad w Polsce. Informacje na temat istniejących i planowanych dróg są dostępne pod adresem: http://pl.wikipedia.org/wiki/Autostrady_w_Polsce.

Należy zwrócić uwagę na to, że sieć planowanych autostrad jest rzeczywiście drzewem, nie ma cykli. Oznacza to, że założenie o braku „nadmiarowych" połączeń nie jest w przypadku Polski bezpodstawne.

Lekcja 3. Występowanie krawędzi w drzewach rozpinających. Czas: 90-120 min.

Na tych zajęciach zajmiemy się kilkoma problemami, których wspólnym mianownikiem jest badanie występowania wybranych krawędzi w drzewie rozpinającym. Rozwiązania dwóch pierwszych z nich są na tyle proste, że uczniowie powinni je wymyślić w czasie wspólnej pracy. Wszystkie rozwiązania kładą nacisk na zrozumienie istoty działania poznanych algorytmów zachłannych, a nie tylko umiejętność ich zakodowania.

Problem na rozgrzewkę

Wybierzmy w grafie jedną ustaloną krawędź. Czy należy ona do jakiegoś minimalnego drzewa rozpinającego?

Rozwiązanie jest bardzo proste – na początku liczymy wielkość minimalnego drzewa rozpinającego, a następnie sprawdzamy, czy jakiekolwiek drzewo rozpinające zawierające ustaloną krawędź ma tę samą wielkość co minimalne (robimy to za pomocą algorytmu Kruskala, w którym jako pierwszą przetwarzamy ustaloną krawędź, a następnie wszystkie pozostałe w kolejności niemalejących wag). Złożoność rozwiązania jest taka, jak dwukrotnego użycia algorytmu Kruskala, czyli $O(m \log m + m \log^* n)$. Po wspólnym wymyśleniu rozwiązania uczestnicy powinni zmodyfikować napisany wcześniej algorytm Kruskala, by rozwiązywał ten problem.

Bajtocja, I OIG

Drugim problemem jest udzielenie odpowiedzi na powyższe pytanie dla **wszystkich** krawędzi w grafie. Zadanie to pojawiło się na I Olimpiadzie Informatycznej Gimnazjalistów. Treść zadania Bajtocja znajduje się pod adresem: http://main.edu.pl/user.phtml?op=showtask&task=baj&con=OIG1. Dostępna jest też tam możliwość automatycznej weryfikacji rozwiązania tego zadania na zestawie testów użytych podczas zawodów.

Oczywistym rozwiązaniem, które będzie pewnie pierwszą propozycją uczniów, jest m-krotne zastosowanie poprzedniego algorytmu. Niestety złożoność takiego rozwiązania to $O(m \log m + m^2 \log^* n)$. Należy zachęcić uczniów do znalezienia szybszego rozwiązania. Jest nim inna modyfikacja algorytmu Kruskala. Zauważmy, że jedyną możliwością, by dostać różne drzewa rozpinające w wyniku działania algorytmu Kruskala, jest istnienie kilku krawędzi o tej samej wadze, z których tylko część zostanie wybrana do drzewa. Jest to jedyna możliwość gdyż, jeżeli pominiemy jakąś krawędź,

a weźmiemy inną o większej wadze, to nie dostaniemy minimalnego drzewa, zgodnie z dowodem poprawności algorytmu Kruskala. Aby zatem stwierdzić, czy dana krawędź **może** pojawić się w jakimś minimalnym drzewie rozpinającym wystarczy sprawdzić, czy byłaby wybrana przez algorytm Kruskala, jeżeli byłaby przetwarzana jako pierwsza spośród krawędzi o tej samej wadze. Zauważmy też, że niezależnie od tego jaki podzbiór krawędzi o tej samej wadze weźmiemy do drzewa rozpinającego, otrzymamy ten sam podział na zbiory rozłączne. Jest tak ponieważ wszystkie krawędzie o wadze nie większej niż ustalona wartość w wyznaczają podział na zbiory rozłączne, taki sam jak minimalny las rozpinający z nich wybrany.

Złożoność tego rozwiązania jest asymptotycznie taka sama jak algorytmu Kruskala. Po wspólnej analizie uczniowie powinni przystąpić do implementacji. Ważne jest, by każdy z programów przeszedł przez zestaw testów z Olimpiady (jeżeli nie uda się tego osiągnąć na zajęciach, to pozostawiamy jako pracę domową).

Wszystkie drzewa rozpinające

Omawiany problem można jeszcze trochę utrudnić: dla każdej krawędzi stwierdź, czy występuje we **wszystkich** minimalnych drzewach rozpinających. Ten problem warto omawiać tylko pod warunkiem, że uczniowie mają większą wiedzę na temat grafów, w szczególności znają pojęcie **mostu** i potrafią zaimplementować algorytm służący do znajdowania mostów w grafie.

Rozwiązanie jest podobne do poprzedniego. Należy rozważać kolejne grupy krawędzi o tych samych wagach. Budujemy z nich graf, którego wierzchołkami są otrzymane do tej pory zbiory rozłączne. Nie trudno zauważyć, że jeżeli dana krawędź ma być obecna w dowolnym minimalnym drzewie rozpinającym, to nie może być zastąpiona (w sensie łączenia wierzchołków) przez inny podzbiór krawędzi o tej samej wadze. Oznacza to, że będzie ona mostem w zbudowanym przed chwilą grafie. Mosty możemy oczywiście wyznaczyć za pomocą standardowego algorytmu.

Na chwilę uwagi zasługuje pytanie o czas działania otrzymanego algorytmu. Na pierwszy rzut oka wydawać by się mogło, że trzeba pesymistycznie m razy wykonywać algorytm wyszukiwania mostów o złożoności $O(n + m)$, czyli sumaryczna złożoność wyniosłaby $O(mn + m^2)$. Zauważmy jednak, że kolejne wyszukiwania mostów odbywają się w grafach indukowanych przez podzbiory krawędzi, które to podzbiory stanowią podział wszystkich krawędzi grafu. Oznacza to, że w rzeczywistości sumaryczny czas wyszukania wszystkich mostów wyniesie tyle, co suma wielkości tych indukowanych grafów, czyli $O(m)$. Ponieważ musimy jeszcze posortować krawędzie otrzymamy całkowitą złożoność $O(m \log m)$.

Przejmowanie linii lotniczych

Kolejny ciekawy problem wymagający modyfikacji algorytmu Kruskala nazywa się *Aviamachinations* i można go znaleźć na stronie: http://acm.sgu.ru/problem.php?contest=0&problem=323 (tam też można automatycznie przetestować rozwiązanie).

Mamy daną sieć złożoną z k połączeń lotniczych podzieloną między m linii lotniczych. Chcemy, by jedna linia lotnicza przejęła od pozostałych połączenia tak, by mogła transportować podróżnych między dowolnymi lotniskami (być może z przesiadkami). Przejęcie każdego połączenia kosztuje pewną kwotę. Którą linię lotniczą należy wybrać, by koszty przejęcia były jak najmniejsze?

Pierwszym rozwiązaniem, które przychodzi do głowy, jest wywołanie algorytmu Kruskala osobno dla każdej linii lotniczej, na początku wrzucając za darmo połączenia obsługiwane przez daną linię. Jest to oczywiście rozwiązanie poprawne, jednak zbyt wolne. Istnieje jednak algorytm szybszy: najpierw znajdujemy minimalne drzewo rozpinające T dla całej sieci, nie biorąc pod uwagę podziału na

linie lotnicze. Następnie dla każdej linii lotniczej najpierw wstawiamy za darmo własne połączenia, a następnie dobieramy pozostałe za pomocą algorytmu Kruskala, ale tylko z puli krawędzi wykorzystanych w *T*. Suma wag tak wybranych krawędzi stanowi najmniejszy sumaryczny koszt przejęcia dla danej linii lotniczej.

Dowód tego faktu nie jest zupełnie prosty, ale opiera się na standardowej metodzie dowodzenia dla algorytmów zachłannych. Dla linii lotniczej *p* weźmy nasze rozwiązanie (zbiór krawędzi) *Z* oraz rozwiązanie optymalne *O*. Załóżmy nie wprost, że wielkość (suma wag krawędzi) *O* jest mniejsza niż *W*. W obu zbiorach znajdują się (a jeżeli nie, to je dodamy bez kosztu) darmowe krawędzie, czyli połączenia należące do linii *p*. Część wspólną zbiorów *Z* i *O* oznaczmy przez *W* (w szczególności w zbiorze *W* są zawarte wszystkie darmowe krawędzie), natomiast $Z - W$ przez Z_0 i $O - W$ przez O_0. Z naszego założenia wynika, że sumaryczna waga krawędzi w O_0 jest mniejsza niż w Z_0. Wówczas jednak waga zbioru $T - Z_0 + O_0$ byłaby mniejsza niż waga minimalnego drzewa rozpinającego. Uzyskana sprzeczność dowodzi poprawności naszego algorytmu.

Powyższy dowód nie jest chyba zbyt skomplikowany, zaciemnia go jedynie trochę duża liczba zbiorów, którymi operujemy.

Lekcja 4. Problemy aproksymacyjne. Czas: 90 min.

W ramach tej lekcji uczniowie prawdopodobnie po raz pierwszy spotkają się z algorytmami aproksymacyjnymi, czyli rozwiązującymi problemy w sposób przybliżony. Choć ten temat nie pojawia się zwykle na zawodach informatycznych, to warto skrótowo przedstawić go uczniom, ze względu na jego ważne miejsce w aktualnym rozwoju algorytmiki.

Pokrycie wierzchołkowe

Jako wprowadzenie do tematu rozważmy znany problem pokrycia wierzchołkowego: mając dany graf, wybrać taki zbiór wierzchołków, by każda krawędź była incydentna z co najmniej jednym wybranym wierzchołkiem. Należy poinformować uczniów, że jest to problem NP-zupełny, czyli nie jest znamy dla niego algorytm rozwiązania, które działałby wydajnie dla dużych danych.

To, co możemy próbować robić, to znaleźć rozwiązanie przybliżone. Odruchem jest często wskazanie jakiegoś algorytmu heurystycznego[1] (w skrócie **heurystyki**). Plusem heurystyki jest to, że często działa dość dobrze, ale nie jesteśmy w stanie stwierdzić, jak dobrze. Aby uświadomić uczniom ten problem rozważmy następującą heurystykę rozwiązującą problem pokrycia wierzchołkowego grafu *G*:

```
na początku pokrycie jest zbiorem pustym;
while G nie jest pusty do
   w:=dowolny wierzchołek G;
   zwiększ pokrycie o w;
   usuń z G wierzchołek w oraz wszystkie krawędzie
     incydentne z nim
```

Weźmy graf *G* będący *n*-wierzchołkową gwiazdką (jeden wierzchołek *w* jest połączony ze wszystkimi pozostałymi wierzchołkami i brak jest krawędzi między pozostałymi wierzchołkami). Gdy w pierw-

[1] Na ogół przyjmuje się, że algorytm aproksymacyjny tym różni się od algorytmu heurystycznego. że w przypadku tego pierwszego można określić, jak bardzo generowane rozwiązania są odległe od optymalnego, zaś w przypadku algorytmu heurystycznego na ogół nie można podać takiego oszacowania generowanych rozwiązań.

szej iteracji pętli zostanie wybrany wierzchołek *w*, to otrzymamy rozwiązanie optymalne. Jeżeli jednak będziemy mieli pecha wybierając wierzchołki leżące na końcach ramion gwiazdy, to możemy skonstruować pokrycie bardzo dalekie od optymalnego, nawet o wielkości $n - 1$.

Algorytmy aproksymacyjne mają natomiast tę zaletę, że jesteśmy w stanie wskazać konkretne ograniczenie, jakie będzie spełniało wygenerowane przez nie rozwiązanie. Zwykle żądamy, by rozwiązanie było co najwyżej *k* razy gorsze od optymalnego, gdzie *k* jest określoną stałą, lub funkcją od wielkości danych wejściowych. Okazuje się, że dla problemu pokrycia wierzchołkowego istnieje bardzo prosty algorytm aproksymacyjny, który jest 2-aproksymacją, czyli zwraca pokrycie co najwyżej dwa razy większe od optymalnego.

Oto skrótowy opis algorytmu:

```
na początku pokrycie jest zbiorem pustym;
while G nie jest pusty do
   (w1, w2):=dowolna krawędź w G;
   zwiększ pokrycie o w1 i w2;
   usuń z G wierzchołki w1 i w2 i wszystkie krawędzie
     incydentne z nimi
```

Z pozoru, ten algorytm jest bardzo podobny do poprzedniej heurystyki. Niemniej teraz jesteśmy już w stanie pokazać, że zwracane pokrycie będzie co najwyżej dwa razy większe od optymalnego. Jest to bardzo proste – dla każdej krawędzi wybieranej przez algorytm co najmniej jeden jej wierzchołek musi należeć do pokrycia (wynika to wprost z definicji pokrycia wierzchołkowego). My natomiast zamiast jednego bierzemy oba, zatem za każdym razem bierzemy co najwyżej dwa razy za dużo.

Aby uczniowie lepiej zrozumieli działanie tego algorytmu, powinni wykonać go na kilku małych grafach, a także ręcznie znaleźć wynik optymalny i porównać go z przybliżonym. W szczególności, warto sprawdzić, jak działa nasz algorytm dla uprzednio rozważanej „gwiazdy".

Koleje, XIV OI

Uprzednie rozważania stanowiły jedynie wprowadzenie do tematyki algorytmów aproksymacyjnych. Teraz zajmiemy się trochę trudniejszym problemem, który pojawił się w finale XIV Olimpiady Informatycznej, jako zadanie próbne. Ponownie, treść zadania oraz możliwość automatycznej oceny rozwiązania są dostępne na stronie: http://main.edu.pl/user.phtml?op=showtask&task=kol&con=OI14

W zadaniu dany jest spójny graf *G* wśród którego wierzchołków wyróżniono zbiór *V'* mocy *p*. Zadanie polega na wybraniu takiego podzbioru krawędzi *G* o minimalnej łącznej sumie wag, aby istniała ścieżka złożona wyłącznie z wybranych krawędzi łącząca dowolne dwa wierzchołki z *V'*.

Po poinformowaniu uczniów, że jest to także problem NP-zupełny, można zachęcić ich do prób wspólnego rozwiązania tego problemu. Na pewno pojawi się wiele różnych propozycji heurystyk, trzeba wtedy stanowczo zażądać dowodu 2-aproksymacji. Po pewnym czasie należy zaprezentować poprawne rozwiązanie, lub umiejętnie doprowadzić do niego podpowiedziami.

Najpierw znajdujemy minimalne odległości między wierzchołkami z *V'* (ze względu na dodatkowe ograniczenia w treści zadania, najszybszym rozwiązaniem jest *p* krotne razy użycie algorytmu Dijkstry). Następnie w otrzymanym grafie pełnym *G'*, w którym krawędź (*a*, *b*) ma wagę równą

najkrótszej ścieżce między *a* i *b* w *G*, znajdujemy minimalne drzewo rozpinające. Aby uzyskać wynik wystarczy z powrotem przekształcić krawędzie tego drzewa rozpinającego w ścieżki złożone z oryginalnych krawędzi grafu *G* (i usunąć duplikaty).

Algorytm jest bardzo prosty, trochę bardziej złożony jest dowód, dlaczego rzeczywiście jest to algorytm 2-aproksymacyjny. Aby to uzasadnić posłużymy się następującą konstrukcją: weźmy wynik optymalny dla tego problemu. Jest to z pewnością drzewo w wyjściowym grafie *G*. Teraz zamieńmy je na cykl poprzez podwojenie każdej krawędzi i przejście standardowym algorytmem znajdującym cykl Eulera. Uzyskany cykl ma oczywiście wielkość dwa razy większą od rozwiązania optymalnego. Pokażemy teraz, że jest on większy od wyniku naszego algorytmu.

Gdy za pomocą krawędzi z *G'* połączymy w listę pierwsze wystąpienia kolejnych wierzchołków z *V'* na cyklu uzyskamy jakieś drzewo rozpinające grafu *G'* (lista jest w szczególności drzewem). Zauważmy, że wielkość tego drzewa jest z pewnością mniejsza niż długość cyklu: po pierwsze nie dodajemy krawędzi od ostatniego wierzchołka, po drugie – krawędzie w *G'* mają długość najkrótszych ścieżek w *G*, więc w szczególności nie dłuższych od ścieżek idących wzdłuż jakiegoś ustalonego drzewa w *G*.

Znaleźliśmy zatem jakieś drzewo rozpinające w grafie *G'*. Wielkość tego drzewa jest nie większa niż wielkość dwukrotnego rozwiązania optymalnego. Tymczasem rozwiązanie zwracane przez nasz algorytm jest najmniejszym drzewem rozpinającym w *G'*, zatem w szczególności mniejszym niż to przed chwilą znalezione. Kończy to dowód faktu, że nasz algorytm jest 2-aproksymacją postawionego problemu.

Bardziej szczegółowy dowód poprawności, lepsze rozwiązania, a także dyskusja nad metodą implementacji rozwiązań tego zadania znajdują się w książce *XIV Olimpiada Informatyczna 2006/2007*, Warszawa 2007. Z pewnością warto się zapoznać z tym materiałem przed przeprowadzeniem zajęć – dla ułatwienia zastosowałem tutaj takie same oznaczania, jak w oryginalnym opracowaniu.