תכנון וניתוח אלגוריתמים – הרצאה 5 (יום שני 28.11.11 שבוע 5)

אלגוריתמים משערכים (המשך)

בעיית k – מרכזים (K-Center):

נניח ונרצה להחליט על מוקדי שירות מתוך מרחב מסוים של מוקדים (לדוגמא מוקדי תחנות כיבוי עירוניות במרחב הערים). נרצה שבתקציב הנתון אף מוקד לא יהיה מרוחק מדי ממוקדי השירות (לא כל עיר תקבל מוקד תחנת כבאות).

קלט:

1. מוקדים $P=(p_1,\ldots,p_n)$
2. מספר k שלם וחיובי
3. מטריקה $d(p_i,p_j)$

פלט: קבוצת מוקדי שירות (מתוך קבוצת המוקדים) S כך שמתקיים: $S\subseteq P,|S|=k$.

המטרה: למזער את $\max_{p\in P}(d(p,S))$ כאשר $d(p,S)\overset{def}{=}\min(d(p,s):s\in S)$.

המטריקה יכולה להיות אוקלידית או נתונה לפי גרף.

אלגוריתם: נמקם את המרכזים אחד אחרי השני. נתחיל שרירותית:

$S:=\{p_1\}$

for i := 2 to k

$s_i=\arg\max_{p\in P}\{d(p,S)\}$ (נבחר נקודה שהמרחק ממנה ליתר נקודות השירות הוא הגדול ביותר)

$S=S\cup\{s_i\}$

end for-loop

return S

איכות הפתרון משתפרת ככל שהאלגוריתם מתקדם. ככל שמוסיפים עוד מוקדי שירות, איכות הפתרון מונוטונית (במובן החלש).

משפט - רדיוס הכיסוי של הפתרון (המרחק הגדול ביותר של מוקד ממוקד השירות הקרוב אליו ביותר) על ידי האלגוריתם המוצע, הוא לכל היותר פעמיים רדיוס הכיסוי האופטימאלי.

הערה: בעיית הפתרון האופטימאלי הוא NP-קשה.

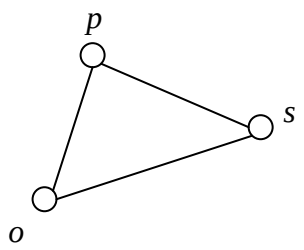
הוכחה למשפט - נקבע פתרון אופטימאלי כלשהוא. נשייך כל נקודה s_i בפתרון של האלגוריתם המוצע לנקודה הקרובה ביותר אליה בפתרון האופטימאלי $OPT(s_i)$.

נמשיך לפי מיקרים:

מקרה ראשון – כל נקודה באלגוריתם המוצע שויכה לנקודה שונה בפתרון האופטימאלי. כלומר $s_i \neq s_j \Rightarrow OPT(s_i) \neq OPT(s_j)$.

ניקח נקודה $p \in P$ כלשהיא ונניח שהאופטימום ממפה אותה למרכז $o \in OPT$.

כלומר יש נקודה $s \in S$ כך שמתקיים $OPT(s) = o$:



מאי שוויון המשולש $d(p,s) \leq d(p,o) + d(o,s)$ ומאחר וכל אחד מהמרחקים, בצד הימני של אי-השוויון, חסום על ידי רדיוס הכיסוי האופטימאלי r_{OPT} (מאחר וכל אחת מהנקודות עברה מיפוי על ידי האלגוריתם האופטימאלי) מתקיים:

$$d(p,s) \leq 2r_{OPT}$$

מקרה שני – קיימות שתי נקודות $s_i, s_j \in S$ כך שמתקיים $OPT(s_i) = OPT(s_j)$.

נניח, ללא הגבלת הכלליות, כי s_j נוספה אחרי s_i. נסמן S_{j-1} את הפתרון החלקי, לפני הוספת s_j. כאשר s_j נוספה, היא הייתה הגרועה ביותר לאלגוריתם. כלומר התקיים:

$$\forall p \in P,$$

$$d(p, S_{j-1}) \leq d(s_j, S_{j-1}) \leq d(s_j, s_i) \overset{triangle\text{-}ineq}{\leq} d(s_j, o) + d(o, s_i) \leq 2r_{OPT}$$

כלומר רדיוס הפתרון, כאשר מוסיפים את s_j, כבר חסום על ידי $2r_{OPT}$ ובגלל המונוטוניות, הפתרון הסופי בוודאי חסום על ידי $2r_{OPT}$.

QED

עד כאן אלגוריתמים משערכים.

תכנות דינאמי (Dynamic Programming)

נתחיל מדוגמא – תזמון אינטרוולים (Interval Scheduling)

ניתן לחשוב על שרת, אליו מגיעים ג'ובים כאשר כל ג'וב רץ על השרת למשך זמן מסוים ובתמורה מייצר ערך כלשהו. המגבלה היא שלא ניתן להריץ על השרת יותר מג'וב אחד בו-זמנית ולא ניתן לעצור ג'וב במהלך ריצתו.

קלט: אוסף של n אינטרוולים (ג'ובים) $\{(s_i, f_i, v_i)\}_{i=1}^{n}$, כך שלכל ג'וב יש שלושה מספרים:

(1) זמן ההתחלה s_i

(2) זמן סיום f_i

(3) ערך v_i

ומתקיים $s_i \le f_i, v_i > 0$

פלט: קבוצה S של ג'ובים זרים בזוגות (אם $i, j \in S$, אזי $([s_i, f_i) \cap [s_j, f_j) = \phi$).

מטרה: למצוא תזמון בעל ערך מרבי ב-S, כלומר למקצן את הסכום $\sum_{i \in S} v_i$.

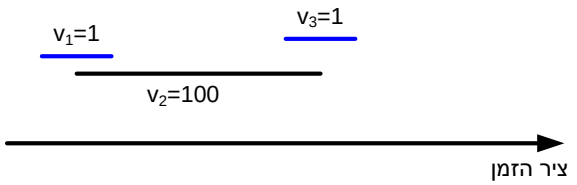
הערה: זהו מקרה פרטי של MIS שגם עד כדי קירוב לא טריביאלי הוא NP-קשה.

נתבונן על מקרה פרטי:

אם $v_i = 1$ לכל i, כלומר נמקצן את מספר האינטרוולים שלא נחתכים.

פתרון אופטימאלי (חמדן) – EDF or Earliest Deadline First. מצא את האינטרוול שזמן הסיום שלו הוא מזערי, הוסף לפתרון, הרחק מהפתרון את כל האינטרוולים שהוא חותך והמשך. מאחר והאינטרוול מסתיים ראשון, הוא חותך הכי פחות אינטרוולים שמתחילים אחריו.

דוגמא רעה - במקרה של v_i כללי הפתרון החמדן אינו עובד:



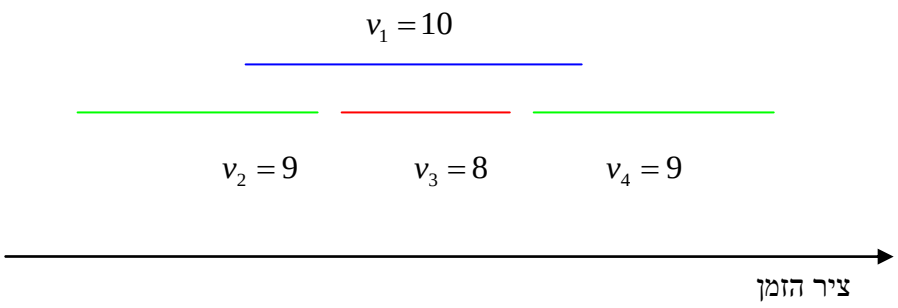
על פי האלגוריתם החמדן, נשרת את הג'וב הראשון והשלישי ונקבל ערך 2. לעומת זאת, ניתן היה לשרת רק את הג'וב השני ולקבל תועלת של 100.

מקרה פרטי נוסף:

ערך כל אינטרוול הוא האורך שלו (משך הזמן שהוא לוקח) $v_i = f_i - s_i$.

פתרון חמדן אפשרי: נתבונן על האינטרוול הארוך ביותר, נכניס לפתרון, נשמיט כל אינטרוול שהוא חותך ונמשיך.

דוגמא רעה לאלגוריתם החמדן:



האלגוריתם יבחר באינטרוול שערכו 10 במקום שלושת האינטרוולים שערכם כמעט פי 3 מערכו. האלגוריתם הוא 3-קירוב.

הפתרון לבעיה הכללית:

נניח כי האינטרוולים ממוינים לפי זמני סיום $\forall 1 \le i \le n,\ f_i \le f_{i+1}$. נסמן לכל אינטרוול i את האינטרוול האחרון שלא נחתך איתו:

$$pre(i) = \max\left(j : f_j < s_i\right)$$

בנוסף, נסמן את הפתרון האופטימאלי לאינטרוולים $1 \ldots i$ בסימון $OPT(i)$.

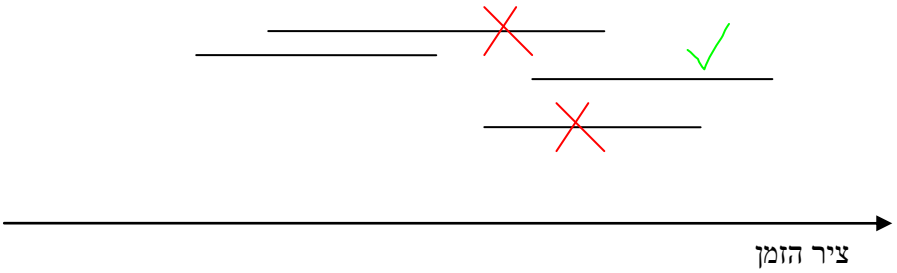
נתבונן על האינטרוול שמסתיים אחרון. קיימים שני מקרים:

האינטרוול האחרון **אינו נמצא** בפתרון האופטימאלי. במקרה זה, הפתרון האופטימאלי על כל האינטרוולים הוא אותו פתרון שנכון לבעיה זו ללא האינטרוול האחרון.

האינטרוול האחרון **נמצא** בפתרון האופטימאלי. במקרה זה יש לסלק מהבעיה את כל האינטרוולים שנחתכים עם האינטרוול האחרון (ולכן אינם יכולים להיבחר) ולהמשיך לפתור את הבעיה באופן רקורסיבי:

$$OPT(i) = \begin{cases} v_i + OPT(pre(i)), i \in S \\ OPT(i-1), i \notin S \end{cases}$$

כלומר האלגוריתם הרקורסיבי $S(i) = \max\left(v_i + S(pre(i)), S(i-1)\right)$ וכאשר $i = 0$ אז $S = 0$.



או:

$S(i)$:

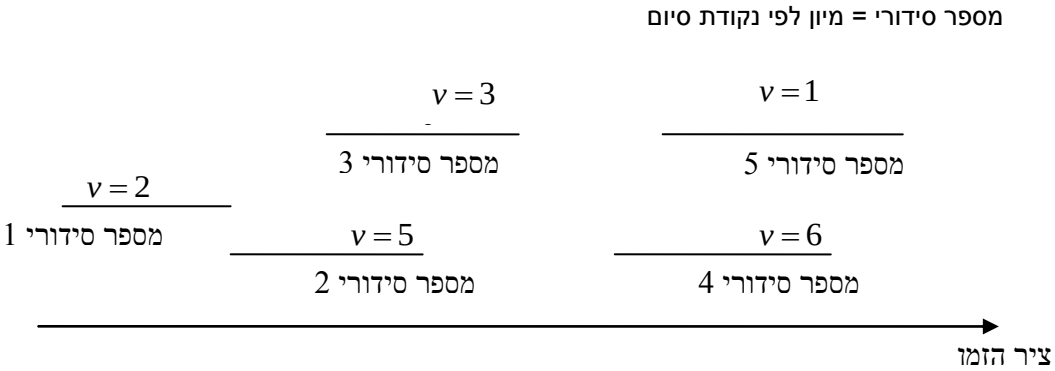
if $i=0$ return $(0,\varnothing)$

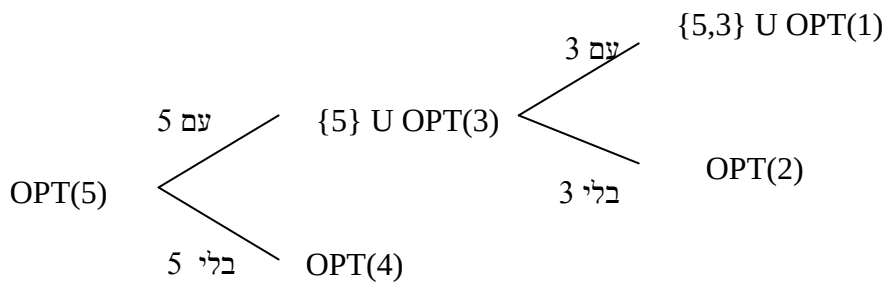
else if $v_i + S(pre(i)).value < S(i-1).value$

return $S(i-1)$

else return $(v_i + S(pre(i)).value, \{i\} \cup S(pre(i)).Intervals)$

דוגמא (ההחלטה שהתקבלה לכל מקטע, לפי המספר הסידורי שלו, מסומנת על גבי הקווים)

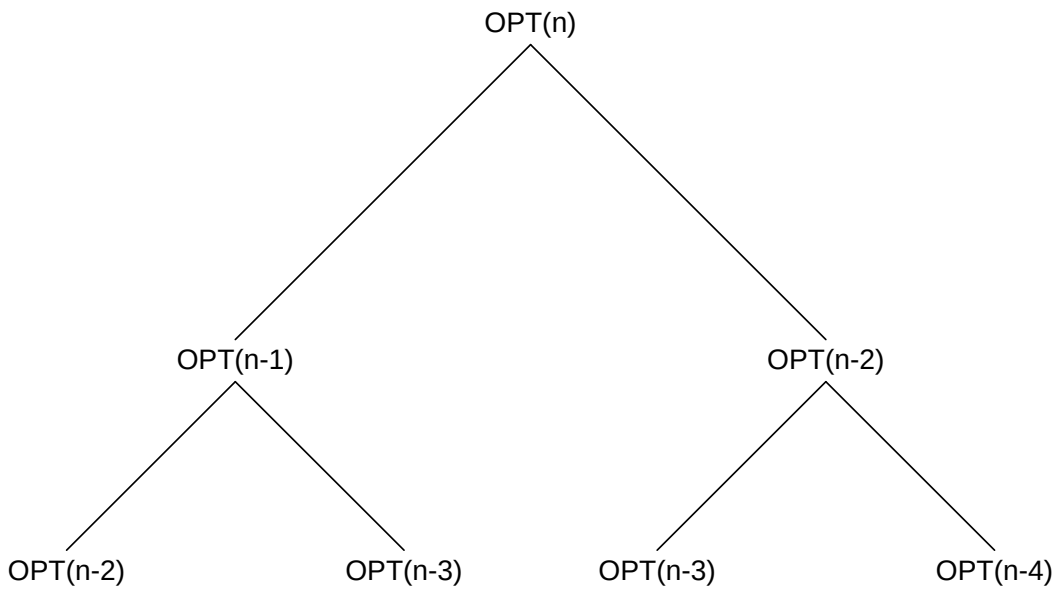




<u>סיבוכיות האלגוריתם</u>: נסמן ב $T(n)$ את זמן הריצה ל $S(n)$. כדי לחשב את $OPT(n)$ יש לחשב את $OPT(n-1)$ ואולי אף את $OPT(n-2)$. כלומר, זמן הסיבוכיות ניתן ע"י המשוואה:

$$T(n) \le T(n-1) + T(n-2) + O(1) \le 2T(n-1) = 2^n$$

(כאשר $T(0)=1$). זוהי משוואת פיבונאצ'י שעבורה הפתרון הנאיבי הוא אקספוננציאלי (לא ישים). לכן, ניסוח הבעיה אלגנטי אך אינו מספק מבחינת סיבוכיות. נשים לב שבעוד שישנם n איברים אנו מבצעים 2^n חישובים. מדוע?



נשים לב כי למרות שגודל העץ אקספוננציאלי, יש בו הרבה מאד <u>חזרות</u>. ניתן לראות כי בסה"כ יש לחשב רק n ערכים שונים של OPT. ניתן יהיה לשפר את הסיבוכיות באמצעות שימוש **בטבלה בה נשמור ערכים שכבר חישבנו**.

נייצר מערך $M[0\ldots n]$ אותו נאתחל עם האיבר הריק ($\perp$) בכל מקום. האלגוריתם:

$m_S(i)$:
  $if\ \ i=0\ return\ 0$
  $if\ \ M[i]=\perp\ then$
    $M[i]:=\max(m_S(i-1), v_i + m_S(pre(i)))$
  $return\ M[i]$

<u>סיבוכיות האלגוריתם</u>: אם לא מתבוננים על הקריאות הרקורסיביות, הסיבוכיות של כל שלב היא $O(1)$. לכן יש לספור את מספר ההפעלות הרקורסיביות. לכל ערך i יוצרים פעם אחת שתי קריאות רקורסיביות. מתחילים לרדת בעץ שקיבלנו לצד מסוים. בדרך חזרה, המערך מתמלא בערכים וכאשר מקבלים קריאה רקורסיבית לנוע בכיוון אחר בעץ, אם הערך קיים במערך, לא מתבצע זימון רקורסיבי (לא עובר את שורת $if\ \ M[i]=\perp\ then$). מאחר וישנם n ערכים ל- i זמן הריצה הוא $O(n)$.

יש לציין שיש עוד למיין את הקבוצות ולכן סיבוכיות קדם-חישוב (pre-processing) הוא $O(n\log n)$.

הערה: הצעה לחישוב $pre(i)$ - נמיין לפי זמני סיום. נקבל i. באמצעות s_i נסרוק ונמצא את f_j הראשון שקטן ממנו.

<u>בעיה - התאמת ליניארית למקוטעין (Piece-wise linear).</u>

<u>קלט</u>: סדרת נקודות על המישור $\{(x_i, y_i)\}_{i=1}^{n}$

<u>דרישה</u>: נקרב את הדגימות על ידי סגמנטים של קווים ישרים. נמצא מספר ישרים $L_j = a_j x + b_j$ והתחום שבו הם רלוונטי $[s_j, f_j]$.

<u>מטרות</u>:

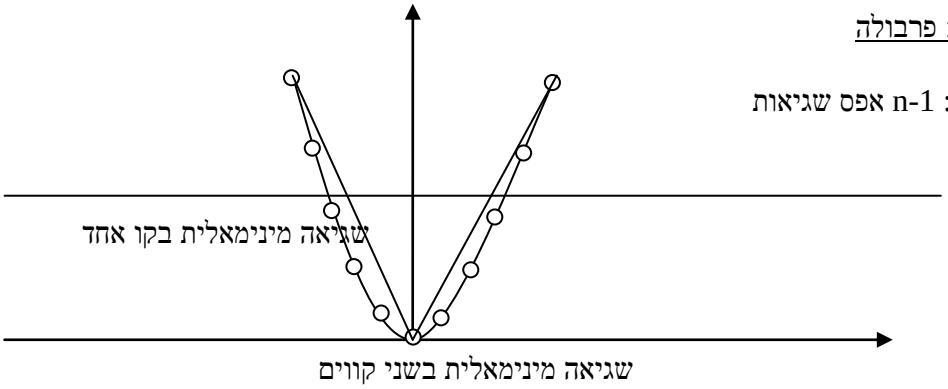
(1) מזעור השגיאה הריבועית : $\sum_{i=1}^{n}\left(y_i - \left(a_{j(i)}x_i + b_{j(i)}\right)\right)^2$

כאשר $j(i): x_j \in [s_j, f_j]$

(2) מזעור מספר הסגמנטים (אחרת ניתן היה להעביר קו ישר בין כל זוג נקודות סמוכות ולהבטיח בכך שגיאה ריבועית אפס).

<u>דוגמה: פרבולה</u>

קווים : n-1 אפס שגיאות



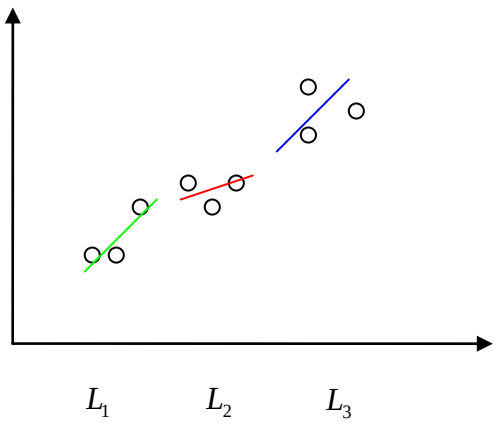
ניתן להתבונן על הבעיה כעל בעיית איתור נקודות השבירה, בהן עוברים מסגמנט אחד לשני. כל נקודה יכולה להיות נקודת שבירה ולכן מספר החלוקות האפשרי הוא 2^n.

נניח כי על כל סגמנט נוסף שמכניסים משלמים קנס המיוצג ע"י פרמטר C. כלומר, כל סגמנט עולה לנו C יחידות של שגיאה ריבועית.

בהינתן קבוצת נקודות משואת היישר הממזערת את השגיאה הריבועית נתונה לפי:

$$a = \frac{n\sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n\sum_{i=1}^{n} x_i^{\,2} - (\sum_{i=1}^{n} x_i)^2} \qquad b = \frac{\sum_{i=1}^{n} y_i - a(\sum_{i=1}^{n} x_i)}{n}$$

<u>תצפית:</u> לא כל תתי הקבוצות רלוונטיות. כיצד מחלקים למקטעים? בכל מקטע נוכל לקרב בפשטות באמצעות הנוסחא למעלה:



ל-x-ים יש סידור לאורך הישר.

<u>נסמן</u>: $E[i,j]$ = ערך השגיאה הריבועית הכוללת אם נקרב את אוסף הנקודות $(x_i, y_i)\ldots(x_j, y_j)$ ע"י ישר יחיד.

כאמור, יש למצוא נקודת שבירה. אם ננסה את כל האופציות, נגדיר בתור נקודת שבירה את נקודה k. כל הנקודות מימין לה ישוערכו באמצעות ישר יחיד ועל התחום שנמצא משמאלה נגדיר שוב, רקורסיבית, נקודה נוספת. אם $OPT(n)$ מציין את המחיר של הפתרון האופטימאלי ל-n נקודות אז מתקיים:

$$OPT(n) = \min_{1 \le k \le n-1} \{E[k,n] + C + OPT(k)\}$$

מקרה בסיס:

$$OPT(1) = 0$$

הזמן לפתור בעיה בגודל n:

$$T(n) \le T(0)+T(1)+\cdots+T(n-1)+1=\sum_{k=0}^{n-1}T(k)+1$$

$$T(n) \le 2^n$$

איך נוכל לשפר? אלגוריתם יעיל בעזרת תכנות דינאמי. ניצור מערך $M[1\ldots n]$ כאשר $M[i]$ מכיל את הפתרון האופטימאלי לתת הבעיה עם הנקודות $1\ldots i$.

1. נחשב $E[i,j]$ בטבלה לכל i<j. כל חישוב לוקח $j-i+1$ חישובים (לפי מספר הנקודות בקטע). לכן $\sum_{j=1}^{n}\sum_{i=1}^{j-1}(j-i+1)\sim n^3$ חישובים.
2. נאתחל טבלה $M[n]\leftarrow\perp$ לכל n
3. נחשב:

$p(n)$:

$if \quad n=1 \quad return \quad 0;$

$if \quad M[n]=\perp \quad then$

$\quad M[n]:=\min_{1\le k\le n-1}\{C+E[k,n]+p(k)\}$

$return \quad M[n];$

עבור סיבוכיות האלגוריתם שוב נתעניין בקריאה הרקורסיבית (כל קריאה אחרת בשלב 3 היא $O(1)$):

שלב 1: מילוי הטבלה כ n^3 חישובים

שלב 2: n חישובים

שלב 3: עבור כל איבר ב M יש לבצע קריאה למינימום שסורק טבלה לאורך n ערכיה (רץ על k). הזימון הרקורסיבי יתבצע רק n פעמים (בגלל תנאי $if \quad M[n]=\perp \quad then$) ולכן סה"כ n^2 חישובים.

<u>הערה</u>: ניתן לבחור את פונקצית השגיאה כרצוננו (אין האלגוריתם תלוי בשגיאה ריבועית דווקא).