Algoritmizálás

Horváth Gyula
Szegedi Tudományegyetem
Természettudományi és Informatikai Kar
horvath@inf.u-szeged.hu

0.1. Az algoritmikus tudás szintjei

- Ismeri (a megoldó algoritmust)
- Érti
- Le tudja pontosan írni (mondatszerű, pszeudókód nyelven)
- Tudja alkalmazni
- Meg tudja valósítani konkrét környezetben (Pascal nyelven)
- Önállóan képes algoritmust készíteni: tervezni, elemezni, megvalósítani

0.2. A problémamegoldás fázisai

1. A probléma megértése.
2. A modell - absztrakt feladat - megfogalmazása.
3. Összefüggések kiderítése és megfogalmazása.
4. Algoritmustervezés.
5. Helyesség igazolása.
6. Hatékonysági elemzés (futási idő és tárigény).
7. Kódolás - az algoritmus leírása a választott programozási nyelven (Pascal, C).
8. Tesztelés.
9. Véglegesítés.

0.3. Az adatkezelés szintjei:

1. Probléma szintje.
2. Modell szintje.
3. Absztrakt adattípus szintje.
4. Absztrakt adatszerkezet szintje.
5. Adatszerkezet szintje.
6. Gépi szint.

Absztrakt adattípus: $A = (E, M)$

1. E: értékhalmaz,
2. M: műveletek halmaza.

"Absztrakt" jelző jelentése:

i. Nem ismert az adatokat tároló adatszerkezet.

ii. Nem ismertek a műveleteket megvalósító algoritmusok, a műveletek specifikációjukkal definiáltak.

0.4. Prioritási Sor

Értékhalmaz: $PriSor = \{ S : S \subseteq E \}$, E-n értelmezett a $\leq$ lineáris rendezési reláció.
Műveletek:

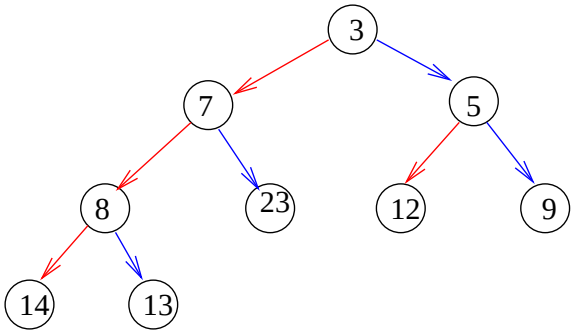
$$S : PriSor, x : E$$

$$\begin{array}{rcl} \{S = S\} & SorBa(S,x) & \{S = Pre(S) \cup \{x\}\} \\ \{S \neq \emptyset\} & SorBol(S,x) & \{x = \min(Pre(S)) \wedge Pre(S) = S \cup \{x\}\} \\ \{S = \{a_1, \ldots, a_n\}\} & Elemszam(S) & \{Elemszam = n\} \\ \{S \neq \emptyset\} & Elso(S,x) & \{x = \min(Pre(S)) \wedge Pre(S) = S\} \end{array}$$

0.5. Absztrakt adatszerkezetek

Olyan $\mathtt{A} = (M, R, Adat)$ rendezett hármas, ahol
1. M az absztrakt memóriahelyek, *cellák* halmaza.
2. $R = \{r_1, \ldots, r_k\}$ a cellák közötti *szerkezeti kapcsolatok*, $r_i : M \to (M \cup \{\perp\})^*$
3. $Adat : M \to E$ parciális függvény, a cellák adattartalma.
$x \in M$, $r \in R$ és $r(x) = \langle y_1, \ldots, y_i, \ldots, y_k \rangle$ esetén az x cella r kapcsolat szerinti szomszédai $\{y_1, \ldots, y_k\}$, y_i pedig az x cella i-edik szomszédja.
Ha $y_i = \perp$, akkor azt mondjuk, hogy x-nek hiányzik az r szerinti i-edik szomszédja.
Példa absztrakt adatszerkezetre: (Minimumos)Kupac. $\mathtt{A} = (M, R, Adat)$
1. M az absztrakt memóriahelyek, *cellák* halmaza.
2. $R = \{bal, jobb\}$ a zerkezeti kapcsolatok.

- A $\{bal, jobb\}$ kapcsolatok bináris fát határoznak meg.
- $(\forall x \in M)(Adat(x) \leq Adat(bal(x)) \wedge Adat(x) \leq Adat(jobb(x)))$

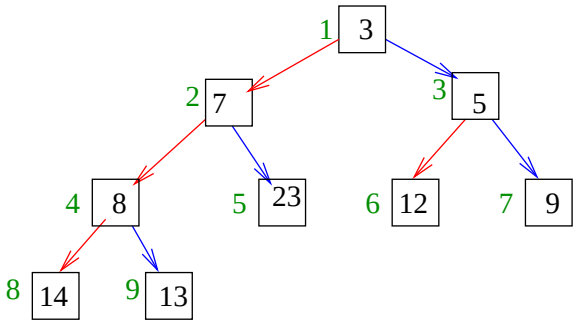


1. ábra. Kupac absztrakt adatszerkezet szemléltetése

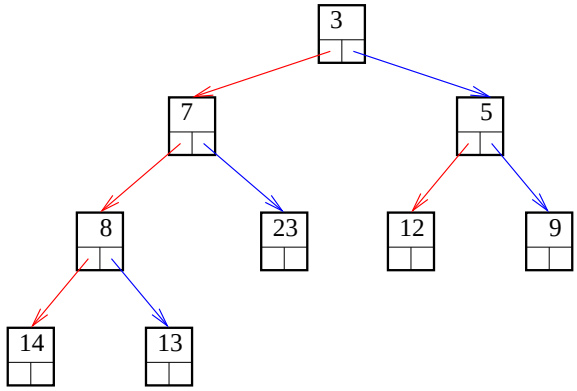
0.6. Adatszerkezetek

Egy $\mathtt{A} = (M, R, Adat)$ absztrakt adatszerkezet megvalósítása:
1. Konkrét memória allokálás az M-beli absztrakt memória cellák számára.
2. Az R szerkezeti kapcsolatok ábrázolása.
3. Alapműveletek algoritmusainak megadása.
Pl. a kupac megvalósítása konkrét adatszerkezettel:

1. A memóriahelyek tömbelemek. 2. A $\{bal, jobb\}$ kapcsolatokat számítási eljárással adjuk meg:
$bal(x) = 2*x$, $jobb(x) = 2*x+1$.
Teljesül a kupac tulajdonság:
$(\forall x)(Adat(x) \leq Adat(bal(x)) \wedge Adat(x) \leq Adat(jobb(x)))$



2. ábra. Kupac ábrázolása tömbbel. A szerkezeti kapcsolatot nem tároljuk; számítjuk.



3. ábra. Kupac ábrázolás dinamikus memóriacellákkal; a szerkezeti kapcsolatot táruljuk.

0.7. Alapfeladatok

0.7.1. Összegzés

Bemenet:

$$a_1, a_2, \ldots, a_n$$

Kimenet:

$$s = \sum_{i=1}^{n} a_i$$

Megvalósítás:

```
1  const
2    maxN=1000;
3  type
4    Tomb=array[1..maxN] of integer;
5  function Osszegez(var a:Tomb; n:integer):integer;
6  var
7    s,i:integer;
8  begin
```

```
9     s:=0;
10    for i:=1 to n do
11      s:=s+a[i];
12    Osszegez:=s;
13  end{Osszegez};
```

0.7.2. Számlálás

Bemenet:

$$a_1, a_2, \ldots, a_n$$

T tulajdonság

Kimenet: a sorozat azon elemeinek száma, amelyekre teljesül a T tulajdonság

$$s = \sum_{i=1}^{n} 1 : T(a_i)$$

Megvalósítás:

```
1   const
2     maxN=1000;
3   type
4     Tomb=array[1..maxN] of integer;
5   function T(x:integer):boolean;
6   begin end;
7
8   function Megszamlal(var a:Tomb; n:integer):integer;
9   {A T tulajdonság globális}
10  var
11    s,i:integer;
12  begin
13    s:=0;
14    for i:=1 to n do
15      if T(a) then
16        inc(s);
17    Megszamlal:=s;
18  end{Megszamlal};
```

0.7.3. Maximum kiválasztás

Bemenet:

$$a_1, a_2, \ldots, a_n$$

Kimenet:

$$b = \max_{i=1}^{n} a_i$$

Megvalósítás:

```
1   const
2     maxN=1000;
3   type
4     Tomb=array[1..maxN] of integer;
5   function Maximum(var a:Tomb; n:integer):integer;
6   var
7     maxi,i:integer;
8   begin
9     maxi:=a[1];
```

```
10    for i:=2 to n do
11      if a[i]>maxi then
12        maxi:=a[i];
13    Maximum:=maxi;
14  end{Maximum};
```

0.7.4. Lineáris keresés (Kiválasztás)

Bemenet:

$$a_1, a_1, \ldots, a_n$$

$$F \text{ feltétel}$$

Kimenet: a legkisebb olyan i index, hogy $F(a_i)$ teljesül. Ha nincs ilyen, akkor 0.
Megvalósítás:

```
1   const
2     maxN=1000;
3   type
4     Tomb=array[1..maxN] of integer;
5     Feltetel=function (x:integer):boolean;
6   function Kivalaszt(var a:Tomb; n:integer; F:Feltetel):integer;
7   var
8     i:integer;
9   begin
10    i:=1;
11    while (i<=n) and (not F(a[i])) do
12      inc(i);
13    if i<=n then
14      Kivalaszt:=i
15    else
16      Kivalaszt:=0;
17  end{Kivalaszt};
```

0.7.5. Rendezés: kiválasztó-rendezés

Bemenet:

$$a_1, a_2, \ldots, a_n$$

Kimenet: a bemenet elemeinek növekvő (nemcsökkenő) felsorolása :

$$b_1 \leq b_2 \leq \cdots \leq b_n$$

Elv: ismételten válasszuk ki a rendezetlen halmazból a rendezésben soron következő elemet (a legkisebbet), és rakjuk a már rendezett részsorozat végére.
Megvalósítás:

```
1   const
2     maxN=1000;
3   type
4     Tomb=array[1..maxN] of integer;
5   procedure KivalasztoRend(var a:Tomb; n:integer);
6   var
7     i,j,mi,x:integer;
8   begin
9     for i:=1 to n-1 do begin
10    {a[1..i-1] a bemenet i-1 legkisebb eleme rendezetten}
11      mi:=i;
12      for j:=i+1 to n do{a[i..n] minimumának megkeresése}
```

```
13        if a[j]<a[mi] then
14          mi:=j;
15      x:=a[i]; {a[i] és a[mi] cseréje}
16      a[i]:=a[mi];
17      a[mi]:=x;
18    end{for i}
19  end{KiválasztoRend};
```

0.7.6. Rendezés: beszúró-rendezés

Bemenet:

$$a_1, a_2, \ldots, a_n$$

Kimenet: a bemenet elemeinek növekvő (nemcsökkenő) felsorolása :

$$b_1 \leq b_2 \leq \cdots \leq b_n$$

```
1   const
2     maxN=1000;
3   type
4     Tomb=array[1..maxN] of integer;
5   procedure BeszuroRend(var a:Tomb; n:integer);
6   var
7     i,j,x:integer;
8   begin
9     for i:=2 to n do begin{a[i] beszúrása az a[1..i-1] rendezett sorozatba}
10      x:=a[i];
11      j:=i-1;
12      while (j>0)and(x<a[j]) do begin
13        a[j+1]:=a[j];
14        dec(j);
15      end{while};
16    end{for i}
17  end{BeszuroRend};
```

0.7.7. Két rendezett sorozat összefésülése

Bemenet:

$$a_1 \leq a_2 \leq \ldots \leq a_m$$

$$b_1 \leq b_2 \leq \cdots \leq b_n$$

Kimenet: a bemenet elemeinek növekvő (nemcsökkenő) felsorolása :

$$c_1 \leq c_2 \leq \cdots \leq c_{m+n}$$

```
1   const
2     maxN=1000;
3   var
4     m,n:integer;
5     a,b:array[1..maxN] of integer;
6     c:array[1..maxN+maxN] of integer;
7     i,j,k:integer;
8   begin
9     readln(m);
10    for i:=1 to m do
11      read(a[i]);
12    readln;
13    readln(n);
```

```
14    for i:=1 to n do
15      read(b[i]);
16  //összefésülés
17    i:=1; j:=1; k:=0;
18    while (i<=m) and (j<=n) do begin
19      inc(k);
20      if a[i]<b[j] then begin
21        c[k]:=a[i];
22        inc(i);
23      end else begin
24        c[k]:=b[j];
25        inc(j);
26      end;
27    end{while};
28    for i:=i to m do begin{az a sorozat maradékának másolás}
29      inc(k);
30      c[k]:=a[i];
31    end;
32    for j:=j to n do begin{a b sorozat maradékának másolás}
33      inc(k);
34      c[k]:=b[j];
35    end;
36  //kiíratás
37    for i:=1 to m+n do
38      write(c[i],' ');
39    writeln;
40  end.
```

0.7.8. Keresés rendezett sorozatban

Adott egész számoknak egy $a = \langle a_0 \leq a_1 \leq \cdots \leq a_{n-1} \rangle$ rendezett sorozata és egy x egész szám.
Keressük meg azt az i indexet, amelyre $a[i] = x$, ha nincs ilyen, akkor a -1 értéket adja az algoritmus!

```
1   const    maxN=1000;
2   type
3     tomb=array[1..maxN] of integer;
4   function BinKeres(a:tomb; n, x:integer):integer;
5   //bináris keresés rendezett sorozatban
6   var
7     bal,jobb,k:integer;
8   begin
9     bal:=1; jobb:=n;
10    while (bal<jobb)do begin
11      k:=(bal+jobb) div 2;
12      if (a[k]<x) then
13        bal:=k+1
14      else
15        jobb:=k;
16    end;
17    if (a[bal]=x) then
18      BinKeres:=bal
19    else
20      BinKeres:=-1;
21  end{BinKeres};
```