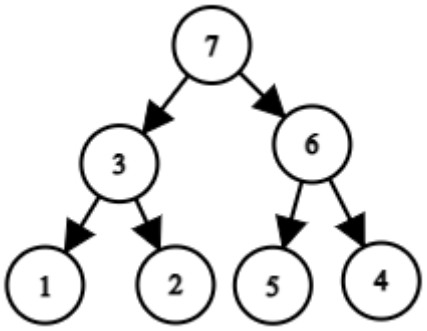
Permutation Transformation(Divide et Impera)

Enunț:

O permutare este o secvență de n numere întregi, de la 1 la ***n***, în care toate numerele apar exact o singură dată. De exemplu, [1], [3,5,2,1,4], [1,3,2] - sunt permutări, iar [2,3,2], [4,3,1], [0] - nu.

Recent, lui Polycarp i-a fost oferită o permutare ***a[1…n]*** de lungime ***n***. Polycarp preferă arborii în locul permutărilor, așa că dorește să transforme permutarea ***a*** într-un arbore binar. El transformă un tablou de numere întregi diferite într-un arbore în următorul mod:

- elementul maxim al tabloului devine rădăcina arborelui;
- toate elementele din stânga maximului formează un subarbore stâng (care este construit conform aceloraşi reguli aplicate la partea stângă a tabloului), dar dacă nu există elemente în stânga maximului, atunci rădăcina nu are fiu stâng;
- toate elementele din dreapta maximului formează un subarbore drept (care este construit conform aceloraşi reguli aplicate la partea dreaptă a tabloului), dar dacă nu există elemente în dreapta maximului, atunci rădăcina nu are fiu drept.



Un exemplu pentru permutarea ***a=[1,3,2,7,5,6,4]*** arată astfel:
Adâncimea vârfului reprezintă numărul de muchii de-a lungul cărora se poate ajunge de la rădăcină la vârf. Dată fiind o permutare, pentru fiecare vârf, găsiți adâncimea.

Input:

- Un număr întreg ***t (1≤t≤100)*** - numărul de cazuri de testare. Apoi urmează t cazuri de testare.

- Un număr întreg ***n (1≤n≤100)*** - lungimea permutării.

- n numere ***a1, a2, ..., an*** - elementele din ***a***.

Soluție:
Deși problema implica lucrul conceptual cu arbore binar, construcția acestuia poate fi evitată. Rezolvarea se bazează pe Divide et Impera:

- Se parcurge vectorul pentru a identifica valoarea maximă. Elementul găsit devine rădăcina arborelui, înălțimea acestuia este 0.

- Vectorul este divizat pe poziția rădăcinii, se obțin doi vectori cu elementele din stânga și din dreapta. Se identifică valoarea maximă din fiecare vector în parte, acestea devin noii copii în arbore (înălțimea 1). Vectorii sunt divizați la valorile găsite.

- Se repetă procedeul până când sunt identificate înălțimile tuturor elementelor.

Complexitatea: $O(n)$

Exemplu:
Se dă următoarea permutare: a = [3,5,2,1,4]

$ans[\,] <- [\,0,\ 0,\ 0\ldots]$ (*vectorul ce contine răspunsul*)
$v[\,] <- input\ (a)$

Funcția ajutătoare void find_depth(int l, int r, int cur) primește ca parametri limitele vectorului și adâncimea elementului curent. Apelul din main( ) se realizeaza cu limitele întregului vector și adâncimea 0 cu scopul de a identifica rădăcina arborelui find_depth(0, n-1, 0). Aceasta are următoarea funcționalitate :

- găsește elementul cu valoare maximă iterând prin vector și o stochează cu înălțimea 0:
  index_elem = (1, 5)
  answ[1] = 0

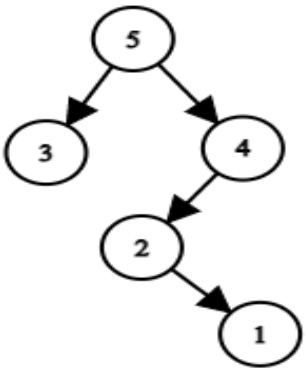
- se apelează funcția pentru partea stanga a vectorului find_depth(0, 0, 1),este satisfăcută condiția pentru cazul de baza, ans[0] = 1 și returnează.

- se apeleaza functia pentru partea dreaptă a vectorului
  find_depth(2, 4, 1),
  se găsește elementul maxim 4:
  index_elem = (4 , 4)
  ans[4] = 1

- se apelează funcția pentru partea stânga find_depth(2, 3, 2),
  se găsește elementul maxim 2:
  index_elem = (3 , 2)
  ans[3] = 2

- nu este satisfăcută condiția pentru apelul de stanga, se apelează find_depth(4, 4, 3) (la dreapta), se trece pe cazul de baza: ans[4] = 3 și returnează.

Obținem următorul vector ca răspuns: ans[ ] = [1, 0, 3, 2, 1] ce reprezintă înălțimile nodurilor din arborele:



Teleports (Greedy)

Enunț:

Se dau punctele ***0, 1, ..., n+1*** de-a lungul axei numerice. Există un portal aflat pe fiecare dintre punctele ***1, 2, ..., n***. La punctul ***i***, poți face următoarele:

- Să te deplasezi la stânga cu o unitate: costă 1 monedă.
- Să te deplasezi la dreapta cu o unitate: costă 1 monedă.
- Să folosești portalul de la punctul i, dacă există: costă a(i) monede. În urma acestei acțiuni, poți alege să te teleportezi la punctul 0 sau la punctul n+1. Odată ce ai folosit un portal, nu-l mai poți folosi.

Ai la dispoziție ***c*** monede și începi de la punctul 0. Care este numărul maxim de teleportoare pe care le poți folosi?

Input:

- Un număr întreg t($1 \le t \le 1000$) - numărul de cazuri de testare. Urmează descrierea cazurilor de testare.

- Două numere întregi n și c ($1 \le n \le 2 \cdot 105$; $1 \le c \le 109$) - lungimea tabloului și numărul de monede pe care le aveți.

- n numere întregi pozitive separate prin spațiu a1, a2, ..., an ($1 \le a(i) \le 109$) - costurile utilizării portalurilor.

Soluție:

- Se observa că este optim să folosim portalul odată ce am ajuns la el, indiferent de direcția de deplasare.

- Atunci când ne teleportăm, problema este independentă de alegerile anterioare. Ajungem la 0 sau n + 1 și avem câteva portaluri disponibile. Prin urmare, putem afla costul individual al fiecărui portal (pentru portalului i este min(a(i) + i, a(i) + n + 1 − i) deoarece putem veni la un portal de la punctul 0 sau de la n + 1, plătim a(i) pentru a-l folosi și avem nevoie de i deplasări pentru a ajunge la el).

- Se sortează portalurile după costul individual de la cel mai mic la cel mai mare, deplasarea spre primul începe la punctul 0.

- Pentru a determina prima teleportare se itereaza prin toate portalurile și se verifica numărul maxim de portaluri care pot fi utilizate unei secvențe, folosind sumele de prefixe peste matricea de costuri minime și căutarea binară. Se

verifica și cazul în care portalul ales este inclus atât ca inițial și în prefixul de cost minim.

Complexitatea: $O(n^2)$

Exemplu:
Input:

```
1
5 9
2 3 1 4 1
```

- Se dau 5 portale, 9 monede și costurile de teleportare].
  tp[ ] = [2, 3, 1, 4, 1]

- Se calculează costul total de teleportare și se sortează crescător obținem vectorul total_costs[ ] = [0, 2, 3, 4, 5, 6] (0 pentru indexare, poate fi ignorat)

- Se determina suma cu prefix a elementelor din vector:
  s[ ] = [0, 2, 5, 9, 14, 19].

- Pentru portalul 1 ales primul obținem:
  t = 3
  k = 3
  total_costs[3] = 4 > t (portalul a fost inclus în suma prefixelor)
  k = 3
  ans = 3

- Procedăm la fel pentru portalurile rămase și obținem următoarele nr. de teleportari:

portal(i):	nr teleportari:
1	3
2	2
3	3
4	1
5	2

Nr. maxim de teleportări este 3, utilizând portalul 1 sau 4 primul.

Longest Bitonic Subsequence (Programare Dinamica)

Enunț:
LBT este cea mai lungă subsecvență în care elementele sunt mai întâi sortate în ordine crescătoare, apoi în ordine descrescătoare.De exemplu, cea mai lungă subsecvență bitonică ***a [3, 1, 4, 7, 6, 10, 9, 5, 8, 2]*** este ***[3, 4, 7, 10, 9, 8, 2***]. Sa se afle lungimea LBT pentru un vector ***v*** dat.

Input:
Vectorul ***nums[ ]*** dat ca parametru.

Soluție:
Idea constă în a menține doi vectori, I[ ] și D[ ]:

- I[i] stochează lungimea celei mai lungi subsecvențe crescătoare, care se termină la nums[i].
- D[i] stochează lungimea celei mai lungi subsecvențe descrescătoare, care începe de la nums[i].
- Lungimea celei mai lungi subsecvențe bitonice este maximă dintre toate I[i] + D[i] - 1.

Complexitatea: $O(n^2)$

Exemplu:
nums = [4, 2, 5, 9, 7, 6, 10, 3, 1]
Obținem următoarele valori pentru tablouri:

	I[i]	D[i]
(i = 0)	1	3
(i = 1)	1	2
(i = 2)	2	3
(i = 3)	**3**	**5**
(i = 4)	3	4
(i = 5)	3	3
(i = 6)	4	3
(i = 7)	2	3
(i = 8)	1	1

Calculăm I[i] + D[i] - 1 pentru fiecare index i, cea mai lungă secvența se obține pentru i = 3 cu lungimea 7.

Find the longest possible route in a matrix (Backtracking)

Enunț:
Având o matrice binară, găsiți cea mai lungă rută posibilă de la sursă la destinație prin parcurgerea pozițiilor adiacente non-zero, adică putem forma ruta din poziții care au valoarea lor ca fiind 1. Nu trebuie să existe niciun ciclu în calea de ieșire, parcurgerea se realizeaza pe direcțiile sus, jos, stanga, dreapta. Întoarceți lungimea rutei găsite.

Input:

- Matricea ***mat[ ]*** dată ca parametru, poziția de început ***src*** și poziția destinație ***dest***

Soluție:
Putem folosi backtracking pentru a rezolva această problemă.

- Începem de la celula sursă dată în matrice și explorăm toate cele patru căi posibile și verificăm recursiv dacă vor duce sau nu la destinație.

- Trebuie să ținem evidența distanței celulei curente față de sursă și să actualizăm valoarea celei mai lungi căi găsite până acum atunci când ajungem la celula destinație.

- Dacă o cale nu ajunge la destinație sau a explorat toate rutele posibile din celula curentă, facem backtracking. Pentru a ne asigura că calea este simplă și nu conține cicluri, ținem evidența celulelor implicate în calea curentă într-o matrice și, înainte de a explora orice celulă, ignorăm celula dacă este deja acoperită în calea curentă.

Complexitatea: $O(4^{X \cdot Y})$ - unde X și Y sunt dimensiunile matricei.

Exemplu:

```
Input:
mat =
    {
        { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
        { 1, 0, 1, 0, 1, 1, 1, 0, 1, 1 },
        { 1, 1, 1, 0, 1, 1, 0, 1, 0, 1 },
        { 0, 0, 0, 0, 1, 0, 0, 1, 0, 0 },
        { 1, 0, 0, 0, 1, 1, 1, 1, 1, 1 },
        { 1, 1, 1, 1, 1, 1, 1, 1, 1, 0 },
        { 1, 0, 0, 0, 1, 0, 0, 1, 0, 1 },
        { 1, 0, 1, 1, 1, 1, 0, 0, 1, 1 },
        { 1, 1, 0, 0, 1, 0, 0, 0, 0, 1 },
        { 1, 0, 1, 1, 1, 1, 0, 1, 0, 0 }
    };
src = (0, 0)
dest = (5, 7)
```

- Programul începe din colțul stânga sus al matricii. Se apelează recursiv modificand x și y, simuland mișcarea în una din direcții - dacă poziția următoare este diferită de 0 și se încadrează în limitele matricii.
- Evitarea ciclurilor se asigură prin schimbarea valorii elementelor vizitate în 0.
- Dacă se ajunge la destinație se memorează distanța parcursa.
- Prin apelul recursiv se parcurg toate rutele posibile, în final fiind memorată valoarea cea mai mare a distanței.

Astfel pentru matricea dată se obține răspunsul 22 și următoarea cale:

	0	1	2	3	4	5	6	7	8	9
0	1	0	1	1	1	1	0	1	1	1
1	1	0	1	0	1	1	1	0	1	1
2	1	1	1	0	1	1	0	1	0	1
3	0	0	0	0	1	0	0	1	0	0
4	1	0	0	0	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	0
6	1	0	0	0	1	0	0	1	0	1
7	1	0	1	1	1	1	0	0	1	1
8	1	1	0	0	1	0	0	0	0	1
9	1	0	1	1	1	1	0	1	0	0

Referințe:

https://codeforces.com/problemset/problem/1490/D
https://codeforces.com/contest/1791/problem/G2
https://www.geeksforgeeks.org/longest-bitonic-subsequence-dp-15/
https://www.geeksforgeeks.org/longest-possible-route-in-a-matrix-with-hurdles/