Приближённые алгоритмы

Никита Гаевой (102, 106)
Иван Казменко (101, 103)
Лиана Хазалия (104, 105)

Санкт-Петербургский Государственный Университет

Понедельник, 28 марта 2022 года

Ссылка на задачи тренировки:
http://acm.math.spbu.ru/~gassa/bachelor-2021/220328_m21.pdf

Ссылка на эти слайды:
http://acm.math.spbu.ru/~gassa/bachelor-2021/220328_m21_slides.ru.pdf

Соответствие тем в слайдах и задач в тренировке:

1. Задача коммивояжёра	A. Задача коммивояжёра
2. Смежная задача	B. Путь и сливы
3. Задача «Студенты»	C. Студенты
4. Задача «Универсальный путь»	D. Универсальный путь
5. Задача «Игра с цифрами»	E. Игра с цифрами
6. Задача «Цифры в таблице»	F. Цифры в таблице

Содержание

Условие

Задача коммивояжёра

- Есть n городов и m двусторонних дорог.
- Для каждой дороги известна её длина.
- Мы начинаем в фиксированном городе.
- Мы хотим посетить каждый город ровно один раз.
- При этом мы хотим минимизировать суммарную длину пройденных дорог.

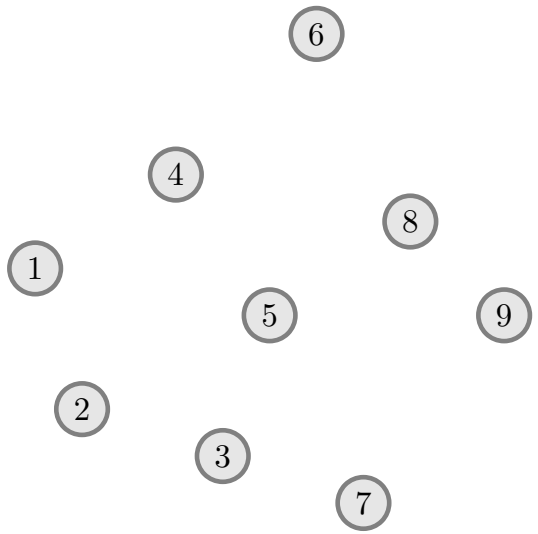
Условие

Метрическая задача коммивояжёра

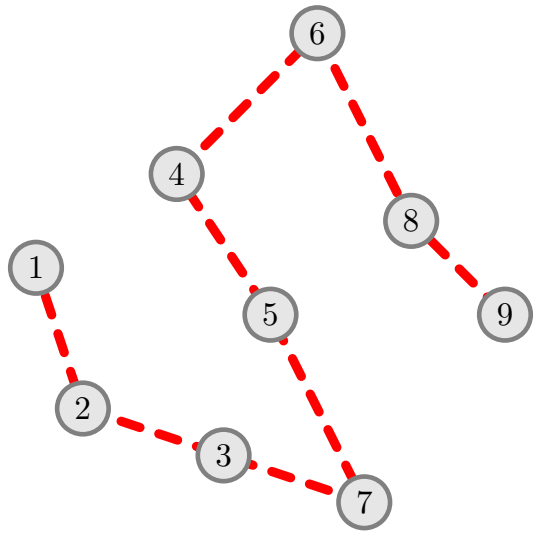
- Есть n городов — пусть это точки на плоскости.
- Между каждой парой городов есть дорога.
- Её длина — расстояние между городами.
- Мы начинаем в фиксированном городе.
- Мы хотим посетить каждый город ровно один раз.
- При этом мы хотим минимизировать суммарную длину пройденных дорог.

Пример

Пример



n=9

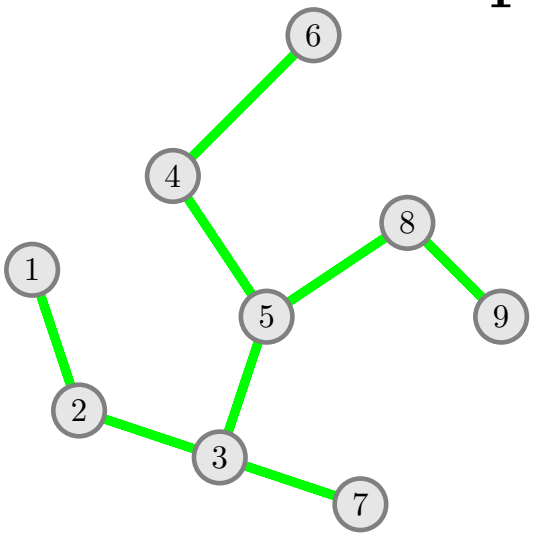


B=29.1077

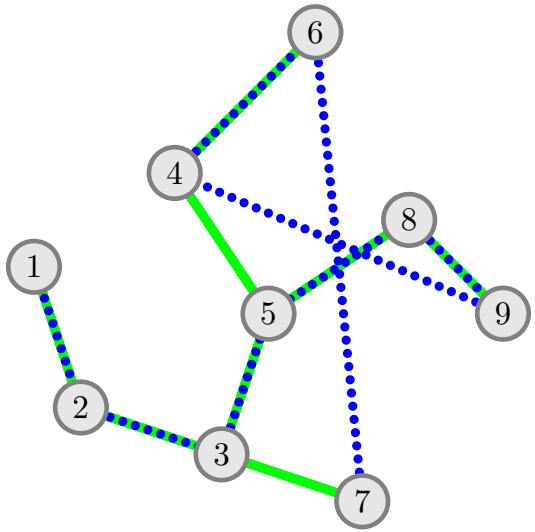
- Мы начинаем в городе 1.
- Наилучший путь имеет длину B и обозначен красной линией.
- Найти наилучший путь сложно — попробуем вместо этого найти какое-нибудь решение, про которое получится доказать, что оно достаточно хорошее.

Приближённый алгоритм

2-приближение



T=26.93129

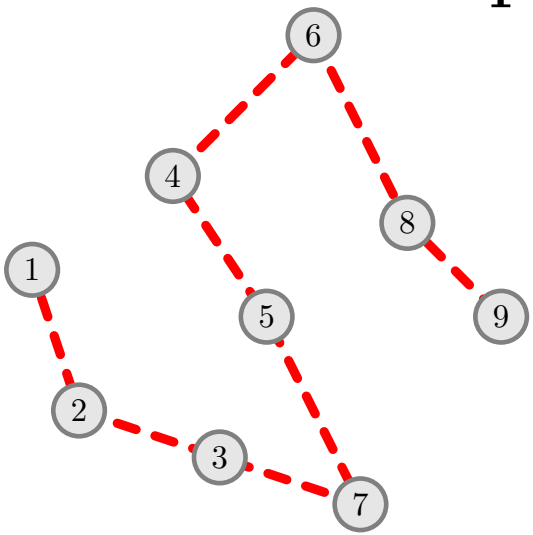


D=37.82912

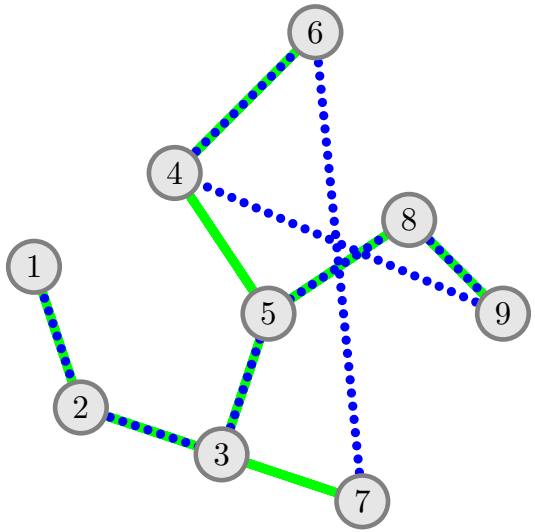
- Суммарная длина рёбер в минимальном остовном дереве — его рёбра выделены зелёным цветом — равна T.
- Обойдём дерево в порядке посещения поиском в глубину, получим синий путь длины D.

Приближённый алгоритм

2-приближение



B=29.1077



D=37.82912

- Оптимальный путь B является остовным деревом, но не обязательно минимальным. Значит, $T \leqslant B$.
- По неравенству треугольника длина обхода дерева не больше удвоенной суммы его рёбер. Поэтому $D \leqslant 2T$.
- Получается, что $D \leqslant 2B$, то есть найденный путь не более чем в два раза хуже оптимального.

Сложность

Сложность

- Если начальный город зафиксирован, существует $(n-1)!$ различных путей.
- Из них нужно выбрать кратчайший.
- $n = 10$: всего $9! = 362\,880$ путей.
- $n = 20$: всего $19! = 121\,645\,100\,408\,832\,000$ путей.
- $n = 100$: всего $99! \approx 9.332 \cdot 10^{157}$ путей.

Сложность

Сложность

- Это NP-трудная задача.
- Это значит, что решение за время Poly(n) неизвестно.
- Более того, если удастся решить её за такое время, это будет значить, что многие задачи, считающиеся трудными, также имеют решение за полиномиальное время.
- Практический смысл: скорее всего, не удастся найти оптимальное решение для достаточно больших n.
- Значит, будем искать решение, близкое к оптимальному, или просто как можно лучшее.

Какие бывают решения

Идеальный план занятия:

- Рассмотрим задачу.
- Разберём несколько решений.
- Докажем, что они работают быстро и правильно.
- Научимся реализовывать их в программе коротко и понятно.

План этого занятия:

- Рассмотрим задачу.
- Разберём несколько решений.
- Ничего не будем доказывать.
- ?!?!!
- Попробуем объяснить интуитивно, почему одни решения должны работать лучше других.

Какие бывают решения

План этого занятия:

- Рассмотрим задачу.
- Разберём несколько решений.
- Ничего не будем доказывать.
- ?!?!!
- Попробуем объяснить интуитивно, почему одни решения должны работать лучше других.

Почему так:

- Доказывать, что у нас получатся насколько-то хорошие решения, сложно.
- Наша конечная цель сейчас — не теоретическое обоснование, а практическое решение, получающее хороший результат.

Метод Монте-Карло

1. Метод Монте-Карло (Monte-Carlo Method)

- Выберем случайный путь и посчитаем его длину.
- Сделаем это много раз и выберем наилучший ответ.

Локальное изменение

Локальное изменение

- Придумаем *локальное изменение*: как немного поменять путь.
- Окрестность пути p — какие пути можно получить из p одним локальным изменением.
- Сам путь p должен поменяться не очень сильно.
  Цель — поменять должно быть быстрее, чем строить новый путь.
- Длина пути $f(p)$ тоже должна поменяться не очень сильно.
  Цель — в окрестности пути p значения f похожи на $f(p)$.

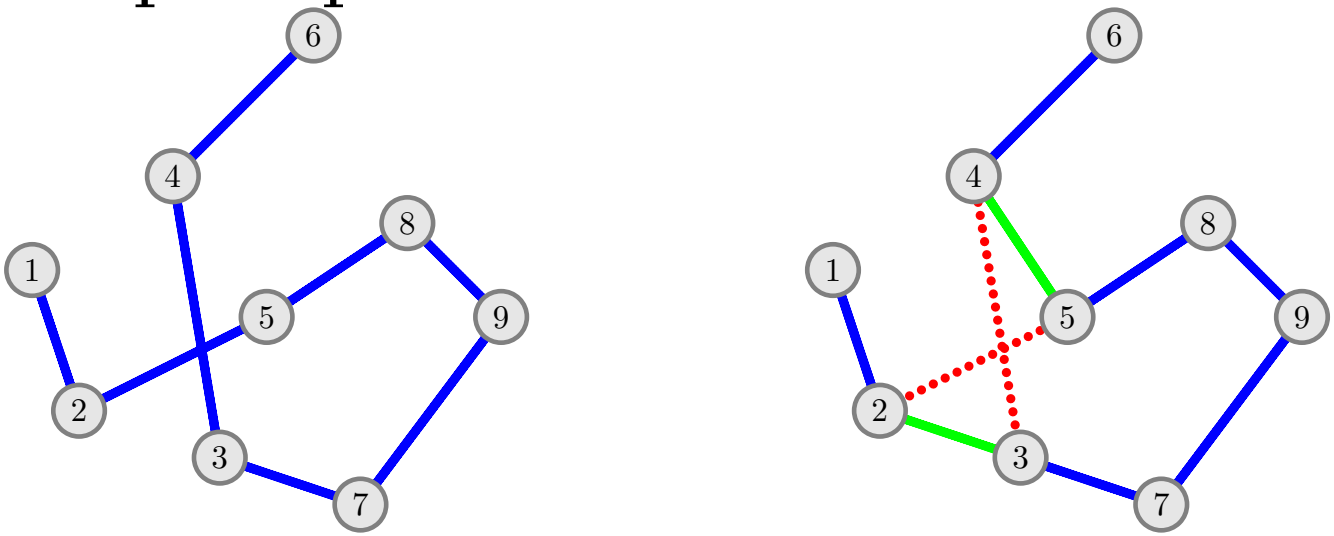
Локальное изменение

Локальное изменение

- Хорошее *локальное изменение*: перевернуть отрезок пути.
- Из пути $a \cdots b \cdot c \cdots d \cdot e \cdots f$ делаем путь $a \cdots b \cdot \overline{d \cdots c} \cdot e \cdots f$.
- Изменение длины: рёбра bc и de заменяются на рёбра bd и ce.
- Если хранить путь в массиве, придётся перевернуть отрезок.
- Чтобы это получалось быстро, можно выбирать короткие отрезки.

Локальное изменение

Пример локального изменения



- Старый путь: $1 \cdot 2 \cdot 5 \cdot 8 \cdot 9 \cdot 7 \cdot 3 \cdot 4 \cdot 6$ (синяя линия слева).
- Новый путь: $1 \cdot 2 \cdot \overline{3 \cdot 7 \cdot 9 \cdot 8 \cdot 5} \cdot 4 \cdot 6$ (сплошная линия справа).
- Удалённые рёбра: $2 \cdot 5$ и $3 \cdot 4$ (красные пунктирные).
- Добавленные рёбра: $2 \cdot 3$ и $5 \cdot 4$ (зелёные).

Случайное блуждание

2. Случайное блуждание (Random Walk)

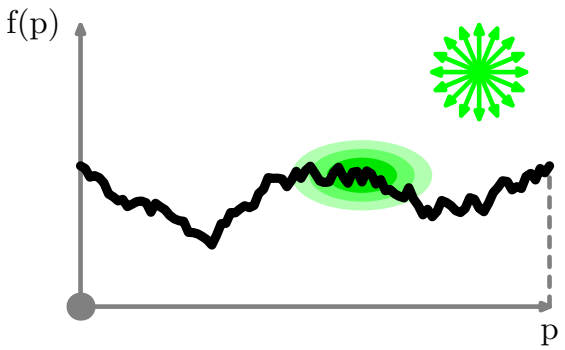
- Выберем случайный путь и посчитаем его длину.
- Сделаем случайное локальное изменение и пересчитаем длину.
- Повторим предыдущий шаг много раз и выберем наилучший ответ.
- Преимущество: мы успеем рассмотреть больше путей.

Случайное блуждание

2. Случайное блуждание (Random Walk)

```
s := randomState
f := score (s)
fBest := f;  sBest := s
for  step = 0;  step < steps;  step++:
    c := randomLocalChange
    f := applyAndRescore (s, c)
    if  fBest > f:
        fBest := f;  sBest := s

Constants example:
steps = 1000000
```



Поиск локального оптимума

3. Поиск локального оптимума (Hill Climbing)

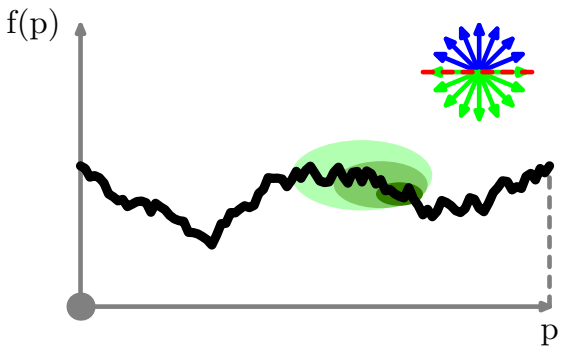
- Выберем случайный путь и посчитаем его длину.
- Сделаем случайное локальное изменение и пересчитаем длину.
- Если длина пути стала больше, отменим это изменение.
- Повторим предыдущие два шага много раз.
- Рано или поздно мы найдём локальный оптимум: такой путь, в окрестности которого нет путей лучше.

Поиск локального оптимума

3. Поиск локального оптимума (Hill Climbing)

```
s := randomState
f := score (s)
fBest := f;  sBest := s
for  step = 0;  step < steps;  step++:
    c := randomLocalChange
    fOld := f
    f := applyAndRescore (s, c)
    if  f > fOld:
        f := fOld;  revert (s, c)
    else if  fBest > f:
        fBest := f;  sBest := s

Constants example:
steps = 1000000
```



Как совместить

Случайное блуждание:

- Рано или поздно посетим все состояния.
- Но это займёт слишком много времени.

Поиск локального оптимума:

- Быстро найдём локальный оптимум.
- Но никогда из него не выйдем.

Как совместить преимущества этих подходов?

- Действовать примерно как в случайном блуждании.
- Но двигаться к хорошим решениям.

Пороговое принятие

4. Пороговое принятие (Threshold Accepting)

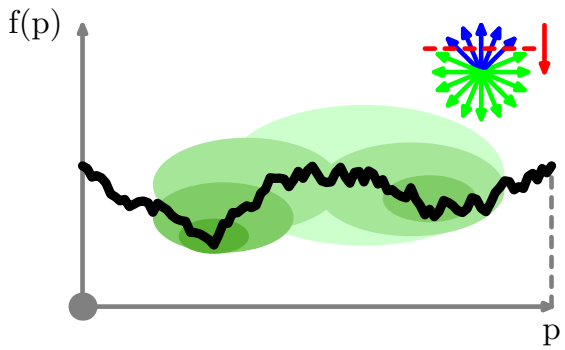
- Выберем случайный путь и посчитаем его длину.
- Сделаем случайное локальное изменение и пересчитаем длину.
- Если длина пути стала хуже более чем на пороговое значение t (threshold), отменим это изменение.
- Повторим предыдущие два шага много раз.
- При этом значение t уменьшается:
  - в начале оно очень большое, и принимаются почти любые изменения;
  - в конце оно очень маленькое, и принимаются почти только улучшения;
  - в середине алгоритм в среднем движется к более хорошим решениям.

Пороговое принятие

4. Пороговое принятие (Threshold Accepting)

```
s := randomState
f := score (s)
fBest := f;  sBest := s
for  t := tMax;  t > tMin;  t -= tStep:
    c := randomLocalChange
    fOld := f
    f := applyAndRescore (s, c)
    if  f > fOld + t:
        f := fOld;  revert (s, c)
    else if  fBest > f:
        fBest := f;  sBest := s
```

```
Constants example:
tMax = 2.0 * averageScoreDelta
tMin = 0.0 * averageScoreDelta
tStep = 0.00001 * averageScoreDelta
```



Метод отжига

5. Метод отжига (Simulated Annealing)

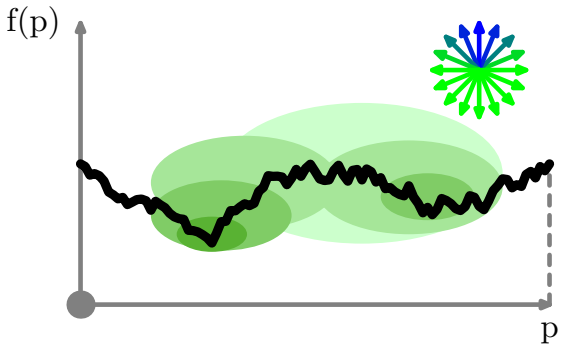
- Выберем случайный путь и посчитаем его длину.
- Сделаем случайное локальное изменение и пересчитаем длину.
- Если длина пути стала хуже, с вероятностью, зависящей от того, насколько хуже, и от температуры t (temperature), отменим это изменение.
- Повторим предыдущие два шага много раз.
- При этом значение t уменьшается:
  - в начале оно очень большое, и принимаются почти любые изменения;
  - в конце оно очень маленькое, и принимаются почти только улучшения;
  - в середине алгоритм в среднем движется к более хорошим решениям.

Метод отжига

5. Метод отжига (Simulated Annealing)

```
s := randomState
f := score (s)
fBest := f;  sBest := s
for  t := tMax;  t >= tMin;  t *= tMult:
    c := randomLocalChange
    fOld := f
    f := applyAndRescore (s, c)
    if  random (0, 1) >= exp ((fOld - f) / t):
        f := fOld;  revert (s, c)
    else if  fBest > f:
        fBest := f;  sBest := s
```

```
Constants example:
tMax = 1.0 * averageScoreDelta
tMin = 0.1 * averageScoreDelta
tMult = 0.9999
```



Выбор констант

Как выбрать константы:

- В алгоритмах есть константы `tMax`, `tMin`, ...
- Как выбрать для них «правильные» значения?
- Они сильно зависят от задачи и решения: как устроена целевая функция, как на неё влияет локальное изменение.
- С первого раза, скорее всего, не получится угадать.
- Придётся экспериментировать:
  - написать общую часть решения,
  - написать генератор случайных тестов,
  - попробовать разные значения констант на одном и том же большом наборе тестов.

Выбор констант

Как выбрать константы:

- В псевдокоде `averageScoreDelta` — на сколько в среднем меняется целевая функция при локальном изменении, то есть среднее значение `abs (f - fOld)`.
- В нашем решении локальное изменение удаляет два ребра и добавляет два других ребра.
- Грубая оценка: `averageScoreDelta` — это средняя длина ребра в исходном графе.

Содержание

Условие

Метрическая задача коммивояжёра (было)

- Есть n городов — пусть это точки на плоскости.
- Между каждой парой городов есть дорога.
- Её длина — расстояние между городами.
- Мы начинаем в фиксированном городе.
- Мы хотим посетить каждый город ровно один раз.
- При этом мы хотим минимизировать суммарную длину пройденных дорог.

Условие

Задача коммивояжёра, «вид сбоку» (стало)

- Есть n городов — пусть это точки на плоскости.
- Между каждой парой городов есть дорога.
- Её длина — расстояние между городами.
- Мы начинаем в фиксированном городе.
- Мы хотим посетить как можно больше городов.
- При этом максимальная длина нашего пути ограничена числом L.

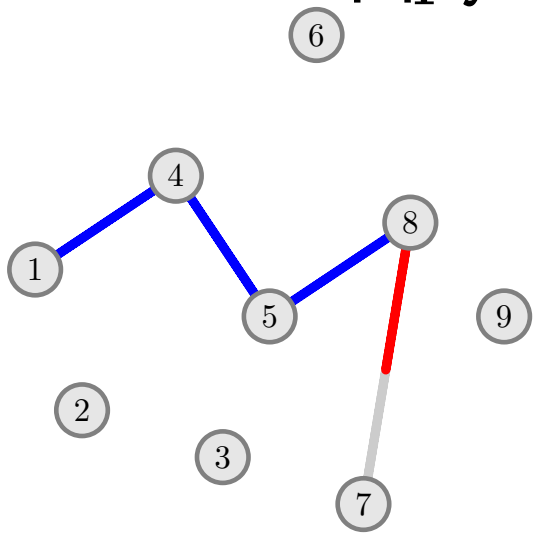
Сведение

Решение сведением к предыдущей задаче

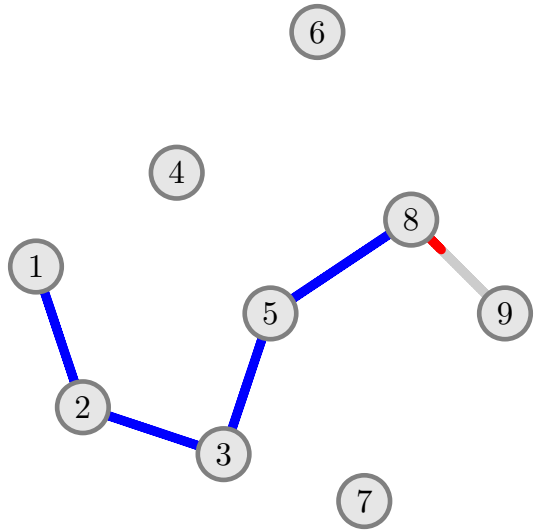
- Устроим *двоичный поиск по ответу*: сколько городов (пусть r) получится посетить.
- Когда желаемое количество городов r зафиксировано, минимизируем длину пути с таким количеством городов.
- Можно решать точно так же, как задачу коммивояжёра, но в перестановке из n городов важен только порядок первых r городов.
- И длина пути — это длина первых $r - 1$ дорог.
- В зависимости от того, получилось ли обойти r городов путём длины не более L, двоичный поиск двигает границы возможных r вправо или влево.

Другое решение

Другое решение



f=3.52333



f=4.32089

- Выберем другую целевую функцию: сколько городов (пусть f) получится посетить.
- Эта функция «слишком дискретная», так что добавим вещественное число от 0 до 1: какую часть следующего ребра удастся пройти.
- Применимы те же методы, что и в задаче коммивояжёра.

Содержание

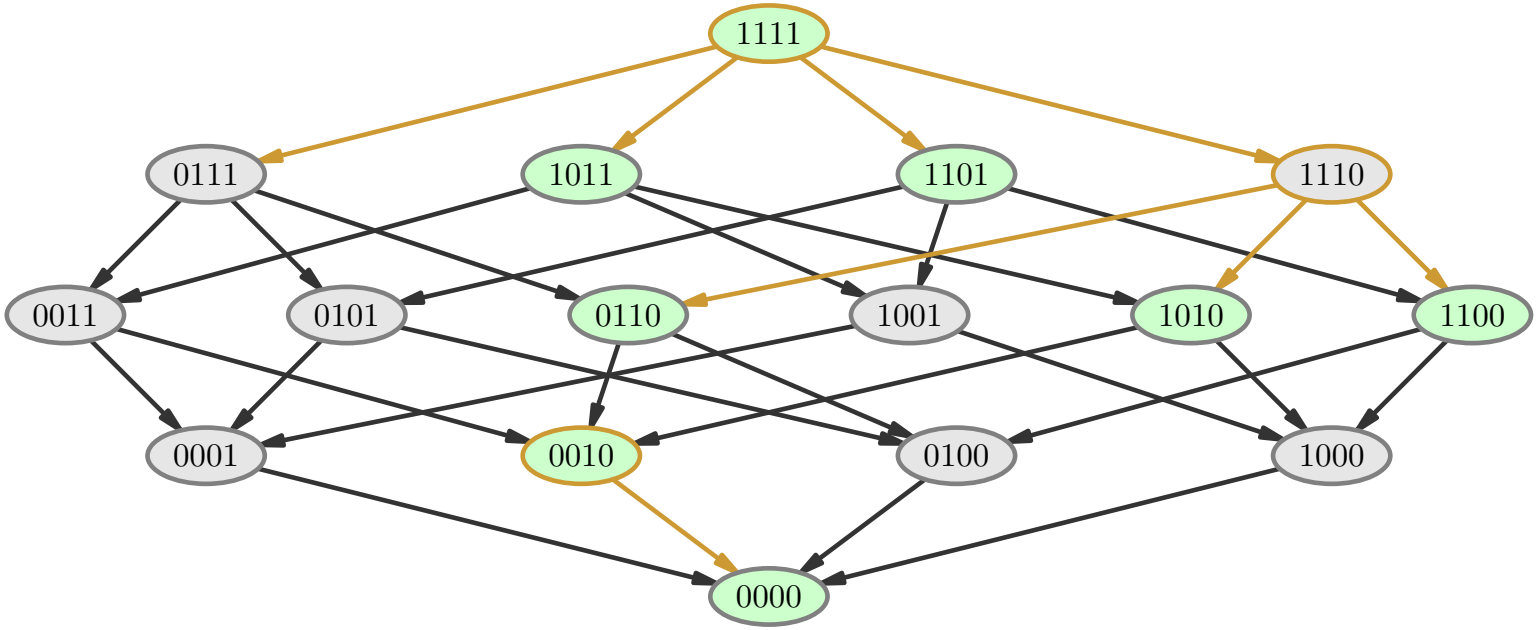
Условие

Формальное условие задачи

- Есть $n \leqslant 6$ тем.
- Рассмотрим граф из всех подмножеств тем, всего их 2^n.
- Из каждой вершины-подмножества U проведём дугу $U \to V$ во все $U \subset V$, в которых не хватает ровно одного элемента из V.
- Некоторые вершины графа отмечены: это студенты, желающие узнать такое подмножество тем.
- Другие вершины не отмечены: таких студентов нет.
- Можно завести группу для студентов: выбрать вершину A и покрыть её, а также все B, в которые ведёт дуга $A \to B$.
- Задача — покрыть все отмеченные вершины минимальным количеством групп.
- Это пример конкретной задачи о покрытии множествами.

Пример

Пример



- В примере $n = 4$.
- Зелёные вершины — студенты.
- Оранжевые эллипсы и стрелки из них — группы.

Точное решение

Точное решение

- Динамика по профилю на подмножествах.
- Идём по «уровням» подмножеств (количеству элементов) сверху вниз.
- Состояние:
  - на уровнях выше все отмеченные вершины уже покрыты,
  - на текущем уровне храним битовую маску (профиль): какие вершины уже покрыты группами с предыдущего уровня.
  - на уровнях ниже ничего ещё не покрыто.
- Целевая функция: сколько уже заведено групп.
- ...
- Сложное, не будем подробнее его разбирать.

Приближённое решение

Приближённое решение

- Состояние: какие группы заведены, а какие нет (всего 2^{2^n} возможных вариантов).
- Изначально все группы заведены.
- Локальные изменения:
  - добавить случайную группу, которой нет;
  - удалить случайную группу, которую можно удалить (все отмеченные вершины остаются покрытыми).
- Добавление пробуем с вероятностью P:
  - $P = 1$ в начале работы алгоритма, $P = 0$ в конце;
  - подберём константы так, чтобы уложиться в ограничения по времени;
  - или просто сделаем вероятность зависящей от времени.
- Пространство возможных решений очень маленькое — не более 2^{64} состояний — и с большой вероятностью найдётся оптимальный ответ.

Содержание

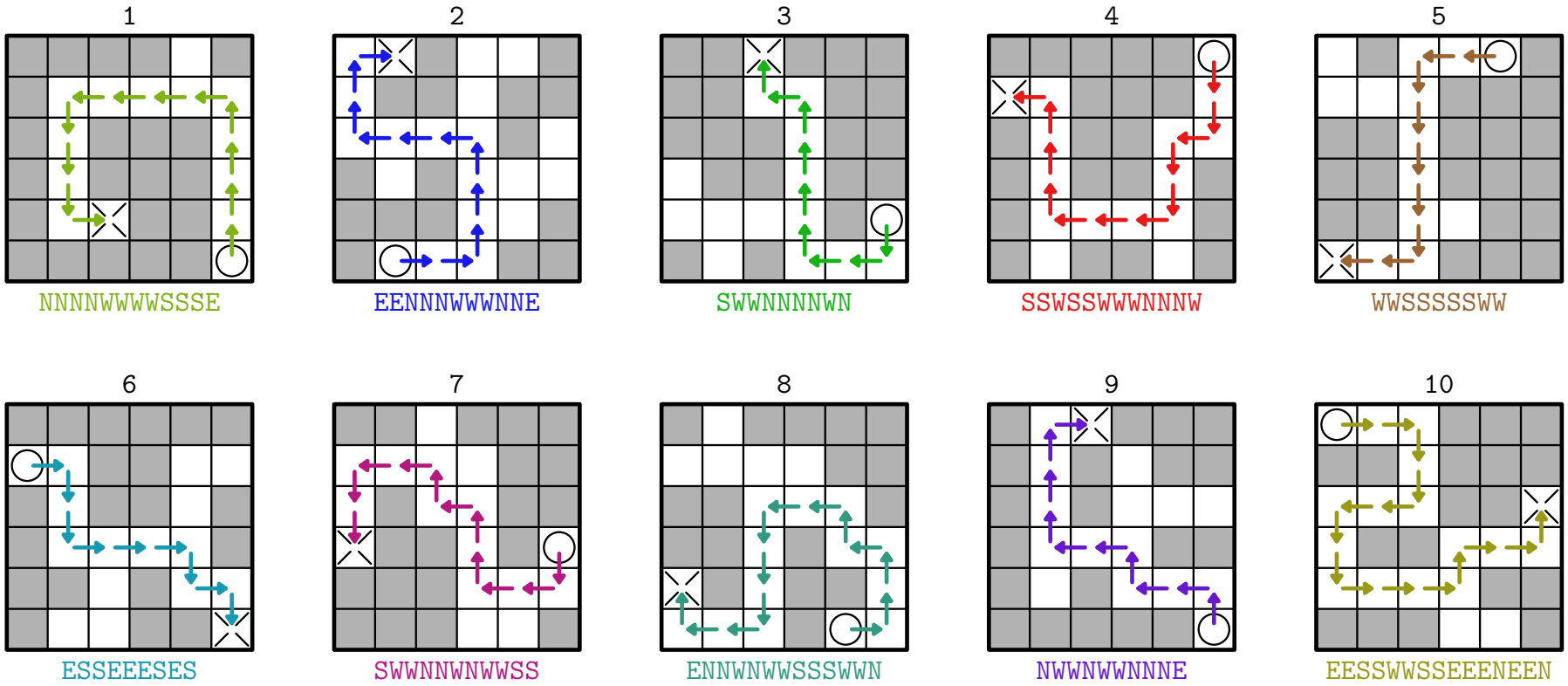
Условие

Условие задачи

- Робот ходит по клетчатой плоскости.
- Есть программа для робота — строка, состоящая из команд «N», «W», «S», «E»:
  - если соседняя клетка в этом направлении свободна, робот перемещается туда;
  - если соседняя клетка в этом направлении занята стеной, робот остаётся на месте.
- Есть случайные лабиринты из 10×10 клеток.
  - в каждом зафиксированы начальная и конечная клетки;
  - робот стартует в начальной клетке и выполняет всю программу слева направо;
  - лабиринт считается пройденным, если при выполнении программы робот хоть раз посетил конечную клетку;
  - алгоритм построения лабиринтов дан в условии.
- Задача — написать такую программу, чтобы она в среднем быстро проходила случайные лабиринты.

Пример

Пример



- Для наглядности размер уменьшен с 10×10 до 6×6.
- Каждый лабиринт пройден отдельно.

Как пройти все лабиринты

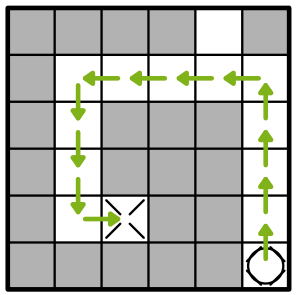
Как пройти все лабиринты

- Сначала пройдём все возможные лабиринты хоть как-то.
- Количество возможных лабиринтов конечно.
- Рассмотрим какой-то из них («первый»):
  - напишем программу P_1, которая его проходит.
- Рассмотрим какой-то другой лабиринт («второй»):
  - поставим робота в начальную клетку, выполним все команды P_1;
  - допишем к программе строку P_2, которая его проходит;
  - например, если второй лабиринт уже пройден во время выполнения P_1, то P_2 — пустая строка.
- Рассмотрим какой-то ещё лабиринт («третий»):
  - учитывая, что программа уже начинается на строку P_1P_2, пройдём и его.
- Переберём так все лабиринты и пройдём их.

Как пройти все лабиринты

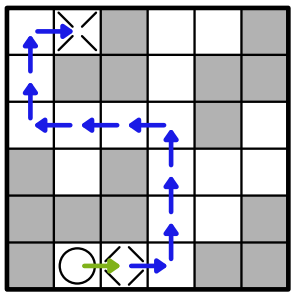
Как пройти все лабиринты

1



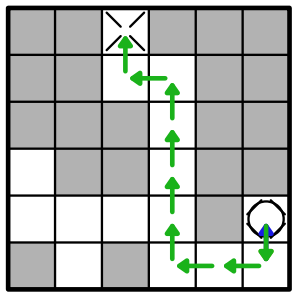
NNNNWWWWSSSE

2



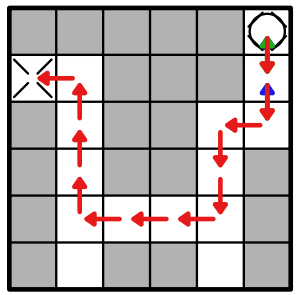
ENNNWWWNNE

3



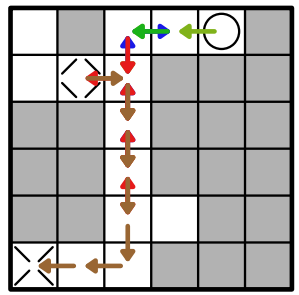
SWWNNNNWN

4



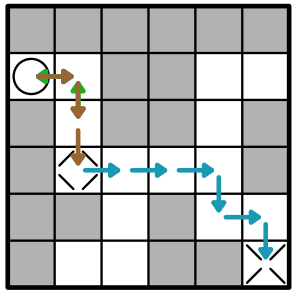
SSWSSWWWNNNW

5



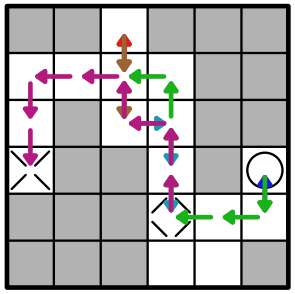
ESSSSWW

6



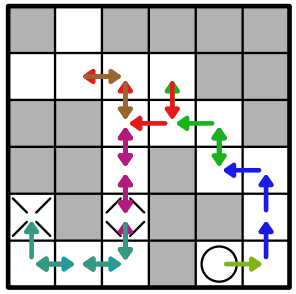
EEESES

7



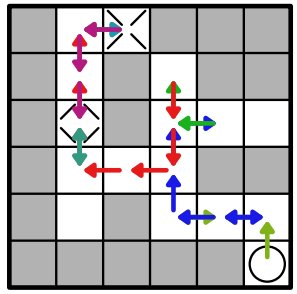
NNWNWWSS

8

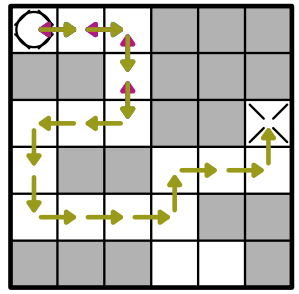


SWWN

9



10



EESSWWSSEEENEEN

- Под лабиринтами написаны строки $P_1, P_2, \ldots, P_{10}$.
- Строка P_9 пуста, потому что девятый лабиринт уже пройден строкой $P_1 P_2 \ldots P_8$.

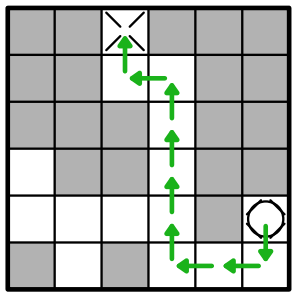
Как пройти много лабиринтов быстро

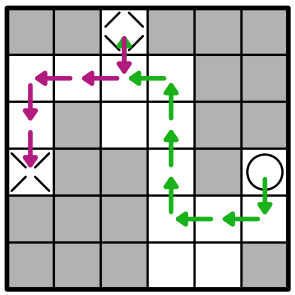
Как пройти много лабиринтов быстро

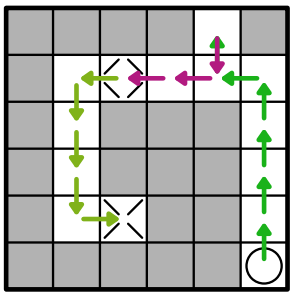
- Теперь придумаем, как укоротить получаемую строку.
- Сгенерируем большое количество лабиринтов заранее (например, $P = 10\,000$ штук).
- Будем проходить их в каком-то порядке.
- В каждый момент жадно выбираем следующий лабиринт для прохождения:
  - тот, в котором осталось меньше всего шагов до финиша;
  - при равенстве идём в том направлении, в котором больше лабиринтов с минимальным количеством шагов до финиша.

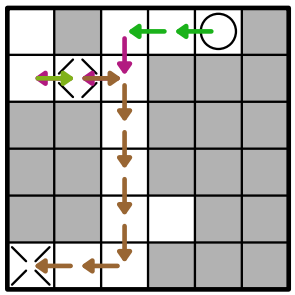
Как пройти много лабиринтов быстро

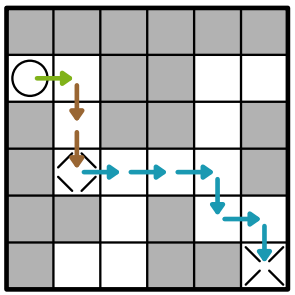
Как пройти много лабиринтов быстро

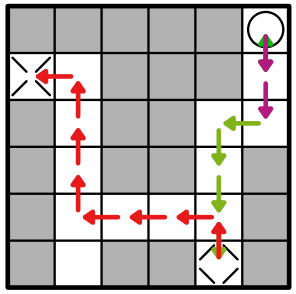
3

SWWNNNNWN

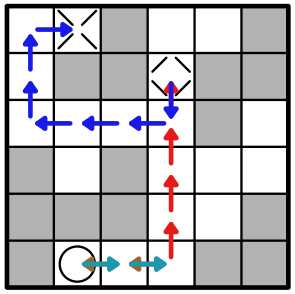
7

SWWSS

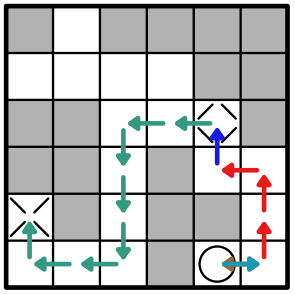
1

WSSSE

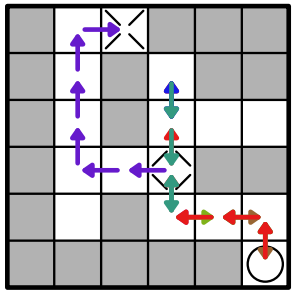
5

ESSSSWW

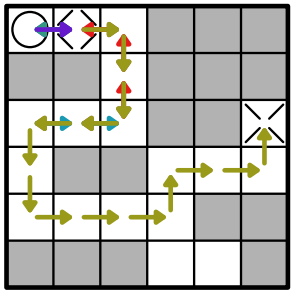
6

EEESES

4

NWWWNNNW

2

SWWWNNE

8

WWSSSWWN

9

WWNNNE

10

ESSWWSSEEENEEN

- Лабиринты расставлены в порядке прохождения.

Как пройти много лабиринтов быстро

Как пройти много лабиринтов быстро

- Улучшение: после прохождения очередного лабиринта сгенерируем следующий, чтобы всегда поддерживать $P = 10\,000$ непройденных лабиринтов.
  - Придётся перегенерировать следующий лабиринт заново, пока оказывается, что сгенерированный уже пройден.
- В таком решении каждый ход, кроме первой сотни, проходит десятки или даже сотни хранимых лабиринтов.
- Делаем всё это, пока не получим строку нужной длины.
  - Чем лучше наше решение, тем дольше генерируется следующий непройденный лабиринт.
  - Предподсчёт: если решение работает долго, можно запустить его локально, а на проверку отправить только вывод полученной строки.

Содержание

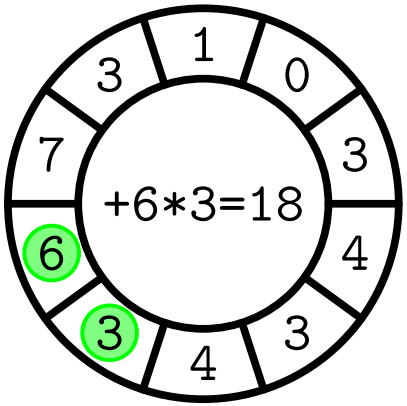
Условие

Условие задачи

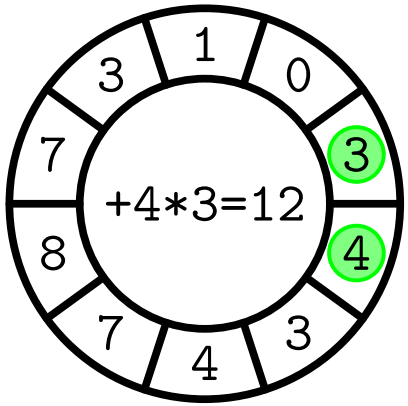
- Есть короткая зацикленная лента из $K = 10$ клеток.
  - В каждой клетке написана цифра от 0 до 9.
- Есть длинная последовательность из $L = 10\,000$ цифр.
  - Каждый элемент последовательности сгенерирован псевдослучайно, все цифры от 0 до 9 равновероятны.
- Ход:
  - взять две цифры из соседних клеток ленты,
  - прибавить к баллам произведение этих цифр,
  - поставить в эти две клетки на ленте две следующие цифры из последовательности.
- Задача — к моменту, когда последовательность кончится, получить как можно больше баллов.

Пример

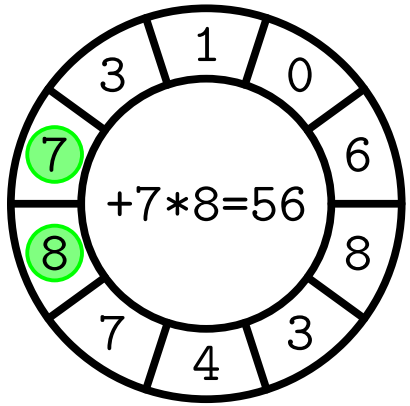
Пример



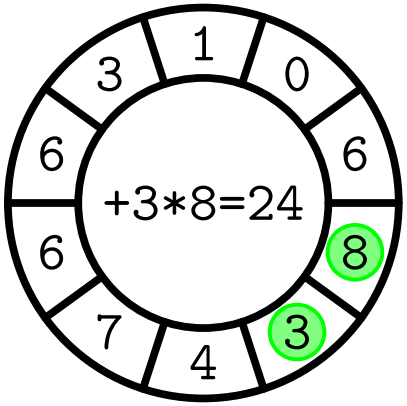
next=8786663999...



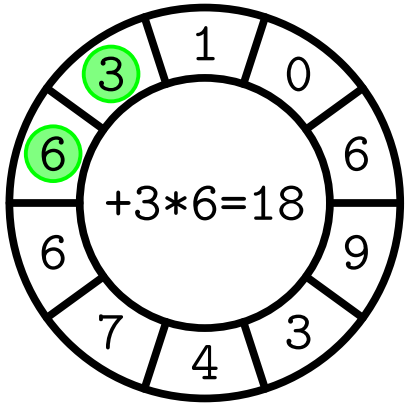
next=86663999...



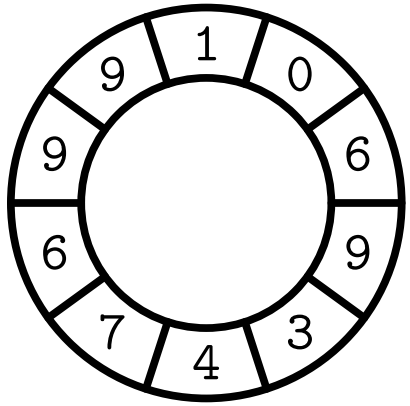
next=663999...



next=3999...



next=99...



next=...

Простые решения

Решение 1: константное

- Можно просто всегда брать первую и вторую цифры.
- Последовательность случайная, так что результат будет такой же, как у решения, выбирающего каждый раз случайную пару.
- Такое решение уже набирает больше половины баллов!
- Баллы: ≈ 65 из 100.

Простые решения

Решение 2: жадное

- Можно брать пару с наибольшим произведением.
- Как ни странно, результат улучшился не сильно...
- Баллы: ≈ 70 из 100.

Локальные изменения и время

Локальные изменения и время

- Бывают задачи, в которых сложно придумать локальное изменение.
- Например, так часто бывает, когда в решении участвует время.
- Изменение решения в один момент может влиять на много следующих моментов.
- Для задач со временем есть другой метод приближённого решения: лучевой поиск.

Локальные изменения и время

Мотивация

- Рассмотрим задачу, где решение — результат некого выбора в несколько последовательных моментов времени.
- Будем строить решения в порядке, соответствующем времени: сначала сделаем выбор в первый момент, затем в следующий, и так далее.
- Если в каждый момент делать один локально оптимальный выбор, получится жадное решение — а оно может быть далеко от глобально оптимального.
- Если в каждый момент рассматривать все возможные варианты, получатся все решения, в том числе глобально оптимальное — но возможных решений слишком много, чтобы это успеть.
- Попробуем в каждый момент рассматривать несколько локально оптимальных вариантов, но не очень много.

Лучевой поиск

Лучевой поиск (Beam Search)

- Будем строить решения в порядке, соответствующем времени: сначала сделаем выбор в первый момент, затем в следующий, и так далее.
- Изначально есть только одно пустое решение.
- Пусть в предыдущий момент у нас было несколько решений (пусть r).
- Для каждого из них всеми способами (пусть их c) сделаем выбор и получим $r \cdot c$ решений.
- Из этих решений оставим w наилучших и перейдём к следующему моменту.

Лучевой поиск

Ширина пучка

- Образно: решения — это лучи, и мы рассматриваем пучок (beam) этих лучей.
- Число w называется шириной пучка (beam width).
- Жадный выбор — это частный случай, в котором $w = 1$.
- Полный перебор — это частный случай, в котором $w = \infty$.
- Чем больше w, тем больше времени займёт поиск — но, вероятно, тем более хорошее решение удастся найти.

Лучевой поиск

Применение в задаче

- Выберем такое w, чтобы решение укладывалось в ограничения по времени с небольшим запасом.
- Баллы: ≈ 90 из 100.
- Как решить ещё лучше?

Произведения цифр

Произведения цифр

- Часто факты из предметной области задачи оказываются важнее, чем выбор метода для перебора.
- Представим, что мы могли бы разбить все цифры на пары как угодно.
- Как было бы выгодно это сделать?
- Для четырёх чисел $a \leqslant b \leqslant c \leqslant d$ оптимальное разбиение — это пары (a, b) и (c, d). Это можно проверить, рассмотрев все три возможных разбиения, или понять из общих соображений.
- Например, для $a = 2$, $b = 3$, $c = 5$ и $d = 7$:
  - $2 \cdot 3 + 5 \cdot 7 = 6 + 35 = 41$,
  - $2 \cdot 5 + 3 \cdot 7 = 10 + 21 = 31$,
  - $2 \cdot 7 + 3 \cdot 5 = 14 + 15 = 29$.

Произведения цифр

Произведения цифр

- В общем случае оптимально собирать пары из близких чисел. То есть отсортировать все числа (цифры) и разбить отсортированную последовательность на пары. Это доказывается рассмотрением любой четвёрки чисел, разбитых на пары в «неправильном» порядке.
- Значит, чем более равные числа мы берём в пару, тем ближе в итоге решение будет к оптимальному.
- Хорошая стратегия для всей игры, кроме нескольких последних ходов — из десяти доступных пар выбирать такую, в которой числа как можно ближе.
- Последние несколько ходов мало влияют на результат, для первой попытки ими можно пренебречь.
- Баллы: ≈ 95 из 100.

Произведения цифр

Скомбинируем решения

- Скомбинируем идею о парах близких чисел с лучевым поиском.
- Вместо максимизации баллов будем минимизировать штраф:
  - например, в каждой паре вычислим квадрат разности чисел;
  - штраф — сумма этих величин во всех выбранных парах.
- Устроим лучевой поиск, в каждый момент оставляя w решений с наименьшим штрафом.
- Баллы: ≈ 99 из 100.

Содержание

Условие

Условие задачи

- Есть таблица из $n \times n$ клеток ($5 \leqslant n \leqslant 50$).
- В каждой клетке написана случайная цифра от 1 до 9.
- Нужно выбрать подмножество клеток.
- Но выбранные клетки не могут иметь общих точек.
- Задача — максимизировать сумму цифр в выбранных клетках.

Пример

Пример

3	4	9	1	3
9	9	1	9	6
3	1	2	6	7
1	7	2	9	4
8	4	1	9	6

Пример

Пример

3	4	9	1	3
9	9	1	9	6
3	1	2	6	7
1	7	2	9	4
8	4	1	9	6

Пример

Пример

3	4	9	1	3
9	9	1	9	6
3	1	2	6	7
1	7	2	9	4
8	4	1	9	6

Решения

Решение 1: сетка

- Можно выбрать клетки, у которых обе координаты (считая с нуля) чётны.
- Поскольку числа случайные, математическое ожидание результата уже не меньше половины математического ожидания в идеальном случае (если бы мы смогли выбрать максимально возможное количество девяток).

Решения

Решение 2: динамика по профилю

- Можно брать клетки по строкам.
- А в каждой строке слева направо.
- Когда мы уже смотрим на клетку, важны только сумма (значение динамики) и выбранные клетки в предыдущих двух строках (профиль динамики).
- Наивная реализация работает за $2^{2n} \cdot n^2$.
- На самом деле достижимых состояний в каждой клетке гораздо меньше, чем 2^{2n}, но всё-таки больше, чем, например, $2^{\frac{2}{3}n}$.

Как решать дальше

Локальное изменение:

- Выберем случайную не выбранную клетку.
- Отменим выбор всех её соседей (их не больше восьми).

Теперь можно сделать:

- случайное блуждание,
- поиск локального максимума,
- пороговое принятие,
- метод отжига...

На примере этой задачи удобно это всё попробовать, потому что объект простой, и писать программу легко.

Вопросы?

Вопросы?