Алгоритмы и структуры данных

Лекция 8. Графы (перебор вариантов).

(c) Глухих Михаил Игоревич, glukhikh@mail.ru

Поиск в глубину: псевдокод

```
DFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (NOT_VISITED, prev = null)
  DFS-VISIT(G, s, info, prev = null)

DFS-VISIT(G = (V, E), v in V, info, prev):
  info[v] = (VISITED, prev = prev)
  for (u in neighbors(v)):
    if (info[u] NOT_VISITED):
      DFS-VISIT(G, u, info, prev = v)
```

Поиск в глубину: трудоёмкость

```
DFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (NOT_VISITED, prev = null)
  DFS-VISIT(G, s, info, prev = null)

DFS-VISIT(G = (V, E), v in V, info, prev): // V calls
  info[v] = (VISITED, prev = prev)
  for (u in neighbors(v)):
    if (info[u] NOT_VISITED): // Total E iterations
      DFS-VISIT(G, u, info, prev = v)
```

Поиск пути в глубину: псевдокод

```
DFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (distance = INF, prev = null)
  DFS-VISIT(G, s, info, depth = 0, prev = null)

DFS-VISIT(G = (V, E), v in V, info, depth, prev):
  info[v] = (distance = depth, prev = prev)
  for (u in neighbors(v)):
    if (info[u].distance > depth + 1):
      DFS-VISIT(G, u, info, depth = depth + 1, prev = v)
```

Поиск пути: что лучше?

- В ширину / В глубину?

Поиск пути: что лучше?

- В ширину / В глубину?
  - Для невзвешенного графа ~ без разницы: O(V) + O(E)

Поиск пути: что лучше?

- В ширину / В глубину?
  - Для невзвешенного графа ~ без разницы: O(V) + O(E)
  - Для взвешенного графа алгоритм Дейкстры даёт O((V+E)logV)

Поиск пути: что лучше?

- В ширину / В глубину?
  - Для невзвешенного графа ~ без разницы: O(V) + O(E)
  - Для взвешенного графа алгоритм Дейкстры даёт O((V+E)logV)
  - Что может дать (для взвешенного графа) поиск в глубину?

Поиск пути: что лучше?

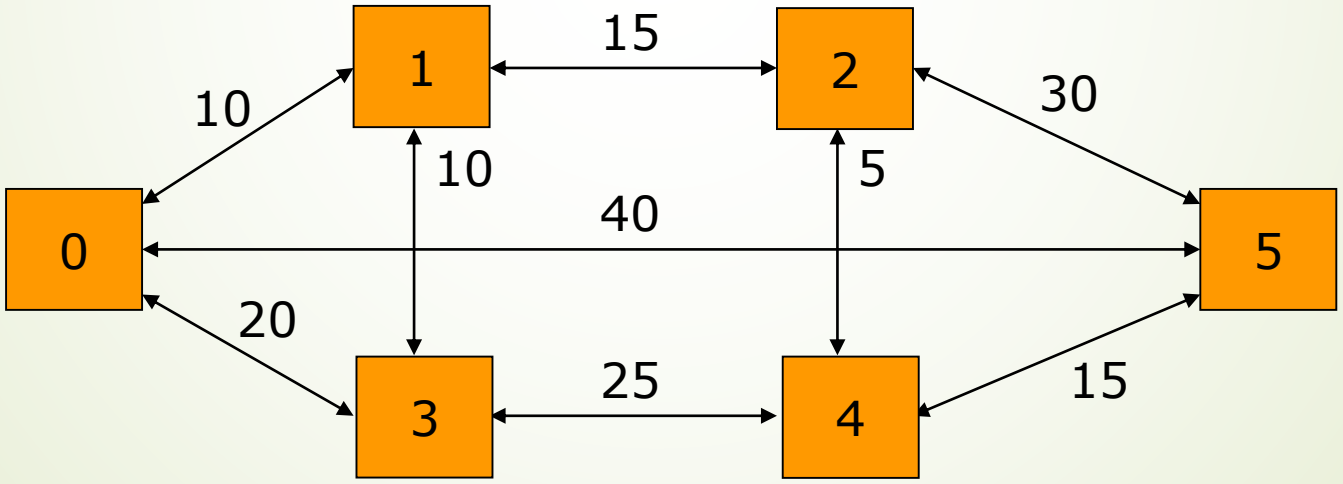
- В ширину / В глубину?
  - Для невзвешенного графа ~ без разницы: O(V) + O(E)
  - Для взвешенного графа алгоритм Дейкстры даёт O(VlogV)+ O(E)
  - Что может дать поиск в глубину?
    - Нам придётся смотреть одни и те же вершины / рёбра несколько раз…

Применение поиска в глубину

- Задача коммивояжёра

Задача коммивояжёра

- Пусть дан взвешенный граф. Требуется найти кратчайший кольцевой путь, проходящий через все вершины графа по одному разу



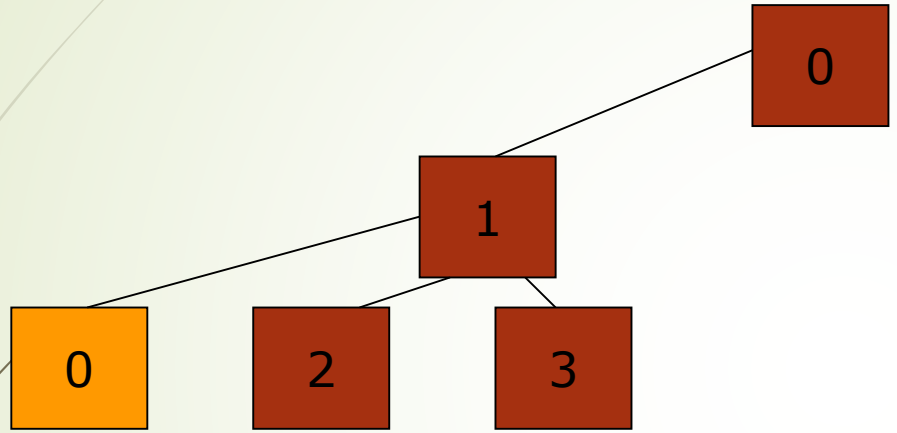
Задача коммивояжёра

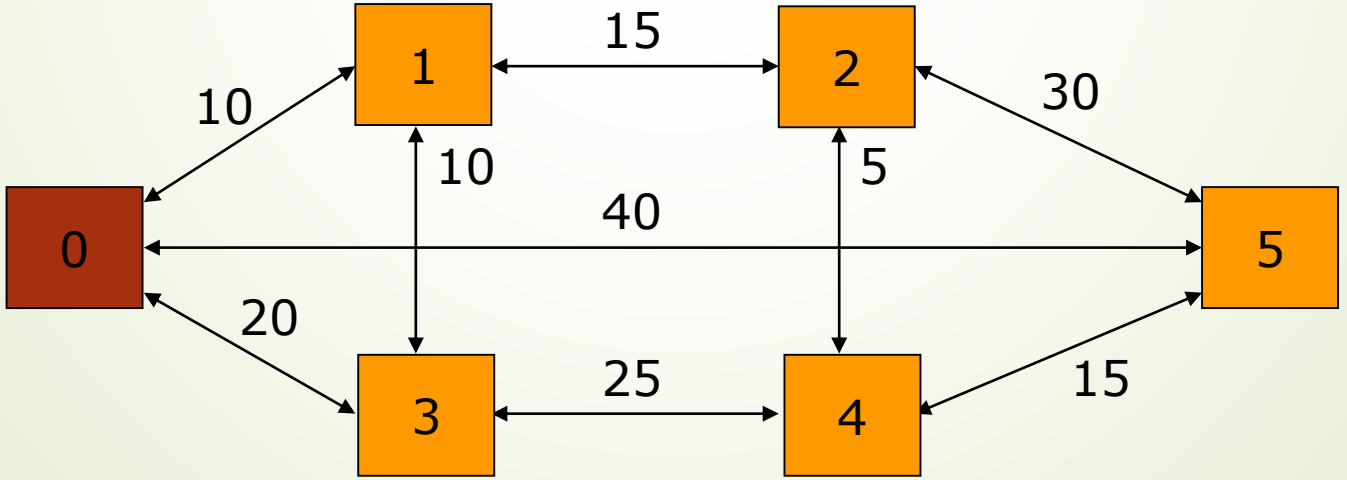
- Варианты формулировки
  - Классическая: граф полносвязный (каждая вершина связана с каждой)
  - Более общая: граф может быть не полносвязным (сводится к классической введением дополнительных дуг с большим весом)

Задача коммивояжёра: точное решение

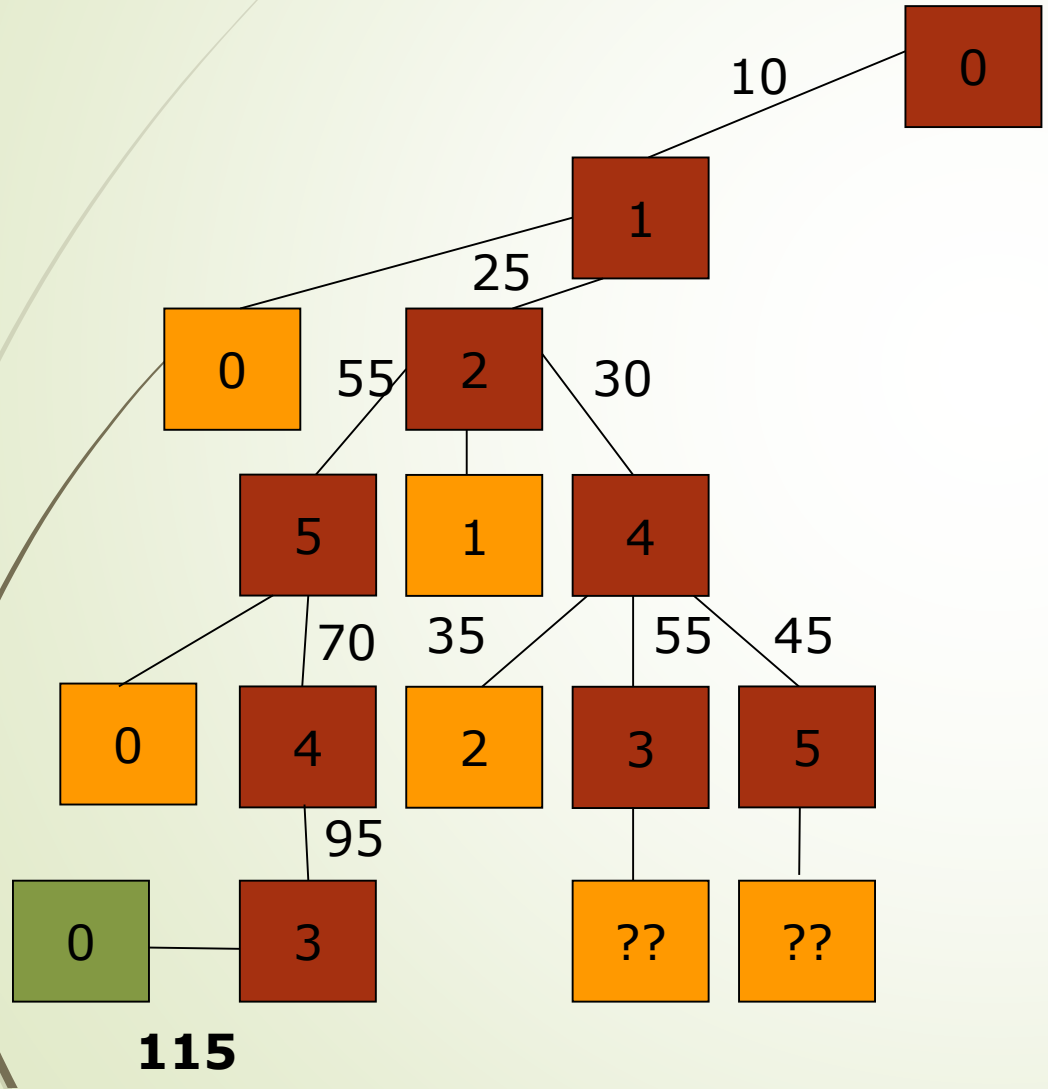
- Используем поиск в глубину (отбрасывая вершины, уже вошедшие в данный путь)
- Запоминаем самый короткий маршрут
- Отсечение: останавливаем текущую ветку поиска в глубину, если её длина уже превысила длину самого короткого маршрута

Перебор вариантов

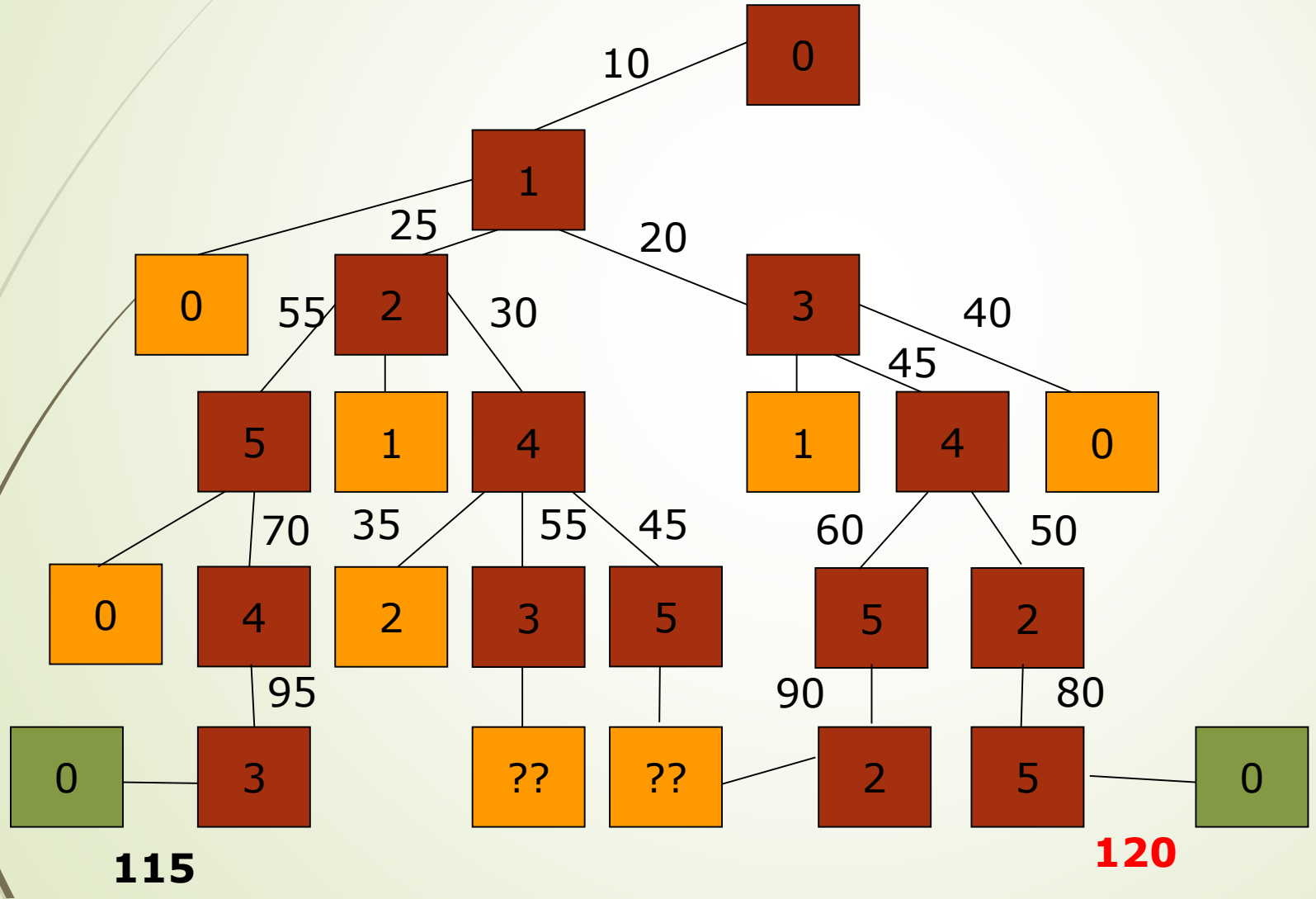




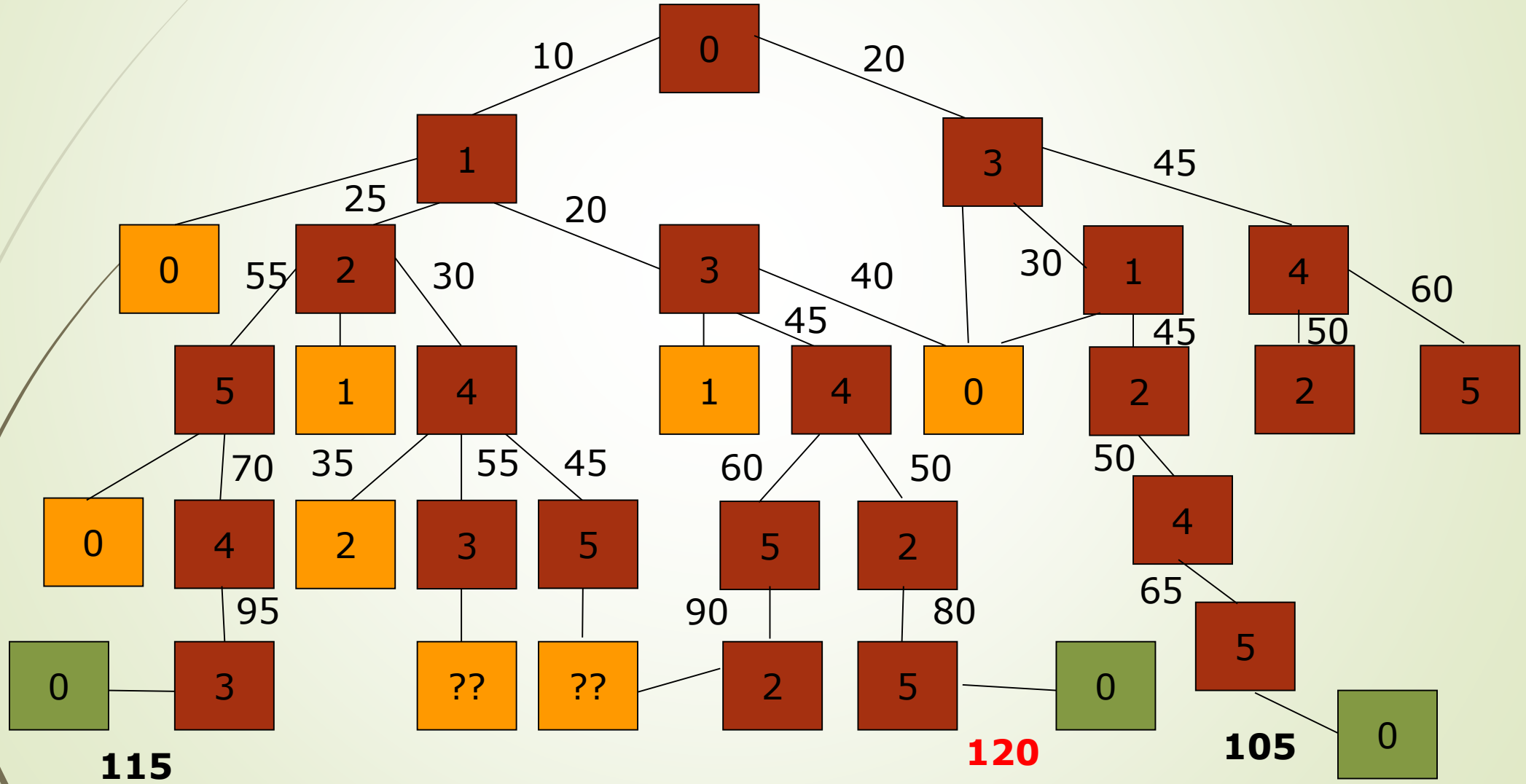
Перебор вариантов



Перебор вариантов



Перебор вариантов



Задача коммивояжёра в целом

- Почему не поиск в ширину?

Задача коммивояжёра в целом

- Почему не поиск в ширину?
  - Очередь может быстро увеличиваться в размерах

Задача коммивояжёра в целом

- Почему не поиск в ширину?
  - Очередь может быстро увеличиваться в размерах
- Трудоёмкость решения

Задача коммивояжёра в целом

- Почему не поиск в ширину?
  - Очередь может быстро увеличиваться в размерах
- Трудоёмкость решения
  - O(V!)
  - Классический пример задачи без быстрого решения
  - По-научному: NP-полная задача

Задача коммивояжёра в целом

- Почему не поиск в ширину?
  - Очередь может быстро увеличиваться в размерах
- Трудоёмкость решения
  - O(V!)
  - Классический пример задачи без быстрого решения
  - По-научному: NP-полная задача
    - В настоящее время для таких задач неизвестно решение хотя бы с полиномиальной сложностью

Задача коммивояжёра в целом

- Почему не поиск в ширину?
  - Очередь может быстро увеличиваться в размерах
- Трудоёмкость решения
  - O(V!)
  - Классический пример задачи без быстрого решения
  - По-научному: NP-полная задача
    - В настоящее время для таких задач неизвестно решение хотя бы с полиномиальной сложностью
    - Учёные склоняются к выводу, что подобное решение для них и не будет известно

Задача коммивояжёра в целом

- Почему не поиск в ширину?
  - Очередь может быстро увеличиваться в размерах
- Трудоёмкость решения
  - O(V!)
  - Классический пример задачи без быстрого решения
  - По-научному: NP-полная задача
- Можно ли решить быстрее?
  - Да, но только неточно – эвристические алгоритмы

Применение поиска в глубину

- Задача коммивояжёра
- Перебор вариантов
  - Перебор в логических играх (с ограничением глубины)

Алгоритмы выбора наилучшего хода (решения)

- Можно разделить на:
  - эмпирические алгоритмы – используют приблизительные правила, основанные на опыте
  - аналитические алгоритмы – используют результаты теоретического анализа
  - переборные алгоритмы – используют перебор вариантов
  - …

Пример эмпирического алгоритма для игры крестики-нолики 3x3

- Возможно ли выиграть одним ходом?
  - если ДА, выполнить этот ход
- Может ли соперник выиграть одним ходом?
  - если ДА, выполнить ход в клетку угрозы
- Свободна ли центральная клетка?
  - если ДА, занять ее
- Свободна ли хотя бы одна угловая клетка?
  - если ДА, занять одну из угловых клеток
- Выполнить ход в любую свободную клетку

- NB: можно ли обыграть этот алгоритм?

Пример аналитического алгоритма для игры в Ним

- Есть три кучки камней, два игрока
- Игроки ходят по очереди
  - Каждый ход – взятие любого числа камней (в том числе всех) из одной кучки
- Выигрывает тот, кто забирает последние камни

Пример аналитического алгоритма для игры в Ним

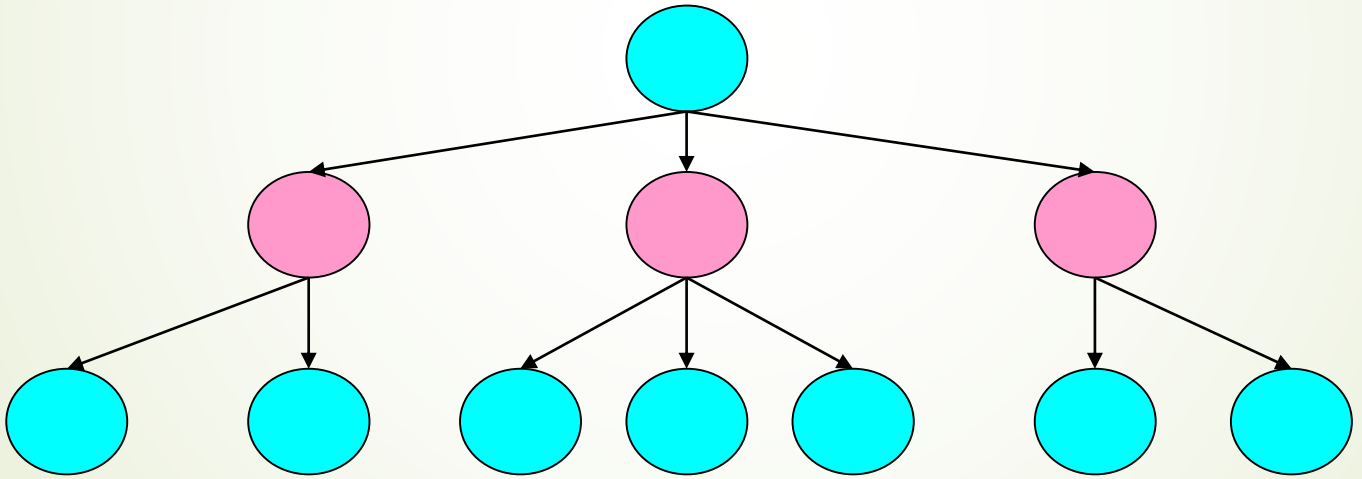
- Есть три кучки камней, два игрока
- Игроки ходят по очереди
  - Каждый ход – взятие любого числа камней (в том числе всех) из одной кучки
- Выигрывает тот, кто забирает последние камни
- Алгоритм
  - Представим число камней в кучках в двоичной системе счисления
  - Позиция *проигрышна*, если побитовый XOR этих трёх чисел равен 0
  - Пример: (12, 10, 7) → (1100, 1010, 0111) → 0001
    - Выигрышна, делаем ход в (12, 10, 6)

Ним: пример

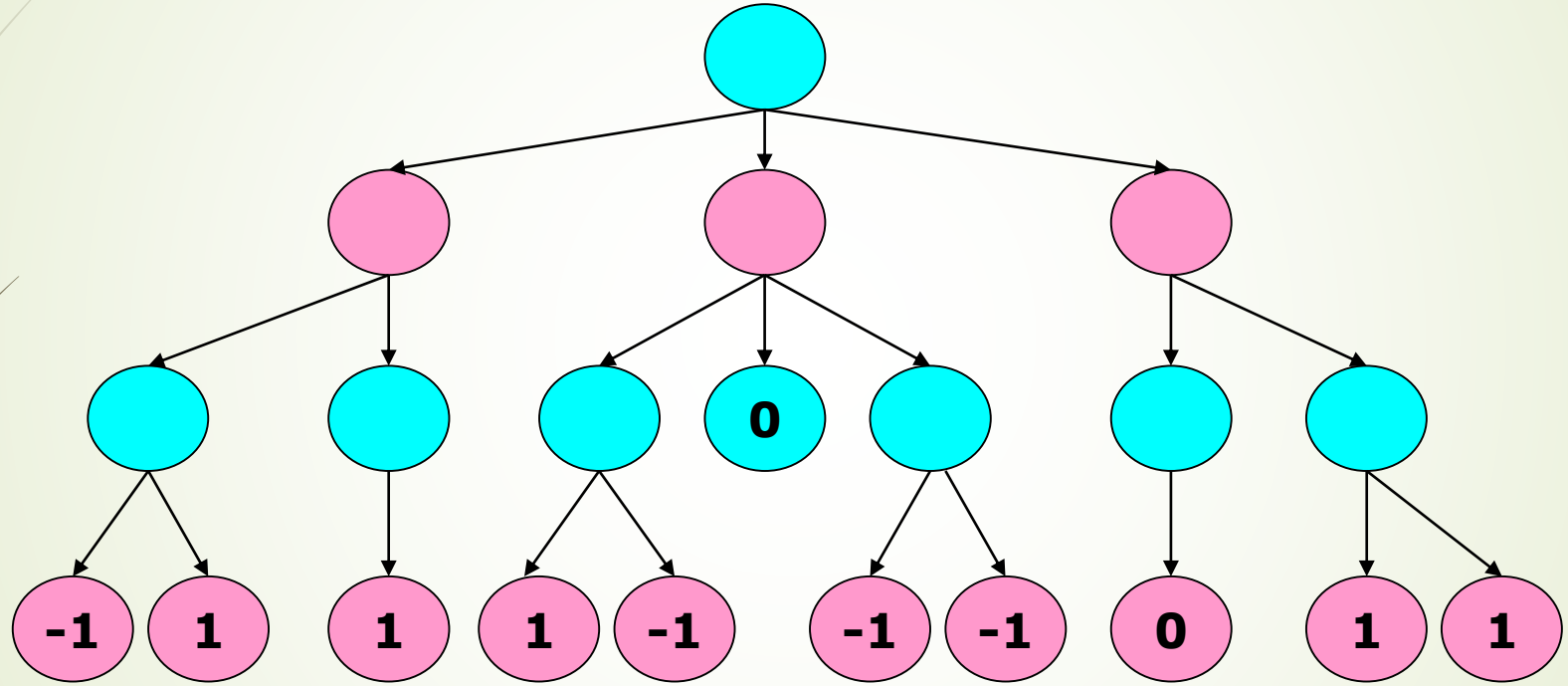
- 12, 10, 7 → 1100, 1010, 0111 → 0001
  - 12, 10, 6 → 1100, 1010, 0110 → 0000
- 9, 10, 6 → 1001, 1010, 0110 → 0101
  - 9, 10, 3 → 1001, 1010, 0011 → 0000
- 9, 7, 3 → 1001, 0111, 0011 → 1101
  - 4, 7, 3 → 0100, 0111, 0011 → 0000
- 4, 5, 3 → 0100, 0101, 0011 → 0010
  - 4, 5, 1 → 0100, 0101, 0001 → 0000
- 4, 3, 1 → 0100, 0011, 0001 → 0110
  - 2, 3, 1 → 0010, 0011, 0001 → 0000

Переборные алгоритмы

- Предполагают полный или частичный перебор вариантов ходов

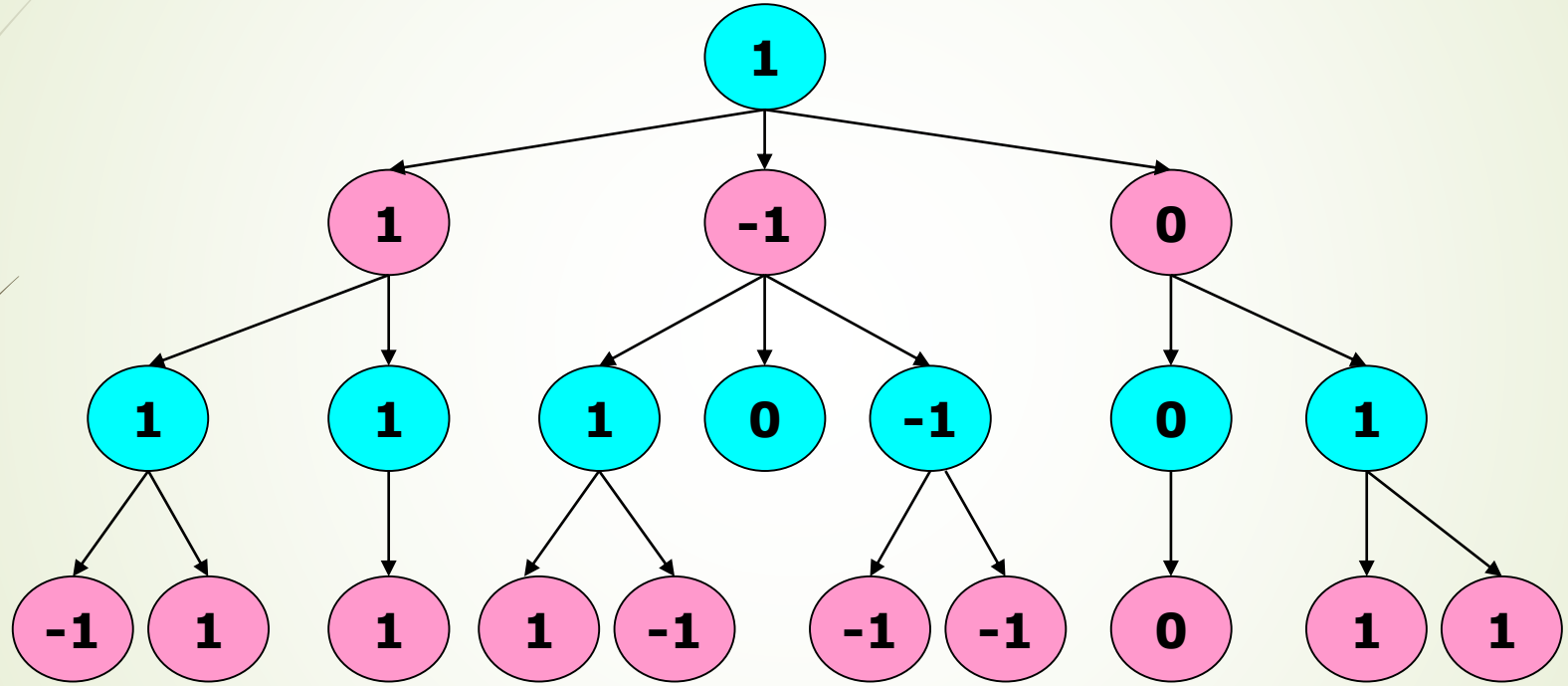


Переборные алгоритмы – идеальный вариант



- Красный выбирает минимум, а синий – максимум

Переборные алгоритмы - идеальный вариант

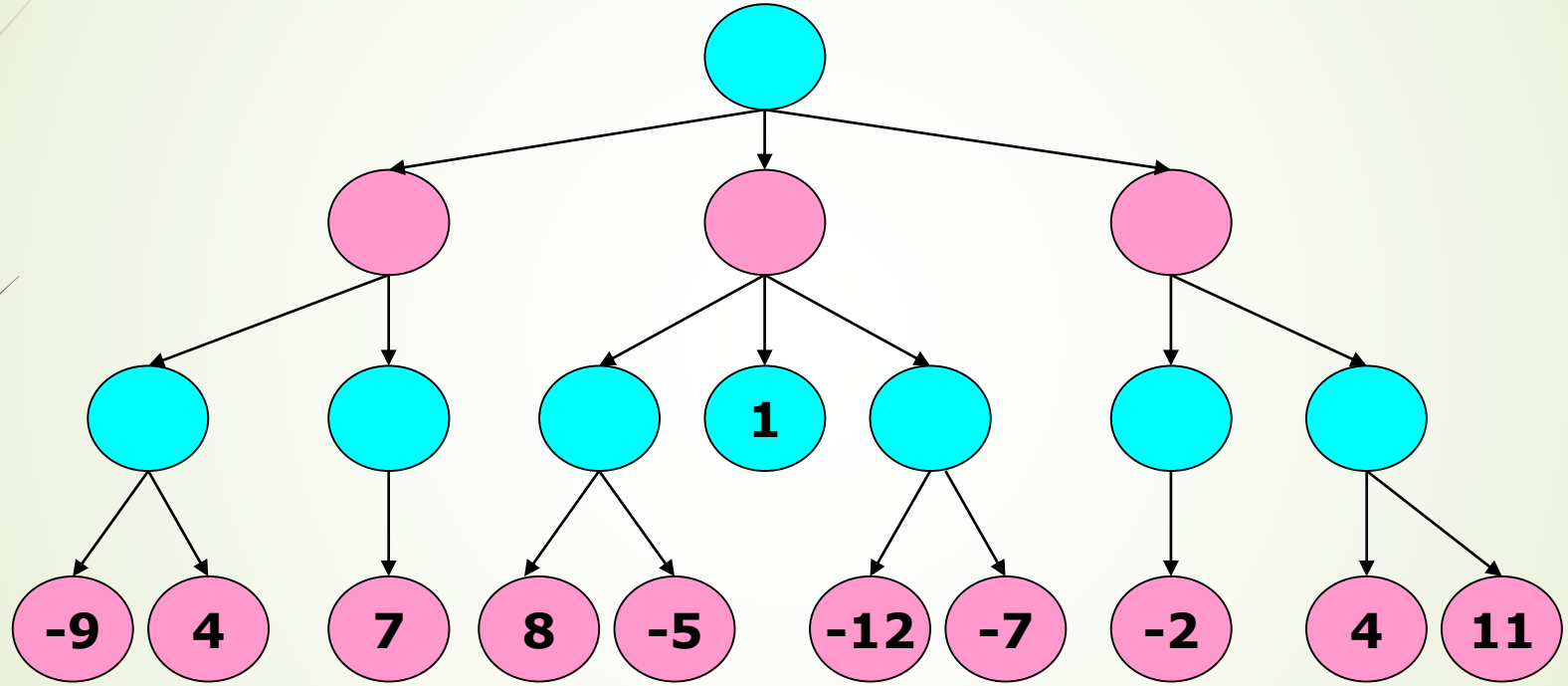


- Позиция выигрышна, нужно сделать левый из трех ходов

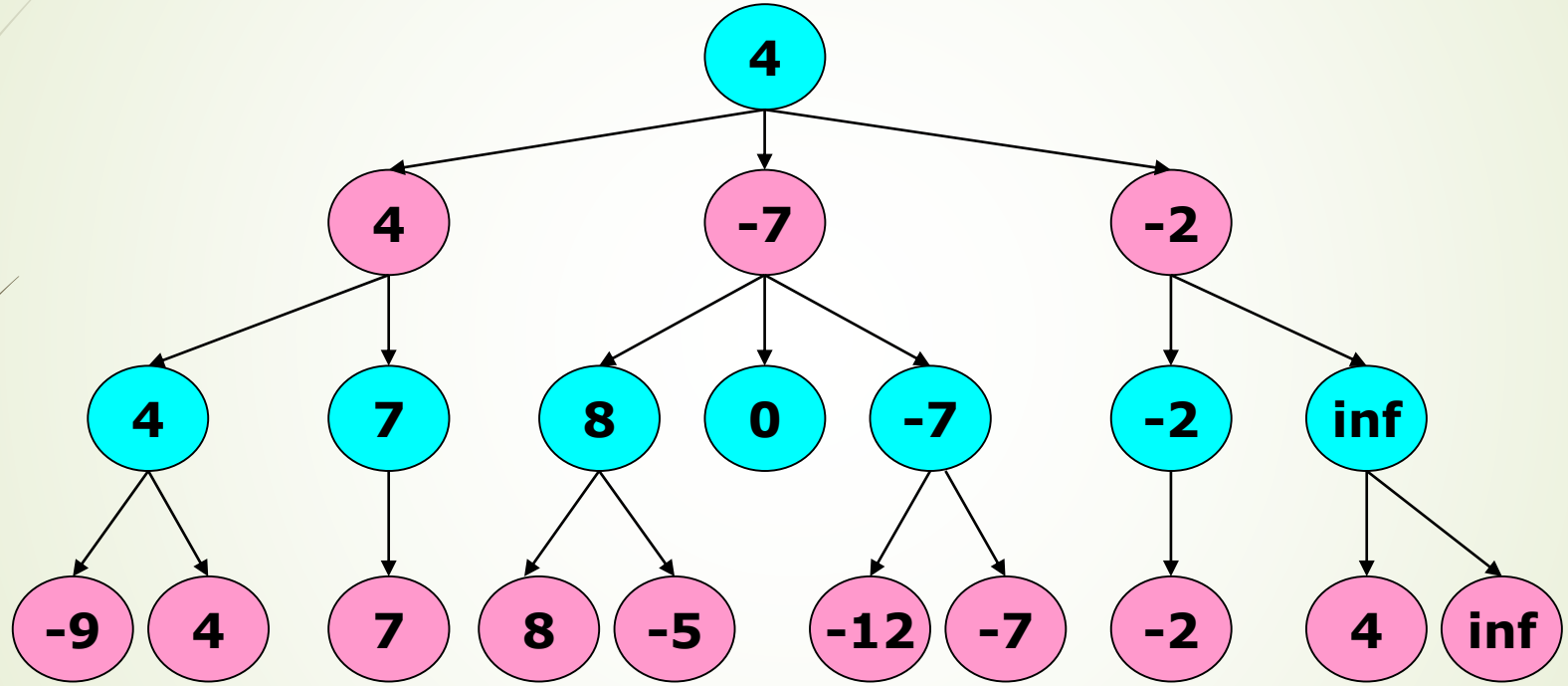
Пример: крестики-нолики 3x3

- См. пятый урок учебного проекта

Переборные алгоритмы – реальный вариант



Переборные алгоритмы - реальный вариант



- Нам лучше сделать левый из трех ходов - ***кажется***, результат там более симпатичен

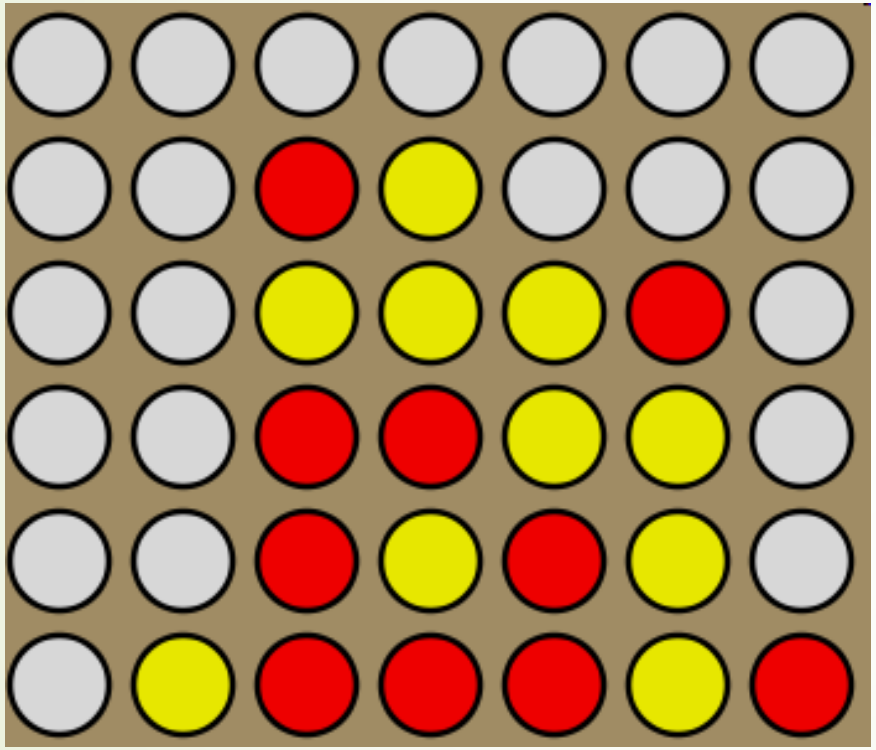
Как организовать перебор (минимакс)?

- Используется рекурсия
- Рекурсивная функция перебирает все варианты ходов из данной позиции, ее результатом должна быть оценка позиции и лучший ход
- Получив позиции уровнем ниже, для каждой из них мы вызываем ту же функцию
  - Точнее: мы либо имеем две функции, одна из которых формирует лучшую оценку, а другая худшую
  - Либо инвертируем оценку при рекурсивном вызове
- Выбираем ход, дающий наилучшую оценку

Как сформировать оценку позиции на самом нижнем уровне?

- Используется так называемая **оценочная функция**

Пример: четыре в ряд



Пример: четыре в ряд

- Оценочная функция – идеи?
  - Необходимо сформировать целое число, которое тем больше, чем лучше позиция для первого игрока, и тем меньше, чем лучше она для второго игрока

Пример: четыре в ряд

- Оценочная функция – идеи?
  - Необходимо сформировать целое число, которое тем больше, чем лучше позиция для первого игрока, и тем меньше, чем лучше она для второго игрока
- Примеры
  - Большой бонус обладателю четвёрки
  - Бонус поменьше обладателю тройки

Пример: четыре в ряд

- Оценочная функция – идеи?
  - Необходимо сформировать целое число, которое тем больше, чем лучше позиция для первого игрока, и тем меньше, чем лучше она для второго игрока
- Примеры
  - Большой бонус обладателю четвёрки
  - Бонус поменьше обладателю тройки
- Пример реализации:
  см. https://github.com/Kotlin-Polytech/FromKotlinToJava

Способы улучшения качества игры

- Совершенствование оценочной функции
- Использование алгоритмов ускорения перебора
- Гибкий выбор глубины перебора
- Оптимизация программы с целью увеличения глубины перебора

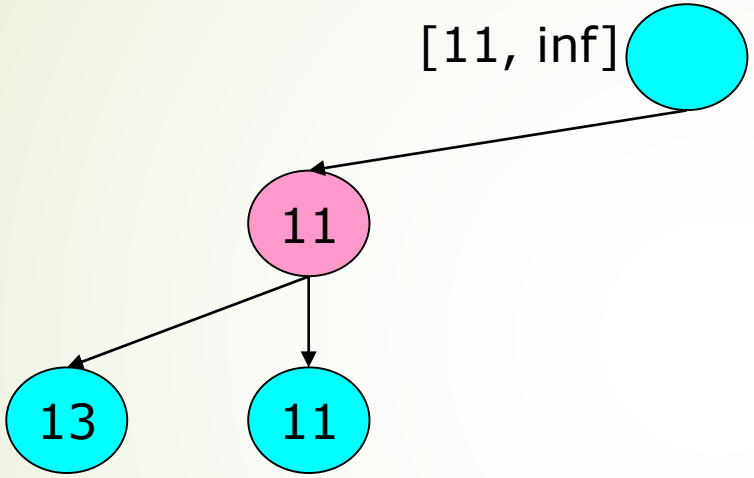
Совершенствование оценочной функции

- Перебираем все возможные четвёрки
- Ищем среди них те, в которых отсутствуют фишки одного цвета, и смотрим на количество фишек другого
  - 4 = Победа: 10000 (с модификацией на глубину)
  - 3: 500
  - 2: 10
  - 1: 1

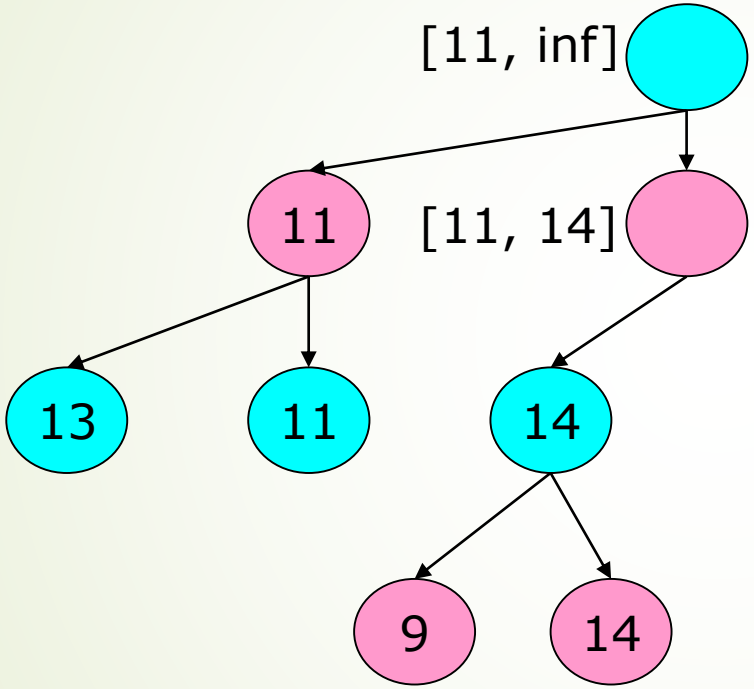
Алгоритмы ускорения перебора

- Альфа-бета процедура (он же - метод ветвей и границ)
- Алгоритм NEGASCOUT
- Алгоритм MTD(f)

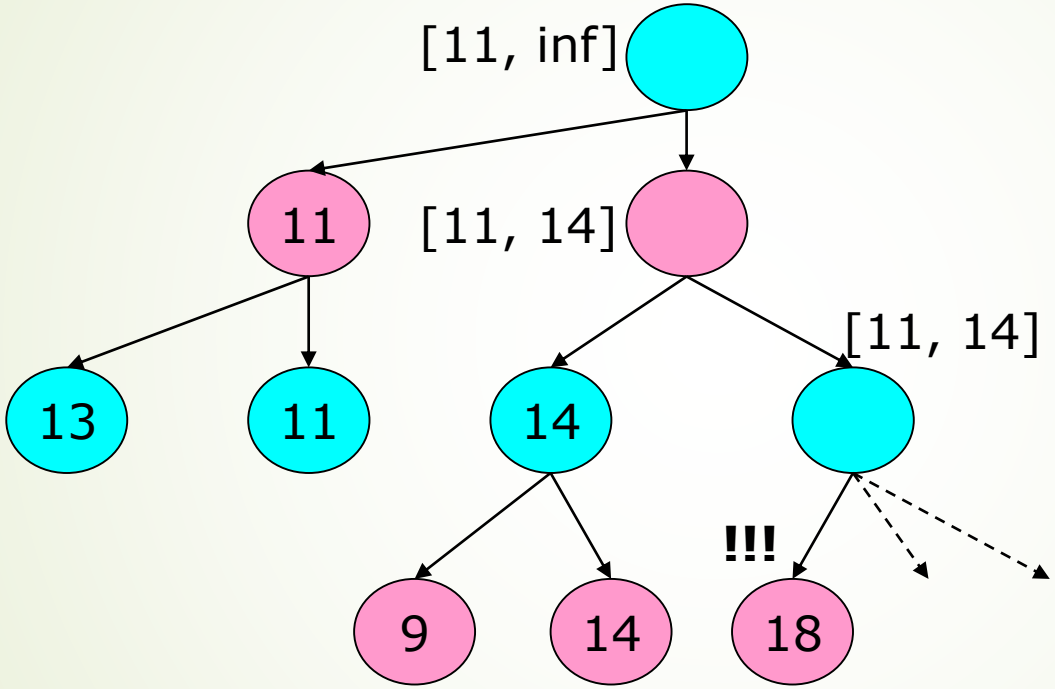
Альфа-бета процедура



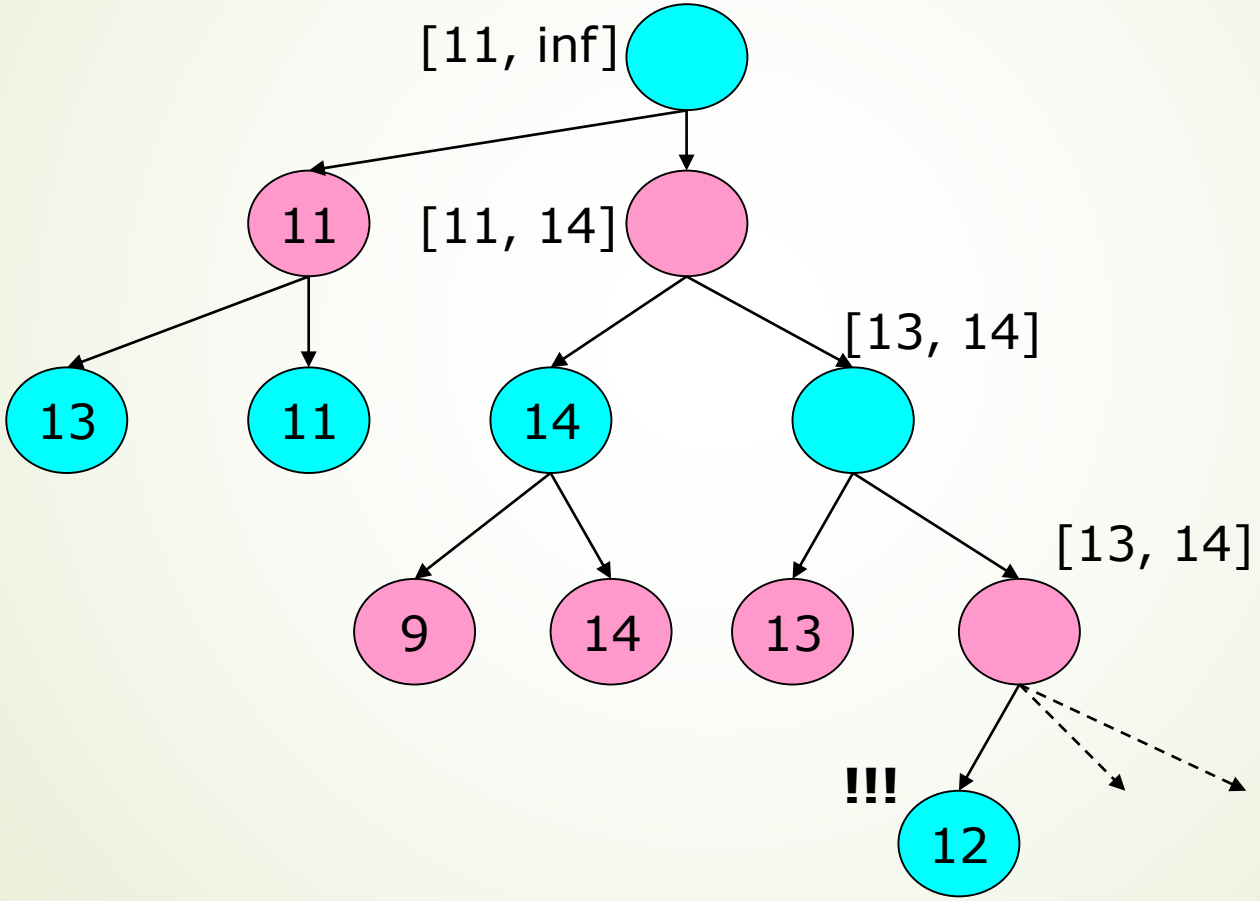
Альфа-бета процедура



Альфа-бета процедура



Альфа-бета процедура



Альфа-бета процедура - организация

- В функции перебора добавляются два аргумента - [alpha, beta]
- На верхнем уровне
  alpha=-inf, beta=+inf
- При переборе вариантов *alpha* является нижней границей; если нашли вариант с лучшей оценкой - *alpha* увеличивается
- При переборе вариантов *beta* является верхней границей; если нашли вариант с лучшей оценкой - перебор можно прервать
- На следующий уровень передаются текущие значения [alpha, beta] или [-beta, -alpha], в зависимости от того, чей на нем ход

Гибкий выбор глубины перебора

- Увеличивать глубину перебора, если число перебираемых вариантов невелико
  - Итеративный перебор с увеличением глубины
- Проверять глубже некоторые варианты
- …

Итоги

- Рассмотрено
  - Графы
  - Алгоритмы на основе поиска в ширину
    - Дейкстра, Волновой, A Star, Беллман-Форд
  - Алгоритмы на основе поиска в глубину
    - Перебор вариантов, минимакс, альфа-бета
- Далее
  - Разные задачи на графе
  - Динамическое программирование