בס"ד

אלגוריתמים - סיכום

תכנות חמדני:

* **אלגוריתם חמדני** - הוא אלגוריתם שבכל רגע נתון מחפש את הבחירה שנראית הטובה ביותר כרגע.
* **אלגוריתם אופטימלי** - הוא אלגוריתם שמקיים את התכונות הבאות:
  ○ הבעיה מקיימת את תכונת הבחירה החמדנית.
  ○ הבעיה מקיימת את תכונת תת המבנה האופטימלי.
- אלגוריתם חמדני מבצע רצף של בחירות, כל אחת מהבחירות שנראית הכי טובה כרגע - נבחרת.
- אלגוריתם חמדני לא תמיד מייצר פתרון אופטימלי!

○ **הבחירה החמדנית - תבנית כללית:**
- עם הבחירה החמדנית...
- נראה שאומנם קיים פתרון אופטימלי שבו...
- ניקח פתרון אופטימלי x לבעיה:
  ○ אם x מכיל את הבחירה החמדנית - סיימנו!
  ○ אם x אינו מכיל את הבחירה החמדנית, נראה איך אפשר לשנות אותו כך שכן יכיל את הבחירה החמדנית.
  ולהגיע לפתרון חדש ש:
  - או שהוא פתרון טוב יותר. כלומר, סתירה להנחה שx אופטימלי.
  - או שהוא שקול באופטימליות לx. כלומר, קיים גם פתרון אופטימלי שמכיל את הבחירה החמדנית.
  ○ מסקנה: קיים פתרון אופטימלי המכיל את הבחירה החמדנית!

○ **תת מבנה אופטימלי - תבנית כללית:**
- נגדיר מהי הבעיה ומהי תת הבעיה.
- נראה שאם x הוא פתרון אופטימלי לבעיה, אזי x' שהוא הפתרון x בלי הבחירה הראשונה הוא אופטימלי לתת בעיה:
  ○ נניח בשלילה שx הוא אופטימלי לבעיה, אבל x' אינו אופטימלי לתת בעיה.
  ○ אז קיים פתרון y' אחר שהוא אופטימלי לאותה תת בעיה, כלומר...
  ○ נוסיף לy' את הבחירה הראשונה שהייתה בx...
  ○ ונקבל פתרון y לבעיה המקורית, שהוא יותר טוב מאשר x, כיון ש...
- בסתירה להנחה שx אופטימלי!

בס״ד

אלגוריתמים - סיכום

תכנות חמדן

* קוד הופמן - שיטה לדחיסת נתונים.

המטרה: מקבלים סדרה של תווים, נרצה לשמור אותה בצורה היעילה ביותר - מבחינת מקום

- איך האלגוריתם עובד? כותבים לכל איבר את השכיחות שלו ומתחילים לבנות את העץ מכל האיברים בתור עלים, בכל שלב לוקחים את שני האיברים בעלי השכיחות הנמוכה ביותר וקודקוד פנימי (המשמש שורש) שערכו סכום השכיחויות, מכניסים אותו לתור וכך בונים את העץ. לאחר מכן כל צלע שמאל תהיה 0 וצלע ימין 1, וכך מקודדים את האותיות כאשר כל אות רק בעלים וקוראים את הקוד מהשורש כלפי מטה.
- לוקחים קוד עם תווים רגילים והופכים אותו לתווים בינאריים, כאשר התו שמופיע הכי הרבה פעמים ניתן לו מינימום תווים בינאריים.
- שלבים: מיון השכיחויות מהקטן לגדול, מחברים את 2 הקטנים ביותר, ממיינים שוב עם הקודקוד החדש וחוזרים על התהליך. אחרי שבנינו את העץ ניתן לדעת מה הקוד של כל עלה.
- שואלים מה אורך הטקסט החדש - אורך הפלט כזה, נחשב את השכיחויות כפול מס׳ התווים של האות המסוימת. דוגמה חוקית: $\overset{\text{שכיחות}}{A:10} \to \overset{\text{קוד}}{1}$, $e:5 \to 10$, $b:2 \to 110 \Rightarrow 10\times1+5\times2+2\times3=\underline{26}$
- שואלים מה יחס הקידוד בין קוד הופמן לקוד הקבוע - נניח יש לנו 5 תווים נצטרך $5 \leq 2^x = \underline{3}$ תווים ואז קידוד באורך קבוע בה כל אחד מהתווים כפול 3 נניח: $10\times3+5\times3+5\times3+2\times3+3\times3=75$

יחס הקידוד/יחס הדחיסה: $\frac{55}{75}$ $\frac{\text{אורך (משתנה/הופמן)}}{\text{אורך קבוע}}$

בסד

אלגוריתמים - סיכום

תכנון דינאמי

* תכנון דינאמי פותר בעיות אופטימיזציה ע"י שימוש בצירוף פתרונות של תתי בעיות.

- סדרת בחירות חייבת להוביל לפתרון אופטימלי עבור הבעיה.
- הבחירות שאנחנו מבצעים משאירות אותנו הרבה פעמים עם המשך פתרון לתת בעיה שהיא אותו הדבר כמו הבעיה המקורית.
- המטרה לפתור בעיות אלו תוך כדי שמירת הפתרונות של תתי הבעיות בצורה מתאימה.

- מה זה תכנון דינאמי? משתמשים בשיטת "הפרד ומשול":
  - מחלקים את הבעיה לתתי בעיות בלתי תלויות.
  - פותרים את תתי הבעיות בצורה רקורסיבית.
  - מאחדים את תתי הפתרונות לפתרון כולל לבעיה הכוללת.
- תכנון דינאמי מתאים במקרים של תתי בעיות בלתי תלויות.
- 4 השלבים של פתרון הבעיה:

1) נאפיין את מבנה הפתרון האופטימלי
2) נגדיר רקורסיבית את הערך של הפתרון האופטימלי.
3) נחשב את הערך של הפתרון האופטימלי מלמטה למעלה.
4) נבנה פתרון אופטימלי מהמידע המחושב - פתרון של תתי הבעיות הרלוונטים.

* ההבדל בין השיטה של תכנון דינאמי לבין השיטה החמדנית - בשיטה החמדנית יש מצבים שפתרון ביניים נראה טוב מפתרונות אחרים באותו שלב, אז הפתרון הזה נבחר ואין דרך לשנות אותו לאור תוצאות שמתקבלות יותר מאוחר. לעומת זאת בתכנון דינאמי עד לפני השלב האחרון לא ידוע אילו שלבי ביניים יבחרו לפתרון הסופי.

* מאפייני תכנות דינאמי - פתרון דינמי מתאים לבעיות הבאות:

- מתקיים תת מבנה אופטימלי - פתרון אופטימלי לבעיה מכיל בתוכו פתרונות אופטימליים לבעיות משנה.
- בעיות משנה חופפות - מרחב תתי הבעיות חייב להיות קטן. כלומר, הבעיות משנה חוזרות שוב ושוב.

- בד"כ מה שמשפיע על זמן הריצה של האלגוריתם זה:

1) מס' תתי הבעיות הכולל , כמה אפשרויות פתרון יש לכל תת בעיה.

- תכנות דינאמי משתמש בתת מבנה אופטימלי מלמטה למעלה. (Bottom-Up)
- בד"כ באלגוריתם של תכנות דינמי נשלם במקום על מנת לחסוך בזמן

בס"ד

אלגוריתמים - סיכום

* בעיית תרמיל הגב בשלמים - משתמשים בתכנות דינאמי כאשר בכל שלב אנחנו בודקים האם כדאי לנו לקחת פריט שלם או לא. בזמן ביצוע שלב של הפתרון התשובה בריבוע הכי ימין למטה של המטריצה. כאשר משחזרים את הפתרון הולכים לריבוע הבא ואם הוא יותר גדול מהריבוע שמעליו סימן שבחרנו לקחת את הפריט הזה, אז אנחנו נחסיר מהמשקל הקיים את המשקל של הפריט הזה כיון שבחרנו לקחת אותו, ונמשיך הלאה לעלות בטבלה עד שנגיע לרשימת הפריטים שלקחנו.

$$C[i,W] = \begin{cases} 0 & \text{if } i=0 \text{ or } W=0 \\ \overset{\text{אם לא ניתן לקחת את הפריט נעתיק את הערך שבריבוע מעל}}{C[i-1,W]} & \text{if } w_i > W \\ \max\left[\overset{\text{בחרנו לקחת את פריט } i}{v_i + C[i-1,W-w_i]},\ \overset{\text{לא בחרנו לקחת את פריט } i}{C[i-1,W]}\right] & \text{if } i>0 \text{ and } W \geq w_i \end{cases}$$

תנאי עצירה! לא לקחת

לא לקחת

אפשר לקחת

- זמן הריצה עם n שורות ו-m עמודות $\Theta(m*n)$
- השיטה החמדנית תעבוד במקרה שהמשקלות הן חזקה אחת של השניה.

* LCS תת סדרה ארוכה ביותר - יש לנו שני קלטים x ו-y ואנחנו רוצים למצוא את תת הסדרה המשותפת הארוכה ביותר אם אנחנו במשבצת שבו $x_i=y_i$ אז ניקח את הערך מהאלכסון ונוסיף 1, במידה ו-$y_i \neq x_i$ ניקח את הערך המקסימלי מבין המשבצת מעל או המשבצת מצד שמאל. הפתרון נמצא במשבצת הכי ימנית למטה. האלגוריתם הרקורסיבי:

$$C[i,j] = \begin{cases} 0 & j=0 \text{ OR } i=0 \\ \overset{\text{ניקח את הערך באלכסון ונוסיף 1}}{C[i-1,j-1]+1} & x_i=y_i \quad i,j>0 \\ \overset{\text{ניקח את המקסימלי מבין המשבצת למעלה או משמאל}}{\max(C[i,j-1],C[i-1,j])} & x_i \neq y_i \quad i,j>0 \end{cases}$$

תנאי עצירה

אותיות שוות

אותיות שונות

זמן הריצה עם n שורות ו-m עמודות $O(m*n)$

- במידה ונותנים לנו סדרה מסוימת ואומרים לנו למצוא את תת סדרה הממוינת הגדולה ביותר אז אנחנו פשוט שולחים את הסדרה עצמה כ-x ואת הסדרה הממוינת כ-$y=sort(x)$ והאלגוריתם ימצא.
- לא לשכוח בטבלה שורה ראשונה ועמודה ראשונה אפסים בגלל תנאי עצירה.

- בכל שאלות של תכנות דינאמי צריך לבדוק איך עוברים על הקלט מהסוף או מההתחלה ואז התשובה הסופית נמצאת ב-$S[n]$ או ב-$S[1]$ בהתאם.
- תיאור נוסחה שבה בחרנו את הקלט - כאשר התשובה נמצאת בהתחלה התשובה נמצאת ב-$S[1]$.

$$S[i] = \max(\overset{\text{הסכום}}{p[i]} + S[\overset{\text{אורך הקלט}}{f[i]}+i+1],\ S[i+1])$$

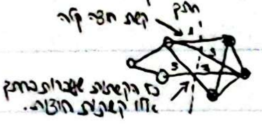
בס"ד

אלגוריתמים - סיכום

עץ פורש מינימלי MST

* עץ פורש - תת קבוצה של צלעות כך שהגרף קשיר (יש מסלול מכל קודקוד לכל קודקוד) ואין מעגלים.

* עץ פורש מינימלי - בהינתן גרף ופונק' משקל, עץ פורש מינימלי הוא עץ אשר סכום משקלי הצלעות שבו נמוך מסכום משקלי הצלעות של שאר העצים הפורשים.



- חתך מכבד - קבוצת צלעות אם אף אחת מהצלעות לא חוצה את החתך.
- צלע בטוחה - קשת חוצה (שעוברת את החתך) בעלת המשקל הנמוך ביותר בחתך מכבד.

* אלגוריתם Kruskal - הרעיון: נמיין את כל משקלי הצלעות מהנמוך ביותר לגבוה, לאחר מכן נפעיל את האלגוריתם למציאת קבוצות זרות לפי סדר הצלעות מהקטן לגדול. כלומר, בכל שלב ניקח את הצלע בעלת המשקל הנמוך ביותר שלא סוגרת מעגל עד שנגיע ל $v-1$ צלעות (גרף קשיר ללא מעגלים). אלגוריתם חמדני. זמן ריצה $\Theta(|E|\log|E|)$

* אלגוריתם Prim - נתחיל מקבוצת צלעות ריקה A ומקודקוד התחלתי. כאשר בכל שלב נבחר צלע בעלת משקל מינימלי מבין כל הצלעות שעדיין לא נבחרו שמחברת קודקוד שכבר הגענו אליו עם קודקוד שעדיין לא הגענו אליו ומוסיפים לקבוצה A.

מבחינת המימוש: לכל קודקוד יש 2 שדות: * key (הערך המינימלי המוביל אליו), * π (מצביע לאבא).

בתחילת הריצה נאפס את כל הקודקודים $key=\infty$, $\pi = null$ (חוץ מקודקוד ההתחלה). וניצור תור קדימויות מינימלי לפי ערכי ה key. כאשר בכל הוצאה נעדכן את המשקלים. אלגוריתם חמדני.

זמן ריצה: $O(E\log V)$ בערימה (נשתמש בגרף דליל). $O(V^2)$ במערך (נשתמש בגרף צפוף).

- בגרף דליל (כאשר $|E|=\Theta(V)$) פריים וקרוסקל יעילים באותה מידה.
- בגרף צפוף ($|E| = \Theta(V^2)$) פריים יותר יעיל.

* תכונת המעגלים הקלים - כאשר יש לנו מעגל בגרף, ניתן להסיר את הצלע בעלת המשקל הגדול ביותר מבלי להשפיע על עץ פורש מינימלי.

- לבעיית ה MST יש את תכונת תת המבנה האופטימלי, מוצגים 2 אלגוריתמים: פריים וקרוסקל ובשניהם בוחרים בכל שלב צלע בעלת משקל מינימלי שמתאים לעץ.
- חשוב! תכונת תת מבנה אופטימלית לא אומרת שפתרון אופטימלי של כל תת בעיה בהכרח יהיה חלק מהפתרון האופטימלי של הבעיה כולה. אלא תכונה זו אומרת שבהינתן פתרון אופטימלי לבעיה, פתרון זה מכיל פתרונות אופטימלים של תתי בעיות.
- כאשר נתון לנו גרף נוסף H שהוא תת גרף של G, אז בבניית עץ פורש מינימלי ל G, לא נבחן מחדש את כל הצלעות, אלא רק את הצלעות של העץ פורש המינימלי של H ואת הצלעות "החדשות" שלא נבחנו בכלל - לא נמצאות בH. (כמו כן, לא "נחזור" על צלעות שלא לקחנו בH).

אלגוריתמים - המשך סיכום

עץ פורש מינימלי

* <u>צואר בקבוק</u> - בעץ פורש מינימלי צואר הבקבוק הוא הצלע בעלת המשקל הגדולה ביותר בעץ הפורש המינימלי.

- כל עץ פורש מינימלי של G הוא עץ עם צואר בקבוק מינימלי של G! אבל לא כל עץ עם צואר בקבוק מינימלי של G הוא עץ פורש מינימלי של G.

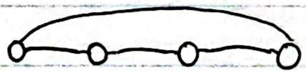
• בעיית העץ פורש המינימלי זו בעיית אופטימיזציה! (כמובן שיכולים להיות יותר מ MST אחד).

<u>חשוב לזכור!</u>

- עץ פורש מינימלי - בהכרח גרף לא מכוון.
- עץ פורש שמקיים את תכונת המעגלים הקלים הוא מינימלי.
- עץ פורש מינימלי מקיים את תכונת המעגלים הקלים.
- אם מוצאים עץ פורש מינימלי אז בגלל תכונת החתך המינימלי - לפי העקרון שראינו <u>זה לא מחייב</u> שעבור כל שני קודקודים בגרף המסלול היחיד בעץ הפורש המינימלי הוא גם מסלול בעל משקל מינימלי בין שני הקודקודים בגרף.
- עץ פורש מינימלי אינו בהכרח יחיד, אבל כאשר לכל קשת בגרף יש משקל שונה - העץ יחיד.
- אם נוסיף קבוע a לכל הקשתות העץ הפורש המינימלי לא ישתנה (כיון שמה שקובע אם קשת תהיה חלק מה MST הוא השוואת המשקלים היחסיים ולא הערכים המוחלטים שלהם).
- אם נכפיל את כל הקשתות בקבוע חיובי גם לא ישתנה לנו העץ הפורש שנקבל. (אבל בקבוע שלילי סדר הקשתות ישתנה ולכן העץ לא בהכרח יישאר אותו הדבר).

* <u>האלגוריתם הכללי למציאת עץ פורש</u> - מוצאים אוסף צלעות שמחברות בין כל קודקודי הגרף, כך שיווצר עץ.
האלגוריתם: יוצרים קבוצת צלעות ריקה, ובכל פעם מוסיפים קשת "בטוחה" - שלא פוגעת בתכונות העץ - כלומר לא סוגרת מעגל. לקבוצה.

○ דבר פשוט בשביל דוגמה נגדית למקרה קצה נשתמש בגרף כזה:



- בעץ פורש מינימלי עבור כל שני קודקודים בגרף, המסלול היחיד בעץ פורש המינימלי <u>אינו בהכרח</u> המסלול בעל המשקל המינימלי בין שני קודקודים בגרף!

בס"ד

אלגוריתמים - סיכום

המסלולים הקצרים ביותר

* <u>בעיית המסלולים הקצרים ביותר</u> - נתון גרף מכוון ופונקצית משקל כאשר המשקל הוא סכום משקלות הקשתות המרכיבות את המסלול.

* <u>מסלול קצר ביותר</u> - מקודקוד u אל קודקוד v מוגדר כמסלול p כלשהו שמשקלו $\delta(u,v) = w(p)$.

□ <u>למה 1</u>: תת שבילים של שבילים קצרים ביותר הם שבילים קצרים ביותר.

פורמלית: יהי $p=<v_1,v_2...v_k>$ שביל קצר ביותר מ-v_1 עד v_k. אזי השביל $p_{ij}=<v_i,...,v_j>$ $1\le i,j\le k$ הוא שביל קצר ביותר מ-v_i ל-v_j.

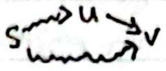
הוכחה: אחרת, היה שביל קצר יותר מ-v_1 ל-v_k

תוצאה: יהי שביל p הקצר ביותר בין s ו-v מורכב כך: $\underbrace{s \rightsquigarrow u \to v}_{p}$

אזי: $\delta(s,v) = \delta(s,u)+w(u,v)$

הוכחה: $\delta(s,v)=w(p) = w(p')+w(u,v) = \delta(s,u)+w(u,v)$

□ <u>למה 2</u>: לכל קשת $(u,v)\in E$. $\delta(s,v)\le\delta(s,u)+w(u,v)$



* Relax - משווה בין מרחק נוכחי לבין מסלול מוצע

```
Relax(u,v,w)
    if(d[v] > d[u] + w(u,v)) then      אם המרחק החדש קטן מהקיים
        d[v] = d[u] + w(u,v)           מעדכנים את d[v] להיות שווה למרחק החדש
        π[v] = u                       מעדכנים את האבא במסלול החדש.
```

□ <u>למה 3</u>: מיד לאחר ביצוע Relax מתקיים: $d[v] \le d[u] + w(u,v)$

□ <u>למה 4</u>: לאחר האתחול ולאחר ביצוע סדרת פעולות Relax מתקיים: $d[v] \ge \delta(s,v)$ $\forall v$, בנוסף לאחר ש-$d[v]$ נעשה שווה ל-$\delta(s,v)$ הוא אינו משתנה יותר.

תוצאה: כשאין שביל מ-s ל-v אזי לאחר האתחול $d[v]=\delta(s,v)=\infty$ וערך זה נשמר.

□ <u>למה 5</u>: יהי $s \rightsquigarrow u \to v$ שביל קצר ביותר. נניח שהגרף אותחל והתבצעה סדרת פעולות Relax וביניהם Relax(u,v,w) אם $d[u]=\delta(s,u)$ לפני הקריאה לרילקס אזי $d[v]=\delta(s,v)$ בכל רגע לאחר הקריאה.

□ <u>למה 6</u>: נניח שG לא מכיל מעגלים שליליים הניתנים להשגה מ-s, אזי לאחר האתחול תת הגרף G_π הוא עץ עם שורש s ולכל סדרה של פעולות Relax על G שומרת על תכונה זו.

□ <u>למה 7</u>: נניח שG לא מכיל מעגלים שליליים הניתנים להשגה מ-s לאחר ביצוע $\overset{\text{אתחול}}{ISS(G,s)}$ וסדרת פעולות Relax עד ש-$d[v]=\delta(s,v)$ לכל $v\in V$, מתקבל תת גרף G_π שהוא עץ המסלולים הקצרים ביותר עם שורש s.

בס״ד

אלגוריתם - סיכום

מסלולים קצרים ביותר

- האלגוריתם דייקסטרה, חמדני, מוצא לנו מסלול ממקור יחיד לכל הצמתים, עם אילוץ של משקלות לא שליליים, זמן ריצה $O(E \cdot \log V)$
- האלגוריתם בלמן פורד, חמדני, מוצא מסלול ממקור יחיד לכל הצמתים, עובד גם עם משקלות שליליים, זמן ריצה $O(VE)$
- האלגוריתם פלויד וורשל, דינאמי, מוצא מסלול מכל המקורות לכל הצמתים, עובד גם עם משקלות שליליים, זמן ריצה $O(V^3)$

* <u>האלגוריתם של דייקסטרה</u> - אלגוריתם למציאת מסלול קצר ביותר ממקור יחיד לכל הצמתים, פועל רק על משקלות חיוביים בלבד! איך האלגוריתם עובד? בקצרה אנחנו מקבלים גרף וצומת התחלה והפלט הוא המרחק המינימלי מצומת ההתחלה לכל הצמתים בגרף.

השלבים: - מעדכנים את כל הצמתים להיות במרחק ∞ וזאת חוץ מ s שמרחקו 0.

- יוצרים תור עדיפויות המכיל את הצמתים שטרם ביקרנו בהם.
- לאחר מכן נבחר את הצומת בעלת המרחק המינימלי מ s אותה "נבקר" ובכך נמשיך הלאה לכל השכנים, כאשר כל פעם נעדכן את המרחקים ע"י ריליקס (כלומר, עבור כל שכן v של הצומת הנבחר u אם המרחק מ v ל u בתוספת המשקל של הקשת (u,v) קטן מהמרחק הנוכחי לv, נעדכן את המרחק לv). נחזור על התהליך עד שכל הצמתים מבוקרים / עד שהתור ריק. זמן ריצה במימוש של ערימה: $O(E \log V)$.

* <u>האלגוריתם של בלמן פורד</u> - אלגוריתם למציאת מסלול קצר ביותר ממקור יחיד לכל הצמתים, בתוספת משקלות שליליים! - ברגע שהאלגוריתם מוצא מעגל שלילי הוא מחזיר לנו תשובה בוליאנית ע"כ ולא מציין איפה הבעיה או פותר אותה.

רעיון האלגוריתם: עוברים V-1 פעמים על כל הקשתות ומנסים לבצע עליהן שיפור (Relax). לאחר מכן מבצעים בדיקה יחידה על כל קשת (האם עדיין ניתן לעשות עוד relax) ובמידה והבדיקה חיובית כלומר יש לנו מעגל שלילי.

השלבים: - אתחול לכל הקודקודים את π=nil ו d=∞ חוץ מצומת ההתחלה שבו d=0.

- לאחר מכן מבצעים V-1 פעמים Relax על הקשתות,
- עבור כל קשת יש: אם $d[v] > d[u] + w$ אז מעדכנים $d[v] = d[u] + w$.
- לאחר מכן מבצעים מעבר נוסף על הקשתות ואם עדיין ניתן לשפר מרחק כלשהו כי קיים מעגל שלילי ומחזירים False.

זמן ריצה $O(VE)$.

בס״ד

אלגוריתמים - סיכום

מסלולים קצרים ביותר.

* <u>אלגוריתם פלויד וורשל</u> - אלגוריתם דינמי למציאת המסלול הקצר ביותר מכל המקורות אל כל היעדים, האלגוריתם תומך גם במשקלות שליליים. המיוחד באלגוריתם הזה שהוא לא מקבל את הגרף, אלא את טבלת הסמיכויות של הגרף - מטריצה בגודל n*n (v=n). המתארת באופן בסיסי את המעברים מקודקוד לקודקוד.

הרעיון המרכזי: מתבסס על דינמיקה חישובית, בכל שלב האלגוריתם בודק האם ניתן לשפר את המרחק בין צומת i לצומת j דרך צומת ביניים k.

השלבים: - בתחילה מאתחלים את מטריצת המרחקים ב0 מכל צומת לעצמו (באלכסון), W[i][j] אם יש קשת (ישירה) מ-i ל-j ו-∞ אם לא קיימת קשת ביניהם.

- לאחר מכן האלגוריתם עובר על כל צומת ביניים k ועבור כל זוג צמתים i,j מעדכן את המרחק לפי משוואת המעבר: ($D[i][j] = min(D[i][j], D[i][k] + D[k][j])$)
- האלגוריתם מזהה מעגלים שליליים אם $D[i][j] < 0$ (יש ערכים שליליים באלכסון).
- בסוף האלגוריתם קיימות 2 מטריצות בגודל n*n האחת מטריצה D שמכילה את המשקלות מכל קודקוד לכל קודקוד והשנייה מטריצה π שמכילה את ההורים.
- כל איטרציה תעבור על כל האפשרויות הקיימות בין 2 קודקודים כאשר בכל פעם נוסיף קודקוד למעגל הבדיקות. זמן ריצה $O(v^3)$.

* <u>גרף מכוון ללא מעגלים DAG</u> - אלגוריתם למציאת מסלולים קצרים ביותר בגרף מכוון ללא מעגלים. משתמש במיון טופולוגי שמפעיל DFS.

השלבים: - ביצוע מיון טופולוגי - כל צומת מופיע לפני כל הצמתים שהוא מצביע עליהם,

- לאחר מכן יוצרים מערך שמכיל את המרחק הקצר ביותר מכל צומת לצומת מקור
- מאתחלים את כל הצמתים ל-∞ מלבד צומת המקור שהוא 0. לאחר מכן נחשב את המרחקים, נעבור על כל הצמתים לפי סדר טופולוגי ולכל צומת נעדכן את המרחק (ממנו לצמתים השכנים) על סמך משקל הקשתות. זמן ריצה $O(E+V)$
- האלגוריתם יעיל יותר מדייקסטרה בזמן ריצה בגרף.
- אחד השימושים הנפוצים של האלגוריתם הזה הוא מציאת מסלול בעל משקל מקסימלי (מסלול קריטי) בגרף מכוון בעל משקלות חיוביים ללא מעגלים, כאשר לכל קודקוד יש מסלול מהמקור.

* <u>BFS</u> - ניתן להשתמש ב-BFS לחישוב מרחק ממקור לכל הצמתים בגרף <u>ללא משקלות</u>
ובחישוב מרחק בין כל זוג צמתים - נפעיל BFS |V| פעמים. זמן ריצה: $O(V+E)$
ניתן לקבל מ-BFS עץ רוחב המושרש בקודקוד מקור ומכיל את המסלולים הקצרים ביותר.

אלגוריתמים - סיכום

מסלולים קצרים ביותר

* חשוב לשים לב!

- בעץ פורש מינימלי - כאשר נוסיף/נחסיר משקל אחיד מכל הצלעות לא תשתנה. לעומת זאת
- בעץ מרחקים קצרים ביותר הוספה/חיסור של ערכים עלולה להיות הרסנית, כיון שהיחס עלול להשתנות. ואם יש מסלול שעובר בכמה קודקודים לעומת מסלול ישיר יקר יותר או להפך החסרה/הוספה של קבוע עלולה לגרום להעדפה של אותה הקשת על פני מסלול אחר.
- באלגוריתם פלויד וורשאל כל איטרציה נוסף קודקוד נוסף לחישוב, תחילה רק A אח"כ AB וכו'.
- בעץ מרחקים קצרים כאשר עושים שינוי ששומר על היחס (הכפלה/חילוק של קבוע) ברגע שהיחס בין המשקלים נשמר - כך העץ לא ישפיע על התוצאה!
אבל בחיסור או בחיבור היחס עלול להשתנות ועלול להשפיע על העץ.

לסיכום:

* <u>חישוב מרחקים ממקור לכל הצמתים:</u>

- BFS - גרף ללא משקלות - זמן ריצה $O(V+E)$
- Dijkstra - גרף עם משקלות חיוביים - זמן ריצה $\overset{\text{ערימה}}{O(E\log V)}$ / $\overset{\text{מערך}}{O(V^2)}$
- Belman-Ford - גרף עם משקלות כלליים - זמן ריצה $O(VE)$

* <u>חישוב המרחקים בין כל זוג צמתים:</u>

- BFS - הרצה $|V|$ פעמים - ללא משקלות - $O(V(V+E))$
- Dijkstra - הרצה $|V|$ פעמים - משקלות חיוביים - $O(VE\log V)$
- Belman Ford - הרצה $|V|$ פעמים - משקלות כלליים - $O(V^2E)$
- Floyed-Warshall - משקלות כלליים - $O(V^3)$

* <u>חישוב מרחק קצר ביותר מכל הקודקודים למקור יחיד:</u>

- מכיון שדייקסטרה ובלמן פורד מחשבים את אותו הדבר רק בכיוון ההפוך כל שעלינו לעשות הוא פשוט להפוך את כל כיווני הקשתות בגרף ולהריץ את אחד האלגוריתמים מקודקוד s ונהפוך בחזרה את הקשתות בעץ שהתקבל.

בס"ד

אלגוריתמים - סיכום

רשתות זרימה

* רשת זרימה - הגדרה: גרף מכוון $G(V,E)$ כאשר לכל קשת יש קיבולת חיובי: $c(u,v) > 0$
הקיבולת זה הזרימה המקסימלית שאפשר להעביר דרך קשת זו.
- עבור כל קודקוד $v \in V$ קיים מסלול שמתחיל ב s (צומת המקור) ועובר ב v ומגיע עד t (צומת הבור).
- קשת שלא קיימת ב E הקיבול שלה הוא 0.

* זרימה ברשת - הגדרה: תהי G רשת זרימה עם פונק' קיבולת c, זרימה flow ברשת G היא פונק' $f: V*V \rightarrow R$ המקיימת את התכונות:

1. אילוץ קיבולת, לכל 2 צמתים $f(u,v) \le \overset{\text{הזרימה חסומה ע"י הקיבולת}}{c(u,v)}$
2. סימטריה נגדית, לכל 2 צמתים $f(u,v) = -f(v,u)$
3. שימור הזרימה, לכל צומת u פרט למקור ולבור $\sum_{v \in V} f(u,v) = 0$ — לא נוספת/נעלמת זרימה בדרך - הזרימות מתקזזות.

- $f(u,v)$ היא הזרימה נטו מ u ל v.
- הערך של הזרימה f מוגדר $|f| = \sum_{v \in V} f(s,v)$ כלומר סך הזרימה נטו מהמקור.

המטרה: למצוא זרימה בעלת ערך מקסימלי מ s ל t.

° תכונת שימור הזרימה: הזרימה נטו חיובית הנכנסת לקודקוד v שווה לזרימה נטו חיובית היוצאת מהקודקוד v (פרט ל s ו t).

* הרשת השיורית - בהינתן רשת זרימה וזרימה הרשת השיורית מורכבת מקשתות שיכולות להכיל זרימה נטו נוספת, כלומר הכמות הנוספת של זרימה נטו שניתן לדחוף מ u ל v מבלי לעבור את הקיבולת $c(u,v)$. $\overset{\text{שיורי}}{c_f(u,v)} = \overset{\text{קיבול}}{c(u,v)} - \overset{\text{זרימה נטו}}{f(u,v)}$

* מסלול שיפור - הגדרה: בהינתן רשת זרימה וזרימה. מסלול שיפור p הוא מסלול פשוט מ s ל t ברשת השיורית G_f, דרך כל קשת (u,v). על מסלול שיפור ניתן להעביר מ u ל v זרימה נטו חיובית נוספת מסוימת מבלי להפר את אילוץ הקיבול על קשת זו.

* קיבול שיורי - הגדרה: כמות מקסימלית של זרימה נטו שניתן להעביר דרך הקשתות של מסלול שיפור p. נתון ע"י: $c_f(p) = \min\{c_f(u,v) \mid (u,v) \text{ נמצאת על } p\}$

* חתך ברשת זרימה - הגדרה: חתך ברשת זרימה הוא חלוקה של V (הקודקודים) ל 2 קבוצות S ו $T = V-S$. הזרימה נטו דרך החתך $f(S,T)$ הקיבול של החתך $c(S,T)$
חתך מינימלי של רשת הוא חתך שקיבולו מינימלי מבין החתכים.
- הזרימה נטו דרך חתך = הזרימה נטו ברשת.

מסקנה: ערכה של זרימה f כלשהי ברשת זרימה G חסום מלמעלה ע"י קיבולו של חתך כלשהו ב G.
- מחתך מסלולים את הגרף $\overset{c_f}{G_f}$ ל 2 כאשר המקור בקודקוד אחד והבור בשני והחתך עובר דרך הקשתות "רוויות".

ספ

סיכום אלגוריתמים

רשתות זרימה

- למה 1: תהי $G=(V,E)$ רשת זרימה ותהי f זרימה ב G, אזי עבור $X\subseteq V$ מתקיים $f(X,X)=0$
- עבור $X,Y\subseteq V$ מתקיים $f(X,Y) = -f(Y,X)$
  עבור $X,Y,Z\subseteq V$ כאשר $X\cap Y=\phi$ מתקיים: $f(X\cup Y,Z)=f(X,Z)+f(Y,Z)$
  וכן: $f(Z,X\cup Y)=f(Z,X)+f(Z,Y)$
- למה: ברשת זרימה, הזרימה נטו שיוצאת מצומת המקור שווה לזרימה נטו שנכנסת לקודקוד הבור.
- למה 2: תהי $G=(V,E)$ רשת זרימה עם מקור s ובור t, ותהי f זרימה ב G.
  תהי G_f רשת השיוריות של G המושרית ע"י f, ותהי f' זרימה ב G_f.
  אזי סכום הזרימות $f+f'$ הוא זרימה ב G שערכה $|f+f'|=|f|+|f'|$
- למה 3: תהי $G=(V,E)$ רשת זרימה, תהי f זרימה ב G ויהי p מסלול שיפור ב G_f
  נגדיר פונק' $f_p: V\times V\to R$ ע"י:

$$f_p(u,v) = \begin{cases} C_f(p) & \text{(u,v) נמצאת על p אם} \\ -C_f(p) & \text{(v,u) נמצאת על p אם} \\ 0 & \text{אחרת} \end{cases}$$

  אזי f_p היא זרימה ב G_f שערכה $|f_p| = C_f(p) > 0$
- למה 5: תהי f זרימה ברשת זרימה G עם מקור s ובור t ויהי (S,T) חתך של G אזי הזרימה נטו דרך (S,T) הוא $|f|=f(S,T)$, כלומר הזרימה נטו דרך החתך = זרימה נטו ברשת.

* משפט: זרימה מקסימלית - חתך מינימלי

הגדרה: אם f היא זרימה ברשת זרימה $G=(V,E)$ עם מקור ובור, אזי התנאים הבאים שקולים:

1. f היא זרימה מקסימלית ב G.
2. הרשת השיורית G_f אינה מכילה שום מסלול שיפור.
3. קיים חתך שעבורו $\overset{\text{זרימה נטו ברשת}}{|f|} = \overset{\text{קיבולת החתך}}{C(S,T)}$

- למה 8: אם מריצים את האלגוריתם של אדמונדס קארפ על רשת זרימה $G=(V,E)$ עם מקור s ובור t, אזי עבור כל הקודקודים מרחק המסלול הקצר ביותר ברשת השיורית G_f גדל מונוטונית עם כל שיפור של הזרימה.
- משפט 9: אם מריצים את אלגוריתם אדמונדס קארפ על רשת זרימה עם מקור s ובור t, אזי המס' הכולל של שיפורי זרימה שמבצע האלגוריתם הוא לכל היותר $O(VE)$.
- חוק שימור הזרימה: סך הזרימה הנכנסת לקודקוד = סך הזרימה היוצאת מהקודקוד.

בס״ד

אלגוריתמים - סיכום

רשתות זרימה

* **אלגוריתמים למציאת זרימה מקסימלית:**

- **שיטת פורד פולקרסון** - שיטה איטרטיבית, מתחילים עם זרימה התחלתית שערכה 0. בכל איטרציה מגדילים את ערך הזרימה ע"י מציאת "מסלול שיפור" ולאחר מכן שיפור הזרימה דרך מסלול זה. חוזרים על התהליך עד שלא ניתן למצוא עוד מסלולי שיפור. הזרימה המתקבלת בסיום התהליך הינה מקסימלית.

זמן הריצה של האלגוריתם תלוי באופן שבו נקבע מסלול השיפור, אם הוא נבחר בצורה גרועה הוא עלול שלא להתכנס כלל. אם הוא נבחר באמצעות BFS הוא רץ בזמן פולינומיאלי.

- **אלגוריתם של אדמונדס-קארפ** - שיפור לאלגוריתם של פורד פולקרסון, אם מסלול השיפור הוא מסלול קצר ביותר מs לt ברשת השיורית כאשר אורך של כל קשת הוא יחידה אחת. כלו' נממש את השיפור p באמצעות אלגוריתם חיפוש לרוחב BFS. זמן ריצה $O(VE^2)$.

השלבים: 1) התחל מזרימה ריקה $f=0$.

2) באיטרציה: כל עוד קיים מסלול שיפור מs לt ברשת השיורית - הגבר זרימה דרך מסלול השיפור הקצר ביותר. ניתן להוכיח את נכונות האלגוריתם לפי משפט חתך מינימלי-זרימה מקסימלית. הסיבוכיות ריצה נובעת מכך שמס' הפעמים שקשת עולה להיות צוואר בקבוק של מסלול שיפור הוא לכל היותר $\frac{|V|}{2}$.

* **הרחבה**: כאשר רוצים לבנות רשת עם מקורות ובורות מרובים, נוסיף 2 צמתים חדשים מקור על ובור על, את מקור העל מחברים לכל המקורות עם קשתות בעלות קיבולת אינסוף, ואת כל הבורות מחברים לבור העל עם קשתות בעלות קיבולת של אינסוף.

* **הרחבה**: כאשר רוצים לבנות שידוך בגרף דו צדדי נבנה אלגוריתם למציאת שידוך מקסימום שזמן ריצתו $O(VE)$, הרעיון: נגדיר גרף מכוון כך שהקיבול של כל הקשתות בגרף הוא 1.

- **חתך מינימלי** - מחלק את הגרף ל2 חלקים חלק עם קודקוד המקור וחלק שני עם קודקוד היעד חיב לעבור דרך קשתות רוויות כאשר $f=c$, חתך מינימלי = זרימה מקסימלית.

- כמות הקשתות ברשת השיורית ≥ כמות הקשתות ברשת הזרימה (לא יכולה להיות קטנה יותר!!!).

בס"ד

אלגוריתמים - סיכום

סיבוכיות NP ו NPC

* בעיות הכרעה - בעיה שקבוצת הפתרונות שלה היא {0,1}
* בעיית אופטימיזציה - בעיה שבה נדרש למצוא ערך מקסימלי או מינימלי של גודל מסוים.
* הכרעת שפה - אלגוריתם A מכריע שפה L אם A מקבל כל מחרוזת בשפה ודוחה כל מחרוזת שאינה בשפה.
- קבלה והכרעה בזמן פולינומיאלי - שפה מתקבלת בזמן פולינומיאלי אם עבור כל מחרוזת באורך n האלגוריתם מקבל את המחרוזת x בזמן $O(n^k)$ כאשר k הוא קבוע כלשהו.
- מחלקת סיבוכיות - קבוצת שפות שהשייכות אליה נקבעת לפי מדד סיבוכיות.
* אלגוריתם אימות - אלגוריתם המקבל מחרוזת קלט x ומחרוזת בינארית y הנקראת אישור.

○ מחלקת הסיבוכיות P - קבוצת השפות המוכרעות בזמן פולינומיאלי.

○ מחלקת הסיבוכיות NP - מחלקת השפות שניתן לאמת אותן באמצעות אלגוריתם זמן פולינומיאלי.

○ מחלקת הסיבוכיות Co-NP - קבוצת השפות L כך שהשפה המשלימה ל-L שייכת ל-NP.

○ מחלקת הסיבוכיות EXP - קבוצת כל השפות הניתנות להכרעה בזמן אקספוננציאלי.

○ מחלקת הסיבוכיות NPH - קבוצת כל השפות L המקיימות: לכל $L'\in NP$ מתקיים $L'\leq_p L$.
כלומר NP קשות - ניתן לעשות רדוקציה מכל בעיית NP אליה.

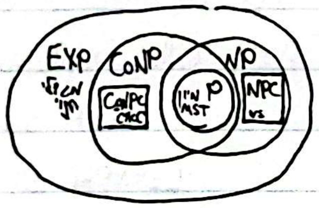
○ מחלקת הסיבוכיות NPC - קבוצת כל השפות L המקיימות: 1) $L\in NP$, 2) $L\in NPH$.

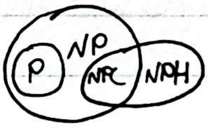
○ מחלקת הסיבוכיות CoNPC - המשלימה של NPC, כלומר כל השפות שהן: 1) $L\in CoNP$
2) לכל $L'\in CoNP$ מתקיים $L'\leq_p L$.

קבוצות הבעיות:

$P\subseteq NP\cap CoNP$

$NPC\subset NPH$





$P\subset NP$

$P\subset Exp$

$NP\subset Exp$

- P - קבוצת הבעיות שקיים להן אלגוריתם הכרעה פולינומי. P סגורה תחת איחוד, חיתוך, משלים.
- NP - קבוצת הבעיות שקיים להן אלגוריתם אימות פולינומי.
- Exp - קבוצת הבעיות שקיים להן אלגוריתם הכרעה אקספוננציאלי.

○ אם $L_1\in NPH$ וגם $L_2\in P$ וגם $L_1\leq_p L_2$ אזי $P=NP$

* $CNF\text{-}TAUT\in P$ (לפי משפט: נוסחה ב CNF היא טאוט' אמ"מ בכל פסוקית קיים משתנה ושלילתו).
* $DNF\text{-}COUNTR\in P$ (לפי משפט: נוסחה ב DNF סתירה אמ"מ בכל פסוקית יש משתנה ושלילתו).

בס"ד

אלגוריתמים - סיכום

NP, NPC ורדוקציות:

* רדוקציה - הגדרה: שפה L_1 ניתנת לרדוקציה בזמן פולינומיאלי לשפה L_2, נסמן $L_1 \le_p L_2$ אם קיימת פונקציה f הניתנת לחישוב בזמן פולינומיאלי כך שעבור כל $x \in \{0,1\}^*$ מתקיים $x \in L_1 \Leftrightarrow f(x) \in L_2$

הגדרה אינטואיטיבית: בעיה Q ניתנת לרדוקציה לבעיה Q' אם ניתן "לנסח מחדש" בקלות כל מופע של Q למופע של Q' שפתרונו מספק פתרון למופע של Q.

אופן הרדוקציה: many-1. המשמעות: השפה L_2 קשה לפחות כמו השפה L_1.

מתקיים: $x \in A \Leftrightarrow f(x) \in B$

$x \notin A \Leftrightarrow f(x) \notin B$

למה 1: אם מתקיים $L_1 \le_p L_2$ אזי $L_2 \in P$ גורר $L_1 \in P$.

למה 2: אם מתקיים $L_1 \le_p L_2$ אזי $L_2 \in NP$ גורר $L_1 \in NP$.

משמעות אינטואיטיבית לרדוקציה: אם קיים אלגוריתם פולינומיאלי המכריע את L_2, קיים גם אלגוריתם פולינומיאלי המכריע את L_1.

* משפט קוק לוין - $SAT \in NPC$

טענות נוספות שחשוב לזכור!

- $TAUT \in CoNPC$, המשלים של TAUT הינו NPC.
- בעיות CLIQUE וקבוצות בלתי תלויות (IS) שייכות NP וגם CoNP.
- המחלקה P סגורה למשלים, חיתוך, איחוד, שרשור, כוכבית.
- מחלקות NP וגם CoNP סגורות לאיחוד וחיתוך.
- כל שפה ששייכת ל P (מוכרעת בזמן פולינומי) בוודאי שייכת ל NP (ניתנת לאימות בזמן פולינומי).
- אם $L_1 \le_p L_2$ וגם $L_2 \in P$ אזי $L_1 \in P$.
- אם $L_1 \le_p L_2$ וגם $L_1 \in NPC$ אזי $L_2 \in NPH$.
- אם $L_1 \le_p L_2$ וגם $L_2 \in NP$ אזי $L_1 \in NP$
- אם $L_1 \le_p L_2$ וגם $L_2 \in NPC$ אזי $L_1 \in NP$
- אם $L_1 \le_p L_2$ וגם $L_2 \in CoNPC$ אזי $L_1 \in CoNP$
- אם $L_1 \in CoNP$ ו $L_2 \in CoNPC$ אזי $L_1 \le_p L_2$
- אם $L \in CoNP$ וגם $L \in CoNPH$ אזי $L \in CoNPC$
- אם $L_1 \in P$, $L_2 \in NPC$ אזי $L_1 \cup L_2 \in NPC$
- אם $L_1, L_2 \in NP$ אזי $L_1 \cup L_2 \in NP$

בס״ד

אלגוריתמים - סיכום

אלגוריתמי קרוב

* אלגוריתם קרוב - אלגוריתם המחזיר פתרונות כמעט אופטימלים.
(ישנן בעיות שניתן למצוא עבורן אלגוריתמים פולינומיאליים המספקים פתרונות כמעט אופטימלים.)

- A אלגוריתם קרוב עבור בעיית אופטימיזציה שבה לכל פתרון יש עלות חיובית,
c - עלות של הפתרון שמפיק האלג' A , c^* העלות של הפתרון האופטימלי לבעיה.
אומרים שאלג' קרוב A הינו בעל חסם יחס של $p(n)$ אם עבור כל קלט בגודל n:
היחס בין c ל-c^* קטן שווה ל $p(n)$ *חסם יחס: $\max\left(\frac{c}{c^*}, \frac{c^*}{c}\right) \le p(n)$
- אלגוריתם אופטימלי הוא בעל חסם יחס של 1
- אלגוריתם קרוב בעל חסם יחס גדול עלול להחזיר פתרון גרוע.

* השגיאה היחסית של אלג' קרוב: $\left|\frac{c-c^*}{c^*}\right|$ * חסם שגיאה יחסית של ε: $\left|\frac{c-c^*}{c^*}\right| \le \varepsilon(n)$

- ראינו שקיימות בעיות NPC, זה אומר שלא מוכר עבורן פתרון בזמן פולינומיאלי, וע״כ כדי להתמודד עם אותן בעיות אנחנו עושים קרובים.

* יחס הקרוב של האלגוריתם הוא: *בבעיות מינימיזציה c/c^* , *בבעיות מקסימיזציה c^*/c
- נרצה לחסום את היחס, כאשר יחס קרוב 1 משמעות $c=c^*$, כלומר האלגוריתם אופטימלי.
בכל מצב אחר, יחס הקרוב יהיה גדול מ 1. ככל שיחס הקרוב גדול יותר, כך הקרוב גרוע יותר.
- בבעי״ת Set-Cover לדוג' ככל שהקלט גדל יותר, כך יחס הקרוב גדל יותר.
- יתכן מצב שבו יחס הקרוב הוא קבוע ולא תלוי בגודל הקלט כמו ב VC, Bin-Packing.
- מוטיבציה: אלגוריתם קרוב בודק עד כמה הפתרון שלנו "גרוע" ביחס לפתרון האופטימלי.
כאשר $p(n)$ הוא החסם העליון לכמה הפתרון לא טוב, הוא לא יכול להיות קטן מ 1 כי אז זה אומר שהפתרון שמצאנו טוב יותר מהפתרון האופטימלי - לא יכול להיות!
וגם הוא לא יכול להיות שווה ל 1 כי אז זה אומר שפתרנו בזמן פולינומי בעיה עם פתרון אקספוננציאלי.

* $p(n)$ - מבטא את חסם היחס, כלומר פי כמה הפתרון שלנו רע יותר מהפתרון האופטימלי.
הגדרה: $\max\left(\frac{c}{c^*}, \frac{c^*}{c}\right) \le p(n)$

- חסם יחס כמובן שלא יכול להיות קטן מ 1 כי המשמעות של זה הוא שמצאנו פתרון יותר טוב מהפתרון האופטימלי מה שלא יכול להיות, כי אם היינו מוצאים פתרון טוב יותר מהאופטימלי הוא בעצמו היה הפתרון האופטימלי.

בס"ד

אלגוריתמים - סיכום

אלגוריתמי קירוב

* $\varepsilon(n)$ מבטא את חסם השגיאה היחסית. יכול להיות לנו סדר גודל של הטעות עצמה.
כלומר, עד כמה אנחנו רחוקים מהפתרון האופטימלי.
הגדרה: $\frac{|c^*-c|}{c^*} \le \varepsilon(n)$

- בבעית מקסימיזציה הפתרון האופטימלי הוא הכי גדול ולכן השגיאה היא פ' $\frac{1}{\rho}$
כלומר הוא במקרה יהיה יותר גרוע מהפתרון האופטימלי.
- בבעית מינימיזציה הפתרון האופטימלי הוא מינימלי ולכן השגיאה היא פ' ρ

אלגוריתמים - סיכום

הסבר בקצרה על בעיות אלגוריתמיות מוכרות שהכרנו בקורס.

NPC * בעיית CIRCUIT-SAT - הבעיה הראשונה שהוכח שהיא NPC.
תאור הבעיה - בהינתן מעגל לוגי בוליאני עם קלטים ויציאה אחת, צריך לקבוע האם קיימת השמה שמביאה לנו ערך True. לדוג: XORy (אם נציב (0,1) נקבל T).

NPC * בעיית SAT - בעיית סיפוק בוליאנית שבה אנחנו בודקים האם קיימת השמה לערכים הבוליאניים (0 או 1) של משתנים כך שנוסחה לוגית נתונה תהיה אמת.
כלומר האם בעיה מסוימת ניתנת לסיפוק. זו הבעיה הראשונה שהוכחה כ NPC (משפט קוק לוין).

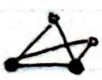
NPC * בעיית NOT-TAUT - לא טאוטולוגיה, בהינתן נוסחה בוליאנית ϕ האם קיימת לפחות השמה אחת של False. אם קיימת השמה של false - התשובה היא כן, כלומר הנוסחה לא טאוט.
אם כל השמה אפשרית תוביל את ϕ ל true - התשובה היא לא, כלומר הנוסחה טאוט.

NPC * בעיית 3-CNF-SAT - גרסה של SAT שבה הנוסחה מיוצגת בצורת 3-CNF, כלומר יש 3 ליטרלים בכל פסוקית. לדוגמה: $(x_2 \vee \neg x_3 \vee \neg x_5) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_4 \vee x_5)$
דוגמה לנוסחה לא ספיקה $(x_3 \vee x_2 \vee \neg x_1) \wedge (x_3 \vee \neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2 \vee x_1) \wedge (\neg x_1 \vee x_2 \vee x_3)$

NPC * בעיית HAM-CYCLE - בעיית מעגל המילטון בעיה בגרפים בה אנחנו מחפשים מעגל המילטוני,
כלומר מעגל שעובר בכל הצמתים של הגרף בדיוק פעם אחת וחוזר לצומת ההתחלה. 
בעיה זו היא NPC.

NPC * בעיית הקליק היא בעיה בגרפים שבה אנחנו מנסים למצוא קבוצה מקסימלית CLIQUE - 
של קודקודים בגרף, כך שכל קודקוד מחובר לכל קודקוד. (כלומר קודקודים בקבוצה מחוברים בקשת).
בעיה זו היא NPC

NPC * בעיית ה TSP - בעיית הסוכן הנוסע, הבעיה מתארת סוכן שמבקר בכמה ערים ומחפש את המסלול הקצר ביותר שמבקר בכל עיר פעם אחת וחוזר לנקודת המוצא.
מדובר בגרף שלם, לא ידוע על אלגוריתם פולינומי, שפותר את הבעיה עבור k מסוים. NPC.

NPC * בעיית VC - בעיית כיסוי הקודקודים היא בעיה בגרפים שבה אנחנו מחפשים קבוצה

מינימלית של קודקודים שמכסה את כל הקשתות בגרף, בעיה זו היא NPC

בעיה פולינום
- * בעיית EC - בעיית כיסוי הצלעות, כלומר קבוצה מינימלית של קשתות שמכסות את כל הקודקודים.

NPC * בעיית SUBSET SUM - בבעיית סכום תת קבוצה אנחנו מקבלים קבוצה של מספרים וקבוע t ואנחנו צריכים לבדוק האם קיימת תת קבוצה של מס׳ שסכומם שווה ל-t.

NPC * בעיית PAR - בבעיית ה partition - החלוקה, אנחנו מקבלים קבוצה של מספרים שלמים חיוביים וצריך לבדוק האם ניתן לחלק את הקבוצה ל2 קבוצות שוות בסכומן.

בס״ד

אלגוריתמים - סיכום בעיות מוכרות בקורס

NPC - * בעיית Knaspsack - בעיית תרמיל הגב. בבעיה זו יש לנו תרמיל עם קיבולת מוגבלת וקבוצה של חפצים שלכל אחד מהם משקל וערך. המטרה היא לבחור קבוצת חפצים כך שהערך הכולל יהיה מקסימלי, מבלי לעבור את מגבלת המשקל של התרמיל.

- ניתן לפתור בעיה זו באמצעות אלגוריתם דינאמי (ניתן גם בחמדני אבל רק בגרסת השברים).

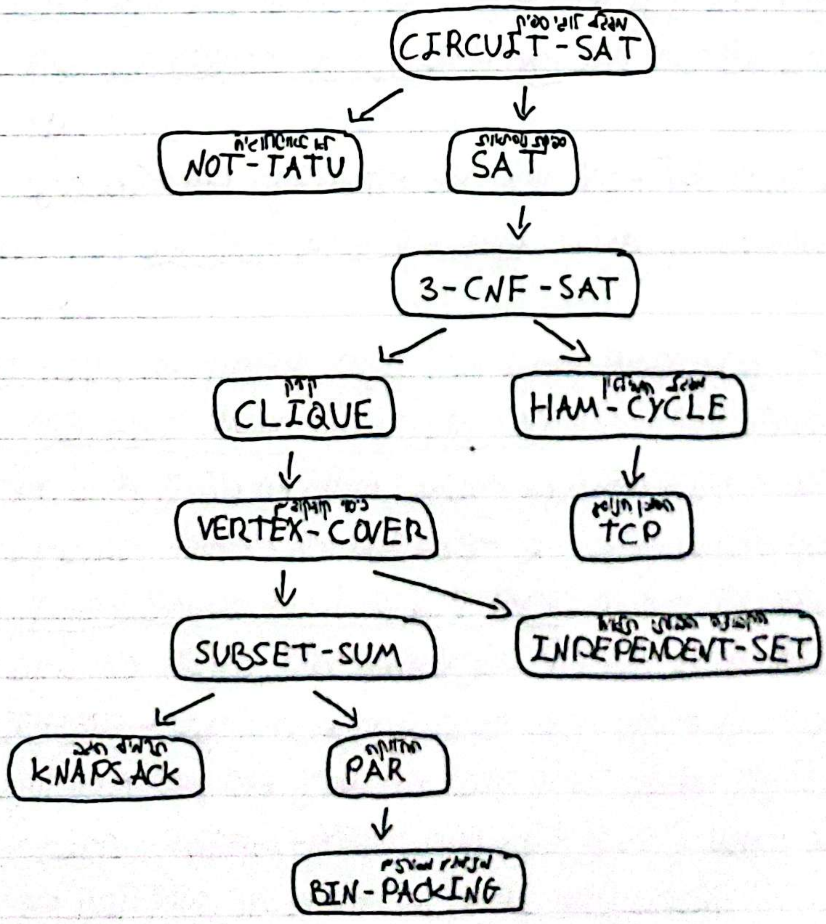
NPC * בעיית Bin-packing - בעיית אופטימיזציה שבה בהינתן קבוצת פריטים שגודלם $0<s_i\leq 1$ יש לשבץ אותם למס׳ מינימלי של מארזים (BINS) כאשר גודל כל מארז הוא 1.
בעיית האופטימיזציה היא NPH, בבעיית ההכרעה (ניתן עד k מארזים). זו בעיית NPC.

- חשוב לזכור!

כל בעיות ה NPC באופטימיזציה שלהן הן NPH (אין לאמת בעיית אופטימיזציה!) ואילו בעיות ההכרעה שלהן הן NP ולכן הן גם NPC (בהרצ׳ יש חסם ניתן לאמת את הפתרון, כי אם בעיות אלו הן בעיות הכרעה ולא אופטימיזציה).

NPC * בעיית IS - הקבוצה הבלתי תלויה הקודקודים כך שאין קשת המחברת בין שני קודקודים בגרף

מפת הרדוקציות



בס״ד

סיכום אלגוריתמים על גרפים

אלגוריתמים כלליים:

שם האלגוריתם	מטרה/תוצאה	פעולת האלגוריתם	זמן (מבנה נתונים).
חיפוש לרוחב **BFS** (גרף לא ממושקל, מקור יחיד)	אלגוריתם חיפוש למציאת עץ רוחב לפי מס׳ צלעות מסלול קצר מבחינת צלעות	מכניסים את השורש לתור, מוציאים מראש התור קודקוד, ומכניסים את כל שכניו לסוף התור תוך כדי עדכון המרחק מהשורש.	$\Theta(V+E)$ מימוש באמצעות תור רשימת שכנויות/מטריצת שכנויות
חיפוש לעומק **DFS** (גרף לא ממושקל)	יער עצי עומק סקירה לעומק ומציאת מעגלים (אם יש קשת אחורית) מציאת רכיבי קשירות	מתחילים מקודקוד מסוים ומעדכן לכל בן את זמן הגילוי שלו, ובסוף את זמן הסיום f(v).	$O(V+E)$ מימוש באמצעות מחסנית רשימת שכנויות/מטריצת שכנויות
גרף דו צדדי **דו״צ**	בדיקה האם גרף נתון הוא דו״צ	מריצים BFS על קודקוד מקור. בודקים אם יש קשת בין קודקוד במרחק זוגי לזוגי אחר: או אי זוגי לאי זוגי	$O(V+E)$ DFS / BFS
מיון **טופולוגי** מתבצע על DAG - גמ״ל גרף קשיר ללא מעגלים	סידור לינארי של הקודקודים לפי זמני סיום	מפעילים DFS, ממיינים לפי סדר יורד של זמני סיום, ומסדרים קשתות לפי הגרף המקורי	$\Theta(V+E)$ משתמשים ב DFS
מציאת רכיבים קשירים **CSS** (גרף לא ממושקל)	חיפוש רכיבי קשירות בגרף מכוון/לא מכוון	מריצים DFS, ממיינים את הקשתות לפי סדר יורד של זמני סיום, מריצים DFS על G^T	$\Theta(V+E)$

אלגוריתמים לעצים פורשים מינימלים:

שם האלגוריתם	מטרה/תוצאה	פעולת האלגוריתם	זמן (מבנה נתונים)
kruskal (מבוסס מיון)	עץ פורש מינימלי (סכום משקל קשתות מינימלי בגרף)	מיון הקשתות בסדר עולה, הוספת כל קשת שלא סוגרת מעגל לתוך העץ.	$\Theta(E\log E)$ מבנה נתונים - קבוצות זרות
Prim (מבוסס על ״צביעת החתך״)	עץ פורש מינימלי (בעל משקל קשתות מינימלי בגרף)	מתחילים בקודקוד לפי בחירה, בכל פעם מוסיפים את הקשת הקלה ביותר המחוברת לאחד הקודקודים (ולא סוגרת מעגל!).	$O(E\log V)$ בערימה (בגרף דליל) $O(V^2)$ במערך (בגרף צפוף $E=\Theta(V^2)$)

אלגוריתמים למציאת מסלולים קצרים ביותר:

שם האלגוריתם	מטרה/תוצאה	פעולת האלגוריתם	זמן (מבנה נתונים)
Dijkstra בגרף ממושקל אי שלילי!	מציאת מסלול קצר ביותר מקודקוד יחיד	איטרטיבי, בכל איטרציה מוציאים את הקודקוד המינימלי ומבצעים Relax לכל השכנים עד שהערימה ריקה	$O(E\log V)$ ערימת מינימום
Bellman-Ford בגרף כללי ממושקל	מציאת מסלול קצר ביותר ממקור יחיד	מבצעים V-1 הקלות על כל הקשתות, ואחת נוספת לראות אם יש שינוי (מעגל שלילי).	$O(VE)$ טוב עם משקלים כללים איתור מעגל עם סכום שלילי.
Floyd-Warshall גרף ממושקל - אלגוריתם דינמי	מציאת מסלולים קצרים ביותר מכל קודקוד לכל קודקוד	בכל איטרציה בודקים אם יש שינוי במרחק הקודקודים בהתחשב בקודקוד נוסף החל מ-1 ועד V.	$O(V^3)$ מטריצת מרחקים $d^{(k)}$ מטריצת אבות $\pi^{(k)}$

דברים שחשוב לזכור באלגוריתמים

* אם נתון לנו שאלגוריתם מסוים A פותר את הבעיה L בזמן X, אז זמן הריצה של האלגוריתם שמחשב את הבעיה המשלימה של L הוא גם X, כיוון שניקח את האלגוריתם A וכל קלט שבו הוא מחזיר true נחזיר false ולהפך.
לכן זמן הריצה ישאר X, משום שלא עשינו שינוי בA אלא רק הפכנו את התשובה.
* אם נמצא אלגוריתם פולינומי לבעיית הCoNPC-ית, אז זה יוכיח ש P=coNP , P=NP.
* הבעיה האלגוריתמית: נתונה נוסחה בוליאנית C בצורת CNF. האם הנוסחה ספיקה?
בעיה זו היא הבעיה המוכרת SAT שהיא הבעיה הNPC הראשונה שהוכחה במשפט קוק-לוין.
* הבעיה האלגוריתמית: נתונה נוסחה בוליאנית C, האם הנוסחה היא טאוטולוגיה?
ניתן לומר על בעיה זו שהיא CoNPC
* נתונות שתי בעיות L_1,L_2 ויש שתיהן שייכות לNPC אז קיימת רדוקציה פולינומית מL_1 לL_2 וגם מL_2 לL_1.
* בעת רדוקציה ממעגל המילטון לסוכן הנוסע - מס' הצלעות בגרף שנקבל יכול להיות גדול מאוד מהצלעות בגרף המקורי!
* הבעיה האלגוריתמית: נתון גרף לא מכוון G=(V,E) ומס' k. האם קיים בגרף מעגל כלשהו?
בעיה זו היא פולינומית וניתן לפתור אותה בזמן O(V+E)
* הבעיה האלגוריתמית: נתון גרף G דו חלקי ומס' טבעי k. האם קיים שידוך בG שגודלו גדול/שווה ≥ k ?
בעיה זו פולינומית כי ניתן למצוא את התשובה באמצעות רדוקציה לבעיית הזרימה המקסימלית.
* נתונות שתי בעיות L_1,L_2, ידוע כי $L_1 \le_p L_2$ - איזה משפט יהיה נכון?
אם $L_2 \in NPC$ אז גם $L_1 \in NPC$.
* הבעיה האלגוריתמית: נתונה נוסחה בוליאנית C בצורה DNF. האם הנוסחה ספיקה?
ניתן לומר על בעיה זו שהיא פולינומית.
* אם נתון אלגוריתם שמכריע את בעיה L בזמן $O(n^k)$ ונתון אלגוריתם B שמכריע את בעיה M בזמן $O(m^q)$, אז זמן הריצה של הרדוקציה הפולינומית מבעיה A לבעיה B הוא $O(n^k)$ כיוון שזה הזמן הארוך ביותר שניתן לפתור את A ולהעביר למופע מתאים של B.
ויש אפשר לעשות את המעבר ולפתור בפחות זמן אך עדיין יהיה זמן ריצה של $O(n^k)$
* בעיית ספיקות המשפט "קוק-לוין" היא SAT והיא שייכת לNPC
* כדי להוכיח ש$P \neq NP$ צריך להראות שקיימת לפחות בעיית NP אחת שאינה ניתנת לפתרון בזמן פולינומיאלי. אפשר להראות את זה ע"י הוכחה פורמלית של חסם תחתון או שנראה שאחת מבעיות הNPC לא ניתנת לפתרון פולינומי.
* כל מה שגורם לכך שבעיה ב NPC תפתר עם אלגוריתם הרצה פולינומי יגרום ש$P=NP$. כולל:
  - ○ רדוקציה מבעיה ב NPC אל P.
  - ○ אם נוכיח שבעיה ב NPH פולינומית.
  - ○ רדוקציה מבעיה ב CoNPC אל P.
  - ○ מס' אלגוריתמי קירובים עבור חסם יחס נמוך מספיק (לדוג' חסם יחס קבוע ל TCP המטרית, חסם יחס נמוך מ 1.5 ל bin packing וכו'..).
- כדי להוכיח ש $P \neq NP$ צריך למצוא חסם תחתון אקספוננציאלי לבעיה ב NPC או ב CoNPC.

נעמה

סיכום שאלות אמריקאיות ממבחנים באלגוריתמים

* בבעיית פריטת המטבעות המוכללת: נתון סכום שלם של A ומטבעות עם הערכים $c^0, c^1, \ldots, c^k$ ורוצים לפרוט את הסכום A תוך שימוש במינימום מטבעות. $c>1$, $k\ge1$ ערכים שלמים.

- קיים אלגוריתם שפותר את הבעיה בזמן $O(k)$.

* נתונה הנוסחה הרקורסיבית הבאה: היכן משתמשים בה?

$$d_{i,j}^{(k)} = \begin{cases} w(v_i,v_j) & k=0 \\ \min\left(d_{i,j}^{(k-1)},\ d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right) & k>0 \end{cases}$$

- זו הנוסחה הרקורסיבית בה משתמשים עבור האלגוריתם של פלויד וורשל, שמטרתו חישוב המסלול הקצר ביותר מכל מקור לכל יעד.

* נתון גרף מכוון עם קשתות שמשקלן 1,2,3 בלבד. מעוניינים למצוא מסלולים קצרים ביותר מקודקוד מוצא s לכל היעדים. איזה מהטענות הבאות נכונה?

- ניתן לפתור את הבעיה האלגוריתמית תוך שימוש באלגוריתם המבוסס על BFS, בזמן $\Theta(|V|+|E|)$ ע"י פיצול קשתות "יקרות" ל2 או 3 קשתות עם משקל 1.

* הכרנו בעיות שונות ב NPC. בין אילו 2 בעיות ניתן להגדיר את הרדוקציה $x \le f(x)$?

- רדוקציה מ 3-SAT ל SAT.

* נתונה הבעיה האלגוריתמית הבאה: נתון גרף $G=(V,E)$ עם משקלות על הקודקודים וקבוע k. האם קיים בגרף G מסלול פשוט שסכום המשקלות שלו הוא k?

- הבעיה היא NPC, ניתן להראות זאת באמצעות רדוקציה מבעיית הסכום החלקי.

* הגדירו מהו חתך <u>מינימלי</u> ברשת זרימה:

- זוהי חלוקת קודקודי הרשת לקבוצה ובה נמצא המקור ולקבוצה ובה נמצא הבור, כך שסכום קיבולות הקשתות החוצות את החתך בכיוון מ S ל T יהיה מינימלית.

* בהינתן קלט עבור בעיית השידוך המקסימלי בגרף דו"צ המירו את הקלט לרשת זרימה. מה המשמעות של הקיבולות של הקשתות שהוספו במהלך בניית רשת הזרימה?

- ערכן 1, בכדי להבטיח שלא יהיה קודקוד שיותאם יותר מפעם אחת.

* נתונה בעיית האופטימיזציה של Vertex Cover. מה ניתן לומר עליה?

- יש לה אלגוריתם קרוב עם יחס 2.

<u>חשוב לזכור!</u>

$\rho(n)$ - מבטא את <u>חסם היחס</u>, כלומר פי כמה הפתרון שמצאנו רע יותר מהפתרון האמיתי.

הגדרה: $\max\left(\frac{c}{c^*}, \frac{c^*}{c}\right) \le \rho(n)$

- חסם יחס כמובן שלא יכול להיות קטן מ 1 כי המשמעות של זה היא שמצאנו פתרון יותר טוב מהאופטימלי וזה לא יכול להיות, כי אם היינו מוצאים פתרון טוב יותר מהאופטימלי הוא היה האופטימלי בעצמו.

$\varepsilon(n)$ - מבטא את <u>חסם השגיאה היחסית</u>. יכול להבטא לנו סדר גודל של הטעות עצמה. כלומר עד כמה אנחנו רחוקים מהפתרון האופטימלי. חישוב השגיאה היחסית מחושב כך:

הגדרה: $\left|\frac{c^*-c}{c^*}\right| \le \varepsilon(n)$

- אם הפתרון רע בצורה מקורבת בלי קשר לגודל הקלט סימני היחסים יהיו לא ב n (רק ε ו ρ).

* מה המשמעות חסם יחס $\rho \ge 1$ עבור אלגוריתם קרוב לבעיית <u>מקסימיזציה</u>?

- האלגוריתם מחזיר פתרון שערכו לכל הפחות $1/\rho$ כפול ערך הפתרון האופטימלי.

בס״ד

המשך סיכום שאלות אמריקאיות ממבחנים באלגוריתמים

* בבעיות של תכנות דינמי, קיים תהליך של שחזור הפתרון האופטימלי, להלן משפטים נכונים לגבי שחזור הפתרון האופטימלי שחושב באמצעות תכנות דינמי:

- סדר הגודל של זמן הריצה של שחזור הפתרון של כפל מטריצות אם ידועה מטריצת עזר עבור ערכי k שונה מסדר הגודל של זמן הריצה של שחזור הפתרון של כפל מטריצות אם לא ידועה את מטריצת העזר.
- סדר הגודל של זמן הריצה של שחזור הפתרון של שיבוץ תמליל הכי משתלם אם ידועה מטריצת עזר עבור ההחלטה שהתקבלה בכל תת בעיה, זהה לסדר הגודל של זמן הריצה של שחזור הפתרון אם לא ידועה את מטריצת העזר.
- סדר הגודל של זמן הריצה של שחזור הפתרון של תת הסדרה המשותפת הארוכה ביותר אם ידועה מטריצת עזר b, יהיה זהה לסדר הגודל של זמן הריצה של שחזור הפתרון אם לא ידועה את מטריצת העזר.

* דברים שחשוב לשים לב בשאלות של הרצה וכו׳:

- ניתן להריץ את Dijkstra (אלגוריתם למציאת מסלולים קצרים ביותר בגרף מכוון) על גרף שאינו מכוון!
- כאשר נתונה לנו מטריצה שהתקבלה מסיום הריצה של פלויד וורשל ניתן להסיק:
  - ◦ אם יש ערכים שליליים באלכסון ⇐ יש בגרף מעגל שלילי.
  - ◦ אם העמודה של קודקוד מסוים כולה אינסוף* ⇐ דרגת הכניסה (מס׳ הקשתות הנכנסות) של קודקוד זה היא 0.
  - ◦ אם השורה של קודקוד מסוים מלאה באינסוף* ⇐ דרגת היציאה (מס׳ הקשתות היוצאות) היא 0.

* כמובן שחוץ מהאיבר שמייצג את הקודקוד עצמו!

- טיפ לזכור! שורות = דרגת יציאה, עמודות = דרגת כניסה.

* בהינתן 2 בעיות A,B קיימת רדוקציה פולינומית מ A ל B אם קיימת פונק׳ f הניתנת לחישוב בזמן פולינומי כך ש: $x \in A \Leftrightarrow f(x) \in B$. האם קיימות 2 בעיות ב NPC עבורן אותה פונק׳ f מתאימה גם לרדוקציה מ A ל B וגם לרדוקציה מ B ל A?

- כן, לדוגמא: הבעיות: Clique ו- Independence Set.

* נתון גרף $G=(V,E)$ שהוא גרף מכוון ללא מעגלים, עם משקלים חיוביים ושליליים על הקשתות. נתון קודקוד $v \in V$ ורוצים למצוא את המסלולים הקצרים ביותר מכל הקודקודים אל הקודקוד v. מה מהתשובות הבאות הוא האלגוריתם היעיל ביותר שיפתור את הבעיה?

- נחשב את G^T ונפעיל את האלגוריתם למציאת מסלולים קצרים ב DAG מנקודת מוצא v בגרף G^T. זמן הריצה $\Theta(|V|+|E|)$.

* מה מבצע אלגוריתם האימות של בעיית 3-CNF-SAT

- האלגוריתם מקבל נוסחת 3CNF והצבת ערכי אמת למשתנים, ובודק האם ערך הנוסחה נותן אמת עבור הצבה זו.

* אם יצליחו להוכיח חסם תחתון אקספוננציאלי לבעיה M השייכת ל NPH, המשמעות:

- ואם $\bar{M}$ שייכת ל NPC, המשמעות של זה היא כי $P \neq NP$.

(הסבר: בגלל שלM יש חסם תחתון אקספוננציאלי אז M בהכרח לא שייכת לP. מכיון שP סגורה למשלים אז בהכרח גם $\bar{M}$ לא שייכת לP. עכשיו ש$\bar{M}$ היא NPC ולא שייכת לP אזי שקיבלנו חסם תחתון לבעיה בNPC שזה אומר ש $P \neq NP$.)

תזכורות ומסקנות מהבחנים.

* בבעיית בחירת הפעילויות אופטימלי נמיין את הפעילויות לפי הזמני סיום מהמוקדם ביותר ובכל פעם נבחר את הפעילות שמסתיימת בזמן הכי מוקדם ובנוסף שאינה חופפת-מתנגשת עם פעילות אחרת-קודמת שבחרנו.

* באלגוריתם הדינמי שבחרנו-בבעיית כפל המטריצות זמן הריצה הנדרש לחישוב הערך של משבצת $[i,j]$ בטבלה m עבור תת בעיה הוא כמס' האפשרויות להצבת סוגריים חיצוניים בביטוי $A_i..A_j$

* בבעיית עץ פורש מינימלי הרעיון של הפתרון הוא למיין את כל הקשתות לפי המשקלים מהקטן לגדול, לאחר מכן נפעיל את האלגוריתם למציאת קבוצות זרות כך: נבדוק אם הקודקודים שייכים כבר לאותו עץ (כלומר יש לנו כבר מסלול אל הקודקוד הזה). אם לא נוסיף את הקשת לעץ.
לכן במידה ויש לנו 2 קשתות שיוצאות מאותו קודקוד הקשת שמשקלה קטן יותר היא תופיע בעץ פורש מינימלי. במקרה והגרף מכיל קשתות שוות במשקל יתכן ויהיו 2 עצים פורשים מינימלים.

* האלגוריתם של בלמן פורד פותר את הבעיה הכללית יותר של המסלולים הקצרים ביותר. מה שמיוחד באלגוריתם הזה שלא חלה עליו מגבלת ערכים שליליים, הוא יודע להתמודד עם ערכים שליליים. דבר נוסף, האלגוריתם בודק ומחזיר לנו תשובה בוליאנית האם יש לנו מעגל שלילי שמחזיר לנו בעצם תשובה לא מדויקת. מחזיר אם קיים מעגל שלילי בגרף.

* אלגוריתם פלויד-וורשל הוא אלגוריתם של תכנות דינאמי.

* אלגוריתם דייקסטרה - למציאת המסלולים הקצרים ביותר. האלגוריתם מאתחל את כל המרחקים בגרף ל∞ ואת קודקוד המקור ל0. לאחר מכן מבצע הקלה על הקשתות שיוצאות מקודקוד המקור ולוקח את הקשת המינימלית, האלגוריתם ממשיך לעבור על השכנים של הקשת שבחרנו ומבצע עליהם הקלה כאשר בכל פעם האלגוריתם יבחר לעבור דרך הקשת המינימלית ביותר בגרף שנמצאת בראש התור. לאחר שסיימנו לעבור על כל הקודקודים תור הקדימויות ריק וקיבלנו את גרף עץ המסלולים הקצרים ביותר מS לכל הקודקודים.

* נתון גרף $G=(V,E)$ לא מכוון קשיר עם משקלות שונות זו מזו, נתון T עץ פורש מינימלי של E ונתון חתך הטענות הבאות נכונות: * הקשת הקלה ביותר שחוצה את החתך שייכת לT, * יתכן שיותר מקשת אחת של T חוצה את החתך, * יתכן והקשת הכבדה ביותר שחוצה את החתך שייכת לT.

* אלגוריתם בלמן פורד - הסבר האלגוריתם: בשונה מאלגוריתם דייקסטרה שמבצע הקלה על כל השכנים אלגוריתם זה רץ על כל הקודקודים בסדר טופולוגי ומבצע הקלה, באופן של לולאה מקוננת כך:
- בתחילה האלגוריתם מאתחל את כל הקודקודים, ערכי d לאינסוף (חוץ מהמקור ל0) וערכי הπ לnull.
- חוזר $|V|-1$ פעמים על הפעולה (הלולאה): - מבצע הקלה על כל הקשתות בגרף.

במילים: אנו מבצעים איטרציות כמס' הקודקודים פחות 1, בכל איטרציה אנו עוברים על כל הקודקודים בגרף ועושים להם Relax (תזכורת: $d[v] > d[u] + w(u,v)$). כאשר בכל איטרציה אנו משתמשים בנתונים המעודכנים מאיטרציה קודמת. לאחר $|V|-1$ איטרציות אנחנו מובטחים שאין מעגל שלילי ע"י בדיקה שבה אם ניתן לבצע הקלה על כל אחת מהקשתות, אם כן יש מעגל שלילי בגרף, אחרת אין מעגל. (אם לפני שהגענו ל $|V|-1$ איטרציות לא נעשו שינויים בגרף או מבחינה טכנית יכולנו לעצור את ההרצה מאותו הרגע והעץ שהתקבל הוא עץ המסלולים הקצרים ביותר ובוודאי שאין לנו מעגל שלילי בגרף.)

* בבעיית מתן העודף בסך A ש"ח, תוך שימוש במס' מינימלי של מטבעות וכו' לפתרון בעיה זו קיים אלגוריתם של תכנות דינאמי וגם של הפרד ומשול וגם אלגוריתם חמדן לפתרון הבעיה.

* אלגוריתם פרים למציאת עץ פורש מינימלי - הסבר האלגוריתם: האלגוריתם מתחיל מקודקוד כלשהו מגדיר את ה d שלו להיות 0 ואת ההורה שלו להיות Null כאשר שאר הקודקודים מאותחלים $d=\infty$ ו π = null נשתמש בתור עדיפויות ונבצע הרצה מהקודקוד ההתחלתי כאשר בכל פעם נוציא את הקודקוד עם המשקל המינימלי. לאחר מכן נמשיך לבדוק את השכנים של אותו קודקוד כאשר בכל פעם נוציא מהערימה את הקודקוד המינימלי ונעדכן את המשקלים עד שנקבל עץ פורש מינימלי.

בס"ד

דברים שחשוב לזכור באלגוריתמים - מתוך מבחנים

* אם P=NP אז NP = CO-NP.
* אם $\overline{SAT} \in P$ אז TAUT∈P
* בהינתן סדרה של n מטריצות שונות ויש לכפלן, מה ניתן לומר על האלגוריתם הנלמד בהרצאה?
- הקלט הוא n+1 ערכים והאלגוריתם יחשב את מס' המכפלות הסקלריות המינימלי הנדרש לביצוע מכפלת המטריצות.

* בהינתן רשת זרימה ובה זרימה מקסימלית:
-

* נבצע רדוקציה מ VC המוגדרת כקלט גרף G ומס' טבעי k ל SUS המוגדרת כקלט קבוצת מספרים S ומס' יעד t. אילו נכונות:
- כמות המס' שיש בS במהלך הרדוקציה תהיה שווה ל |E|+|V|.
- בגרף לא ריק, חלק מהמספרים שיוצרו בS יהיו בדיוק חזקה של 2.
- t (ערך המטרה שיוצר) יהיה קטן ממש מסכום כל המספרים בקבוצה S.
* הבעיות שבהן מיון מוקדם עוזר לפתרון הבעיה:
- בבעיית בחירת מקסימום פעילויות
- בבעיית תזמון המשימות
- בבעיית תרמיל הגב בשברים (באמצעות האלגוריתם החמדן).
* נתונה נוסחת 3-CNF, הפעלנו עליה את הרדוקציה לבעיית הקליק, וקיבלנו ממנה גרף ובו 12 קודקודים וקליק מקסימלי בגודל 3. מה ניתן להסיק מכך?
- הנוסחה המקורית לא הייתה ספיקה.
* נתון גרף קשיר לא מכוון עם משקלים על הקשתות וידוע ש|V|=|E|. אילו מהשיטות הבאות נכונה לגבי האופן היעיל ביותר למציאת עץ פורש מינימלי לגרף זה.
- נפעיל DFS על הגרף, וכשנגיע לקשת אחורה, נסיר את הקשת היקרה ביותר מהמעגל שמצאנו.
* נתונה רשת זרימה, עם מקור s ויעד t עליה מריצים את האלגוריתם של אדמונדס-קארפ. מה יהיה תנאי העצירה של האלגוריתם?
- כאשר BFS לא ימצא מסלול שיפור ברשת השיורית.
* מהי סיבוכיות הזמן של אלגוריתם הקירוב שלמדנו עבור הסוכן הנוסע, עם חסם יחס 2 תחת הנחת אי שוויון המשולש? - תשובה: $\Theta(E\log V)$
* נתונה בעיה אשר פתרון של תת בעיה שלה נקבל בנוסחה הבאה:

$$A[i,j] = \max_{k=i..j-1}\{A[i,k] + A[k+1,j] + C[k]\}$$

כאשר המערך C הוא מערך מספרים נתון.
רוצים לפתור את הבעיה באמצעות תכנות דינמי. מה ניתן לומר על פתרון הבעיה?
- סדר מילוי הטבלה בגישת התכנון הדינמי יהיה כמו סדר מילוי הטבלה בבעיית כפל מטריצות.
* נתונה הבעיה האלגוריתמית הבאה:
נתונה נוסחה בוליאנית. האם כל ההשמות עבור המשתנים שלה נותנים ערך אמת.
- הבעיה היא CoNPC הבעיה המשלימה היא NPC. (הסבר: בעיה זו היא בעיית TAUT שהיא CoNPC, המשלימה היא SAT∈NPC).
* נתון גרף DAG ורוצים לבדוק האם קיים בו קודקוד שיש ממנו מסלול לכל יתר הקודקודים. איך ניתן לפתור בעיה זו בצורה היעילה ביותר?
- ניתן לפתור את הבעיה בזמן $O(|V|+|E|)$, באמצעות האלגוריתם של מציאת מסלולים קצרים בDAG.