Alapvető algoritmusok

9. előadás

Dr. Pătcaș Csaba

BABEŞ-BOLYAI TUDOMÁNYEGYETEM
Matematika és Informatika Kar

Tartalom

Feladattípusok

- Az eddigi feladatokban az összes megoldást kellett generálnunk, de a backtracking ennél általánosabb módszer, más típusú feladatokra is alkalmazható.
- Említettük, hogy azoknál a feladatoknál, amelyek egyetlen megoldást kérnek, a kiírás után kiléphetünk. Ebbe a kategóriába tartoznak a `MindenTávolság` és `Sudoku` feladatok.
- A visszalépéses keresést alkalmazhatjuk optimumkeresési feladatokra is, ebben a kategóriában említjük majd az `ABCLefedés`, `Kivonások` és a `Dáma 1D` feladatokat.
- Számolási feladatokat is megoldhatunk backtrackinggel: az eredményt számolhatjuk egyesével (pl. hányféleképpen helyezhetünk fel n királynőt egy $n \times n$-es sakktáblára), vagy nagyobb lépésekben (pl. számoljuk ki egy adott szám osztóinak összegét). Ide tartoznak még a `HuszárFutár` és `Díjazás` feladatok.

Kivonások

Feladat

Egy természetes számon végrehajthatjuk a következő műveletet: válasszuk ki a szám valamely nem-nulla számjegyét és vonjuk azt ki a számból. Így például a 40451-ből kaphatunk 40450-et, 40447-et, vagy 40446-ot. Határozzuk meg egy adott n számra, hogy legkevesebb hány lépésben változtatható nullává!

- Minden lépésben meghatározzuk a lehetséges lépések halmazát, sorban végigpróbáljuk őket és rekurzívan meghívjuk az alprogramot az így kapott számra.
- Vegyük észre, hogy az így kapott fa minden ága megoldáshoz vezet, de a levelek különböző mélységben helyeszkednek el. Például a 19-et átalakíthatjuk nullává 11 lépésből, de 3 lépésből is.

Megjegyzés: A feladat megoldásához nem szükséges a visszalépéses keresés módszere, ez csak egy didaktikai példa

Kivonások

Első változat

```
ALGORITMUS Kivonások1(n, lépés, megoldás)
  HA (n = 0) akkor
    megoldás = min(megoldás, lépés)
  KÜLÖNBEN
    SzámjegyekHalmaza(n, halmaz, halmazMéret)
    MINDEN i = 1, halmazMéret végezd el:
      Kivonások1(n - halmaz[i], lépés + 1, megoldás)
    VÉGE(Minden)
  VÉGE(Ha)
VÉGE(Algoritmus)
```

Kezdetben `megoldás = VÉGTELEN` (cím szerint átadott paraméter), hívás:
`Kivonások1(n, 0, megoldás)`

Kivonások

Második változat

Ha meghaladtuk az eddig talált minimális megoldást, már biztosan nem találunk jobbat, így nincs értelme folytatnunk. Így megszabadulunk a `min` függvény hívásától is.

```
ALGORITMUS Kivonások2(n, lépés, megoldás)
  HA (lépés >= megoldás)
    VISSZATÉRÍT
  VÉGE(Ha)
  HA (n = 0) akkor
    megoldás = lépés
  KÜLÖNBEN
    ... (ugyanaz mint az előbb)
```

Kivonások

Harmadik változat

A legnagyobb érték amivel egy lépésben csökkenhet a szám egyenlő a legnagyobb számjeggyel, ami 9. Vagyis a legjobb esetben is $[n/9]$ lépésre van szükségünk, hogy eljussunk a nullához. Ha ebben az optimista feltételezésben is meghaladjuk az eddigi legjobb megoldást, nincs értelme folytassuk, mert azt jelenti, hogy olyan ágra kerültünk, amelyben minden levél túl mélyen van.

```
ALGORITMUS Kivonások3(n, lépés, megoldás)
  HA (lépés + (n / 9) >= megoldás)
    VISSZATÉRÍT
  VÉGE(Ha)
  HA (n = 0) akkor
    megoldás = lépés
  KÜLÖNBEN
    ... (ugyanaz mint az előbb)
```

Kivonások

Negyedik változat

Nagyobb eséllyel jutunk kevesebb lépésből álló megoldáshoz, ha nagyobb számjegyeket választunk, ezért érdemes az ágakat ennek megfelelő sorrendben bejárni, így az eddigi optimalizálások is többször alkalmazhatóak lesznek majd.

```
ALGORITMUS Kivonások4(n, lépés, megoldás)
  HA (lépés + (n / 9) >= megoldás)
    VISSZATÉRÍT
  VÉGE(Ha)
  HA (n = 0) akkor
    megoldás = lépés
  KÜLÖNBEN
    SzámjegyekHalmaza(n, halmaz, halmazMéret)
    RendezCsökkenőbe(halmaz, halmazMéret)
    ... (ugyanaz mint az előbb)
```

Dáma 1D

Feladat

Adott egy n elemből álló sorozat, melynek elemei a 0, 1, vagy 2 értéket vehetik fel. A 0 szabad pozíciót jelöl, az 1 világos bábut és a 2 sötét bábut. A világos bábuk csak jobbra léphetnek, a sötétek csak balra. Egy bábu léphet egyetlen pozíciót, ha üres helyre kerül, vagy kettőt, ha ezzel átugrik egy tetszőleges színű bábut és üres pozícióra kerül. Határozzuk meg a minimális lépések számát, amellyel olyan konfigurációba jutunk, amelyben a baloldali pozíciókat csak sötét bábuk foglalják el és a jobboldali pozíciókat csak világos bábuk (tehát az üres pozíciók középre kerülnek).

Példa: $n = 5$

1 2 0 1 0

0 2 1 1 0

0 2 0 1 1

2 0 0 1 1

Dáma 1D

Elemzés

- A `Kivonások` feladathoz hasonlóan itt is egy optimumkeresési feladattal van dolgunk.
- Erre a feladatra is adaptálhatóak az ott látott optimalizálási ötletek.
- Ha több lépést hajtottunk végre, mint az eddigi legjobb megoldás, nincs értelme tovább menni.
- Egy bábu legtöbb két pozíciót haladhat egy lépésben. Ezt az észrevételt felhasználva adhatunk egy alsó becslést a szükséges lépések számára.
- A fenti kezdő konfigurációban [1 2 0 1 0] két világos és egy sötét bábu volt, vagyis egy sötét bábu legalább az 1. indexig kell lépjen és egy világos legalább a 4. indexig. Felírhatjuk, hogy melyik pozíción lévő bábu milyen messze van a legközelebbi végcéljától: [3 1 0 0 0]. Mivel egy lépésben két pozíciót is elmozdulhat egy bábu, a minimális lépések száma: [2 1 0 0 0], vagyis 3.
- A rekurzív hívásokat érdemes olyan sorrendben végrehajtani, hogy először azokat a lépéseket hajtjuk végre, ahol két pozíciót ugrik egy bábu, mivel így nagyobb eséllyel jutunk jobb megoldáshoz.

Sudoku

Feladat

Adott egy részben kitöltött 9×9 méretű négyzetrács. Töltsük ki úgy a megmaradt mezőket, hogy minden sorban, minden oszlopban és mind a 9 darab 3×3-as kis négyzetben, minden 1 és 9 közötti természetes szám pontosan egyszer jelenjen meg!

- A feladat egyetlen helyes kitöltést kér, így az első kiíratás után megállhatunk.
- Az első ötlet az lenne, hogy sorról-sorra haladva minden kitöltetlen mezőbe megpróbáljuk beírni az összes értéket 1-től 9-ig, majd a végén ellenőrizzük, hogy teljesülnek-e a belső feltételek.
- Ez tulajdonképpen egy nyers erő (brute force) megközelítés lenne, visszalépéses kereséssel írhatunk hatékonyabb megoldást is.
- Vegyük észre, hogy az eredmény természetes kódolásához nem elegendő egy egyszerű számsorozat.

- Minden üres mezőre számontartjuk, hogy milyen értékeket írhatunk belé, figyelembe véve a részlegesen kitöltött négyzetrácsot.
- Ahelyett, hogy mind a 9 lehetőséget végigpróbálnánk, csak ezekből az értékekből választunk. Ezzel tulajdonképpen beépítettük a belső feltételeket a folytatási feltételekbe és amint megoldáshoz jutottunk, az helyes is lesz.
- Megfigyelhetjük, hogy a backtracking által felépített fának van néhány általános tulajdonsága: az azonos szinten lévő csúcsok leszármazottainak a száma változó (1-től 9-ig) és csak az ágak nagyon kis százaléka vezet végeredményhez.

- A cél az, hogy a fában minél hamarább építsünk fel egy maximális hosszúságú ágat, amely végeredményhez vezet.
- Ehhez az kell, hogy az első végeredményt megelőzően, minél kevesebb állapotot járjunk be.
- Ezt úgy érhetjük el, hogy a végeredménynek megfelelő ágtól „balra” eső (korábban felépített) részfa minél kevesebb csúcsból álljon, vagyis minél „keskenyebb” és minél „alacsonyabb” legyen.

Sudoku

Hogyan tehetjük a fát „keskenyebbé”?

- Megpróbáljuk minimalizálni egy csúcs gyerekeinek a számát. A `Sudoku` feladatánál ennek érdekében tekintünk a 9 gyerek helyett csak annyit, ahány értéket az adott mezőre írhatunk, de a `Permutációk` vagy a `Királynők` feladatánál is alkalmaztuk ezt az elvet a folytatási feltételek megfogalmazásakor.
- Nem mindegy, hogy egy csúcs gyerekeit milyen sorrendben járjuk be, a `Sudoku` feladatánál nem feltétlenül a leghatékonyabb választás az üres mezőkbe növekvő sorrendbe beírni a lehetséges értékeket.
- Általában véve érdemesebb a kisebb részfákat bejárni először (ha mindegyik részfa egyenlő valószínűséggel visz megoldáshoz).
- Például ha egy csúcsnak van két gyereke, az egyik részfája 100 állapot bejárását feltételezi, a második 10 állapot bejárását és a két ág 50%-50% eséllyel visz megoldáshoz, akkor jobban járunk ha a 10 állapot bejárásával kezdjük.

Sudoku

Hogyan tehetjük a fát „alacsonyabbá” és „keskenyebbé”?

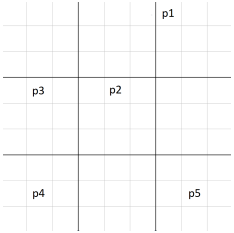
- Az sem mindegy, hogy mit rendelünk a fa különböző szintjeihez, az $M_1, \ldots, M_n$ halmazokból milyen sorrendben választunk értéket, nem feltétlenül az $1, \ldots, n$ sorrend a leghatékonyabb.
- Például a `Sudoku` feladatánál az üres mezőket ne sorról sorra töltsük ki, hanem kezdjük azokkal, amelyekbe a legkisebb számú értéket írhatjuk. Így potenciálisan hamarabb bejárjuk azokat az ágakat, amelyek nem vezetnek eredményhez és kevesebb konfiguráció kigenerálása után találunk megoldást.

Sudoku

Hogyan tehetjük a fát „alacsonyabbá” és „keskenyebbé”?

Példa:

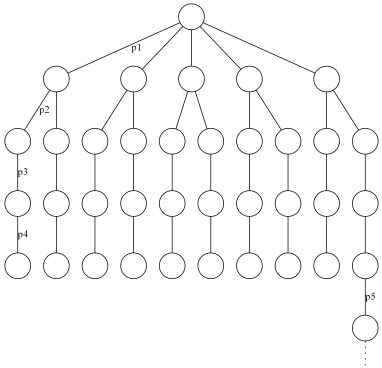
- Tegyük fel, hogy a Sudoku táblán keressük az értéket a $p_1, \ldots, p_5$ üres mezőkhöz.
- A p_1 mező 5 lehetséges értéket kaphat még, a $p_2, \ldots, p_4$ mezők mindegyike két lehetséges értéket, és a p_5 mező csak egy értéket kaphat.
- A tábla konfigurációja olyan, hogy azonos soron vagy oszlopon vannak egymással a p_2 és a p_3, a p_3 és a p_4, valamint a p_4 és a p_5. Vagyis ha valamelyikük értéket kap, a másik eggyel kevesebb értéket kaphat.



Sudoku

Hogyan tehetjük a fát „alacsonyabbá” és „keskenyebbé”?

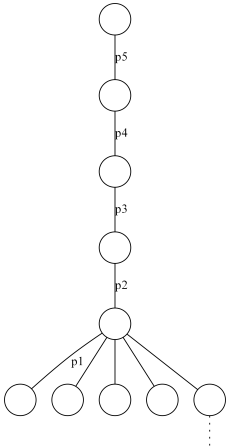
- Feltételezzük, hogy csak akkor jutunk megoldáshoz, ha a p_1 az 5. ágnak megfelelő értéket kapja és a p_2, p_3 és p_4 a második ágnak megfelelő értéket.
- Ha $p_1, \ldots, p_5$ sorrendben rendeljük a mezőket a fa szintjeihez, a következő fát kapjuk mielőtt a megoldáshoz vezető értékeket rendeljük az öt mezőhöz.



Sudoku

Hogyan tehetjük a fát „alacsonyabbá” és „keskenyebbé”?

- Ha $p_5, \ldots, p_1$ sorrendben rendeljük a mezőket a fa szintjeihez, 31 állapot helyett csak 4-et kell feleslegesen bejárnunk, mielőtt a helyes ágon megyünk tovább.



Minden távolság

Feladat

Ismervén a távolságot minden pontpár között n darab Ox tengelyen található pont közül, határozzuk meg a pontok nemnegatív koordinátáit, tudván, hogy az origó biztos közöttük van!

- A legnagyobb távolság biztosan csak egyszer fog szerepelni, ez megadja az origótól legtávolabbi pont *max* helyzetét.
- Ahhoz, hogy több száz pontra is időben fusson a program, szükség van a `Sudoku` feladatánál látott optimalizálási ötletekre.

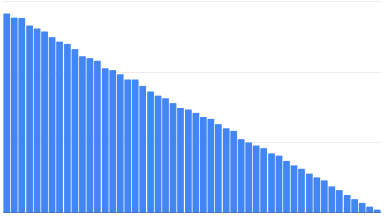
Minden távolság

Milyen sorrendben érdemes a fa szintjeit elképzelni?

- Intuitíven „érezzük”, hogy érdemesebb a nagyobb távolságokkal kezdeni.
- Ha egyenletes eloszlású koordináták közötti távolságok gyakoriságát nézzük, könnyen látható, hogy a legnagyobb értékek a legritkábbak.

Minden távolság

Milyen sorrendben érdemes a fa szintjeit elképzelni?



- Ebből az $\frac{n(n-1)}{2}$ értékből, minket csak n érdekel, mivel ezek egyenletesen oszlanak el, a legnagyobb értékek közül választjuk ki őket a legnagyobb eséllyel.
- Észrevesszük, hogy ha létezik egy x koordinátájú pont, akkor nem csak az x, hanem a $max - x$ is szerepel távolságok között, vagyis ezekből az értékekből is n darab található a hisztogramon. Ezek figyelmevételével duplára növelhetjük az esélyeinket arra, hogy egy helyes x pozíciót találunk.

Backtracking a síkban

- Az előző feladatoknál már láttuk, hogy az eredmény kódolása nem mindig lehetséges egy egyszerű számsorral.
- Például amikor egy kétdimenziós tömbben való adatokat kell feldolgoznunk, abban kell utakat keressünk, minden szinten egy (x, y) koordinátapárt kell majd tárolnunk (esetleg más adatokkal egyetemben).
- Ekkor tekinthetjük úgy is, hogy az $M_1 \times M_2 \times \ldots M_n$ Descartes-szorzat minden eleme $M_k = (x_k, y_k, ?)$ alakú.
- Ilyenkor a módszer neve backtracking a síkban.

Labirintus

Feladat

Egy labirintust egy $n \times m$ méretű `A` mátrixszal kódolunk, amelyben az 1-es érték jelzi a folyosót és a 0-ás értékek a falat. Egy személyt ejtőernyővel leengednek az (x, y) pozícióra, amelyen biztosan folyosó található. Írjuk ki a labirintusból kivezető összes utat, tudván, hogy egy út nem érintheti kétszer ugyanazt a helyet. A labirintusból a tömb szélén léphetünk ki, egy adott mezőről a négy szomszédos mezőre léphetünk.

- Az eredményt a `v` tömbben kódoljuk, amelynek minden eleme egy struktúra, melynek `x` és `y` mezői vannak.
- Belső feltételek: az út folyosón kell haladjon végig és nem léphetünk kétszer ugyanarra a helyre. Vagyis $v[1] = (x, y), v[k]$ és $v[k-1]$ szomszédosak, `A[v[k].x][v[k].y] = 1`, $\forall k$, v utolsó eleme a mátrix szélén van és `v[i]` $\neq$ `v[j]`, $\forall i \neq j$

Labirintus

- A folytatási feltételek egyszerűsítése végett körbevesszük az `A` mátrixot 0-ás értékekkel.
- Egy `volt` logikai mátrixban tartjuk számon, hogy az aktuális útvonal során érintettük-e az adott mezőt.
- Egy pozícióra akkor léphetünk, ha ott folyosó található és még nem jártunk ott, ez lesz a folytatási feltétel.

Labirintus

```
ALGORITMUS Labirintus(x, y, k, v, volt, A)
  HA ((A[x][y] = 1) ÉS (volt[x][y] = HAMIS)) akkor
    volt[x][y] = IGAZ
    v[k] = (x, y)
    HA ((x = 1) VAGY (x = n) VAGY (y = 1) VAGY (y = m)) akkor
      Kiír(v, k)
    VÉGE(Ha)
    Labirintus(x + 1, y, k + 1, v, volt, A)
    Labirintus(x - 1, y, k + 1, v, volt, A)
    Labirintus(x, y + 1, k + 1, v, volt, A)
    Labirintus(x, y - 1, k + 1, v, volt, A)
    volt[x][y] = HAMIS
  VÉGE(Ha)
VÉGE(Algoritmus)
```

Labirintus

Második megoldás

- Bemutatunk egy másik megoldást, amely másképpen implementálja az algoritmus részleteit.
- A `volt` logikai tömböt felváltja egy `eredm` mátrix, melynek kezdetben minden eleme 0, majd egy adott koordinátán azt fogja tárolni, hogy hányadik lépésben értünk oda.
- Így a `v` tömbre már nincs szükségünk, az `eredm` tárolja az út visszakereséséhez szükséges információkat.
- Deklarálunk két konstans tömböt:
  `dx[4] = [-1 0 0 1], dy[4] = [0 -1 1 0]`
- Ezúttal nem használunk keretet, hanem ellenőrizzük, hogy kilépnénk-e a labirintusból a következő lépéssel.

Labirintus

Második megoldás

```
ALGORITMUS Labirintus2(x, y, k, eredm, A)
  MINDEN irány = 1, 4 végezd el:
    újx = x + dx[irány]
    újy = y + dy[irány]
```

Labirintus

Második megoldás

```
    HA ((újx >= 1) ÉS (újx <= n) ÉS (újy >= 1) ÉS (újx <= m)) akkor
      HA (A[újx][újy] = 1) ÉS (eredm[újx][újy] = 0) akkor
        eredm[újx][újy] = k
        HA ((újx = 1) VAGY (újx = n) VAGY
            (újy = 1) VAGY (újy = m)) akkor
          Kiír(eredm)
        VÉGE(Ha)
        Labirintus2(újx, újy, k + 1, eredm, A)
        eredm[újx][újy] = 0
      VÉGE(Ha)
    VÉGE(Ha)
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Fénykép

Feladat

Egy $n \times m$ méretű `A` mátrixban egy egyszerű fényképet ábrázolunk. Ott ahol a fényképen valamilyen tárgy egy része látható, a megfelelő elem értéke 1, különben 0. Ugyanahhoz a tárgyhoz tartozó megfelelő elemek szomszédosak soronként, oszloponként, vagy átlósan. Írjuk ki minden tárgy egy tetszőleges koordinátáját és a tárgy méretét (az 1-esek számát amik alkotják).

Fénykép

Megoldás

- A megoldáshoz kombináljuk az előbb látott ötleteket.
- Az `A` mátrix köré ismét nullásokból álló keretet rajzolunk, azon tárgyak lekezelésére, amelyek érintik a mátrix szélét.
- Ezúttal a `dx` és `dy` tömbök mérete 8 lesz.
- Ha elronthatjuk az `A` mátrix tartalmát, akkor 2-re állítjuk majd azokat a mezőket ahol már jártunk.
- Ha egy helyen jártunk, oda már nincs értelme más úton visszamenni, így az `A` elemeit többet nem állítjuk vissza 1-re.
- Emiatt ennek az algoritmusnak polinomiális futási ideje lesz, tulajdonképpen egy „álcázott” mélységi bejárásról van szó.
- Mivel ehhez hasonló algoritmusokat megtalálhatunk grafikus alkalmazásokban, zárt területek színezésekor, az algoritmust fill algoritmusnak is nevezik.

Fénykép

Pszeudokód

```
ALGORITMUS Fill(x, y, A, méret)
  HA (A[x][y] != 1) akkor
    VISSZATÉRÍT
  VÉGE(Ha)
  A[x][y] = 2
  méret = méret + 1
  MINDEN irány = 1, 8 végezd el:
    Fill(x + dx[irány], y + dy[irány], A, méret)
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Fénykép

Pszeudokód

Az algoritmust a következőféleképpen hívjuk meg:

```
...
MINDEN i = 1, n végezd el:
  MINDEN j = 1, m végezd el:
    HA (A[i][j] = 1) akkor
      méret = 0
      Fill(i, j, méret)
      KI: i, j, méret
    VÉGE(Ha)
  VÉGE(Minden)
VÉGE(Minden)
...
```

Tartalom

Mohó módszer (Greedy)

- Láttuk a `Sudoku`, `Kivonás` és `Dáma 1D` feladatoknál, hogy ha nem tetszőleges sorrendben jártuk be a backtracking által épített fát, hanem bizonyos döntések alapján választottuk meg, hogy merre induljunk, javíthattuk az esélyeinket arra, hogy hamarabb eredményhez jussunk.
- Tulajdonképpen lokálisan hoztunk döntéseket, annak az információnak az ismeretében, ami az adott csúcsban rendelkezésünkre állt.
- Ez a greedy módszer lényege, amely során mohó módon mindig a legígéretesebb irányba indulunk, viszont a backtrackinggel ellentétben, soha nem lépünk vissza.
- Megmaradva a fás analógiánál, tulajdonképpen a legbaloldalabbi ágon lefutunk a gyökértől a levélig.
- A greedy módszert leggyakrabban optimalizálási feladatok esetén használjuk.

Mohó módszer (Greedy)

- Bizonyos feladatok esetén optimális megoldáshoz juthatunk, ha megfelelő döntések alapján választjuk meg az első ágat amin lefelé haladunk a fában.
- A mohó módszer tárgyalását ilyen feladadok bemutatásával kezdjük majd.
- Ha egy feladatra greedy módszert akarunk alkalmazni, bizonyítanunk kell, hogy a módszer az optimális megoldáshoz vezet. Ezt általában a *reductio ad absurdum* módszerével kézenfekvő megtenni.
- Ennek az ellenkezője könnyen bizonyítható egy ellenpéldával.
- A bemutatott feladatok esetén kihagyjuk a bizonyításokat, de ezek megtalálhatóak a jegyzetben.

Greedy heurisztika

- Más feladatok esetében nem garantált, hogy a mohó módszerrel optimális megoldáshoz jutunk, ekkor a kapott megoldással csak közelíteni próbáljuk az optimumot.
- Ilyenkor greedy heurisztikáról beszélünk, erre is adunk majd néhány példát a fejezet végén.
- Ezt a jelenséget szemléltethetjuk a mindennapi életből vett példával is: ha az aktuális pozíciónkról indulva mindig a legmagasabb pontra mászunk amit látunk (vagyis a lokális ismereteink alapján mohó módon választunk), nem garantált, hogy a Mount Everest csúcsán kötünk ki (viszont vannak olyan kiindulási pontok, ahonnan igen).
- Ezt matematikai nyelvezettel úgy is szoktuk mondani, hogy a lokális optimum nem garantálja a globális optimumot. Ezekkel a fogalmakkal ismét találkozni fogunk a dinamikus programozás módszerénél.

Összeg

Feladat

Adott egy n elemű valós számokból álló sorozat. Határozzuk meg az adott sorozat azon részsorozatát, amelynek összege a lehető legnagyobb!

- Megoldás: Könnyen belátható, hogy optimális megoldást kapunk, ha a sorozatból kiválasztjuk az összes szigorúan pozitív elemet.
- Speciális eset: Ha minden szám negatív?

Várakozási idő

Feladat

Egy ügyvédi irodába egyszerre érkezik n személy, akiknek az intéznivalóit az ügyvéd ismeri, így azt is tudja, hogy egy-egy személlyel hány percet fog eltölteni. Állapítsuk meg azt a sorrendet, amelyben fogadnia kellene a személyeket ahhoz, hogy az átlagos várakozási idő minimális legyen!

Mivel az átlagos várakozási idő az n személy várakozási idejének számtani középarányosa, tulajdonképpen a várakozási idők összegét kell minimalizálni.

Várakozási idő

Példa: $n = 3$, várakozási idők: [60 10 30]

Sorrend	Várakozási idők összege	Átlagos várakozási idő
1 2 3	$0 + 60 + (60 + 10)$	$130 \ / \ 3$
1 3 2	$0 + 60 + (60 + 30)$	$150 \ / \ 3$
2 1 3	$0 + 10 + (10 + 60)$	$80 \ / \ 3$
2 3 1	$0 + 10 + (10 + 30)$	$50 \ / \ 3$
3 1 2	$0 + 30 + (30 + 60)$	$120 \ / \ 3$
3 2 1	$0 + 30 + (30 + 10)$	$70 \ / \ 3$

- A fenti példa alapján az első ötlet az lehetne, hogy visszalépéses kereséssel generáljuk az összes permutációt és ezek közül kiválasztjuk a minimális összeget adót. A megismert optimalizálási módszerek is alkalmazhatóak lennének.
- A feladat viszont megoldható sokkal hatékonyabban: a várakozási idők növekvő sorrendjében fogadjuk a személyeket.

Buszmegállók

Feladat

Egy közszállítási vállalat olyan gyorsjáratot szeretne indítani, amely csak a város főutcáján közlekedne és már létező megállókat használna. Ezeket a megállókat úgy kell kiválasztanunk, hogy két megálló között a távolság legkevesebb x méter legyen (mivel gyorsjáratról van szó) és a megállók száma legyen a lehető legnagyobb (minél több utas használhassa).

Példa: $n = 10, x = 60$
A megállók koordinátái: [0 100 150 175 200 250 260 270 350 370]
A gyorsjáratnak 5 megállója lesz, ezek sorszámai: 1, 2, 4, 6, 9.
Koordinátáik: 0, 100, 175, 250, 350

Buszmegállók

Megoldás

- Bizonyítható, hogy mindig létezik optimális megoldás, amelyhez hozzátartozik az 1-es, vagyis a legkisebb koordinátával rendelkező megálló (lásd jegyzet).
- Ennek a tulajdonságnak a segítségével jön a megoldási ötlet.
- Az első megállót betesszük a megoldásba.
- Minden lépésben a legközelebbi megállót választjuk a megoldásba, amely legalább x távolságra van a legutóbb hozzáadott megállótól.
- A pszeudokódban feltételezzük, hogy a megállók rendezve vannak a koordinátáik szerinti növekvő sorrendbe.

Buszmegállók

Pszeudokód

```
ALGORITMUS Busz(n, megállók, x, megoldás)
  Hozzáfűz(megoldás, 1)
  táv = 0
  MINDEN i = 2, n végezd el:
    újTáv = táv + megállók[i] -  megállók[i - 1]
    HA (újTáv >= x) akkor
      Hozzáfűz(megoldás, i)
      táv = 0
    KÜLÖNBEN
      táv = újTáv
    VÉGE(Ha)
  VÉGE(Minden)
VÉGE(Algoritmus)
```