Laboratorium: projekt nr 4 *26 stycznia 2006*

Algorytmy i Struktury Danych

ZBIORY DYNAMICZNE

Instytut Informatyki Uniwersytetu Wrocławskiego *Paweł Rzechonek*

1 Kopce złączalne

Zadanie:

Zaimplementuj *złączalne kolejki priorytetowe* na dwa sposoby:

- kopce dwumianowe (ang. *binomial heaps*) ;
- kopce lewicowe (ang. *leftist heaps*) .

W kopcach tych mają być przechowywane liczby całkowite.

Następnie napisz programy testujące Twoje struktury danych. Program testujący powinien interpretować i realizować następujące rozkazy (jeden rozkaz w jednej linii):

- Sprawdzenie liczby elementów w kolejce (przykładowo w kolejce `A`):
  `size A`
  W odpowiedzi program wypisuje liczbę wszystkich elementów w kolejce.
- Wstawienie elementu do kolejki (przykładowo 127 do kolejki `B`):
  `insert B 127`
  W odpowiedzi program wypisuje wartość elementu wstawionego do kolejki.
- Sprawdzenie jaka jest maksymalna wartość w kolejce (przykładowo w kolejce `D`):
  `max D`
  W odpowiedzi program wypisuje wartość elementu maksymalnego w kolejce.
- Usunięcie z kolejki elementu maksymalnego (przykładowo z kolejki `E`):
  `extract E`
  W odpowiedzi program wypisuje wartość elementu maksymalnego usuniętego z kolejki.
- Usunięcie wszystkich elementów z kolejki (przykładowo `F` $\leftarrow \emptyset$, kolejka `F` jest opróżniana):
  `clear F`
  Procedura ma być zaimplementowana w taki sposób, że powtarzana jest funkcja `extract` aż do opróżnienia kolejki. W odpowiedzi program wypisuje w kolejności nierosnącej wartości wszystkich elementów z opróżnianej kolejki (elementy wypisywane są w jednej linii i pooddzielane pojedynczymi spacjami).

- Skopiowanie kolejki (przykładowo `R` ← `S`, stara zawartość kolejki `R` jest usuwana, a następnie zawartość kolejki `S` jest przenoszona do kolejki `R`):
  `copy R S`
  W odpowiedzi program wypisuje liczbę wszystkich elementów w nowej kolejce.
- Przyłączenie kolejki (przykładowo `T` ← `T`∪`S` oraz `S` ← ∅, do starej zawartości kolejki `T` jest dopisywana zawartość kolejki `U`, a kolejka `U` zostaje opróżniona):
  `meld T U`
  W odpowiedzi program wypisuje liczbę wszystkich elementów w nowej kolejce.
- Wyjście z programu:
  `exit` Program nie wypisuje żadnej odpowiedzi.

Każda odpowiedź programu ma być wypisana w oddzielnej linii.

Zadanie polega na zaaplikowaniu do obu złączalnych kolejek priorytetowych takich samych danych (sprawdź czy wyniki się zgadzają), dokonania pomiaru czasów działania tych struktur danych i rozstrzygnięciu, która z nich jest szybsza.

Uwaga:

Powinieneś mieć dwa osobne programy i uruchamiać je oddzielnie. Początkowo w każdym programie ma być utworzonych 26 pustych kolejek priorytetowych oznaczonych kolejnymi literami alfabetu od `A` do `Z`.

Dane:

W pierwszym wierszu z danymi jest podana liczba n ($1 \leq n \leq 1000000$) oznaczająca liczbę instrukcji wykonywanych na początkowo pustych kolejkach priorytetowych, a w kolejnych n wierszach zapisane są rozkazy zgodnie ze składnią opisaną wcześniej.

Wyniki:

W wyniku należy wypisać w n wierszach wszystkie n odpowiedzi generowane przez program testujący (każda odpowiedź w osobnym wierszu).

Sprawozdanie:

W sprawozdaniu należy zamieścić opis struktury danych wraz z opisem wykonywanych na tej strukturze operacji. Napisz, jakie czasy otrzymałeś/otrzymałaś testując swoje kolejki priorytetowe na dużych danych.

2 Zbiory rozłączne

Zadanie:

Zaimplementuj struktury do pamiętania *zbiorów rozłącznych* (ang. *disjoint sets*) na dwa sposoby:

- reprezentacja listowa z łączeniem według rozmiarów;
- reprezentacja drzewiasta z łączeniem według rozmiarów i kompresją ścieżek.

W strukturach tych mają być przechowywane liczby całkowite.

Następnie napisz programy testujące Twoje struktury danych. Program testujący powinien interpretować i realizować następujące rozkazy (jeden rozkaz w jednej linii):

- Sprawdzenie ile jest zbiorów:
  `sets`
  W odpowiedzi program wypisuje liczbę wszystkich zbiorów.
- Sprawdzenie ile jest elementów w zbiorze (przykładowo w zbiorze, do którego należy 31):
  `size 31`
  W odpowiedzi program wypisuje liczbę elementów znalezionego zbioru.
- Sprawdzenie, do którego zbioru należy element (przykładowo 63):
  `find 63`
  W odpowiedzi program wypisuje wartość elementu, który jest reprezentantem zbioru.
- Połączenie zbiorów (przykładowo zbiorów, do których należą 127 i 255):
  `union 255 127`
  W odpowiedzi program wypisuje wartość elementu, który jest reprezentantem nowopowstałego zbioru. Jeśli łączone zbiory mają taki sam rozmiar, to reprezentanta o większej wartości podczepiamy do reprezentanta o mniejszej wartości.
- Wyjście z programu:
  `exit` Program nie wypisuje żadnej odpowiedzi.

Każda odpowiedź programu ma być wypisana w oddzielnej linii.

Zadanie polega na zaaplikowaniu do obu reprezentacji zbiorów rozłącznych takich samych danych (sprawdź czy wyniki się zgadzają), dokonania pomiaru czasów działania tych struktur danych i rozstrzygnięciu, która z nich jest szybsza.

Uwaga:

Powinieneś mieć dwa osobne programy i uruchamiać je oddzielnie.

Dane:

W pierwszym wierszu z danymi jest podana liczba k ($1 \leq k \leq 1000000$) oznaczająca początkową liczbę elementów i zbiorów (elementy są kolejnymi liczbami całkowitymi z zakresu $0, \ldots, k-1$), w drugim wierszu jest podana liczba n ($1 \leq n \leq 1000000$) oznaczająca liczbę instrukcji wykonywanych na początkowo jednoelementowych zbiorach, a w kolejnych n wierszach zapisane są rozkazy zgodnie ze składnią opisaną wcześniej.

Wyniki:

W wyniku należy wypisać w n wierszach wszystkie n odpowiedzi generowane przez program testujący (każda odpowiedź w osobnym wierszu).

Sprawozdanie:

W sprawozdaniu należy zamieścić opis struktury danych wraz z opisem wykonywanych na tej strukturze operacji. Napisz, jakie czasy otrzymałeś/otrzymałaś testując swoje struktury dla zbiorów rozłącznych na dużych danych.

3 Drzewa SPLAY i drzewa R–B

Zadanie:

Zaimplementuj *zrównoważone drzewa BST* (ang. *ballanced BST*) na dwa sposoby:

- drzewa rozchylane SPLAY;
- drzwa czerwono–czarne R–B.

W drzewach tych mają być przechowywane liczby całkowite.

Następnie napisz programy testujące Twoje struktury danych. Program testujący powinien interpretować i realizować następujące rozkazy (jeden rozkaz w jednej linii):

- Sprawdzenie liczby elementów w drzewie:
  `size`
  W odpowiedzi program wypisuje liczbę wszystkich elementów w drzewie.
- Sprawdzenie czy element jest w drzewie (przykładowo 17):
  `search 17`
  W odpowiedzi program wypisuje wartość 1, gdy element został znaleziony, lub 0 gdy nie było go w drzewie.
- Wstawienie elementu do drzewa (przykładowo 33):
  `insert 33`
  W odpowiedzi program wypisuje wartość 1, gdy element został wstawiony, lub 0 gdy elementu nie wstawiono (w drzewie już był taki element).
- Usunięcie elementu z drzewa (przykładowo 65):
  `delete 65`
  W odpowiedzi program wypisuje wartość 1, gdy element został usunięty, lub 0 gdy elementu nie usunięto (w drzewie takiego elementu nie było).
- Wypisanie wszystkich elementów zapamiętanych w drzewie w porządku *inorder*:
  `print`
  W odpowiedzi program wypisuje w kolejności niemalejącej wartości wszystkich elementów zapamiętanych w drzewie BST (elementy wypisywane są w jednej linii i pooddzielane pojedynczymi spacjami).
- Wyjście z programu:
  `exit` Program nie wypisuje żadnej odpowiedzi.

Każda odpowiedź programu ma być wypisana w oddzielnej linii.

Zadanie polega na zaaplikowaniu do obu drzew BST takich samych danych (sprawdź czy wyniki się zgadzają), dokonania pomiaru czasów działania tych struktur danych i rozstrzygnięciu, która z nich jest szybsza.

Uwaga:

Powinieneś mieć dwa osobne programy i uruchamiać je oddzielnie. Początkowo drzewo ma być puste.

Dane:

W pierwszym wierszu z danymi jest podana liczba n ($1 \leq n \leq 1000000$) oznaczająca liczbę instrukcji wykonywanych na początkowo pustym drzewie BST, a w kolejnych n wierszach zapisane są rozkazy zgodnie ze składnią opisaną wcześniej.

Wyniki:

W wyniku należy wypisać w n wierszach wszystkie n odpowiedzi generowane przez program testujący (każda odpowiedź w osobnym wierszu).

Sprawozdanie:

W sprawozdaniu należy zamieścić opis struktury danych wraz z opisem wykonywanych na tej strukturze operacji. Napisz, jakie czasy otrzymałeś/otrzymałaś testując swoje drzewa BST na dużych danych.