Algorytmy i struktury danych

Minimalne drzewa rozpinające

©Marcin Sydow

Spis zagadnień

- drzewa i lasy rozpinające
- cykle fundamentalne i rozcięcia fundamentalne
- własności cykli i rozcięć
- minimalne drzewa rozpinające
- algorytm Kruskala
- algorytm Prima

Drzewo rozpinające

drzewo rozpinające spójnego, nieskierowanego grafu prostego $G = (V, E)$ to taki podgraf T tego grafu, który jest drzewem i zawiera wszystkie wierzchołki danego grafu.

Graf niespójny nie posiada drzewa rozpinającego. Jeśli graf G jest niespójny, to graf będący sumą drzew rozpinających jego składowych spójnych (po jednym na składową) nazywamy **lasem rozpinającym**.

przykład

Drzewo rozpinające grafu spójnego można otrzymać kolejno usuwając krawędzie grafu tak aby uzyskać drzewo.

Dla danego grafu spójnego może istnieć wiele drzew rozpinających.

Każde drzewo rozpinające danego grafu ma tyle samo krawędzi (i wierzchołków)

przypomnienie: **rozcięcie** to minimalny zbiór rozspajający grafu

Problem: Minimalne drzewo rozpinające (ang. MST)

Wejście: nieskierowany graf G z wagami na krawędziach (liczby wymierne).

Wyjście: drzewo rozpinające o minimalnym łącznym koszcie krawędzi (tzw **minimalne drzewo rozpinające**)

(ang. Minimum Spanning Tree - MST)

Problem ten ma rozliczne zastosowania. Jest on rozwiązywalny w czasie wielomianowym. Algorytmy znajdowania MST oparte są na własnościach cykli i rozcięć (Kruskal) lub na modyfikacji algorytmu BFS i Dijkstry (Prim).

The "Własność rozcięcia"

Definition

Dla spójnego grafu $G = (V, E)$ z wagami na krawędziach i podzbioru $S \subseteq V$ jego wierzchołków, jego *rozcięcie* to podzbiór jego krawędzi $E' \subseteq E$ mających dokładnie jeden koniec w zbiorze S i drugi koniec w zbiorze $V \setminus S$.

Lemma

(własność rozcięcia) Załóżmy, że E' jest rozcięciem i e jest jego krawędzią o minimalnej wadze w E'. Wtedy, istnieje minimalne drzewo rozpinające grafu G zawierające e. Dodatkowo, jeśli T' jest zbiorem krawędzi zawartym w pewnym MST i T' nie zawiera żadnej krawędzi z E', to wtedy $T' \cup \{e\}$ jest również zawarte w pewnym MST (tzn. krawędź e jest "przydatna" w MST grafu G)

Dowód znaleźć można np. w podręczniku K.Melhorna w stosownym rozdziale o MST.

Własność cyklu dla MST

Lemma

Załóżmy, że S jest podzbiorem krawędzi pewnego MST pewnego spójnego grafu G z wagami na krawędziach G oraz $C \subseteq E$ jest pewnym cyklem tego grafu. Jeśli $e = (u, v) \in C$ jest krawędzią tego cyklu o maksymalnym koszcie w C taką, że u jest incydenty (styka się z) S, oraz v nie, to wtedy istnieje drzewo MST T' grafu G zawierające S i nie zawierające krawędzi e (tzn. krawędź ta jest zbędna w MST grafu G).

Dowód znaleźć można np. w podręczniku K.Mehlhorna.

Od własności cyklu i rozcięcia do znajdowania MST

Przedstawione własności cyklu i rozcięcia są przydatne do uzasadnienia, że podane algorytmy Kruskala i Prima znajdowania MST są poprawne.

Ogólny schemat algorytmu znajdującego MST może być przedstawiony następująco:

1. inicjalizuj $T = \emptyset$ (to będzie zbiór krawędzi poszukiwanego MST)
2. dopóki T nie wyznacza MST, dodaj pewną krawędź e o minimalnym koszcie z pewnego rozcięcia E' rozłącznego z T

(własność rozcięcia gwarantuje poprawność powyższego ogólnego podejścia)

Różne wybory odnośnie konkretnego rozcięcia E' rozpatrywanego w każdej iteracji powyższego schematu prowadzą do różnych implementacji algorytmów znajdujących MST.

Omówimy teraz 2 algorytmy będące takimi realizacjami: Prima i Kruskala.

Agorytm Prima

Zaczyna od wierzchołka startowego s i stopniowo powiększa drzewo rozpinające. Niech S oznacza zbiór wierzchołków rosnącego drzewa. Poczatkowo $S = \{s\}$. Zauważmy, że zbiór krawędzi o dokładnie jednym końcu w S stanowi rozcięcie. W każdym kroku dodawany jest wierzchołek będący drugim końcem najlżejszej krawędzi z tego rozcięcia.

Używana jest kolejka priorytetowa, aby efektywnie znaleźć taki wierzchołek (priorytetem jest waga najlżejszej krawędzi łączącej ten wierzchołek ze zbiorem S). Po wybraniu, wszystkie krawędzie wychodzące z nowo-dodanego wierzchołka poddawane są relaksacji.

Algorytm Prima

$w(u, v)$ oznacza wagę krawędzi (u, v), w atrybucie `dist` przechowywana jest najkrótsza aktualnie znana odległość wierzchołka do zbioru S, a pq oznacza kolejkę priorytetową. Używamy list sąsiedztwa. Wynikowe drzewo reprezentowane jest w atrybutach `parent`.

```
MSTPrim(V,w,s){
  PriorityQueue pq
  s.dist = 0
  s.parent = null
  pq.insert(s)
  for each u in V\{s}:
    u.dist = INFINITY

  while(!pq.isEmpty()):
    u = pq.deleteMin()
    u.dist = 0
    for each v in u.adjList:
      if (w(u,v) < v.dist):
        v.dist = w(u,v)
        v.parent = u
        if (pq.contains(v)): pq.decreaseKey(v)
        else pq.insert(v)
}
```

Analiza algorytmu Prima

rozmiar danych: n=|V|, m=|E|

dominująca operacja: przypisanie (w inicjalizacji) i porównanie priorytetów (w tym ukryte w kolejce) i odległości

inicjalizacja: $O(n)$
pętla: (n $\times$ delMin()) + (m $\times$ decreaseKey())

Jeśli kolejka zaimplementowana jako kopiec binarny:
pętla: $O(nlog(n)) + O(mlog(n)) = \mathbf{O((n+m)log(n))}$

Jeśli używamy kopca Fibonacciego (amortyzowany koszt stały operacji decreaseKey()):
$O(nlog(n) + m)$

Algorytm Kruskala

1. początkowo $T = \emptyset$
2. rozpatruj krawędzie w kolejności niemalejących wag i dodawaj do T te, które nie tworzą cyklu z poprzednio dodanymi, pozostałe odrzucaj, do momentu, gdy T nie tworzy drzewa rozpinającego

Główny problem to efektywne sprawdzanie, czy rozpatrywana krawędź nie tworzy cyklu z dotychczasowo dodanymi.

Pomysł polega na używaniu pomocniczej struktury danych typu *union-find*. Ponieważ w każdej iteracji T stanowi las, każda nowa krawędź (u, v), która utworzyłaby cykl ma tę własność, że oba jej końce u i v należą do tego samego drzewa w lesie T.

Algorytm Kruskala


```
kruskalMST(V,E,w){
  T = 0
  UnionFind uf
  foreach edge (u,v) in non-decreasing order of weight:
    if (uf.find(u) != uf.find(v)):
      T = T + (u,v)
      uf.union(uf.find(u),uf.find(v))
  return T
}
```

Istnieje bardzo szybka (drzewowa) implementacja struktury union-find, która zapewnia stały czas operacji `union` i prawie[1] stały amortyzowany czas operacji `find`. Analiza złożoności czasowej tej implementacji nie jest jednak matematycznie łatwa. Przy takiej implementacji złożoność jest $O(mlog(m))$ (i jest zdominowana przez początkowe posortowanie krawędzi po wagach)

[1] jest to pewna funkcja, która bardzo wolno rośnie

Podsumowanie

- drzewo rozpinające
- problem minimalnego drzewa rozpinającego (MST)
- własność rozcięcia i własność cyklu
- algorytm Prima
- algorytmj Kruskala

Przykładowe pytania/zadania

- podaj definicję drzewa i lasu rozpinającego
- podaj definicję rozcięcia i cyklu
- podaj własność rozcięcia i jej interpretację w problemie MST
- podaj własność cyklu i jej interpretację w problemie MST
- wyjaśnij ideę algorytmu Prima
- wyjaśnij ideę algorytmu Kruskala
- wyjaśnij jak własność cyklu i rozcięcia używana jest w w/w algorytmach dla MST
- dokonaj analizy złożoności czasowej algorytmu Prima/Kruskala
- mając dany konkretny graf z wagami zastosuj algorytm Prima/Kruskala i wypisz krawędzie tego grafu w kolejności w jakiej zostały zaakceptowane (w przypadku arbitralności zastosuj kolejność alfabetyczną etykiet)

Dziękuję za uwagę