Matematyka dyskretna

Andrzej M. Borzyszkowski

Instytut Informatyki
Uniwersytet Gdański

sem. zimowy 2025/26

`inf.ug.edu.pl/~amb/MaD`

Rekursywne typy danych

Listy

- lista: kolejne elementy
- wiele języków programowania: tablica np. liczb, często elementów tego samego typu
  ale Python: lista dowolnych elementów również różnych typów
- operacje związane z listą
  - utworzenie listy (pustej)
  - sprawdzenie, czy lista jest pusta
  - dodanie elementu do listy
  - pobranie elementu z listy
- dostęp do elementów
  - tablica: podanie numeru dowolnego elementu
  - albo tylko wybrane miejsca na liście
- definicja rekursywna
  - lista może być pusta
  - dodanie elementu do listy tworzy nową listę

Stosy

- lista ma jeden koniec (wierzchołek) na który dodajemy elementy i skąd pobieramy elementy
  LIFO: *last in first out*
- implementacja: tablica $[0..max]$ oraz $top \leqslant max$
  - pusta jeśli $top = 0$
  - pobranie elementu o indeksie $top-1$
  - usunięcie elementu: $top--$
  - wstawienie nowego elementu pod indeks top i przesunięcie $top++$ (o ile $top \leqslant max$, w p.p. nie ma miejsca na dalsze wstawianie)

Stosy c.d.

- implementacja przez wskaźniki w C

```
struct list {
  int elem;
  struct list *next;
  };
struct list *mylist = NULL;
```

- dodanie elementu na początku

```
struct list *add(struct list *m, int e) {
  list *t;
  t = malloc(sizeof(struct list));
  t->elem = e;
  t->next = m;
  m = t;
  return m;
}
```


Kolejki

- lista ma dwa końce: do pobierania elementów (początek) i dokładania nowych (koniec)
  FIFO: *firt in first out*
- implementacja: tablica $[0..max]$ oraz *pocz* i *koniec*
  zajęta część kolejki ma albo indeksy od *pocz* do $koniec-1$ (jeśli $0 \leqslant pocz \leqslant koniec \leqslant max$) albo od *pocz* do *max* i od 0 do $koniec-1$ jeśli $0 \leqslant koniec < pocz \leqslant max$
  - nowa kolejka: $pocz = koniec = 0$
  - pusta jeśli $pocz = koniec$
  - pobranie elementu o indeksie *pocz*
  - usunięcie elementu: $pocz{+}{+} \mod max{+}1$
  - wstawienie nowego elementu pod indeks *koniec* i przesunięcie $koniec{+}{+} \mod max{+}1$ (o ile nie spowoduje to $koniec = pocz$, wówczas nie ma miejsca na dalsze wstawianie)

Kolejki c.d.

- implementacja przez wskaźniki: dodanie elementu na końcu listy

```
struct list *add(struct list *m, int e) {
  if (m == NULL) {
    m = malloc(sizeof(struct list));
    m->elem = e;
    m->next = NULL;
  } else  m->next = add(m->next, e);
  return m;
}
```

- pobranie początkowego elementu

```
int get(struct list *m) {
  return m->elem;
}
```

- usunięcie początkowego elementu

```
struct list *get(struct list *m) {
  return m->next;
}
```


Drzewa z korzeniem

- definicja rekursywna (drzewo binarne)
  - puste drzewo jest drzewem
  - drzewem jest korzeń wraz z dwoma potomkami

```
struct tnode {
  int elem;
  struct tnode *left;
  struct tnode *right;
  };
struct tnode *root = NULL;
```

- można przyjąć inną stałą liczbę potomków, np. trzy dla testów dających trzy wyniki
- można założyć, że każdy węzeł ma listę potomków a nie daną z góry stałą liczbę
- wówczas węzeł ma wskaźnik na początek listy potomków i wskaźnik na kolejnego brata

Drzewo rozbioru gramatycznego

- wyrażenie arytmetyczne
  - liczba i zmienna są wyrażeniami
  - $(w_1) \cdot (w_2)$ jest wyrażeniem
  - $(w_1) + (w_2)$ jest wyrażeniem
- każde wyrażenie definiuje drzewo rozbioru gramatycznego
  - liczba i zmienna są liśćmi (drzewa bez potomków)
  - iloczyn, suma, inne operatory binarne definiują drzewo, którego korzeń ma etykietę operatora, a potomkami są dwa podwyrażenia
  - w tej wersji węzły albo nie mają potomków albo mają dwa
  - można też rozpatrywać operatory z różnymi liczbami argumentów

Notacja polska (beznawiasowa)

- notacja polska prosta – najpierw operator, potem argumenty

```
void treeprint(struct tnode *p) {
 if (p != NULL) {
  printf(" ", p->elem);
  treeprint(p->left);
  treeprint(p->right);
 }}
```

- notacja polska odwrotna – najpierw argumenty, potem operator

```
void treeprint(struct tnode *p) {
 if (p != NULL) {
  treeprint(p->left);
  treeprint(p->right);
  printf(" ", p->elem);
 }}
```


Trzy algorytmy wypisywania drzewa

- notacja polska prosta (prefiksowa/przedrostkowa)
- notacja polska odwrotna (postfiksowa/przyrostkowa/RPN)
- notacja infiksowa
  - dla drzewa rozbioru gramatycznego wymaga nawiasów bez nich zapis będzie niejednoznaczy
  - dla drzewa poszukiwań binarnych gwarantuje wydruk uporządkowany
- algorytm ewaluacji dla notacji polskiej odwrotnej
  - jeśli pobrany element jest liściem, to na stos wkładamy jego wartość
  - jeśli pobrany element jest operatorem, to ze stosu zdejmujemy dwie wartości, używamy operatora i wynik wkładamy na stos

Przeszukiwanie drzew binarnych

- przeszukiwanie wgłąb
  - jeśli istnieje korzeń, zaznaczamy jako odwiedzony i wkładamy na pusty stos
  - badamy element na wierzchu stosu, sprawdzamy, czy istnieje nieodwiedzony potomek
  - jeśli tak, zaznaczamy potomka jako odwiedzony i wkładamy na stos
  - jeśli nie, zdejmujemy ten element ze stosu
  - koniec, jeśli stos został pusty
- przeszukiwanie wszerz
  - jeśli istnieje korzeń, zaznaczamy wkładamy na koniec (pustej) kolejki
  - usuwamy element z początku kolejki ale od razu wstawiamy wszystkich potomków na koniec kolejki
  - koniec algorytmu, jeśli kolejka jest pusta