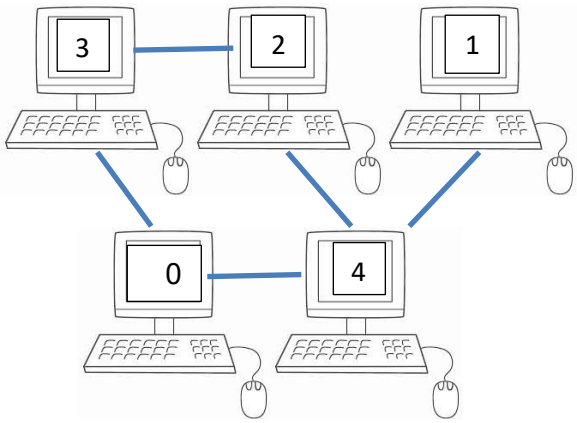
# פרוייקט סיום תכנות מתקדם תשפ"ה סמסטר ב'

## תיאור הבעיה - כללי

נתונה רשת מחשבים, כאשר מחשבים שונים מחוברים בקווי תקשורת דו-כיווניים ביניהם.

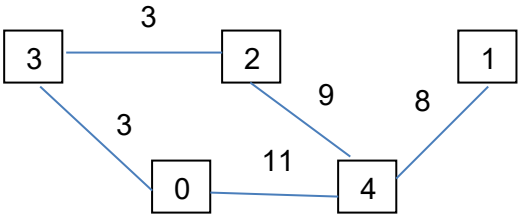
לא כל המחשבים מחוברים לכולם.

לדוגמא:

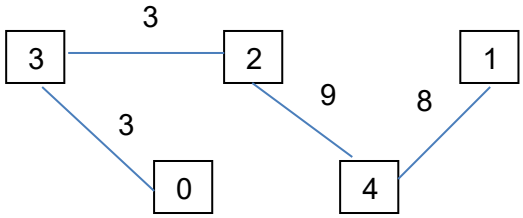



מחשב שיש בידיו הודעה – יכול להעביר אותה למחשבים אליהם הוא מחובר, והם יכולים להעביר אותה הלאה באותו האופן. ברשת תקינה יש דרך להעביר הודעה מכל מחשב לכל מחשב. לדוגמה, ברשת לעיל מחשב מס' 1 יכול להעביר הודעה למחשב מס' 3 בדרך הבאה: 3-0-4-1 או: 3-2-4-1.

לכל קו יש עלות בשקלים. למשל, ברשת לעיל ייתכן שהמחירים הם:




חברת SpaceY מעוניינת לרכוש חלק מקוי התקשורת כך שעדיין תתאפשר העברת הודעות מכל מחשב לכל מחשב. החברה מעוניינת שסך הרכישה תהיה בעלות מינימלית. למשל, בדוגמה אם החברה תרכוש את כל הקווים, העלות תהיה 8+9+11+3+3=34 אבל אם היא תרכוש רק את הקווים הבאים:




העלות תהיה 8+9+3+3=23 ועדיין תהיה אפשרות להעביר הודעות בין כל המחשבים. שימו לב שבדוגמה 23 היא אכן העלות המינימלית.

בהינתן רשת מחשבים ובה n מחשבים ו-m קוי תקשורת (בדוגמה n=5 ו: m=5) מטרת התכנית היא למצוא n-1 קוי תקשורת כך שאם החברה SpaceY תרכוש אותם, כל מחשב יוכל להעביר הודעה לכל מחשב ברשת והעלות הכוללת של רכישת הקווים תהיה מינימלית.

עליכם לכתוב תכנית שתקבל את מספר המחשבים ברשת, וקוי התקשורת כולל עלות כל קו ותבצע את ארבע המשימות שיפורטו בהמשך:

א. קלט מהמשתמש של הקווים המחברים ועלויותיהם ובניית **רשת מחשבים**.
ב. מעבר על הרשת ובניית אוסף הקוים שכדאי לחברה SpaceY לרכוש. אוסף זה של קוי תקשורת נקרא **עץ פורש מינימלי** והוא יישמר כמערך כפי שיתואר להלן.
ג. מעבר על העץ הפורש המינימלי, ובניית **רשת מסלולים** בפורמט שבו, בהינתן שני מחשבים התכנית תדע לחשב את המסלול ביניהם.
ד. קבלת שני מספרים של מחשבים ואת רשת המסלולים, וחישוב והדפסת המסלול ביניהם.

להלן המשימות בפירוט. בסוף המסמך מצורף קוד שמכיל את פונקציית ה-main ו-prototypes של פונקציות המשימות.

## משימה א' – קלט ובניית רשת המחשבים

הפונקציה תקבל כקלט את מספר המחשבים, n, (לצורך פשטות נניח שהם ממוספרים מ- 0 עד n-1) ותקלוט מהמשתמש את מספר הקווים, m.

לאחר מכן, הפונקציה תקלוט מהמשתמש שלשות שמייצגות קוי תקשורת ועלות : a,b,price . למשל בדוגמה למעלה יתקבל הקלט הבא:

5

1 4 8

3 2 3

3 0 3

2 4 9

4 0 11

במהלך הקליטה, תיבנה רשת המחשבים.

<u>רשת המחשבים</u>: הרשת תאוחסן במערך בשם Net של רשימות מקושרות באופן הבא:

המערך יהיה בגודל n וכל תא u במערך יכיל רשימה מקושרת של המחשבים שמחשב u מחובר אליהם (ראו להלן EdgeList). שימו לב שקו (a,b) וקו (b,a) הם למעשה אותו קו ולכן גם ברשימה של a יופיע b וגם ברשימה של b יופיע a.

<u>הכנסה ממוינת</u>: על מנת לקבל תוצאות אחידות, עליכם לבצע הכנסה לרשימות באופן ממוין באופן עולה על פי מספר המחשב. כך שבסוף אם נניח, כמו בדוגמה, למחשב 2 ישנם שני מחשבים המחוברים אליו: 3 ו-4, אז 3 יופיע לפני 4 ברשימה המקושרת גם אם החיבור (2,4,9) התקבל בקלט לפני (3,2,3).

לצורך כך, השתמשו במבנים הבאים:

```
typedef struct {
        int neighbor;
        int cost;
} Edge;

typedef struct edge_node {
        Edge e;
        struct edge_node *next;
} EdgeNode;

typedef struct {
        EdgeNode *head;
        EdgeNode *tail;
} EdgeList;
```

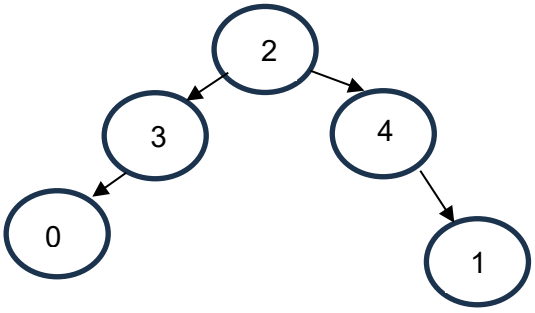
## משימה ב' – בניית עץ פורש מינימלי

במשימה זו הפונקציה תקבל את הרשת ותייצר ממנה את אוסף הקוים שכדאי לחברה SpaceY לרכוש. אוסף זה של קוי תקשורת נקרא **עץ פורש מינימלי**. העץ יישמר כמערך של הורים בשם **Prim**, כך שלשורש העץ אין הורה, ולכל איבר אחר v יישמר במערך Prim באינדקס v, שם ההורה של v. למשל, עבור הדוגמה למעלה, המערך יכול להיות:

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 3 | 4 | -1 | 2 | 2 |

ההורה של מחשב 0 הוא 3, ההורה של מחשב 1 הוא 4, מחשב 2 הוא שורש העץ, לכן הוא היחיד ללא הורה ובמערך זה מסומן כ – 1-, וההורה של מחשב 3 ושל מחשב 4 הוא 2.

ניתן לצייר זאת כך:




שימו לב שלאו דווקא מדובר בעץ בינארי.

האלגוריתם שבו תשתמשו יבנה את העץ הפורש Prim בהדרגה. יתחיל ממחשב כלשהו (שורש העץ, ללא הורה), ויוסיף אליו בכל שלב מחשב נוסף עד שיצורפו כל המחשבים. בכל שלב ייבחר המחשב שעלות הוספתו היא הזולה ביותר.

## אלגוריתם בניית Prim

לכל מחשב v שאינו בעץ הפורש עדיין, יש אפשרויות שונות להצטרף לעץ דרך מחשבים שכבר קיימים בו. המחשב בעץ שדרכו החיבור הוא הזול ביותר ייקרא Prim(v) ועלות הקו המחבר בין v לבין Prim(v) ייקרא min(v). במידה ול-v ברגע נתון אין אפשרות להצטרף לעץ כי אינו מחובר לאף מחשב בו, אז min(v) יכיל את הערך INT_MAX (הקבוע מוגדר בקובץ limits.h) ו- Prim(v) יכיל 1-. בנוסף, הערך הבולאני InT(v) מציין האם v נמצא בעץ הפורש המינימלי או עדיין לא.

בתחילת האלגוריתם אף מחשב לא נמצא בעץ הפורש שהתכנית בונה, ולכן מערך inT מכיל false ולכל v מתקיים min[v]=INT_MAX, Prim[v] = -1.

במהלך האלגוריתם ערכים אלו מתעדכנים למחשבים שעדיין לא נמצאים בעץ הפורש. עבור מחשב v שהצטרף לעץ הפורש מתקיים ש- Prim[v] הוא ההורה שלו בעץ ועלות החיבור בינו לבין ההורה הינה min[v] וערכים אלו נותרים עד סוף האלגוריתם.

## מבני נתונים בהם משתמש האלגוריתם

1. מערך inT: **מערך ביטים** בגודל $\lceil n/8 \rceil$ המכיל ערך 1 בביט ה-v אם המחשב ה-v כבר נמצא בעץ הפורש. הביטים מתחילים מה- least significant bit ב- least significant byte.
2. מערך min: מערך המכיל לכל מחשב v את הערך המינימלי שמחבר אותו לעץ הפורש.
3. רשימת העדיפויות priority: רשימה מסוג CandidateList של המחשבים שטרם הצטרפו לעץ הפורש. הרשימה לא תהיה ממוינת. שימו לב שרשימה זו היא **דו-כוונית**.
לצורך כך השתמשו במבנים הבאים:

```
typedef struct {
        int computer;
        int min;
} Candidate;

typedef struct candidate_node {
        Candidate c;
        struct candidate_node *next;
        struct candidate_node *prev;
} CandidateNode;

typedef struct {
        CandidateNode *head;
        CandidateNode *tail;
} CandidateList;
```

4. מערך location: מערך המכיל לכל מחשב v מצביע לאיבר ב- Priority שמכיל אותו.

כלומר זה מערך של מצביעים ל- CandidateNode.

## פונקציות עזר

1. הפונקציה DeleteMin מקבלת את הרשימה priority ומחזירה את v עבורו הערך min הנמוך ביותר מבין המחשבים v שנמצאים ב- רשימה priority. הפונקציה מוחקת את האיבר v מ- priority.
2. הפונקציה DecreaseKey מקבלת את המערך location ואת האיבר v ואת הערך החדש ומקטינה את ערך ה- min שלו לערך החדש.

## פסאודו קוד של האלגוריתם:

For every v: min value is INT_MAX and Prim value is -1.

Build priority by inserting all elements to it and make location[v] point to the element in priority that contains computer number v.

Build inT such that for every v the value is false.

Pick initial computer from user, call it $v_0$

```
min[v0] ← 0                    //this is the first computer to join the tree Prim

Prim[v0] ← -1

while  priority ≠ <empty-list>  do                    // Grow Tree

        u ← DeleteMin(PQ)       // Get the lowest element in PQ

        if min[u]=INT_MAX

                STOP the program with the output "Cannot build Prim"

        Else continue as follows

        InT[u] ← true           // Set the u's bit to 1

        for each v ∈ Net[u]  do

                if  (not InT[v])  and cost(u,v) < min[v]) then

                        min[v] ← cost(u,v)

                        Prim[v] ← u

                        DecreaseKey(location,v)

                End-if

        End-for
```

**End-while**

לצורך אחידות הניחו ש- $v_0$ הוא המחשב 0.

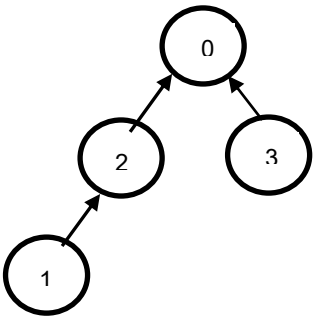
## משימה ג' – בניית רשת מסלולים

במשימה זו הפונקציה תקבל את המערך Prim ותבנה רשת בשם PrimPaths באופן הבא:

לכל מחשב v "שכן" של v הוא

א. הערך Prim[v]
ב. u כך ש: Prim[u]=v.

למשל, אם בשלב הקודם התכנית ייצרה עץ פורש מינימלי שנראה כך:

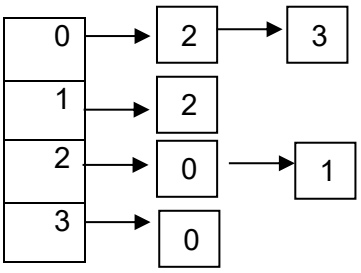



כזכור הייצוג הוא במערך Prim שנראה כך:

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| -1 | 2 | 0 | 0 |

(כי 0 הוא שורש העץ, ל-2 ול-3 ההורה הוא 0 ול-1 ההורה הוא 2.)

הפונקציה תייצר מבנה רשת בדומה לרשת מחשבים של המשימה השנייה. בדוגמה להלן, תיווצר הרשת הבאה ששמה PrimPaths:



השתמשו באותם מבנים כמו במשימה הראשונה.

## משימה ד' –מציאת מסלול בין זוג קדקודים והדפסתו

במשימה זו, הפונקציה תקבל את הרשת PrimPaths ומספרים של שני מחשבים, first ו-last, ותמצא את המסלול ביניהם על פי האלגוריתם המתואר בהמשך.

## מבני נתונים

1. מערך Color: מערך המכיל לכל מחשב v את הצבע שלו: לבן, אפור או שחור.
2. מערך Parent: מערך המכיל לכל מחשב v את ה- parent שלו במסלול מ- first ל-last.

## רעיון האלגוריתם

כל מחשב יהיה צבוע באחד מהצבעים הבאים:

- "לבן" – טרם ביקרנו בו
- "אפור" – ביקרנו בו וטרם בדקנו את כל המחשבים שמחוברים אליו
- "שחור" – ביקרנו בו וסיימנו לבדוק את כל המחשבים שמחוברים אליו

בתחילת האלגוריתם כל המחשבים יהיו לבנים ואז תופעל הפונקציה הרקורסיבית FindPath עם הפרמטר first. בסיום נשתמש במערך Parent על מנת להדפיס את המסלול מ- first ל-last.

## פונקציות עזר – פסאודו קוד

```
FindPath(Vertex u)

Color[u] ← gray          // begin processing of u

for each v ∈PrimPath[u] do

        if Color[v] = white then
                Parent[v] ← u
                if (v = last)
                                        // we have reached the destination
                        return;
                else
                        FindPath(v)     // recursive call
        End-if

End-foreach
```

```
Color[u] ← black                    // end processing of u
```

**<u>printPath(v)</u>**

```
if (v = first)
        print v
else
        printPath(Parent[v])      // print path up to v
        print v
end-if
```

לאחר הקריאה ל- FindPath(first) הפונקציה תקרא ל- PrintPath עם הפרמטר last.

**השתמשו בקוד בעמוד הבא:**

```c
typedef struct  {
    int neighbor;
    int cost;
} Edge;

typedef struct edge_node {
    Edge e;
    struct edge_node* next;
} EdgeNode;

typedef struct {
    EdgeNode* head;
    EdgeNode* tail;
} EdgeList;

EdgeList *build_net(int n);
int* build_prim_tree(EdgeList net[ ], int n);
EdgeList *build_paths(int* tree, int n);
void find_and_print_path(EdgeList primpaths[ ], int n, int first, int last);

int main()
{
    int n;
    EdgeList *Net;
    int* Prim;
    EdgeList *PrimPath;
    int first, last;

    scanf("%d", &n);
    Net = build_net(n);
    Prim = build_prim_tree(Net, n);
    PrimPath = build_paths(Prim, n);

    scanf("%d%d", &first, &last);
    find_and_print_path(PrimPath, n, first, last);

    return 0;
}
```