**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**“Московский государственный университет геодезии и картографии” (МИИГАИК)**

**Факультет геоинформатики и информационной безопасности**

**Кафедра геоинформационных систем и технологий**

**Лабораторная работа №7**

**"Методы решения алгоритмических задач"**

**Преподаватель: Лебедев Евгений Денисович**

**Работу выполнил: Зазыкин Иван Дмитриевич**

**Группа: 2024-ФГИИБ-ПИ-1б**

**Вариант: 11**

**Москва 2025**

**Раздел 1. Задача на деревья**

**2.1. Описание подхода к решению**

**Дерево** — это одна из базовых структур данных в информатике, представляющая собой связный ациклический граф. Это иерархическая структура, состоящая из **узлов** (вершин), соединенных **рёбрами**. У дерева есть один выделенный узел — **корень**. Каждый узел, кроме корня, имеет ровно одного "родителя" и может иметь ноль или более "детей". Узлы, у которых нет детей, называются **листьями**.

**Применение:** Деревья удобно применять для задач, где есть иерархия или вложенность. Например, для представления файловой системы, структуры HTML-документа (DOM), синтаксического разбора выражений в компиляторах, а также для эффективного поиска и хранения данных (например, в двоичных деревьях поиска).

**2.2. Формулировка задания (Задача №5 "Числовые пути")**

Вася и его друзья решили поиграть в игру. Дано дерево, в узлах которого записаны цифры от 0 до 9. Таким образом, каждый путь от корня до листа содержит число, получившееся конкатенацией цифр пути (склеиванием цифр пути в одно число). Нужно найти сумму всех таких чисел в дереве. Гарантируется, что ответ не превосходит 20 000.

**2.3. Примеры тестовых входных и выходных данных**

* **Пример 1:**
  + Дерево: корень 1, дети 3 и 5.
  + Пути: 1 -> 3 (число 13), 1 -> 5 (число 15).
  + Результат: 13 + 15 = 28.
* **Пример 2:**
  + Дерево: корень 1, дети 2 и 3. У узла 3 дети 2 и 1.
  + Пути: 1 -> 2 (12), 1 -> 3 -> 2 (132), 1 -> 3 -> 1 (131).
  + Результат: 12 + 132 + 131 = 275.

**2.4. Ссылка на GitHub-репозиторий с кодом**

https://github.com/re-side/Inf\_university/blob/main/ALG/lab7/Tree.cpp

**2.5. Описание решения задачи**

Для решения задачи используется рекурсивный обход дерева в глубину (DFS). Создается вспомогательная функция, которая принимает два аргумента: текущий узел и число, составленное на пути от корня до родителя этого узла.

1. При посещении каждого узла его значение (цифра) добавляется в конец текущего числа путем умножения числа на 10 и прибавления значения узла.
2. Если функция доходит до листового узла (у которого нет потомков), это означает, что один полный путь от корня до листа завершен. Функция возвращает итоговое число, собранное на этом пути.
3. Если узел не является листовым, функция рекурсивно вызывается для его левого и правого дочерних узлов. Результаты этих вызовов (суммы путей в поддеревьях) суммируются и возвращаются наверх.

**2.6. Листинг решения**

**Раздел 2. Динамическое программирование**

**2.1. Описание подхода к решению**

**Динамическое программирование (ДП)** — это метод решения сложных задач путем их разбиения на более простые, перекрывающиеся подзадачи. Ключевая идея состоит в том, чтобы решать каждую подзадачу только один раз и сохранять ее решение (мемоизация). Когда та же подзадача встречается снова, используется уже сохраненный результат, что позволяет избежать повторных вычислений.

**Применение:** ДП особенно эффективно для оптимизационных задач, где нужно найти минимальное или максимальное значение, или для комбинаторных задач, где нужно посчитать количество способов чего-либо. Примеры: задача о рюкзаке, поиск наибольшей общей подпоследовательности, вычисление чисел Фибоначчи.

**2.2. Формулировка задания (Задача №11 "Одинаковые суммы")**

На Алгосах устроили турнир по настольному теннису. Гоша выиграл n партий, получив при этом некоторое количество очков за каждую из них. Гоше стало интересно, можно ли разбить все заработанные им во время турнира очки на две части так, чтобы сумма в них была одинаковой.

**2.3. Примеры тестовых входных и выходных данных**

Пример 1

4 true

1 5 7 1

Пример 2

3 false

2 10 9

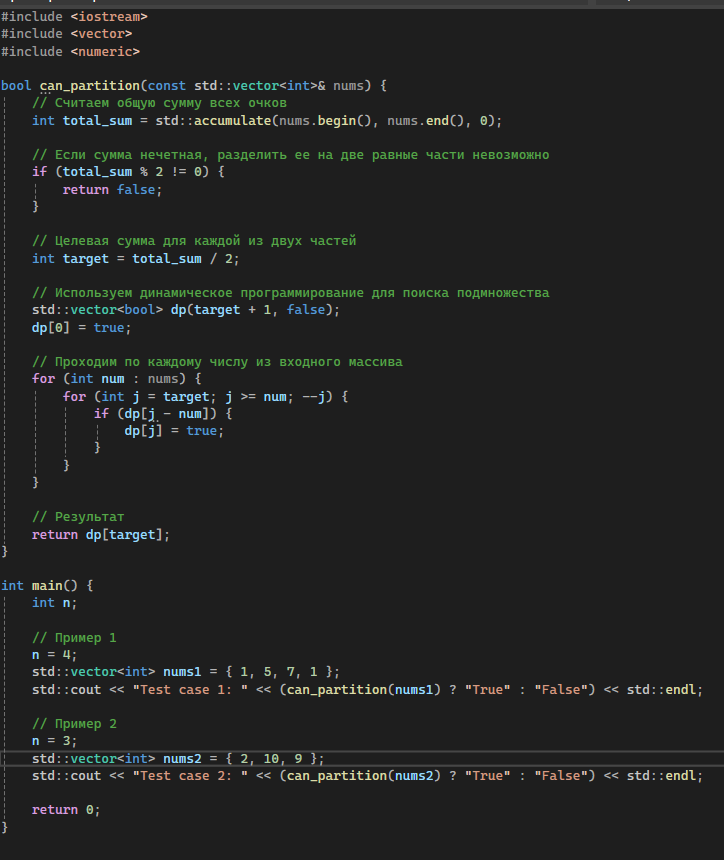
**2.4. Ссылка на GitHub-репозиторий с работающим кодом**

https://github.com/re-side/Inf\_university/blob/main/ALG/lab7/Dynamic.cpp

**2.5. Описание решения задачи**

Задача сводится к классической задаче о разбиении множества.

1. Сначала вычисляется общая сумма всех очков. Если она нечетная, разделить ее на две равные части невозможно, поэтому функция возвращает False.
2. Если сумма четная, целевая сумма для каждой части будет равна total\_sum / 2. Задача теперь в том, чтобы найти подмножество очков, сумма которых равна этой целевой сумме.
3. Для этого используется динамическое программирование. Создается булев массив dp размером target + 1, где dp[i] означает, можно ли набрать сумму i. dp[0] устанавливается в true.
4. Далее, для каждого числа из входного массива очков, мы итеративно обновляем dp, отмечая новые суммы, которые становятся достижимыми.

**2.6. Листинг решения**

**Раздел 3. Решение задачи методом декомпозиции**

**2.1. Описание подхода к решению**

**"Разделяй и властвуй"**— это парадигма проектирования алгоритмов, которая состоит из трёх шагов:

1. **Разделяй:** Исходная задача разбивается на несколько меньших подзадач того же типа.
2. **Властвуй:** Подзадачи рекурсивно решаются. Если они достаточно малы, они решаются напрямую.
3. **Соединяй:** Решения подзадач объединяются для получения решения исходной задачи.

**Применение:** Этот метод лежит в основе многих эффективных алгоритмов, таких как сортировка слиянием, быстрая сортировка и бинарный поиск.

**2.2. Формулировка задания (Задача №7 "Поиск пика")**

В Алгосах появился таинственный массив, в котором нужно найти пиковый элемент — такой, который не меньше своих соседей. Используйте алгоритм бинарного поиска (вариант «разделяй и властвуй») для эффективного поиска. Гарантируется, что существует хотя бы один пик.

**2.3. Примеры тестовых входных и выходных данных**

* **Пример 1:**
  + Вход: 6, 1 3 20 4 1 0
  + Вывод: 20

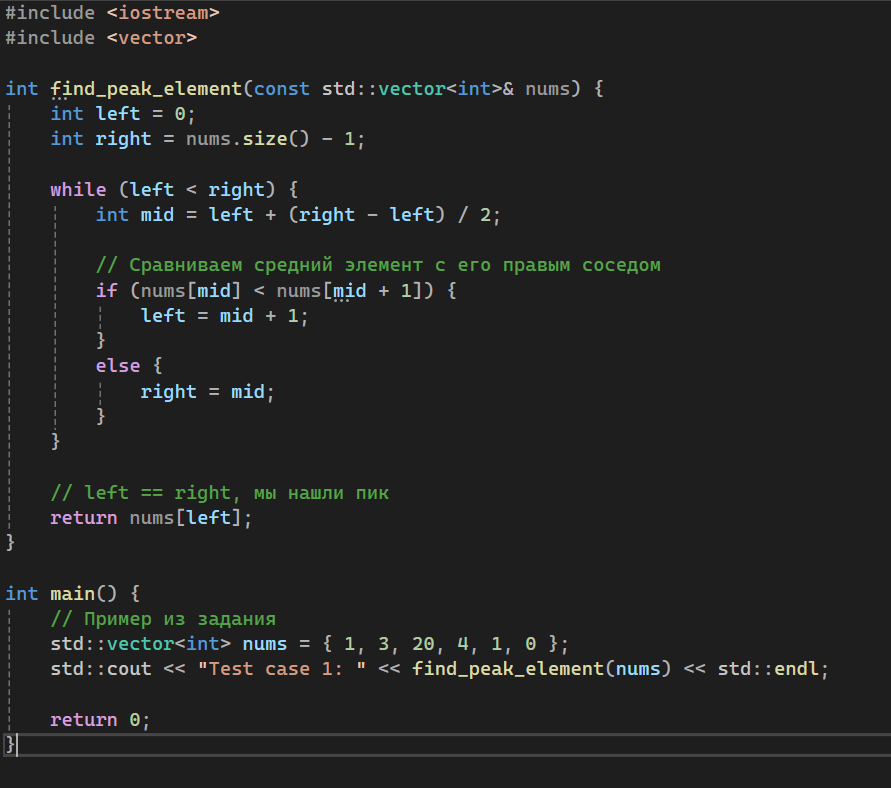
**2.4. Ссылка на GitHub-репозиторий с работающим кодом**

<https://github.com/re-side/Inf_university/blob/main/ALG/lab7/Divide.cpp>

**2.5. Описание решения задачи**

Для поиска пикового элемента применяется алгоритм, основанный на бинарном поиске, что является классическим примером метода "разделяй и властвуй".

1. Определяются две границы поиска: left и right, изначально равные первому и последнему индексам массива.
2. В цикле вычисляется средний индекс mid.
3. Элемент nums[mid] сравнивается с его правым соседом nums[mid + 1].
   * Если nums[mid] < nums[mid + 1], значит, мы на "восходящем склоне", и пик точно находится правее. Левая граница сдвигается на mid + 1.
   * В противном случае, мы либо на пике, либо на "нисходящем склоне", и пик находится в левой части, включая mid. Правая граница сдвигается на mid.
4. Цикл сужает интервал поиска вдвое на каждой итерации, пока left и right не сойдутся. Итоговый элемент и будет пиковым.

**2.6. Листинг решения**

**Раздел 4. Решение задачи методом поиска с возвратом**

**2.1. Описание подхода к решению**

**Поиск с возвратом (Backtracking)** — это общая алгоритмическая техника для нахождения всех решений некоторой вычислительной проблемы. Алгоритм постепенно строит кандидатов в решения и отказывается от кандидата ("возвращается назад"), как только определяет, что он не может быть достроен до верного решения.

**Применение:** Этот метод является "умным" полным перебором, так как отсекает заведомо проигрышные ветви дерева поиска. Он часто используется в задачах, связанных с комбинаторикой, например, в решении судоку, задаче о восьми ферзях, и для генерации всех перестановок или комбинаций.

**2.2. Формулировка задания (Задача №2 "Разбиение числа")**

Гоша любит числа. Он хочет узнать, сколькими способами можно представить число n как сумму положительных целых чисел, в порядке неубывания. Используйте поиск с возвратом для генерации всех разбиений.

**2.3. Примеры тестовых входных и выходных данных**

* **Пример 1:**
  + Вход: 4
  + Вывод:

1 1 1 1

1 1 2

1 3

2 2

4

**2.4. Ссылка на GitHub-репозиторий с работающим кодом**

<https://github.com/re-side/Inf_university/blob/main/ALG/lab7/Backtracking.cpp>

**2.5. Описание решения задачи**

Для генерации всех разбиений числа n на слагаемые используется рекурсивный алгоритм с возвратом.

1. Создается рекурсивная функция, принимающая параметры: target (оставшаяся сумма), start\_num (минимальное слагаемое на текущем шаге) и current\_partition (вектор с текущей комбинацией).
2. **Базовый случай:** Если target равен 0, значит, найдена корректная комбинация слагаемых, и она выводится на печать.
3. **Рекурсивный шаг:** Функция перебирает все возможные слагаемые i, начиная от start\_num.
   * Число i добавляется в текущее разбиение.
   * Запускается рекурсивный вызов для оставшейся суммы target - i. В новый вызов передается i в качестве start\_num, чтобы все последующие слагаемые были не меньше текущего, что обеспечивает порядок и уникальность комбинаций.
   * После возврата из рекурсии, последнее добавленное число удаляется из разбиения — это и есть шаг "возврата", позволяющий исследовать другие варианты.

**2.6. Листинг решения**

