Алгоритмы и структуры данных

Disjoint Set Union

CS Center, Новосибирск

DSU|система непересекающихся множеств

$E = \{1, 2, \ldots n\}$

DSU|система непересекающихся множеств

E = {1, 2, ... n}

«~» - отношение эквивалентности на E

DSU|система непересекающихся множеств

E = {1, 2, ... n}

«~» - отношение эквивалентности на E

S - разбение E на классы эквивалентности

DSU|система непересекающихся множеств

E = {1, 2, ... n}

«~» - отношение эквивалентности на E

S - разбение E на классы эквивалентности

Изначально ~ = {(1, 1), (2, 2), ... (n, n)}, S = { {1}, {2}, ... {n} }

DSU|система непересекающихся множеств

Изначально ~ = {(1, 1), (2, 2), ... (n, n)}, S = { {1}, {2}, ... {n} }

DSU|система непересекающихся множеств

Изначально ~ = {(1, 1), (2, 2), ... (n, n)}, S = { {1}, {2}, ... {n} }

Пусть x, y из E

DSU|система непересекающихся множеств

Изначально ~ = {(1, 1), (2, 2), ... (n, n)}, S = { {1}, {2}, ... {n} }

Пусть x, y из E

Обозначим C(x) – класс, которому принадлежит элемент x

DSU|система непересекающихся множеств

Изначально ~ = {(1, 1), (2, 2), ... (n, n)}, S = { {1}, {2}, ... {n} }

Пусть x, y из E

Обозначим C(x) – класс, которому принадлежит элемент x

Методы DSU:

- unite(x, y)
- bool check(x, y) const

DSU|система непересекающихся множеств

Изначально ~ = {(1, 1), (2, 2), ... (n, n)}, S = { {1}, {2}, ... {n} }

Пусть x, y из E

Обозначим C(x) – класс, которому принадлежит элемент x

Методы DSU:

- unite(x, y) : S := S \ { C(x), C(y) } ∪ { C(x) ∪ C(y) }
- bool check(x, y) const

DSU|система непересекающихся множеств

Изначально ~ = {(1, 1), (2, 2), ... (n, n)}, S = { {1}, {2}, ... {n} }

Пусть x, y из E

Обозначим C(x) – класс, которому принадлежит элемент x

Методы DSU:

- unite(x, y) : S := S \ { C(x), C(y) } ∪ { C(x) ∪ C(y) }
- bool check(x, y) const : x ~ y

DSU|система непересекающихся множеств

Методы DSU:

- unite(x, y) : $S := S \setminus \{ C(x), C(y) \} \cup \{ C(x) \cup C(y) \}$
- bool check(x, y) const : $x \sim y$

DSU|система непересекающихся множеств

Методы DSU:

- unite(x, y) : $S := S \setminus \{ C(x), C(y) \} \cup \{ C(x) \cup C(y) \}$
- bool check(x, y) const : $x \sim y$

Пример:

$E = \{1, 2, 3, 4, 5\}$, $S = \{ \{1\}, \{2\}, \{3\}, \{4\}, \{5\} \}$

DSU|система непересекающихся множеств

Методы DSU:

- unite(x, y) : S := S \ { C(x), C(y) } ∪ { C(x) ∪ C(y) }
- bool check(x, y) const : x ~ y

Пример:

E = {1, 2, 3, 4, 5}, S = { {1}, {2}, {3}, {4}, {5} }
unite(1, 2): S = { {1, 2}, {3}, {4}, {5} }

DSU|система непересекающихся множеств

Методы DSU:

- unite(x, y) : S := S \ { C(x), C(y) } ∪ { C(x) ∪ C(y) }
- bool check(x, y) const : x ~ y

Пример:

E = {1, 2, 3, 4, 5}, S = { {1}, {2}, {3}, {4}, {5} }
unite(1, 2): S = { {1, 2}, {3}, {4}, {5} }
unite(3, 5): S = { {1, 2}, {3, 5}, {4} }

DSU|система непересекающихся множеств

Методы DSU:

- unite(x, y) : S := S \ { C(x), C(y) } ∪ { C(x) ∪ C(y) }
- bool check(x, y) const : x ~ y

Пример:

E = {1, 2, 3, 4, 5}, S = { {1}, {2}, {3}, {4}, {5} }
unite(1, 2): S = { {1, 2}, {3}, {4}, {5} }
unite(3, 5): S = { {1, 2}, {3, 5}, {4} }
unite(2, 3): S = { {1, 2, 3, 5}, {4} }

DSU|система непересекающихся множеств

Методы DSU:

- unite(x, y) : S := S \ { C(x), C(y) } ∪ { C(x) ∪ C(y) }
- bool check(x, y) const : x ~ y

Пример:

E = {1, 2, 3, 4, 5}, S = { {1}, {2}, {3}, {4}, {5} }
unite(1, 2): S = { {1, 2}, {3}, {4}, {5} }
unite(3, 5): S = { {1, 2}, {3, 5}, {4} }
unite(2, 3): S = { {1, 2, 3, 5}, {4} }
unite(1, 5): S = { {1, 2, 3, 5}, {4} }

DSU|система непересекающихся множеств

Методы DSU:

- unite(x, y) : S := S \ { C(x), C(y) } ∪ { C(x) ∪ C(y) }
- bool check(x, y) const : x ~ y

Пример:

E = {1, 2, 3, 4, 5}, S = { {1}, {2}, {3}, {4}, {5} }
unite(1, 2): S = { {1, 2}, {3}, {4}, {5} }
unite(3, 5): S = { {1, 2}, {3, 5}, {4} }
unite(2, 3): S = { {1, 2, 3, 5}, {4} }
unite(1, 5): S = { {1, 2, 3, 5}, {4} }
unite(1, 4): S = { {1, 2, 3, 5, 4} }

Применение DSU

Задача incremental dynamic connectivity

- Есть неориентированный граф

Применение DSU

Задача incremental dynamic connectivity

- Есть неориентированный граф
- Обрабатываем два вида запросов:
  - Добавить ребро

Применение DSU

Задача incremental dynamic connectivity

- Есть неориентированный граф
- Обрабатываем два вида запросов:
  - Добавить ребро
  - Проверить в одной ли компоненте связности две вершины

Применение DSU

Задача incremental dynamic connectivity

- Есть неориентированный граф
- Обрабатываем два вида запросов:
  - Добавить ребро
  - Проверить в одной ли компоненте связности две вершины

- Задача сводится к DSU

Применение DSU

Задача decremental dynamic connectivity

- Есть неориентированный граф

Применение DSU

Задача decremental dynamic connectivity

- Есть неориентированный граф
- Обрабатываем два вида запросов:
  - Удалить ребро

Применение DSU

Задача decremental dynamic connectivity

- Есть неориентированный граф
- Обрабатываем два вида запросов:
  - Удалить ребро
  - Проверить в одной ли компоненте связности две вершины

Применение DSU

Задача decremental dynamic connectivity

- Есть неориентированный граф
- Обрабатываем два вида запросов:
  - Удалить ребро
  - Проверить в одной ли компоненте связности две вершины

- Offline decremental сводится к incremental

Реализация DSU

Наивная реализация

Подход	unite	check
Добавление ребра	O(1)	O(n)
Перекраска	O(n)	O(1)

Реализация DSU на основе леса

- Каждому элементу E соответсвует вершина леса

Реализация DSU на основе леса

- Каждому элементу E соответсвует вершина леса
- Каждой дерево леса представляет один класс эквивалентности

Реализация DSU на основе леса

- Каждому элементу E соответсвует вершина леса
- Каждой дерево леса представляет один класс эквивалентности

- В каждой вершине храним ссылку на родителя

Реализация DSU на основе леса

- Каждому элементу E соответсвует вершина леса
- Каждой дерево леса представляет один класс эквивалентности

- В каждой вершине храним ссылку на родителя
- Используем функцию getRoot – для указанной вершины получает корень её дерева

Реализация DSU на основе леса

- Каждому элементу E соответсвует вершина леса
- Каждой дерево леса представляет один класс эквивалентности

- В каждой вершине храним ссылку на родителя
- Используем функцию getRoot – для указанной вершины получает корень её дерева

- check(x, y) – проверяет, что x и y в одном дереве

Реализация DSU на основе леса

- Каждому элементу E соответсвует вершина леса
- Каждой дерево леса представляет один класс эквивалентности

- В каждой вершине храним ссылку на родителя
- Используем функцию getRoot – для указанной вершины получает корень её дерева

- check(x, y) – проверяет, что x и y в одном дереве
  - getRoot(x) == getRoot(y)

Реализация DSU на основе леса

- Каждому элементу E соответсвует вершина леса
- Каждой дерево леса представляет один класс эквивалентности

- В каждой вершине храним ссылку на родителя
- Используем функцию getRoot – для указанной вершины получает корень её дерева

- check(x, y) – проверяет, что x и y в одном дереве
  - getRoot(x) == getRoot(y)
- unite(x, y) – если x и y в разных деревьях, сделать из этих деревьев одно

Реализация DSU на основе леса

- Каждому элементу E соответсвует вершина леса
- Каждой дерево леса представляет один класс эквивалентности

- В каждой вершине храним ссылку на родителя
- Используем функцию getRoot – для указанной вершины получает корень её дерева

- check(x, y) – проверяет, что x и y в одном дереве
  - getRoot(x) == getRoot(y)
- unite(x, y) – если x и y в разных деревьях, сделать из этих деревьев одно
  - getRoot(x) подвесить к getRoot(y) (или наоборот)

Реализация DSU на основе леса

- check(x, y) – проверяет, что x и y в одном дереве
  - getRoot(x) == getRoot(y)
- unite(x, y) – если x и y в разных деревьях, сделать из этих деревьев одно
  - getRoot(x) подвесить к getRoot(y) (или наоборот)

- Всё зависит от времени работы getRoot

Реализация DSU на основе леса

- check(x, y) – проверяет, что x и y в одном дереве
  - getRoot(x) == getRoot(y)
- unite(x, y) – если x и y в разных деревьях, сделать из этих деревьев одно
  - getRoot(x) подвесить к getRoot(y) (или наоборот)

- Всё зависит от времени работы getRoot = высота дерева

Реализация DSU на основе леса

- check(x, y) – проверяет, что x и y в одном дереве
  - getRoot(x) == getRoot(y)
- unite(x, y) – если x и y в разных деревьях, сделать из этих деревьев одно
  - getRoot(x) подвесить к getRoot(y) (или наоборот)

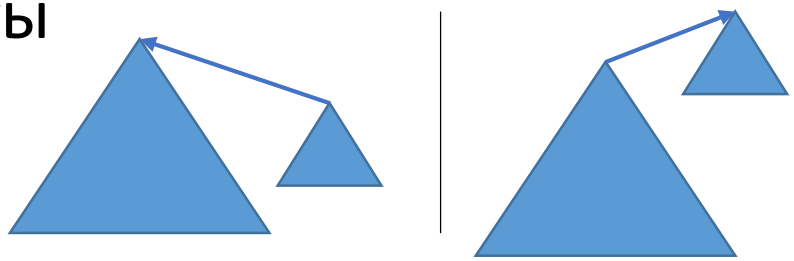
- Всё зависит от времени работы getRoot = высота дерева
  - Зависит от стратегии подвешивания при unite

DSU на лесе: эвристика ранга

- Стремимся получать деревья небольшой высоты

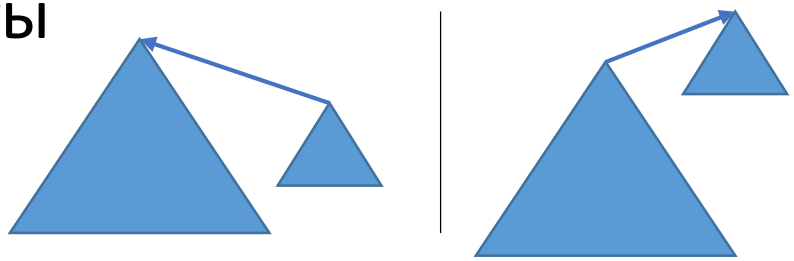
DSU на лесе: эвристика ранга

- Стремимся получать деревья небольшой высоты
- Как определить кого к кому подвесить?



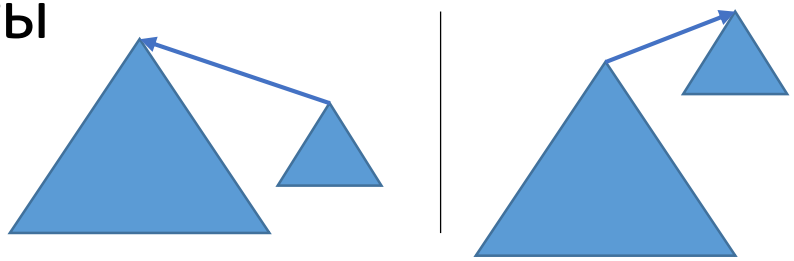
DSU на лесе: эвристика ранга

- Стремимся получать деревья небольшой высоты
- Как определить кого к кому подвесить?
  - Введём ранг вершины r(v):
    - r(v) = 0, у v нет детей



DSU на лесе: эвристика ранга

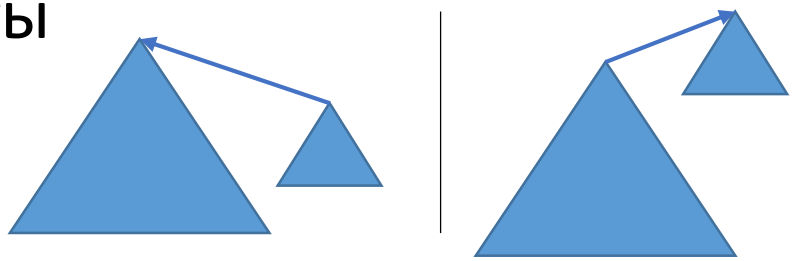
- Стремимся получать деревья небольшой высоты
- Как определить кого к кому подвесить?
  - Введём ранг вершины r(v):
    - r(v) = 0, у v нет детей
    - r(v) = 1 + max { r(u) | p(u) = v }



DSU на лесе: эвристика ранга

- Стремимся получать деревья небольшой высоты
- Как определить кого к кому подвесить?
  - Введём ранг вершины r(v):
    - r(v) = 0, у v нет детей
    - r(v) = 1 + max { r(u) | p(u) = v }

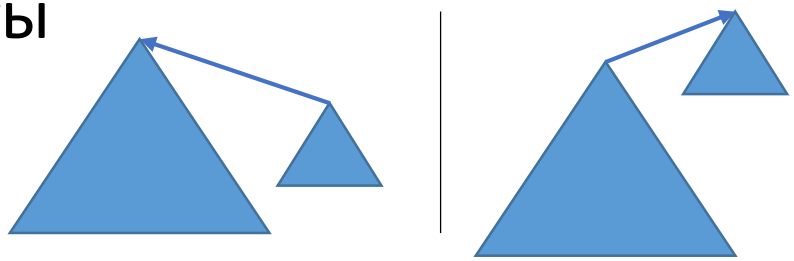
  // Ранг – это высота



DSU на лесе: эвристика ранга

- Стремимся получать деревья небольшой высоты
- Как определить кого к кому подвесить?
  - Введём ранг вершины r(v):
    - r(v) = 0, у v нет детей
    - r(v) = 1 + max { r(u) | p(u) = v }

  // Ранг – это высота
- Подвешиваем дерево с меньшим рангом к дереву с большим рангом

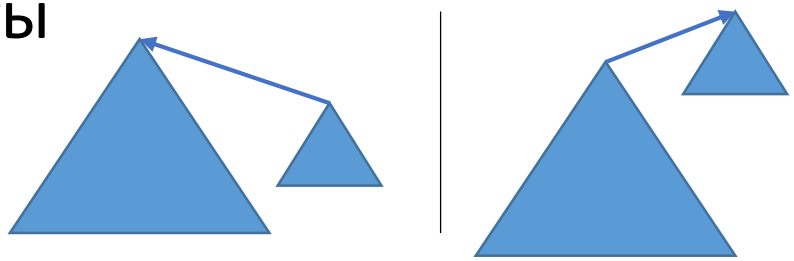


DSU на лесе: эвристика ранга

- Стремимся получать деревья небольшой высоты
- Как определить кого к кому подвесить?
  - Введём ранг вершины r(v):
    - r(v) = 0, у v нет детей
    - r(v) = 1 + max { r(u) | p(u) = v }

  // Ранг – это высота
- Подвешиваем дерево с меньшим рангом к дереву с большим рангом

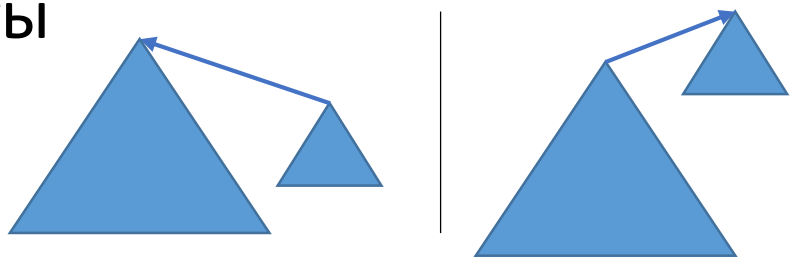
  Когда растёт ранг?



DSU на лесе: эвристика ранга

- Стремимся получать деревья небольшой высоты
- Как определить кого к кому подвесить?
  - Введём ранг вершины r(v):
    - r(v) = 0, у v нет детей
    - r(v) = 1 + max { r(u) | p(u) = v }

  // Ранг – это высота



- Подвешиваем дерево с меньшим рангом к дереву с большим рангом

  Когда растёт ранг?
    - Только при совпадении рангов объединяемых деревьев

DSU на лесе: эвристика ранга

Оценка ранга

- $N \geq 2^{r(v)}$

DSU на лесе: эвристика ранга

Оценка ранга

- $N \geq 2^{r(v)}$
  - По индукции

DSU на лесе: эвристика ранга

Оценка ранга

- $N \geq 2^{r(v)}$
  - По индукции:
    - База: $r(v) = 0$ => в поддереве v одна вершина

DSU на лесе: эвристика ранга

Оценка ранга

- $N \geq 2^{r(v)}$
  - По индукции:
    - База: $r(v) = 0$ => в поддереве v одна вершина
    - Инд.переход: если ранг увеличился «r -> r + 1», то число вершин стало $2^{(r+1)}$

DSU на лесе: эвристика ранга

Оценка ранга

- $N \geq 2^{r(v)}$
  - По индукции:
    - База: r(v) = 0 => в поддереве v одна вершина
    - Инд.переход: если ранг увеличился «r -> r + 1», то число вершин стало $2^{(r+1)}$

- $r(v) \leq \log N$

DSU на лесе: эвристика ранга

Оценка ранга

- $N \geq 2^{r(v)}$
  - По индукции:
    - База: r(v) = 0 => в поддереве v одна вершина
    - Инд.переход: если ранг увеличился «r -> r + 1», то число вершин стало $2^{(r+1)}$

- $r(v) \leq \log N$
- $T_{getRoot} = O(\log N)$

DSU на лесе: эвристика ранга

Оценка ранга

- $N \geq 2^{r(v)}$
  - По индукции:
    - База: r(v) = 0 => в поддереве v одна вершина
    - Инд.переход: если ранг увеличился «r -> r + 1», то число вершин стало 2^(r+1)

- $r(v) \leq \log N$
- $T_{getRoot} = O(\log N)$
  - $T_{unite} = O(\log N)$
  - $T_{check} = O(\log N)$

DSU на лесе: сжатие путей

Когда повторно вызвали getRoot(v),

можно ли переиспользовать результат с прошлого раза?

DSU на лесе: сжатие путей

Когда повторно вызвали getRoot(v),

можно ли переиспользовать результат с прошлого раза?

- getRoot переподвешивает к корню все пройденные вершины

DSU на лесе: сжатие путей

Когда повторно вызвали getRoot(v),

можно ли переиспользовать результат с прошлого раза?

- getRoot переподвешивает к корню все пройденные вершины
  - не будем ходить по одному пути много раз

DSU на лесе: сжатие путей

Когда повторно вызвали getRoot(v),

можно ли переиспользовать результат с прошлого раза?

- getRoot переподвешивает к корню все пройденные вершины
  - не будем ходить по одному пути много раз
- Ранги + сжатие путей => учётное время растёт медленнее итерированного логарифма

DSU на лесе: сжатие путей

Когда повторно вызвали getRoot(v),

можно ли переиспользовать результат с прошлого раза?

- getRoot переподвешивает к корню все пройденные вершины
  - не будем ходить по одному пути много раз
- Ранги + сжатие путей => учётное время растёт медленнее итерированного логарифма

N	2	4	16	65536	2^65536
(log*)N	1	2	3	4	5