Beadandó feladatok

Feladatok a Haladó Alkalmazott Programozás gyakorlathoz, 2019.

A gyakorlati jegy megszerzéséhez mindenkinek 1 feladat megoldását kell elkészítenie az alábbi feladatok közül és azt az alábbi határidőig elküldenie a gpu (kukac) wigner.mta.hu címre. Aki nem küld be elfogadható megoldást, az nem szerezhet jegyet. A megoldásnak szabványos C++17-ben kell elkészülnie, ezért célszerű több fordítóval is ellenőrizni a helyességet. A feladatoknál elvárás, hogy a standard library-t használjuk, ahol lehet és célszerű. **Elvárás, hogy a programok C++ konstrukciókkal és stílusban íródjanak, hibák és figyelmeztetések nélkül forduljanak, helyesen működjenek, a foglalt memóriát felszabadítsák, ne indexeljék túl a tömböket, ne segfaultoljanak, stb.**

Lehetőség van rá, hogy a teljesen vagy részben megoldott feladatokat ellenőrzésre, vagy segítségkérésre elküldjétek nekünk a végső beadás előtt, amely az értékelésbe nem számít bele, de segít tisztázni félreértéseket, vagy hiányosságokat a megoldásban, javasoljuk, hogy használjátok ki ezt a lehetőséget.

Egyéb hasonló jellegű feladat is választható (pl. mert az valakinek kell/hasznos a szakdolgozatához, tdk-hoz, stb), ezt kérjük előre egyeztessétek legalább 1 oldalas részletes leírással és program tervvel együtt!

Szeretnénk, ha **minden feladatot csak 1 ember választ**ana. A választott feladat számát és a variációját küldjétek el email-ben, a már választott feladatok jelzésre kerülnek a weboldalon. Aki változtatni szeretne, az a kiírás utáni első hétben ezt megteheti e-mailben.

A feladatok nehézségére igyekszünk utalni az alábbi sorrendben, de kérdezzetek, ha nem világos, hogy mit kell csinálni. **A feladatkiírás nem feltétlen tartalmaz minden részletet**, ha valamit úgy éreztek hiányzik, vagy nincs kellően definiálva, kérdezzetek rá. A nehezebb feladatokat nagyobb súllyal vesszük figyelembe az év végi jegyben, a könnyebbeknél minden kis hiba nagyobb súllyal számít.

Beküldési határidő: **Május 5. Vasárnap 23:59.**

Az ez után beérkező feladatok naponta 20%-al kevesebbet érnek (tehát május 9 után nem fogadjuk el őket).

Feladatok egyszerűtől a nehezebbekig:

1. Monte-Carlo integrálás (link)
   Írjunk integrátort, ami az alábbi függvényként hívható, és kiszámolja egy 3D-s integrál közelítő értékét. Az első argumentum az integrandus, a második az integrálási tartomány (igazat ad vissza, ha a pont benne van a tartományban), az utolsó 6 az integrálási tartomány bennfoglaló téglatestje.

   ```
   MonteCarlo( [](auto x, auto y, auto z){ return exp(-x*x-y*y-z*z); },
               [](auto x, auto y, auto z)->bool{ return x*x+y*y+z*z<16.0; },
                -4.0, 4.0, -4.0, 4.0, -4.0, 4.0);
   ```

2. 2x2 Mátrix struktúra

Kompatibilisen a házi feladatban megírt 2-es Vektor struktúrával, az alapműveleteken túl inverz, determináns, egyenletrendszer megoldás.

3. Stack-en tárolt Vektor és négyzetes Mátrix struktúra
   Típus szinten adott méretű, hasonlóan az std::array-hez. A szokásos matematikai műveleteket kell támogatni közöttük. Mátrixon legalább kettőt kell az alábbiak közül: lineáris egyenletrendszer megoldása ismert jobboldallal, determináns számítás, inverz számítás.

4. 0-1-2 dimenziós intervallum osztály 2 dimenzióban
   1, 2, ill. 4 skalár értéket tárol, amelyek egy pontot/szakaszt/téglalapot határoznak meg (utóbbi kettő mindig párhuzamos a koordináta tengelyekkel), ezek között metszet, unió, különbség, eltolás, adott pont/szakasz körül nagyítás, kicsinyítés, pont/szakasz benne van, nincs, 1D-ből direkt szorzat->2D műveleteket támogat.

5. 3 dimenziós forgatások reprezentálása több féle módon
   Kvaterniók, 3x3-as mátrixok, szög-tengely reprezentáció és az ezek közötti konverziók Link, Link, Link

6. Racionális szám osztály
   Számláló és nevező egész számként tárolva, és rajta az alapműveletek legalább a gyökvonásig (Newton iteráció). Legyen kompatibilis a saját Vector2 és Mátrix osztályunkkal!

7. 3 dimenziós egyenes osztály két féle reprezentációban
   Egyik struktúrában két pont határozza meg az egyenest, a másikban használjuk a Plücker féle reprezentációt. Írjuk meg a konverziós, és a geometriailag hasznos műveleteket (egyenesek távolsága, metszése, viszonya, stb.)

8. Polinom osztály (1 változós)
   Az osztály a polinom együtthatóit tárolja (dinamikusan változhat, tehát pl. std::vector), és az alapműveleteket támogatja: összeg, különbség, szorzat, kompozíció, derivált, integrál értéke adott helyen, gyökei (Newton iterációval), stb. Move aware!

9. Függvény osztály (1 változós)
   Az osztály valamilyen általunk választott reprezentációban tárolja egy folytonos függvény valamilyen diszkretizációját, a cél, hogy lehessen kiértékelni bárhol (interpoláció), határozott és határozatlan integrált számolni, deriválni, gyököt, minimumot, maximumot keresni. Move aware!

10. Hisztogram osztály (1 változós)
    Dinamikusan állítható határok és felosztás. Lehessen két azonos hisztogramot összeadni, kivonni, skalárral szorozni, osztani, és random mintavételezni belőle, mint eloszlásból! Move aware!

11. Gráf osztály (irányítatlan)
    Lehessen megadni csúcsokból és élekből, vagy az összekötöttségi mátrixból, lehessen bármikor bővíteni új csúccsal vagy éllel, vagy egy ilyet kivenni belőle. Támogasson legrövidebb utat két csúcs között, tudja eldönteni, hogy teljes, vagy nem, összefüggő, vagy ha nem, hány darab nem összefüggő komponense van, fa, vagy nem fa. Bináris műveletek: unió, legalább egy fajta szorzat. Move aware!

12. std::list vs std::vector benchmarking

    Írjunk programot, ami azonos műveleteken hasonlítja össze a lista és a vector hatékonyságát, mint például, elemekből megkonstruálás, elejére/végére/közepére n elem beszúrása, vagy törlése, rendezés, indexelés, stb. Szabványos időméréssel és véletlenszám-generálással legyen megírva.

13. Random engine benchmarking

    Hasonlítsuk össze a beépített random generátorok sebességét és tulajdonságait, főleg, hogy magasabb dimenziókban tudunk-e felfedezni korrelációt a generált számok között.

14. Kártya pakli keverésének vizsgálata

    Definiáljunk egy típust, ami képes reprezentálni egy kártyalap tulajdonságait egy kiválasztott fajta pakliból. (a kártya színeire pl. használhatunk enumokat). Definiáljunk egy rendezést a kártya tulajdonságai felett (pl.: először szín szerint, azon belül érték szerint), és hozzunk létre egy pakli kártyát és rendezzük eszerint a reláció szerint. Definiáljunk egy keverő függvényt, ami valahogy modellezi azt, ahogy kézzel keverjük a lapokat. Vizsgáljuk meg, hogy kiválasztott „érdekes" mintázatok (pl. 5 egyforma szín egymás után) milyen gyorsan (n függvényében) érik el teljesen random pakliban mérhető előfordulási valószínűségeiket a keverés paramétereinek függvényében. Használjunk standard algoritmusokat a keverésre (shuffle, de NE a deprecated változatokat!), és amire csak lehet.

15. Mértékegységek és prefixumok követése

    Csináljunk egy olyan osztályt, amely adott T típust tárol, de követi azt is, hogy 3 kiválasztott SI alapmennyiség (pl. hosszúság, idő, tömeg) milyen kombinációját jelenti. Ezt pedig úgy éri el, hogy tárolja a három mennyiség kitevőjét. Tehát pl. a sebesség reprezentálása:

    ```
    template<typename T, int L, int T, int M> class ValueWithDimension;
    template<typename T> using Velocity = ValueWithDimension<T, 1, -1, 0>;
    ```

    Írjuk meg erre az osztályra az alapműveleteket, úgy, hogy összeadni, kivonni csak olyan mennyiségeket lehessen, amelyek azonos mértékegységűek! Szorzásnál, osztásnál pedig megfelelően transzformálódjanak a kitevők. Terjesszük ki az osztályt azzal a funkcionalitással, hogy az ismert SI prefixumokat is tárolja és követi a mennyiségekre, és a műveleteknél igyekszik a legjobb köztes prefixumot választani (pl. 1km + 1mm -> 1000.001 m).

16. Elosztott fájlokban tárolt 2D adattábla

    Írjunk olyan osztályt, amely hatalmas 2 dimenziós adattáblát reprezentál, amely nem fér el a memóriában, blokkokra van osztva, amelyek valahogy fájlokban vannak tárolva, és mindig az éppen használt blokkot tölti be a memóriába, dolgozik rajta, majd írja ki újra fájlba. Move aware!

17. Blokk-szekvenciális tároló

    A láncolt lista (std::list) és a vektor (std::vector) két végletként tekinthető a lineáris tárolás szempontjából, ugyanis el lehet képzelni egy olyan tárolót, amely láncolt blokkokat tárol, ahol minden blokk *b* darab elemet tárol folytonosan a memóriában (mint a vector), és a blokkok pointerekkel mutatnak egymásra (mint a listában). Ha *b=1*, akkor visszakapjuk a listát, ha *b=n,* ahol *n* az elemszám, akkor a tároló ugyan az, mint a vektor. Implementáljuk ezt a tárolót, b legyen template paraméter, és írjuk meg rá a legszükségesebb konstruktorokat, tagfüggvényeket, begin, end metódusokat. Hasonlítsuk össze különböző reprezentatív *b*-k esetére a következő eljárások átlagos

idejeit (az időméréshez az std::chrono szolgáltatásait használjuk!): Véletlenszerű indexelés, Elemek beszúrása a tároló elejére, Elemek beszúrása a tároló végére, n egymásutáni elem törlése a tároló random pozícióból. Move aware!

18. BKK adatbányászat
    Töltsük le a BKK menetrend adatokat, és írjunk beolvasó rutint, hogy az alábbi kérdéseket automatikusan megválaszolhassuk: Melyik a leghosszabb busz, villamos járat (megállószámra, vagy hosszra)? Melyik járatból közlekedik a legtöbb egy átlagos munkanapon? Egy adott időpontban mennyi járat van éppen úton? Hány villamos/buszmegálló van összesen Budapesten?

19. Vízhőmérséklet adatok feldolgozása
    Adott egy adatsor, amely naponta több pillanatban tárol több mérőpontból vízhőmérséklet adatokat. Olvassuk be és dolgozzuk át úgy, hogy naponként 1 mért érték legyen, ha volt megfelelő adat az adatsorban. Írjunk pár lekérdezést: maximum, minimum, legnagyobb hőingás 1 hét alatt, stb.

20. N test szimuláció
    Írjunk 4-ed rendű Runge-Kuttával N-test szimulációt, és próbáljuk meghatározni azt, hogy a Naprendszeren kívülről érkező kis tömegű test mikor ütközik nagyobb valószínűséggel a Földdel: ha van a Naprendszerben Jupiter, vagy ha nincs.

21. Kettős inga szimulálása
    Írjunk a 4-ed rendű Runge-Kuttával kettős inga szimulációt, úgy, hogy grafikusan megjelenítjük az ingát, és egérrel meglökhetőek a súlyok. Rajzoljuk ki a fázisteret is.

22. Lorenz pillangó kirajzolása
    Számoljuk ki és jelenítsük meg a Lorenz rendszert, és számoljunk Ljapunov exponenst, amely alapján színezzük ki.

23. Ideális gáz gravitációs térben
    A 4-ed rendű Runge-Kutta felhasználásával írjunk 2D szimulációt, amelyben kis kör alakú gázrészecskék rugalmasan ütköznek egymással, és az alsó felülettel, vízszintesen periodikus határfeltétel van, és felfelé nyitott a rendszer, lefelé pedig a magassággal csökkenő gravitációs erő hat. Számoljuk ki az átlagos és a magassági régiókra külön-külön számolt termodinamikai állapotjelzőket (hőmérséklet, nyomás).

24. Cahn-Hilliard egyenlet szimulációja (link)
    Vegyünk fel egy 2D-s periodikus rácsot ($N \times N$ cella), minden cellában egy random $[-1.0; +1.0]$ közötti lebegőpontos értékkel. Véges differencia módszerrel, kis lépésközzel léptessük a rácsot, rajzoljuk ki grafikusan és néhány lépésenként közelítsük a kétpont korrelációs függvényt random mintával: $C(r) = < P, Q >$, ahol P és Q két rácspont, aminek a távolsága r ($r = 0..N$) és rajzoljuk ki ezt is.

25. Kontúr keresés és prominencia számolás 2D-s adaton
    Írjunk marching-squares algoritmust, amivel kontúr vonalakat lehet készíteni, majd egy betöltött 2D-s adatsoron készítsük el a kontúr vonalakat, és írjunk függvényt, amely képes meghatározni a

csúcsok prominenciáját.

26. Lefolyási területek határai
    Töltsünk be egy 2D-s magasság térképet, és határozzuk meg rajta a lefolyási területeket (hány független van belőlük, mekkorák, és határozzuk meg a határaikat is).

27. Mandelbrot fraktál
    Írjunk programot, amely kirajzolja, valamint nagyíthatóan és navigálhatóan megjeleníti a Mandelbrot fraktált.

28. Szivárvány: 2D sugárkövetés
    Számoljunk szivárványt 2D-ben: egy párhuzamos sugárnyaláb érkezik egy körre, ahol a sugarakat az irány vektoruk és a λ hullámhosszuk jellemzi. A sugarak egy n_1 törésmutatójúközegben terjednek, a kör belseje viszont n_2 a törésmutatójú. A sugarak a Snellius-Descartes törvény szerint törnek, illetve teljes visszaverődést szenvednek a körön. Addig kövessük a sugarakat, amíg el nem hagyják a kört, és ekkor készítsünk 2D hisztogramot a hullámhossz-irányszög szerint. Az RGB értékeket a hullámhosszból a Color Matching Function-ökkel való átfedésből számítsuk.

29. Neurális Hálós számfelismerés
    Írjunk egy egyszerű 2 rétegű neurális hálót, amelyet az MNIST kézzel írott számjegyeken tanítunk be.

30. Társasjáték AI
    Írjunk AI-t egy általunk választott (lehetőleg egyszerű) társasjátékhoz. Pl.: UNO, Pandemic.