Alapvető algoritmusok

2. előadás

Dr. Pătcaș Csaba

BABEŞ-BOLYAI TUDOMÁNYEGYETEM
Matematika és Informatika Kar

Tartalom

1. Laborfeladatokra vonatkozó alapszabályok
   - Plagizálás
   - Programozási stílus
   - Fejlesztői környezet
   - Tömbök
   - Gyakori hibák
   - Tesztelés
2. Könyvészet
3. Maximális összegű tömbszakasz

Laborházik

Mi számít másolásnak?

- Átvett programban változónevek cseréje
- Átvett programban alprogramok nevének, sorrendjének megváltoztatása
- Bármi, ami átvett program módosításával indul
- Átvett program származhat évfolyamtárstól, középiskolából, korábbi évfolyamokról, ismerősöktől, könyvből, mesterséges intelligenciától (pl. ChatGPT), internetről stb.
- Egy program(részlet) akkor is másoltnak számít, ha nem „copy-paste” módszerrel lett másolva, csak „rápillantásos módszerrel”.
- Másolás esetén mindkét fél hibás és büntetve lesz. Figyelem a jóindulatú képernyőmegosztásos magyarázatokra is!

Laborházik

Mi nem számít másolásnak?

- Ötletek, algoritmusok, implementálási részletek szóbeli megbeszélése
- Már megírt programban való közös hibakeresés (a javítás a szerző dolga)
- Az a legbiztosabb, ha nem küldjük át az elkészült házifeladatokat másoknak.
- A félreértések elkerülése végett javasolt óra elején jelenteni, hogy ki kivel dolgozott együtt.

Példák

Másolás vagy nem?

```
bool illeszkedik(int i, int domino[][2]) {
  if (i > 0)
    if (domino[i - 1][1] != domino[i][0])
      if (domino[i - 1][1] == domino[i][1])
        csere(domino[i][0], domino[i][1]);
      else return false;
  return true;
}
bool szabad(int i, int domino[][2]) {
  for (int j = 0; j < i; j++)
    if ((domino[j][0] == domino[i][0] &&
    domino[j][1] == domino[i][1]) ||
    (domino[j][0] == domino[i][1] &&
    domino[j][1] == domino[i][0])) return false;
  return true;
}
void variacio(int i, int n, int domino[][2], int x[][2]) {
  for (int j = 0; j < n; j++) {
    domino[i][0] = x[j][0];
    domino[i][1] = x[j][1];
    if (illeszkedik(i, domino) && szabad(i, domino))
      if (i < n - 1) variacio(i + 1, n, domino, x);
      else kiirEredmeny(n, domino);
  }
}
```

```
bool ellenorzes(int i, int d[][2]) {
    if (i > 0)
        if (d[i - 1][1] != d[i][0]) {
            if (d[i - 1][1] == d[i][1])
                felcserel(d[i][0], d[i][1]);
            else return false; }
    return true;
}
bool ellenorzes2(int i, int d[][2]) {
    for (int j = 0; j < i; j++) {
        if ((d[j][0] == d[i][0] && d[j][1] == d[i][1]) ||
            (d[j][0] == d[i][1] && d[j][1] == d[i][0]))
            return false;
    }
    return true;
}
void back(int i, int n, int d[][2], int x[][2]) {
    for (int j = 0; j < n; j++) {
        d[i][0] = x[j][0]; d[i][1] = x[j][1];
        if (ellenorzes(i, d) && ellenorzes2(i, d)) {
            if (i < n - 1)  back(i + 1, n, d, x);
            else {
                kiir(n, d);
                ok = 1; }
        }
    }
}
```

Példák

Másolás vagy nem?

Válasz: IGEN!
Magyarázat: Csak átnevezték az azonosítók neveit

Algoritmika
Dr. Pătcaș Csaba
Laborfeladatok
Plagizálás
Stílus
IDE
Tömbök
Gyakori hibák
Tesztelés
Könyvészet
Maximális összegű tömbszakasz

Példák

Másolás vagy nem?

```
int main()
{
int a, b, c;
cout << "a haromszog elso oldala:" << endl;
cin >> a;
cout << "a haromszog masodik oldala:" << endl;
cin >> b;
cout << "a haromszog harmadik oldala:" << endl;
cin >> c;

double felkerulet = (double) (a + b + c) / 2;
cout << felkerulet << endl;
double terulet =
    sqrt(felkerulet * (felkerulet - a) *
    (felkerulet - b) * (felkerulet - c));
cout << "Terulet:" << terulet;

return 0;
}
```

```
int main(){

    float a, b, c; //oldalak hossza
    float felkerulet, terulet;

    cout << "Add meg a haromszog oldalainak a hosszat!" << endl;
    cin >> a >> b >> c;

    felkerulet = (a + b + c) / 2;
    terulet = sqrt(felkerulet * (felkerulet - a) *
    (felkerulet - b) * (felkerulet - c));
    cout << terulet;

    return 0;
}
```

Példák

Másolás vagy nem?

Válasz: NEM!
Magyarázat: A feladat túl egyszerű ahhoz, hogy lényegesen különböző megoldásokat lehessen írni rá

Algoritmika
Dr. Pătcaș Csaba
Laborfeladatok
Plagizálás
Stílus
IDE
Tömbök
Gyakori hibák
Tesztelés
Könyvészet
Maximális összegű tömbszakasz

Példák

Másolás vagy nem?


```
long long kruskal(pair<long long, pair<int, int> > p[])
{
    int x, y;
    long long cost, minimumCost = 0;
    for(int i = 0;i < edges;++i)
    {
        // Selecting edges one by one in increasing order from the beginning
        x = p[i].second.first;
        y = p[i].second.second;
        cost = p[i].first;
        // Check if the selected edge is creating a cycle or not
        if(root(x) != root(y))
        {
            minimumCost += cost;
            union1(x, y);
        }
    }
    return minimumCost;
}
```

Példák

Másolás vagy nem?

Válasz: IGEN!
Magyarázat: Internetről másolva

Példák

Másolás vagy nem?

```
int keres(int *v[], int m, int n) {
    int ossz = 0, i = 0, c = 0, j;
    int *tomb = new int [n]{0};
    while(c != n-1) {
        if(tomb[v[i][0]] != tomb[v[i][1]] ||
        tomb[v[i][0]] == 0 || tomb[v[i][1]] == 0)  {
            ossz = ossz + v[i][2];
            c++;
            if(tomb[v[i][0]] != 0) {
                if(tomb[v[i][1]] != 0) {
                    for(j = 0; j < n; j++)
                        if(tomb[j] == tomb[v[i][1]])
                            tomb[j] = tomb[v[i][0]];
                } else tomb[v[i][1]] = tomb[v[i][0]];
            } else {
                if(tomb[v[i][1]] != 0)
                    tomb[v[i][0]] = tomb[v[i][1]];
                else {
                    tomb[v[i][0]] = v[i][0];
                    tomb[v[i][1]] = v[i][0];
                }
            }
        }
        i++;
    }
    return ossz;
}
```

```
int keres(int *t[], int m, int n) {
    int s=0,i=0,k,db=0; int *kotes=new int [n]{0};
    while(db!=n-1){
        if(kotes[t[i][0]]!=kotes[t[i][1]] ||
        kotes[t[i][0]]==0 || kotes[t[i][1]]==0) {
            s+=t[i][2];
            db++;
            if(kotes[t[i][0]]==0){
                if(kotes[t[i][1]]==0){
                    kotes[t[i][0]]=t[i][0];
                    kotes[t[i][1]]=t[i][0];
                }else{
                    kotes[t[i][0]]=kotes[t[i][1]];
                }
            }else{
                if(kotes[t[i][1]]==0){
                    kotes[t[i][1]]=kotes[t[i][0]];
                }else{
                    k=kotes[t[i][1]];
                    for(int j=0;j<n;j++){
                        if(kotes[j]==k)kotes[j]=kotes[t[i][0]];
                    }
                }
            }
        }
        i++;
    }
```

Példák

Másolás vagy nem?

Válasz: IGEN!
Magyarázat: Azonos gyökérből indult a két program

Példák

Másolás vagy nem?

```
ido++; belep[csomopont] = ido; low[csomopont] = ido;
vermen[csomopont] = true; verem.push(csomopont);
for (int i = 0; i < graf[csomopont].size(); i++) {
  int w = graf[csomopont][i];
  if (belep[w] == -1) {
    DFS_SCC(w, graf, belep, low, vermen, ido,
    komponens ,komponensek ,verem);
    low[csomopont] = min(low[w] ,low[csomopont]);
  } else {
      if ((belep[csomopont] > belep[w]) &&
          (vermen[w] == true)) {
        low[csomopont] = min(low[csomopont], belep[w]);
      }
    }
  }
  if (low[csomopont] == belep[csomopont]) {
    komponens++;
    komponensek.resize(komponens + 1);
    while ((verem.empty() == false) &&
           (belep[verem.top()] >= belep[csomopont])) {
      int csucs = verem.top();
      komponensek[komponens].push_back(csucs);
      vermen[csucs] = false;
      verem.pop();
    }
  }
```

```
ido++; belep[el] = ido; low[el] = ido;
vermen[el] = true; verem.push(el);
for (int i = 0; i < graf->sz_Lista[el].size(); i++)
{
  int aktualis_El = sz_Lista[el][i];
  if (belep[aktualis_El] == -1) {
    DFS_SCC(graf, aktualis_El, ido, belep, low,
    vermen, verem,komponensek);
    low[el] = min(low[el], low[aktualis_El]);
  }
  else {
    if ((belep[aktualis_El] < belep[el]) &&
        vermen[aktualis_El])
      low[el] = min(low[el], belep[aktualis_El]);
    }
  }
  if (low[el] == belep[el]) {
    komponensek++;
    SCC.resize(komponensek);
    while (!verem.empty()
          && belep[verem.top()] >= belep[el]) {
      graf->SCC[komponensek - 1].push_back(verem.top());
      vermen[verem.top()] = false;
      verem.pop();
    }
  }
```

Példák

Másolás vagy nem?

Válasz: NEM!

Magyarázat: Mindketten az előadásban szereplő pszeudokód alapján dolgoztak

Példák

Másolás vagy nem?

```
intime++; stepin[node] = intime; low[node] = intime;
child = 0;  separator = 0; int top = 0;
int hossz = szom[node].size();
for (int i = 0; i < hossz; i++)
  if (stepin[szom[node][i]] == 0) {
     bot = bot + 2; stack[bot - 1] = node;
     stack[bot] = szom[node][i]; child++; top = bot-1;
     depth(n, szom, db, komp, h, hidak, p, pontok, bot, stack,
     stepin, low, szom[node][i], intime, node, child, separator);
     low[node] = min(low[node], low[szom[node][i]]);
     if (low[szom[node][i]] >= stepin[szom[node][i]]) {
       h = h + 1; hidak.push_back(node);
       hidak.push_back(szom[node][i]);
     }
     if (low[szom[node][i]] >= stepin[node]) {
       separator = 1; db = db + 1; vector<int> connection;
       for (int j = top; j <= bot; j++)
         connection.push_back(stack[j]);
       komp.push_back(connection); bot = top - 1;
     }
  } else {
    if (szom[node][i] != parent and
        stepin[szom[node][i]] < stepin[node]) {
      bot = bot + 2; stack[bot - 1] = node;
      stack[bot] = szom[node][i];
      low[node] = min(low[node], stepin[szom[node][i]]);
    }
```

```
ido++; vermen[v] = ido; low[v] = ido; gyerekek = 0;
valaszt = 0; int top = 0; int hossz = graf[v].size();
for (int i = 0; i < hossz; i++) {
    if (vermen[graf[v][i]] == 0) {
        bot = bot + 2; verem[bot - 1] = v;
        verem[bot] = graf[v][i]; top = bot - 1; gyerekek++;
        melysegi(graf, komp, hidak, pontok, bot, verem,
        vermen, low, graf[v][i], ido, v, gyerekek, valaszt);
        low[v] = min(low[v], low[graf[v][i]]);
        if (low[graf[v][i]] >= vermen[graf[v][i]]) {
            hidak.push_back(v); hidak.push_back(graf[v][i]);
        }
        if (low[graf[v][i]] >= vermen[v]) {
            valaszt = 1; int l = komp.size();
            komp.resize(l + 1);
            for (int j = top; j <= bot; j++)
                komp[l].push_back(verem[j]);
            bot = top - 1;
        }
    } else {
        if (graf[v][i] != szulo &&
            vermen[graf[v][i]] < vermen[v]) {
          bot = bot + 2; verem[bot - 1] = v;
          verem[bot] = graf[v][i];
          low[v] = min(low[v], vermen[graf[v][i]]);
        }
    }
```

Példák

Másolás vagy nem?

Válasz: IGEN!
Magyarázat: Külön kezdtek el dolgozni, de utólag hibakeresés céljával addig alakították a programokat, míg „túl hasonlóak” lettek

Példák

Másolás vagy nem?

```
vector <unordered_map<int,int>> lista;
stack<int> euler;
void printEuler(int v, int n, int m)
{
    stack<int> rek_verem;
    rek_verem.push(v);
    while(!rek_verem.empty())
    {
        int u = rek_verem.top();
        if(lista[u].size()==0)
        {
            euler.push(u);
            rek_verem.pop();
        }
        else
        {
            rek_verem.push(lista[u].begin()->first);
            lista[u].erase(lista[u].begin()->first);
        }
    }
}
```

```
int vertex;
vector<unordered_map<int, int>> szomszedsagiLista;
stack<int> eulerVonal;
void Euler(int v, int n, int m)
{
  stack<int> verem;
  verem.push(v);
  while (!verem.empty())
  {
    int u = verem.top();
    if (szomszedsagiLista[u].size() == 0)
    {
      eulerVonal.push(u);
      verem.pop();
    }
    else
    {
      verem.push(szomszedsagiLista[u].begin()->first);
      szomszedsagiLista[u].erase(szomszedsagiLista[u].begin()->first);
    }
  }
}
```

Példák

Másolás vagy nem?

Válasz: „Inkább igen”
Magyarázat: Ugyan az algoritmus egyszerű, de sokféleképpen implementálható. Együtt dolgoztak és túl sok volt az inspiráció egymástól és az internetről.

Programozási stílus

Változók

- A változók jelentése legyen pontosan definiálva.
- Az azonosítók legyenek beszédesek! Engedélyezett a több mint két karakteres változónevek használata is!
- Ne használjunk inicializálatlan (kezdőérték nélküli) változókat!

Gyakran használt változónevek:

- Ciklusváltozók: `i, j(, k)`
- Bemeneti adatok, tömbök mérete: `n, m`
- Tömbök: `a, b, v`
- Számok: `x, y, z`
- Karakterek: `c, ch`
- Karakterláncok: `s`
- Állományok: `f, fp`

Algoritmika
Dr. Pătcaș Csaba
Laborfeladatok
Plagizálás
Stílus
IDE
Tömbök
Gyakori hibák
Tesztelés
Könyvészet
Maximális összegű tömbszakasz

Programozási stílus

Konstansok

- Kerüljük a szám vagy karakterlánc konstansok „elrejtését” a programkód belsejében. Ezt a jelenséget „magic numbers”-nek nevezzük, ugyanis egy kívülálló számára tetszőleges értéknek tűnhetnek ezek a mágikus számok.
  Pl. `if (kor < 18)`
  `nev = "Laci"`
- Ehelyett emeljük ki a forráskód elejére ezeket az értékeket, ahol adjunk nekik értelmes azonosítókat.
  Pl.

```
const int FELNOTT_KOR = 18;
...
if (kor < FELNOTT_KOR)

#define KIINDULASI_SZEMELY "Laci"
...
nev = KIINDULASI_SZEMELY
```

Algoritmika
Dr. Pătcaș Csaba
Laborfeladatok
Plagizálás
Stílus
IDE
Tömbök
Gyakori hibák
Tesztelés
Könyvészet
Maximális összegű tömbszakasz

Programozási stílus

Alprogramok

- A legfontosabb alapelv: D.R.Y. (Don't Repeat Yourself)
- A programot bontsuk minél kisebb (az ésszerűség határain belül) alprogramokra, úgy, hogy ezeknek még logikailag legyen értelme (egyértelműen megfogalmazható részfeladatot oldjanak meg). Egy olyan alprogram, amely egyszerre beolvas és ki is számolja a megoldást, nem egy logikailag különálló feladatot old meg.
- Kerüljük a „spagetti kódot"!
- Az alprogramok nevei legyenek beszédesek, a név alapján legyen tiszta, hogy mit old meg az adott alprogram.
- Többszavas azonosítók esetén C/C++-ban nincs egyértelmű konvenció, mindegy, hogy melyiket választjuk, de legyünk következetesek! pl. `GyorsHatvany`, `gyorsHatvany`, `gyors_hatvany`

Két alapvető stílust különböztethetünk meg, ami a paraméterlista megválasztását illeti:

1. Mindent küldünk paraméterként amire szüksége van az adott alprogramnak, vagyis mindent ami az alprogram által megvalósított algoritmus bemenete.
   pl. `keresTömbben(a, n, x)`
2. Csak azokat a bemeneti változókat küldjük paraméterként, amelyek az alprogram többszöri hívásai között változnak, más értékre hívódnak meg.
   pl. `keresTömbben(x)`

- A második stílussal inkább nagyobb projektek esetén találkozhatunk, ahol egy 5-10-15 tagú paraméterlista sokat rontana a kód olvashatóságán és növelné a hibák kockázatát.
- Ennek a tárgynak a keretein belül rövidebb programokat fogunk írni, ezért az első stílust várjuk el a házi feladatokban és laborvizsgán is.

Programozási stílus

continue, break, return, exit

- A felsorolt utasítások megszakítják az ismétlődő struktúrák „természetes folyását”, ilyen szempontból tekinthetőek a goto utasítással egy kategóriába.
- Ilyen szempontból nem tartják be a strukturált programozás elveit, melyeket E. W. Dijkstra fogalmazott meg 1972-ben.
- Túlzott, nem megfelelő használatuk ronthatja a programunk olvashatóságát.
- Viszont a modern programozási nyelvekben is jelen vannak és megfelelően használva egyszerűbb és hatékonyabb programokat írhatunk.
- A fentiek miatt a tárgy keretein belül használatuk nem tilos, de ajánlatos őket óvatosan kezelni, helyzettől függően a javítótanár indokolatlannak találhatja alkalmazásukat.

Algoritmika

Dr. Pătcaș Csaba

Laborfeladatok

Plagizálás

Stílus

IDE

Tömbök

Gyakori hibák

Tesztelés

Könyvészet

Maximális összegű tömbszakasz

Beépített függvények (pl. rendezés, bináris keresés) vagy adattípusok (pl. `set`, `map`) használata a következő feltételek teljesülésekor engedélyezett:

- nem az a feladat lényege, vagy nem vesződik el a feladat lényege a használat által
- a diák tudja és érti, hogy mi van mögötte
- nem romlik a megoldás időbonyolultsága vagy memóriabonyolultsága a használat által (pl. egy `sort` elrontja a lineáris időbonyolultságot)

A `pow` függvény használata több szempontból is problémás, ide tartoznak a pontossági és hatékonysági okok, ezért a programozási stílusra adott pontszámból legalább 5% levonást eredményez.

Programozási stílus

Egyéb alapelvek

- A kódot tördeljük (indentáljuk)! Több stílus létezik, válasszunk egyet! (lásd Syllabus oldalt Canvas-en)
- Megjegyzések: ideális esetben a kód „olvastatja magát”, de ez nem mindig lehetséges. Ilyenkor rövid megjegyzésekkel magyarázhatjuk az alprogram célját, a programblokk működését.
- Ha egy `if`-ben hiányzik valamelyik ág, az legyen az `else` ág!
- Az egymásba ágyazott `if`-ek következzenek a megvalósulás valószínűsége szerinti csökkenő sorrendben!
- A `for` ciklusokban ne módosítsuk a ciklusváltozót, de a kezdő- és végső értéket sem!
- Az alprogramokban ne módosítsuk a bemeneti paraméterként kapott változókat, még akkor sem ha ezeket érték szerint adtuk át!

Milyen fejlesztői környezetet (IDE) használjunk?

- Tetszőleges programozási nyelvben lehet dolgozni, de mivel a többség C / C++-ban fog, ezért az alábbiakban lesznek specifikusan erre vonatkozó megjegyzések.
- Minden diáknak ismernie kell és aktívan kell tudja használni a következő alapvető hibakeresési (debug) módszereket: step over, step into, step out, run to cursor, breakpoint, conditional breakpoint, watch, call stack.
- Nincs megszabott fejlesztői környezet, de olyat kell választani, amelyben a fentieket meg lehet valósítani. Pl. Visual Studio Community, Visual Studio Code, CLion
- A Codeblocks instabil és elavult, használata erősen ellenjavallott.

Tartsuk be a C/C++ nyelv standardjait!

- Ugyan egyes fordítók elfogadják a `void main()`-t is, a standard `int main()`-t ír elő, így használjuk ezt.
- Egy n elemű egész számokat tartalmazó tömböt soha ne deklaráljunk `int a[n]` formában!

Tömbök

Egy $n \leq 100$ hosszúságú tömböt többféleképpen deklarálhatunk C/C++-ban:

1. Statikus tömb: `int a[100];`
2. Változó hosszúságú tömb: `int a[n];`
3. Dinamikus tömb (C): `malloc, calloc, realloc` $\rightarrow$ `free`
4. Dinamikus tömb (C++): `a = new int[n];` $\rightarrow$ `delete`
5. Vector template: `vector <int> a(n);`

Előnyök:

- Egyszerű használat
- Az alap C része (nem kell include)
- Többdimenziós tömbök deklarálása egyszerű

Hátrányok:

- Mindig a maximális memóriát foglalja le
- Nincs hibakezelés (range checking)
- Nehéz hibakeresés (debugging)

Tömbök

Változó hosszúságú tömbök

Just don't

Algoritmika
Dr. Pătcaș Csaba
Laborfeladatok
Plagizálás
Stílus
IDE
Tömbök
Gyakori hibák
Tesztelés
Könyvészet
Maximális összegű tömbszakasz

Előnyök:

- Precízen kontrolálható a felhasznált memória mérete
- Legfennebb csak pár alap include-ra van szükség

Hátrányok:

- Nincs hibakezelés (range checking)
- Nagyon nehéz hibakeresés (debugging)
- Körülményes / időigényes ismeretlen darabszámú elemet egyenként hozzáadni
- Többdimenziós tömbök deklarálása, lefoglalása, felszabadítása nagyon körülményes

Tömbök

Vector template

Előnyök:

- Van hibakezelés (range checking)
- Debug módban könnyebb a hibakeresés
- Ismeretlen darabszámú elemet egyszerű egyenként hozzáadni (`a.push_back(x)`)

Hátrányok:

- Szükséges a STL (include-ok és memória)
- Az adatszerkezet maga is (jellemzően konstans méretű) plusz memóriát használ
- Az előnyökből adódoan valamivel (konstans szorzóval) lassúbb a többi változatnál
- Többdimenziós tömbök deklarálása relatív körülményes

Gyakori hibák a forráskódokban

Lokális változók kezdőérték nélkül

```
void fuggveny(int ertek_szerinti_parameter)
{
...
}
int main()
{
  int x;
  fuggveny(x); //HIBA: uninitialized local variable 'x' used
}
```

- A fenti hiba miatt nem lehet az adott programot lefordítani bizonyos kompilátorverziókkal, a DOMJudge sem fogja elfogadni.
- Más esetekben a program lefut, de különböző eredményt ad különböző környezetekben, mert a kezdőérték nélküli lokális változók értékére támaszkodik, amely C/C++-ban nem definiált.

Algoritmika

Dr. Pătcaș Csaba

Laborfeladatok

Plagizálás

Stílus

IDE

Tömbök

Gyakori hibák

Tesztelés

Könyvészet

Maximális összegű tömbszakasz

Gyakori hibák a forráskódokban

A vector template használata paraméterként

```
void fuggveny(...)
{
...
}

int main()
{
  vector <int> a;
  fuggveny(a);
}
```

Gyakori hibák a forráskódokban

A `vector` template használata paraméterként

Egy alprogram fejléce, melyet így hívunk meg többféleképpen is kinézhet:

1. `void fuggveny(vector <int> v)`
   Minden hívásnál másolat készül a tömbről (hasonlóan az érték szerinti átadáshoz), így az időbonyolultság és a memóriabonyolultság is romolhat.
2. `void fuggveny(vector <int> &v)`
   Csak a referenciát adjuk át (ami konstans méretű memóriát igényel), így minden módosítás a `v`-ben, megtörténik az `a`-ban is.
3. `void fuggveny(const vector <int> &v)`
   Fordítási hibát kapunk, ha módosítani próbáljuk `v`-t (pl. `push_back()` vagy `clear()` metódusokkal), de leolvasni továbbra is tudjuk (pl. egy adott pozíción lévő elemet lekérhetjük, vagy a `size()` metódust meghívhatjuk).

Gyakori hibák a forráskódokban

Logikai változó értékének ellenőrzése

Algoritmika

Dr. Pătcaș Csaba

Laborfeladatok

Plagizálás

Stílus

IDE

Tömbök

Gyakori hibák

Tesztelés

Könyvészet

Maximális összegű tömbszakasz

`if (változó == true)` vagy `if (változó == 1)` helyett elég csak

```
if (változó)
```

`if (változó == false)` vagy `if (változó == 0)` helyett elég csak

```
if (!változó)
```

Gyakori hibák a forráskódokban

Logikai értékek

- Ha egy `x` változó csak igaz vagy hamis értéket vehet fel, ne deklaráljuk `int`-nek!
- Értékadáskor használjuk a `true` és `false` konstansokat 1 és 0 helyett!

Tehát

`x = 1;` helyett `x = true;` és `x = 0;` helyett `x = false;`

és

`int x;` helyett `bool x;`

Hogyan teszteljük a programunkat?

- NEM billentyűzetről beírva / copy-paste-elve minden alkalommal a bemenetet. Egy félév során mennyi időt veszítesz ezzel vajon?
- Lehetséges megoldás az alábbiakat a program elejére tenni, majd felküldés előtt megjegyzésbe tenni, vagy törölni.

```
freopen("a.in", "r", stdin);
freopen("a.out", "w", stdout);
```

A modernebb fordítóprogramok reklamálni fognak a `freopen` használata miatt, erre egy megoldás a `_CRT_SECURE_NO_WARNINGS` használata (Visual Studioban: Project - Properties - C/C++ - Preprocessor - Preprocessor Definitions).

Hogyan teszteljük a programunkat?

Valamivel elegánsabb megoldás a saját gépünkön definiálni egy makrót (Visual Studioban: Project - Properties - C/C++ - Command Line - `/DSAJAT_GEP`), így nem kell mindig módosítani a kódot beküldés előtt:

```
#ifdef SAJAT_GEP
freopen("a.in", "r", stdin);
freopen("a.out", "w", stdout);
#endif
```

Hogyan gyártsunk teszteket?

Ötletek

- Extrém kicsi tesztek (pl. $n = 0, 1, 2$)
- Extrém nagy tesztek (pl. maximális n, az eredmény nem fér be az `int` típusba)
- Speciális struktúrájú bemenetek, melyekre az eredmény könnyen ellenőrízhető (pl. egyenlő számokat tartalmazó számsorozatok)
- Kézzel gyártott kisebb tesztek
- A megoldási algoritmusra specifikusan gyártott tesztek („átlátszó doboz módszerét” alkalmazva - erről később)
- Véletlenül generált tesztek
- Ha „minden kötél szakad”: *stress testing*

Tartalom

1. Laborfeladatokra vonatkozó alapszabályok
   - Plagizálás
   - Programozási stílus
   - Fejlesztői környezet
   - Tömbök
   - Gyakori hibák
   - Tesztelés
2. Könyvészet
3. Maximális összegű tömbszakasz

Miért van szükség könyvészetre?

Könyvészet

Egyetemi jegyzet

Algoritmika

Dr. Pătcaș Csaba

Laborfeladatok

Plagizálás

Stílus

IDE

Tömbök

Gyakori hibák

Tesztelés

Könyvészet

Maximális összegű tömbszakasz

Ionescu, Klára. *Bevezetés az algoritmikába.* Kolozsvári Egyetemi Kiadó, 2007.

Könyvészet

A nagy klasszikusok

- CLR: Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest *Algoritmusok*. Műszaki Könyvkiadó, Budapest, 1997.
- CLRS: Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Új algoritmusok*. Scolar kiadó, Budapest, 2003.
- TAOCP: Knuth, D. *Arta programării calculatoarelor*, vol. III, Sortare și Căutare, Editura Teora, București.

Könyvészet

Az alulírottól

- Ionescu, Klára, and Pătcaș, Csaba. *Algoritmusok hatékonyságának növelése a bináris keresés elvének alkalmazásával.* Műszaki szemle, 43(3):7–14, 2008.
- Pătcaș, Csaba, and Ionescu, Klára. *Algorithmics of the knapsack type tasks.* Teaching Mathematics and Computer Science INFODIDACT:37–71, 2008.
- Csaba, Pătcas, *Informatikafeladatok megoldása haladó módszerekkel (Feladatgyűjtemény)*, Kolozsvári Egyetemi Kiadó, 2019.

Könyvészet

A „szomszédból”

- *Gazeta de informatică*, Computer press Agora, Tg. Mureș, 1993-2004.
- Kátai Zoltán, *Algoritmusok felülnézetből*, Scientia Kiadó, Kolozsvár, 2007.
- Kátai Zoltán, *Algoritmustervezési stratégiák*, Scientia Kiadó, Kolozsvár, 2020.

Könyvészet

Steal this book!

Algoritmika

Dr. Pătcaș Csaba

Laborfeladatok

Plagizálás

Stílus

IDE

Tömbök

Gyakori hibák

Tesztelés

Könyvészet

Maximális összegű tömbszakasz

Jeff Erickson, Algorithms. `http://algorithms.wtf`

Könyvészet

Figyelem!

Az olyan oldalak, melyeket önkéntesek írnak és nincsenek ellenőrizve tudományos szempontból (pl. Wikipedia, geeks4geeks, Stack Overflow, tutorialspoint) számos hibát tartalmazhatnak és nem számítanak megbízható forrásnak, inkább csak kiindulópontnak jók! Személyesen, több hibás algoritmust is kijavítottam az angol Wikipédián, de hagytam bőven másoknak is. A ChatGPT még ezeknél is rosszabb.

Tartalom

1. Laborfeladatokra vonatkozó alapszabályok
   - Plagizálás
   - Programozási stílus
   - Fejlesztői környezet
   - Tömbök
   - Gyakori hibák
   - Tesztelés
2. Könyvészet
3. Maximális összegű tömbszakasz

Maximális összegű tömbszakasz

Feladat

Adott egy n egész számból álló számsorozat, amely biztosan tartalmaz legalább egy pozitív számot. Írjunk programot, amely meghatározza azt a tömbszakaszt, amelynek összege a lehető legnagyobb.

- Egy tömb részsorozatának nevezzük a tömb egy olyan rendezett részhalmazát, mely nem feltétlenül egymás utáni pozíciókon található elemeket tartalmaz.
  Példa: 1, 2, -6, 3, 4, 5, -2, 10, -5, -6
- Egy tömbszakasz olyan részsorozat, amely csak egymás utáni pozíciókon található elemeket tartalmaz.
  Példa: 1, 2, -6, 3, 4, 5, -2, 10, -5, -6
- Más elnevezéseket ne használjunk a továbbiakban (pl. vizsgán)!

Leghatékonyabban hogyan tudod megoldani a feladatot?

Algoritmika

Dr. Pătcaș Csaba

Laborfeladatok

Plagizálás

Stílus

IDE

Tömbök

Gyakori hibák

Tesztelés

Könyvészet

Maximális összegű tömbszakasz

Maximális összegű tömbszakasz

Első megoldás

- Generáljuk az a tömb összes lehetséges tömbszakaszát, mindegyiknek kiszámoljuk az összegét és kiválasztjuk a legnagyobbat.
- A `maxOsszeg` változóba számoljuk ki a maximális összegű tömbszakasz összegét, ezt mindig frissítjük mikor jobb megoldást találtunk az eddiginél.
- A `bal` változóval jelöljük az aktuálisan generált tömbbszakasz legbaloldalibb elemének indexét.
- A `jobb` változóval jelöljük az aktuálisan generált tömbbszakasz legjobboldalibb elemének indexét.
- Az `osszeg` változóba számoljuk ki az aktuálisan generált tömbszakasz hosszát.
- Az `i` változóval járjuk be az aktuálisan generált tömszakaszt az összeg kiszámításához.

Forráskód: `maxTombszakasz1.cpp`

Maximális összegű tömbszakasz

Második megoldás

- Kiszámoljuk a részösszegek `sum` tömbjét, melynek `i.` eleme az a tömb első `i` elemének összegével lesz egyenlő.
- Észrevesszük, hogy `a[bal..jobb] = sum[jobb] - sum[bal - 1]`
- Átírjuk az első megoldást úgy, hogy megszabadulunk a belső `for` ciklustól, melyet helyettesítünk a fenti képlettel.

Forráskód: `maxTombszakasz2.cpp`

Maximális összegű tömbszakasz

Harmadik megoldás

- Megszabadulhatunk a `sum` tömbtől és a belső (`i` ciklusváltozót használó) ciklustól is, ha észrevesszük a következőt.
- `a[bal..jobb] = a[bal..jobb-1] + a[jobb]`
- Ezt felhasználva, az `osszeg` változót frissíthetjük egyetlen összeadással minden lépésben mikor a `jobb` változó növekszik és nem kell a teljes intervallumra újraszámolni.

Forráskód: `maxTombszakasz3.cpp`