שפות תכנות - ש.ב. 3

עמרי גיא 065982415

גיא שקד 036567055

12 בדצמבר 2010

.1

.2

(א) הסיבוכיות של $a@b$ היא $O(\text{length}(a))$. לכן הסיכוביות של האיטרציה הראשונה של nrev היא $O(n)$, האיטרציה השניה - $O(n-1)$ וכך הלאה. בסך הכל סיבוכיות זמן הריצה של הפונקציה -

$$O(n) + O(n-1) + \ldots + O(1) = O\left(\frac{n(n-1)}{2}\right) = O\left(n^2\right)$$

(ב) סיבוכיות זמן הריצה של $irev$ כפי שכתבנו - $O(n)$.

השתמשנו בפונקציית עזר (למימוש רקורסיית זנב) המקבלת שתי רשימות ומחזירה את הרשימה השניה ובסופה הרשימה הראשונה הפוכה. כל צעד של רקורסית הזנב מעביר איבר אחד מתחילת הרשימה הראשונה לסוף השניה, וכאשר הרשימה הראשונה ריקה - השניה מוחזרת כפלט.

על מנת להשתמש בפונקציית העזר הפונקציה $irev$ קוראת לפונקציית העזר עם הרשימה שהתקבלה כקלט בתור משתנה ראשון ורשימה ריקה בתור משתנה שני.

.3 על מנת לממש את $compList$ השתמשנו בפונקציית עזר בשם $comp$ המקבלת שתי פונקציות $f, g :' a \to' a$ ומחזירה פונקציה שהיא הרכבה של f על g. כלומר -

$$comp(f,g)(x) = f(g(x))$$

כעת $compList$ מרכיבה את f על הרשימה שהוחזרה מ-$compList$ עם אותה הפונקציה ו-i קטן באחד, ומשרשרת את f בתחילת הרשימה שהתקבלה.

.4 הפונקציה $mySort$ ממיינת רשימה על ידי ביצוע pivoting. בכל איטרציה האיבר הראשון ברשימה נבחר כ-$pivot$, אנו משתמשים ב-$filter$ כדי לפרק את הרשימה לאיברים הקטנים וגדולים מה-$pivot$ ומפעילים את הפונקציה רקורסיבית על כל אחד משני החלקים, האיבר המקורי מוכנס בין שתי הרשימות.

.5 השתמשנו בפונקצית עזר בשם $union$ המקבלת רשימה של רשימות, ומחזירה רשימה שהיא שרשור הרשימות.

הפונקציה $getPaths$ בונה את רשימת כל המסלולים מ-v_1 ל-v_2 על ידי קריאה רקורסיבית לעצמה - בכל איטרציה אנו יוצרים "מסלול" מכל אחת מהקשתות היוצאות מ-v_1, ולכל מסלול כזה משרשרים את כל המסלולים המגיעים מהצומת אליו הוא הגיע ועד ל-v_2.

על מנת למנוע היווצרות מעגלים, בכל קריאה רקורסיבית כזו אנו מסירים מרשימת הקשתות את כל הקשתות הנכנסות ל-v_1 (ממנו יצאנו). הרקורסיה תעצור כאשר $v_1 = v_2$ (אז המסלול היחיד שאינו מכיל מעגלים הוא כמובן $[v_1]$).

.6

.7

(א) קיימות שתי שיטות לבניית string literals בשפת go -

i. שיטת **Interpereted** -
על ידי שימוש ב־double qoutes. למשל - "Hello world". ב־string literals כאלו ניתן להכניס תווים מיוחדים על ידי שימוש ב־escape charcter מסוג קו נטוי, למשל - "FirstLine \n SecondLine", בדומה ניתן להכניס למחרוזת כזו את הסימן " על ידי - "\"World\" Hello". כל תו יוניקוד ניתן להכניס על ידי שימוש ב־$U\backslash$ ואחריו קוד היוניקוד שלו, למשל עבור הסימן ש"ח שקוד היוניקוד שלו הוא 20AA ניתן לכתוב - "\U000020AA".

ii. שיטת **raw** -
על ידי שימוש ב־apostrophe. למשל -'Hello world'. ב־string literals כאלו לא ניתן להכניס תווים מיוחדים (כל תו שנכתב בין ה־apostrophe יתפרש כפי שהוא), ובפרט לא ניתן להכניס את הסימן ' (כיוון שמופע שלו יסיים את המחרוזת). כיוון שכל תו מתפרש כפי שהוא ניתן להכניס "ירידת שורה" פשוט באמצעות "אנטר" באמצע המחרוזת.

כאשר מדובר במחרוזת ארוכה, המתפרשת על פני מספר שורות, ניתן לשרשר מחרוזות המתפרשות על פני מספר שורות באמצעות + בין השורות (המחרוזות) השונות. "ירידת שורה" באמצע מחרוזת כזו צריכה להתבצע באופן מפורש כפי שתואר לעיל.

קוד מקור בשפת go נכתב בקידוד UTF-8, בקידוד זה כל תו מיוצג על ידי מספר שונה (ולא חסום) של בתים. טיפס char הרומז (כמו בשפות תכנות רבות אחרות) על מימוש בן גודל קבוע בשפה שבה מראש משתמשים בתווים שהייצוג שלהם לא חסום עלול להביא לתוצאות לא הגיוניות ולא סבירות (תו שיכול להופיע בקוד, אבל לא בטיפוב char). לכן go אינה מציעה את הטיפוס char אלא מספר טיפוסים חלופיים - unicode_char, unicode_letter, unicode_digit.

(ב) אורך המערך ב־go ידוע בזמן קומפילציה, מכאן שהמערכים ב־go הם סטטיים. מערך הוא לא גמיש (ניתן להציב מערך אחד לתוך השני רק אם הם זהים בגודלם, ולא ניתן להגדיל מערך על ידי הצבה מעבר לגבולותיו). ומכאן שהמערכים ב־go הם **Static arrays**

בשפת go קיימים מערכים רגילים (בשם array) האינדקס למערך הוא רק מספר שלם גדול או שווה לאפס (או טיפוס שקול) מערכים אלו הם **Ordinary arrays**.

נוסף למערכים שתוארו לעיל יש בשפת go גם מערכים אסוציאטיביים (בשם map) שבהם האינדקס יכול להיות מכל טיפוס, ובפרט מטיפסים שאין עליהם יחס סדר או שאי אפשר להגדיר עליהם "טווח" של ערכים דיסקרטיים, מערכים אלו הם Generalized (Associative) Array, ובפרט הם בהכרח flexible array.

(ג) שני טיפוסים פרמיטיביים T1 ו־T2 ב־go יהיו שקולים אם הם ההכרזה (decleration) עליהם זהה, בפרט אם יש להם אותו טיפוס (פרמיטיבי). חריגים הם טיפוסים פרמיטיביים שהם שני שמות שונים (alias) לאותו הטיפוס. למשל - uint8 ו־byte הם למעשה אותו טיפוס ולכן שקולים.

שני טיפוסים שאינם פרמיטיביים יהיו שקולים אם הם בנויים מאותו סט של משתנים פרמיטיביים. בפרט -

i. פונקציות הן שקולות אם הן מקבלות את אותו מספר משתנים, מטיפוסים שקולים, ומחזירות משתנה מטיפוס שקול.

ii. מערכים הם שקולים אם טיפוס הבסיס שלהם שקול ומספר האיברים בהם זהה.

iii. טיפוסי slice הם זהים אם טיפוס הבסיס שלהם זהה (אך ללא תלות במספר האיברים).

iv. מבנים (struct) זהים אם הם מכילים את אותם שדות (בעלי אותו טיפוס, שם וסדר). שדות חסרי שם נחשבים זהים.

v. מצביעים לטיפוסים שקולים הם שקולים.

vi. שני ממשקים יהיו שקולים אם הם מכילים את אותו סט של מתודות עם שמות זהים והכרזות על פונקציות שקולות, סדר ההכרזה (על פונקציות שונות) לא משנה.

vii. שני מערכים אסוציאטיביים (map) שקולים אם יש להם טיפוסים שקולים בתור מפתח וערך (key, value).

viii. שני channels יחשבו שקולים אם יש להם אותו טיפוס ערך (value) ואותו כיוון.