Орграфы

Филипп Воронов

Филипп Воронов

Teamlead, Поиск Mail.ru

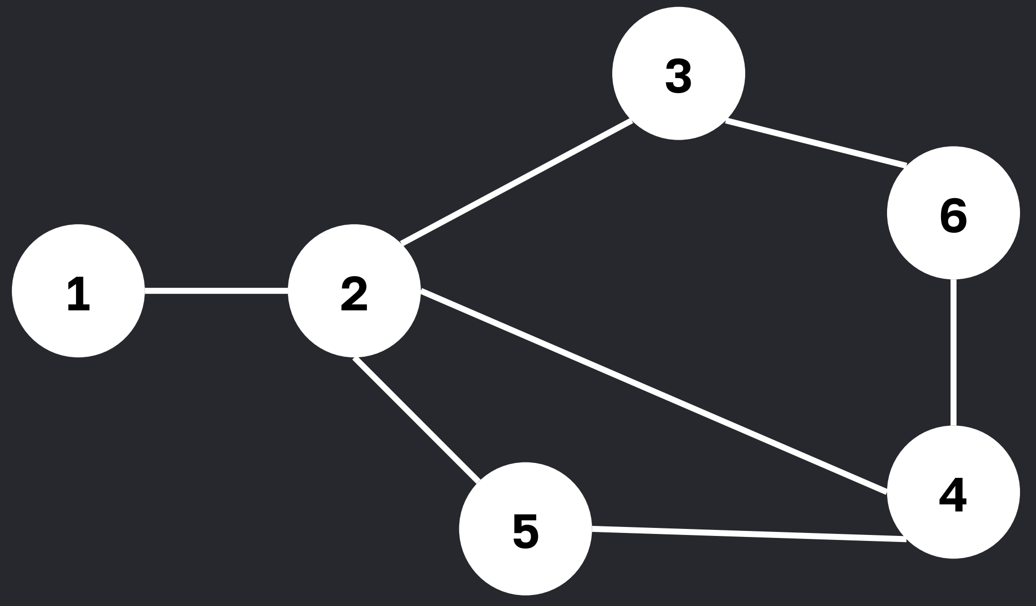
Аккаунты в соц.сетях

@Филипп Воронов

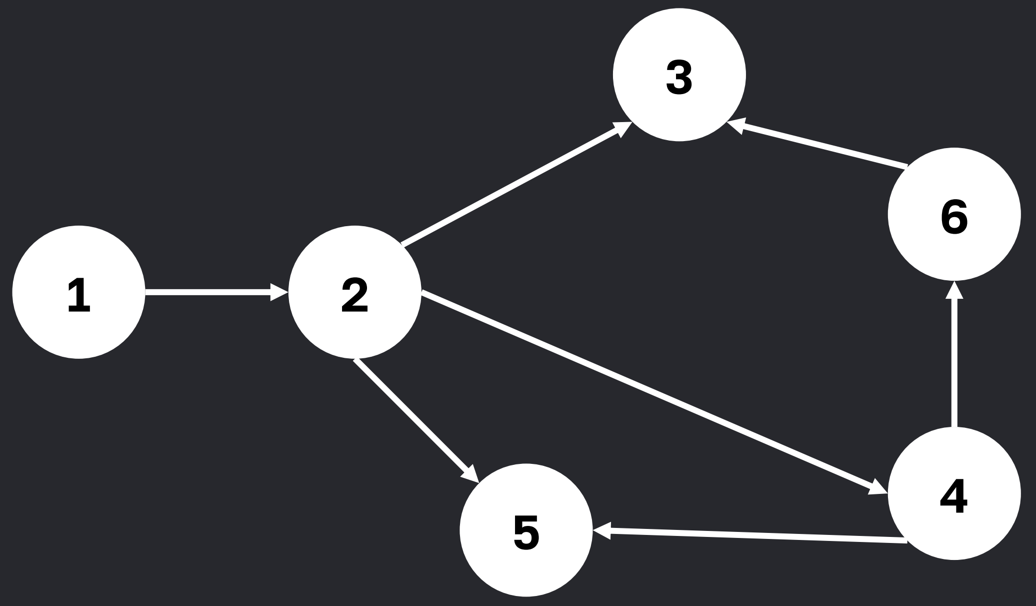
Что такое орграф?

Что такое орграф?

Орграфы или ориентированные графы — это обычные графы, у которых рёбра имеют направление.
Рёбра орграфов называются дугами.



граф



орграф

Если у обычных графов из одной вершины можно перейти в другую, это означает, что и обратно можно пройти. А в орграфах это не обязательно так.

Что такое орграф?

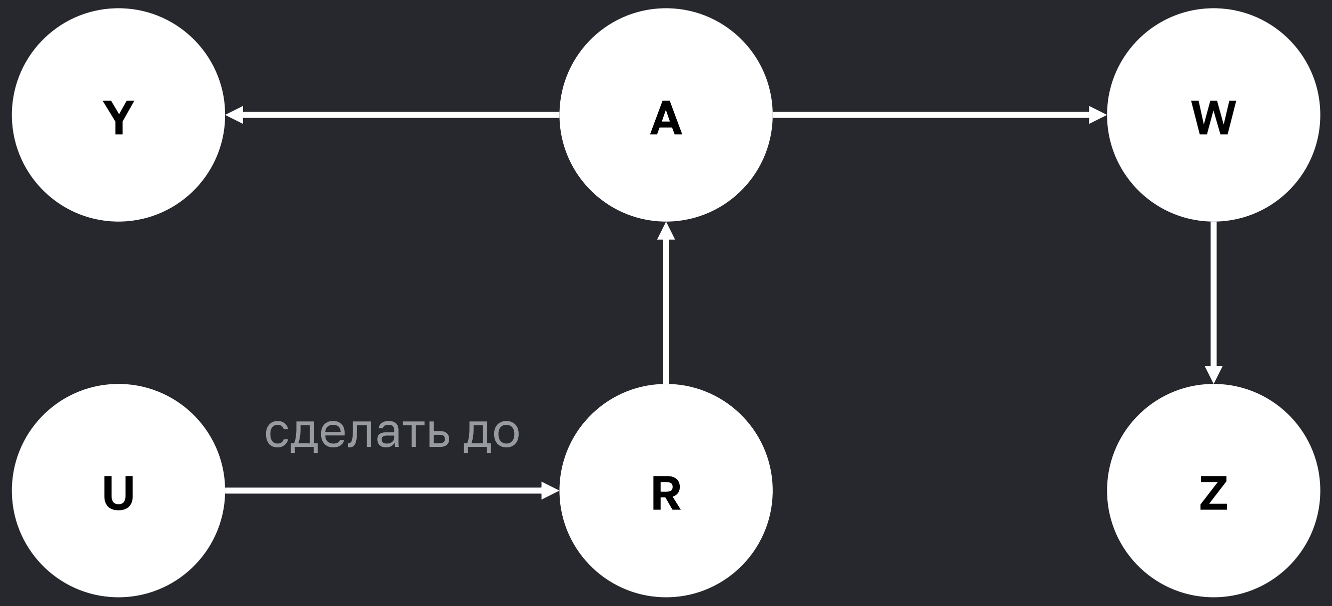
Многие понятия обычных графов переопределяются
и усложняются из-за направленности дуг.

Потому наборы вершин, где любые две достижимы друг из друга, называются компонентами сильной связности. Алгоритм их поиска гораздо сложнее обхода в глубину (см. алгоритм Косарайю).

Топологическая сортировка

Топологическая сортировка

Задача. Есть ориентированный граф.

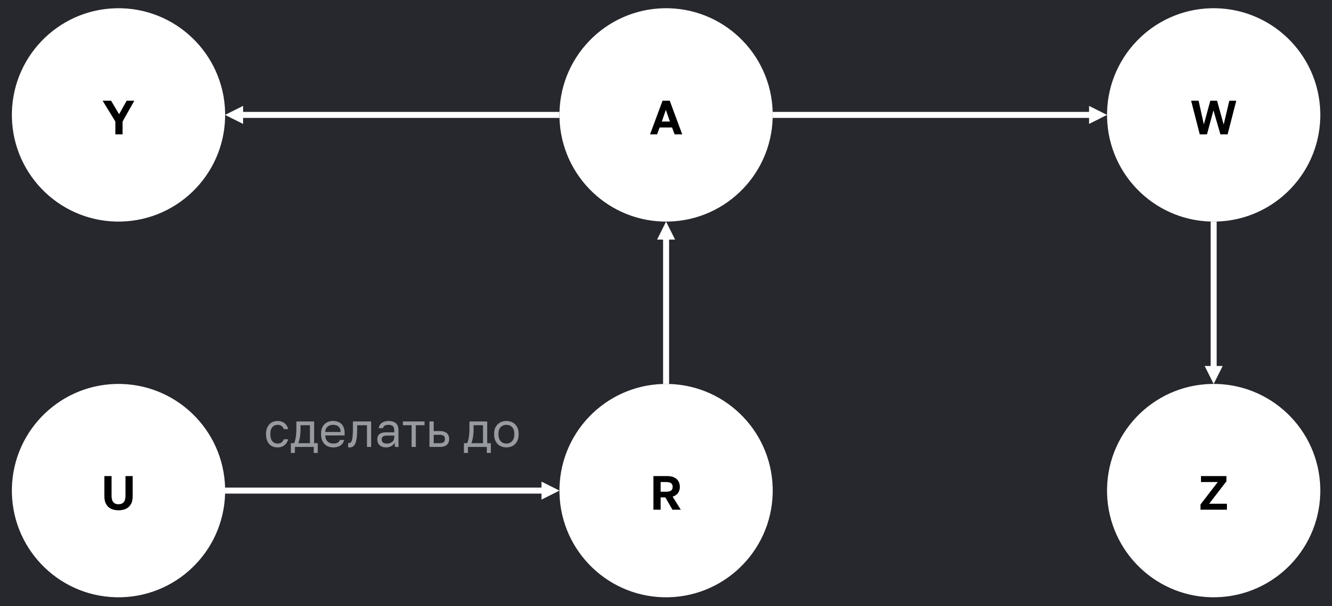


В вершинах хранятся задания, которые мы должны выполнить.
Дуга из задания «a» в задание «b» означает, что нельзя выполнять «b» до выполнения задания «a».

Необходимо написать программу, выводящую задания в порядке, возможном для исполнения.

Топологическая сортировка

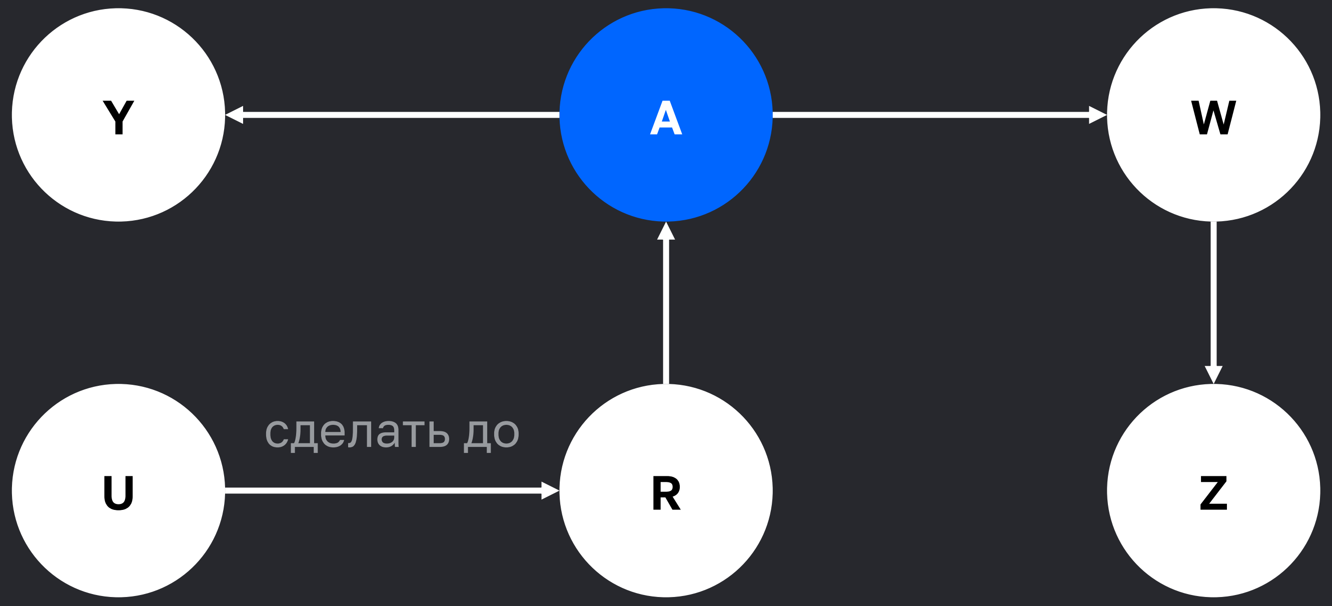
Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).



Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).

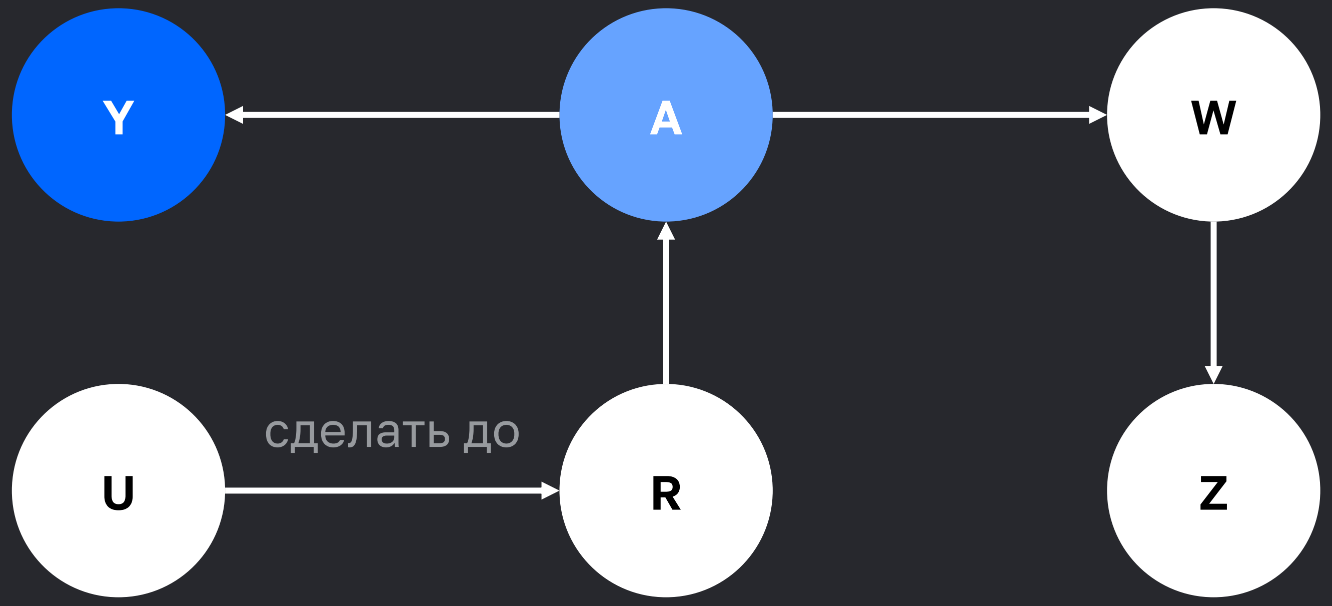


пустой список

Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).

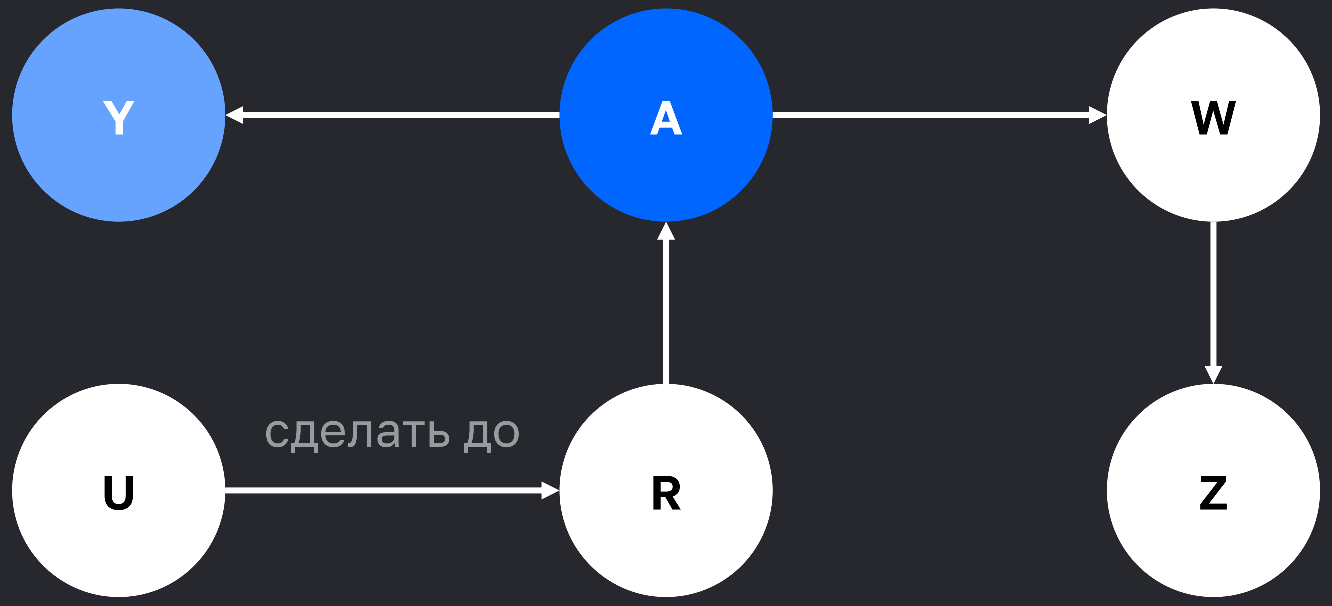


пустой список

Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).

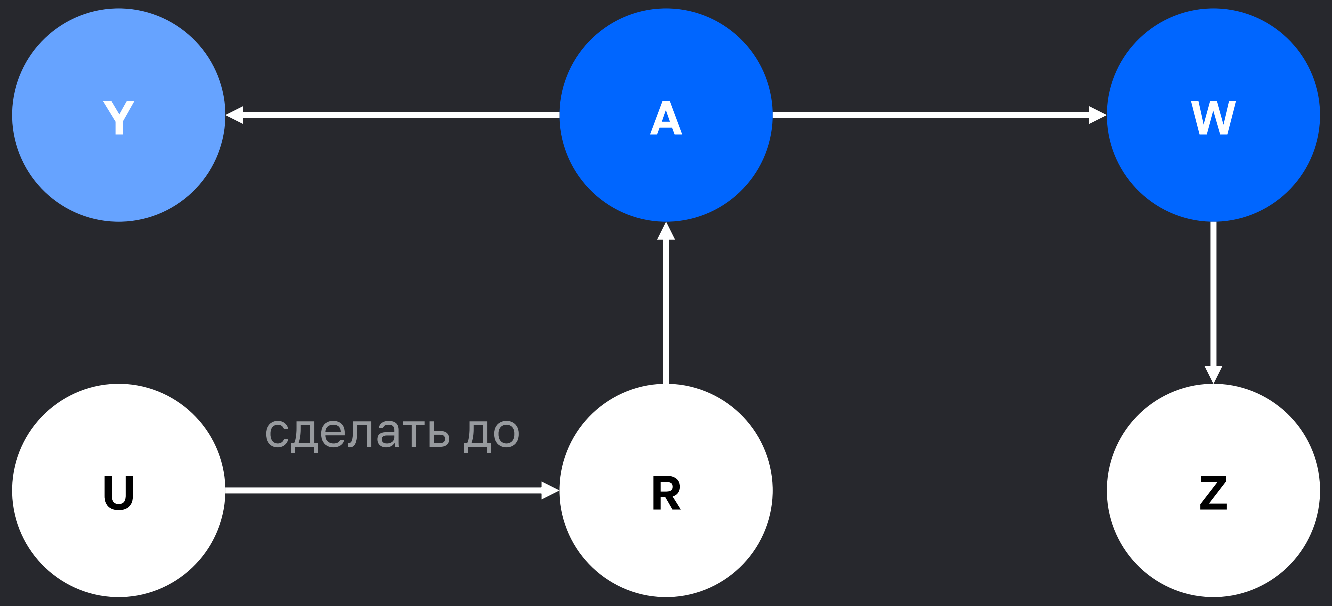


Y

Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).

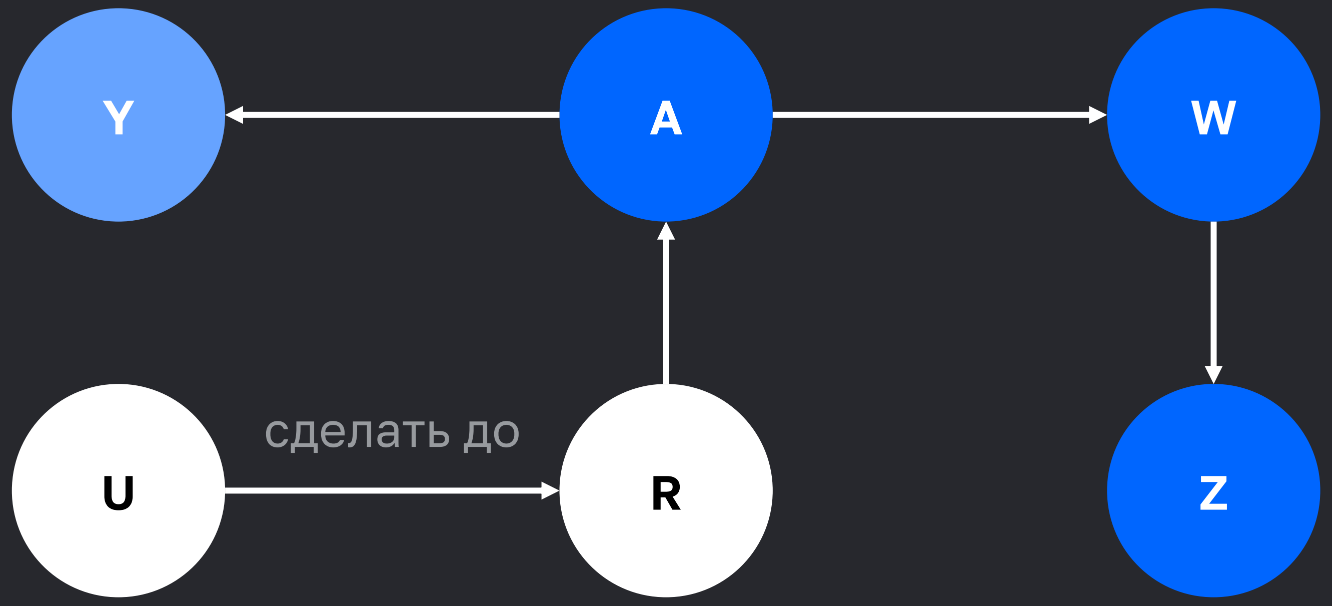


Y

Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).

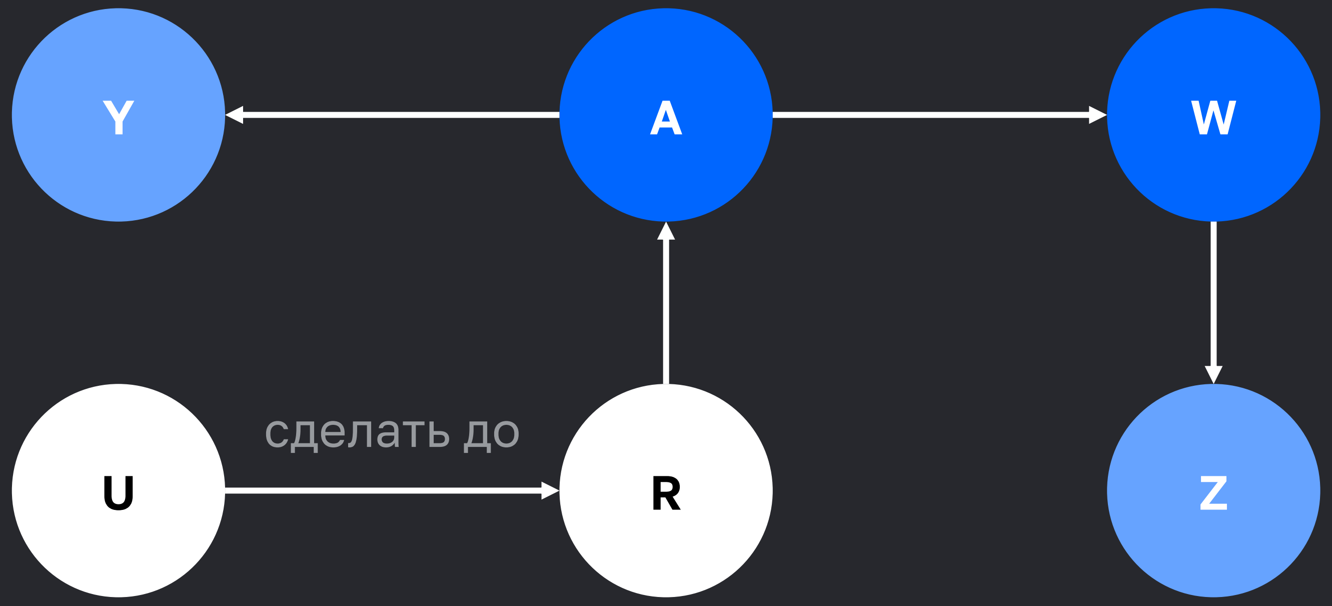


Y

Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).

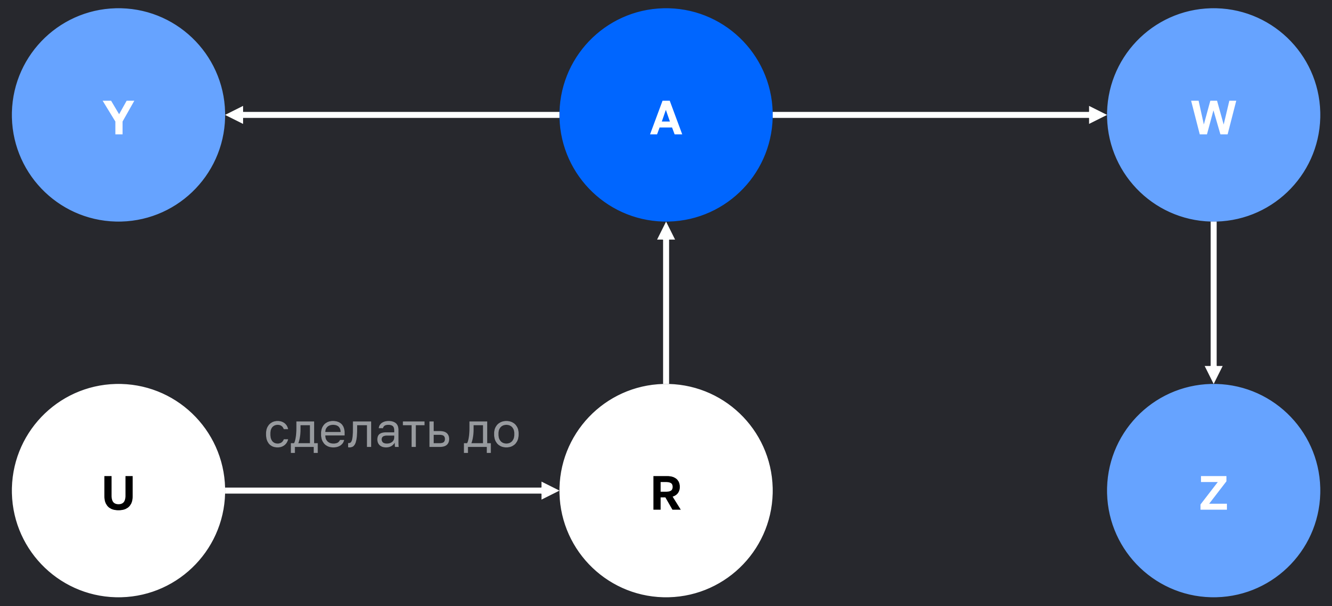


ZY

Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).

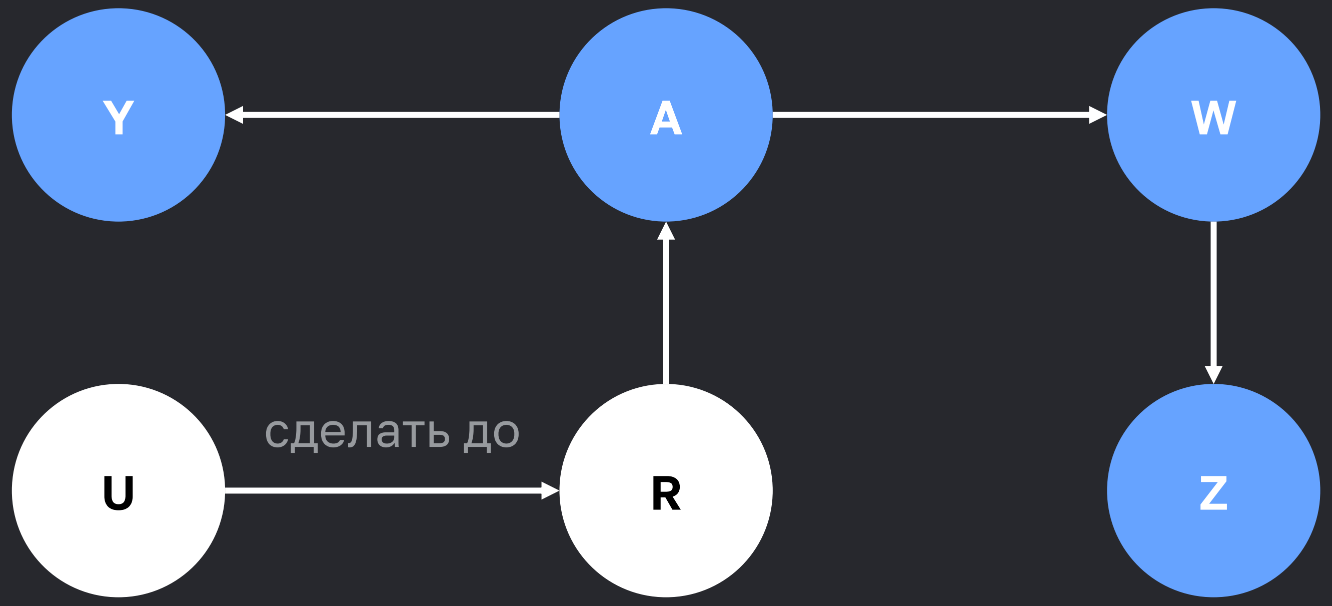


WZY

Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).

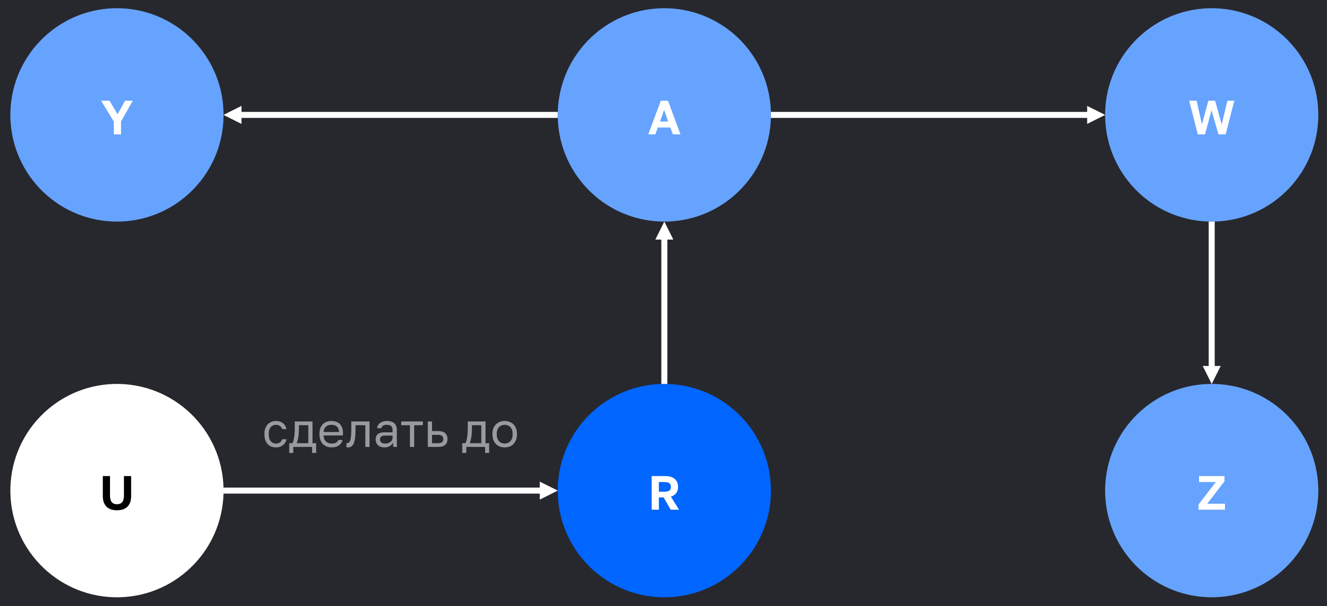


AWZY

Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).

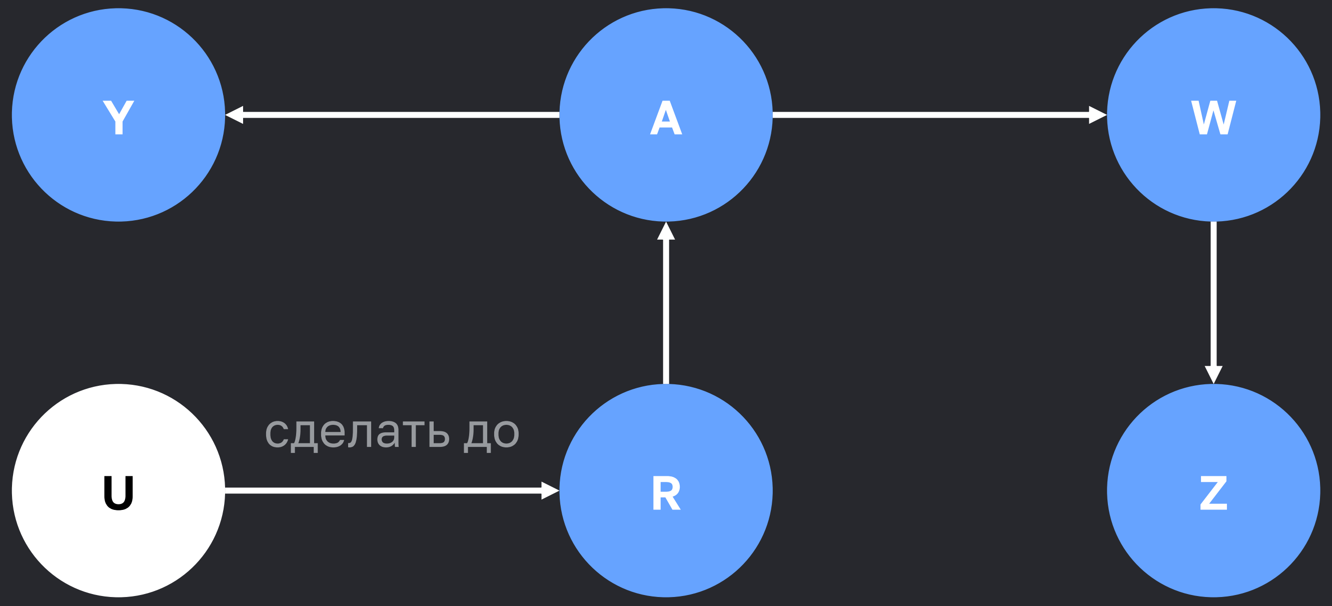


AWZY

Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).

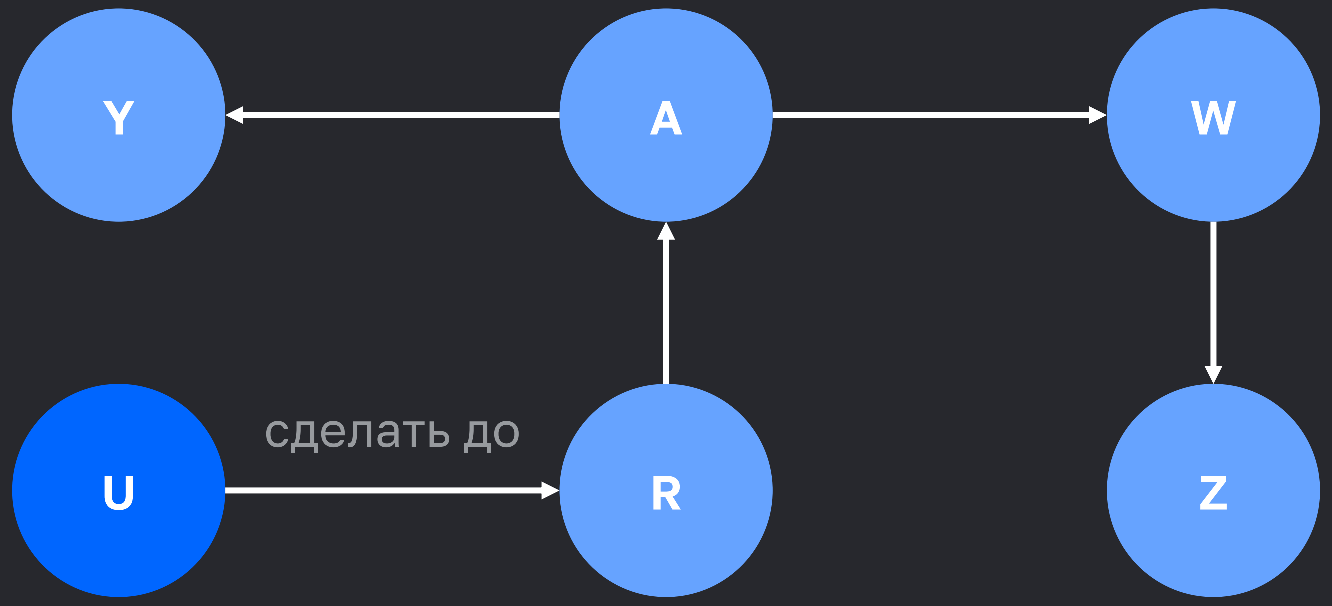


RAWZY

Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).

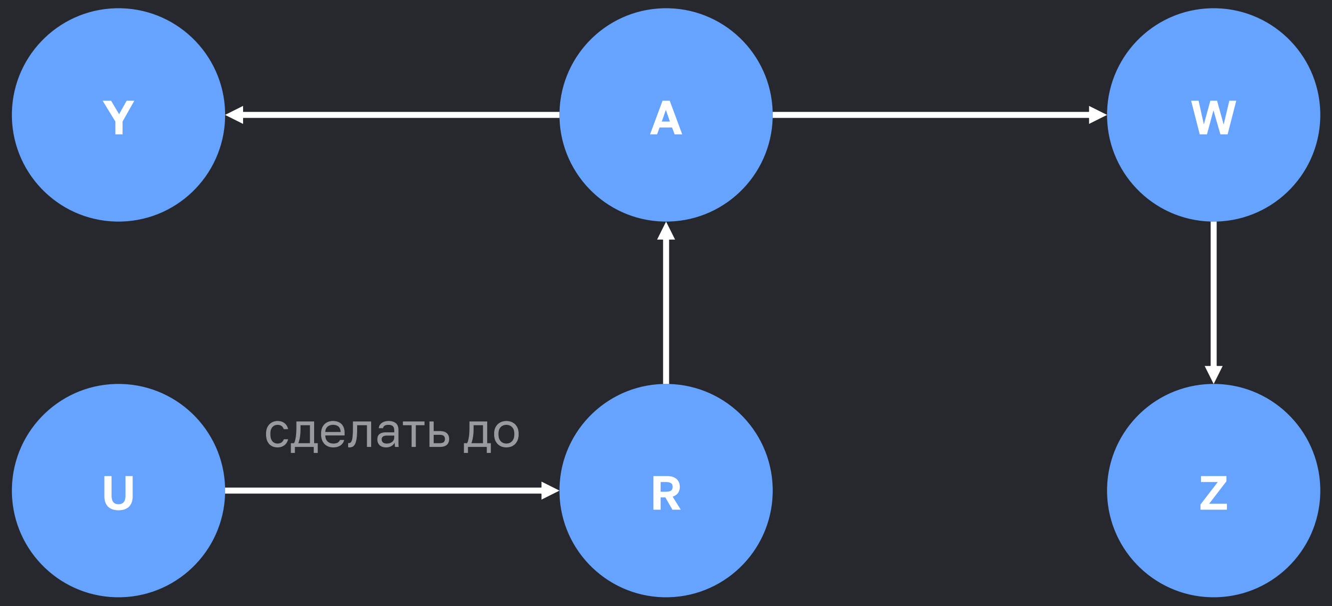


RAWZY

Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Решение: Топологическая сортировка. Допустим, что решение существует (что нет циклов).



URAWZY

Модифицируем опять обход в глубину. Будем строить ответ в списке order, который будем заполнять посещённой вершиной после посещения всех смежных из неё, вставляя её в начало списка.

Топологическая сортировка

Этот принцип гарантирует нам, что если вершина «b» достижима из «a», то «a» будет в списке раньше, чем «b».

Асимптотика работы совпадает с асимптотикой обхода в глубину.

Топологическая сортировка

Псевдокод:

```
dfs(graph, vertex, visited, order):
  visited[vertex] = да
  for v из смежные(vertex)
    if не visited[v]
      dfs(graph, v, visited, order)
  вставить vertex в начало order

top_sort(graph):
  order = []
  visited = [V раз нет]
  for v из вершины(graph)
    if не visited[v]
      dfs(graph, v, visited, order)
  return order
```

Комментарий
Модифицированный рекурсивный вызов dfs

Визуализация алгоритма https://visualgo.net/en/dfsbfs

Топологическая сортировка

Псевдокод:

```
dfs(graph, vertex, visited, order):
  visited[vertex] = да
  for v из смежные(vertex)
    if не visited[v]
      dfs(graph, v, visited, order)
  вставить vertex в начало order

top_sort(graph):
  order = []
  visited = [V раз нет]
  for v из вершины(graph)
    if не visited[v]
      dfs(graph, v, visited, order)
  return order
```

Комментарий

Всё как в старом dfs

Топологическая сортировка

Псевдокод:

```
dfs(graph, vertex, visited, order):
  visited[vertex] = да
  for v из смежные(vertex)
    if не visited[v]
      dfs(graph, v, visited, order)
  вставить vertex в начало order

top_sort(graph):
  order = []
  visited = [V раз нет]
  for v из вершины(graph)
    if не visited[v]
      dfs(graph, v, visited, order)
  return order
```

Комментарий
После обхода всех смежных вершин саму вершину добавляем в начало списка, в котором будем хранить вершины в топологическом порядке

Топологическая сортировка

Псевдокод:

```
dfs(graph, vertex, visited, order):
  visited[vertex] = да
  for v из смежные(vertex)
    if не visited[v]
      dfs(graph, v, visited, order)
  вставить vertex в начало order

top_sort(graph):
  order = []
  visited = [V раз нет]
  for v из вершины(graph)
    if не visited[v]
      dfs(graph, v, visited, order)
  return order
```

Комментарий
Топологическая сортировка вершин графа

Топологическая сортировка

Псевдокод:

```
dfs(graph, vertex, visited, order):
 visited[vertex] = да
 for v из смежные(vertex)
  if не visited[v]
   dfs(graph, v, visited, order)
 вставить vertex в начало order

top_sort(graph):
 order = []
 visited = [V раз нет]
 for v из вершины(graph)
  if не visited[v]
   dfs(graph, v, visited, order)
 return order
```

Комментарий
Список для ответа

Топологическая сортировка

Псевдокод:

```
dfs(graph, vertex, visited, order):
  visited[vertex] = да
  for v из смежные(vertex)
    if не visited[v]
      dfs(graph, v, visited, order)
  вставить vertex в начало order

top_sort(graph):
  order = []
  visited = [V раз нет]
  for v из вершины(graph)
    if не visited[v]
      dfs(graph, v, visited, order)
  return order
```

Комментарий

Обычный запуск рекурсивных обходов dfs

Топологическая сортировка

Псевдокод:

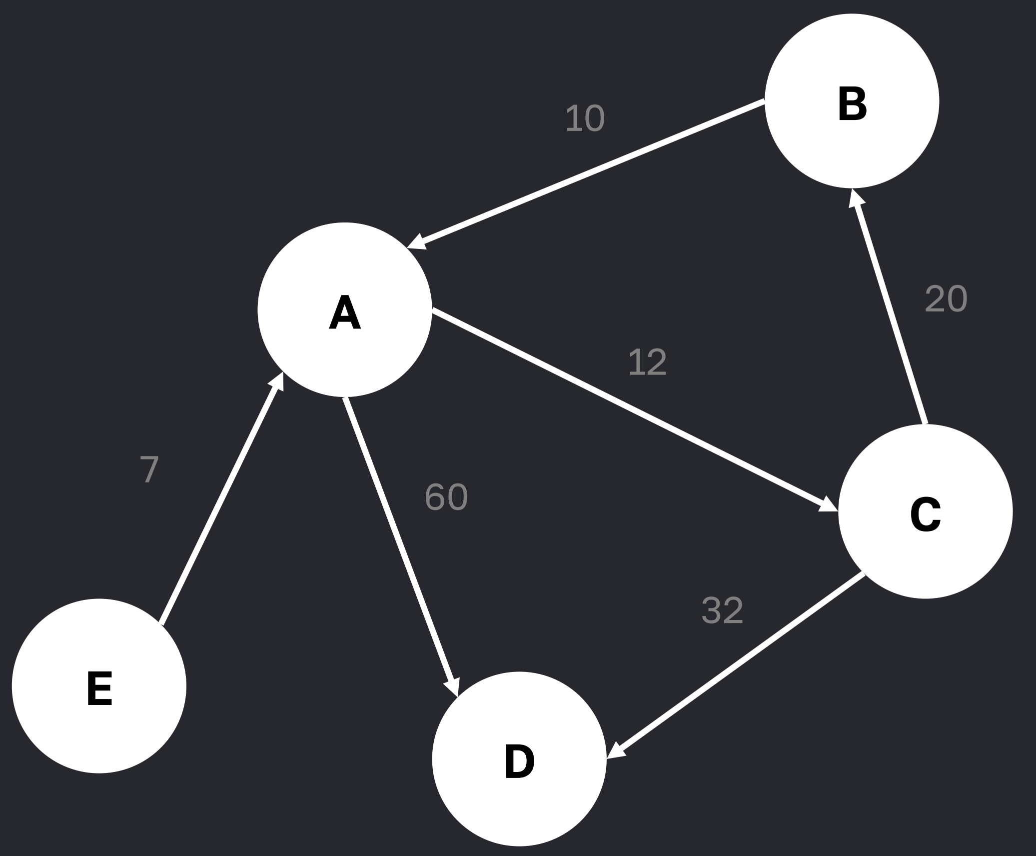
```
dfs(graph, vertex, visited, order):
  visited[vertex] = да
  for v из смежные(vertex)
    if не visited[v]
      dfs(graph, v, visited, order)
  вставить vertex в начало order

top_sort(graph):
  order = []
  visited = [V раз нет]
  for v из вершины(graph)
    if не visited[v]
      dfs(graph, v, visited, order)
  return order
```

Комментарий
После обхода в order у нас список вершин в топологическом порядке

Алгоритм Дейкстры

Взвешенный орграф

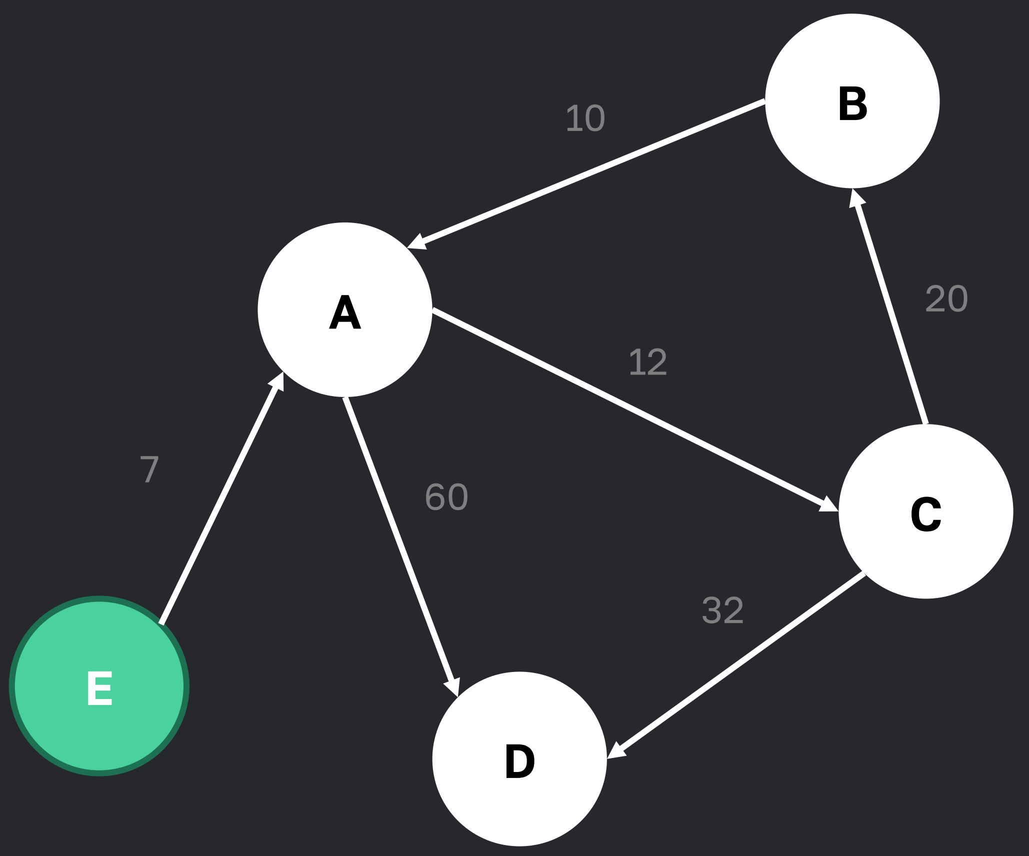
Если мы припишем каждому ребру или дуге число, то такой граф или орграф называется взвешенным, а эти числа — весами рёбер или дуг.



Сумма весов всех рёбер или дуг в пути называется весом пути.

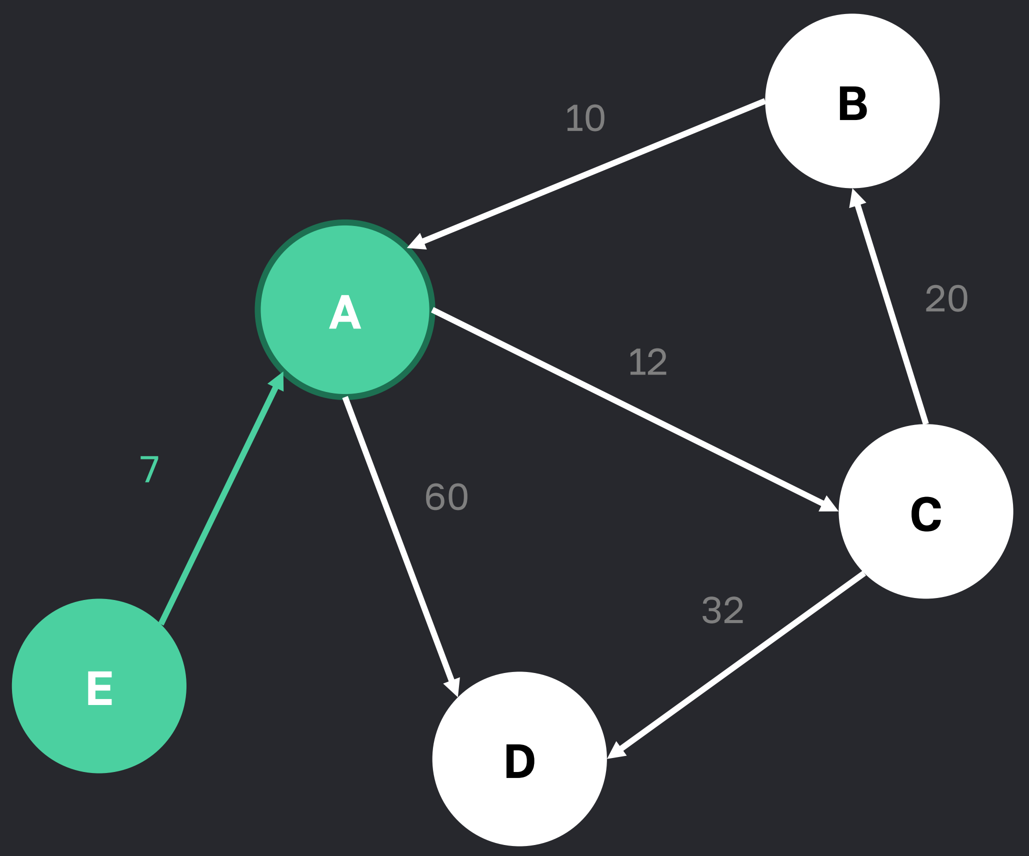
Кратчайшие пути

Задача. Путь из одной вершины в другую называется кратчайшим, если у него минимальный вес из всех таких путей.



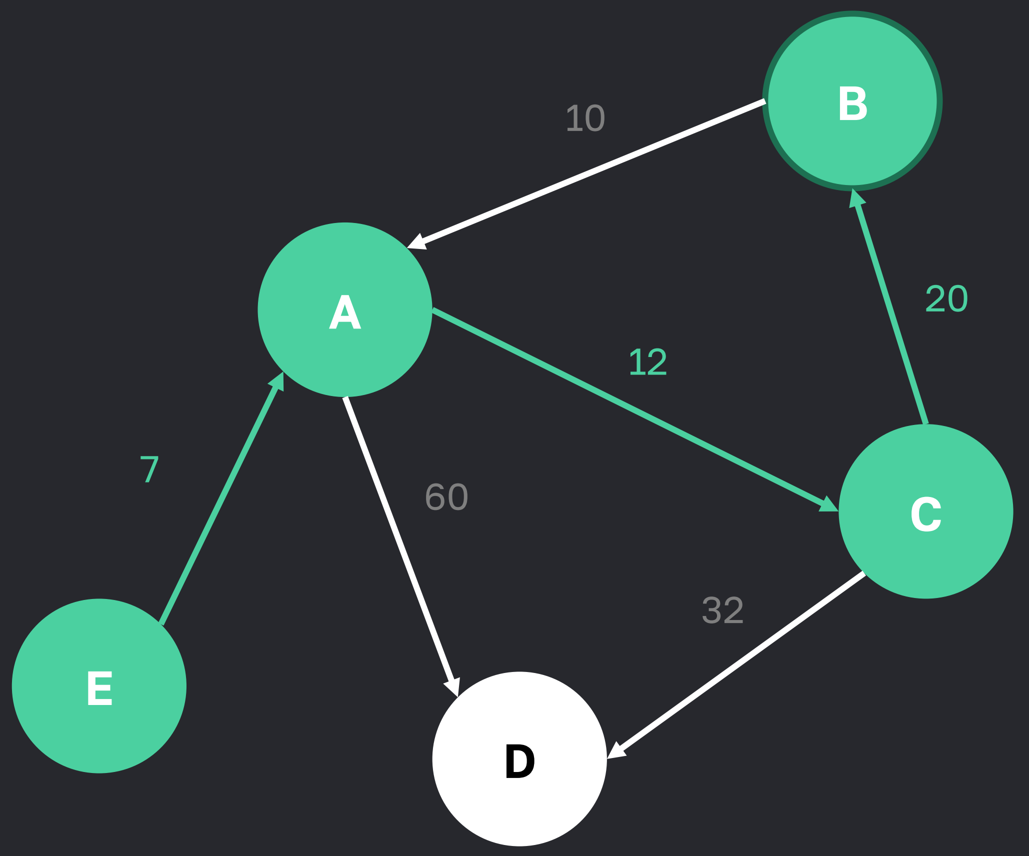
Необходимо найти для заданной вершины все кратчайшие пути до всех остальных вершин.

Кратчайшие пути



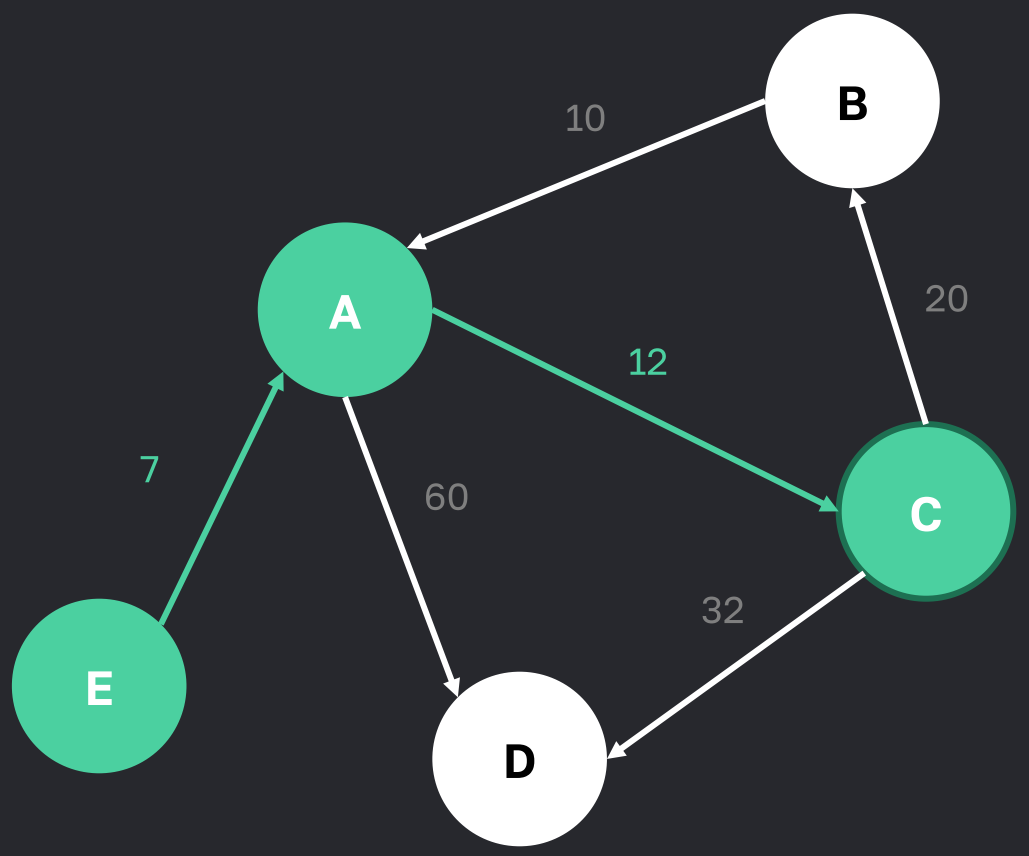
Кратчайшие пути из «E»		
до вершины	вес	путь
A	7	→ A

Кратчайшие пути



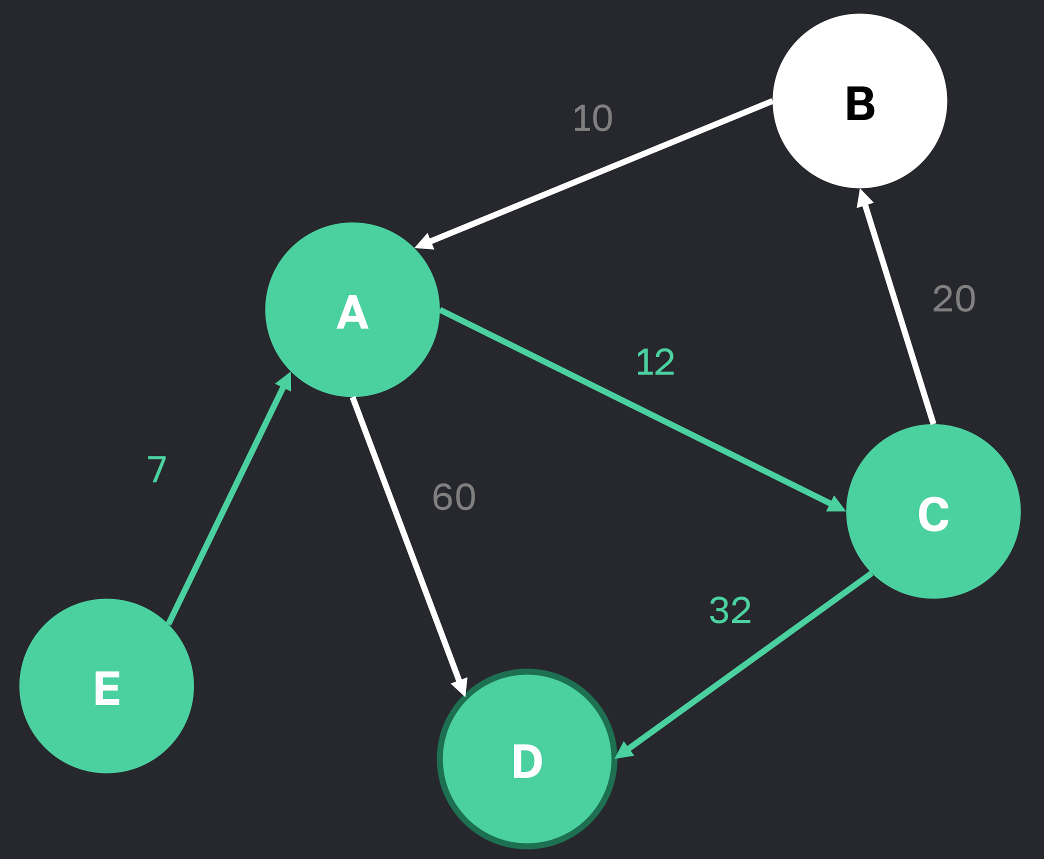
Кратчайшие пути из «E»		
до вершины	вес	путь
A	7	→ A
B	39	→ A → C → B

Кратчайшие пути



Кратчайшие пути из «E»		
до вершины	вес	путь
A	7	→ A
B	39	→ A → C → B
C	19	→ A → C

Кратчайшие пути



Кратчайшие пути из «E»		
до вершины	вес	путь
A	7	→ A
B	39	→ A → C → B
C	19	→ A → C
D	51	→ A → C → D

Алгоритм Дейкстры

Алгоритм имеет сложный инвариант и использует несколько вспомогательных структур. Сперва рассмотрим упрощённую задачу: найти для каждой вершины не кратчайший путь, а лишь его длину.

Алгоритм Дейкстры

Алгоритм имеет сложный инвариант и использует несколько вспомогательных структур. Сперва рассмотрим упрощённую задачу: найти для каждой вершины не кратчайший путь, а лишь его длину.

Помеченные вершины — ассоциативный массив с вершинами как ключами и весами минимальных путей как значениями. Для них минимальный путь уже найден, также мы учли все пути, в которых они были предпоследними вершинами. Изначально пустое.

Алгоритм Дейкстры

Алгоритм имеет сложный инвариант и использует несколько вспомогательных структур. Сперва рассмотрим упрощённую задачу: найти для каждой вершины не кратчайший путь, а лишь его длину.

Помеченные вершины — ассоциативный массив с вершинами как ключами и весами минимальных путей как значениями. Для них минимальный путь уже найден, также мы учли все пути, в которых они были предпоследними вершинами. Изначально пустое.

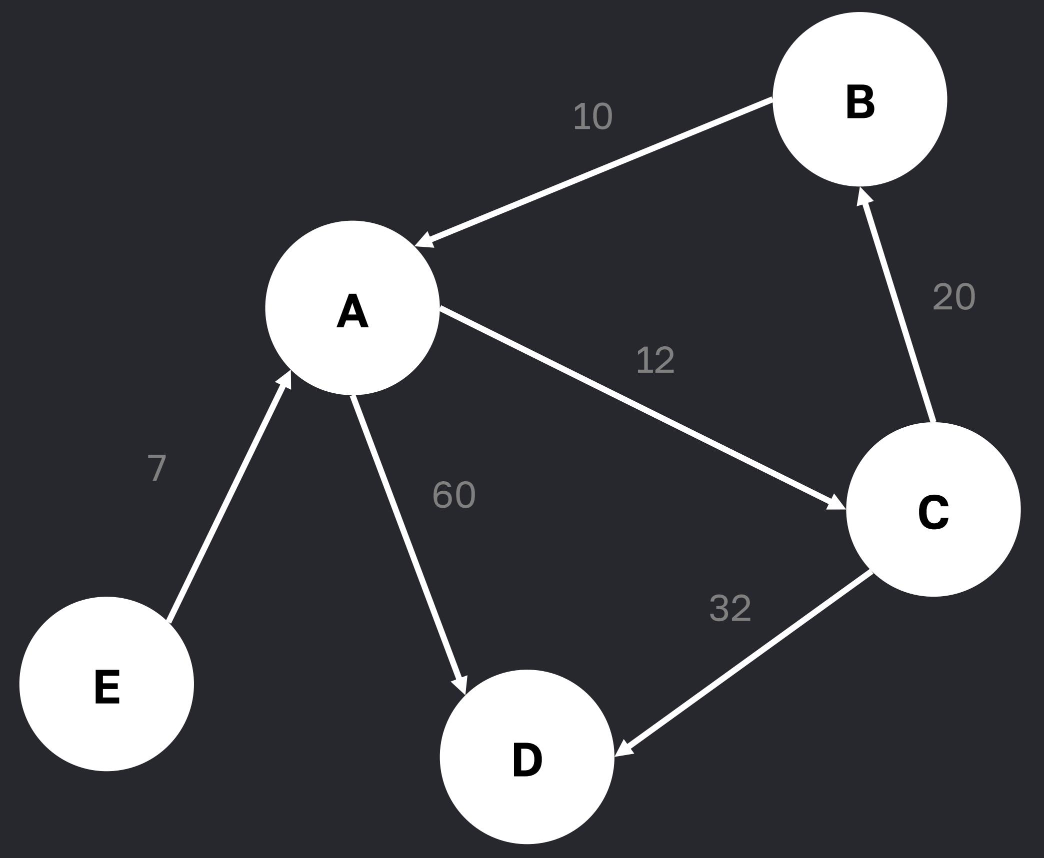
Пирамида прикидок — пирамида на минимум непомеченных вершин, где ключом является наша текущая прикидка длины самого короткого пути. Изначально только у начальной вершины ключ 0 (кратчайший путь до неё самой = никуда ходить не надо), у остальных ключ — бесконечность (про остальные пока не знаем).

Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

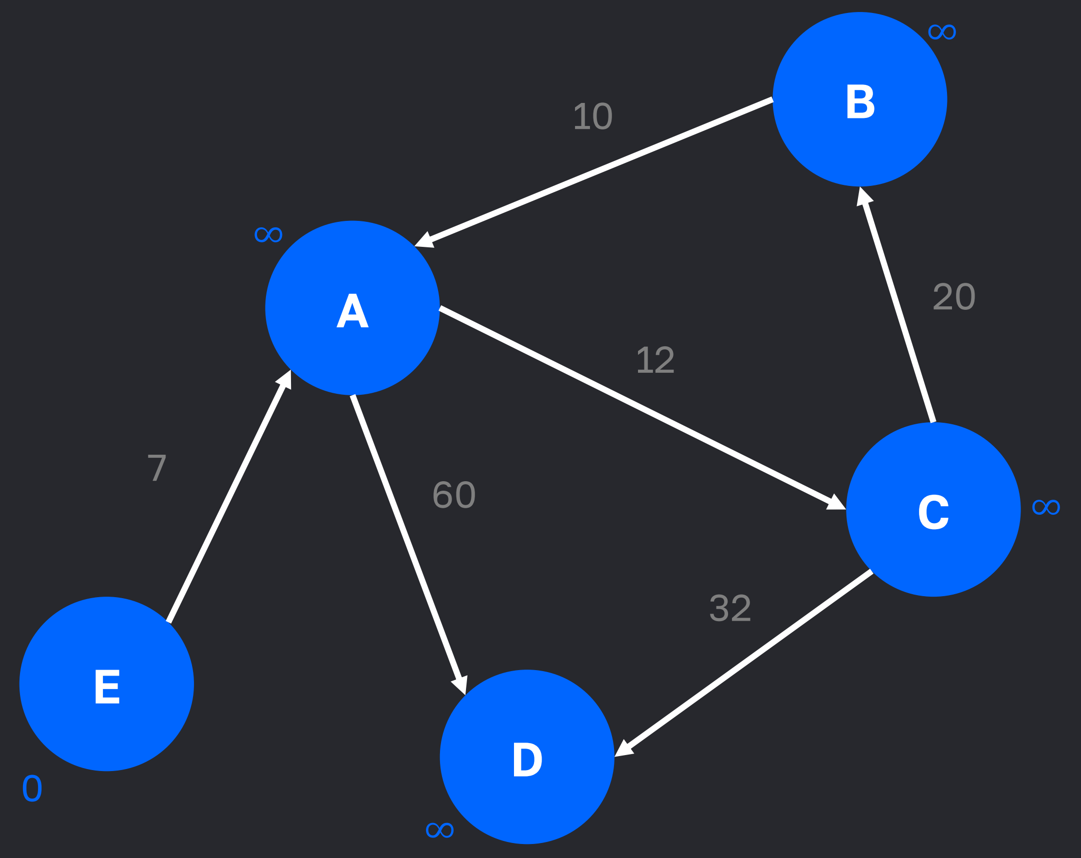


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

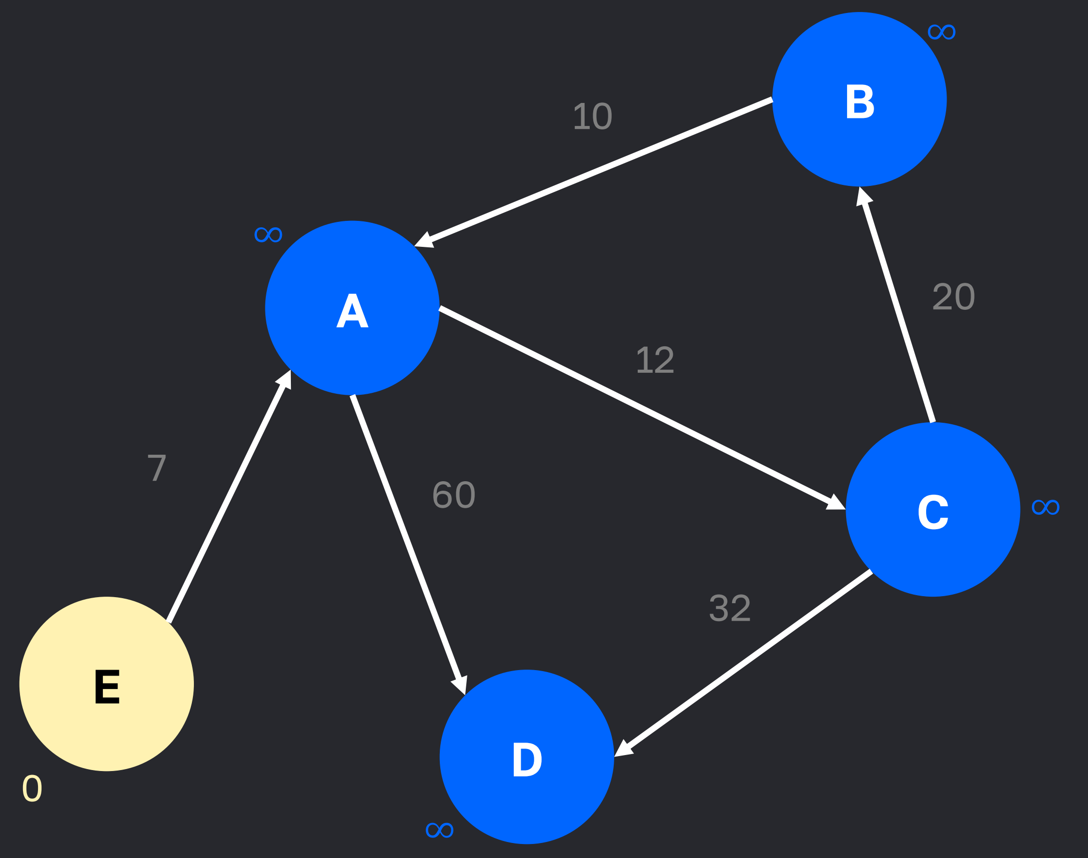


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

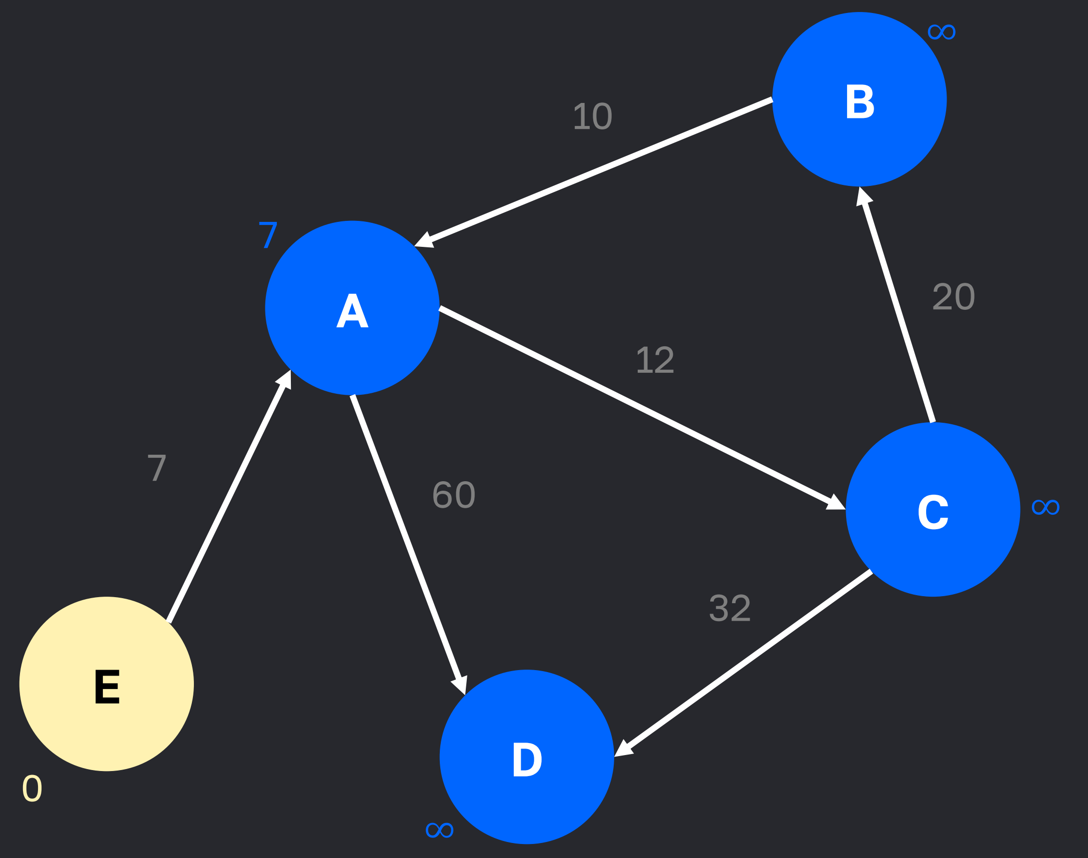


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

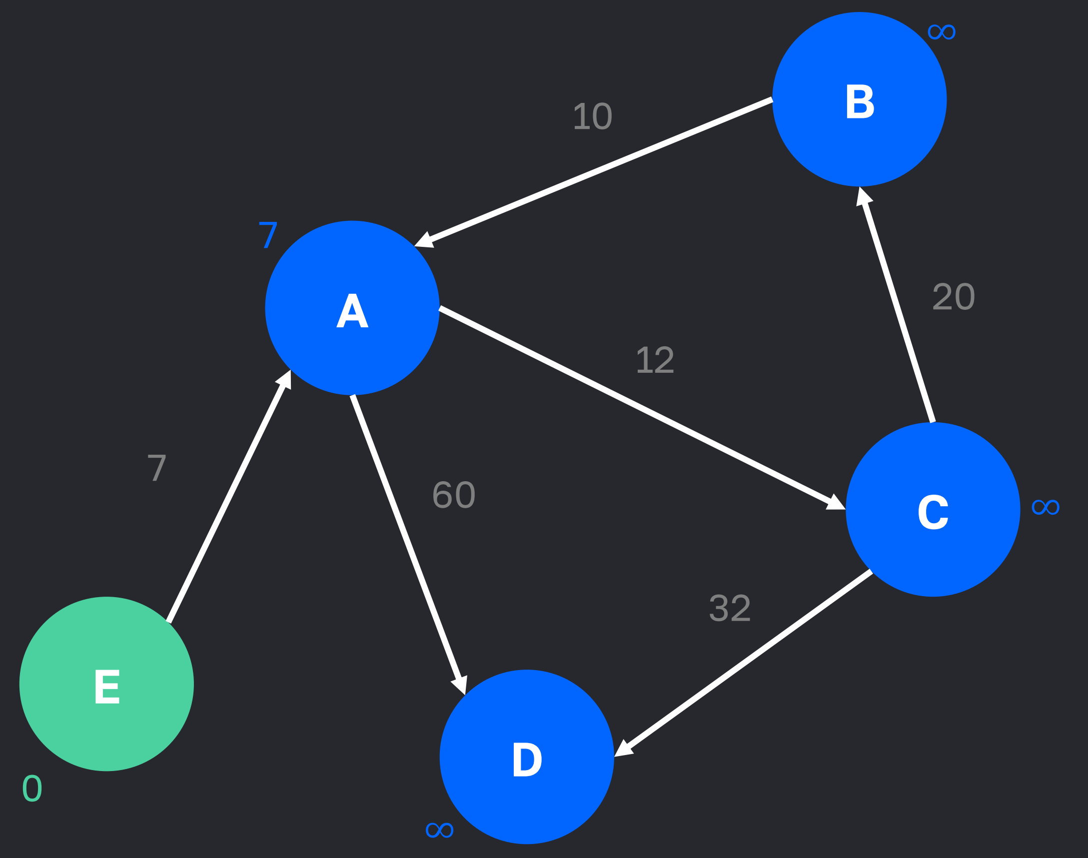


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

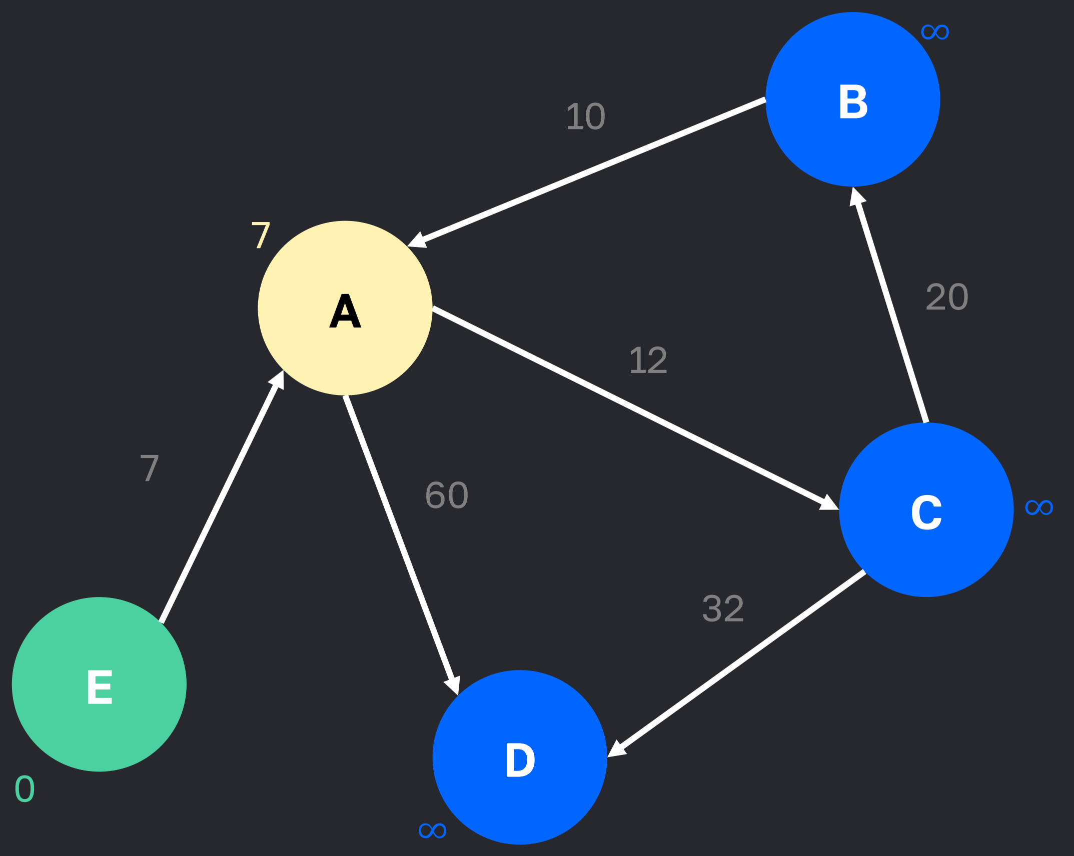


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

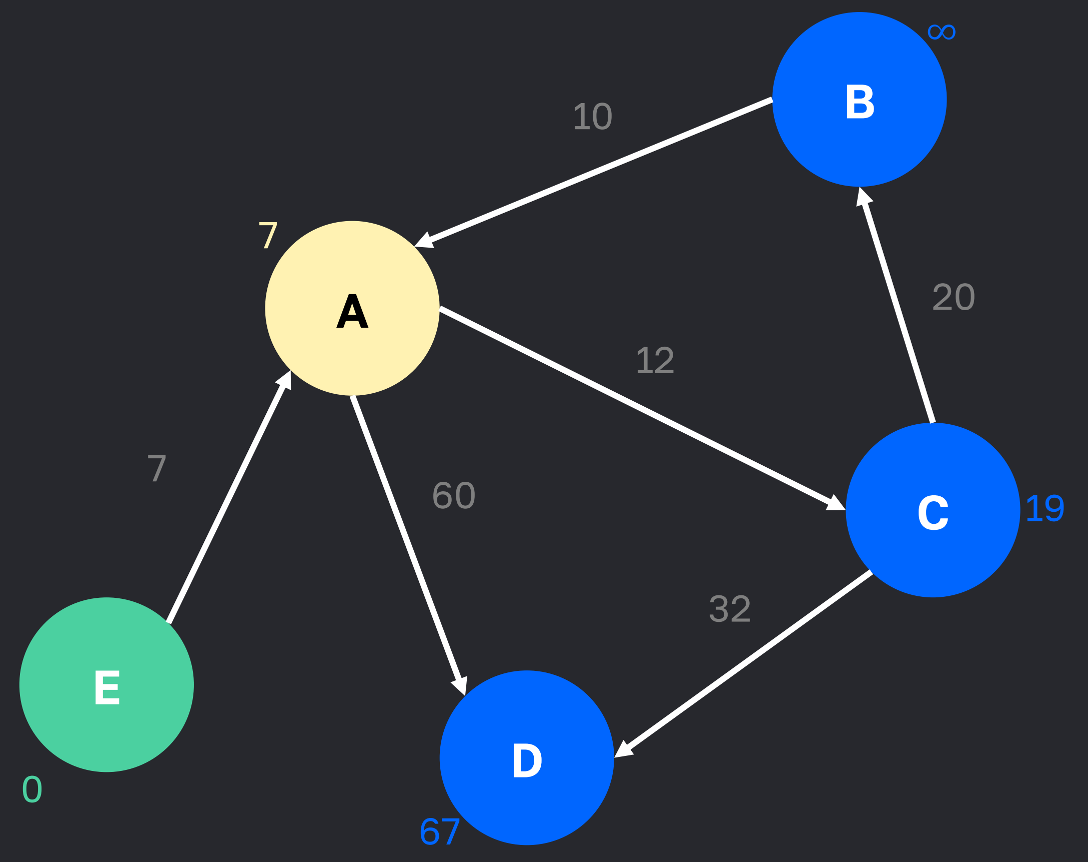


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

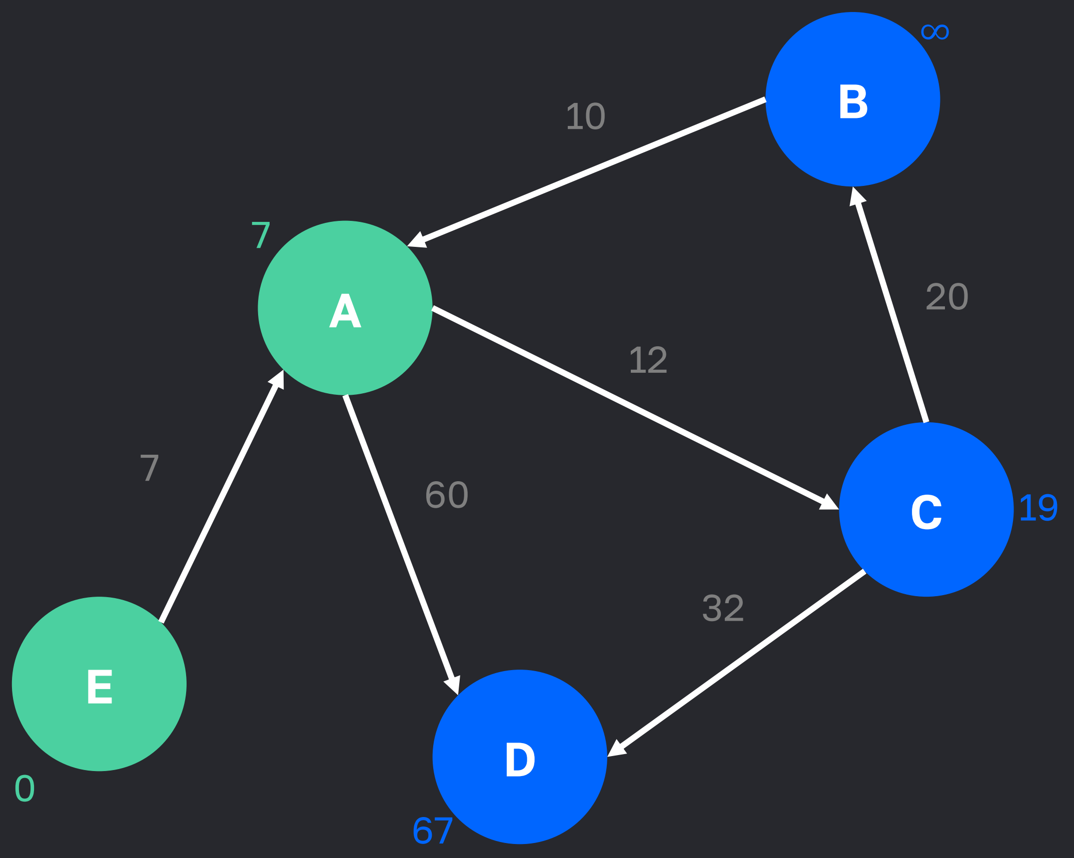


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

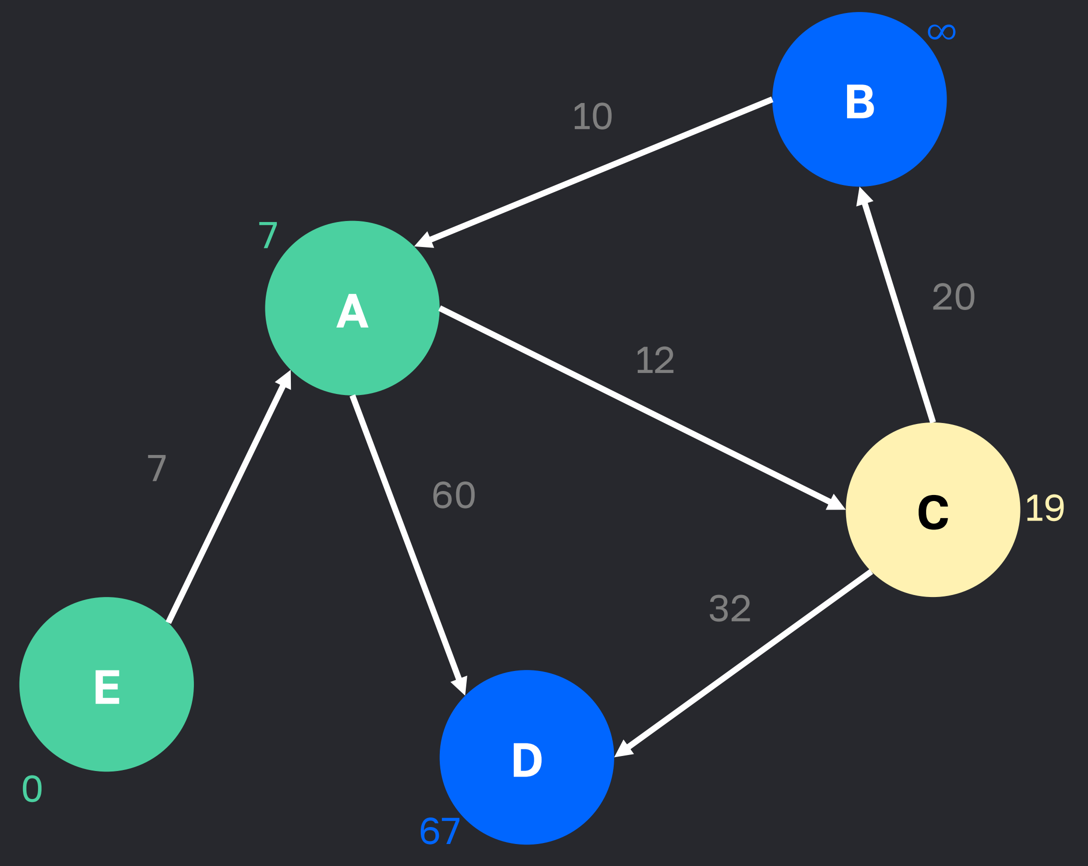


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

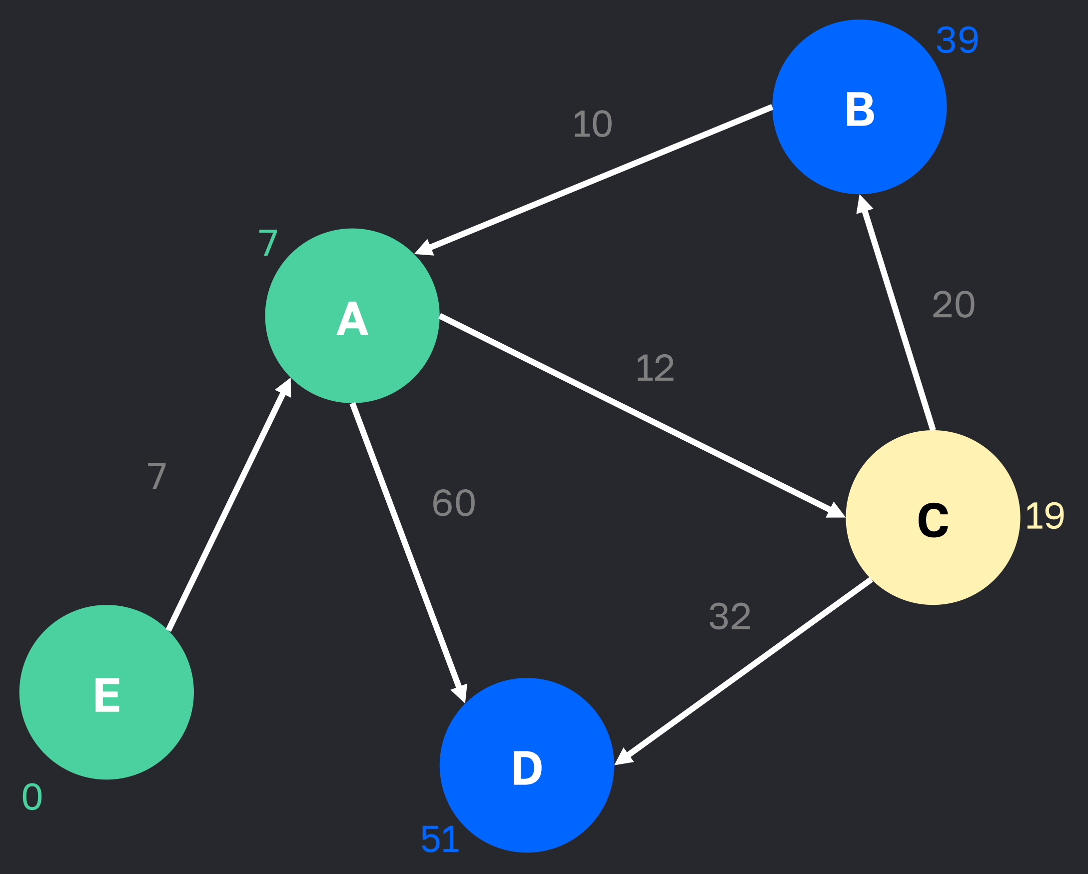


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

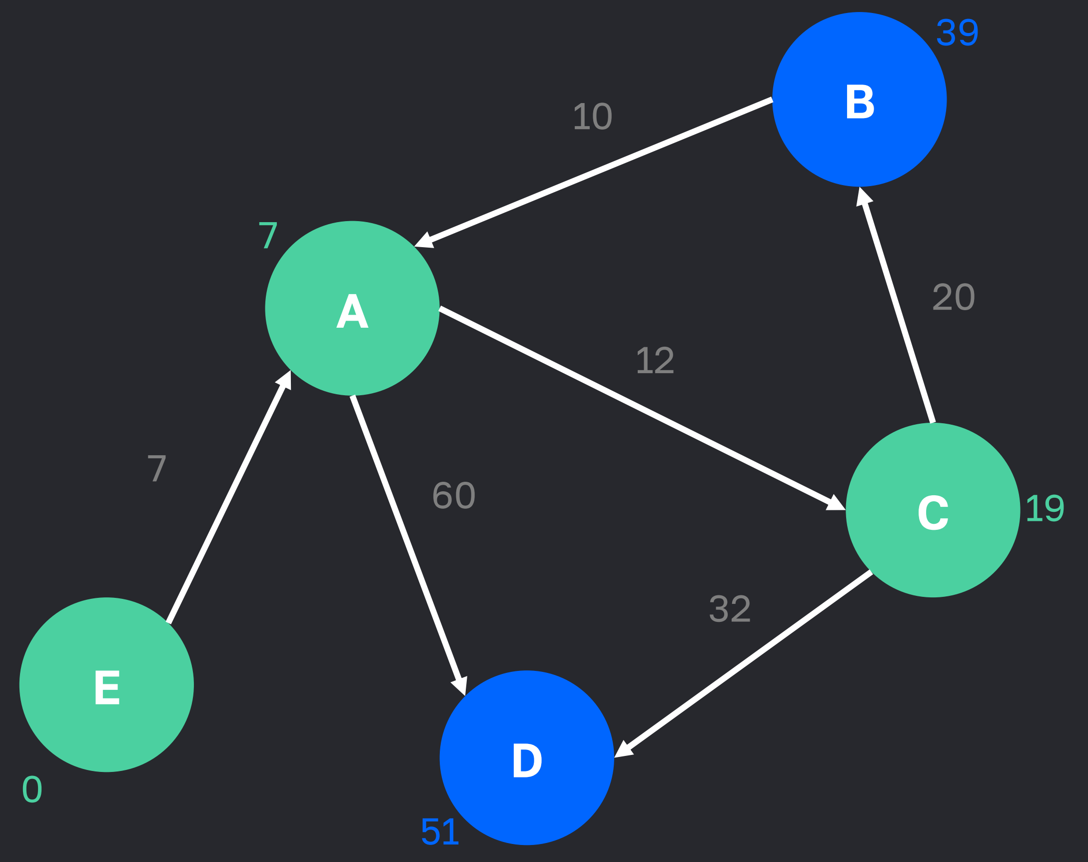


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

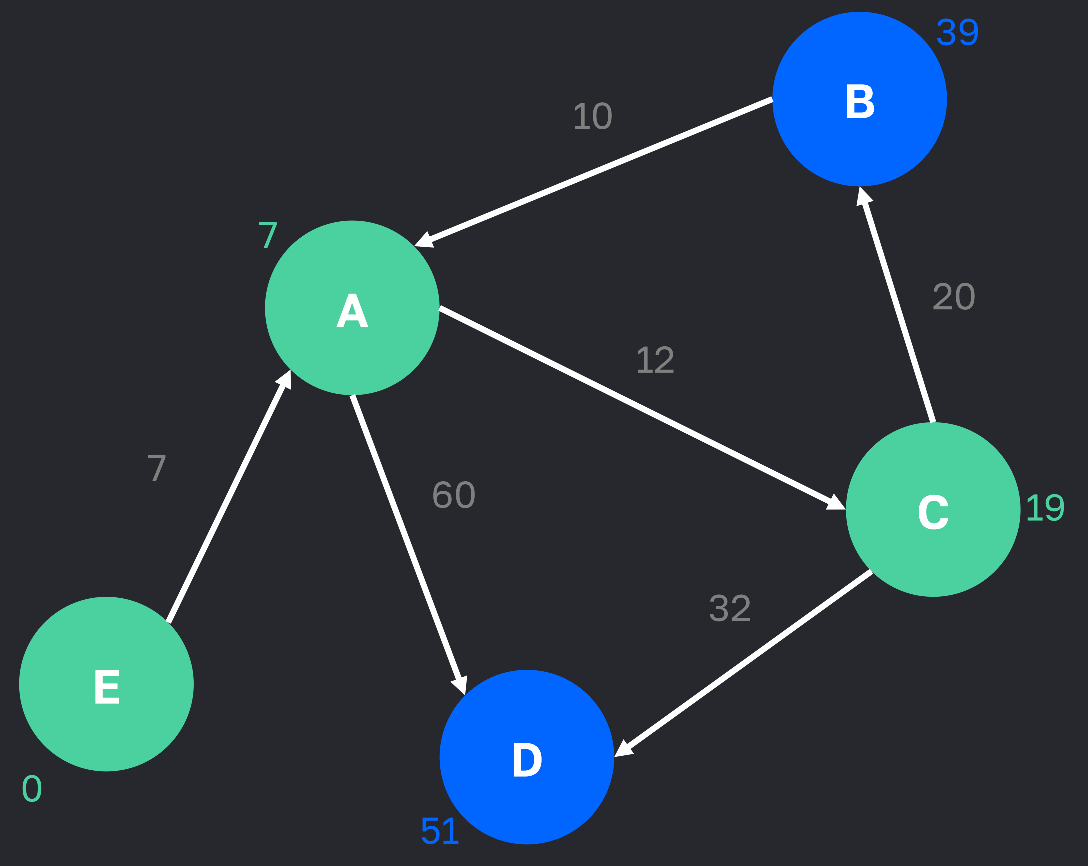


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

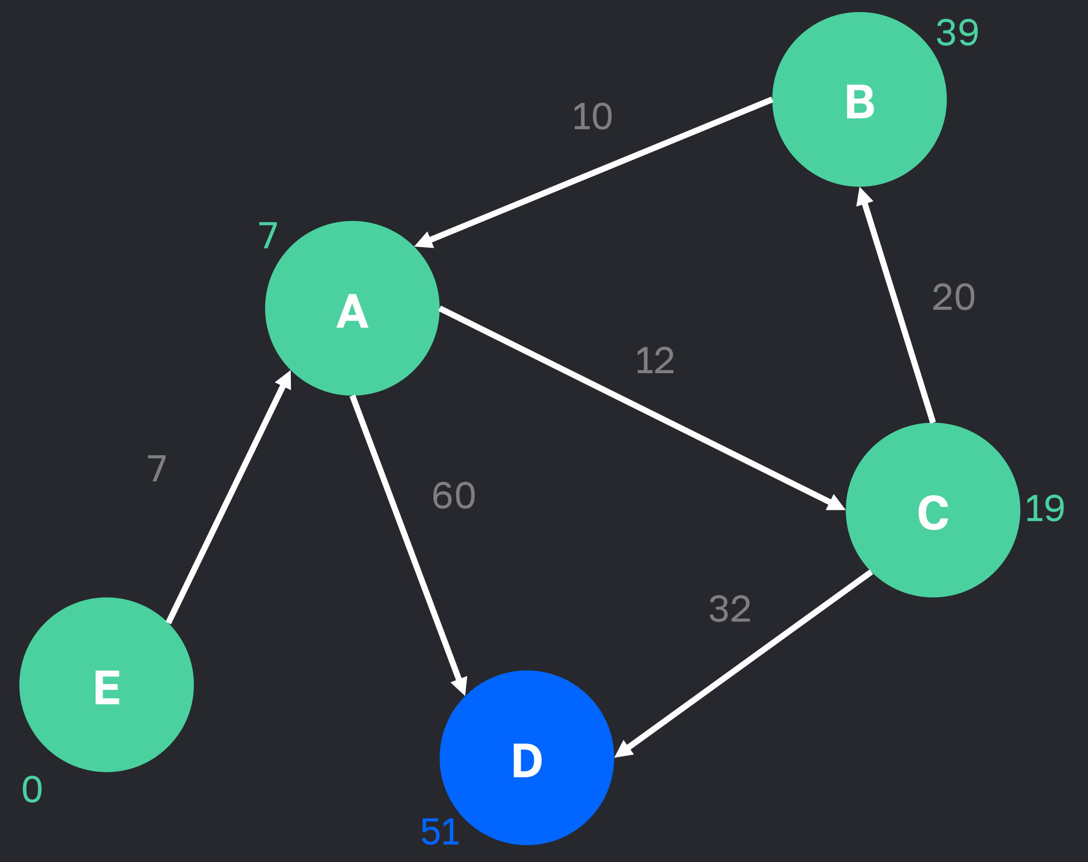


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

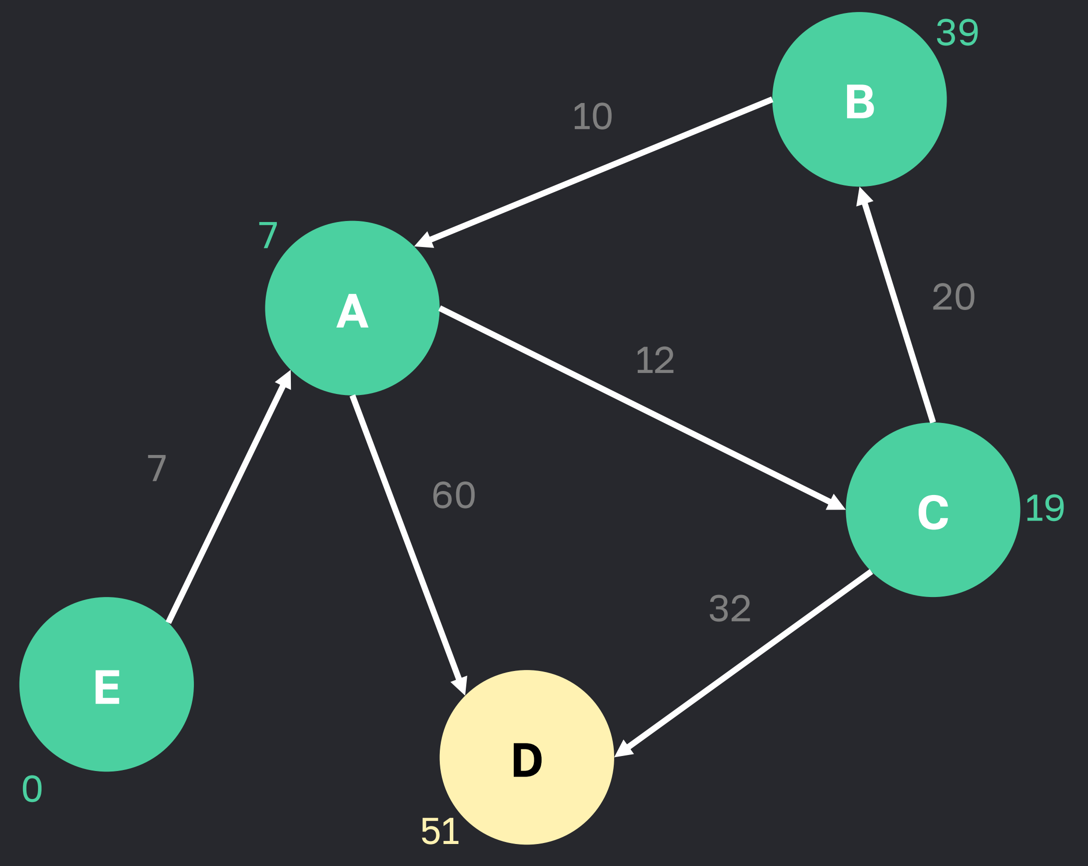


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.

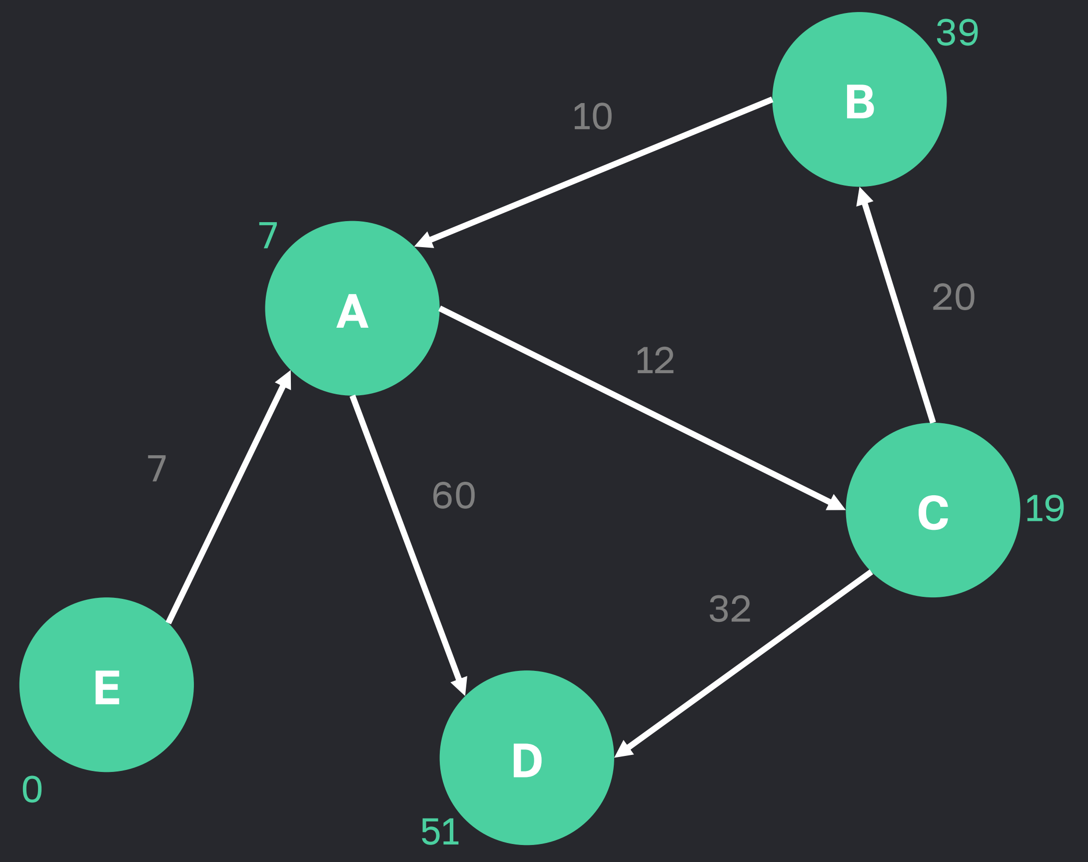


Алгоритм Дейкстры: визуализация

На каждом шаге мы будем доставать из пирамиды минимум. Алгоритм устроен так, что ключ у этого минимума и будет минимальным путём до неё (позже проиллюстрируем идею, а сейчас просто поверьте).

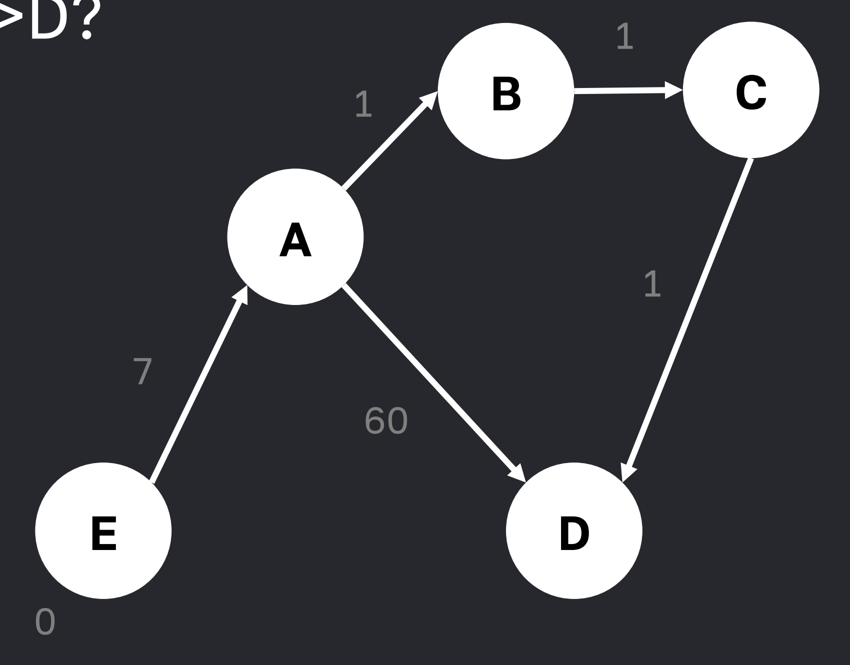
Затем просто переберём все дуги из этой вершины и обновим наши знания о соседних вершинах в пирамиде, если путь через нас до них будет короче.

После чего добавляем эту извлечённую вершину в помеченные.



Алгоритм Дейкстры: визуализация

Почему для извлекаемого на каждом шаге минимума пирамиды мы можем быть уверены, что это минимальный путь до него? Почему у нас никогда не получится извлечь, например, D с прикидкой в 60 раньше, чем мы найдём путь E->A->B->C->D?



Примерная идея объяснения такая.

Пусть на всех шагах правило «извлекаемый из пирамиды элемент имеет минимальный путь» работало, почему сработает и дальше?

Если и есть более короткий путь E->A->B->C->D, то по определению его вес меньше 67, как и, очевидно, любой кусок этого пути: E->A->B и т. п. Мы точно обработали как минимум одну вершину этого пути (например, начальную). Первая необработанная вершина этого пути (B) извлечётся раньше, чем D, т. к её значение в пирамиде точно не больше, чем путь до предыдущей обработанной (A) + вес дуги между ними, что по определнию меньше, чем ключ в пирамиде у D. Поэтому B и C извлекутся раньше D и алгоритм сработает.

Алгоритм Дейкстры

Псевдокод:

```
dijkstra(graph, s):
  d = [s = 0, остальные +inf]
  heap = пирамида(s = 0, остальные +inf)
  marked = пустой ас. массив
  repeat V раз
    cur = извлекаем минимум из heap
    for v из смежные(cur)
      if d[cur] + вес(cur->v) < d[v]
        d[v] = d[cur] + вес(cur->v)
    marked[cur] = d[cur]
```

Комментарий

Выбираем среди ещё нерассмотренных вершину с минимальным значением в «d» и рассматриваем её. При рассмотрении пробегаемся по всем дугам из этой вершины и проверяем, не короче ли путь до смежной вершины через рассматриваемую

Алгоритм Дейкстры

Асимптотика. При использовании пирамид асимптотика $O(E + V \log_2 V)$.

```
dijkstra(graph, s):
  d = [V раз бесконечность]
  d[s] = 0
  marked = пустое множество
  parents = [V раз пусто]
  repeat V раз
    cur = вершина не в marked с мин. d
    for v из смежные(cur)
      if d[cur] + вес(cur->v) < d[v]
        d[v] = d[cur] + вес(cur->v)
        parents[v] = cur
    marked[cur] = да
```

Алгоритм работает, только если веса дуг неотрицательны

Другие алгоритмы для кратчайших путей

Алгоритм	Что ищет
Дейкстры	Из одной вершины до всех остальных с неотрицательными дугами
Форда — Беллмана	Из одной вершины до всех остальных с любыми дугами
Флойда — Уоршелла	Нахождение кратчайших путей между всеми парами вершин графа