Osnove:
osnovne podatkovne strukture, algoritmi in tehnike

Andrej Brodnik, Matevž Jekovec

Načrt sprehoda

- uvod
- osnovne podatkovne strukture: sklad, vrsta, slovar, vrsta s prednostjo, množica
- drevesa za okus
- delo z nizi
- veliki *O* in prijatelji
- urejanje (*sorting*)
- osnovne tehnike načrtovanja algoritmov

- Literatura:
  - Steven S. Skiena, Miguel A. Revilla: Programming Challenges – The Programming Contest Training Manual. Springer 2003.

Uvod

- od problema do rešitve:
  - idejni načrt rešitve (invariante)
  - ocena idejnega načrta
  - programska realizacija načrta
- izbira programskega jezika
  - raba sistemskih knjižnic in okolij

Uvod

- preverjanje in razhroščevanje rešitve
  - na danih primerih
  - robni pogoji
  - preverjanje na velikih primerih
  - invariante (!!)
  - sledenje programu:
    - z uporabo razhroščevalnika
    - z izpisi

Osnovne podatkovne strukture

- vsemogočno polje
- pri večini primerov so podane velikosti v naprej in lahko na podlagi tega sklepamo o njihovi velikosti
- statično zasedanje pomnilnika
- osnovne podatkovne strukture:
  - sklad
  - vrsta
  - slovar
  - vrsta s prednostjo
  - množica
- obstaja sistemska podpora zanje

Sklad *(stack)*

- naloži: push
- poberi: pop
- poln, prazen: full, empty
- naredi/nastavi: initialize

- realizacija s poljem
  - samo polje za hranjenje podatkov
  - dodatne spremenljivke za knjigovodstvo – ohranjanje strukture

Vrsta *(queue)*

- dodaj: enqueue
- izloči: dequeue
- polna, prazna: full, empty
- naredi/nastavi: initialize

- realizacija s poljem
  - samo polje za hranjenje podatkov
  - dodatne spremenljivke za knjigovodstvo – ohranjanje strukture

  - krožno polje/vmesnik

Slovar *(dictionary)*

- vstavi: insert
- izloči: delete
- najdi: find
- naredi/nastavi: initialize

- v slovar se vstavljajo sestavljeni elementi: <ključ, podatki>

- statični slovar: naredimo enkrat za vselej
- poldinamični slovar: samo vstavljamo
  - zelo primerne so razpršene tabele
- dinamični slovar
  - še vedno razpršene tabele, le uporabimo veriženje za razreševanje sovpadanja

Vrsta s prednostjo *(priority queue)*

- vstavi: insert
- izloči najmanjšega: delmin
- največji: min
- naredi/nastavi: initialize

- v vrsto s prednostjo se vstavljajo sestavljeni elementi: <prednost, podatki>

- lahko še druge operacije
  - spremeni prednost: change priority

Množica *(set)*

- unija: union
- presek: intersection
- vstavi / izloči: insert / delete
- element množice: member
- naredi/nastavi: initialize

- zelo raznolike zahteve za delo z množicami
- običajno implementirano z uporabo drugih podatkovnih struktur

Osnovne podatkovne strukture

- redko se jih ponovno implementira, ampak se uporablja sistemske knjižnice
  - C++ Standard Template Library – STL
  - Java java.util package
- pojem vmesnika (*interface*, *abstract class*, ...) in implementacije (*class*)

Osnovne podatkovne strukture

- C++:
  - sklad: S.push, S.pop, S.top, S.empty
  - vrsta: Q.front, Q.back, Q.push, Q.pop, Q.empty
  - slovar: npr. hash_map: H.erase, H. find, H.insert
  - vrsta s prednostjo: PQ.top, PQ.push, PQ.pop, PQ.empty
  - množice: urejeni asociativni vsebovalniki
    - set<key, comparison> S;
- javanske knjižnice imajo več različnih implementacij: abstraktni razred, konkretni razred

Drevesa za okus

- opravka imamo z elementi oblike <ključ, podatek>
- osnovna splošna definicija drevesa *(tree)* je rekurzivna:
  - koren z neko vrednostjo
  - levo poddrevo z elementi, ki so manjši od elementa v korenu
  - desno poddrevo z elementi, ki so večji od elementa v korenu
- uporabno za implementacijo slovarja, vrste s prednostjo, ...
- prihajajo v različnih oblikah in okusih
  - dvojiška drevesa, dvojiška iskalna drevesa, uravnotežena dvojiška iskalna drevesa (AVL, RČ)
  - večsmerna drevesa (B drevesa, B+ drevesa, ...)
- ni namen tega predavanja podroben pregled

Drevesa za okus

- opravka imamo z elementi oblike <$k_1 k_2 k_3 ... k_m$, podatek> in ki je iz neke abecede Σ, $|\Sigma| = s$
- številsko drevo *(trie)*
  - iz besede *retrieval*
- osnovna splošna definicija je rekurzivna:
  - koren
  - levo poddrevo z elementi, ki se prično s prvo črko Σ,
  - levo poddrevo z elementi, ki se prično z drugo črko Σ, ...
- uporabno za delo s črkovnimi nizi
- izboljšave: stiskanje po poti in po plasteh (*path* in *level compression*)
- PATRICIA

Delo z nizi

- dve osnovni predstavitvi:
  - z zaključnim znakom (*null terminated*)
  - z dolžino
  - (kot seznam črk)
- branje: scanf, getchar
- osnovne operacije:
  - iskanje črke, podniza, podzaporedja
  - lepljenje (*concatenation*)
  - zamenjava podniza
  - ...

Delo z nizi – C

```
#include <ctype.h>       /* include the character library */

int isalpha(int c);      /* true if c is either upper or lower case */
int isupper(int c);      /* true if c is upper case */
int islower(int c);      /* true if c is lower case */
int isdigit(int c);      /* true if c is a numerical digit (0-9) */
int ispunct(int c);      /* true if c is a punctuation symbol */
int isxdigit(int c);     /* true if c is a hexadecimal digit (0-9,A-F) */
int isprint(int c);      /* true if c is any printable character */

int toupper(int c);      /* convert c to upper case -- no error checking */
int tolower(int c);      /* convert c to lower case -- no error checking */
```

```
#include <string.h>      /* include the string library */

char *strcat(char *dst, const char *src);     /* concatenation */
int strcmp(const char *s1, const char *s2);   /* is s1 == s2? */
char *strcpy(char *dst, const char *src);     /* copy src to dist */
size_t strlen(const char *s);                 /* length of string */
char *strstr(const char *s1, const char *s2); /* search for s2 in s1 */
char *strtok(char *s1, const char *s2);       /* iterate words in s1 */
```

Delo z nizi – C++

```
string::size()          /* string length */
string::empty()         /* is it empty */
string::c_str()         /* return a pointer to a C style string */

string::operator [](size_type i)        /* access the ith character */

string::append(s)       /* append to string */
string::erase(n,m)      /* delete a run of characters */
string::insert(size_type n, const string&s) /* insert string s at n */

string::find(s)
string::rfind(s)        /* search left or right for the given string */

string::first()
string::last()          /* get characters, also there are iterators */
```

Delo z nizi – Java

- razred String – statični nizi
- razred StringBuffer
- paket java.text

Veliki O in prijatelji

- osnovna težava:
  - Peter in Špela zapišeta enak algoritem, a Špela je spretnejša in piše učinkoviteje in bolj jedrnato – bolje pozna jezik
  - Peter nato uporabi hitrejši procesor
  - kako primerjati izvedbe med seboj? Kaj merimo?

Veliki O in prijatelji

- definiramo družino funkcij $O(f(n))$:
  - v družini funkcij z imenom $O(f(n))$, so vse funkcije $g(n)$ nad spremenljivko n, za katere velja: $f(n) \geq c\ g(n)$ za neko konstanto c (za dovolj velik n)
- primeri:
  - $100\ n \in O(n)$
  - $0{,}000001\ n \in O(n)$
  - $\log 10\ n \in O(n)$
  - $3{,}14\ n + \log 10\ n \in O(n)$
- $f(n)$ predstavlja na nek način največjo možno funkcijo v družini funkcij
- obstajata še prijatelja:
  - Ω: najmanjša funkcija v družini
  - θ: hkrati največja in najmanjša funkcija

Veliki O in prijatelji

- osnovna težava:
  - Peter in Špela zapišeta enak algoritem, a Špela je spretnejša in piše učinkoviteje in bolj jedrnato – bolje pozna jezik
  - Peter nato uporabi hitrejši procesor
  - kako primerjati izvedbe med seboj? Kaj merimo?
- obe rešitvi sta enako zahtevni, če sta njuni časovni zahtevnosti v isti družini funkcij
  - časovna zahtevnost je funkcija, ki šteje število primerjav, ki jih naredi program pri n podatkih

Urejanje *(sorting)*

- pogosto uporabljamo:
  - preverjanje enoličnosti
  - brisanje dvojnikov
  - razporejanje dogodkov po prednosti
  - srednji element (mediana) – poseben primer (!)
  - štetje pogostnosti
  - rekonstrukcija izvorne urejenosti
  - presek množic in (prava) unija
  - iskanje vsote dveh elementov
  - učinkovito (večkratno) iskanje
  - ...

Urejanje *(sorting)*

- stabilno in nestabilno urejanje
- različni načini urejanja:
  - $O(n^2)$: urejanje z vstavljanjem (*insertion*), z izbiranjem (*selection*), z mehurčki (*buble*), ...
  - $O(n \log n)$: z zlivanjem (*merge*), z vrsto s prednostjo / kopico (*heap*)
  - $O(n \log n)$ – pričakovano: hitro urejanje (*quick*)
  - ...
  - $O(m+n)$: korensko (*radix*), z vedri (*bucket*), s štetjem (*counting*), ...
- lahko napišemo svojo lastno implementacijo ali uporabimo knjižnico

Urejanje – C

```
#include <stdlib.h>
void qsort(void *base, size_t nel, size_t width,
     int (*compare) (const void *, const void *));

int intcompare(int *i, int *j) {
  if (*i > *j) return (1);
  if (*i < *j) return (-1);
  return (0);
}

qsort((char *) a, cnt, sizeof(int), intcompare);

bsearch(key, (char *) a, cnt, sizeof(int), intcompare);
```

Urejanje – C++

```
void sort(RandomAccessIterator bg,
    RandomAccessIterator end)
void sort(RandomAccessIterator bg,
    RandomAccessIterator end, BinaryPredicate op)


void stable_sort(RandomAccessIterator bg,
    RandomAccessIterator end)
void stable_sort(RandomAccessIterator bg,
    RandomAccessIterator end, BinaryPredicate op)
```

Urejanje – Java

```
static void sort(Object[] a)
static void sort(Object[] a, Comparator c)

binarySearch(Object[] a, Object key)
binarySearch(Object[] a, Object key, Comparator c)
```

Osnovne tehnike načrtovanja algoritmov

- pregledovanje:
  - izberemo najboljšo možno rešitev.
    - možne rešitve so že naračunane vnaprej, le najti jo moramo
  - običajno to pomeni pregledovanje polja ali podobno
  - primer: *iskanje najmanjšega elementa*

Osnovne tehnike načrtovanja algoritmov

- požrešna tehnika:
  - v vsakem koraku poskušamo narediti največji korak proti cilju (smo požrešni in hočemo največ v dani situaciji)
  - skoraj vedno se uporablja neka vrsta s prednostjo
  - primer: *iskanje najcenejšega vpetega drevesa v grafu, ... – sledi jutri*
- deli in vladaj:
  - v vsakem koraku razdelimo problem na enega ali več podproblemov, ki jih ločeno in neodvisno rešimo
  - primer: *urejanje z zlivanjem (merge sort); iskanje elementa v iskalnem drevesu*

Osnovne tehnike načrtovanja algoritmov

- z vračanjem:
  - ko pridemo do razpotja imamo dve možnosti: (i) če obstaja naslednji korak ali več različnih korakov na poti k rešitvi, izberemo enega in nadaljujemo; ali (ii) če ni naslednjega koraka, se vrnemo nazaj do tam, kjer smo zadnjič izbrali enega od možnih korakov ter izberemo tam naslednji možni korak
  - potrebujemo podatkovno strukturo za beleženje razpotij in le-ta je običajno vrsta ali sklad
  - primer: *iskanje ciklov v grafu, iskanje premera grafa, pregledovanje grafov – sledi jutri*

Osnovne tehnike načrtovanja algoritmov

- dinamično programiranje:
  - v vsakem koraku moramo izbrati najboljšo možno rešitev (pregledovanje) med vsemi podproblemi, ki pa na začetku še niso naračunane (so dinamične). Imamo dve možnosti, da jih v naračunamo vnaprej ali sproti.
  - vedno imamo dvorazsežnostno tabelo/polje, v katerem shranjujemo najboljše možne rešitve podproblemov.
  - primer: *iskanje razdalje urejanja (edit distance), ... – sledi v četrtek*