NYELVFÜGGETLEN PROGRAMOZÁS

1. forduló

A kategória támogatója: SAP Hungary Kft.

Ismertető a feladathoz

Kérjük, hogy a feladatlap indítása előtt mindenképp olvasd el az alábbi útmutatót:

- MINDEN kérdésre **van helyes válasz**.
- Olyan kérdés **NINCS**, amire az összes válasz helyes, ha mégis az összes választ bejelölöd, arra a feladatra automatikusan 0 pont jár.
- A **radio button-os kérdésekre egy helyes válasz van**.
- **Ha lejár a feladatlap ideje, a rendszer AUTOMATIKUSAN** beküldi azt az addig megjelölt válaszokkal.
- Azokat a feladatlapokat, amelyekhez **csatolmány** tartozik, javasoljuk **NEM mobilon** elindítani, erre az érintett feladatlapok előtt külön felhívjuk a figyelmet.
- Az **adatbekérős feladatokra NEM jár részpontszám**, csak a feleletválasztósakra.
- **Helyezéseket a 4. forduló után mutatunk**, százalékos formában: adott kategóriában a TOP 20-40-60%-hoz tartozol.
- **Badge-ket** szintén a 4.forduló után kapsz majd először.
- Ha egyszerre több böngészőből, több ablakban vagy több eszközről megnyitod ugyanazt a feladatlapot, **nem tudjuk vállalni** az adatmentéssel kapcsolatban esetlegesen felmerülő anomáliákért a felelősséget!
- A hét forduló során az egyes kategóriákban (de nem feltétlenül mindegyikben) **könnyű-közepes-nehéz kérdésekkel** egyaránt találkozhatsz majd.

Jó versenyzést kívánunk!

1.forduló

A feladatlap több csatolmányt is tartalmaz, ezért a megoldását asztali gépen javasoljuk!

- Minden fordulóban két algoritmikus feladat lesz.
- Minden feladat esetében 5 "éles" inputra kell előállítanod az outputokat, amelyeket aztán a versenyfelületen a megfelelő szövegmezőbe kell illesztened.
- A megoldásod ellenőrzését segítendő, minden feladathoz tartozik 2 db példa input és output.
- Pl. ha egy feladat címe "Cica", akkor a cica.peldaX.in.txt-ben lesz a példa input, a cica.peldaX.out.txt-ben pedig az ehhez tartozó példa output (X egy egész szám). A cicaX.in.txt fájlokban lesznek a pontokért megoldandó inputok, ahol X: 1..5.
- Mindezeket a txt fájlokat a csatolt tömörített archívum tartalmazza, melyet a feladatsor indítása után tölthetsz le.
- A megoldásokat bármilyen programnyelven elkészítheted.
- A forráskódot nem kell beküldeni, csak az outputokat.
- Ne add fel, ha nem tudod a megadott időkereten belül mind a két feladatot megoldani, a kategória nem feltétlenül könnyű. :)

Jó szórakozást!

Felhasznált idő: 39:59/40:00 **Elért pontszám:** 0/16

Indítás utáni csatolmányok

1. feladat 0/1 pont

Kódzár

Móricka a logikai áramkörökről tanult az iskolában. Elhatározta, hogy kódzárat készít biciklije lezárására. A kódzár egy N bináris számjegyből álló jelszóval működik, melynek számjegyeit N db kapcsolóval állíthatjuk be. A jelszó ellenőrzését XOR kapuk végzik. A XOR kapu olyan digitális áramköri elem, melynek két bemenete és egy kimenete van. A kimenet pontosan akkor lesz 1, ha a két bemenet egyike 0, a másika pedig 1. Ellenkező esetben (két db 0, vagy két db 1-es bemenet esetén) a kapu kimenete 0 lesz. Az egyik XOR kapunak kitüntetett szerepe van, ugyanis a zár pontosan akkor nyílik ki, ha ennek a kapunak a kimenete 1.

A kódzár áramköri leírását Móricka egy szigorúan titkos szövegfájlban tárolja, melynek sorait 1-től számozzuk. A fájl első sora N-et tartalmazza, azaz a bináris jelszó hosszát. A fájl további sorai egy-egy XOR kaput definiálnak. Minden ilyen sorban szóközzel elválasztva 3 db azonosító található: az adott kapu neve, valamint a két inputjának definíciója. Mindkét inputdefiníció vagy egy 0 és N-1 közötti egész szám lehet (N-1-et beleértve), mely azt jelenti, hogy az adott input a beírt jelszó adott sorszámú számjegye lesz (balról számozzuk a számjegyeket, 0-tól kezdve); vagy egy másik XOR kapu azonosítója, mely esetben az adott inputra a hivatkozott XOR kapu outputja van rákötve. Semelyik XOR kapu sem hivatkozhat az őt követően definiált kapuk outputjaira, csak a megelőzőekére. A zárat megvalósító áramkör outputját a legutolsó sorban definiált kapu állítja elő. Ha ez 1, a zár kinyílik, különben nem.

Móricka sajnos elég hamar elfelejtette a jelszót. Segítsünk neki: az áramköri leírás alapján fejtsük vissza azt a jelszót, amely nyitja a zárat, és a zárat nyitó jelszavak közül bináris számként értelmezve a legkisebb. Ha ilyen nem létezik, akkor a NINCS szöveg legyen az output (csupa nagybetűkkel).

Mi a **kodzar1.in.txt**-hez tartozó output?

Válasz

100

A helyes válasz:

00001

Magyarázat

Minden kapura igaz az, hogy a bemeneti bitek valamely adott részhalmazába eső 1-esek számának paritását számolja ki: 1-et ad ki, ha páratlan sok 1-es van ott, és 0-t, ha páros. Ez abból következik, hogy a XOR művelet kommutatív és asszociatív, és saját magával XOR-ozva a 0 és 1 is 0-t ad, tehát ha két XOR-kapu outputját összeXORoljuk, akkor a két kapuhoz tartozó bit-részhalmazok szimmetrikus differenciája fogja meghatározni azt a részhalmazt, amelyben az új kapu meghatározza az 1-esek számának paritását.

Az áramkör outputja tehát a bemeneti bitek egy adott részhalmazának összeXORolásával kapható. A maradék bitek nem számítanak, ezek tehát lehetnek 0-k. A releváns bitek közül elég csak a legutolsót 1-esre állítani. Így kapható a legkisebb szám, amely nyitja a zárat. Végig kell tehát menni növekvő sorrendben a 2-hatványokon, azaz a one-hot vektorokon (az 1-es számnak megfelelő bináris vektorral kezdve). Ha ezek közül egyik jelszó sem vált be, akkor az utolsó kapuhoz tartozó részhalmaz az üres halmaz, tehát a zár egyetlen kódszóval sem nyitható ki, ilyenkor tehát a NINCS-et fogjuk kiírni.

```python
#!/usr/bin/env python3

def tryToInt(s:str):
  try:
    return int(s)
  except ValueError:
    return s

def solveFile(fn:str, fOut):
```

```
  gates = {} # név -> inputok
  gateOrder = [] # előfordulási sorrend a fájlban
  with open(fn) as f:
    n = int(f.readline().strip())
    assert n >= 1
    for line in f:
      line = line.strip()
      if not line:
        continue
      gateName, input1, input2 = line.split()
      assert gateName not in gates
      gates[gateName] = tryToInt(input1), tryToInt(input2)
      gateOrder.append(gateName)

  result = "NINCS"
  # One-hot vektorok felsorolása "növekvő" sorrendben
  # (csak 1 db bit 1-es, az, amelyik balról az iHotInput-adik helyen van, 0-tól számozva a biteket)
  for iHotInput in range(n-1, -1, -1):
    gateOutputs = {}

    for iBit in range(n):
      gateOutputs[iBit] = (iBit == iHotInput)

    for gateName in gateOrder:
      input1, input2 = gates[gateName]
      input1 = gateOutputs[input1]
      input2 = gateOutputs[input2]
      assert input1 in (0, 1)
      assert input2 in (0, 1)
      gateOutputs[gateName] = input1 != input2

    if gateOutputs[gateOrder[-1]] == 1:
      result = "0" * iHotInput + "1" + "0" * (n-1 - iHotInput)
      assert len(result) == n
      break

  message = "Output for %s: %s" % (fn, result)
  print(message)
  fOut.write(message+"\n")

  if "pelda" in fn:
    fnPeldaOut = fn.replace(".in.", ".out.")
    assert fnPeldaOut != fn
    with open(fnPeldaOut, "w") as fPeldaOut:
      fPeldaOut.write(str(result))

def main():
  with open("out.txt", "w") as fOut:
    for i in range(1, 6):
      solveFile("kodzar%s.in.txt" % (i,), fOut)
    for i in range(1, 3):
      solveFile("kodzar.pelda%s.in.txt" % (i,), fOut)

if __name__ == "__main__":
  main()
```

2. feladat 0/1 pont

Mi a **kodzar2.in.txt**-hez tartozó output?

Válasz

A helyes válasz:

00000000000000000000000001000000

Magyarázat

Ld. fent.

3. feladat 0/1 pont

Mi a **kodzar3.in.txt**-hez tartozó output?

Válasz

A helyes válasz:

00000000000000000000000000000000000000000000000000000000000000010

Magyarázat

Ld. fent.

4. feladat 0/2 pont

Mi a **kodzar4.in.txt**-hez tartozó output?

Válasz

A helyes válasz:

00000000000000000000000000000000000000000000000000000000000000000000000000000001

Magyarázat

Ld. fent.

5. feladat 0/3 pont

Mi a **kodzar5.in.txt**-hez tartozó output?

Válasz

A helyes válasz:

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010

Magyarázat

Ld. fent.

6. feladat 0/1 pont

Kavics

Adott N db kavics, melyek egyenlő távolságra találhatóak egymástól egy egyenes mentén: bármely két szomszédos kavics távolsága 1 méter. Pistike a kavicsokat K db kupacba szeretné rendezni a távirányítós markolójával (K <= N). Minden kupac egy-egy meglévő kavics helyén lesz. Ha az i koordinátán lévő m tömegű kavicsot át szeretnénk szállítani a j koordinátán található kupacba, akkor a markoló m * abs(i-j) egységnyi energiát fogyaszt. Számoljuk ki, hogy minimálisan mekkora energiafogyasztással valósítható meg a kavicsok kupacokba rakása!

A bemenet első sora két számot tartalmaz, szóközzel elválasztva: az N ill. K számokat. A második sorban N db nemnegatív egész szám található, szóközzel elválasztva: az i. szám (1-től számozva) az i koordinátán lévő kavics tömegét jelöli.

A kimenet egyetlen nemnegatív egész szám legyen, az optimális átrendezés energiafogyasztása.

Mi a **kavics1.in.txt**-hez tartozó output?

Válasz

100

A helyes válasz:

54566

Magyarázat

Dinamikus programozás, a második kupacot léptetjük, és a második kupactól kezdődő kavicsok pakolási energiafogyasztását egy másik részfeladat megoldásából vesszük. A nehézség az, hogy hogyan számoljuk ki az első és második kupac közé eső kavicsok mozgatási költségét (a példakódban ez az extraCost változóban van tárolva). A példakód azt használja ki, hogy ha a második kupacot eggyel léptetjük, akkor némelyik kavicsok költsége nem változik, némelyeké meg a tömegükkel nő, de ők egy intervallumba esnek, tehát kumulatív összeg számításával gyorsan meg lehet oldani extraCost frissítését.

```
#!/usr/bin/env python3

def calcCumulativeSum(w:list):
  result = []
```

```
  s = 0
  for elem in w:
    s += elem
    result.append(s)

  assert len(result) == len(w)
  assert result[-1] == sum(w)

  return result

def solve(weights:list, kMax):
  assert weights
  assert 1 <= kMax <= len(weights)

  cumulativeSum = calcCumulativeSum(weights)

  dp = {}
  for k in range(1, kMax+1):
    for startIndex in range(len(weights)-k+1):
      if k == 1:
        cost = 0
        for i in range(startIndex+1, len(weights)):
          cost += (i - startIndex) * weights[i]
        dp[k, startIndex] = cost
      else:
        # extraCost: az 1. és 2. kupac közötti kavicsok szállítási költsége.
        extraCost = 0
        minCost = 1e100

        # index2: a 2. kupac helyének indexe.
        for index2 in range(startIndex+1, len(weights)-(k-1) + 1):
          cost = extraCost + dp[k-1, index2]
          if cost < minCost:
            minCost = cost
          extraCost += cumulativeSum[index2] - cumulativeSum[(startIndex + index2) // 2]

        dp[k, startIndex] = minCost

  # Most k=kMax esetén ki kell számolni a minimumot aszerint, hogy hová rakjuk az 1. kupacot.
  result = 1e100
  extraCost = 0
  for startIndex in range(len(weights)-kMax+1):
    cost = extraCost + dp[kMax, startIndex]
    if cost < result:
      result = cost
    extraCost += cumulativeSum[startIndex]

  return result

def solveFile(fn:str, fOut):
  with open(fn) as f:
    n, k = map(int, f.readline().strip().split())
    assert n
    assert 1 <= k <= n
    weights = list(map(int, f.readline().strip().split()))
    assert len(weights) == n

  result = solve(weights, k)

  message = "Output for %s: %s" % (fn, result)
  print(message)
  fOut.write(message+"\n")

  if "pelda" in fn:
    fnPeldaOut = fn.replace(".in.", ".out.")
    assert fnPeldaOut != fn
```

```
    with open(fnPeldaOut, "w") as fPeldaOut:
      fPeldaOut.write(str(result))

def main():
  with open("out.txt", "w") as fOut:
    for i in range(1, 6):
      solveFile("kavics%s.in.txt" % (i,), fOut)
    for i in range(1, 3):
      solveFile("kavics.pelda%s.in.txt" % (i,), fOut)

if __name__ == "__main__":
  main()
```

7. feladat 0/1 pont

Mi a **kavics2.in.txt**-hez tartozó output?

Válasz

A helyes válasz:

241378

Magyarázat

Ld. fent.

8. feladat 0/1 pont

Mi a **kavics3.in.txt**-hez tartozó output?

Válasz

A helyes válasz:

496404

Magyarázat

Ld. fent.

9. feladat 0/2 pont

Mi a **kavics4.in.txt**-hez tartozó output?

Válasz

A helyes válasz:

1771270

Magyarázat

Ld. fent.

10. feladat 0/3 pont

Mi a **kavics5.in.txt**-hez tartozó output?

Válasz

A helyes válasz:

5125600

Magyarázat

Ld. fent.