2 skyrius

Duomenų struktūros

2.1. Tiesinis sąrašas

Dažnai sprendžiame uždavinius, kuriems būdingos tokios savybės:

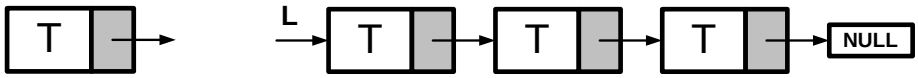
- maksimalus saugomų duomenų skaičius nėra iš anksto žinomas arba jis labai keičiasi uždavinio sprendimo metu,
- nauji elementai gali būti įterpiami ne tik į aibės pabaigą, bet ir į bet kurią jos vietą,
- dažnai tenka šalinti elementus, esančius įvairiose aibės vietose.

Tokio uždavinio pavyzdžiu yra bibliotekoje saugomų knygų sąrašas: biblioteka gauna vis naujas knygas, o skaitytojai pasirenka tiek naujai išleistas, tiek ir senesnes knygas, kai kurios knygos yra nurašomos ir biblioteka jų daugiau nesaugo. Norėdami lengvai rasti reikalingą knygą jas rūšiuojame pagal autoriaus pavardę. Tarkime, kad knygų sąrašą saugosime masyve. Tada šio masyvo ilgis turi būti pakankamai didelis, kad galėtume ilgai naudoti šį sąrašą. Tačiau pradžioje, kol bibliotekoje yra nedaug knygų, toks didelis masyvas yra nereikalingas, todėl rezervuota kompiuterio atmintis bus naudojama neefektyviai. Tarkime, kad knygų sąrašo ilgis yra 10000 knygų, o biblioteka gavo naują Jono Avyžiaus romaną. Jį reikia įrašyti į 214 sąrašo vietą, todėl teks perstumti 99786 įrašus, o tai pakankamai ilgai trunkanti operacija. Panaši situacija bus ir tada, kai teks pašalinti iš sąrašo pradžios vieną iš knygų. Tokiais atvejais naudojamos *dinaminės* duomenų struktūros.

2.1.1. Vienakryptis tiesinis sąrašas

Pirmiausia apibrėžiame vieną atskirą *elementą* (angl. *node*), kurį sudaro informacinė dalis T (ją patogu realizuoti kaip įrašą) ir rodyklė, rodanti į elemento tipo objektą (žr. 2.1 *a* paveikslą):

```
struct node{
  struct T;
  node* next;
}
```



a) b)

2.1 pav. a) Atskiras sąrašo elementas, b) vienakryptis tiesinis sąrašas.

Tada *tiesinis vienakryptis sąrašas* (angl. *linked list*) – tai pradžios rodyklė ir rodyklėmis susietų elementų seka. Paskutinio elemento rodyklė rodo į tuščią elementą, tai ir yra sąrašo pabaigos požymis (žr. 2.1 *b* paveikslą). Pavyzdžiui, C++ kalboje tuščia rodyklė yra žymima NULL.

Aptarsime svarbiausias tiesinio sąrašo savybes

- tiesinis sąrašas yra sukūriamas apibrėžus rodyklę, kuri rodo į sąrašo pabaigą, bet kuriuo momentu jam skiriame tik tiek atminties, kiek reikia sąrašo elementų saugojimui,
- kompiuterio atmintyje sąrašo elementai gali būti saugomi bet kurioje vietoje, o gretimi sąrašo elementai nebūtinai turi būti saugomi gretimose atminties ląstelėse,
- bet kuris sąrašo elementas yra pasiekiamas tik iš sąrašo pradžios, paeiliui perėjus visus prieš jį esančius elementus.

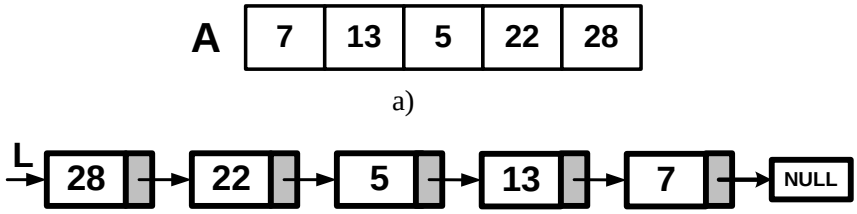
2.1.2. Pagrindiniai vienakrypčio sąrašo veiksmai

Kaip ir kievienos duomenų struktūros atveju reikia realizuoti procedūras (metodus), leidžiančias atlikti pagrindinius veiksmus: sudaryti naują sąrašą, atlikti elemento paiešką, įrašyti naują bei pašalinti jau egzistuojantį elementus.

Naujo sąrašo sudarymas. Sakykime, kad duomenys yra masyve A (dažniausiai duomenis skaitome iš failo) ir juos reikia perkelti į dinaminį sąrašą. Tada patogiausia naują elementą įrašyti į sąrašo pradžią. Kiekviename ciklo žingsnyje pirmiausia sukuriame naują elementą, į jį užrašome reikalingą informaciją, o po to šį elementą prijungiame prie sąrašo.

```
constructList(node* listStart){
  T A[20];
  node *n;
  listStart = NULL;
  for (i=0; i<20; i++){
    n = new node;  // Naujas elementas
    (*n).T = A[i];
    (*n).next = listStart;
    listStart = n;
  }
}
```

Šiuo algoritmu sukurtame sąraše elementai yra saugomi atvirkščia duomenų skaitymui tvarka. 2.2 paveiksle pavaizduotas vienakrypčio sąrašo formavimas, kai pradiniai duomenys buvo saugomi A masyve.



a)

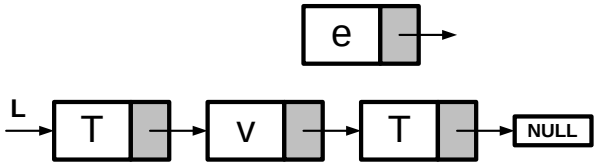
b)

2.2 pav. a) Pradinių duomenų masyvas, b) vienakryptis tiesinis sąrašas, į kurį perkelti masyvo duomenys.

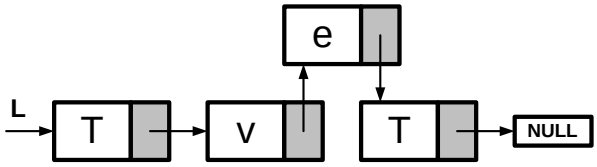
Elemento paieška. Tarkime reikia patikrinti ar tiesiniame sąraše L yra saugomas įrašas, kurio id laukas yra lygus t. Tikrinti pradedame nuo sąrašo pradžios ir paeiliui pereiname visus sąrašo elementus, tol kol randame ieškomą įrašą, arba pasiekiame sąrašo pabaigą.

```
node* find(node* listStart, int t) {
  node *n;
  n = listStart;
  while ( (n <> NULL) && ((*n).T.id <> t) )
    n = (*n).next;
  return (n);
}
```

Jeigu sąraše yra N elementų, tai blogiausiu atveju teks patikrinti visus elementus. Kai tikriname daug įrašų, tai vidutinis paieškos ilgis yra $\frac{N}{2}$. Šis įvertis mažai pasikeičia net ir tada, kai duomenys yra surūšiuoti. Nagrinėdami efektyvius paieškos algoritmus įsitikinsime, kad surūšiuotoje aibėje paieškos sudėtingumas net ir blogiausiu atveju yra tik $\log N$. Aišku, kad realizuodami sparčiuosius paieškos algoritmus naudosime kitas duomenų struktūras.



a)



b)

2.3 pav. a) Tiesinis sąrašas prieš elemento e įterpimą, b) tiesinis sąrašas po įterpimo veiksmo.

Naujo elemento įtraukimas. Šis veiksmas įrašo naują elementą į nurodytą tiesinio sąrašo vietą. Tai gali būti sąrašo pradžia, pabaiga arba vieta, kurią suradome vykdydami paieškos algoritmą.

Jeigu reikia įrašyti elementą e į sąrašo pabaigą ir nėra saugoma nuoroda į paskutinį jo elementą, tai pirmiausia surandame šį elementą, o paskui įtraukiame naują elementą.

```
InsertRear(node* listStart, node* e) {
  node *n, *e;
  n = listStart;
  while ( (*n).next <> NULL )
    n = (*n).next;
  (*e).next = NULL;
  (*n).next = e;
}
```

Ieškodami paskutinio sąrašo elemento patikriname visus jo elementus. Tačiau skirtingai nuo įterpimo į masyvą nereikia keisti elementų vietomis, todėl pats įterpimo veikimas yra labai efektyvus.

Nesunku realizuoti elemento e įterpimą po elemento v, užtenka tik pakeisti dviejų rodyklių reikšmes (žr. 2.3 paveikslą).

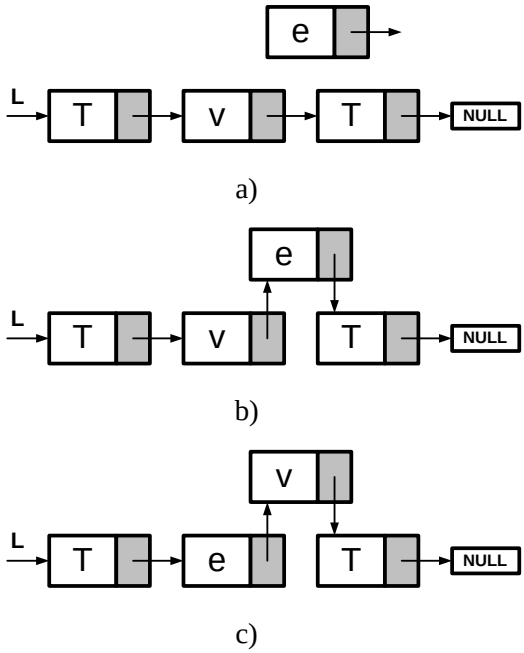
```
InsertAfter(node* v, node* e) {
  (*e).next = (*v).next ;
  (*v).next = e;
}
```

Sudėtingesnis veiksmas yra įterpti naują elementą e prieš vienakrypčio tiesinio sąrašo elementą v. Taip yra todėl, kad žinodami v adresą mes negalime pasiekti prieš tai esantį sarašo elementą. Pateiksime du tokio veiksmo algoritmus.

Pirmajame algoritme iš pradžių e įterpiame po elemento v, o paskui sukeičiame šių elementų informacinių dalių turinius (žr. 2.4 paveikslą).

```
InsertBefore1(node* v, node* e) {
  InsertAfter(v, e);
  T tmp = (*v).T;
  (*v).T = (*e).T;
  (*e).T = tmp;
}
```

Antrajame algoritme visą sąrašą tikriname nuo pradžios ir saugome prieš tai buvusio elemento adresą. Tada suradę elementą v, vėl galime pasinaudoti įterpimo po duotojo elemento veiksmu.

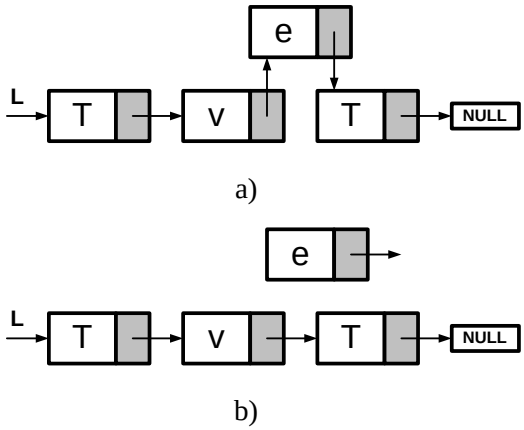


2.4 pav. a) Tiesinis sąrašas prieš elemento e įterpimą, b) tiesinis sąrašas po įterpimo veiksmo, c) tiesinis sąrašas po informacinių dalių turinių sukeitimo vietomis.

```
InsertBefore2(node* listStart, node* v, node* e) {
  node *previuos;
  previous = listStart;
  while ( (*previuos).next <> v )
    previuos = (*previous).next;
  InsertAfter(v, previuos);
}
```

Elementą e nesunku įterpti prieš kitą elementą v ir tada, kai sąrašas yra dvikryptis, tai yra kiekvienas jo elementas turi dvi rodykles: pirmoji rodyklė yra nukreipta į sekantį sąrašo elemetą, o antroji – į prieš tai įrašytą elementą.

Elemento pašalinimas iš tiesinio sąrašo. Kaip ir elemento įterpimo atveju, nesunku pašalinti iš sąrašo elementą, įrašytą po duotojo elemento v. Užtenka patikrinti ar v nėra paskutinis elementas, o po to pakeičiame dviejų rodyklių reikšmes, atkabiname reikalingą elementą ir išlaisviname jam išskirtą kompiuterio atmintį (žr. 2.5 paveikslą).



2.5 pav. a) Tiesinis sąrašas iki elemento, įrašyto po v, pašalinimą, b) tiesinis sąrašas po pašalinimo veiksmo.

```
DeleteAfter(node* v) {
  if ( (*v).next <> NULL){
    (*v).next = (*v).next.next;
    delete (*v).next;
  }
}
```

Daug sunkiau iš vienakrypčio sąrašo pašalintį patį elementą v, nes nežinome, koks elementas yra įrašytas prieš jį. Jeigu nenorime tikrinti visų sąrašo elementų pradėdami nuo pradžios, tol kol rasime v ir pakeliui įsiminsime prieš tai stovinčio elemento adresą, tai galime panaudoti tokį algoritmą:

```
DeleteAt(node* v) {
  if ( v <> NULL){
    if ( v == listStart ){ // Pirmas elementas
      listStart = (*v).next;
      delete v;
    }else{
      if ((*v).next==NULL){ // Paskutinis elementas
        node *e;
        e = listStart;
        while ( (*e).next <> v )
           e = (*e).next;
        (*e).next = NULL;
```

```
        delete v;
      }else{ // Vidurinis elementas
        (*v).T = (*v).next.T;
        (*v).next = (*v).next.next;
        delete (*v).next;
      }
    }
  }
}
```

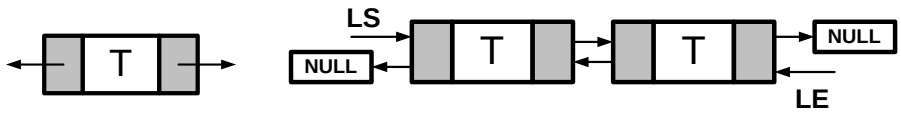
Jeigu elementas v yra sąrašo viduryje, tai kopijuojame informaciją į v iš po jo einančio elemento e ir pašaliname e. Specialūs veiksmai atliekami, kai v yra pirmasis ar paskutinis sąrašo elementas. Pirmuoju atveju sąrašo pradžios rodyklę nukreipiame į po v einantį elementą. Sąrašo pabaigoje esantį elementą šaliname neefektyviu pagrindiniu algoritmu, kai pereiname visą sąrašą ir randame prieš v įrašytą elementą.

2.1.3. Dvikryptis tiesinis sąrašas

Daugelis tiesinio sąrašo veiksmų būtų efektyvesni, jei žinotume ne tik sekančio, bet ir prieš tai įrašyto elemento adresus. Todėl apibrėžkime *dvikryptį* tiesinį sąrašą, kurio kiekvienas elementas turi dvi rodykles: pirmoji rodyklė yra nukreipta į sekantį sąrašo elemetą, o antroji rodo prieš jį įrašytą elementą.

Pirmiausia apibrėžiame vieną atskirą *elementą* kurį sudaro informacinė dalis T ir dvi rodyklės (žr. 2.6 *a* paveikslą):

```
struct node{
  struct T;
  node* next;
  node* previous;
}
```



a) b)

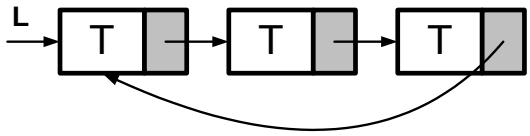
2.6 pav. a) Atskiras dvikrypčio sąrašo elementas, b) dvikryptis tiesinis sąrašas.

Tada *tiesinis dvikryptis sąrašas* – tai pradžios ir pabaigos rodyklės (pažymėkime jas `listStart, listEnd`) ir rodyklėmis susietų elementų seka. Pirmojo

elemento *previous* rodyklė ir paskutiniojo elemento *next* rodyklė rodo į tuščią elementą, tai ir yra sąrašo pradžios bei pabaigos požymiai (žr. 2.6 *b* paveikslą).

2.1.4. Ciklinis sąrašas

Ciklinis sąrašas – tai vienakryptis tiesinis sąrašas, kurio paskutinis elementas yra gretimas pirmajam. Jis pavaizduotas 2.7 paveiksle.



2.7 pav. Ciklinis tiesinis sąrašas.

Tokia duomenų struktūra, pavyzdžiui, patogi aprašant žiediniu maršrutu judančio autobuso sustojimų aibę. Tada, pabaigus pilna ratą, vėl bus skelbiamas pirmojo sustojimo pavadinimas.

Ciklinio sąrašo paskutinio elemento *next* rodyklė sutampa su sąrašo pradžios rodykle, tai ir yra sąrašo pabaigos požymis.

Realizuodami ciklinio sąrašo veiksmus turime neužmiršti, kad paskutinis jo elementas rodo pirmąjį sąrašo elementą. Pavyzdžiui nagrinėkime procedūrą, kuria elementą e įterpiame į ciklinio sąrašo pradžią.

```
InsertFront(node* e) {
  if ( listStart == NULL){ // Pirmas elementas
    (*e).next = e;
  }else{
    (*e).next = (*listStart).next;
    node* n;
    n = (*e).next;
    while ( (*n).next <> listStart )
      n = (*n).next;
    (*n).next = e;
  }
  listStart = e;
}
```

Parodysime, kaip ciklinis sąrašas gali būti efektyviai naudojamas vieno loginio žaidimo algoritmo realizacijoje.

2.1 pavyzdys. Kuris vaikas bus pasirinktas paskutinis? Tarkime, kad N vaikų sustoja ratu, jie atsitiktinai pasirenka skaičių m ir vaiką, nuo kurio bus pradėta skaičiuotė. Tada laikrodžio judėjimo kryptimi suskaičiuojams m-tasis vaikas, kuris palieka ratą. Skaičiuotė tęsiama nuo sekančio vaiko tol, kol lieka vienas laimėtojas.

Pavyzdžiui, tarkime, kad žaidžia penki vaikai A, B, C, D, E ir pasirinktas skaičius $m = 4$. Skaičiuoti pradedame nuo C, tada pirmasis ratą palieka A. Toliau skaičiavimą tęsiame nuo B ir iš žaidimo iškrenta E. Lieka trys vaikai B, C, D, vėl skaičiuojame nuo B, kuris ir palieka ratą. Paskutinėje dvikovoje skaičiuoti pradedame nuo C, o pralaimi D.

Pateikite šio algoritmo realizaciją!