# מערכות מבוזרות - הקדמה

## הגדרה

מערכת מבוזרת היא אוסף של מכשירי מחשוב נפרדים היכולים לתקשר זה עם זה. לכל מעבד במערכת מבוזרת בדרך כלל יש את האג'נדה העצמאית שלו, אך מסיבות שונות כגון: צורך בשיתוף משאבים, העלאת זמינות וסובלנות לקריסות, על המעבדים לתאם את פעולותיהם. כיום, כמעט כל חישוב שנעשה הוא במערכות מבוזרות, הסיבה לכך היא שמערכות מבוזרות מאפשרות לנו:

* שיתוף נתונים.
* לטפל בכמויות מידע גדולות יותר, שכן אין צורך שכל יחידת מחשוב תחזיק את כל המידע אלא ניתן לפזר את המידע ביניהם או שכולם יפנו למקומות מוגדרים ששם מאוחסן המידע.
* חישוב מקבילי על פני מכונות רבות.
* לבנות תשתיות תקשורת ומערכות המשתרעות על פני מרחקים גדולים.
* לבנות מערכות שעמידות לתקלות, כך שאם יחידת מחשוב אחת או יותר יקרסו המערכת כולה עדיין תמשיך לתפקד.

הקשיים העיקריים במערכות מבוזרות הם: כיצד לחלק את העבודה בין יחידות המחשוב? כיצד לשתף חישוב, נתונים, ותשתיות תקשורת? כיצד מתמודדים עם הודעות שנאבדו או עיכובים בהודעות? וכיצד מתמודדים עם יחידות מחשוב שקרסו, נפרצו, או שעובדות באיטיות?

## מודלים תיאורטיים של מערכות מבוזרות

בקורס זה נתמקד בשני סוגי מודלים: העברת הודעות וזיכרון משותף. כל אחד מהם יכול להיות מסונכרן או לא מסונכרן. שני המודלים שקולים, כלומר כל מה שמודל אחד יכול לעשות גם המודל השני יכול, ולהיפך, אמנם ביעילויות שונות.

**העברת הודעות**: זהו מודל שבו כל יחידות המחשוב, הנקראות במודל זה צמתים, מתקשרות זו עם זו באמצעות הודעות. ניתן לייצג מערכת כזו בגרף שבו כל יחידת מחשוב היא צומת, וקיימת צלע בין כל שני צמתים שיכולים לתקשר ביניהם. מודל זה משמש למערכות גדולות או רשתות תקשורת.

**זיכרון משותף**: זהו מודל שבו כל יחידות המחשוב מתקשרות על ידי כתיבה/קריאה ממקום בזיכרון המשותף לכל יחידות המחשוב. מודל זה משמש בעיקר במקרים של מחשב יחיד שיש בו כמה מעבדים, או מעבד שרצים בו מספר תהליכים.

### מערכת מסונכרנת

זוהי מערכת שמחלקת את הזמן ליחידות זמן קבועות הנקראות rounds וממוספרות . כל round הוא בין הזמן ל-. כל התהליכים במערכת יודעים בדיוק מתי מתחיל ונגמר כל סיבוב באמצעות שדר שמקבלים כל התהליכים באותו זמן. אנו מניחים שכל פעולה במערכת כזו לוקחת בדיוק סיבוב אחד, אין עיכובים.

**העברת הודעות מסונכרנת**: כל תהליך שולח הודעות בזמן . אנו מניחים שכל ההודעות שמתקבלות בתהליכי היעד נקראות ומעובדות עד סוף זמן , אמנם יכול להיות מצב שהודעות לא מגיעות.

**שיתוף זיכרון מסונכרן**: בכל round כל תהליך יכול לגשת לתא זיכרון אחד המשותף לכל התהליכים, ולכתוב או לקרוא אליו.

סיבוכיות הזמן לאלגוריתם במערכת מסונכרנת הוא מספר הסיבובים הדרוש כדי לסיים את האלגוריתם.

### מערכת לא מסונכרנת

זוהי מערכת שבה הזמן שלוקחת פעולה אינו ידוע אך כן ידוע שהזמן סופי. במערכת כזו יכולים להיות עיכובים. אנו צריכים לשקול את המקרה הגרוע ביותר, לכן אנו מניחים שהעיכובים נקבעים על ידי יריב שמחפש להכשיל את המערכת.

**העברת הודעות לא מסונכרנת**: כל תהליך שולח הודעות וידוע שהן אכן מתקבלות ביעדן, אך משך העיכוב הוא שרירותי בין הודעה להודעה ונקבע ע"י יריב. אמנם היריב אינו יכול לקבוע ש-B יקבל הודעה מ-A לפני ש-A שלח אותה, הסדר נשמר.

**שיתוף זיכרון לא מסונכרן**: כל התהליכים לבסוף מסיימים את פעולתם, אך משך הזמן הדרוש שרירותי ונקבע ע"י יריב.

כיצד נגדיר סיבוכיות זמן לאלגוריתם במערכת לא מסונכרנת? לא נרצה שהגדרת הסיבוכיות תהיה תלויה בזמן שבוחר היריב שכן זמן זה לא תלוי באלגוריתם. לכן את כל טווח הזמנים שהיריב יבחר לעכב, ננרמל לערכים בין 0 ל-1 ביחידות זמן מתאימות. בדרך כלל נתעניין במקרה הגרוע ביותר שבו זמן העיכוב מקסימלי, לכן בדרך כלל נניח שכל עיכוב הוא בדיוק 1.

## סוגי תקלות

1. Crash - אחד התהליכים קרס בשלב מסוים בביצוע. במערכת מסונכרנת יכול להיות שהקריסה הייתה באמצע ה-round ולכן חלק מהתהליכים קיבלו הודעה וחלק לא.
2. Byzantine - תהליך לא צפוי ועושה מה שבא לו. זה יכול להיות בגלל פריצת האקרים אליו. במצב כזה ניתן לעשות יותר נזק מאשר אם התהליך היה קורס. מערכת שיודעת להתמודד עם תקלה כזו תהיה עמידה בפני כל התקלות.
3. Omission - תהליך או קו תקשורת הפסיקו לעבוד אך רק באופן זמני. לדוגמה, חלק מההודעות הולכות לאיבוד.
4. Resilience - מספר תהליכים קרסו ולא רק אחד. במקרה זה המערכת צריכה להראות עמידות ולהמשיך לתפקד.

## תכונות נצרכות

כאשר מתמודדים עם מערכות ופרוטוקולים מבוזרים, ישנם מאפייני נכונות מסוגים שונים שצריכים להיות למערכת:

1. Safety - המערכת לעולם לא תבצע דברים רעים שאנו רוצים להימנע מהם. כדי להוכיח שהמערכת מקיימת תכונה זו יש להוכיח באמצעות אינווריאנטה שהמצב ההתחלתי בטוח, ולאחר מכן כל פעולה חוקית גם כן מביאה למצב בטוח.
2. Liveness - זוהי תכונה משלימה לבטיחות, שכן כדי לקיים בטיחות אפשר גם לא לעשות כלום. תכונה זו דורשת שמשהו טוב כן יקרה למישהו, כלומר המערכת תיתן לפחות לתהליך אחד לפעול. שקול ל-deadlock free.
3. Fairness - מוסיפה על Liveness שלא מספיק שמשהו טוב יקרה לתהליך אחד אלא לבסוף צריכה לעשות טוב לכולם, כלומר כל התהליכים צריכים לפעול. שקול ל-starvation free.

לדוגמה, עבור מערכת של רמזורים בצומת, המערכת תהיה בטוחה אם בכל זמן לכל היותר רמזור אחד מציג אור ירוק. כדי שהמערכת תקיים Liveness צריך שתמיד לפחות רמזור אחד יציג אור ירוק. וכדי שהמערכת תהיה הוגנת כל הרמזורים צריכים להיות מתישהו ירוקים.

## העברת הודעות - Massage Passing

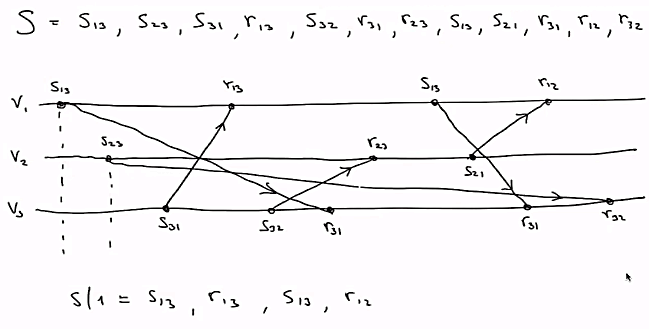
נגדיר את מודל העברת הודעות בצורה פורמלית יותר כדי שנוכל בהמשך לדון על נכונות מערכת כזו.

**הגדרת המודל**: במערכת העברת הודעות מכילה n צמתים , אנו מניחים שאנו יודעים את n. בכל זמן כל צומת נמצא באיזשהו מצב , התלוי בקלט לצומת, משתנים מקומיים שלו, שעון פנימי, והיסטוריית האירועים שהצומת הזו גילה. יש שלושה סוגים של אירועים:

* אירוע שליחה - של הודעה. האירוע גלוי אך ורק לצומת ששלח את ההודעה ולא למקבל (אצלו יהיה אירוע קבלה).
* אירוע קבלה - של הודעה. האירוע גלוי אך ורק לצומת שקיבל את ההודעה, השולח לא יודע שההודעה התקבלה.
* אירוע תזמון - זהו אירוע שהופעל בצומת על ידי שעון פנימי כלשהו.

### הגדרות נוספות

* Configuration: היא וקטור של כל המצבים של n הצמתים בזמן נתון - . מסומנת ב-C.
* Execution Fragment: קטע הרצה שמיוצג באמצעות רצף קונפיגורציות ואירועים לסירוגין - . כאשר לא חייבים להתחיל מ- שהיא קונפיגורציה התחלתית, וכל שלשה היא עקבית, כלומר כל אירוע הוא שמוביל מקונפיגורציה אל .
* Execution: זהו קטע הרצה שחייב להתחיל מ-.
* Schedule: זהו Execution רק בלי הקונפיגורציות, רק האירועים - . במילים אחרות, לוח הזמנים מכיל את האירועים של שליחה וקבלת הודעות עם הקלטים שלהם. במידה והרצף גם הגיוני, כלומר אין הודעה שנשלחת לפני שמתקבלת, נאמר שזהו Admissible Schedule, כלומר לוח זמנים קביל. כיוון שהמודל דטרמיניסטי מרצף האירועים ניתן לדעת גם את הקונפיגורציות, לכן השימוש ב- Scheduleיותר נוח. אנו תמיד מניחים שהאירועים נקבעו על ידי יריב ומסודרים לפי המקרה הגרוע ביותר.
* Schedule Restriction: בהינתן לוח זמנים S נגדיר את ההגבלה של צומת , המסומן ב-, להיות כל האירועים ב-S ש- יודע עליהן. נזכיר שאם צומת שלח הודעה לצומת , אזי במידה ו- יקבל את ההודעה לא ידע על כך. כמו כן כמובן שאינו יודע על אירועי שליחה של צמתים אחרים. דוגמה בשקף 26 במצגת 1.

צורה גרפית נוחה שמראה את ה-של כל הצמתים, היא לצייר לכל צומת i ציר זמן שעליו נכתוב את כל המאורעות שראה לפי הסדר. בין כל אירוע שליחה מ-i ל-j ואירוע קבלה מתאים , נמתח חץ כדי להראות את כיוון השליחה. באמצעות גרף זה אפשר לקבל את שכל צומת ראה בציר הזמן שלו, וכן להבין את סדר השליחה והקבלה בהרצה ולבדוק האם סדר זה אכן הגיוני.

### משפט אי ההבחנה (Indistinguishability)

בהינתן שני לוחות זמנים S ו-S’ של שתי הרצות שונות על אותה מערכת מבוזרת, ותהליך (צומת) כך שמתקיים: , כלומר רואה אותם אירועים וקלטים בשתי ההרצות, אזי יבצע בדיוק אותם פעולות ב-S וב-S’ שכן הוא אינו יכול להבחין בהבדל ביניהם. *במערכת רגילה משפט זה טריוויאלי. אמנם במערכת מבוזרת, שבה יכול להיות שכל שאר התהליכים מלבד בשני ההרצות עשו פעולות שונות, החידוש הוא ש- יעשה בשניהם בדיוק אותו דבר.*

***הגדרה:*** *עבור שתי הרצות* E *ו*-E’*, אם קיים צומת שבשניהם רואה את אותם* אירועים וקלטים, , אזי נאמר ששתי ההרצות דומות עבור , בסימון .

# בעיית שני הגנרלים

## תיאור הבעיה

שתי צבאות נלחמים זה בזה. הצבא הכחול, החזק יותר, בעמדה נחותה, ואילו הצבא האדום, החלש יותר, פיצל את כוחותיו לשתי עוצבות בגבעות שונות המכתרים את הצבא הכחול. לכל עוצבה יש גנרל. הצבא האדום ינצח רק אם שני הגנרלים של שני העוצבות יצליחו לתאם ביניהם לתקוף באותו הזמן. לשם כך הם מתקשרים ביניהם באמצעות יונות דואר. אמנם הצבא הכחול רואה את היונים ומנסה לירות בהן כדי לשבש את התקשורת של הצבא האדום. כיוון שהמרחק בין העוצבות גדול, גנרל ששולח יונה לא מצליח לראות האם היונה הגיע לצד השני, וכן הגנרל בצד השני לא יודע כאשר יונה נשלחת. הבעיה היא כיצד הגנרלים יכולים לתקשר ביניהם במצב כזה כדי להבטיח תקיפה של שניהם באותו הזמן שתוביל לניצחון?

הבעיה המתוארת לעיל היא אנלוגיה של הבעיה הבאה במערכות מבוזרות: עבור שני צמתים במערכת מבוזרת **מסונכרנת** שבה חלק מההודעות הנשלחות יכולות ללכת לאיבוד (Omission failure) וכל אחד מהצמתים יכול לקבל כקלט 0/1 ולפלוט 0/1, האם קיים אלגוריתם עבור שני הצמתים שיקיים את שלושת הדרישות הבאות:

1. Agreement - שני הצמתים צריכים להחזיר את אותו הפלט (0 או 1).
2. Termination - שני הצמתים מסיימים את פעולתם במספר סופי של סיבובים (rounds).
3. Validity - אם שני הצמתים קיבלו אותו קלט וגם לא היה איבוד של הודעות בשני הכיוונים אזי שני הצמתים יפלטו את הקלט שלהם. אם אחד משני התנאים לא מתקיים אין חובה כזו.

נשים לב שללא Validation יש אלגוריתם מאוד פשוט שפותר את הבעיה - אלגוריתם שתמיד מחזיר 0. בעיית שני הגנרלים מייצגת בעיה כללית יותר של כיצד תהליכים מגיעים להסכמה במערכת מבוזרת שבה לא בטוח שכל ההודעות מגיעות ליעד. ישנם מקרים רבים בהם תהליכים צריכים להגיע להסכמה, כמו לדוגמה: מי מוביל חישוב, כיצד לחלק סדר פעולות, האם לבצע פעולה או לא, מי הראשון שניגש למשאבים, וכו'.

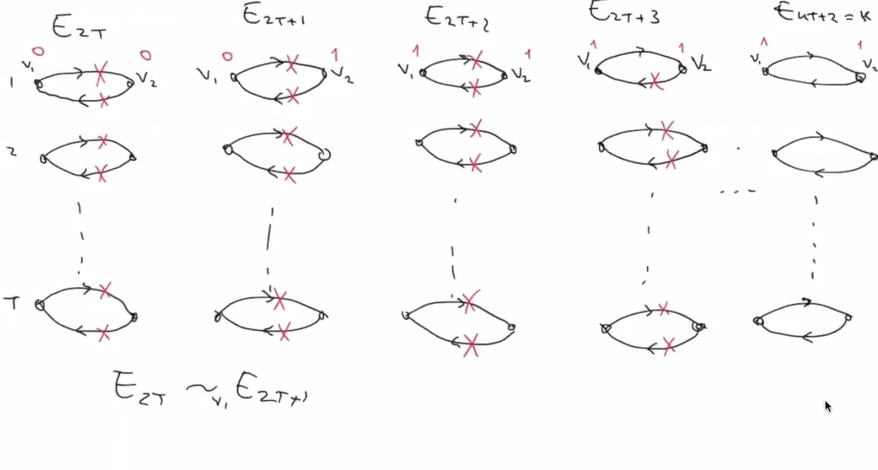
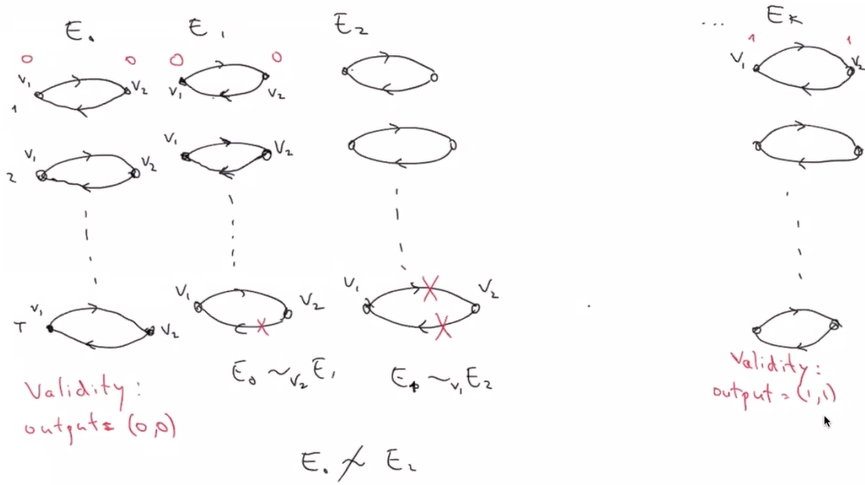
## הוכחת אין פתרון

**משפט:** לבעיה המוצגת לעיל, למרות שנראית פשוטה, לא יכול להיות פתרון!

**הוכחה:** נניח בשלילה שיש אלגוריתם כזה שעבור כל הרצה אפשרית עונה על כל דרישות הבעיה. נגדיר רצף של הרצות (Executions) כך שכל שתי הרצות סמוכות [דומות](#_משפט_אי_ההבחנה), כלומר לכל קיים צומת עבורה . נניח בנוסף שכל הרצה מחזירה פלט לאחר בדיוק T סיבובים. נוכל להניח זאת שכן ההרצות חייבות להסתיים לאחר מספר לא סופי של סיבובים ותמיד ניתן להוסיף סיבובים שלא משפיעים על הפלט.

אם נצליח להוכיח שעבור הרצה שני הצמתים חייבים להחזיר 0 ועבור הרצה שני הצמתים צריכים להחזיר 1 סיימנו, שכן אם בהרצה שניהם החזירו 0 אזי גם בהרצה שניהם יהיו חייבים להחזיר 0, שהרי ב- אחת הצמתים בוודאי החזירה 0. כך נמשיך ברצף עד שנגיע ל- שגם בו שני הצמתים יהיו חייבים להחזיר 0. אמנם כיוון שנוכיח שחייבים להחזיר 1 נקבל שהאלגוריתם יכשל עבור הרצה זו, בניגוד להנחה.

נוכיח שקיים רצף של הרצות כזה. בהרצה שני הצמתים יקבלו כקלט 0 וכל ההודעות יצליחו. הרצה זהה ל- אלא שההודעה האחרונה מ- לא הגיעה ל-. מבחינת אין הבדל ולכן ההרצות דומות. כך נמשיך כל הרצה לבטל את ההודעה האחרונה עד שבה כל ההודעות לא נשלחו. ב- נשנה רק את הקלט של ל-1, וב- נשנה את הקלט של ל-1. מ- כל הרצה נוסיף הודעה שבהרצות הקודמות נכשלה. בהרצה ה- נקבל ששני הצמתים קיבלו כקלט 1 וכל ההודעות נשלחו, לכן שניהם חייבים להחזיר 1. אמנם ה צריכים להחזיר 0. סתירה!



## תיאור חדש לבעיית שני הגנרלים

כיוון שהוכחנו שלבעיה זו אין אלגוריתם שפותר אותה נשנה את הבעיה מעט. ניתן לפתור את בעיית שני הגנרלים אם:

1. אנו מרשים לגנרלים להטיל מטבע ולקבל החלטות על סמך התוצאה. מספיק שגנרל אחד יקיים תנאי זה.
2. אנו מסופקים גם אם תנאי Agreement (שני הצמתים צריכים להחזיר את אותו הפלט) מתקבל בהסתברות .

בשני תנאים אלו אנו גם מגדילים את היכולת וגם מקילים את הדרישה. נראה אלגוריתם מקרי (Randomized Algorithm) שפותר את הבעיה החדשה. אך לפני כן נצטרך להגדיר אלגוריתם עזר, שנקרא Level Algorithm, שנשתמש בו באלגוריתם המקרי שנציג.

### Level Algorithm

עבור שני צמתים u, v השולחים הודעות אחד לשני (לא בטוח שכל ההודעות מגיעות ליעד), נגדיר לכל צומת מספר הנקרא "הרמה (Level) של הצומת". מספר זה מייצג את מספר ההודעות שנשלחו וחזרו בהצלחה. נרצה אלגוריתם שבסוף ריצתו שתי הרמות של הצמתים יהיו בהפרש של לכל היותר אחד. Level Algorithm עונה על דרישה זו. הוא מבצע את הפעולות הבאות:

1. שתי הרמות מאותחלות ל-0.
2. בכל סיבוב שני הצמתים שולחים את הרמה הנוכחית שלהם אחד לשני.
3. כאשר צומת u עם רמה ℓu מקבל הודעה מצומת v, עם הרמה ℓv, אזי u מעדכן את רמתו להיות: ℓu = max {ℓu, ℓv+1}.

אבחנות:

* הרמה של צומת לעולם לא יורדת.
* בכל סיבוב הרמות של שני הצמתים יהיו בהפרש של לכל היותר 1.
* אם כל ההודעות מתקבלות אזי רמות שני הצמתים שוות למספר הסיבובים.
* הרמה של צומת u שווה ל-0 אם ורק אם u לא קיבל אף הודעה. לכן אם הרמה לכל הפחות 1, בהכרח u קיבל הודעה.

ההוכחה לכל אחת מאבחנות אלו היא באמצעות אינדוקציה פשוטה ונמצאת במצגת 2.

## Randomized Two Generals Algorithm

1. נניח שצומת u יכול להטיל מטבע ולקבל החלטות על סמך התוצאה, אך צומת v לא יכול.
2. צומת u בוחר מספר באופן אחיד ואקראי.
3. שני הצמתים מריצים את Level Algorithm למשך r סיבובים. בכל סיבוב כל אחד מהם שולח את ערך הקלט ההתחלתי שלו. צומת u בנוסף גם כולל את הערך t שבחר באקראי בכל הודעה.
4. צומת תפלוט 1 אם מקיימת שלושה תנאים:
5. הצומת יודע מהו t וגם ראה את הקלט של הצומת המקבילה.
6. הקלט של שני הצמתים הוא 1.
7. הרמה של הצומת היא לפחות t.
8. אחרת, הצומת תפלוט 0.

אבחנות:

* אם לפחות אחד הקלטים 0 שני הצמתים יחזירו 0.
* אם שני הקלטים 1 ואף צומת לא נאבדה - שניהם יחזירו 1.
* אם שני הקלטים 1 ויש הודעות שנאבדו - שני הצמתים יחזירו אותו הפלט אלא אם כן , שאז יחזירו פלט שונה והאלגוריתם יכשל. ההוכחה נובעת מהגדרת האלגוריתם (ניתן לראות פירוט בשקף 23).

**טענה:** האלגוריתם מקיים Agreement עם הסתברות לכל הפחות . ככל שנבחר r גדול יותר כך נוכל לקרב את ההסתברות ל-1.

**הוכחה**: האלגוריתם יסתיים לאחר r סיבובים ולכן מקיים Termination. בנוסף, משתי האבחנות הראשונות ניתן לראות שמקיים Validation. נוכיח את הטענה. נדמיין שהיריב מנסה להכשיל את האלגוריתם שלנו. הוא שולט בקלט ובאיזה הודעות ישלחו ואיזה לא. היריב יבחר קלט (1,1) אחרת האלגוריתם תמיד יצליח. כעת, הוא ירצה להכשיל מספר הודעות כך שהרמות של הצמתים יהיו , אחרת האלגוריתם שוב תמיד יצליח. אמנם היריב אינו יודע מהו t שכן נבחר באקראי, ולכן הוא חייב לבחור כלשהו. היריב אכן יצליח להכשיל את האלגוריתם בהסתברות .

\*להשלים סוף הרצאה 3 (15 דק אחרונות) וסוף תרגול 3 (בקובץ) הוכחת חסם תחתון על ההסתברות לטעות .

# בעיית השידור ברשתות

## הגדרת הבעיה

בפרק זה נדון בבעיה הבאה: צומת v רוצה לשלוח הודעת שידור (Broadcast) M לכל שאר הצמתים בגרף לא מכוון G. הבעיה היא שהגרף הוא לא בהכרח קליקה וכל צומת יכול לשלוח הודעה רק לשכנים שלו בגרף, אליהם הוא מחובר דרך צלע. לכל צומת יש ID ייחודי וכל צומת מכיר את ה-ID של כל השכנים שלו, כך שכל צומת שמקבל הודעה יודע ממי הוא קיבל. המערכת היא א-סינכרונית, כלומר זוהי מערכת שבה כל ההודעות תמיד מגיעות ליעד (ללא תקלות), אך זמן העיכוב של כל הודעה אינו ידוע ונקבע על ידי יריב. כן ידוע שזמן העיכוב הוא סופי. בנוסף, זוהי מערכת מסוג event-based שבה כאשר צומת מקבל הודעה משכן הוא מבצע חישובים מקומיים ושולח הודעה לשכן אחד או יותר. נרצה למצוא אלגוריתם שידע לבצע שידור במערכת כזו עם סיבוכיות זמן וסיבוכיות שליחת הודעות מינימליים.

## אלגוריתם Flooding

זהו אחד האלגוריתמים המבוזרים הפשוטים ביותר. הרעיון הוא שכל צומת המקבל את M בפעם הראשונה ישלח אותו לכל השכנים שלו מלבד הצומת ממנה קיבל את ההודעה. שלבי האלגוריתם הם:

* צומת v שולח את ההודעה M לכל השכנים שלו.
* כל שאר הצמתים בקבלת M משכן כלשהו w: אם M לא התקבל בעבר שלח את M לכל השכנים חוץ מ-w. אחרת, אל תעשה כלום.

לפי שנחשב את סיבוכיות הזמן וסיבוכיות שליחת ההודעות נגדיר מספר הגדרות על מרחקים בגרף.

### הגדרת מרחקים בגרף

* - זהו המרחק הקצר ביותר בגרף G בין צומת v לצומת u.
* *- זהו המרחק המקסימלי מבין כל הצמתים אל* v*. .*
* *- זהו הרדיוס המינימלי מבין כל צמתי הגרף. .*
* *- זהו הרדיוס המקסימלי מבין כל צמתי הגרף או במילים אחרות המרחק המקסימלי בין כל שני צמתים. .*

*נשים לב שמתקיים: . האי-שוויון הראשון הוא בגלל שנוכל ליצור מסלול בין כל שני צמתים בגרף* v, u *באמצעות המרחק מ-*v *אל צומת הרדיוס ומצומת הרדיוס אל* u*, שהוא לכל היותר פעמיים הרדיוס של* G*. מכל אי-שוויוניים אלו נוכל להסיק שמבחינת סיבוכיות כל גדלים אלו שקולים.*

### סיבוכיות האלגוריתם

לצורך חישוב הסיבוכיות ננרמל את כל הזמנים שבהם היריב יכול לעכב את קבלת ההודעות לערכים בין 0 ל-1, נניח שכל החישובים הפנימיים לוקחים 0 שניות ושההודעה הראשונה נשלחה בזמן 0.

**סיבוכיות זמן** - של אלגוריתם Flooding מצומת v במערכת א-סינכרונית הוא (לא O של אלא בדיוק).

הוכחה: נוכיח בשני כיוונים שהסיבוכיות גדולה שווה ל-וגם קטנה שווה ל-, ולכן שווה לה.

כיוון 1 - עבור צומת כלשהי u עבורה בוודאי ש-u יקבל את ההודעה לפחות בזמן , שכן מניחים כל עיכוב הוא לכל היותר 1. טענה זו בעצם מלמדת אותנו שסיבוכיות זמן במערכת א-סינכרונית תמיד תהיה לפחות הסיבוכיות במערכת סינכרונית.

כיוון 2 - עבור צומת כלשהי u עבורה , בוודאי מתקיים . נוכיח באינדוקציה על t כי הזמן שלוקח ל-u לקבל את M הוא לכל היותר t.

בסיס: עבור נקבל כי ולכן הטענה מתקיימת.

צעד: נניח שהטענה נכונה עבור ונוכיח עבור t+1. יהי מסלול באורך t+1 מ-v אל u. לפי הנחת האינדוקציה, M יגיע אל בזמן לכל היותר t. לכן M יגיע אל u בזמן t+1 או לפני כן. מ.ש.ל.

**סיבוכיות הודעות** - יהי . לפי תיאור האלגוריתם u יכול לשלוח הודעה ל-w לכל היותר פעם אחת וכן בכיוון ההפוך. לכן מספר ההודעות הוא לכל היותר פעמיים מספר הצלעות - .

### עץ פורש

אלגוריתם Flooding יוצר לנו עץ פורש בגרף G. שורש העץ הוא v, ועבור כל צומת אחר - צומת האב שלו הוא הצומת הראשון ממנו קיבל את M. אמנם נשים לב שיכול להיות הבדל בצורת העץ בין מערכת סינכרונית למערכת א-סינכרונית. במערכת סינכרונית העץ הפורש שנוצר על ידי האלגוריתם תמיד ייראה אותו הדבר, ובו הדרגה של כל צומת u היא בדיוק , כלומר המרחק הקצר ביותר ממנו אל v. עץ כזה נקרא “עץ המסלולים הקצרים” או "עץ BFS". אולם במערכת א-סינכרונית כל עץ פורש יכול להיווצר על ידי אלגוריתם Flooding. מי שמחליט איזה עץ ייווצר הוא היריב. המקרה הגרוע ביותר הוא עץ פורש שמחבר את כל הצמתים בענף אחד ועומקו הוא n-1.

## Convergecast

הפעולה ההפוכה משידור (Broadcast) היא Convergecast שבה כל הצמתים שולחים הודעה לצומת v. סיבה אפשרית לרצות לעשות Convergecast היא כדי להודיע ל-v שכולם קיבלו את הודעת השידור שלו.

### אלגוריתם עבור Convergecast:

* לבנות עץ פורש שהשורש שלו הוא v.
* כל עלה u בעץ הפורש: שלח את ההודעה רק לצומת האב שלך.
* כל צומת פנימית w בקבלת הודעה מילד x: אם w קיבל הודעות **מכל** ילדיו שולח הודעה לצומת האב שלו, אחרת ממתין.
* צומת השורש v בקבלת הודעה מילד x: אם v קיבל הודעות **מכל** ילדיו אזי ה- Convergecastמסתיים, אחרת ממתין.

### סיבוכיות האלגוריתם:

**סיבוכיות הזמן** - בדיוק **עומק העץ** הפורש.

**סיבוכיות הודעות** - בדיוק שכן בעץ פורש כל צלע מוסיפה צומת.

## אלגוריתם Flooding/Echo

זהו אלגוריתם שקודם מבצע Broadcast משורש v ולאחר מכן Convergecast על העץ הפורש ש-Flooding יצר. תיאור מפורט של האלגוריתם:

1. הפעל אלגוריתם Flooding כדי לשלוח את M מ-v אל כל הצמתים.
2. Flooding במהלך ריצתו גם מייצר עץ פורש ששורשו v, כמו שתיארנו לעיל, על ידי כך שכל צומת מגדיר את הצומת הראשון ממנו קיבל את M להיות צומת האב בעץ הפורש. בנוסף, כדי שכל צומת ידע מי הבנים שלו, כל צומת המקבל את M במהלך Flooding, מודיע לצומת ממנה קיבל את ההודעה שהוא צומת האב שלו אם זוהי הפעם הראשונה, אחרת מודיע לו שהוא לא צומת האב שלו. צומת שכל הצמתים אליהם שלח את M מודיעים לו שהוא לא צומת האב שלהם, מגדיר את עצמו כעלה בעץ הפורש.
3. הפעל אלגוריתם של Convergecast כדי להחזיר חזרה הודעה מכל הצמתים אל v (זהו שלב ב-echo).

### סיבוכיות האלגוריתם:

**סיבוכיות הזמן** - חיבור הסיבוכיות של Flooding עם Convergecast. במערכת סינכרונית העץ הפורש שנוצר מאלגוריתם Flooding הוא עץ BFS ולכן הסיבוכיות היא . אולם במערכת א-סינכרונית היריב יכול לקבוע במקרה הגרוע שהעץ הפורש יהיה בעומק n-1. לכן הסיבוכיות היא . אלגוריתם זה זוהי דוגמה לכך שסיבוכיות במערכת א-סינכרונית יכול להיות גרוע הרבה יותר מסיבוכיות במערכת סינכרונית.

בסעיפים הבאים נחקור אלגוריתמים נוספים לבניית עץ פורש במערכת מבוזרת, שאמנם הסיבוכיות זמן שלהם יותר גדולה אך העץ הפורש שנבנה מהם יותר מאוזן, ואז אלגוריתם Convergecast ייקח פחות זמן.

**סיבוכיות הודעות** - בכל סוגי המערכות היא .

## אלגוריתם Dijkstra

זהו אלגוריתם שבו כל צומת מתחזק אומדן של המרחק שלו מ-v, בהתחלה כל המרחקים מאותחלים לאינסוף מלבד v המאותחל ל-0. בנוסף האלגוריתם מתחזק תור עדיפויות או ערימת מינימום שבתחילת האלגוריתם מכילה את כל הצמתים. בכל איטרציה מוציאים מהתור את הצומת u שהאומדן שלו הכי נמוך ומעדכנים את האומדן של כל השכנים שלו ממנו. כלומר לכל האומדן יהיה . נרצה להמיר אלגוריתם זה למערכת מבוזרת עם שמירה על המאפיינים הייחודים לו, שהם: בנייה של העץ הפורש רמה אחר רמה ושאין אפשרות לתקן אומדן של צומת לאחר שמוציאים אותו מהתור.

### אלגוריתם Dijkstra מבוזר

באלגוריתם זה בונים את "עץ המסלולים הקצרים" כך שבכל איטרציה r בונים את העץ עד רמה r.

* עבור בלולאה עד שהשורש v מסיים את האלגוריתם:
* צומת v שולח broadcast לכל הצמתים בעץ הפורש שנבנה עד שלב זה. בשידור זה מודיע "התחילו את שלב “r+1.
* השידור מגיע עד לעלים בעץ (הנמצאים ברמה r). עלים אלו שולחים הודעה לכל השכנים שלא קיבלו מהם את ההודעה "התחילו את שלב r+1", ובה הם מודיעים להם "הצטרפו לרמה r+1". הודעה אחרונה זו יכולה להגיע לצמתים ברמה r+1 או צמתים ברמה r.
* כל צומת u שמקבל את ההודעה "הצטרפו לרמה r+1" מצומת w, אם זו הפעם ראשונה (u ברמה r+1) מגדיר את w כצומת האב שלו ומחזיר לו ACK, אם זו לא הפעם הראשונה (u ברמה r) מחזיר לו NACK.
* כאשר עלה מקבל ACK/NACK מכל מי ששלח לו הודעה מתחיל Convergecast עם הודעה "התחילו שלב r+2". אמנם אם קיבל רק NACK מכל מי ששלח לו הודעה, הוא מבין שהוא עלה אמיתי בעץ הפורש הסופי, ומתחיל Convergecast עם ההודעה "סיימתי".
* כאשר v מקבל מכל הבנים את הודעת ה-Convergecast, אם בחלק מהן יש "התחילו שלב r+2" עובר לאיטרציה הבאה. אחרת, כל ההודעות הן "סיימתי", מסיים את האלגוריתם (אפשר להוסיף שמתחיל שידור לכל הצמתים עם הודעה "השידור הסתיים").

### סיבוכיות האלגוריתם

**סיבוכיות הזמן** - בכל איטרציה r מבצעים קודם Flooding בעץ הנוכחי T, שהסיבוכיות שלו היא . ולבסוף מבצעים Convergecast שהסיבוכיות שלו היא עומק העץ, שבמקרה הזה גם כן שווה ל-r. בנוסף, צריך להוסיף שתי יחידות זמן שבהם העלים שולחים לשכנים שלהם "הצטרפו לרמה r+1" ומקבלים חזרה ACK/NACK. סך הכל הסיבוכיות של כל איטרציה r היא . כדי לקבל את סיבוכיות הזמן של האלגוריתם נסכום את סיבוכיות כל האיטרציות, שמספרם לכל היותר , ונקבל **.**

**סיבוכיות הודעות** - נחלק לשני סוגי הודעות:

1. סיבוכיות ההודעות של Flooding ו-Convergecast בכל איטרציה הוא מספר הצלעות בעץ הפורש הנוכחי, שהוא n-1, כאשר n הוא מספר הצמתים בעץ. n הוא לכל היותר בגודל . אם נחשב זאת עבור כל איטרציה נקבל .
2. נספור את מספר ההודעות שהעלים בכל איטרציה r שולחים לשכנים שלהם ואת התגובה חזרה. נשים לב שניתן לחלק כל צלעות הגרף G בהתאם לרמות העץ לפי הודעות אלו, שכן כל ההודעות מרמה r לרמה r או לרמה r+1 ישלחו רק באיטרציה ה-r אך לא באיטרציות אחרות. לכן סכום ההודעות מסוג זה הוא , שכן לכל הודעה יש גם תגובה.

סך הכל, סיבוכיות ההודעות היא .

## אלגוריתם Bellman-Ford

גם באלגוריתם זה כל צומת מתחזק אומדן של המרחק שלו מ-v, בהתחלה כל המרחקים מאותחלים לאינסוף מלבד v המאותחל ל-0. עוברים בלולאה פעמים, כך שבכל איטרציה כל צומת מעדכן את המרחק שלו בהתאם לאומדן של כל השכנים שלו. כלומר האומדן של כל צומת u יהיה . נרצה להמיר אלגוריתם זה למערכת מבוזרת עם שמירה על המאפיינים הייחודים לו.

### אלגוריתם Bellman-Ford מבוזר

גם כאן כל צומת 𝑢 מאחסן מספר שלם המייצג את הניחוש הנוכחי שלו למרחק לשורש 𝑣. בכל פעם שצומת 𝑢 יכול לשפר את , הוא עושה זאת ואז מודיע לכל שכניו.

* אתחול: לכל צומת מאתחלים ולשורש v מאתחלים .
* השורש v שולח "1" לכל השכנים שלו.
* כל עוד האלגוריתם לא נעצר:
* כל צומת u בקבלת הודעה מצומת w עם ערך t המקיים : מעדכן , קובע w להיות צומת האב של u, ושולח הודעה “t+1” לכל השכנים מלבד w.

ידוע כי צריך לכל היותר סיבובים של השלב האחרון כדי שכל האומדנים יהיו מדויקים ונקבל את "עץ המסלולים הקצרים". אמנם כיצד נדע במקרים בהם מספיקים פחות סיבובים? יש לכך פתרון אך לא נלמד עליו בקורס זה.

### סיבוכיות האלגוריתם

**סיבוכיות זמן** - של אלגוריתם Bellman-Ford מבוזר הוא .

הוכחה: נוכיח בשני כיוונים שהסיבוכיות גדולה שווה ל-וגם קטנה שווה ל-, ולכן שווה לה.

כיוון 1 - עבור צומת כלשהי u עבורה בוודאי ש-u יקבל את ההודעה לפחות בזמן , שכן מניחים כל עיכוב הוא לכל היותר 1.

כיוון 2 - עבור צומת כלשהי u עבורה , בוודאי מתקיים . נוכיח באינדוקציה על t כי עד זמן t אכן יתקיים כנדרש.

בסיס: עבור נקבל כי ואכן בזמן 0 מתקיים .

צעד: נניח שהטענה נכונה עבור ונוכיח עבור t+1. יהי מסלול באורך t+1 מ-v אל u. לפי הנחת האינדוקציה, בזמן t יתקיים , ולכן לאחר לא יותר מסיבוב אחד u יקבל את ההודעה t מ- ויעדכן כנדרש.

**סיבוכיות הודעות**: יהי . לפי תיאור האלגוריתם u יכול לעדכן את האומדן שלו לכל היותר פעמים, שכן זהו המרחק הגדול ביותר האפשרי בעץ. לכן u יכול לשלוח הודעה ל-w לכל היותר פעמים. כן הוא גם בכיוון ההפוך מ-w ל-u. טענה זו נכונה לכל צלע בגרף, לכן מספר ההודעות המקסימלי האפשרי הוא **.**

## סיכום

**במערכת סינכרונית:**

* כדאי להשתמש באלגוריתם Flooding עם סיבוכיות זמן , וסיבוכיות הודעות .

**במערכת א-סינכרונית**:

* אלגוריתם Dijkstra מבוזר - סיבוכיות זמן , וסיבוכיות הודעות .
* אלגוריתם Bellman-Ford מבוזר - סיבוכיות זמן , וסיבוכיות הודעות .

נשים לב כי דיאקסטרה עדיף כאשר מתחשבים בעיקר בסיבוכיות הודעות ואילו בלמן פורד עדיף כאשר מתחשבים בעיקר בסיבוכיות זמן. האלגוריתם הטוב ביותר הקיים היום המציג פשרה בין סיבוכיות הזמן לשליחת הודעות הוא אלגוריתם המציג סיבוכיות זמן , וסיבוכיות הודעות . אלגוריתם זה מבוסס על “synchronizers” שהם כלי כללי לתרגום אלגוריתמים סינכרוניים לא-סינכרוניים.

# סיבתיות, זמן לוגי ומצבים גלובליים

## הגדרת הבעיה