Способы хранения графов
Обходы
Задачи

Способы хранения

задачи

И немножко C++17

Способы хранения графов

Способы хранения графов

- Матричный способ
  - Матрица смежности (вершины × вершины)
  - Матрица инцидентности (вершины × рёбра)

Способы хранения графов

- Матричный способ
  - Матрица смежности (вершины × вершины)
  - Матрица инцидентности (вершины × рёбра)
- Списки
  - Список смежности ([вершина → список смежных])
  - Список рёбер ([начала – конец – свойство(вес)])

Способы хранения графов

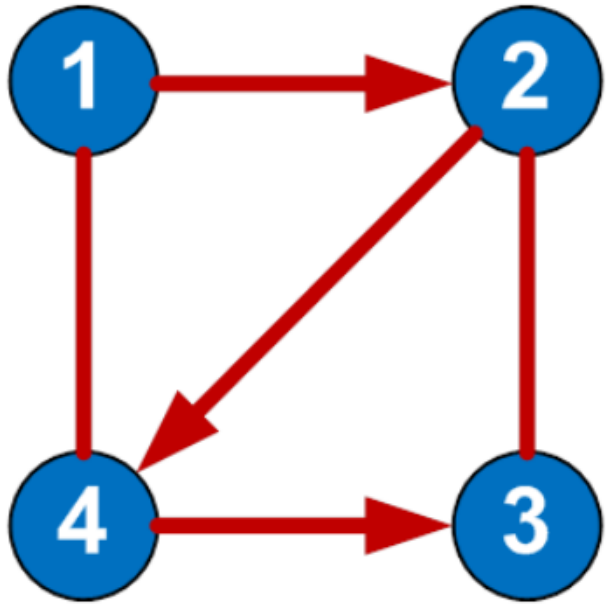
- Матричный способ
  - Матрица смежности (вершины × вершины)
  - Матрица инцидентности (вершины × рёбра)
- Списки
  - Список смежности ([вершина → список смежных])
  - Список рёбер ([начала – конец – свойство(вес)])
- Экзотика или узкоспециализированные
  - Вектор смежности, ассоциативный массив смежности

Способы хранения графов

- Матричный способ
  - Матрица смежности (вершины × вершины)
  - Матрица инцидентности (вершины × рёбра)
- Списки
  - Список смежности ([вершина → список смежных])
  - Список рёбер ([начала – конец – свойство(вес)])
- Экзотика или узкоспециализированные
  - Вектор смежности, ассоциативный массив смежности

Матрица смежности

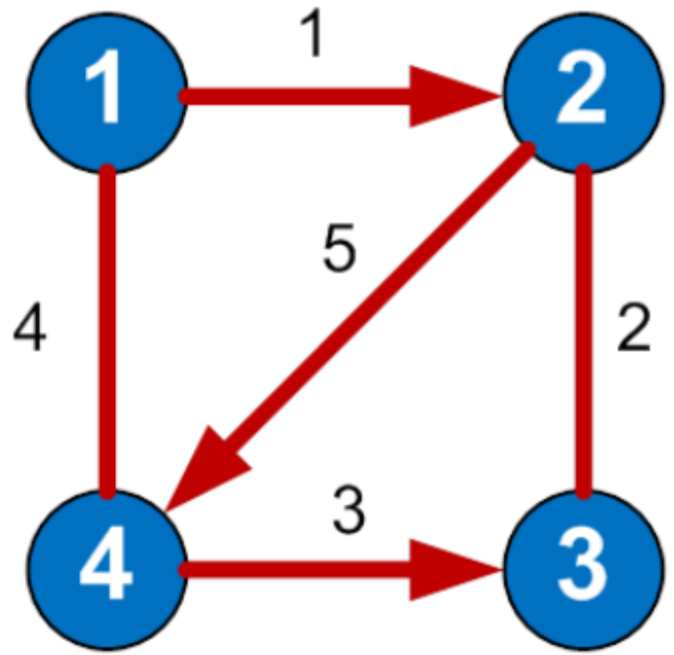
	1	2	3	4
1	0	1	0	1
2	0	0	1	1
3	0	1	0	0
4	1	0	1	0



- Размер не зависит от кол-ва рёбер → плохо хранить разреженные графы
- НО! Для индексации по массиву за O(1) требует нумерации вершин
- Легко добавлять/удалять рёбра - O(1)
- Сложно добавлять/удалять вершины - $O(|V|^2)$

Матрица инцидентности

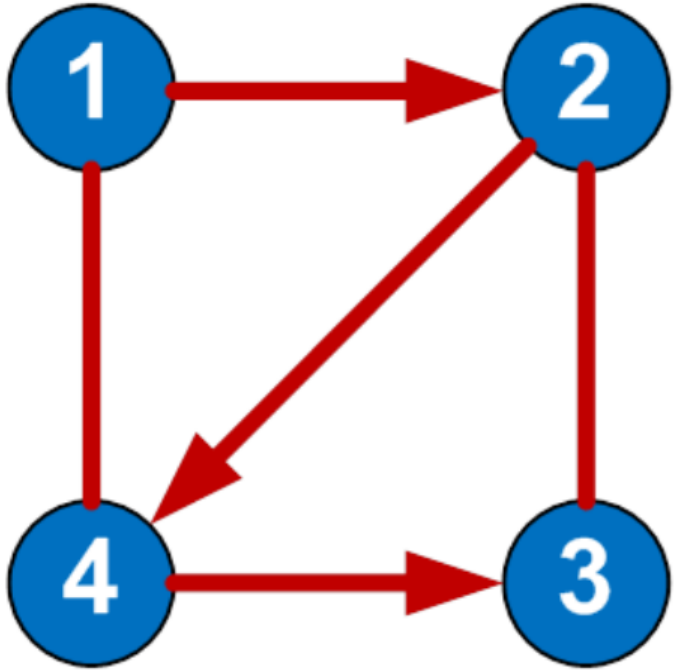
	1	2	3	4	5
1	1	0	0	1	0
2	-1	1	0	0	1
3	0	1	-1	0	0
4	0	0	1	1	-1



- Размер зависит от кол-ва и рёбер, и вершин
- Для использования требует явной нумерации рёбер
- Сложно изменять. Любое изменение – O(|V||E|)
- Для разреженных матриц чуть лучше по памяти

Список смежности

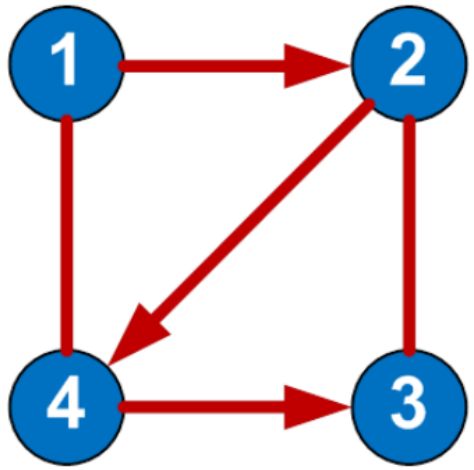
1	2, 4
2	3, 4
3	2
4	1, 3



- Экономно по памяти
- Просто изменять: добавление за O(1), удаление за O(|E|+|*V*|)
- Быстрый перебор соседей, но сложно проверить смежность вершин
- При большой связности графа константа увеличивается

Список рёбер

	Начало	Конец	Вес
1	1	2	
2	1	4	
3	2	3	
4	2	4	
5	3	2	
6	4	1	
7	4	3	



- Ужасно, но универсально
- Можно использовать для хранения промежуточного представления

Постановка задачи

- Данные о графе хранятся в структуре справа
- Подаются на std::cin (как в курсе на stepik)
- Необходимо создать представление графа в памяти по этим входным данным

```
a b 7
a c 6
b d 6
c f 9
d e 3
d f 4
e c 2
```

Создадим промежуточное представление в виде списка рёбер

```
14  // Список рёбер
15  class EdgesList {
16  private:
17      std::vector<std::tuple<std::string, std::string, int>> edges;
18
19  public:
20      EdgesList() = default;
21      EdgesList(std::istream &dataStream){
22          handleDataStream(dataStream);
23      }
24
25      void addEdge(std::tuple<std::string, std::string, int> edge){
26          edges.emplace_back(std::move(edge));
27      }
```

Немного страшно?

Немного страшно?

Решение есть:

Немного страшно?

Решение есть:

```
using VertexName = std::string;
using VertexID = int;
using EdgeWeight = int;
using Edge = std::tuple<VertexName, VertexName, EdgeWeight>;
using SemiEdge = std::pair<VertexName, EdgeWeight>;
using ListOfVertices = std::vector<VertexName>;
```

Теперь:

```
20  // Список рёбер
21  class EdgesList {
22  private:
23      std::vector<Edge> edges;
24
25  public:
26      EdgesList() = default;
27      EdgesList(std::istream &dataStream){
28          handleDataStream(dataStream);
29      }
30
31      void addEdge(Edge edge){
32          edges.emplace_back(std::move(edge));
33      }
```

Обработка входных данных с помощью std::stringstream

```
60  private:
61  void handleDataStream(std::istream &dataStream){
62      std::string curEdge;
63      std::stringstream iss;
64
65      while (std::getline(dataStream, curEdge)){
66          iss << curEdge;
67          std::string from;
68          std::string to;
69          int defaultEdgeLength = 1;
70
71          iss >> from >> to >> defaultEdgeLength;
72          assertm(from.size() != 0 and to.size() != 0,"Sizes must be not zeros");
73
74          addEdge(Edge{std::move(from), std::move(to), defaultEdgeLength});
75          iss.clear();
76      }
77  }
```

Обработка входных данных с помощью std::stringstream

```
60  private:
61  void handleDataStream(std::istream &dataStream){
62      std::string curEdge;
63      std::stringstream iss;
64
65      while (std::getline(dataStream, curEdge)){
66          iss << curEdge;
67          std::string from;
68          std::string to;
69          int defaultEdgeLength = 1;
70
71          iss >> from >> to >> defaultEdgeLength;
72          assertm(from.size() != 0 and to.size() != 0,"Sizes must be not zeros");
73
74          addEdge(Edge{std::move(from), std::move(to), defaultEdgeLength});
75          iss.clear();
76      }
77  }
```

Также в дальнейшем понадобится получить список вершин:

```
35    std::vector<std::string> getVertices() const {
36        std::vector<std::string> UniqueVertices;
37        for (auto [from, to, weight] : edges){
38            UniqueVertices.emplace_back(std::move(from));
39            UniqueVertices.emplace_back(std::move(to));
40        }
41        UniqueVertices.erase(std::unique(std::begin(UniqueVertices), std::end(UniqueVertices)),
42                             std::end(UniqueVertices));
43        return UniqueVertices;
44    }
```

И вывести всё это добро на консоль:

```
56    friend std::ostream& operator<<(std::ostream &out, const EdgesList& edgesList){
57        for (const auto &[from, to, weight] : edgesList.edges)
58            out << from << " - " << to << ": " << weight << "\n";
59        return out;
60    }
```

Также в дальнейшем понадобится получить список вершин:

```
35  std::vector<std::string> getVertices() const {
36      std::vector<std::string> UniqueVertices;
37      for (auto [from, to, weight] : edges){
38          UniqueVertices.emplace_back(std::move(from));
39          UniqueVertices.emplace_back(std::move(to));
40      }
41      UniqueVertices.erase(std::unique(std::begin(UniqueVertices), std::end(UniqueVertices)),
42                           std::end(UniqueVertices));
43      return UniqueVertices;
44  }
```

И вывести всё это добро на консоль:

```
56  friend std::ostream& operator<<(std::ostream &out, const EdgesList& edgesList){
57      for (const auto &[from, to, weight] : edgesList.edges)
58          out << from << " - " << to << ": " << weight << "\n";
59      return out;
60  }
```

Используем заранее созданный поток:

```
std::stringstream stream{"A B\nA C\nB D\nC F\nD E\nD F\nE C\n"};
```

Также можем читать сразу с консоли:

```
std::string inputLine;
std::stringstream stream;
while (std::getline(std::cin, inputLine)){
    stream << inputLine << "\n";
}
```

Используем заранее созданный поток:

```
std::stringstream stream{"A B\nA C\nB D\nC F\nD E\nD F\nE C\n"};
```

Также можем читать сразу с консоли:

```
std::string inputLine;
std::stringstream stream;
while (std::getline(std::cin, inputLine)){
    stream << inputLine << "\n";
}
```

Использование:

```
EdgesList obj(stream);
std::cout << obj;
```

Используем заранее созданный поток:

```
std::stringstream stream{"A B\nA C\nB D\nC F\nD E\nD F\nE C\n"};
```

Также можем читать сразу с консоли:

```
std::string inputLine;
std::stringstream stream;
while (std::getline(std::cin, inputLine)){
    stream << inputLine << "\n";
}
```

Использование:

```
EdgesList obj(stream);
std::cout << obj;
```

```
λ .\a.exe
A - B: 1
A - C: 1
B - D: 1
C - F: 1
D - E: 1
D - F: 1
E - C: 1
```

Используем заранее созданный поток:

```
std::stringstream stream{"A B\nA C\nB D\nC F\nD E\nD F\nE C\n"};
```

Также можем читать сразу с консоли:

```
std::string inputLine;
std::stringstream stream;
while (std::getline(std::cin, inputLine)){
    stream << inputLine << "\n";
}
```

Использование:

```
EdgesList obj(stream);
std::cout << obj;
```

```
λ .\a.exe
A - B: 1
A - C: 1
B - D: 1
C - F: 1
D - E: 1
D - F: 1
E - C: 1
```

Реализация списка смежности:

```
28  // Список смежности
29  class AdjacencyListImp {
30  private:
31      std::map<VertexName, std::list<SemiEdge>> AdjList;
32
33  public:
34      AdjacencyListImp() = default;
35      AdjacencyListImp(const EdgesList &graf_repr){
36          auto allVertices = graf_repr.getVertices();
37          for (auto vertex : allVertices)
38              AdjList[vertex].clear(); // Костыль для инициализации всех значений
39
40          for (auto [from, to, weight]: graf_repr){
41              AdjList[from].emplace_back(SemiEdge{std::move(to), std::move(weight)});
42          }
43      }
```

Как используем, что получилось:

```
AdjacencyListImp graf(obj);
std::cout << graf;
```

Для графа 1:

```
Adj List:
A: (B - 1) (C - 1)
B: (D - 1)
C: (F - 1)
D: (E - 1) (F - 1)
E: (C - 1)
F:
```

Для графа 2:

```
Adj List:
A: (B - 1)
B: (C - 1) (D - 1)
C:
D: (E - 1)
E: (G - 1)
G: (D - 1)
```

Обход в глубину

Для каждой не пройденной вершины необходимо найти все не пройденные смежные вершины и повторить поиск для них.

Решаемые задачи:

- обход графа
- проверка на ацикличность
- проверка связности
- топологическая сортировка

Обходим граф 1 и граф 2

Для каждой не пройденной вершины необходимо найти все не пройденные смежные вершины и повторить поиск для них.

Граф 1:

Стек:

Граф 2:

Стек:

Обходим граф 1 и граф 2

Для каждой не пройденной вершины необходимо найти все не пройденные смежные вершины и повторить поиск для них.

Граф 1: A B D E C F

Стек:

Pop A
Push C
Push B
Pop B
Push D
Pop D
Push F
Push E
Pop E
Push C
Pop C
Push F
Pop F
Pop F
Pop C

Граф 2:

Стек:

Обходим граф 1 и граф 2

Для каждой не пройденной вершины необходимо найти все не пройденные смежные вершины и повторить поиск для них.

Граф 1: A B D E C F

Стек:

Pop A
Push C
Push B
Pop B
Push D
Pop D
Push F
Push E
Pop E
Push C
Pop C
Push F
Pop F
Pop F
Pop C

Граф 2: A B C D E G

Стек:

Pop A
Push B
Pop B
Push D
Push C
Pop C
Pop D
Push E
Pop E
Push G
Pop G

Реализация DFS - рекурсивная

```
45  ListOfVertices recursiveDFS(){
46      auto firstVertex = AdjList.begin()->first;
47      ListOfVertices result;
48
49      dfs(result, firstVertex);
50      return result;
51  }
103 void dfs(ListOfVertices& visited, VertexName vertex){
104     visited.push_back(vertex);
105     for (const auto [adj_vertex, _] : AdjList[vertex]){
106         if (std::find(std::begin(visited),
107                 std::end(visited), adj_vertex) == std::end(visited)){
108             dfs(visited, adj_vertex);
109         }
110     }
111 }
```

Реализация DFS – итеративная

```
53  ListOfVertices iterativeDFS(){
54      using namespace std;
55      auto firstVertex = AdjList.begin()->first;
56      ListOfVertices result;
57      stack<VertexName> stack; stack.push(firstVertex);
58
59      while (stack.empty() == false){
60          auto vertex = stack.top(); stack.pop();
61
62          if (find(begin(result), end(result), vertex) == end(result))
63              result.push_back(vertex);
64          for (auto it = AdjList[vertex].rbegin(); it != AdjList[vertex].rend(); ++it){
65              auto [adj_vertex, _] = *it;
66              if (find(begin(result), end(result), adj_vertex) == end(result))
67                  stack.push(adj_vertex);
68          }
69      }
70      return result;
71  }
```

Поиск цикла - псевдокод

```
func dfs(v: vertex):
    color[v] = grey
    for (u: vu ∈ E)
        if (color[u] == white)
            dfs(u)
        if (color[u] == grey)
            print()
    color[v] = black
```

Цвет вершины:

Белая - вершина не пройдена

Серая — вершина активная (в текущем dfs)

Чёрная — вершина пройдена (все итерации dfs от нее завершены)

Поиск цикла - реализация

```
114  void dfs_cycle(ListOfVertices& visited, std::map<VertexName, Color>& colored,
115                                                  VertexName vertex){
116      colored[vertex] = Color::GREY;
117      visited.push_back(vertex);
118
119      for (const auto [adj_vertex, _] : AdjList[vertex]){
120          if (colored[adj_vertex] == Color::WHITE){
121              if (std::find(std::begin(visited),
122                      std::end(visited), adj_vertex) == std::end(visited)){
123                  dfs_cycle(visited, colored, adj_vertex);
124              }
125          }
126          if (colored[adj_vertex] == Color::GREY){
127              std::cout << "Cycle!\n";
128              visited.push_back(adj_vertex);
129          }
130      }
131      colored[vertex] = Color::BLACK;
132  }
```

Поиск цикла - использование

Запуск для графа 1:

```
λ .\a.exe
A B D E C F
```

Запуск для графа 2:

```
λ .\a.exe
Cycle!
A B C D E G D
```

Запуск для графа 1-штрих:

```
λ .\a.exe
Cycle!
Cycle!
A B D E C F A B
```

2. Покрасьте вершины неориентированного графа в два цвета, чтобы все ребра соединяли вершины разных цветов, за $\mathcal{O}(E + V)$.

5. Инспектору нужно проверить состояние дорог в городе, для этого он хочет проехать по каждой дороге в каждую сторону (все дороги двусторонние). Постройте кратчайший путь. $\mathcal{O}(E + V)$.

Чётный лес. Дан ацикличный граф. Найти максимальное кол-во рёбер, которые можно удалить так, чтобы каждая компонента связности графа содержала чётное кол-во вершин.