Adattípus, Konténer, Iterátor, Felsoroló, Generikus programozás

Horváth Gyula
horvath@inf.elte.hu

1. Adattípusok

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.

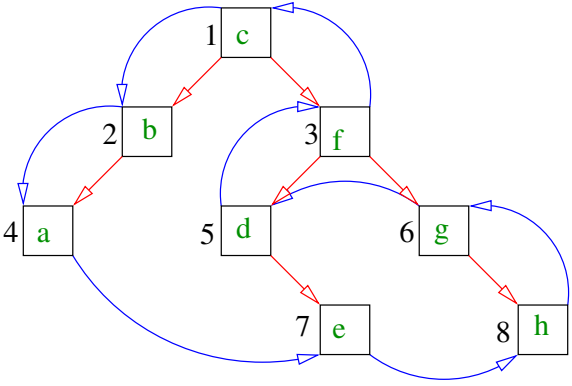
ii. Nem ismertek a műveleteket megvalósító algoritmusok, a műveletek specifikációjukkal definiáltak.

1.1. Általános absztrakt adatszerkezet

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 szomszédjai $\{y_1, \ldots, y_k\}$, 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.
”Absztrakt” jelző jelentése:

i. Nem ismert, hogy a celláknak hogyan foglalunk memóriát.

ii. Nem ismert, hogy a szerkezeti kapcsolatokat hogyan valósítjuk meg.



1. ábra. Adatszerkezet piros és kék szerkezeti kapcsolattal

$M = \{1,2,3,4,5,6,7,8\}$
kék : $M \to M$
piros : $M \to (M \cup \{\bot\}) \times (M \cup \{\bot\})$

$$\text{kék} : 1 \mapsto 2, 2 \mapsto 4, 3 \mapsto 1, 4 \mapsto 7, 5 \mapsto 3, 6 \mapsto 2, 7 \mapsto 8, 8 \mapsto 6,$$

$$\text{piros} : 1 \mapsto (2,3), 2 \mapsto (4,\bot), 3 \mapsto (5,6), 4 \mapsto (\bot,\bot), 5 \mapsto (\bot,7), 6 \mapsto (\bot,8), 7 \mapsto (\bot,\bot), 8 \mapsto (\bot,\bot)$$

$$Adat(1) =' c', Adat(2) =' b', Adat(3) =' f', Adat(4) =' a', Adat(5) =' d', Adat(6) =' g', Adat(7) =' e',$$

1.2. Konténer

Értékhalmaz: $Kontener = \{a_1, \ldots, a_n\} : a_i \in E, i = 1, \ldots, n, n \geq 0\}$
Műveletek argumentumai: $K : Kontener, x : E$

Előfeltétel	Művelet	Utófeltétel
$K = K$	HozzáAd(K, x)	$K = Pre(K) \cup \{x\}$
$K = K$	ÜresE(V)	$= (Pre(K) = \{\}) \wedge K = Pre(K)$
$K = \{a_1, \ldots, a_n\}$	Elemszám(K)	$= n \wedge K = Pre(K)$
$K = K$	$for\,(E\,x : K)\; M(x)$	$K = Pre(K)$

A

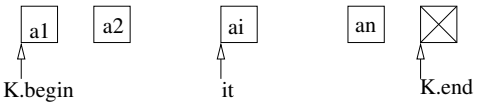
$$for(E\,x : K)\; M(x)$$

kiterjesztett for-ciklus hatása: A K konténer minden adatelemére pontosan egyszer végrehajtja az M műveletet.
A végrehajtás sorrendje nem meghatározott, kivéve, ha a konténer rendezetten tartalmazza az adatelemeket, ekkor a rendezés szerint sorrendben hajtja végre az adatelemeken az M műveletet.
Konténer adatelemei más módszerrel is felsorolhatók: **iterátor** vagy **felsoroló**.

1.3. Konténer elemeinek felsorolása Iterátorral

Az iterátor fogalmán mást és mást értenek a különböző programozási nyelvekben (pl. C++, java, C#). Mi most a C++ iterátor fogalmát ismertetjük, ami jelentősen eltér más nyelvek iterátor fogalmától. A C++ iterátor (továbbiakban iterátor) a kurzor fogalmán alapszik. A kurzor egy referencia (hivatkozás, pointer), ami egy összetett adat valamely adatelemére való hivatkozás. Ha az összetett adat az $a_1, \ldots, a_n$ adatelemeket tartalmazza (ezekből épül fel), akkor létezik az adatelemeknek egy logikai (az adatelemek tárolásától független) sorrendje. Ezt szemlélteti az alábbi ábra.

Minden K konténer esetén $K.begin$ a felsorolás első elemét tartalmazó cellára mutat (ha üres a



2. ábra. Iterátor konténer elemeire

konténer, akkor $K.begin$ értéke=$K.end$, $K.end$ pedig a felsorolás után álló fiktív cellára mutat. Egy it itarátor által mutatott cella tartalmára a C++-ban szokásos $*it$ dereferenciával hivatkozhatunk.
Az iterátorral való továbblépés a $++$ művelettel valósítható meg. Tehát az alábbi programséma a K konténer elemein végighaladva végrehajtja az M műveletet (mindegyikre pontosan egyszer).

1.3.1. C++ megvalósítás

```
1 it=K.begin();
2 while(it != K.end() ) {
3       x=*it;   //
4       M(x);    //
5       it++;    //
6 }
```

1.4. Konténer elemeinek felsorolása Felsorolóval

A C++-tól különböző programozási nyelvek nem (C++-értelemben vett) iterátor alkalmazásával oldják meg konténer adatelemeinek sorravételét, hanem felsorolót biztosítanak. A felsoroló olyan három művelet együttese, amelyek a sorravételt biztosítják. C# esetén ezen három művelet a *GetEnumerator* *MoveNext* és a *Current*. Ezekkel a műveletekkel az alábbi programséma szerint lehet egy M műveletet végrehajtani a K konténer elemein.

```
1 e=K.GetEnumerator();
2 while (e.MoveNext()){
3       x=e.Current;
4       M(x);
5 }
```

Ellentétben az iterátorral, a felsorolás az alábbi ábrával szemléltethető. Tehát úgy tekinthetjük, hogy az elején van egy fiktív cella és a kezdetben a felsoroló erre a fiktív cellára mutat. Az *MoveNext* művelet igaz értéket ad, ha van még következő elem, és a kurzort arra állítja át, egyébként hamis a visszaadott érték, és a kurzor nem mozdul.

3. ábra. Felsoroló

1.5. Konténer elemeinek felsorolása kiterjesztett for ciklussal

1.5.1. C++ megvalósítás

```
1 for (auto x : K)
2     M(x);
```

1.5.2. C# megvalósítás

```
1 foreach (T x in K)
2     M(x);
```

```
1 foreach (var x in K)
2     M(x);
```

A sémában T a konténer elemeinek a típusa kell legyen.

1.6. Generikus programozás fogalma

A generikus programozás célja, hogy olyan algoritmusokat általános formában adjunk meg. Például, ne kelljen egy algoritmust a bemenő adatok minden egyes konkrét típusára külön-külön leírni, ha az minden típusra ugyanazt a tevékenységet végzi. Ilyenkor az elemek típusát is paraméternek tekintjük, és ha egy adott konkrét típusra akarjuk használni az algoritmust, akkor előbb ”behelyettesítjük” a típus paraméter helyére a konkrét típust.
A különböző pr. nyelvek különböző mechanizmust biztosítanak a generikusság megvalósítására. Az is eltérő, hogy milyen programelemek generikusságát biztosítják. Először a C++ generikus mechanizmusát vesszük.
A C++ a template (sablon) alkalmazásával teszi lehetővé generikus program elemek alkalmazását. Ezek az programelemek: osztály és függvény. Először lássunk a generikus függvények használatát programozási tételek általános megadására.
Az általánosítás a következő jelenti:

- Mindegy hogy milyen adattároló tárolja az adathalmaz elemeit, csak az a kikötés, hogy konténer legyen
- Mindegy, hogy mi az elemek típusa
- A tulajdonság tetszőleges logikai értékű függvény lehet

A C# nyelvben függvény és típus lehet generikus. Generikus függvény esetén a generikus (típus) paramétert az függvény azonosítója után ¡¿ zárójelek között kell megadni. Például a

```
static T osszegzes<T>(T[] A) {...}
```

függvénynek T típusparamétere, ami azt jelenti, hogy aktuális paramétere tetszőleges elemtípusú tömb lehet. A generikus típust osztály (class) definícióval lehet megadni. Az általános formája a következő:

```
public class Tipusnev<T1,...,Tk>{
  //lokális típusdefiníciók
  //adattagok
  //metódusok
}
```

A definícióban $T1,\ldots,Tk$ a típus paraméterek, amelyek az osztály tagok definíciójában mint típus azonosítok szerepelhetnek.

1.6.1. Generikus típus használata

Konkrét típus úgy származtatható a $Tipusnev$ generikus típusból, hogy a típus $T1,\ldots,Tk$ típus paraméterek mindegyikébe konkrét típust helyettesítünk az alábbi formában:

```
  Tipusnev<E1,...,Ek>;
```

Ahol $E1,\ldots,Ek$ konkrét típusok. A specializálás (típus behelyettesítés) közvetlenül változó deklarációjában is lehet: $Tipusnev < E1,...,Ek > V;$

1.7. A Megszámolás programozási tétel C++ generikus megvalósítása

1.7.1. A Megszámolás hagyományos megvalósítása

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int Megszamol(int A[], int n, bool f(int) ){
5    int s=0;
6    for(int i=0;i<n;i++)
7      if(f(A[i])) s++;;
8        return s;
9    }
10
11   bool paros(int x){
12     return x%2==0;
13   }
14 int main(){
15     int A[] = {1,2,3,6,8};
16     cout<<Megszamol(A, 5, paros)<<endl;
17     cout<<Megszamol(A, 5, [](int x) -> bool{return x%2==1;});
18     return 0;
19 }
```

1.7.2. A Megszámolás megvalósítása tömb-típusparaméterrel

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  template<typename E>
4  int Megszamol(E A[], int n, bool f(E) ){
5    int s=0;
6    for(int i=0;i<n;i++)
7      if(f(A[i])) s++;;
8        return s;
9    }
10   bool paros(int x){
11     return x%2==0;
12   }
13   bool Sparos(string x){
14     return x.size()%2==0;
15   }
16 int main(){
17   int A[]={1,2,3,6,8};
18   string S[]={"alfa","bet","gamma","delt"};
19
20   cout<<Megszamol(A, 5, paros)<<endl;
21   cout<<Megszamol(S, 4, Sparos)<<endl;
22   return 0;
23 }
```

1.7.3. A Megszámolás megvalósítása iterátor típusparaméterrel

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  template<class InputIt, typename E>
5  int Megszamol(InputIt e,InputIt u,bool T(E)){
6    int s=0;
7    while(e!=u){
8      if(T(*e)) s++;
9      e++;
10   };
11   return s;
12 }
13 bool paros(int x){return x%2==0;}
14 bool sparos(string x){return x.size()%2==0;}
```

```
15 int main(){
16    int A[]={6,3,3,5,7,8,8,9,10}; //mezei tömb konténer
17    array<int, 9>AA={6,3,3,5,7,8,8,9,10};
18    vector<int> V;
19    for(int x:A) V.push_back(x);
20    string S[]={"a","ab","abc","xxx"};
21    int n=9;
22    int hany=Megszamol(&A[0], &A[9], paros);
23    cout<<"A:"<<hany<<endl;
24
25    hany=Megszamol(&S[0], &S[4], sparos);
26    cout<<"S:"<<hany<<endl;
27
28    hany=Megszamol(AA.begin(), AA.end(), paros);
29    cout<<"AA:"<<hany<<endl;
30
31    hany=Megszamol(&V[0], &V[9], paros);
32    cout<<"V:"<<hany<<endl;
33 }
```

1.7.4. A Megszámolás megvalósítása konténer típusparaméterrel

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  template<class KontenerT>
5  int Megszamol(const KontenerT& K, bool T(typename KontenerT::value_type) ){
6     int s=0;
7     for (auto x : K)
8        if(T(x))
9           s++;
10    return s;
11 }
12 bool paros(int x){return x%2==0;}
13 bool Fparos(pair<const int,int> x){return x.first%2==0;}
14
15 int maxi(int x, int y){return x>y ? x:y;}
16 pair<int,int> Mmaxi(pair<int,int> x, pair<int,int> y){return x.first>y.firs
```

```
17 int main(){
18     int A[]={6,3,3,5,7,8,8,9,10};              //mezei tömb
19     array<int, 9> AA={6,3,3,5,7,8,8,9,10}; //tömb
20     vector<int> V;                             //vektor
21     set<int> H;                                //halmaz
22     multiset<int> MH;                          //multihalmaz
23     map<int,int> F;                            //(parciális)függvény
24     multimap<int, int> R;                      //(rendezett)reláció
25     unordered_set<int> UH;                     //nem-rendezett halmaz
26     unordered_multiset<int> UMH;               //nem-rendezett multihalmaz
27     unordered_map<int,int> UF;                 //nem-rendezett függvény
28     unordered_multimap<int,int> UMF;           //nem rendezett reláció
29     list<int> L;                               //lista
30     forward_list<int> FI={1,2,3};              //egyirányú lista
31     deque<int> DQ;                             //kétvégü sor
32     int hany;
33     for(int x:A){
34         V.push_back(x);
35         H.insert(x); MH.insert(x);
36         UH.insert(x); UMH.insert(x);
37         R.insert(pair<int,int>(x,2*x));
38         UMF.insert(pair<int,int>(x,2*x));
39         L.push_back(x);
40         DQ.push_back(x);
41     }
```

```
42     hany=Megszamol(AA,paros); cout<<"AA:"<<hany<<endl;      //tömb
43     hany=Megszamol(V,paros); cout<<"V:␣"<<hany<<endl;       //vektor
44     hany=Megszamol(H,paros); cout<<"H:␣"<<hany<<endl;       //halmaz
45     hany=Megszamol(MH,paros); cout<<"MH:"<<hany<<endl;      //multihalmaz
46     hany=Megszamol(UH,paros); cout<<"UH:␣"<<hany<<endl;     //nem-rendezett h
47     hany=Megszamol(UMH,paros); cout<<"UMH:␣"<<hany<<endl;   //nem-rendezett m
48     hany=Megszamol(F,Fparos); cout<<"F:␣"<<hany<<endl;      //függvény
49     hany=Megszamol(R,Fparos); cout<<"R:␣"<<hany<<endl;      //reláció
50     hany=Megszamol(UF,Fparos); cout<<"UF:␣"<<hany<<endl;    //nem-rendezett f
51     hany=Megszamol(UMF,Fparos); cout<<"FF:␣"<<hany<<endl;   //nem-rendezett r
52     hany=Megszamol(L,paros); cout<<"L:␣"<<hany<<endl;       //lista
53     hany=Megszamol(Fl,paros); cout<<"Fl:␣"<<hany<<endl;     //egyirányú lista
54     hany=Megszamol(DQ,paros); cout<<"DQ:"<<hany<<endl;      //kétvégü sor
55
56 return 0;
57 }
```

1.8. A Megszámolás programozási tétel C# generikus megvalósítása

1.8.1. A Megszámolás hagyományos megvalósítása

```
1  using System;
2  using System.Collections.Generic;
3
4  class Program{
5     static int Megszamol(int [] A, Func<int,bool> f ){
6        int s=0;
7        foreach(int x in A)
8           if(f(x)) s++;;
9        return s;
10    }
11    static bool paros(int x){
12       return x%2==0;
13     }
14    static void Main(){
15       int[] A=new int[] {1,2,3,6,8};
16       Console.WriteLine(Megszamol(A, paros));
17       Console.WriteLine(Megszamol(A,x=>x %2!=0));
18    }
19 }
```

1.8.2. A Megszámolás megvalósítása tömb-típusparaméterrel

```
1  using System;
2  using System.Collections.Generic;
3
4  class Program{
5     static int Megszamol<T>(T[] A, Func<T,bool> f ){
6        int s=0;
7        foreach(T x in A)
8           if(f(x)) s++;;
9        return s;
10    }
11    static void Main(){
12       int[] A=new int[] {1,2,3,6,8};
13       Console.WriteLine(Megszamol(A,x=>x %2==0));
14    }
15 }
```

1.8.3. A Megszámolás megvalósítása felsoroló típusparaméterrel

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4
5  class Program{
6     static int Megszamol<T>(IEnumerator<T> e, Func<T,bool> f ){
7        int s=0;
8        while(e.MoveNext()){
9           if( f(e.Current) ) s++;
10       }
11       return s;
12    }
13    static bool paros(int x){
14       return x%2==0;
15    }
16    static void Main(){
17       List<int> lista = new List<int>();
18       lista.Add(1); lista.Add(4); lista.Add(9);
19       Console.WriteLine(Megszamol(lista.GetEnumerator(), x=>x %2!=0));
20       Console.WriteLine(Megszamol(lista.GetEnumerator(), paros));
21    }
22 }
```

1.8.4. A Megszámolás megvalósítása konténer típusparaméterrel

```
1  using System;
2  using System.Collections.Generic;
3
4  class Program{
5     static int Megszamol<T>(ICollection<T> K, Func<T,bool> f ){
6        int s=0;
7        foreach(T x in K)
8           if(f(x)) s++;;
9        return s;
10    }
11    static bool paros(int x){
12       return x%2==0;
13    }
14    static void Main(){
15       int[] A=new int[] {1,2,3,6,8};
16       Console.WriteLine(Megszamol(A, paros));
17
18       List<int> lista = new List<int>();
19       lista.Add(1); lista.Add(4); lista.Add(9);
20       Console.WriteLine(Megszamol(lista , x=>x %2!=0));
21    }
22 }
```