① 12/3/09
DB

נושאים מתקדמים במסדי נתונים

* השנה הדגש הוא מידע ברשת

הנושאים:

* רקע - איך האינטרנט עובד } ואז משקל את הלק הרחב בקורס
  - שימושים של הרשת
* מנועי חיפוש - אינדוקס, שאילתות, איסוף מידע וכו'
* XML

בדרך המרצה מקדישת זמן בקורסים שה לתת מוטיבציה ולהסביר למה בכלל יש לדון בנתחום. אבל ראן זה די ברור. אנחנו משתמשים באינטרנט כל הזמן, ההשפעה של האינטרנט עלינו היא עצומה. התומר מאוד חדש ומה שמאוד יפה הוא שזה תחום שמאוד משפיע על חיינו אבל הוא גם מתבסס מאוד הרבה על תומר תיאורטי שלמדנו באוניברסיטה. חוץ מזה יש גם המון מקום למחקר ויצירה.

תרגילים:

* אין מתרגל ואין בודק. המרצה מעבירה את התרגול
* ברוב השעורים יינתן תרגיל בית לשבוע.
התרגילים אופציונליים ועל תרגיל שמגישים מקבלים בונוס בציון הסופי (עומר ו.ו). אפשר גם להגיש אר הפתרון בכיתה ואת זה מקבלים אותנו בנקודה. סהכ אפשר לבור עד יום בעל לגבי התרגילים.

לא כל מה שנלמד בקורס מופיע במשקפים - הרבה פעמים יש משקפים שאלות שתתחות.

איך האינטרנט עובד?

המטרה הכללית היא להבין איך אינטרנט עובד ומה יכול אולי יכול לקרות באינטרנט.

מה קורה כשאתם שולחים לינק בדפדפן web? מאחורי הדף יש למעשה קוד html. ללינק יש שני חלקים - הידוע שזו הטקסט שאנחנו רואים על המסך. אבל כשאתם שולחים את הטקסט אנחנו רוצים להגיע לכתובת, אלא שזה רק שם. מה שבאמת צריך זה כתובת IP. כתובת זו אנחנו יכולים לקבל מה- DNS. הוא מקשר בין שמות טקסטואליים לכתובות IP.

הדפדפן רוצה לבקש את הדף מהשרת. שרת web היו תוכנה שיודעת להתייחס משאבים אבל בשביל זה צריך לדעת את ה- IP וה- port שעליו רשת מאזין. בדרך יש שימוש ב- port 80, אבל לפעמים יש הפניה למקום אחר. בדרך המתשמש מקונפג רק שהוא יודע מראש כמה IP-ים של DNS-ים. ה- DNS מקבל שם של שרת ומחזיר את הכתובת שלו. איך? ה- DNS מחזיק בתוכו כמה מיפויים ואם הוא לא מוצא את הכתובת הנתונה הוא יכול לפנות ל- DNS אחר ורק הלאה. בסוף הכתובת תוצרת למשתמש.

הדפדפן שולח חבילה עם כל הפרטים הנחוצים. הפרוטוקול שם שליחת החבילות נקרא IP.

אחרי שקיבלנו את ה- IP של השרת שרצינו צריך לשלוח לו בקשה למשאב. השרת מחלק את המשאב לחבילות קטנות ושולח אותו למשתמש. הפרוטוקול הזה נקרא TCP. הפרוטוקול הזה בודק שהמשאב אכן הגיע למשתמש ואם לא הוא שולח שוב. החבילות מגיעות למשתמש עם מספרים

סידוריים, המתשמ מסדר אותן לפי הסדר ומציג בדפדפן.

משאב זה פיסת אינפורמציה שיכולה להיות ע"י URL. זה יכול להיות קובץ (טקסט, תמונה ...) או דף דינאמי. את שאנחנו רואים בדפדפן היום שלום של כמה משאבים - בקוד html של הדף יש הפניה למשאבים אחרים ואז הדפדפן צריך לבקש אותם מהשרת במיוחד. לכן הרבה פעמים רשת עולה דף web אנחנו רואים את הטקסט לפני התמונות.

הצורה הבסיסי של URL היא

protocol :// domain / path

ב- URL יכול להיות גם עוגן שאפשר לעשות לינק לאמצע הדף ולא להתחלתו. אפשר להוסיף פרמטרים שמשמשים בעיקר בדפים דינאמיים.

את הצורה הכללי היא

protocol :// domain / path # anchor ? parameters

עוד פרטים טכניים במצגת. וגם בוגמאות.

הערות:

* רשתות עוגן ב- URL מקבלים את אותו המשאב כמו בלי העוגן. זו פשוט נקודת כניסה שונה אבל זה אותו המשאב.
* רשמבקשים URL עם פרמטרים אלו (קב) אשהו שונה ואולי לא - זה תלוי בשרת.

לינקים יחסיים -

אפשר ב- html לא לרשום את הכתובת המלאה אלא רק path ואז הדפדפן מניח שזה ביחס למקום שהוא עכשיו נמצא בו. מה שהוא עושה זה להחליף את מה שאחרי ה- slash האחרון ב- path.

הערות:

* ה-URL מגדה באופן ייחודי משאב. בהינתן URL אפשר להגיע רק למשאב אחד.
* לאותו משאב יהיו להיות כמה URL-ים שונים. למשל
www.notes-heaven.com
www.notes-heaven.com/index.html

זה נראה כמו ענין מחשבתי אבל זה יוצר בעיות למנועי חיפוש. למשל google אומרים שדף הוא חשוב אם הרבה דפים אחרים מלבינים אליו. אם ה-URL-ים שונים זו פוגעת בחישוב הזה.

דפים דינאמיים: עד עכשיו הנחנו שהשרת מקבל בקשה, ניגש למערכת הקבצים ומחזיר את מה שצריך.
אבל יש גם דפים שמורים במערכת הקבצים. למשל, תוכנות חיפוש לא שמורות ורשת amazon כתבים לנו
hello dina
הם בודאי לא שומרים באופן מיוחד לכל משתמש.
יש כמה מיני טכנולוגיות שונות וכדי לייצר דפים דינאמיים בצד השרת. יהיים יכם להיות דפים דינאמיים בצד הלקוח - זה כשהדפדפן מסוגל לעשות זאת. למשל, בדיקת תקינות שקרה זה משהו שהדפדפן יכול לעשות אלא דרוש לתקשר עם השרת בשביל זה. או למשל כמה מיני תפריטים שנפתחים וכו' זה יכול בצד הלקוח.
כמובן, אפשר גם לחבר בין שתי הגישות.

דוגמה: counter שסופר נכנסים לדף web הוא אמור בצד השרת- רק הוא יודע כמה אנשים נכנסו.
- חיפוש מחשבות בצד של השרת.
- באופן כללי אם צריך לגשת לאינטרנט מידע באמצע נתונים זה חייב להיות בצד השרת.

היתרון בלהריץ דברים בצד הלקוח הוא שצה מוריד עומס מהשרת אבל החיסרון הוא שיש לקוחות שלא מאפשרים למשל להריץ אצלם javascript (משיקולי security) ואז בעדם אי אפשר לתת להם שירות.

איך השרת מתקשר עם המתשמש שלנו? הפרוטוקול נקרא http. הדפדפן יודע לשוחח בקשר http והשרת web מאזין לבקשות ויודע לענות. אנחנו נדבר על http 1.0.

השלבים העיקריים הם:

- הלקוח פותח ערוץ תקשורת
- הלקוח שולח בקשה
- השרת שמאזין מקבל את הבקשה
- השרת שולח תשובה
- הערוץ נסגר

הפרוטוקול הזה stateless - עומד אתרגע שהבקשה נענתה הוא שוכח שבכלל הייתה תקשורת. לכן, אם אנחנו רוצים איזושהי דרך לצבור מידע אי אפשר להשתמש בפרוטוקול וצריך למצוא דרך תבנותית אחרת. למשל - cookies.

הפרוטוקול עם דיאלוגים נמצא במצב. אבל בגדול יש לו מיני הקדמות - headers וגם גוף אופציונלי. למשל, יכול להיות בבקשה השם מאסר הדפדפן. זה טוב כי אולי יש מאגר שנור לאתר עמוד בהתאם לדפדפן, אבל במובן הכי הוא ה- URL הוא לגמרי ציבורי חד-ערכי של המשאב. עוד דוגמא של headers במצב.

אפשר להפעיל פעולות שונות על הרתובת (get, put, delete וכו'). במובן הטהור בדב"ר זו יאפשר לנו לעשות כל מיני דברים מוזרים.

ויהיו להיות שנרצה לשנות את השדות של ה- headers כאופן גדול:

- cookies
- referer - למשל אם המפנה מקבל רק על הפניה אחרי נרצה לשנות אותו.
- user-agent - לראות את היתרון השונה לדפדפנים שונים

לתשובה שם השרת יש מהנה דומה - הקדמות ואז המשאב.
למשל יש header שאומר את אורך המשאב שזה תשובה לדי שהדפדפן יידע שאתן קיבלנו את הכל ותהיה לו סוף המשאב
כדי שהדפדפן יידע איך להציג אותו.
באלגוריתם רשימת של הקבצים שמותרים ע"י השרת.

תופעה מעניינת: רשתות ללא -
www.cs.huji.ac.il/~jeff
הדפדפן משנה את הכתובת ל-
www.cs.huji.ac.il/~jeff/
מה שקורה הוא שה שרת web אומר לדפדפן שהכתובת של האתר השתנתה ואז הדפדפן מבקש שוב ומקבל את הכתובת החדשה עם ה- slash
זה טוב בגלל הלינקים היחסיים. אם יש לנו לינקים יחסים בלי והכתובת הייתה נשארת בלי ה- slash אז הם לא היו עובדים!

לפי מה שאמרתי עד עכשיו נראה כ משאב הדפדפן שולח בקשה לשרת. זה כמובן לא היעיל כי לפעמים נכנסים אלינו פעמים לאותו מקום. למשל ל-google יש סתם קטן ותמוך. הבל כ ל דקות להביא אותו אתרש. אז בשביל לחסוך בתעבורה באינטרנט עושים caching. אפשר לעשות את זה מקומית של המחשב או באמצעות proxy שעושה caching בתוכו. כמובן, יש היררכיה של proxy-ים.

היתרונות של caching הם ברורים. הבעיה העיקרית היא שאנחנו עלולים לקבל מידע לא עדכני.
יש עוד כמה אני שימושים לשרת proxy - למנוע גישה לדפים מסוימים, לקלף אתרי עבודים, להגן מפני ווירוסים.

תוכנת proxy המיוסמת מאוצנה לגישה port ואז בלואר אינטרפיר מקבלת בקשות ועונה להן. באלגר יש שתי דוגמאות למימושים פשוטים בjava.

עוגיות:
זה איזשהו מנגנון שמאפשר לשרת ללקוח לשמור אם עתל בקשות ארוך - משהו משותל שנראה לנו כמו בקשות לאותם מקום.
כשאנחנו שולחים בקשה השרת יכול לבקש מאיתנו לעדכן עוגיה במחשב. העוגיה נשמרת אצלנו במחשב במערכת הקבצים.
עוגיה היא למעשה אוסף משתנים עם ערכים. לכל אתר יש עוגיות משלו, לפי המידע הרלוונטי. ואז כשאנחנו פונים לאתר שוב אנחנו שולחים לו את ערכי העוגיה רק נוכלת בקשות אורכבר טוות.

הערות:
* בשיטה יכולה להכיל כמה עוגיות
* עוגיה עם אותו שם דורסת עוגיה קודמת

אחת ש- http הוא stateless, אבל כל לאתרים שצריכים דברים. למשל, סל קניות ב- amazon. הרעיון הוא שהשרת והלקוח שלחים איזה פיס מידע ייחודי לכל אורך ה- session. את זה אפשר לעשות בעזרת עוגיות (אבל יש עם דרכים אחרות). העוגיות האלה פעמים תיקל ברגע שסוגרים את הדפדפן. המידע עדיין נשמר בדב רק בלב השרת והעוגיה מכילה אז שהיו מפתח למידע הזה.

session יכול להיגמר באחת משתי דרכים:

- סוגרים את הדפדפן. במקרה זה השרת לא יודע שה-session נגמר והוא ממשיך לשמור אצלו את המידע למשך זמן מסוים שנקבע מראש (expiration) או נשלחים לו logout או משהו כזה.

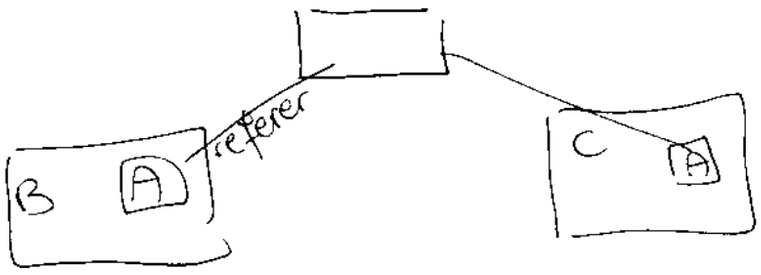
האם אזהרה זה בטוח? אזהרה זה סתם טקסט. הוא לא יכול להרוס וירוס. אולי רשום שם קוד נגדי, אבל זה לא יחל לעזור. אזהרות לא יכולות לבדוק את הקוד בדפדפן כי הפעלה של אבטחה ובמות האזהרות.

אבל, אפשר יכולות להכיל מידע סודי. למשל יכול להיות שמספר כרטיס האשראי שלנו מופיע באזהרה. בכל אופן, אזהרה יכולה להכיל רק מידע סודי שאותנו סיפקנו.

מניעת אזהרות: יש תקשורת בין שרת לדפדפן ויש packet sniffer שיושב על אחת התקשורת. אבל אם הוא רואה את תוכן האזהרה אולי הוא יכול להתחבר לי. אבל אפשר לבדוק...

הרצת אזהרות: יש שרת ומתקיים שמכוונה ובנדון שולח באזהרה לכם לא נכונים. למשל אם יש אתר ששולח אליכם את שלום באזהרה - אפשר לשנות את הת שלום...

אזהרות בלי שימוש - בעזרת אזהרות אתרים יכולים לגלות באילו אתרים אתרים היינו. למשל, באתרים עם פרסומות הפרסומות מבקש לא ישמור באותו שרת של האתר אלא באיזה שרת אחר. בשביל מה שה תמונה של הפרסומת נשלחת אותה ה- referer.



(5)

השפה שמאפשרת הפרסומת יכולה לשלוח לנו עוגיה וכך
אם נישנו אל אותה הפרסומת דרך כמה אתרים שונים
השרת של הפרסומת יכולה לדעת את זה.
אבל יש דרך בדפדפן למנוע third-party cookies.

⑥ 19/03/09
DB

שמוש שלה למדנו את הטכנולוגיות הבסיסיות שבלעדיהן קובץ האינטרנט. נשאר לנו להשלים קטע קטן של אמיתה באינטרנט.
כולנו מכירים את אתר המידע האישי. סביר שאנחנו לא נרצה שכולם יוכלו לראות המידע האישי שלנו (משיקולי בטיחות ומובן) נצטרך שלוגיות מסוימות לנו תהליך תקשורת בטוח ומוכרן. הבעיה היא שלהצפיר את מידע ההצפנות הלה בלעדיה זו לא מאובטח. כל מי שמאזין לתקשורת יכול לראות את תוכן הלוגיה ולהתחזות אלינו. נרצה גם לוודא שאף אחד לא יתחזה לשרת. למשל, לה יכול להיות מאוד חכוץ את מישהו מתחזה לאתר הבנק שלנו.
הבעיה הבסיסית היא שאנחנו רוצים להגביל את הגישה לאתרים מסוימים. יש כמה שאלות:
- איזה דפים אנחנו רוצים להגביל?
- מי יכול לגשת בכל מקום?
- איך לבדוק את הזהות המתחברים?
- האם צריך לבטיח את המידע?
יש כמה אסטרטגיות
- programmatic security - מי שכתב את התוכנית מממש את הלוגיקה של האמיתה. היתרון של זה הוא שזה מאוד גמיש - ממש הכל אפשרי. אבל הבעיה הגדולה היא שצריך לכתוב הרבה קוד מסובך.
- declarative security - אפשר להגדיר למי מותר לגשת לאתר והם HTTP דואגים לישם את זה. זה מאוד פשוט אבל מאוד מוגבל, כי כל מה שיש לנו זה אוסף של קשרים בין הדפים למשתמשים המותרים שלהם.
אנחנו נדבר רק על סוג זה של אמיתה.

מחלקה (Realm) זה אוסף דפים עם שם. לכל מחלקה

מגדירים אופן משתמשים וסיסמאות. האופן שבו מגדירים
את זה תלוי בשרת. הרעיון הבסיסי היה שרשת משתמש
מבקש משאב מאוך מאלכה היה שולח שם משתמש וסיסמה.
יש פרוטוקולים שונים לניהול של המעבר של שם המשתמש
והסיסמה. ב- http יש שני פרוטוקולים:

- Basic
  - המשתמש מבקש משאב
  - השרת שולח תגובה שהבקשה נדחתה ושלוך לאמת את המשתמש
  - הדפדפן שומר תלוי ומבקש שם משתמש וסיסמה לאלכה המבוקשת (יכולות להיות רמה מאלכות)
  - המשתמש מכניס את הפרטים המבוקשים
  - השרת בודק את הנכונות ומחזיר תשובה בהתאם.

במהלך כל ה session של התקשורת, הדפדפן שומר
את שם המשתמש והסיסמה של המאלכה הזאת והיא שולחת
אותם באופן אוטומטי אם צריך. זה חוסך לנו זבוזה
זמן. הדפדפנים גם יכולים לשמור את הסיסמאות
לתמיד אם נמקש מאנו.

בפרוטוקול הזה מה שנשלח הוא username:password
וזה נשלח בבסיס 64. זה לא הצפנה. זה ממש נשלח
חשוף לכל הרשתות. וזאת הבעיה העיקרית של השיטה הזאת!
כל מי שמאזין לתקשורת או עוסק בדרך יכול לגלות את
הסיסמה.

- Digest - כאן המידע נשלח מוצפן וזה פותר את
הבעיה העיקרית של השיטה הקודמת.

עדיין יש מאלכות וכל אלו שאת הסיסמה שולחים אותה
שמפעילים עליה פונקציה לרבות חד-כיוונית (כמו MD5).
הרעיון היה שקל לוודא את הסיסמה אבל קשה לשחזר אותה
מה ערכן hash שלה.

השרת מקבל את שם המשתמש וה- hash ומוצא
שה- hash של הסיסמה ששמורה אצלו מתאים למה
שהגיע בתוך שורה.

אבל, למרבה הצער כפתרון הזה שמדבר, כי אפשר להאזין
ל- hash ולדעת להתחזות לנו (Replay attack).
בשביל להתגבר על זה משתמשים במשתנה שנקרא nonce.
השרת שולח לנו דרך אקראי חדש והמשתמש צריך
להחזיר את ה hash של user:password:nonce
וכה מונע את ה- replay. כי מי שרוצה להתחזות לנו
כמה יקבל מהשרת nonce שונה ועל דרך ה hash
הישן לא יהיה תקף יותר.

אבל אף אחד לא יכול להתחזות ללקוח או לגלות את הסיסמה
שלו אבל לדעתי מישהו יכול להתחזות לשרת. בשביל
למנוע את זה גם הלקוח שולח nonce משלו, והשרת
צריך להחזיר את ה- hash של user:password:nonce:resoure-content

רק אנחנו גם בטוחים שעל אחד לא הגיל את התוכן של הדף.
אחרי לדעתי הבעיה כבר אי שאפשר להקשות יכול לראות
את המידע הסודי שלנו. בשביל זה צריך להצפין את המידע.
זה נעשה באמצעות פרוטוקול SSL (ר' קורס באמצעות
מידע).

לדעתי יש לנו הבנה הסיסטמי אל אם עובד האינטרנט.

Website Usability

זה לא חלק אינטרנטי מהחומר של הקורס אבל אולי חלק
מאיתנו נבנה אתרי web וכחלק מעבודות עבודה נצטרך
איך לעשות אתרים טובים.

הרבה מאוד אתרים מושקעים ומתוחכמים יכולים להיות מאוד יעילים מבחינת נוחות השימוש. למשל אתר לאנות לאבהברה לא מושקע. למשל - www.brillpublications.com -
אתר מאוד מושקע אבל בכל לא נוח...
עוד דוגמה היא השלטים הנוראיים של HOT או של YES,
שלא דבר כל זה שום אחד רוצים להקליט תוכנית כי דאו
צריך לקרוא את כל הוראות ההוראות כמה פעמים כדי להבין מה
לעשות. בעולם הפיסי אנחנו קונים מוצר ורק אז מסלים שאולו
לא נוח. אבל ב- web זה ממש קריטי: אף אחד בכל
לא ישתמש במוצר שלנו אם האתר לא נוח מאוד ברור. מה
קורה שם.

מה זה שימושיות?

- learnability - כמה קל להבין איך משתמשים
- efficiency - כמה מהר הדברים קורים
- memorability - כמה קל לזכור איך משתמשים
- errors - כמה שגיאות משתמשים עושים
- satisfaction - כמה נעים השימוש
- utility - האם האתר אכן עושה מה שצריך

מתקדם מראים ש-יוסף מכל האתרים לא לוקחים בחשבון
שיקולי שימושיות כשמתכננים אותם. הנה עוד דוגמה לאתר
מאוד מאוד יפה - האתר הישן של מעו של בנק. הדוגמה
מעולה.

איזה כלים דרך לקחת בה שמין כשמתכננים אתר?

- עיצוב הדפים - אומלל שכל הדפים יהיו בנויים לפי איזה
template - זה עוזר למשתמש להבין איך לעבוד דברים.
כדאי שהקונטרסט יהיה טוב, שהרקע לא יהיה רועש, פונטים
סטנדרטיים. בכל - סטנדרטי זה טוב - כולם כבר מכירים.

הגודל של הדף חשוב! אנשים לא אוהבים לרדת למטה, אבל לגלול ימינה ושמאלה זו מלא הקטסטרופה. אז דרוש לבאים שהאתר יתאים גם למסכים הקטנים יותר.
הה מלדה היא לבתוכה מלא - אנשים קוראים יותר לאט ממסך והם גם שמים לב בדיוק רק לחלקים המודגשים. את המידע הכי חשוב דרוש לשים למעלה.
הכל הכי חשוב הוא התוכן - מה שמחדיר אנשים לאתר זה התוכן המעניין!

- עיצוב מאוד הבית - זה הדבר הראשון שהמבקרים רואים. במלבן יש פירוט של תוקרת השטח של המסך לתכנים. מה דריך להיות בעיצוב הבית?
  - איך שפה משפט שמתאר את מה שהאתר עושה.
  - הכותרת (מה שמופיע בדפדפן למעלה) דרוש להיות בעלת משמעות
  - אולי אפשרות חיפוש
  - להדגיש דברים חשובים
  - splash screen זה לא טוב!
- מבנה האתר - ה- navigation זה הדבר הכי חשוב. זה דריך להיות נוח! מחקרים מראים שהרבה פעמים אנשים מתיאשים ויוצאים מהאתר לפני שהם אוצאים את מה שדריך. חשוב גם שבכל שלב המשתמש ידע איך לחזור לעמוד הבית! ואם יש תגובה של הליקים שעברנו זה עוזר למשתמש להבין איפה הוא נמצא.
מהליקים דריך להיות ברור לאן שותנו מגיעים. דוגמה טובה למה לא לעשות אפשר לראות ב www.cuhza.com או ב- www.zaha-hadid.com.
- נגישות לאנשים עם מוגבלות. יש אופן קבוע WAI:
  - האתרים דרוים להיות נגישים לדפדפנים ישנים
  - האתרים דרוים להיות נגישים לעיוורים

- האתרים צריכים להיות נגישים ל- mobile devices
כל מיני טכנולוגיות מתקדמות כמו Flash או
JavaScript יכולים ליצור בעיות נגישות לדפדפנים
ישנים או למכשירים ניידים.
צריכת להיות אפשרות להגדיל את הגופנים - זה
חשוב לאוטוסיה והנגישות.
עדיף שרשמות של הלינקים יהיו אינדיקטיביים לתוכן.
רצוי להדגיש את האתר שלנו נגיש אפשר לראות איך הוא
נראה עם דפדפנים שונים ויש גם כל מיני כלים
שאפשר להוריד.

אז בנינו אתר מדהים. צריך לעשות לו בדיקות!!!
צריך לעשות בדיקות בשלבים הראשונים, לפני הפרסום
ואחרי הפרסום.
סוגים שונים של בדיקות:
- איפוש אצורים של בלבול
- לבדוק שאפשר לעשות אשימה בזמן מסוים
- השוואה בין אתרים שונים עם תוכן דומה.

בעיקר הבדיקות הוא פשוט - לוקחים לקוחות פוטנציאליים
ונותנים להם משימות ומסתכלים איך הם עומדו עליהם
ומתקשים משהו.

באלגת יש לינק לאתר שיש בו רשימה ממלה של הדברים
שנבאי לבקוק כשבונים אתר.

השית באופן סטנדרטי עושה logs של מה שקורה באתר
ואז אפשר לנתח אותם ולבדוק כל מיני דברים - זמן הביקור,
הדפדפן, מסלול לינקים וכו'. זו מאפשר לנו לראות
מידע על המשתמשים באתר ואולי אפשר אז להתאים
להם את האתר בצורה טובה יותר.

תרגול בית: להחזיר אתר ידוע ולנתח 5 בעיות שיש באתר ולהציע פתרונות.

נכנסים לחלק הכיפי יותר של הקורס

מנועי חיפוש

תופעת מנועי החיפוש כמו - google בפרט מאמש הרעידה את העולם. למעשה יש השערה שגוגל זה אלוהים (וכנראה יש דיון אם אולי גוגל זה השטן) המטרה שלנו תהיה להבין איך זה קורה שגוגל יודע הכל ומראה לנו מידע כ"כ מהר.

אנחנו רוצים לבנות מנוע חיפוש. מהם הקשיים העיקריים שצריך להתמודד איתם?

- כמות המידע
- אחזור העדכון של המידע
- כימות הרלוונטיות, דירוג
- אינדוקס המידע
- מציאת המידע מלכתחילה
- התמודדות עם סוגים שונים של מדיה
- האנשים הרעים שמנסים להטות את מנוע החיפוש (spam)
- כיסוי של המידע

מנוע החיפוש מתקיים משהו שנקרא index repository שמכיל מידע על הדפים וגם יש crawler ו- indexer שם הכל רצים ברקע, מחפשים מידע ומכניסים ל-repository. קומפוננטות נוספות הן

- query processor - מנתח שאילתות
- ranker - מדרג תוצאות

האתגר הוא איך שאי לתפוס את כל הדברים שקורים
בסביבה לאן.

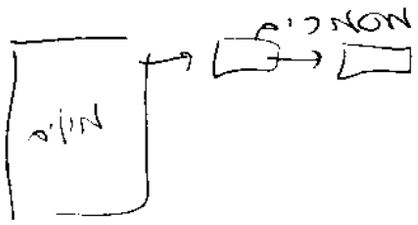
בשביל הפס (דבר ב) ה- index repository.

(10) 26/3/09
מבני נתונים

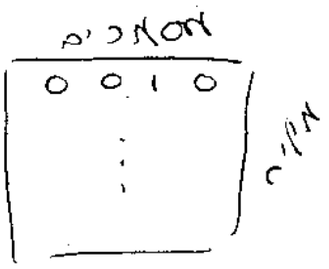
שבוע הבא לא תהיה הרצאה רגילה ובמקום זה יהיה באתר לינק להרצאות של אנשים בגוגל שאנחנו מתבקשים להקשיב להן.
ההרצאה הנ"ל חומר אינטרנטי בקורס.

היום נתחיל לדבר על storage - איך שומרים מידע? נתחיל במקרה הפשוט שיש לנו רק טקסט (לא תמונות וכו') אבל מאוד מהר הרבה מאוד ונרצה לתשוב איך לשומר אותו כדי שאפשר יהיה לעשות חיפוש יעיל.
קודם נראה כמה דברים מעניינים לסטטיסטיקות לא טובים כדי שנפתח אינטואיציה למה שכן תשוב.
נגדיר את הבעיה בצורה מדויקת: יש אוסף מסמכים ואוסף מילים ואנחנו רוצים למצוא את כל המסמכים שמכילים את כל המילים. מה נרצה לקחת בחשבון? ברור שנרצה שזה יהיה כמה שיותר מהר. מצד שני, בדרך יש tradeoff בין זיכרון לזמן. כמובן, צריך לקחת בחשבון שבכל שהשאילתא מסובכת יותר זה בא על חשבון זמן וזיכרון. למשל, לחפש "hello world" זה יותר מסובך מלחפש את המילים בנפרד.
בכל אופן, נרשיו נדון בבעיה הפשוטה של מסמכים ומילים.
כמה רעיונות:

- hash table



- bit map



- מסד נתונים

• סתם קבצי טקסט ומהיונתן שאותם מחפשים בהם, למשל בעזרת grep שפשוט תכנה מאוד יעילה ומהירה.

כדי להבין את הבעיה בכל שתרון כדאי להבין כמה זמן לוקח למחשב לעשות כל מיני פעולות. במציאות יש פירוט אבל מה שבולט הוא שאי אפשר לשמור את כל הנתונים בזיכרון הראשי כי הוא לא מספיק גדול ומצד שני הזמן שלוקח לגשת לדיסק הוא הרבה יותר גדול מפעולות עיבוד שאנחנו רוצים לעשות. אבל נראה שכמה שיותר נתונים ישבו בזיכרון הראשי.

<u>פתרון 1</u>: לשמור את הקבצים כמו שהם ולחפש בהם עם כלי כמו grep. פה יש שני חלקים. מצד אחד, צריך לפתוח את הקבצים - להעביר אותם לזיכרון ואז לעבד אותם.

גניח שיש שני קבצים בגודל 5MB שיושבים בדיסק בזיכרון.

seek time - $5ms = 5\times10^{-3}s$

transfer time - $2\cdot10^{-8}\cdot10^{7} = 0.25\,s$

זה רק בשביל להביא את הקבצים לזיכרון.

אם הקבצים לא נכנסים בזיכרון והוא די גדול ובמצב הרבה יותר גרוע.

אז מבחינת זמן זה לא יעיל ומבחינת זיכרון זה תלוי באפליקציה.

<u>פתרון 2</u>: לשמור במסד נתונים

יש כמה דרכים להשתמש במסד נתונים.

DocID	Doc

בדרך הכי פשוטה היא פשוט טבלה אז להשתמש בשאילתות sql. (בעזרת LIKE).

מבחינה רעיונית זה נהיה לשימוש כ- grep אבל שהקבצים זה יותר גרוע כי מסד נתונים לא באמת מיועד לזה אלא

מאופטמת לחיפושים כאלה.

אופציה שניה היא לשמור גם את | DocID | word |

ואופציה שלישית היא לשמור שתי טבלאות של גוגל:

word index	appears
word | wid	DocId | wid

מה עדיף? מבחינת זיכרון האופציה השלישית טובה יותר כי סביר שה wordId תופס פחות מקום מהמילים עצמן ומבחינת זמן, אחרי שמצאנו את ה- wid אנחנו צריכים לעשות רק השאילתא מספרית וזה מהיר יותר. חוץ מזה, היקטנת המקום משפרת את הזמנים של השאלות הנתונים מהדיסק.

נשים לב שזה גם חוסך מקום לגוגל האופציה זאת הקבצים כבר לא מלה בסוף נשארת רק פעם אחת והרי טבעי שבמסמכים עצמם המילים תוצרות של עצמן.

מה לגבי חישוב השאילתא? אם מסמכים את השאילתא מוסיפים עוד מילה ומעדכן אם מוסיפים שלילה הזמן יכול להגדגר די מהר ואולי בקבצים זה די יעיל.

לסיכום, הפתרון של מסד נתונים מתנהג יותר טוב כשיש הרבה מאוד מידע כי את השיפורים של הזיכרון משפיעים יותר, אבל אם יש יתרון מעט מידע עדיף כנר לשמור את הקבצים וכהו.

ובכל זאת לפתרון המסד יש חסרונות - אם יש מילה שמופיעה בהרבה מאוד מסמכים זה אומר שמומרים את ה- wid שלה הרבה מאוד פעמים ודרוש לגשת להרבה שורות. וכמובן יש את הבעיה שזה לא תומך בעילות בשאילתות עם הרבה מלים.

פתרון 3: Bitmaps

הפתרון הבדוק הוא שפעולות על ביטים מאומשות במתשבה בזכרון מסוגל יעילה.

למשל, אם רוצים למצוא rain and pain נעשה and על השורות המתאימות. מסוגל פשוט ויעיל.

אבל מה לגבי הציבור? ענית שיש 500,000 אולים שונים ו- 1000000 מסמכים, ובכל אחד מהם 1000 אולים. אז ב- 1% מהטבלה יש ערכים. אז זה בבזבוז מקום מטורף!!! כל זה מביא אותנו ל-

Repository Index

יש שני מבני נתונים:

- לקסיקון - אוסף כל האולים במסמכים וכל אולה יש פוינטר ל- inverted index שלה
- inverted index - לכל אולה בלקסיקון יש רשימה של פוינטרים לכל המופעים שלה במסמכים

נשמע מסוגל פשוט. אם רוצים למצוא מסמכים שיש בהם שתי אולים צריך להתיק את הרשימות. אבל,

The Devil is in the Details

הלקסיקון בשלב הבא ענית שאולות מסובכות ובכל פעם מתפשים פשוט מלה אחת. המטרה היא למנוע גישה יקרה ואז קטן- רק שאולי נצליח לדחוס אותו לזיכרון הראשי. לכל אולה נרצה לשמור את הפוינטר לרשימה שלה ואת אורך הרשימה. הרעיון לשמש לכל מיני דברים (למשל דירוג) אבל כרגע זה פשוט מספיק לנו לדעת מתי לעצור את קריאות הרשימה.

אלתנו נניח ש- יש 1M מילים

כל מילה באורך לכל היותר 20

מילה מוגדרת באורך 8

כל תו זה 1byte

כתובת בזיכרון זה 4 byte

ערך ה frequency 4 byte

אופציה 1: לשמור את הלקסיקון בשרשור ~~#~~ של מחרוזות שוות אורך. אז כל רשומה תהיה באורך $20\cdot 4+4+4$

ואז סה"כ זה $1M(20\cdot 4+4+4)=28M$

זה בהנחה כי מילה מוגדרת היטב באורך 8 ...

אופציה 2: אפשר לשמור את כל המילים ב- string אחד ארוך ולשמור פוינטרים לתוך ה string הזה שמסמל איפה כל מילה מתחילה. זה יתן:

$1M(8+4+4+4)=20MB$

אז תסכון 8M

אופציה 3: יתרון זה יהיה המילים עם אותה תחילית (prefix) ואז נוכל לשמור אותה פעם אחת. זה מאפשר להשתמש בטכניקה שנקרא Front Coding.

בהנחה שתחילית ממוצעת משותפת היא באורך 3 נקבל

$1M(5+4+4+1+4)=18M$

הבעיה היא שבטכניקה הזאת אי אפשר לעשות חיפוש בינארי.

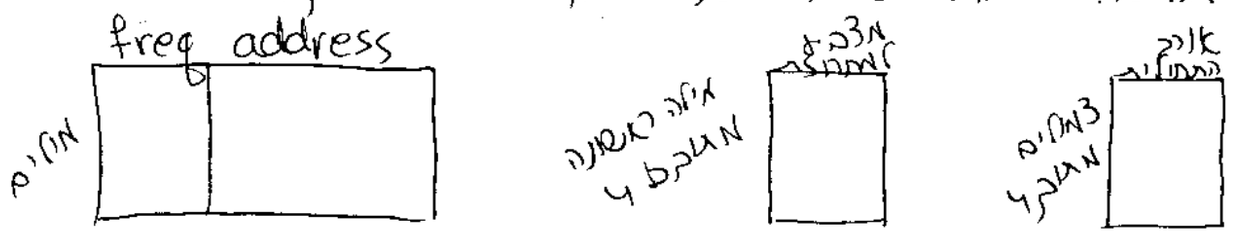
אופציה 4: Frond Coding 4-in-3 - בכל בלוק של 4 מילים הראשונה נתונה בשלמותה ושאר האחרות מקודדות. למשל

jezebel	jeziel
jezer	jezliah
jezerit	jezoar
jezrah	jezrahiah

אז יש string ארוך

7jezebel 1r 2itiah 6jeziel 4liah 3oarrahiah

חוץ מהמילה הראשונה לשלוש האחרונות מותקיבו את התחילית המשותפת עם המילה הקודמת. ולכל המילים פרט לאחרונה כתבים את אורך ה- suffix שנשאר. למילה הראשונה אורך ה suffix הוא פשוט אורך המילה. וכך לכל 4 מילים. ברור שצה בהפני עצמן לא מספיק. אז יש עם



איך עושים פה חיפוש? אפשר לעשות חיפוש בינארי על המילים ואחרי שמוצאים מילים - אחרי עדין נשארים עם סיבוכיות לוגריתמית. החיפוש באיך המילים הוא פשוט כי יש לנו את כל המידע הדרוש.

מה אבעיה לה חיפוש? לכל מילה שומרים מספר מופעים והתחלת שלה $(4+4)\ M$

רבע מהמילים שומרים את כל המילה - $\frac{1}{4} M \cdot 8$

לשאר המילים שומרים את ה suffix שאורכו הממוצע 5 -

$\frac{3}{4} M \cdot 5$

רבע מהמילים שומרים מצביע לאתרולא - $\frac{1}{4} M \cdot 4$

ל- $\frac{3}{4}$ מהמילים שומרים את אורך התחילית - $\frac{3}{4} M \cdot 1$

ול- $\frac{3}{4}$ מהמילים שומרים את אורך ה suffix $\frac{3}{4} M \cdot 1$

סה"כ לה יוצא $16.25\ M$ אז אפילו קלת חסכנו מקום אם הכנסנו את האובדה לעשות חיפוש יעיל.

עכ"ל יש מבנה נתונים בשם trie שלפעמים משתמשים בו בשביל לקסיקון. יש מאמר שמדבר עליהם. יש לקרוא אותו ולנתח שאלות מעניינות

ה- Inverted Index

לכל השומה מאינדקס נקראת posting list - זו רשימת המסמכים שבהם מופיעה המילה. אפשר גם לשמור את המקום במסמך שבו מופיעה המילה. למשל

2 2 4 3 1 2

המילה מופיעה במסמך 2 פעמיים

המילה מופיעה במסמך 3 פעם אחת

התועלת של המקום מתבטאת בזיכרון !!

אבל זה לדיון רשימה של מספרים.

אפשרות נוספת שהרשימה תכלול רק את מספרי המסמכים (בלי מיקום) וגם נניח שאין מונחים מהקטן לגדול - זה יעזור לנו בחישוב של שאילתות.

סביר שאינדקס מסוג גדול, אין סיכוי שהוא יכנס בזיכרון הבשר, אבל בכל זאת נרצה לשמור אותו בדיסק הכי יעיל שאפשר.

דרך אחת להקטין היא לא לשמור את המספרים עצמם אלא את ההפרשים ביניהם. אבל מאחר שההפרשים הם גדולים הוא מסדר גודל מספר המסמכים, אם משתמשים בfixed length כדי לייצג את המספרים זה לא חוסך מקום. אבל אם משתמשים בקידוד באורך משתנה זה יכול לחסוך!

סוגי דחיסה: מדובר על מתחלק לשני סוגים: מס פרמטרים ומלבי פרמטרים. למשל, פרמטר יכול להיות תדירות המופע של עניין בקבץ.

עוד חלוקה היא דחיסה אלומסולית או לוקאלית. בדחיסה אלומסלית כל רשימה נדחסת בפני עצמה ובדחיסה לוקאלית אפשר להשתמש בשיטות שונות לכמה רשימות שונות

נשים לב שאולי נפוצות שמופיעות בהרבה מאוד מסמכים
ההפרשים של הרשימה יהיו קטנים יחד. יש מספר אולם לא נפוצות
וכולם להיות הפרשים קטנים אם למסמכים יש איזשהו סדר
לוגי ביניהם.
זכן כמובן המטרה שלנו תהיה לקודד מספרים קטנים בספרות
ביטים!!

מה בכלל אפשר לכווץ? יש תוצאה מפורסמת של Shannon
שאומרת שהתוכן התתתון הוא $\sum_s -P(s)\log P(s)$
כאשר s זה הסמן! - $P(s)$ תדירות המופע שלו.
אבל יותר טוב מזה אי אפשר. ובהמשך אנתוני נראה שיטה
שממש מתקרבת לזה (arithmetic coding).

שיטת הקידוד הכי פשוטה היא קידוד אונרי - את $x>0$
מקודדים ע"י $x-1$ אחדות ואז 0. למשל

1	0
2	10
3	110
⋮	

את 0 אי אפשר לקודד רק וזה גם לא משנה לנו כי
אין לנו את 0 בתור הפרש בין מספרי מסמכים.
האם השיטה הזו יעילה לנו? מה ההסתברות של x אולו
קידוד זה היה אופטימלי? אם כן, אורך הקידוד של x הוא
x ולכן $x=-\log P(x)$, אז אם $P(x)=2^{-x}$ זה
אכן קידוד אופטימלי!

עוד שיטת קידוד היא γ-coding. משיטה זו x מיוצג ע"י
הקידוד האונרי של $\lfloor \log x \rfloor + 1$ ואחריו הקידוד הבינארי
באורך $\lfloor \log x \rfloor$ של $x-2^{\lfloor \log x \rfloor}$. למשל,

0	0
5	11001
9	1110001

איך מפענחים את זה? אותנו יודעים שההתחלה היא קידוד אונרי אז זו מסתיימת ב-0 הראשון. אבל אז המספר הזה בדיוק מגדיר את האורך של התדי השני של הקידוד של המספר הבא ואז מוציאים את המספר הראשון. ואז ממשיכים מאותו אופן. למשל, נניח ששלחנו 0111010011101111

הלוקח האונרי של המספר הבא היה 0111. שזה שווה ל- $1+\lfloor \log x \rfloor$. לכן האורך של התדי השני הבא 4 והוא 1001 בבינארית. עכשיו לפי הנוסחאות אפשר לפענח בדיוק את המספר הזה.

אלא, הקידודים האלה יותר יקרים מסתם קוד בינארי. אבל קוד בינארי אי אפשר לשפר סתם ככה ולפענח. למשל אם יש 111 איך נדע אם זה 11, 1 או 1, 11 או 111?

מה האורך של הייצוג של x בקידוד-γ? $1+2\lfloor \log x \rfloor$
ואם זה היה אופטימלי פירושו ש- $\log P(x) = 1+2\lfloor \log x \rfloor$
לכן (אם נזניח את הערך התחתון) $P(x) = \frac{1}{2x^2}$.

שיטת קידוד נוספת היא Simple Variable Length
לכל בית יש ביט אז אפשר להשתמש באחד מהם כדי להגיד אם זה הבית האחרון או לא ובשאר להשתמש כפי לקודד. זה נוטה להיות יקר כי המתשב עובד טוב עם בתים.
למשל, 256 בבינארית זה 100000000. אז נייצג אותו ע"י 000000 10 | 1 00000000

לציין אחרון

בית אחרון

אז כאמה מתנים דניק בשביל לידע את x? $\left\lceil \frac{\lceil \log x \rceil}{7} \right\rceil$

אז כאן שיטות ללא פרמטרים. יש שתי שיטות מעניינות עם פרמטרים - קוד האמן קנוני וקידוד ארית מטי...

(15) 23/4/09
DB

- מאמר הקורס יש רשימה של הנושאים שיעורי הבית. נא לוודא שהכל בסדר.
- מי שעדיין לא היה שייך להרצאות האלה מתבקש לעשות זאת.

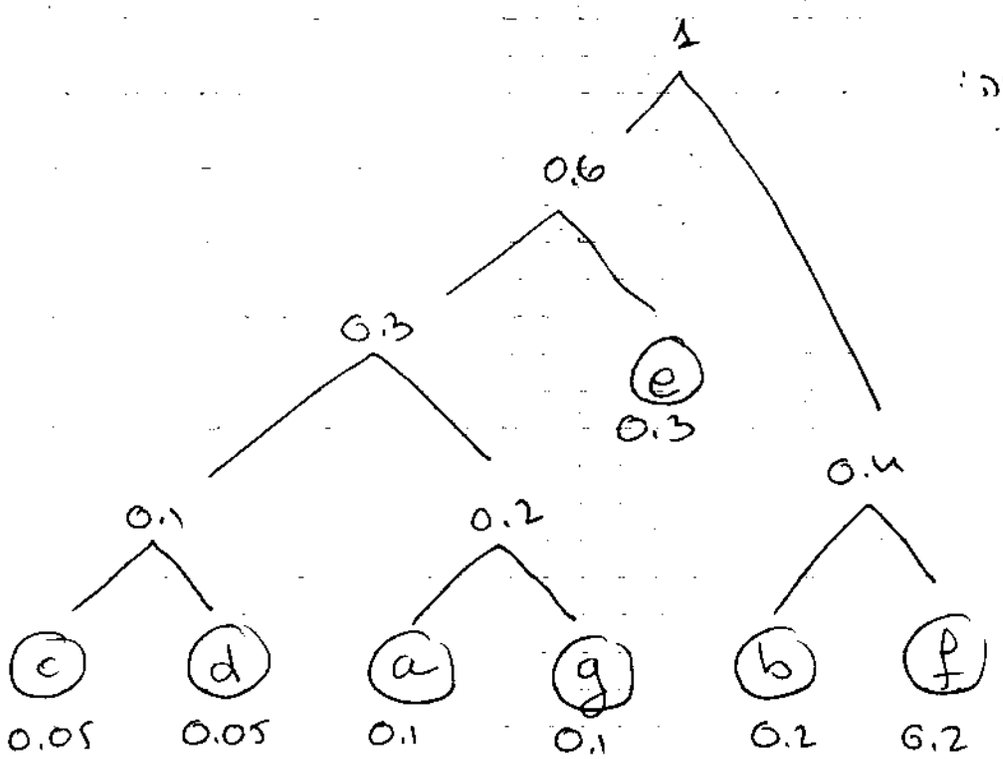
תזכורת לעניינו: אנחנו מנסים להבין איך נראה לשמור כמה גדלה של טקסט הקשור לבדוק חפוש באופן יעיל.
ראינו שאפשר לשמור אינדקס מהופך - מילים שמצביעות לרשימות המסמכים שהן מופיעות בהם.

קוד האפמן

היסטוריה מעניינת - דיוויד האפמן המציא את קוד האפמן כדי לא לעשות למהתן בקורס שעשה מתואר שני...

יש רשימת סמלים שכל אחד מהם נתונה הסתברות מופע. אנחנו רוצים למצוא prefix code - קידוד לסמלים כך שאין קוד שהוא רישא של קוד אחר (זה מאפשר לשמור פיענוח בקלות). כולנו מכירים את האלגוריתם של קוד האפמן. מישהו אלגוריתמאים רשאי לבין שקוד האפמן הוא אופטימלי באשר רוצים למזער קוד משלו.

דוגמה:



התיסרון הגדול של השיטה הזו הוא שמחייבים לשמור את הע"ל כדי לפענח את הקידוד וזה מנשל את הזיכרון שצריך. בשמקודדים טקסט אז יש 26 תווים וזה לא גדול, אבל בשביל אינדקס אולי גדלה של מספר מסמך יהיה סימבול ואז יש המון המון סמלים. הבעיה הבאמת גדולה היא שיכול להיות שאחרי כל רק בגדל שהוא לא נכנס בזיכרון הראשי...

קוד האצמן קנוני הוא קוד שצבורו לא צריך לשמור על כדי לפענח. הרעיון הוא שיש רשימת סמלים ונותן שמספרים לשמור לכל אחד אותם את אורך הקידוד של הסמל.

תכונות של הקוד:

- אם נתבונן במילות הקוד באותו אורך אז הן למעשה מספרים בינריים עוקבים.
- בהינתן שני סמלים s, s' בעלי מילות קוד באותו אורך אז $s < s' \Leftrightarrow cw(s) < cw(s')$, כלומר הסדר הפנימי של מילות הקוד מתאים לסדר הפנימי של הסמלים
- מילת הקוד הראשונה הקצרה ביותר היא מתרוצת של אפסים
- מילת הקוד האחרונה הארוכה ביותר היא מתרוצת של אחדות.
- נניח ש- d היא מילת הקוד האחרונה באורך i ונניח שהאורך הכי גבוה של מילת קוד כלשהי הוא j אז מילת הקוד הראשונה בעלת אורך j היא $c = 2^{j-i}(d+1)$ (כלומר מוסיפים ל- d 1 ואז דוחפים לו הכפה שאלה באורך ההפרש $j-i$)

דוגמה: רשימת סמלים עם אורכי הקידוד שלהם:

a	3	e	2
b	2	f	3
c	4	g	3
d	4		

אנחנו יודעים שהקוד הראשון הקל ביותר צריך להיות אפסים.
לכן b או e מקבלים 00. b < e
אם כן $\boxed{b - 00}$.
הקודים צריכים להיות עוקבים אם כן $\boxed{e - 01}$
האורך הבא הוא 3 ויש לנו את a, f, g (לפי הסדר).
לפי הנוסחה נקבל שהקידוד של a הוא $\boxed{a - 100}$
ואז f, g הם מספרים עוקבים $\boxed{f - 101}$
! $\boxed{g - 110}$.
עכשיו שוב לפי הנוסחה נקבל $\boxed{c - 1110}$
! $\boxed{d - 1111}$ שהוא גם סדרה של אחדות, כמו שרצינו.

איזה יופי! נתנו לנו את האותיות ופיענחנו איזה את הקוד.
אבל תשיפה מגיעים האורכים? יש לנו עוד צריכים מתוחכמות
להמציא אותם (לא לנו בחיים שרירותית של אורכים עובדת)
אבל הדרך הכי פשוטה היא לעשות קוד האפמן רגיל ואז
להשתמש באורכים שהתקבלו אמנם. (אנחנו לא נוכיח שזה עובד
אבל זה יוצא מאיזשהו נתון מתמטי של המבנה של העץ של
קוד האפמן).

טוב, אז מצאנו את טבלת הפיענוח אבל זה לא יכול בכלל!!
זה אומנם תופס המון מקום אולי אפשר אז כדי לפענח צריך לנו
פעם לעבור על כל הטבלה.
אז אנחנו נרצה לפענח בלי שנ לשמור את כל הטבלה.
בשביל לפענח נשתמש במידע הבא לכל אורך li שמופיע
באלגוריתם הקוד:

אורך	- - -
מילת הקוד הראשונה	- - -
כמה עלים יש באורך זה	- - -

2	3	4
00	100	1110
2	3	2

הבית אחרי שלנו

מתוך מצה רמזון צריכים לדעת את האורך של הקוד של כל סימן שיש לנו.

decoding

- $i=0$
- Repeat
  - $i=i+1$
  - let d be the word derived by reading l_i symbols

  until $d \le c_i + n_i - 1$
- Return the $d-c_i+1$-th symbol (in lexicographical order) of length l_i

דוגמה: נפענח את 1000110

l_i	2	3	4
c_i	00	100	1110
n_i	2	3	2

3	a
2	b
4	c
4	d
2	e
3	f
3	g

בהתחלה $i=1$ ואז $l_i=2$! $d=10$.

האם $d \le c_i + n_i - 1$ האם

$10 \overset{?}{\le} 00 + (10-1) = 01$

לא! לכן מאשרים $i=2$! $d=100$.

אז $100 \le 100 + 11 - 1$ אם אנחנו ארוגים!

100 אחר להתאים לסמל הראשון שהקידוד שלו באורך 3 שזה a. אז יש לנו 1110 שזה a

וכבר מאשרים. משל אפשר להגיע ל- aeg

שזה בדיוק מה שהיה יוצא אם היינו משתמשים בטבלה.

באמצעות טבלה להיות שאפשר ליצור קוד האפמן הנוני -
אז קודם מייצרים קוד האפמן ואז את הנוני

כדי לבנות קוד האפמן צריך לדעת את ההסתברות של הסמלים.
אם זה טקסט צריך לעבור שלב. ולספור תדירות.
אבל אנחנו רוצים את זה בשביל לקודד את ה inverted index:
והסמלים שלנו הם המספרים שמופיעים ברשימת שלנו.
יש כמה אפשרויות

- לפי רשימה לעשות סטטיסטיקה משלה ולקודד עם קוד אחר.
כאן הקידוד יצא טוב כי הסטטיסטיקות טובות ומדויקות ומותאמות
"מאופן אישי" אבל אז צריך לשמור המון נתונים לפי אחת
מהרשימות כדי לפענח.
- לעשות סטטיסטיקה של כל הרשימות ביחד - אז צריך לשמור רק
טבלה אחת בשביל הפענוח אבל הקידוד יצא פחות טוב.
- מה שאומרים שמקובל בספרות: לעשות סטטיסטיקות על רשימות
רשימות שהן ארוכות באותו אורך. האינטואיציה היא שברשימות
כאלה המספרים צריכים להיות דומים (נניח שהמספרים
שמופיעים הם הפרשים בין מספרי מסמכים. מספרי המסמכים
הכללי היא קבועה וחסום, אז אם יש לנו רשימות מאוד ארוכות
סביר שיופיעו בהן הרבה מספרים קטנים ...)

קוד האפמן לא טוב ויפה ואפילו אופטימלי. אם מקודדים כל סמל בנפרד,
אבל יש מקרים שבהם זה לא יעיל - למשל אם יש סמלים מאוד
מאוד נפוצים וסמלים מאוד מאוד מאוד נדירים.

<u>Arithmetic Coding</u> - הסברים בעמ'

<u>דוגמה:</u> נתונים שלושה סמלים עם הסתברויות:

A - 0.2 B - 0.3 C - 0.5

ורוצים לקודד את ACCB

ל סאם מדגדגם אותנו יתחום ויחסי שהוא מיצג לפי התדירות.

Symbol	low	high
	0	1
A	$0+1\cdot 0=0$	$0+1\cdot 0.2=0.2$
C	$0+0.2\cdot 0.5=0.1$	$0+0.2\cdot 1=0.2$
C	$0.1+0.1\cdot 0.5=0.15$	$0.1+0.1\cdot 1=0.2$
B	$0.15+0.05\cdot 0.2=0.16$	$0.15+0.05\cdot 0.5=0.175$

אנו מגדירים את הטווח והקידוד הוא איזשהו מספר בתוכו, לדוגמא 0.17. אנו בהינתן הקוד ואורך המילה שקיבלנו מאפשר לפענח.

הפיענוח פשוט. הרעיון הוא שלפי מנסים סמלים שונים ומודקים שהקידוד שיש לנו מתאים לטווח שמתקבל בינתיים.

נשים לב שרק שלא התמודדנו כאן עם הבעיה של מדיאר מספר בטווח מסוים שאפשר לקודד אותו בהמון דרכים הייתה בינתיים.

הקידוד הארתמטי מתקרב לחסם התחתון של שנון. האורך של הטווח הסופי הוא $\prod_{i=1}^{n} P(S_i)$ ומספר הביטים שצריך בשביל לקודד מספר בטווח הזה הוא:

בעיה: בשביל לפענח צריך לשמור את ההסתברויות שלכל הסימבולים וזה מוסיף לדרישות הזיכרון, אבל בעיה שיותר קשה אולי יותר הוא שלאורך מסוים אפשרויות יגדלו לה שמנו וזה לא משהו שהמודל שלנו מאפשר להתמודד איתו.

בשביל לפתור את הבעיות הללו אפשר להשתמש בקידוד אריתמטי אדפטיבי.

התהליך קומה ובכל שלב משלבים את ההסתברויות לפי השילוב שנו מחלקים את הטווחים. בהתחלה אנחנו שלבו התווים הסתברות לה וכל שקוראים תווים מעדכנים את ההסתברויות לפי הטקסט שקראנו עד כה. הרעש לא יפגע את ההסתברויות והן נוצרות on the fly כל שקוראים את הטקסט.

הפיענוח מדמים בדיוק את אותו התהליך - בהתחלה התווים שווי הסתברות ואז מתאימים לנתון. כמה שלב הטווחים צמצמים ומוצאים בטווח מהשלב הקודם עד אין לנו מעשה אפשרות לטעות בפענוח.

אינדקסים לגלישה ברשת ה- web

אנחנו הולכים לדבר על מאמר של Boldi & Vigna.
המטרה: לשמור על מבנה ה- web ועל גם כמובן ברמה שפחות מקום.

מדרון נאלמי: נניח שכל URL יש מספר יחודי. אז למשל ל- 411 מיליון דפים צריך 27 ביטים רק לקודד את המספר בקידוד בינארי קבוע.
אם נשמור פשוט את הרשימות בלבד אז צריך 64 ביטים לכל URL. אם בלי נשפר ונניח נשמור לו מאוד פשוט <u>הסימן</u> של הלינקים שמופיעים בו אז אפשר להוריד בחצי.
אבל במאמר של Boldi & Vigna הם הצליחו להוריד את מספר הביטים ללינק ל- 3 !!!

מבחנות מעניינות של לינקים:

- לוקליות - בדרך לינקים מובילים לדפים עם אותו ה- host אז אם URLs נשמרים לקסיקוגרפית אז המקום והמרחק בניהם יהיו קרובים.

- דימון - URLs שנמצאים לקסיקוגרפית סמוך שיהיו להם הרבה לינקים משותפים

- לוקליות - הרבה פעמים להרבה עמודים שמקושרים מעמוד משותף יש URLs לוקליים.

"גוגל נאומי":

node	out degree	successors

בשביל מה לדחוס לדעת את מספר הלינקים שיוצאים? וגם לדעת את הרשימה של ה successors עצמה?

זה לה כמעט קשה לדחוס ביעילות. אבל השפור הראשון הוא לשמור את ה successors ברשימת הפרשים

$$S(x) = \{ s_1 - x, s_2 - s_1 - 1, \dots, s_k - s_{k-1} - 1 \}$$

$$2(s_1 - x) \quad s_1 - x \geq 0$$

$$2|s_1 - x| - 1 \quad s_1 - x < 0$$

אבל לפי הלוגיות של המספר הראשון אפשר לפענח את s_1 ואת השאר אחרי זה כבר לא בעיה לפענח.

השפור הבא הוא כזה: יכול להיות שלפני x היה קודקוד y שרשימת ה successors שלו דומה לזו של y. ואז אולי לא כדאי פשוט לשמור את $S(x)$ אלא להגיד משהו מסתכן ה- successors של x הוא כמו של y אבל עתה השינויים הבאים...

אז יש לנו - $x-y$ - reference number

- בוטים באורך $S(y)$ שמגדירים איזה מה successors של y הם גם של x ← copy list

- רשימה של קודקודים חסרים - $S(x) - S(y)$

ואז הטבלה שלנו הופכת ל-

node	out-d	ref. #	copy list	Extras

את הימים הלה אנחנו רוצים לדחוס כי השמור המוקם הן די בזבוז. אז במקום זה נסתכל לה כסדרה של חילופים בין רלפים של 1 ל-0. אז אפשר לה שתמש בקידוד הפשוט שאומר פשוט מה האורכים של הרלפים הלה. בהנחה שהרלפ הראשון הוא תמיד של 1 (אורכו לא יכול להיות 0) אין בעיה לפענח אותה. ואז מקבלת טבלה

Node	Out-d	Ref	#blocks	Copy blocks	Extras

ברשיון נרצה לדחוס גם את הרשימה של הקלדקדים הנותרים. נרצה לנצל את זה שהרבה פעמים יש שם רלפים עוקבים.

* היום יצאו RC של Windows 7

אלים לבקשה להצלם שם בבית- באתר.

פיברגי אל דרכים לאינדקס את ה- web וראינו פתרון שמשתמש ב-3 ביטים עבור link.
נא לשים לב שבמצבת יש טבלאות שמועתקות מהמטמון המקומי ויש שם כמה typos. חוץ מזה את הזמנה שלא לא באתר דרוש לשמור כי אלה בשום מקום עיקבים.

ענין שרודים לדעת אם יש לינק בין x ל-y. הינו רוצים ללכת לשורה x ולבדוק אם מופיע שם לינק ל-y. אבל פה יש שתי בעיות:
קודם כל, אי אפשר סתם ככה ללכת לשורה x כי אנחנו לא יודעים איפה היא מתחילה. שנית, גם אם יהיה ללכת יש לשורה x אבל הקידוד הוא כזה שה רשימות של שורה כשהי מלוות ברשימות של השורות הקודמות. לכן כדי לענות את הטבלה אין ברירה אלא לקרוא באופן רציף מההתחלה.
אבל המבנה מאוד קומפקטי ב- storage אבל קשה להגיע אמנם מיד ל- random access.

Wild Cards

כאינו איך מוצאים את רשימת המאמרים שמכילים את המילה t. זה היה פשוט מאוד- לחפש בלקסיקון ואז להסתכל ב inverted index.
אבל היינו רוצים גם לענות על שאילתות מהצורה lab* או *or, כלומר רשום לנו wild cards במילה.
אבל לענות על lab* זה קל כי הלקסיקון ממוין לפי אבל אנחנו פשוט יכולים למצוא את כל המילים שמתחילות ב- lab

ואז לפעול בדרך המקורית. אבל רק לענות על *ro אנחנו נצטרך לעבור לינארית על כל הלקסיקון. בשביל לפתור את זה אפשר לשמור את האינדקס פעמיים - פעם אחת כמו קודם ופעם אחת כשהלקסיקון הפוך. אבל זה לא עוזר לנו לענות על שאילתות מהצורה *lab* ...

n-grams

n-gram זה רצף של n אותיות עוקבות במילה. נסמן ב- $ תחילת מילה או סוף. אז ה-digram של dog הן

d , do , og , g

אז אם יש לנו רשימת מילים אפשר לשמור בשביל את רשימת כל ה- digrams עם הרשימות לאילו מילים שבהן הם מופיעים. אז כדי לחפוש את *r lab אנחנו צריכים לחפוש את כל המילים שמשתמשות לרשימות של l, la, ab, r ולסנן צריך לוודא שמה שמצאנו אכן מתאים לשאילתא...

מה הסיבוכיות של הדבר הזה? נניח ש-

* 30 אותיות כולל $
* 10^6 מילים
* אורך מילה ממוצעת (ללא $) הוא 5 אותיות
* הנתח אותיות הכפול ביותר מופיע ב- 10,000 מילים
* 600 זוגות אותיות מופיעים במילים בכל
* מדבים תופס 4 bytes = 32 bits

אם נחנו

* צריך 5 ביט לאות כי $\lceil \log_2 30 \rceil = 5$
* צריך 26 ביט למילה (מלבד מילה) כי $\lceil \log_2 10^6 \rceil = 26$
* הרשימה הארוכה ביותר היא באורך שיכולה תופס 14 ביט $\lceil \log_2 10000 \rceil = 14$

המבנה הכי פשוט הוא טבלה

letter pair	length	words

אז יצא

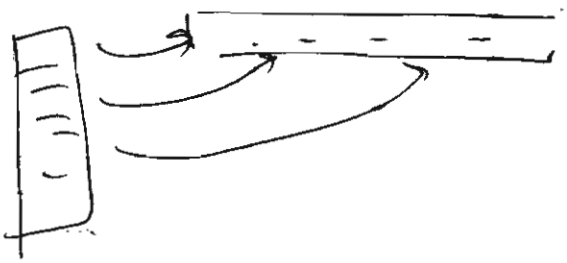
$$\underbrace{600}_{\text{סך כל האותיות}} \left(\underbrace{5+5}_{\text{גם אותיות}} + \underbrace{14}_{\text{אורך הרשימה}}\right) + \underbrace{1000000}_{\text{מספר המילים}} \cdot \underbrace{6}_{\text{בגלל מחרוזים 6 digrams}} \cdot \underbrace{20}_{\text{אורך מילה ממוצע}} =$$

$$= 120,014,400 \text{ bits}$$

מה זמן הגישה למבנה? אי אפשר לעשות חיפוש בינארי כי הרשימות לא באורך קבוע. לכן זמן הגישה הוא לינארי. (תשובה להבין את הדברים יכולה כי שאלות דומות עלולות להופיע במבחן!).

אפשר לשפר - נוסיף 600 מצביעים. נוספים כאלו נניח מבחינת הזיכרון אבל היו מאפשרים חיפוש בינארי. ב-scope של הקורס כן הבעיה מבחינת אבל תשים לב שבמציאות אי אפשר להביע על מקום סתם ככה. בדרך הבעיה היא ברמה של byte ואילו אנחנו מדברים הבעיה ברמה של bit. זה עלול להכניס סיבוכים...

הבעיה אחרת: להוסיף 30·30=900 מצביעים - מצביע לכל זוג אותיות אפשרי והם מצביעים לתוך הרשימות. אז נשמור 900 פוינטרים. יפשוט הרשימה ארוכה של רשימות.



ואז הזיכרון שלדיק הוא

$900 \cdot 32 + 6 \cdot 10^6 \cdot 20$

וזמן הגישה הוא לינארי!!

Rotated Lexicon

רעיון שהבנו בפתרון הקודם שאחרי שמצאנו את הרשימה צריך לעבור על כל אותיות שמה שקיבלנו אכן מתאים לשאילתא. עם Rotated Lexicon אפשר לענות על השאילתא. עם חיפוש בינארי אחד.

אם יש לנו מילה נשמור את כל ה"סיבובים" שלה לפי סדר האב. לכל סיבוב כזה מתאים גם (אינדקס הסיבוב, המילה, מספר).

אמשיך לשמור את הצורות והמילה מספיק כדי לשחזר את המילים המסובבות ולכן אפשר לעשות חיפוש בינארי.

אז למשל בשביל לחפש את *lab נחפש בטבלה את $lab

נחפש את	כדי למצוא את
$lab	lab*
or$	*or
ab	*ab*
r$l	l*r
r$l b	l*b*r

אז אם יש שני wild cards באמצע זה מלכתחילה שני חיפושים ותיתכן, אבל בשאר המקרים מספיק רק חיפוש אחד.

נניח רגע שבנוסף לנתונים הקודמים, המילה הארוכה ביותר היא באורך 30. אז כדי לייצג סיבוב צריך 5 ביט. ואז גודל הקסקון הוא

$$1000000 \cdot 6 \cdot (20 + 5) = 150{,}000{,}000$$

מילה, הפעמים, מספר מצבים, סיבוב

וזה הרבה יותר גדול מאשר הנמבנה הקודם. אבל אחד שני צמצם השאילתא לחיפוש מהיר.

עיבוד מקדים

כשמקבלים מסמך צריך להחליט מה בדיוק לשמור וזה בדרך משפיע גם התשובות שהמשתמש מקבל.

- דבר הראשון שמקבל עשות הוא case folding - הכל עובר ל- lower case - הלקסיקון שמור באותיות קטנות והשאילתות מתורגמות לאותיות קטנות לפני שעונים עליהן. זה כמובן תוסך בדיוקן. לגבי איכות התשובות אפשר להתווכח. למשל Apple או בושמה טובה שבה לא מתקבל הפירוש המתבקש...

- Stemming - מלישות שורש של המילים וזה מאפשר לחפש מסמכים עם וריאציה של המילים שלה השאילתא. אז יקבל יותר מידע - חלקו רלוונטי וחלקו לא.

התכנה הידועה ביותר בתחום זה היא Porter Stemmer. זו פשוט רשימה של כללים שמפעילים אותם זה אחר כך ומסרל מתקבל השורש. במלבנים יש ליקים למאמר. לעשות זה לא ממש מבטל את השורש. אלא מין בורה נורמלית ומה שחשוב הוא שעושים את זה אם בבניית הלקסיקון וגם לפני חישוב השאילתא אז מגיעים לאותו המקום.

בדרך לא עושים את זה היום כדי לחסוך בזמן חישוב.

- Stop words - מילים מאוד נפוצות ובדרך לא חשובות - the, a, to וכו'. הן תופסות המון מקום בגודל הרשימות האלה מאוד ארוכות והן לא יכולות לבאוס את חישוב השאילתא. למשל אם מחפשים the bird אז הרשימה של the כנראה מאוד ארוכה וכדי לחתוך אותה עם הרשימה של bird זה יקח הרבה זמן, עדיף פשוט לחסל רק את bird. בדומה, להתעלם מהמילים האלה לא תמיד טוב, למשל ... to be or not to be

תיקון שגיאות כתיב

שני מטרות עיקריות:

- לתקן שגיאות כתיב לפני שמאנדקסים אסמך (בדרך לאינדוקס)
- לתקן שגיאות כתיב בשאילתא (בדרך לא מתוגים אוטומטית אלא נתנים הצעה).

יש שני סוגים:

- isolated word - מסתכלים על כל מילה בנפרד
- Context Sensitive - מסתכלים על מילה בתוך ההקשר שלה. למשל זה יכול להיות טוב לאיפוס כמו

I flew form Israel to Italy

אנחנו נדבר רק על תיקון של אחים בודדות.

חשוב לשים לב: אפשר להשתמש במידע שאנחנו יודעים מאיפה הגיע המסמך. למשל אם המסמך הגיע מ-OCR אז יש סיכוי גדול להתבלבל בין O ל-D כי הן דומות ואם המסמך הוקלד (כ-QWERTY) אז יש סיכוי גדול להתבלבל בין S ל-D (כי הן דמויות על המקלדת).

הנחת הבסיס שלנו הוא שיש לקסיקון שמכיל מיליון המילים בדרך הנכון. את הלקסיקון הזה אפשר להביא משני מקומות

- לקסיקון סטנדרטי כמו מילון או לקסיקון מבוסס לתחום מסוים לשהו
- לקסיקון שנבנה מהסתכלות בקורפוס מסמכים (למשל על האינטרנט) כולל שגיאות כתיב שמופיעות שם גם.

edit distance

בהינתן שני רצפים של אותיות, מרחק העריכה שלהן הוא מספר הפעולות המינימלי (הכנסות, מחיקות, והחלפות) שצריך לבצע כדי לעבור מ- s_1 ל- s_2.

לדוגמא, edit-distance("cat", "dog") = 3

edit-distance(s, t) $\le$ max(|s|, |t|)

חישוב המרחק:

1) n = length(s)
   m = length(t)
   if n = 0 return m and exit
   if m = 0 return n and exit

2) המשך במין אלגוריתם דינאמי שרלה...

דוגמה:

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2	3	4
A	2	1	1	2	3	4
M	3	2	2	1	2	3
B	4	3	3	2	1	2
O	5	4	4	3	2	1
L	6	5	5	4	3	$\boxed{2}$

יש שלב יותר טוב לאמצע!

המרחק! הפעולות שצריך לעשות הן להחליף את U ב- A ולהוסיף L.

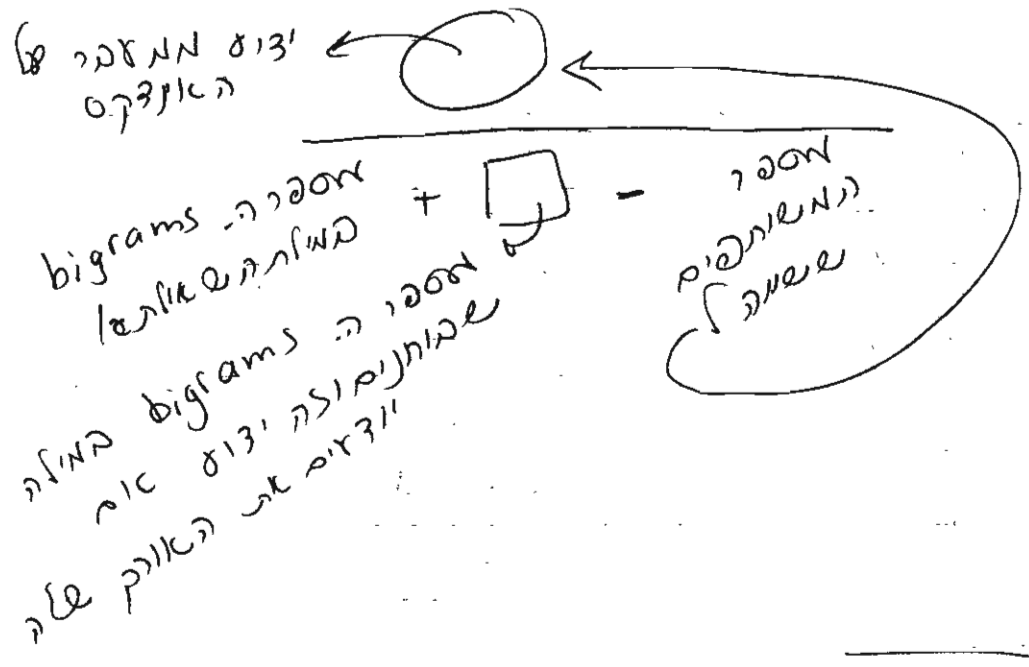
אפשר לעשות משהו קצת יותר מתוחכם - מרחק מחושק שבו התשובות מסויימות גדולות יותר מאחרות והאלגוריתם פועל.

בס אפשר בוודאי לא נהיה לעבור על כל הקטבון ולחלק את המילה הכי קרובה לשאילתא שלנו עם האלגוריתם הדיבאי.

אז אפשר להסתכל על n-grams של שני מילים ואז להסתכל על מקדם Jaccard - $\frac{|X \cap Y|}{|X \cup Y|}$.

אז קלדה נתפוש לוגות מילים שמקדם Jaccard שלהן הוא מעל סף מסוים. ואותן נבדוק ביתר פירוט. (בדיקת המקדם תהא מהירה).

נשים לב שם - n-gram index לא מופיעות המילים עצמן אלא המספרים המזהיים שלהן. אבל אם אנחנו יודעים את אורכי המילים אז אפשר להשב:



שאילתא היא ביטוי בוליאני של מילים באמצעות AND, OR, ! - NOT.

שאילתא יכולה קונקטיבית אם ישנה שימוש רק ב- AND.

רגע לפני שאלתה ראלה בדבר דרך לעבור על הלקסיקון ולשמור את התדירויות של המילים והרתיבות שלהן ואז אוברים על המילים מהנדירה ביותר לתדירה ביותר ומורידים מהרשימה שלנו את המסמכים שהמילים הראחרות לא מופיעות בהם.

מתחילים מהנדיר כי לא תוסיף זמן. אותנו רוצים להגיע לרשימות הארוכות עם כמה שפחות דברים להבדיק.

נניח ששלנו שתי המון מסמכים C_1, C_2 הראשון C_1 קטנה מ- C_2.

אם הרשימות ממוינות אפשר למצוא את החיתוך שלהן ב- $O(|C_1|+|C_2|)$.

אם הן לא ממוינות וב- C_2 אפשר לעשות חיפוש בינארי אז אפשר לחתוך אותן ב $O(|C_1|\cdot\log|C_2|)$.

בפועל במבני הנתונים שאיתנו בקורס אין לנו Random Access אבל אי אפשר לעשות חיפוש בינארי ונשארנו רק עם האופציה הלינארית.

הבעיה היא שצה לא כל כך יעיל...

(25) 7/5/09
DB

- התרגילים לא נבדקים.
- באתר הקורס יש הקלטות של ההרצאות.
- ננסה להעביר את פתרון התרגיל לסוף השעה השישית

Query Processing

אנחנו עתה יודעים דברים שניתן לשאול אידע מקבצים שבהם אנחנו רוצים לחפש. זה נשאר פשוט. אבל רשים כמויות גדולות של מידע רבה צריך לחפש פתרונות לא טריוויאליים...

בעיבוד שאילתות הצמצמנו לשאילתות קונייקטיביות. לה פתרון היו תהליך חותוך בין posting lists.
דיברנו על שני אפשרויות:

- מעבר לינארי על שני הרשימות (המיונים) ומציאת החיתוך אבל מעבר לינארי יכול לקחת הרבה זמן
- לו הייתה לנו יכולת לגשת רנדומלית ל- posting list השני, לכל מילה בראשון אפשר לעשות חיפוש בינארי בשני ואז זה יעבוד $(\log|L_2| \cdot |L_1|)$ שזה טוב אם $|L_1|$ קטן !- $|L_2|$ גדול.

אבל אין לנו יכולת לגשת רנדומלית לרשימות ולכן משתמשים בסבבה

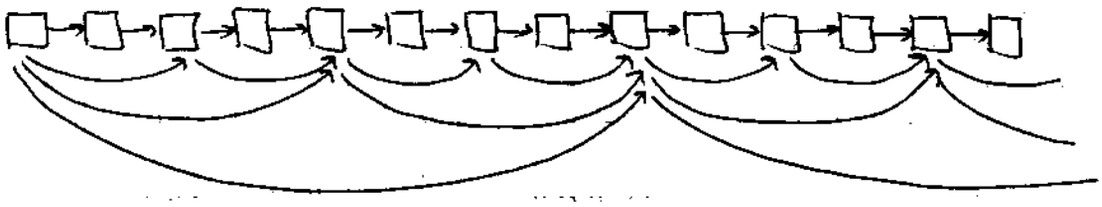
- הרשימות הינן רשימות הפרשים וצריך לרוץ אחורה רק לפעמים
- הרשימות אינן באורך קבוע אלא אי אפשר לדעת אפילו האמצע של הרשימה.

איך בכל זאת נוכל לעשות גישה מהירה יותר?
אינטואיציה: נניח שיש לנו רשימה מקושרת ממוינת אם יש לנו

רק מדפיסים ל- node הבא. אז כדי להגיע לאיזה node באמצע חייבים לעבור על כל אלה שמאחריו.

נוסיף עוד קצת פוינטרים:

- לכל קודקוד שני יהיה פוינטר לכל קודקוד שני הבא
- לכל קודקוד רביעי יהיה פוינטר לקודקוד רביעי אחרון
- ובקיצור לכל קודקוד 2^i יש פוינטר לקודקוד שנמצא 2^i קודקודים אחרון.



זה כמובן אופטימלי מן חיפוש לוגריתמי, אבל חישוב הוספה של כל הפוינטרים

כי כל המצביעים אופטימליים לגשור קפיצות לאמצע

הוספה / הכנסה לרשימה בצורה הזו אמור בעיות כי צריך לעדכן את כל הפוינטרים. אבל כמבנה נתונים סטטי זה אמור לתמוך.

יש איזדיה של המבנה הזה שנקראת skip list (יש מאמר קצר ונחמד) שמשתמש מרנדומליות ומבטיח מבנה אמור יעיל מהסתברות גבוהה.

בדורה ישירה אי אפשר להשתמש במבנה הזה כדי לשמור את ה- posting lists כי ה- posting list הוא רשימת הפרשים. אבל ניתן שלה לא שמור בהפרשים בכל זאת, במשך רמה שבועות התאמצו מאוד לחסוך כ"כ מבנה נתונים ורשיו הם אנו מבנה נתונים שמרפיו את רמות הדירון. אז אנחנו לא נשתמש ב- skip list.

אבל נשתמש ברעיון של הפוינטרים כדי לשפר את זמן הגישה.

נקודות סינכרון

נקודת סינכרון ב- posting list היא זוג <docId, bitaddress>
כאשר docId זה ה- id של המסמך הנוכחי ו- bit address
זו כתובת לנקודת הסינכרון הבאה. כדי לחסוך אפשר לשמור
את כל הפרשים מנקודת הסינכרון הקודמת. וכך אפשר
להתקדם ברשימה.

אבל מה אם השתנו בשיטות קידוד? קידוד גאמא וקידוד האפמן
מקודדים כל מספר בנפרד אבל זה מספר. גם קידוד אריתמטי
מגייתם לכל הרשימה כאובייקט אחד ומקודד אותה כמספר אחד,
אבל שם אי אפשר יהיה לקפוץ לאמצע. אבל זה משתו שקרים
לקחת בחשבון...

ושוב כרגיל יש trade off בין מספר רמות המצביעים
לבין הזיכרון. נניח שהמצביעים קופצים כל פעם אחד X אומנטים
ויהי k אורך הרשימה. מספר ההשוואות הוא $\frac{k}{x} + x - 1$ ואת
זה אנחנו רוצים למזער. אבל בואתרים $x = \sqrt{k}$.

שאילתות לא קונג'נקטיביות

כאמור היינו רוצים יכולת לשאילתות OR או NOT.
אם יש קונג'נקציה של דיסג'נקציות משלבים את הגדלים של
הדיסג'נקציות ע"י סכום הגדלים שכל אחת בתוכן ואז
מחשבים בדומה לשאילתא קונג'נקטיבית נשמסתכלים על כל דיסג'נקציה
כביטוי בודד. NOT זה דומה לשאילתא רגולרית רק שמוגאים שלה לא
מופיע.
נניח שיש לנו קונג'נקציה עם wild cards. אמרנו שמהרה של
wildcards אנחנו למעשה מתפשטים את כל המילים בלקסיקון
שיכולות להתאים. ואז למשל *b* and *he הופכת למעשה
לשאילתא שאינה קונג'נקציה של דיסג'נקציות.

בשאילתות כדה שאינו כאן הרוחה מאיך היה לכל היותר לינארי
באורך הרשימות של המילים. האמת מאיך נסון? לא!!

שאילתות שארכיות NOT יהיות לקחת יותר זמן מאורכן הרשימה
של האולה, כי יהיו להיות שרשימת המסמכים שאינה לא
מופיעה בהם הרבה יותר ארוכה מרשימת המסמכים שהיו
כן מופיעה בהם. לכן אנו חפשים בדר לא מופשטים
שאולתות כאלה. זה נקרא שאילתא לא טהורה.

יצירת האינדקס

יש שני נושאים שדרוש לדון בהם
- מציאת המסמכים עצמם (crawling)
- בניית מבני הנתונים. וזה קשה!!
- אין מספיק זיכרון להחזיק את כל האינדקס ואז יש הרבה page faults.
- posting list צריך להיות רציף בזיכרון. אבל בשלב שאנחנו עושים בחיסה אנחנו לא יודעים מראש כמה מקום זה יקח. אנ אנחנו לא יודעים כמה מקום להקצות. אי אפשר לבנות את זה בלוח הנאיבית - לקרוא מילה מילה ולהוסיף למקום הנכון ב- posting list. יש ממש תשש רבד של thrashing - שהתכנית תשלה את רוב הזמן ב הקריאה וכתיבה לדיסק.

אינטואיציה לפתרון:
- בניית הלקסיקון היא לא בשלב כל הוא קטן
- בשביל ה- inverted index קודם יוצרים גרסא של מילה ומסמך ואז אם ממיינים את זה לפי המילים אז מקבלים מישהו סדרתי שאפשר לעבוד בקלות. החלק הקשה הוא המיון כי כל המידע לא נכנס לזיכרון הראשי. אבל גם ספר ב- DB תופס איב אושים מיון במקרה הזה. אל סימן!

Crawler

מטרות:

- להוריד כמות גדולה של דפים (את כל הרשת אז לא היה אפשרי)
- לעדכן את הדפים שכבר הורדו
- לאלוץ דפים חדשים
- לוודא שיש כמה שיותר "דפים טובים" שלה משאבים שצריך להקדיר.

רעיון בסיסי: מתחילים עם קבוצת גרעין של URL-ים שמים אותם בתור ואז עובדים על התור כך:

- מוציאים URL מהתור ומורידים אותו
- מוציאים את כל הלינקים שאותו מצביע עליהם
- מוסיפים לתור את כל ה-URL-ים שעוד לא היו בפה ואת כל זה עושים עד שהתור ריק.

התהליך הזה לא באמת מסתיים כי אפשר לחשוב על הרשת כמשהו אינסופי - דפים כל הזמן משתנים / נוספים.

אבל אם מקימים אתר אינטרנט חדש אבל אף אחד לא מצביע עליו, אז הדף הזה בתיאוריה לא יגיע למנוע חיפוש. כלומר דף יתום. אפשר אמנם להיות שיש דרכים למנוע חיפוש אבל אומרים שדף הזה לא יישאר לזמן רב כי לא שאל אחד לא מצביע עליו זה כנראה אומר שהדף לא חשוב...

איך לארגן את התור של ה-URL-ים? יש מנגנונים הרבה מתקדמים. האסטרטגיות הבסיסיות:

- מחסנית - מכניסים לתחילת התור ומוציאים מתחילת התור. זה לא עושה חיפוש לעומק.
- תור - מכניסים לסוף התור ומוציאים מהתחלה. זה לא עושה חיפוש לרוחב.

רוחב VS עומק - אין אחד מהם שיותר טוב באופן מוחלט. לכל אחד יתרונות ומשלו. חיפוש לרוחב "הוגן" יותר - לא "נתקעים" רק באתר אחד. ואילו בחיפוש לעומק יש יקרירות של הבקשות וזה חוסך זמן על DNS. עוד יתרון של חיפוש לרוחב הוא שעוברים קודם על הדפים ה"רדודים" יותר - בעומק נמוך יותר וסביר שהם חשובים יותר כי בדרך שעוברים לינקים חשובים למשל. חיסרון יחודי של חיפוש לעומק הוא שהוא על הזמן שנותן לאתר שפר ומעמס אותו מאוד וזה ממש יכול לגרור ל denial of service, וזה לא רצוי כי הם שלנים להתעלבן ואל לחסום למשל את השרות לשפת של האוניברסיטה או מישהו כזה. אל מי שפותח crawler - שיילהר.

- תור קדימויות
  - interest driven - יש רשימת מילים שמתאר נושא שמעניין אותנו ואז לפי זה לתת דירוג לפי הדומיין שלו לרשימת המילים הנ"ל. ואז לפי הדירוג להכניס לתור. הבעיה היא שאנחנו רודפים לתת דירוג גבוה לקדם את הדף אל אפשר אולי להסתמך על הדף שמגבים אליו...
  - popularity driven - הדירוג נקבע לפי כמות הלינקים לדף הזה
  - location driven - לפי מילים שמופיעות בכתובת (כמו .ac, .edu) או כמות ה"\" בכתובת - ככל שיש יותר הדף עמוק יותר וכנראה שפחות חשוב פחות...

איך יש לעקב את ה- crawler?
- צריך לפענח את ה- IP בעזרת DNS
- צריך להתחבר לשרת ולשלוח לו בקשה
- צריך לחכות לתשובה

בשביל לקצר את זמן ה- DNS lookup אפשר לעשות

caching של תגובות DNS. חוץ מזה,
אפשר גם לעשות prefetching (לבקש אתרים שמכניסים לתור ולא נשמרו)
לפעמים אנשים רוצים לתת הרשאות ל-crawlers - לא לאפשר
להם לגשת בכלל, לא לאפשר להם לאינדקס, לאפשר רק לחלק
מה-crawlers-ים לגשת ולזוז. crawlers מנומס אלוף
להקשיב לבקשות המלים. אפשר לשים ב-root directory
של האתר קובץ בשם robots.txt ושם יש הוראות -
גישות של crawlers-ים ו-URL-ים שאסור להם לגשת
אליהם. אין אחד לא אוכף את התנאים האלה!
חוץ מזה בתוך ה-HTML אפשר לתת הוראות לרובוטים וגם כדי
באמצע מיקרה בהקשר מלמעלה. לאמת יש meta-tag שאומר ל-crawler
לתגיר לדף בקר בו שום אדם לא מאה.
כמו שאמרנו אין דרך לאכוף את הבקשות אבל אם מישהו
לא לתת ל-crawlers-ים להיכנס. דרך להשתמש בטכניקות
של אמצעי מיגון.

שאלות מעניינות -
- נניח שאנחנו עושים crawler והוכנו לדף מאיזה אתר תוך
שנשמענו ל-robots.txt. האם יכולים לתבוע אותנו על הפרת
זכויות יוצרים? כן!! אפילו היה איזה משפט שאז הפסידו
בו לאיזה אתר חדשות או משהו כזה.
- בוודאי אם אנחנו לא נשמעים ל- robots.txt יכולים לתבוע אותנו.
- בשל אתר יכול לוודא באמצעות חלקים שה-crawler שהגיע אליך
הוא אכן מי שהוא טוען שהוא. למשל, אתרי חיפוש יכולים
מפרסמים אותו כתובות IP וכו'.
- אין דרך למנוע מאופן מוחלט מ-crawlers-ים ספציפיים
להיכנס...

במדינה יש תרשים כריאה של crawler

באינטרנט יש המון דפים עם URL-ים שונים אבל התוכן שלהם זהה או דומה מאוד זהה (mirror sites).

האתגר הוא להמנע מלהוריד את אותם הדפים ממקומות שונים-
כך שאבל המון משאבים יותר יהם להוריד את איכות התוצאות.
איך אפשר להתגבר על זה?

- לשמור איזו גרסה מנורמלת של ה URL-ים. למשל db~ ! db/index.html~ זה אותו הדבר.
- לשמור hash של הדף ואז כשמגיע דף חדש לבדוק שה- hash שלו לא קיים. אמנם זה לא מונע מאיתנו להוריד את הדף אבל לפחות לא נשמור אותו. רק נשים לב שיש פונקציות דומות (למשל MD5) שמסכנות להתנגשות אבל מאוד קטן. זה עובד רק לזיהוי דפים זהים ממש ולא "דומים זהים".
- לנסות לאתר דפים דומים זהים באמצעות edit distance. זה יעבוד לא יעיל כשיש מיליוני דפים והם גם לא קרובים מדי.
אז אחרי נדבר על פתרון אחר - לפי דף יודעים בקביר רק שאם דפים הם דומים אז בהסתברות גבוהה יש להם בדיוק אותו תקציר.

Shingling

מהגדרת מסמך D, w-shingle יהיו את הדף של המסמך שמעניק w. S(D,w) הם כל ה- w-shingles של D. נגדיר דימיון של A ו- B

$$sw(A,B) = \frac{|S(A,w) \cap S(B,w)|}{|S(A,w) \cup S(B,w)|}$$

זה ה- Jaccard Coefficient !! ככל ש- w קטן יותר אנחנו מתיחסים יותר ויש יותר סיכוי שמסמכים רבים יראו דומים.
בדרך משתמשים ב- w=7.

הבעיה בסתם לבחור את (שווים) הסט שהקבוצה יהיה להיות מאוד גדולה. אז אנחנו רוצה לדגום את הקבוצות הגדולה רק שדימיון בקבוצות יהיה על דימיון במאמרים.
לדגום shingle סתם רנדומלית זה לא טוב כי אם יש לנו שני מאמרים מאוד דומים עם n shingle-ים, וההסתברים לבחור את אותו אחד היא $\frac{1}{n}$ זה לא טוב לנו.
אפשר להניח שכל ה- shingle-ים הפשוטים מוגדר סדר מלא. אז למשל התקדיר יכול להיות ה- shingle הכי בסין (הבא שיש בסדר המלא) ואז ה- shingle הכי גדול (האחרון בסדר המלא). להיות מאמרים דומים אם יש סכוי גדול שנתשום שמאמרים דומים לא דומים יבאו דומים.
אז אנחנו רוצים להסתכל על רמה shingle-ים. בפרט אשתמשים ב- 100. (הסבר באלגו).

אנו יחיפוש בדפי אשתמשים בסמה crawler-ים אז צריך לבאים לסדר את ה URL-ים בין הקורים השנים כדי שהם יכלו לעבוד בו.

(30) 14/5/09
DB

Ranking

המטרה שלנו היא להבין איך לדרג דפים. עד כה הסתכלנו על שאילתא כמשהו בולבאני - או שדף מתאים או שלא. בלי משמעות לסדר של התוצאות. אבל ברור שב- web או אפילו באופן לא מאוד גדול של קבצי טקסט זה לא מספיק. היינו רוצים לתת ציון לכל תוצאה ולהתעניין את התוצאות מסדר לפי ציונים.

אלא שגם רלוונטיות זה משהו סובייקטיבי, שקשה למשתמש יחיד. מנוע החיפוש מנסה לחשוב על ערכה של מה רלוונטי ומה לא.

השמנסים למדוד טיב של מנוע יש כמה מדדים סטנדרטיים

- דיוק - אחוז התשובות הרלוונטיות מכל התשובות שהוחזרו
  כמובן נרצה שה דיוק יהיה רמה שיותר גבוה
- recall - אחוז התשובות הרלוונטיות שהוחזרו מתוך כל התשובות הרלוונטיות.
- דיוק ב- k - אחוז המסמכים הרלוונטיים מבין k התוצאות הראשונות.

איך אפשר לקבל דיוק מושלם? - לא להחזיר כלום
איך אפשר לקבל recall מושלם? - להחזיר הכל
אז הדיוק יה recall בעיה אחד של תשובה השני.

אי אפשר לחשב את המדדים האלה אוטומטית כי מנוע החיפוש לא יודע מה רלוונטי ואולי הוא היה יודע אז התוצאות שלו תמיד היו מושלמות.

אז איך בכל זאת?

את פונקציות הדירוג אפשר מגדרים לחלק לשני קבוצות - גלויות שבחרנו (בהינתן שאילתא כל דף מקבל ציון) ומלמדות גלויות בשאילתא (לכל דף יש ציון אבסולוטי). פונקציות גלויות שלוקחות בחשבון הרבה יותר יקרות לחישוב, אולם האינטואיציה אומרת

שצה דירוג יותר מוצלח. PageRank של גוגל היא מתבי תלויה
בשאילתא ורוצה הדרך הדגלה של גוגל היתה בתק שהם שמו לב
שצה עובד יותר טוב מדירוג שתלוי בשאילתא.

TF-IDF

לכל מילה בכל מסמך יש משקל $W_{t,d}$ ולכל מילה בשאילתא
יש משקל W_d. המטרה היא להגדיר את W_{td}.
אם המילה t לא מופיעה כלל במסמך d אז $W_{td}=0$.
לשאר אפשר לתת משקל 1 אבל זה מתשפשטני כי סביר
שמילה תדירה יש משמעות גבוהה יותר ממילה נדירה. אז אולי
היינו רוצים ש- W_{td} יגיד את מספר המופעים של t ב-d.
אבל יש זה אולי לא מסוף מוצלח כי יוצא שמילה שהופיעה
10000 פעמים טובה ממילה שהופעה 5000 פעמים.
מספר המופעים
בגל המשקל הסטנדרטי שמשתמשים בו היא $W_{td} = 1 + \log f_{td}$
נשים לב שלא כל המילים שוות. לדיק לתת איבר דומה
למילים נדירות (בשפה).
למילה t נגדיר את f_t להיות מספר המסמכים שבהם מופיעה
מהם ואז נגדיר $W_t = \log\left(1+\frac{N}{f_t}\right)$
לרשון יש לנו משקלות, מה נעשה אם משתמשים במודל
מרחב וקטורי. מסמכים ושאילתות הם וקטורים ב- n ממדים,
כאשר n מספר המילים בלקסיקון. במקום ה-k של הוקטור
של מסמך d מופיע המשקל W_{td} כאשר t היא
המילה ה-k בלקסיקון. ובמקום ה-k של הוקטור של
השאילתא מופיע W_t כאשר t המילה ה-k בלקסיקון.

דוגמה: נניח ש- $N=210,\ f_a=30,\ f_b=210,\ f_c=50$

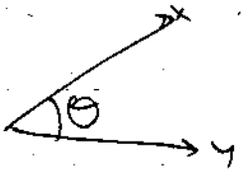
Doc 1: aabbd Doc 2: bcccc

השאילתא היא abc

	A	B	C	D
Doc 1	2	2	0	1
Doc 2	0	1	3	0
Query	3	1	2	0

יתכן שבמבחן תהיה שאלה
דומה. מומלץ להכיר מושגים

אז יש לנו וקטורים של מסמכים ושל שאילתא. נבדוק כמה הוקטורים של המסמכים קרובים לוקטור השאילתא. את זה נמדוד ע"י הזווית שבין הוקטורים.



$$\cos\theta = \frac{\sum x_i y_i}{\sqrt{\sum x_i^2}\sqrt{\sum y_i^2}} = \frac{\langle x,y\rangle}{\|x\|\|y\|}$$

במסמכים שלנו אותקבל...

$$\cos\theta = \frac{1}{|Q||D_d|}\sum_{t=1}^{n} W_t W_{td}$$

אם $\frac{1}{|Q|}$ אפיין בכל הדירוגים בין המסמכים ומסמך שמעניינת אותנו רק הה שוואה אפשר בשביל להתעלם ממנו.

נמשיך דוגמה:

	A	B	C	D
Doc1	2	2	0	1
Doc2	0	1	3	0
Query	3	1	2	0

⇐ הציונים של המסמכים: $\frac{\sum W_t W_{td}}{|D_d|}$

$$Doc1: \frac{6+2}{\sqrt{9}} = \frac{8}{3}$$

$$Doc2: \frac{7}{\sqrt{10}}$$

אז Doc1 מדרג גבוה יותר.

עד עכשיו הנחנו שיש לנו רק טקסט אבל ב- web יש אלמנטים של html (כמו הכותרת של הדף, מילים ב- bold או italic) שכדאי לעצור לנו בהתלטה. אז אפשר לשנות את המשקל של מילים שמופיעות במקום מסוים. למשל מילה מכותרת ניתן לתת משקל גבוה יותר.

עוד רעיון ומאד הוא להשתמש ב- anchor text של לינקים רבים לדברים משהו כל הדף. אז אפשר למשל לתשוב כל המילים האלה ואולי הן ממש מתוכן הדף אלה הנכנים אותם לוקטור ומשם להמשיך

כרגע.

אך עדיין אינו שיטת פירוק עליות שאותו.
נעבור לדירוגים כלפי עליים. הרעיון הוא לעשות איטרציה
של מבנה הלינקים.
אפשר להסתכל על ה- web כגרף עם קשת מ- P_1 ל- P_2
אם ל- P_1 יש לינק ל- P_2. במידה מסוימת P_1 אומר שחוא
חושב ש- P_2 הוא טוב. אז אפשר בדרך מאוד פשוטה להגדיר
- פופולריות מכוונת - הדירוג של P הוא מספר הקשתות הנכנסות
ל- P
- פופולריות לא מכוונת - הדירוג של P הוא מספר הקשתות הכולל
הנכנס / יוצא מ- P.

אבל אולי הינו רוצים לתת יותר משקלות לקשתות של דפים חשובים.

אחרת מקרי - יש אלף שאתחול אדל רנדומלי ומשם ביתר
אקראית לינק ותולך אליו וכן הלאה.
הבעיה היא שבאינטרנט יש dead-ends ומלכודות שופשף
להיתקע שם.

dead-end

אל הבעיה האקראי שלנו אם יכול לעשות טלפורטציה אם
בא לו. כלומר מבטיחים שהמעברים הם שרשרת מרקובית
ארגודית - אם מתחילים בדף כלשהו, ולכן מספיק ברגע
אל יש הסתברות חיובית להיות בכל דף אחר והיא בלתי
תלויה במקום שהתחלנו בו.
לשרשרת הזו יש מטריצת מעברים - ההסתברות להגיע מאתר
i לאתר j עי לינק.
המיפוי של דפר כדי לחשב את הדירוגים דרוש לפתור מערכת משוואות
לינאריות שהן גם פשוט אבל לא רשמדבר בכמות הדפים

שיש ב- web, אנ משתמשים ב-Power Method.
מתחילים מדירוג אקראי לכל אתר ומשתמשים בקונבולוציות
של הנוסחה כדי לעדכן כמה איטרציות. התהליך הזה
מתכנס לדירוג האמיתי. בדרך פשוט עושים מספר קבוע
לכל אתר של איטרציות.

איך אתר לכל הקונקורנטים יש אותו דירוג!

אפשר רמות לבנות page rank שבהן תלוי נושא, למשל
מנוע חיפוש לספורט. זה לא מתקבל אוטומטית מ-
page rank כי בדירוגים שלו אמצעים ופופולריים.
קודם הגדרנו האקראי. רק לעשות טלפורטציה לכל דף. אנ
במקום זה אפשר להגדיר את הבולט הדפים שאפשר לעשות
אליהם טלפורטציה להיות רק דפים שאנחנו יודעים שהם
איכותיים לנושא שלנו. מנועים שמשתמשים בזה בשביל
להילחם נגד spam וזה נקרא TrustRank שם הבולט
הטלפורטציה היא הבולט דפים שאנחנו יודעים שהם לא spam.

איך אפשר להשתמש פירוש של דף?
- ב- TF-IDF - להתבונן באתר הרבה מילים שונות
שלאו דווקא קשורות לאתר.
- ב-Page Rank - לנסות הרבה לינקים שמקשרים לאתר.
שם - לקבוע מאמר ולעקוב אחרי שלות.

עוד כמה שיטות דירוג
- פופולריות הקלקה - אספר הפעמים שמשתמשים לוחצים על
לינק בתוצאות החיפוש מגדיל את הרלוונטיות.
- stickiness - להבין כמה זמן משתמש נמצא מאיזשהו
אתר - לחקור כמה זמן. אמר עד שהמשתמש חוזר לדף
תוצאות החיפוש.

הם מקרה של השיטות האלה סיכלו מבעיה ש-
get richer rich - מי שרבר יש לו
pagerank גבוהה בגראה ימשיך לגדול כי הרבה
ימצאו אותו ויקשרו אליו ובוודאי יהיו לו יותר קליקים.

XML

XML זה סוג קובץ עם tags ואכן HTML זה
סוג של XML אלא שב-XML אפשר לתת לtags
כל שם שנרצה.

<u>מוגיבה</u>

הרעיון של מי שהמציא את ה-web הוא שהוא יהיה סטנדרט
שאפשר יהיה לראות לאחשב. ממש להבין אתרים אלא סתם
לחפש מילים. אז צריך למצוא דפים שהוגשו מהם מידע
רלוונטי. אבל זה בעל לא פשוט

- איך נדע שהאתר מתאר את מה שצריך?
- איך נדע איזה מידע להוציא?
- דפים משתנים כל זמן אז כל שנבנה היום לא
יעבוד מחר.

אם הרשת הייתה מסוג נתונים גדול הכל היה יותר מסובך אבל זה לא
המצב. אז יש שתי גישות

- בינה מלאכותית
- להפוך את ה-web למאגר נתונים - לרתום אותו כל
המידע כ-XML.

כאן נמונה הגישה שנתמקד בה.

כדי שמחונה תוכל לובמין לא מספיק רק לכתוב
מידע אלא לכתוב אותו יחד עם הסמנטיקה שלו.
XML זה סינטקס שמאפשר לתת לטקסט סמנטיקה (להגיד
שזה שם, טלפון, אי מייל וכו')
RDF מגדיר אובייקטים ויחסים ביניהם.
OWL מגדיר אונטולוגיה שמקשרת מושגים וראיונות שונים.
שירותי ווב מאפשרים למחשבים להתקשר מידע אוטומטית
אחרון נתמקד בטכנולוגיה הראשונה.

יש גרסה XML הנשמרת בדרך של RSS.

(34) 4/6/09
DB

בשיעור הקודם התחלנו לדבר על XML. אמרנו שהמוטיבציה היא semantic-web, שה-web יהיה מובן לארץ לאנשים אלא גם לאחשבים. וזה ההבדל בין XML ל-HTML. HTML נותן לנו אינפורמציה רק על ה style של הטקסט ולא על המשמעות שלו.

עוד מוטיבציה היא החלפת מידע. יש הרבה מקורות של מידע אבל כל אחד מסוגים שונים עם סכמות שונות. למשל, יכולים להיות מאגרי מידע של חברות. ואם החברות רוצות להתאחד הן רוצות איך שהו לחבר את המידע. איך לעשות זאת? כל חברה יכולה לעבור למבנה ביניים XML וכך הן יכולות להבין אחת את השניה.

XML מתאר רק תוכן ולא style. למה זה טוב? יכול להיות שנרצה שה אתר ישתנה עם הזמן או שאופנות העיצוב משתנות. אם האתר שלנו כתוב ב-HTML בלבד אז אם נרצה לשנות את המראה זה מסובך ויקחו לשבש את הטקסט. XML מאפשר לרתוב את הטקסט בנפרד ואת המעלב הגרפי יכול להתעסק רק עם העיצוב. קובץ XSL לוקח XML ואומר איך להציג אותו.

- XML הוא למעשה מבנה היררכי של אלמנטים שמגדיר מחזיק אותו כעץ. לקודקודים של העץ באותה רמה יש סדר. לפעמים אפשר לעשות הבטחות פנימיות לתוך העץ ואז נוצר עץ כללי.
- לתגים של XML יכולים להיות אטריבוטים למשל אפשר לציין מה השפה של הטקסט
- לתגים של XML תאריך יש בן כמה
- XML הוא רגיש לסדר ורגיש ל-case.

אם יש לנו טבלה אפשר בקלות להפוך אותה ל- XML-
פשוט כל שורה בטבלה תהיה אלמנט ומתנבי תת-
אלמנטים שמתאימים לעמודות.
הכיוון השני קצת יותר קשה. קשיים:
- צריך יכול להופיע מספר פעמים (בניגוד למודל
הרגיל שבו כל אטריבוט בטבלה היא צריך בודד)
- לרשומות שונות יכולים להיות אטריבוטים שונים
וגם יכול הרבה זכרון נשמע בטבלאות, אם עושים
תרגום נאיבי
- כמות מלאכי מיגבלה של רמות היררכיה
- הסדר הוא מהותי

DTD - Document Type Descriptor

זו שפת סכמה שאותה אפשר להגדיר מה צריך להיות במסמך
XML. אפשר להשתמש בכל רגי להגדיר שפת XML
מיוחדות לצרכים ספציפיים. ואז אם קבל תוכן DTD
מסוים אז אלגוריתם שמכירות את ה- DTD הזה יכולות
להבין בדיוק מלא את ה- XML.
את ה- DTD כותבים באמצעות ביטויים רגולרים.
דוגמה במציאות - ספר טלפונים.

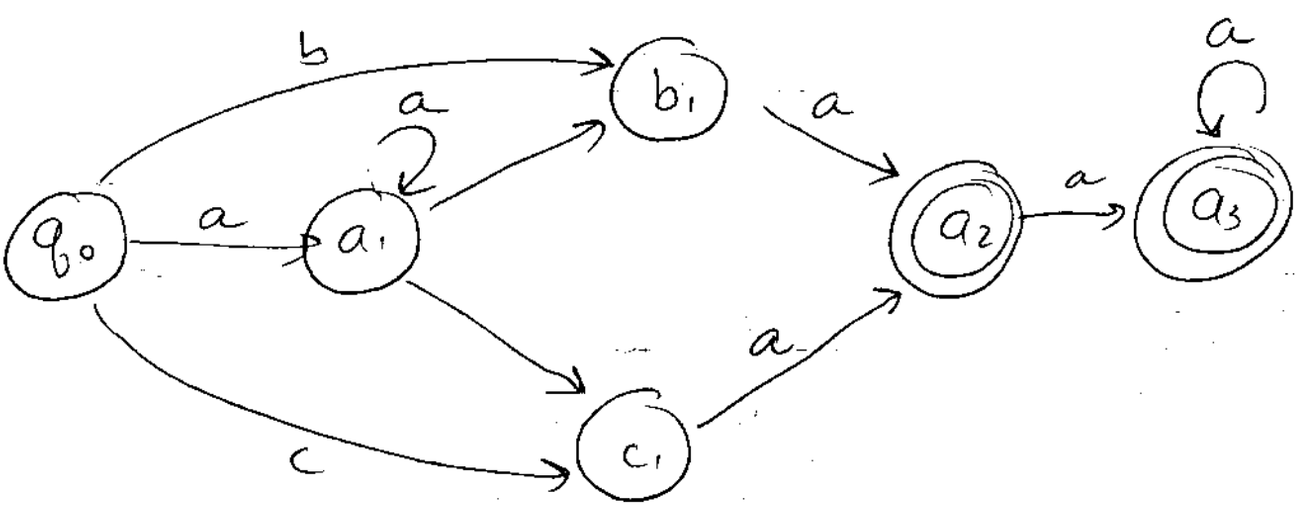
תכונה חשובה שצריכה להיות ל- DTD היא 1-unambiguity
בכל רגע נתון ב- parsing אנחנו מגיעים לצעד באיזה חלק
של הביטוי הרגולרי אנחנו נמצאים.
למשל $(a,b)|(a,c)$ לא מקיים את התכונה
$a,(b|c)$ מקיים את התכונה.

דוגמה: יש ביטויים רגולריים שבלי אי אפשר לכתוב
בלוחות 1- חד-משמעיות. למשל $(a|b)^*a(a|b)$.
אבל יש אלגוריתם שמזהה אם אפשר ואם כן, עושה תרגום.

שפה רגולרית אפשר להגדיר באמצעות אוטומט.
אנחנו נשתמש באוטומט מיוחד שנקרא מצלושקה.

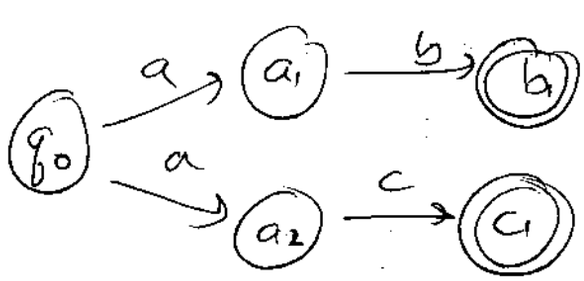
שלבים:

1. נירמול הביטוי ע"י החלפת e^+ ב- ee^*
2. אפיון של כל המופעים של כל אות: $a^*, (b|c)a^+$
   $\hookrightarrow a^*, (b|c)a, a^* \rightarrow a_1^*(b_1|c_1)a_2, a_3^*$
   ערשיו אין אותיות שחוזרות על עצמן.
3. יוצרים מצב לכל אות ממופעת ועוד מצב q_0
4. המצבים המקבלים הם כל המצבים שמייצגים אותיות שמילה יכולה להסתיים בהן.
5. יוצרים מעבר ממצב l_j ל- k_j אם יש אות שבה k_j נמצאת אחרי l_i. המעבר הוא עם k בדיוק את שלו.



שפה היא 1-חד משמעית אם האוטומט מצלושקה שלה הוא דטרמיניסטי!!

למשל, $(a,b)|(a,c)$ הופך ל-



לא דטרמיניסטי!!

דוגמה: נתונים כ-

```
<!element country
    (president | king | (king, queen) | queen)>
<!element president (#PCDATA)>
<!element king (#PCDATA)>
<!element queen (#PCDATA)>
```

לצה יש ביטוי שקול אז-אשמע:

president | (king, queen?) | queen

(כי שפה סופית אינה יכולה לבטא שלה שקול)

דוגמה: אפשר לשלם באשראי ואתה מאומן. אפשר באשראי רמה שרוצים ומאומן לכל נימר פעם אחת תיכנים לשלם. — (card+, cash?) | cash

מאגרים יש פירוט של הסינטקס של DTD ודוגמאות.

חסרונות של DTDs

- קשה לעשות לזה parsing
- אי אפשר לעשות מפה של "יושב" מאבנים - כמו
אי אפשר לשנות שדות של אלמנט מלבד מבנה שו
- צריך מקום אקסטנסיבי כדי לתאר של - a
יש שלושה בנים b, c, a לא משנה באיזה סדר.
- צריך לבדוק לא לעשות הגדרות שאין אל אמת טיפי
שמתאים להן.
- ועוד... (במאגרים)

הבדל בין אלמנט לאטריבוט:

element	attribute	
✓	✗	את אלמנטים
✗	✓	ID, IDREF
✓	✗	תכונה entities

XPATH

XML שומר את המידע. אבל אנחנו צריכים דרך לשאול את הנתונים מה- XML. בשביל זה יש את שפת XPATH. פירוט של הסינטקס ופונקציות נמצאים במצגת.

נעב - יקרא הסבר על XPath ושפה מעניינת של ה- (xpqx) להביא את יחס ההצלה בין ביטויים.

XML Storage

אמרנו רמזנו מדברים של היקפים מאוד גדולים ואז אופן שמירת המידע מתבול להשפיע, כמובן של הציבור אבל גם של סוגי הפעולות שבוצעו עליו בינלוי.

יש שלוש אפשרויות מקובלות

- קבצים - כל מסמך הוא קובץ
- לשמור אירשפו את המסמך למסד נתונים רלציוני
- Natine XML Database - מסד נתונים שמרול עצם מה טמלאות.

החיסרון המובהק של שימוש במסד נתונים רלציוני הוא שקשה להמיר את המידע כדי להכניס להיוראי. אבל מצד שני יש כבר תשתיות מוכנות ושנים של מחקר ואופטימיזציות. הבעיה עם Natine XML DB היא שצה תחום שעדיין לא מפותח כל כך ואם אין אופציה אולר מסתרר.

(37) 11/6/09
DB

XML Storage

- ישלנו את כל ה- XML הנהדר הזה, אבל איפה נשים ונשמור אותו?

יש מספר אסטרטגיות בירושות לבניית יעולות

- קבצים - פשוט וסטנדרטי
- אורשפו להכניס את הנתונים למסד נתונים רלציוני
- לבנות מסד נתונים מיוחד ל- XML

אותנו נעזר קצת משנושת האסטרטגיות.

קבצים

בפעם ששאלתם שאלתכו דרוש לעשות parsing של הקבצים.
אין שום אינדקסים שעוזרים לחפש. כל פעם דרוש לעשות parsing
מחדש. יש שני טעמים סטנדרטיים של parsers ל-XML

- DOM - אוסר על מחוק המסמך (מחומש ל' SAX)
- SAX - לא מאיר אל מבנה נתונים אלא שולח הודעה
בכל פעם שהוא רואה אלמנט כלשהו

• DOM - אז יש לנו על איך עונים על שאלות?

- /transaction/buy - בודקים אם השורש הוא transaction ואם כן אז בודקים אם יש לו בן buy (אולי כדאי לבנות parsing כל המסמך רק בשביל לחפש בשורש...)
- /ticker - עושים טוב על כל הדף ומחפשים את כל ה ticker שמופיעים מעל

חסרונות: - די כבדי וקל לשימוש

- מספיק לעשות parsing פעם אחת כדי לענות על הרבה שאלות

חסרונות: - דורש הרבה מאוד זיכרון כי רוצים שכל הדף יהיה בזיכרון הראשי!

נראה בעיה בעקר כי ב-SMX יש אינו נוח
לעדכן מסמכים אמש לקוחות של מאות MB ואז
עבכנות מתבולות לקרוס...
- צריך לעבור על המסמך לפני שאפשר לענות על
שאילתא לראשו, אם השורש לא היה transaction
אפשרי היה לקרוא קבצ מהקובל כדי לבין שהשאילתא
ריקה.

- SAX - עובר על המסמך פעם אחת ומודען כל פעם שהוא
רואה משהו מעניין. איך ענים על שאילתא?
- /transaction/buy - אודאים שה. הראשון
הוא transaction ואז עוקבים עד שנגיע buy
רק שצריך לעקוב אחרי הרמה שאנחנו נמצאים בה.
- בגדול יש הרבה book keeping יעשיר ואם עוקבים
אחרי זה מספיק בנהירות זה לא מסוך מסובך.
מרינות - יאז דורש הרבה זיכרון, מהיר
חסרונות - אי אפשר לקרוא אחורנית

<u>מסך נתונים הלבאני</u>
הן שם העקריים הוא בתרגום ה- SMX לטבלאות
ואז בתרגום השאילתא ל- SQL.
DTD יכול להיות מסוך מורכב שבלא בעיה יתר קשה הוא שטבלה
יש שני מאדם ואילו ב-SMX יכול להיות קינון בעומק בלתי מוגבל.
מלבד זה ב- SMX ... ובלי בעיות ערכים מרובים לאותו attribute
כן יכולה להיות בקלות.

אופציה 1 - אין לנו DTD
נשמור את מבנה העץ בתוך הטבלה. לכל קודקוד נתן id ייחודי,
נשמור לו את הסוג והערך ואת אבא שלו. כך אפשר לשחזר
את מבנה העץ. פגיעה במהירות. נכון שמסמך SMX

הוא סדור לכן נראה לתת ids שמאפשרים לשמור גם
את הסדר של הקודקודים.
אין צורך גם שאילתות?

- /a/b/c/d/e/ - עושים join לטבלה עם עצמה
5 פעמים ומוודאים שמתקיימים כל התנאים הדרושים.
ב- SQL אפשר לעשות גם select אחרון...

- //a/b - כה דומה לי פשוט. עושים join
אחד ומחפשים את כל הילדים שהם a.

- //a/b/e - דרוש לאלו b-ים! - e-ים שיש
מסלול ביניהם. זהו בדיוק הסאב הטרנזיטיבי. והבעיה
היא שאנחנו לא יודעים את אורך המסלול המקסימלי לכן
לעשות סתם join מיותר מאורך לא יצור לנו, כי אולי
מדיים נפספס ב-1. למעשה, אפשר להוכיח שגם אלגוריתם
רקורסיבי אי אפשר לפתור את הבעיה הזאת.
אולם, SQL כה יתר תקן ומסטנדרט הנוכחי אפשר לעשות
שאילתות רקורסיבות ולכן אפשר לענות על כה.

שיפורים

- נאספר את הקודקודים ב- DFS אבוב- BFS ובנוסף
נשמור לכל קודקוד את הגבוה (הסדור) האחרון שלו.
אם רוצים נגיד ל //a/b/e/ ? e הוא בן אבוי של
b אם ה- id של e גדול מ- id של b ל- id
של הבן האחרון של b. כך מתאפשר בעצם ה- DFS !!!
בשביל שאילתות sibling נשמר שהשם הוא
הבן אבוי האחרון + 1.

אופציה 2 - להשתמש ב DTD
מקבלים DTD ומגדירים מאנן סכמה רגיונית. נשים לב
שמתוך האסד ובנים אנחנו רוצים לשמור רק את המידע
לאשל -

```
<book>
    <title> ...  </title>
    <author>
        <firstname> ..  </firstname>
        <lastname> ...  </lastname>
    </author>
    <address>
        <city> ...  </city>
        <country> ...  </country>
    </address>
</book>
```

אז אפשר להסתפק בטבלה עם עמודות
title, firstname, lastname, city, country
לה נתן את כל המידע הדרוש אבל אנחנו מאבדים
את מבנה הקבלה. אז אם רוצים לשמר את ה- SAX
צריך לשמור מידע עם המבנה שלו.

בהגדרת DTD הוא קובץ איזהסשי שאת מגדירים אותו.
בעצם הוא אם בהגדרת DTD לאבטוא סכמה בלבדיות
שיכולה לדבר לאמת שאמאים ל- DTD.
DTD-ים וכללים להגדרת מסמכים. לכן קובעים תהליך
פישוט - D_2 פישוט של D_1 אם כל מסמך שתואם את
D_1 הוא גם כמעט תואם את D_2. כמעט פירושו תואם לא
נדי סדר הבנים.
תנאי פישוט:

$(e_1, e_2)^* \longrightarrow e_1^*, e_2^*$

$(e_1, e_2)? \longrightarrow e_1?, e_2?$

$(e_1 | e_2) \longrightarrow e_1?, e_2?$

$e_1^{**} \rightarrow e_1^*$

$e_1^{*?} \rightarrow e_1^*$

$e_1^{?*} \rightarrow e_1^*$

$e_1^{??} \rightarrow e_1?$

$e_1^{+} \rightarrow e_1^*$

וכן ...

המטרה בחוקי הפישוט היא לגרום שבתוך יהיה רק אופרטור אחד של כל אלמנט ושלא יהיו סוגריים אלא רק אותם אחרים...

דוגמה

(b|c|e)?, (e?|f+)

→ (b?, c?, e?)?, e??, f+?

→ b??, c??, e??, e??, f+?

→ b??, c??, e??, e??, f*?

→ b?, c?, e?, e?, f*

→ b?, c?, e*, f*

אפשר גם להפעיל את החוקים גם לתוך הביטוי המקורי ולראות מה פועלים לכל הדרך. כל אלמנט יהופיע -

(b|c|e)?, (e?|(f?, (b,b)*))*

→ b*, c?, e*, f*

אי אפשר לנשים לפי הגדרה של אלמנט ב- DTD. ורשיון אפשר לייצר גרף ל- DTD.

הקודקודים הם אלמנטים, אטריבוטים ואופרטור (*, ?) כל אלמנט מופיע פעם אחת. ואטריבוטים מופיעים כמספר הפעמים שהם מופיעים ב- DTD.

מהגרף הזה אפשר לייצר את הסכמה. טבלה מיוצרים עבור כל קודקוד אלמנט

- עם דרגת כניסה 0 (כי אין צורך לאן לשים את המידע שלו)
- עם דרגת כניסה גדולה מ- 2 (כי אין מקום טבעי לשים מן את המידע)
- שנמצאת תחת * (כי אי אפשר לתת מספר בלתי מוגבל של רשומה בשורה של טבלה)
- קודקוד שמשתמש במטבל

דוגמה במציאות.

יתרונות לשומרת האובייקט הקודמת

- הרבה יותר יעיל לבנות שאילתות מסוימות כמו שם משפחה ושם פרטי של כתבת מסוים - צריך פחות חיפוש בין טבלאות והטבלאות יהיו יותר
- יעיל לשאילתות כמו "כל הכותבות בשם jack" - לא צריך לעבור על כל הנתונים אלא רק את טבלת אתר.

חסרונות

- עדיין יש שאילתות שצריך לעשות בשביל הרבה חיפוש-ים הלוך ושוב.
- הבעיה כבר לא משתנה באפשרות בדים כדי לקשר בין ילדים להורים? אולי היינו רוצים אבל לה בלתי אפשרי כי יכולים להיות כמה הורים ואין לנו צריך להגיד שמפתח ישתנו (מדובר בלקשרות טבלה אחת לצריך להגיד באיזה טבלה מדובר הא צריך להיות).

- אפליה 3 -

דומה לשיטה הקודמת אבל התוקים לילדות הטבלאות קבוע שונים - פירוש במציאות. השיטה הזו מאפשרת את מספר הצרופים שצריך לעשות. לעומת זאת לפעמים נדרש דרוך בשאילתות אחוד כי מידע יכול להיות מפוזר בין כמה טבלאות.

אמצעי נתונים מיוחדים ל-XML

הרעיון הבסיסי הוא לשמור את המסמך כולל והבעיה המרכזית היא לידור שיטה יעילה לעדכון של שאילתות.

Natix

המתשב זובד בבלוקים אבל מנסים לשמור כמה שיותר
קודקודים קרובים בתוך אותו הבלוק. אנחנו לא נתמקד
באסטרטגיה של הסדר של הדיסק לאחרי שיש זבודה —
מעניינת את הנושא הזה.

אינדקס - נדבר על שתי אסטרטגיות.
- רשימות של אלמנטים ומצביעים -
נשמור שני סוגי אינדקסים -
value - לפי איך שמפתח בפנים יש רשימת קודקודים
שמכילים אותו
element - לפי אלמנט (שם של אלמנט) שמפתח
בפנים נשמור רשימת של קודקודים עם האלמנטים
המתאימים

את הרשימה הזאת נוכל אפשר לדחוס בשיטות שלמדנו
אבל אנחנו לא נתמקד בזה.
structure - זבור איזשהו מבנה סבבי - הקודקודים
המתאימים (למשל זבור שאילתות נפוצות).
לפי זמנים

נניח שיש לנו רק value ו-element. אין אינדקס על
שאילתא exch // buy ? שני אפשרויות:
- לחפש buy ואז לחפש בתוך זה
- לחפש exch ואז לחפש בקנאיים.
מה יותר יעיל? נוכל לבל תלוי בתכונות הסבבי. אבל
בדרך לאסטרטגיה השניה יש יתרון מסוימת כי
בלי שם רשימת התשובות ונפוטנציאלית אינטנסיבית ואולי
באסטרטגיה הראשונה היא גדולה.
מצד שני, אם buy הרבה פחות נפל מ- exch אז
אולי כדאי לדחוף אותו קודם.

האמת היא ששתי האסטרטגיות האלה שרושמות כל בשתיהן צריך לזכור מה כל הזמן, אבל הרעיון הוא שקל מאוד ואולי לא נכנס לסיכום הראשי.

לכן נרצה לענות על השאילתות רק בעזרת האינדקסים.
בשביל זה נעשיר קלטים המידע ששומרים באינדקס.
לכל אלמנט נשמור שלשה (start, end, level)
וזה מאפשר לנו להבדיל קינון, אבל גם שקלוד הוא בן של קלוד אחר.

מטרה: לענות על שאילתות מהצורה $h//m$
אינטואיציה: מעבר ב-DFS על עץ אפשרי לבדוק
בצמוד ינאבי ובשימוש במחסנית שגודלה לכל היותר
גובה העץ. המחסנית מייצגת יחס של אב-קדמון-
צאצא.

תשובות לשיעורי בית-
- בביטוי שהקיטוי $(b|c)^?, b, (b|c)$
לא תצמשמעי כי אם מתחלים ב-b לא ברור
אם זה הראשון או השני מהביטוי.
אחרי שזה מגדיר שפה סופית ברור אם שיש ביטוי
אז משמעי שקול:
$(b, (b, (b|c)^? | c)) | ((c, b, (b|c)), b)$
האסטרטגיה במקרה הסופי היא לתלק את המילים לחלקים
צרים שאין בהם חוסר חד משמעיות
- $*//*/$ שקול ל- $*/*//$ והם שניהם מחציים את
כל הקלוקלים שונים השורש.
- $*/$ המשלים של הביטויים הקלדמים ומחציר רק את השוחש
- $*//$ מחציר את כל הקלוקלים.

- //*[//a]//a[/a][a]

אם מתקיימים מהתנאים הקודמים שצריך להתיר
הם //*//a

התנאי הראשון אומר שיש לשורש צאצא a אבל
אפשר להתקיים מצב כי בלאו הכי אנחנו דורשים
//*//a ועל מה שמקיים אמ"ה רבר ברור
ממנו שיש לשורש צאצא a.
התנאי [/a] דורש שהשורש של הרקע הוא a.
סה"כ זה מתיר קודקודי a שנמצאים בכל מקום
פרט לשורש שיש להם צאצא a ורק אם
השורש של הרקע a

ש"ב - רק חלק ג'

(42) 18/6/09
DB

את טבלת הקודקודים

שבוע שעבר עלינו בבעיה הבאה:
בהינתן שני labels רצינו למצוא גיאור קודקודים שמקיימים קשר אב-קדמון – צאצא או אב-בן.
נניח שלכל label יש לנו רשימת מיקומים של קודקודים עם ה- label הזה. האינטואיציה היא שאם נעבור על הגרף ב- DFS אנו המתסנרים תגיע לכל היותר עד גובה הגרף וקשר אב-קדמון – צאצא מתבטא בכך שהאב הקדמון מוכנס למחסנית לפני הצאצא.
אבל אנחנו רוצים לסמלץ מעבר על הגרף ע"י מעבר על האינדקסים.
באופן מאוד שמגיע את האלגוריתם הזה יש עם גרסה של האלגוריתם שמחזיר תוצאות בצורה ממוינת לפי האב הקדמון. זה עוזר למשל אם רוצים לחשב את $a//b//c$ אז אפשר לחשב את $a//b$ ואז לנסות לחבר אותו ל- c. בשביל זה צריך שהרשימה $*//c$ תהיה ממוינת לפי האב של c ואילו מחשבים את $*//a$ ואת $a//b$ אז צריך ש- $*//a$ תהיה ממוינת לפי הבן של a. אז בקיצור, שני הכיוונים של הסידור חשובים ומי שרוצה יכול להסתכל במאמר. במילים אפשר למצוא את האלגוריתם שממיין לפי צאצא ואז לפי אב קדמון.

איך אפשר להשתמש באלגוריתם דומה כדי לחשב צירוף אחרות? עבור יחס אב-בן זה מאוד פשוט לשנות את האלגוריתם בהתאם.
following זה קצת יותר מורכב... אבל אפשרי (היה פעם זה פעם שאלה במבחן).

כמובן, רפי לתשב שאילתת XPath שמה לה מאוד פשוט להרחיב. מחשבים מה ואז מרחיבים אותו (כמו שאמרנו קודם על a//b//c).

שב - האם אפשר לעשות אופטימיזציה (על אורכם) כאשר הי- XML הוא גרף אלא רק (רשימה שימוש ב- IDREF).

לאמיתי XML יש נטייה לאמבנה שחוזר על עצמו ואת זה אפשר לנצל רפי לתיאור בעזרת אות המסמך עם מרכל בשם dataguide. נתמקד כרגע בשאילתות מסלול- שאילתות מהצורה a//b//c//d//...

ב- dataguide כל קודקוד הוא למעשה אינדקס שמצביע לכל הקודקודים המתאימים במרכל הגדול. ואז רפי לחסוך אפשר לחפש את המסלול במרכל הקטן ואז הקודקודים שנמצאו מו נמנים לנו את התוצאה עבור המרכל הגדול.

פורמלית:

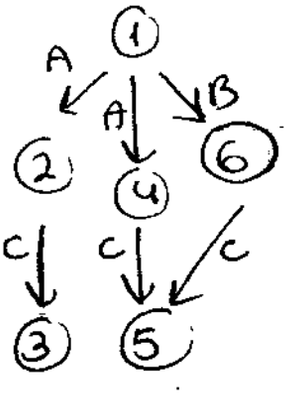
אם X מסמך XML נאמר ש- D מרכל הוא dataguide של X אם:

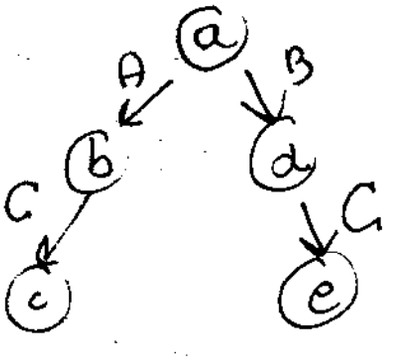
- כל מסלול של labels ב- X מופיע בדיוק פעם אחת ב- D.
- כל מסלול של labels שמופיע ב- D מופיע לפחות פעם אחת ב- X.

* מסלול מהתחלתו תמיד מתחיל בשורש.

נניח X מסמך XML ! D - dataguide.
יהיו p, p' שני מסלולים.
נסמן ב- p(X) את קבוצת הקדקודים שמגיעים אליהם ב- X כאשר הולכים על המסלול p.
נאמר ש- $p \equiv_X p'$ אם $p(X) = p'(X)$. למעשה, זה אומר שב- X אין אפשר להבחין בין p ל- p'.
נאמר ש- D הוא dataguide עבור X אם לכל זוג מסלולים p, p' אם $p \equiv_X p'$ אז גם $p \equiv_D p'$ ואם $p \equiv_D p'$ אז גם $p \equiv_X p'$.
זה אומר ש- D חד-משמעי במובן שתאינדקס בעזרתו מלבים על מה שצריך.
לצערנו dataguide עבור אלו להיות מאוד גדול (מעריכי). אבל אם ה- XML הוא עץ אז גודלו הבעייתי יכולה להיות בזמן לינארי.

במציאות נמצא אלגוריתם יצירת ה- strong dataguide דוגמה:





$(\{1\}, a), (\{2,4\}, b), (\{3,5\}, c), (\{6\}, a), (\{5\}, e)$

$p = \{(A,2), (A,4), (B,6)\}$

A: $t_2 = \{2,4\}$ נאנמצא מפתח hash מאיזרים b

Recursive Make $(\{2,4\}, b)$
$p = \{(C,3), (C,5)\}$
C: $t_2 = \{3,5\}$ נמצא hash מפתח יוצרים c

B: t_2 - {6} לא נכנס hash
אז יוצרים d

Recursive make

p = {G, 5}
G: t_2 = {5} לא נכנס hash
אז יוצרים את
e

תסרוקות:
- יכול לבנות מאוד גדול
- מתאים רק לשאילתות מסלול

מגרעות: אם ה- dataguide מכל מאר נכנס לזיכרון אז
זה מאוד יפה (ולזה עמדה שקורה בדב כי XML הוא לרוב
רק אם המון תגיות)

ש"ב - ל"ד strong dataguide לגרף
אחרת דוגמה לשאילתא שמקבלת תשובה לא נכונה
בשימוש בו.

שאילתות מעל XML

דרוש לאפשר לאנשים לשאול שאילתות בשפה יותר ברורה -
keyword search (סמנטי בנוסף לא תמקבל שאילתות
XPath).

הנה דרך פשוטה להגדיר חיפוש מילות מפתח - גם רשימה של מונחים
לחיפוש שיכולים להיות מ- 3 דרכים
- label וערך : author: smith
- ערך : smith:
- label : author:

קשיים:

- להבין מה להתייחס למשתמש. ב- XML בדרך הכוונה היא איזה תת עץ שמתאים לשאלה. למשל, אפשר להתייחס את תת העץ האינדקס שמכיל את מונחי החיפוש.
- להימנע מלהתייחס לאלמנטים שאין להם שום קשר אחד לשני. בשביל זה צריך לדעת למה המשתמש מתכוון...
- איך לעשות פירוט של התוצאות - כי בעזרת שפות לחלוטין זה בעיה והבעיה היא שאין מספיק מסמכי XML כדי לעשות User study מהתלמיד אין שיטה באמת טובה.

XRank

שאילתות הן מילות חיפוש כמו במנוע חיפוש רגיל.
תוצאות של שאילתות הן חלקים של ה- XML שמכילים את מילות החיפוש והכי קטנים שיכולים להיות.

ב- HTML נותנים פירוט לכל מסמך. ב- XML נרצה לתת פירוט לכל אלמנט. אז הרעיון הוא להרחיב את שיטות הפירוט שאנחנו מכירים כך שאפשר יהיה לדרג אותן XML.

מה רוצים לקחת בחשבון?

- לינקים בין רכיבים
- קרבה בין אלמנטים

אנחנו נתמקד בפירוט לפי לינקים.
איזה לינקים יש במסמך XML?

- אב שמצביע לבן
- בן שמצביע לאב
- קשרי IDREF
- קשרי XLINK (הצבעות למסמך XML אחר)

הרעיון זה דומה ל- PageRank

יש מקרה של מסמכי XML. כל אלמנט במסמך הוא קודקוד.
הבעיות מיוצגות בקשתות לעבור בין אלמנטים - אב לבן, בן לאב, hyperlinks או רפרנסים לקבוץ.
ולבסוף מתקבלת נוסחה דומה לנוסחה של Page Rank.

השלכות:

- נניח שיש מסמך שיש לו מספר חלקים אבל הרבה רפרנסים. אבל רק שיש יותר רפרנסים החשיבות של חלק יורדת. אבל זה אפשרי לתקן זי שנסתכל על לינקים חיצוניים ולינקים מבנים אחרת וניתן להם הסתברויות שונות.
- למעשה, אולי אם נרצה להתייחס ללינקים אב-בן בשונה מלינקים של בן-אב

כמובן, אפשר לתת לזה אינטרפרטציה של הלוך מקרי.
ה-rank של אלמנט e הוא ההסתברות שנגיע ל-e אם נתחיל בקודקוד מקרי ואז
- בהסתברות $d_3-d_2-d_1-1$ נקפוץ לקודקוד אקראי
- בהסתברות d_1 נעבור ללינק חיצוני
- בהסתברות d_2 נעבור לבן
- בהסתברות d_3 נעבור לאב

האם השיטה הצליחה הגיונית?
- נניח ש-IDREFS מתבטאים מסמכים למאמרים שהם מצטטים. נראה די סביר שאם אישהו חשוב מצטט אותך אז גם אתה חשוב.
- IDREFS מתבטאים בעזרת מקיון לא המחלקות שלהם - אישהו מקיון חשוב אם המחלקה שלו חשובה מלהפך - לא הבר מתחול להיות רופא.
- IDREFS מתבטאים ארצות לא מדינות שכנות. אבל מדינה גשובה אם היא מוכרת בהרבה מדינות חשובות. ומה לגבי אוסטרליה?

אינדקס

רוצים להתיר את האלמנטים הכי ספציפיים עבור מילות חיפוש בשאילתא.

שיטה נאיבית היא להתייחס לאלמנטים כמסמכים ולעשות inverted index רגיל. הבעיה כאן היא שצה תופס המון מקום אבל מצמר לצה סתם חיתוך רשימות לטוראפשר להתיר את האלמנטים היותר ספציפיים.

Dewey Encoding - אפשר לתת לקודקודים מספר כך שאלמנט A הוא אב קדמון של B אמ המספר של A הוא prefix של המספר של B

(46) 25.6.09
DB

מה שלא נספיק ללמוד מהשקפים לא יכלל בחומר לבחינה.

אנחנו מדברים על key word search מול XML.

קבלו שתי שאלות

- איך להחליט מה להחזיר למשתמש?
- איך לדרג את התוצאות?

תשובות

- קלבקלדם n יכן שם נוריך 1-n את הלאלאים שלו שמרלים את b מלות המפתח, n ילדים יכול את b מלות המפתח
- האינו איך לשנות את PageRank כדי שיתאים ל- XML

היום נלמד איך למצוא את b הקלדקלדים הרלוונטיים בצורה יעילה.
כמובן אפשר לנסות בשימוש באינדקס מסבר ויהיה מאוד נחמד אם נצטרך לעבור עליו רק פעם אחת.
בשביל שזבר דעכנו על Dewey Encoding - דרך לאפיין את ה- nodes בעל כך שנוכל לדעת מ המאפיין יחסים של אב-קדמון - צאצא בין הקלדקלדים. אלה ה- ids של הקלדקלדים ואז אפשר לבנות inverted index לפי מלה.

אבל אם יש לנו שאלתא עם כמה מלים נבנה טבלה:

node ID	contains word 1?	...	contains word m?	contains all?

במילים יש פסאודו קוד לאלגוריתם שממש את הטבלה בצורה יעילה.

המשך הדוגמה מהמילים:

deweyID	XQL	language	all
1	✓		
~~0~~	~~✓~~	~~✓~~	~~✓~~
~~2~~			
~~0~~	~~✓~~	~~✓~~	~~✓~~
0			
47			

אחרי שאמרנו ש- 47.0.0.2.0 מקיים את מה שצריך אנחנו מגיעים ל- 47.0.1 והתחלים המשמעות הארוכה ביותר היא 47.0 לכן מוחקים את השורות של 0.2.0 ומכניסים שורה ל- 1 ומסמנים רק שהוא מכיל את XQL.

שאילתות לא מדויקות

השיטה הנ"ל מועדפת לאנשים שיודעים לכתוב שאילתות ב- XPath וכו' אבל יכול להיות שהם לא מדויק יודעו את המבנה של הקובץ ואז אולי בטעות השאילתא פשוט לא מתאימה ל- XML. למשל, ב תוצאות הספרים הבדוקות שמור מסמך XML יכול ב- web אם לא המידע הרלוונטי. בגלל שלא הגיוני לדפוס שלא המסמכים והיה אותו מבנה אם לא הגיוני לכתוב שאילתא שונה לכל מסמך כי זה לא scalable. אז אנחנו נוכל שנדע לענות על שאילתות שדומות לשאילתא מתאימה.

צריך לאפשר למשתמש לכתוב שאילתא "מקורבת" וה- query processor צריך לדעת להתמודד עם זה.

יש הרבה מחקר בנושא ויש המון הגדרות ואלגוריתמים שונים.
אבל יש רמה מסוימת משותפת
- להרשת שלחלק מה קודקודים בשאילתא לא תהיה התאמה
- להרשות לאסוף בשאילתא להתאים לכל ע. (למשל במקום לחפש בנים אפשר לחפש גולגולים)
- לאפשר להתאים קודקודים עם משמעות דומה (למשל film בשאילתא לעומת movie ב- XML).

אנחנו נלמד מאמר ספציפי (שמו במצגת).
שאילתא היא לדשה לך. המאמר מאפשר ערבוב סוגי הקלות של השאילתא.
- הטלת קודקודים - אם יש לנו מידע אונטולוגי ויש לנו מידע על יחס type / super-type ואז אפשר להתאיל קודקוד בשאילתא בקודקוד אחר יותר.
- מחיקת עלה מה עץ (והבן שנכנס לתוכו
- החלפת יחס אב-בן ביחס אב-קדמון - גולגולו
- קידום תת עץ - לקחת בן להפוך אותו לבן של השכנו

את ההקלות האלה אפשר לשרשר בכל מיני דרכים.

אבל תשובה משודרגת היא תשובה מדוייקת להקלה כלשהי של השאילתא. אבל לא כל ההקלות טובות באותה מידה. אם לכל קודקוד וכל בן יש שני ציונים - למקרה שהתשובה מדוייקת ולמקרה שהתשובה משודרגת. וכך לכל תשובה אפשר לתת ציון ולהחליט מה יותר טוב. את הציונים נותן המשתמש.

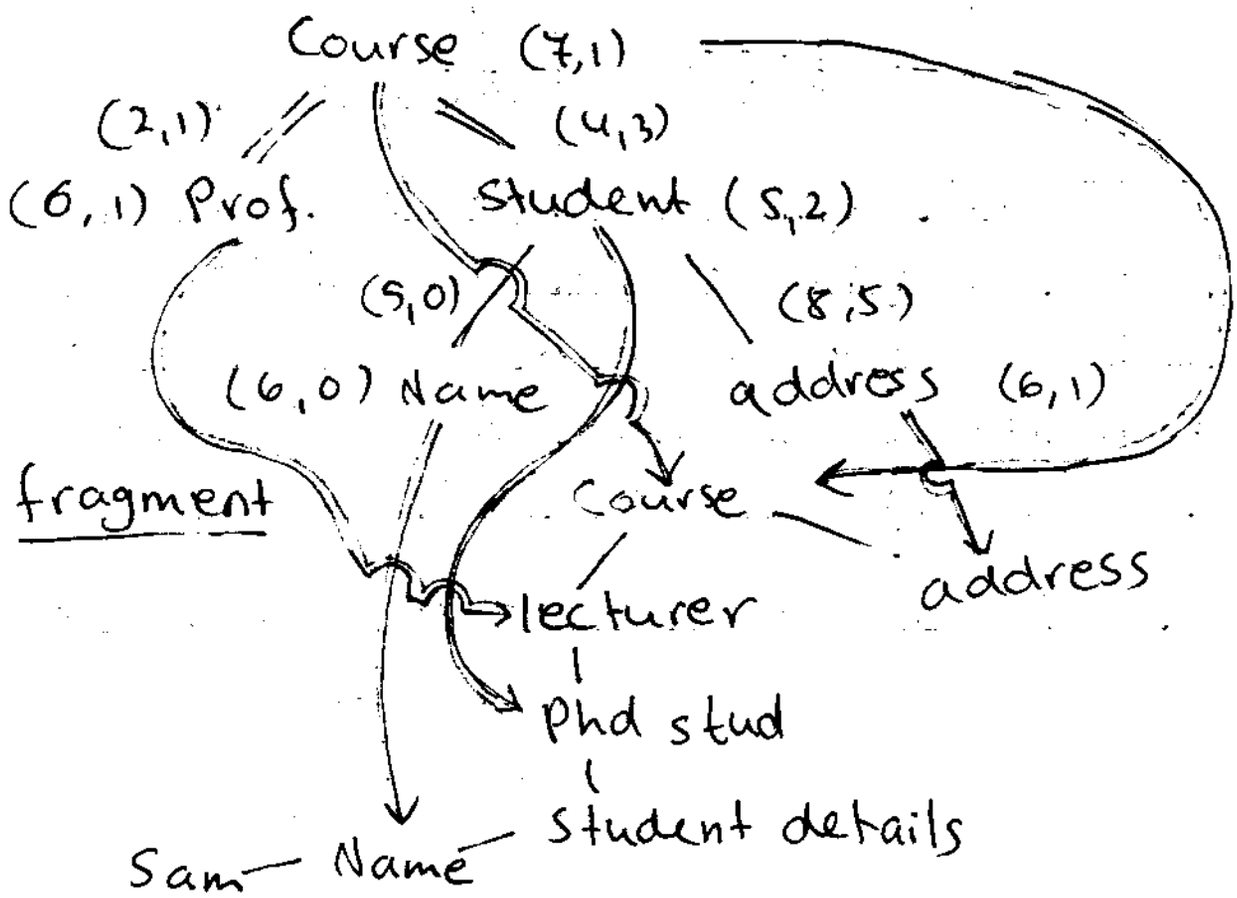
אישה נאיבית: לרדוא את כל ההקות, לכל אחת לחשב את הציון ולהחזיר את הטובה ביותר.

אתנתתא אדמיניסטרטיבית -

שפה עיך לא החיטה ליבי אבנה הבתינה אובל הפילוסופיה
שלת הנאו בכאר: אבתן ליבך לבול שאושפ סגי שאלות
- שאלות אוטומטיות - סתם דברים שליבך לדעת פשוט כי
ישבנו בקורס
- שאלות תרגיל - עשינו משהו בבית, הפעלנו אלגוריתם
ורשינו ליבך לעשות אותו הדבר אבל עם דוגמה אחרת
- שאלות הבנה - לקחת משהו שלמדנו בבית ולהשליך אותו
על מקרה שונה. אם באמת מבינים היטב את מה שלמדנו
בבית אז זה לא אמור להיות מאוד קשה.

הדרך הנכונה ללמוד בבית היא לעבור על כל מה שנלמד
בבית ובתרגולי הבית ולפי פעם שנתקעים באלגוריתם פשוט
להפעיל אותו על דוגמה ווודא שמבינים.

דוגמה:



הבאו לבכון אתגלגון הבי שלא יהיו טעויות חישוב.

יש ל- Prof תת טיפוס של Lecturer
Phd stud תת טיפוס של Student

יש רק דרך אחת לחפוש את הקשרים. מה שמלוכר בעמוד הקודם.
מה הדרך?
- אם ה course מקבלים 7
- אם ה Student מקבלים 5 כי phd-stud הוא טיפוס של Student.
- אם Prof קיבלנו 1 כי lecturer הוא תת סוג Prof
- על name קיבלנו 6
- אם Address מקבלים 6
- על B ע course-student ועל course-prof קיבלנו דרך אחת - 4 ; 2 כי באמת יש יתרון שזה אם קדמון (לא משנה ש- prof קיבל קודם דרך חלקי)
- אם student-name מקבלים דרך חלקי כי יש יתרון שלא קדמון אלא אב ממש.
- על address-student לא מקבלים כלום כי הוא לא קיים בטבלה.

בשאילתות לעדכון QueryPlan נותן את רשימת ה joins שדרך עצות כדי לענות עליה. ל Plan תכנה דרך להוסיף את ההקלות - פירוט מחלקה.

איך נמדל אתם התשובות שמודות בדרך סלם אסוים?
לפי הקודקוד בתבניות החישוב ניתן דרך איזה סך הנקודות המקסימלי שאפשר להשיג לשאר הקודקודים.
רק אפשר לצרוך תוצאות ביניים מהחישוב כי אותן יודעים שכבר אין להן סיכוי להגיע לסוף הדרוש.

זה נכון הסתכלנו במקרה שהמסמך והשאילתא הם זהים.
כלומר זה נכון במצב שאשתמש רק ב- node deletion
אבל שם המסמך והשאילתא יכולים להיות יצא לגמרי - אין מכוון
מתנים עם שורש.

שאילתא נותנת אוילרים, השהיא מדיקתהם חייבים להתקיים
- אילוץ שורש- שורש השאילתא חייב להיות שורש הגרף
- אילוץ בלא - יחסי אב-בן שחייבים להתקיים
בשפת הגרפים זה נקרא subgraph isomorphism
שורש מתאים לשורש ובלא מתאימה לבלא.

אנחנו נרצה לאפשר לחת תשובות חלקיות - מיפויים שנותנים
לחלק מהדרכים NULL, אבל דריך לעשות אותה בדרך
הגיונית וסבירה.
בשפוא זה יש אילוץ ישיבות - לא מאפשרים דברים סתם
בדרכה תלושה מהמציאות. אפשר למפות קודקוד v ל אובייקט
o אם יש מסלול מהשורש של השאילתא v רק שכל
אילוצי הדרך על המסלול מסופקים.

תשובה חלקית היא מסוים AND אם אילוץ השורש מתקיים
ואילוץ הישיבות מתקיים לכל קודקוד בשאילתא שממופה לקודקוד
במסמך נתון. כמו כן, אם קודקוד ממופה אבל חלק הדרך
הנכנסות שלו דרכות להיות ממופות - כלומר אחד מההורים הנכנסים
שלו.

בלא היא מסופקת חלש אם היא מסופקת או שאחד או יותר
מהקודקודים שלה ממופה ל- NULL.
מיפוי חלקי הוא מיפוי חלש אם השורש מסופק, אילוץ הנישות
מתקיים לכל קודקוד ממופה וכל בלא בשאילתא מסופקת חלש.

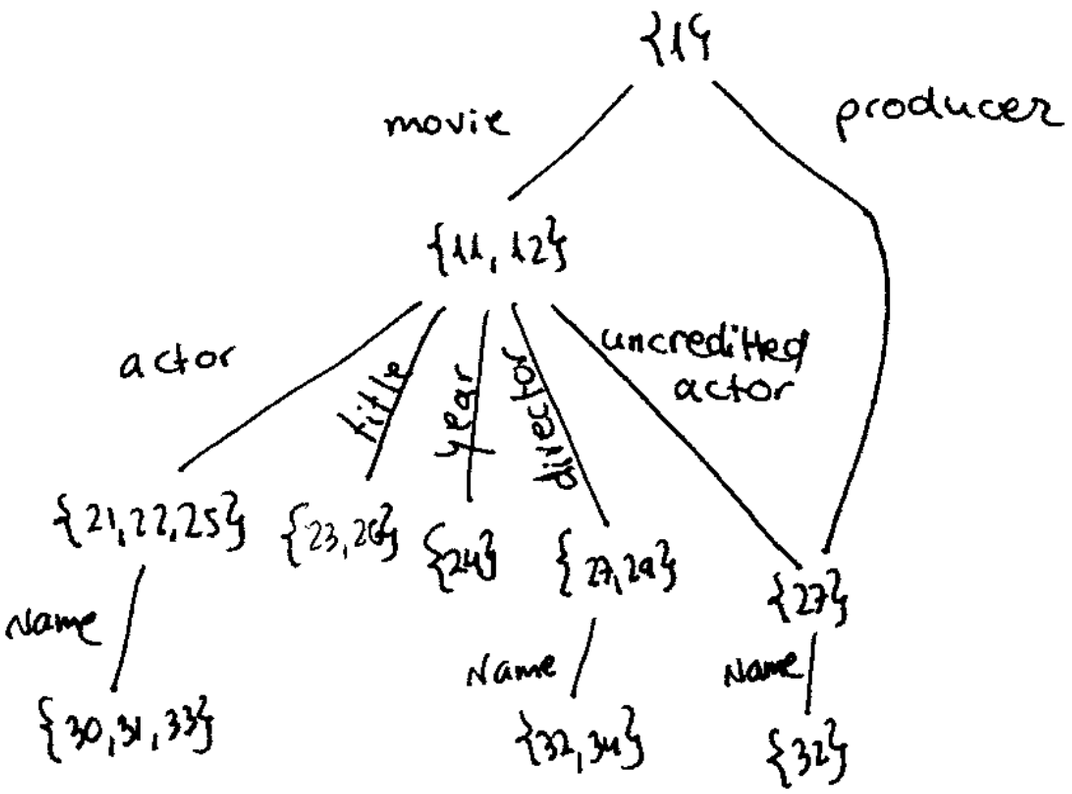
תשובה יחלקית היא אמיתית. OR אם היא מספקת את
אילוץ השורש ואילו הנגישות.

אנחנו רוצים להתאים אדם לשחקן המתאים. נאמר שמישהו תואם
מקסימלי אם לא קיים מישהו אחר שמתאים אותו ובנתוך תשובה
לשאילתא אנחנו נחפש רק מישהו מקסימלים.

תשובות לשב"

(1) ראינו איך להשתמש באינדקס שמאפשר ב- DFS רפי למצוא
יחס של קונקטור - גאנגא. האם אפשר להחיות את הקלות למקרה
של גרף כללי ולא עץ? האינטואיציה היא שלא כל האלגוריתם
מתבסס על זה שאפשר למצוא רק שורש יחס
קונקטור - גאנגא. זה לא אפשרי בגרף כללי.
ישנם מחקרים שמתנאים מסוימים רק מנסים לפתור את הבעיה.

(2) דרך לחשוב strong data guide לעץ נתון.



(תמונה: בשאילתא /movie[year]/title

בשאר שלנו year אפשרי בלי אותו ב- 24 יש לו
את title ב- 23. אבל ה- data guide למעשה
עדיין לנו יתרון שובה מעה שלריך.