PROGRAMOZÁSMÓDSZERTAN
6. ELŐADÁS'2004
(VÁZLAT)

TOVÁBBI KERESÉSI TÉTELEK

1. Általános keresés

Induljunk ki a lineáris keresés már megismert algoritmusából, és hántsunk le róla minden konkrétat, adjuk meg a „leglényegét"!

Mik azok, amik túl konkrétak az adott algoritmusban?

- a sorozat éppen **N** elemű
- a sorozat „bejárása" (elemeinek sorravétele) éppen **növekvő indexek** irányába halad (persze nem lenne jobb a fordított sorrendű sem)
- a sorozat **tömb**(szerű) struktúrában tárolódik

Absztrakció:

Vezessük be az alábbi fogalmakat:

- $L \in H^*$ ***lehetségesek sorozata*** az eredeti X egy alkalmasan választott (algoritmus meghatározta) rész-sorozata ($L \subseteq X$)
- $\|.\|: H^* \to \mathbb{N}$ ***hossz*** a sorozat hossza (elemszáma)
- $\sigma: H^* \to H^*$ ***szűkítés*** $L \in H^*: (x \in \sigma(L) \Rightarrow x \in L) \wedge \|\sigma(L)\| < \|L\|$ – azaz a σ L-ből elhagy valahány elemet
- $\varepsilon: H^* \to \mathbb{N}$ ***elemkijelölés*** $L \in H^*: \varepsilon(L) \in [1..\|X\|]$ – azaz ε L egy lehetséges elemének indexét adja

A σ és az ε persze egymással szorosan összefüggnek. Ezt fejezi az alábbi állítás:

$$\mathbf{x_{\varepsilon(L)} \in L \ \wedge \ x_{\varepsilon(L)} \notin \sigma(L)}.$$

Azaz a szűkítés során legalább az *elemkijelölő függvény* által meghatározott elemmel „rövidül meg" a sorozat.

Megjegyzés:

Sok esetben ennél több is teljesül: $\sigma(L) \subset L$. Ekkor nyilvánvalóan teljesül a fenti, azaz $\sigma(L) \subset L \Rightarrow ((x \in \sigma(L) \Rightarrow x \in L) \wedge \|\sigma(L)\| < \|L\|)$.

Az absztrakt keresés tétel algoritmusa

Ezen fogalmakkal az általános keresés tétel algoritmusa az alábbi lesz:

```
1) L:=X
2) Ciklus amíg ‖L‖>0 és nem T(X(ε(L)))
3)   L:=σ(L)
4) Ciklus vége
5) Van:=‖L‖>0
6) Ha Van akkor Sorsz:=ε(L)
```

Ellenőrzésképp, valamint hogy jobban megértsük a fenti kulcsfontosságú függvények hatását, kövessük működés közben az algoritmust! Alkalmazzuk egy $X=(x_1,x_2,\ldots x_N)$ sorozatra, amelynek –tegyük föl– nincs T-tulajdonságú eleme:

az algoritmus éppen végrehajtott utasítása	*L*	*a ciklusfeltétel kiértékelése*
1) L:=X	$(x_1,x_2,\ldots x_N)$	
2) **Ciklus amíg** ‖L‖>0 **és nem** T(X(ε(L)))		$\|(x_1,x_2,\ldots x_N)\|=N>0$ [≡Igaz] **nem** $T(X(\varepsilon((x_1,x_2,\ldots x_N))))=$ **nem** $T(\mathbf{x}_{\iota_1})$ [≡Igaz]
3) L:=σ(L)	$(x_1,\ldots x_{\iota_1-1},x_{\iota_1+1},\ldots x_N)$	
2) **Ciklus amíg** ‖L‖>0 **és nem** T(X(ε(L)))		$\|(x_1,\ldots x_{\iota_1-1},x_{\iota_1+1},\ldots x_N)\|=N-1>0$ [≡Igaz] **nem** $T(X(\varepsilon((x_1,\ldots x_{\iota_1-1},x_{\iota_1+1},\ldots x_N))))=$ **nem** $T(\mathbf{x}_{\iota_2})$ [≡Igaz]
3) L:=σ(L)	$L: x_{\iota_1},x_{\iota_2}\notin L$	
...		
3) L:=σ(L)	$L: x_{\iota_1},\ldots x_{\iota_{N-1}}\notin L$	
2) **Ciklus amíg** ‖L‖>0 **és nem** T(X(ε(L)))		$\|(x_{\iota_N})\|=1>0$ [≡Igaz] **nem** $T(X(\varepsilon((x_{\iota_N}))))=$ **nem** $T(\mathbf{x}_{\iota_N})$ [≡Igaz]
3) L:=σ(L)	$L: x_{\iota_1},\ldots x_{\iota_N}\notin L=()$	
2) **Ciklus amíg** ‖L‖>0 **és nem** T(X(ε(L)))		$\|()\|=0>0$ [≡Hamis]
5) Van:=‖L‖>0		
6) **Ha** Van **akkor** Sorsz:=ε(L)		

A lineáris keresés tétel

Nézzük a lineáris keresés tétel újrafogalmazását az általános keresés algoritmusa alapján!

- L $L:=(x_i,\ldots x_N)$ *i*: ahol tartunk az algoritmusban (i=1..N+1)
  L egyértelműen jellemezhető, sőt helyettesíthető az algoritmus végrehajtásakor éppen érvényes első elemének indexével: $L:=L(i)\rightarrow i$

Azaz minden L-re vonatkozó műveletben i-vel operál(hat)unk.

- σ σ(L):=σ(x$_i$,…x$_N$) ⊆ (x$_{i+1}$,…x$_N$) – i eggyel növelését jelenti
  Vagyis σ(L)=σ(L(*i*))=σ(*i*)→*i*+1
- ε ε(L):=*i* – a „mindenkori” vizsgált elem: L(*i*)-beli első elem indexe
  Azaz ε(L)=ε(L(*i*))=ε(*i*)→*i*

Ekkor ‖L‖=‖L(*i*)‖:=N-*i*+1, s az algoritmus maga:

Az absztrakt keresés	***A lineáris keresés***
`L:=X`	`i:=1` [X 1. eleménél kezdődik a keresés]
Ciklus amíg ‖L‖>0	**Ciklus amíg** N-i+1>0 [1] [van még hol keresni]
és nem T(X(ε(L)))	**és nem** T(X(i)) [az i. olyan-e]
L:=σ(L)	i:+i [a következőnél kezdődik a sorozat]
Ciklus vége	**Ciklus vége**
Van:=‖L‖>0	Van:=N-i+1>0
Ha Van **akkor** Sorsz:=ε(L)	**Ha** Van **akkor** Sorsz:=i [a keresett]

Hatékonyság (N>0):

Hasonlításszám	***Min***	***Átlag***	***Max***	
Van keresett	1	N/2	N	O(N)
Nincs keresett	N	N	N	Θ(N)

„Szokásos” σ/ε-függvénydefiníciók

A közismert keresésekben az alábbi értelmezésű függvényeket szoktuk párban használni:

ε	***σ***
ε$_e$((x$_1$,...x$_N$)):=*1* – *első*	σ$_e$((x$_1$,...x$_N$)):=(x$_2$,...x$_N$) – *első elem nélkül*
ε$_u$((x$_1$,...x$_N$)):=*N* – *utolsó*	σ$_u$((x$_1$,...x$_N$)):=(x$_1$,...x$_{N-1}$) – *utolsó elem nélkül*
ε$_k$((x$_1$,...x$_k$,…x$_N$)):=*k*, ahol *k*=(*1*+*N*) *Div 2* – *középső*	σ$_{<k}$((x$_1$,...x$_N$)):=(x$_1$,...x$_k$), ahol *k*=((*1*+*N*) *Div 2*)-1 – *középső elem előttiek* (világos, hogy (x$_1$,...x$_k$)⊂(x$_1$,...x$_N$) teljesül) σ$_{>k}$((x$_1$,...x$_N$)):=(x$_k$,...x$_N$), ahol *k*=((*1*+*N*) *Div 2*)+1 – *középső elem utániak* (világos, hogy (x$_k$,...x$_N$)⊂(x$_1$,...x$_N$) teljesül)

[1] N-i+1>0 ⇔ N>i-1 ⇔ **N≥i** . S ez a lineáris keresésben előforduló ciklusfeltétel első tényezője.

$\varepsilon_e((x_1,...x_N)):=1$ – *első*	$\sigma_<((x_1,...x_N)):=(x'_1,...x'_k)$, ahol $\forall i\in[1..k]: x'_i\in X \wedge x'_i<x_1$ – *elsőnél kisebbek* (világos, hogy $(x'\in\sigma(L) \Rightarrow x'\in L) \wedge \|\sigma(L)\|<\|L\|$ teljesül, hiszen $\forall i\in[1..k]: x'_i\neq x_1$)

Megjegyzés:

Talán sejthető, hogy a 2. ε/σ-pár a *hátulról* „bejárás" kellékei. A 3. a hamarosan sorra kerülő *logaritmikus keresés*hez, a 4. pedig a rekurzió-típus kapcsán szóba kerülő *Quicksort rendezés*hez tartoznak.

2. Lineáris keresés rendezett sorozatban

LinKeresésRendezettben$(H^*,H,F(H\times H,\mathbb{L})):\mathbb{L}\cup\mathbb{L}\times\mathbb{N}$

Be: $N\in\mathbb{N}$, $X\in H^*$, $y\in H$, $\leq\in F(HxH,\mathbb{L})$ $(T_y:H\to\mathbb{N},\ T_y(h):=(h=y))$

Ki: $Van\in\mathbb{L}$, $Sorsz\in\mathbb{N}$

Ef: $N=Hossz(X) \wedge RendezettHalmaz_{\leq}(H) \wedge RendezettSorozat_{\leq}(X)$

Uf: $Van \equiv \exists i\in[1..N] : x_i=y \ \wedge\ Van \Rightarrow (Sorsz\in[1..N] \wedge x_{Sorsz}=y \wedge \forall i\in[1..Sorsz) : x_i<y)$

Az absztrakt algoritmus elé:

- L $L:=(x_i,\ldots x_N)$ – elejétől végig, i: ahol tartunk a végrehajtás során = 1..N+1

  Figyelem, a *rendezettség* kihasználandó! A végrehajtás véget érhet még annak előtte, hogy az L sorozat N. eleméhez érnénk! Pontosan akkor, amikor az $x_i>y$ első ízben bekövetkezik (elölről haladva).

  L egyértelműen jellemezhető, sőt helyettesíthető az algoritmus végrehajtásakor éppen érvényes első elemének indexével: $L:=L(i)\to i$

  Azaz minden L-re vonatkozó műveletben *i*-vel operál(hat)unk.

- σ **(1)** $\sigma(L(i))\to(x_{i+1},...x_N)$, ha $x_i\leq y$

  (2) $\sigma(L(i))\to(x_{N+1},...x_N)$, ha $x_i>y$

  (Az világos, hogy **(1) és (2)** $\Rightarrow \sigma(L(i))\subseteq(x_i,...x_N)=L(i)$)

  Azaz $\sigma(L)=\sigma(L(i))=\sigma(i)\to$Ha $x_i\leq y$ akkor i:+1 különben i:=N+1.

- ε $\varepsilon(L)=\varepsilon_1(L(i)):=i$ – a „mindenkori" vizsgált elem: L(i)-beli első elem indexe

  Azaz $\varepsilon(L)=\varepsilon_1(L(i))=\varepsilon_1(i)\to i$

Ekkor $\|L(i)\|=N-i+1$, s így $\|L(i)\|>0 \Leftrightarrow N-i+1>0 \Leftrightarrow N\geq i$.

„Vezessük le” az absztraktból:

Az absztrakt keresés	***Lineáris keresés rendezettben***
`L:=X`	`i:=1` [X 1. eleménél kezdődik a keresés]
Ciklus amíg ‖L‖>0	**Ciklus amíg** N≥i [van még hol keresni]
és nem T(X(ε(L)))	**és** X(i)≠y [az i. olyan-e]
L:=σ(L)	**Elágazás**
	X(i)≤y esetén i:+1
	X(i)>y **esetén** i:=N+1
	Elágazás vége
Ciklus vége	**Ciklus vége**
Van:=‖L‖>0	Van:=N≥i
Ha Van **akkor** Sorsz:=ε(L)	**Ha** Van **akkor** Sorsz:=i [a keresett]

Az algoritmus bizonyos redundanciákat rejt, amiket az alábbiakban felderítünk és megszüntetünk.

Vegyük észre, hogy

A ciklusból két ok miatt léphetünk ki: „**N<i vagy X(i)=y**”

N<i	találtunk nagyobbat, azaz a *korábbi* ***i***-re: **X(i)>y** (l. σ definíciójának 2. ágát) vagy „simán” a sorozat végére értünk: **N+1=i**
X(i)=y	megtaláltuk a keresettet.

Az első olyan i, amelyre **X(i)>y** teljesült, rögvest beállítjuk az **N+1=i** ciklusfeltétel (egy) terminálló értékét. Tehát figyelembe véve a **X(i)≠y** feltételt, világos, hogy eddig az elágazásnak csak az **X(i)≤y**-hoz tartozó ága lett végrehajtva. Ezért „logikusnak” látszik redukálni a ciklusmagot az elágazás ezen ágára. Persze ez csak úgy történhet, hogy a hozzá tartozó feltétellel bővítjük a ciklusfeltételt… Azaz

```
Ciklus N≥i és X(i)≠y és X(i)≤y
  i:+1
Ciklus vége
```

E lépéssel viszont összekavartunk két eddig könnyen szétválasztható esetet. Figyeljük meg, mi teljesült a ciklus után a korábbi (még helyes) algoritmusban, és mi a mostaniban:

Ha …	*Korábban*	*Most*
…benne van	N≥i [**és** X(i)=y]	**N≥i és** X(i)=y
…nincs benne	i>N	**N≥i és** X(i)>y **[**mert sebtében kiléptünk**]** **vagy** i>N **[**mert a sorozat végére értünk**]**

Az **N≥i** feltétel már egymaga nem képes megválaszolni a keresés sikerességének kérdését. Ebből következőleg a folytatást is módosítani kell!

```
Van:=N≥i és X(i)=y
```

Az algoritmus ezek után:

Alg:

```
Eljárás LinKeresRendben(Konstans N:Egész, X:THk, Y:TH
                        Változó Van:Logikai, Sorsz:Egész):
  Változó
    i:Egész
  i:=1
  Ciklus amíg N≥i és X(i)<y
    i:+1
  Ciklus vége
  Van:=N≥i és X(i)=y
  Ha Van akkor Sorsz:=i
Eljárás vége.
```

Hatékonyság (N>0):

Hasonlításszám	***Min***	***Átlag***	***Max***	
Van keresett	1+1[2]	(N+3)/2[3]	N+1	O(N)
Nincs keresett	1+1	(N+5)/2[4]	N+1	O(N)

Vö. a lineáris kereséssel!

3. Logaritmikus keresés

LogaritmikusKeresés$(H^*,H,F(H\times H,\mathbb{L})):\mathbb{L}\cup\mathbb{L}\times\mathbb{N}$

Be: $N\in\mathbb{N}$, $X\in H^*$, $y\in H$, $\le\in F(H\times H,\mathbb{L})$ $(T_y:H\to\mathbb{L},\ T_y(h):=(h=y))$

Ki: $Van\in\mathbb{L}$, $Sorsz\in\mathbb{N}$

Ef: $N=Hossz(X)\wedge RendezettHalmaz_{\le}(H)\wedge RendezettSorozat_{\le}(X)\wedge$

$\exists elem:H^*\times\mathbb{N}\to H$ *szelekciós függvény:* $elem(X,i)=x_i$ [5]

Uf: $Van\equiv\exists i\in[1..N]: x_i=y \wedge Van \Rightarrow (Sorsz\in[1..N]\wedge x_{Sorsz}=y)$

Az absztrakt algoritmus elé:

- L L:=(x_e,…x_u) – a mindenkori elejétől az utolsóig, kezdetben (e,u)=(1,N)
  L egyértelműen jellemezhető, sőt helyettesíthető az algoritmus végrehajtásakor éppen érvényes első + utolsó elemének indexével: L:=L(e,u)→(e,u)
  Azaz minden L-re vonatkozó műveletben *e-vel és/vagy u*-val operál(hat)unk.

[2] A „+1” a ciklus utáni hasonlítást számlálja.

[3] ((1+1) `[X(1)=y]`+ (2+1) `[X(2)=y]`+…+(N+1) `[X(N)=y]`)/N=(N*(N+1)/2+N)/N = (N/2*((N+1)+2))/N = (N+3)/2

[4] ((1+1) `[X(1)>y]`+ (2+1) `[X(2)>y]`+…+(N+1) `[X(N)>y]`+(N+0) `[X(N)<y]`)/N=((N*(N+1)/2+N)+N)/N = =(N/2*((N+1)+4))/N = (N+5)/2

[5] Ugyanez kifejezhető úgy is, hogy X közvetlen elérésű.

- σ **(1)** $\sigma(L(e,v)):=\underline{\sigma_{<k}}(L(e,v))$, ha $x_{\varepsilon(L(e,v))}<y$
  (2) $\sigma(L(e,v)):=\sigma_{>k}(L(e,v))$, ha $x_{\varepsilon(L(e,v))}>y$
  Azaz $\sigma(L)=\sigma(L(e,v))=\sigma(e,v)\rightarrow$ **Elágazás**
  $x_{\varepsilon(L(e,v))}<y$ **esetén** $(e,v):=(e,\varepsilon(L(e,v))-1)$
  $x_{\varepsilon(L(e,v))}>y$ **esetén** $(e,v):=(\varepsilon(L(e,v))+1,v)$
  Elágazás vége.
- ε $\varepsilon(L(e,v)):=\varepsilon_k(L(e,v)):=(e+v)$ Div 2 – a mindenkori sorozat középső elemének indexe
  Azaz $\varepsilon(L)=\varepsilon_k(L(e,v))=\varepsilon_k(e,v)\rightarrow(e+v)$ Div 2

Ekkor $\|L(e,v)\|=v-e+1$, s így $\|L(e,v)\|>0 \Leftrightarrow v-e+1>0 \Leftrightarrow v\geq e$.

„Vezessük le” az absztraktból:

Az absztrakt keresés	***Logaritmikus keresés***
`L:=X`	`(e,v):=(1,N)`
Ciklus amíg `‖L‖>0`	**Ciklus amíg** `v≥e` [van még hol keresni]
és nem `T(X(ε(L)))`	**és** `X((e+v) Div 2)≠y` [az i. olyan-e]
`L:=σ(L)`	**Elágazás**
	`X((e+v)` **Div** `2)<y` **esetén**
	`(e,v):=(e,(e+v)` **Div** `2-1)`
	`X((e+v)` **Div** `2)>y` **esetén**
	`(e,v):=((e+v)` **Div** `2+1,v)`
	Elágazás vége
Ciklus vége	**Ciklus vége**
`Van:=‖L‖>0`	`Van:=v≥e`
Ha `Van` **akkor** `Sorsz:=ε(L)`	**Ha** `Van` **akkor** `Sorsz:=(e+v)` **Div** `2`

Megjegyzés:

Az egyszerűbb leírhatóság és a *többszörös kiszámolás* elkerülése érdekében tároljuk az ε(L(e,v))-dik (=(e+v) DIV 2) elem indexét a *k* változóban, e leválasztás viszont a külön számolást jelent! Másrészt az „X(k)≠y” feltétel a ciklusfeltételben *egy feltétel vizsgálattal elintézhetőre* egyszerűsíti a ciklusmagbeli elágazást. Harmadrészt, az *üres értékadások*at elhagyjuk. Így az alábbi algoritmus marad:

Alg:

```
Eljárás LogKeresés(Konstans N:Egész, X:THk, Y:TH
                   Változó Van:Logikai, Sorsz:Egész):
  Változó
    e,v,k:Egész
```

```
  (e,v):=(1,N); k:=(e+v) Div 2
  Ciklus amíg v≥e és X(k)≠y
    Ha X(k)<y akkor  (e,v):=(e,k-1)
             különben (e,v):=(k+1,v)
    k:=(e+v) Div 2
  Ciklus vége
  Van:=v≥e
  Ha Van akkor Sorsz:=k
Eljárás vége.
```

Hatékonyság (N>0):

Hasonlításszám	***Min***	***Átlag***	***Max***	
Van keresett	1	$\sim 2*(\log_2(N)-1)$[6]	$\lceil 2*\log_2(N) \rceil$	$O(\log_2(N))$
Nincs keresett	$\lceil 2*\log_2(N) \rceil$	$2*\log_2(N)$	$\lceil 2*\log_2(N) \rceil$	$\Theta(\log_2(N))$

Vö. a lineáris keresés rendezettben hatékonyságával!

Megjegyzés:

A fenti számításban szereplő kettesalapú logaritmus (is) magyarázza, miért emlegetik e keresést gyakorta „bináris keresés”-ként.

3. Visszalépéses keresés (backtrack)

3.1. A „backtrack” gondolat

Bevezetésként gondolkozzunk el egy-két tipikus feladaton és ezek megoldásvázlatain.

1. feladat:

Adott egy N×N-es sakktábla. Helyezzünk el rajta N vezért úgy, hogy ne üssék egymást!

1. megoldásvázlat:

Helyezzük el valahogy az N vezért, majd döntsük el, ütik-e egymást! Ha igen, helyezzük el másként. Addig próbálkozzunk, amíg sikert nem érünk.

Bajok:

1. Hogyan helyezzük el a vezéreket úgy, hogy nyugodtak lehessünk: nem marad ki véletlenül egy lehetséges elhelyezés.
2. Nagyon sok lehetőséget kell kipróbálni. Jelen esetben: $\binom{N^2}{N}$.

   Ui.: az 1. vezérnek N^2 lehetősége van, a 2.-nak eggyel kevesebb, az N.-nek N^2-N+1.

2. megoldásvázlat:

Minden vezérhez rendeljünk egy oszlopot, amelybe el próbáljuk helyezni. Az elhelyezés után ellenőrizzük, ütik-e egymást! Ha igen, válasszuk a következő helyet a saját oszlopában az N.-nek. Ha összes lehetőségét kimerítettük az N.-nek, akkor válasszuk az N-1. kö-

[6] Az átlagos hasonlításszám kalkulálásához meg kell határozni az összehasonlítás fa átlagosághosszat. Ez egyenletes eloszlást feltételezve n! számú ág hosszának átlagolását jelenti. Mivel a fa „lefelé” exponenciálisan terebélyesedik, ezért az átlag alig lesz kevesebb, mint a maximális.

vetkezőjét. Így gondolkozunk, ha az N-1. helyeit is kimerítettük… Addig próbálkozzunk, amíg sikert nem érünk.

Bajok közül az 1.-n túl tehetjük magunkat, a 2. is mérséklődött, de azért elégedettek nem lehetünk. Hiszen még mindig nagyszámú lehetőség végiggondolásával érhetünk csak célt: N^N.

3. megoldásvázlat:

Ha magunknak kellene kézzel a vezéreket elhelyezni, bizonyára az alábbi szisztéma mellett döntenénk:

Sorszámozzuk meg az oszlopokat és a sorokat 1-től N-ig. Helyezzük el az 1. vezért az 1. oszlopba! Keressük meg a 2. vezérnek az első adandó mezőt a 2. oszlopba, éít. Ha egyik vezérnél már nem találunk alkalmas mezőt, akkor –azt gondolván, hogy az eddigiek vannak rossz helyen– az előzőnek keresünk egy „jobbnak" ígérkező helyett (az eddig még nem vizsgáltak között). Vagy, ha már neki sincs további alkalmasnak tűnő, akkor az azt megelőzővel foglalkozunk, vagy az azt megelőzővel éít., addig amíg meg nem találtuk az igazi elhelyezést…

Valahogy így: nézzük meg a demonstrációt!

2. feladat:

Hányféleképpen lehet címletezni N Ft-ot? Nyilván adottak a címletezésben szereplő bankjegyek, ill. pénzérmék.

1. megoldásvázlat:

Vegyük a legkisebb névértékű pénzérmét vagy bankjegyet, s határozzuk meg, belőle hány kellene az N Ft felváltásához. (Megengedjük, hogy ez pontosan nem sikerül, ekkor azt a legkisebb számot kerestük, amely darabnyi ebből nem kevesebb, mint N Ft.) Az biztos, hogy minden megoldás ennél nem több címletet használ föl. Állítsuk tehát elő valamilyen szisztematikus sorrendben az összes legfeljebb ennyi címletet tartalmazó sorozatot, majd számoljuk meg hánynak éppen N az értéke.

2. megoldásvázlat:

Rakjuk –mondjuk érték szerint csökkenő– sorrendbe a címletezésben felhasználható bankjegyeket és pénzérméket. Állítsunk elő egy lehetséges felváltását az N Ft-nak az alábbiak szerint.

Válasszuk ki azt a címletet, amely nem nagyobb, mint a címletezendő összeg, és vonjuk le belőle a címlet értékét. Járjunk így a maradék összeggel. Világos, hogy egyre kisebb címletekkel próbálkozunk. Ha a maradék összeg már nem címletezhető, akkor a legutóbbi választásunkat visszavonjuk (visszaadva értékét a maradék összeghez) és a következő (kisebb értékű) címlettel teszünk kísérletet.

Ha egy címletezést megkaptunk, akkor a megoldás-számlálót növeljük, majd a következő megoldást ebből kiindulva úgy próbáljuk megtalálni, hogy „eldobjuk” az utolsó címletet – mintha rossz lett volna– és keresünk egy másikat, éppenúgy, ahogy eddig tettük.

Az biztos, hogy ugyanazt a címletezést kétszer nem kaphatjuk meg, mert címlet*kombináció*kat (azaz sorrendfüggetlen címlet-összeállítást) generáltunk (a rendezettség és a szisztematikus választás miatt).

3. feladat:

Egy éhes egérnek egy labirintusban elhelyeznek egy darab sajtot. Segítsünk az egérnek megkeresni a sajthoz vezető utat!

Megoldásvázlat:

Kiindulási ponttól lépésről lépesre a következőt tesszük. A labirintus folyosójának egy-egy pontjából odébb lépni többnyire 2 irányba lehet: előre és vissza. A visszafelé lépést el kell kerülni, sőt általában is elkerülendő egy olyan helyre lépés, ahol már játunk. (Ellenkező esetben végtelen ciklusban kódorognánk a labirintusban.) Ha azonban egy útelágazáshoz érünk, akkor szisztematikusan válasszunk az irányok közül. Mindaddig haladunk, amíg zsákutcába nem jutunk. Ekkor visszavonogatjuk lépéseinket mindaddig, amíg egy olyan útelágazáshoz nem érünk, ahol van még eddig nem választott irány.

Valahogy így: nézzük meg a demonstrációt! (Vajon milyen sorrendben[7] „néz körül” az egér? Állapítsa meg a demonstráció alapján!)

3.2. A backtrack alapú keresés

Milyen feladat tehát a „backtrack”-feladat? Vonjunk le következtetéseket a példákból!

1. ***Kimenet***: mindegyiknél *egy sorozat*ot állítunk elő.	**1. feladat**: ~~mező koordináták /~~ sor-indexek sorozata **2. feladat**: a címletek sorozata **3. feladat**: a lépésirányok sorozata
2. ***Bemenet***: *több sorozat*, amelyekből (mindegyikből egy-egy) állítjuk elő a kimenet egyetlen sorozatát.	**1. feladat**: i. alapsorozat=i. oszlop **2. feladat**: i. alapsorozat=címletek sorozata **3. feladat**: i. alapsorozat=lépésirányok sorozata
3. Valamiféle ***összefüggés*** kapcsolja össze az eddig kiválasztott értékeket és a következőként kiválasztható értéket.	**1. feladat**: az (i-1).-ig letettek ne üssék az i.-et **2. feladat**: az (i-1).-ig kiválasztott címleteknél nem lehet nagyobb az i. címlet, és össz értékük nem lehet nagyobb N Ft-nál **3. feladat**: az i. lépés után nem lehet az egér olyan helyen, ahol már járt.

[7] A „körülnézés” abszolút sorrendben történik (azaz nem függ attól, hogy az egér éppen milyen irányban halad): Nyugat, Észak, Kelet, Dél.

BacktrackKeresés$\left(\times_{(i=1..K)}H_i^*,\ F(\cup_{(j=1..K)}(\times_{(i=1..j)}H_i),\mathbb{L})\right):\mathbb{L}\cup\mathbb{L}\times\mathbb{N}^*$[8]

Egy másik szignatúra is rendelhető a feladathoz:

BacktrackKeresés$\left(\times_{(i=1..K)}H_i^*,\ F(\mathbb{N}^*,\mathbb{L})\right):\mathbb{L}\cup\mathbb{L}\times\mathbb{N}^*$[9]

Be: $K\in\mathbb{N}$, $M\in\mathbb{N}^*$ – K elemszámú méretsorozat,

$Y_i\in H_i^*$ $(i\in[1..K])$ – m_i elemszámú lehetőségsorozatok,

$l:\ \bigcup_{i=1}^{K}\ \times_{j=1}^{i} H_j\to\mathbb{L}$ – lehetőségek összeférése,

Ki: $Van\in\mathbb{L}$, $X\in\mathbb{N}^*$ – az összeférő lehetőségek indexeinek sorozata

Ef: $K=Hossz(M)\wedge K\geq 2\ \wedge$

$\forall i\in[1..K]:\ (\ M_j=Hossz(Y_j)\wedge$

$HalmazFölsorolás(Y_j)\ )\wedge$

$\forall j\in[1..K],\ \forall i\in[1..j]:\ l(h_1,...h_j)\Rightarrow l(h_1,...h_i)$

az első változat előfeltétele

Uf: $Van\equiv\exists Z\in Y_1\times Y_2\times...\times Y_K:\ l(Z)\ \wedge$

$Van\Rightarrow(\forall i\in[1..K]:\ y^{(i)}_{x_i}\in Y_i)\wedge l(y^{(1)}_{x_1},...y^{(K)}_{x_K})$

az első változat utófeltétele

Ef: $K=Hossz(M)\wedge K\geq 2\ \wedge$

$\forall i\in[1..K]:\ (\ M_j=Hossz(Y_j)\wedge$

$HalmazFölsorolás(Y_j)\ )\wedge$

$\forall j\in[1..K],\ \forall i\in[1..j]:\ l(n_1,...n_j)\Rightarrow l(n_1,...n_i)$

a második változat előfeltétele

Uf: $Van\equiv\exists Z\in[1..m_1]\times[1..m_2]\times...\times[1..m_K]:\ l(Z)\ \wedge$

$Van\Rightarrow(\forall i\in[1..K]:\ x_i\in[1..m_i])\wedge l(x_1,...x_K)$

a második változat utófeltétele

Az absztrakt algoritmus elé:

- L A keresett megoldás a $\times_{i=1}^{K} Y_i$ K-dimenziós (L-) tér egy pontja, amely kielégíti az l-t.

  Ennek megadását az X-tömb végzi a Y_i-beli komponensek *indexei*nek felsorolásával. A megoldás keresése során e teret kell ügyesen bejárnunk, lehetőleg úgy, hogy ne kelljen mind az M_1*...* M_K „pontjának” vizsgálatát elvégeznünk. Ezt úgy tesszük, hogy szisztematikusan szűkítjük a teret egyre szűkülő alterek uniójára, azáltal, hogy újabb és újabb komponensét kötjük meg a leendő megoldásnak. Kezdetben csak az 1. dimenzióban lesz kötött érték. Ekkor tehát a térnek azt a *K-1* dimenziós alterét szemeltük ki további vizsgálatra, amelynek 1. komponense éppen a rögzített érték. A vizsgálatra szoruló pontok halmaza még a teljes tér. Amikor kiderül a komponensek konkretizálása során, hogy a „koncentrált” altér nem tartalmazhatja a megoldást, akkor a teljes alteret elvetve vesszük a „szomszédos” (valamelyik soronkövetkező, azaz a megfelelő Y_i -beli későbbi elem által meghatározott) alteret.
  Ezek után az L a következő indexsorozatok *halmaz*a: az (1,1,...1), első elemek

[8] A szignatúráról:
1. paraméter – K db alaphalmazon nyugvó sorozatok direktszorzata (K db sorozat)
2. paraméter – egy logikai értékű függvény, amely az alaphalmazokból vett egy-egy elemből alkotott sorozat kezdőszeleteihez rendel Igaz/Hamis értéket.

[9] A szignatúráról:
1. paraméter – K db alaphalmazon nyugvó sorozatok direktszorzata (K db sorozat)
2. paraméter – egy logikai értékű függvény, amely egy indexsorozat kezdőszeleteihez rendel Igaz/Hamis értéket. Az i. index az i. alapsorozathoz tartozik, s azt fejezi ki, hogy abból éppen az i.-ket választottuk ki.

sorozatától az $(m_1,m_2,...m_K)$ utolsó sorozatig „tart”. (Az $m_i=\|Y_i\|$.) (Formális okok miatt egészítsük ki a fenti teret olyan elemekkel is, amelyek bármely koordinátája a 0 is lehet.) Ezt a halmazt tesszük pontok *sorozat*ává az ε és σ függvények segítségével. A gondolatmenetből nyilvánvaló, hogy a pontfelsorolás alapja az L-beli sorozatok *kezdőszelet*ei lesznek. Az aktuális L azonosítására mód van a „kiinduló” sorozat nem triviális (azaz rögzített) elemeket tartalmazó kezdőszelete által:

$L:=L(x_1,x_2,...x_i)$ (az x_i -k az *indexek*et jelentik).

Azaz minden L-re vonatkozó műveletben *i*-vel és X-szel operálunk.

Egyszerűség kedvéért elvárjuk, hogy

$$L(x_1,x_2,...x_i)=L(x_1,x_2,...x_i,0,...0).$$

Tehát $L(x_1,x_2,...x_i)$ az alábbi halmazt jelöli valójában:

$$L(x_1,x_2,...x_i) = \{(x_1,x_2,...x_i,z_{i+1},...z_K) \mid \forall j\in[i+1..K]: z_j\in[1..m_j]\} \cup_{(z=x_i+1..M_i)} L(x_1,x_2,...x_{i-1},z)$$

– pirossal emeltük ki a rögzített paramétereket.

Például az L-tér az

1. *feladatban: i. dimenziója az i. vezér sorkoordinátáinak halmaza*[10]
2. *feladatban: i. dimenziója az i.-ként kiválasztható címletek halmaza, a tér annyi „dimenziós”, ahány címlet szükségeltetik maximálisan a címletezéshez*
3. *feladatban: az i. (és az összes) dimenziója a lépésirányok halmaza, a „dimenziószám” az út hossza.*

Vegyük észre az L alábbi tulajdonságait!

1. Az összes lehetőség L halmaza: L(1,1,...,1).
2. Monoton csökkenés:

a. $L(x_1,...,x_i)=L(x_1,...x_{i+1}) \quad \forall x_{i+1}\in[1..m_{i+1}]$,
azaz „előrehaladáskor” nem csökken a vizsgálatban szereplők száma;

b. $L(x_1,...,x_i)\supset L(x_1,...x_{i-2},z_{i-1}) \quad \forall z_{i-1}\in(x_{i-1}..m_{i-1}]$.
azaz „visszalépéskor” éppen annyival csökken, ahány „dimenziós” alteret sikerült kizárni a további vizsgálatból, pontosabban éppen az
$\{(x_1,...,x_{i-1},\mathbf{z_i,...,z_K}) \mid \forall j\in[i..K]\ \mathbf{z_j}\in[1..m_j]\}$
alteret, ami elemszáma $\prod_{j=i}^{K} m_j$;

c. $L(x_1,...,x_i)\supset L(x_1,...x_{i-1},z_i) \quad \forall\ z_i\in(x_i..m_i]$,
azaz többszörös visszalépés után az „átlépett” alterekbeli elemekkel csökkenthető a vizsgálat;

[10] Jobb lenne *sorozat*ot mondani a *halmaz* helyett, hiszen, mint hamarosan kiderül, ebből kell *valahányadik*at kiválasztani, ami egy halmazból lehetetlen.

d. a b. és a c.-ből következik, hogy a „legkisebb” $L(x_1,...,x_i)$ az L(), amely az $L(x_{m_1})$-ből való visszalépés után adódik. Ebben 0 darab elem van.

Az „L-mérték” nyomonkövetését végzi a $\|.\|$ *függvény*, így elkerülhetetlen ennek matematikai megragadása. Nyilvánvaló lehetőség lenne az előbbiekre építve a még ki nem szórt pontok számával mérni. Ennél lényegesen egyszerűbb mód is van. A d. figyelembe vételével az L üressé válását észlelhetjük az éppen a lerögzített elemek számának figyelésével: amikor az 0-ra csökken, akkor ürült ki a tér is.
Vagyis, ha i jelöli a kezdőszelet aktuális hosszát ($L=L(x_1,...,x_i)$), akkor

$$\|L\|>0 \Leftrightarrow i\geq 1$$

illetve ha a végrehajtás során a K.-et is kielégítettük, akkor találtuk meg a keresett elemét a térnek:

$$\mathbf{i>K}$$

- σ **(1)** $\sigma(L(x_1,...x_i)):=L(x_1,...x_i,x_{i+1})$, ha $i\leq K \wedge l(x_1,\ldots x_{i-1},x_i)$

  – *előrelépés*

  (2) $\sigma(L(x_1,...x_i)):=L(x_1,...z_j,0)$, ha $i\geq 1 \wedge$
  $\neg l(x_1,\ldots x_{i-1},x_i) \wedge \exists^* j\in[1..i): \exists_* z_j\in(x_j..m_j]: l(x_1,\ldots z_j)$

  – *visszalépés*

  ($\exists_*$,$\exists^*$jelentése:

  $\exists_* p\in[a..f]$: létezik és ha nem egyértelmű, akkor közülük az első, azaz az *a*-hoz legközelebbi)…

  $\exists^* p\in[a..f]$: létezik és ha nem egyértelmű, akkor közülük az utolsó, azaz az *f*-hez legközelebbi)

  (3) $\sigma(L(x_1,...x_i)):=(0,0,\ldots 0)$, egyéb esetben

  – *leállás, kielégíthetetlen esetben*.

- ε $\varepsilon\big(L(x_1,...,x_i)\big):=(x_1,...,x_i,0,...,0)$

Alg:

```
Konstans MaxN:Egész(???)
Típus TMk=Tömb(1..MaxN:Egész)
      TXk=Tömb(1..MaxN:Egész)
      TKeresett=Rekord(van:Logikai, melyik:Egész)
Eljárás BacktrackKeresés(Konstans K:Egész
                         Változó  Van:Logikai, X:TXk):
  Változó
    i:Egész
    ker:TKeresett
```

```
  i:=1; X(1..K):=0 [az i. kezdőszeletnél tartunk]
  Ciklus amíg i≥1 és i≤K
    ker:=JóElem(i)
    Elágazás
      i≤K és ker.van esetén X(i):=ker.melyik; i:+1
          i≥1        esetén X(i):=0; i:-1
    Elágazás vége
  Ciklus vége
  Van:=i≥1
  [X-ben található a megoldás, ha létezik]
Eljárás vége.
```

A ciklusmagban a σ-transzformáció nem „eredeti” megfelelője található, hanem annak egy módosított (egyszerűsített, de ekvivalens) változata. (A **(2)**-ág bonyolult feltétele egy keresés tétellel lenne megvalósítható. Ennek ciklusa „egyesíthető” a befoglaló külső ciklussal. S így nyertük a fenti megoldást.)

Mivel a ciklusban csak i≥1 és i≤K esetben vagyunk, s csak ekkor történhet az i növelése, vagy csökkentése, ezért a kétirányú elágazás helyett írható **Ha**-típusú elágazás is, egyszerűbb feltétellel. Így kapjuk (csak a ciklusbelüli részt ismételjük meg):

```
  Ciklus amíg i≥1 és i≤K
    ker:=JóElem(i)   [az i.-ig lehetséges-e a kiválasztás?]
    Ha ker.van akkor  X(i):=ker.melyik; i:+1
               különben X(i):=0; i:-1
  Ciklus vége
```

A JóElem függvény finomítása:

```
Függvény JóElem(i:Egész): TKeresett
  [i.-ben választható-e jó komponens?
   M(1..K) tartalmazza az egyes bemeneti sorozatok elemszámát]
  Változó
    j:Egész
  j:=X(i)+1
  Ciklus amíg j≤M(i) és nem Lehetséges(i,j)
    j:+1
  Ciklus vége
  JóElem.van:=j≤M(i)
  Ha JóElem.van akkor JóElem.melyik:=j
Függvény vége.
Függvény Lehetséges(Konstans i,j:Egész):Logikai
  [az i. komponensként a j választása összefér-e az eddigiekkel?]
  Lehetséges:=l(Y(1)(X(1)),...,Y(i-1)(X(i-1)),Y(i)(j))
Függvény vége.
```

Érdemes meggondolni az egyes konkrét feladatmegoldásoknál, mekkora a hatékonyságnövekedés a „naiv” megoldáshoz képest.

3.3. A backtrack gondolat további tételekben

Ha nem keresnünk kell backtrack alapon, akkor sincs nagy vész, mivel mechanikus átírási sablonok segítenek nekünk. Mindössze annyi az észreveendő, hogy

Absztrakt keresés algoritmusának fogalma	*Backtrack algoritmus kelléke*
$\|L\|>0$	`i≥1`
$T(\varepsilon(L))$	`i≤K`
$L:=\sigma(L)$	`ker:=JóElem(i)`
	Ha `ker.van` **akkor**
	`  X(i):=ker.melyik; i:+1`
	Különben
	`  X(i):=0; i:-1`
	Elágazás vége

Nincs más teendőnk, mint az algoritmus legfelsőbb szintjét minimálisan újraalkotni. Arra persze oda kell figyelni, hogy ugyanaz a változó az alapalgorimusban esetleg nem ugyanazzal a jelentéssel bír, mint a backtrack-es változatban.

A kiválasztás:

Alapalgoritmus	*Backtrack-változat*
`i:=1`	`i:=1; X(1..K):=0`
Ciklus amíg nem `T(X(i))`	**Ciklus amíg** `i≤K`
`  i:+1`	`  ker:=JóElem(i)`
Ciklus vége	`  Ha ker.van akkor`
`Sorsz:=i`	`  X(i):=ker.melyik; i:+1`
	`  Különben`
	`  X(i):=0; i:-1`
	`  Elágazás vége`
	Ciklus vége
	`[a keresett elem az X-ben]`

A megszámlálás:

Alapalgoritmus	*Backtrack-változat*[11]
`Db:=0`	`Db:=0`
Ciklus `i=1-től N-ig`	`i:=1; X(1..K):=0`
`  Ha T(X(i)) akkor Db:+1`	**Ciklus amíg** `i≥1`
Ciklus vége	`  Ha i>K akkor`
	`  Db:+1;`
	`  i:-1 [visszalépés]`
	`  Elágazás vége`
	`  ker:=JóElem(i)`
	`  Ha ker.van akkor`
	`  X(i):=ker.melyik; i:+1`
	`  Különben`
	`  X(i):=0; i:-1`
	`  Elágazás vége`
	Ciklus vége

A maximum-kiválasztás:

Itt feltételezzük, hogy $\exists \leq, <$ rendezés az L téren, ami X megoldásokat összevethetővé teszi. Az egyszerűség kedvéért föltesszük, hogy az L legkisebb eleme az (0,...,0), azaz $\forall(x_1,x_1,\ldots x_K)\in L: (0,...,0)<(x_1,x_1,\ldots x_K)$.

[11] Először a számlálásos ciklust amíg-ossá alakítottuk, s csak azután álltunk a mechanikus átíráshoz.

Alapalgoritmus

```
max:=X(1)
Ciklus i=2-től N-ig
  Ha X(i)>max akkor max:=X(i)
Ciklus vége
```

Backtrack-változat

```
i:=1; X(1..K):=0
max:=X
Ciklus amíg i≥1
  Ha i>K akkor
    Ha X>max akkor
      max:=X
      i:-1 [visszalépés]
    Elágazás vége
  Elágazás vége
  ker:=JóElem(i)
  Ha ker.van akkor
      X(i):=ker.melyik; i:+1
  Különben
      X(i):=0; i:-1
  Elágazás vége
Ciklus vége
[max-ban a „leg”]
```

TARTALOM