אלגו - קודים למבחן

מבוסס על השיעורים של פרופסור לויט, והסיכומים של אליזבת

לתיקונים/הערות/הארות - נעם דומוביץ - $ndomovich@gmail.com$

תוכן עניינים

complexity = O(n*(n+logn))=O(n^2)

LIS(int[] a) {

```
    // init
    int n = a.length();
    mat = new int[n][n];
    int lis = 1, val,idx;
    mat[0][0] = a[0];
    for(int i = 1 ; i< n ; ++i) {
        val = a[i];
        // find greatness relative to diagonal
        idx = binary_serach_between(val,mat,lis);
        mat[idx][idx] = val;
        // copy the line above
        for(int k = 0 ; k < j ; k++) {
            mat[idx][k]=mat[idx-1][k];
        }
        // update lis
        if(idx == lis) {
            lis++
        }
        // lis = the len, a[lis-1] = the sequence
        return lis, a[lis-1];
}
```

i-f-c-u

```
// str1.length >= str2.length , // String could contain A-Z,a-z
// Cluster is Class with vector for saving idx per letter
lcs_with_lis(String str1, String str2) {
  Cluster[] letters_dic_idx = new Cluster[122 - 65 + 1]; // = 57 := 'z' - 'A'
    //( In Java needs to initialize the array )
    int[] to_lis = new int[str1.length() * str2.length()];
    int right = 0;
    // enter idx of shortest string
    for (int l = 0 ; l < str2.length() ; ++l) {
        letters_dic_idx[str2.charAt(l) - 'A'].idx.add(l);
    }
    // create array of idx - for LIS - by longest string
    for ( int l = 0 ;  l < str1.length() ; ++l) {
        int letter = str1.charAt(l) - 'A'; // give as idx of letter in dic
        // get all idx of letter in str2
        Vector<Integer> letter_idx_vec = letters_dic_idx[letter].idx;
        for( int idx = letter_idx_vec.size() - 1 ; idx >= 0 ; --idx) {
```

```
                to_lis[right] = letter_idx_vec.get(idx);
                right++;
            }
        }
        b = new int[right]; // remove empty cells
        b = Arrays.copyOfRange(to_lis, 0, right);
        // use lis
        int[] lis_arr = LIS(b);
        String ans = '';
        for( int n = 0 ; n < lis ; n++) {
            int i = lis_arr[n];
            ans += str2.charAt(i);
        }
        return ans;
    }
```

הוכחת נכונות: <u>**טענה:**</u> **האלגוריתם מחזיר את המחרוזת המשותפת הגדולה ביותר**

- $LIS(b) \leq LCS(str1, str2)$: מנכונות LCS תוחזר המחרוזת המשותפת <u>הארוכה</u> ביותר, ומ LIS תוחזר מחרוזת כלשהי הבנויה מאותיות משותפת של $str1, str2$
- $LCS(str1, str2) \leq LIS(b)$, כי b הוא מערך של אינדקסים לאותיות משותפות, מ $str1$ ומ $str2$, לכן הוא לפחות כמו המחרוזת המשותפת הארוכה ביותר

עולה יורדת

```
lis[i] ←1 , lds[i] ←1 ;
for i=1 to n{
    for j= 0  to i − 1{
        if(arr[i] > arr[j] &&  lis[i] ≤ lis[j] + 1) {
            lis[i]=lis[j]+1;
        }
    }
}
for i=1 to n{
    for j= 0  to i − 1{
        if(arr[i] < arr[j] &&  lds[i] ≤ lds[j] + 1) {
            lds[i]=lds[j]+1;
        }
    }
}
max(lis[i] = lds[i])
```

```
buildMatrix(String a, String b):
    // init - row size as strings + 1 row/col of 0
    int row = a.length() + 1;
    int col = b.length() + 1;
    int[][] mat = new int[row][col];
    for(int i = 1 ; i < row ; ++i) {
        for(int j = 1 ; j <col ; ++j) {
        // insert [ (i − 1, j) ; (i, j − 1) (i, j) ]
            if(a.charAt(i-1) == b.charAt(j-1) { // d_1
                mat[i][j] = mat[i-1][j-1] + 1;
            } else { // l_max
                mat[i][j] = math.max(mat[i-1][j], mat[i][j-1];
            }
        }
    }
    return mat;
}
// i-i-d_1-l_max
LCS (String a , String b):
    // init_a - max len by matrix
    mat[][] = buildMatrix(a,b);
    int i = a.length , j = b.length;
    // init_b data to backward matrix loop
    String max_sub = “”;
    int max_sub_len = mat[i][j];
    int count = max_sub_len - 1;
    // get max sub by reverse
    while(count >= 0 ) { // in our max sub
        // check [ (i − 1, j) ; (i, j − 1) (i, j) ]
        if(a.charAt(i-1) == b.charAt(j-1) { // i-1 == j-1 - take
            max_sub = a.charAt(i-1) + max_sub;
                --count; --i; --j;
        }else if(mat[i][j] == mat[i][j-1]){ // j == j-1 - permote
            --j;
        }else{ // i == i-1
            --i;
            }
        }
    }
    return max_sub_len, max_sub
    //i_a-i_b-r-c-t-p
```

0.3 מציאת 2 מקסימלים

רקורסיה

```
complexity = O(2^n)
MAX_MAX_REC(int[] a, Vector b, int l , int r) {
    // stop conodition
    if(l == r )
        return a[l];
    else
        // each time chack half array
        int mid = (int)l + (l-r)/2;
        int max1 = MAX_MAX_REC(a,b,l,mid);
        int max2 = MAX_MAX_REC(a,b,mid+1,r);
        if(max1 > max2) {
            b.push(max2); // remeber the loser , and check later for secondary
            return max1
        else
            b.push(max1);
            return max2;
}
    // In the end:
        1. max1 = is what we get back from MAX_MAX_REC
        2. max2 = check what is greatest number in b, by simple loop
```

complexity = O(2^n)
comperations: = 1.5n

MAX_MAX_IND(int[]) {

```
    // init
    int max1, max2, first, second;
    if( a[0] > a[1]) { max1 = a[0]; max2 = a[1] }
    else { max1 = a[1]; max2 = a]0]; }
    //
    for ( int i = 2 ; i < a.length() ; i +=2) {
        first = a[i];
        second = a[i+1];
        if ( first > second ) { // 1
            if ( first > max1) { // 2
                if( second > max1) { // 3
                    max2 = second;
                else {
                    max2 = max1;
                }
                max1 = first;
            } else {
                if ( first > max2) { // 3
                    max2 = first;
                }
        } else {
            if(first > max1 ) {
                if( second > max1 ) {  // 2
                    max2 = first;
                } else {
                    max2 = max1;
                }
                max1= second;
            } else {
                if( second > max2 ) { // 3
                    max2 = second;
            }
        }
    } return max1, max2;
}
```

0.4 משחק המספרים

אסטרטגיות:

- <u>חמדן נאיבי</u>

```
// _left = 0, _right = a.length
choose_strategy(int[] a , _left, _right) {
    int sum_even = sum_even(a, _left, _right);
    int sum_odd = sum_odd(_left, _right);
    if (sum_even > sum_odd) {
        return EVEN;
    else {
        return RIGHT;
    }
}
```

- <u>חמדן אדפטיבי:</u>
  - בכל תור נקרא ל $Choose\_strategy(a, \_left, \_right)$ העדכניים
- <u>חיפוש שלם - רקורסיבי:</u>

```
// at first turn = 1 , next calls {-1,1} by turns
calc_gains_rec(int[] a, int l, int r, int turn) {
    if (l == r) {
        return (turn) * a.in[l];
    } else {
        int gainL = (turn) * a.[l];
        gainL += calc_gains_rec(a,l + 1, r, -turn);
        int gainR = (turn) * a.[r];
        gainR += calc_gains_rec(a, l, r - 1, -turn);
        if (turn > 0) { // my turn -> take Max
            if (gainL > gainR) {
                side = LEFT;
                return gainL , side;
            } else {
                side = RIGHT;
                return gainR , side;
            }
        } else { // user turn -> take Min
            if (gainL < gainR) {
                side = LEFT;
                return gainL,side;
            } else {
```

```
                side = RIGHT;
                return gainR, side;
            }
        }
    }
}
```

- חיפוש שלם - דינאמי
  - אתחול:

```
init_dynamic(int a[]) {
    int n = a.length();
    int[][] gainMat = new int[n][n];
    // init - diagonal
    for (int i = 0 ; i < n ; ++i) {
        gainMat[i][i] = a.[i];
    }
    // build mat - only upper right traingle
    for( int i = n - 1 ; i >= 0 ; --i) {
        for (int j = i + 1 ; j < n ; ++j) {
            gainMat[i][j] = Math.max(a[i] - gainMat[i+1][j], a[j] - gainMat[i][j-1]);
        }
    }
}
```

  - מהלך בודד:

```
dynamic_move(int[][] gainMat , int _left, int, _right) {
    int l = gainMat[_left][_left] - gainMat[_left + 1][_right];
    int r = gainMat[_right][_right] - gainMat[_left][_right - 1];
    if (l > r ) {
        return LEFT;
    } else {
        return RIGHT;
    }
}
```

0.5 כדור הזכוכית

- בעיה זו היא בעצם משל לבעיית חיפוש במערך ממוין עם הגבלה על מספר הבדיקות שניתן לבצע, כאשר מן הקצה האחד מותרת בדיקה אחת ואז אין ברירה אלא להשתמש בחיפוש שלם, ומן הקצה השני מספר הבדיקות הוא ככל שנרצה, ומעבר לחיפוש בינארי לא נמצא פתרון יעיל יותר. כעת נשאלת השאלה, מה קורה בטווח?
- בהנתן שתי בדיקות = כדורים : פתרון ראשון - חלוקה ל $\sqrt{levels}$
- בהנתן שתי בדיקות = כדורים : פתרון שני - מספרים משולשים:

```
triangle_number_steps(int levels) {
    int jump = 1;
    int i;
    // get max size of jumps
    count++;
    while (levels > (jump * (jump + 1) / 2)) {
        ++jump;
    }
    for (i = jump; i <= levels; i += jump) { // slice by triangle number jumps
        if (i >= BROKEN_LEVEL) {
            for (i = i - jump; i < i + jump && i < levels; ++i) { // check this slice one by one
                if (i == BROKEN_LEVEL) {
                    return i;
                }
            }
        }
        --jump;
    }
}
```

```
k_checks() {
    int numChecks = 0 , min , max, broked,unbroke , count = 0;
    int[][] mat = new int[balls+ 1][levels + 1];
    //init
    for(int j = 0 ; j <= levels ; ++j) {
        mat[0][j] = 0; //no ball - no checks
        mat[1][j] = j; // 1 ball - only whole search is suitable
    }
    //build mat
    for (int i = 2; i < balls + 1; ++i) {
        mat[i][0] = 0; mat[i][1] = 1;
        if (levels >= 2) { mat[i][2] = 2; }
        //
        for (int j = 2; j <= levels ; ++j) {
            min = levels + 1;
            //
            for (int p = 1; p <= j - 1; ++p) {
                // the ball broke on level i, so check the levels below with one less ball
                broked = mat[i - 1][p - 1];
                // the ball wasn't broken on level i, so check the level above with same ball number
                unbroke = mat[i][j - p];
                // take the worst case
                max = Math.max(broked, unbroke);
                // take the best step
                min = Math.min(max, min) + 1;
                mat[i][j] = min;
            }
        }
        print( 'num of checks for k balls and n levels is: ' + mat[balls][levels]);
}
```

0.6 בעית המזכירה:

- מנהל צריך לקבל n תלמידים ־ נסמן ב t_i
- המזכירה צריכה לסדר את סדר הפגישות עם המנהל, למנהל ולמזכירה (בניגוד לתלמידים...) , אין חשיבות לסדר.
- התלמידים = העובדים מרויחים אותו דבר, וכל אחד רוצה להכנס ראשון.
- נאמר שהזמן שתלמיד מחכה הוא הפסד שלנו, מטרתנו לצמצם את ההפסד הזה = זמן ההמתנה הכולל

<u>פתרון ראשון ־ חיפוש שלם:</u>

כלומר

$$\min\left(\frac{t_1+(t_1+t_2)+....+(t_1+....+t_n}{n}\right)=\min\left(\frac{nt_1+(n-1)t_2+...+t_n}{n}\right)$$

סיבוכיות יצירת כל האפשרויות (פרמטוציות) = $O(n!)$ + סיבוכיות החישוב לכל פרמטוציה: $O(n)$

בסה"כ: $O\left(n!\cdot n\right)$

<u>פתרון שני ־ חיפוש חמדן ־ מיון המערך:</u>

מהגדרה החמדן בוחר בכל שלב את המשתלם ביותר ־ נגדיר שבחירה זו היא הפגישה שתגמר הכי מוקדם. אם נבצע זאת על המערך שוב ושב נקבל מערך ממוין.

<u>הוכחה:</u>

נתבונן במקרה כללי:

$$\sum\nolimits_1=\cancel{t_1+(t_1+t_2)+....+}(\cancel{t_1+...}+\boldsymbol{t_{i-1}})+\cancel{(t_1+...+t_{i-1}+t_{i-1}+t_i)}+....+\cancel{(t_1+....+t_n)}$$

$$\sum\nolimits_2=\cancel{t_1+(t_1+t_2)+....+}(\cancel{t_1+...+\boldsymbol{t_{i-2}}}+\boldsymbol{t_i})+\cancel{(t_1+...+t_{i-2}+t_i+t_{i-1})}+....+\cancel{(t_1+....+t_n}$$

כלומר אנו מחפשים$\sum_1 < \sum_2$ וזה קורה אס"ם $t_i < t_{i-1}$

מכאן שהאלגוריתם פשוט ממיין את המערך, כמו ב$bubble-sort$ ־ וזה $O(n^2)$

מכיון ואנו יודעים למיין בסיבוכיות טובה יותר, נקבל ש: $O(n\cdot\log n)$.

0.7 המשך בעיית המזכירה ־ בעיית הקומפיילר

- נניח כי יש $l_1,l_2,....,l_n$ תוכניות, ו$p_1,p_2,.....$ הסתברות שימוש בתכוניות
- פונקציית המטרה:

$$\min\left(p_1\cdot l_1+p_2(l_1+l_2)+.....+p_2(l_1+....+l_n)\right)$$

<u>פתרון:</u>

- נגדיר פרמטר חדש $h_i=\frac{p}{l_i}$, ונמיין על פיו, מדוע?

<u>אבחנות:</u>

- במקרה ו $p = p_1 = p_2 = ... = p_n$ אז:

$$\text{בעיית המזכירה} = p_1\left(\cdot l_1 + (l_1 + l_2) + ... + (l_1 + ... + l_n)\right)$$

$$\text{ולכן נמיין } l_1 \le l_2 \le ..... \le l_n$$

- במקרה ו $l_1 = l_2 = ... = l_n$ אז:

$$\text{בעיית המזכירה (בסדר הפוך)} = l\left(p_1 + 2p_2 + ... + np_n\right)$$

$$\text{ולכן נמיין (בסדר יורד) } \boldsymbol{p_n \le p_{n-1} \le .... \le p_1}$$

- על פי שני מקרי קצה אלה, נרצה לצור פרמטר שלישי:

$$\sum\nolimits_1 = \cancel{p_1 l_1 + p_2 (l_1 + l_2) + ...} + \cancel{p_i (l_1 + ... + l_{i-1}} + l_i) + \boldsymbol{p_{i+1}} (\cancel{l_1 + ...} + \boldsymbol{l_i} + \cancel{l_{i+1}}) + ...$$

$$\sum\nolimits_2 = \cancel{p_1 l_1 + p_2 (l_1 + l_2) + ...} + \cancel{p_{i+1} (l_1 + ... + l_{i-1}} + L_i) + \boldsymbol{p_i} (\cancel{l_1 + ...} + \boldsymbol{l_{i+1}} + \cancel{l_{i+1}}) + ...$$

$$\sum\nolimits_1 > \sum\nolimits_2 \iff p_{i+1} \cdot l_i > p_i \cdot l_{i+1} \iff \frac{p_{i+1}}{l_{i+1}} > \frac{p_i}{l_i}$$

$$h_{i+1} = \tfrac{p_{i+1}}{l_{i+1}} > \tfrac{p_i}{l_i} = h_i$$

- ולכן אם נמיין $\left\{\frac{p_i}{l_i}\right\}$ בסדר יורד, נקבל את זמן ההמתנה המינמלי
- סיבוכיות: $O(n) + O(n \cdot \log n) = O(n \cdot \log n)$= חישוב הסדרה + מיון

0.8 בעיית הסופגניה

יש לנו מחבת שניתן לשים בו 2 סופגניות. הכנת סופגניה דורשת 1 דקה על על צד (סהכ 2)

- עבור $n = 1$ (מספר הסופגניות) הזמן $T = 2$,
- עבור $n = 2$, $T = 2$. עבור $n = 4$, $\begin{array}{l} A1, B1 \to 1 \\ A2, B2 \to 1 \\ C1, D1 \to 1 \\ C2, D2 \to 1 \end{array}$ $T = 4$
- עבור $n = 3$, A, B, C רצף אפשרי $\begin{array}{l} A1, B1 \to 1 \\ A2, B2 \to 1 \\ C1, C2 \to 2 \end{array}$ סה"כ $T = 3$. עבור $n = 5$ $T = 5$ = $3 + 2$
- מקרה כללי: n סופגניות $(P = 2)$: n דקות
  - $n = 2k$ ־ זוגי , $2k = n$
  - $n = 2k + 1$ אי־זוגי אז $n = (2k - 2) + 3 = 2k + 1$
- מקרה כללי: n סופגניות, P מקום על המחבת, מה סך הזמן הנדרש?
  - אם $n \le p$: $T = 2$
  - אם $p < n \le 2p$: $T = 2p$
- אם $n > 2p$ אז:
  - אם $n = pk$ אז pk דקות
  - אם $n = pk + r$ אז $n = (pk - p) + p + r = pk + r$

0.9 בעיית הפיצה

הבעיה:

- פיצה מחולקת למשולשים שווים, שני אנשים אוכלים במהירויות שונות.
- אבי אוכל פי 2 מהר יותר מבני, אסור להגיע בו זמנית למשולש האחרון.
- אבי קובע את מספר המשולשים, השאלה מהי החלוקה שבה מספר המשולשים שאבי יקבל הכי הרבה מהפיצה?

פתרון:

- עבור $N=2$ - לכל אחד יש $\frac{1}{2}$, עבור $N=3$ לאבי $\frac{2}{3}$ בני $\frac{1}{3}$, עבור $N=4$ לאבי $\frac{2}{4}$ בני $\frac{1}{4}$ נותר משולש - לא תקין
- עבור $N=5$ לאבי $\frac{3}{5}$ בני $\frac{2}{5}$,
  - נשים לב ש: $\frac{3}{5}<\frac{2}{3}$
- נכליל - טענה: החלק של אבי הגדול ביותר בחלוקה ל $3k$ חלקים
- ואז אבי $2k$ ובני k
- חלוקה ל $3k+1$ חלקים - לא חוקי כי $3k$ הראנו מה יקרה, ו1 ישאר
- חלוקה ל $3k+2$ חלקים. אבי $\frac{2k+1}{3k}$, בני
- מה החלוקה הטובה ביותר?

$$\begin{gathered}\frac{2k+1}{3k+2} < \frac{2}{3}\\ 3\left(2k+1\right)<2\left(3k+2\right)\\ \vdots\\ 3<4\end{gathered}$$

- נכליל את המהירות ל $X\geq 3$
- כלומר $V_A=X\cdot V_B$
- אם החלוקה היא $X+1$
  - אז אבי עם X , ובני 1
  - אם $X+2$ חלקים - לא תקין כי , ישאר 1
- $N=X+r+1$ חלקים $2\leq r\leq X$ אז:

$$\frac{X}{X+1} > \frac{X+r-1}{X+r+1}$$

$$X\left(X+r+1\right)>\left(X+1\right)\left(X+r-1\right)$$

$$X>r-1$$

$$X+1>r\ \checkmark$$

0.10 בעיית המטריצה

- מציאת ריבוע:

```
findBigSquareInMat(int[][] mat, int n , int m) {
    int[][] dynamic = new int[n][m];
    int maxS = 0, row = 0 , col = 0 , square = 0;
    // init
    for (int i = 0; i < n; ++i) { dynamic[i][0] = mat[i][0]; }
    for (int j = 0; j < m; ++j) { dynamic[0][j] = mat[0][j]; }
    // calc
    for ( int i = 1 ; i < n ; ++i ) {
        for ( int j = 1 ; j < m ; ++j) {
            if ( mat[i][j] == 0) {
                dynamic[i][j] = 0;
            } else {
                square = Math.min(dynamic[i-1][j-1], Math.min(dynamic[i-1][j], dynamic[1][j-
                1])) + 1;
                dynamic[i][j] = square;
                if ( square > max) {
                    maxS = square; row = i; col = j;
                }
            }
        }
    }
    return maxS, (row - maxS + 1), (col - maxS + 1);
}
```

- מציאת מלבן:

```
int Max__Histogram__Area(int[] hist) {
    int n = hist.length;
    Stack<Integer> s = new Stack<>();
    int maxArea = 0 , top = 0 , area__with__top = 0, i = 0;
    while (i < len) {
        if (s.empty() || hist[s.peek()] <= hist[i]) { // increasing steps
            s.push(i);
            ++i;
        } else { // found decreasing
            top = s.pop();
            area__with__top = hist[top] * (s.empty() ? i : i - s.peek() - 1); // usually ver-
            tical area
            if (maxArea < area__with__top) {
```

```
                maxArea = area_with_top;
            }
        }
    }
    while (!s.empty()) {
        top = s.pop();
        area_with_top = hist[top] * (s.empty() ? i : i - s.peek() - 1); // usually horizontal area
        if (maxArea < area_with_top) {
            maxArea = area_with_top;
        }
    }
    return maxArea;
}
```

findMaxRectangle() {

```
    int rows = mat.length, cols = mat[0].length;
    int maxArea = 0, area = 0;
    if (rows == 0 || cols == 0 ) { return 0; }
    int[] help = new int[cols];
    //first row
    for(int j = 0; j < cols ; j++) { help[j] = 0; }
    //
    for (int i = 0 ; i < rows; ++i) {  // get histograms by rows
        for( int j = 0 ; j < cols ; ++j) {  // each row depende row above
            if(mat[i][j] == 1 ) {
                help[j] = help[j] + 1;
            } else {
                help[j] = 0 ;
            }
        }
        area = Max_Histogram_Area(help);
        if (area > maxArea) {
            maxArea = area;
        }
    }
    System.out.println("maxArea: " +  maxArea);
    return maxArea;
}
```

```
// WASNT == OFF == false
// WAS == ON = true
// VISIT_TWICE == 0 ; WASNT_T == 2
boolean check_visits(int n) {
    int count = 0;
    int[] visits = new int[n]; Arrays.fill(visits, WASNT_T);
    boolean all = false;
    while (!all) {
        int p = get_random_num();
        // if (vists[p] == WASNT && lamp == ON) { // p his first time and lamp is on
        if (visits[p] > VISIT_TWICE && lamp == ON) { // p his first or second time and lamp is on
            lamp = OFF; // turn off
            --visits[p]; // count this one // visits[p] = WAS; // only once
        }
        if (p == 0 && lamp == OFF) { // the representative chosen
            lamp = ON; // turn on
            count++; // count it
            if (count == 2*n) { // if (count == n) {
                all = true;
            }
        }
    }
    return all; // == true
}
```

0.12 בעיית החזקה

```
complexity = O(log n)
pow(int x, int n) {
    int ans = 1;
    while (n > 0) {
        if (n % 2 == 1) {
            ans *= x;
        }
        x = x * x;
        n = n / 2;
    }
    return ans;
}
```

0.13 מספרי פיבונאצ'י

- נגדיר $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ ויתקיים: $A^n = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix}$
- נשתמש באלגוריתם היעיל לחזקה, עם כפל מטריצות.
- פונקציית כפל מטריצות נממש על פי הגדרה: אם $[A]_{n\times m}, [B]_{m\times p}$ אז: $(AB)_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj}$
- ניתן להוכיח טענות לפי חוקי מטריצות: למשל: $|A| = -1$

$$|A^n| = |A|^n = (-1)^n = f_{n+1}f_{n-1} - f_n^2 = (-1)^n$$

```
// complexity = O(log n)*O(2^2)= O(log n)
fibb_in_(int[][]a ,int n) {
    int[][] ans = new int[][]{{1, 0}, {0, 1}};
    while (n > 0) {
        if (n % 2 == 1) {
        ans = multMat(ans, a);
        }
        a = multMat(a,a);
        n = n / 2;
    }
    return
    a[0][1] // f_n
    a[1][1] // f_n - 1
    a[1][1]; // f_n + 1
}
```

0.14 בעיית החיציון

הגדרת חציון: האיבר שחצי סדרה גדולה ממנו , וחצי קטנה ממנה, (לא בהכרח איבר בסדרה)

1. נתון מערך A . המטרה: למצוא מספר הגדול מהחציון

פתרונות

(א) למיין את המערך לקחת את האיבר $a_{\frac{n}{2}+1}$ - סיבוכיות $O\left(n\log n\right)$

(ב) לחשב את המספר המקסימאלי $O(n)$,

(ג) להשתמש ב $\frac{n+1}{2}$ ולחשב את המקסימאלי $O(\frac{n}{2})$

(ד) לפעול לפי הרעיון הבא: $P\left(MAX(a_1,...a_k)\geq medion\right)=\frac{2^k-1}{2^k}$ ו- $P\left(MAX(a_1,...a_k)<medion\right)=\frac{1}{2^k}$, ולכן לכל מערך שהוא $m=MAX(a_1,.....a_{64})$

- סיבוכיות $O(64)=O(1)$

2. נתונים שני מערכים שווי אורך ממוינים, A,B , המטרה למצוא את האיברים האיברים שהם גדולים מהחציון

(א) $merge(A,B)=O(2n)$ - כל פעולה תלויה בפעולה קודמת.

(ב) נגדיר $c_i-max(a_i,b_{n-i})$ - יתרון - כל פעולה פועלת באופן עצמאי, ואז ניתן לפצל משימה למספר מעבדים - כלומר במחשב עם n ליבות , הסיבוכיות $O(1)$

הוכחה: באינדוקציה

- בסיס: נפצל למקרים:
  - אם $max(a_1,b_n)=a_1$: אז $a_1\geq b_n$ לכן $a_1\geq\{b_i\}$ לכל n האיברים לכן $a_1\geq median(Merge(A,B))$, a_1 נמצא במקום $n+1$ במערך הממוצג.
  - אם $max(a_1,b_n)=b_n$: אז המקום של b_n במערך הממוזג $\underbrace{b_1....b_{n-1},a_1}_{\text{n values}},b_n$ הוא $n+1$ ולנו יש $2n$ איברים, ומכאן שגדול מהחציון.
- צעד: באופן דומה לכל $i<n$

3. (שאלה של אור) נתונים שני מערכים שוני אורך ממוינים, A,B , המטרה למצוא את האיברים האיברים שהם גדולים מהחציון

הצעה לפתרון:

- אתחול: קח את החציון של A , קח את החציון של B השווה בינהם
- (בה"כ $A>B$) שמור את את איברי A הגדולים מהחציון + S_A מוגדר כמספר האיברים שלקחנו
- זרוק את איברי B הקטנים מהחציון + S_B מוגדר כמספר האיברים שזרקנו
- כעת נותר למצוא $S_{reset}=Size(A)+Size(B)-S_A-S_B$ איברים
  - פתרון נאיבי מיין את הנותר וקח S_{reset} איברים מהסוף $O(\frac{Size(A)}{2})+O(\frac{Size(B)}{2})$
  - פתרון רקוריסיבי :
    - ∗ צעד: קח חציון+ השווה + שמור וזרוק + כל עוד שמרתי פחות איברים מהדרוש חזור על הצעד
    - ∗ בסיס: אם שמרת יותר איברים מהדרוש קח חציון עליהם

find_cars_counting() {

```
int count, car_num = 0 ;
parking_spaces.head.data = 1;
Node temp = parking_spaces.head.next;
boolean ans = false;
while(!ans) {
    count = 1;
    while(temp.data != 1) {
        count++;
        temp = temp.next;
    }
    // We found a match for a mark
    temp.data = 0; // remove mark
    car_num = count;
    while ( count > 0 ) { // go backward
        temp = temp.prev;
        count--;
    }
    if (temp.data == 0 ) {
        ans = true;
    } else {
        temp = temp.next;
    }
}
return car_num;
```

}

find_cycle() {

```
if(parking_spaces2.head == null) {
    return false;
} else {
    Node fast = parking_spaces2.head;
    Node slow = parking_spaces2.head;
    while(fast != null && fast.next != null && slow != null ) {
        if (fast == slow) {
            return true;
        } else {
            fast = fast.next.next;
            slow = slow.next;
        }
    }
    return false;
```

}

}

הוכחות נכונות:

- נניח שישנה נקודת מפגש. ויהיה k מספר הצעדים מנ' ההתחלה.
- נסמן ב n את מספר הקודקודים, p מספר המעגלים של הצב, q מס המעגלים של הארנב.
- נסמן i כמות הצעדים שנעשו

  כעת:

  – עבור הצב , כמות הצעדים שנעשו $i = pn + k$
  – עבור הארנב, כמות הצעדים שנעשו $2i = qn + k$

- נשווה בינהם

$$\begin{aligned} 2pn + 2k &= qn + k \\ k &= qn - 2pn \\ k &= (q - 2p)n \end{aligned}$$

- מכיון ו n מספר זה מספר הקודקודים , ו k היא כפולה של n, ולכן אם יש נקודת מפגש היא בדיוק נקודת ההתחלה
- אצלנו: $p = 1$ ו ־ $q = 2$, ואכן $k = 0$, כלומר 0 ישנו מרחק 0 בין נקודת המפגש לנקודת ההתחלה $\Longleftrightarrow$ נקודת המפגש היא נקודת ההתחלה

<u>נכונות למעגל עם זרוע:</u>

- נניח שישנה נקודת מפגש. ויהיה k מספר הצעדים מנ' ההתחלה.
- נסמן ב n את מספר הקודקודים, ב m את אורך הזרוע, p מספר המעגלים של הצב, q מס המעגלים של הארנב.
- נסמן i כמות הצעדים שנעשו, כאשר iצעדים לצב, $2i$ צעדים לארנב:
- לכן:

$$i = m + np + k \qquad 2i = m + nq + k$$

- כעת :

$$\begin{aligned} 2m + 2np + 2k &= m + nq + k \\ m + k &= n\left(q - 2p\right) \\ k &= n\left(q - 2p\right) - m \end{aligned}$$

- כלומר נקודת המפגש נמצאת $n - m$ מתחילת המעגל . (הערה: אם $m < n$ נגדיר $m = [m]_n$)
- נראה שקבוצת הפתרונות לא ריקה: אם נחבר את $p = 0$, $q = m$, $k = mn - m$ נקבל:

$$i = m + mn - m = mn$$

- <u>מסקנה מיידית:</u> אם מנקודת המפגש שלהם, ילך הצב לתחילת המסלול, ושניהם יקדמו במהירות צב, לאחר m צעדים הם יפגשו $\Longleftrightarrow$ אנחנו יודעים את אורך הזרוע
- <u>מסקנה מיידית:</u> מנקודת המפגש בין המעגל לזרוע, הארנב ישאר במקומו והצב יתקדם בצעדיו הרגילים, לאחר p צעדים הם יפגשו $\Longleftrightarrow$ אנחו יודעים את היקף המעגל

- **private void find_cycle() {**

```
int arm_length = 0 ;
int cycle_perimeter = 0;
if(parking_spaces2.head == null) {
    System.out.println("no cycle");
    return;
} else {
    Node fast = parking_spaces2.head.next;
    Node slow = parking_spaces2.head;
    while(fast != null && fast.next != null && slow != null ) {
        if (fast == slow) {
            // they met - calc arm length
            slow = parking_spaces2.head;
            fast = fast.next; // E = V - 1 ⟺ m − 1 steps to intersection
            while (fast != slow) {
                arm_length++;
                //both slow steps
                fast = fast.next;
                slow = slow.next;
            }
            System.out.println("arm length: "+ arm_length);
            // both in meeting point with cycle - calc perimeter
            slow = slow.next;
            while (slow != fast) {
                cycle_perimeter++;
                slow = slow.next;
            }
            cycle_perimeter++;
            System.out.println("cycle perimeter: "+ cycle_perimeter);
            return;
        } else {
            fast = fast.next.next;
            slow = slow.next;
        }
    }
    System.out.println("no cycle");
    return;
}

}
```

האלגוריתם של ניבש

- תהיה פונקציה $F:D\to D$ ואיבר x_0 , $n=|D|$
- וסדרה:

$$x_0 \qquad F(x_0)=x_1 \qquad F(x_1)=x_2 \qquad F(x_2)=x_2 \quad ...$$

- אלגוריתם עם מחסנית אחת - זמן ריצה $O(n)$
- k מחסניות - פחות הוצאות - זמן ממוצע בהסתברות $O\left(log\ n\right)$

```
nivash (x_0 ,F){
    x = x_0;
    Stack s := init
    time =  1;
    s.push(x_0 , time);
    while(true) {
        x= f(x);
        if(x == s.top().val){
            return time - s.top().t;
        while(x < s.top.val) {
            s.pop;
            if(x == s.top().val) {
                return time - s.top.t
        s.push(x,time)
        time++;
}
```

- חישוב המחירים

```
//mat include x,y prices we need to calc entries and pathes
clc_prices(int[][] mat) {
    int x_price, y_price;
    //init 2D entries - init (0,0)
    mat[0][0].entry = 0;
    // init first row
    for (int i = 1; i < n; ++i) {
        mat[i][0].entry = mat[i - 1][0].entry + mat[i - 1][0].y;
        mat[i][0].pathes_num = 1;
    }
    // init first col
    for (int j = 1; j < m; ++j) {
        mat[0][j].entry = mat[0][j - 1].entry + mat[0][j - 1].x;
        mat[0][j].pathes_num = 1;
    }
    for (int i = 1; i < n; ++i) {
        for (int j = 1; j < m; ++j) {
            // get x price + prev entry
            x_price = mat[i - 1][j].entry + mat[i - 1][j].y;
            // get y price + prev entry
            y_price = mat[i][j - 1].entry + mat[i][j - 1].x;
            // peak max
            if (x_price < y_price) {
              mat[i][j].entry = x_price;
              mat[i][j].pathes_num = mat[i - 1][j].pathes_num;
            } else if (y_price < x_price) {
              mat[i][j].entry = y_price;
              mat[i][j].pathes_num = mat[i][j - 1].pathes_num;
            } else {
              // arbitrary choice
              mat[i][j].entry = x_price;
              // equal, so take both:
              mat[i][j].pathes_num = mat[i - 1][j].pathes_num + mat[i][j - 1].pathes_num;
            }
        }
    }
    System.out.println("cheapest price is: " + mat[n - 1][m - 1].entry);
    return mat;
}
```

- קבלת מסלול בודד

```
get_path(int[][] prices) {
    // get (n.m)
    int [][] mat = calc_prices(prices);
    int i = mat.length - 1; j = mat[0].length - 1;
    // array for path
    int path_len = i+j;
    int[] path = new int[path_len];
    // each loop check which point is match to current entry
    for (int s = 0; s < path_len; ++s) {
        if (i > 0) {
            if (mat[i][j].entry == mat[i - 1][j].y + mat[i - 1][j].entry) {
            path[s] = 1;
            --i;
            } else {
              path[s] = 0;
              --j;
              }
        } else {
            path[s] = 0;
              --j;
        }
    }
    return path;
}
```

- קבלת כל המסלולים ברקורסיה

```
private void getAllPathes() {
    Vector<String> allTracks = new Vector<String>();
    getAllPathesRec(new String(), mat.length - 1, mat[0].length - 1, allTracks);
    for( String track: allTracks) {
        System.out.println(track);
    }
}
getAllPathesRec(String track, int i, int j, Vector<String> allTracks) {
    if (i > 0 && j > 0) {
        int maybe1 = mat[i - 1][j].entry + mat[i - 1][j].y;
        int maybe2 = mat[i][j - 1].entry + mat[i][j - 1].x;
        if (maybe1 < maybe2) {
            getAllPathesRec("1" + track, i - 1, j, allTracks);
```

```
        } else if (maybe1 > maybe2) {
            getAllPathesRec("0" + track, i, j - 1, allTracks);
        } else { // found intersection
            getAllPathesRec("1" + track, i - 1, j, allTracks);
            getAllPathesRec("0" + track, i, j - 1, allTracks);
        }
    }
    if (i == 0 && j == 0) { // starting point
        allTracks.add(track);
    }
    else if (i == 0) { // reached to bottom limit
        String temp = "";
        for (int t = j; t > 0; --t) {
            temp += "0";
        }
        allTracks.add(temp + track);
    } else if (j == 0) { // reached to left limit
        String temp = "";
        for (int t = i; t > 0; --t) {
            temp += "1";
    }
    allTracks.add(temp + track);
}
```

0.17 בונוס

חיפוש בינארי(אריה במדבר):

RBS(A,v,left,right) ·

```
if left > right
    return null
mid = ⌊left + right/2⌋
if v == A[mid] //                  נשווה את s לאיבר ה ⌊n/2⌋ בגודל המערך.
    return mid //                  אם יש שוויון - מצאנו!
elif v > A[mid] //                 אם s גדול יותר נמשיך בחיפוש על תת המערך שמימין לאיבר ה ⌊n/2⌋
    return RBS(A,v,mid+1,right)
else //                            אם s קטן יותר נמשיך בחיפוש על תת המערך שמשמאל לאיבר ה ⌊n/2⌋
    return RBS(A,v,left,mid-1)
```

$merge - sort$

```
Merge Sort([1,....,n])
if (n>1)
    merge sort([A[1...n/2])          //          פירוק "החצי הראשון"
    merge sort([A[n/2...n])          //          פירוק ה"חצי השני"
    merge([A[1...n/2]),A[n/2...n],A[1...n]) //  קריאה לאלגוריתם המחבר את המערכים
merge(A,B,C[1....n]
    i,j,l = 1                        //          איפוס המונים
    while i<n/2 or j<n/2
        c[l] = min (A[i],B[j])
        if min was A[i]:             //          בדיקה וקידום האינדקס הנבחר
            i += 1
        else
            j += 1
        l += 1                       //          התקדמות במערך הממוין
```