Algoritmi in podatkovne strukture – 2

Grafi

vpeta drevesa

Uteženi grafi

Graf je definiran kot:

- Imamo množico vozlišč, ki imajo svoje oznake: $V = \{v_1, v_2, \ldots, v_n\}$.
- Imamo množico povezav $E = \{(v_i, v_j; w_{ij}) \mid i, j = 1, 2, \ldots, n\}$, kjer povezava $(v_i, v_j; w_{ij})$ povezuje vozlišči v_i in v_j ter ima utež $w_{ij} \in \mathcal{R}$.
- Potem je utežen graf definiran kot $G = (V, E)$.

Utež predstavlja lahko ≫dolžino≪, ≫težo≪, ≫propustnost≪, ...

Dopolnitev definicije:

- utež $w_{ij} \in \{0, \ldots, M-1\}$.
- utež $w_{ij} \in \mathcal{R}$ in utež $w_{ij} > 0$.
- graf je usmerjen ali ne.

Najcenejša vpeta drevesa

- Predpostavka: $G = (V, E)$ je povezan neusmerjen utežen graf.
- Naloga: poiščimo povezan vpet podgraf T tako, da je vsota

$$w(T) = \sum_{uv \in E(T)} w_{uv}$$

minimalna. Ta graf (drevo) imenujemo *najcenejše vpeto drevo*.

Ideja algoritma

- Če iz vpetega drevesa odstanimo katerkoli povezavo, nam graf razpade na dva dela
- V drevesu mora biti najcenejša povezava grafa $(v_a, v_b; w_{ab})$, ker če bi ne bila, potem:
  - namesto nje povezuje dva dela grafa neka druga, dražja povezava $(v_i, v_k; w_{ik})$, kjer $w_{ik} > w_{ab}$
  - če sedaj povezavo $(v_i, v_k; w_{ik})$ nadomestimo s povezavo $(v_a, v_b; w_{ab})$, smo s tem ohranili drevo (ki je še vedno vpeto) in hkrati ga tudi pocenili
- IDEJA: drevo gradimo tako, da mu dodajamo vedno naslednjo najcenešo povezavo, če ta ohranja lastnost drevesa.
- Takšni metodi pravimo POŽREŠNA METODA, ker vedno obdela najprej nekaj kar je najmanjšega/največjega a hkrati ohranja minimalnost.

Definicije in teorija

- Naj bo na nekem koraku algoritma A podmnožica povezav nekega minimalnega vpetega drevesa.
- Naj bo $(u, v) \in E(G)$ povezava z lastnostjo: $A \cup (u, v)$ je še vedno podmnožica povezav (minimalnega) vpetega drevesa. Potem taki povezavi rečemo *dopustna povezava*.
- skica algoritma:

```
public Tree MVD(Graph G) {
  A= { };
  while !(A celo vpeto drevo) {
    poišči v G povezavo (u, v), ki je dopustna;
    A+= (u, v)
  };
  return A;
}
```

- Kako poiskati dopustno naslednjo povezavo?
- Glede na idejo požrešnosti želimo, da je ta še najcenejša – primer *požrešnega algoritma*.

Definicije in teorija – nadalj.

- *Prerez* $(S, V \setminus S)$ neusmerjenega grafa $G = (V, E)$ je particija množice vozlišč V.
- Pravimo, da povezava $(u, v) \in E$ *veže* prerez $(S, V \setminus S)$, če je eno izmed krajišč v S, drugo pa v $V \setminus S$.
- Pravimo, da prerez A *ohranja* množico $A \subseteq E$, če nobena povezava iz A ne veže prereza.
- Vezni povezavi rečemo *minimalna povezava*, če ima izmed vseh veznih povezav minimalno težo (minimalnih povezav je lahko več).

Definicije in teorija – nadalj.

Izrek 1. *Naj bo $G = (V, E)$ povezan neusmerjen graf. Naj bo A podmnožica E, ki je vsebovana v nekem minimalnem vpetem drevesu, naj bo $(S, V \setminus S)$ nek prerez G, ki ohranja A, in naj bo $(u, v; w_{uv})$ minimalna povezava, ki veže $(S, V \setminus S)$. Potem je $(u, v; w_{uv})$ dopustna povezava za A.*

Posledica 1. *Naj bo $G = (V, E)$ povezan neusmerjen graf. Naj bo A podmnožica E, ki je vsebovana v nekem minimalnem vpetem drevesu in naj bosta $C_i = (V_i, E_i)$ in $C_j = (V_j, E_j)$ povezani komponenti (drevesi) v gozdu $G_A = (V, A)$. Če je $(u, v; w_{uv})$ minimalna povezava, ki veže C_i in C_j, potem je $(u, v; w_{uv})$ dopustna povezava za A.*

Požrešni algoritmi

- spoznali smo *deli in vladaj* algoritme
- algoritem je požrešen, ko v vsakem koraku želi narediti *največji možen korak* k rešitvi – npr., naslednje izbrano povezavo, ki jo obravnavamo, je tista, ki je dopustna in ima *najnižjo ceno*

Kruskalov algoritem

Ideja: Kruskalov (J. Kruskal, 1956) algoritem gradi vpeto drevo kot gozd (množica A):

- Na vsakem koraku vzamemo za *dopustno povezavo* minimalno povezavo izmed vseh, ki povezujejo dve drevesi v gozdu. Da je ta povezava res dopustna, sledi po posledici 1.
- Vsaka komponenta C_i predstavlja množico in ko dve komponenti združimo, naredimo v resnici unijo dveh množic – prim. *Union-Find*.

Kruskalov algoritem

```
pubic tree MVD-Kruskal(Graph G) {
  A= { };
  for (v ∈ G.V()) UF.MakeSet(v);
  for (e= (u, v; w) ∈ G.E(), i++) polje[i]= (w, e);
  Uredi(polje);
  for (i=0; i < polje.length(); i++) {
    e= polje[i];
    if (UF.FindSet(e.u) != UF.FindSet(e.v)) { // dopustna povezava
      A+= e;
      UF.Union(UF.FindSet(e.u), UF.FindSet(e.v));
  }
  return A;
}
```

- Najprej $O(|V|)$ `MakeSet` operacij $O(n)$
- Urejanje $O(m \log m) = O(m \log n)$ $O(m \log n)$
- $O(m)$ `FindSet` in `Union` operacij $O(m \log^* n)$
- SKUPAJ: $O(m \log n)$

Primov algoritem

Ideja: Primov algoritem gradi rešitev kot drevo; t.j. množica A je vedno drevo:

- Drevo začnemo graditi iz poljubnega vozlišča.
- Na vsakem koraku dodamo minimalno povezavo, ki veže A z vozliščem, ki ga A ne pokriva. Da je ta povezava res dopustna, zopet sledi po posledici 1.
- Vozlišča ki niso pokrita z A (niso v trenutnem drevesu) hranimo v vrsti s prednostjo (kopici) Q.
- $v.key$ naj bo minimalna teža povezave, ki povezuje v z drevesom A (če take povezave ni, je vrednost $v.key = \infty$).
- $v.p$ označuje očeta vozlišča v v grajenem vpetem drevesu.

Primov algoritem

```
pubic tree MVD-Prim(Graph G) {
  for (v ∈ G.V()) { v.key= ∞; v.p= null }
  r.key= 0;              // r je poljubno vozlišče
  for (v ∈ G.V()) PQ.Insert( (v.key, v) );
  while (! PQ.Empty() ) {
    u= PQ.ExtractMin();
    for (v ∈ u.Adj())
      if (PQ.Find(v) && w_uv < v.key()) {
        v.p= u;  v.key= w_uv
      }
  }
```

- gradnja kopice $O(n \log n)$
- zanka `while` n-krat
  - vsak ukaz `ExtractMin` $O(\log n)$
  - vse zanke `for` skupaj $O(m)$-krat in vsakem ponavljanju implicitni `DecreaseKey`, $O(m \log n)$.

  $O(n \log n + m \log n)$
- SKUPAJ: $O(m \log n)$

Zahtevnost

problem	zahtevnost
najcenejše vpeto drevo – Kruskal	$O(E \log E)$
najcenejše vpeto drevo – Prim	$O(E \log V)$