Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Кафедра ПМиК

Лабораторная работа 2

по дисциплине «Современные проблемы информатики»

Выполнил: ст. гр. ЗМП-41 Лёвкин И. А.

Проверил: Лихачёв А.В.

**Новосибирск 2025**

**Постановка задачи**

**Вариант 1.**

1. Разработать компьютерную программу решения задачи о коммивояжере методом полного перебора.
2. Разработать компьютерную программу решения задачи о коммивояжере оптимизированным методом, указанным преподавателем.
3. Результаты работы представить в виде зависимости от числа пунктов маршрута отношения времени решения задачи методом полного перебора ко времени её решения оптимизированным методом.

**Вариант 2.**

1. Разработать компьютерную программу решения задачи Джонсона для двух станков методом полного перебора.
2. Разработать компьютерную программу решения задачи Джонсона для двух станков алгоритмом Джонсона.
3. Результаты работы представить в виде зависимости от числа деталей отношения времени решения задачи методом полного перебора ко времени её решения алгоритмом Джонсона.

**Результаты:**

**Вариант 1.**

Вывод программы:

|  |
| --- |
| Города: 4  Полный перебор: 0.000020 сек  Held-Karp (DP): 0.000028 сек  Отношение времени перебор/динамическое программирование: 0.70  Города: 5  Полный перебор: 0.000027 сек  Held-Karp (DP): 0.000039 сек  Отношение времени перебор/динамическое программирование: 0.68  Города: 6  Полный перебор: 0.000122 сек  Held-Karp (DP): 0.000072 сек  Отношение времени перебор/динамическое программирование: 1.70  Города: 7  Полный перебор: 0.000731 сек  Held-Karp (DP): 0.000164 сек  Отношение времени перебор/динамическое программирование: 4.46  Города: 8  Полный перебор: 0.004574 сек  Held-Karp (DP): 0.000525 сек  Отношение времени перебор/динамическое программирование: 8.72  Города: 9  Полный перебор: 0.036676 сек  Held-Karp (DP): 0.000987 сек  Отношение времени перебор/динамическое программирование: 37.15  Города: 10  Полный перебор: 0.351454 сек  Held-Karp (DP): 0.002693 сек  Отношение времени перебор/динамическое программирование: 130.50  Города: 11  Полный перебор: пропущен (слишком долго)  Held-Karp (DP): 0.004528 сек  Города: 12  Полный перебор: пропущен (слишком долго)  Held-Karp (DP): 0.012136 сек  Города: 13  Полный перебор: пропущен (слишком долго)  Held-Karp (DP): 0.027218 сек  Города: 14  Полный перебор: пропущен (слишком долго)  Held-Karp (DP): 0.071902 сек  Города: 15  Полный перебор: пропущен (слишком долго)  Held-Karp (DP): 0.146836 сек  Города: 16  Полный перебор: пропущен (слишком долго)  Held-Karp (DP): 0.358237 сек |

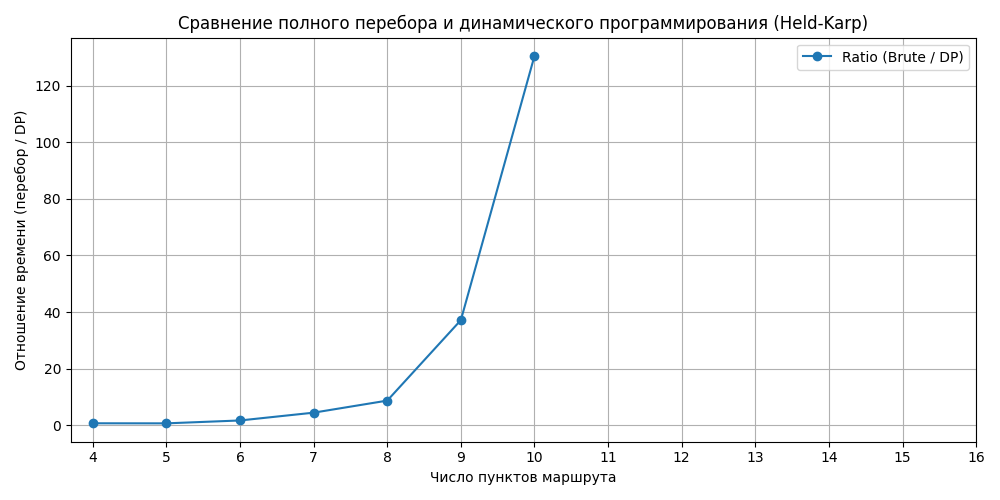


Рис. 1. Отношение метода перебора к методу динамического программирования

**Вариант 2.**

Вывод программы:

|  |
| --- |
| Деталей: 4  Полный перебор: 0.000016 сек  Алгоритм Джонсона: 0.000009 сек  Отношение времени перебор/алгоритм Джонсона: 1.74  Деталей: 5  Полный перебор: 0.000056 сек  Алгоритм Джонсона: 0.000006 сек  Отношение времени перебор/алгоритм Джонсона: 9.15  Деталей: 6  Полный перебор: 0.000332 сек  Алгоритм Джонсона: 0.000005 сек  Отношение времени перебор/алгоритм Джонсона: 72.07  Деталей: 7  Полный перебор: 0.002301 сек  Алгоритм Джонсона: 0.000004 сек  Отношение времени перебор/алгоритм Джонсона: 547.74  Деталей: 8  Полный перебор: 0.019535 сек  Алгоритм Джонсона: 0.000012 сек  Отношение времени перебор/алгоритм Джонсона: 1627.95  Деталей: 9  Полный перебор: 0.184292 сек  Алгоритм Джонсона: 0.000010 сек  Отношение времени перебор/алгоритм Джонсона: 17551.65  Деталей: 10  Полный перебор: 1.970357 сек  Алгоритм Джонсона: 0.000011 сек  Отношение времени перебор/алгоритм Джонсона: 177509.60  Деталей: 11  Полный перебор: 23.409337 сек  Алгоритм Джонсона: 0.000014 сек  Отношение времени перебор/алгоритм Джонсона: 1684125.05  Деталей: 12  Полный перебор: пропущен (слишком долго)  Алгоритм Джонсона: 0.000009 сек  Деталей: 13  Полный перебор: пропущен (слишком долго)  Алгоритм Джонсона: 0.000007 сек  Деталей: 14  Полный перебор: пропущен (слишком долго)  Алгоритм Джонсона: 0.000006 сек  Деталей: 15  Полный перебор: пропущен (слишком долго)  Алгоритм Джонсона: 0.000006 сек  Деталей: 16  Полный перебор: пропущен (слишком долго)  Алгоритм Джонсона: 0.000005 сек |

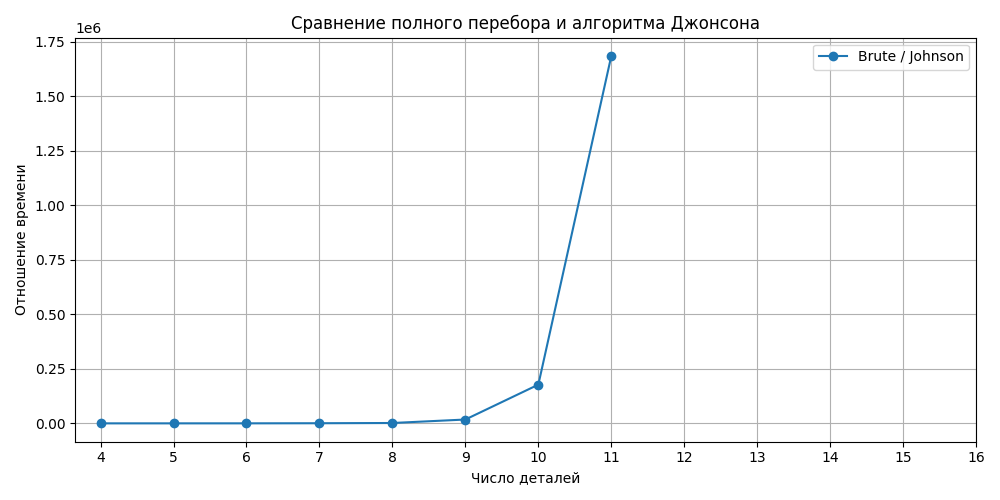


Рис. 2. Отношение метода перебора к алгоритму Джонсона

**Листинг**

**tsp.py**

import itertools

import time

import random

import matplotlib.pyplot as plt

def generate\_distance\_matrix(n, max\_dist=100):

return [

[random.randint(1, max\_dist) if i != j else 0 for j in range(n)]

for i in range(n)

]

def calc\_distance(route, dist\_matrix):

return (

sum(dist\_matrix[route[i]][route[i + 1]] for i in range(len(route) - 1))

+ dist\_matrix[route[-1]][route[0]]

)

def tsp\_brute\_force(dist\_matrix):

n = len(dist\_matrix)

min\_distance = float("inf")

best\_route = None

for perm in itertools.permutations(range(1, n)):

route = [0] + list(perm)

dist = calc\_distance(route, dist\_matrix)

if dist < min\_distance:

min\_distance = dist

best\_route = route

return best\_route + [0], min\_distance

def tsp\_held\_karp(dist\_matrix):

n = len(dist\_matrix)

memo = {}

for k in range(1, n):

memo[(1 << k, k)] = (dist\_matrix[0][k], 0)

for subset\_size in range(2, n):

for subset in itertools.combinations(range(1, n), subset\_size):

bits = sum(1 << k for k in subset)

for k in subset:

prev = bits & ~(1 << k)

res = []

for m in subset:

if m == k:

continue

res.append((memo[(prev, m)][0] + dist\_matrix[m][k], m))

memo[(bits, k)] = min(res)

bits = (1 << n) - 2

res = [(memo[(bits, k)][0] + dist\_matrix[k][0], k) for k in range(1, n)]

opt\_cost, parent = min(res)

path = [0]

bits = (1 << n) - 2

last = parent

for \_ in range(n - 1):

path.append(last)

new\_bits = bits & ~(1 << last)

\_, last = memo[(bits, last)]

bits = new\_bits

path.append(0)

return path, opt\_cost

def compare\_algorithms():

brute\_times = []

dp\_times = []

ratios = []

cities\_range = range(4, 17)

for n in cities\_range:

matrix = generate\_distance\_matrix(n)

print(f"\nГорода: {n}")

if n <= 10:

start = time.perf\_counter()

tsp\_brute\_force(matrix)

brute\_time = time.perf\_counter() - start

print(f" Полный перебор: {brute\_time:.6f} сек")

else:

brute\_time = None

print(f" Полный перебор: пропущен (слишком долго)")

start = time.perf\_counter()

tsp\_held\_karp(matrix)

dp\_time = time.perf\_counter() - start

print(f" Held-Karp (DP): {dp\_time:.6f} сек")

brute\_times.append(brute\_time)

dp\_times.append(dp\_time)

if brute\_time is not None:

ratio = brute\_time / dp\_time if dp\_time > 0 else float("inf")

print(

f" Отношение времени перебор/динамическое программирование: {ratio:.2f}"

)

else:

ratio = None

ratios.append(ratio)

plt.figure(figsize=(10, 5))

plt.plot(cities\_range, ratios, marker="o", label="Ratio (Brute / DP)")

plt.xlabel("Число пунктов маршрута")

plt.ylabel("Отношение времени (перебор / DP)")

plt.title("Сравнение полного перебора и динамического программирования (Held-Karp)")

plt.grid(True)

plt.xticks(cities\_range)

plt.legend()

plt.tight\_layout()

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

compare\_algorithms()

**johnson.py**

import itertools

import time

import random

import matplotlib.pyplot as plt

# Вычисление времени выполнения последовательности

def calc\_makespan(sequence, A, B):

    time\_A = 0

    time\_B = 0

    for i in sequence:

        time\_A += A[i]

        time\_B = max(time\_A, time\_B) + B[i]

    return time\_B

def johnson\_brute\_force(A, B):

    n = len(A)

    best\_seq = None

    best\_time = float("inf")

    for perm in itertools.permutations(range(n)):

        t = calc\_makespan(perm, A, B)

        if t < best\_time:

            best\_time = t

            best\_seq = perm

    return list(best\_seq), best\_time

def johnson\_algorithm(A, B):

    n = len(A)

    jobs = list(range(n))

    left = []

    right = []

    for job in jobs:

        if A[job] <= B[job]:

            left.append((A[job], job))

        else:

            right.append((B[job], job))

    left.sort()

    right.sort(reverse=True)

    sequence = [job for \_, job in left] + [job for \_, job in right]

    return sequence, calc\_makespan(sequence, A, B)

# Генерация случайных данных

def generate\_tasks(n, max\_time=20):

    A = [random.randint(1, max\_time) for \_ in range(n)]

    B = [random.randint(1, max\_time) for \_ in range(n)]

    return A, B

def compare\_johnson\_algorithms():

    brute\_times = []

    johnson\_times = []

    ratios = []

    n\_values = range(4, 17)

    for n in n\_values:

        A, B = generate\_tasks(n)

        print(f"\nДеталей: {n}")

        if n <= 11:

            start = time.perf\_counter()

            johnson\_brute\_force(A, B)

            t\_brute = time.perf\_counter() - start

            print(f"  Полный перебор: {t\_brute:.6f} сек")

        else:

            t\_brute = None

            print(f"  Полный перебор: пропущен (слишком долго)")

        start = time.perf\_counter()

        johnson\_algorithm(A, B)

        t\_johnson = time.perf\_counter() - start

        print(f"  Алгоритм Джонсона: {t\_johnson:.6f} сек")

        brute\_times.append(t\_brute)

        johnson\_times.append(t\_johnson)

        if t\_brute is not None:

            ratio = t\_brute / t\_johnson if t\_johnson > 0 else float("inf")

print(f" Отношение времени перебор/алгоритм Джонсона: {ratio:.2f}")

        else:

            ratio = None

        ratios.append(ratio)

    # Построение графика

    plt.figure(figsize=(10, 5))

    plt.plot(n\_values, ratios, marker="o", label="Brute / Johnson")

    plt.xlabel("Число деталей")

    plt.ylabel("Отношение времени")

    plt.title("Сравнение полного перебора и алгоритма Джонсона")

    plt.grid(True)

    plt.xticks(n\_values)

    plt.legend()

    plt.tight\_layout()

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    compare\_johnson\_algorithms()