Algoritmer og Datastrukturer (NDAA04010U)
Ugeopgave 5 med svar og forklaringer

Københavns Universitet

2024

1 Disjunkte mængder

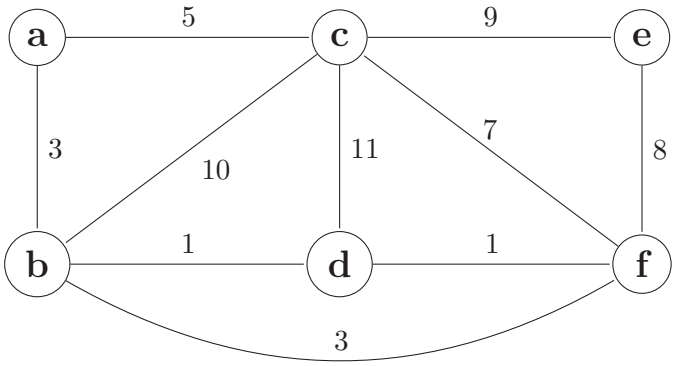
Vi betragter trærepræsentationer af disjunkte mængder (eng. *disjoint sets*), som beskrevet i CLRS sektion 19.3, ved brug af heuristikkerne *union by rank* og *path compression*. Antag at MAKE-SET er kørt n gange og lad k være en variabel. Hvilke af følgende udsagn er altid sande? Vælg ét eller flere korrekte svar og beskriv hvordan du kom frem til dem (både positive og negative svar).

1. Tiden for et kald til UNION(x, y) er $O(\lg n)$. ✓
2. k kald til FIND-SET(42) lige efter hinanden tager tid $O(k + \lg n)$. ✓
3. Kaldene MAKE-SET(x_1),...,MAKE-SET(x_k) tager total tid $O(k)$. ✓
4. En LINK(x, y) operation øger den maksimale værdi af *rank* i træerne for x og y med 1.
5. I værste fald tager en FIND-SET operation $\Omega(n)$ tid.
6. UNION(x, x) tager konstant tid.

Svar: Dybden af trærepræsentationen er $O(\lg n)$, så det gælder også tiden for FIND-SET og dermed UNION. Kalder vi FIND-SET(42) igen efter at den lige er blevet kaldt vil operationen tage konstant tid, dvs. tiden for k kald bliver $O(k + \lg n)$. En LINK operation giver kun en større *rank* hvis røddernes rank er identisk. Med den givne implementation af UNION(x, x) laves der to kald til FIND-SET, der ikke nødvendigvis tager konstant tid. △

2 Mindste udspændende træ

Betragt mindste udspændende træ (eng. *minimum spanning tree*) problemet på denne graf:



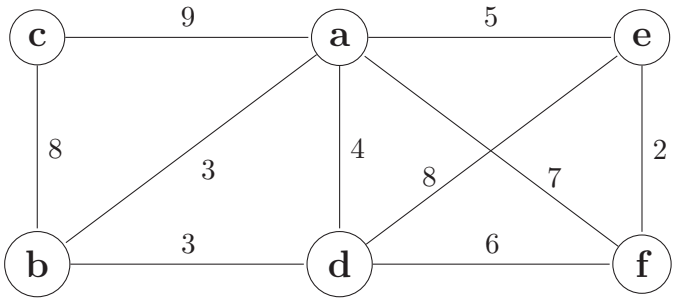
Hvilke kanter er med i mindste udspændende træ? Angiv for hver kant i træet en snitmængde S hvor kanten er let (eng. *light edge*) i snittet $(S, V - S)$.

1. {a,b} ✓
2. {a,c} ✓
3. {b,c}
4. {b,d} ✓
5. {b,f}
6. {c,d}
7. {c,e}
8. {c,f}
9. {d,f} ✓
10. {e,f} ✓

Svar: Mindste udspændende træ kan findes ved at køre fx Kruskál's eller Prim's algoritme. Prim's algoritme med startknude **a** giver disse snitmængder: {a,b} er en let kant for snitmængden $\{a\}$, {b,d} er en let kant for snitmængden $\{a, b\}$, {d,f} er en let kant for snitmængden $\{a, b, d\}$, {a,c} er en let kant for snitmængden $\{a, b, d, f\}$, {e,f} er en let kant for snitmængden $\{a, b, c, d, f\}$. $\triangle$

3 Prim's algoritme

Betragt følgende vægtede, uorienterede graf:



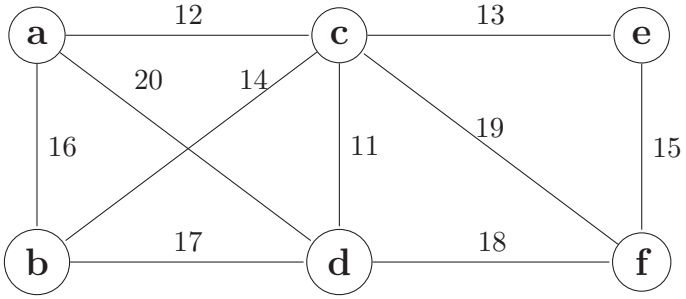
Hvis Prim's algoritme køres på denne graf med start i knude **a**, i hvilken rækkefølge bliver knuderne inkluderet i det mindste udspændende træ? (Med andre ord, i hvilken rækkefølge bliver de taget ud af prioritetskøen?)

1. a, b, d, e, f, c ✓
2. a, b, e, d, f, c
3. a, b, d, c, f, e
4. a, b, e, f, d, c

Svar: Håndkørsel af Prim's algoritme og giver den angivne rækkefølge. Bemærk at rækkefølgen *ikke* er den samme som for Dijkstra's algoritme til at finde korteste veje. △

4 Snit og sikre kanter

Betragt følgende uorienterede, vægtede graf med knudemængde $V = \{a, b, c, d, e, f\}$:



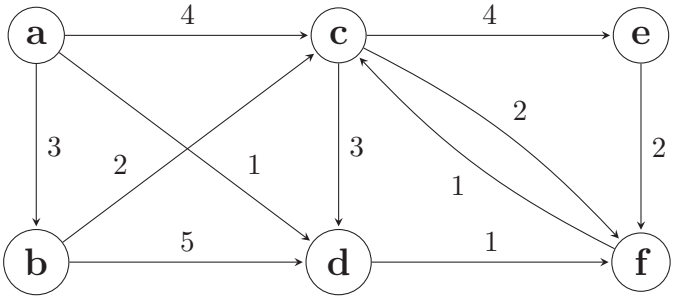
Betragt kantmængden $A = \{\{a, c\}, \{c, d\}\}$, som er en delmængde af et minimalt udspændende træ (eng. *minimum spanning tree*). Vi er interesserede i snit (eng. *cuts*), sikre kanter (eng. *safe edges*) for A, og snit der respekterer (eng. *respects*) en kantmængde A. Hvilke af følgende udsagn er sande? Vælg ét eller flere korrekte svar og beskriv hvordan du kom frem til dem (både positive og negative svar).

1. Lad $S_1 = \{a, b, c, d\}$. Snittet $(S_1, V - S_1)$ respekterer kantmængden A. ✓
2. Lad $S_2 = \{a, c, e\}$. Snittet $(S_2, V - S_2)$ respekterer kantmængden A.
3. Lad $S_3 = \{b, e, f\}$. Snittet $(S_3, V - S_3)$ respekterer kantmængden A. ✓
4. $\{d, f\}$ er en sikker kant for A.
5. $\{c, e\}$ er en sikker kant for A. ✓
6. $\{a, d\}$ er en sikker kant for A.

Svar: Kanten $\{c, d\}$ krydser snittet $(S_2, V - S_2)$, så det respekterer ikke A. Snittene med S_1 og S_3 har ikke nogen kant i A, der krydser, så de respekterer A. Kanten $\{d, f\}$ har ikke mindst vægt i snittet $(\{a, c, d\}, \{b, e, f\})$, så den er ikke sikker. Kanten $\{d, f\}$ har mindst vægt i snittet $(\{a, c, d\}, \{b, e, f\})$, så den er sikker. Kanten $\{a, d\}$ krydser ikke snittet $(\{a, c, d\}, \{b, e, f\})$, så den er ikke sikker. △

5 Korteste veje

Betragt korteste veje problemet (eng. *single-source shortest-paths problem*) med startknude **a** på følgende graf:



Hvilke kanter er med i korteste-vej træet? Hvilken algoritme er velegnet til at beregne korteste-vej træet?

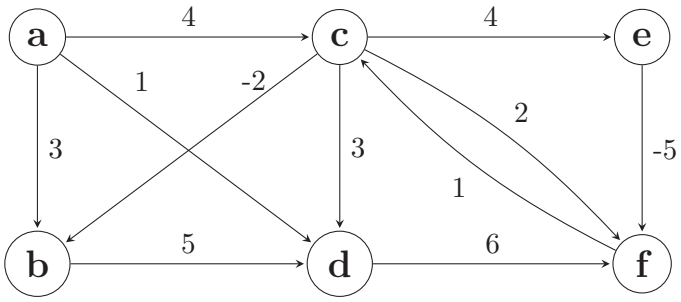
1. (a,b) ✓
2. (a,c)
3. (a,d) ✓
4. (b,c)
5. (b,d)
6. (c,d)

7. (c,e) ✓
8. (c,f)
9. (d,f) ✓
10. (e,f)
11. (f,c) ✓

Svar: Da der kun er positive vægte kan kanterne i træet findes ved at køre Dijkstra's algoritme, der giver det angivne output (som i dette tilfælde er unikt). △

6 Korteste veje

Betragt korteste veje problemet (eng. *single-source shortest-paths problem*) med startknude **a** på følgende graf:



Hvilke kanter er med i korteste-vej træet? Hvilken algoritme er velegnet til at beregne korteste-vej træet?

1. (a,b)
2. (a,c) ✓
3. (a,d) ✓
4. (b,d)
5. (c,b) ✓
6. (c,d)
7. (c,e) ✓
8. (c,f)

9. (d,f)
10. (e,f) ✓
11. (f,c)

Svar: Da der er negative vægte kan kanterne i træet findes ved at køre Bellman-Ford algoritmen, der giver det angivne output (som i dette tilfælde er unikt). △

7 Korteste-veje træ

Betragt et korteste veje problem på en orienteret graf $G = (V, E)$ (eng. *single-source shortest-paths problem on a directed graph*) med vægtfunktion $w : E \to \mathbb{R}$ og startknude s. Hvilke af følgende egenskaber har et korteste-veje træ (eng. *shortest-paths tree*) $G' = (V', E')$ altid? Vælg ét eller flere korrekte svar og beskriv hvordan du kom frem til dem (både positive og negative svar).

1. I G' kan alle knuder i V nås fra s.
2. Knuden s har kun udgående kanter i G'. ✓
3. G' har $|V'| - 1$ kanter. ✓
4. Hvis alle vægte er unikke indeholder G' af de $|E'|$ kanter i G, der har lavest vægt.
5. G' indeholder den kant i G, der har lavest vægt.
6. Enhver sti i G' er en korteste vej i G. ✓
7. Alle korteste veje i G svarer til en sti i G'.
8. G' er et binært træ med s som rod.
9. G' er et mindste udspændende træ (eng. *minimum spanning tree*).
10. Kan beregnes med Dijkstra's algoritme.

Svar: G' forbinder kun s med de knuder knuder i V hvortil der findes en vej i G. Der er kun udgående kanter fra s fordi træet ikke har nogen cykler. G' er et træ, og derfor er antal kanter lig med antal knuder minus 1. En lav vægt er ikke en garanti for at være med i en korteste vej fra s — for eksempel kan visse kanter ikke nås fra s. Der kan være flere korteste veje mellem s og en knude v (med samme vægt), og korteste-vej træet indeholder netop én af disse. Korteste-vej træet er ikke nødvendigvis binært, fx kan vi have en graf hvor alle korteste veje fra s består af én kant, dvs. at s har kanter til $|V'| - 1$ knuder. Et mindste udspændende træ (MST) minimerer summen af kantvægte og indeholder ikke nødvendigvis de korteste veje fra en bestemt knude s; betragt fx en uorienteret graf med tre

knuder og kanter af vægt 2, 2 og 3 hvor et MST vil indeholde de to kanter af vægt 2, men hvor kanten af længde 3 udgør den korteste vej mellem to af knuderne. Dijkstra's algoritme er kun garanteret korrekt når der ikke er negative vægte, så hvis dette ikke er tilfældet kan output være forkert. (Man kan i stedet bruge en anden algoritme, fx Bellman-Ford.) △

8 Valg af datastruktur

Antag at vi gerne vil skabe en datastruktur VENTETID, der indeholder en mængde af n afgangstidspunkter (ugedag og klokkeslet, fx for en bus). For enkelheds skyld skal der blot holdes styr på én rute og ét stoppested, så den eneste information er afgangstidspunkter. Datastrukturen skal understøtte to operationer: INSERT(x), der indsætter et nyt afgangstidspunkt, og HVORLÆNGE(x) der returnerer antal minutter fra tidspunkt x til den næste afgang.

Hvad er det mest passende valg af datastruktur til løsning af dette problem blandt nedenstående, og hvilken worst-case (amortiseret) køretid for HVORLÆNGE(x) bliver resultatet af dette valg? Udtryk svaret som funktion af n. Vælg præcis ét svar og beskriv hvordan du kom frem til det.

1. Disjunkte mængder (eng. *disjoint sets*), tid $O(1)$.
2. Disjunkte mængder (eng. *disjoint sets*), tid $O(\lg n)$.
3. Fibonacci hob (eng. *Fibonacci heap*), tid $O(\lg n)$.
4. Fibonacci hob (eng. *Fibonacci heap*), tid $O(1)$.
5. Rød-sort binært søgetræ (eng. *red-black binary search tree*), tid $O(\lg n)$. ✓
6. Rød-sort binært søgetræ (eng. *red-black binary search tree*), tid $O(1)$.

Svar: Hvis vi indsætter afgangstiderne i et rød-sort søgetræ så kan HVORLÆNGE(x) implementeres som et kald til SUCCESSOR(x). Hvis der ikke findes nogen efterfølger til x (fordi der ikke er nogen senere afgang i samme uge), returneres minimum, dvs. den første afgang i næste uge. I begge tilfælde tager det tid $O(\log n)$. De andre datastrukturer kan ikke umiddelbart bruges til at løse VENTETID hurtigere. △