Отчёт по лабораторной работе №2. Комбинаторные алгоритмы решения оптимизационных задач

Выполнил: Алексей Шимко

Задание №1. Разобрать и разработать генератор подмножеств заданного множества. Исходный код:

namespace Combinatorics

{

public class Subsets

{

public static List<List<string>> getSubsets(string[] elements)

{

List<List<string>> subsets = new List<List<string>>();

int n = elements.Length;

for (int i = 0; i < (1 << n); i++)

{

List<string> subset = new List<string>();

for (int j = 0; j < n; j++)

{

if ((i & (1 << j)) != 0)

{

subset.Add(elements[j]);

}

}

subsets.Add(subset);

}

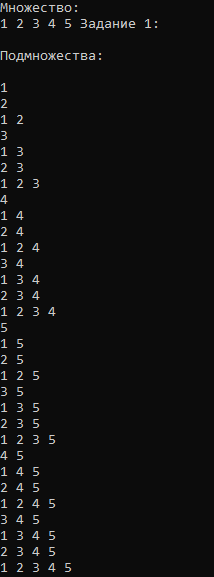
return subsets;

}

}

}

Результат выполнения:



Задание №2. Разобрать и разработать генератор сочетаний.

Исходный код:

namespace Combinatorics

{

public static class Combinations

{

public static List<List<string>> GenerateCombinations(string[] elements, int k)

{

List<List<string>> combinations = new List<List<string>>();

int n = elements.Length;

void Generate(int start, List<string> currentCombination)

{

if (currentCombination.Count == k)

{

combinations.Add(new List<string>(currentCombination));

return;

}

for (int i = start; i < n; i++)

{

currentCombination.Add(elements[i]);

Generate(i + 1, currentCombination);

currentCombination.RemoveAt(currentCombination.Count - 1);

}

}

Generate(0, new List<string>());

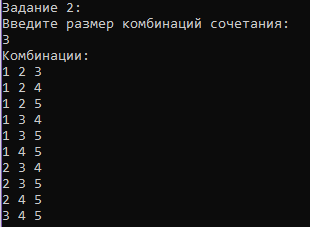
return combinations;

}

}

}

Результат:



Задание №3. Разобрать и разработать генератор перестановок.

Исходный код:

using System;

using System.Collections.Generic;

namespace Combi

{

public class Accommodation

{

private List<List<int>> arrangements;

private List<int> elements;

private int currentArrangementIndex;

public int Na { get; private set; }

public Accommodation(List<int> elements, int m)

{

if (elements == null || m <= 0 || m > elements.Count)

{

throw new ArgumentException("Invalid arguments");

}

this.elements = elements;

arrangements = new List<List<int>>();

currentArrangementIndex = -1;

GenerateArrangements(m);

Reset();

}

private void GenerateArrangements(int m)

{

arrangements.Clear();

GenerateArrangements(new List<int>(), m);

}

private void GenerateArrangements(List<int> currentArrangement, int remaining)

{

if (remaining == 0)

{

arrangements.Add(new List<int>(currentArrangement));

return;

}

for (int i = 0; i < elements.Count; i++)

{

if (!currentArrangement.Contains(elements[i]))

{

currentArrangement.Add(elements[i]);

GenerateArrangements(currentArrangement, remaining - 1);

currentArrangement.RemoveAt(currentArrangement.Count - 1);

}

}

}

public void Reset()

{

Na = 0;

currentArrangementIndex = -1;

}

public List<int> GetFirst()

{

if (arrangements.Count > 0)

{

currentArrangementIndex = 0;

Na = 0;

return arrangements[currentArrangementIndex];

}

return null;

}

public List<int> GetNext()

{

if (currentArrangementIndex < arrangements.Count - 1)

{

currentArrangementIndex++;

Na++;

return arrangements[currentArrangementIndex];

}

return null;

}

public int Count()

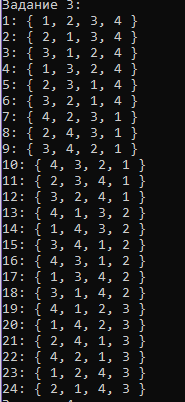
{

return arrangements.Count;

}

}

}

Результат:  


Задание №4. Разобрать и разработать генератор размещений.

Исходный код:

using System;

using System.Collections.Generic;

namespace Combinatorics

{

public class Permutation

{

private readonly List<int> elements;

private readonly List<List<int>> permutations;

private int currentPermutationIndex;

public Permutation(List<int> elements)

{

this.elements = elements ?? throw new ArgumentNullException(nameof(elements));

permutations = new List<List<int>>();

currentPermutationIndex = -1;

}

public void GeneratePermutations()

{

permutations.Clear();

GeneratePermutations(elements.Count);

currentPermutationIndex = -1;

}

private void GeneratePermutations(int n)

{

if (n == 1)

{

permutations.Add(new List<int>(elements));

return;

}

for (int i = 0; i < n; i++)

{

GeneratePermutations(n - 1);

if (n % 2 == 0)

{

Swap(i, n - 1);

}

else

{

Swap(0, n - 1);

}

}

}

private void Swap(int i, int j)

{

int temp = elements[i];

elements[i] = elements[j];

elements[j] = temp;

}

public List<int> GetNextPermutation()

{

if (currentPermutationIndex < permutations.Count - 1)

{

currentPermutationIndex++;

return permutations[currentPermutationIndex];

}

return default;

}

public void Reset()

{

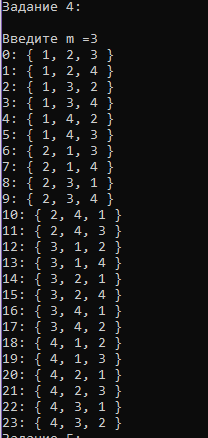
currentPermutationIndex = -1;

}

}

}

Результат:



Задание №5 . Решить в соответствии с вариантом задачу и результат занести в отчет (Вариант распределяется по списку):

Задача о рюкзаке.

Исходный код:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab\_2

{

public class KnapsackSolver

{

public static int SolveKnapsack(int capacity, int[] weights, int[] values)

{

int n = weights.Length;

List<List<int>> allSubsets = GetSubsets(n);

List<int> bestSubset = new List<int>();

int maxValue = 0;

foreach (var subsetIndices in allSubsets)

{

int subsetWeight = CalculateSubsetWeight(subsetIndices, weights);

if (subsetWeight <= capacity)

{

int subsetValue = CalculateSubsetValue(subsetIndices, values);

if (subsetValue > maxValue)

{

maxValue = subsetValue;

bestSubset = new List<int>(subsetIndices);

}

}

}

Console.WriteLine("Максимальный результат: {" + string.Join(", ", bestSubset) + "}");

Console.WriteLine("Максимальная стоимость: " + maxValue);

return maxValue;

}

private static List<List<int>> GetSubsets(int n)

{

List<List<int>> subsets = new List<List<int>>();

string[] elements = new string[n];

for (int i = 0; i < n; i++)

{

elements[i] = i.ToString();

}

List<List<string>> stringSubsets = Combinatorics.Subsets.getSubsets(elements);

foreach (var stringSubset in stringSubsets)

{

List<int> intSubset = new List<int>();

foreach (var element in stringSubset)

{

intSubset.Add(int.Parse(element));

}

subsets.Add(intSubset);

}

return subsets;

}

private static int CalculateSubsetWeight(List<int> subsetIndices, int[] weights)

{

int weight = 0;

foreach (int index in subsetIndices)

{

weight += weights[index];

}

return weight;

}

private static int CalculateSubsetValue(List<int> subsetIndices, int[] values)

{

int value = 0;

foreach (int index in subsetIndices)

{

value += values[index];

}

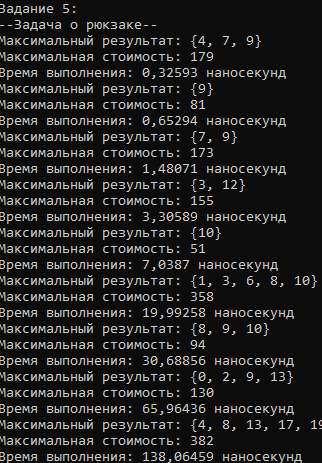
return value;

}

}

}

Результат:



Задание №6.

Исследовать зависимость времени вычисления необходимое для решения задачи (в соответствии с вариантом) от размерности задачи и результат в виде графика с небольшим пояснением занести в отчет:



Таким образом, проведя линию тренда мы можем сделать, что время выполнения алгоритма зависит от количество рассматриваемых элементов экспоненциально. Примечание: такой результат справедлив при решении задачи комбинаторным образом.