22 פרויקט 6: ניתוב באמצעות אלגוריתם דייקסטרה

פרוטוקולי שער פנימי (IGPs) משתפים מידע אחד עם השני כך שכל נתב מכיל את כל המידע על הרשת שהוא חלק ממנה. כך, כל נתב יכול לקבל החלטות ניתוב אוטונומיות בהנחה שיש לו כתובת IP. לא משנה לאן הוא הולך, הנתב תמיד יכול להעביר את המנות בכיוון הנכון.

IGPs כמו Open Shortest Path First (OSPF) משתמשים באלגוריתם דייקסטרה כדי למצוא את הנתיב הקצר ביותר דרך גרף משוקלל.

בפרויקט זה, אנחנו נמדל את הניתוב הזה. אנחנו ניישם את אלגוריתם דייקסטרה כדי להדפיס את הנתיב הקצר ביותר מכתובת IP אחת לכתובת IP אחרת, תוך הצגת כתובת ה-IP של כל הנתבים בדרך.

לא נשתמש ברשת אמיתית עבור זה. במקום זאת, התוכנית שלך תקרא קובץ JSON שמכיל את תיאור הרשת ולאחר מכן תחשב את הנתיב מהמידע הזה.

22.1 פונקציות אסורות

אתה תיישם את אלגוריתם דייקסטרה בעצמך. אין זיכוי על שימוש בספרייה קיימת!

22.2 רענון גרפים

גרפים עשויים מקודקודים (vertices) וקשתות (edges). לפעמים מקודקודים קוראים "קודקודים" או "קודקודים" או "צמתים". קשתות הן חיבורים בין צומת אחד לצומת אחר.

כל צומת בגרף יכול להיות מחובר לכל מספר של צמתים אחרים, גם לאפס. צומת יכול להיות מחובר לכל צומת אחר. הוא אפילו יכול להתחבר לעצמו.

קשת יכולה להיות עם משקל שמייצג את העלות של חציית הקשת כאשר אתה עובר בגרף.

למשל, דמיין מפת כבישים שמציגה ערים וכבישים בין הערים. כל כביש מתויג עם אורך שלו. בדוגמה הזו, הערים יהיו קודקודים, הכבישים יהיו קשתות, ואורך הכביש יהיה משקל הקשת.

כאשר עוברים בגרף, המטרה היא למזער את סך כל משקלי הקשתות שאתה נתקל בהן בדרך. במפה שלנו, המטרה תהיה לבחור קשתות מהעיר ההתחלתית דרך כל הערים הביניים עד לעיר היעד כך שנעבור את המרחק הכולל המינימלי.

22.3 סקירה על אלגוריתם דייקסטרה

אדגר דייקסטרה היה מדען מחשבים מפורסם שהמציא הרבה דברים, אבל אחד מהם היה כל כך משפיע שהוא הפך להיות ידוע רק בשמו: אלגוריתם דייקסטרה.

טיפ מקצועי: הסוד לאיות "Dijkstra" הוא לזכור ש-"ijk" מופיע בסדר.

אם אתה רוצה למצוא את הנתיב הקצר ביותר בין צמתים בגרף חסר משקל, אתה פשוט צריך לבצע חיפוש לפי רוחב עד שתמצא את מה שאתה מחפש. המרחקים נמדדים רק ב"הופס".

אבל אם אתה מוסיף משקלים לקשתות בין הצמתים, חיפוש לפי רוחב לא יכול לעזור לך להבדיל ביניהם. אולי יש נתיבים שהם מאוד רצויים, ואחרים מאוד לא רצויים.

אלגוריתם דייקסטרה יכול להבדיל. זהו חיפוש לפי רוחב עם twist.

מהות האלגוריתם היא זו: לחקור החוצה מהנקודה ההתחלתית, רודפים רק אחרי הנתיב עם אורך כולל הקטן ביותר עד כה.

המשקל הכולל של כל נתיב הוא סכום המשקלים של כל הקשתות שלו.

בגרף מחובר היטב, יהיו הרבה נתיבים פוטנציאליים מההתחלה אל היעד. אבל מכיוון שאנחנו רק רודפים אחרי הנתיב הקצר ביותר הידוע עד כה, אנחנו לא נרדוף אחרי נתיב שייקח אותנו מיליון מיילים בדרך, כל עוד אנחנו יודעים על נתיב שיותר קצר ממיליון מיילים.

22.4 יישום אלגוריתם דייקסטרה

דייקסטרה בונה מבנה עץ על גבי גרף. כאשר אתה מוצא את הנתיב הקצר ביותר מצומת כלשהו חזרה לעבר ההתחלה, הצומת רושם את הצומת הקודם בנתיב שלו כהורה שלו.

אם מאוחר יותר יתגלו נתיב קצר יותר, ההורה משתנה לצומת בנתיב הקצר החדש.

ויקיפדיה מציעה כמה דיאגרמות מצוינות שמציגות את זה בפועל.

אז איך לגרום לזה לעבוד?

אלגוריתם דייקסטרה עצמו רק בונה את העץ שמייצג את הנתיבים הקצרים ביותר חזרה להתחלה. מאוחר יותר נעקוב אחרי עץ הנתיבים הקצרה הזה כדי למצוא נתיב מסוים.

אלגוריתם דייקסטרה לחישוב כל הנתיבים הקצרים ביותר בגרף מנקודת מקור: התחלה: צור קבוצת to_visit ריקה. זו קבוצת הצמתים שעדיין לא ביקרנו בהם. צור מילון מרחקים. עבור כל צומת נתון (כמפתח), הוא יחזיק את המרחק מאותו צומת לנקודת ההתחלה. צור מילון הורים. עבור כל צומת נתון (כמפתח), הוא מציין את המפתח של הצומת שמוביל חזרה לנקודת ההתחלה (על הנתיב הקצר ביותר). עבור כל צומת: הגדר את ההורה שלו ל-None. הגדר את המרחק שלו לאין סוף. (ל-Python יש אינסוף ב-math.inf, אבל אתה יכול גם להשתמש במספר מאוד גדול, לדוגמה 4 מיליארד). הוסף את הצומת לקבוצת to_visit. הגדר את המרחק לנקודת ההתחלה ל-0.

הרצה: כשהקבוצה to_visit אינה ריקה: מצא את הצומת עם המרחק הקטן ביותר ב-to_visit. קרא לו "הצומת הנוכחי". הסר את הצומת הנוכחי מקבוצת to_visit. עבור כל אחד מהשכנים של הצומת הנוכחי שעוד נמצאים ב-to_visit: חשב את המרחק מנקודת ההתחלה לשכן. זהו המרחק של הצומת הנוכחי בתוספת משקל הקשת לשכן. אם המרחק המחושב קטן מהערך הנוכחי של השכן במרחק: הגדר את ערך השכן במרחק כמרחק המחושב. הגדר את ההורה של השכן לצומת הנוכחי. [תהליך זה נקרא "ריכוך". המרחקים של הצמתים מתחילים באין סוף ו"רכים" עד למרחקים הקצר ביותר.]

ויקיפדיה מציעה את הקוד המזוייף הזה, אם זה יותר נעים לעיכול:

python
Copy code

```
function Dijkstra(Graph, source):

for each vertex v in Graph.Vertices:
    dist[v] ← INFINITY
    prev[v] ← UNDEFINED
    add v to Q
```

```
dist[source] ← 0
while Q is not empty:
    u ← vertex in Q with min dist[u]
    remove u from Q
    for each neighbor v of u still in Q:
        alt ← dist[u] + Graph.Edges(u, v)
        if alt < dist[v]:
            dist[v] ← alt
            prev[v] ← u
return dist[], prev[]
```

בשלב זה, בנינו את העץ שלנו שמורכב מכל הצמתים ההורים.

כדי למצוא את הנתיב הקצר ביותר מנקודה אחת חזרה להתחלה (בגובה שורש העץ), אתה צריך פשוט לעקוב אחרי מצביעי ההורים מאותה נקודה חזרה לעבר השורש.

קבל את הנתיב הקצר ביותר מנקודת מקור ליעד: הגדר את הצומת הנוכחי לצומת היעד. הגדר את הנתיב שלנו כמערך ריק. כאשר הצומת הנוכחי לא הוא הצומת ההתחלתי: הוסף את הצומת הנוכחי לנתיב. הצומת הנוכחי = ההורה של הצומת הנוכחי. הוסף את הצומת ההתחלתי לנתיב. כמובן, זה יבנה את הנתיב בסדר הפוך. זה חייב להיות כך, כי כל מצביעי ההורים מצביעים חזרה לנקודת ההתחלה בשורש העץ. תוכל להפוך אותו בסוף, או להריץ את האלגוריתם הראשי של דייקסטרה כשאתה מעביר את היעד כנקודת מקור.

22.4.1 קבלת המרחק המינימלי

חלק מהאלגוריתם הוא למצוא את הצומת עם המרחק המינימלי שעדיין נמצא בקבוצת to_visit.

לפרויקט הזה, אתה יכול לעשות חיפוש לינארי O(n) כדי למצוא את הצומת עם המרחק הקצר ביותר עד כה.

בחיים האמיתיים, זה יקר מדי – ביצועים $O(n^2)$ על פי מספר הקודקודים. אז יישומים ישתמשו ב-heap מינימלי שיביא לנו את המינימום בזמן הרבה יותר טוב O(log n). זה מביא אותנו ל-O(n log n) לפי מספר הצמתים.

אם תרצה את האתגר הנוסף, השתמש ב-heap מינימלי.

22.5 מה לגבי הפרויקט שלנו?

[כל כתובת ה-IP בפרויקט זה היא כתובת IPv4.]

הורד את קוד השלד כאן.

אוקי, אז זה היה הרבה מידע כללי.

מה זה מתורגם לפרויקט?

22.5.1 הפונקציה, הקלטים והפלטים

עליך ליישם את הפונקציה הזאת:

python
Copy code
```
:(def dijkstras_shortest_path(routers, src_ip, dest_ip
```

הקלטים של הפונקציה הם:

- routers: מילון המייצג את הגרף.
- src_ip: כתובת IP מקורית כמיתר נקודות ומספרים.
- dest_ip: כתובת IP יעד כמיתר נקודות ומספרים.

הפלט של הפונקציה הוא:

- מערך של מיתרים שמראה את כל כתובות ה-IP של הנתבים לאורך הנתיב.

הערה: אם כתובת ה-IP של המקור וכתובת ה-IP של היעד נמצאות באותה תת-רשת, החזר מערך ריק. לא יהיו מעורבים נתבים במקרה הזה.

הקוד להניע את הפונקציה שלך כבר כלול בקוד השלד למעלה. זה יפלט לקונסול שורות כאלה שמראות את המקור, היעד, וכל הנתבים בדרך:

python
Copy code
```
,'10.34.98.1' ,'10.34.46.1']    10.34.166.228 <- 10.34.46.25
['10.34.166.1'
```

22.5.2 ייצוג הגרף

מילון הגרף ב-routers נראה כך:

json
Copy code
```
}
"10.34.98.1": {
"connections": {
"10.34.166.1": {
"netmask": "/24",
"interface": "en0",
"ad": 70"
},{
"10.34.194.1": {
"netmask": "/24",
```

```
                    ,"interface": "en1"
                               ad": 93"
                                      ,{
                            } :"10.34.46.1"
                      ,"netmask": "/24"
                    ,"interface": "en2"
                               ad": 64"
                                      {
                                         ,{
                          ,"netmask": "/24"
                             ,if_count": 3"
                          "if_prefix": "en"
                                         ,{
                          # ... וכן הלאה ...
                                         {
```

המפתחות ברמה העליונה (למשל "10.34.98.1") הם כתובות ה-IP של הנתבים. אלו הם הקודקודים של הגרף.

עבור כל אחד מהם, יש רשימת "connections" שהיא הקשתות של הגרף.

בכל חיבור, יש שדה "ad" שהוא משקל הקשת.

"AD" הוא ראשי תיבות של Administrative Distance (מרחק מנהל). זהו משקל שנקבע ידנית או אוטומטית (או שילוב של שניהם) שמגדיר עד כמה יקר הוא קטע מסוים בנתיב. ברירת המחדל היא 110. מספרים גבוהים יותר הם יותר יקרים.

הנתון "netmask" עבור כתובת ה-IP של הנתב נמצא בשדה "netmask", וישנם שדות "netmask" נוספים עבור כל הנתבים המחוברים.

ה"שדה"interface" מציין איזה התקן רשת בנתב משמש כדי להגיע לנתב שכן. שדה זה אינו בשימוש בפרויקט הזה.

שדות "if_count" ו-"if_prefix" גם הם אינם בשימוש בפרויקט הזה.

22.6 קובץ קלט ודוגמת פלט

הקובץ של השלד כולל דוגמת קובץ קלט (example1.json) ופלט צפוי עבור אותו קובץ (example1_output.json).

22.7 רמזים

סמוך מאוד על פונקציות הרשת שכתבת בפרויקט הקודם!
הבן היטב את תיאור הפרויקט הזה לפני שתתחיל לתכנן!
תכנן כמה שיותר לפני שתכתוב קוד!