Kereső algoritmusok a diszkrét optimalizálás problémájához

A. Grama, A. Gupta, G. Karypis és V. Kumar: „Introduction to Parallel Computing”, Addison Wesley, 2003. könyv anyaga alapján

A kereső eljárások pontosabb tárgyalása Futó Iván (szerk.): Mesterséges Intelligencia, Aula, 1999. fejezetéből származnak

Vázlat

- A diszkrét optimalizálási probléma
- Soros megoldás
- Párhuzamos megoldási lehetőségek és problémák

Diszkrét optimalizálás

- A diszkrét optimalizálási problémák komoly számítási igényűek. Jelentős elmélettel és gyakorlati fontossággal rendelkeznek (ütemezési feladatok, pályatervezés, digitális áramkörökhöz tesztpéldák generálása, logisztikai feladatok, stb.).
- A kereső algoritmusok a feltételeknek megfelelő megoldások terét szisztematikusan vizsgálják.

Definíció

- A *diszkrét optimalizációs probléma (DOP)* a következő módon fejezhető ki: (S, f).
  - S bizonyos feltételeknek eleget tévő megoldások véges, vagy megszámlálhatóan végtelen halmaza,
  - f költségfüggvény S minden eleméhez valós R értéket rendel.
- A DOP megoldása olyan x_{opt} megoldás keresése, hogy $f(x_{opt}) \leq f(x)$ minden $x \in S$ esetén.

- Gyakran NP-teljes probléma.

Diszkrét optimalizáció: példa

- *Egyértékű lineáris programozási feladat* (0/1 integer-linear-programming problem): adott A $m \times n$ mátrix, b $m \times 1$ vektor és c $n \times 1$ vektor.
- A cél egy olyan $\overline{x}$ $n \times 1$ vektor meghatározása, amelynek elemei csak 0 és 1 értékekből állhatnak,
  a vektor teljesíti a következő feltételt: $A\overline{x} \geq b$

  és az f függvénynek minimuma kell legyen: $f(\overline{x}) = c^T \overline{x}$

*: Duális megfogalmazás

DOP és a gráfkeresés

- Az S lehetséges elemeinek száma nagyon nagy.
- A DOP gyakran úgy is megfogalmazható, hogy egy gráfban minimum költségű utat keressünk egy kezdőponttól egy vagy több lehetséges cél csomópontig.
- Az S minden x elemét tekinthetjük egy útnak a kezdőponttól valamely célig.
  - A gráf csomópontjai állapotok
  - Az állapotok vagy végpontok, vagy nem végpontok
  - Bizonyos állapotok lehetséges megoldáshoz tartoznak, mások nem
  - Az élekhez költségek tartoznak, amelyek az egyik állapotból a másikba történő átmenet ráfordításai.
- A gráfot *állapottérnek* nevezzük.

Állapottér gráf: példa

- Egyértékű lineáris programozási feladat
  - Az állapotok az *x* vektor egyes koordinátáihoz rendelt értékek => fa

Example 11.3 The 0/1 integer-linear-programming problem revisited

Consider an instance of the 0/1 integer-linear-programming problem defined in Example 11.1. Let the values of A, b, and c be given by

$$A = \begin{bmatrix} 5 & 2 & 1 & 2 \\ 1 & -1 & -1 & 2 \\ 3 & 1 & 1 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} 8 \\ 2 \\ 5 \end{bmatrix}, \quad c = \begin{bmatrix} 2 \\ 1 \\ -1 \\ -2 \end{bmatrix}.$$

The constraints corresponding to A, b, and c are as follows:

$$\begin{aligned} 5x_1 + 2x_2 + x_3 + 2x_4 &\geq 8 \\ x_1 - x_2 - x_3 + 2x_4 &\geq 2 \\ 3x_1 + x_2 + x_3 + 3x_4 &\geq 5 \end{aligned}$$

and the function $f(x)$ to be minimized is

$$f(x) = 2x_1 + x_2 - x_3 - 2x_4.$$

$$\sum_{x_j \text{ is free}} \max\{A[i, j], 0\} + \sum_{x_j \text{ is fixed}} A[i, j]x_j \geq b_i, \quad i = 1, \ldots, m$$

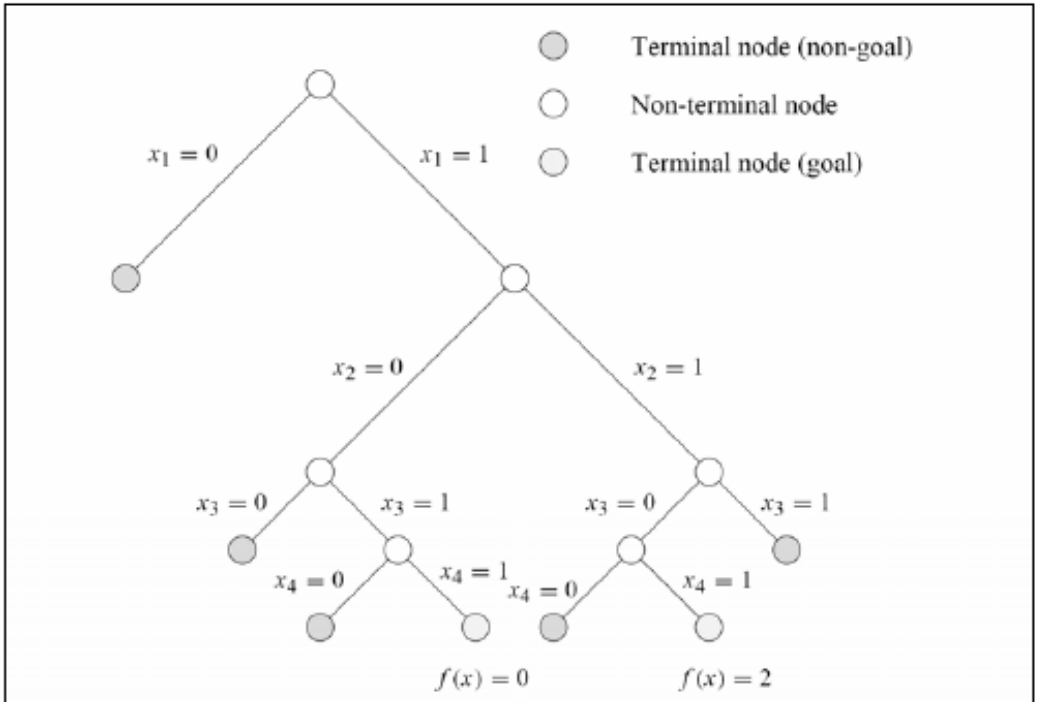
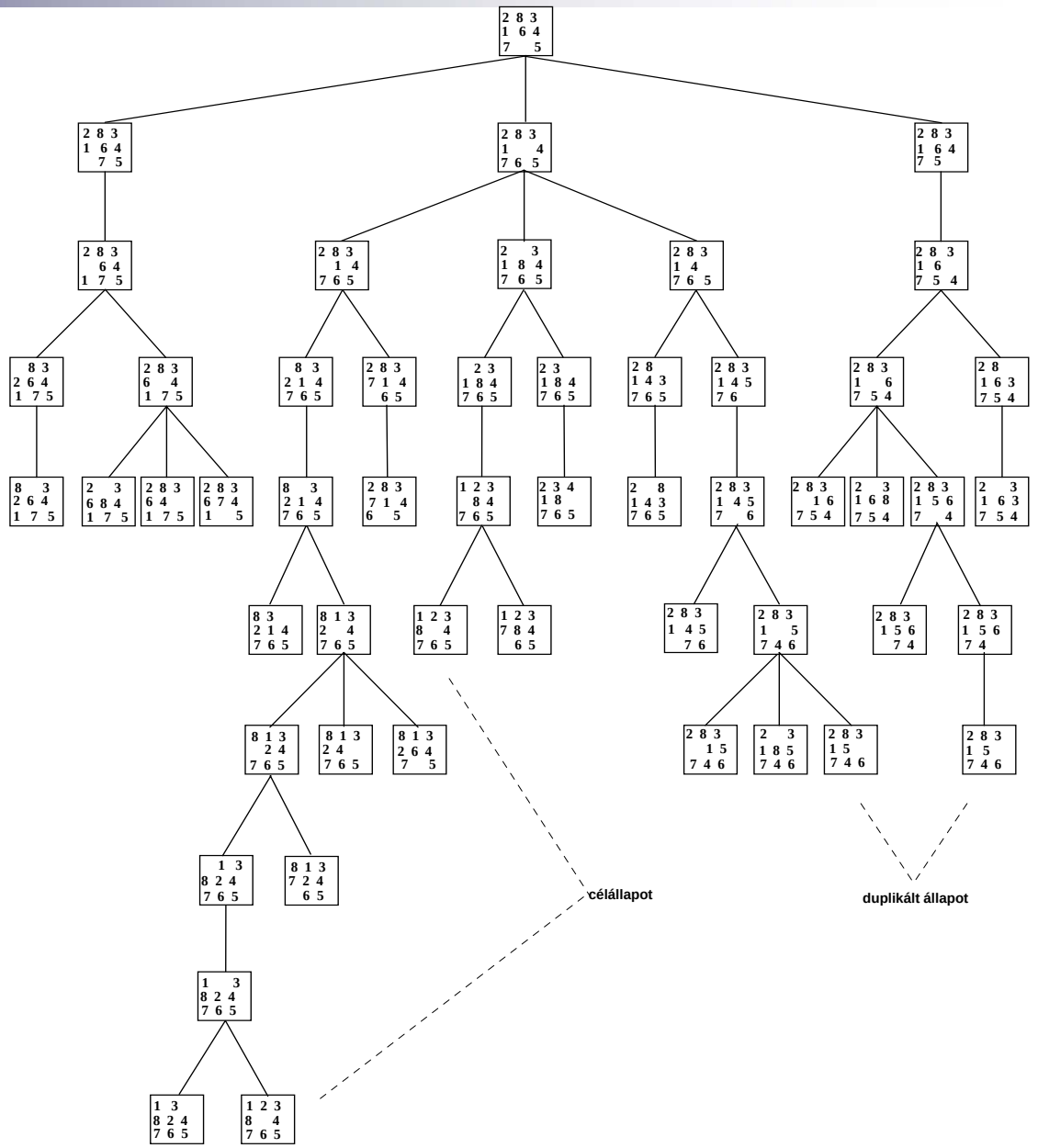


Figure 11.2 The graph corresponding to the 0/1 integer-linear-programming problem.

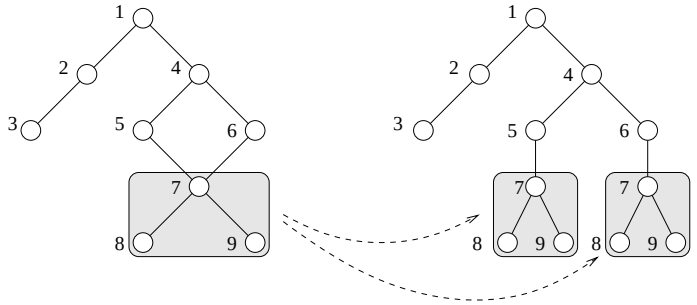
8-as kirakó

- Egy kezdeti állapotból találjuk meg azt a legrövidebb lépéssorozatot, amely a célkonfigurációba vezet.
- *Állapottér:* gráf

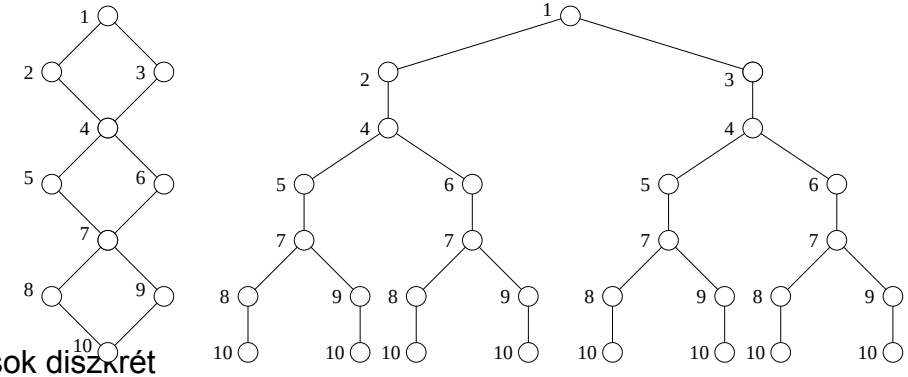


Kereső algoritmusok tere

- A keresési tér gráf vagy fa?
- A gráf fává történő kiterítése gyakran nagyméretű feladathoz vezet.



(a)



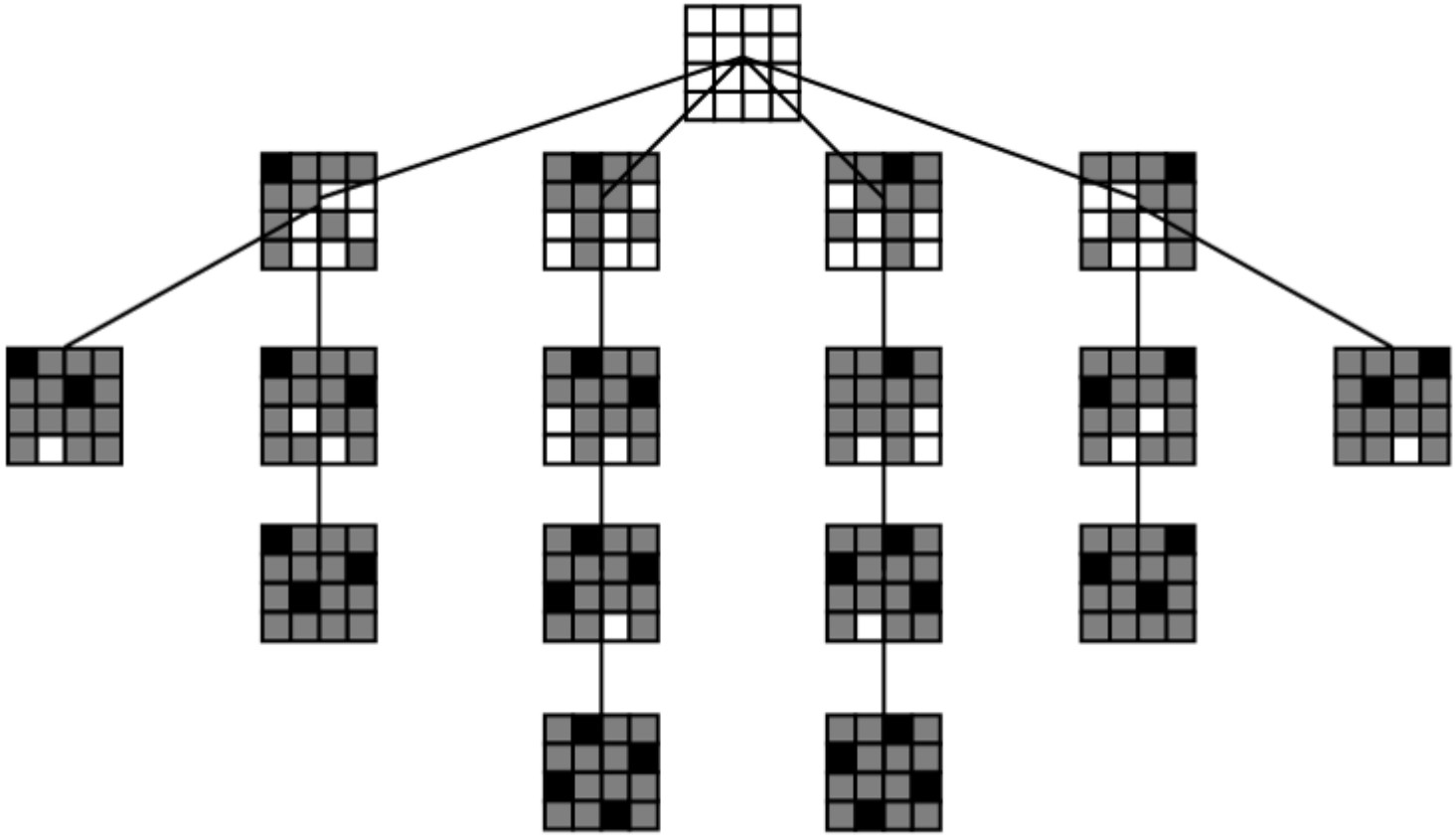
(b)

Kereső stratégiák I.

- Nem módosítható keresés
  - Nincs visszalépés. Pl. hegymászó algoritmus (gradiens keresés).
- Visszalépéses keresés
  - Törli a zsákutcát.
  - Ha körre futunk, akkor visszalépés.
  - Mélységi korlátot használhatunk (heurisztika).
  - Azonos szinten célszerű sorrendi heurisztikát használni.
  - Kis memória igényű.
  - Nem garantálja az optimális megoldást (csak növekvő mélységi korláttal és iterálva).

Négy-királynő probléma

- Visszalépéses keresés + azonos sorban haladva, mindig balról jobbra helyezzük el a királynőket.



Kereső stratégiák II. *Nem informált gráfkeresés*

- *Nem informált gráfkeresések* – a gráf kiértékelő függvényben *(f(n) = g(n) + h(n)) beépülő* heurisztikát nem használunk.
- Csúcsok NYÍLT és ZÁRT listája.
- Mélységi keresés
  - Egységnyi élköltséget tekintünk.
  - *f(n):= -g(n)* minden *n NYÍLT* csúcsra.
  - Mélységi korlát használható, iterációs változatban is.
  - Fa állapottérnél előnyös.
- Szélességi keresés
  - Egységnyi élköltséget tekintünk.
  - *f(n):= g(n)* minden *n NYÍLT* csúcsra.

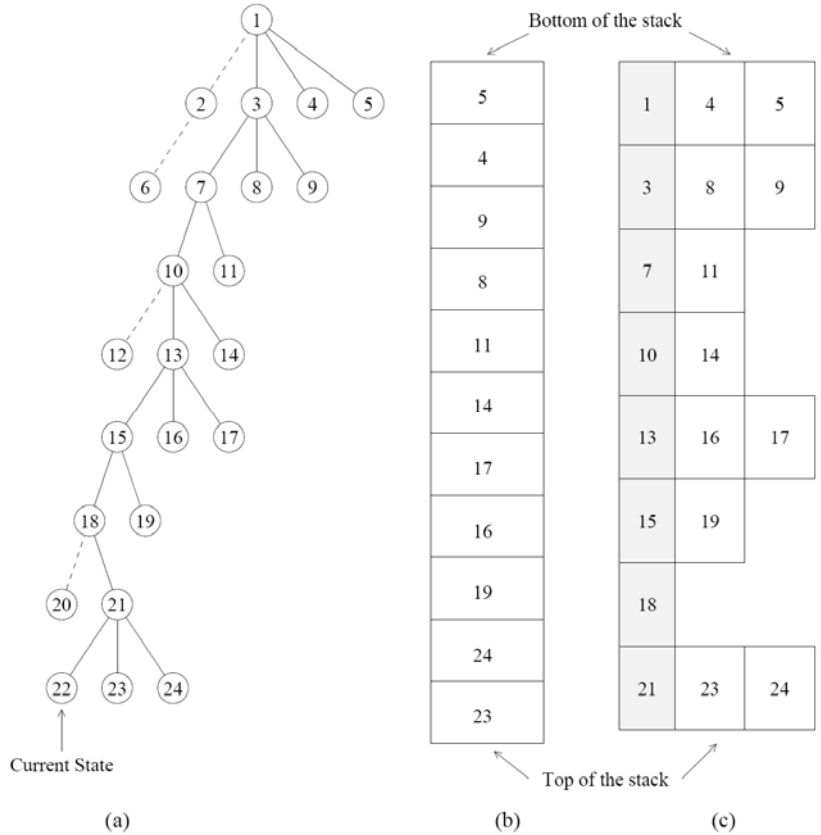
Iteratív mélységi keresés

- Gyakran a megoldás a gyökérhez közeli, de másik ágon van.
- Az egyszerű mélységi keresés nagy részt dolgozna fel mielőtt erre az ágra jut.
- Iteratív módon növeljük a mélységi határt, ameddig keresünk.
- Ha nem találunk megoldást, akkor a határt növeljük és ismételjük a folyamatot.

Mélységi keresés: tárolási követelmény és adatstruktúra

- A mélységi keresés minden lépésénél a még ki nem próbált alternatívákat el kell tárolni.
- Ha m egy állapot tárolásához szükséges adatterület, és d a maximális mélység, akkor a mélységi kereséshez szükséges teljes tér $O(md)$ nagyságrendű.
- Ha a keresendő állapottér fa, akkor a mélységi keresést veremmel hatékonyan lehet reprezentálni.
- A verem memória szükséglete egyenesen arányos a fa mélységével.

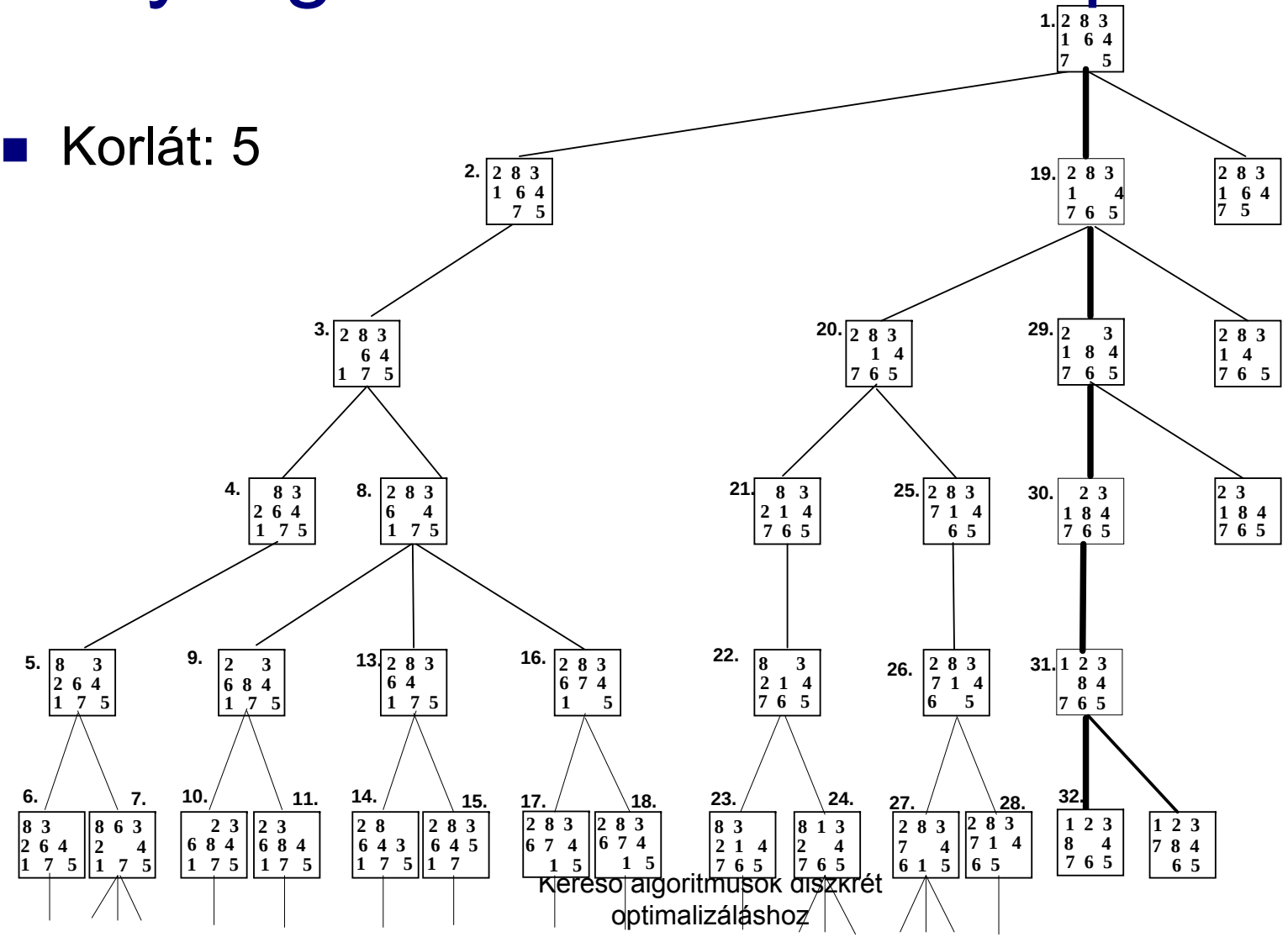
Mélységi keresés: tárolási követelmény é



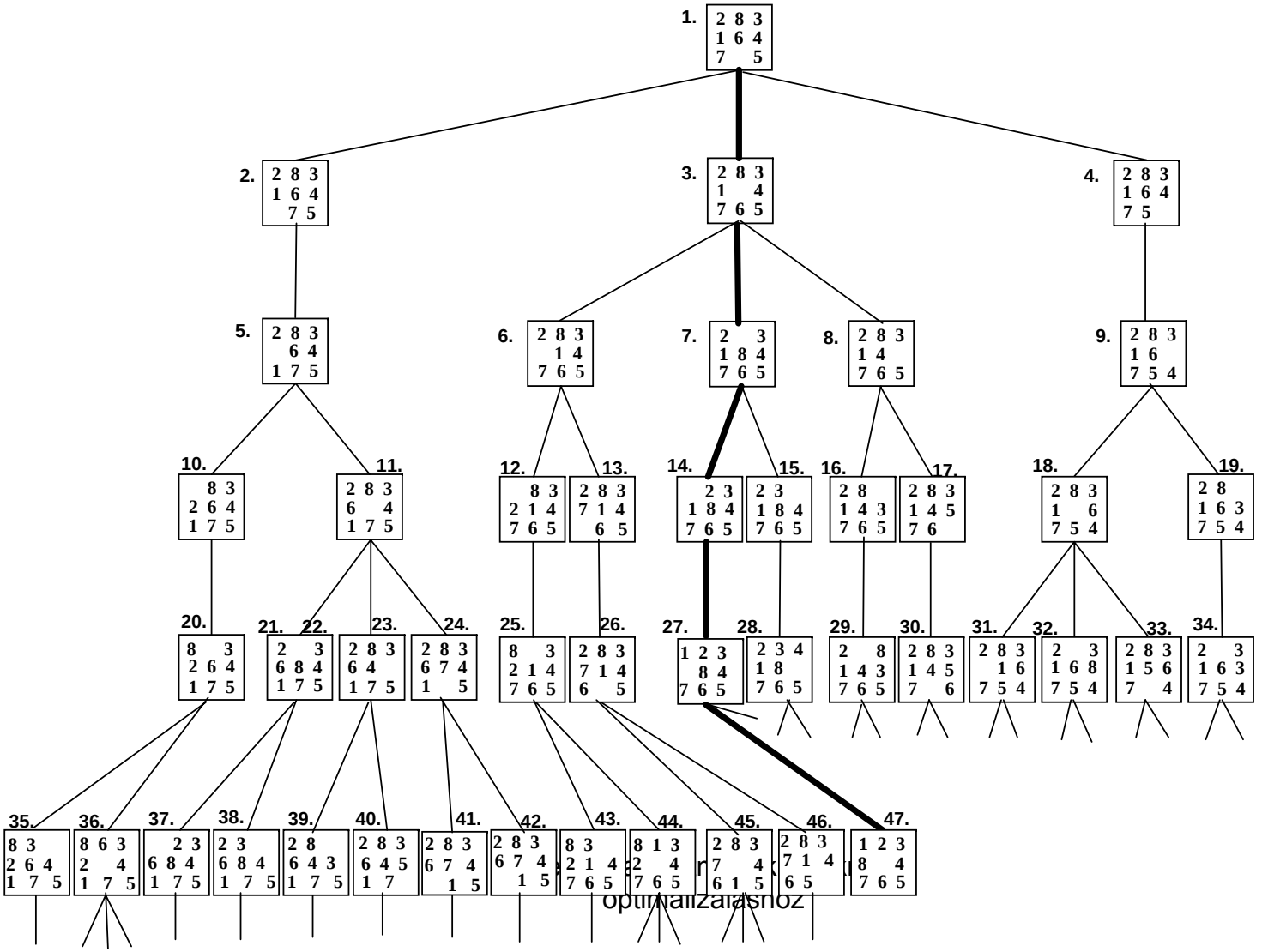
(a) Mélységi keresés fa esetén; pontozottan jelennek meg a már felkeresett csúcsok; (b) A verem a még ki nem próbált alternatívákat tárolja csak; (c) a verem a ki nem próbált alternatívákat és szüleiket (szürkített) tárolja.

Mélységi keresés korláttal: példa

- Korlát: 5



Szélességi keresés: példa



Heurisztikus gráfkereső stratégiák

- Az *f(x)* kiértékelő függvénybe valamilyen heurisztikát építünk be (*f(x)* = *g(x)* + *h(x)*). Gyakran például becsülhető a cél elérésének költsége az aktuális csúcsból.
- A becslést *heurisztikus becslésnek* nevezik.
- Ha a becslés garantáltan alábecsül, akkor *megengedő heurisztikáról* beszélünk.
- Elképzelés: ha heurisztikát használunk, akkor a „rossz”, vagy „nem sokat ígérő” utakra nem költünk.

Diszkrét optimizálás példához heurisztika

A 8-as kirakóhoz (megengedő) heurisztika

- *W(n)* (*n NYÍLT*), ahol *W* a rossz helyek száma

A 8-as kirakóhoz megengedő heurisztika

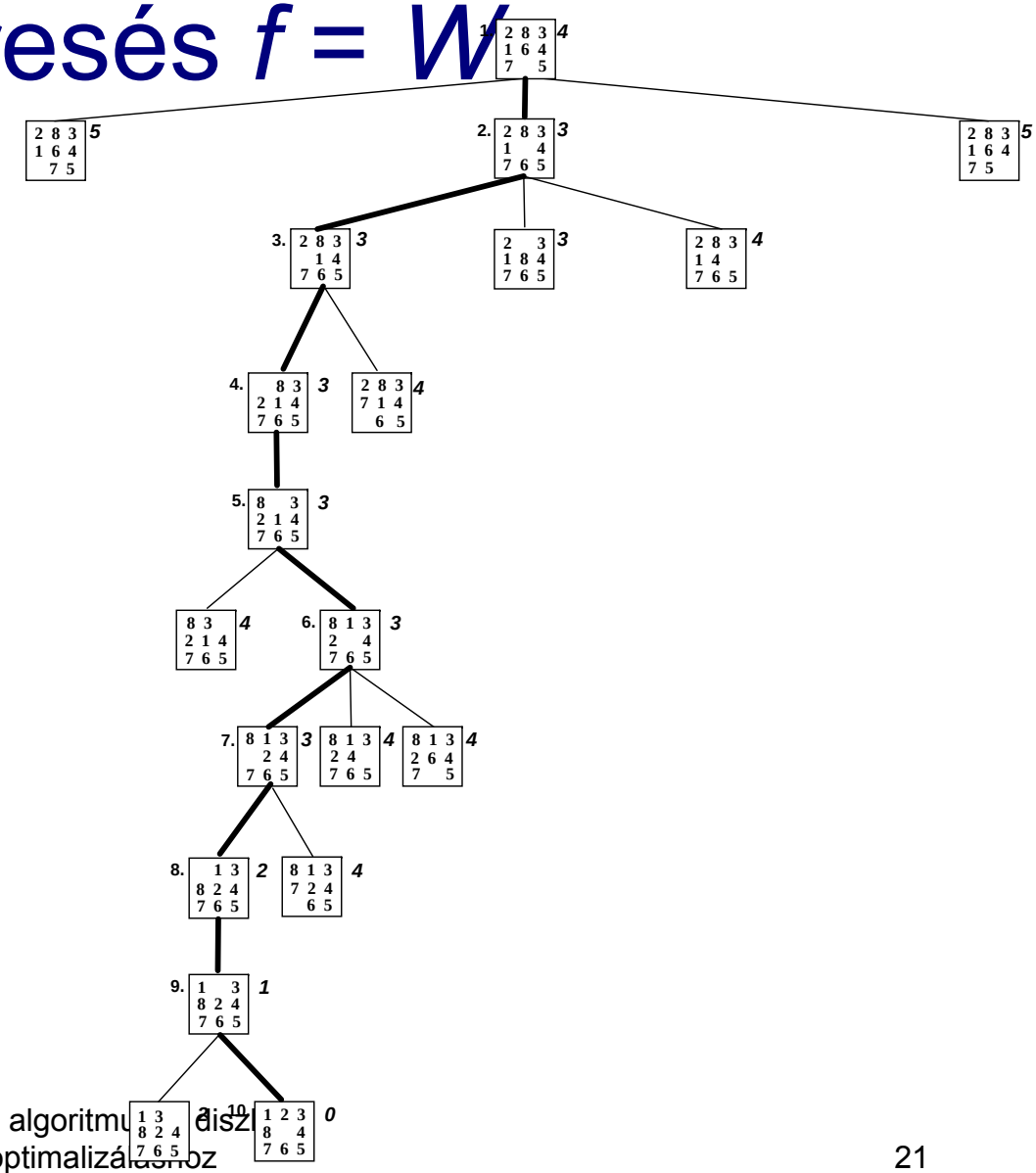
- A rács minden pontja a koordinátáival azonosítható.
- Az (i, j) és a (k, l) pozíciók távolsága: $|i - k| + |j - l|$. Ez a Manhattan távolság.
- A kezdeti és cél pozíciók közötti Manhattan távolságok összege (*P*-vel fogjuk jelölni) megengedő heurisztika.

Kereső stratégiák III. Heurisztikus gráfkereső stratégiák

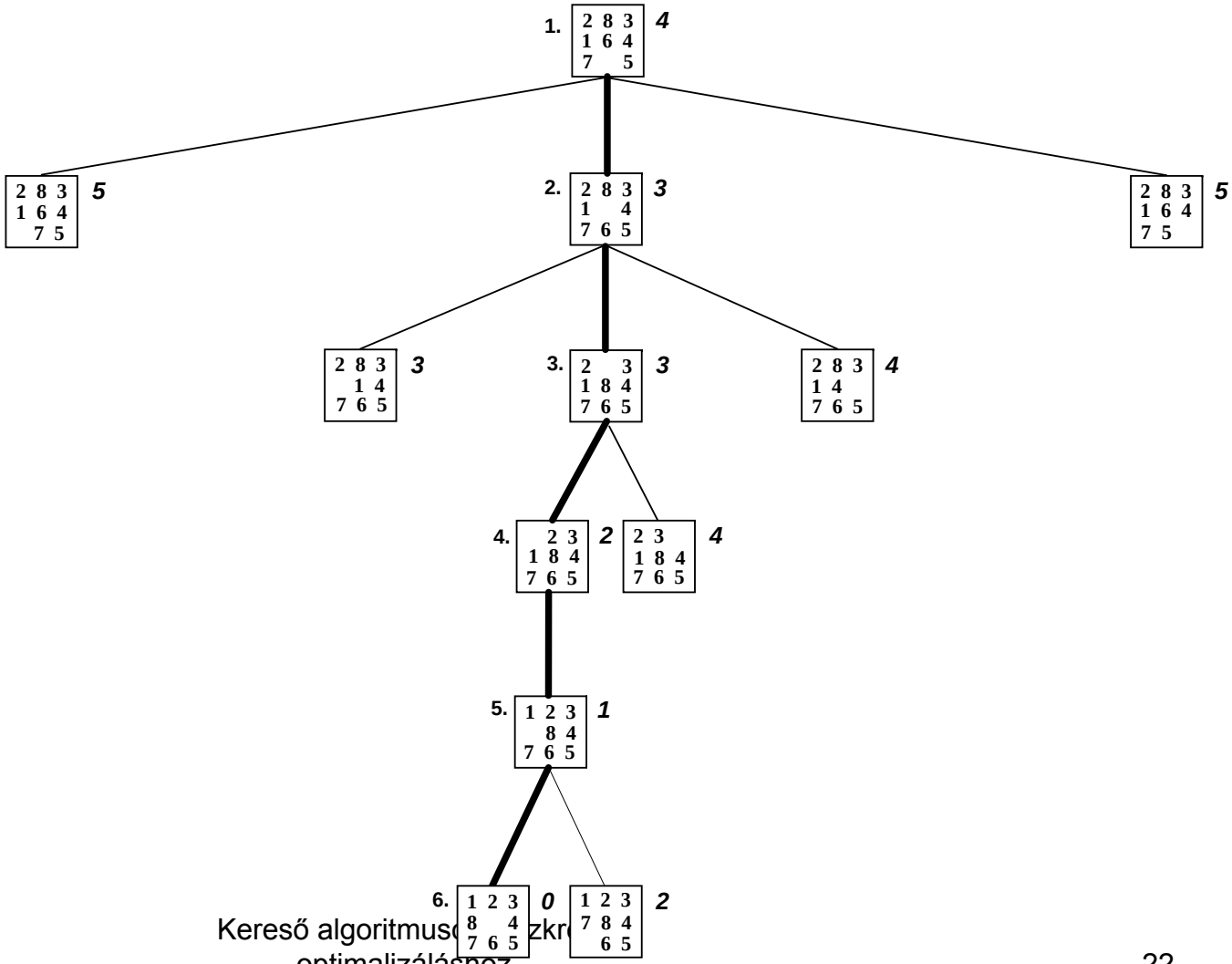
- Előretekintő keresés
  - *f(n)* = *h(n)* minden *n NYÍLT* csúcsra.
- A*
  - Kiértékelő fgv. *f(n)* = *g(n)* + *h(n)* minden *n NYÍLT* csúcsra, ahol *h(n)* megengedő heurisztika.
  - Garantálja az optimális megoldást.
  - Nagy a memória igény.
    - Meglátogatott csúcsokkal (állapotokkal) arányos.
  - Mind fák, mind gráfok estében hatékony.

Előretekintő keresés *f* = *W*

- *W* hibásak száma, hátralévő mozgások alsó becslését adja.



A* példa *P* heurisztikával



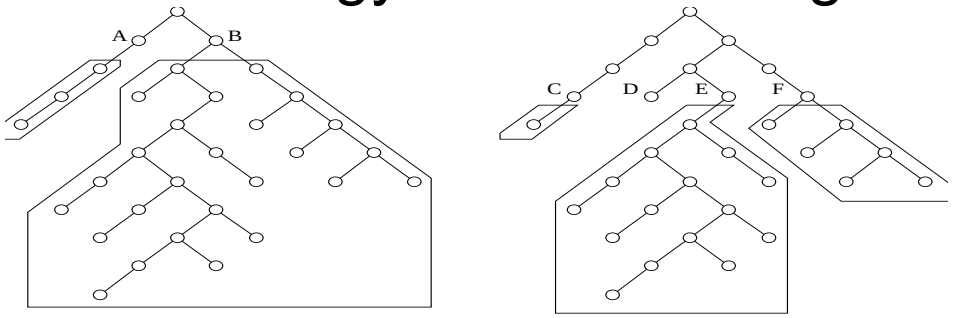
Párhuzamos diszkrét optimalizáció: motiváció

- DOP általában NP-teljes problémák. Segít-e a párhuzamosítás?
- Sok problémánál az átlagos eset polinomiális idejű.
- Sokszor viszonylag kis állapotterünk van, de valós idejű megoldásra van szükségünk.

- Többletráfordítás
  - A soros és párhuzamos keresés közötti munkamennyiség gyakran különböző.
  - Ha W a soros munka és W_P a párhuzamos, a többletráfordítás $s = W_P/W$.
  - A sebességnövekedés határa $p \times (W/W_P)$.

Párhuzamos mélységi keresés

- A keresési teret miként osszuk fel processzorok között?
- A különböző ágakat lehet párhuzamosan keresni.
- De az ágak nagyon eltérő méretűek lehetnek.
- Nehéz becsülni az ágak méretét a gyökerükből.
- Dinamikus terhelés kiegyenlítés szükséges.



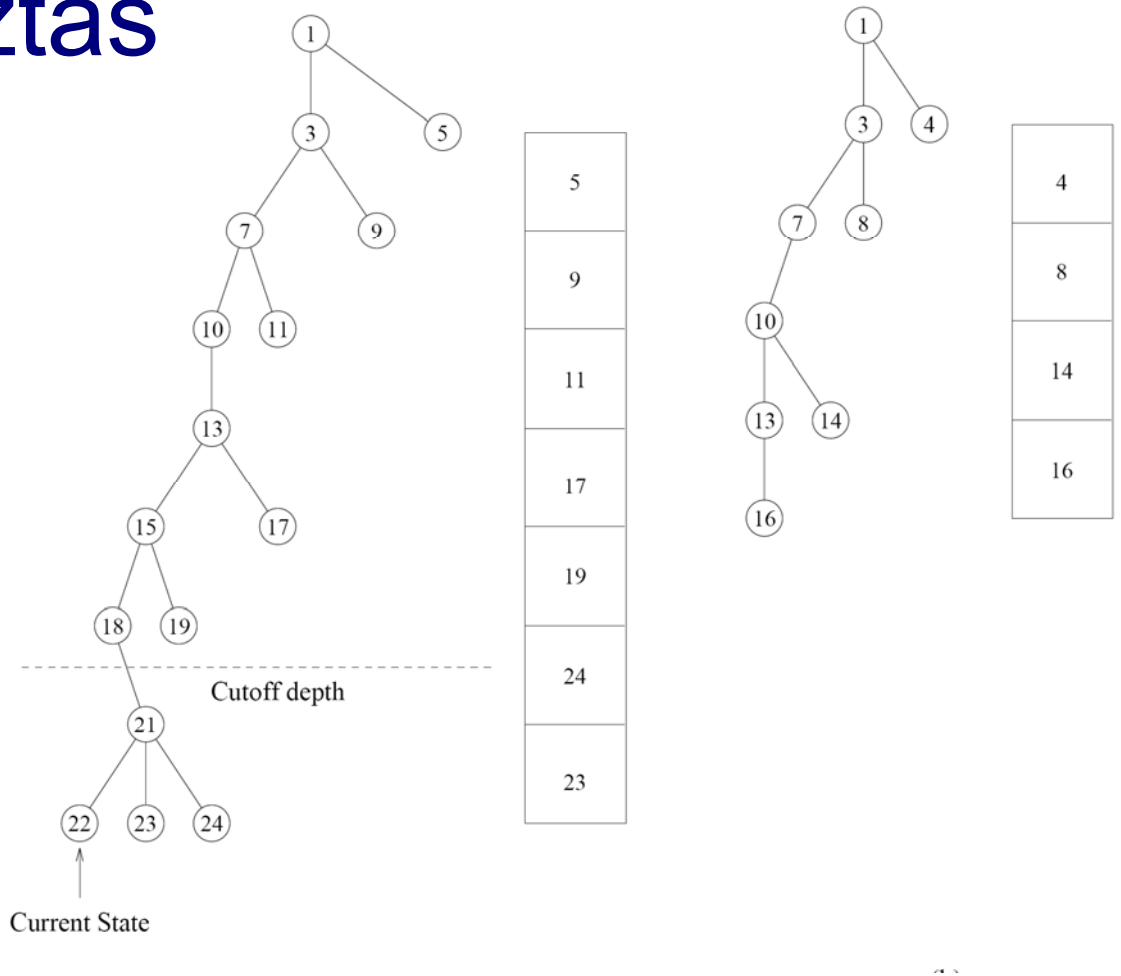
A statikus dekompozíció strukturálatlan fakeresést és nem kiegyensúlyozott terheléshez vezet.

Párhuzamos mélységi keresés: dinamikus terhelés kiegyenlítés

- Amikor egy processzor befejezi munkáját, akkor egy másiktól kér.
- Osztott memóriás modellnél ez lockolással és a munka szétvágásával történik (lásd később).
- Ha valamely processzor megoldáshoz és, akkor a többi terminál.
- Az eddig fel nem dolgozott részeket saját veremben tárolhatja minden processzor.
- Kezdetben az egész teret egy processzorhoz rendeljük.

Párhuzamos mélységi keresés: dinamikus terhelés kiegyensúlyozás

Párhuzamos mélységi keresés: munkafelosztás



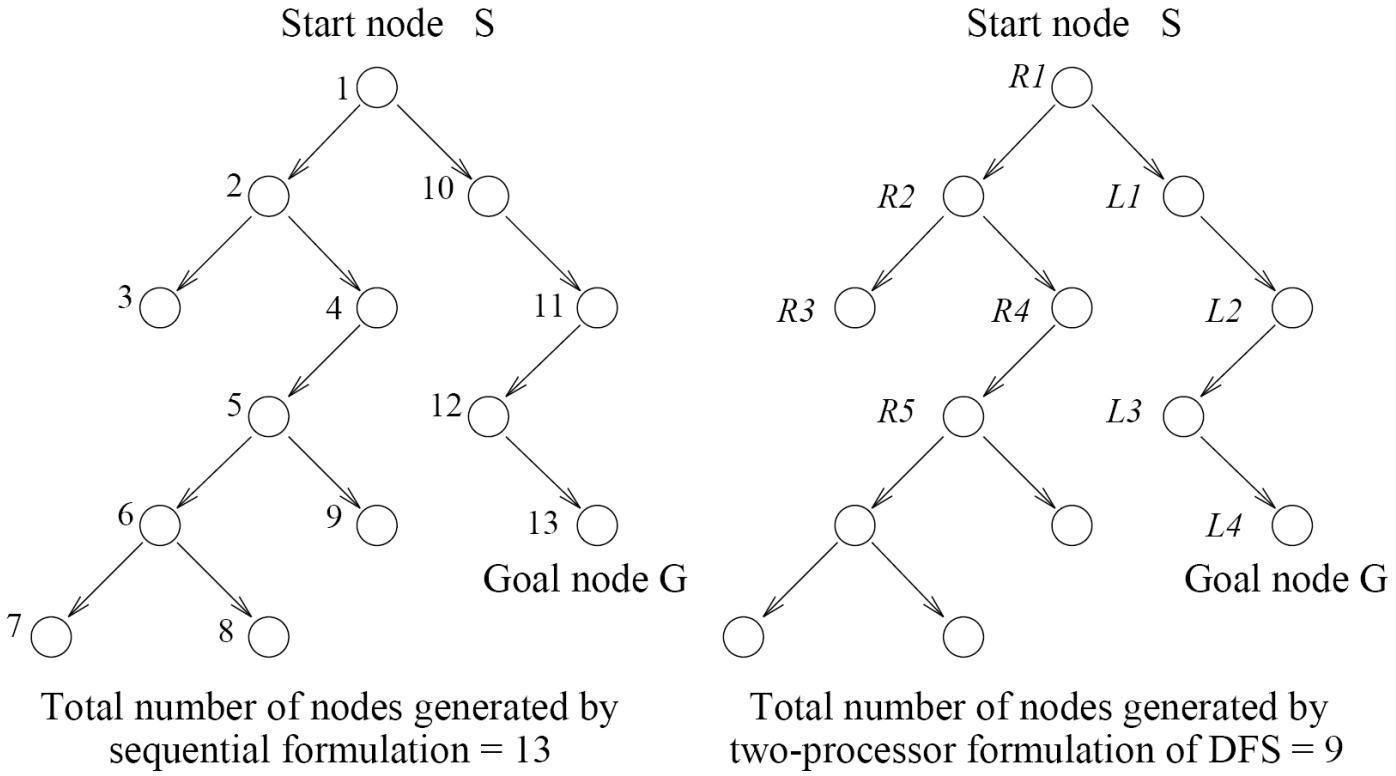
Mélységi keresés: a két részfa a verem reprezentációval.

Terhelés kiegyenlítés

A munka kiosztásának sémája:

- Aszinkron (lokális) körbeforgó: minden processzor egy számlálót tart karban ((*target* +1) mod *p*) és az igényét a felé közvetíti.
  - Nagyszámú munkaigényt generál.
- Globális körbeforgó: A rendszer tart karban egy számlálót és az igények körbeforgó módon kerülnek kezelésre.
  - Legkevesebb munkaigényt generálja.
- Véletlen ciklikus: Véletlenszerűen kerül kiválasztásra egy processzor.
  - Megfelelő kompromisszum.

Sebesség anomáliák párhuzamos keresésnél

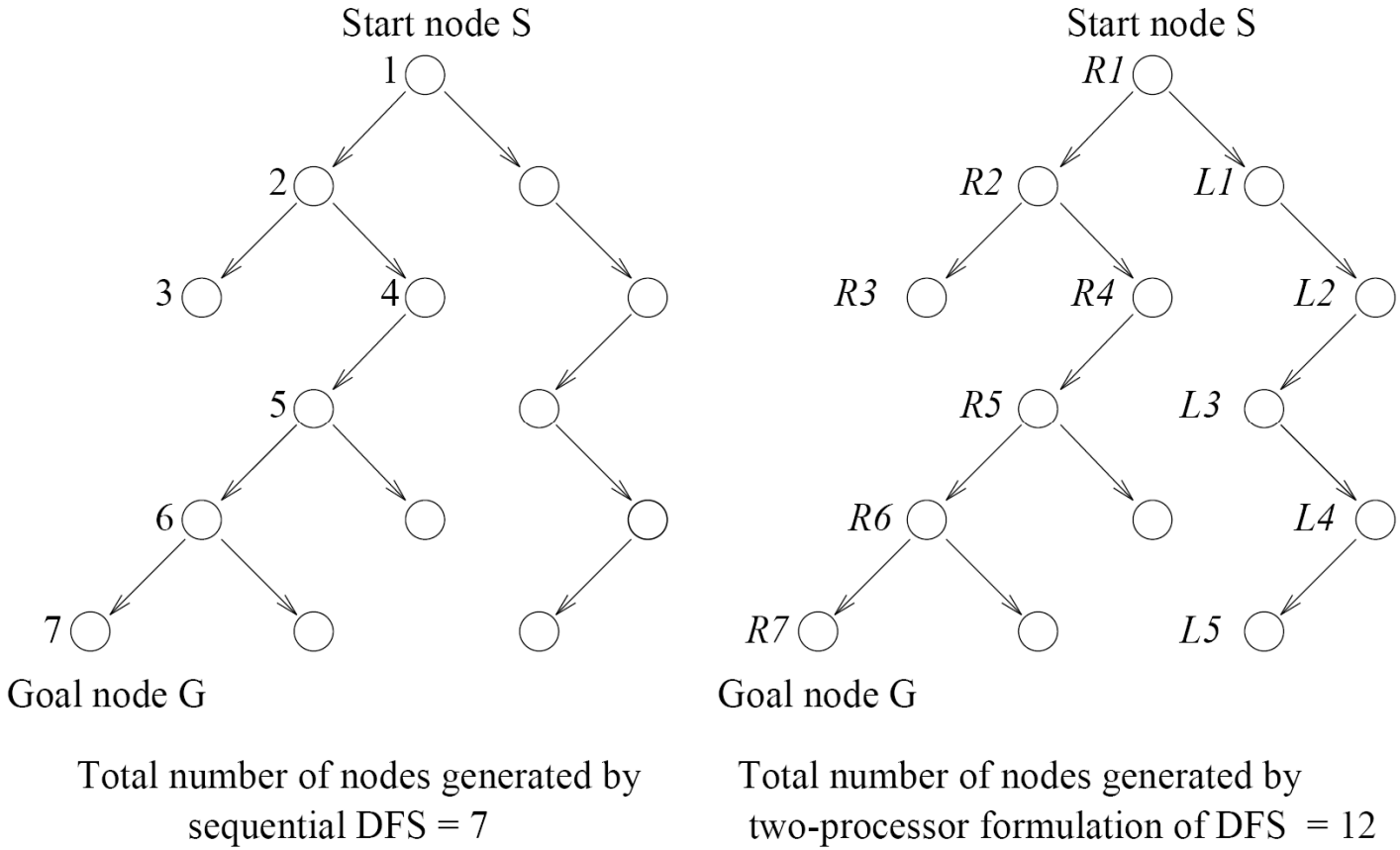


Total number of nodes generated by sequential formulation = 13

Total number of nodes generated by two-processor formulation of DFS = 9

A párhuzamos mélységi keresés kevesebb csomópontot jár be, mint a soros.

Sebesség anomáliák párhuzamos keresésnél



A párhuzamos mélységi keresés több csomópontot jár be, mint a soros.