

АДРИАНОВ Н.М.
ИВАНОВ А.Б.

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

МИНИМАЛЬНЫЕ ПОКРЫВАЮЩИЕ ДЕРЕВЬЯ.
АЛГОРИТМ ПРИМА. АЛГОРИТМ КРУСКАЛА.
СИСТЕМА НЕПЕРЕСЕКАЮЩИХСЯ МНОЖЕСТВ

ПОКРЫВАЮЩИЕ ДЕРЕВЬЯ

(V, E) – связный неориентированный граф

$l : E \rightarrow \mathbb{R}$ – длины ребер

Покрывающее (остовное) дерево T :

вершины $V(T) = V$

ребра $E(T) \subset E$

Минимальное покрывающее дерево: $\sum_{e \in E(T)} l(e) \rightarrow \min$

MST = Minimal Spanning Tree

АЛГОРИТМ ПРИМА

```
procedure Prim( G ):

    visited = array of size G.V
    T = graph( G.V )
    Q = priority queue

    x = any vertex from V
    for z in G.neighbours(x):
        Q.insert((x,z), l(x,z))
    end

    while not Q.isEmpty:
        (x,y) = Q.deleteMin()
        if not visited[y]:
            visited[y] = true
            T.addEdge(x,y)
            for z in G.neighbours(y):
                Q.insert((y,z), l(y,z))
            end
        end
    end
end
end
```

СВОЙСТВО РАЗРЕЗА

Разрез: вершины разбиты на 2 части: S и $V \setminus S$.

- ребра X входят в минимальное покрывающее дерево T
- ни одно ребро из X не пересекает разрез
- e – ребро минимальной длины, пересекающее разрез

\Rightarrow Ребра $X \cup \{e\}$ входят в минимальное покрывающее дерево T' .

АЛГОРИТМ КРУСКАЛА

```
procedure Kruskal( G ):
    E' = E отсортированный по
           возрастанию длины
    X =  $\emptyset$ 
    for e in E':
        if X  $\cup$  e не содержит циклов:
            X = X  $\cup$  e
        end
    end
end
```

НЕПЕРЕСЕКАЮЩИЕСЯ МНОЖЕСТВА (UNION-FIND)

```
class UnionFind {
    UnionFind(int N)

    void union(int x, int y)
    int find(x)

    boolean
        connected(int x, int y)
    int count()
}
```

```
connected(x,y):
    find(x) == find(y)
```

```
procedure Kruskal( G ):
    E' = E отсортированный по
        возрастанию длины
    T = graph( G.V )
    UF = UnionFind( G.V )
    for (u,v) in E':
        if !UF.connected(u,v):
            T.addEdge(u,v)
            UF.union(u,v)
        end
    end
end
```

UNION-FIND: РЕАЛИЗАЦИЯ

- Храним номер компоненты для каждой точки.
union точек из разных компонент меняет номер в одной из компонент.
find: $O(1)$
union: $O(n)$
- Храним номер родителя для каждой точки.
union точек из разных компонент прописывает одну точку как родителя другой.
find поднимается до корневой вершины, номер корневой вершины и будет номером компоненты.
find: $O(n)$
union: $O(1)$

UNION-FIND: РЕАЛИЗАЦИЯ

```
public class UnionFind {
    private int[] parent;

    public UnionFind(int N) {
        parent = new int[N];
        for (int i = 0; i < N; i++)
            parent[i] = i;
    }

    public int find(int x) {
        while (x != parent[x])
            x = parent[x];
        return x;
    }

    public void union(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) return;
        parent[x] = y;
    }
}
```

UNION-FIND: РАНГИ

`find` имеет сложность $O(n)$, так как могут получаться “высокие” деревья.

Идея: будем хранить высоту дерева (ранг). Будем прицеплять меньшее дерево к большему.

- если $\pi(x) \neq x$, то $rank(x) < rank(\pi(x))$
- если $rank(x) = k$, то в дереве под x – не менее 2^k вершин
- максимально возможный ранг равен $\lceil \log n \rceil$

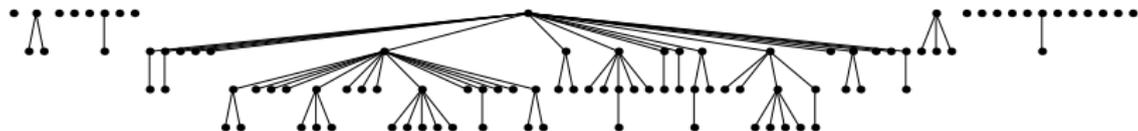
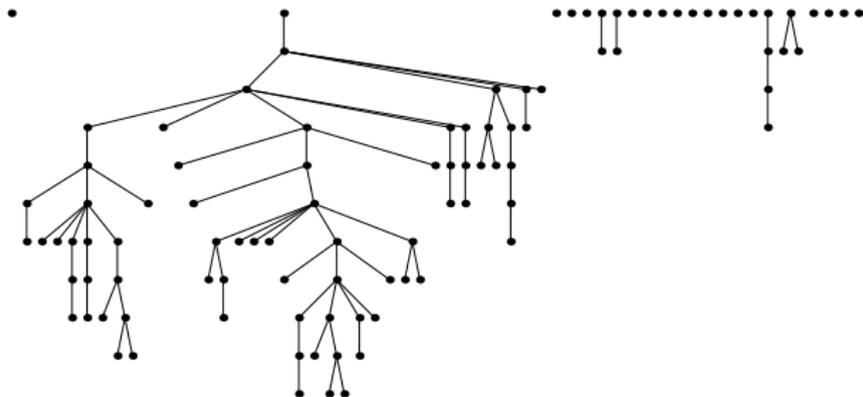
UNION-FIND: РАБГИ

```
public class UnionFind {
    private int[] rank;

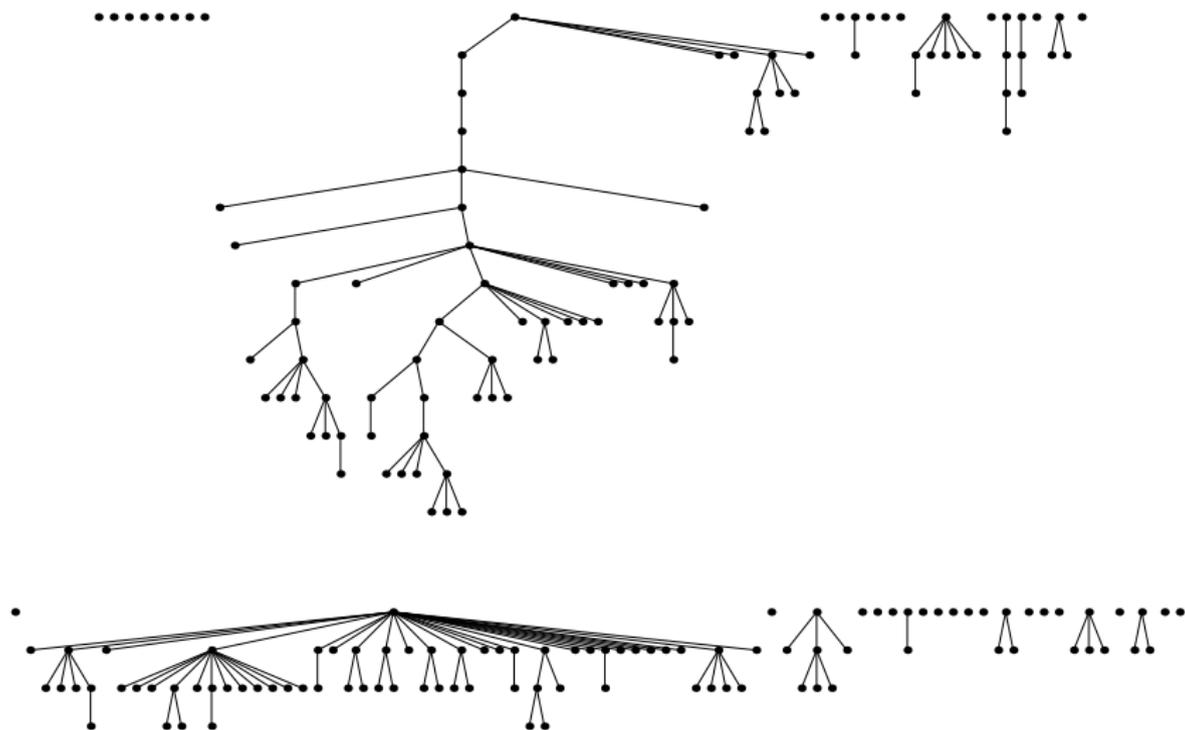
    public UnionFind(int N) {
        ...
        rank = new int[N];
        for (int i = 0; i < N; i++)
            rank[i] = 0;
    }

    public void union(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) return;
        if (rank[x] < rank[y])
            parent[x] = y;
        else {
            parent[y] = x;
            if (rank[x] == rank[y])
                rank[x]++;
        }
    }
}
```

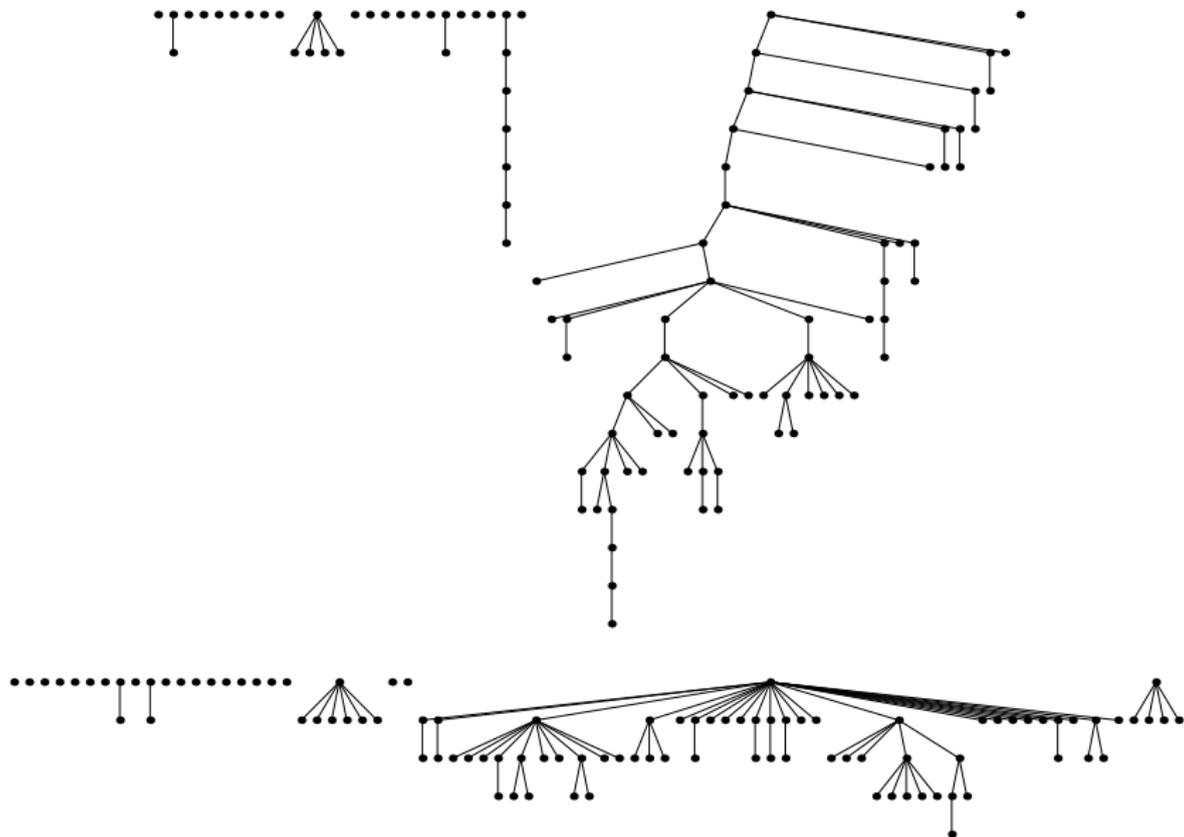
UNION-FIND: РАНГИ (СИМУЛЯЦИЯ 1)



UNION-FIND: РАНГИ (СИМУЛЯЦИЯ 2)



UNION-FIND: РАНГИ (СИМУЛЯЦИЯ 3)



UNION-FIND: СЖАТИЕ ПУТЕЙ

Идея: При вызове `find(x)` мы проходим все вершины до корня. Можем все эти вершины перевесить к корню.

```
public class UnionFind {  
  
    ...  
  
    public void find(int x) {  
        if (parent[x] == x) return x;  
        return parent[x] = find(x);  
    }  
  
    ...  
}
```

UNION-FIND: СЖАТИЕ ПУТЕЙ – СЛОЖНОСТЬ

$$\text{TOWER}(k) = 2^{2^{\dots^2}}$$

$\log^* n =$ минимальное k такое, что $\underbrace{\log \log \dots \log n}_{k \text{ раз}} \leq 1$

$$\text{TOWER}(1) = 2$$

$$\text{TOWER}(2) = 2^2 = 4$$

$$\text{TOWER}(3) = 2^4 = 16$$

$$\text{TOWER}(4) = 2^{16} = 65536$$

$$\text{TOWER}(5) = 2^{65536}$$

n	$\log^* n$
1	0
2	1
3, 4	2
5, 6, ... 16	3
17, 18, ... 65536	4
65537, ... 2^{65536}	5

UNION-FIND: СЖАТИЕ ПУТЕЙ – СЛОЖНОСТЬ

Когда вершина x перестает быть корнем – ее ранг больше не меняется. Если $rank(x) \in \{k + 1, \dots, 2^k\}$, где $k = \text{TOWER}(i)$, то “выдаем” вершине 2^k рублей.

Если $rank(x) = k$, то в дереве под x не менее 2^k вершин. Следовательно, число вершин ранга k не более $n/2^k$.

Число вершин ранга $> k$

$$\leq \frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \dots = \frac{n}{2^k} \cdot \left(\frac{1}{2} + \frac{1}{4} + \dots \right) = \frac{n}{2^k}$$

Значит, всего “выдали” не более $n \log^* n$ рублей.

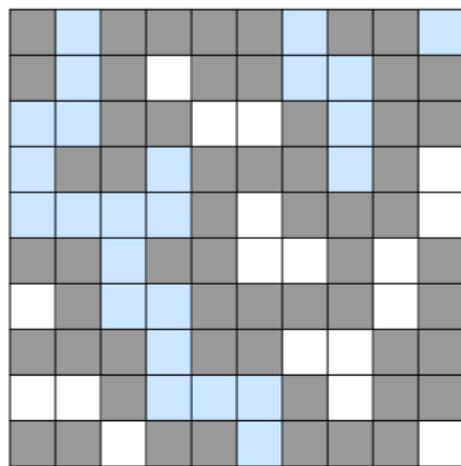
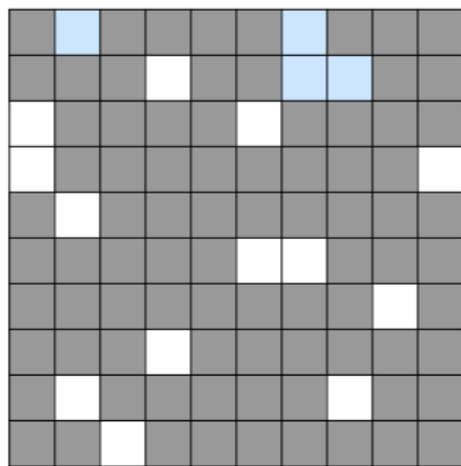
При вызове метода `find` движемся вверх к корню – если $rank(x)$ и $rank(\pi(x))$ лежат в одном интервале, то берем рубль у вершины x . Переходов между интервалами – не более $\log^* n$.

Итого: m вызовов `union` / `find` требуют $O(n \log^* n + m \log^* n)$ операций. Амортизированная стоимость = $O(\log^* n)$.

ПРОСАЧИВАНИЕ (PERCOLATION)

Модель:

- Квадратная решетка $N \times N$.
- “Открываются” случайные клетки.
- Просачивание: сверху вниз через “открытые” клетки.



ПОРОГ ПРОСАЧИВАНИЯ

$$p = (\text{количество открытых клеток})/N^2$$

Какую долю p от всех клеток надо открыть, чтобы возникло просачивание?

$$p^* = 0.592746 \dots$$

ПРОСАЧИВАНИЕ: СИМУЛЯЦИЯ

```
public class Percolation {  
  
    public Percolation(int N)  
    public void open(int x, int y)  
    public boolean isOpen(int x, int y)  
    public boolean percolates()  
}
```

