Цой Олег Дмитриевич.

Даты выполнения: 09.07.2022

Время выполнения: 5 часов

1. Напишите функцию, которая принимает на вход знаковое целое число и печатает его двоичное представление, не используя библиотечных классов или функций.

// вариант, где будет печать в прямой форме

template <typename Int>  
void printIntegerInBinary(Int num) {  
 const size\_t bitsInByte = 8;  
 const size\_t count = bitsInByte \* sizeof(Int);  
  
 int result[count] = {};  
 Int degree = 1;  
 const size\_t last\_idx = count - 1u;  
  
 if (num < 0) {  
 num \*= (-1);  
 result[last\_idx] = 1;  
 }  
  
 for (size\_t i = 0u; i < last\_idx; ++i) {  
 const Int bit = (num & degree) >> i;  
 result[i] = bit;  
 degree = degree << 1;  
 }  
  
 for (int i = last\_idx; i >= 0; --i) {  
 printf("%d", result[i]);  
 }  
}

2. Напишите функцию, удаляющую последовательно дублирующиеся символы в строке:

void RemoveDups(char\* str) {  
 if (str[0] == '\0') {  
 return;  
 }  
  
 auto pattern = str[0];  
 size\_t write\_idx = 1u;  
 for (size\_t i = 1u; str[i] != '\0'; ++i) {  
 if (str[i] != pattern) {  
 str[write\_idx] = str[i];  
 pattern = str[i];  
 ++write\_idx;  
 }  
 }  
 str[write\_idx] = '\0';  
}

3. Реализуйте функции сериализации и десериализации двусвязного списка в бинарном формате в файл. Алгоритмическая сложность решения должна быть меньше квадратичной.

#include <vector>  
#include <unordered\_map>  
#include <string>  
  
// структуру ListNode модифицировать нельзя  
struct ListNode {  
 ListNode \* prev;  
 ListNode \* next;  
 ListNode \* rand; // указатель на произвольный элемент данного списка, либо NULL  
 std::string data;  
};  
  
class List {  
public:  
 List()  
 : head(nullptr)  
 , tail(nullptr)  
 , count(0)  
 {}  
  
 void Serialize (FILE \* file); // сохранение в файл (файл открыт с помощью fopen(path, "wb"))  
 void Deserialize (FILE \* file); // загрузка из файла (файл открыт с помощью fopen(path, "rb"))  
  
private:  
 using NodesIds = std::unordered\_map<ListNode \* , size\_t>;  
  
 ListNode \* head;  
 ListNode \* tail;  
 int count;  
  
 /\*\*  
 \* @brief Сериализует узел листа из файла.  
 \* @details Поле узла, отвечающее за случайный узел, будет сериализоваться как идентификатор  
 \* этого случайного узла.  
 \* @param file Файл, в который будет происходить сериализация.  
 \* @param node Узел, который будет сериализован.  
 \* @param randId Идентификатор случайного узла.  
 \*/  
 void SerializeListNode(FILE \* file, ListNode \* node, size\_t randId) const;  
  
 /\*\*  
 \* @brief Десериализует узел листа из файла.  
 \* @param node Узел листа, в который будет произведена десериализация.  
 \* @param nodes Массив узлов листа.  
 \*/  
 void DeserializeListNode(FILE \* file, ListNode \* node) const;  
  
 /\*\*  
 \* @brief Создает хеш-таблицу сопоставляющую идентификатор узла к самому узлу.  
 \* @details Идентификатор начинается с 1.  
 \* @return хеш-таблицу: id -> node.  
 \*/  
 NodesIds makeNodesIds() const;  
};  
  
void List::Serialize(FILE \*const file) {  
 const NodesIds nodesIds = makeNodesIds();  
  
 // будем считать что запись проходит без ошибок  
 fwrite(&count, sizeof(int), 1u, file);  
 for (ListNode \* curNode = head; curNode != nullptr; curNode = curNode->next) {  
 const size\_t randId = (curNode->rand ? nodesIds.at(curNode->rand) : 0u);  
 SerializeListNode(file, curNode, randId);  
 }  
}  
  
void List::SerializeListNode(FILE \*const file, ListNode \*const node,  
 const size\_t randId) const {  
 fwrite(&randId, sizeof(size\_t), 1u, file);  
  
 const size\_t data\_size = node->data.size();  
 fwrite(&data\_size, sizeof(size\_t), 1u, file);  
  
 for (const auto c : node->data) {  
 fwrite(&c, sizeof(std::string::value\_type), 1u, file);  
 }  
}  
  
List::NodesIds List::makeNodesIds() const {  
 NodesIds nodes;  
 nodes.reserve(count);  
  
 size\_t id = 0u;  
 for (ListNode \* curNode = head; curNode != nullptr; curNode = curNode->next) {  
 nodes[curNode] = ++id;  
 }  
 return nodes;  
}  
  
void List::Deserialize(FILE \*const file) {  
 std::unordered\_map<size\_t, ListNode \*> idsToNodes;  
 idsToNodes[0] = nullptr;  
  
 fread(&count, sizeof(int), 1, file);  
 const auto size = static\_cast<size\_t>(count);  
 idsToNodes.reserve(size);  
 std::vector<size\_t> nodesRandIds;  
 nodesRandIds.reserve(size);  
  
 for (size\_t i = 1u; i <= size; ++i) {  
 idsToNodes[i] = new ListNode{};  
 auto \*const node = idsToNodes[i];  
  
 size\_t randId = 0u;  
 fread(&randId, sizeof(size\_t), 1u, file);  
 nodesRandIds.push\_back(randId);  
  
 DeserializeListNode(file, node);  
  
 auto \*const prevNode = idsToNodes[i - 1u];  
 if (prevNode) {  
 prevNode->next = node;  
 }  
 node->prev = prevNode;  
 }  
  
 for (size\_t i = 0u; i < size; ++i) {  
 const size\_t randId = nodesRandIds[i];  
 idsToNodes[i + 1u]->rand = idsToNodes[randId];  
 }  
  
 head = idsToNodes[1u];  
 tail = idsToNodes[size];  
}  
  
void List::DeserializeListNode(FILE \*const file, ListNode \*const node) const {  
 size\_t str\_size = 0u;  
 fread(&str\_size, sizeof(size\_t), 1u, file);  
 for (size\_t i = 0u; i < str\_size; ++i) {  
 std::string::value\_type c = '\0';  
 fread(&c, sizeof(std::string::value\_type), 1u, file);  
 node->data.push\_back(c);  
 }  
}