Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Лабораторная работа № 5

по дисциплине ОАиП

Вариант 10

Выполнил студент гр. 150502: Снитко Д.А

Проверил: Дулько П.А.

Минск 2022

**1. Постановка задачи**

Преобразовать двунаправленную очередь в сбалансированное бинарное дерево.

**2. Исходный код**

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

// узел двунаправленного списка, который также будет использоваться как узел дерева

struct queue

{

int data;

// указатель правого поддерева

struct queue\* next;

// указатель левого поддерева

struct queue\* prev;

};

// функция для подсчета узлов в связанном списке

int countNodes(struct queue\* head)

{

int count = 0;

struct queue\* temp = head;

while (temp)

{

temp = temp->next;

count++;

}

return count;

}

/\* Основная функция, которая строит сбалансированное СБ и возвращает его корень.

head\_ref --> Указатель на указатель на головной узел двусвязного списка

n --> Количество узлов в двусвязном списке \*/

struct queue\* sortedListToBSTRecur(struct queue\*\* head\_ref, int n)

{

if (n <= 0)

return NULL;

// Рекурсивно построить левое поддерево

struct queue\* left = sortedListToBSTRecur(head\_ref, n / 2);

// head\_ref теперь относится к среднему узлу, сделать средний узел корневым СБ

struct queue\* root = \*head\_ref;

// Установить указатель на левое поддерево

root->prev = left;

// Изменить заголовок связанного списка для родительских рекурсивных вызовов

\*head\_ref = (\*head\_ref)->next;

/\* Рекурсивно строим правое поддерево и связываем его с корнем

Количество узлов в правом поддереве равно общему количеству узлов - узлов в

левое поддерево - 1 \*/

root->next = sortedListToBSTRecur(head\_ref, n - n / 2 - 1);

return root;

}

/\* функция подсчитывает количество узлов в списке, а затем вызывает sortedListToBSTRecur() для построения СБ \*/

struct queue\* sortedListToBST(struct queue\* head)

{

//Подсчет количества узлов в связанном списке

int n = countNodes(head);

// Построение СБ

return sortedListToBSTRecur(&head, n);

}

// инициализация

void add(struct queue\*\* pt, struct queue\*\* ph,int num)

{

struct queue\* n;

if (!(n = (struct queue\*)calloc(1, sizeof(struct queue))))

{

printf("Нет свободной памяти");

return;

}

n->data=num;

if (!\*pt || !\*ph)

{

\*pt = n;

\*ph = n;

}

else

{

n->next = \*pt;

(\*pt)->prev = n;

\*pt = n;

}

fflush(stdin);

}

// функция для вывода узлов в очереди

void printList(struct queue\* node)

{

while (node != NULL)

{

printf("%d ", node->data);

node = node->next;

}

}

void print(struct queue\* dr, int level)

{

if (dr) print(dr->prev, level + 1);

for (int n = 0; n < level; ++n) printf(" ");

if (dr) printf("(%d)\n", dr->data);

if (dr) print(dr->next, level + 1);

}

int main()

{

setlocale(LC\_ALL, "Russian");

struct queue\* head, \*tail;

head = tail = NULL;

// cоздадим отсортированный связанный список состоящий из случайных чисел

for (int i = 1; i < 100; i+=rand()%20)

add(&head, &tail, i);

printf("Исходная очередь\n");

printList(head);

//преобразование списка в СБ

struct queue\* root = sortedListToBST(head);

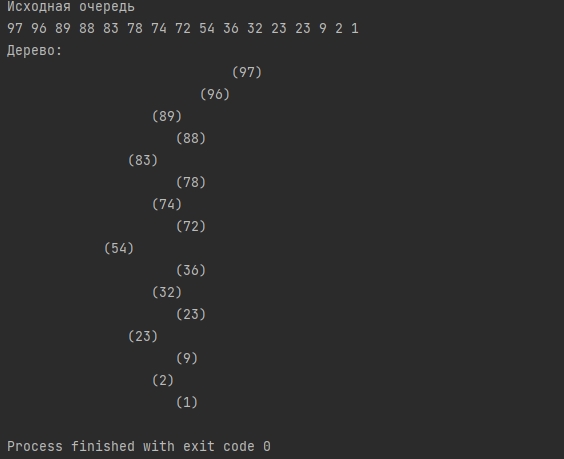
printf("\nДерево: \n ");

print(root, 0);

return 0;

}

**3. Демонстрация работы**

****