Kruskal (minimális feszítőfa): igazán különleges részfa - A megoldás magyarázata

A feladat megoldását a `kruskalmstrsub.py` szkript fájl tartalmazza. Az eredményt a `kruskals` függvény szolgáltatja, amely segédfüggvényként felhasználja a `halmazt_keres` függvényt.

Először a `kruskals` függvényt tárgyaljuk, amely Kruskal algoritmusának egy módosított változata.

Első lépésben beállítjuk, hogy a kiinduló teljes súly nulla legyen az alábbiak szerint:

```
# beallitjuk, hogy a kezdo teljes suly nulla legyen
teljes_suly = 0
```

Ezután a következő előkészítő lépésben a bemenetként kapott gráf minden egyes csúcsát eltároljuk egy-egy halmazba a következők szerint:

```
# a bemenetkent kapott graf minden egyes csucsat eltaroljuk
# egy halmazba
halmazok = []
for i in range(1, g_nodes + 1):
    halmazok.append({i})
```

Majd az élekre vonatkozó információkat átalakítjuk, hogy tudjuk alkalmazni rá Kruskal algoritmusát. Ehhez az éleket egy `dict`-be rakjuk, ahol az él által összekötött két csúcs együttesen alkotja a kulcsot, míg a hozzá tartozó érték az él súlya. A vonatkozó kódrészlet az alábbiak szerint néz ki.

```
# atalakitjuk a bemenetet, hogy konnyebben kezelheto legyen
# az eleket egy dict-be helyezzuk, ahol a kulcs az el
# altal osszekotott ket csucs, mig az ertek az el sulya
elek = {}
for i in range(len(g_from)):
    elek[(g_from[i], g_to[i])] = g_weight[i]
```

Ezt követően Kruskal algoritmusának megfelelően rendezzük az éleket súlyuk szerinti növekvő sorrendbe. Ehhez a `sorted` függvényt hívjuk meg, amely a `dict` kulcs-érték párjait (vö. `elek.items()`) az alapértelmezés szerint növekvő sorrendbe rakja, amelyhez az értékeket, azaz a súlyokat (vö. `key=lambda x: x[1]`) használja. Az eredményül kapott listát pedig eltároljuk egy új változóban (vö. `rendezett_elek`). A releváns kódrészlet alább látható.

```
# rendezzuk az eleket sulyuk szerint novekvo sorrendbe
rendezett_elek = sorted(elek.items(), key=lambda x: x[1])
```

Ezután bejárjuk a rendezett éleket (vö. `for (ki, be), suly in rendezett_elek`). Minden iterációban megkeressük az aktuális él két csúcsához (vö. `ki` és `be`) tartozó két csúcshalmazt (vö. `ki_halmaz = halmazt_keres(ki, halmazok)` és `be_halmaz = halmazt_keres(be, halmazok)`), amelyekhez felhasználjuk a `halmazt_keres` segédfüggvényt. Ha a két csúcshalmaz nem egyezik meg (vö. `ki_halmaz != be_halmaz`), azaz az aktuális él hozzávétele nem hoz létre kört, akkor hozzáadjuk az aktuális élt az eddig kiválasztott élekhez, a súlyát beszámítjuk a teljes súlyba (vö. `teljes_suly += suly`), illetve a két csúcshalmazt egyesítjük (vö. `halmazok[ki_halmaz].update(halmazok[be_halmaz])` és `del halmazok[be_halmaz]`). A teljes vonatkozó kódrészlet az alábbiak szerint néz ki.

```
# vegigmegyunk a rendezett eleken
for (ki, be), suly in rendezett_elek:
    # meghatarozzuk, hogy az el egyik csucsa melyik halmazban van
    ki_halmaz = halmazt_keres(ki, halmazok)
    # meghatarozzuk, hogy az el masik csucsa melyik halmazban van
    be_halmaz = halmazt_keres(be, halmazok)
    # ha a ket halmaz kulonbozik, azaz az el hozzavetele nem hoz
    # letre kort
    if ki_halmaz != be_halmaz:
        # hozzavesszuk az elt, igy a sulya beleszamit a teljes sulyba
        teljes_suly += suly
        # a ket csucshalmazt egyesitjuk
        halmazok[ki_halmaz].update(halmazok[be_halmaz])
        del halmazok[be_halmaz]
```

Végül pedig visszaadjuk a kiszámított teljes súlyt az alábbiak szerint:

```
# visszaadjuk a teljes sulyt
return teljes_suly
```

A `halmazt_keres` függvény pedig csak annyit tesz, hogy megkeresi azt a csúcshalmazt `halmazok`-ban, amelyik tartalmazza `csucs`-ot, és visszaadja a halmaz indexét:

```
def halmazt_keres(csucs, halmazok):
    # megkeressuk, hogy melyik csucshalmaz tartalmazza
    # a keresett csucsot
    for i in range(len(halmazok)):
        if csucs in halmazok[i]:
            return i
```