Imperatív Programozás Labor ZH – 2025.01.02.

Elvárások a programmal szemben

- A megoldáshoz semmilyen segédeszköz nem használható, kivéve a C referenciát.
- A program végleges verziójának működőképesnek kell lennie. Forduljon és fusson! **A nem forduló kód 0 pontot ér!**
- Ne használj globális változókat! Csak a makrók megengedettek!
- Törekedj a szép, áttekinthető kódolásra, használj indentálást, kerüld a kódismétlést!
- Kommunikáljon a program! Legyen egyértelmű a felhasználó számára, hogy mit vár a program, illetve pontosan mi történik!
- Logikusan tagold a megoldást (használj függvényeket, külön fordítási egységeket)!
- Ne foglalj szükségtelenül memóriát, és kerüld a memóriaszivárgást!
- Kerüld a nem definiált viselkedést okozó utasításokat!
- Ahol nincs pontosan leírva egy feladatrész, annak egyedi megvalósítását rád bízzuk.
- A struktúrák paraméterként átadásánál gondold át, hol érdemes mutatót használni!
- **A végleges programban a be nem tartott elvárások pontlevonással járnak!**

Logisztika

avagy hogyan tároljunk és adjunk vissza növekvő sorrendben egy bemeneti egyedi egész értékekből álló számsort

A MindentRendez Kft raktárában némi felfordulás történt, összekeveredtek a kiszállításra váró raklapos áruk. Mivel nagy és nehéz árukról van szó, ezért a rakodás során fontos lenne, hogy a minimálisnál többször ne mozgassuk az árukat, az egyes címeken a kirakodás során pedig már egyáltalán ne kelljen keresgélni, rendezgetni – erre ugyanis se hely, se idő nincsen.

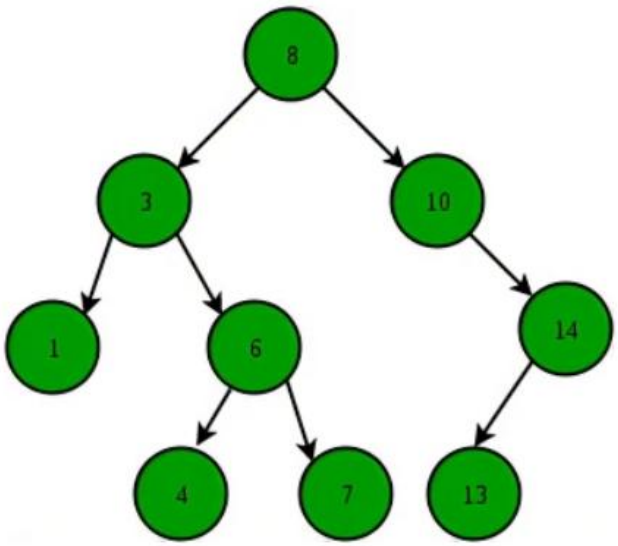
Segíts nekik a kiszállítási távolság alapján sorrendbe állítani a raklapokat, hogy a kiszállítás során minden esetben az éppen soron következő címhez tartozó áru legyen a következő elérhető raklapon a kamionon.

Szerencsére nem maradsz teljesen egyedül a feladattal, mert a MindentRendez Kft egyik logisztikusa, Mr Igor, szabadidejében különböző algoritmusokról szokott olvasgatni, az egyik nemrég olvasott pedig pont alkalmas lehet ebben a helyzetben. Sajnos Mr Igor nem emlékszik az algoritmus/adatszerkezet pontos nevére, de részletesen ismerteti az ötletét az alábbiakban. *(szerkesztői megjegyzés: bináris keresőfa adatszerkezetről van szó, de ahogyan Mr Igornak, úgy neked sincs, kedves Hallgató, szükséged erre az infóra – a megoldáshoz szükséges minden információ szerepel a feladatleírásban)*

Mr Igor ötlete szerint, mivel az összekeveredett raklapokhoz csak egyesével, egymás után férünk hozzá és így nem lehetséges ezeket eleve jó sorrendben kiszedni a raktárból, ezért, egyesével szükséges a raklapokat az udvarra kivinni, ott viszont úgy helyezik el őket, hogy további mozgatás nélkül kialakuljon a helyes sorrendjük az alábbi logika szerint:

- a megértést segíti a jobb oldali ábra, mely a 8, 10, 3, 14, 1, 13, 6, 4, 7 raklapos áruk udvari elhelyezését mutatja – minden egyes szám az adott áru kiszállítási címének depótól való távolságát mutatja
- a legelső raklapot elhelyezik az udvar egyik szélén, középre rendezve – az ábrán a 8-as szám

- a következő raklapokat eleve rendezve rakják le, ha az első raklaptól közelebbi a cél, akkor bal oldalra, ha távolabbi akkor jobb oldalra – példában először a 10-est rakjuk le a 8-astól jobbra (mivel 10>8), majd a 3-ast a 8-astól balra (mivel 3<8)
- ezt a balra-jobbra rendezést nem csak az első raklaphoz képest, hanem folyamatosan végezzük, amíg találunk egy „üres” helyet – ugyanis minden raklaphoz maximum két ilyen hely van, egy tőle jobbra, egy pedig balra. Azaz, ha már lehelyeztük a 8, 10, 3 raklapokat, akkor a következő 14-es távolságú raklapot biztosan a 8-astól jobbra kell helyeznünk, de ott közvetlenül már nincs hely, hiszen azt a 10-es elfoglalja. Ezért megnézzük, hogy a 10-es alá – tőle jobbra, hiszen 14>10 - elhelyezhető-e a 14-es raklap, szerencsére itt még találunk üres helyet



- Következőkben lehelyezzük az 1-es raklapot a 8-astól balra, majd a 3-astől is balra. A 13-as raklapot a kezdeti 8-astól jobbra, az ott lévő 10-estől szintén jobbra, az ott lévő 14-estől viszont már balra helyezzük
- Az eddig ismertetett lerakási logikát alkalmazzuk, míg el nem fogy az összes raklap. Amikor végzünk, akkor nagy valószínűséggel nem lesz teljesen „tömör” a felépített szerkezet – lesznek benne üres helyek, pl a fenti példában 10-től balra nincsen semmi – de ez egyáltalán nem probléma

A feladat összesen 50 pontot ér. Legalább 10 pontot kell gyűjteni a tárgy sikeres teljesítéshez. A megoldásra 180 perc áll rendelkezésre. Beadáskor csak a .c, .h fájlokat kell becsomagolva feltölteni a következő formában: <neptun_kód>.zip (Az utoljára feltöltött megoldást pontozzuk.) A következőkben felsorolásra kerülnek a kötelezően megvalósítandó alprogramok, de ezeken túl ajánlott további „segéd” alprogramok elkészítése is.

Főprogram – main() – 15 pont

Elsődleges feladata a program összefogása és kommunikáció a felhasználóval. Köszöntsük a felhasználót, és röviden ismertessük a programot. (Saját szavakkal, pár mondatban elegendő.) Ezután egy menü segítségével ajánljuk fel a lehetséges funkciókat a felhasználónak, nevezetesen:

1) Hozzáadás – egy új raklap hozzáadása az adatszerkezethez az insert_pkg(...) függvény segítségével, a depótól való távolságot a felhasználótól kérjük be
2) Kirajzolás – az adatszerkezet aktuális tartalmának kirajzolása a print_tree(...) függvény segítségével
3) Rakodási lista – az adatszerkezet tartalma alapján a get_manifest(...) felhasználásával írjuk ki egy fájlba a (memóriacím, depótól való távolság) párokat, soronként csak egy párt.
4) Csomag törlése – töröljünk az adatszerkezetből egy csomagot a delete_pkg(...) függvény segítségével

5) Kilépés – a program befejezése, itt ne felejtsd el a használt memóriát felszabadítani, ehhez érdemes egy külön delete_all_pkgs(...) függvényt készíteni. Építhetsz a delete_pkg() függvényre, de attól függetlenül is megoldható.

> *Mr Igor tippje: a teljes adatszerkezet lebontása és az összes memória felszabadítása rekurzióval a legegyszerűbb. Ehhez először nézd meg a print_tree()-nél szereplő tippet, annak egy módosított verziójára lesz itt szükség, ahol először megtörténik rekurzíven mind a bal, mind a jobb gyerekből elérhető teljes szerkezet feldolgozása (jelen esetben felszabadítása) és csak ezt követően történik meg az aktuális elem törlése.*

A program készüljön fel arra is, ha nem létező menüpontot adna meg a felhasználó

Tipp: kezdetben érdemes felvenni a {8, 10, 3, 14, 1, 13, 6, 4, 7} tömböt a main()-be, hogy egy ciklussal könnyen, gyorsan feltölthető és így kipróbálható, tesztelhető legyen a készülő megoldás.

Beszúrás – insert_pkg(...) – 10 pont

Az eddig ismertetettek alapján készíts egy láncolt adatszerkezetet, melyben minden raklapot egy node-ként reprezentálsz, tárolva a saját értékét (depótól való távolság) illetve a tőle balra illetve jobbra elhelyezkedő node címét. Az insert_pkg(...) függvény egy újabb csomag raktárból az udvarra való kitolását és az udvaron a megfelelő pozícióba történő elhelyezését jelenti. A függvény egymás utáni többszöri hívásával felépíthető az előző oldalon szereplő példa teljes szerkezete.

Szerencsére a lehetséges címzettek úgy helyezkednek el a depohoz képest, hogy nem lehet két címzettnek azonos távolsága, így erre az adatszerkezet építése/kezelése során nem kell felkészülni, ugyanakkor hibás adatbevitel (egy olyan távolságú raklapot próbálunk hozzáadni amilyen már van) előfordulhat, ilyenkor ezt vissza kell utasítani/közölni a felhasználóval – az adatszerkezet változatlan marad.

Tipp#1: célszerű egy create_pkg(...) függvényt is készíteni, mely csak egy független node-ot inicializál.

Tipp#2: gondold végig, hogy a teljesen üres adatszerkezetet hogyan reprezentálod (NULL pointer vagy egy leíró szerkezet) – mindkettő lehet jó, ha a további függvényeidet ennek megfelelően készíted el.

Kiírás – print_tree(...) – 10 pont

Készíts egy függvényt mely az adatszerkezethez eddig hozzáadott raklapokat kiírja/rajzolja a képernyőre. Az egyszerűség kedvéért használhatjuk a jobb oldalon szereplő példa formázását, ahol a legfelső sor az adatszerkezet legbaloldalibb, a legutolsó sor pedig a legjobboldalibb eleme.

Ha van kedved/időd akkor választhatsz ettől fancybb, jobban formázott kiírási formát is.

Mr Igor tippje: a felépített adatszerkezet legkönnyebben rekurzívan járható be – jelen esetben ha először a bal oldali részt dolgozzuk fel rekurzívan, majd az aktuális elemet – itt éppen kiírjuk az értéket, a mélységgel (szülő csúcstól való távolság) indentálva –, végül a jobb oldali részt ismét rekurzívan, akkor pont a példában szereplő kiírást kapjuk az 5, 3, 4, 2, 7, 12, 8, 18, 6, 1 távolságú raklapsorozatra.

Rakodási lista – get_manifest(...) – 10 pont

A get_manifest(...) függvény adjon vissza egy node-okra (raklapok) mutató pointereket tartalmazó tömböt, melynek pontosan annyi eleme van, ahány elem van az adatszerkezetünkben (raklap az udvaron). A tömb a raklapok depótól való távolsága alapján növekvő sorrendben tartalmazza az egyes node-ok memóriacímeit, ezzel megadva a bepakolás sorrendjét.

Mr Igor tippje: Figyeld meg, hogy a print_tree() kimenetét ha fentről lefelé soronként vizsgálod akkor pont növekvő sorrendet kapsz, így ha itt is ugyanazt a rekurzív bejárást használod, csak kiírás helyett a memóriacímet mented ki egy tömbbe, akkor már meg is van az eredmény.

Sérült csomag eltávolítás – delete_pkg(...) – 5+5 pont

Úgy néz ki a MindentRendez Kft raktárában ma semmi nem megy simán, az udvarra kihúzott raklapok között ugyanis egy utólagos ellenőrzés talált néhány lejárt szavatosságú árut – melyek így sajnos nem szállíthatóak ki, azonnal el kell őket tüntetni az udvarról. Készíts egy delete_pkg() függvényt, mely a már felépített adatszerkezetből (udvarra kihordott raklapok) töröl egy node-ot (raklapot) olyan módon, hogy az adatszerkezet belső tulajdonságai továbbra is érvényesek maradnak:

- minden node-nak maximum egy jobb és egy bal gyereke van
- minden nodra igaz, hogy a saját értéke (depótól való távolság) nagyobb, mint az összes bal gyerek mentén elérhető érték, és kisebb, mint az összes jobb gyerek mentén elérhető érték

Mr Igor tippje: Egy node törlésénél az alábbi 3 eset lehetséges – ebből az első kettő igazán egyszerű, az utolsó picit trükkösebb ezért azt részletesen elmagyarázom:

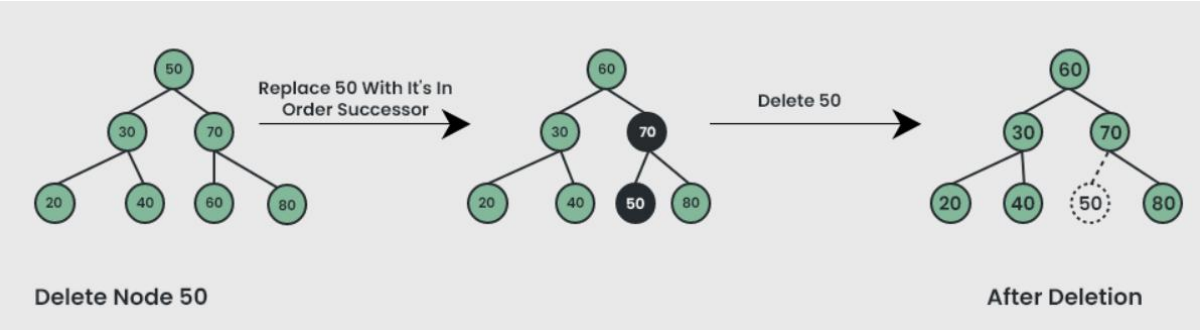
***Gyerek nélküli node törlése**: itt csak arra kell figyelni, hogy a szülő nodeból „kikössük" a gyereket még a node törlése előtt*

***Egy gyerekes (akár bal akár jobb) node törlése**: itt egy szülő és egy gyerek van, őket kell összekötni – a törlendő node-ot kifűzve – pont mintha egy sima láncolt listából törölnénk egy nem utolsó elemet*

***Két gyerekes node törlése**: itt a sima kifűzés nem működik, hiszen egy helyre kellene befűzni a két gyerek node-ot. Helyette az alábbiakat csinálhatjuk:*

Keressük meg a törlendő node értékhez legközelebbi (akár kisebb akár nagyobb) értéket az adatszerkezetben, ezt a bal gyerekből indulva végig a jobb gyerekeket választva eljutunk egy levél node-hoz, melynek már nincs gyereke. Amit így találunk az az egyik legközelebbi érték. Hasonlóan, ha a jobb gyerekből indulva követjük a bal gyerekeket akkor pedig a másik legközelebbi elemet találjuk meg, de igazából nekünk csak az egyikre van szükségünk –

bármelyikre. A fenti példában az 50-es törlendő értékhez a két legközelebbi érték a 40 és a 60.



A megtalált egyik legközelebbi értéket cseréljük ki a törlendő elemmel (fenti példán az 50 -> 60 csere). Ezzel átrendeztük olyan módon az adatszerkezetet, hogy a fentebb említett elvárt belső tulajdonságai nem sérültek, viszont a törlés már könnyebb, mert a törlendő elemnek immár nincs gyereke, így tudjuk az erre vonatkozóan már korábban leírtakat alkalmazni.

A zh 100%-os teljesíthető a 0 és 1 gyerekes törlési esetek implementálásával, a 2 gyerekes esettel +5 pont szerezhető.