3 מאי, 1999

הרצאה 7

עצים דינמיים

מוטיבציה

הבעיה: בעיית הזרימה בגרפים.

נתונים: גרף מכוון בעל קיבולים לקשתות: $cap : E \to R^+$, מקור-s ובור-t.

<u>הגדרה</u>: זרימה היא פונקציה $f : E \to R$ שמקיימת:

1. חוק הקיבול - לכל $(u,v) \in E$ מתקיים: הזרימה על הקשת אינה שלילית והיא קטנה מקיבול הקשת.
$$0 \le f(u,v) \le cap(u,v)$$
2. חוק הצומת – לכל צומת v (חוץ מ-s ו-t) מתקיים: סכום הזרימה לתוך הצומת שווה לסכומי הזרימות מתוך הצומת
$$\forall v \in V - \{s,t\} \qquad \sum_u f(u,v) = \sum_w f(v,w)$$

נניח ש- s מקור (אף קשת אינה נכנסת אליו)
t בור (אף קשת אינה יוצאת ממנו).

<u>ערך הזרימה</u> מוגדר כסכום הזרימה היוצאת מ-s

$$|f| = \sum_{w:(s,w)\in E} f(s,w)$$

(ערך הזרימה שווה לסכום הזרימה הנכנסת ל-t
$|f| = \sum_{u:(u,t)\in E} f(u,t)$.)

<u>מטרה</u>: מצא זרימה f עבורה $|f|$ הוא מכסימום.

שיטת דיניץ

<u>זרימה חוסמת</u> היא זרימה f שבכל מסלול מ-s ל-t קיימת קשת רוויה e. כלומר, $f(e) = cap(e)$.

נניח שאם קשת $(x,y) \in E$ אזי גם הקשת ההפוכה $(y,x) \in E$ (אם זו קשת חדשה $cap(y,x) = 0$.)

לכל קשת נגדיר קיבול שיורי:
$$res(x,y) = cap(x,y) - f(x,y) + f(y,x)$$
בעזרת קשתות אלו, מוגדר גרף שיורי:
$$G^{res} = (V, \{(x,y) | res(x,y) > 0\})$$

<u>הגדרה</u>: *גרף שכבות* L
גרף BFS מ-s על הגרף השיורי $(L \subseteq G^{res})$.

אלג' דיניץ עובד בפאזות. בכל פאזה בונים גרף שכבות L ומוצאים בו זרימה חוסמת f^{block} . בסוף הפאזה מעדכנים את הזרימה f בעזרת הזרימה החוסמת. כלומר, לכל קשת e

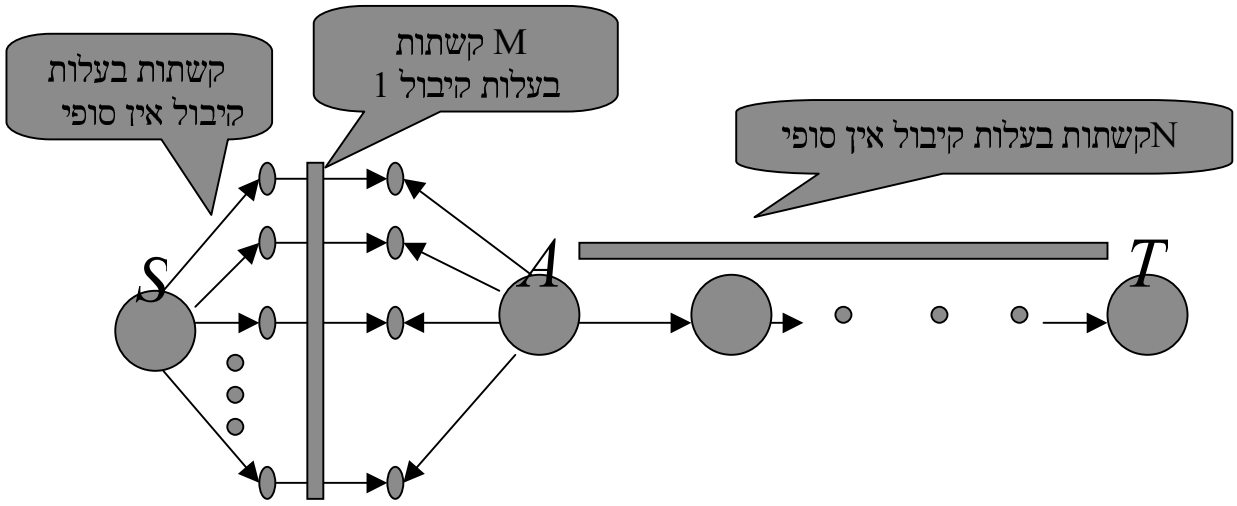
$$f(e) = f(e) + f^{block}(e)$$

כדי למצוא זרימה חוסמת, נחפש מסלולים מ-s ל-t ע"י ביצוע DFS תוך כדי סילוק קשתות שאינן מובילות ל- t.
כשנמצא מסלול P נחפש את צוואר הבקבוק שלה $\Delta = \min_{e \in P}\{cap(e) - f^{block}(e)\}$, ונגדיל את הערך של f^{block} לאורך המסלול. כשנתקל בקשת עם קיבול שיורי 0, נסלק אותה מהגרף.

מציאת זרימה חוסמת בשיטה זו, דורשת זמן $O(nm)$ כאשר $n = |V|$ ו- $m = |E|$. כי לא נלך קדימה יותר מ- n צעדים מבלי לסלק קשת או להגיע ל- t. ובכל מסלול שיימצא לפחות קשת אחת תסולק.
הערה: במקום n אפשר לכתוב את אורך המסלול מ-s ל-t בגרף L.

אחרי העדכון, נחזור על התהליך. ניתן להוכיח שבפאזה הבאה מספר השכבות יגדל ממש. על-כן, מספר הפאזות הוא $< n$. כלומר, האלגוריתם כולו דורש זמן $O(n^2 m)$.

החסם לעיל הדוק. נתבונן במקרה:



המקרה הזה חוסם מלרע את הסיבוכיות, מפני שניאלץ תמיד לצעוד על n הקשתות מ-A ל-t בכל הרוויה של מסלול. כלומר, אם בגרף יש $2N + M$ קשתות, האלגוריתם ידרוש M חיפושי DFS, שכל אחד מהם דורש זמן $\theta(N)$, סה"כ זמן $\theta(NM)$.

ננסה להתגבר על מעברים מיותרים אלו ע"י הגדרת מבנה נתונים.

עצים דינמיים

יער של עצים מכוונים זרים בצמתים. העץ מכוון לכיוון השורש. ועל-כן מכל צומת יוצאת לכל היותר קשת אחת. באלגוריתם העץ יהיה חלקי לגרף השכבות, והשורש יהיה בכיוון t.

פעולות מוגדרות

1. make_tree(v) – צור עץ חדש בעל צומת אחד, v.
2. find_root(v) – מצא את שורש העץ שמכיל את v.
3. find_cost(v) = (u,c) – c הוא ערך צוואר הבקבוק של המסלול מ-v ל-find_root(v) והקשת בה מתקבל המינימום היא $(u,w) \in T$.
4. Add_cost(v,Δ) – הוסף את הערך Δ לכל הקשתות במסלול מ-v ל-find_root(v)
5. Link(v,w,Δ) – v שורש של עץ דינמי, w צומת שאינו שייך לעץ של v. הוסף את הקשת (v,w) ליער עם ערך Δ.
6. Cut(v) – v אינו שורש של עץ דינמי. סלק את הקשת שיוצאת מ-v לאביו. (את הקשת נסלק גם מהעץ וגם מגרף השכבות).

אלגוריתם למציאת זרימה חוסמת בעזרת עצים דינמיים

שמורה: v מצביע בכל שלב של האלגוריתם על שורש של עץ דינמי.

1. אתחול

2. חוג{
   2.1. אם יש קשת שיוצאת מ-v
   קדימה
   2.2. אחרת{
   אם $v = s$ גמרנו
   אם $v = t$ **עדכן**
   אחרת **אחורה**
   }
}

פרוט הפעולות:

אתחול

לכל צומת $u \in V$ בצע make_tree(u).
$v = s$

עדכון

מצא קשת רוויה (x,Δ) = find_cost(s); //
הפחת זרימה בצמתים במסלול add_cost(s, -Δ);//
נתק את קטע המסלול המיותר cut(x);//
שמור על השמורה v = x;//

אחורה

תהי (u,v) קשת שנכנסת ל-v (ברור לנו שאנו לא רוצים להגיע שוב ל-v מפני שאין ממנה יציאה)
אפשרות I

cut(v)
v=find_root(s)

אפשרות II

לכל קשת (u,v) שנכנסת ל-v בעץ בצע cut(v)
v = find_root(s)

אנו נשתמש באפשרות השניה, מפני שמבנה הנתונים אינו תומך במציאת הקשת e שנכנסת ל-v במסלול מ-s.

טענה

האלגוריתם למציאת זרימה חוסמת בעזרת עצים דינמיים, משתמש ב-$O(m)$ פעולות על עצים דינמיים.

הסבר: כל קשת המוצאת מעץ דינמי מוצאת גם מגרף השכבות. על-כן, כל קשת מוכנסת לכל היותר פעם אחת ליער של עצים דינמיים, ומוצאת פעם אחת. כל אחת מהפעולות (קדימה, אחורה עדכן) מכניסה או מוציאה לפחות קשת אחת.

משפט

כל סידרה של m פעולות על עצים דינמיים על יער בן n צמתים דורשת זמן $O(m \log n)$.

מסקנה

האלג' מוצא זרימה חוסמת בגרף השכבות בזמן $O(m \log n)$.

כיון שכדי למצוא זרימה יש לחשב לכל היותר n גרפי שכבות $\Leftarrow$ ניתן לחשב זרימה מכסימום בזמן $O(nm \log n)$.

מימוש עצים דינמיים

נחלק את הקשתות של עץ דינמי לשניים: קשתות שלמות וקשתות שבורות, כך שלכל צומת נכנסת לכל היותר קשת שלמה אחת. הקשתות השלמות, אם כן, מהוות אוסף של מסלולים זרים בצמתים שנקראים מסלולים שלמים.

פעולות על מסלולים

1. make_path(v) = אתחול מסלול עם צומת בודד.
2. find_path(v) - מצא את המסלול ש-v משתתף בו.
3. find_tail(p) - מצא את הצומת הקרוב ביותר לשורש במסלול.
4. find cost (p) = (v,Δ) – מצא את צוואר הבקבוק במסלול מ- v (find_tail(find_path(v)).
5. add_path_cost (p, Δ) - הוסף את המחיר Δ לכל הצמתים במסלול.
6. join (p,v,q) - נתון המצב הבא: .-.-.-.-.-.-0-.-.-.-.-.-.-. , ז"א, נתונים שני מסלולים שהצומת v מפריד ביניהם, נאחד את שני המסלולים לאחד.
7. split(v) = (p,q) – ההפך מ-join

בהגדרות לעיל p,q מסלולים. v,w מציינים צמתים.

- פעולה מורכבת: p = expose(v)

פעולה זו מוגדרת בעצים דינמיים – יצירת מסלול שלם מ- v לשורש תוך שמירה על האינווריאנט שלכל צומת נכנסת קשת ששלמה אחת.

אחרי ביצוע פעולת expose על צומת, קיים מסלול שלם מהצומת לשורש ואפשר לבצע עליו add_path_cost או find_path_cost.

- פעולת splice – הפיכת קשת לשלמה והוספתה למסלול תוך כדי פעולת expose, בתהליך מחליפים את הקשתות השלמות בשבורות, כדי לשמור על האנווריאנט, במידת הצורך.

משפט

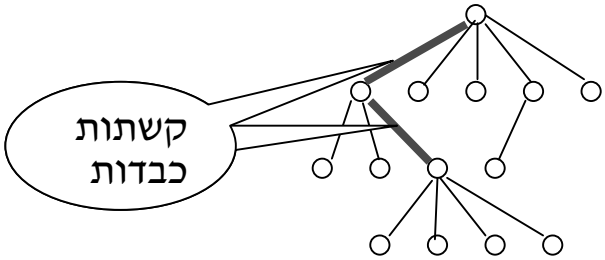
ניתן לבצע סדרה של m פעולות על עצים דינמיים ע"י $O(m)$ פעולות על מסלולים הכוללות $O(m \log n)$ פעולות splice.

הוכחה

הגדרות:

- עבור צומת v בעץ T, size(v) = מספר צאצאים כולל v עצמו.
- קשת (u,w) היא כבדה אם: 2*size(u) > size(w)

דוגמה:



למה 5.1:

בכל מסלול מצומת v לשורש יש לכל היותר $\lfloor \log n \rfloor$ קשתות קלות.

הוכחה

אם (u,w) היא קשת קלה, אזי size(w) ≥ 2size(u) ומאורע זה אינו יכול לקרות יותר מאשר $\log n$ פעמים, לאורך מסלול מצומת v לשורש.

למה

לכל צומת נכנסת לכל היותר קשת כבדה אחת.

הסבר

אחרת, שני תתי העצים היו גדולים מתת עץ האב.

הוכחת המשפט

1. O(m) פעולות נובע מהקוד.
2. O($m \log n$) פעולות splice.

כל פעם שמבצעים splice הופכים קשת שבורה לשלמה.
נסתכל על 2 סוגי קשתות בהפיכת קשת שבורה לשלמה

1. שלמות כבדות
2. שלמות קלות

בכל פעולת expose רק הקשתות על המסלול יכולות להפך משבורות לשלמות. מספר הקשתות השלמות שנוצרות שווה למספר הקשתות השלמות הקלות + מספר הקשתות השלמות הכבדות.

שלמות קלות: יש לכל היותר $\log n$ קשתות קלות לאורך מסלול ולכן גם לאורך מסלול יש לכל היותר $\log n$ שלמות קלות. לכן בכל פעולת expose נוצרות $O(\log n)$ קשתות שלמות קלות. כיון שיש $O(m)$ פעולת expose, במהלך האלגוריתם נוצרות $O(m \log n)$ קשתות שלמות קלות.

שלמות כבדות: נחסום את מספר הקשתות השלמות הכבדות שנוצרות. יהי N מספר שלמות כבדות, במהלך כל האלגוריתם:

$$N_{final} = N_{init} + N_{created} - N_{destroyed}$$

$$N_{created} \le N_{destroyed} + n \qquad <=$$

- expose

הקשתות השלמות על המסלול לא יהרסו. אם (u,w) קשת שלמה כבדה שנהרסה אזי (u',w) היא קשת של המסלול מ-v לשורש ועל כן (u',w) קלה. => מספר הקשתות הכבדות השלמות שנהרסו $\geq$ מספר הקשתות הקלות על המסלול מ-v לשורש $\geq \log n$.

- כתוצאה מ-cut/link משתנה מבנה העצים ביער ועל כן, קשת יכולה להפוך מכבדה לקלה (ולהפך).

Cut

כיוון שלפני הניתוק מבצעים expose כל המסלול מ-v לשורש הוא של קשתות שלמות, לכן הקשתות שליד המסלול הן שבורות ולא כבדות שלמות שנהרסות. כיוון שלאחר הניתוק יתכנו $\geq \log n$ קשתות קלות=> על המסלול היו לכל היותר $\log n$ קשתות כבדות שנהרסו.

Link

קודם מבצעים expose לכן כל קשתות המסלול שלמות וכל אלה לצד המסלול שבורות. כיוון שלצמתי המסלול ה-size גדל, אף קשת שלמה כבדה לא נהרסת.

מימוש פעולות על עצים דינמיים בעזרת פעולות על מסלולים

```
procedure make_tree(vertex v);
     make_path(v);
     successor(v) := NULL;
end make_tree;

vertex function find_root(vertex v);
     p = expose(v);
     return find_tail(p);
end find_root;

list function find_cost(vertex v);
     p = expose(v);
     return find_path_cost( p );
end find_cost

procedure add_cost(vertex v, real x);
     p = expose(v);
     add_path_cost(p, x);
end add_cost;

procedure link(vertex v, w);
     p = expose(v);
     q = expose(w);

     1.  join(NULL, p, q);
end link;

procedure cut(vertex v);
     path p,q;
     2.  expose( v );
     3.  [p,q] := split( v);
end cut;
```

Note:
The expose in line 1 of link makes v into a one vertex solid path. After line 2 in cut, vertex v has no entering solid edge, thus the path p returned by the split in line 3 is empty. Since in cut vertex v is not a tree root, the path q returned in line 3 is nonempty.

```
path function expose(vertex v);
     path p, q, r;
     vertex w;

     p = find_path(v);
     v = find_tail(p);

     while(v != root)
     {
        w = parent(v);
              //i.e., w is the next vertex up the path to the root of the tree.

        [q, r] := split(w);
              // r  is the upper part of the path containing  w
       p = join(p, w, r);
       v = find_tail(p);
     }
     return p;
end expose;
```