**ממן 14 – אלון גולדמן**

**שאלה 1**

**רעיון האלגוריתם**:

בעזרת תכנון דינמי, נבנה טבלה מקבילה לשריג בשאלה, ובה נחזיק את מחירי המסלולים בשריג. בסיום, נוכל על פי הטבלה למצוא את המסלול עם המחיר המזערי.

**הסבר**:

נבנה טבלה בגודל n\*n, שמקבילה לשריג הנתון. כל תא בטבלה ייצג את המחיר של המסלול מ[1, j] כלשהו (עמודת התאים השמאלית ביותר), ועד לנקודה [i,j] בשריג המקורי (לפי הקורדינאטות של התא).

כדי למנוע בלבול, מכיוון שהמחיר בשריג המקורי מסומן בc[i,j], נסמן כל ערך בטבלה החדשה כp[i,j].

דוגמה:

1. אם השריג המקורי הוא שריג בגודל 4\*4, למשל השריג הבא (כל תא בטבלה מייצג קודקוד בשריג):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i=4 | i=3 | i=2 | i=1 |  |
| 8 | 0 | 5 | 3 | J=1 |
| 1 | 7 | 3 | 0 | J=2 |
| 6 | 2 | 6 | 4 | J=3 |
| 1 | 9 | 1 | 6 | J=4 |

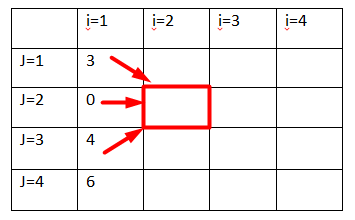
אז תא [1,3] יהיה המסלול המזערי בין תא שמאלי כלשהו לבין תא [1,3] – מכיוון ש[1,3] גם הוא הכי שמאלי, אזי המסלול הכי קצר יהיה הערך בתא עצמו – ולכן הערך שישב בטבלה שנבנה יהיה 4.

1. דוגמה נוספת: בתא [3,3] יהיה המסלול המזערי בין תא שמאלי כלשהו לבין תא זה. במקרה שלנו, יהיה זה המסלול היוצא מתא j=2, שסכומו 5 (0+3+2).

בניית הטבלה:

מכיוון שכל תא בעמודה השמאלית בטבלה הוא המרחק המזערי בינו לבין אותה הנקודה בשריג (ראו דוגמה (1)), בשלב האתחול של הטבלה נכניס עמור כל תא בעמודה השמאלית את הערך המקביל אליו בשריג המקורי (בקודקוד [i,j]).

מכל תא בעמודה השמאלית, המסלול יכול להתקדם ל3 כיוונים – "ימינה", "למעלה וימינה", "למטה וימינה". ז"א, שעבור כל תא בעמודה הבאה, ניתן להגיע מ3 כיוונים: מהתא השמאלי אליו, מהתא השמאלי למעלה, ומהתא השמאלי למטה. המחשה:



לכן בבירור, כדי שבתא המודגש באדום יהיה המסלול עם המחיר המינימלי, המסלול יגיע מהתא השמאלי אליו (מבין שלושת האופציות הנ"ל) שבו המחיר הנמוך ביותר. (בדוגמא המודגשת, תא p[1,2]=0 ולכן המסלול יגיע מתא זה). לכן הערך בתא כלשהו יהיה:

או במילים: המחיר של כל תא בטבלה יהיה המינימום מבין 3 התאים השמאלים הסמוכים + הערך המקורי בתא המקביל בשריג.

בצורה זו נבנה את הטבלה עמודה אחד עמודה, משמאל לימין, עד לסוף הטבלה.

מציאת המסלול המזערי: נמצא את המינימום מבין העמודה הימנית ביותר, וננוסיף אותו למסלול הסופי. מבין 3 התאים השמאליים אליו ( כמו בדוגמה לעיל, התאים שמהם יוצאים החצים האדומים), נבחר את המינימום ונוסיף למסלול. כך נבצע עבור כל תא במסלול, עד שנגיע לעמודה השמאלית ביותר. נחזיר את המסלול שמצאנו.

**האלגוריתם**:

01: find\_minimum\_path(lattice):

02: // init

03: n = length\_of(lattice)

04: table = create a table with size n\*n

05: for j from 1 to n do:

06: table[1,j] = lattice[1,j]

07:

08: // fill table

09: for i from 2 to n do:

10: for j from 1 to n do:

11: p[i,j] = min(p[i-1, j-1], p[i-1, j], p[i-1, j+1]) + c[i,j]

12:

13: // find path

14: cheapest\_path\_starting\_cell = the cell with min p of column i=n

15:

16: j = cheapest\_path\_starting\_cell.j // the j index of the cell

17: for i from n to 1 do:

18: cheapest\_path.add(table[i, j])

19: // get the j of the cheapest path on the left

20: j = min(p[i-1, j-1], p[i-1, j], p[i-1, j+1]).j

21:

22: return cheapest\_path

**נכונות**:

לאלגוריתם 2 חלקים – בחלק הראשון ממלאים את הטבלה במסלולים הנמוכים, ובחלק השני מוצאים את המסלול הזול ביותר. נוכיח את 2 החלקים:

1. נוכיח באידוקציה – בכל תא כלשהו (נסמן בA) בטבלה, יהיה המסלול הזול ביותר מתא כלשהו בעמודה i=1 לתא A.

עבור k=1 זה נכון בוודאי – המחיר הזול ביותר מתא בעמודה 1 לתא בעמודה 1 זה התא עצמו.

נניח שזה נכון עבור k-1. נראה עבור k:

הערך שיהיה בתא כלשהו (נסמן בA) יהיה הערך בשריג המקורי של התא (c[I,j]), בנוסף למינימום מבין שלושת התאים השמאליים יותר אליו (לפי הגדרת השאלה – "שמאלה ולמעלה", "שמאלה" ו"שמאלה ולמטה"). המינימום מבין 3 התאים הנ"ל הוא מסלול באורך k-1, ולכן לפי הנחת האינדוקציה הוא מסלול מינימלי מאותו התא לעמודה i=1 השמאלית. לכן הערך בתא A יהיה גם הוא המסלול המינימלי מעמודה i=1 לתא A.

הוכחנו שכל תא בטבלה הוא המחיר של המסלול המינימלי מהתא לתא כלשהו בעמודה i=1.

1. נוכיח באינדוקציה – כשנתונה טבלה מלאה בערכי מסלולים, האלגוריתם ימצא את המסלול המזערי מתא כלשהו בעמודה i=n לתא כלשהו בעמודה i=1.

עבור k=1 זה בוודאי נכון – הטבלה מייצגת את ערכי המסלולים המזעריים, ובשורה 14 באלגוריתם נמצא את המינימום הזה ונחזיר אותו.

נניח עבור k-1 ונוכיח עבור k:

הערכים בעמודה i=n הם הערכים של המסלולים הסופיים הקצרים ביותר – בשורה 14 נמצא את המסלול המזערי מביניהם. לאחר מכן, נבדוק מהו המינימום מבין 3 התאים השמאליים אליו – לפי הנחת האינדוקציה כל תא מבין ה3 מייצג את המסלול הזול ביותר בינו לבין עמודה i=1. לכן המסלול המזערי מתא בi=n שמצאנו, עובר דרך המינימלי מבין השלושה. המינימלי מבין השלושה הוא מסלול באורך k-1, ולכן לפי הנחת האינדוקציה זהו המסלול המינימלי עד לעמודה i=1.

הוכחנו שההאלגוריתם מוצא את המסלול המינימלי בשריג.

**סיבוכיות**:

1. לולאת האתחול היא בגודל Ɵ(n)
2. יש n^2 תאים, ועבור כל תא מתבצע זמן קבוע של פעולות אלמנטריות (חיבור ומציאת מינימום מבין 3 ערכים). לכן סיבוכיות הזמן היא (n^2)Ɵ
3. מציאת התא המינימלי מבין n תאים עורכת Ɵ(n)
4. מציאת המסלול המינימלי – עבור כל אחד מn התאים במסלול, יש זמן קבוע של פעולות (מציאת מינימום מבין 3), ולכן הזמן הוא Ɵ(n).

קיבלנו בסך הכל Ɵ(n^2) סיבוכיות זמן.

**שאלה 2**

**רעיון**:

נמיין את התיבות בסדר עולה לפי השטח שלהן, ולאחר מכן נחשב בעזרת תכנון דינאמי עבור כל תיבה, מהו הגובה המירבי שניתן לערום כך שהיא תהיה בראש המגדל. נמצא את המגדל הגבוה ביותר.

**הסבר**:

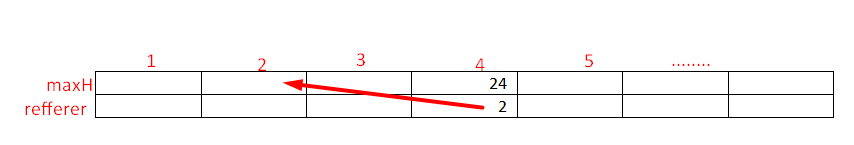
ראשית, נמיין את רשימת התיבות. נמיין את התיבות לפי **שטח** הבסיס של כל תיבה (חישוב של L\*W), בסדר עולה (ז"א התיבה בעלת השטח הגדול ביותר – ראשונה). מיון זה **לא** יוודא שניתן להניח כל תיבה באינדקס גדול על התיבות מאינדקס קטן יותר (שהרי יכולה להיות תיבה עם שטח קטן יותר אך אורך/רוחב גדול יותר מתיבה אחרת). לעומת זאת, המיון יאפשר לנו לדעת שעבור כל תיבה כלשהי, לא יכולה להיות תיבה מ**שמאל אליה** שניתן להניח על גבה.

במילים אחרות – לא יכולה להיות תיבה בעלת שטח **גדול** מתיבה אחרת, שמונחת מעל התיבה בעלת השטח הקטן. אם נתקדם ברשימת התיבות משמאל לימין (מהשטח הגדול לשטח הקטן), לא יכולה להיות תיבה i מעל תיבה j כך שהשטח של i גדול מהשטח של j.

נגדיר את maxH(i) להיות הגובה המקסימלי של מגדל שתיבה i בראשו.

לאחר מכן, נבנה מערך דו מימדי עם 2\*n תאים. בשורה הראשונה (באורך n) יהיו ערכי maxH עבור כל האינדקסים. בשורה השניה יהיה האינדקס של התיבה שמתחת לתיבה i במגדל.

דוגמה (חלקית):



בשורה הראשונה באינדקס 4, זהו הגובה המקסימלי של מגדל שבראשו תיבה מס' 4. במגדל זה, התיבה שמתחת לתיבה 4 היא תיבה שבאינדקס 2.

נאתחל את הטבלה, כך שבכל תא maxH(i) יופיע הגובה של התיבה h(i), ובכל refferer יופיע i.

עבור כל תיבה, משמאל לימין, נבדוק האם הגובה הנוכחי נמוך יותר מהגובה של התיבה + הגובה של תיבה כלשהי משמאל. (ז"א נבדוק האם אפשר לקבל גובה גבוה יותר של מגדל אם נציב את התיבה בראש מגדל אחר, משמאל). במידה וגובה זה גבוה יותר וגם המגדל יציב, נעדכן את הגובה המקסימלי.

בסוף הריצה, נוכל למצוא את גובה המגדל המקסימלי ע"י סריקה של מערך הגבהים. נוכל לשחזר את התיבות המרכיבות את המגדל ע"י שורת הrefferers.

**האלגוריתם**:



(מצ"ב האלגוריתם כאן אם יש הערות, אבל בתמונה יותר נוח לקרוא):

01: tallest\_tower(boxes\_list):

02: sort(boxes\_list) by the base's area // w(i)\*l(i)

03: reverse(boxes\_list) // the biggest area on index 1

04:

05: // init results matrix

06: heights\_array = 2\*n size

07: for i from 1 to n:

08: heights\_array[1,i] = h(boxes\_list(i)) // maxH

09: heights\_array[2,i] = i // refferer

10:

11: // calculate heights

12: for i from 2 to n: // index 1 cannot be on top of every other box

13: for j from 1 to i: // all the boxes to the left of box i

14: if w(i) < w(j) and l(i) < l(i): // the tower will be stable

15: if heights\_array[1,i] < heights\_array[1,j] + h(boxes\_list(i)):

16: // update the maxH

17: heights\_array[1,i] = heights\_array[1,j] + h(boxes\_list(i))

18:

19: // find max tower

20: max = 1

21: for i from 2 to n:

22: if maxH[i] > maxH[max]:

23: max = i

24:

25: while refferer[max] is not the index of max:

26: print(max)

27: max = refferer[max]

28: print(max) // print the last box

**נכונות**:

נוכיח באידוקציה – בכל תא i בטבלה יש את המגדל הגבוה ביותר שיכול להיות כשתיבה i בראש המגדל.

עבור k=1 זה נכון מכיון שתיבה יחידה היא בוודאי המגדל הכי גבוה שבראשו התיבה.

נניח עבור k-1, ונראה עבור k:

עבור כל אחת מהתיבות שמשמאל לתיבה k, נבדוק האם ניתן להניח את תיבה k עליה. במידה וניתן (מבחינת נתוני השאלה) להניח, והגובה שמתקבל גבוהה מהערך הקודם – נעדכן ערך זה. ז"א שהגובה שיהיה בסוף הלולה יהיה הגובה ביותר החוקי כאשר תיבה k בראש. כל אחד מבין הגבהים משמאל הוא מגדל בגודל k-1, ולכן לפי הנחת האינדוקציה הערך הזה מייצג את הגובה של המגדל בהתאם לתא.

ז"א הוספנו את הערך h הנוכחי לגובה המגדל הגבוה ביותר, ובכך קיבלנו שבתא k יהיה המגדל הגבוה ביותר האפשרי.

**סיבוכיות**:

1. חישוב בשטח אורך זמן קבוע, ולכן מיון השטחים עולה Ɵ(nlgn).
2. שינוי הסדר של רשימת התיבות יכול להתבצע בזמן המיון עצמו (ע"י שינוי קל של המיון), אבל ניתן גם לסדר בזמן Ɵ(n) באופן פשוט.
3. לאחר מכן, עבור כל אחת מn התיבות, נבצע בדיקה של כל אחת מהתיבות משמאל. הבדיקה אורכת זמן קבוע, ולכן סך הזמן הוא Ɵ(n^2).
4. מציאת המקסימום אורכת Ɵ(n)
5. עבור כל ערך שנדפיס מתקדמים שמאלה ("למטה" במורד המגדל), ולכן יש לכל היותר n תיבות להדפיס. לכן הזמן הוא Ɵ(n)

קיבלנו בסך הכל Ɵ(n^2) זמן.

**שאלה 3**

סעיף א

פולינום הוא פולינום כלשהו שמקיים אינטרפולאציה של הנקודות .

באופן דומה, הפולינום מקיים אינטרפולציה של הנקודות , הכוללות גם את נקודה . (באופן דומה עבור כל נקודה כלשהי שכלולה בטווח המשותף).

לפי הסבר זה, אם נציב בפולינום שבשאלה את הערך , נקבל:

ונשים לב שהערכים המודגשים באדום הם אותו הערך.

כדי ששוויון זה יהיה נכון, נוכל למשל להציב ו- .

המשוואה אומרת שהפולינום r צריך להתאפס בנקודה . למשל, הפולינום

מקיים תנאי זה.

באופן דומה נציב בפולינום שבשאלה את הערך , ונקבל:

*באופן דומה הערכים האדומים זהים, ולכן נוכל להציב*  ו-

המשוואה אומרת שהפולינום q צריך להתאפס בנקודה . למשל, הפולינום

מקיים תנאי זה.

מ ו- נובע . נחשב:

הביטוי השמאלי לא יכול להתאפס כי נתון שכל הX של הנקודות שונים. לכן הביטוי השני חייב להתאפס. נכתוב את הפולינום s כפולינום פשוט (משוואת קו ישר):

הביטוי הימני לא יכול להיות 0, ולכן קיבלנו שa=0. מכן נובע . נציב זאת במשוואה ונקבל:

קיבלנו:

סעיף ב'

רעיון: נתחזק טבלה בגודל n\*n עם חישובי פולינומים, בתכנון דינאמי, כך שבסוף הריצה נוכל להחזיר את הערך המבוקש.

**הסבר**:

לפי נוסחת הנסיגה מסעיף א', חישוב אינטרפולציה של פולינום מורכב מחישוב של 2 אינטרפולציות קטנות יותר. לכן, נחזיק טבלה בגודל n\*n שבה יהיו הפולינומים של תתי נקודות מסך n הנקודות שקיבלנו. התא p[i,j] בטבלה ייצג את הפולינום .

נשים לב שרק חצי טבלה יהיה מלא, מכיוון שיש חשיבות לסדר הנקודות ברשימה. (אם i<j אז קיים אבל לא קיים .

נאתחל את הטבלה כך שבכל תא p[i,i] יהיה ערך הנקודה עצמו.

לאחר מכן נרוץ על רשימת הנקודות **משמאל לימין**, ועבור כל נקודה נחשב את הפולינום של הנקודה והנקודות משמאל אליה, בריצה **מימין לשמאל**. בצורה זו, בכל שלב של הריצה, הערכים הנדרשים לנו לחישוב נוסחת הנסיגה יהיו כבר מלאים בטבלה.

בסיום הריצה נלך לתא p[1,n] ונחזיר את הפולינום הסופי.

**האלגוריתם**:

01: interpolation(points\_list):

02: // init table

03: n = size\_of(points\_list)

04: create table p with size of n\*n

05: for i from 1 to n:

06: p[i,i] = y(points\_list[i])

07:

08: // fill table

09: for j from 2 to n:

10: for i from j-1 to 1:

11: p[i,j]=

12:

13: return p[1,n]

**נכונות**:

הנכונות מבוססת על נוסחת הנסיגה הנתונה בסעיף א'.

נוכיח באינדוקציה שהאלגוריתם מחזיר את הפולינום הרצוי.

עבור K=1 האלגוריתם מחזיר את הערך בנקודה – שמייצג פולינום יחיד מדרגה 0.

נניח שהאלגוריתם נכון עבור k-1. נראה שהוא נכון עבור k:

לפי נוסחת הנסיגה, כדי לחשב את הפולינום באורך k עלינו לחשב 2 פולינומים באורך k-1:

הפולינום והפולינום . שניהם פולינומים באורך k-1 ולכן לפי הנחת האינדוקציה חישבנו אותם כבר. מכאן שהאלגוריתם נכון.

**סיבוכיות**:

מכיוון שכל החישובים החשבונאים בזמן קבוע (נתון בשאלה), הסיבוכיות היא:

1. אתחול הטבלה הוא ריצה אחת מ1 עד n ולכן בסיבוכיות Ɵ(n)
2. מילוי הטבלה הוא ריצה על n איברים, כאשר עבור כל אחד רצים לכל היותר על n-1 איברים, ולכן הסיבוכיות היא Ɵ(n^2)
3. החזרת הפולינום בזמן קבוע.

קיבלנו בסך הכל Ɵ(n^2)

סעיף ג'

נתון הפולינום , נציב את הנקודות בשאלה ונקבל:

נריץ את האלגוריתם על 5 נקודות אלו:

אחרי האתחול, הטבלה תיראה ככה:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 4 | 3 | 2 | 1i= |  |
|  |  |  |  | 46 | 1j= |
|  |  |  | 2 |  | 2 |
|  |  | 0 |  |  | 3 |
|  | 10 |  |  |  | 4 |
| 98 |  |  |  |  | 5 |

(התאים המסומנים באפור לא יתמלאו – הסברתי זאת בסעיף ב')

נחשב כל תא בטבלה:

בסוף הריצה נחזיר את תא p[1,5] שהוא באמת הפולינום המבוקש

**שאלה 4**

סעיף א'

האלגוריתם מחשב את מחירי המסלולים המינימליים מהצומת הנתונה r לכל שאר הצמתים בגרף. אם יש צמתים שאין מסלול כלל מקודקוד r אליהם, אזי המחיר יהיה אינסוף.

**נוכיח באינדוקציה**:

**טענת האינדקוציה**:

נסמן בk את מספר הקשתות בגרף. עבור k קשתות, האלגוריתם מוצא את המסלול הזול ביותר מקודקוד הנתון לשאר הקודקודים בגרף.

**בסיס האינדוקציה**: עבור גרף עם קשת אחת (k=1), נניח שקשת זו יוצאת מקודקוד r לקודקוד אחר כלשהו, נסמן בu. (שהרי אם הקשת מחברת 2 קודקודים אחרים או **נכנסת** לr (כי הגרף מכוון), אז אין משמעות לקשת, ועדיין לא ניתן להגיע מקודקוד r לאף קודקוד. לכן זהו המקרה היחיד המעניין).

בלולאה הפנימית באלגוריתם נבדוק האם . מכיוון שA[u]=∞ וA[r]=0, נעדכן את הערך A[u] לערך סופי כלשהו. זהו המחיר המינימלי (ואפשרות היחידה) להגיע מקודקוד r לקודקוד u, והטענה נכונה.

כעת נניח עבור גרף עם k-1 קשתות, ונוכיח עבור גרף עם k קשתות.

נבחר קשת כלשהי בגרף, ונסיר אותה. קיבלנו גרף עם k-1 קשתות, ולכן לפי הנחה האינדוקציה כל המרחקים במערך A נכונים. כעת נחזיר את הקשת שהסרנו. כאשר נגיע אליה במהלך הסריקה שבלולאה הפנימית, נבדוק את התנאי .

* אם התנאי לא מתקיים (ז"א הצד השמאלי של אי השווין קטן יותר), אז הקשת לא מאפשרת להגיע לקודקוד v במחיר זול יותר, ולכן לא נעדכן כלום. האלגוריתם יחזיר את אותה התשובה שהוא היה מחזיר ללא קשת זו, ולפי הנחת האינדוקציה אלו המחירים הזולים ביותר
* אם התנאי מתקיים (ז"א הצד הימני קטן מהשמאלי באי השוויון), ניתן להגיע לקודקוד v בצורה זולה יותר, ע"י הגעה לצומת u ואז שימוש בקשת e. לפי הנחת האינדקציה, המחיר לצומת u הוא הזול ביותר (מכיוון שבלי הקשת שהוספנו, קיבלנו גרך עם k-1 קשתות ולכן המחירים הזולים מעודכנים במערך). נעדכן את המחיר במערך A. בסבבים הבאים של הלולאה החיצונית, יעודכנו כל המחירים בהתאם, במידה ושינוי המחיר ישפיע עליהם.

קיבלנו שבמידה והוספת קשת מורידה את מחירי המסלולים – האלגוריתם יעדכן את המערך.

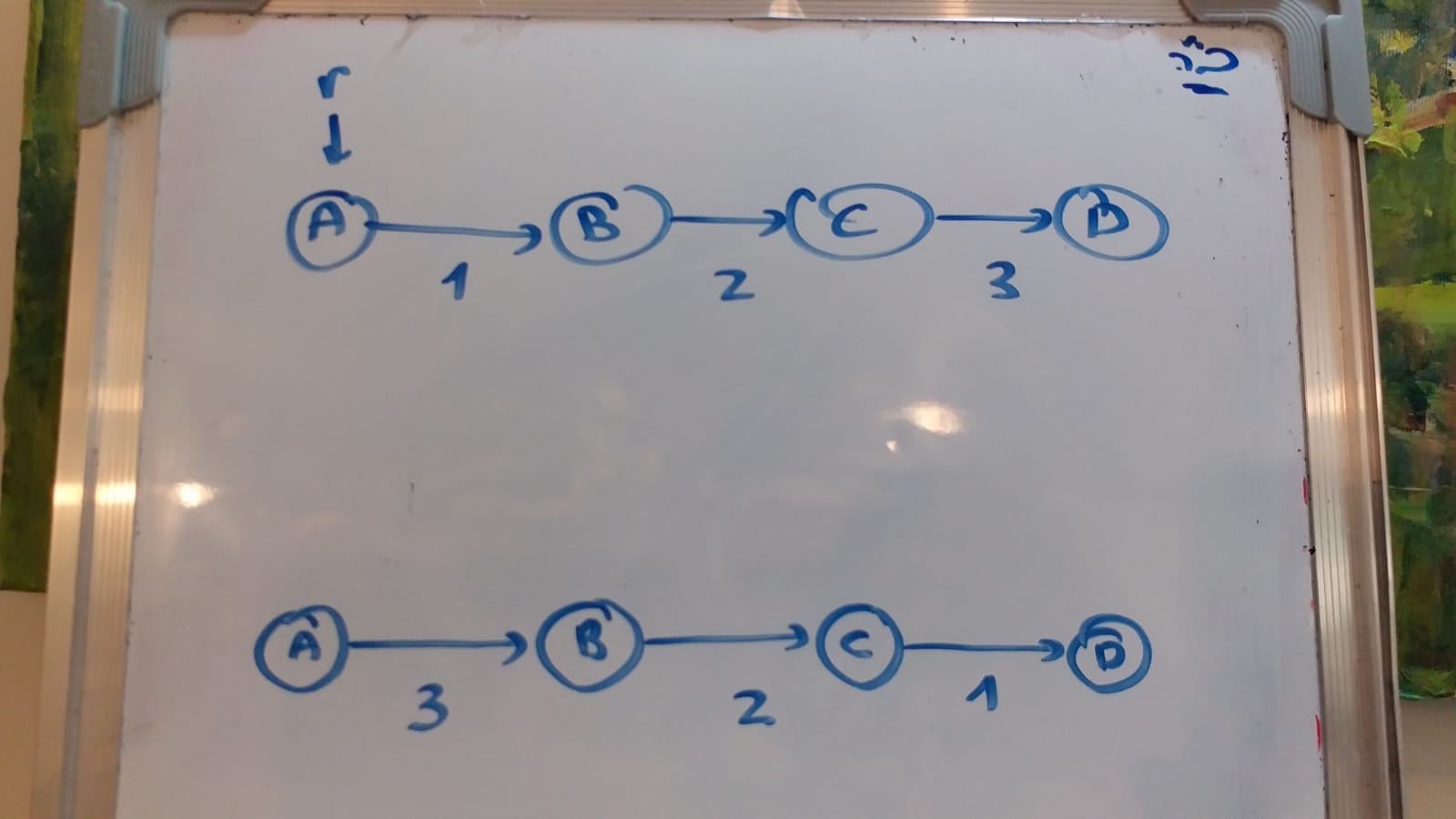
הוכחנו שהאלגוריתם מוצא את מחירי המסלולים הזולים ביותר מקודקוד r לכל הקודקודים האחרים.

סעיף ב'

בכל איטרציה של הלולאה החיצונית, חייב לכל היותר להתעדכן מחיר של תא אחד במערך לפחות (שאחרת האלגוריתם היה עוצר). במקרה הגרוע ביותר, יעודכן בכל איטרציה רק ערך אחד במערך. כדי לחשב את מספר הפעמים שזה יכול לקרות, נתבונן באורכי המסלולים מקודקוד r לכל שאר הקודקודים (**אורכי** המסלולים, ולא **מחירי** המסלולים!).

אם המסלול הארוך ביותר בגרף כלשהו הוא באורך k קשתות, אז מס' האיטרציות המירבי הוא k+1 פעמים (k פעמים לעדכון המערך, ואיטרציה נוספת שבה לא יתעדכן אף ערך והאלגוריתם יעצור). המסלול הארוך ביותר האפשרי בגרף כלשהו עם n צמתים הוא בעל n-1 קשתות (שהרי אם יש עוד קשת נוספת אז נוצר מעגל, ואז ניתן להשמיט קשת אחרת מאורך המסלול). מכאן שערך B המירבי הוא n איטרציות.

משפחה של גרפים שעליהם יתבצעו B(n) איטרציות הם שורה של צמתים המחוברים ביניהם בקשתות, כאשר מחירי הקשתות מסודרות בסדר **הפוך** מסדר הגרף. דוגמה:

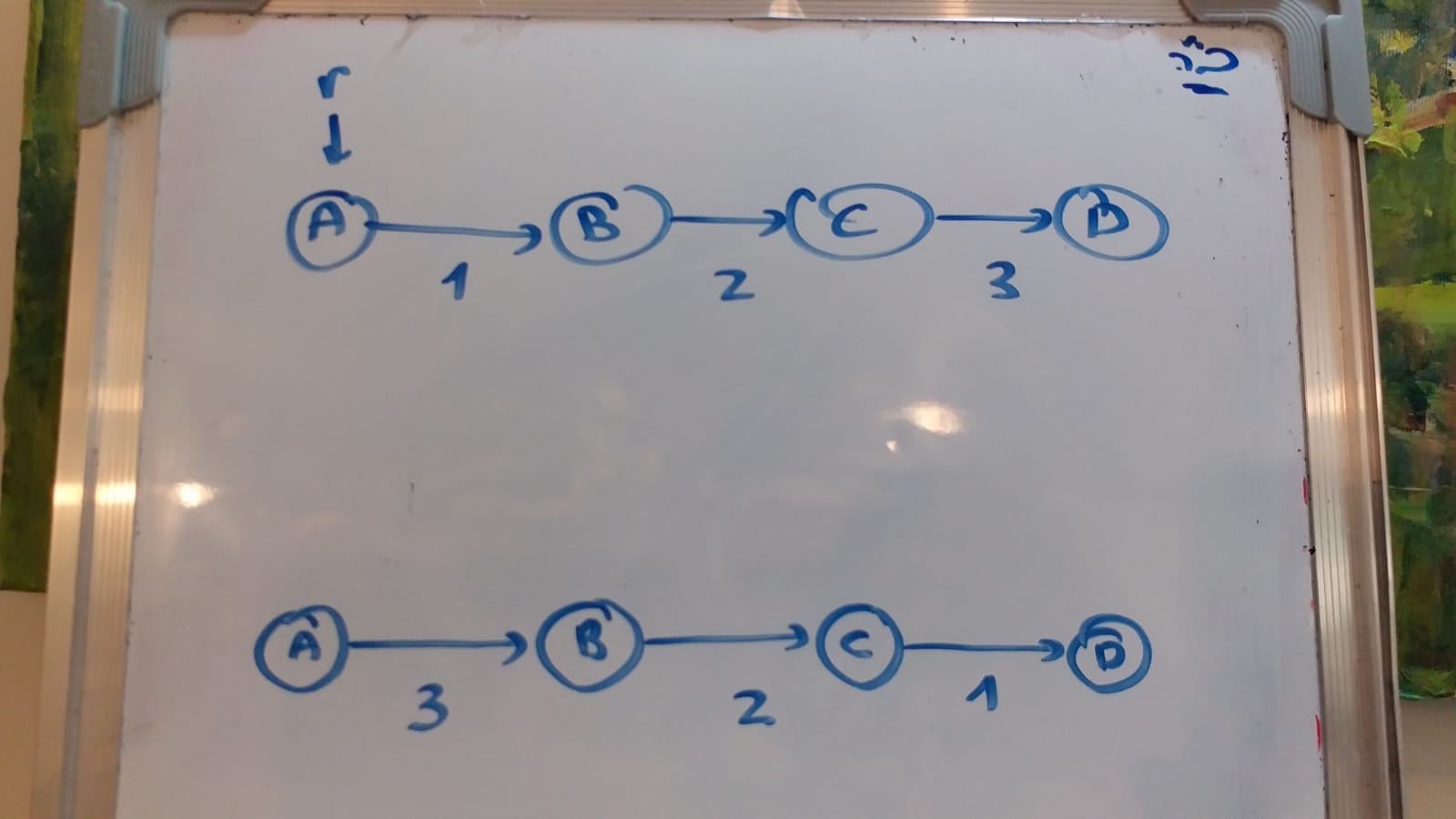


(כאשר הצומת A היא הצומת הנתונה r).

באיטרציה הראשונה יתעדכן הערך הצמוד לr בלבד; לאחר מכן הערך העוקב אליו; לאחר מכן העוקב אליו וכל הלאה, במשך n-1 איטרציות. באיטרציה האחרונה לא יתעדכן אף תא במערך, והאלגוריתם יעצור.

סעיף ג'

משפחה אפשרית של גרפים שבהם יהיו רק 2 איטרציות, היא גרפים דומים לגרפים בסעיף ב', רק שהקשתות מסודרות הפוך – המחיר הכי זול צמוד לr וכל הלאה. דוגמה:



באיטרציה הראשונה יעודכנו כל התאים במערך, ואלו גם המסלולים היחידים האפשריים. בריצה לאחר מכן לא יתעדכן אף ערך, והריצה תיעצר.