Топологическая сортировка

- это последовательность вершин t[1] … t[n], такая, что нет ребра из более правой вершины в более левую.

1 вариант.
Добавляем вершины в список в самом конце дфса. Получится перевернутая топологическая сортировка

```
List<Integer> topsort = new ArrayList<>();
void dfs(int x) {
  if (vis[x]) return;
  vis[x] = true;
  for (int y : graph[x]) {
    dfs(y);
  }
  topsort.add(x);
}
Collections.reverse(topsort);
```

2 вариант.
Храним в очереди все вершины, в которых не идет ни одно ребро. На каждом шагу достаем из очереди вершину, добавляем ее в топологическую сортировку и “удаляем” все ребра, ведущие из нее (уменьшаем ее степень на 1).

```
List<Integer> topsort = new ArrayList<>();
Queue<Integer> q = new ArrayDeque<>();
int[] in = new int[n]; // in[i] - сколько ребер входит в i
// заполнить in, добавить в q
while (!q.isEmpty()) {
      int x = q.poll();
      topsort.add(x);
      for (int y : graph[x]) {
            in[y]--;
            if (in[y] == 0) q.add(y);
      }
}
```

Задача: построить лексикографически минимальную топологическую сортировку.
(массив a лексикографически меньше массива b, если до какого-то индекса k их элементы были равны: a[0]=b[0], …, a[k-1]=b[k-1], но a[k]<b[k])
Решение: Храним в *очереди с приоритетами* все вершины, в которых не идет ни одно ребро. На каждом шагу достаем из *очереди с приоритетами* вершину, добавляем ее в топологическую сортировку и “удаляем” все ребра, ведущие из нее (уменьшаем ее степень на 1).

Поиск в ширину на графе с ребрами веса 0 и 1

```
int[] dist = new int[n];
Arrays.fill(dist, INF);
dist[start] = 0;
Deque<Integer> q = new ArrayDeque<>();
q.addLast(start);
while (!q.isEmpty()) {
      int x = q.poll();
      for (Edge e : graph[x]) {
            if (dist[e.to] > dist[x] + e.w) {
                  dist[e.to] = dist[x] + e.w;
                  if (e.w == 0) {
                        q.addFirst(e.to);
                  } else {
                        q.addLast(e.to);
                  }
            }
      }
}
```

если вес ребра равен 0, то можно добавить новую вершину в начало очереди
Свойство “В любой момент времени, если A - первая вершина в очереди, а B - последняя, то либо d[A] +1 == d[B], либо d[A] == d[B].” не нарушается при этом

Диаметр дерева
(это самый длинный путь в дереве)

Алгоритм поиска:

1. возьмем любую вершину x, найдем самую удаленную от нее, пусть она y (если несколько можно взять любую).
2. возьмем вершину y (которую только что нашли), найдем самую удаленную от нее, пусть она z (если несколько можно взять любую).
3. путь y-z - диаметр дерева

Работает только в деревьях. Если в графе есть циклы, этот алгоритм неверен.

Нахождение разных величин в дереве дфсом

Пусть вершина r выделена и является корнем. (дерево подвешено за вершину r)
найти

- расстояние от r до всех остальных
- предка каждой вершины на пути от корня
- размер поддерева (количество вершин в поддереве)
- порядок обхода вершин (order[i] - в какую вершину dfs зашел на i-ом запуске, revOrder[i] - на каком запуске dfs зашел в i-ую вершину)

```
int[] height = new int[n];
int[] parent = new int[n];
int[] size = new int[n];
List<Integer> order = new ArrayList<>();
int[] revOrder = new int[n];
dfs(r, -1, 0);
void dfs(int x, int p, int h) {
  parent[x] = p;
  height[x] = h;
  order.add(x);
  size[x] = 1;
  for (int y : graph[x]) {
    if (y != p) {
      dfs(y);
      size[x] += size[y];
    }
  }
}
for (int i = 0; i < n; i++) revOrder[order.get(i)] = i;
```

1-4, 4-2, 4-3
order = {1 4 2 3}
revOrder = {1 3 4 2}

Можно сопоставлять поддереву вершины отрезок в массиве. Для i-ой вершины это отрезок [revOrder[i], revOrder[i] + size[i] - 1]

Здесь revOrder[4] = 2, size[4] = 3. отрезок - [2, 4]

LCA (least common ancestor, наименьший общий предок)
Задача. Дано дерево из 1e5 вершин. Ответить на 1e5 запросов: найти LCA вершин x и y.
LCA(x, y) - это такая вершина v, что v содержится на пути от и корня до x, и от корня до y, и при этом глубина v максимальна)

```
int[] height = new int[n];
int[][] parent = new int[logN][n]; // logN == log_2(n) округленный вверх
dfs(r, -1, 0);
void dfs(int x, int p, int h) {
  parent[0][x] = p;
  height[x] = h;
  for (int y : graph[x]) {
    if (y != p) dfs(y);
  }
}
parent[0][r] = r; // предок корня - сам корень. здесь это удобно
```

Можно построить структуру данных, которая будет отвечать на запрос LCA(x,y) за logN времени, и потреблять N*logN памяти.

Подзадача: построить структуру данных, которая будет находить k-ого предка вершины за logN времени, и потреблять N*logN памяти.

Храним для каждой вершины ее 1, 2, 4, 8, …. предков.
Теперь можно скакать сразу по много шагов по пути к корню.

Заполнение:
p[lvl][x] - (1<<lvl)-ой предок вершины x
p[0][x] посчитано в dfs-е.
Остальные уровни считаем так:

```
for (int lvl = 1; lvl < logN; lvl++) {
      for (int x = 0; x < n; x++) {
            // зная 1, 2, 4, …, (1<<(lvl-1))-го предков
            // надо посчитать (1<<lvl)-го предка
            parent[lvl][x] = parent[lvl-1][parent[lvl-1][x]];
      }
}
```

Чтобы найти k-ого предка вершины x:
посмотрим на единичные биты в числе k. пройдем по парентам этих лвлов.
Например, чтоб найти 11-го предка: 11 = 8+2+1, значит нам надо:
x = parent[3][x]; x = parent[1][x]; x = parent[0][x]

```
for (int i = logN - 1; i >= 0; i--) {
	if ((k & (1 << i)) != 0) x = parent[i][x];
}
```

Как найти LCA(x,y).

1. Вершины x и y могут быть расположены на разной глубине. Надо поднять ту из них, что глубже, на уровень другой.
2. Поднимаем их синхронно на уровни, начиная с наибольшего. Если после подъема они не совпадают, поднимаем, а если совпадают, не поднимаем. Подбираемся к LCA(x,y) снизу, но не превышаем ее.
3. Теперь parent[0][x] == parent[0][y] == LCA(x,y)

```
int LCA(int x, int y) {
	if (height[x] > height[y]) {
		int t = x; x = y; y = t;
	}
	// теперь height[x] <= height[y]

	// 1. поднимаем y на height[y] - height[x]
	int dh = height[y] - height[x];
	for (int i = logN - 1; i >= 0; i--) {
		if ((dh & (1 << i)) != 0) y = parent[i][y];
	}
	// если x - предок y (они на одном пути к корню)
	if (x == y) return x;

	// теперь x и y на одной глубине
	// поднимаем их синхронно
	for (int i = logN - 1; i >= 0; i--) {
		int nx = parent[i][x];
		int ny = parent[i][y];
		if (nx != ny) {
			x = nx;
			y = ny;
		}
	}

	// теперь parent[0][x] == parent[0][y] == LCA(x,y)
	return parent[0][x];
}
```

Задача. Дано дерево из 1e5 вершин. Ответить на 1e5 запросов: найти расстояние между x и y.

Задачи

http://acm.timus.ru/problem.aspx?space=1&num=1930
http://acm.timus.ru/problem.aspx?space=1&num=1025

http://acm.timus.ru/problem.aspx?space=1&num=1056
http://acm.timus.ru/problem.aspx?space=1&num=1329 (хотя ее можно сдать и проще)
http://acm.timus.ru/problem.aspx?space=1&num=1471
http://codeforces.com/problemset/problem/519/E