ADASALES – Ada és utazó értékesítő / ADASALES - Ada and Travelling Salesman

Probléma összefoglalása:

A probléma az, hogy megtaláljuk azt a maximális profitot, amelyet Ada a katicabogár képes megkeresni, mint utazó értékesítő (TSP[1]), aki városok között utazik. A városok egy fa szerkezetben kapcsolódnak össze, ahol minden városhoz sajátos ár tartozik. Ada egyszerre csak egy terméket tud szállítani, és az a célja, hogy egy olyan városba utazzon, ahol egy adott városból indulva a lehető legnagyobb profitot érheti el. A városokat közvetlen utak kötik össze, Ada minden lekérdezéshez egy várost kap kiindulópontként, a feladat pedig az, hogy meghatározzuk, mekkora profitot tud keresni egy másik városba utazva.

A probléma részletei:

Ada az egyik városban tartózkodik, és szeretné tudni, hogy mekkora árat kaphat egy másik városba vezető úton.

A városokat egy fa köti össze, ami azt jelenti, hogy bármely két város között pontosan egy út vezet.

Minden városnak fix ára van.

A probléma több olyan lekérdezéssel jár, ahol Ada különböző városokban indul, és ki kell számolnunk azt a maximális árat, amelyet az úton bármely másik városba utazva kereshet.

Megoldás megközelítése:

A megoldás a Depth-First Search[2] (DFS) segítségével járja be a fát, és keresi meg azt a maximális árat, amelyet Ada kaphat, ha minden kiindulási pontról egy másik városba utazik. Íme a megközelítés lebontása:

Faábrázolás: A városok és kapcsolataik egy fát alkotnak, amelyet szomszédsági lista segítségével ábrázolhatunk.

DFS-bejárás: DFS-t használunk a fa bejárására, és kiszámítjuk az adott városból elérhető maximális árat. A DFS során fenntartjuk a várostól a leszármazott városokig vezető útvonalon található maximális árat.

Lekérdezés megválaszolása: Miután a DFS-bejárás kiszámítja az összes város maximális árait, az egyes lekérdezések megválaszolása állandó idejű műveletté válik, mivel az eredmények előre kiszámításra kerülnek.

Megoldás lépésről lépésre:

1. <u>Bemenet elemzése és inicializálása:</u>

Először a városok számát és a lekérdezések számát olvassuk le.

Ezután kiolvassuk az egyes városok árait, és eltároljuk egy tömbben.

A fát (gráfot) úgy építjük fel, hogy kiolvassuk a városok közötti kapcsolatokat, amely egy szomszédsági listát ad.

[1] https://en.wikipedia.org/wiki/Travelling_salesman_problem
[2] https://en.wikipedia.org/wiki/Depth-first_search

2. DFS-bejárás a maximális nyereség kiszámításához:

Egy tetszőleges csomópontból (általában a 0-s csomópontból) kiindulva végzünk egy mélységi keresést (DFS), hogy kiszámítsuk azt a maximális árat, amelyet minden városból kiindulva bármely másik városba utazva elérhetünk.

A dfs függvény a következőképpen működik:

- Inicializálja a max_profit[node] értéket az aktuális város árára (prices[node]).
- Ezután meglátogatja az aktuális csomópont minden gyermekét (szomszédját). Ha a gyermeket nem látogattuk meg (még nem dolgoztuk fel), akkor rekurzív módon meghívja a dfs függvényt a gyermeken.
- Az összes szomszéd meglátogatása után frissítjük a max_profit[node] értéket az aktuális csomóponttól és gyermekeitől származó maximális nyereségre.

Ez biztosítja, hogy minden csomópont a lehető legjobb profitot tárolja, amelyet bármelyik leszármazottjához való utazással elérhet.

3. Lekérdezés feldolgozása:

A DFS-bejárás után kiszámítottuk a max_profit[] tömbben lévő városok maximális nyereségét.

Minden olyan lekérdezéshez, ahol a város meg van adva, azonnal visszaadhatjuk a max_profit[város] értékét, mivel az adott város maximális nyereségét előre kiszámítottuk a DFS-bejárás során.

4. Több lekérdezés hatékony megválaszolása:

Mivel minden lekérdezés egyszerűen csak a max_profit[város] értékét kéri, minden lekérdezés megválaszolása O(1) állandó időben történik.

Ez rendkívül hatékony, különösen nagy számú lekérdezés esetén (akár 5×1055×105).

Dinamikus programozási megközelítés:

Átfedő részproblémák: A probléma több lekérdezéssel jár, amelyek a maximális profitot kérik különböző városokból kiindulva. Ahelyett, hogy minden egyes lekérdezésnél külön-külön újraszámolnánk a maximális nyereséget, egyszer megoldjuk a problémát az összes város esetében a DFS használatával, és eltároljuk az eredményeket.

Optimális alstruktúra: Egy városból származó maximális profit a gyermekei maximális profitjától függ. A DFS használatával egyesítjük az alproblémák (gyermekvárosok) eredményeit, hogy megoldjuk a szülőváros problémáját. Ez a dinamikus programozás klasszikus jellemzője: kisebb részproblémák megoldása, eredményeik felhasználása nagyobb problémák megoldására.

Memorizáció: Ebben az esetben a DFS bejárás eredményei a max_profit[] tömbben tárolódnak, hatékonyan megjegyzik az egyes városok maximális profitját. Ez lehetővé teszi, hogy minden kérdésre állandó időben válaszoljunk.

Miért illik ez a megoldás a dinamikus programozáshoz:

Dinamikus programozási alapelvek:

Memorizáció: A DFS-bejárás során eltároljuk az egyes városok maximális hasznát, hogy az újraszámítás nélkül újra felhasználható legyen a következő lekérdezésekben.

Optimális alstruktúra: A probléma megoldása (maximális profit egy városból) a részproblémák megoldásától függ (maximális haszon gyermekvárosaiból).

Átfedő részproblémák: Az egyes városok maximális nyereségét egyszer számítják ki és tárolják. Ahelyett, hogy minden lekérdezéshez újraszámítanánk, az előre kiszámított értékeket közvetlenül használjuk fel.

Hatékonyság:

A megoldás O(N) időt használ a DFS bejáráshoz, ahol N a városok száma. Ez a probléma megoldásának összetettsége minden város számára.

Minden lekérdezésre O(1) idő alatt válaszolunk, egyszerűen megkeresve az előre kiszámított eredményt a max_profit[] tömbben.

Így a teljes folyamat teljes időbonyolultsága O(N + Q), ahol N a városok száma, Q pedig a lekérdezések száma, így még nagy bemenetek esetén is hatékony.