מבוא למדעי המחשב
בחינת מועד א', סמסטר ב' תשס"ה, 24.6.2005

מרצה: גב' יעל כהן-סיגל.
מתרגל: מר עזרא דאיה.

משך המבחן: שעתיים וחצי.
חומר עזר: מותר כל חומר עזר, מלבד מחשב.
הנחיות:
1. יש לענות על כל השאלות.
2. קראו היטב כל שאלה. ודאו כי אתם מבינים את השאלה לפני שתתחילו לענות עליה.
3. כתבו בכתב יד ברור וקריא. תשובות לא קריאות לא תיבדקנה.
4. הערות לתשובותיכם ניתן לכתוב בעברית, גם בגוף פונקציות C.
5. ניתן ונדרש להגדיר פונקציות עזר לפי הצורך.
6. ניתן להשתמש בכל פונקציה המופיעה במצגות ההרצאות והתרגולים ע"י הצהרה עליה בלבד (אין צורך להגדירה). כמו כן, ניתן להשתמש בפונקציות מתוך הספריות stdio.h, stdlib.h ו- string.h. לא ניתן להשתמש בפונקציות אחרות בלא להגדיר אותן במפורש.

שאלה 1 (25 נק')

בהינתן קבוצת מספרים, עקבה היא תת-קבוצה של מספרים עוקבים.
למשל עבור הקבוצה {4,7,10,34,5,33,6,35,1} תת הקבוצה {34,33,35} היא עקבה ותת הקבוצה {4,7,5,6} היא העקבה הגדולה ביותר. תת הקבוצה {5,6,10} אינה עקבה כיון שאיננה מורכבת מקבוצה של מספרים עוקבים. הקבוצה {3,4,5} אינה עקבה כיון שאיננה תת-קבוצה.
כתבו פונקציה המקבלת מערך של מספרים שלמים ללא חזרות (כלומר, כל מספר מופיע לכל היותר פעם אחת) ואת גודלו ומדפיסה את העקבה המקסימלית שלו.

דרישות סיבוכיות:
סיבוכיות זמן: O(n log n) סיבוכיות מקום: O(n)
פתרונות בסיבוכיות גבוהה יותר לא יתקבלו.

שאלה 2 (25 נק')

בהינתן מספר שלם חיובי N המקיים N > 0 נרצה למקם על לוח שחמט בגודל N*N, N מלכות שחורות ו- N מלכות לבנות כך שיתקיים:
1. אף מלכה לא תאיים על מלכה אחרת בעלת צבע זהה.
2. בכל משבצת תמוקם לכל היותר מלכה אחת.
תזכורת: מלכה מאיימת על כל כלי משחק הנמצא בקו ישר כלשהוא מהמשבצת בו היא ממוקמת (קו אנכי, אופקי או אלכסוני).
עליכם לכתוב תכנית (הנעזרת ב- backtracking) המדפיסה את כל הפתרונות האפשריים (אם קיימים כאלו) עבור N כלשהוא.

ניתן להניח כי N מוגדר כקבוע בתכנית (ע"י define).

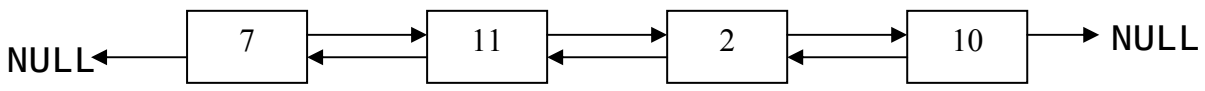
שאלה 3 (50 נק')

נתונה רשימה מקושרת דו-כיוונית אשר כל תא בה מוגדר באופן הבא:

```
struct cell{
    const int num;
    struct cell *prev;
    struct cell *next;
};
```

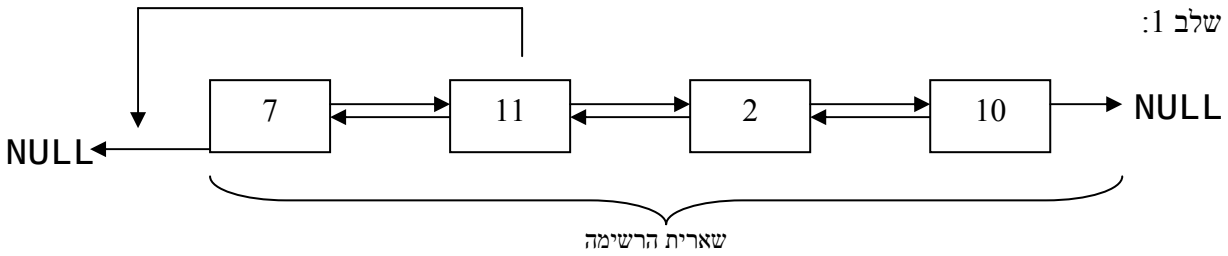
אלגוריתם המיון Max Sort מקבל כקלט רשימת מספרים וממיין אותה באופן הבא: בכל שלב נחפש את האיבר הגדול ביותר בשארית הרשימה ונעביר אותו לראש הרשימה. בסוף התהליך נקבל רשימה ממוינת.

למשל: בהינתן הרשימה

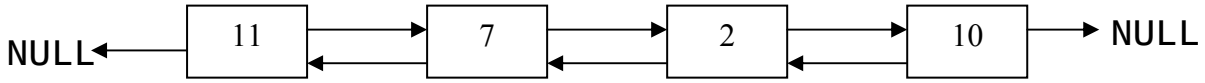


יתבצע המיון כדלהלן:

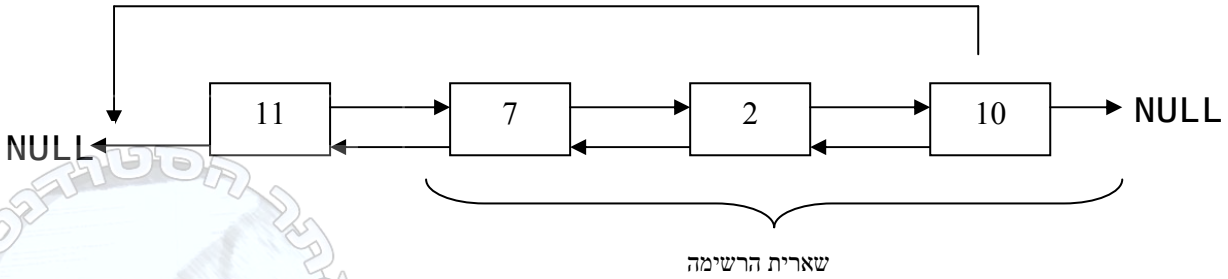
שלב 1:



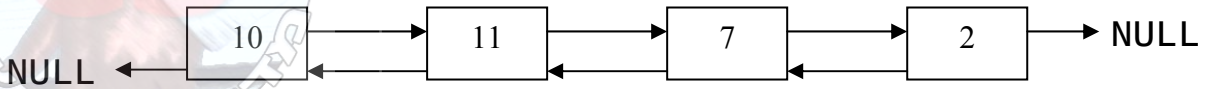
לאחר השלב ה- 1 נקבל את הרשימה



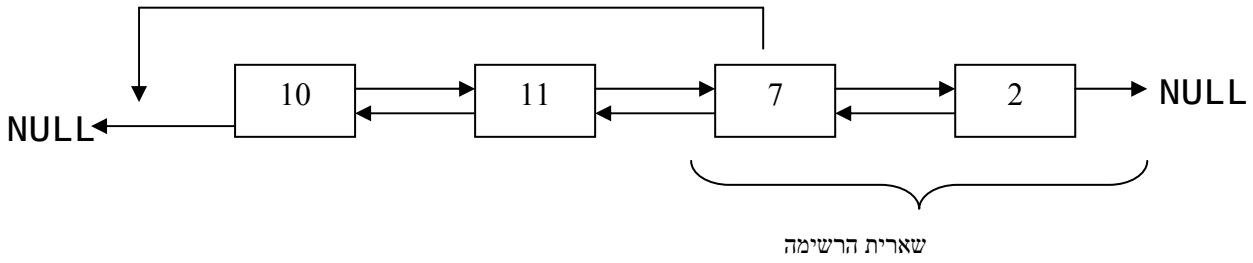
שלב 2:



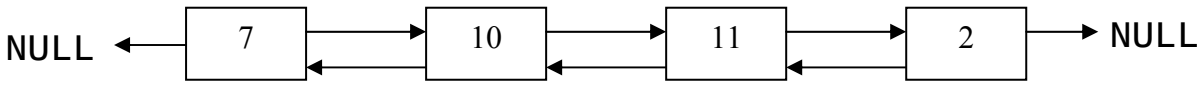
לאחר השלב ה- 2 נקבל את הרשימה



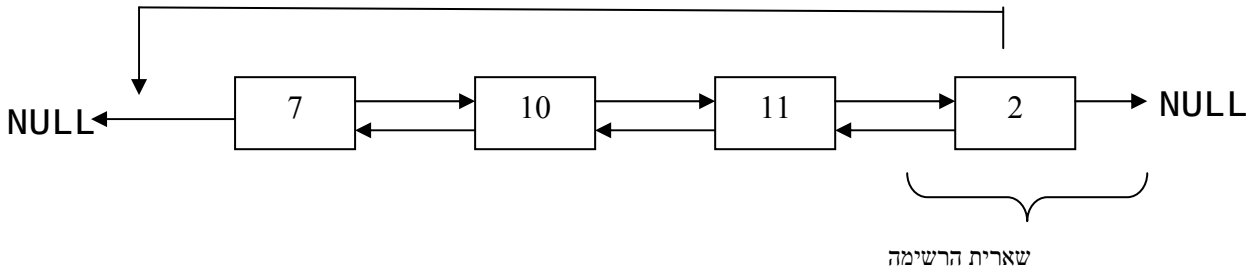
שלב 3:



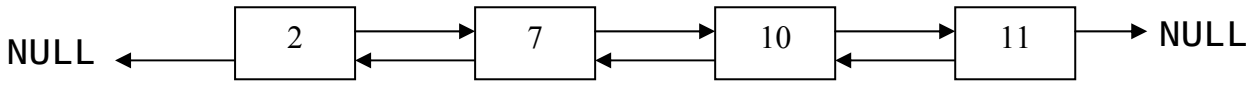
לאחר השלב ה- 3 נקבל את הרשימה



שלב 4:



לאחר השלב ה- 4 נקבל את הרשימה הממוינת



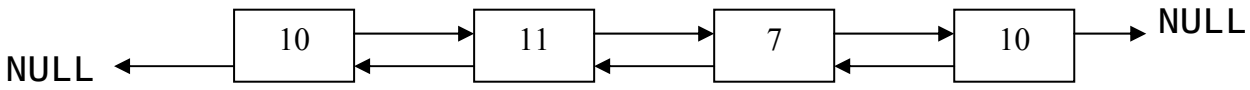
א. הגדירו פונקציה המקבלת מצביע לתחילת רשימה מקושרת דו-כיוונית כנ"ל, ממיינת את איבריה בעזרת אלגוריתם המיון Max Sort ומחזירה מצביע לתחילת הרשימה ממוינת.
שימו לב: השדה num ברשומה cell מוגדר כ- const ולכן לא ניתן לשנותו!.

סיבוכיות נדרשת:
סיבוכיות זמן: $O(n^2)$ (n הוא מספר האיברים ברשימה)
סיבוכיות מקום: O(1)

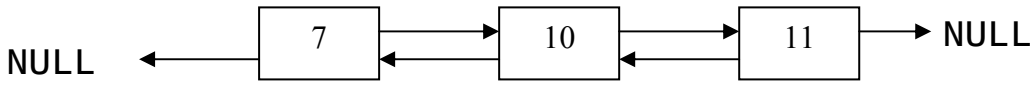
ב. הגדירו פונקציה המקבלת מצביע לתחילת רשימה מקושרת דו-כיוונית כנ"ל, ומוחקת מן הרשימה מופעים חוזרים של תאים בעלי ערך זהה. הפונקציה מחזירה מצביע לתחילת הרשימה החדשה. סדר האיברים ברשימה המוחזרת אינו משנה.

למשל:

עבור הרשימה



תוחזר הרשימה



הערות: ניתן להשתמש בפונקציה של סעיף א' גם אם לא פתרתם אותו.
ניתן להניח כי כל אברי הרשימה הוקצו בעזרת הקצאה דינאמית.

סיבוכיות נדרשת:
סיבוכיות זמן: $O(n^2)$ (n הוא מספר האיברים ברשימה)
סיבוכיות מקום: O(1)

<u>בהצלחה!</u>

Number1:

```
#include <stdio.h>

void sort (int array[], int size); /* mergesort - O(n log n) */

void consecutive(int arr[], int n) {
        int i,j;
        int *tmp, max=0;
        sort(arr,n);
        if (NULL==(tmp=(int*)malloc(n*sizeof(int)))) {
                printf("Memory allocation error\n");
                exit(1);
        }
        for(i=0 ; i<n ; i++)
                tmp[i]=0;
        for(i=0,j=1 ; i<n ;i++){
                if(arr[i+1] == (arr[i] +1))
                        j++;
                else {
                        tmp[i] = j;
                        j=1;
                }
        }
        for (i=0; i<n; i++) {
                max=(tmp[i]>tmp[max])? i:max;
        }
        for (i=0; i<tmp[max]; i++) {
                printf("%d ",arr[(max - tmp[max]) + 1 + i]);
    }
}
```

Number 2:

```
#include <stdio.h>
#define N 5

enum {EMPTY,WHITE,BLACK};
enum {FALSE, TRUE};

void solve (char board[N][N], int row);
void print_board (char board[N][N]);
int threatens (char board[N][N], int row, int column, int color);

int main()
{
        char board[N][N]={0};
        solve(board,0);
        return 0;
}
```

```
void solve (char board[N][N], int row)
{
	int wcol, bcol;
	if (row == N) {
		print_board(board);
	}
	else {
		for (wcol=0; wcol < N; wcol++) {
			if (!threatens(board, row, wcol, WHITE)) {
				board[row][wcol] = WHITE;
				for (bcol=0; bcol < N; bcol++) {
					if (board[row][bcol]==EMPTY &&
!threatens(board, row, bcol, BLACK)) {
						board[row][bcol] = BLACK;
						solve(board, row+1);
						board[row][bcol] = EMPTY;
					}
				}
				board[row][wcol] = EMPTY;
			}
		}
	}
	return;
}

void print_board(char board[N][N])
{
	int i,j;
	static int counter=0;
	printf("Solution number %d:\n", ++counter);
	for (i=0; i<N; i++) {
		for (j=0; j<N; j++) {
			if(board[i][j]==EMPTY){
				printf(" . ");
			}
			else if(board[i][j]==WHITE){
				printf(" W ");
			}
			else{
				printf(" B ");
			}
		}
		printf("\n");
	}
	printf("\n");
}

int threatens(char board[N][N], int row, int column, int color)
{
```

```
        int r;
        /* check if there's a queen on this column */
        for (r=0; r<row; r++) {
                if (board[r][column] == color) {
                return TRUE;
                }
        }
        /* check if there's a queen on the diagonals */
        /* first, check the upper-left diagonal: [row-r,column-r] */
        for (r=1; row-r>=0 && column-r>=0; r++) {
                if (board[row-r][column-r] == color) {
                return TRUE;
                }
        }
        /* then, check the upper-right diagonal: [row-r,column+r] */
        for (r=1; row-r>=0 && column+r<N; r++) {
                if (board[row-r][column+r] == color) {
                return TRUE;
                }
        }
        return FALSE;
}
```

Number3:

```
#include <stdio.h>

#define SIZE 100

struct cell *MaxSort  ( struct cell *list);
struct cell *RemoveDuplicate ( struct cell *list );
void RemoveItem ( struct cell *item );

struct cell{
        const int num;
        struct cell *next;
        struct cell *prev;
};


struct cell  *MaxSort  ( struct cell *list )
{
        struct cell *tmp1, *tmp2, *max, *prev, *next;
        prev=tmp1=list;
        while(tmp1!=NULL){
                max=tmp1;
                for (tmp2=tmp1; tmp2!=NULL; tmp2=tmp2->next){
                        if (tmp2->num > max->num){
                                max=tmp2;
                        }
```

```
                }
                if (tmp1==max){
                        tmp1=tmp1->next;
                }
                prev=max->prev;
                next=max->next;
                if (prev!=NULL){ /* max is not the first element */
                        prev->next=next;
                        list->prev=max;
                        max->next=list;
                        max->prev=NULL;
                        if (next!=NULL){ /* max is not the last element */
                                next->prev=prev;
                        }
                }
                list=max;
        }
        return list;
}


struct cell *RemoveDuplicate ( struct cell *list )
{
        struct cell *tmp;
        list=MaxSort(list);
        for (tmp=list; tmp!=NULL; tmp=tmp->next){
                while ( (tmp->next!=NULL) && (tmp->num == tmp->next->num) )
{
                        RemoveItem (tmp->next);
                }
        }
        return list;
}

void RemoveItem ( struct cell *item )
{
        item->prev->next=item->next;
        if (item->next!=NULL){
                item->next->prev=item->prev;
        }
        return;
}
```