Általános alapelvek. A pontozási útmutató célja, hogy a javítók a dolgozatokat egységesen értékeljék. Ezért az útmutató minden feladat (legalább egy lehetséges) megoldásának főbb gondolatait és az ezekhez rendelt részpontszámokat közli. Az útmutatónak nem célja a feladatok teljes értékű megoldásának részletes leírása; a leírt lépések egy maximális pontszámot érő megoldás vázlatának tekinthetők. Az útmutatóban feltüntetett részpontszámok csak akkor járnak a megoldónak, ha a kapcsolódó gondolat egy áttekinthető, világosan leírt és megindokolt megoldás egy lépéseként szerepel a dolgozatban. Így például az anyagban szereplő ismeretek, definíciók, tételek puszta leírása azok alkalmazása nélkül nem ér pontot (még akkor sem, ha egyébként valamelyik leírt tény a megoldásban valóban szerephez jut). Annak mérlegelése, hogy az útmutatóban feltüntetett pontszám a fentiek figyelembevételével a megoldónak (részben vagy egészében) jár-e, teljes mértékben a javító hatásköre. Részpontszám jár minden olyan ötletért, részmegoldásért, amelyből a dolgozatban leírt gondolatmenet alkalmas kiegészítésével a feladat hibátlan megoldása volna kapható. Ha egy megoldó egy feladatra több, egymástól lényegesen különböző megoldást is elkezd, akkor legfőljebb az egyikre adható pontszám. Ha mindegyik leírt megoldás vagy megoldásrészlet helyes vagy helyessé kiegészíthető, akkor a legtöbb részpontot érő megoldáskezdeményt értékeljük. Ha azonban több megoldási kísérlet között van helyes és (lényeges) hibát tartalmazó is, továbbá a dolgozatból nem derül ki, hogy a megoldó melyiket tartotta helyesnek, akkor a kevesebb pontot érő megoldáskezdeményt értékeljük (akkor is, ha ez a pontszám 0). Az útmutatóban szereplő részpontszámok szükség esetén tovább is oszthatók. Az útmutatóban leírttól eltérő jó megoldás természetesen maximális pontot ér, de bizonyítás nélkül csak az előadáson szereplő tételekre és állításokra lehet hivatkozni.

1. Legyen
$$f(n) = n^3 + 5n\log_2^2 n, \qquad g(n) = n^3 \log_2 n.$$
Igazoljuk vagy cáfoljuk, hogy $f(n) = O(g(n))$.

Megoldás: A reláció igaz, azaz $n^3 + 5n\log_2^2 n \in O(n^3 \log_2 n)$.

Ehhez belátjuk, hogy létezik olyan c valós szám és n_0 egész szám, hogy $n \geq n_0$ esetén: **2 pont**
$$f(n) = n^3 + 5n\log_2^2 n \leq c \cdot n^3 \log_2 n = c \cdot g(n).$$
Azt fogjuk belátni, hogy $c = 6$ és $n_0 = 2$ kielégíti a fenti feltételt. **2 pont**
Ezután nézzük a következő egyenlőtlenségsort:
$$n^3 + 5n\log_2^2 n \leq 6n^3 \leq 6n^3 \log_2 n.$$
2+2 pont
Az első egyenlőtlenség azért igaz, mert $n \geq 2$ esetén $\log_2 n \leq n$, **1 pont**
míg a második azért, mert $n \geq 2$ esetén $\log_2 n \geq 1$. **1 pont**

2. Egy vállalat 6 karakter hosszú termékazonosítókat használ. Az azonosítók kizárólag a következő 8 nagybetűből állhatnak:
`O, Z, A, B, E, M, S, T`
A vállalatnál ezekre a karakterekre nem a szokásos ABC sorrend érvényes, hanem egy speciális, fontossági sorrend:
`O < Z < A < B < E < M < S < T`
Adott pontosan 8 darab, ilyen szabály szerint képzett, 6 karakter hosszú termékazonosító:
`[BASTET, AMETSA, ZEMOBE, SOMBTA, AZAZAZ, MESOBE, TAMTOM, ZEBESA]`
Rendezzük a fontossági sorrendet használva lexikografikus sorrendbe ezeket az azonosítókat a Radix rendezés segítségével. A megoldást részletesen fejtsük ki, azaz minden fázis után írjuk le a termékazonosítók aktuális sorrendjét.

Pontozás Ha tudja, hogy radix rendezésben oszloponként (karakterenként) kell rendezni. **4 pont**
Ha tudja, hogy hátulról előre kell. **2 pont**
Helyes számolás. **4 pont**
Ha sima ABC sorrendjét alkalmazza, max. **8 pont**

Megoldás: A radix rendezést a jobb szélső (6. karakter) pozíciótól kezdjük:
A karakterek rendezett sorrendje (a kisebb értékű előrébb kerül) a 6. karakter utáni rendezés után:
`AZAZAZ, AMETSA, SOMBTA, ZEBESA, ZEMOBE, MESOBE, TAMTOM, BASTET.`

5. karakter utáni rendezés végén:
`TAMTOM, AZAZAZ, ZEMOBE, MESOBE, BASTET, AMETSA, ZEBESA, SOMBTA.`

4. karakter utáni rendezés végén:
`ZEMOBE, MESOBE, AZAZAZ, SOMBTA, ZEBESA, TAMTOM, BASTET, AMETSA.`

3. karakter utáni rendezés végén:

`AZAZAZ, ZEBESA, AMETSA, ZEMOBE, SOMBTA, TAMTOM, MESOBE, BASTET.`

2. karakter utáni rendezés végén:

`SOMBTA, AZAZAZ, TAMTOM, BASTET, ZEBESA, ZEMOBE, MESOBE, AMETSA.`

1. karakter utáni rendezés végén:

`ZEBESA, ZEMOBE, AZAZAZ, AMETSA, BASTET, MESOBE, SOMBTA, TAMTOM.`

3. Adott egy fekete-fehér, pixeles kép, amelyen egymás melletti m darab épület látható. A kép egy $n \times m$ méretű mátrixként van reprezentálva. A mátrix minden eleme 0 vagy 1 (, ahol az 1 az épületeket, a 0 az égboltot jelenti)

   - Az oszlopok az épületeket reprezentálják, soronként felfelé haladva az épületek aljától a tetejük felé.
   - Minden oszlopban az 1-esek alulról kezdődnek és egybefüggőek: azaz minden oszlopban először szerepelhet néhány 1, majd egy összefüggő 0-s szakasz a kép tetejéig (de 0 után már nem lehet újra 1).

   A feladat az, hogy határozzuk meg annak az oszlopnak az indexét, amelyben a legtöbb egymás felett álló 1-es található (azaz a legmagasabb épület helyét). Ha több ilyen oszlop is van, bármelyik helyes választnak tekinthető.

   Adjunk meg egy algoritmust, amely legfeljebb $O(m \log n)$ mezőt vizsgál meg a mátrixban, és meghatározza a legmagasabb épületet.

   Pontozás:
   jó algoritmus **6 pont**
   indoklás **2 pont**
   lépésszám **2 pont**
   Megjegyzés: Létezik $O(m+n)$ lépésszámú algoritmus is, ami persze szintén elfogadható.

   Megoldás: Ha az adott oszlopban az első 1-es a r-edik sorban (1-gyel kezdve a sorok indexelését) található, akkor az oszlop "épületmagassága" $n - r + 1$.

   Algoritmus: Egy bináris keresés típusú algoritmussal tudjuk megoldani a feladatot.

   1. Menjünk végig minden oszlopon (m db).
   2. **Bináris keresésszerű algoritmus:** Az aktuális oszlopra alkalmazzuk a következőt a sorindexek tartományán $[0, n-1]$:
      - Állítsuk be $\texttt{alsó} = 1$ és $\texttt{felső} = n$.
      - Ismételjük, míg $\texttt{alsó} < \texttt{felső}$:
        - Számoljuk ki a $\texttt{középső} = \lceil (\texttt{alsó} + \texttt{felső})/2 \rceil$ indexet.
        - Ha $A[\texttt{középső}][c] = 1$, akkor $\texttt{felső} = \texttt{középső}$; különben $\texttt{alsó} = \texttt{középső} + 1$.
      - A végén `alsó` jelöli az első 1-es helyét (vagy n, ha nincs 1-es).
   3. **Magasság számítása:** Az oszlop magassága $h = n - \texttt{alsó} + 1$.
   4. **Maximum keresése:** Válasszuk ki a tanult maximum kereső algoritmussal azt az oszlopot, amelyiknél h maximális.

   Helyesség és futási idő: Az oszlopok értékei monoton nőnek (0-k, majd 1-esek), így a bináris keresésszerű algoritmus garantáltan jól működik és legfeljebb $O(\lceil \log_2 n \rceil)$ mezőt vizsgál meg. Ezt m oszlopon futtatjuk, majd a kapott m érték közül választjuk ki a maximumot, de ehhez már nem kell újabb mezőket megvizsgálni. Tehát az algoritmus összesen $O(m \log n)$ mezőt vizsgál meg.

4. Az alábbi irányított gráfokat, G_1-et és G_2-t szomszédsági listáik írják le. Futtassunk mélységi keresést G_1-en és G_2-n és számítsuk ki minden csúcs elérési és befejezési számát, majd az órán tanult módszer használatával döntsük el, hogy melyik gráf tartalmaz irányított kört, azaz melyik nem irányított aciklikus gráf, és adjunk meg egy topologikus rendezést abban a gráfban, amelyben ez lehetséges.

   - G_1: **a**: b, c; **b**: d; **c**: d; **d**: e; **e**: a.
   - G_2: **a**: g, f; **b**: a, g; **c**: - ; **d**: - ; **e**: c, d; **f**: e; **g**: f, e.

   Megoldás:

G_1

Csúcs	Mélységi	Befejezési
a	1	5
b	2	3
c	5	4
d	3	2
e	4	1

3 pont

Egy él $x \to y$ visszaél, ha a `mélységi szám`$(x) >$ `mélységi szám`(y) és `befejezési szám`$(x) <$ `befejezési szám`(y), így G_1-ben az $e \to a$ él visszaél, azaz G_1 nem DAG. **1 pont**

G_2

Csúcs	Mélységi	Befejezési
a	1	6
g	2	5
f	3	4
e	4	3
c	5	1
d	6	2
b	7	7

3 pont

Mivel G_2-ben nincs visszaél, ez egy DAG. **1 pont**
A topologikus sorrendet a befejezési számok szerinti csökkenő sorrend adja. **1 pont**
Topologikus sorrend: b, a, g, f, e, d, c. **1 pont**

5. Adott egy irányított, élsúlyozott gráf $G = (V, E)$, amely Algoritmisztán városait és közúti útvonalait reprezentálja. Minden él $e \in E$ két várost köt össze egy adott irányban, és súlya $w(e)$ azt az időt jelenti, amely az adott irányú utazáshoz szükséges. A gráf éllistában van megadva.

   Két barát, akik jelenleg Algoritmisztán két különböző városában tartózkodnak (jelölje ezeket A és B), találkozót szeretne szervezni egy harmadik algoritmisztáni városban. A találkozás feltételei a következők:

   - Mindketten ugyanabban az időpillanatban indulnak el,
   - mindketten a gráf élei mentén haladhatnak csak, az él által adott irányban, az élhez tartozó idő alatt és a városokban nem kell várniuk az utazás során (pl. kocsival mennek),
   - nem hagyják el Algoritmisztán területét, azaz a gráf csúcsai között kell mozogniuk.

   A cél az, hogy megtaláljuk azt a várost $C \in V$, ahol a két barát a *leghamarabb találkozhat* (azaz a közös indulástól a találkozásig eltelt idő a legkevesebb). Nem kell, hogy egyszerre érjenek oda, az egyik barát várhat a másikra. Ha több ilyen város is van, bármelyik elfogadható megoldás.

   Adjunk meg egy algoritmust, amely legfeljebb $O((m + n) \log n)$ idő alatt meghatároz egy ilyen találkozási helyet, ahol n a gráfban lévő csúcsok számát jelenti.

   Megoldás:

   Algoritmus:

   I. Futtassuk Dijkstra algoritmusát a gráfban G az A kezdőpontból, hogy minden $v \in V$ esetén kiszámoljuk $d_A(v)$, azaz az A-ból v-be vezető legrövidebb (idejű) út hosszát. **2 pont**

   II. Futtassuk Dijkstra algoritmusát a B kezdőpontból, így meghatározzuk $d_B(v)$ értékeit. **1 pont**

   III. Minden $v \in V$ esetén számoljuk ki $T(v) = \max\{d_A(v), d_B(v)\}$. **2 pont**

   IV. Válasszuk azt a $C \in V$-t, amelyre $T(C)$ minimális, és adjuk vissza ezt C-t. **1 pont**

   Indoklások:
   - Miért a Dijkstra?
   - $T(v)$ indoklása **2 pont**

   Lépésszám:
   A Dijkstra algoritmus szomszédsági mátrix esetén $O((m+n) \log n)$ időben fut, és mivel kétszer hajtjuk végre (egyszer A-ból, egyszer B-ből), az össz idő $O((m + n) \log n)$. A $T(v)$ értékek számítása és a minimum kiválasztása $O(n)$ időt vesz igénybe, így a teljes algoritmus futási ideje $O((m + n) \log n)$. **2 pont**

6. Egy hosszú autóútra készülünk, amelyet a 0 kilométernél kezdünk meg. Az út mentén n darab szálloda található, amelyek a következő, növekvő sorrendben megadott kilométerpontokon helyezkednek el:

$$0 < a_1 < a_2 < \cdots < a_n.$$

(Itt a_i az indulási ponttól mért távolságot jelöli kilométerben, $i = 1, 2, \ldots, n$. A számok nem feltétlen egészek.) Az út során csak ezek közül néhány kiválasztott helyen állunk meg, máshol nem, azonban az utolsó szállodánál (a_n) kötelezően meg kell állnunk, mivel az a célállomásunk. Ideális esetben naponta 350 kilométert szeretnénk utazni. A tényleges napi megtett távolság $x \geq 0$ esetén az adott naphoz tartozó *büntetés* a következőképpen alakul:

$$(350 - x)^4.$$

(**Fontos**: az x lehet kisebb és nagyobb is 350-nél; a képlet mindkét esetet bünteti, mivel az abszolút eltérés negyedik hatványát méri.)

Adjunk olyan $O(n^2)$ futásidejű algoritmust, amely meghatározza, hogy mely szállodákban érdemes megállni úgy, hogy az összes napra eső büntetések összege minimális legyen, míg elérjük a célállomást! (Például a 0, 300, 420, 770 esetben az optimális megállóhelyek a 420 és 770.)

Megoldás:

`Első megoldás`

A következő élsúlyozott gráffal reprezentáljuk a szállodákat és a köztük lévő utazásokat:

Az út minden lehetséges megállási pontját (szállodák) csúcsokként értelmezzük, tehát a csúcsok halmaza

$$V = \{0, 1, 2, \ldots, n\},$$

ahol i a kezdőponttól a_i-re lévő szállodát jelenti. Minden olyan (i,j) csúcs-párra hozzuk létre az él súlyát

$$w(i,j) = \big(350 - \big(a_j - a_i\big)\big)^4.$$

Ez a súly a két megálló közötti utazás büntetését méri (bármely irányban).

Ezzel a gráf reprezentációval a cél az, hogy megtaláljuk a kezdőcsúcstól (0) a célcsúcsig (n) vezető olyan utat, melyen az élsúlyak összege minimális.

Algoritmus **5 pont**

A megoldás során alkalmazzuk a Dijkstra algoritmust az alábbi módon:

1. Futtassunk Dijkstra-t 0-ból (azt a verziót, ahol tömbben tároljuk D-t) és nézzük meg a legrövidebb távolságot n-be.
2. A Dijkstra algoritmus "előző" tömbjét használva rekonstruáljuk visszafelé menve az optimális útvonalat a 0 csúcstól a célcsúcsig (n).

Helyesség **2 pont**

Mivel az élek súlyfüggvénye $(350 - (a_j - a_i))^4$ nem negatív értékeket vesz fel, a Dijkstra algoritmus alkalmazása helyes.

Futásidő **3 pont**

Legyen a csúcsok száma $N = n + 1$. A gráfban a lehetséges élek száma legfeljebb

$$\frac{N(N-1)}{2} = O(n^2),$$

így a gráf szomszédossági mátrixának megkonstruálása $O(n^2)$. Mivel a mátrixos implementációt alkalmazzuk, így a Dijkstra futásideje (az út rekontruálásával együtt) $O(n^2)$. Így az egész futásidő $O(n^2)$.

`Második megoldás` - bár ez csak akkor használható, ha **csak** előrefelé megyünk, elfogadhatónak tekintjük.

Algoritmus: Legyen $a_0 = 0$, és definiáljuk a `Bünt` tömböt (részfeladatokat), ahol `Bünt`$[j]$ a minimális összes büntetés, hogy eljussunk a j. szállodáig ($j = 0, 1, \ldots, n$). **3 pont**

A kezdet és a továbblépés képlete:

$$\texttt{Bünt}[0] = 0, \qquad \texttt{Bünt}[j] = \min_{0 \leq i < j} \Big\{ \texttt{Bünt}[i] + \Big(350 - (a_j - a_i)\Big)^4 \Big\}, \qquad j = 1, \ldots, n.$$

indoklással: **4 pont**

Egy `előző` tömb segítségével nyomon a szokásos módon követjük, melyik i-nél vette fel a minimumot. **1 pont**

A végén `Bünt`$[n]$ adja a minimális büntetés mértékét és az `előző` tömbben visszalépkedve megkapjuk a konkrét szállodákat. **1 pont**

Lépésszám: Minden j ($1 \leq j \leq n$) esetén az összes i ($0 \leq i < j$) lehetőséget végigvizsgáljuk, így egy konkrét j-re `Bünt`$[j]$ kiszámításának a futásideje $O(n)$, az algoritmus össz futásideje $O(n^2)$. **1 pont**

7. Adott éllistával egy irányított gráf. Adjunk $O(m+n)$ futásidejű algoritmust, ami eldönti, hogy van-e olyan csúcs a gráfban, ahonnan minden csúcs elérhető irányított úton. (n - szokás szerint - a gráfban lévő csúcsok számát, m pedig az élek számát jelöli.)

[*Segítség: próbáljuk meg a feladatot először egy DAG-ra megoldani.*]

Megoldás:
Algoritmus:

(a) Futtassunk egy DFS-t az egész gráfon. Az utolsóként befejezett csúcsot nevezzük `esetleg`-nek.

(b) Indítsunk egy újabb DFS-t az `esetleg` csúcsból.

(c) Amennyiben a második DFS minden csúcsot elér, akkor `esetleg` egy olyan csúcs, ahonnan a gráf összes csúcsa elérhető. Egyébként nincs ilyen csúcs. **4 pont**

Helyesség:
Használjuk azt a tanult tételt, hogy a $DFS(G,v)$ pontosan azokat a csúcsokat éri el, ahova irányított út van v-ből és még nem lettek felfedezve.

Ha DFS-t többször kell „újrakezdeni", akkor az utolsó újrakezdésnél elért pontokba biztosan nem vezet irányított út az előzőleg elértekből a fenti tétel szerint. Így csak az utolsó "elkezdésnél" elért csúcsok lehetnek esélyesek, hogy van belőlük irányított út a többiekhez. Ha ezen pontok közül bármely w-ből van irányított út az összes többibe, akkor `esetleg`-ből is van, hiszen `esetleg`-ből van w-be (mivel `esetleg` befejezési száma a legnagyobb, az utolsó „újrakezdés" `esetleg`-ből történt) és összefűzzük w-ből más csúcsokba menő utakkal, így lesz egy irányított élsorozat (tehát út is). **5 pont**

Lépésszám:
Az első DFS bejárása $O(n+m)$ időt vesz igénybe, a második DFS szintén $O(n+m)$ időben fut, így az algoritmus teljes futásideje $O(n+m)$. **1 pont**