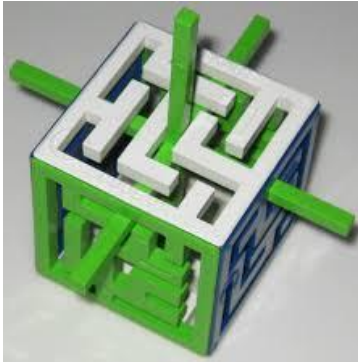


גנרציה של מבוך תלת- מימדי (Oskar's Cube) – ג'יי טננבאום – 323639476



תקציר - Abstract:

המבוך Oskar's Cube הינו מבוך תלת- מימדי שמורכב מ-6 מבוכים הממסגרים תיבה חלולה, ומצלב תלת- מימדי. את הצלב יש להזיז בתוך המבוך מנקודת ההתחלה התלת- מימדית לנקודת הסיום, כאשר מבוכי המסגרת מאלצים את כיווני התנועה של הצלב בכל מצב. המבוך מתואר במאמר הבא [Oskar's Cube](#) (העתק- הדבק לדפדפן) שפורסם ב-Scientific American. מטרת הפרוייקט לגרנט באופן אקראי מבוך מסוג Oskar's Cube ופיתרון מתאים, ובמימדים שונים, ולייצא את המבוך אל קובץ STL, שניתן להדפסה תלת- מימדית ויהיה שמיש.

האלגוריתם היוצר את המבוך הוא אלגוריתם מבוסס DFS, כאשר עיקר האתגר הוא לקחת בחשבון את זה שביצירת המבוך יוצרו מבוכים מושרים [המבוכים של הפאות] חוקיים, וכיצד ליצור את קירות המבוכים תוך דאגה שלא יהיו "חלקים מרחפים" ושיהיה ניתן להדפסה כיחידה אחת פונקציונלית, שהחלק נע בקלות.

תיאור התוכנה – קלט/ פלט

קלט: מימדי המבוך (כאשר עובי הקיר 0, לפני הניפוח - יוסבר פורמלית בהמשך) לאורך הצירים X, Y, Z , ואופציונלית אורך מסלול הפיתרון המינימלי [ננסה 1,000 פעמים ליצור מבוך עם מסלול פיתרון גדול שכזה ואז נפרוש אם לא נצליח].

פלט: בתיקיה **outputs שתיווצר ב-directory של קובץ הפיתרון שרץ ייווצר:** קובץ printableMaze המוכן להדפסה תלת- מימדית (STL) של מבוך אוסקר תלת- מימדי במימדי הקלט [ב-3 פאות סמוכות, נקודת ההתחלה תחומה ע"י 4 קוביות בולטות, והסיום נתחם ע"י 4 קוביות חצי- בולטות], קובץ STL בשם solutionMaze שמכיל את המבוך עם מסלול הפיתרון [ה- trace של מרכז הקרוס מההתחלה לסיום], וקובץ בשם outputSequence שמתאר את הפיתרון של המבוך בטקסט.

סקירת ספרות

התחלתי במחקר אודות אופן היצירה של מבוך אקראי. יש כמה דרכים נפוצות לעשות זאת.

- **עבור גרפים:** נדמיין גרף קשיר, כאשר כל קשת מייצגת קיר בין הצמתים עליהם חלה. יצירת מבוך מעניין היא למעשה מציאת תת- גרף שקשה למצוא מסלול בין 2 צמתים [ההתחלה והסוף]. אם הגרף אינו אציקלי, אז ייתכן מספר דרכים בין זוג הצמתים הנבחרים, ולכן גנרוס המבוך שקול למציאת עץ פורש אקראי. אופן יצירת המבוך הוא במעבר כלשהו על הגרף, דוגמאת DFS, כאשר לכל קשת, כאשר עברנו עליה למעשה "שוברים את הקיר" על קשת זו, ומפנים את הקשר בין קודקוד המוצא והיעד.
- **חיפוש DFS:** נדמיין שהמרחב הוא גריד [מלבני] גדול של תאים [דוגמאת לוח שחמט], ולכל תא יש 4 קירות. נתחזק מבנה של מחסנית שתחילה תכיל תא אקראי. בכל צעד, ניקח את התא הנוכחי כתא בתחתית המחסנית, נבחר תא שכן אקראי שטרם ביקרנו בו, ונסיר את הקיר ביניהם, נסמן את התא החדש שביקרנו בו, ונוסיפו למחסנית לשם backtracking. כאשר אין שכנים חדשים שטרם ביקרנו בהם, מתייחסים כאל נקודת dead end, ונבצע חזרה לאחור עד לתא שיש לו שכן שטרם ביקרנו בו, וניצור junction חדש בתא זה. התהליך יעצור כאשר ביקרנו בכל תא, ונגיע ב-backtracking חזרה לתא הראשון, בידיעה שביקרנו בכל תא. אלג' שכאלה יכולים לא ליצור הרבה junctions, ולהיות בעלי מסדרונות ארוכים, שהרי ממצים עד הסוף כל ענף עד ל-backtracking.
- **קרוסקל אקראי:** עבור גריד מלבני של תאים, ניצור רשימת כל הקירות בין תאים סמוכים, וקבוצה עבור כל תא, שיכיל אך ורק את עצמו. כעת לכל קיר בסדר אקראי: אם שני התאים הסמוכים לו הם בקבוצות שונות, נוריד את הקיר הנוכחי, ונאחד את הקבוצות של שני תאים אלו. אפשר לממש זאת עם union find, בזמן כמעט לינארי. כיוון שהוא יוצר עפ"מ לגרף עם משקלים זהים, האלג' נוטה לבצע תבניות חוזרות, שיחסית קלות לפיתרון.
- **שיטת החלוקה הרקורסיבית:** עבור גריד מלבני של תאים, מתחילים עם מבוך ללא קירות, נקרא לו חדר. מחלקים את החדר עם קיר [או קירות] אקראי, כאשר לכל קיר ממקמים באקראי פתח למעבר. חוזרים רקורסיבית על כל תת- חדר, עד שכל חדר בגודל מינימלי [לדוג' שכל חדר בעובי תא יחיד בכיוון כלשהו מה-2]. שיטה זו מביאה למבוכים עם קירות ארוכים, וקל יחסית לראות לאילו חדרים עדיף לא להיכנס.

לצרכים של פרוייקט זה, צריך ליצור מבוך בתלת- מימד, ולהכליל את אחד האלגוריתמים שלעיל לתלת –מימד. אנו מסתמכים על ההבנה שאם לכל פאה ניצור מבוך דו- מימדי אקראי, המבוך שיתקבל יכיל רצפים חוקיים קצרים מאוד, והפיתרון של המבוך יהיה קצר [כיוון שמהר מאוד לא יהיה לאן לזוז, כי צריך ששלושת המבוכים בפאות השונות יסכימו לגבי הצעד הבא, שזה קורה בסבירות נמוכה יחסית]. אציין שיש באינטרנט כלים שיוצרים מבוכים דו- מימדיים כדוגמאת [A](#), או 3D/2D כדוגמאת [B](#).

תיאור האלגוריתם בפירוט

אנו מתבוננים במבוך מסוג Oskar's Cube. האתגר שאותו פתרתי היה גיורט אוטומטי ורנדומלי של מבוך שכזה. מעבר ליצירת מבוך אוסקר חוקי כלשהו, אנו מעונינים שהפיתרון יהיה מסלול "מעניין" [ארוך מספיק] בתוך הקובייה, כך ששלושת המבוכים של 3 פאות סמוכות יסכימו עם מסלול זה ויאפשרו ל-cross לבצע אותו. יש לציין שזו בבירור משימה יותר מעניינת ומורכבת מגנרט מבוך תלת-מימדי רגיל בגריד. הגעתי ל-3 אבחנות שסייעו לי בקבלת אינטואיציה והבנה כיצד ליצור מבוך שכזה שניתן להדפסה:

- שמתי לב שלמעשה כל פאה של המבוך של אוסקר היא גריד של משבצות, שחלקן חלולות וחלקן מלאות.
- בפיתרון של המבוך, בכל צעד בפיתרון יש להזיז את הקרוס בדיוק 2 משבצות לכיוון מסויים.
- לכל פאה בקוביית אוסקר, כש"מכווצים" את המשבצות לקירות עם עובי 0, מתקבל מבוך רגיל במימדים בערך פי 2 יותר קטנים מהמקורי $[size_{deflated} = \frac{1}{2}(size_{inflated} + 1)]$. ניצור מבוך deflated עם קירות בעובי 0, וננפח למבוך הפלט שניתן להדפסה.

לכן השתמשתי בוריאציה על בניית מבוכים בעזרת DFS באופן הבא:

1. מתחילים מנקודה התחלתית תלת-מימדית כלשהי initialPoint [אצלי זה מקודד כמרכז הפאה העליונה, כי קל להדפיס ככה עם הקרוס, אך אם היינו רוצים, אפשר להתחיל מכל נקודה], ומכניסים ל-stack בצירוף העומק שלו, 0.
2. מסמנים ב-grid שהיינו בנקודה initialPoint. [ולפי GRID, 3 הפאות xy,xz,yz יעדכנו שביקרו בתאים המתאימים במבוכים המושרים]
3. מאתחלים באופן ריק את prevMap, שממפה לכל צומת את הצומת שגילה אותו ב-DFS [מייצג את עץ ה-DFS]. נוסף נתחזק משתנים maxDepth, ו-farthest שמייצגים את העומק המקסי של צומת שביקרנו בו ב-DFS [מאתחל למינוס אחת], ואת הצומת הכי עמוקה.
4. כל עוד stack אינו ריק:
 - a. $(vertex, depth) = stack.pop()$.
 - ב. אם $depth > maxDepth$, נעדכן $maxDepth = depth$, ונעדכן את farthest להיות vertex.
 - b. מחפשים ומוסיפים שכנים חדשים של vertex בקובייה התלת-מימדית [Grid]:
 - א. מעבר על 6 השכנים מעל-מתחת-ימין-שמאל-בפנים-בחוץ של vertex, ובדיקה:
 - השכן לא חורג מגבולות המבוך.
 - וגם טרם גילינו את השכן.
 - וגם המעבר אליו חוקי מבחינת המבוכים המושרים - לכל מבוך מושרה לפחות אחד הבאים:
 - טרם גילינו אותו במבוך המושרה (ואז אפשר לשבור אליו קיר במבוך המושרה).
 - או שכדי להגיע לשכן אנו נעים אנכית למבוך המושרה (ואז לא צריך לדאוג לאף שבירת קיר במבוך המושרה).
 - או שכבר הורדנו את הקיר במבוך המושרה (ואפשר לעבור בפתח זה שוב).
 - ב. לכל שכן חדש חוקי מסעיף א' נשבור קירות מתאימים במבוכים המושרים במידת הצורך [ב-grid נוסיף את הקשת כשבורה בפאות המושרות בהתאם] ונסמן שביקרנו במיקום התלת-מימדי שלו.
 - ג. נערבב את סדר השכנים חדשים החוקיים, וכל אחד שכזה נוסיף לתחילת ה-stack בצירוף עומקו $depth+1$, ונוסיף ל-prevMap את המיפוי ממנו ל-vertex.
5. מסלול הפלט [הפיתרון] יהיה הצומת העמוקה ביותר בעץ ה-DFS, שיחושב בסוף ה-DFS עצמו בעזרת טיול חזרה מ-farthest ועד initialPoint עם prevMap, והפיכת המסלול.
6. סה"כ בידינו המבוכים המושרים ומסלול הפיתרון. כעת, כדי שנקבל מבוך שניתן להדפסה [אי-אפשר להדפיס קירות עם עובי 0] מבצעים "ניפוח" של הקירות כך שכל קיר מקבל עובי 1 עבור המבוכים [זה למעשה סקיל של פי 2 פחות אחת של המבוך המקורי], ונחשב (בעזרת כך שלכל פאה מושרית זוכרים אילו קירות הורדנו) את הגריד של 3 הפאות הסמוכות של המבוך [ה-3 הנוספות סימטריות]. כאשר לכל משבצת בכל גריד מסומן אם יש קיר או ריק שם. עושים בדומה עם המסלול, ומנפחים את האינדקסים פי 2, כאשר בין כל זוג איברים עוקבים במסלול כעת גדל המרחק להיות 2, ולכן מוסיפים גם את התא שביניהם למסלול.

סיבוכיות זמן ומקום:

נחשב את הסיבוכיות של האלגוריתם, כאשר הקלט הוא מימדי המבוך X, Y, Z לאורך הצירים. יצירת הגריד נעשית ע"י אלג' ה-DFS שתואר לעיל. שלבי האתחול 1,2,3 בבירור לוקחים זמן ומקום קבוע. השלב העיקרי הוא 4: אנו מבצעים DFS על הקובייה במימדים X, Y, Z , כאשר מסמנים כל תא שהיינו בו, ולכן לא נבקר בו פעמיים. לכן סה"כ כמות התאים בהם נעבור הוא $O(XYZ)$. לכל תא שנחפש לו שכנים חדשים מתבוננים ב-6 השכנים שלו,

כאשר לכל אחד בודקים את התנאים המתוארים לעיל באלג', שכולם ביחד לוקחים זמן ומקום קבוע, ואז שוברים לכל היותר 6 קירות בזמן קבוע, מערבים אותם [לכל היותר 6], ומוסיפים לתחילת ה-stack בזמן ומקום קבוע [נעדכון קבוע בתוחלת של prevMap - הכנסת לכל היותר 6 מיפויים ל-hash, שלאורך האלג' יכול כל צומת לכל היותר פעם אחת => זיכרון בתוחלת $O(XYZ)$]. לכן לאורך האלג' נשלם עלות זמן בתוחלת של $O(XYZ)$. לאורך האלג' אנחנו מתחזקים את ה-stack הזה שיכול להכיל כל תא לכל היותר פעם אחת [כי סימנו תאים לאורך הדרך] ולכן בגודל $O(XYZ)$ לכל היותר. אנו גם מתחזקים לאורך הדרך לכל אחת משלושת הפאות את הקירות ששברנו ואת התאים בהם ביקרנו במבוך המושרה. כיוון שכמות הקירות היא לינארית בגודל המבוך, סה"כ לכל מבוך מושרה נשמור לאורך האלג' זיכרון בגודל המבוך => סה"כ זיכרון $O(XYZ) = O(XY + XZ + YZ)$.

בשלב 5 אנו עוברים מהצומת העמוקה ביותר farthest ועד לנקודת ההתחלה [מסלול באורך לכל היותר $O(XYZ)$], וכל פעם מחפשים ב-prevMap את השכן הקודם, ובסופו הפכים את הרשימה => זמן $O(XYZ)$ בתוחלת, ומקום $O(XYZ)$.

בשלב 6 לבסוף מבצעים עבור המבוכים המושרים ומסלול הפיתרון ניפוח פי 2 ופחות פי 1, ומשלימים קוביות באמצע, שזה כמובן יגדיל (מבחינת זיכרון) כל גריד של פאה פי 2^2 , ואת הקוביה פי 2^3 , ולכן ייקח לכל היותר $O(2^3XYZ + 2^2XY + 2^2XZ + 2^2YZ) = O(XYZ)$ זמן ומקום.

סה"כ עלות הזיכרון והזמן שניהם $O(XYZ)$ בתוחלת.

[נשים לב שזה בעצם יוצא לינארי בתוחלת בנפח התיבה התלת-מימדית, ועבור קלטים הגיוניים להדפסה שבהם $X, Y, Z \leq 100$ אז $XYZ \leq 10^6$ ומקבלים בפועל זמן ריצה סביר (כי הקבוע של ה- $O(XYZ)$ הוא קטן יחסית)].

הערה: הייצוא לפלט של קובץ STL הוא כמובן לינארי בגודל של הפלט [כמות ה-voxels], ולכן לוקח $O(\max(XY, XZ, YZ))$, אך בפועל לוקח המון זמן מפני שאנו שומרים את הפלט לדיסק.

נכונות: נרצה להראות שאחרי הפעלת האלגוריתם מתקבל מבוך אוסקר תקין שניתן בו להגיע מנקודת ההתחלה לנקודת הסיום. ואכן, נסתמך על הנכונות של dfs ביצירת מבוך בדו-מימד לצרכינו.

האלגוריתם מתחיל מנקודת התחלה שרירותית, ובכל צעד אנחנו אפקטיבית צועדים לעומק המבוך כאשר אנחנו שוברים קירות במבוכים המושרים ומסמנים תאים בהם ביקרנו לאורך הדרך.

קיים מסלול במבוך המושרה מההתחלה לסיום כיוון שלכל צומת שכנה חדשה טובה שמגלים שוברים אליה קיר מתאים בשני המבוכים מתוך ה-3 המושרים במידת הצורך, ולכן באינדוקציה על עומק ה-dfs קל לראות שכל תא שהגענו אליו נגיש מההתחלה. כיוון שבחרנו את הסיום להיות תא שביקרנו בו נובע קיום מסלול שכזה.

חוקיות המבוך המתקבל ועצירת האלג' ינבעו מהתבוננות על התנאי לגילוי שכנים:

בהינתן צומת, השכנים החדשים הטובים הם אלו מבין 6 הסמוכים לו כך שטרם ביקרנו בהם במבוך התלת-מימדי, [שזה בדיוק מסכים עם אלג' dfs, וזו הסיבה שהאלג' עוצר] ומוגדרים כך שהם מסכימים עם כל המבוכים המושרים בתהליך הגנרוס. הסכמה עם המבוכים המושרים יכולה לבוא באחת מ-3 דרכים: או שטרם גילינו אותו במבוך המושרה, ואז בהכרח אפשר לשבור אליו קיר במבוך המושרה ולא נהרוס את המבוך המושרה, או שכדי להגיע לשכן אנו נעים אנכית למבוך המושרה, ואז אנחנו לא שוברים קירות במבוך המושרה ובפרט שומרים על חוקיות, או שכבר הורדנו את הקיר במבוך המושרה ועוברים בפתח ולכן לא הורסים את המבוך המושרה. התקניות של המבוך נובעת כיוון שלא פותחים פעמיים קירות שונים לאותו התא, ומכך גם נובע כי לא נפתח גישה למעגל בתוך המבוך הלא מנופח, כלומר אין חלקים מרחפים והמבוך ניתן להדפסה כיחידה אחת. קל לראות שלכל מבוך מושרה אם נתבונן בהיטל של התקדמות האלג' עליו, זה יראה כמעין DFS שלפעמים עוצר במקום ולפעמים חוזר חזרה לטיולים בצמתי העץ אחרי שביקר בהם כבר, ולכן יוצר מבוך תקין [משיקולי הנכונות של ה-DFS בגנרוס מבוך דו-מימדי].

העובדה שכל המבוכים המושרים יוצאים מבוכים שלמים ולא תקועים באמצע נובעת מאינדוקציה פשוטה וההבחנה שאם המבוכים המושרים בתיבה X-Y-Z יפותחו עד הסוף אז גם בתיבה X+1-Y-Z, כיוון שבשכבת ה-X האחרונה נפתח את שורת ה-Z וה-X באינדקס ה-X+1 של הפאות ה-XZ, XY בהתאמה, כיוון שמאינדוקציה פיתחנו לחלוטין עד השכבה ה-X וקעת נוכל בהכרח לשבור קיר לכל אינדקס חדש בשכבה ה-X+1 מהתנאי "או שטרם גילינו אותו במבוך המושרה" ל-2 המבוכים המושרים XY, XZ והתנאי על תנועה אנכית למבוך המושרה YZ, ונמשיך לשבור את שאר הקירות בשורה בשני המבוכים המושרים בעזרת צירוף של אותו הכלל עבור מבוך מושרה אחד, והכלל "או שכבר הורדנו את הקיר במבוך המושרה ועוברים בפתח" עבור המבוך השני המושרה [ותנועה אנכית לשלישי].

יצירת קבצי ה-STL

הנתון בידינו הוא הפלט של אלגוריתם בניית הגריד. הפלט של האלגוריתם הוא מהצורה של המסלול התלת-מימדי שמבצע הקרוס, ושלושת המבוכים המושרים על הפאות XY, XZ, YZ הנתונים בצורה מטריצינית כאשר 1 משמעו שיש בתא קיר, ו-0 משמעו חלל. כעת נראה כיצד ליצור את קבצי ה-STL היוצאים כפלט:

יצירת הקובץ להדפסה - printableMaze: [המבוך עם הקרוס בנקודת ההתחלה, ועם סימוני ההתחלה והסיום]

7. המבוך: כדי ליצור את המבוך עצמו, כל שיש לעשות הוא לעבור על כל אחד משלושת הגרידים, ולבנות בהתאם קוביות יחידה ב-2 הפאות בתלת-מימד איפה שיש קירות במבוכים המושרים המתאימים. עושים זאת לכל פאה XY, XZ, YZ, ולבסוף מוסיפים קוביות יחידה לאורך מסגרת המבוך כדי לחבר את הפאות.

8. המבור עם הקרוס: כעת כדי להוסיף את הקרוס, כל שיש לעשות הוא לקחת את קוביית היחידה, ולבצע SCALE, TRANSLATION במשך 3 פעמים, כדי לבנות את רכיב ה-cross שמקביל לכל ציר, תוך דאגה שמרכז הקרוס יהיה ב-start [שנתון מרצף הפיתרון], ודאגה שיהיה מרווח מספק בינו לבין הקירות האחרים, כדי שבהדפסה זה לא יתמזג.
9. המבור הסופי להדפסה [הכולל סימוני התחלה וסיום]: בשביל המבור הסופי להדפסה, ניקח את המבור עם הקרוס, ולפי הקלט של המסלול מההתחלה לסיום, ניקח את נקודות ההתחלה והסיום בכל פאה, ונוסיף ב-3 פאות סמוכות 4 קוביות גבוהות המקיפות את נקודת ההתחלה, ו-4 נמוכות המקיפות את הסיום.

יצירת הפיתרון של המבור - solutionMaze:

10. outputSequence: זהו קובץ טקסט שמכיל את מסלול הפיתרון, שנוצר תוך כדי האלגוריתם.
11. המבור עם הפיתרון: כדי לבנות את הפיתרון, אנו לוקחים רק את המבור עצמו מקודם (ללא הקרוס וסימוני ההתחלה והסיום), ומוסיפים קוביות לאורך המסלול של הפיתרון [שחזר כפלט מהאלג]. אך המסלול הניתן הוא מסלול בגרסא הלא מנופחת של המבור, ולכן לכל מיקום לאורך הפיתרון נבצע SCALE של פי 2, ונוסיף גם את הקוביות המחוברות בין זוג מיקומים עוקבים, עקב הניפוח.

בחירת סיום המבור

מובן שמטרתנו להכין מבור שתחת אילוצי גדלים נתונים יוביל לפיתרון ארוך ככל האפשר, כיוון שמבור שפיתרוננו קצר- באורך של מספר צעדים בודד, איננו מרשים ואיננו קשה לפיתרון. תחילה כתבתי את האלגוריתם כך שייקח את הפיתרון להיות המסלול ב-DFS מנקודת ההתחלה ועד הצומת הראשונה שלא גילתה שכנים חדשים טובים [עלה ה-DFS הראשון], אך התברר לי כי אורכי מסלולי הפלטים אינם כ"כ גדולים. לכן, בתיקון יחסית פשוט של ה-DFS הצלחתי למצוא, במקום את עומק העלה הראשון, את עומק העלה הכי עמוק, ואותו לקחת כנקודת היעד של המבור, ושחזרתי את המסלול.

סטטיסטיקות: אורכי מסלולי הפלט הממוצעים [1000 ניסויים] ב-2 השיטות לפי גדלי המבור בצירים השונים:

שיטה/גודל	5x5x5	8x8x8	10x10x10	10x10x2	10x2x2
עלה ראשון	18.214	32.175	41.839	19.56	9.56
עלה בעומק מקסימלי	29.985	65.482	91.522	58.394	11.3

מסקנות: בגדלי מבוכים סבירים להדפסה, השיטה של לקחת את העלה העמוק ביותר מובילה לפתרונות באורך ממוצע של סביבות פי 2 יותר ארוך מאשר השיטה של לקחת את היעד כעלה ה-DFS הראשון. במבוכים שטוחים שיטת העלה העמוק תוביל ליחסים אפילו יותר טובים, ובמבוכים צרים וארוכים התפקוד של 2 השיטות נהייה דומה יחסית. בתוכנה עצמה מכניסים כקלט גם את אורך המסלול המינימלי הנדרש [במונחי המבור הלא מנופח], והתוכנה תנסה 1,000 פעמים להגיע לאורך המסלול הנדרש, ואחרת תוותר ותצא. כך אפשר לנסות לכפות על אורך מסלול פיתרון ארוך.

סקירת הקוד

Main זו התכנה העיקרית, ש:

1. תחילה קוראת ל-createGrid שיוצר את המבור: הוא מנסה כמות מסויימת של פעמים ליצור מבור וקבלת מסלול מספיק ארוך מההתחלה לסוף, כאשר ליצירת מבור ומסלול פיתרון באקראי קוראים ל-dfs, שהוא: מבצע את לוגיקת ה-dfs ונעזר ב-discoverNeighbors שכל פעם מגלה שכנים חדשים (בעזרת legalNeighborsAxisDir שמגלה שכנים סמוכים במבור, ו-isGoodNewNeighbor שאומר האם מותר לעבור לשכן החדש לוגית, כלומר האם יש התאמה מבחינת תקינות המבוכים המושרים ואלג' ה-DFS), ולבסוף קוראים ל-getSolutionPath שמחשב את המסלול של הפיתרון בדיעבד אחרי ה-DFS עצמו.
2. קוראת ל-createEnlargedGrids שמבצעת את הניפוח של 3 המבוכים המושרים
3. קוראת לסדרה של פונקציות שיוצרות בעזרת numpy-stl את הפלט ה-stl [gridsToVoxelLocations] מחשב את מיקומי ה-voxels של המבור עצמו, voxelizeLocationsMesh הופך מיקומים אלו ל-mesh, addCrossMesh מוסיף את הקרוס בנקודת ההתחלה, ו-addStartAndEndAndSave מוסיף את נקודת ההתחלה והסיום למבור. לבסוף, get3dvoxelsFromSequence מחשב את מיקומי ה-voxels של מסלול הפיתרון במבור המנופח של הפלט, ומאחדים מסלול זה עם המבור עצמו לפלט נוסף - solutionMaze].

ספריות נחוצות

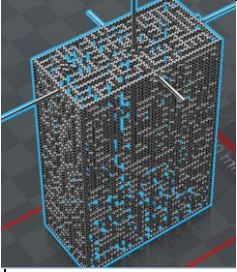
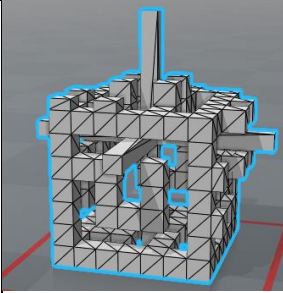
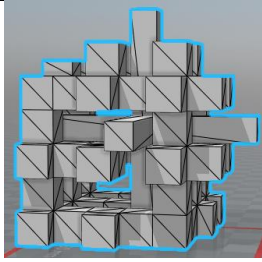
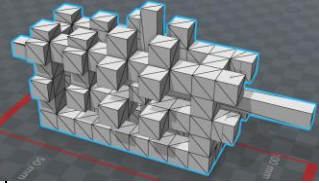
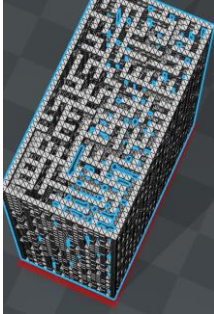
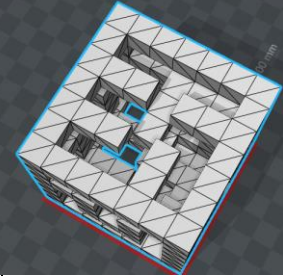
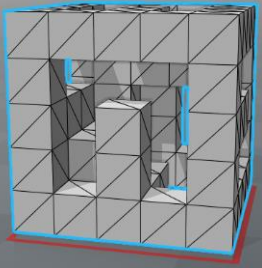
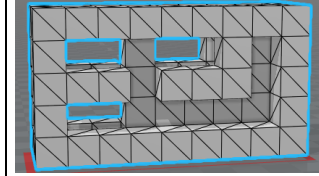
נבנה בפיתרון 2.7.12.

ספריות מובנות בפיתרון: sys, os, random, datetime.

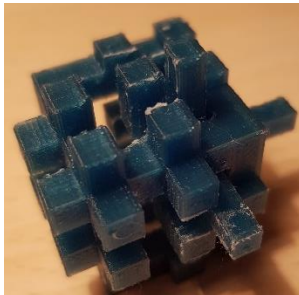
ספריה נוספת: numpy-stl v2.2.0 שב-url: <https://github.com/WoLpH/numpy-stl> [דורש את הספריות הסטנדרטיות

[numpy, python-utils]

פלטמים לדוגמא (ראה התיקיה המצורפת של Outputs Examples בשביל הפלטמים מלאים)

מבוכ 10x20x30	מבוכ 3x3x3	מבוכ 2x2x2	מבוכ 1x2x4	מבוכ
				קובץ של STL המבוכ
				קובץ של STL הפיתרון
[(10, 20, 58) ..., (6, 4, 36)] לפלט מלא ראה - תיקיה - Outputs Examples	[(2, 2, 4), (3, 2, 4), (4, 2, 4), (4, 1, 4), (4, 0, 4), (4, 0, 3), (4, 0, 2), (4, 0, 1), (4, 0, 0), (4, 1, 0), (4, 2, 0), (3, 2, 0), (2, 2, 0), (2, 1, 0), (2, 0, 0), (1, 0, 0), (0, 0, 0), (0, 0, 1), (0, 0, 2)]	[(2, 2, 2), (1, 2, 2), (0, 2, 2), (0, 2, 1), (0, 2, 0), (0, 1, 0), (0, 0, 0)]	[(0, 4, 2), (0, 4, 1), (0, 4, 0), (0, 3, 0), (0, 2, 0), (0, 1, 0), (0, 0, 0), (0, 0, 1), (0, 0, 2)]	מסלול הפיתרון במבוכ המודפס (המונפח) 1

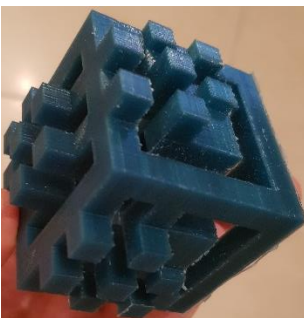
תצלום ההדפסות וסרטון הדגמה והסבר באינטרנט:



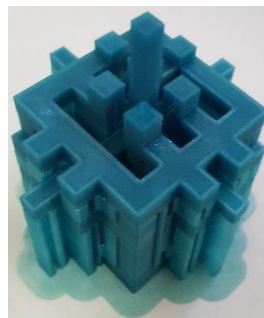
ולאחר הניקוי:



קובייה 2x2x2 (מימדים טרם ניפוח):



ולאחר הניקוי:



קובייה 3x3x3 (מימדים טרם ניפוח):

סרטון ההדגמה: <https://goo.gl/photos/oWNSMRYzefciyiQSA>