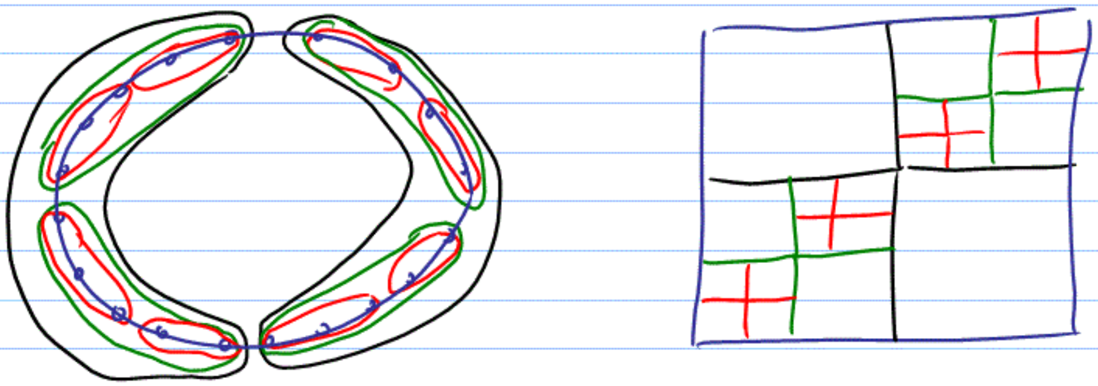
הסרת ההנחה:

בהינתן G_i', נפרק אותו לגרפים דו-צדדיים.



יש $2 + 4 + \dots + N = O(N)$ גרפים דו-צדדיים (בגדלים שונים)

כל קדקד נמצא ב-$O(\log N)$ גרפים דו-צדדיים.

לכל גרף דו צדדי יהיה Monge heap משלו, שתומך ב- Activate וב- ExtractMin.

כדי לממש את Dijkstra, כל גרף דו צדדי יתרום את הנציג המינימלי שלו לערימה הגלובלית Q של Dijkstra.

כלומר, ב- Q יהיו $O(N)=O(\sqrt{n})$ איברים.

יש לנו כעת יותר מעותק אחד של כל קדקד. כשנממש Dijkstra עותקים שונים של אותו קודקוד יכולים להופיע כמינימום הגלובלי. ב- Dijkstra כל קדקד מופיע בדיוק פעם אחת כמינימום הגלובלי. רק ההופעה הראשונה היא אמיתית. השאר הן תוצאה של השכפול ואין צורך לעשות דבר, פרט להוצאת העותק מה- Monge heap בו הופיע.

FR-Dijkstra(G'_{in}, G'_{out}, s)

$d(s) = 0$; $d(v) = \infty$ $\forall v \neq s$

decompose into bipartite graphs

initialize a Monge heap for each bipartite graph

initialize heap Q w/ min represetative from each Monge heap

$S = \{s\}$

While $S \neq V(G)$:

  $v, \mathcal{M}, d_v \leftarrow Q.\text{ExtractMin}$

  $\mathcal{M}.\text{ExtractMin}$ — להוציא את v מ-$\mathcal{M}$

  $Q.\text{updateHeap}(\mathcal{M}.\text{FindMin})$ — לעדכן את הנציג של $\mathcal{M}$ ב-Q

  if $v \notin S$: — אם זהו המרחק הראשון של v שהוצא מינימום גלובלי. צריך לטפל ב-v

    $d(v) \leftarrow d_v$

    $S \leftarrow S \cup \{v\}$

    for each $\mathcal{M}$ that contains v: — הפעל את v בכל הגרפים שמכילים אותו

      $\mathcal{M}.\text{Activate}(v)$

      $Q.\text{UpdateHeap}(\mathcal{M}.\text{FindMin})$ — ועדכן את הערימה הגלובלית

זמן ריצה :

כל עותק של G קדקד מוצא לכל היותר פעם אחת כמינימום

של הערימה Q. לכן מספר האיטרציות $O(N \log N)$

כל פעולה $Q.ExtractMin$

$M.ExtractMin$

$M.Activate$

$M.FindMin$

לוקחת $O(\log N)$, כך שסך הכל זמן הריצה

$$O(N \log^2 N) = O(\sqrt{n} \log^2 n)$$

כפי שטענו.

Note Title

Exact Distance Oracles

אני רוצה להראות די בקצרה אפליקציה של FR-Dijkstra.

לבעיית MSSP.

Distance Oracle - מבנה נתונים שעונה על שאילתות מרחק בין זוגות קדקודים בגרף ביעילות.

הפרמטרים שיעניינו אותנו:

- גודל מבנה הנתונים (space)
- זמן הבניה של מבנה הנתונים (preprocessing time)
- זמן השאילתות (query-time)

באפליקציות רבות (למשל mobile devices), השטח וזמן השאילתא יותר חשובים מאשר זמן הבניה של מבנה הנתונים (נעשה off line).

נתעניין בעיקר ב- tradeoff בין space ל- query time.

פתרונות טריוויאלים

	preprocessing	space	query
SSSP	—	$O(n)$	$O(n)$
APSP	$O(n^2)$	$O(n^2)$	$O(1)$

פתרון נאיבי מבוסס r-division:

preprocessing:

חשב r-division עם $O(\frac{n}{r})$ אזורים, כל אחד עם $O(r)$ קדקודים

ו- $O(\sqrt{r})$ קדקדי שפה.

חשב $d(u,v)$ - APSP בין כל קדקדי השפה (מרחקים בגרף G כולו)

המקום הנדרש הוא $O((\frac{n}{\sqrt{r}})^2)=O(\frac{n^2}{r})$.

אם משתמשים ב- r-division עם מספר קבוע של חורים,

לחשב המרחקים בזמן $O(\frac{n}{r}\cdot n\log n)=\tilde{O}(\frac{n^2}{r})$ (כמו פר-דייקסטרה, אבל יש לכך פרטים לדעתה נלאה.)

בזמן שאילתה למרחק מ-u ל-v:

יהיו R_u, R_v האיזורים בחלוקה שמכילים את u, v בהתאמה.

- חשב מרחקים קדקוד $d_u(\cdot)$ מ-u ב- R_u $O(r)$

- חשב מרחקים קדקוד $d_v(\cdot)$ אל v ב- R_v $O(r)$

מצא את הזוג $x\in\partial R_u$, $y\in\partial R_v$ שממזערים את $O(\sqrt{r}\cdot\sqrt{r})=O(r)$

$$d^* = d_u(x)+d(x,y)+d_v(y)$$

החזר את הקטן מבין d^* ו $d_u(v)$ ($d_u(v)$ מוגדר רק אם $R_u=R_v$)

זמן שאילתה $O(r)$.

קיבלנו tradeoff מקום $O(\frac{n^2}{r})$ עם שאילתא $O(r)$.

בפרט, אם רוצים מקום לינארי, זמן השאילתא הוא $O(n)$.

נשתמש ב- FR-Dijkstra כדי לקבל זמן שאילתא יותר טוב.

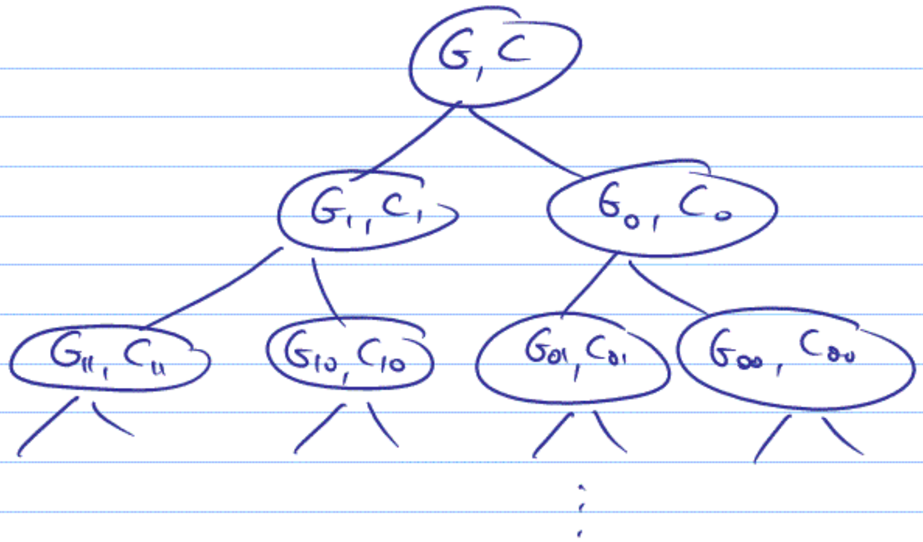
preprocessing: חלק את הגרף G רקורסיבית באמצעות מפריד מעגלי. כשמפרידים את גרף G' באמצעות מפריד מעגלי C לתת-גרף G'_{in} ו- G'_{out}, חשב מבנה MSSP עבור C ב- G'_{in} וב- G'_{out} ו- DDG עבור C (כולל RMQ) ב- G'_{in} וב- G'_{out}.

הזמן הנדרש הוא $O(n\log n)$ בכל רמה של הרקורסיה, ולכן $O(n\log^2 n)$ בסך הכל.

המקום הנדרש הוא $O(n\log n)$ עבור ה- MSSPs, ו- $O(n)$ עבור ה- DDGs, בכל רמה של הרקורסיה, ולכן $O(n\log^2 n)$ בסה"כ.

<u>שאילתא</u>: מרחק מ- u ל- v.

חשבו על העץ הבינארי T שמתאר את החלוקה הרקורסיבית של G. כל צומת של העץ מתאים לתת-גרף G' של G. אם הצומת אינו עלה של העץ, אז מתאים לו גם המעגל C שמפריד את G'.

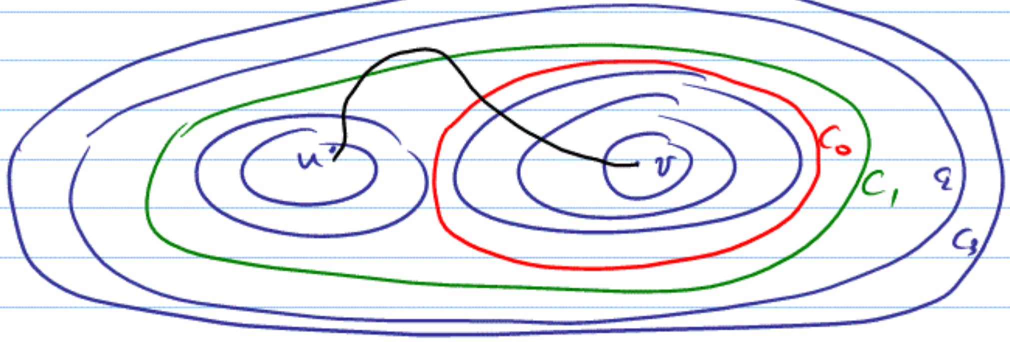


יהי C_0 המעגל הקרוב ביותר לשורש של T המפריד בין u ל-v

(זהו ה-LCA של u ו-v ב-T).

יהיו $C_1, \dots, C_k$ האבות של C_0 ב-T, כאשר C_k הוא המעגל בשורש.

יהי $0 \le i$ האינדקס המקסימלי עבורו המסלול הקצר ביותר מ-u ל-v חוצה את C_i. i מוגדר היטב מכיוון שכל מסלול מ-u ל-v, ובפרט המסלול הקצר ביותר, חייב לחצות את C_0.



נניח שאנו יודעים את C_i. אז ניתן לחשב את המרחק מ-u ל-v

ע״י FR-Dijkstra על ה-DDGs של C_i (פנימי וחיצוני)

במספר $2|C_i|$ קשתות בין u לקודקודים של C_i

וכן הקודקודים של C_i ל-v. האורכים של הקשתות הללו נתונים

ע״י מבנה ה-MSSP של C_i.

זמן החישוב הוא $O(|C_i| \cdot \log^2 n) = O(\sqrt{n}\log^2 n)$

מכיוון שאיננו יודעים את C_i, ננסה את כל הערכים האפשריים. יש $O(\log n)$

כאלה (עומק העץ T הוא $O(\log n)$).

לכן, זמן השאילתא הוא $O(\sqrt{n}\log^3 n)$ (למעשה $O(\sqrt{n}\log^2 n)$

כי גדלי המעגלים קטנים גיאומטרית).

קיבלנו $\tilde{O}(n)$ מקום! $\tilde{O}(\sqrt{n})$ שאילתא (קודם השאילתא היתה $O(n)$)

למעשה, ניתן לשלב r-divisions עם מבנה הנתונים הזה כדי

לקבל $\tilde{O}(\frac{n^2}{r})$ מקום! $\tilde{O}(\sqrt{r})$ שאילתא (בתרגול)

Branch Decomposition & Baker's technique

Note Title

אנו עוברים לדבר על קירובים לבעיות NP קשות.

היום נלמד טכניקה כללית לאלגוריתמי קירוב בגרפים מישוריים שמבוססת

על פירוק הגרף המקורי לתתי-גרפים שעליהם "קל" לפתור את הבעיה,

ומהפתרונות האופטימליים לתתי-הגרפים ניתן להרכיב פתרון מקורב לבעיה

המקורית.

נעבוד על Vertex Cover כדוגמה לאורך ההרצאה, הטכניקה עובדת

לבעיות רבות אחרות (..., dominating set, independent set)

ויכולה לעבוד גם בבעיות ממושקלות.

נתחיל בבעיה קלה: VC של עץ.

ניתן לפתור באמצעות תכנות דינאמי:

עבור כל קודקוד v בעץ T מהעלים לכיוון השורש נשמור שני גדלים:

X_v - גודל הפתרון האופטימלי בתת העץ של v

Y_v - גודל הפתרון האופטימלי בתת העץ של v, כש-v חייב להיות בפתרון.

עבור עלה v, $x_v = 0$, $y_v = 1$

עבור קדקד פנימי v עם ילדים $v_1, v_2 \dots v_k$:

$$y_v = 1 + \sum_i x_{v_i}$$

$$x_v = \min \{ y_v, \sum_i y_{v_i}$$

VC מינימלי לכל T ששורשו r הוא בגודל x_r.

מה ניתן לעשות כשהגרף אינו עץ?

בפתרון לעצים השתמשנו בפתרון לתתי גרפים (תתי-עצים) קטנים כדי לקבל פתרון לתת-גרף גדול יותר. הסיבה שקל לעשות זאת היא שהאינטראקציה בין תתי עצים מוגבלת מאוד - כל תת עץ מתחבר לשאר העץ באמצעות קשת אחת שמחוברת לשורש של תת העץ.

ננסה לפרק גרף שאינו עץ לתתי-גרפים הולכים וקטנים, שהאינטראקציה ביניהם מוגבלת.

ניתן לעבוד עם תתי גרפים המושרים על ידי:

קבוצות של קדקודים $\Leftarrow$ carving decomposition

או קבוצות של קשתות $\Leftarrow$ branch decomposition

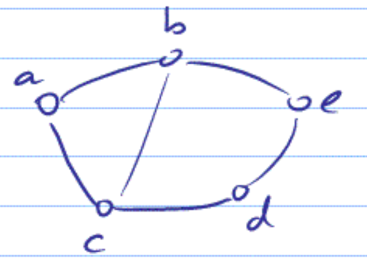
נתחיל עם תתי גרפים מושרי קדקודים. Carving Decomposition

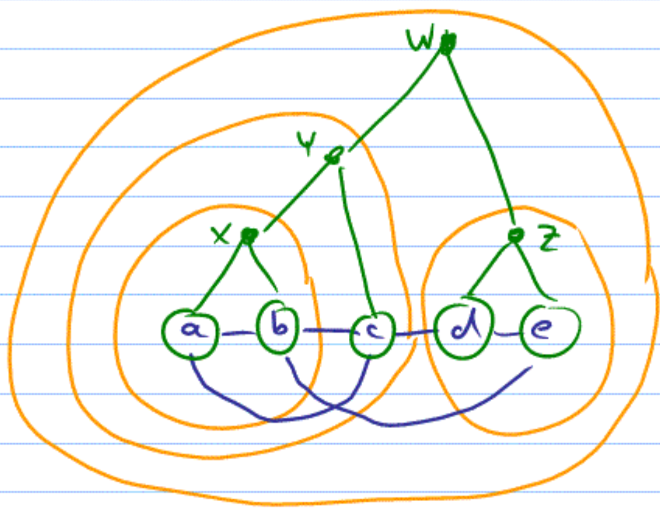
יהי $G=(V,E)$ גרף.

יהי T עץ בינארי שעליו הם הקדקודים של G

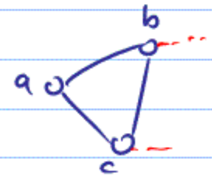
כל צומת $t\in T$ מתאים לתת גרף של G המושרה ע"י העלים

בתת העץ T_t של T המושרש ב-t.





למשל - תת הגרף שמתאים ל-T_Y הוא $G_Y=$



מהי האינטראקציה של G_Y עם $G\setminus G_Y$?

הקשתות בחתך $\delta_G(Y)=\delta_G(\{a,b,c\})$.

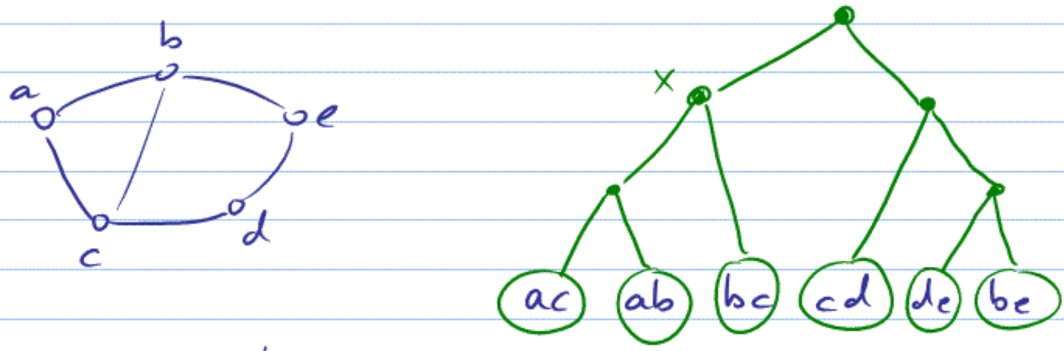
הרוחב של T הוא $\max_{X\in T}|\delta_G(X)|$

ה-carvingwidth של G הוא הרוחב המינימלי של carving decomp. של G.

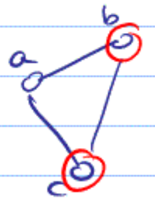
טענה: carvingwidth $\geq$ max degree

כעת - Branch decomposition - חלוקה לתת גרפים מושרי קשתות

ההבדל הוא שהעלים של העץ הבינארי T הם כעת הקשתות של G.



תת הגרף שמתאים ל- T_x הוא

$G_x =$ 

ש: האינטראקציה של G_x עם שאר הגרף?

$\partial_G(x)$ - הקדקודים של G_x שתלוי לידם קשת שאינה ב- G_x

הרוחב של T הוא $\max_{x \in T} \partial_G(x)$.

ה. branchwidth של G הוא הרוחב המינימלי של branch decomp. של G.

דוגמה: חישוב VC באמצעות DP על branch-decomposition,

לכל $X \in T$, לכל $S \subseteq \partial_G(X)$,

$M_X[S] =$ גודל VC מינימלי של $G(X)$ שמשתמש ב-S
ולא משתמש ב-$\partial_G(X) \setminus S$.

עבור קדקוד בודד - טריוויאלי. אם W אבא של X, Y

$$M_W[S] = \min_{S_X, S_Y} M_X[S_X] + M_Y[S_Y] - |S_X \cap S_Y|$$

where: $S_X \subseteq \partial_G(X)$, $S_Y \subseteq \partial_G(Y)$ $\quad (S_X \cup S_Y) \cap \partial_G(W) = S$

מה זמן הריצה?

לבנות את $X \cup Y$ מתוך X, Y לוקח $|\partial_G(X)| \cdot |\partial_G(Y)| = O(2^{bw} \cdot 2^{bw})$

כמה צמתים יש ב-T? $O(n)$

$\Leftarrow$ זמן ריצה $O(4^{bw} \cdot n)$

ליניארי אם branchwidth קבוע!