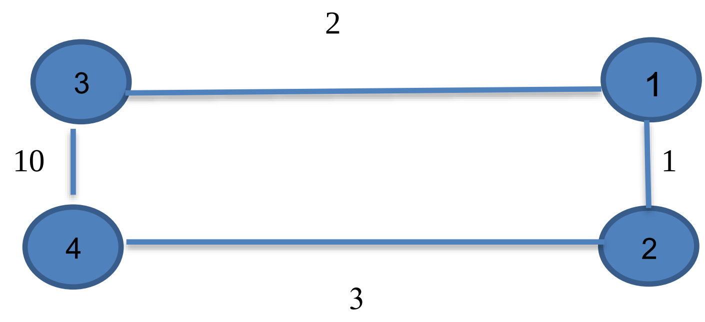
עבודה 2 – תכנון אלגוריתמים

ההצעה אינה עובדת, נפריך ע"י דוגמה נגדית : יהיה גרף G הגרף הבא:



נבחין כי אם החלוקה השרירותית תבחר כקבוצות את קוד' V1={ 1,2} ולקבוצה השנייה את קוד V2={3,4} נקבל כי

משקל הקשתות (1,2) ו (3,4) הינן פתרון חוקי לתת הבעיות ולכן קשת עם משקל מינימלי המחברת ביניהן הינה (1,3)

שמשקלה 2 כלומר, בסה"כ נקבל משקל עפ"מ לפי אלגוריתם זה : 10+1+2=13 אבל ניתן לראות כי עפ"מ הנכון לגרף

זה הינו אוסף הקשתות: (1,2), (2,4) (1,3) אשר משקלן הינו 6 בסתירה למשקל המתקבל מההצעה.

1)ב)ההצעה נכונה, נוכיח את הטענה:

<u>טענה ראשית</u>: בהינתן גרף קשיר G=(V,E) האלגוריתם מחזיר עפ"מ חוקי, T של הגרף G.

<u>ט.ע:</u> עבור כל k, $1 \le k \le |v-1|$ האלגוריתם בחר קב' צלעות T_k שמוכלות בפתרון של עפ"מ T*.

<u>הוכחת טענה ראשית:</u> יהיה גרף G=(V,E) בכל שלב k באלגוריתם, לפי טענת העזר נבחין כי קב' הצלעות T_k מוכלת בפתרון חוקי לבעיית העפ"מ. לכן, גם עבור השלב ה*v-1* (האחרון באלגוריתם) נקבל שהוא מוכל בפתרון חוקי ואופטימלי כלשהו *T*,כלומר אם $T_k \subseteq E_{T^*}$ אזי בהכרח,

$W(T_k) \subseteq W(E_{T^*})$ אנו יודעים כי עבור *v-1* שלבים באלגוריתם, נקבל *v-1* צלעות לכן,

$|T_{v-1}| = v-1$ בנוסף, אנו יודעים כי $W(E_{T^*})$ הינו מינימלי מהגדרתו כפתרון אופטימלי, לכן מכל אלה נקבל *T=T** כלומר, *T* עפ"מ שהאלגוריתם מחזיר הינו עפ"מ חוקי כנדרש. האלגוריתם אינו נתקע מפני שכאשר אנו נמצאים בשלה ה-1 באלגוריתם, נפעיל את האלגוריתם של פרים בצורה רגילה, ולכן , מהיותו גרף קשיר, תמיד תהיה לנו צלע חוצת חתך שתמשיך את האלגוריתם, כך גם לגבי השלב ה-2, כאשר נפעיל את האלגוריתם של קרוסקל.

<u>הוכחת טענת עזר</u>: נוכיח באינדוקציה על השלב הk באלגוריתם, כלומר על בחירת הצלע e_k

<u>בסיס</u>: עבור k=0: מתקבלת הקב' הריקה שמוכלת בכל פתרון חוקי T* לבעיית העפ"מ כנדרש.

<u>ה.א:</u> עבור השלב הk-1 קיים עפ"מ T* עבורו קב' הצלעות $T_{k-1} \subseteq E_{T^*}$

<u>צעד</u>: תהי הצלע הנבחרת בשלב ה-k עבור הצעד הk , e_k . נחלק ל2 מקרים:

1) $k \le \left\lfloor \frac{|v|}{2} \right\rfloor$ כלומר, בשלב ה-k הצלע הנבחרת היא כחלק מהאלגוריתם של פרים. נחלק ל2 מקרים:

א) $e_k \in E_{T^*}$ אזי, נוכל לבחור עבור האלגוריתם את העץ T*=T וסיימנו.

ב) $e_k \notin E_{T^*}$ אזי, נתבונן בגרף $Z=(V, E_{T^*} \cup \{e_k\})$ קיים בZ מעגל, לפי משפט שנלמד בכיתה. נסמנו בC. נבחין כי $e_k \in C$ מפני שאם לא היה שייך למעגל, היה מעגל כבר לפני, בסתירה להיות *T עפ"מ ללא מעגלים. לפי אותו משפט, לכל קשת ששייכת למעגל, אם נסיר אותה נקבל עדיין עץ פורש. נסמן את הצלע $e_k = (u, v)$ כאשר לפי האלגוריתם של פרים אנו יודעים כי אם יש לנו בחתך קב' S של קוד' שנבחרו(בגודל *k-1*) אזי, נקבל כי $u \in S$ ו- $v \in V/S$.נבחין כי מפני שהצלע e_k אינה שייכת לE_{T^*}, אזי קיימת צלע נוספת $e'=(u', v')$ אשר מקיימת את התנאי $u' \in S$ ו- $v' \in V/S$, הינה חלק מהמעגל בהכרח, מפני שמסלול מ*v* ל*u* שאינו כולל את הצלע עצמה, מתחיל ב-*S* ומסתיים ב- *S/V*. לכן לפי אותו משפט, שנלמד, הגרף המתקבל $T'=(V, E_{T^*} \cup \{e_k\} \setminus e')$ הינו גרף קשיר בעל *v-1* צלעות, ולכן הוא עץ פורש לגרף *G*. נבחין כי $e' \notin T_{k-1}$ מפני ש*e'* נמצא באותו חתך בקבוצות קוד' כמו צלע e_k. לכן לפי ה.א נוכל לקבל כי: $T_k = T_{k-1} \cup \{e_k\} \subseteq E_{T'} = (E_{T^*} \cup \{e_k\}) \setminus \{e'\}$ בנוסף, נבחין כי מפני שבשלב ה*k* האלגוריתם יכל לבחור גם ב e_k וגם ב*e'* כי שתיהן מאותו החתך, מהעובדה שהאלגוריתם בחר בe_k נסיק ש$w(e_k) \le w(e')$. לכן בסה"כ בחישוב המשקלים נקבל:

$w(T') = W(T^*) - w(e') + w(e_k) \le w(T^*)$ (לפי מה שראינו חיסור המשקלים יביא לתוצאה שלילית והקטנת התוצאה) לכן, נקבל שגם *T'* הינו עפ"מ עם משקל קל לפחות כמו T^* , ומכיל את T_k.

2) $k > \left\lfloor \frac{|v|}{2} \right\rfloor$, כלומר בשלב ה-*k* הצלע הנבחרת היא כחלק מהאלגוריתם של קרוסקל. נחלק ל2 מקרים:

א)אם הצלע e_k כאשר נרצה להוסיפה , סוגרת מעגל יחד עם T_{k-1} ולכן לא ניקח אותה, כלומר ,

$T_k = T_{k-1} \subseteq E_{T^*}$ ולכן נבחר $T = T^*$ להיות הפתרון שמכיל את T_k.

ב)אחרת, $T_k = T_{k-1} \cup \{e_k\}$. כאן נחלק ל2 מקרים נוספים:

ב1) $e_k \in E_{T^*}$ אזי, נוכל לבחור עבור האלגוריתם את העץ T*=T וסיימנו.

ב2) $e_k \notin E_{T^*}$ אזי, נתבונן בגרף $Z=(V, E_{T^*} \cup \{e_k\})$ קיים בZ מעגל, לפי משפט שנלמד בכיתה. נסמנו בC. נבחין כי $e_k \in C$ מפני שאם לא היה שייך למעגל, היה מעגל כבר לפני, בסתירה להיות *T עפ"מ ללא מעגלים. לפי אותו משפט, לכל קשת ששייכת למעגל, אם נסיר אותה נקבל עדיין עץ פורש, נבחין כי במעגל קיימת צלע $e' \notin T_{k-1} \cup \{e_k\}$, אחרת המעגל היה מתקבל כבר ב T_{k-1} בסתירה לבחירת האלגוריתם. הגרף המתקבל $T'=(V, E_{T^*} \cup \{e_k\} \setminus e')$ הינו גרף קשיר בעל *v-1* צלעות, ולכן הוא עץ פורש לגרף *G*. לכן לפי ה.א נוכל לקבל כי: $T_k = T_{k-1} \cup \{e_k\} \subseteq E_{T'} = (E_{T^*} \cup \{e_k\}) \setminus \{e'\}$ בנוסף, נבחין כי מפני שבשלב ה*k* האלגוריתם יכל לבחור גם ב e_k וגם ב*e'* כי שתיהן ניתנות לבחירה בשלב ה- *k*, מהעובדה שהאלגוריתם בחר בe_k נסיק ש$w(e_k) \le w(e')$. לכן בסה"כ בחישוב המשקלים

$w(T') = W(T^*) - w(e') + w(e_k) \le w(T^*)$ (לפי מה שראינו חיסור המשקלים יביא לתוצאה שלילית והקטנת התוצאה) לכן, נקבל שגם *T'* הינו עפ"מ עם משקל קל לפחות כמו T^* , והינו מכיל את T_k כנדרש.

2)א) נגדיר את תת הבעיה , עבור כל קוד' מ-$v1,...,vn$ נגדיר תת בעיה לכל v_i, $1 \le i \le n$, צריך למצוא מסלול מקוד' s לקוד v_i בעל הפרש מינ' בין צלעות ירוקות וצהובות.

נגדיר את קב' הפתרונות $sol(v_i)$: $I \in sol(v_i)$ כאשר($I=(v_s, \dots, v_i$ כאשר זהו מסלול אפשרי בגרף מקוד' s לקוד' v_i . כאשר $I \subseteq \{v_1, \dots, v_n\}$(ללא חשיבות לסדר, הכוונה היא לכל קוד' הגרף, כאשר פתרון I מכיל תת קב' של קוד' G).

הגדרת OPT: OPT(v_i)= מסלול I בעל הפרש מינ' בין צלעות C(e) ירוקות לצהובות מבין כל המסלולים בין s לv_i, כלומר, $|\{e \in I: c(e) = green\}| - |\{e \in I: c(e) = yellow\}|$ $Min_{I \epsilon sol(v_i)}$ = OPT(v_i) .
נגדיר עבור קוד' כלשהו $v_i = j$
נגדיר עבור $c(e) = green = 1$, $c(e) = yellow = -1$ (צלע ירוקה מוסיף 1,צהובה מורידה 1)
בנוסף, נגדיר עבור $I \in sol(v_i)$ את diff(I) להיות ההפרש בין הצלעות הירוקות לצהובות במסלול I.

ב) נגדיר את sol לתתי קבוצות בצורה הבאה:
Sol(j)=$\{I \cup (k,j) | I \in sol(k), k \in V, (k,j) \in E\}$

ג)נגדיר את נוסחת המבנה לחישוב *opt(j)*:

$$OPT(j) = \begin{cases} -\infty, & in\ path\ P\ exist\ \ most\ yellow\ cycle \\ else & \\ 0, & j = s \\ Min_{(k,j)\epsilon E}\{OPT(k) + c(k,j)\} & \end{cases}$$

ד) <u>אבחנה</u>: נראה כי אם יש לנו מעגל עם יותר צלעות צהובות בגרף, נוכל לעבור עליו אינסוף פעמים ללא קשר למס' הצלעות הירוקות ובכך תמיד להגיע להפרש מינ' $-\infty$. האלגוריתם אכן עובר על מצב זה, אך זהו מקרה מיוחד שראוי לציון. אם אנחנו הקוד' s עצמו, לפי הנתון שאין צלעות נכנסות אל s, בהכרח לא נוכל לבצע מעגל ולכן המסלול המינ' הוא 0, אחרת נחפש לפי המינ' בין כל השכנים, אשר מגדירים פתרון מינימלי עבור j.

$$OPT(j) = Min_{I \epsilon sol(j)} diff(I) = Min_{(k,j)\epsilon E}\left\{Min_{I' \epsilon sol(k)}\{diff\left(I' \cup (k,j)\right)\}\right\} =$$

<u>אבחנה 1</u>(שהשתמשנו בה במעבר מלעיל): כל מסלול מ- s לj יכיל את כל הפתרונות למסלול מ-s לשכניו של j (כל k) והצלע האחרונה במסלול תהיה הצלע (k, j).
<u>אבחנה 2:</u> הצבע המינימלי יהיה -1 (צהוב) ומקס' 1(ירוק) עבור כל צלע, בכל מקרה, גם אם אין לנו צלע צהובה, ניקח בהכרח צלע ירוקה, לכן בכל שלב בנוסחה, נבחר צלע כלשהי.

$$= Min_{(k,j)\epsilon E}\left\{Min_{I' \epsilon sol(k)}\{diff(I') + c(k,j)\}\right\}$$
$$= Min_{(k,j)\epsilon E}\{Min_{I' \epsilon sol(k)}\{diff(I')\} + c(k,j)\} =$$

לפי הגדרה אנו יודעים כי $Min_{I' \epsilon sol(k)}\{diff(I')\} = OPT(k)$ לכן נוכל לקבל:

$$= Min_{(k,j)\epsilon E}\{OPT(k) + c(k,j)\} \blacksquare$$

כאמור הוכחנו את הנוסחה, והאלגוריתם יעצור במידה ולא קיים מעגל צהוב, אחרת כפי שאמרנו, נקבל מסלול אינסופי עם הפרש מינוס אינסוף.

ה) <u>אבחנה</u>: על גרף מכוון חסר מעגלים (DAG) ניתן לעשות מיון טופולוגי.

שלבי האלגוריתם:

1)נאתחל מערך A בגודל |V|, מאותחל בכל תא עם הערך 0.
2)נבצע מיון טופולוגי על הגרף אשר מתחיל מקוד' s. מצב זה אפשרי מהאבחנה, בנוסף אנו שמ-s יש רק צלעות יוצאות ולא נכנסות, לכן בהכרח המצב אפשרי. נסמן את המיון: $s = v_{1,\dots,} v_n$
3)עבור כל קוד' v_j, $2 \le j \le n$ נשמור רשימה של הצלעות הנכנסות אליו.(נקרא לה InListj)
4)עבור j , $1 \le j \le n$, נחשב במערך A שיצרנו: $A[j] = Min_{(v_{i,}v_j)\epsilon InListj}\{A[i] + c(v_i, v_j)\}$
5)נחזיר את המערך, כאשר בכל תא אנחנו נקבל עבור קוד' מסוים את המרחק המינימלי מ-s.

ו)הסבר נכונות האלגוריתם: לפי האבחנה ,המיון הטופולוגי יכול להתקיים, ובפרט יכול להתחיל מ s אשר נתון שצלעות היוצאות ממנו הינן יוצאות בלבד. כאשר נבצע את המיון, מכיוון שביצענו מיון טופולוגי, כאשר נרצה לגשת לקוד' v_j מסוים , כבר יחושבו כל הקודמים אליו בגרף, כלומר המינימום של כל הקוד' שקיימת להם קשת נכנסת אל v_j, ובכך נוכל לבחור את הצבע המינ' (צהוב) במידה וקיים, כך בכל תא , ככל שנשיך, נשמור את המסלול המינימלי מבין כל שכניו כולל אותו, בכך האלגוריתם מתקדם בצורה איטרטיבית. כאשר נתקדם בהמשך המיון, נמצא את המסלול עם ההפרש המינימלי עבור כל קוד' בגרף, כאשר הערך המינימלי נשמר בתא המתאים במערך, כשנרצה להחזיר את המערך, אם נשמור בכל תא גם את ערך(זיהוי בגרף) הקוד' המקורי(ולא רק את הסדר במיון), נקבל תשובה מלאה עבור כל קוד' בגרף את הפרש הצלעות המינימלי.
<u>ניתוח זמן ריצה:</u> 1)אתחול מערך ריק O(|v|) – A
2)עבור מיון טופולוגי בגרף, בעיה שניתנת לפתירה ע"י סריקת DFS בזמן (|E|+|V|)O.
3)רשימה עבור כל קוד', מעבר על כל הקוד' וכל הצלעות פעם אחת בלבד:(|E|+|V|)O .
4)מעבר על כל קוד' במיון ובדיקת המינ' , כאשר בכל בדיקה יתבצעו פעולות בגודל רשימת הצלעות הנכנסות, לכן בסה"כ זמן הריצה יהיה (|E|+|V|)O.
5)החזרת המערך, כאשר ניגש לכל תא ונשלוף את ערך הקוד' בגרף(ולא דרך המיון) ב- (|v|)O לכן בסה"כ נקבל זמן ריצה : (|E|+|V|)O.

3)א)תת בעיה אופיינית: נגדיר את תת הבעיה במט' M (nXm) עבור כל תא $1 \le i \le n$,$1 \le j \le m$, חישוב מסלול בעל מחיר מינימלי מהנק' (1,1) לנק' (i, j).
קב' פתרונות *sol*: (i, j)*sol*-קב' כל המסלולים האפשריים עבור הגעה אל הנק' (i, j). פורמלית, $P \in sol(i, j)$ כאשר P=$(< 1,1 >, \dots, < i, j >)$ (עבור הבעיה שלנו, $(i, j) = (n, m)$).
נגדיר $[j][i]$ *val*$(i, j) = value\ of\ moving\ from\ cell[i][j]$
(i, j)*OPT*- המסלול מ(1,1) אל (i, j) בעל העלות המינימלית מבין כל המסלולים ב(i, j)*sol*. כלומר: $OPT(i, j) = Min_{P\epsilon sol(i,j)} \{\sum_{(k,l)\epsilon p} val(k, l)\}$.

ב) נחלק את *sol* לתתי קב' בצורה הבאה:
$$sol(i, j) = \{(P \cup (i, j)|P\epsilon(sol(i - 1, j) \cup sol(i, j - 1))\}$$

ג)נמצא נוסחה ל(i, j)*OPT* נגדיר 2 סוגי OPT ומבינם נבחר את האופטימלי בכל שלב: OPT_1- יתייחס לחישוב אופטימלי עבור הכניסה משורה(משמאל), OPT_2 יתייחס עבור כניסה מעמודה(מלמעלה)
$$OPT(i, j) = \{\ 0 \quad , \qquad i, j \le 1$$
$$\{OPT(i, j - 1) + val(i, j - 1) \quad , i = 1, 1 < j \le m$$
$$\{OPT(i - 1, j) + val(i - 1, j) \quad , j = 1, 1 < i \le n$$
$$else$$
$$\{OPT(i, j) = \ Min\{OPT_1(i, j), OPT_2(i, j)\}$$

$$OPT_1(i, j) = \{OPT(i, 1) + 2 * val(i, 1) \qquad , j = 2, 2 \le i \le n$$
$$\{\ Min\ \{\ (OPT_1(i, j - 1) + val(i\ , j - 1)), OPT_2\ (i, j - 1) + 2 * val(i, j - 1)\ , \qquad else$$

$$OPT_2(i,j) = \{OPT(1,j) + 2val(1,j) \qquad i = 2\ ,2 \le j \le m$$

$$\{Min\{(OPT_2(i-1,j) + val(i-1,j)), OPT_1(i-1,j) + 2 * val(i-1,j), \quad else$$

ד)<u>הוכחת נכונות:</u> למעשה עבור כל משבצת במטריצה, נבחר במקרה האופטימלי שיכול להיות עבורו, כלומר עבור כל תא במטריצה נחשב את המקרים האופטימליים שהיו לפני כן בתוספת המקרה הנוכחי ונבחר את התוצאה המינימלית. במקרה הכללי ללא תנאי קצה\בסיס בו נרצה להגיע לתא פנימי מסוים יהיו לנו 4 אופציות (אנכי ללא שינוי כיוון, אנכי עם שינוי כיוון, אופקי ללא שינוי כיוון, אופקי עם שינוי כיוון), אנו נבחר בשילוב ערך האופטימלי הקודם שממנו הגענו את האופציה המינימלית ביותר מבין ה-4 ב *OPT(I,J)*. כמו כן נשים לב למספר מקרי קצה:

1) מקרה בסיס: *i=1=j* לכן נחשב מסלול מ*(1,1)* לעצמו לכן ערכו יהיה 0.

2) כאשר $j = 1{,}1 < i \le n$ או $i = 1{,}1 < j \le m$, כלומר אנו במצב בו יש דרך אחת בלבד להתקדם. והיא בצורה אופקית או אנכית בלבד, לכן בכל צעד נוסיף את הערך האופטימלי הקודם + ערך התא ממנו התקדמנו.

<u>אבחנה:</u> מקרה הקצה הבא מתאר מצב דומה לכן נתייחס למקרה בה"כ עבור OPT_1:

$j = 2\ ,2 \le i \le n$ כל מקרה בו נכנס לתא *(i,2)* מתא *(i,1)* מפני שאנו בOPT_1 אז כל כניסה תהיה מהשורה הנוכחית, במקרה נכנס לתא בוודאות לאחר פנייה ולכן זוהי האופציה היחידה.

המקרה המקביל עבור OPT_2 יתאר את אותה תופעה עבור כניסה מאותה עמודה.

- כל מקרה אחר יהיה מקרה רגיל כמו שתיארנו בהתחלה.

ה. <u>תיאור האלגוריתם</u>:

1) נאתחל מטריצה *optM* בגודל *(nxm)* כאשר כל תא מכיל זוג סדור *(opt1,opt2)* .

2) נאתחל את *optM[1,1].opt1=0* ו *optM[1,1].opt2=0* . כמו כן לכל $i,j \neq 1$ נאתחל *optM[i,j].opt1=∞* וכן *optM[i,j].opt2=∞* .

3) נרוץ בלולאה על השורה במקום ה *i=1* לכל $1 < j \le m$ ונעדכן ערך כל תא בצורה הבאה : *optM[1,j].opt1=opt[1,j-1].opt1+val(1,j-1)* .

4) נרוץ בלולאה על העמודה במקום ה *j=1* לכל $1 < i \le m$ ונעדכן ערך כל תא בצורה הבאה : *optM[1,j].opt2=opt[i-1,1].opt2+val(i-1,1)* .

5) נרוץ בלולאה מ *i=2* עד *n*

נרוץ בלולאה מ*j=2* עד *m*

$$optM[i,j].opt1 = Min\ \{\ (optM[i,j-1].opt1 + val(i\ ,j-1))$$

$$,optM[i,j-1].opt2 + 2 * val(i,j-1)\}$$

$$optM[i,j].opt2 = Min\ \{\ (optM[i-1,j].opt2 + val(i-1\ ,j))$$

$$,optM[i-1,j].opt1 + 2 * val(i-1,j)\}$$

6) עבור התא *optM[n,m]* נחזיר את הערך המינמלי מבין *optM[n,m].opt1* או *optM[n,m].opt2* .

ו) האלגוריתם מאתחל מטריצה עם זוג סדור בכל תא אשר מייצג את ערכי הכניסה האופטימליים משמאל ומלעיל לתא הנבדק. לכן בכל צומת האלגוריתם מחשב את הערך המינימלי ביותר מבין האופציות להגעה אל הנקודה. ראשית נאתחל את התא במקום ה *(1,1)* עם ערכים אופטימליים השווים ל0 , מפני שלפני תא זה לא ביקרנו באף תא קודם. כמו כן נאתחל את שאר ערכי המינימום במטריצה באינסוף, כאשר כל מסלול אפשרי יעדכן אותו בהתאם בהמשך הריצה. כפי שכבר ציינו

עלינו לאתחל שורה/עמודה בערכים מצטברים לפי השורה/עמודה שכן זוהי הדרך היחידה להגיע לנקודה. עבור כל השאר נחשב את הערך הקודם בנוסף לערך הכניסה הקודם ונבחר את הערך המינימלי עבור כל כניסה. (באלגוריתם זה אין צורך לעבור על השורה ה-2 והעמודה ה-2 כפי שביצענו בנוסחת המבנה מפני שבאלגוריתם זה אתחלנו את השדות באינסוף ולכן אם אין מסלול הוא לא יבחר כי הוא ישאר אינסוף והוא ישקול רק מסלולים קיימים על פי האלגוריתם).

<u>ניתוח זמן ריצה:</u>

1) אתחול מטריצה בגודל *(nxm)* , אתחול התא *(1,1)* עם הערכים המתאימים ב*opt1,opt2* של התא, ואתחול כל ערכי *opt1,opt2* בשאר המטריצה יהיה כגודל המטריצה *O(n*m)*.
2) עבור אתחול השורה והעמודה הראשונה זמן הריצה יהיה *O (n)+ O (m)* כאורך השורות והעמודות.
3) ריצה על כל שאר ערכי המטריצה בלולאה בתוך לולאה וחישוב כל ערכי המינימום המתאימים *O (n*m)* .
4) החזרת הערך המינימלי בתא האחרון –*O(1)*

לכן זמן הריצה הכולל יהיה *O (n*m)*.

4)<u>א)תת בעיה אופיינית</u>: לכל $1 \leq i \leq n$ בעל מצבור ארגזים , נגדיר את תת הבעיה לצמצום של *i* מצבורי ארגזים (*i* קומות)ל *j* קומות שניתנות לאחסון כאשר $1 \leq l \leq m$. קב' הפתרונות *sol(i,j)* תהיה קב' כל הדרכים האפשריות לבצע העברה זו (ולאחסן את *i* המצבורים ב-*j* קומות)

I נגדיר *OPT(i,j)* להיות הפתרון בעל משקל מינימלי בין כל משקלי פתרונות ב*sol(i, j)* פורמלית:

נגדיר עבור פתרון כלשהו I את "עלות" הפתרון להיות $W(\ I)$

$$OPT(i,j) = Min_{I\epsilon sol(i,j)} \sum_{k=1}^{j} \sum_{s\in F_k} c_s \cdot (s - l_k) = Min_{I\epsilon sol(i,j)} W(I)$$

ב) תחילה, נבחין כי עבור פתרון אפשרי, תהי קומה l_j, הקומה ה-*j* בפתרון, כלומר הקומה הגבוהה ביותר בה ניתן לאחסן. לכן, בכל פתרון סביר, אם מס' הקומות הכולל l_j, *i>* כל הקומות בין *i* לl_j ירדו אל הקומה l_j.

$$sol(i,j) = \{(S_1 \cup (l_j = i) | S_1 \in (sol(i-1, j-1)) \cup \left(S_2 \cup (c_i \in F_{l_j}\right)$$

$$|S_2 \in (sol(i-1,j), (l_{j-1} < l_j < i)$$

כלומר, הפתרונות האפשריים הן עבור פתרונות בהם הקומה ה-*i* היא קומת אחסון בעצמה+ הפתרונות בהן היא לא (כלומר כל האפשרויות לקומות אחרונות l_j עד לקומה ה-*i*)

ג)

$$OPT(i,j) = \{0 \qquad , \ j \geq i$$

$$\{\textstyle\sum_{x=1}^{i-1} x \cdot c_{x+1}, \qquad j=1 \ ,else$$

$$\{Min\{OPT(i-1,j-1), OPT(i-1,j) + c_i \cdot \left(i - l_{last_{(i-1,j)}}\right), OPT(i-2,j-1) + c_i$$

($l_{last_{(i-1,j)}}$ -הקומה האחרונה בפתרון). הסבר קצר לפירוק הנוסחה: מבלי להוסיף את *OPT* עבור *i-2* היינו מקבלים מצב בו יכול להיות שתת הפתרון לא מכיל את *i*, אבל פוסל אופציה בו העדיפות

היא בקומה אחת מתחתיו, כך שלא מתחשבת במשקל של הקומה *i* והכפלה ב"מחיר" אם נרד יותר ,לכן, נרצה בכל שלב לבחון גם את הפתרון האופטימלי עם הקומה מתחתיו (*i-1*) היא הגבוהה ביותר.

ד)הוכחת נוסחת המבנה: עבור המקרה בו $j \geq i$, כלומר יש הקצאת קומות אפשריות לפחות כמו מס' הקומות בפועל, לא נזיז דבר. עבור המקרה בו יש לנו אך ורק הקצאה של קומה 1 שבה נוכל לאחסן, ע"מ שבאמת נוכל לאחסן הקומה חייבת להיות הקומה ה-1 (אחרת אין פתרון, לא נוכל להוריד את כל הארגזים), ולכן הפתרון האופטימלי יהיה הורדת כל הארגזים אל הקומה ה-1 לפי המחיר הרלוונטי.
במקרה הכללי:

<u>אבחנה 1:</u> עבור כל פתרון חוקי , או שהקומה ה-*i* (בתור קומת אכסון)מוכל בו או שלא.

<u>אבחנה 2</u>: קב' הפתרונות ב*sol(i j)* שלא כוללים את הקומה ה-*i* תהיה קב' *sol(i-1,j)*.

<u>הסבר</u>: אם הקומה ה-*i* אינה נכללת בפתרון, אזי ישנן *j* קומות אחרות שבהן ניתן לאחסן(צמצום של שאר הקומות ל*j* אפשרויות)

<u>אבחנה 3</u>: קב' הפתרונות ב- *sol (i,j)* שכוללים את הקומה ה-*i* בפתרון יהיו מהצורה:

$$I' \cup \{i\} | I' \in sol(i-1, j-1).$$

<u>הסבר</u>: אם הקומה ה-*i* קיימת בפתרון הכולל, כלומר הארגזים יישארו במקום, אזי, נשאר לנו למצוא פתרון עבור שאר *i-1* אוספי הארגזים , שלהן יש *j-1* אפשרויות לקיבוץ.

הגדרת sol

$$OPT(i,j) = Min_{I \epsilon sol(i,j)} W(I) =$$

אבחנה 1

$$= Min \left\{ Min_{\substack{I \epsilon sol(i-1,j-1) \\ i \epsilon I}} W(I), Min_{\substack{I' \epsilon sol(i-1,j) \\ i \notin I}} \left(W(I) + c_i * \left(i - l_{last_{(i-1,j)}} \right) \right) \right\}$$

אבחנה 2,3

$$= \ Min \left\{ Min_{\substack{I \epsilon sol(i-1,j-1) \\ i \epsilon I}} W(I), Min_{\substack{I \epsilon sol(i-1,j) \\ i \notin I \\ i-1 \epsilon I}} (W(I) + c_i), \right.$$

$$\left. Min_{\substack{I' \epsilon sol(i-1,j) \\ i \notin I \\ i-1 \notin I}} \left(W(I) + c_i * \left(i - l_{last_{(i-1,j)}} \right) \right) \right\} =$$

$$= Min\{ Min_{\substack{I \epsilon sol(i-1,j-1) \\ i \epsilon I}} W(I), Min_{I \epsilon sol(i-2,j-1)} W(I) + c_i,$$

$$Min_{\substack{I \epsilon sol(i-1,j-1) \\ i \notin I \\ i-1 \notin I}} W(I) + c_i * \left(i - l_{last_{(i-1,j)}} \right) \} =$$

הגדרת OPT

$$= Min\{OPT(i-1,j-1), OPT(i-1,j) + c_i \cdot \left(i - l_{last_{(i-1,j)}} \right), OPT(i-2,j-1) + c_i \blacksquare$$

ה)אלגוריתם איטרטיבי:

1)נאתחל מטריצה בגודל *nXm* ,(נתייחס לאינדקסים מ1 עד *n,m*) ומשתנה מינימום עבור כל תא עם ערכי *Lmax*-עבור קומה מקס' ו- *Vmin* עבור ערך מקסימלי מאותחל עם ערך אינסוף

2)נרוץ בלולאה כפולה על כל (*i,j*) אפשריים כאשר הלולאה החיצונית *j*- $1 \le j \le m$ והלולאה הפנימית *i*- $1 \le i \le n$.

3)אם $j \ge i$ אז *M[i,j].Vmin=0 M[I,j].Lmax=1*

4)אחרת, אם *J=1* נרוץ בלולאה מ-1 עד *i-1* ונכניס ל-*M[i,j].Vmin* את הערך: $\sum_{x=1}^{i-1} x \cdot c_{x+1}$ ונעדכן *M[I,j].Lmax=1*

5)אחרת, (*j>1,i>j*) , נכניס לתא *M[I,j].Vmin* את הערך הבא:

Min{M[i-1,j-1].Vmin, M[i-2,j-1].Vmin+c_i*, M[i-1,j].Vmin+*$c_i * (i - M[i-1,j].Lmax)$*},*

6)כעת עדיין בתוך הלולאה, נבדוק איזה מן הערכים נכנסו מבין ה-3 אל התא *M[I,j].Vmin*:

אם הכנסנו *M[i-1,j-1].Vmin,* אזי, נאתחל את הערך *M[i,j].Lmax,* בערך *i*(קומה מקס' לאחסון)

אם הכנסנו *M[i-2,j-1].Vmin+*c_i*,* אזי, נאתחל את הערך *M[i,j].Lmax,* בערך *i-1*

אם הכנסנו *M[i-1,j].Vmin+*$c_i * (i - M[i-1,j].Lmax)$*,* אזי, נאתחל את הערך *M[i,j].Lmax,* בערך *M[i-1,j].Lmax*.

7) כעת נחזיר את המטריצה *M* כאשר הערך האופטימלי שנרצה יהיה ב*M[n,m].Vmin*.

ו) אלגוריתם לשחזור:

1)נאתחל מערך *A* בגודל *m*(כגודל הפתרון האופטימלי של קומות לאחסון)ונשתמש במטריצה *M*.

2)נתחיל לולאת *while* שלא עוצרת כל עוד $i \ne 1, j \ne 1$,(עבור *M[i,j]* נתחיל ב*i=n, j=m*)

*3)*נבדוק האם: *M[i,j].Vmin= M[i-1,j-1].Vmin* אם כן, נוסיף לA את את קומה *i* , נעבוד כעת על ערכי מטריצה אשר *M[i,j]= M[i-1,j-1].* (כלומר, *i=i-1, j=j-1*)

4)אחרת,*3* נבדוק האם: *M[i,j].Vmin= M[i-2,j-1].Vmin+*c_i אם כן, נוסיף לA את את קומה *i-1* , נעבוד כעת על ערכי מטריצה אשר *M[i,j]= M[i-2,j-1].*

5)אחרת, לא נוסיף כלום למערך, ונתקדם לערכי מטריצה: *M[i,j]= M[i-1,j].*(מקרה של תת פתרון ללא *i* / *i-1* (לא בוודאות) קומות בפתרון, נחפש ערכים בתת הבעיה.

6)לאחר שנצא מהלולאה, נוסיף את 1 לA מפני שהוא בהכרח בפתרון, ולא נגיע אליו בלולאה, ונסיים.

ניתוח זמן ריצה: עבור אתחול המערך – *O(1)*

עבור הלולאה: נבחין כי אנו מתקדמים בריצה בעיקר על *i*, ונפסיק כאשר *j=i* או שהגענו לסוף הקומות, כלומר, נרוץ בלולאה לכל היותר כגובה הבניין, שהוא *n*. כאשר בכל לולאה, אנו עורכים פעולות השוואה ועדכון ב*O(1)*. לכן, בסה"כ נקבל זמן ריצה *O(n)*.