Комбинаторные алгоритмы

Поиск в графе

Гальперин Александр Леонидович

2018 г.

Поиск в графе

Разделы

- Поиск в глубину
- Поиск в ширину

Поиск в графе

- Рассматриваем два стандартных метода систематического обхода графа: поиск в глубину (Depth First Search) и поиск в ширину (Breadth First Search).

Поиск в графе

- Рассматриваем два стандартных метода систематического обхода графа: поиск в глубину (Depth First Search) и поиск в ширину (Breadth First Search).
- Рассматриваем *обыкновенные графы*. На произвольные графы методы распространяются очевидным образом.

Поиск в глубину

Идея метода: поиск в обыкновенном графе G из некоторой начальной вершины v (с этого момента v — *просмотрена*).
Пусть u — последняя просмотренная вершина (возможно, $u = v$).

Поиск в глубину

Возможны два случая:

Поиск в глубину

Возможны два случая:

- **Среди вершин, смежных с u, существует еще непросмотренная вершина w.**

Поиск в глубину

Возможны два случая:

- **Среди вершин, смежных с u, существует еще непросмотренная вершина w.**
  Тогда w объявляется просмотренной, и поиск продолжается из вершины w. При этом назовем вершину u — *отцом* вершины w ($u = father[w]$).

Поиск в глубину

Возможны два случая:

- **Среди вершин, смежных с u, существует еще непросмотренная вершина w.**
  Тогда w объявляется просмотренной, и поиск продолжается из вершины w. При этом назовем вершину u — *отцом* вершины w ($u = father[w]$).
  Ребро uw назовем *древесным*.
- **Все вершины, смежные с u, просмотрены.**

Поиск в глубину

Возможны два случая:

- **Среди вершин, смежных с u, существует еще непросмотренная вершина w.**
  Тогда w объявляется просмотренной, и поиск продолжается из вершины w. При этом назовем вершину u — *отцом* вершины w ($u = father[w]$).
  Ребро uw назовем *древесным*.
- **Все вершины, смежные с u, просмотрены.** Тогда u — *использованная вершина*. Поиск продолжается из вершины $x = father[u]$, т.е. из той вершины, из которой мы попали в вершину u.

Поиск в глубину

Что произойдет, когда все просмотренные вершины будут использованы?

Поиск в глубину

Что произойдет, когда все просмотренные вершины будут использованы?

- Если в графе G не осталось непросмотренных вершин, то поиск заканчивается.

Поиск в глубину

Что произойдет, когда все просмотренные вершины будут использованы?

- Если в графе G не осталось непросмотренных вершин, то поиск заканчивается.
- Если осталась непросмотренная вершина y, то поиск продолжается из этой вершины.

Поиск в глубину

Поиск в глубину просматривает вершины в определенном порядке. Для того, чтобы зафиксировать этот порядок, используем массив $num[v]$.

Поиск в глубину

Поиск в глубину просматривает вершины в определенном порядке. Для того, чтобы зафиксировать этот порядок, используем массив $num[v]$.

- При этом естественно считать, что для начальной вершины $num[v] = 1$.

Поиск в глубину

Поиск в глубину просматривает вершины в определенном порядке. Для того, чтобы зафиксировать этот порядок, используем массив $num[v]$.

- При этом естественно считать, что для начальной вершины $num[v] = 1$.
- Если вершина w просматривается сразу после вершины u, то $num[w] = num[u] + 1$.

Поиск в глубину

Пусть в обыкновенном графе G произведен поиск в глубину. Обозначим через T множество всех древесных ребер.

Поиск в глубину

Пусть в обыкновенном графе G произведен поиск в глубину. Обозначим через T множество всех древесных ребер.
Все оставшиеся ребра будем называть *обратными*. Множество всех обратных ребер будем обозначать через B.

Поиск в глубину

Результат применения поиска в глубину к связному графу G показан на рисунке. Здесь сплошные линии изображают древесные ребра, а пунктирные - обратные ребра.

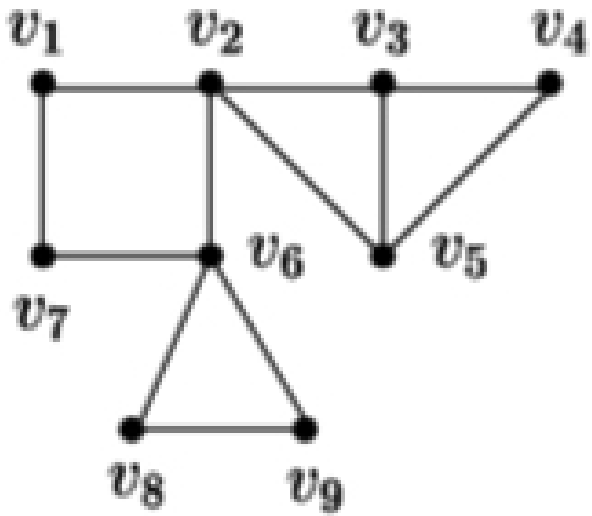


Рис.: Результат применения поиска в глубину

Поиск в глубину

Заметим, что

- нумерация вершин соответствует порядку обхода поиском в глубину;
- множество всех древесных ребер с выделенной начальной вершиной v_1 образует корневое дерево с вершиной в v_1. Это дерево называют *глубинным* деревом или *d–деревом*;
- каждое обратное ребро соединяет в d–дереве предка и потомка.

Поиск в глубину

Пусть G — несвязный граф, $G_1, \dots G_k$ — множество всех его компонент связности.

Поиск в глубину

Пусть G — несвязный граф, $G_1, \ldots G_k$ — множество всех его компонент связности. Обозначим через T_i множество древесных ребер, выделенных поиском в глубину в компоненте G_i, а через v_i — корневую вершину из G_i.
Таким образом, множество всех древесных ребер несвязного графа образует *остовный лес*. Фиксируя в каждом поддереве этого леса корневую вершину, мы получаем *глубинный лес* или *d–лес*.

Поиск в глубину

Формальное описание алгоритма

Поиск в глубину

Формальное описание алгоритма

- В алгоритме используются ранее описанные массивы *father* и *num*;

Поиск в глубину

Формальное описание алгоритма

- В алгоритме используются ранее описанные массивы *father* и *num*;
- *num* используется для распознавания непросмотренных вершин: равенство $num[v] = 0$ означает, что вершина v еще не просмотрена.

Поиск в глубину

Версия алгоритма поиска в глубину, основанная на рекурсивной процедуре $DFS(v)$, осуществляющей поиск в глуюину из вершины v

Рекурсивный алгоритм

```
1. procedure DFS(v);
2. begin
3.     num[v] := 1; i := i + 1;
4.     for u ∈ list[v] do
5.         if num[u] = 0 then
6.             begin
7.                 T := T ∪ {uv}; father[u] := v; DFS(u);
8.             end
9.         else if num[u] < num[v] and u ≠ father[v] then
10.                B := B ∪ {uv};
11. end;
```

Поиск в глубину

Версия алгоритма поиска в глубину, основанная на рекурсивной процедуре $DFS(v)$, осуществляющей поиск в глуюину из вершины v

Рекурсивный алгоритм: продолжение

```
12. begin
13.     i := 1; T := ∅; B := ∅;
14.     for v ∈ V do num[v] := 0;
15.     for v ∈ V do
16.       if num[v] = 0 then
17.         begin
18.           father[v] := ∅; DFS(v)
19.         end
20. end.
```

Поиск в глубину

Этот алгоритм применим к произвольному графу G.

Поиск в глубину

Этот алгоритм применим к произвольному графу G.

- Если граф G — связный, то цикл в строках 15–19 достаточно заменить вызовом процедуры $DFS(v_0)$ применительно к начальной вершине v_0.

Поиск в глубину

Этот алгоритм применим к произвольному графу G.

- Если граф G — связный, то цикл в строках 15–19 достаточно заменить вызовом процедуры $DFS(v_0)$ применительно к начальной вершине v_0.
- В терминах алгоритма вершина v просмотрена с началом просмотра процедуры $DFS(v)$.

Поиск в глубину

Этот алгоритм применим к произвольному графу G.

- Если граф G — связный, то цикл в строках 15–19 достаточно заменить вызовом процедуры $DFS(v_0)$ применительно к начальной вершине v_0.
- В терминах алгоритма вершина v просмотрена с началом просмотра процедуры $DFS(v)$.
- В тот момент, когда процедура $DFS(v)$ закончила работу, вершина v становится использованной.

Поиск в глубину

Теорема 1

Пусть G — связный (n, m)–граф. Тогда
1) *поиск в глубину просматривает каждую вершину в точности один раз*;
2) *поиск в глубину требует $O(n + m)$ операций*;
3) *подграф (V, T) графа G является деревом*.

Поиск в глубину

Доказательство теоремы 1

Доказательство.

1) Проверка в строке 5 гарантирует, что каждая вершина просматривается не более одного раза.

Поиск в глубину

Доказательство теоремы 1

Доказательство.
1) Проверка в строке 5 гарантирует, что каждая вершина просматривается не более одного раза.
Убедимся, что поиск просматривает каждую вершину.

Поиск в глубину

Доказательство теоремы 1

Доказательство.
1) Проверка в строке 5 гарантирует, что каждая вершина просматривается не более одного раза.
Убедимся, что поиск просматривает каждую вершину.

о/п Пусть X — множество просмотренных вершин в тот момент, когда алгоритм закончил работу, $Y = V \backslash X$.

Поиск в глубину

Доказательство теоремы 1

Доказательство.
1) Проверка в строке 5 гарантирует, что каждая вершина просматривается не более одного раза.
Убедимся, что поиск просматривает каждую вершину.

о/п Пусть X — множество просмотренных вершин в тот момент, когда алгоритм закончил работу, $Y = V \backslash X$.
Если $Y \neq \varnothing$, то в силу связности графа G существует ребро xy : $x \in X, y \in Y$.

Поиск в глубину

Доказательство теоремы 1

Доказательство.
1) Проверка в строке 5 гарантирует, что каждая вершина просматривается не более одного раза.
Убедимся, что поиск просматривает каждую вершину.

о/п Пусть X — множество просмотренных вершин в тот момент, когда алгоритм закончил работу, $Y = V \backslash X$.
Если $Y \neq \varnothing$, то в силу связности графа G существует ребро xy : $x \in X, y \in Y$.
Но процедура $DFS(v)$ полностью отработала, поэтому смежная с x вершина y должна быть просмотрена. Противоречие

Поиск в глубину

Доказательство теоремы 1

2)Число повторений цикла в процедуре

начало в строке 4: **for** $u \in list[v]$ **do**

с учетом рекурсивных вызовов равно сумме степеней всех вершин графа, т.е. $2m$. Следовательно, число операций пропорционально m.

Поиск в глубину

Доказательство теоремы 1

2)Число повторений цикла в процедуре

$$\text{начало в строке 4: } \textbf{for } u \in list[v] \textbf{ do}$$

с учетом рекурсивных вызовов равно сумме степеней всех вершин графа, т.е. $2m$. Следовательно, число операций пропорционально m.

Число повторений в цикле в строке 14

$$\textbf{for } v \in V \textbf{ do } num[v] := 0$$

пропорционально n.

Поиск в глубину

Доказательство теоремы 1

2)Число повторений цикла в процедуре

$$\text{начало в строке 4: } \textbf{for } u \in list[v] \textbf{ do}$$

с учетом рекурсивных вызовов равно сумме степеней всех вершин графа, т.е. $2m$. Следовательно, число операций пропорционально m.

Число повторений в цикле в строке 14

$$\textbf{for } v \in V \textbf{ do } num[v] := 0$$

пропорционально n.

Следовательно, поиск в глубину требует $O(n + m)$ операций.

Поиск в глубину

Доказательство теоремы 1

3)Ясно, что условие $num[v] = 0$ (строка 5) выполняется $n - 1$ раз. Следовательно, $|T| = n - 1$.

Поиск в глубину

Доказательство теоремы 1

3)Ясно, что условие $num[v] = 0$ (строка 5) выполняется $n - 1$ раз. Следовательно, $|T| = n - 1$.
Кроме того, из 1) вытекает, что множество ребер T не содержит циклов.

Поиск в глубину

Доказательство теоремы 1

3)Ясно, что условие $num[v] = 0$ (строка 5) выполняется $n - 1$ раз. Следовательно, $|T| = n - 1$.
Кроме того, из 1) вытекает, что множество ребер T не содержит циклов.
Таким образом, граф (V, T) ацикличен и содержит ребер на единицу меньше, чем вершин. Значит, (V, T) — дерево. ■

Поиск в глубину

Пусть G — связный граф, и из его вершины v_0 произведен поиск в глубину. Дерево (V, T) с выделенной вершиной v_0 — корневое (это и есть d– дерево).

Поиск в глубину

Пусть G — связный граф, и из его вершины v_0 произведен поиск в глубину. Дерево (V, T) с выделенной вершиной v_0 — корневое (это и есть d– дерево).

$\mathcal{NB}$

Заметим, что $\forall u \neq v_0$ вершина $father[u]$ является отцом и в d–дереве.

Поиск в глубину

Нерекурсивная версия алгоритма поиска в глубину

Рассмотрим нерекурсивную версию процедуры $DFS(v)$Рекурсия устраняется при помощи **стека** S, элементами которого являются вершины графа.

Поиск в глубину

Нерекурсивная версия алгоритма поиска в глубину

Рассмотрим нерекурсивную версию процедуры $DFS(v)$Рекурсия устраняется при помощи **стека** S, элементами которого являются вершины графа.

- Вершина v является просмотренной, если $num[v] \neq 0$.

Поиск в глубину

Нерекурсивная версия алгоритма поиска в глубину

Рассмотрим нерекурсивную версию процедуры $DFS(v)$Рекурсия устраняется при помощи **стека** S, элементами которого являются вершины графа.

- Вершина v является просмотренной, если $num[v] \neq 0$.
- Вершина v становится использованной с того момента, когда $v = top[S]$ (v находится в вершине стека) и все вершины, смежные с v, уже просмотрены (в этом случае вершина v удаляется из стека).

Поиск в глубину

Нерекурсивная версия алгоритма поиска в глубину

Рассмотрим нерекурсивную версию процедуры $DFS(v)$Рекурсия устраняется при помощи **стека** S, элементами которого являются вершины графа.

- Вершина v является просмотренной, если $num[v] \neq 0$.
- Вершина v становится использованной с того момента, когда $v = top[S]$ (v находится в вершине стека) и все вершины, смежные с v, уже просмотрены (в этом случае вершина v удаляется из стека).

$\mathcal{NB}$

Вычисления, связанные с множеством обратных ребер B, здесь опущены. В качестве упражнения их можно восстановить.

Поиск в глубину

Нерекурсивная версия алгоритма поиска в глубину

Нерекурсивный алгоритм

```
1, procedure DFS(v)
2. begin
3.       num[v] := i;    i := i + 1;
4.       S := nil;    S ⇐ v;
5.       while S ≠ nil do
6.          begin
7.             v := top(S);
```

Поиск в глубину

Нерекурсивная версия алгоритма поиска в глубину

Нерекурсивный алгоритм: продолжение

8. **if** $\exists u \in list[v]$ **and** $num[u] = 0$
9. **then**
10. **begin**
11. $num[u] := i; \quad i := i + 1; \quad T := T \cup \{uv\};$
12. $father[u] := v; \quad S \Leftarrow u;$
13. **end**
14. **else** $v \Leftarrow S$
15. **end**
16. **end;**

Поиск в глубину

$\mathcal{NB}$

Если xy — обратное ребро, то вершины x и y сравнимы в d–дереве, т.е. одна из них является предком другой.

Поиск в глубину

NB

Если xy — обратное ребро, то вершины x и y сравнимы в d–дереве, т.е. одна из них является предком другой.

Действительно, пусть xy — обратное ребро графа G, причем

$$num[x] < num[y].$$

о/п Предположим, что вершины x и y несравнимы в d–дереве. Из описания алгоритма следует, что в процессе работы процедуры $DFS(x)$ будут просмотрены только потомки вершины x. Поскольку $y \in list[x]$ и в момент завершения процедуры $DFS(x)$ вершина y еще не просмотрена, получаем Противоречие

Поиск в ширину

- В этом случае вместо стека будем использовать очередь Q, элементами которой будут являться вершины графа G.

Поиск в ширину

- В этом случае вместо стека будем использовать очередь Q, элементами которой будут являться вершины графа G.
- Поиск начинается с некоторой вершины v.

Поиск в ширину

- В этом случае вместо стека будем использовать очередь Q, элементами которой будут являться вершины графа G.
- Поиск начинается с некоторой вершины v.
- Эта вершина помещается в очередь Q и с этого момента считается *просмотренной*.

Поиск в ширину

- В этом случае вместо стека будем использовать очередь Q, элементами которой будут являться вершины графа G.
- Поиск начинается с некоторой вершины v.
- Эта вершина помещается в очередь Q и с этого момента считается *просмотренной*.
- Затем все вершины смежные с v включаются в очередь и получают статус *просмотренных*, а вершина v из очереди удаляется.

Поиск в ширину

- Более общо, пусть в начале очереди находится вершина u.

Поиск в ширину

- Более общо, пусть в начале очереди находится вершина u.
- Обозначим через $u_1, \ldots, u_p$ еще непросмотренные вершины, смежные с u.

Поиск в ширину

- Более общо, пусть в начале очереди находится вершина u.
- Обозначим через $u_1, \ldots, u_p$ еще непросмотренные вершины, смежные с u.
- Помещаем вершины $u_1, \ldots, u_p$ в очередь Q и с этого момента считаем их *просмотренными*. Удаляем вершину u из очереди.
- Присваем вершине u статус *использованной*.

Поиск в ширину

- Более общо, пусть в начале очереди находится вершина u.
- Обозначим через $u_1, \ldots, u_p$ еще непросмотренные вершины, смежные с u.
- Помещаем вершины $u_1, \ldots, u_p$ в очередь Q и с этого момента считаем их *просмотренными*. Удаляем вершину u из очереди.
- Присваем вершине u статус *использованной*.
- В этой ситуации вершина u называется *отцом* для каждой из вершин $u_1, \ldots, u_p$: $u = father[u_i], (1 \leqslant i \leqslant p)$.

Поиск в ширину

- Более общо, пусть в начале очереди находится вершина u.
- Обозначим через $u_1, \ldots, u_p$ еще непросмотренные вершины, смежные с u.
- Помещаем вершины $u_1, \ldots, u_p$ в очередь Q и с этого момента считаем их *просмотренными*. Удаляем вершину u из очереди.
- Присваем вершине u статус *использованной*.
- В этой ситуации вершина u называется *отцом* для каждой из вершин $u_1, \ldots, u_p$: $u = father[u_i], (1 \leqslant i \leqslant p)$.
- Каждое из ребер $uu_i (1 \leqslant i \leqslant p)$ назовем *древесным*.

Поиск в ширину

- Более общо, пусть в начале очереди находится вершина u.
- Обозначим через $u_1, \ldots, u_p$ еще непросмотренные вершины, смежные с u.
- Помещаем вершины $u_1, \ldots, u_p$ в очередь Q и с этого момента считаем их *просмотренными*. Удаляем вершину u из очереди.
- Присваем вершине u статус *использованной*.
- В этой ситуации вершина u называется *отцом* для каждой из вершин $u_1, \ldots, u_p$: $u = father[u_i], (1 \leqslant i \leqslant p)$.
- Каждое из ребер $uu_i (1 \leqslant i \leqslant p)$ назовем *древесным*.
- В тот момент, когда очередь Q окажется пустой, поиск в ширину обойдет компоненту связности графа G.

Поиск в ширину

- Более общо, пусть в начале очереди находится вершина u.
- Обозначим через $u_1, \ldots, u_p$ еще непросмотренные вершины, смежные с u.
- Помещаем вершины $u_1, \ldots, u_p$ в очередь Q и с этого момента считаем их *просмотренными*. Удаляем вершину u из очереди.
- Присваем вершине u статус *использованной*.
- В этой ситуации вершина u называется *отцом* для каждой из вершин $u_1, \ldots, u_p$: $u = father[u_i], (1 \leqslant i \leqslant p)$.
- Каждое из ребер $uu_i (1 \leqslant i \leqslant p)$ назовем *древесным*.
- В тот момент, когда очередь Q окажется пустой, поиск в ширину обойдет компоненту связности графа G.
- Если остались непросмотренные вершины (граф G несвязен), поиск в ширину продолжается из некоторой непросмотренной вершины.

Поиск в ширину

- Поиск в ширину просматривает вершины в определенном порядке. Как и раньше, фиксируем его в массиве *num*.

Поиск в ширину

- Поиск в ширину просматривает вершины в определенном порядке. Как и раньше, фиксируем его в массиве num.
- Если $u = father[u_i] (1 \leqslant i \leqslant p)$, то

$$num[u_i] = num[u] + i \quad (1 \leqslant i \leqslant p)$$

(для определенности полагаем, что сначала просматривается вершина u_1, затем u_2 и т.д.).

Поиск в ширину

- Поиск в ширину просматривает вершины в определенном порядке. Как и раньше, фиксируем его в массиве num.
- Если $u = father[u_i] (1 \leqslant i \leqslant p)$, то

$$num[u_i] = num[u] + i \quad (1 \leqslant i \leqslant p)$$

(для определенности полагаем, что сначала просматривается вершина u_1, затем u_2 и т.д.).

- Для начальной вершины v естественно положить $num[v] = 1$.

Поиск в ширину

- Поиск в ширину реализует процедура $BFS(v)$ (Breadth First Search).

Поиск в ширину

- Поиск в ширину реализует процедура $BFS(v)$ (Breadth First Search).
- Использует описанные раньше массивы *father* и *num*.

Поиск в ширину

- Поиск в ширину реализует процедура $BFS(v)$ (Breadth First Search).
- Использует описанные раньше массивы *father* и *num*.
- Вычисляет множество всех древесных ребер T.

Поиск в ширину

- Поиск в ширину реализует процедура $BFS(v)$ (Breadth First Search).
- Использует описанные раньше массивы *father* и *num*.
- Вычисляет множество всех древесных ребер T.
- Массив *num* удобно использовать для распознавания всех непросмотренных вершин. Равенство $num[v] = 0$ обозначает, что вершина v не просмотрена.

Поиск в ширину

Алгоритм поиска в ширину

```
1. procedure BFS(v)
2. begin
3.     Q := nil;    Q ⇐ v;    num[v] := i;    i := i + 1
4.     while Q ≠ nil do
5.         begin
6.             u ⇐ Q;
7.             for w ∈ list[u] do
8.                 if num[w] = 0 then
9.                     begin
10.                        Q ⇐ w;    father[w] := u;
11.                        num[w] := i;    i := i + 1;    T := T ∪ {uw};
12.                    end
```

Поиск в ширину

Алгоритм поиска в ширину: продолжение

```
13.         end
14. end
15. begin
16.     i := 1;    T := ∅;
17.     for v ∈ V do num[v] := 0;
18.     for v ∈ V do
19.         if num[v] = 0 then
20.             begin father[v] := 0;    BFS(v) end
21. end.
```

Поиск в ширину

NB1

Полезно убедиться, что нерекурсивная процедура $DFS(v)$ отличается от $BFS(v)$ заменой стека на очередь.

Поиск в ширину

$\mathcal{NB}1$

Полезно убедиться, что нерекурсивная процедура $DFS(v)$ отличается от $BFS(v)$ заменой стека на очередь.

$\mathcal{NB}2$

Заметим также, что если применить этот алгоритм к связному графу G, то можно цикл в строках 18–20 заменить однократным вызовом процедуры BFS.

Поиск в ширину

Теорема 2

Пусть G — связный (n, m)–граф. Тогда:

1. *поиск в ширину просматривает каждую вершину в точности один раз;*
2. *поиск в ширину потребует $O(n + m)$ операций;*
3. *подграф (V, T) графа G является деревом.*

Поиск в ширину

Теорема 2

Пусть G — связный (n, m)–граф. Тогда:

1. *поиск в ширину просматривает каждую вершину в точности один раз;*
2. *поиск в ширину потребует $O(n + m)$ операций;*
3. *подграф (V, T) графа G является деревом.*

Эта теорема доказывает аналогично теореме 1.

Поиск в ширину

- В связном графе G поиск в ширину из вершины v строит корневое дерево с множеством ребер T и корнем v.

Поиск в ширину

- В связном графе G поиск в ширину из вершины v строит корневое дерево с множеством ребер T и корнем v.
- Это дерево называется *деревом поиска в ширину* или *b–деревом*.

Поиск в ширину

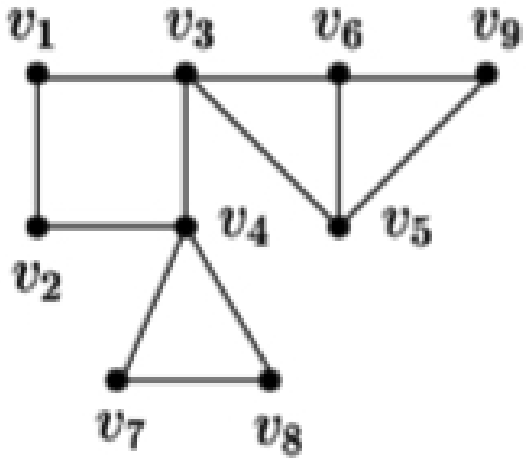
- В связном графе G поиск в ширину из вершины v строит корневое дерево с множеством ребер T и корнем v.
- Это дерево называется *деревом поиска в ширину* или *b–деревом*.
- Аналогично, если G — произвольный обыкновенный граф, то поиск в ширину строит b–дерево в каждой компоненте связности графа G.

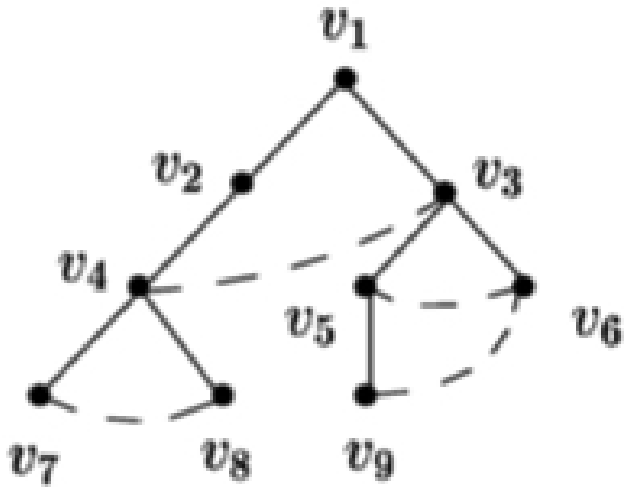
Поиск в ширину

- В связном графе G поиск в ширину из вершины v строит корневое дерево с множеством ребер T и корнем v.
- Это дерево называется *деревом поиска в ширину* или *b–деревом*.
- Аналогично, если G — произвольный обыкновенный граф, то поиск в ширину строит b–дерево в каждой компоненте связности графа G.
- Объединяя эти деревья, мы получим остовный лес графа G, называемый *b–лесом* этого графа.

Поиск в ширину

На рисунке показаны граф G и его b–дерево.





Поиск в ширину

- Пусть в графе G произвели поиск в ширину. Занумеруем вершины графа в соответствии с порядком, в котором поиск в ширину обходит вершины.

Поиск в ширину

- Пусть в графе G произвели поиск в ширину. Занумеруем вершины графа в соответствии с порядком, в котором поиск в ширину обходит вершины.
- Обозначим вершины графа w_i, $1 \leqslant i \leqslant n$, считая, что $num[w_i] = i$.

Поиск в ширину

- Пусть в графе G произвели поиск в ширину. Занумеруем вершины графа в соответствии с порядком, в котором поиск в ширину обходит вершины.
- Обозначим вершины графа w_i, $1 \leqslant i \leqslant n$, считая, что $num[w_i] = i$.

В леммах 1–5 изучаются свойства поиска в ширину, отличающих его от поиска в глубину.

Поиск в ширину

Лемма 1

Вершина w_k является отцом вершины w_l тогда и только тогда, когда $k = \min\{i | w_i \in list[w_l]\}$.

Поиск в ширину

Доказательство.

$\boxed{\Rightarrow}$Пусть w_k — отец вершины w_l. Это значит, что непосредственно перед удалением w_k из очереди Q, вершина w_l не была просмотрена.

Поиск в ширину

Доказательство.

$\boxed{\Rightarrow}$ Пусть w_k — отец вершины w_l. Это значит, что непосредственно перед удалением w_k из очереди Q, вершина w_l не была просмотрена.

$\boxed{\textbf{о/п}}$ Если вершина w_p смежна с w_l в графе G и $p < k$, то w_p была удалена из очереди раньше, чем w_k. Поэтому отцом w_l оказалась бы вершина w_p. $\boxed{\text{Противоречие}}$ $\square$

Поиск в ширину

$\boxed{\Leftarrow}$Пусть k — наименьший из номеров вершин w_i, смежных с w_l в графе G. Ясно, что при $p < k$ вершина w_p не смежна с вершиной w_l и поэтому не может быть ее отцом. Отсюда следует, что w_k – отец w_l. □ ■

Поиск в ширину

$\mathcal{NB}$

Из леммы 1 следует, что ребро, не являющееся древесным, никогда не соединяет предка с потомком в b–дереве. Поэтому такие ребра графа G будем называть *поперечными*.

Поиск в ширину

Лемма 2

Пусть вершины w_k и w_l являются отцами вершин w_p и w_q соответственно. Если $p \leqslant q$, то $k \leqslant l$.

Поиск в ширину

Доказательство.

о/п Предположим, что $l < k$. Из этого неравенства следует, что вершина w_l будет использована раньше, чем w_k. Поэтому вершина w_q, являющаяся сыном w_l, попадет в очередь раньше, чем w_p — сын вершины w_k. Отсюда $q < p$. Противоречие ■

Поиск в ширину

Обозначим через $h(u)$ уровень вершины u в дереве, равный расстоянию этой вершины от корня дерева.

Поиск в ширину

Лемма 3

Если $1 \leqslant p \leqslant q \leqslant n$, *то* $h(w_p) \leqslant h(w_q)$.

Поиск в ширину

Доказательство. Требуемое неравенство очевидно, если w_p — корень b–дерева.

Поиск в ширину

Доказательство. Требуемое неравенство очевидно, если w_p — корень b–дерева.
Пусть w_p не является корнем. Обозначим через s наибольший из номеров p и q и применим индукцию по s.

Поиск в ширину

Доказательство. Требуемое неравенство очевидно, если w_p — корень b–дерева.
Пусть w_p не является корнем. Обозначим через s наибольший из номеров p и q и применим индукцию по s.
База индукции при $s = 2$, очевидно, выполняется.

Поиск в ширину

Доказательство. Требуемое неравенство очевидно, если w_p — корень b–дерева.
Пусть w_p не является корнем. Обозначим через s наибольший из номеров p и q и применим индукцию по s.
База индукции при $s = 2$, очевидно, выполняется.
Рассмотрим вершины w_k и w_l, являющиеся отцами вершин w_p и w_q соответственно.

Поиск в ширину

Доказательство. Требуемое неравенство очевидно, если w_p — корень b–дерева.
Пусть w_p не является корнем. Обозначим через s наибольший из номеров p и q и применим индукцию по s.
База индукции при $s = 2$, очевидно, выполняется.
Рассмотрим вершины w_k и w_l, являющиеся отцами вершин w_p и w_q соответственно. В силу леммы 2: $k \leqslant l$.

Поиск в ширину

Доказательство. Требуемое неравенство очевидно, если w_p — корень b–дерева.
Пусть w_p не является корнем. Обозначим через s наибольший из номеров p и q и применим индукцию по s.
База индукции при $s = 2$, очевидно, выполняется.
Рассмотрим вершины w_k и w_l, являющиеся отцами вершин w_p и w_q соответственно. В силу леммы 2: $k \leqslant l$. Ясно, что к вершинам w_k и w_l применимо предположение индукции. Следовательно, $h(w_k) \leqslant h(w_l)$.

Поиск в ширину

Доказательство. Требуемое неравенство очевидно, если w_p — корень b–дерева.
Пусть w_p не является корнем. Обозначим через s наибольший из номеров p и q и применим индукцию по s.
База индукции при $s = 2$, очевидно, выполняется.
Рассмотрим вершины w_k и w_l, являющиеся отцами вершин w_p и w_q соответственно. В силу леммы 2: $k \leqslant l$. Ясно, что к вершинам w_k и w_l применимо предположение индукции. Следовательно, $h(w_k) \leqslant h(w_l)$.Отсюда

$$h(w_p) = h(w_k) + 1 \leqslant h(w_l) + 1 = h(w_q).$$

Лемма доказана. ■

Поиск в ширину

Лемма 4

Если вершины w_p и w_q смежны в графе G и $p < q$, то

$$h(w_q) - h(w_p) \leqslant 1.$$

Поиск в ширину

Доказательство. Пусть w_l — отец вершины w_q. Тогда из леммы 1 следует, что $l \leqslant p$.

Поиск в ширину

Доказательство. Пусть w_l — отец вершины w_q. Тогда из леммы 1 следует, что $l \leqslant p$.
В силу леммы 3

$$h(w_l) \leqslant h(w_p) \leqslant h(w_q).$$

Поиск в ширину

Доказательство. Пусть w_l — отец вершины w_q. Тогда из леммы 1 следует, что $l \leqslant p$.
В силу леммы 3

$$h(w_l) \leqslant h(w_p) \leqslant h(w_q).$$

Поскольку $h(w_q) - h(w_l) = 1$, получаем, что

$$h(w_q) - h(w_p) \leqslant h(w_q) - h(w_l) = 1,$$

что и требовалось доказать. ■

Поиск в ширину

Лемма 5

Расстояние в графе G от вершины w_1 (т.е. от корня b–дерева) до произвольной вершины u равняется $h(u)$.

Поиск в ширину

Доказательство. Достаточно проверить, что для произвольной (w_1, u)–цепи

$$w_1 = v_0, v_1, \ldots, v_{s-1}, v_s = u$$

выполнено неравенство $s \geqslant h(u)$.

Поиск в ширину

Доказательство. Достаточно проверить, что для произвольной (w_1, u)–цепи

$$w_1 = v_0, v_1, \ldots, v_{s-1}, v_s = u$$

выполнено неравенство $s \geqslant h(u)$.

Поиск в ширину

Рассмотрим последовательность

$$0 = h(v_0), h(v_1), \ldots, h(v_{s-1}), h(v_s) = h(u), \qquad (1)$$

составленную из уровней вершин данной цепи. В силу леммы 4 соседние элементы этой последовательности различаются не больше, чем на единицу.

Поиск в ширину

Рассмотрим последовательность

$$0 = h(v_0), h(v_1), \ldots, h(v_{s-1}), h(v_s) = h(u), \qquad (1)$$

составленную из уровней вершин данной цепи. В силу леммы 4 соседние элементы этой последовательности различаются не больше, чем на единицу.
Отсюда вытекает, что последовательность (1) имеет наименьшую длину, если она является возрастающей. В этом случае последовательность должна иметь вид $0, 1, \ldots, h(u)$.

Поиск в ширину

Рассмотрим последовательность

$$0 = h(v_0), h(v_1), \ldots, h(v_{s-1}), h(v_s) = h(u), \qquad (1)$$

составленную из уровней вершин данной цепи. В силу леммы 4 соседние элементы этой последовательности различаются не больше, чем на единицу.

Отсюда вытекает, что последовательность (1) имеет наименьшую длину, если она является возрастающей. В этом случае последовательность должна иметь вид $0, 1, \ldots, h(u)$.

Значит, для произвольной последовательности (1) выполнено неравенство $s \geqslant h(u)$. ■

Поиск в ширину

Пусть v_0 — корень b–дерева. Лемма 5 показывает, что простая (v_0, u)–цепь в b–дереве является кратчайшей v_0, u–цепью в графе G.

Поиск в ширину

Пусть v_0 — корень b–дерева. Лемма 5 показывает, что простая (v_0, u)–цепь в b–дереве является кратчайшей v_0, u–цепью в графе G.Отсюда следует

NB

Поиск в ширину может быть применен для решения следующей задачи: *В связном графе G найти кратчайшую цепь, соединяющую данную вершину v_0 с произвольной вершиной u.*

Поиск в ширину

Пусть v_0 — корень b–дерева. Лемма 5 показывает, что простая (v_0, u)–цепь в b–дереве является кратчайшей v_0, u–цепью в графе G.Отсюда следует

$\mathcal{NB}$

Поиск в ширину может быть применен для решения следующей задачи: *В связном графе G найти кратчайшую цепь, соединяющую данную вершину v_0 с произвольной вершиной u.*

Для решения этой задачи необходимо в графе G из вершины v_0 произвести поиск в ширину, а затем, используя массив *father*, построить требуемую кратчайшую цепь.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

Замкнутая цепь в графе G называется *эйлеровой*, если она содержит все ребра и все вершины графа.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

Замкнутая цепь в графе G называется *эйлеровой*, если она содержит все ребра и все вершины графа.
Связный неодноэлементный граф *эйлеров* тогда и только тогда, когда каждая его вершина имеет четную степень.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

Этот раздел посвящен построению и анализу алгоритма, позволяющего в обыкновенном связном графе G с четными степенями вершин построить эйлерову цепь.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

Этот раздел посвящен построению и анализу алгоритма, позволяющего в обыкновенном связном графе G с четными степенями вершин построить эйлерову цепь.
В алгоритме используются два стека: *SWork* и *SRes*. Элементы обоих стеков — вершины графа G. Также вводится массив $listW$, элементы которого — списки вершин.
Мы считаем, что $\forall v \in V$ начальное значение $listW[v]$ совпадает со списком смежных с v вершин, т.е. с $list[v]$.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

Алгоритм построения эйлеровой цепи

Вход: связный граф $G = (V, E)$ без вершин нечетной степени, начальная вершина v_0.

Выход: эйлерова цепь, представленная последовательностью вершин в стеке *SRes*.

1. **begin**
2. $SWork := nil$; $SRes := nil$;
3. $SWork \Leftarrow v_0$;
4. **for** $v \in V$ **do** $listW[v] := list[v]$;

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

Алгоритм построения эйлеровой цепи: продолжение

```
5.    while SWork ≠ nil do
6.      begin
7.        v := top(SWork);
8.        if listW[v] ≠ ∅ then
9.          begin
10.           u := первая вершина listW[v];
11.           SWork ⇐ u;
12.           listW[v] := listW[v]\{u};
13.           listW[u] := listW[u]\{v};
14.         end
15.       else
16.         begin v ⇐ SWork;    SRes ⇐ v; end end end.
```

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

Принцип работы алгоритма состоит в следующем.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

Принцип работы алгоритма состоит в следующем.

- Алгоритм начинает работу с некоторой вершины v_0, продвигается по ребрам графа, причем каждое ребро графа удаляется (строки 10–13).

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

Принцип работы алгоритма состоит в следующем.

- Алгоритм начинает работу с некоторой вершины v_0, продвигается по ребрам графа, причем каждое ребро графа удаляется (строки 10–13). Понятно, что последовательное выполнение этой группы операторов позволяет выделить в графе некоторую замкнутую цепь.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

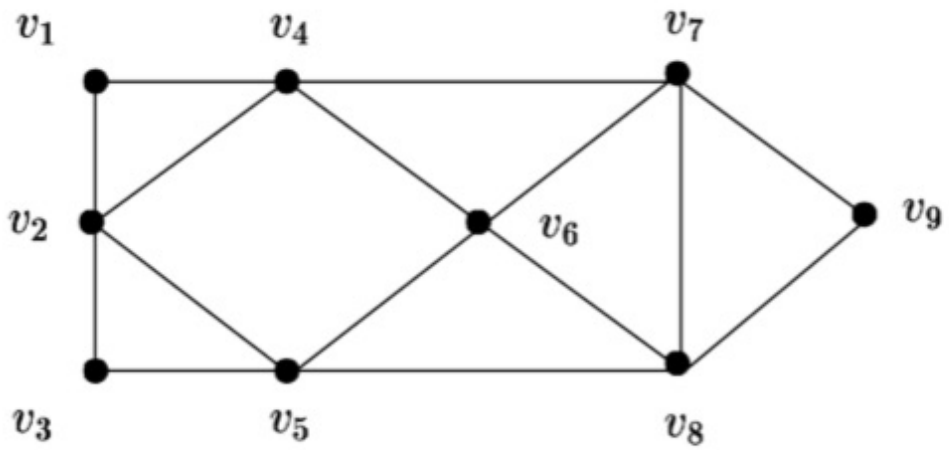
Принцип работы алгоритма состоит в следующем.

- Алгоритм начинает работу с некоторой вершины v_0, продвигается по ребрам графа, причем каждое ребро графа удаляется (строки 10–13). Понятно, что последовательное выполнение этой группы операторов позволяет выделить в графе некоторую замкнутую цепь.
- Затем начинается выполнение группы операторов в строке 16. Эти операторы выталкивают очередную вершину из стека *SWork* в стек *SRes*, пока не выполнится одно из условий:
  - стек *SWork* пуст (конец работы алгоритма)
  - для вершины $v = top(SWork)$ существует непройденное ребро vu. В это случае алгоритм продолжает работу из вершины u.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

На рисунке изображен эйлеров граф и эйлерова цепь, построенная нашим алгоритмом (предполагается, что все вершины упорядочены по возрастанию номеров).



$$v_1,\ v_2,\ v_3,\ v_5,\ v_2,\ v_4,\ v_6,\ v_5,\ v_8,\ v_6,\ v_7,\ v_8,\ v_9,\ v_7,\ v_4,\ v_1$$

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

Теорема

Алгоритм правильно строит эйлерову цепь в эйлеровом графе G.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

Доказательство.

- Заметим сначала, что из стека *SRes* вершины никогда не выталкиваются. Отсюда следует, что начиная с некоторого момента работы алгоритма стек *SRes* перестает изменяться. Это произойдет после выполнения операторов в строке 16.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

Доказательство.

- Заметим сначала, что из стека *SRes* вершины никогда не выталкиваются. Отсюда следует, что начиная с некоторого момента работы алгоритма стек *SRes* перестает изменяться. Это произойдет после выполнения операторов в строке 16.
- **о/п** Предположим, что стек *SWork* в этот момент не пуст. Если $v = top(SWork)$, операторы в строках 10–13 начнут добавлять вершины к стеку *SWork* (т.е. в графе G можно из вершины v построить цепь, состоящую из еще непройденных ребер).

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

- Ясно, что построение такой цепи должно прекратиться. Это означает, что мы придем в вершину u, для которой все инцидентные ей ребра уже пройдены ($listW[u] = \varnothing$). После этого начнут выполняться операторы строки 16, и, следовательно, к стеку $SRes$ добавиться хотя бы одна вершина. Противоречие

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

- Ясно, что построение такой цепи должно прекратиться. Это означает, что мы придем в вершину u, для которой все инцидентные ей ребра уже пройдены ($listW[u] = \varnothing$). После этого начнут выполняться операторы строки 16, и, следовательно, к стеку *SRes* добавиться хотя бы одна вершина. Противоречие
- Таким образом, рано или поздно стек *SWork* станет пустым, что приведет к завершению работы алгоритма.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

- Пусть P — цепь, содержащаяся в стеке $SRes$ после окончания работы алгоритма. Легко понять, что вершина w помещается в стек $SRes$, если все ребра, инцидентные w, уже пройдены.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

- Пусть P — цепь, содержащаяся в стеке *SRes* после окончания работы алгоритма. Легко понять, что вершина w помещается в стек *SRes*, если все ребра, инцидентные w, уже пройдены.
- Отсюда следует, что для любой вершины цепи P все ребра, инцидентные этой вершине, содержатся в цепи P.

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

- Пусть P — цепь, содержащаяся в стеке $SRes$ после окончания работы алгоритма. Легко понять, что вершина w помещается в стек $SRes$, если все ребра, инцидентные w, уже пройдены.
- Отсюда следует, что для любой вершины цепи P все ребра, инцидентные этой вершине, содержатся в цепи P.
- Поскольку G — связный граф, цепь P содержит все ребра графа G. Значит, P — эйлерова цепь. ■

Поиск в ширину

Алгоритм отыскания эйлеровой цепи в эйлеровом графе

- В заключение оценим сложность алгоритма.
- Для этого заметим, что при каждой итерации цикла либо к стеку *SWork* добавляется вершина (это означает прохождение очередного ребра), либо вершина переносится из стека *SWork* в *SRes* (другими словами к строящейся эйлеровой цепи добавляется ребро).
- Отсюда следует, что число повторений цикла равно $O(m)$.
- Если позаботиться о том, чтобы время, необходимое для удаления вершины из списка $listW[v]$, было ограничено константой, то сложность алгоритма будет равняться $O(m)$.