Wojciech Myszka

Laboratorium 5a: Wyszukiwanie binarne — doskonalenie aplikacji

2022-04-03 10:39:40 +0200

1. Wprowadzenie

Jak się wydaje większość stworzonych aplikacji jest w miarę poprawna. To czego im zazwyczaj brakuje to:

1. Dokładne przetestowanie. W szczególności sprawdzić należy:
   — tablice z danymi o długości 1 i 2 (warto też sprawdzić zachowanie algorytmu dla tablic o parzystej i nieparzystej liczbie danych),
   — sprawdzenie czy poprawnie znajdowane są pierwsza i ostatnia wartość z tablicy danych,
   — sprawdzenie czy poprawnie traktowane są szukane wartości mniejsze/większe od najmniejszej/największej wartości w tablicy.
2. Usunięcia wszystkich wydruków poza pętle. W tej chwili część przedstawionych programów drukuje sprzeczne informacje (lub nie drukuje nic).
3. Ewentualnie doprowadzenie do sytuacji, w której do „algorytmu" wyszukiwania jest tylko jedno wejście i jedno wyjście.

2. Zadania do wykonania

1. Wydzielenie algorytmu wyszukiwania binarnego jako osobnej funkcji, o przykładowym wywołaniu:

   ```
   int bin_szuk(int X, int N, int dane[N]);
   ```

   gdzie:
   — X — szukana wartość,
   — N — długość tablicy danych,
   — dane — tablica typu **int** zawierające uporządkowane dane.
   Funkcja zwraca wartości:

— $i \in \{0, 1, \ldots, N-1\}$ — znaleziono wartość na pozycji i,
— $i < 0$ — wartości nie znaleziono.
Dodatkowo, zwracana wartość może zawierać zakodowaną informację gdzie należałoby wstawić wartość. Tu można zakodować numer przedziału, którym należy umieścić wartość:
— przed zerowym elementem
— między zerowym, a pierwszym,
— ...
— między $N-2$ a $N-1$,
— na miejscu N (po $N-1$).

2. Kolejnym etapem jest taka modyfikacja funkcji, aby **nieznalezioną** wartość dopisywała do tabeli w takim miejscu aby dane ciągle były uporządkowane rosnąco (malejąco).
Można to zrobić na dwa sposoby.
— Oprócz tablicy z danymi (załóżmy, że nazywa się ona T i ma długość N deklarujemy tablicę T1 (tego samego typu co T) o długości N + 1.
Tablice T i T1 muszą być parametrami funkcji. Gdy wartość trzeba dodać przed wyjściem z funkcji dokonuje się odpowiedniego przepisania danych.
— Drugi sposób jest nieco bardziej zagmatwany, ale ładniejszy. Skorzystamy z pamięci dynamicznej oraz funkcji malloc() oraz realloc().
Funkcja malloc() wywoływana jest w sposób następujący:

```
#include <stdlib.h>
...
int * T; // To jest nasza tablica
int N;   // W N jest rozmiar tablicy
...
T = malloc(N * sizeof(int)); // Prosba o przydzial pamieci
if (T == 0) // Gdy pamiec nie może byc przydzielona
            // funkcja zwróci wartosc 0
{
    printf("Brak pamieci!\n");
    return -1;
}
```

Funkcja zwraca adres (wskaźnik) początku obszaru pamięci przydzielonego tablicy z danymi. Użyjemy jej w funkcji main(). Niestety tablicom dynamiczny nie można nadawać początkowych wartości tak jak automatycznym. Nie jest to wielkim problemem, gdyż nasz program powinien teraz poradzić sobie z tablicą jednoelementową lub pustą!
Gdy trzeba rozmiar tablicy zwiększyć używamy funkcji realloc. Używamy jej tak:

```
int * T1; // To bedzie ,,nowa'' tablica
...
T1 = realloc(T, (N + 1) * sizeof(int))
// T to ,,stara'' tablica, a N + 1 to potrzebny rozmiar ,,nowej''
if (T1 == 0)
// Oznacza to, ze ne udalo sie zwiekszyc rozmiaru tablicy
{
    printf("Brak pamieci!\nEmergency exit");
    return -1;
}
else
{
    T = T1;
}
...
```

Używamy dodatkowej zmiennej T1 żeby w sytuacji, kiedy nie uda się przydzielić pamięci (realloc() zwraca zero!) nie stracić danych!

Politechnika Wrocławska

Wyszukiwanie ze wstawianiem

Wojciech Myszka

Katedra Mechaniki, Inżynierii Materiałowej i Biomedycznej

2022-04-03 10:39:30 +0200

HR EXCELLENCE IN RESEARCH

Ogólny schemat

1. Zaczynamy z **pustą** tablicą ($N = 0$).
2. Gdy przychodzi pierwsza liczba zwiększamy tablicę do rozmiaru 1 i dodajemy liczbę.
3. Gdy przychodzi kolejna liczba, sprawdzamy czy jest już w tablicy
   - jeżeli TAK przechodzimy do punktu 3
   - jeżeli NIE
     - 3.1 zwiększamy rozmiar tablicy o 1
     - 3.2 dodajemy element
     - 3.3 zwiększamy N o 1 ($N = N + 1$)
     - 3.4 drukujemy tablicę

   przechodzimy do punktu 3

Do rozstrzygnięcia pozostaje jak przerwać program. Proponuję, żeby założyć, że wszystkie wartości są dodatnie, jak przyjdzie wartość ujemna — zatrzymujemy pracę programu.

Odczyt liczby

Do wprowadzania użyjemy funkcji scanf():

```
int x;
// ...
scanf("%d", &x);
if (x < 0)
{
    printf("Koniec programu!");
    return 0;
}
```

Wyszukiwanie

1. W większości przypadków Państwa programy były OK.
2. Zadbać trzeba o to, żeby funkcja zwracała
   - wartość dodatnią lub zero gdy liczba została znaleziona (oznaczająca miejsce, na którym liczba się znajduje)
   - wartość ujemną określającą pozycję, na której liczba ma być dodana, według kodu:
     -1 — miejsce zerowe
     -2 — miejsce pierwsze
     ...
     $-n$ — wartość ma być wstawiona na miejscu $n-1$

Uwaga: Można oczywiście użyć innego sposobu kodowania!

„Zaczynamy z pustą tablicą”

```
int N = 0;
int * dane;
```

1. Korzystamy ze wskaźników (tylko to pozwala na zwiększanie długości tablicy)

„Zwiększamy tablicę do rozmiaru 1...”

```
dane = malloc((N + 1) * sizeof(int));
if (dane == 0)
{
    printf("Blad! Nie dostalem pamieci!\n");
    return 1;
}
N = N + 1;
```

„Zwiększamy rozmiar tablicy o 1”

```
int * dane1;
dane1 = realloc(dane, (N + 1) * sizeof(int));
if (dane1 == 0)
{
    printf("Blad! Nie dostalem pamieci!\n");
    return 1;
}
dane = dane1;
```

Użycie nowej zmiennej (dane1) ma sens, gdyż, w sytuacji gdy nie dostaniemy pamięci — funkcja realloc() zwraca wartość zero. Mamy szanse coś z dotychczas zebranymi danymi zrobić. Tu program kończy pracę.

Dodawanie liczby I

- Zakładam, że tablica ma wystarczającą długość (została „przedłużona” przed wywołaniem funkcji)
- Wiemy ile jest liczb w (starej) tablicy (N)
- Mamy informację na którym miejscu liczbę dodać (k)
  - gdy $k < N$ — „gdzieś między liczby w tablicy”
  - gdy $k == N$ — jako ostatnią wartość

Podczas wstawiania trzeba będzie (czasami) liczby w tablicy przesuwać!
Prototyp funkcji

```
void wstaw(int x, int k, int N, int * T);
// x wstawiana liczba; k miejsce wstawiania
// N dlugosc tablicy;  T tablica danych
```

Dodawanie liczby II

Algorytm:

1. Gdy $k == N$ liczbę trzeba wstawić na końcu (nie trzeba przesuwać danych)
   - `T[k] = x;`
   - koniec
2. W przeciwnym razie trzeba przesunąć wszystkie wartości o indeksach większych niż k „w prawo” (najlepiej robić to od końca)

   ```
   for(int i = N; i > k; i = i - 1)
       T[i] = T[i - 1];
   T[k] = x;
   ```

3. Koniec

Drukowanie tablicy

Łatwizna.
Prototyp funkcji:

```
void drukuj(int N, int * T);
```

3. Wersja PDF tego dokumentu...

...pod adresem.

Wersja: 61 z drobnymi modyfikacjami! data ostatniej modyfikacji 2022-04-03 10:39:40 +0200