

Diskrétne geometrické štruktúry

1.

Martin Florek

florek@sccg.sk

www.sccg.sk/~florek

Členenie

- Zložitosť
- Základné dátové štruktúry
- Stromy
- Geometrické prehľadávanie priestoru
(range trees, search trees, quadtrees, Kd-trees, BSP...)
- Ohraničujúce objekty
- Reprezentácie objektov
- Reprezentácie grafov, manifoldov
(winged-edge, half-edge, quad-edge)
- Algoritmy na grafoch

Literatúra

- Elmar Langetepe, Gabriel Zachmann: Geometric Data Structures for Computer Graphics, AK Peters
- Hanan Samet: Foundations of Multidimensional and Metric Data Structures, The Morgan Kaufmann Series in Computer Graphics
- Donald E. Knuth: The Art of Computer Programming, Addison-Wesley Professional
- N.Wirth: Algoritmy a štruktúry údajov, Alfa

Algoritmus

- Konečný návod ako riešiť problém s použitím daných elementárnych operácií
- Dobre definovaná procedúra, ktorá pre nejakú množinu vstupov vyprodukuje výstupnú hodnotu alebo množinu hodnôt
- Postupnosť krokov výpočtu, ktorý transformuje vstup na výstup
- Algoritmus nazývame správny, ak pre každý príklad vstupu sa zastaví so správnym výstupom

Zložitosť algoritmu

- Miera zložitosti X (čas, pamäť, počet aritmetických operácií...)
- Funkcia veľkosti vstupných dát udávajúca množstvo miery zložitosti X spotrebovanej algoritmom A pri riešení problému
- Veľkosť vstupu: tento pojem závisí od typu problému - počet prvkov vstupu, počet bitov, dva a viac parametrov (počet vrcholov, počet hrán)
- Najčastejšie časová a pamäťová zložitosť

Asymptotická zložitosť

- Čas výpočtu algoritmu pri dostatočne veľkom vstupe
- Správanie sa algoritmu pri zväčšujúcom sa vstupe
- Používanie asymptotickej notácie
- V mnohých prípadoch sa používajú aj iné typy zložitosti, napr. presný počet operácií...

O-notácia

- Asymptotická horná hranica, horné ohraničenie zložitosti
- Zložitosti najhoršieho prípadu
- Definovanie množiny funkcií, do ktorej potom spadá aj skúmaná zložitost'
- $O(g(n)) = \{f(n): \text{existujú kladné konštanty } c \text{ a } n_0 \text{ také, že } 0 \leq f(n) \leq c \cdot g(n) \text{ pre všetky } n \geq n_0\}$
- $a \cdot n^2 + b \cdot n + c \in O(n^2)$, namiesto \in sa píše
=

Ω -notácia

- Asymptotická spodná hranica, dolné ohraničenie zložitosti
- Zložitosti najlepšieho prípadu
- $\Omega(g(n)) = \{f(n): \text{existujú kladné konštanty } c \text{ a } n_0 \text{ také, že } 0 \leq c \cdot g(n) \leq f(n) \text{ pre všetky } n \geq n_0\}$
- $a \cdot n^2 + b \cdot n + c = \Omega(1)$

θ -notácia

- $\theta(g(n)) = \{f(n): \exists \text{ kladné konštanty } c_1, c_2 \text{ a } n_0 \text{ také, že } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ pre } \forall n \geq n_0\}$
- $g(n)$ je asymptoticky tesná hranica pre $f(n)$
- Pre $n \geq n_0$ funkcia $f(n)$ je rovná funkcii $g(n)$, až na konštantný faktor.
- Pre ľubovoľné dve funkcie $f(n)$ a $g(n)$ platí:
 - $f(n) = \theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$

“Malé” notácie

- Nie sú asymptoticky tesné
- $o(g(n)) = \{f(n): \forall c > 0 \exists n_0 > 0 \text{ také, že } 0 \leq f(n) < cg(n) \text{ pre } \forall n \geq n_0\}$
- $\omega(g(n)) = \{f(n): \forall c > 0 \exists n_0 > 0 \text{ také, že } 0 \leq cg(n) < f(n) \text{ pre } \forall n \geq n_0\}$
- $f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$
- $2n = o(n^2)$, ale $2n^2 \neq o(n^2)$
- $n^2/2 = \omega(n)$, ale $n^2/2 \neq \omega(n^2)$

Dátové štruktúry

- Definované najčastejšie na dynamických množinách
- Vnútoraná organizácia dát
- Implementácia operácií
- U - univerzálna množina – množina prvkov, ktoré sa môžu vyskytovať ako prvky dátovej štruktúry
- S – množina prvkov dátovej štruktúry

Operácie

- $\text{MEMBER}(x, S)$ – určiť, či $x \in S$, ak áno, tak určiť miesto pamäti (resp. smerník), kde je x uložený.
- $\text{INSERT}(x, S)$ – vlož x do S
- $\text{DELETE}(x, S)$ – zmaž x z S

Operácie 2

- Ak je univerzálna množina usporiadaná, používajú sa aj ďalšie operácie
 - $\text{MIN}(S)$ – vráti najmenší prvok z S
 - $\text{MAX}(S)$ – vráti najväčší prvok z S
 - $\text{SUCCESSOR}(S, x)$ – nasledovník prvku x v množine S
 - $\text{PREDECESSOR}(S, x)$ – predchodca prvku x v množine S
 - $\text{SPLIT}(x, S)$ rozloží množinu S na 2 disjunktné množiny S_1, S_2 tak, že $S_1 = \{y; y \in S, y \leq x\}$, $S_2 = \{y; y \in S, y > x\}$
 - $\text{CONCATENATE}(S_1, S_2)$ – Ak pre $\forall x' \in S_1, \forall x'' \in S_2$ platí $x' < x''$ vytvorí sa usporiadaná

Zoznam

- Pole
- Jednotlivé položky usporiadané v pamäti za sebou
- member - $O(n)$, insert - $O(1)$, delete - $O(n)$
- Problémy s alokáciou pamäte, presunom veľkých blokov, pretečením
- Utriedený zoznam: member - $O(\log n)$, insert - $O(n)$, delete - $O(n)$

Zoznam - algoritmy

```
ARRAY_MEMBER(x, P)
{
    N=PocetPrvkovPola();
    for(i=0;i<N;i++)
        if(x==P[i])
            return i;
    return -1;
}
```

```
ARRAY_INSERT(x, P)
{
    N=PocetPrvkovPola();
    RealokujViacPamäte();
    P[N]=x;
}
```

```
ARRAY_DELETE(x, P)
{
    pos=MEMBER(x, P);
    N=PocetPrvkovPola();
    for(i=pos;i<N-1;i++)
        P[i]=P[i+1];
    RealokujMenejPamäte();
}
```

```
SORTED_ARRAY_MEMBER(x, P)
{
    N=PocetPrvkovPola();
    low = 0;
    high = N - 1;
    while (low <= high)
    {
        mid = (low + high) / 2;
        if (P[mid] > x) high = mid - 1;
        else if (P[mid] < x) low = mid + 1;
        else return P[mid];
    }
    return -1;
}
```

```
SORTED_ARRAY_INSERT(x, P)
{
    N=PocetPrvkovPola();
    low = 0;
    high = N - 1;
    while (low <= high)
    {
        mid = (low + high) / 2;
        if (P[mid] > x) high = mid - 1;
        else if (P[mid] < x) low = mid + 1;
        else {high=low=mid;break;}
    }
    RealokujViacPamäte();
    for(i=N;i>low;i--)
        P[i]=P[i-1];
    P[low]=x;
}
```

Spájaný zoznam

- Záznamy v pamäti nenadväzujú
- Prepojenie záznamov pomocou smerníkov
- Vhodné pre dynamické štruktúry
- Každý záznam obsahuje:
 - Kľúč jednoznačne identifikujúci záznam (može byť totožný s dátami)
 - Smerník na nasledujúci záznam
 - Smerník na predchádzajúci záznam (pre obojsmerné zoznamy)
 - Dáta
- Uchováваме ešte smerník na prvý prvok – referencia na zoznam

Spájaný zoznam 2

- Smerníky na koncoch zoznamu = NULL
- Cyklický zoznam - prvý a posledný prvok sú prepojený
- Sentinel – prvok, ktorý nenesie dáta (prázdny záznam), určuje začiatok alebo koniec zoznamu, slúži ako smerník na zoznam, môže urýchliť niektoré operácie
- Usporiadáný zoznam

Spájaný zoznam - algoritmy

```
struct LinkedList
{
    LinkedListRecord* head;
}
```

```
struct LinkedListRecord
{
    int key;
    LinkedListRecord* prev;
    LinkedListRecord* next;
    void* data;
}
```

```
LIST_MEMBER(L, k)
{
    x = L->head;
    while (x != NULL && x->key != k)
        do x = x->next;
    return x;
}
```

```
LIST_DELETE(L, x)
{
    if (x->prev != NULL)
        then [x->prev]-> next = x->next;
    else L->head = x->next;
    if (x->next != NULL)
        then [x->next]-> prev = x-> prev;
}
```

```
LIST_INSERT(L, x)
{
    x->next = L->head;
    if (L->head != NULL)
        then L->head->prev = x;
    L->head = x;
    x->prev = NULL;
}
```

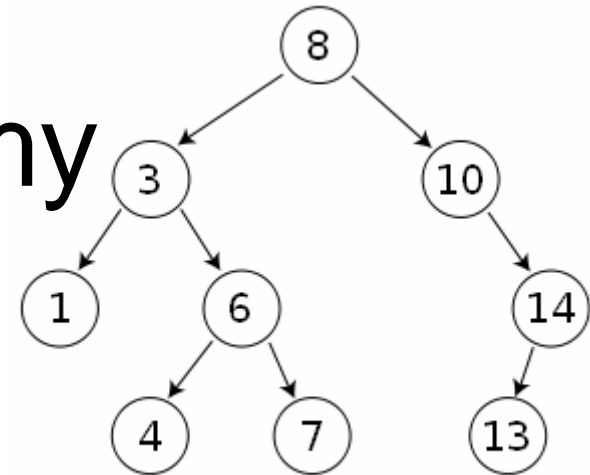
Binárny strom

- Prvok binárneho stromu obsahuje:
 - kľúč (prvok z usporiadateľnej nožiny)
 - smerník na ľavého syna
 - smerník na pravého syna
 - smerník na rodiča (nemusí)
 - dáta
- Koreň – Root – jediný prvok bez rodiča
- List – Leaf – prvok bez synov

Binárny prehľadávací strom (BPS)

- Binárny prehľadávací strom:
 - nech x je ľubovoľný vrchol BPS a y je ľubovoľný vrchol z ľavého resp. pravého podstromu vrchola x
 - potom: $(y \rightarrow \text{key} < x \rightarrow \text{key})$ resp. $(y \rightarrow \text{key} \geq x \rightarrow \text{key})$
- Výška stromu je dĺžka najdlhšej cesty od koreňa k listu +1
- Výška stromu je definovaná ako maximálna úroveň ľubovoľného vrchola stromu. Koreň stromu je na prvej úrovni. Ozn. $h(r)$ výšku stromu s koreňom r .
- Výška stromu s jedným vrcholom je 1.
- Operácie MEMBER, INSERT, DELETE, MIN, MAX na BPS s výškou h sa vykonajú v čase $O(h)$.

BPS - algoritmy



```
struct BPS
{
    BPSNode* head;
}
```

```
struct BPSNode
{
    int key;
    BPSNode* left;
    BPSNode* right;
    BPSNode* parent;
    void* data;
}
```

```
TREE_MEMBER(T, k)
{
    x = T->head;
    while ((x != NULL) && (k != x->key))
    {
        if (k < x->key) x = x->left;
        else x = x->right;
    }
    return x;
}
```

```
TREE_DELETE(T, z)
{
    if (z->left == NULL || z->right == NULL) y = z;
    else y = TREE_SUCCESOR(z);
    if (y->left != NULL) x = y->left;
    else x = y->right;
    if (x != NULL) x->parent = y->parent;
    if (y->parent == NULL) T->root = x;
    else
    {
        if (y == y->parent->left) y->parent->left = x;
        else y->parent->right = x;
    }
    if (y != z)
    {
        z->key = y->key;
        Skopiruj ostatne data;
    }
    return y;
}
```

BPS – algoritmy 2

```
TREE_SUCCESOR(x)
{
    if (x->right != NULL)
        return TREE_MINIMUM (x->right);
    y = x->parent;
    while ((y != NULL) && (x == y->right))
    {
        x = y;
        y = y->parent;
    }
    return y;
}
```

```
TREE_MINIMUM(x)
{
    while (x->left != NULL)
        x = x->left;
    return x;
}

TREE_MAXIMUM(x)
{
    while (x->right != NULL)
        x = x->right;
    return x;
}
```

```
TREE_INSERT(T, z)
{
    y = NULL;
    x ← T->root;
    while (x != NULL)
    {
        y = x;
        if (z->key < x->key) x = x->left;
        else x = x->right;
    }
    z->parent = y;
    if (y == NULL) T->root = z;
    else
    {
        if (z->key < y->key) y->left = z;
        else y->right = z;
    }
}
```

Prehľadávanie BPS

```
PREORDER(x)
{
    if (x == NULL) return;
    print x->key;
    PREORDER(x->left);
    PREORDER(x->right);
}
```

```
INORDER(x)
{
    if (x == NULL) return;
    INORDER x->left);
    print x->key;
    INORDER x->right);
}
```

```
POSTORDERx)
{
    if (x == NULL) return;
    POSTORDERx x->left);
    POSTORDERx x->right);
    print x->key;
}
```

koniec (-:

florek@sccg.sk