Adatszerkezetek és algoritmusok

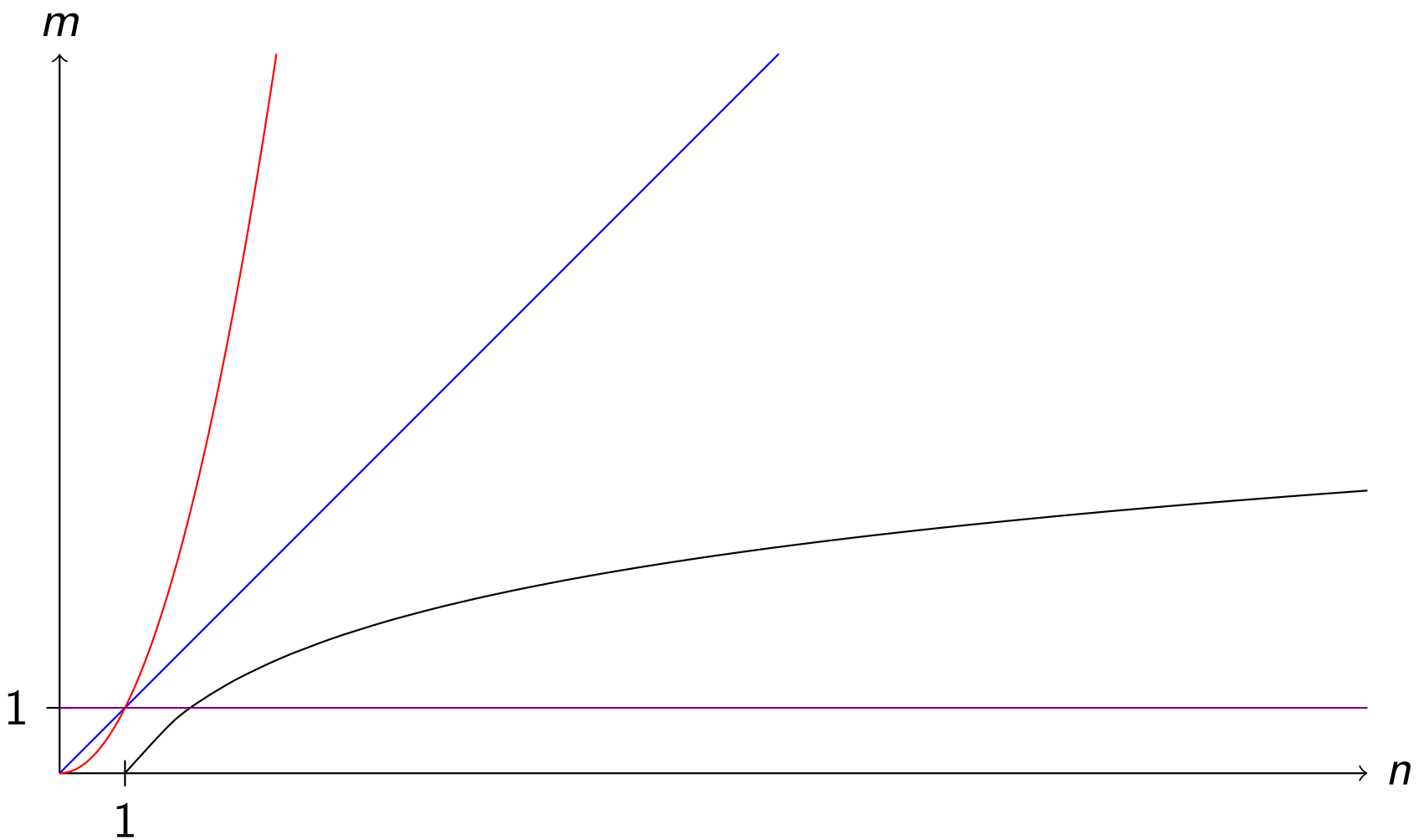
Horváth Géza

tizenkettedik előadás

Előadások témái

1. Az algoritmusokkal kapcsolatos alapfogalmak bevezetése egyszerű példákon keresztül.
2. Az algoritmusok futási idejének aszimptotikus korlátai.
3. Az adatszerkezetekkel kapcsolatos alapfogalmak. A halmaz, a multihalmaz és a tömb adatszerkezet bemutatása.
4. Az adatszerkezetek folytonos és szétszórt reprezentációja. A verem, a sor és a lista.
5. Táblázatok, önátrendező táblázatok, hash függvények és hash táblák, ütközéskezelés.
6. Fák, bináris fák, bináris keresőfák, bejárás, keresés, beszúrás, törlés.
7. Kiegyensúlyozott bináris keresőfák: AVL fák.
8. Piros-fekete fák.
9. B-fák.
10. Gráfok, bejárás, legrövidebb út megkeresése.
11. Párhuzamos algoritmusok.
12. Eldönthetőség és bonyolultság, a P és az NP problémaosztályok.
13. Lineáris idejű rendezés. Összefoglalás.

Polinomiális, lineáris, logaritmikus és konstans futási idő



Számítási kapacitás egységnyi idő alatt megtett lépések száma

We assume a 30 day month and 365 day year.

	1 Second	1 Minute	1 Hour	1 Day	1 Month	1 Year	1 Century
$\lg n$	$2^{1\times10^6}$	$2^{6\times10^7}$	$2^{3.6\times10^9}$	$2^{8.64\times10^{10}}$	$2^{2.592\times10^{12}}$	$2^{3.1536\times10^{13}}$	$2^{3.15576\times10^{15}}$
$\sqrt{n}$	1×10^{12}	3.6×10^{15}	1.29×10^{19}	7.46×10^{21}	6.72×10^{24}	9.95×10^{26}	9.96×10^{30}
n	1×10^6	6×10^7	3.6×10^9	8.64×10^{10}	2.59×10^{12}	3.15×10^{13}	3.16×10^{15}
$n\lg n$	62746	2801417	133378058	2755147513	71870856404	797633893349	6.86×10^{13}
n^2	1000	7745	60000	293938	1609968	5615692	56176151
n^3	100	391	1532	4420	13736	31593	146679
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

Eldönthető, eldönthetetlen, könnyű, nehéz...

Az eddig vizsgált algoritmusok csaknem valamennyien ***polinomiális idejűek,*** azaz n méretű bemeneten futási idejük a legrosszabb esetben is $O(n^k)$, valamely k konstanssal. Természetes a kérdés: vajon *minden* probléma megoldható-e polinomiális időben. A válasz természetesen: nem. Vannak problémák – mint például Turing híres megállási problémája –, melyek egyáltalán nem oldhatók meg számítógéppel, bármennyi idő áll is rendelkezésre. Vannak olyanok is, melyek megoldhatók ugyan, de nem polinomiális időben. A polinomiális idejű algoritmussal megoldható problémákat általában könnyűnek tekintjük, a szuperpolinomiális időt igénylőket pedig nehéznek.

Könnyű vagy nehéz?

E fejezet témája egy rendkívül érdekes problémaosztály – az úgynevezett NP-teljes problémák osztálya. NP-teljes problémára mind ez idáig senki sem adott polinomiális algoritmust, mint ahogy szuperpolinomiális alsó becslést sem. A P ≠ NP sejtés felvetése, azaz 1971 óta az elméleti számítógéptudomány egyik legmélyebb, legnehezebb kutatási területe.

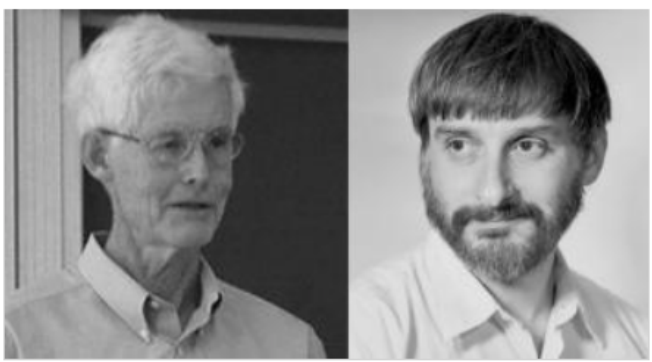
Millenniumi Problémák

claymath.org

CMI

ABOUT PROGRAMS MILLENNIUM PROBLEMS PEOPLE PUBLICATIONS EVENTS EUCLID

P vs NP Problem



Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe! Thus no future civilization could ever hope to build a supercomputer capable of solving the

Rules:

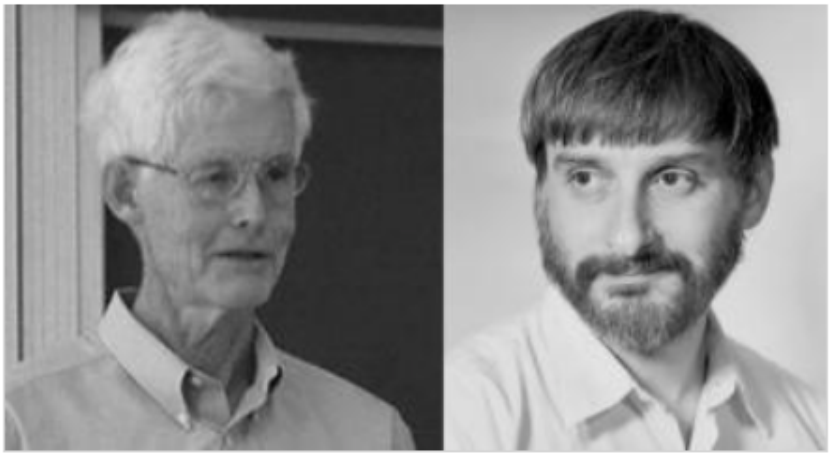
Rules for the Millennium Prizes

Related Documents:

Official Problem Description

Minesweeper

P vs. NP



Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe! Thus no future civilization could ever hope to build a supercomputer capable of solving the problem by brute force; that is, by checking every possible combination of 100 students. However, this apparent difficulty may only reflect the lack of ingenuity of your programmer. In fact, one of the outstanding problems in computer science is determining whether questions exist whose answer can be quickly checked, but which require an impossibly long time to solve by any direct procedure. Problems like the one listed above certainly seem to be of this kind, but so far no one has managed to prove that any of them really are so hard as they appear, i.e., that there really is no feasible way to generate an answer with the help of a computer. Stephen Cook and Leonid Levin formulated the P (i.e., easy to find) versus NP (i.e., easy to check) problem independently in 1971.

P

A P osztály a polinom időben megoldható problémákból áll, azaz olyan problémákból, melyekhez létezik olyan k konstans, hogy a probléma n hosszú bemenet esetén $O(n^k)$ idő alatt megoldható. Az eddigi fejezetekben vizsgált problémák túlnyomó többsége P-be tartozik.

Példa:

INSERTION-SORT(A)

```
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

NP

Az NP osztály olyan problémákból áll, amelyek polinom időben ellenőrizhetők. Ez valami olyasmit jelent, hogy ha valaki egy „bizonyítékot” adna nekünk a megoldásról, akkor annak helyességét polinomiális időben le tudnánk ellenőrizni.

Példa: lásd. 8. oldal.

Bármely P-beli probléma az NP osztályba is beletartozik, hiszen a P-beli problémákat meg tudjuk oldani polinomiális időben, még bizonyíték nélkül is. Ezt az elképzelést később pontosan leírjuk, addig kénytelenek vagyunk elhinni, hogy $P \subseteq NP$. Nyitott kérdés, hogy P valódi részhalmaza-e NP-nek.

NPC

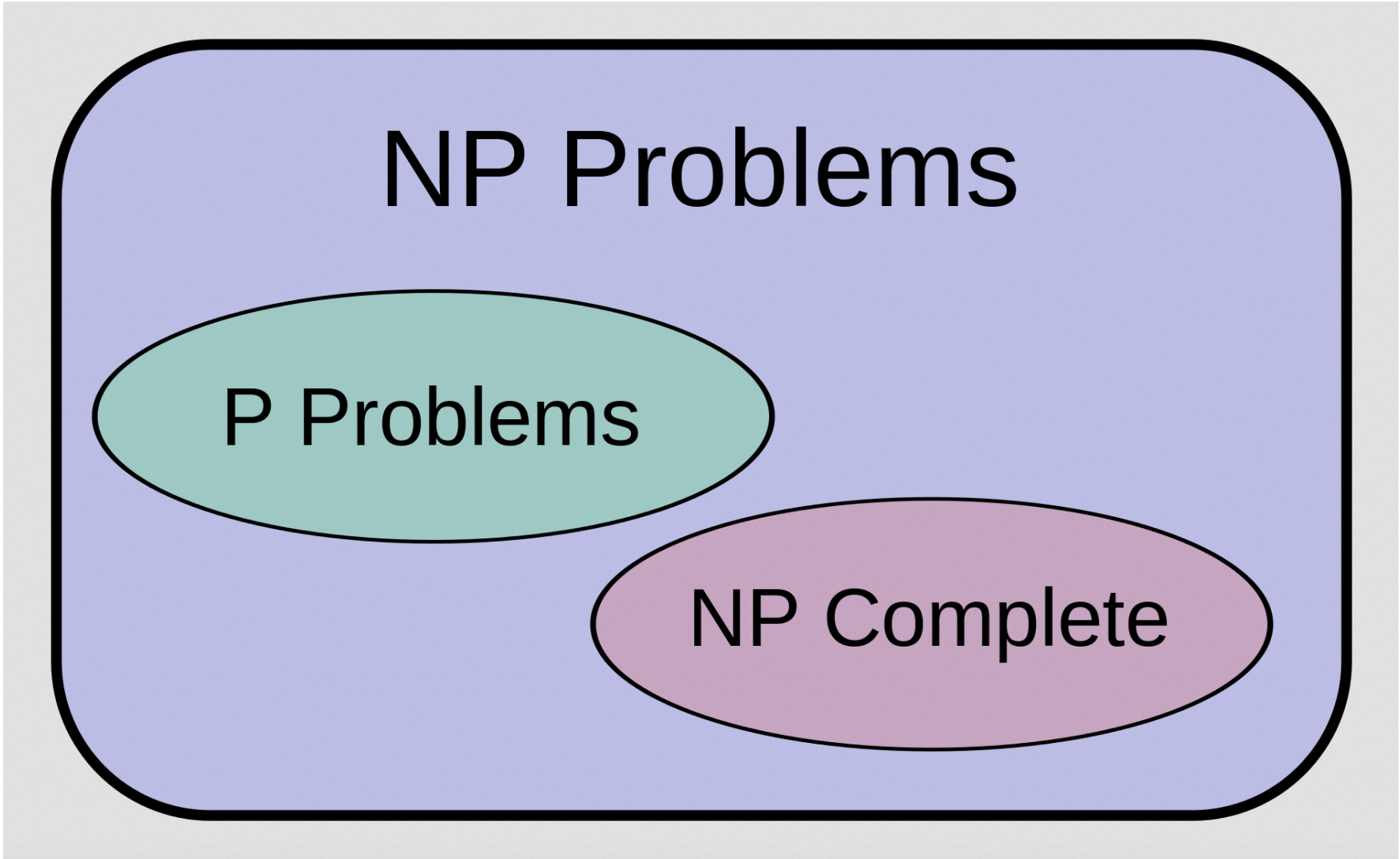
Az ***NP-teljes*** problémák osztályát NPC-vel jelöljük. Ide azok a problémák tartoznak, melyek „legalább olyan nehezek”, mint bármely NP-beli probléma. Hogy a „legalább olyan nehéz” mit is jelent, azt később pontosan definiálni fogjuk. Addig is kimondjuk bizonyítás nélkül, hogy ha az NP-teljes problémák közül *akár csak egy is* megoldható polinomiális időben, akkor *minden* NP-beli probléma megoldható polinomiális időben. A matematikusok nagy része úgy gondolja, hogy az NP-teljes problémák nem polinomiálisak. Napjainkig sokféle NP-teljes problémát vizsgáltak, a polinomiális idejű algoritmus irányába történő legcsekélyebb előrehaladás nélkül, így igencsak meglepő lenne, ha kiderülne, hogy ezen problémák mindegyikére létezik polinomiális megoldás. Tekintetbe véve mindazonáltal azt is, hogy a $P \neq NP$ állítás bizonyításának szentelt jelentős erőfeszítések sem hoztak eredményt, nem zárható ki annak lehetősége, hogy az NP-teljes problémák mégiscsak megoldhatók polinom időben.

NPC

Aki jó algoritmusokat szeretne tervezni, annak feltétlenül értenie kell az NP-teljesség elméletének alapjait. Ha például valaki mérnöki munkája során NP-teljes problémával találkozik, többnyire az a legjobb, ha alkalmas közelítő algoritmust (lásd 35. fejezet) próbál találni, és nem vesződik a gyors, pontos megoldás kitalálásával. Ezen túlmenően igen sok érdekes és természetesen felmerülő probléma, amely első ránézésre nem tűnik nehezebbnek, mint a keresés, a rendezés vagy a hálózati folyamok, NP-teljesnek bizonyul. Ezért fontos, hogy közelebbről megismerkedjünk ezzel a figyelemre méltó problémaosztállyal.

Példa: Az utazóügynök probléma (travelling salesman problem, TSP) a következő kérdésre keresi a választ. Adva vannak városok és a köztük lévő távolságok páronként. Egy kezdő városból kiindulva melyik a legrövidebb út, mely bejárja az összes várost, majd visszatér a kiindulási pontba?

A legtöbb kutató szerint ez a feltépítés a legvalószínűbb



Döntési és optimalizálási problémák

Az érdeklődésünk középpontjában álló problémák jelentős része ***optimalizálási probléma,*** ahol minden megengedett megoldáshoz egy érték tartozik, feladatunk pedig olyan megengedett megoldás megtalálása, amelyhez a legjobb érték van rendelve. Például a LEGRÖVIDEBB-ÚT-nak nevezett probléma esetében adott egy G irányítatlan gráf és az u, v csúcsok, feladatunk pedig olyan u és v közötti út megtalálása, melynek a lehető legkevesebb éle van. (Más szavakkal LEGRÖVIDEBB-ÚT a két adott pont közti legrövidebb út probléma egy súlyozatlan, irányítatlan gráfban.) Az NP-teljesség fogalmát azonban nem optimalizálási problémákra alkalmazzuk közvetlenül, hanem úgynevezett ***döntési problémákra,*** ahol a válasz egyszerűen „igen” vagy „nem” (formálisabban „1” vagy „0”).

Példa: Prímteszt: egy adott természetes szám esetén megadja, hogy az adott szám prím szám-e.

Döntési és optimalizálási problémák

Bár problémák NP-teljességének bizonyításakor tevékenységünket a döntési problémák birodalmára kell korlátoznunk, valójában nyilatkozhatunk optimalizálási problémákról is. Létezik ugyanis egy rendkívül hasznos összefüggés az optimalizálási és a döntési problémák között. Egy adott optimalizálási problémának megfeleltethetünk egy döntési problémát oly módon, hogy az optimalizálandó értékre egy korlátot állapítunk meg. A LEGRÖVIDEBB-ÚT esetében például a megfelelő döntési probléma, melyet ÚT-nak fogunk nevezni, a következő: adott egy G irányítatlan gráf és az u, v csúcsok, továbbá egy k egész szám, döntsük el, hogy létezik-e u és v között olyan út, amely legfeljebb k élből áll.

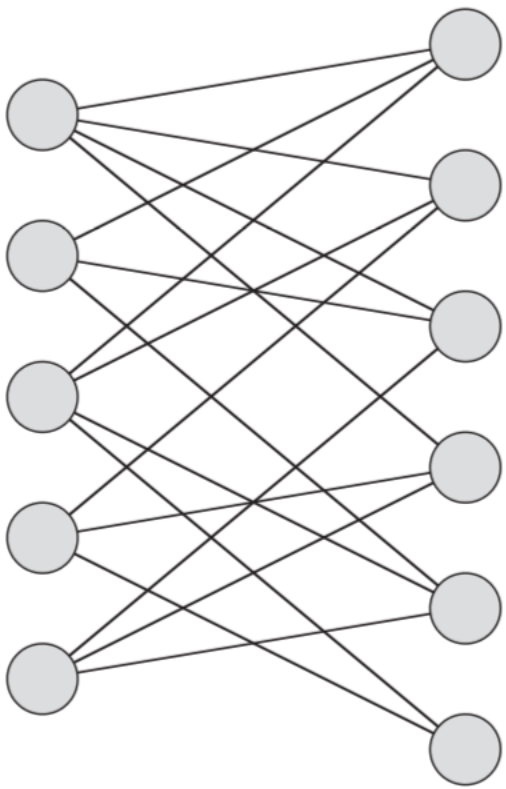
Legrövidebb út vs. leghosszabb út

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2       u.color = WHITE
 3       u.d = ∞
 4       u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       u = DEQUEUE(Q)
12       for each v ∈ G.Adj[u]
13            if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.π = u
17                 ENQUEUE(Q, v)
18       u.color = BLACK
```

- A szélességi keresés megoldja a legrövidebb út problémát $\mathcal{O}(|V| + |E|)$ (lineáris) lépésben egy tetszőleges irányítatlan súlyozatlan G=(V,E) gráf esetén.
- A leghosszabb út megkeresése két csúcs között viszont NP-teljes probléma.

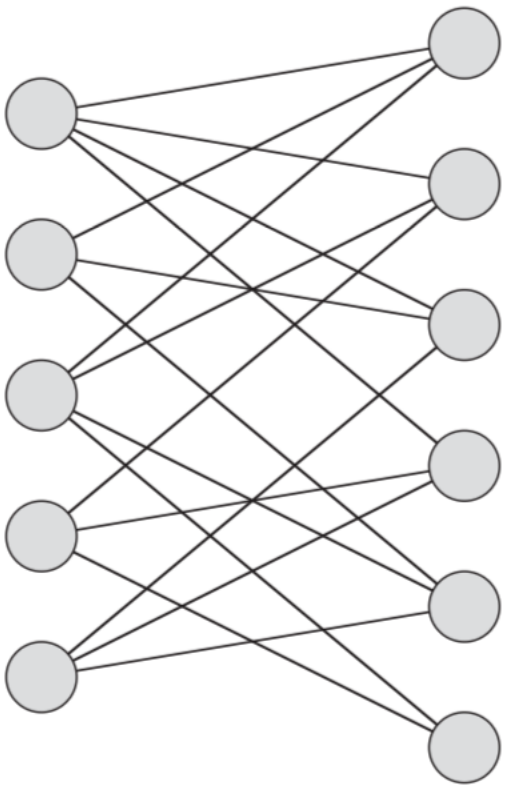
Euler-kör vs. Hamilton-kör

A $G = (V, E)$ összefüggő, irányított gráf ***Euler-köre*** egy olyan kör, amelyben pontosan egyszer szerepel G minden éle. (Egy csúcsot többször is érinthetünk.)



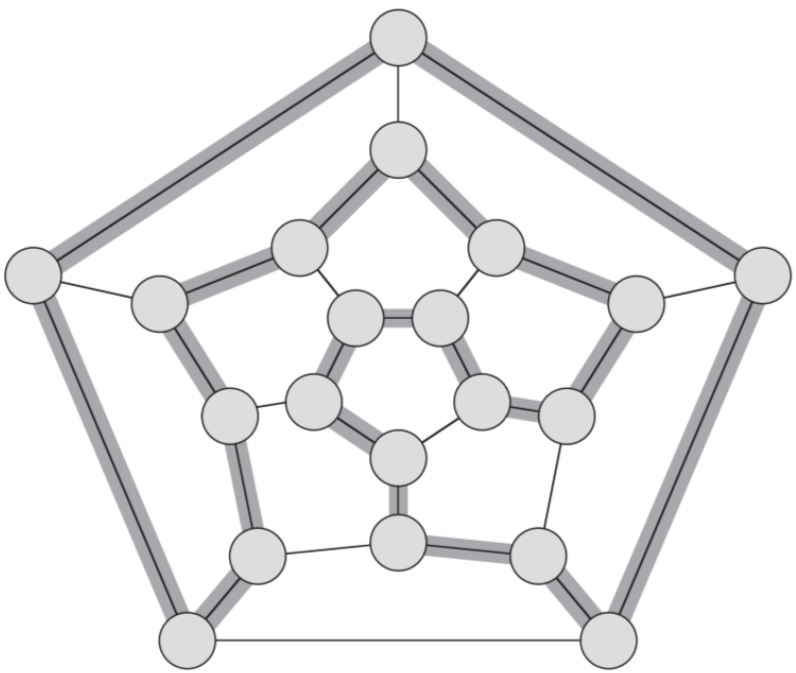
Euler-kör vs. Hamilton-kör

- A G gráf pontosan akkor tartalmaz Euler-kört, ha minden csúcsából páros számú él indul ki.
- Létezik olyan algoritmus, amelyik $\mathcal{O}(|E|)$ lépésben eldönti, hogy egy adott G gráfban található-e Euler-kör.



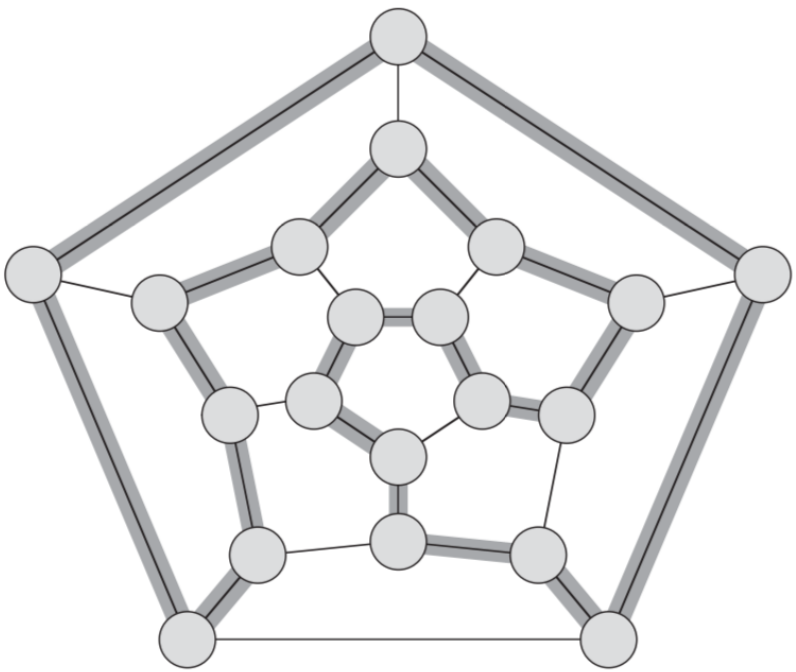
Euler-kör vs. Hamilton-kör

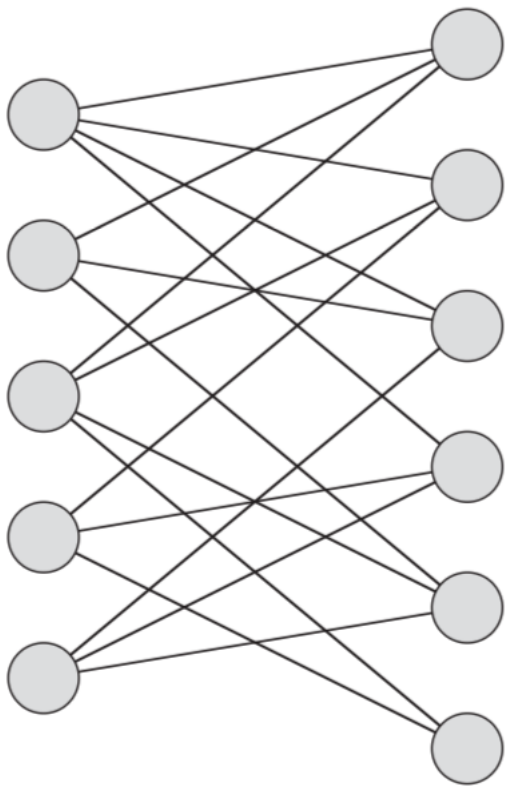
A G=(V,E) gráf Hamilton-köre egy olyan kör, amelyben pontosan egyszer szerepel minden csúcs.



Euler-kör vs. Hamilton-kör

A G=(V,E) gráf Hamilton-köre egy olyan kör, amelyben pontosan egyszer szerepel minden csúcs.





Euler-kör vs. Hamilton-kör

A G=(V,E) gráf Hamilton-köre egy olyan kör, amelyben pontosan egyszer szerepel minden csúcs.

Annak megválaszolása, hogy egy adott gráfban található-e Hamilton-kör, NP-teljes probléma.

Ellenőrző algoritmus Hamilton-körhöz

Annak ellenőrzéséhez, hogy egy adott gráfban található Hamilton-kör, szükségünk van egy "bizonyítékra", – avagy "tanúra", illetve "súgásra", a magyar terminológia nem egységes ebben az elnevezésben, – valamint szükségünk van még egy polinom idejű algoritmusra, ami igazolja a bizonyíték (tanú súgás) segítségével, hogy az adott gráfban létezik Hamilton-kör.

Egy jó tanú lehet a Hamilton-kört alkotó csúcsok sorozata, és van olyan $\mathcal{O}(n^2)$ idejű algoritmus, amely egy adott gráf és az előbb említett csúcssorozat ismeretében polinom idő alatt edönti, hogy az adott gráfban a megadott csúcssorozat Hamilton-kört alkot-e.

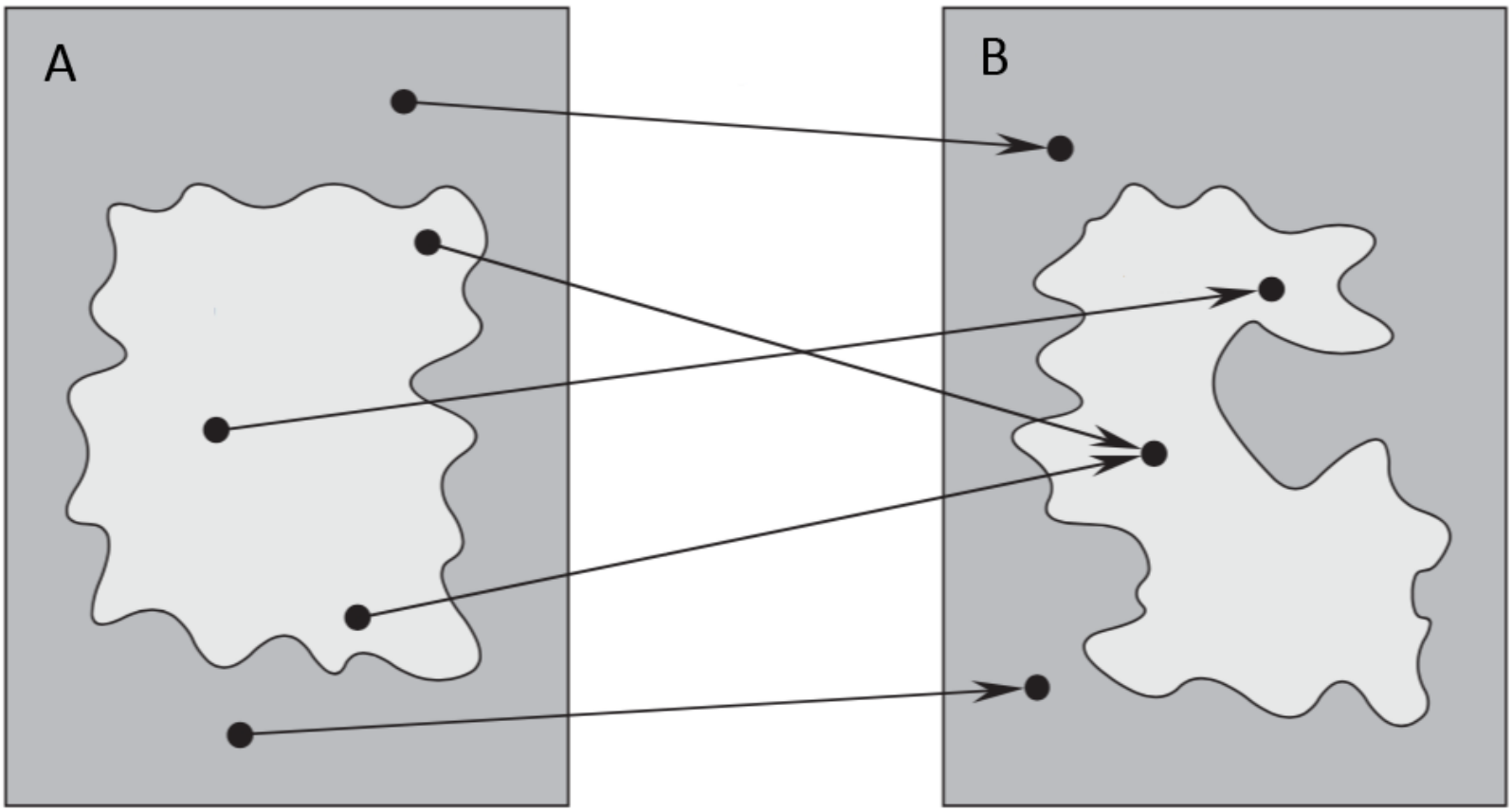
Ezzel igazolást nyert, hogy a Hamilton-körök problémája NP-beli probléma.

Polinom idejű visszavezető algoritmus, avagy a Karp-redukció

- Legyen adott egy A döntési probléma, amit polinom idő alatt szeretnénk megoldani.
- Tegyük fel, hogy van egy B döntési probléma, amit már meg tudunk oldani polinom idő alatt.
- Végezetül legyen egy olyan eljárásunk, ami az A probléma minden α esetét átalakítja a B probléma egy β esetévé, úgy, hogy
  1. az átalakítás polinom idő alatt megtörténik,
  2. a válasz minden esetben azonos lesz, azaz β pontosan akkor igaz, amikor α is igaz, és β pontosan akkor hamis, amikor α is hamis.

A fentebbi eljárást **polinom idejű visszavezető algoritmusnak, vagy egyszerűen csak a Karp-redukciónak** hívjuk.

Karp-redukció



Karp-redukció

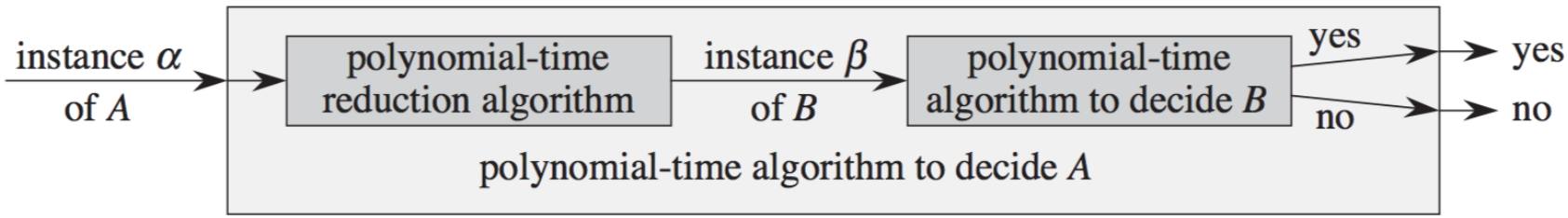


Figure 34.1 How to use a polynomial-time reduction algorithm to solve a decision problem A in polynomial time, given a polynomial-time decision algorithm for another problem B. In polynomial time, we transform an instance α of A into an instance β of B, we solve B in polynomial time, and we use the answer for β as the answer for α.

34.1. ábra. Polinomiális idejű visszavezető algoritmus használata az A döntési probléma megoldására, a B problémára ismert polinomiális algoritmus ismeretében. A egy α esetét átalakítjuk B egy β esetévé, megoldjuk B-t a β esetre, végül a β-ra kapott választ adjuk meg az A α esetéhez tartozó válaszként.

NP-teljesség

A B probléma **NP-nehéz** ha minden NP-beli A probléma esetén létezik az A problémának Karp-redukciója a B problémára.

Egy probléma **NP-teljes,** ha

1. NP-beli, és
2. NP-nehéz.

- Annak igazolására, hogy NP-beli, a polinom idejű ellenőrző algoritmust és a tanúkat használjuk.
- Annak igazolására, hogy NP-nehéz, a Karp redukciót és egy előző, igazoltan NP-teljes problémát használjuk.

Irodalomjegyzék