Университет ИТМО

Факультет программной инженерии и компьютерной техники

# Лабораторная работа №4
по “Алгоритмам и структурам данных”
Доп

*Выполнил*:
Студент группы P3206
Сорокин А.Н.
*Преподаватель:*
Тараканов Д.С.

Санкт-Петербург
2024

**Задача №M “Цивилизация”**
Решение за O(V).

Внезапно, линия.

Алгоритм Дейкстры из-за своей ассимптотики на больших наборах данных может работать медленнее, следовательно, необходимо искать более оптимальное решение.

Рассмотрим **поиск в ширину (BFS)**. Данный алгоритм позволит нам найти кратчайший путь быстрее. Вместо priority_queue, имеющей вставку и удаление элементов за O(log V), будем использовать обычный queue, где вставка и удаление происходит за O(1). BFS не тратит время на приоритизацию вершин, он просто расширяется. Однако поскольку граф взвешенный, следует заметить, что при первом достижении конечной вершины найденный путь не обязательно будет кратчайшим.

Рассчитаем ассимптотику алгоритма:
Мы рассматриваем все вершины V. Для каждой вершины рассматриваются все смежные с ней вершины, однако поскольку максимальная степень вершины равна 4, то у каждой вершины максимум 4 соседа, точнее 4V. Отсюда O(V + 4V) = O(V).

Код решения:
```
#include <fstream>
#include <iostream>
#include <list>
#include <queue>
#include <string>
#include <utility>
#include <vector>

using namespace std;
const long INF = 1e7;

struct Node {
```

```
  Node() : d(-1), predecessor(-1), visited(false) {}
  int d;
  int predecessor;
  bool visited;
};

int kek(vector<string> &grid, int v, int m) {
  char a = grid[v / m][v % m];
  if (a == '.') {
    return 1;
  } else if (a == 'W') {
    return 2;
  }
  return INF;
}

int main() {
  ifstream infile("input.txt");

  int n, m, y_start, x_start, y_end, x_end;
  infile >> n >> m >> y_start >> x_start >> y_end >> x_end;

  vector<string> grid(n);
  for (int i = 0; i < n; ++i) {
    infile >> grid[i];
  }

  vector<Node> visited(n * m, Node());

  int s = (x_start - 1) + (y_start - 1) * m;
  int e = (x_end - 1) + (y_end - 1) * m;
  visited[s].d = 0;
  visited[s].predecessor = s;
  visited[s].visited = true;

  vector<pair<int, int>> directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
```

```
queue<int> q;
q.push(s);

while (!q.empty()) {
  int u = q.front();
  q.pop();

  if (kek(grid, u, m) == INF)
    continue;

  for (auto dir : directions) {
    int next_x = u % m + dir.first;
    int next_y = u / m + dir.second;

    if (next_x < 0 || next_x >= m || next_y < 0 || next_y >= n)
      continue;
    int v = u + dir.first + dir.second * m;

    int new_d = visited[u].d + kek(grid, v, m);

    if (!visited[v].visited || visited[v].d > new_d) {
      visited[v].d = (new_d > INF) ? INF : new_d;
      visited[v].predecessor = u;
      visited[v].visited = true;
      q.push(v);
    }
  }
}

if (!visited[e].visited || visited[e].d >= INF) {
  cout << -1;
  return 0;
}
cout << visited[e].d << "\n";
list<char> res;
```

```
  int curr = e;
  while (curr != s) {
    int dif = curr - visited[curr].predecessor;
    if (dif == 1) {
      res.push_front('E');
    } else if (dif == -1) {
      res.push_front('W');
    } else if (dif == m) {
      res.push_front('S');
    } else {
      res.push_front('N');
    }
    curr = visited[curr].predecessor;
  }

  for (char i : res) {
    cout << i;
  }
  return 0;
}
```

## Алгоритм Прима

## Содержание



## Зачем нужен

Алгоритм Прима – алгоритм поиска минимального остовного дерева (MST) во взвешенном неориентированном графе. Минимальное остовное дерево графа G(V, E) – связный ациклический подграф исходного графа G, состоящий из V вершин исходного графа и V – 1 ребер, имеющих наименьший вес.

То есть алгоритм Прима последовательно строит из исходного графа минимальное остовное дерево.

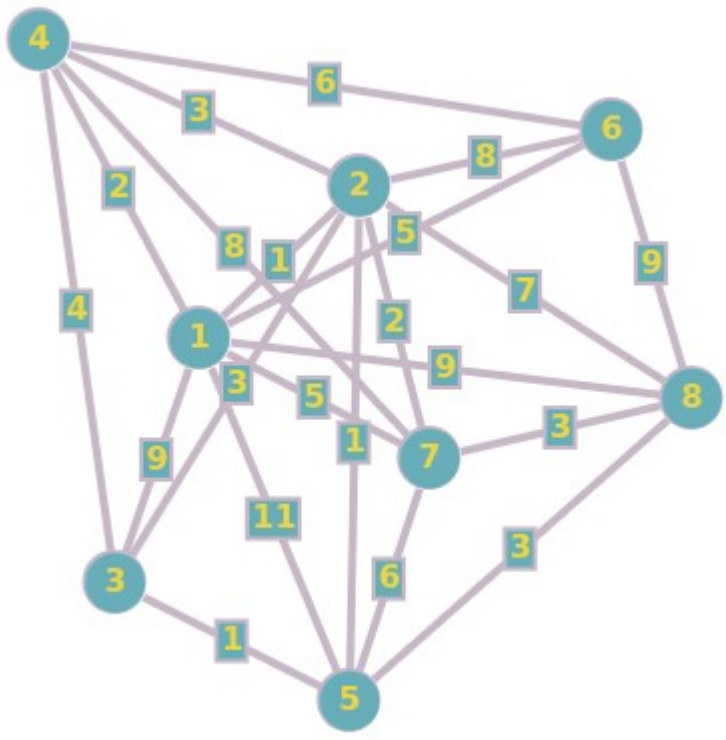
## Как работает

Изначально помечаем произвольную вершину.

1. Проверяем, все ли вершины помечены. Если да, переход к шагу 6.
2. Проводим релаксацию ребер, инцидентных вершинам, смежным с только что добавленной вершиной.
3. Выбираем среди непомеченных вершин вершину с ребром минимального веса, инцидентного ей и помеченной вершине.
4. Помечаем вершину.
5. Переходим к шагу 1.
6. Конец.

## Пример работы

Имеем граф G:




Найдем минимальное остовное дерево данного графа.

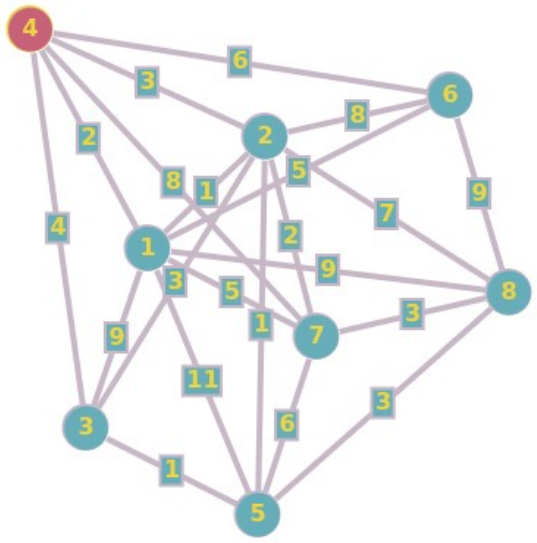
Дана матрица смежности графа G

| V | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | X | 1 | 9 | 2 | 11 | 5 | 5 | 9 |
| 2 | 1 | X | 3 | 3 | 1 | 8 | 2 | 7 |
| 3 | 9 | 3 | X | 4 | 1 | | | |
| 4 | 2 | 3 | 4 | X | | 6 | 8 | |
| 5 | 11 | 1 | 1 | | X | | 6 | 3 |
| 6 | 5 | 8 | | 6 | | X | | 9 |
| 7 | 5 | 2 | | 8 | 6 | | X | 3 |
| 8 | 9 | 7 | | | 3 | 9 | 3 | X |

Начнем обход с вершины v4.

Изначально скажем, что:

| | weight | predecessor | visited |
|---|---|---|---|
| v1 | INF | None | False |
| v2 | INF | None | False |
| v3 | INF | None | False |
| v4 | 0 | v4 | True |
| v5 | INF | None | False |
| v6 | INF | None | False |
| v7 | INF | None | False |
| v8 | INF | None | False |



Проведем релаксацию смежных вершин:

w’(v1) < w(v1): w(v1) = 2

w’(v2) < w(v2): w(v2) = 3

w’(v3) < w(v3): w(v3) = 4

w’(v6) < w(v6): w(v6) = 6

w’(v7) < w(v7): w(v7) = 8

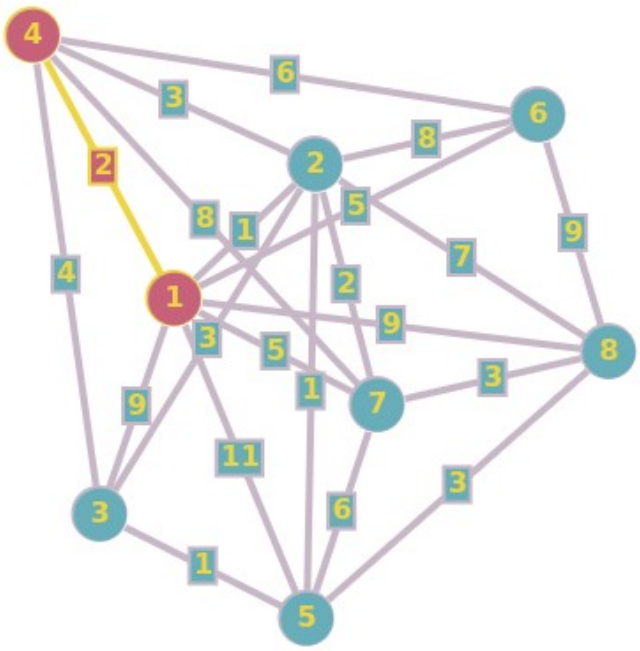
После релаксации получаем:

| | weight | predecessor | visited |
|---|---|---|---|
| v1 | 2 | None | False |
| v2 | 3 | None | False |
| v3 | 4 | None | False |
| v4 | 0 | v4 | True |
| v5 | INF | None | False |
| v6 | 6 | None | False |
| v7 | 8 | None | False |
| v8 | INF | None | False |

Выбираем непосещенную вершину с наименьшим weight. В данном случае – v1.

Обновляем таблицу

| | weight | predecessor | visited |
|---|---|---|---|
| v1 | 2 | v4 | True |
| v2 | 3 | None | False |
| v3 | 4 | None | False |
| v4 | 0 | v4 | True |
| v5 | INF | None | False |
| v6 | 6 | None | False |
| v7 | 8 | None | False |
| v8 | INF | None | False |




Проведем релаксацию смежных вершин:

w’(v2) < w(v2): w(v2) = 1

w’(v3) > w(v3)

v4 уже посещена

w’(v5) < w(v5): w(v5) = 11

w’(v6) < w(v6): w(v6) = 5

w’(v7) < w(v7): w(v7) = 5

w’(v8) < w(v8): w(v8) = 9

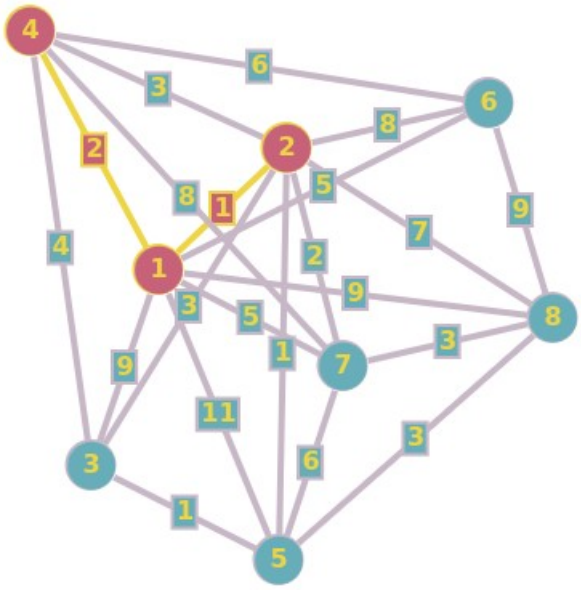
После релаксации получаем:

| | weight | predecessor | visited |
|---|---|---|---|
| v1 | 2 | v4 | True |
| v2 | 1 | None | False |
| v3 | 4 | None | False |
| v4 | 0 | v4 | True |
| v5 | 11 | None | False |
| v6 | 5 | None | False |
| v7 | 5 | None | False |
| v8 | 9 | None | False |

Выбираем непосещенную вершину с наименьшим weight. В данном случае – v2.

Обновляем таблицу.

| | weight | predecessor | visited |
|---|---|---|---|
| v1 | 2 | v4 | True |
| v2 | 1 | v1 | True |
| v3 | 4 | None | False |
| v4 | 0 | v4 | True |
| v5 | 11 | None | False |
| v6 | 5 | None | False |
| v7 | 5 | None | False |
| v8 | 9 | None | False |




Проведем релаксацию смежных вершин:

v1 уже посещена

w’(v3) < w(v3): w(v3) = 3

v4 уже посещена

w’(v5) < w(v5): w(v5) = 1

w’(v6) > w(v6)

w’(v7) < w(v7): w(v7) = 2

w’(v8) < w(v8): w(v8) = 7

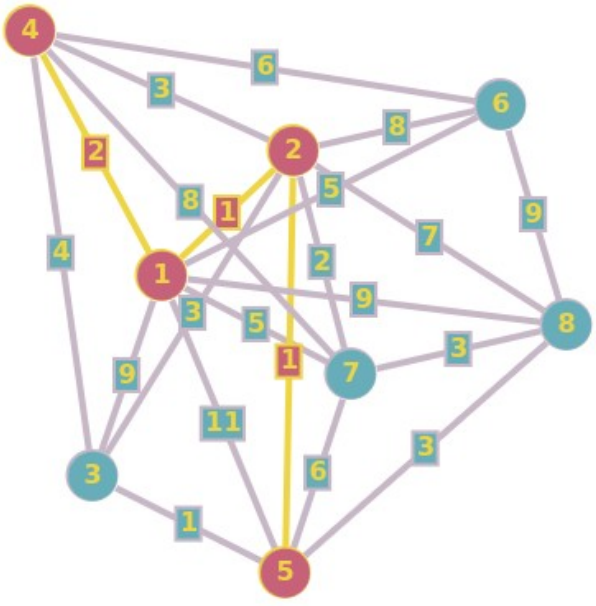
После релаксации получаем:

| | weight | predecessor | visited |
|---|---|---|---|
| v1 | 2 | v4 | True |
| v2 | 1 | v1 | True |
| v3 | 3 | None | False |
| v4 | 0 | v4 | True |
| v5 | 1 | None | False |
| v6 | 5 | None | False |
| v7 | 2 | None | False |
| v8 | 7 | None | False |

Выбираем непосещенную вершину с наименьшим weight. В данном случае – v5.

Обновляем таблицу.

| | weight | predecessor | visited |
|---|---|---|---|
| v1 | 2 | v4 | True |
| v2 | 1 | v1 | True |
| v3 | 3 | None | False |
| v4 | 0 | v4 | True |
| v5 | 1 | v2 | True |
| v6 | 5 | None | False |
| v7 | 2 | None | False |
| v8 | 7 | None | False |



Проведем релаксацию смежных вершин:

v1 уже посещена

v2 уже посещена

w’(v3) < w(v3): w(v3) = 1

v4 уже посещена

w’(v7) > w(v7)

w’(v8) < w(v8): w(v8) = 3

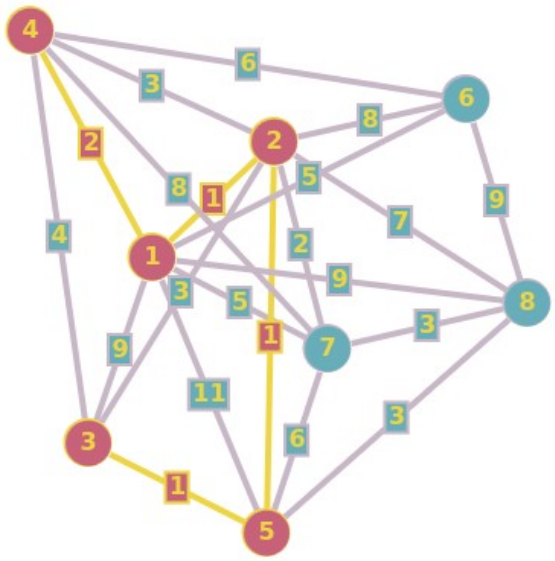
После релаксации получаем:

| | weight | predecessor | visited |
|---|---|---|---|
| v1 | 2 | v4 | True |
| v2 | 1 | v1 | True |
| v3 | 1 | None | False |
| v4 | 0 | v4 | True |
| v5 | 1 | v2 | True |
| v6 | 5 | None | False |
| v7 | 2 | None | False |
| v8 | 3 | None | False |

Выбираем непосещенную вершину с наименьшим weight. В данном случае – v3.

Обновляем таблицу.

| | weight | predecessor | visited |
|---|---|---|---|
| v1 | 2 | v4 | True |
| v2 | 1 | v1 | True |
| v3 | 1 | v5 | True |
| v4 | 0 | v4 | True |
| v5 | 1 | v2 | True |
| v6 | 5 | None | False |
| v7 | 2 | None | False |
| v8 | 3 | None | False |




Проведем релаксацию смежных вершин:

v1 уже посещена

v2 уже посещена

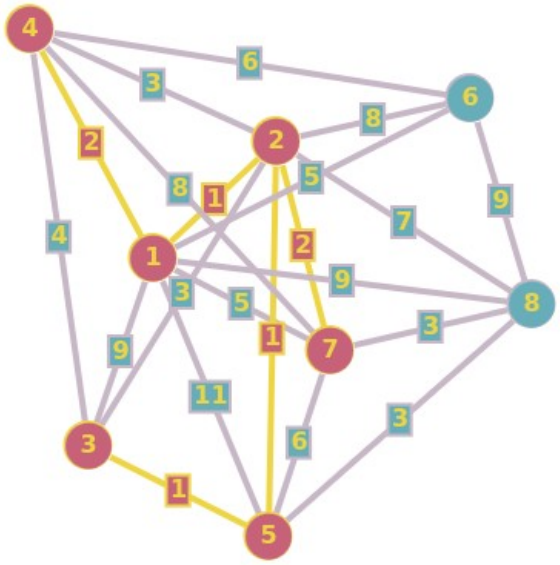
v4 уже посещена

v5 уже посещена

Выбираем непосещенную вершину с наименьшим weight. В данном случае – v7.

Обновляем таблицу.

| | weight | predecessor | visited |
|---|---|---|---|
| v1 | 2 | v4 | True |
| v2 | 1 | v1 | True |
| v3 | 1 | v5 | True |
| v4 | 0 | v4 | True |
| v5 | 1 | v2 | True |
| v6 | 5 | None | False |
| v7 | 2 | v2 | True |
| v8 | 3 | None | False |




Проведем релаксацию смежных вершин:

v1 уже посещена

v2 уже посещена

v4 уже посещена

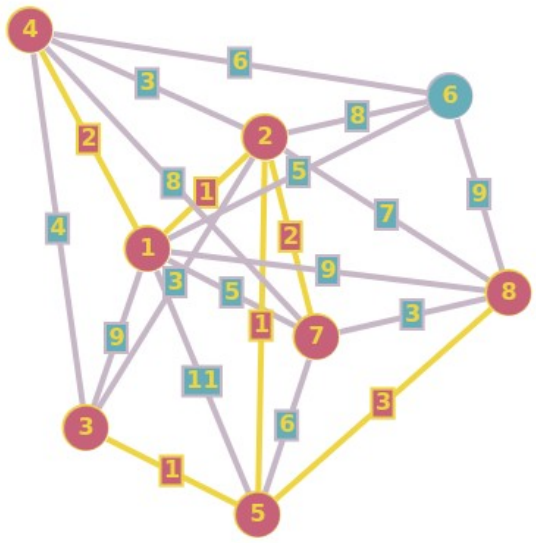
v5 уже посещена

w’(v8) = w(v8)

Выбираем непосещенную вершину с наименьшим weight. В данном случае – v8.

Обновляем таблицу.

| | weight | predecessor | visited |
|---|---|---|---|
| v1 | 2 | v4 | True |
| v2 | 1 | v1 | True |
| v3 | 1 | v5 | True |
| v4 | 0 | v4 | True |
| v5 | 1 | v2 | True |
| v6 | 5 | None | False |
| v7 | 2 | v2 | True |
| v8 | 3 | v5 | True |




Проведем релаксацию смежных вершин:

v1 уже посещена

v2 уже посещена

v5 уже посещена

w’(v6) > w(v6)

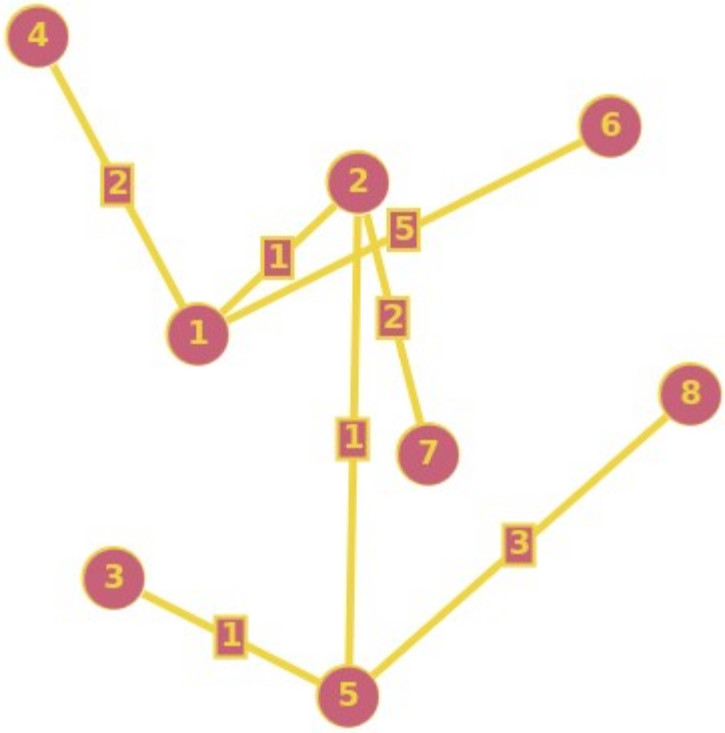
v7 уже посещена

Выбираем непосещенную вершину с наименьшим weight. В данном случае – v6.

Обновляем таблицу.

| | weight | predecessor | visited |
|---|---|---|---|
| v1 | 2 | v4 | True |
| v2 | 1 | v1 | True |
| v3 | 1 | v5 | True |
| v4 | 0 | v4 | True |
| v5 | 1 | v2 | True |
| v6 | 5 | v1 | True |
| v7 | 2 | v2 | True |
| v8 | 3 | v5 | True |

Все вершины посещены. Получено MST:




# Доказательство корректности

Поскольку алгоритм Прима является жадным, доверять ему нельзя.

Докажем, что алгоритм Прима всегда будет находить минимальное остовное дерево в графе.

**Теорема**: Если $S$ – остовное дерево, выбранное алгоритмом Прима из графа $G=(V,E)$, то $S$ – минимальное остовное дерево графа $G$.

**Доказательство**: Докажем от противного.

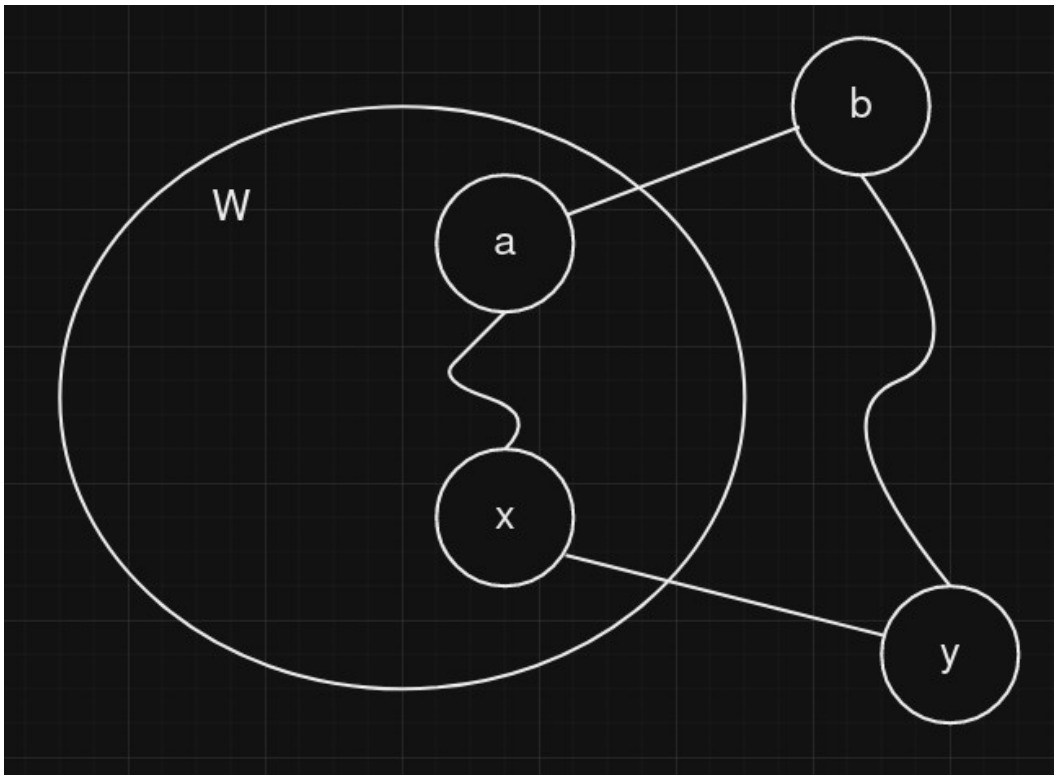
Предположим, что $S$ не имеет минимальный вес (т.е. не является минимальным остовным деревом).

Пусть $ES=(e_1,e_2,...e_{n-1})$ - последовательность ребер, выбранных алгоритмом Прима (ровно в таком порядке). Пусть $U$ – $MST$ **графа** $G$, содержащее ребра из **наибольшего возможного префикса** последовательности $ES$.

Вот алгоритм Прима выбирает ребро $e_i=\{\{x,y\}\}$, которое не входит в $U$. Пусть $W$ – набор вершин до выбора ребра $e_i$. Из этого можем заметить, что $U$ содержит ребра $(e_1,e_2,...e_{i-1})$ (префикс $ES$ длины $i-1$, сказано в абзаце выше), но не ребро $e_i$.

Т.к. в $U$ нет ребра $\{\{x,y\}\}$, но при этом U является $MST$ ( $V(U)=V(G)$ ), то в $U$ существует путь $P$ от вершины $x$ до вершины $y$. Пусть $\{\{a,b\}\}$ - ребро на пути

$P$ , такое, что вершина $a$ уже была помечена алгоритмом Прима, а вершина $b$ – еще нет. Это может выглядеть так:




Пусть набор ребер $T = E(U) + \{\{x, y\}\} - \{\{a, b\}\}$ , Заметим, что $T$ – остовное дерево графа $G$ . Рассмотрим 3 случая:

- $w(\{x, y\}) < w(\{a, b\})$ . Получается, что создавая остовное дерево $T$ мы из $U$ удалили ребро и добавили ребро с меньшим весом, чем то, которое удалили. Выходит, что $w(T) < w(U)$ , что делает $T$ минимальным остовным деревом графа $G$ . Но в начале мы установили, что $U$ – минимальное остовное дерево, т.е. дерево с наименьшим весом. Приходим к противоречию.
- $w(\{x, y\}) = w(\{a, b\})$ . То есть $T$ – минимальное остовное дерево графа $G$ , как и $U$ . Однако, так как алгоритм Прима еще не выбирал ребро $a, b$ , это ребро не является ни одним из ребер $(e_1, e_2, \dots e_{i-1})$ . Из этого следует, что $T$ состоит из ребер $(e_1, e_2, \dots e_{i-1}, e_i)$ . Данный набор ребер является наибольшим префиксом набора $ES$ . Но в начале мы установили, что $U$ содержит наибольший префикс набора $ES$ , а выходит, что для $T$ он больше. Приходим к противоречию.
- $w(\{x, y\}) < w(\{a, b\})$ . То есть алгоритм Прима выбрал ребро $\{\{x, y\}\}$ , которое не является минимальным. Однако алгоритм Прима на каждом шаге выбиут ребро с наименьшим весом, а значит, он выберет ребро $\{\{a, b\}\}$ , чего не может быть, так как мы установили, что на этом шаге он выбрал ребро $\{\{x, y\}\}$ . Приходим к противоречию.

Поскольку все 3 случая дали противоречия, то наше предположение о том, что $S$ не является минимальным остовным деревом, ложно. Значит, $S$ – минимальное остовное дерево графа $G$, выбранное алгоритмом Прима, что доказывает теорему и корректность работы алгоритма.

## Расчет ассимптотики

Время выполнения алгоритма зависит от способа хранения графа и способа хранения вершин, не входящих в дерево.

- Массив, списки смежности/матрица смежности. Если приоритетная очередь $Q$ реализована как обычный массив $d$, то извлечение минимума в ней будет выполняться за $O(n)$, а стоймость релаксации одного ребра занимает $O(1)$ ($d[u] \leftarrow w(v,u)$). То есть ассимптотика составляет $O(V^2)$.
- Бинарная куча, списки смежности. Если приоритетная очередь $Q$ реализована в виде бинарной кучи, то ассимптотика извлечения минимума падает до $O(\log n)$, но стоймость релаксации одного ребра возрастает до $O(\log n)$. Тогда ассимптотика равна $O((V+E)\log V)=O(E\log V)$.
- Фибоначчиева куча, списки смежности. Если приоритетная очередь $Q$ реализована в виде фибоначчиевой кучи, то ассимптотика извлечения минимума равна $O(\log n)$, а время релаксации одного ребра падает до $O(1)$. Тогда ассимптотика равна $O(E+V\log V)$.