

אוגדן בעיות מפורסמות במדעי המחשב

מציאת המסלול הקצר ביותר

כתבה שרה פולק

תיאור הבעיה

הבעיה המתוארת בפרק זה עוסקת במציאת המסלול הקצר ביותר בין שני צמתים בגרף משוקלל. את הבעיה הגדיר דייקסטרה בשנת 1956, כשעבד במרכז למתמטיקה באמסטרדם. דייקסטרה התבקש להדגים, בכנס בינלאומי של מתמטיקאים שעמד להיערך בעיר, את יכולת החישוב של מחשב ARMAC, עליו עבדו במרכז למתמטיקה. דייקסטרה חיפש בעיה מתאימה וחשב כי שימוש בתכנית המחשבת ומוצאת המסלול הקצר ביותר בין שני ערים, תדגים היטיב את יכולת החישוב של ARMAC. דייקסטרה הגדיר את המסלול הקצר ביותר בין שתי ערים, כמסלול שהמרחק שלו בין עיר המוצא ועיר היעד הוא הקטן ביותר, אולם לאלגוריתם הפותר בעיה זו, יש שימושים רבים שחלקם יפורטו בהמשך.

פתרון הבעיה

נגדיר בצורה פורמלית את הבעיה שהציב דייקסטרה:

הקלט הוא גרף מכוון משוקלל G (משקל קשת הוא מספר חיובי), וצומת מקור s . נסמן ב- V את קבוצת הצמתים ב- G וב- E את קבוצת הקשתות. כל קשת (u, v) מייצגת קשר בין צומת u לצומת v . המשקל של קשת (u, v) $w(u, v) \geq 0$ מייצגת את העלות של מעבר מצומת u לצומת v .

עלות המסלול מצומת התחלה s לצומת היעד t הוא סכום המשקלים של הקשתות המרכיבות את המסלול. האלגוריתם מוצא לא רק את המסלול בין s ל- t , אלא מאפשר למצוא גם את כל המסלולים הקצרים ביותר מ- s לכל אחת מהצמתים בגרף (בתנאי שיש מסלול ביניהם).

האלגוריתם שומר לכל צומת v את האורך $d[v]$ של המסלול הקצר ביותר שנמצא עד כה (מצומת s אל צומת v). בשלב ראשון האורך של $d[s]$ הוא 0 (והא מציינ שאורך המסלול מהצומת s לצומת עצמה, הוא 0) ושאר הצמתים מכילים את הערך $d[v] = \infty$ (המציינ שאנו לא יודעים איזה מסלול מוביל לצמתים אלו). כשהאלגוריתם מסתיים, $d[v]$ יכיל את האורך הכולל של המסלול הקצר ביותר מ- s ל- v או שישאר ∞ אם אין שום מסלול ביניהם. בנוסף לכך, אנו משתמשים במערך $previous[v]$ השומר את הצומת הקודם לצומת v שנמצא במסלול הקצר ביותר אליו. מערך זה מאפשר לנו בסיום למצוא את המסלול מ- s ל- t . הפעולה העיקרית באלגוריתם היא **edge relaxation** ומשמעותה: אם יש קשת בין u ל- v , אז המסלול הקצר ביותר בין s ל- u מתעדכן והוא כולל את המסלול מ- s ל- v ע"י הוספת הקשת $edge(u, v)$ לסופו, ואורך המסלול הוא $d[u] + w(u, v)$. אם אורכו קטן מ- $d[v]$, אנו יכולים להחליף את הערך הנוכחי בערך החדש שחשבנו.

אנו מבצעים את הפעולה **edge relaxation** עד שכל הערכים $d[v]$ מייצגים את המחיר של המסלול הקצר ביותר מ- s ל- v . האלגוריתם מאורגן כך שכל קשת $edge(u, v)$ מעודכנת (relax) רק פעם אחת, כאשר $d[u]$ מגיע לערכו הסופי.

מבני הנתונים שבהם משתמשים באלגוריתם זה הם: שתי קבוצות של צמתים S ו- Q . הקבוצה S היא כל הצמתים שעבורם אנו יודעים את הערך $d[v]$ (והוא כזכור מבטא את אורך המסלול הקצר ביותר), והקבוצה Q מכילה את שאר הצמתים. בהתחלה, הקבוצה S היא קבוצה ריקה, ובכל איטרציה צומת אחת מועברת מקבוצה Q לקבוצה S . הצומת שמועבר הוא הצומת עם הערך $d[u]$ הנמוך ביותר. כאשר צומת u מועבר ל- S , האלגוריתם מחשב ומעדכן את המרחקים של כל קשת (u,v) שיוצאת מ- u . באלגוריתם הבא: הפונקציה $u := \text{Extract-Min}(Q)$, מחפש את הצומת u בקבוצת הצמתים Q שיש לה את הערך $d[u]$ הנמוך ביותר. צומת זו מועברת והפונקציה מחזירה את הקבוצה Q המעודכנת.

Dijkstra(G,w,s)

```

1  for each vertex  $v$  in  $V[G]$ 
                                     {אתחול}
2  do  $d[v] = \text{infinity}$ 
3   $\text{previous}[v] = \text{undefined}$ 
4   $d[s] = 0$ 
5   $S = \text{empty set}$ 
6   $Q = \text{set of all vertices}$ 
                                     {כל עוד לא עברנו על כל צמתי הגרף}
7  while  $Q$  is not an empty set
8  do  $u = \text{Extract-Min}(Q)$ 
9   $S = S \text{ union } \{u\}$ 
10 for each edge  $(u,v)$  outgoing from  $u$ 
11 do if  $d[v] > d[u] + w(u,v)$  // Relax  $(u,v)$ 
12 then  $d[v] = d[u] + w(u,v)$ 
13  $\text{previous}[v] = u$ 

```

אם אנו רוצים רק המסלול הקצר ביותר בין s ו- t , נוכל לסיים את האלגוריתם בשורה 8 כאשר $u == t$.

כדי לקרוא את המסלול הקצר מ- s ל- t , נשתמש במערך previous :

$S = \text{empty sequence}$

$u = t$

while defined u

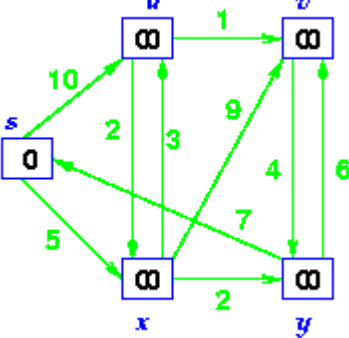
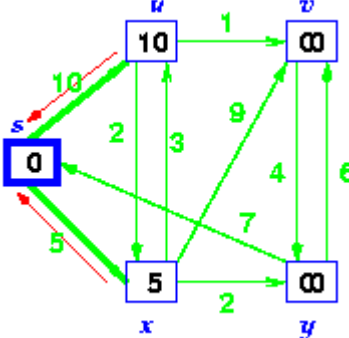
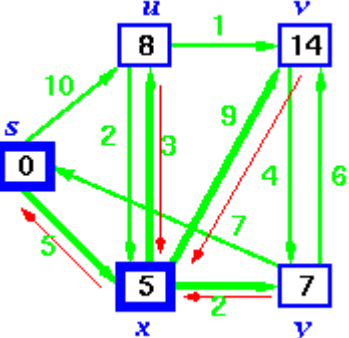
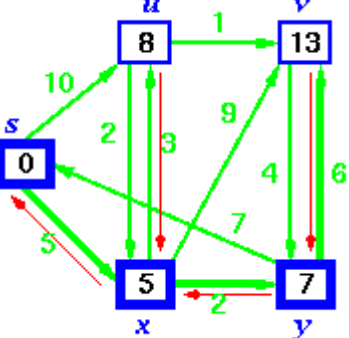
do insert u to the beginning of S

$u = \text{previous}[u]$

הרצף ב- S היא רשימה של צמתים שנמצאים על המסלול הקצר ביותר בין s ל- t .

נדגים את הרצת האלגוריתם באיור הבא :

איור מס. 1. שלבי ביצוע של אלגוריתם למציאת המסלול הקצר ביותר

	<p>שלב 1 : אתחול: צומת מקור ערכו 0 כל שאר הצמתים מאותחלים ∞</p>
	<p>שלב 2 : הצומת הקרוב ביותר ל- s : בשלב הראשון הוא s המרחק הוא $d[s]=0$ העבר את צומת s ל- S והסר אותו מ- Q חשב את המרחק של השכנים של s : $d[s,u]=10$ $d[s,x]=5$ עדכן s צומת קודם ל- u,x (חיצים אדומים) : $previous[u]=s$ $previous[x]=s$</p>
	<p>שלב 3 : הצומת הקרוב ביותר ל- s : הוא x המרחק הוא $d[s,x]=5$ העבר את צומת x ל- S והסר אותו מ- Q חשב את המסלול הקצר ביותר מ- s לשכנים של x : $d[s,x] + d[x,u]=8$ $d[s,x] + d[x,v]=14$ $d[s,x] + d[x,y]=7$ עדכן x צומת קודם ל- u,v,y (חיצים אדומים) : $previous[u]=x$ $previous[v]=x$ $previous[y]=x$ שימו לב שכעת מצאנו מסלול קצר יותר מ- s ל- u</p>
	<p>שלב 4 : הצומת הקרוב ביותר ל- s : הוא y המרחק הוא $d[s,y]=7$ העבר את צומת y ל- S והסר אותו מ- Q חשב את המסלול הקצר ביותר מ- s לשכנים של y : $d[s,x] + d[y,v]=13$ עדכן y צומת קודם ל- v (חיצים אדומים) : $previous[v]=y$ שימו לב שכעת מצאנו מסלול קצר יותר מ- s ל- v</p>

	<p>שלב 5 :</p> <p>הצומת הקרוב ביותר ל- s : הוא u המרחק הוא $d[s,u]=8$ העבר את צומת u ל- S והסר אותו מ-Q חשב את המסלול הקצר ביותר מ- s לשכנים של u : $d[s,x] + d[u,v]=9$ עדכן u צומת קודם ל- v (חיצים אדומים) : $previous[v] = u$ שימו לב שכעת מצאנו מסלול קצר יותר מ- s ל- v</p>
	<p>שלב 6 :</p> <p>הצומת האחרון הוא v המרחק הוא $d[s,v]=9$ העבר את צומת v ל- S והסר אותו מ-Q</p>

סיבוכיות : נסמן את מספר הצמתים בגרף ב- n. את Q ניתן לממש ברשימה לינארית או במערך, ולכן הפעולה Extract-Min (Q) היא חיפוש לינארי ברשימה של כל הצמתים שב- Q, לכן הסיבוכיות היא $\Theta(n^2)$.

נכונות : ההוכחה היא בדרך השלילה. נניח שמצאנו עבור צומת x את המסלול הקצר ביותר והשכן הקודם הוא $previous[x] = y$. נוכיח שלא ניתן למסלול מסלול קצר אחר אל x. על דרך השלילה, נניח שיש מסלול קצר יותר שמוביל משכן אחר והוא z ל- x. אולם הנחה זו מביא לסתירה משום שיש שתי אפשרויות :

- אם המסלול אל z הוא הקצר ביותר, אזי בדקנו כשטפלנו ב- z שאת המסלול

$$d[s,z] + d[z,x]$$

ואם הוא היה קצר יותר, אזי היה נרשם $previous[x] = z$ ולא $previous[x] = y$.

- אפשרות שנייה שעדיין לא מצאנו את המסלול הקצר ביותר ל- z. אם המרחק של המסלול הזמני ל- z גדול מהמרחק של המסלול ל- x, אז לא יתכן שהמסלול דרך z קצר יותר. לכן נוכל להניח שהמסלול אל z קטן מהמסלול של x. אבל במקרה כזה z היה צריך להיבחר בפעולת Extract-Min(Q) ולא x.

אולם מאחר וכבר מצאנו את המסלול הקצר ביותר ל- x, הוא בהכרח קטן יותר מאשר המסלול הקצר ביותר אל z, ולכן זו סתירה. מכאן ניתן להסיק שברגע שחשבנו (relax) מסלול קצר ביותר לצומת, לא ימצא מסלול אחר קצר יותר.

אדגר דייקסטרה (Edsger Wybe Dijkstra)

דייקסטרה נולד ברוטרדם (הולנד) ב-1930 כבן לשני הורים מדענים (אב היה כימאי והאם מתמטיקאית). בשנת 1945 החל ללמוד באוניברסיטה של Leiden כשהוא מתלבט בין לימודי מתמטיקה ולימודי פיזיקה. לבסוף בחר דייקסטרה ללמוד פיזיקה, והוא נימק זאת בכך שמתמטיקה הוא יוכל ללמוד תמיד (מוכר לנו, לא?). במהלך לימודיו נדרש דייקסטרה לבצע חישובים מרובים, ולכן בקיץ 1951 נרשם דייקסטרה לקורס תכנות באוניברסיטת קיימברידג' ובמקביל הוא מצא עבודה חלקית במרכז למתמטיקה באמסטרדם. בתקופה זו החלה התעניינותו הרבה בנושא שהוגדר באותו זמן **כתכנות**. במרכז למתמטיקה באמסטרדם, עבד דייקסטרה על מחשב בשם ARMAC שפותח עבור המרכז ובכתביו הוא תיאר את המחשב, כמחשב ענק שתפס כיתה שלמה. למחשב היה תוף מגנטי ששימש כזיכרון, ולכתבת תוכניות השתמשו בפקודות מכונה (שפות כמו פורטרן או ליספ עדיין לא הומצאו).

דייקסטרה סיים במהירות את תואר בפיזיקה והתמסר יותר ויותר לתכנות. באותן השנים תכנות לא הוכר עדיין כמקצוע, ולכן הוא המשיך לרשום את מקצועו כפיזיקאי. דייקסטרה המשיך לעבוד במרכז למתמטיקה עד שבשנות ה-70 הוצעה לו משרת עמית חוקר ב Burroughs Corporation שבארצות הברית ב-1972 הוא קיבל פרס ACM Turing על תרומתו בפיתוח שפת ALGOL. בשנות ה-80 הוא עבר ל-Austin בטקסס ושם עבד שפרש לגמלאות. דייקסטרה נפטר בשנת 2002.

תרומתו של דייקסטרה למדעי המחשב היתה רבה ואנו נציין מספר הישגים בולטים: ב-1956 פיתח את האלגוריתם למציאת המסלול הקצר ביותר אותו אנו מתארים המאמר זה. אלגוריתם זה ידוע היום גם בשם אלגוריתם דייקסטרה. כמו כן הוא פתר בעיה הקשורה לאלגוריתם של מציאת המסלול הקצר ביותר, והיא מציאת הדרך להעביר חשמל לכל המעגלים החיוניים במחשב תוך שימוש במינימום חוטי חשמל. דייקסטרה כינה בעיה זו בשם:

"shortest subspanning tree algorithm."

בשנות ה-60 המוקדמות דייקסטרה יישם את הרעיון של מניעה הדדית לתקשורת בין המחשב והמקלדת שלו. הוא השתמש באותיות P ו-V לייצג את שתי הפעולות בהן משתמשים עד היום בבעיית המניעה הדדית, ומאז 1964 רעיון זה הוא הבסיס למעבדים מודרניים עם לוח זיכרון. כמו כן הגדיר דייקסטרה את בעיית הפילוסופים הסועדים, בעיה המתייחסת לפתרון של בעיות הרעבה ו-deadlock, הקיימת במחשבים מרובי תהליכים ומעבדים.

דייקסטרה התייחס גם לטכניקות תכנות ובעיקר לשימוש בהוראת GO TO. דייקסטרה פרסם מאמר בשם: "GO TO statement considered harmful". במאמר הוא מעביר ביקורת על השימוש בהוראת GO TO. הוא טען כי תוכנית המכילה פקודות GO TO, קשה להבנה, ובכך הניח את היסודות לתפישה של תכנית מובנית. דייקסטרה המליץ להשתמש במבני בקרה ובלולאת While במקום בהוראת GO TO. דייקסטרה גם התייחס לחשיבות של שימוש במתמטיקה ככלי לתכנות

דייקסטרה התעניין גם באימות פורמלי. הדיעה הרווחת בשנות ה-70 היא שקודם יש לכתוב תוכנית ואז לספק את ההוכחה המתמטית להוכחת הנכונות. דייקסטרה התנגד וטען שהוכחות כאלו הן ארוכות ומסורבלות והן לא מספקות שום תובנה על התוכניות שפותחו. כאלטרנטיבה הוא הציע שיטה בה ההוכחה והתוכנית יפותחו ביחד (program derivation) כאשר בתחילה התוכנית תאופיין מתמטית והגדרה זו

תומר לתכנית מחשב, מה שיאפשר לקבל תכנית נכונה. בהמשך, רוב עבודתו היתה קשורה לדרכים ושיטות לפשט ארגומנטים מתמטיים.

דייקסטרה תיעד את כל הבעיות והמכתבים שכתב והותיר את הכתבים באתר:

<http://www.cs.utexas.edu/users/EWD/>

הרחבות ושימושים

בעיה זו מציגה שיטת חיפוש, שהיא פעולה נפוצה בתחומים רבים במדעי החשב. לדוגמה, אלגוריתם דייקסטרה שימושי בבעיות תעבורה בהן יש לחשב את המסלול הקצר ביותר. בעיות תעבורה הן בעיות של מציאת מסלולי נסיעה בין ערים או העברת מנות בין צמתים ברשת תקשורת (אינטרנט) וכדומה. כאמור שיטות חיפוש היא פעולה נפוצה ביישומים רבים. אנו לדוגמה, משתמשים בה למעבר על עצים בינריים. ניתן לסווג שיטות חיפוש על פי מאפיינים רבים. אחד הסיווגים מגדיר שני סוגים: חיפוש סיסטמתי וחיפוש היוריסטי. אלגוריתם דייקסטרה מבצע חיפוש סיסטמתי על הגרף בדומה לשיטות חיפוש שאנו מלמדים בעיצוב תכנה על עצים בינריים. המשמעות היא שהחיפוש עובד בצורה שיטתית (המוגדרת על ידי האלגוריתם) על כל הצמתים בגרף. למשל בחיפוש לרוחב אנו מתקדמים כל הצמתים בהתאם לרמתם.

שיטות חיפוש אחרות המקובלות בבניה מלאכותית הן חיפושים היוריסטיים. בהם החיפוש מתבצע בהתאם לחישוב של פונקציה היוריסטית שמסייעת למצוא את הצומת הבא אליה נתקדם. כלומר בכל איטרציה, נבחנים הצמתים שאליהם ניתן להגיע מצומת s בה אנו נמצאים, ולכל צומת כזו מחושבת פונקציה היוריסטית. החישוב מעריך את הסיכוי של צומת הבאה להימצא על המסלול הקצר ביותר למטרה. הצומת שציונה הוא הטוב ביותר, נבחרת להיות הצומת הבאה במסלול. שיטה זו לא בהכרח את הפתרון היעיל ביותר. עם זאת, לעיתים כאשר מרחב החיפוש בחיפוש סיסטמתי הוא גדול מאוד וכאשר בעיה לא ניתנת לפתרון בזמן סביר (או בכלל לא), חיפוש היוריסטי יכול להיות שיטה מתאימה לביצוע החיפוש. כמובן ככל שנצליח לייצר פונקציה היוריסטית מתאימה יותר הפלט שנקבל יהיה טוב יותר. שיטות חיפוש היוריסטיות נפוצות בתחומים רבים. אחד מהם הוא משחקים, כאשר אנו רוצים לכתוב תכנית בה מחשב משחק נגד אדם. כאשר נתון מצב במשחק ועל המחשב לבחור את המהלך הבא, הוא מחשב את כל המהלכים האפשריים ולכל מהלך כזה את הסיכוי שלו להוביל לניצחון, ובהתאם הוא בוחר את המהלך המבטיח ביותר.

מקורות

בתכנית הלימודים אנו מתייחסים במפורש לאלגוריתם דייקסטרה בתקשורת נתונים (הנלמד במגמת טכ"מ). ניתן לשלב אלגוריתם זה בתורת המחשב ובבניה מלאכותית. ניתן לשלב בעיה זו בעיצוב תכנה, כהעשרה לשיטות חיפוש על עצים בינריים. חומר עזר נוסף:

<http://www.acm.org/classics/oct95/>

<http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/dijkstra.html>

סימולציות של אלגוריתם דייקסטרה:

<http://carbon.cudenver.edu/~hgreenbe/sessions/dijkstra/DijkstraApplet.html>

http://students.ceid.upatras.gr/~papagel/project/kef5_7_1.htm