תרגילים
מהספר

1. Let $G = (V, E)$ be an undirected graph with n nodes. Recall that a subset of the nodes is called an *independent set* if no two of them are joined by an edge. Finding large independent sets is difficult in general; but here we'll see that it can be done efficiently if the graph is "simple" enough.

Call a graph $G = (V, E)$ a *path* if its nodes can be written as $v_1, v_2, \ldots, v_n$, with an edge between v_i and v_j if and only if the numbers i and j differ by exactly 1. With each node v_i, we associate a positive integer *weight* w_i.

Consider, for example, the five-node path drawn in Figure 6.28. The *weights* are the numbers drawn inside the nodes.

The goal in this question is to solve the following problem:

Find an independent set in a path G whose total weight is as large as possible.

(a) Give an example to show that the following algorithm *does not* always find an independent set of maximum total weight.

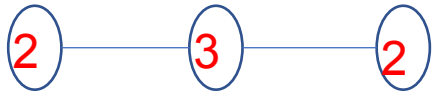
```
The "heaviest-first" greedy algorithm
  Start with S equal to the empty set
  While some node remains in G
    Pick a node v_i of maximum weight
    Add v_i to S
    Delete v_i and its neighbors from G
  Endwhile
  Return S
```

(b) Give an example to show that the following algorithm also *does not* always find an independent set of maximum total weight.

```
Let S_1 be the set of all v_i where i is an odd number
Let S_2 be the set of all v_i where i is an even number
(Note that S_1 and S_2 are both independent sets)
Determine which of S_1 or S_2 has greater total weight,
  and return this one
```

Figure 6.28 A paths with weights on the nodes. The maximum weight of an independent set is 14.

(c) Give an algorithm that takes an n-node path G with weights and returns an independent set of maximum total weight. The running time should be polynomial in n, independent of the values of the weights.



א.

ב.

ג. נגדיר OPT(i) כמשקל של קבוצת הצמתים הבלתי תלויה

Si = {V1..Vi}

נאתחל :

OPT(0)= 0 וכן OPT(1)= W1.

עבור i > 1 יש שני מקרים :

אם Vi שייך ל-Si אז ברור ש-Vi-1 לא שייך ל-Si ולכן

OPT(i)= Wi + OPT(i-2).

אם Vi לא שייך ל-Si אז OPT(i)= Wi + OPT(i-1).

מכאן נוסחת הנסיגה :

OPT(i)= max(OPT(i-1), Wi + OPT(i-2))

עכשיו ניתן לחשב איטרטיבית את OPT(i) עבור i=2...n.

ערך הפתרון, משקלה של הקבוצה הבלתי תלויהבעלת המשקל המקסימלי, נמצא ב-OPT(n).

עבור תוכן הפתרון - הצמתים נמצאים בקבוצה הבלתי תלויה המקסימלית – יש לעקוב אחרי הבחירה של אופרטור ה-max בכל שלב, החל מ- n ומטה.

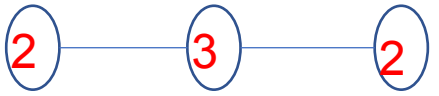
סיבוכיות : מבנה הנתונים הוא בגודל O(n) ועבור כל תא היה צורך בזמן חישוב קבוע, לכן סיבוכיות הזמן היא O(n).

```java
public static int independentSubset(int[] values)
{
    int[] opt = new int[values.length];
    opt[0] = 0;
    opt[1] = values[1];
    for (int i = 2; i < values.length; i++)
        opt[i] = Math.max(opt[i - 1], values[i] + opt[i - 2]);

    int temp = values.length - 1;
    while (temp > 1)
    {
        if (opt[temp] == opt[temp - 2] + values[temp])
        {
            System.out.println("node " + temp + " value " + values[temp]);
            temp = temp - 2;
        }
        else
            temp = temp - 1;
    }
    if (temp == 1)
        System.out.println("node " + temp + " value " + values[temp]);

    System.out.println(Arrays.toString(opt));
    return(opt[values.length - 1]);
}
```

א.



ב.

ג. נגדיר OPT(i) כמשקל של קבוצת הצמתים הבלתי תלויה

Si = {V1..Vi}

נאתחל :

OPT(0)= 0 וכן OPT(1)= W1.

עבור i > 1 יש שני מקרים :

אם Vi שייך ל-Si אז ברור ש-Vi-1 לא שייך ל-Si ולכן

OPT(i)= Wi + OPT(i-2).

אם Vi לא שייך ל-Si אז OPT(i)= Wi + OPT(i-1).

מכאן נוסחת הנסיגה :

OPT(i)= max(OPT(i-1), Wi + OPT(i-2))

עכשיו ניתן לחשב איטרטיבית את OPT(i) עבור i=2…n.

ערך הפתרון, משקלה של הקבוצה הבלתי תלויהבעלת המשקל המקסימלי, נמצא ב-OPT(n).

עבור תוכן הפתרון - הצמתים נמצאים בקבוצה הבלתי תלויה המקסימלית – יש לעקוב אחרי הבחירה של אופרטור ה-max בכל שלב, החל מ- n ומטה.

סיבוכיות : מבנה הנתונים הוא בגודל O(n) ועבור כל תא היה צורך בזמן חישוב קבוע, לכן סיבוכיות הזמן היא O(n).

2. Suppose you're managing a consulting team of expert computer hackers, and each week you have to choose a job for them to undertake. Now, as you can well imagine, the set of possible jobs is divided into those that are *low-stress* (e.g., setting up a Web site for a class at the local elementary school) and those that are *high-stress* (e.g., protecting the nation's most valuable secrets, or helping a desperate group of Cornell students finish a project that has something to do with compilers). The basic question, each week, is whether to take on a low-stress job or a high-stress job.

If you select a low-stress job for your team in week i, then you get a revenue of $\ell_i > 0$ dollars; if you select a high-stress job, you get a revenue of $h_i > 0$ dollars. The catch, however, is that in order for the team to take on a high-stress job in week i, it's required that they do no job (of either type) in week $i-1$; they need a full week of prep time to get ready for the crushing stress level. On the other hand, it's okay for them to take a low-stress job in week i even if they have done a job (of either type) in week $i-1$.

So, given a sequence of n weeks, a *plan* is specified by a choice of "low-stress," "high-stress," or "none" for each of the n weeks, with the property that if "high-stress" is chosen for week $i > 1$, then "none" has to be chosen for week $i-1$. (It's okay to choose a high-stress job in week 1.) The *value* of the plan is determined in the natural way: for each i, you add ℓ_i to the value if you choose "low-stress" in week i, and you add h_i to the value if you choose "high-stress" in week i. (You add 0 if you choose "none" in week i.)

The problem. Given sets of values $\ell_1, \ell_2, \ldots, \ell_n$ and $h_1, h_2, \ldots, h_n$, find a plan of maximum value. (Such a plan will be called *optimal*.)

Example. Suppose $n = 4$, and the values of ℓ_i and h_i are given by the following table. Then the plan of maximum value would be to choose "none" in week 1, a high-stress job in week 2, and low-stress jobs in weeks 3 and 4. The value of this plan would be $0 + 50 + 10 + 10 = 70$.

	Week 1	Week 2	Week 3	Week 4
ℓ	10	1	10	10
h	5	50	5	1

(a) Show that the following algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer.

```
For iterations i = 1 to n
  If h_{i+1} > ℓ_i + ℓ_{i+1} then
    Output "Choose no job in week i"
    Output "Choose a high-stress job in week i+1"
    Continue with iteration i+2
  Else
    Output "Choose a low-stress job in week i"
    Continue with iteration i+1
  Endif
End
```

To avoid problems with overflowing array bounds, we define $h_i = \ell_i = 0$ when $i > n$.

In your example, say what the correct answer is and also what the above algorithm finds.

(b) Give an efficient algorithm that takes values for $\ell_1, \ell_2, \ldots, \ell_n$ and $h_1, h_2, \ldots, h_n$ and returns the *value* of an optimal plan.

	שבוע 1	שבוע 2	שבוע 3
low	2	2	2
high	1	5	10

א.

ב. נגדיר OPT(i) כרווח המירבי שניתן להשיג בשבועות 1...i.
נאתחל

OPT(1) = max(L1, H1)

עבור i > 1 יש שני מקרים :
אם בשבוע i נבחרת משימה במתח נמוך, אזי לבחירה זאת יצטרף פתרון אופטימלי עבור השבועות 1..i-1.
אם בשבוע i נבחרת משימה במתח גבוה, אזי לבחירה זאת יצטרף פתרון אופטימלי עבור השבועות 1..i-2.

מכאן נוסחת הנסיגה :

OPT(i) = max(Li + OPT(i-1), Hi + OPT(i-2))

עכשיו ניתן לחשב איטרטיבית את OPT(i) עבור i=1...n.

ערך הפתרון, ערכה של התוכנית האופטימלית, נמצא ב-OPT(n).

עבור תוכן הפתרון - איזו משימה נבחרה בכל שבוע – יש לעקוב אחרי הבחירה של אופרטור ה-max בכל שלב, החל מ- n ומטה.

סיבוכיות : מבנה הנתונים הוא בגודל O(n)
ועבור כל תא היה צורך בזמן חישוב קבוע, לכן סיבוכיות הזמן היא O(n).

3. Let $G = (V, E)$ be a directed graph with nodes $v_1, \ldots, v_n$. We say that G is an *ordered graph* if it has the following properties.

(i) Each edge goes from a node with a lower index to a node with a higher index. That is, every directed edge has the form (v_i, v_j) with $i < j$.

(ii) Each node except v_n has at least one edge leaving it. That is, for every node v_i, $i = 1, 2, \ldots, n-1$, there is at least one edge of the form (v_i, v_j).

The length of a path is the number of edges in it. The goal in this question is to solve the following problem (see Figure 6.29 for an example).

Given an ordered graph G, find the length of the longest path that begins at v_1 and ends at v_n.

(a) Show that the following algorithm does not correctly solve this problem, by giving an example of an ordered graph on which it does not return the correct answer.

```
Set w = v_1
Set L = 0
While there is an edge out of the node w
  Choose the edge (w, v_j)
     for which j is as small as possible
  Set w = v_j
  Increase L by 1
end while
Return L as the length of the longest path
```

In your example, say what the correct answer is and also what the algorithm above finds.

(b) Give an efficient algorithm that takes an ordered graph G and returns the *length* of the longest path that begins at v_1 and ends at v_n. (Again, the *length* of a path is the number of edges in the path.)

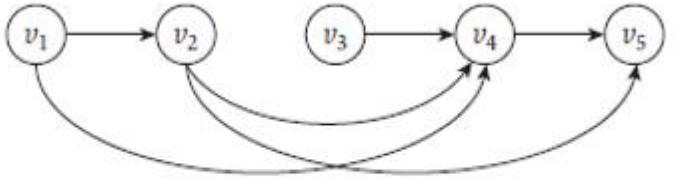
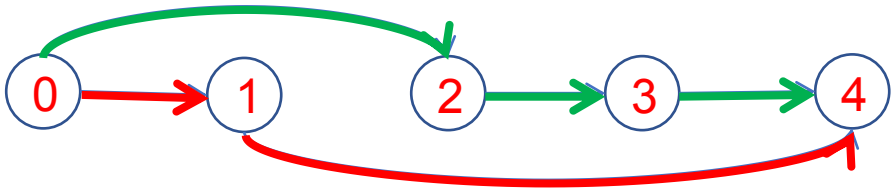


Figure 6.29 The correct answer for this ordered graph is 3: The longest path from v_1 to v_n uses the three edges (v_1, v_2),(v_2, v_4), and (v_4, v_5).

א.



ב. נגדיר OPT(i) כאורך המסלול המרבי מצמתים V0 עד Vi.
נאתחל

OPT(0) = 0

לכל צומת נבדוק את כל הקשתות שנכנסות אליו.
ניקח את הערך המקסימלי מבין הערכים של כל הצמתים שיש קשת מהם אל הצומת הנוכחי, ונוסיף לו 1.

פרט שצריך לשים אליו לב : יש צמתים שאין מסלול בין V1 אליהם, ולכן כשבודקים צומת שיוצאת ממנו קשת צריך לבדוק שהערך שיש לו אינו ברירת המחדל.

ערך הפתרון, אורכו של המסלול הארוך ביותר בגרף, נמצא ב-opt(n).
עבור תוכן הפתרון יש לתחזק מבנה נתונים בו תתועד לכל צומת איזו מהקשתות שנכנסות אליו נבחרה בגלל שערך ה-opt שלה הוא מקסימלי.

```java
private static int longPath(boolean[][] edges)
{
    int[] OPT = new int[edges.length];
    int[] s = new int[edges.length];
    OPT[0] = 0;
    for (int i = 1; i < OPT.length; i++)
    {
        OPT[i] = Integer.MIN_VALUE;
        for (int j = 0; j < OPT.length; j++)
            if (edges[j][i])
                if (OPT[j] != Integer.MIN_VALUE)
                    if (OPT[i] < OPT[j] + 1)
                    {
                        OPT[i] = OPT[j] + 1;
                        s[i] = j;
                    }
    }

    int temp = edges.length - 1;
    while (temp != 0)
    {
        System.out.print(temp + "<-");
        temp = s[temp];
    }
    System.out.println("0");
    return OPT[edges.length - 1];
}
```

ב. נגדיר OPT(i) כאורך המסלול המרבי מצמתים V0 עד Vi.
נאתחל

OPT(0) = 0

לכל צומת נבדוק את כל הקשתות שנכנסות אליו.
ניקח את הערך המקסימלי מבין הערכים של כל הצמתים שיש קשת מהם אל הצומת הנוכחי, ונוסיף לו 1.

פרט שצריך לשים אליו לב : יש צמתים שאין מסלול בין V1 אליהם, ולכן כשבודקים צומת שיוצאת ממנו קשת צריך לבדוק שהערך שיש לו אינו ברירת המחדל.

ערך הפתרון, אורכו של המסלול הארוך ביותר בגרף, נמצא ב-opt(n).
עבור תוכן הפתרון יש לתחזק מבנה נתונים בו תתועד לכל צומת איזו מהקשתות שנכנסות אליו נבחרה בגלל שערך ה-opt שלה הוא מקסימלי.

4. Suppose you're running a lightweight consulting business—just you, two associates, and some rented equipment. Your clients are distributed between the East Coast and the West Coast, and this leads to the following question.

Each month, you can either run your business from an office in New York (NY) or from an office in San Francisco (SF). In month i, you'll incur an *operating cost* of N_i if you run the business out of NY; you'll incur an operating cost of S_i if you run the business out of SF. (It depends on the distribution of client demands for that month.)

However, if you run the business out of one city in month i, and then out of the other city in month $i+1$, then you incur a fixed *moving cost* of M to switch base offices.

Given a sequence of n months, a *plan* is a sequence of n locations—each one equal to either NY or SF—such that the i^{th} location indicates the city in which you will be based in the i^{th} month. The *cost* of a plan is the sum of the operating costs for each of the n months, plus a moving cost of M for each time you switch cities. The plan can begin in either city.

The problem. Given a value for the moving cost M, and sequences of operating costs $N_1, \ldots, N_n$ and $S_1, \ldots, S_n$, find a plan of minimum cost. (Such a plan will be called *optimal*.)

Example. Suppose $n = 4$, $M = 10$, and the operating costs are given by the following table.

	Month 1	Month 2	Month 3	Month 4
NY	1	3	20	30
SF	50	20	2	4

Then the plan of minimum cost would be the sequence of locations

$$[NY, NY, SF, SF],$$

with a total cost of $1+3+2+4+10=20$, where the final term of 10 arises because you change locations once.

(a) Show that the following algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer.

```
For i = 1 to n
  If N_i < S_i then
    Output "NY in Month i"
  Else
    Output "SF in Month i"
End
```

In your example, say what the correct answer is and also what the algorithm above finds.

(b) Give an example of an instance in which every optimal plan must move (i.e., change locations) at least three times.

Provide a brief explanation, saying why your example has this property.

(c) Give an efficient algorithm that takes values for n, M, and sequences of operating costs $N_1, \ldots, N_n$ and $S_1, \ldots, S_n$, and returns the *cost* of an optimal plan.

א. אם M = 10, הוצאות התפעול בניו-יורק הן {1,4,1} והוצאות התפעול בסן-פרנסיסקו הן {20,1,20}, הרי שהתכנון האופטימלי יהיה להישאר כל הזמן בניו-יורק, בשעה שהאלגוריתם החמדן יציע ניו-יורק – סן-פרנסיסקו – ניו-יורק.

ב. אם M = 10, הוצאות התפעול בניו-יורק הן {1,100,1,100} והוצאות התפעול בסן-פרנסיקו הן {100,1,100,1}, הרי שהתכנון האופטימלי יהיה לעבור כל חודש לעיר השנייה, והמחיר יהיה 34 (שלושה מעברים ו-4 חודשי שהות שכל אחד מהם עולה 1). אם ננסה לוותר על אחד המעברים, המשמעות תהיה עלות של לפחות 100, ולכן לא אופטימלית.

ג. סדרת עלויות התפעול האופטימלית מסתיימת בהכרח או בניו-יורק או בסן-פרנסיסקו. נגדיר בנפרד את OPTn(j) כעלות של הסדרה האופטימלית שמסתיימת בניו-יורק בחודש ה-j, ואת OPTs(j) כעלות של הסדרה האופטימלית שמסתיימת בסן-פרנסיסקו בחודש ה-j.

כאשר סידרה מסתיימת בחודש ה-j בניו-יורק, יש שני מקרים :
או להיות גם בחודש שלפני כן בניו-יורק, ואז
OPTn(j) = Nj + OPTn(j-1)
או להיות בחודש שלפני כן בסן-פרנסיסקו ואז לעבור לניו-יורק, ואז
OPTn(j) = Nj + M + OPTs(j-1)
לכן נוסחת הנסיגה עבור OPTn(j) היא
OPTn(j) = Nj + min(OPTn(j-1), M + OPTs(j-1))
ובסימטריה מלאה
OPTs(j) = Sj + min(OPTs(j-1), M + OPTn(j-1))

פסאודו קוד :

```
OPTn(0) = OPTs(0) = 0
For i=1…n
        OPTn(j) = Nj + min(OPTn(j-1), M + OPTs(j-1))
        OPTs(j) = Sj + min(OPTs(j-1), M + OPTn(j-1))
EndFor
Return min(OPTn(n), OPTs(n))
```

סיבוכיות : מבנה הנתונים הוא בגודל O(n) ועבור כל תא היה צורך בזמן חישוב קבוע, לכן סיבוכיות הזמן היא O(n).

20. Suppose it's nearing the end of the semester and you're taking n courses, each with a final project that still has to be done. Each project will be graded on the following scale: It will be assigned an integer number on a scale of 1 to $g > 1$, higher numbers being better grades. Your goal, of course, is to maximize your average grade on the n projects.

You have a total of $H > n$ hours in which to work on the n projects cumulatively, and you want to decide how to divide up this time. For simplicity, assume H is a positive integer, and you'll spend an integer number of hours on each project. To figure out how best to divide up your time, you've come up with a set of functions $\{f_i : i = 1, 2, \ldots, n\}$ (rough

יהי OPT(i,h) הסכום המקסימלי של ציונים שניתן להשיג בקורסים 1..i בלכל היותר h שעות.

איתחול :

לכל h

OPT(0,h) = 0

ולכל i

OPT(i,0) = $\sum_{j=1}^{i} f_j(0)$

נוסחת הנסיגה :

OPT(i,h) = $\max_{0 \le k \le h} (fi(k) + OPT(i-1, h-k))$

התוצאה הסופית נמצאת ב - OPT(n,H).

בכל צעד של הכרעת מקסימום אנו רושמים במבנה נתונים נפרד מהו ה-k שהביא למקסימום, כדי לשחזר בסופו של דבר את פרקי הזמן שמיועדים לכל קורס.

גודל מבנה הנתונים הינו O(nH) ולמילוי כל תא יש צורך ב-O(H) פעולות, ולכן סיבוכיות הזמן הינה $O(nH^2)$.

24. *Gerrymandering* is the practice of carving up electoral districts in very careful ways so as to lead to outcomes that favor a particular political party. Recent court challenges to the practice have argued that through this calculated redistricting, large numbers of voters are being effectively (and intentionally) disenfranchised.

Computers, it turns out, have been implicated as the source of some of the "villainy" in the news coverage on this topic: Thanks to powerful software, gerrymandering has changed from an activity carried out by a bunch of people with maps, pencil, and paper into the industrial-strength process that it is today. Why is gerrymandering a computational problem? There are database issues involved in tracking voter demographics down to the level of individual streets and houses; and there are algorithmic issues involved in grouping voters into districts. Let's think a bit about what these latter issues look like.

Suppose we have a set of n *precincts* $P_1, P_2, \ldots, P_n$, each containing m registered voters. We're supposed to divide these precincts into two *districts*, each consisting of $n/2$ of the precincts. Now, for each precinct, we have information on how many voters are registered to each of two political parties. (Suppose, for simplicity, that every voter is registered to one of these two.) We'll say that the set of precincts is *susceptible* to gerrymandering if it is possible to perform the division into two districts in such a way that the same party holds a majority in both districts.

Give an algorithm to determine whether a given set of precincts is susceptible to gerrymandering; the running time of your algorithm should be polynomial in n and m.

Example. Suppose we have $n = 4$ precincts, and the following information on registered voters.

Precinct	1	2	3	4
Number registered for party A	55	43	60	47
Number registered for party B	45	57	40	53

This set of precincts is susceptible since, if we grouped precincts 1 and 4 into one district, and precincts 2 and 3 into the other, then party A would have a majority in both districts. (Presumably, the "we" who are doing the grouping here are members of party A.) This example is a quick illustration of the basic unfairness in gerrymandering: Although party A holds only a slim majority in the overall population (205 to 195), it ends up with a majority in not one but both districts.

נגדיר OPT(j, p, x, y) (מטריצת בולאנים !) שבה יהיה ערך true אם ניתן להגיע לפחות ל-x מצביעים למפלגה A באזור אחד ולפחות ל-y מצביעים למפלגה A באזור השני, כאשר p מתוך j המחוזות הראשונים מצורפים לאיזור הראשון. אחרת – false.

אם במחוז j + 1 יש z מצביעים, אזי כדי לחשב את OPT(j + 1, p, x, y) צריך לבחון שתי אפשרויות, והתוצאה תהיה OR של שתיהן :
או שמחוז j + 1 יצורף לאיזור 1, ואז יש להתייחס ל-OPT(j, p - 1, x - z, y),
או שמחוז j + 1 יצורף לאיזור 2, ואז יש להתייחס ל-OPT(j, p - 1, x, y - z).

כדי לקבל תשובה סופית – האם קבוצת מחוזות מועדת להטייה – יש לחפש במטריצה OPT אחר הערך true במקומות OPT(n, n/2, x, y) כאשר כל אחד מ-x ו-y גדול מ – mn/4 (שהרי מספר הקולות בכל אחד משני האיזורים הינו mn/2).

גודל מבנה הנתונים הינו $O(n^2m^2)$, וכיון שלחישוב כל תא במבנה הנתונים נדרש זמן קבוע, זוהי גם סיבוכיות הזמן.

מבחנים

שאלה 1 – תכנון כפל מטריצות

<u>תכנון כפל מטריצות</u>. כזכור, המכפלה $A_1 \times A_2 \times \ldots \times A_n$ של סדרת מטריצות מוגדרת רק כשישנה התאמה בין מספרי השורות והעמודות: אם מסמנים ב- r_i את מספר השורות במטריצה A_i, וב- c_i את מספר העמודות שלה, אז חייב להתקיים התנאי $r_{i+1} = c_i$ לכל $1 \le i < n$. במקרה שכזה כל מכפלה $A_i \times A_{i+1}$ הינה מטריצה בת r_i שורות ו- c_{i+1} עמודות, והחישוב שלה (בהתאם להגדרת כפל מטריצות) ניתן לביצוע ע"י $\Theta(r_i \times c_i \times c_{i+1})$ פעולות אלמנטריות בלבד. (כפל וחיבור מספרים נחשב לפעולה אלמנטרית). כזכור, כפל מטריצות הוא גם אסוציאטיבי, כלומר, בהכפלה של סדרת מטריצות, הננו רשאים למקם את הסוגריים כרצוננו. למשל $(A_1 \times A_2) \times A_3 = A_1 \times (A_2 \times A_3)$.

(א) הציגו דוגמה של שלוש מטריצות, שבה מיקום מסוים של הסוגריים דורש פי אלף פעולות אלמנטריות מאשר המיקום האחר.

(ב) הציגו אלגוריתם, שמקבל כקלט רשימה $(r_1, c_1), \ldots, (r_n, c_n)$ של מספרי השורות והעמודות בכל מטריצה, ומפיק כפלט מיקום אופטימלי של הסוגריים עבור ההכפלה $A_1 \times \ldots \times A_n$.
(שימו לב שאיננו מבצעים עדיין את הכפלת המטריצות, אלא רק מנסים **<u>לקבוע את מיקום הסוגריים</u>**, שימזער את מספר הפעולות האריתמטיות בזמן ההכפלה).

תשובה 1 – תכנון כפל מטריצות

<u>תכנון כפל מטריצות</u>. כזכור, המכפלה $A_1 \times A_2 \times \ldots \times A_n$ של סדרת מטריצות מוגדרת רק כשישנה התאמה בין מספרי השורות והעמודות: אם מסמנים ב-r_i את מספר השורות במטריצה A_i, וב-c_i את מספר העמודות שלה, אז חייב להתקיים התנאי $r_{i+1} = c_i$ לכל $1 \le i < n$. במקרה שכזה כל מכפלה $A_i \times A_{i+1}$ הינה מטריצה בת r_i שורות ו-c_{i+1} עמודות, והחישוב שלה (בהתאם להגדרת כפל מטריצות) ניתן לביצוע ע"י $\Theta(r_i \times c_i \times c_{i+1})$ פעולות אלמנטריות בלבד. (כפל וחיבור מספרים נחשב לפעולה אלמנטרית). כזכור, כפל מטריצות הוא גם אסוציאטיבי, כלומר, בהכפלה של סדרת מטריצות, הננו רשאים למקם את הסוגריים כרצוננו. למשל $(A_1 \times A_2) \times A_3 = A_1 \times (A_2 \times A_3)$.

(א) הציגו דוגמה של שלוש מטריצות, שבה מיקום מסוים של הסוגריים דורש פי אלף פעולות אלמנטריות מאשר המיקום האחר.

(ב) הציגו אלגוריתם, שמקבל כקלט רשימה $(r_1, c_1), \ldots, (r_n, c_n)$ של מספרי השורות והעמודות בכל מטריצה, ומפיק כפלט מיקום אופטימלי של הסוגריים עבור ההכפלה $A_1 \times \ldots \times A_n$.
(שימו לב שאיננו מבצעים עדיין את הכפלת המטריצות, אלא רק מנסים **<u>לקבוע את מיקום הסוגריים</u>**, שימזער את מספר הפעולות האריתמטיות בזמן ההכפלה).

$$m[i,j] = \begin{cases} 0 & \text{if } i = j\ , \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j\ . \end{cases}$$

תשובה 1 – תכנון כפל מטריצות

גירסה איטרטיבית

```
MATRIX-CHAIN-ORDER(p)
1  n = p.length − 1
2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
3  for i = 1 to n
4      m[i, i] = 0
5  for l = 2 to n                    // l is the chain length
6      for i = 1 to n − l + 1
7          j = i + l − 1
8          m[i, j] = ∞
9          for k = i to j − 1
10             q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11             if q < m[i, j]
12                 m[i, j] = q
13                 s[i, j] = k
14 return m and s
```

גירסה רקורסיבית

```
RECURSIVE-MATRIX-CHAIN(p, i, j)
1  if i == j
2      return 0
3  m[i, j] = ∞
4  for k = i to j − 1
5      q = RECURSIVE-MATRIX-CHAIN(p, i, k)
           + RECURSIVE-MATRIX-CHAIN(p, k + 1, j)
           + p_{i−1} p_k p_j
6      if q < m[i, j]
7          m[i, j] = q
8  return m[i, j]
```

גירסה רקורסיבית עם תיזכור

```
MEMOIZED-MATRIX-CHAIN(p)
1  n = p.length − 1
2  let m[1..n, 1..n] be a new table
3  for i = 1 to n
4      for j = i to n
5          m[i, j] = ∞
6  return LOOKUP-CHAIN(m, p, 1, n)

LOOKUP-CHAIN(m, p, i, j)
1  if m[i, j] < ∞
2      return m[i, j]
3  if i == j
4      m[i, j] = 0
5  else for k = i to j − 1
6          q = LOOKUP-CHAIN(m, p, i, k)
              + LOOKUP-CHAIN(m, p, k + 1, j) + p_{i−1} p_k p_j
7          if q < m[i, j]
8              m[i, j] = q
9  return m[i, j]
```

הדפסת התוצאה

```
PRINT-OPTIMAL-PARENS(s, i, j)
1  if i == j
2      print "A"_i
3  else print "("
4      PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5      PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6      print ")"
```

שאלה 2 – תת-סדרה מרבית רצופה

***שאלה 3ב* – תת-סדרה מרבית רצופה**. תת-סדרה רצופה של הסדרה $(x_1,x_2,...,x_{n-1},x_n)$, הינה סדרה מהצורה $(x_i,x_{i+1},...,x_{k-1},x_k)$ עבור איזשהם $1\le i\le k\le n$. הציגו אלגוריתם, שבהנתן סדרת מספרים שלמים $(x_1,...,x_n)$, מוצא בתוכה תת-סדרה רצופה שסכומה $x_i+...+x_k$ מרבי. למשל, בסדרת הקלט $(-1,4,-3,5,-1,-1,1,-1)$ תת-הסדרה הרצופה $(4,-3,5)$ מניבה את תת-הסכום המרבי $6=5+3-4$.

על האלגוריתם המוצע לרוץ בזמן $\Theta(n)$, כשבניתוח זמן הריצה מניחים, שכל פעולה אלמנטרית על מספרים (כמו חיבור, חיסור או השוואה) מתבצעת בזמן $\Theta(1)$.

```java
public static int maxContinuousSubseries(int[] values)
{
    int[] opt = new int[values.length];
    opt[0] = values[0];

    for (int i = 1; i < values.length; i++)
        opt[i] = Math.max(values[i], opt[i-1] + values[i]);

    int maxLocation = 0;
    for (int i = 1; i < values.length; i++)
        if (opt[i] > opt[maxLocation])
            maxLocation = i;

    System.out.println("opt : " + Arrays.toString(opt));

    System.out.print(values[maxLocation]);
    int sum = opt[maxLocation] - values[maxLocation];
    for (int i = maxLocation - 1; sum != 0; sum -= values[i], i--)
        System.out.print(" + " + values[i]);
    return(opt[maxLocation]);
}
```

תשובה 2 – תת-סדרה מרבית רצופה

***שאלה 3ב* – תת-סדרה מרבית רצופה**. תת-סדרה רצופה של הסדרה $(x_1,x_2,...,x_{n-1},x_n)$, הינה סדרה מהצורה $(x_i,x_{i+1},...,x_{k-1},x_k)$ עבור איזשהם $1\le i\le k\le n$. הציגו אלגוריתם, שבהנתן סדרת מספרים שלמים $(x_1,...,x_n)$, מוצא בתוכה תת-סדרה רצופה שסכומה $x_i+...+x_k$ מרבי. למשל, בסדרת הקלט $(-1,4,-3,5,-1,-1,1,-1)$ תת-הסדרה הרצופה $(4,-3,5)$ מניבה את תת-הסכום המרבי $4-3+5=6$. על האלגוריתם המוצע לרוץ בזמן $\Theta(n)$, כשבניתוח זמן הריצה מניחים, שכל פעולה אלמנטרית על מספרים (כמו חיבור, חיסור או השוואה) מתבצעת בזמן $\Theta(1)$.

נגדיר OPT(i) כסכום המרבי של תת-סדרה רצופה שמסתיימת ב-Xi וכוללת את Xi.

כל איבר חדש שמוסיפים יכול להמשיך סדרה קודמת או להתחיל תת-סדרה חדשה.

איתחול :

Opt(0) = X0

מכאן נוסחת הנסיגה :

OPT(i) = max(Xi, OPT(i-1) + Xi)

ערך הפתרון נמצא במקום שבו הערך של OPT הוא המקסימלי.

סיבוכיות : O(n)

שאלה 3 – תת-סדרה עולה מרבית

נתונה סדרה של n מספרים ממשיים, $a_1,...,a_n$. תת סדרה של מספרים אלו היא תת קבוצה של מספרים אלו המסודרת בהתאם לסדר הסדרה המקורית, $a_{i_1},...,a_{i_k}$ $(1 \le i_1 < i_2 < .... < i \le_k n)$. תת סדרה היא **עולה** אם $a_{i_1} < a_{i_2} ... < a_{i_k}$. הציעו אלגוריתם תכנון דינמי המוצא תת סדרה עולה באורך מקסימלי. הוכיחו נכונות ונתחו סיבוכיות.

ניסוח אלגוריתם + ניתוח נכונות:

ניתוח יעילות:

תשובה 3 – תת-סדרה עולה מרבית

נתונה סדרה של n מספרים ממשיים, $a_1,...,a_n$. תת סדרה של מספרים אלו היא תת קבוצה של מספרים אלו המסודרת בהתאם לסדר הסדרה המקורית, $a_{i_1},...,a_{i_k}$ $(1 \le i_1 < i_2 < .... < i \le_k n)$. תת סדרה היא **עולה** אם $a_{i_1} < a_{i_2} ... < a_{i_k}$. הציעו אלגוריתם תכנון דינמי המוצא תת סדרה עולה באורך מקסימלי. הוכיחו נכונות ונתחו סיבוכיות.

```
public static int longestIncreasingSubseries(int[] values)
{
    int[] opt = new int[values.length];
    int[] s = new int[values.length];

    opt[0] = 1;

    for (int i = 1; i < values.length; i++)
        for (int j = 0; j < i; j++)
            if (values[i] > values[j] && opt[j] + 1 > opt[i])
            {
                opt[i] = opt[j] + 1;
                s[i] = j;
            }

    int maxLocation = 0;
    for (int i = 0; i < values.length; i++)
        if (opt[i] > opt[maxLocation])
            maxLocation = i;

    int temp;
    for (temp = maxLocation; opt[temp] > 1; temp = s[temp])
        System.out.print(values[temp] + " ");

       System.out.println("opt : " + Arrays.toString(opt));
       System.out.println("s : " + Arrays.toString(s));

    return(values[temp]);
}
```

נגדיר opt(i) כאורך המרבי של תת סדרה עולה בין האיברים a_1.. a_i.

איתחול :

Opt(0)

נוסחת הנסיגה :

$$opt(i) = \max_{\substack{j<i \\ a_j<a_i}} opt(j) + 1$$

סיבוכיות : $O(n^2)$

ערך הפתרון נמצא במקום שבו הערך של OPT הוא המקסימלי.

כדי לקבל את תוכן הפתרון יש לבנות מבנה נתונים מקביל s
ובו לתעד כל החלטה של max בנוסחת הנסיגה.

שאלה 4 – פלינדרום מרבי

פלינדרום מרבי. פלינדרום הינה מחרוזת שנקראת בצורה זהה מימין לשמאל או משמאל לימין. למשל המחרוזת "ABBA" באנגלית, והמחרוזת הבאה בעברית "דעו מאביכם כי לא בוש אבוש שוב אשוב אליכם כי בא מועד" (כשמתעלמים מסימן הרווח). פלינדרום מרבי בתוך מחרוזת נתונה, היא תת-מחרוזת **רצופה**, שמהווה פלינדרום, ושאורכה מרבי. למשל בתוך המחרוזת abcbea, הפלינדרום המרבי הוא bcb (ולא abcba שאיננה תת-מחרוזת רצופה). הציגו אלגוריתם למציאת פלינדרום מרבי בתוך מחרוזת קלט באורך n מעל האלפבית האנגלי. בשאלה זו לא יינתן ניקוד לאלגוריתמים טריוויאליים, שרצים בזמן $\Theta(n^3)$ (הזמן הנדרש לבדיקת כל תתי המחרוזות הרצופות).

```java
public static int longestPalindrome(String s)
{
    boolean opt[][] = new boolean[s.length()][s.length()];
    for (int i = 0; i < s.length(); i++)
        opt[i][i] = true;
    for (int i = 0; i < s.length() - 1; i++)
        if (s.charAt(i) == s.charAt(i + 1))
            opt[i][i + 1] = true;
    for (int length = 2; length < s.length(); length++)
        for (int start = 0; start < s.length() - length; start++)
            opt[start][start + length] =
            opt[start + 1][start + length - 1] &&
            s.charAt(start) == s.charAt(start + length);

    for (int i = 0; i < s.length(); i++)
        System.out.println(Arrays.toString(opt[i]));

    for (int length = s.length() - 1; length >= 1; length--)
        for (int start = s.length() - length - 1; start >= 0; start--)
            if (opt[start][start + length])
            {
                System.out.println("Longest palyndrome is from " +
                    start + " to " + (start + length) + " : " +
                    s.substring(start, start + length + 1));
                return length + 1;
            }
    return 1;
}
```

תשובה 4 – פלינדרום מרבי

פלינדרום מרבי. פלינדרום הינה מחרוזת שנקראת בצורה זהה מימין לשמאל או משמאל לימין. למשל המחרוזת "ABBA" באנגלית, והמחרוזת הבאה בעברית "דעו מאביכם כי לא בוש אבוש שוב אשוב אליכם כי בא מועד" (כשמתעלמים מסימן הרווח). פלינדרום מרבי בתוך מחרוזת נתונה, היא תת-מחרוזת **רצופה**, שמהווה פלינדרום, ושאורכה מרבי. למשל בתוך המחרוזת abcbea, הפלינדרום המרבי הוא bcb (ולא abcba שאיננה תת-מחרוזת רצופה). הציגו אלגוריתם למציאת פלינדרום מרבי בתוך מחרוזת קלט באורך n מעל האלפבית האנגלי. בשאלה זו לא יינתן ניקוד לאלגוריתמים טריוויאליים, שרצים בזמן $\Theta(n^3)$ (הזמן הנדרש לבדיקת כל תתי המחרוזות הרצופות).

נגדיר OPT(i,j) להיות true אם התווים שבמקומות i..j המחרוזת הם פאלינדרום, false אחרת.

איתחול :

Opt(i, i) = 1

Opt(i,j) = 2 אם s[i] == s[j] && j = i+1

Opt(i,j) = 1 אם s[i] != s[j] && j = i+1

נוסחת הנסיגה :

Opt(i,j) = opt(i + 1, j-1) && s[i] == s[j]

עבור התשובה הסופית נחפש מבין כל המקומות במטריצה שבהם נמצא הערך true את המקום שבו ההפרש בין i ל-j הינו המירבי.

סיבוכיות : $O(n^2)$

שאלה 5 – משחק מטבעות

משחק המטבעות מתקיים בין שני שחקנים. שחקן א' משחק בצעדים האי-זוגיים ושחקן ב' בצעדים הזוגיים. בתחילת המשחק נתונה סדרה של מטבעות בעלי ערכים $(v_1,\ldots,v_n)$. בכל צעד במשחק, השחקן הנוכחי בוחר ומשלשל לכיסו או את המטבע שבקצה הימני של הסדרה הנוכחית או, לחלופין, את המטבע שבקצה השמאלי של הסדרה הנוכחית. אם, למשל, בתחילת המשחק נתונה הסדרה $(10,20,5,4)$, אז שחקן א' עשוי לבחור תחילה במטבע שבקצה הימני שערכו 4. כעת שחקן ב' עשוי לבחור מתוך הסדרה שנותרה $(10,20,5)$ את המטבע בקצה השמאלי שערכו 10. אחר כך שחקן א' עשוי לבחור מתוך $(20,5)$ את המטבע שערכו 20, ולבסוף בצעד האחרון שחקן ב' "בוחר" את המטבע שנותר שערכו 5. בדוגמא זו הרווח לשחקן א' הוא $4+20=24$, והרווח לשחקן ב' הוא $10+5=15$. הציגו אלגוריתם תכנון דינמי, שבהנתן סדרת קלט $(v_1,\ldots,v_n)$ מחשב מהו הרווח המרבי, שהשחקן הפותח (שחקן א') יכול להבטיח שישלשל לכיסו. רווח מרבי זה חייב להיות מובטח, ללא תלות בצעדיו של שחקן ב'.

תשובה 5 – משחק מטבעות

משחק המטבעות מתקיים בין שני שחקנים. שחקן א' משחק בצעדים האי-זוגיים ושחקן ב' בצעדים הזוגיים. בתחילת המשחק נתונה סדרה של מטבעות בעלי ערכים $(v_1,\ldots,v_n)$. בכל צעד במשחק, השחקן הנוכחי בוחר ומשלשל לכיסו או את המטבע שבקצה הימני של הסדרה הנוכחית או, לחלופין, את המטבע שבקצה השמאלי של הסדרה הנוכחית. אם, למשל, בתחילת המשחק נתונה הסדרה $(10,20,5,4)$, אז שחקן א' עשוי לבחור תחילה במטבע שבקצה הימני שערכו 4. כעת שחקן ב' עשוי לבחור מתוך הסדרה שנותרה $(10,20,5)$ את המטבע בקצה השמאלי שערכו 10. אחר כך שחקן א' עשוי לבחור מתוך $(20,5)$ את המטבע שערכו 20, ולבסוף בצעד האחרון שחקן ב' "בוחר" את המטבע שנותר שערכו 5. בדוגמא זו הרווח לשחקן א' הוא $4+20=24$, והרווח לשחקן ב' הוא $10+5=15$. הציגו אלגוריתם תכנון דינמי, שבהנתן סדרת קלט $(v_1,\ldots,v_n)$ מחשב מהו הרווח המרבי, שהשחקן הפותח (שחקן א') יכול להבטיח שישלשל לכיסו. רווח מרבי זה חייב להיות מובטח, ללא תלות בצעדיו של שחקן ב'.

נגדיר OPT(i,j) כרווח ששחקן א' יכול להבטיח לעצמו כשבמשחק נותרה תת-סדרה רצופה של מטבעות מ-i עד j.
בכל שלב מספר המטבעות שנשארו במשחק הינו j-i+1.

נניח בלי הגבלת הכלליות שמספר המטבעות זוגי. לכן שחקן א' משחק כשמספר המטבעות שנותרו זוגי, ושחקן ב' משחק כשמספר המטבעות שנותרו אי-זוגי.

איתחול :

$$OPT(i, i) = 0$$

כיון שנותר במשחק מטבע אחד, ושחקן ב' לוקח אותו.

נוסחת הנסיגה :
כאשר תורו של שחקן א' לשחק :

$$OPT(i,j) = \max(Vi + OPT(i + 1, j), Vj + OPT(i, j - 1))$$

כאשר תורו של שחקן ב' לשחק הוא מנסה למזער את הרווח של שחקן א', ולכן :

$$OPT(i,j) = \min(0 + OPT(i + 1, j), 0 + OPT(i, j - 1))$$

התשובה נמצאת ב - opt(1,n)

סיבוכיות : $O(n^2)$

תשובה 5 – משחק מטבעות

נגדיר OPT(i,j) כרווח ששחקן א' יכול להבטיח לעצמו כשבמשחק נותרה תת-סדרה רצופה של מטבעות מ-i עד j.
בכל שלב מספר המטבעות שנשארו במשחק הינו j-i+1.

נניח בלי הגבלת הכלליות שמספר המטבעות זוגי. לכן שחקן א' משחק כשמספר המטבעות שנותרו זוגי, ושחקן ב' משחק כשמספר המטבעות שנותרו אי-זוגי.

איתחול :

$OPT(i, i) = 0$

כיון שנותר במשחק מטבע אחד, ושחקן ב' לוקח אותו.

נוסחת הנסיגה :
כאשר תורו של שחקן א' לשחק :

$OPT(i,j) = \max(Vi + OPT(i + 1, j), Vj + OPT(i, j - 1))$

כאשר תורו של שחקן ב' לשחק הוא מנסה למזער את הרווח של שחקן א', ולכן :

$OPT(i,j) = \min(0 + OPT(i + 1, j), 0 + OPT(i, j - 1))$

התשובה נמצאת ב - opt(1,n)

סיבוכיות : $O(n^2)$

```
public static int coinsGame(int[] values)
{
    int[][] opt = new int[values.length][values.length];
    int[][] s = new int[values.length][values.length];
    if (values.length % 2 == 0)
    {
        for (int i = 0; i < values.length; i++)
        {
            opt[i][i] = 0;
            s[i][i] = i;
        }
        for (int coinsLeft = 2; coinsLeft <= values.length; coinsLeft++)
            for (int i = 0; i <= values.length - coinsLeft; i++)
            {
                int j = i + coinsLeft - 1;
                if (coinsLeft % 2 == 0) // PLAYER A
                {
                    opt[i][j] =
                    Math.max(values[i] + opt[i + 1][j],
                        values[j] + opt[i][j - 1]);
                    if (values[i] + opt[i + 1][j] > values[j] + opt[i][j - 1])
                        s[i][j] = i;
                    else
                        s[i][j] = j;
                }
                else // PLAYER B
                {
                    opt[i][j] = Math.min(opt[i + 1][j], opt[i][j - 1]);
                    if (opt[i + 1][j] < opt[i][j - 1])
                        s[i][j] = i;
                    else
                        s[i][j] = j;
                }

            }
```

תשובה 5 – משחק מטבעות

נגדיר OPT(i,j) כרווח ששחקן א' יכול להבטיח לעצמו כשבמשחק נותרה תת-סדרה רצופה של מטבעות מ-i עד j.
בכל שלב מספר המטבעות שנשארו במשחק הינו j-i+1.

נניח בלי הגבלת הכלליות שמספר המטבעות זוגי. לכן שחקן א' משחק כשמספר המטבעות שנותרו זוגי, ושחקן ב' משחק כשמספר המטבעות שנותרו אי-זוגי.

איתחול :

$OPT(i, i) = 0$

כיון שנותר במשחק מטבע אחד, ושחקן ב' לוקח אותו.

נוסחת הנסיגה :
כאשר תורו של שחקן א' לשחק :

$OPT(i,j) = \max(Vi + OPT(i + 1, j), Vj + OPT(i, j - 1))$

כאשר תורו של שחקן ב' לשחק הוא מנסה למזער את הרווח של שחקן א', ולכן :

$OPT(i,j) = \min(0 + OPT(i + 1, j), 0 + OPT(i, j - 1))$

התשובה נמצאת ב - opt(1,n)

סיבוכיות : $O(n^2)$

```
    int j = values.length - 1;
    int i = 0;
    int sumA = 0, sumB = 0;
    while (j >= i)
    {
        if ((j - i + 1) % 2 == 0) // PLAYER A
        {
            sumA += values[s[i][j]];
            System.out.println("Player A took coin number " + s[i][j] +
                " value = " + values[s[i][j]] + " sum so far : " + sumA);
            if (s[i][j] == i)
                i = i +1;
            else
                j = j - 1;
        }
        else // PLAYER B
        {
            sumB += values[s[i][j]];
            System.out.println("Player B took coin number " + s[i][j] +
                " value = " + values[s[i][j]] + " sum so far : " + sumB);
            if (s[i][j] == i)
                i = i +1;
            else
                j = j - 1;
        }
    }

    return opt[0][values.length - 1];
}
```

שאלה 6 – מסלול בין מלונות

***שאלה 3* – תכנון דינאמי – בחירת מלונות לאורך מסלול** (25 נק'). ברצוננו לערוך מסע לאורכו של מסלול ישר מנקודת התחלה s לנקודת סיום f. נתונה רשימה $p_1 < \ldots < p_n$ של מיקומי מלונות, כך שמלון i ממוקם בדיוק p_i קילומטרים מתחילת המסלול. במהלך המסע לנים בכל לילה במלון אחר. החופשה מוגבלת בזמן, ולכן חייבים להשלים את המסע תוך לכל היותר t ימים ($t < n$ נתון). ידוע שכמות המאמץ שנדרש ביום הליכה הינה הריבוע d^2 של המרחק d, שהולכים באותו יום. ברצוננו לבחור את נקודות הלינה, כך שנמזער את סכום המאמצים בכלל ימי המסע. הציגו אלגוריתם יעיל ככל האפשר לבעיה, שרץ בזמן פולינומי ביחס ל-t וביחס ל-n). נדרשת תשובה של 4-5 שורות, שכוללת נוסחה רקורסיבית לפתרון בשיטה של תכנון דינאמי.

תשובה 6 – מסלול בין מלונות

***שאלה 3* – תכנון דינאמי – בחירת מלונות לאורך מסלול** (25 נק'). ברצוננו לערוך מסע לאורכו של מסלול ישר מנקודת התחלה s לנקודת סיום f. נתונה רשימה $p_1 < \ldots < p_n$ של מיקומי מלונות, כך שמלון i ממוקם בדיוק p_i קילומטרים מתחילת המסלול. במהלך המסע לנים בכל לילה במלון אחר. החופשה מוגבלת בזמן, ולכן חייבים להשלים את המסע תוך לכל היותר t ימים ($t < n$ נתון). ידוע שכמות המאמץ שנדרש ביום הליכה הינה הריבוע d^2 של המרחק d, שהולכים באותו יום. ברצוננו לבחור את נקודות הלינה, כך שנמזער את סכום המאמצים בכלל ימי המסע. הציגו אלגוריתם יעיל ככל האפשר לבעיה, שרץ בזמן פולינומי ביחס ל-t וביחס ל-n). נדרשת תשובה של 4-5 שורות, שכוללת נוסחה רקורסיבית לפתרון בשיטה של תכנון דינאמי.

נגדיר OPT(i,j) כסכום המזערי של מאמצים הנדרש למסלול בן j ימים שמסתיים במלון i.

כיון שבכל לילה לנים במלון אחר, חייב להתקיים i >= j, כלומר מספר המלון שבו לנים בלילה ה-j גדול או שווה ל-j.

אם i=j משמעות הדבר שבכל לילה עוברים למלון הבא, ולכן סכום המאמצים הינו סכום ריבועי המרחקים בין מלון למלון ב-i המלונות הראשונים.

כיון שפונקציית הריבוע קמורה, אם אין מגבלה על מספר הימים – המאמץ המינימלי יתקבל אם בכל יום נתקדם מלון אחד קדימה.

איתחול :

אם ביום הראשון כבר הגענו למלון i, המאמץ הוא ריבוע המרחק אליו

$OPT(i, 1) = p_i^2$

נוסחת הנסיגה : $OPT(i, j) = \min_{k<i} (OPT(k, j-1) + (p_i - p_k)^2)$

ערך הפתרון נמצא ב-opt(n, t), שהוא המאמץ המינימלי הנדרש כדי להגיע למלון האחרון ב-t ימים.

כדי לקבל את תוכן הפתרון יש לבנות מבנה נתונים מקביל s ובו לתעד כל החלטה של min בנוסחת הנסיגה.

תשובה 6 – מסלול בין מלונות

נגדיר OPT(i,j) כסכום המזערי של מאמצים הנדרש למסלול בן j ימים שמסתיים במלון i.

כיון שבכל לילה לנים במלון אחר, חייב להתקיים i >= j, כלומר מספר המלון שבו לנים בלילה ה-j גדול או שווה ל-j.

אם i=j משמעות הדבר שבכל לילה עוברים למלון הבא, ולכן סכום המאמצים הינו סכום ריבועי המרחקים בין מלון למלון ב-i המלונות הראשונים.

כיון שפונקציית הריבוע קמורה, אם אין מגבלה על מספר הימים – המאמץ המינימלי יתקבל אם בכל יום נתקדם מלון אחד קדימה.

איתחול :

אם ביום הראשון כבר הגענו למלון i, המאמץ הוא ריבוע המרחק אליו

$OPT(i, 1) = p_i^2$

נוסחת הנסיגה : $OPT(i, j) = \min_{k<i} (OPT(k, j-1) + (p_i - p_k)^2)$

ערך הפתרון נמצא ב-opt(n, t), שהוא המאמץ המינימלי הנדרש כדי להגיע למלון האחרון ב-t ימים.

כדי לקבל את תוכן הפתרון יש לבנות מבנה נתונים מקביל s ובו לתעד כל החלטה של min בנוסחת הנסיגה.

```
public static int hotelsTour(int []distances)
{
    int[][] opt = new int[distances.length][distances.length + 1];
    int[][] s = new int[distances.length][distances.length + 1];
    for (int i = 0; i < distances.length; i++)
        opt[i][1] = (int)Math.pow(distances[i], 2);

    for (int i = 1; i < distances.length; i++)
        opt[i][i + 1] = opt[i - 1][i] + (int)Math.pow(distances[i] - distances[i-1], 2);

    for (int i = 2; i < distances.length; i++)
        for (int j = 2; j <= i; j++)
        {
            opt[i][j] = opt[i - 1][j-1] + (int)Math.pow(distances[i] - distances[i - 1], 2);
            for (int k = 2; k <= i - j + 2; k++)
            {
                if (i == 4 && j == 2)
                    System.out.println("here " + opt[i][j]);
                if (opt[i - k][j-1] + Math.pow(distances[i] - distances[i - k], 2) < opt[i][j])
                {
                    opt[i][j] = opt[i - k][j-1] + (int)Math.pow(distances[i] - distances[i - k], 2);
                    s[i][j] = k;
                }
            }
        }

    for (int i = 0; i < distances.length; i++)
        System.out.println(Arrays.toString(opt[i]));
    for (int i = 0; i < distances.length; i++)
        System.out.println(Arrays.toString(s[i]));

    int minLocation = 0;
    for (int i = 1; i < distances.length; i++)
        if (opt[distances.length - 1][i] < opt[distances.length - 1][minLocation])
            minLocation = i;

    return distances[minLocation];
}
```

שאלה 7 – ספירת סידורים אפשריים

יהי $f(n)$ מספר הדרכים לסדר n עצמים באמצעות שני היחסים $<$ וכן $=$. למשל $f(2)=3$ משום שעבור 2 עצמים a,b ישנם בדיוק 3 סידורים אפשריים: $a<b$ או $a=b$ או $b<a$.

בדומה, $f(3)=13$ משום שעבור 3 עצמים a,b,c ישנם כבר 13 סידורים אפשריים:

$$\left\{\begin{array}{lll} a=b=c, & b=c<a, & c<a=b \\ a=b<c, & b<a=c, & c<a<b \\ a<b=c, & b<a<c, & c<b<a \\ a<b<c, & b<c<a, & \\ a=c<b, & & \\ a<c<b, & & \end{array}\right\}$$

הציגו אלגוריתם שעל קלט n מחשב את $f(n)$ תוך ריצה בזמן $\Theta(n^2)$ וצריכת זיכרון $\Theta(n)$.

(החשיבו פעולה אריתמטית (חיבור, כפל והשוואה של מספרים) כפעולה שמתבצעת בזמן $\Theta(1)$, וצורכת $\Theta(1)$ תאי זיכרון בלבד).

הדרכה: העזרו בעובדה שכל סידור משרה חלוקה למחלקות שקילות, כשכל מחלקה מורכבת מעצמים שהיחס ביניהם הוא $=$. למשל עבור 4 עצמים, הסידור $d<a=b<c=e$ משרה חלוקה ל-3 מחלקות שקילות: d במחלקה נפרדת, a,b במחלקה נפרדת, ו-c,e במחלקה נפרדת.

תשובה 7 – ספירת סידורים אפשריים

יהי $f(n)$ מספר הדרכים לסדר n עצמים באמצעות שני היחסים $<$ וכן $=$. למשל $f(2)=3$ משום שעבור 2 עצמים a,b ישנם בדיוק 3 סידורים אפשריים: $a<b$ או $a=b$ או $b<a$.

בדומה, $f(3)=13$ משום שעבור 3 עצמים a,b,c ישנם כבר 13 סידורים אפשריים:

$$\left\{\begin{array}{lll} a=b=c, & b=c<a, & c<a=b \\ a=b<c, & b<a=c, & c<a<b \\ a<b=c, & b<a<c, & c<b<a \\ a<b<c, & b<c<a, & \\ a=c<b, & & \\ a<c<b, & & \end{array}\right\}$$

הציגו אלגוריתם שעל קלט n מחשב את $f(n)$ תוך ריצה בזמן $\Theta(n^2)$ וצריכת זיכרון $\Theta(n)$. (החשיבו פעולה אריתמטית (חיבור, כפל והשוואה של מספרים) כפעולה שמתבצעת בזמן $\Theta(1)$, וצורכת $\Theta(1)$ תאי זיכרון בלבד).

הדרכה: העזרו בעובדה שכל סידור משרה חלוקה למחלקות שקילות, כשכל מחלקה מורכבת מעצמים שהיחס ביניהם הוא $=$. למשל עבור 4 עצמים, הסידור $d<a=b<c=e$ משרה חלוקה ל-3 מחלקות שקילות: d במחלקה נפרדת, a,b במחלקה נפרדת, ו-c,e במחלקה נפרדת.

נגדיר OPT(i,j) להיות מספר יחסי הסדר המלאים על j עצמים כשיש בדיוק i מחלקות שקילות.

איתחול :

OPT(i, 1) = 1

נוסחת הנסיגה :

OPT(i, j) = OPT(i, j – 1) * i + opt(i – 1, j – 1) * i

כיון שכל איבר חדש יכול להצטרף למחלקת שקילות קודמת (ואז צריך לבחור עבורו מחלקה) או לפתוח מחלקה חדשה (ואז צריך לבחור עבורה מיקום בין המחלקות האחרות.)

שאלה 8 – פרוק למכפלות

***שאלה 3* – תכנון דינאמי – פרוק למכפלות** (25 נק').

נתון פרוק $x = p_1 \times p_2 \times .... \times p_k$ של מספר טבעי x לגורמים ראשוניים $p_1 < p_2 < .... < p_k$, כשאף חזקה בפרוק איננה גדולה מ-1. (למשל, הפרוק "$30 = 2^1 \times 3^1 \times 5^1$" הינו קלט חוקי, אבל הפרוק "$300 = 2^2 \times 3^1 \times 5^2$" איננו קלט חוקי). נסמן ב-$f(x)$ את <u>מספר הדרכים</u> להציג את x כמכפלה של מספרים טבעיים (לאו דווקא ראשוניים). למשל $f(30) = 5$ משום שישנן בדיוק 5 הצגות שונות של 30 כמכפלה של מספרים טבעיים: $1 \times 30 = 2 \times 15 = 3 \times 10 = 5 \times 6 = 2 \times 3 \times 5$. שימו לב שאין חשיבות לסדר של המוכפלים. למשל $2 \times 3 \times 5$ ו-$3 \times 2 \times 5$ נחשבות לאותה הצגה, (אבל 3×10 ו-$3 \times 2 \times 5$ נחשבות להצגות שונות). הציגו אלגוריתם שבהינתן קלט חוקי $x = p_1 \times p_2 \times .... \times p_k$ מחשב את $f(x)$. <u>הדרכה</u>: מה הקשר בין $f(x)$ לבין $f(y)$ כאשר $y = q_1 \times q_2 \times .... \times q_k$ הינו קלט חוקי עם אותו מספר של גורמים ראשוניים k. נדרשת תשובה של 4-5 שורות בלבד, שכוללת נוסחה רקורסיבית לפתרון בשיטה של תכנון דינאמי.

הרעיון המרכזי_____

_____

נוסחת הנסיגה_____

_____

_____

אלגוריתם וזמן ריצה_____

_____

תשובה 8 – פרוק למכפלות

***שאלה 3* – תכנון דינאמי – פרוק למכפלות** (25 נק').

נתון פרוק $x = p_1 \times p_2 \times .... \times p_k$ של מספר טבעי x לגורמים ראשוניים $p_1 < p_2 < .... < p_k$, כשאף חזקה בפרוק איננה גדולה מ-1. (למשל, הפרוק "$30 = 2^1 \times 3^1 \times 5^1$" הינו קלט חוקי, אבל הפרוק "$300 = 2^2 \times 3^1 \times 5^2$" איננו קלט חוקי). נסמן ב-$f(x)$ את מספר הדרכים להציג את x כמכפלה של מספרים טבעיים (לאו דווקא ראשוניים). למשל $f(30) = 5$ משום שישנן בדיוק 5 הצגות שונות של 30 כמכפלה של מספרים טבעיים: $30 = 2 \times 15 = 3 \times 10 = 5 \times 6 = 2 \times 3 \times 5 = 1 \times 30$. שימו לב שאין חשיבות לסדר של המוכפלים. למשל $2 \times 3 \times 5$ ו-$3 \times 2 \times 5$ נחשבות לאותה הצגה, (אבל 3×10 ו-$3 \times 2 \times 5$ נחשבות להצגות שונות). הציגו אלגוריתם שבהינתן קלט חוקי $x = p_1 \times p_2 \times .... \times p_k$ מחשב את $f(x)$. הדרכה: מה הקשר בין $f(x)$ לבין $f(y)$ כאשר $y = q_1 \times q_2 \times .... \times q_k$ הינו קלט חוקי עם אותו מספר של גורמים ראשוניים k. נדרשת תשובה של 4-5 שורות בלבד, שכוללת נוסחה רקורסיבית לפתרון בשיטה של תכנון דינאמי.

הרעיון המרכזי_____

_____

נוסחת הנסיגה_____

_____

_____

אלגוריתם וזמן ריצה_____

_____

f(x) הינו למעשה מספר הדרכים האפשריות לחלק את הקבוצה {1...k} לתת קבוצות. כיון שהמספרים $p_1 \dots p_k$ הם ראשוניים, הרי שכל תת-קבוצה של {1...k} משרה למעשה מספר טבעי שהוא מכפלה של הגורמים הראשוניים המתאימים. ולכן כל חלוקה של הקבוצה {1...k} לתת קבוצות הינה למעשה פרוק של x למכפלה של מספרים טבעיים.

דוגמה :

החלוקה של {1,2,3}	{1},{2},{3}	{3},{1,2}	{2},{1,3}	{1},{2,3}	{1,2,3}
ההצגה של 30 כמכפלת טבעיים	$p_1 \times p_2 \times p_3 = 2 \times 3 \times 5$	$p_3 \times (p_1 p_2) = 5 \times 6$	$p_1 \times (p_1 p_3) = 3 \times 10$	$p_1 \times (p_1 p_2) = 2 \times 15$	$p_1 p_2 p_3 = 30$

נגדיר OPT(i, j) כמספר החלוקות של הקבוצה {1...j} ל-i תת קבוצות זרות ולא ריקות.

איתחול :

OPT(1,j) = 1

כאשר מצרפים את הגורם הראשוני p_i, יש שתי אפשרויות :
או שהוא מצטרף לאחת מתת הקבוצות הקיימות,
או שהוא פותח תת קבוצה חדשה.
מכאן נוסחת הנסיגה :

OPT(i, j) = OPT(i, j-1) * i + OPT(i – 1, j – 1)

תשובה 9 - מיקום סוגריים בביטויים בוליאניים

Let **T(i, j)** represents the number of ways to parenthesize the symbols between i and j (both inclusive) such that the subexpression between i and j evaluates to true.

Let **F(i, j)** represents the number of ways to parenthesize the symbols between i and j (both inclusive) such that the subexpression between i and j evaluates to false.

Base Cases:

```
T(i, i) = 1 if symbol[i] = 'T'
T(i, i) = 0 if symbol[i] = 'F'

F(i, i) = 1 if symbol[i] = 'F'
F(i, i) = 0 if symbol[i] = 'T'
```

$$T(i,j) = \sum_{k=i}^{j-1} \begin{Bmatrix} T(i,k) * T(k+1,j) & if & operator & [k] is' \& ' \\ Total(i,k) * Total(k+1,j) - F(i,k) * F(k+1,j) & if & operator & [k] & is' | ' \\ T(i,k) * F(k+1,j) + F(i,k) * T(k+1,j) & if & operator & [k] & is' \oplus ' \end{Bmatrix}$$

$\text{Total(i,j)= T(i,j)+F(i,j)}$

$$F(i,j) = \sum_{k=i}^{j-1} \begin{Bmatrix} Total(i,k) * Total(k+1,j) - T(i,k) * T(k+1,j) & if & operator[k] & is' \& ' \\ F(i,k) * F(k+1,j) & if & operator[k] & is' | ' \\ T(i,k) * T(k+1,j) + F(i,k) * F(k+1,j) & if & operator[k] & is' \oplus ' \end{Bmatrix}$$

$\text{Total(i,j)=T(i,j)+F(i,j)}$

מיקום סוגריים בביטויים בוליאניים

ביטוי בוליאני ללא סוגריים, הינה סדרה של הקבועים הלוגיים T,F (המייצגים, כרגיל, את הערכים $True,False$), כך שבין כל שני קבועים, מופיע אחד מהאופרטורים הלוגיים $\wedge,\vee,\otimes$ (המייצגים כרגיל את האופרטורים, and, or, xor). אותו ביטוי בוליאני, עשוי לקבל ערך אמת שונה בהתאם למיקום הסוגריים. למשל, בביטוי $F \wedge T \otimes T$ ניתן למקם סוגריים בדיוק בשתי דרכים שונות: $(F \wedge T) \otimes T = T$ בעוד ש- $F \wedge (T \otimes T) = F$. הציגו אלגוריתם תכנון דינמי, שבהיתן ביטוי בוליאני ללא סוגריים, מחשב את מספר הדרכים למיקום סוגריים עבורן מתקבל הערך T. הניחו, לשם פשטות, שהקלט נתון במערך של הקבועים $A[1...n]$, ובמערך של האופרטורים $B[1...n-1]$. למשל הקלט $F \wedge T \otimes T$ נתון במערכים, שבהם $A[1]=F, A[2]=T, A[3]=T$ וכן $B[1]=\wedge, B[2]=\otimes$.

לקט

שאלה 1 - LCS

3.1 בעיית תת־המחרוזת המשותפת הארוכה ביותר LCS

קלט: 2 מחרוזות $X = x_1, \ldots, x_n$ ו־$Y = y_1, \ldots, y_m$.
מחפשים את תת־המחרוזת המשותפת (אפשר עם דילוגים, אבל לשמור על הסדר) באורך מירבי.
לדוגמא: $X = abccba, Y = bacbc$, אז abc תת־מחרות משותפת.

3.1 בעיית תת־המחרוזת המשותפת הארוכה ביותר LCS

קלט: 2 מחרוזות $X = x_1, \ldots, x_n$ ו־$Y = y_1, \ldots, y_m$.
מחפשים את תת־המחרוזת המשותפת (אפשר עם דילוגים, אבל לשמור על הסדר) באורך מירבי.
לדוגמא: $X = abccba, Y = bacbc$, אז abc תת־מחרות משותפת.

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1,j-1]+1 & \text{if } i,j > 0 \text{ and } x_i = y_j\,, \\ \max(c[i,j-1], c[i-1,j]) & \text{if } i,j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

```
LCS-LENGTH(X, Y)
 1  m = X.length
 2  n = Y.length
 3  let b[1..m, 1..n] and c[0..m, 0..n] be new tables
 4  for i = 1 to m
 5      c[i, 0] = 0
 6  for j = 0 to n
 7      c[0, j] = 0
 8  for i = 1 to m
 9      for j = 1 to n
10          if x_i == y_j
11              c[i, j] = c[i − 1, j − 1] + 1
12              b[i, j] = "↖"
13          elseif c[i − 1, j] ≥ c[i, j − 1]
14              c[i, j] = c[i − 1, j]
15              b[i, j] = "↑"
16          else c[i, j] = c[i, j − 1]
17              b[i, j] = "←"
18  return c and b
```

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2      return
3  if b[i, j] == "↖"
4      PRINT-LCS(b, X, i − 1, j − 1)
5      print x_i
6  elseif b[i, j] == "↑"
7      PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```

שאלה 2 - כנס

בעקבות התגיגות על ניצחונה של הפועל באר שבע השבוע, נדחה הכנס הבינלאומי שתוכנן. הפעם דאגה המחלקה למדעי המחשב לבקש את שני האודיטוריומים בבניין 26 (5 ו-6). להזכירכם סיגלית וזהבה רוצות לשבץ באותו היום מספר גדול ככל האפשר של הרצאות, מבין n הרצאות אפשריות, **בשני האודיטוריומים**. משך ההרצאה ה-i הוא l_i ($l_i \in \mathbb{N}$) שעות. אין הגבלה על זמני ההתחלה והסיום של ההרצאות. יורם מסר למזכירות כי באותו יום ניתן להשתמש במשך T ($T \in \mathbb{N}$) שעות בכל אחד משני האודיטוריומים. בעקבות ההצלחה בפעם הקודמת, התבקשתם לעזור לסיגלית וזהבה לשבץ **ללא התנגשויות** מספר מקסימלי של הרצאות בזמן T בשני האודיטוריומים. כאשר מתחילים הרצאה מסוימת, חייבים לסיימה (באופן רצוף).

תכננו אלגוריתם מבוסס תכנון דינמי לפתרון הבעיה עם שני אולמות

תשובה 2 - כנס

בעקבות התגיגות על ניצחונה של הפועל באר שבע השבוע, נדחה הכנס הבינלאומי שתוכנן. הפעם דאגה המחלקה למדעי המחשב לבקש את שני האודיטוריומים בבניין 26 (5 ו-6). להזכירכם סיגלית וזהבה רוצות לשבץ באותו היום מספר גדול ככל האפשר של הרצאות, מבין n הרצאות אפשריות, **בשני האודיטוריומים**. משך ההרצאה ה-i הוא l_i ($l_i \in \mathbb{N}$) שעות. אין הגבלה על זמני ההתחלה והסיום של ההרצאות. יורם מסר למזכירות כי באותו יום ניתן להשתמש במשך T ($T \in \mathbb{N}$) שעות בכל אחד משני האודיטוריומים. בעקבות ההצלחה בפעם הקודמת, התבקשתם לעזור לסיגלית וזהבה לשבץ **ללא התנגשויות** מספר מקסימלי של הרצאות בזמן T בשני האודיטוריומים. כאשר מתחילים הרצאה מסוימת, חייבים לסיימה (באופן רצוף).

תכננו אלגוריתם מבוסס תכנון דינמי לפתרון הבעיה עם שני אולמות

נבנה מטריצה תלת-מימדית $M_{n\times T\times T}$, כך שהתא במטריצה $m[i,j,k]$ יכיל את מספר ההרצאות המקסימלי שניתן לשבץ מתוך הקבוצה $\{a_1, a_2, \ldots, a_i\}$ כאשר באולם הראשון ניתן לשבץ הרצאות במשך j שעות, ובשני במשך k שעות.

איתחול :

עבור J < 0 או K < 0

m[i,j,k] = -1

עבור i = 0

m[i,j,k] = 0

מילוי מבנה הנתונים :

$$m[i,j,k] = \max \begin{Bmatrix} m[i-1,j,k], \\ m[i-1,j-l_i,k]+1, \\ m[i-1,j,k-l_i]+1 \end{Bmatrix}$$

פסאודו קוד :

for $i = 1$ to n do
  for $j = 0$ to T do
    for $k = 0$ to T do

$$m[i,j,k] = \max \begin{Bmatrix} m[i-1,j,k], \\ m[i-1,j-l_i,k]+1, \\ m[i-1,j,k-l_i]+1 \end{Bmatrix}$$

מילוי התא $m[i,j,k]$ תלוי רק בתאים בשכבה ה-$i-1$ (או בערכי הקצה כפי שהוגדרו בסעיף iii), ותאים אלו מולאו כבר באיטרציה הקודמת של הלולאה החיצונית.

שאלה 3 – לוכד החלומות

- על חישוק (ההיקף של עיגול) ממוקמים n חרוזים מחמישה צבעים שונים, ובנוסף חרוז יחיד הצבוע בשחור (המיקומים נקבעים לפי כוכבים ומזלות). החרוזים ממוספרים בתחום $n..1$, כך שהחרוז השחור מצוי בין החרוז 1 לבין החרוז n.
- מחברים בין זוגות של חרוזים מצבעים זהים על ידי חוטים, כך ששני חוטים שונים (שהגם שני מיתרים בעיגול) לא יחצו זה את זה, ולכל חרוז מחובר לכל היותר חוט אחד.
- כדי שלוכד החלומות יהיה בעל יעילות מרבית יש למתוח מספר מקסימלי של חוטים (מיתרים).

<u>בעיית לוכדי החלומות</u>: בהינתן חישוק אשר ממוקמים עליו n חרוזים כמתואר, כיצד ניתן לחבר את החרוזים ע"י חוטים כך שלוכד החלומות יהיה בעל יעילות מרבית?

תשובה 3 – לוכד החלומות

- על חישוק (ההיקף של עיגול) ממוקמים n חרוזים מחמישה צבעים שונים, ובנוסף חרוז יחיד הצבוע בשחור (המיקומים נקבעים לפי כוכבים ומזלות). החרוזים ממוספרים בתחום $1..n$, כך שהחרוז השחור מצוי בין החרוז 1 לבין החרוז n.
- מחברים בין זוגות של חרוזים מצבעים זהים על ידי חוטים, כך ששני חוטים שונים (שהגם שני מיתרים בעיגול) לא יחצו זה את זה, ולכל חרוז מחובר לכל היותר חוט אחד.
- כדי שלוכד החלומות יהיה בעל יעילות מרבית יש למתוח מספר מקסימלי של חוטים (מיתרים).

<u>בעיית לוכדי החלומות</u>: בהינתן חישוק אשר ממוקמים עליו n חרוזים כמתואר, כיצד ניתן לחבר את החרוזים ע"י חוטים כך שלוכד החלומות יהיה בעל יעילות מרבית?

במטריצה $M_{n\times n}$, התא $m[i,j]$ יכיל את המספר המקסימלי של המיתרים שניתן להעביר באופן חוקי בין חרוזים בתת הסדרה המתחילה בחרוז ה-i ומסתיימת בחרוז ה-j.

נגדיר $\delta(i,j)=1$ אם החרוזים i ו-j באותו הצבע, אחרת $\delta(i,j)=0$. אזי:

$$m[i,j]=\max\begin{cases} m[i+1,j], \\ m[i,j-1], \\ m[i+1,j-1]+\delta(i,j), \\ \max\limits_{i<k<j}\{m[i,k]+m[k+1,j]\} \end{cases}$$

```
for i = 1 to n do
    m[i,i] = 0
for l = 2 to n do
    for i = 1 to n − l + 1 do
        let j ← i + l − 1
```

$$m[i,j]=\max\begin{cases} m[i+1,j], \\ m[i,j-1], \\ m[i+1,j-1]+\delta(i,j), \\ \max\limits_{i<k<j}\{m[i,k]+m[k+1,j]\} \end{cases}$$

האלגוריתם ממלא מטריצה בגודל $n\times n$. מילוי כל תא לוקח $O(n)$ זמן, ולכן סה"כ זמן הריצה הוא $O(n^3)$.

שאלה 4 – מסיבה עליזה

התבקשת לייעץ למנהל חברה אשר מתכנן מסיבה לעובדיו. לחברה יש מבנה היררכי שבו המנהל הוא שורש העץ. מחלקת כוח אדם נתנה לכל עובד ציון ב-"עליזות" – מספר ממשי. כדי שהמסיבה תהיה כיפית, המנהל מבקש שלא יזמנו אליה גם עובד וגם הבוס הישיר שלו. כתוב אלגוריתם אשר יניב רשימת אורחים כזו שסכום ציוני ה-"עליזות" של כל המוזמנים יהיה המירבי. נתח את זמן הריצה של האלגוריתם.

תשובה 4 – מסיבה עליזה

התבקשת לייעץ למנהל חברה אשר מתכנן מסיבה לעובדיו. לחברה יש מבנה היררכי שבו המנהל הוא שורש העץ. מחלקת כוח אדם נתנה לכל עובד ציון ב-"עליזות" – מספר ממשי. כדי שהמסיבה תהיה כיפית, המנהל מבקש שלא יזמנו אליה גם עובד וגם הבוס הישיר שלו. כתוב אלגוריתם אשר יניב רשימת אורחים כזו שסכום ציוני ה-"עליזות" של כל המוזמנים יהיה המירבי. נתח את זמן הריצה של האלגוריתם.

נגדיר T(x,c) כסכום מידת העליזות בצמתים שצבעם c בעץ ששורשו הוא x.

אם x הוא עלה אזי

$$T(x,c)=\begin{cases} v & \text{if } x=\text{RED} \\ 0 & \text{if } x=\text{WHITE} \end{cases}$$

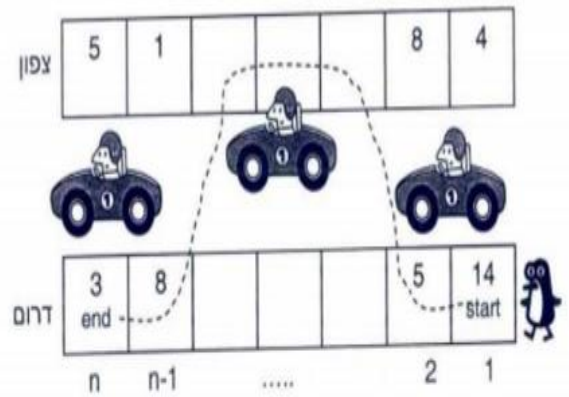
אחרת

$$T(x,c)=\begin{cases} v+\sum_i T(x.child_i,\text{WHITE}) & \text{if } x=\text{RED} \\ \sum_i \max(T(x.child_i,\text{WHITE}),T(x.child_i,\text{RED})) & \text{if } x=\text{WHITE} \end{cases}$$

התשובה הסופית היא $\max(T(root,\text{WHITE}),T(root,\text{RED}))$

שאלה 5 – מסע הפינגווין

נתון כביש שמשני צדדיו מדרכות משובצות. על כל משבצת מונחת כמות מסוימת של גרגירי תירס. פינגווין רעב מגיע למשבצת הראשונה בצד הדרומי של הכביש. בכל פעם שהפינגווין דורך במשבצת מסוימת הוא אוכל את כל גרגירי התירס שעליה.



הפינגווין מעוניין לאכול כמה שיותר גרגירי תירס בדרכו מהמשבצת הראשונה למשבצת ה-n-ית. בכל צעד הוא יכול להתקדם משבצת אחת קדימה (מערבה) באותו צד, או לחצות את הכביש (מצפון לדרום או להיפך) תוך התקדמות משבצת אחת קדימה (מערבה).

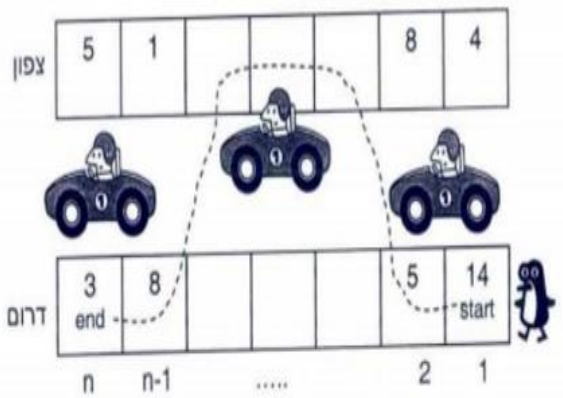
במהלך הטיול מותר לפינגווין לחצות את הכביש **d**<n פעמים לכל היותר. עליו לסיים את הטיול במשבצת המערבית (ה- n-ית) הדרומית. הקו המרוסק בציור מתאר מסלול אפשרי עם שתי חציות.
נסמן ב-a_i את מספר גרגירי התירס במשבצת ה-i-ית בצד בדרומי וב-b_i את מספר גרגירי התירס במשבצת ה-i-ית בצד בצפוני. (לכל i בין 1 ל-n). למשל בציור $a_2=5, b_1=4$. ערכי a_i ו-b_i ידועים מראש לכל i.

בשאלה זו עליכם לתאר אלגוריתם המבוסס על **תכנות דינאמי** העוזר לפינגווין לתכנן את מסלולו כך שבסך הכל יאכל במהלך הטיול את הכמות המירבית האפשרית של גרגירי תירס. נסתפק בחישוב הכמות ואין צורך לכלול בפלט כיצד להשיגו.

נסמן ב-S(i,k) את כמות הגרגרים המירבית שניתן לצבור עד (וכולל) המשבצת ה-i-ית אם בוצעו בדיוק k חציות עד משבצת זו. שימו לב שלערכי S(i,k) יש משמעות רק עבור k<i.

תשובה 5 – מסע הפינגווין

נתון כביש שמשני צדדיו מדרכות משובצות. על כל משבצת מונחת כמות מסוימת של גרגירי תירס. פינגווין רעב מגיע למשבצת הראשונה בצד הדרומי של הכביש. בכל פעם שהפינגווין דורך במשבצת מסוימת הוא אוכל את כל גרגירי התירס שעליה.



הפינגווין מעוניין לאכול כמה שיותר גרגירי תירס בדרכו מהמשבצת הראשונה למשבצת ה-n-ית. בכל צעד הוא יכול להתקדם משבצת אחת קדימה (מערבה) באותו צד, או לחצות את הכביש (מצפון לדרום או להיפך) תוך התקדמות משבצת אחת קדימה (מערבה).

במהלך הטיול מותר לפינגווין לחצות את הכביש **d<n** פעמים לכל היותר. עליו לסיים את הטיול במשבצת המערבית (ה- n-ית) הדרומית. הקו המרוסק בציור מתאר מסלול אפשרי עם שתי חציות.
נסמן ב-a_i את מספר גרגירי התירס במשבצת ה-i-ית בצד בדרומי וב-b_i את מספר גרגירי התירס במשבצת ה-i-ית בצד בצפוני. (לכל i בין 1 ל-n). למשל בציור $a_2=5$, $b_1=4$. ערכי a_i ו-b_i ידועים מראש לכל i.

בשאלה זו עליכם לתאר אלגוריתם המבוסס על **תכנות דינאמי** העוזר לפינגווין לתכנן את מסלולו כך שבסך הכל יאכל במהלך הטיול את הכמות המירבית האפשרית של גרגירי תירס. נסתפק בחישוב הכמות ואין צורך לכלול בפלט כיצד להשיגו.

נסמן ב-S(i,k) את כמות הגרגרים המירבית שניתן לצבור עד (וכולל) המשבצת ה-i–ית אם בוצעו בדיוק k חציות עד משבצת זו. שימו לב שלערכי S(i,k) יש משמעות רק עבור k<i.

איתחול :

$S(i, 0) = a_i$

$S(i, i) = 0$

נוסחת הנסיגה :

$$S(i, k) = \begin{cases} a_i + \max(S(i-1, k), s(i-1, k-1)) & \text{when } k \% 2 == 0 \\ b_i + \max(S(i-1, k), s(i-1, k-1)) & \text{when } k \% 2 == 1 \end{cases}$$

שאלה 6 – צילומי לווין

לוויין Google Earth נשלח לגיחת צילום. הלוויין מסוגל לצלם 2 סוגי תמונות – תמונה ברזולוציה גבוהה (HD) ותמונה ברזולוציה רגילה (SD). בידיכם רשימה של n אתרים אפשריים לצילום, נסמנם $\{1,2,\dots,n\}$. לכל אתר נקבע רווח מצילום ברזולוציה גבוהה $\{h_1,h_2,\dots,h_n\}$ ורווח מצילום ברזולוציה רגילה $\{s_1,s_2,\dots,s_n\}$ בהתאמה. עליכם לתכנן גיחת צילום רווחית ככל הניתן בזמן קצוב.

שימו לב: לכל $1 \le i,j \le n$ מעבר הלוויין מאתר i לאתר j **אינו** דורש זמן.

עבור כל אתר i, יש ללוויין **<u>בדיוק אחת</u>** משלוש האפשרויות הבאות:

1. לא לצלם את האתר ה-i.
2. לצלם את האתר ה-i ברזולוציה רגילה. משך הפעולה: **3 דק'**.
3. לצלם את האתר ה-i ברזולוציה גבוהה. משך הפעולה: **5 דק'**.

הלוויין עומד לרשותכם למשך T דקות (מספר טבעי כלשהו, ניתן להניח כי $T \le 5n$).

תכנית צילום חוקית מוגדרת כשתי תת-קבוצות **זרות** $H,S \subseteq \{1,\dots,n\}$. $H \cap S = \emptyset$ ומתקיים - $5|H| + 3|S| \le T$.

שווי תכנית צילום (H,S) מוגדר כ- $\sum_{i\in H} h_i + \sum_{i\in S} s_i$.

הצע אלגוריתם תכנון דינמי שימצא את השווי המקסימלי של תוכנית צילום חוקית.

תשובה 6 – צילומי לווין

לוויין Google Earth נשלח לגיחת צילום. הלוויין מסוגל לצלם 2 סוגי תמונות – תמונה ברזולוציה גבוהה (HD) ותמונה ברזולוציה רגילה (SD). בידיכם רשימה של n אתרים אפשריים לצילום, נסמנם $\{1,2,\dots,n\}$. לכל אתר נקבע רווח מצילום ברזולוציה גבוהה $\{h_1,h_2,\dots,h_n\}$ ורווח מצילום ברזולוציה רגילה $\{s_1,s_2,\dots,s_n\}$ בהתאמה. עליכם לתכנן גיחת צילום רווחית ככל הניתן בזמן קצוב.

שימו לב: לכל $1 \le i,j \le n$ מעבר הלוויין מאתר i לאתר j **אינו** דורש זמן.

עבור כל אתר i, יש ללוויין **<u>בדיוק אחת</u>** משלוש האפשרויות הבאות:

1. לא לצלם את האתר ה-i.
2. לצלם את האתר ה-i ברזולוציה רגילה. משך הפעולה: **3 דק'**.
3. לצלם את האתר ה-i ברזולוציה גבוהה. משך הפעולה: **5 דק'**.

הלוויין עומד לרשותכם למשך T דקות (מספר טבעי כלשהו, ניתן להניח כי $T \le 5n$).

תכנית צילום חוקית מוגדרת כשתי תת-קבוצות **זרות** $H,S \subseteq \{1,\dots,n\}$. $H \cap S = \emptyset$ ומתקיים - $5|H| + 3|S| \le T$.

שווי תכנית צילום (H,S) מוגדר כ- $\sum_{i\in H} h_i + \sum_{i\in S} s_i$.

הצע אלגוריתם תכנון דינמי שימצא את השווי המקסימלי של תוכנית צילום חוקית.

לכל $1 \le j \le n$, $0 \le t \le T$, נגדיר את $OPT(j,t)$ להיות שווי מקסימאלי של תוכנית צילום עבור אתרי הצילום $\{1,2,\dots,j\}$ כאשר הזמן הנותר לצילום הינו t.
עבור $j = 0$ נגדיר כי $OPT(0,t) = 0$ לכל $0 \le t \le T$.
פתרון לבעיה המקורית הינו $OPT(n,T)$.

נוסחאת המבנה לחישוב $OPT(j,t)$ הינה:

$$OPT(j,t) = \begin{cases} (i) & j = 0 & : & 0 \\ (ii) & j \ge 1 \wedge t \ge 5 & : & max\begin{Bmatrix} OPT(j-1,t-5)+h_j, \\ OPT(j-1,t-3)+s_j, \\ OPT(j-1,t) \end{Bmatrix} \\ (iii) & j \ge 1 \wedge 3 \le t < 5 & : & max\begin{Bmatrix} OPT(i-1,t-3)+s_j, \\ OPT(i-1,t) \end{Bmatrix} \\ (iv) & j \ge 1 \wedge t < 3 & : & OPT(j-1,t) \end{cases}$$

הסבר:

(*i*) מקרה בסיס – נובע ישירות מההגדרה.

(*ii*) במידה ויש לצלם j אתרי צילום ראשונים והזמן הנותר גדול מ-5, ישנן שלוש אופציות:
<u>ראשונה:</u> לצלם את האתר j ברזולוציה גבוהה – ואז הרווח ממנו הוא h_j ונותר לצלם את $j-1$ האתרים ב- $t-5$ דקות. לכן נוסיף את הרווח h_j לשווי תוכנית צילום אופטימאלית עבור מה שנותר.
<u>שנייה:</u> לצלם את האתר j ברזולוציה נמוכה – וההסבר דומה עבור s_j וזמן נותר של $t-3$ דקות.
<u>שלישית:</u> לא לצלם את האתר כלל, ואז ערך הפתרון זהה לשווי תוכנית צילום עבור $j-1$ האתרים הראשונים עם הזמן המקורי t, שכן לא צילמנו את האתר j.

(*iii*) במקרה זה ההסבר דומה להסבר של המקרה השני, אך כיוון שהזמן הנותר קטן מ-5, האופציות מצומצמות יותר.

(*iv*) במקרה זה ההסבר דומה להסבר של המקרה השני, אך כיוון שהזמן הנותר קטן מ-3, האופציות מצומצמות יותר.

נשתמש במערך דו-מימדי בגודל $(n+1)\times(T+1)$.
נאתחל: $M[0,t] = 0$ לכל $0 \le t \le T$.
צעד: עבור $t = 0$ עד T בצע:
עבור $j = 1$ עד n בצע:

$$\begin{aligned} &if\ t \ge 5: && M[j,t] = max\begin{Bmatrix} M[j-1,t-5]+h_j \\ M[j-1,t-3]+s_j \\ M[j-1,t] \end{Bmatrix} \\ &else\ if\ t \ge 3: && M[j,t] = max\begin{Bmatrix} M[j-1,t-3]+s_j \\ M[j-1,t] \end{Bmatrix} \\ &else: && M[j,t] = M[j-1,t] \\ &end. \end{aligned}$$

סיום: החזר את M (כאשר שווי תוכנית צילום אופטימאלית הינו $M[n,T]$).

זמן ריצה: אתחול: $O(T)$. ישנם $O(n\cdot T)$ צעדים ובכל צעד $O(1)$ פעולות. בסה"כ $O(n\cdot T)$.

שאלה 7 – סכום סידרה

<u>מופע:</u> זוג סדרות של מספרים שלמים חיוביים: $P=\{p_1,...,p_m\}$ ו $H=\{h_1,...,h_n\}$ כך ש $n \le m$.
ניתן להניח ש H ממוינת בסדר עולה.

<u>פתרון חוקי:</u> סדרה $\{i_1,i_2,...,i_n\} \subseteq [1,m]$ של אינדקסים של איברים ב P.

<u>יש למצוא:</u> פתרון חוקי $M=\{i_1,i_2,...,i_n\}$ כך ש $d(M)=\sum_{k=1}^{n} | p_{i_k} - h_k |$ מינימאלי.

סעיף א: הראו כי האלגוריתם החמדן הבא נכשל:

1. מיין את P בסדר עולה.
2. החזר $\{i_1,i_2,...,i_n\}$ האינדקסים של איברי P שמתאימים ל n האיברים הראשונים לפי המיון.

סעיף ד: הגדירו תת-בעיה אופיינית. כלומר הגדירו $OPT(i,j)$ כאשר i מייצג את P ו j את H. ציינו איזה ערך אנו מעוניינים לחשב.

סעיף ה: נסחו נוסחת נסיגה ותנאי בסיס עבור ערך של פתרון אופטימאלי עבור הבעיה לעיל.
<u>הדרכה:</u> עבור $OPT(i,j)$ שהגדרתם שקלו שני מקרים: $i \ge j$, ו $i < j$.

תשובה 7 – סכום סידרה

מופע: זוג סדרות של מספרים שלמים חיוביים: $P=\{p_1,\dots,p_m\}$ ו $H=\{h_1,\dots,h_n\}$ כך ש $n\le m$.
ניתן להניח ש H ממוינת בסדר עולה.
פתרון חוקי: סדרה $\{i_1,i_2,\dots,i_n\}\subseteq[1,m]$ של אינדקסים של איברים ב P.

יש למצוא: פתרון חוקי $M=\{i_1,i_2,\dots,i_n\}$ כך ש $d(M)=\sum_{k=1}^{n}|p_{i_k}-h_k|$ מינימאלי.

סעיף א: הראו כי האלגוריתם החמדן הבא נכשל:

1. מיין את P בסדר עולה.
2. החזר $\{i_1,i_2,\dots,i_n\}$ האינדקסים של איברי P שמתאימים ל n האיברים הראשונים לפי המיון.

סעיף ד: הגדירו תת-בעיה אופיינית. כלומר הגדירו $OPT(i,j)$ כאשר i מייצג את P ו j את H. ציינו איזה ערך אנו מעוניינים לחשב.

סעיף ה: נסחו נוסחת נסיגה ותנאי בסיס עבור ערך של פתרון אופטימאלי עבור הבעיה לעיל.
הדרכה: עבור $OPT(i,j)$ שהגדרתם שקלו שני מקרים: $i\ge j$, $i<j$.

סעיף א'

$$H=\{1,14,20\} \qquad P=\{1,2,3,15,16,17\}$$

פתרון חמדני: $|1-1|+|2-14|+|3-20|=0+12+17=29$

פתרון טוב יותר: $|1-1|+|15-14|+|17-20|=0+1+3=4$

סעיף ד'

הגדרת תתי הבעיות ו – OPT.

נגדיר $OPT(i,j)$ להיות ערך של פתרון אופטימאלי עבור $H^j=\{h_1,\dots,h_j\}$, $P^i=\{p_1,\dots,p_i\}$.
אנו מעוניינים ב – $OPT(m,n)$.

סעיף ה'

טענה(:מבנה של פתרון אופטימאלי)

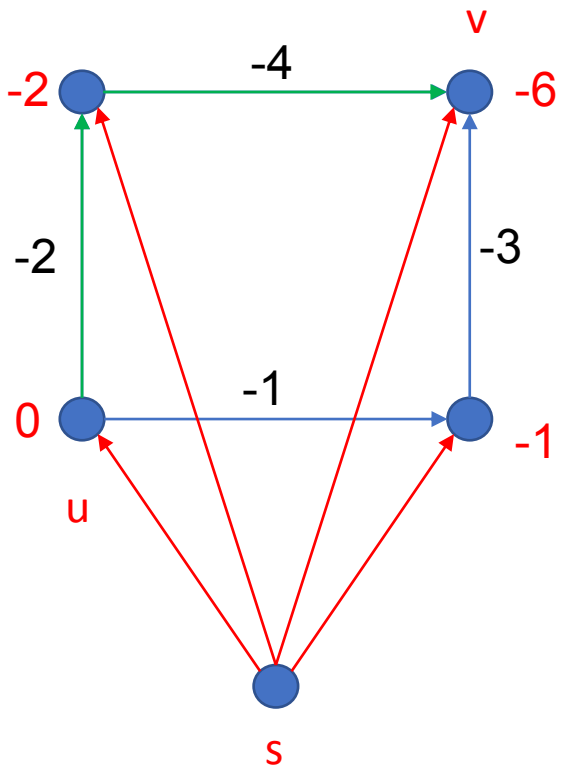
עבור $0\le i\le m$ ו $0\le j\le n$ -
1. אם $i<j$, $OPT(i,j)=\infty$
2. אחרת, אם $i\ge j$ אזי
   a. $OPT(i,0)=0$ לכל i.
   b. אם $j>0$
$$OPT(i,j)=min\{OPT(i-1,j),|p_i-h_j|+OPT(i-1,j-1)\}$$

שאלה 8 - מסלול קצר ביותר בגרף

נתון גרף מכוון עם משקלים כלשהם על הקשתות (חלקם חיוביים, חלקם שליליים) אך ללא מעגלים מכוונים שליליים. עומד לרשותכם אלגוריתם בלמן-פורד אשר יודע לחשב מסלולים קצרים ביותר בין צומת לבין שאר הצמתים בגרף גם כאשר המשקלות שליליים (חלקם או כולם.).
הציעו אלגוריתם יעיל למציאת אורך המסלול הקצר ביותר (כלומר בעל הערך הנמוך ביותר, ואם שלילי - השלילי ביותר) בין שני צמתים בגרף. כלומר על האלגוריתם למצוא את זוג הצמתים u ו-v בגרף אשר אורך המסלול ביניהם הינו קצר יותר מאשר כל מסלול בין שני צמתים אחרים כלשהם בגרף.

תשובה 8 - מסלול קצר ביותר בגרף

נתון גרף מכוון עם משקלים כלשהם על הקשתות (חלקם חיוביים, חלקם שליליים) אך ללא מעגלים מכוונים שליליים. עומד לרשותכם אלגוריתם בלמן-פורד אשר יודע לחשב מסלולים קצרים ביותר בין צומת לבין שאר הצמתים בגרף גם כאשר המשקלות שליליים (חלקם או כולם.)
הציעו אלגוריתם יעיל למציאת אורך המסלול הקצר ביותר (כלומר בעל הערך הנמוך ביותר, ואם שלילי - השלילי ביותר) בין שני צמתים בגרף. כלומר על האלגוריתם למצוא את זוג הצמתים u ו-v בגרף אשר אורך המסלול ביניהם הינו קצר יותר מאשר כל מסלול בין שני צמתים אחרים כלשהם בגרף.



נוסיף צומת חדש s לגרף ונחבר קשת עם משקל 0 ממנו לכל אחד מהצמתים בגרף המקורי.

נריץ את בלמן-פורד מהצומת החדש.

בין כל הצמתים בגרף המקורי נבחר את זה שמרחקו מ-s הינו הקצר ביותר, נקרא לו v.

ניקח את המסלול שבלמן-פורד מצא בין s לבין v. לצומת הראשון במסלול זה אחרי s נקרא u.

המסלול בין u לבין v בגרף המקורי הינו המסלול המבוקש, שאורכו הינו הקצר ביותר בין כל המסלולים בגרף המקורי. זאת כיון שהקשתות החדשות שהוספנו משקלן 0 והן אינן מוסיפות לאורכו של אף מסלול.