Структуры данных

Гаевой, Казменко, Макаров

Санкт-Петербургский Государственный Университет

Четверг, 26 ноября 2020 года

Содержание

Содержание

Задача

Задача:

- Рассмотрим граф из n вершин, изначально без рёбер.
- В него добавляются рёбра по одному.
- Мы хотим следить за компонентами связности в этом графе.

Задача

Задача:

- Рассмотрим граф из n вершин, изначально без рёбер.
- В него добавляются рёбра по одному.
- Мы хотим следить за компонентами связности в этом графе.
- Для очередного ребра:
  - Если ребро соединяет вершины из одной компоненты связности, ничего не делать.
  - Если ребро соединяет вершины из разных компонент связности, объединить их в одну.
  - Выдать, что из этого произошло.
- Возможно, иногда мы ещё хотим узнавать, в какой компоненте связности лежит вершина.
- Такая структура называется «система непересекающихся множеств» (СНМ) или «disjoint set union» (DSU).

Цвета множеств: решение

Решение:

- Пусть у каждой вершины есть цвет.
- Изначально вершина i имеет цвет i.

Цвета множеств: решение

Решение:

- Пусть у каждой вершины есть цвет.
- Изначально вершина i имеет цвет i.
- Если очередное ребро соединяет вершины одного цвета, ничего не делаем.

Цвета множеств: решение

Решение:

- Пусть у каждой вершины есть цвет.
- Изначально вершина i имеет цвет i.
- Если очередное ребро соединяет вершины одного цвета, ничего не делаем.
- Если очередное ребро соединяет вершины разных цветов, перекрашиваем все вершины одного цвета в другой.

Цвета множеств: код

```
 1  struct Dsu {
 2      vector <int> c;
 3      int n;
 4
 5      Dsu (int n_) {
 6          n = n_;
 7          c = vector <int> (n);
 8          iota (c.begin (), c.end (), 0);
 9      }
10
11      bool unite (int u, int v) {
12          if (c[u] == c[v])
13              return false;
14          for (int i = 0; i < n; i++)
15              if (c[i] == c[u])
16                  c[i] = c[v];
17          return true;
18      }
19  };
```

Цвета множеств: код

```
 1  #include <iostream>
 2  #include <numeric>
 3  #include <vector>
 4  using namespace std;
 5
 6  struct Dsu { /* ... */ };
 7
 8  int main () {
 9      int n, m;
10      cin >> n >> m;
11      Dsu dsu (n);
12      int res = n;
13      for (int j = 0; j < m; j++) {
14          int u, v;
15          cin >> u >> v;
16          u -= 1;
17          v -= 1;
18          res -= dsu.unite (u, v);
19      }
20      cout << ((res == 1) ? "YES" : "NO") << endl;
21      return 0;
22  }
```

Цвета множеств: код

```cpp
#include <iostream>
#include <numeric>
#include <vector>
using namespace std;

struct Dsu { /* ... */ };

int main () {
    int n, m;
    cin >> n >> m;
    Dsu dsu (n);
    int res = n;
    for (int j = 0; j < m; j++) {
        int u, v;
        cin >> u >> v;
        u -= 1;
        v -= 1;
        res -= dsu.unite (u, v);
    }
    cout << ((res == 1) ? "YES" : "NO") << endl;
    return 0;
}
```

Ввод:

```
3 3
1 2
1 3
2 3
```

Вывод:

NO

Это что, баг?!

Цвета множеств: код

```
1   struct Dsu {
2       vector <int> c;
3       int n;
4
5       Dsu (int n_) {
6           n = n_;
7           c = vector <int> (n);
8           iota (c.begin (), c.end (), 0);
9       }
10
11      bool unite (int u, int v) {
12          if (c[u] == c[v])
13              return false;
14          for (int i = 0; i < n; i++)
15              if (c[i] == c[u])
16                  c[i] = c[v];
17          return true;
18      }
19  };
```

Ввод:

```
3 3
1 2
1 3
2 3
```

Вывод:

NO

Это что, баг?!

Цвета множеств: код

```
 1  struct Dsu {
 2      vector <int> c;
 3      int n;
 4
 5      Dsu (int n_) {
 6          n = n_;
 7          c = vector <int> (n);
 8          iota (c.begin (), c.end (), 0);
 9      }
10
11      bool unite (int u, int v) {
12          if (c[u] == c[v])
13              return false;
14          int prev = c[u];
15          for (int i = 0; i < n; i++)
16              if (c[i] == prev)
17                  c[i] = c[v];
18          return true;
19      }
20  };
```

Ввод:

```
3 3
1 2
1 3
2 3
```

Вывод:

```
YES
```

Цвета множеств: асимптотика

Дополнительная память:

Время работы:

- Узнать цвет вершины:
- Объединить два множества:
- Общее время работы:

Цвета множеств: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины:
- Объединить два множества:
- Общее время работы:

Цвета множеств: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(1)$.
- Объединить два множества:
- Общее время работы:

Цвета множеств: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(1)$.
- Объединить два множества: $\Theta(n)$.
- Общее время работы:

Цвета множеств: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(1)$.
- Объединить два множества: $\Theta(n)$.
- Общее время работы: $O(m + n^2)$.

Цвета множеств: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(1)$.
- Объединить два множества: $\Theta(n)$.
- Общее время работы: $O(m + n^2)$.
- Можно довести идею с цветами до $O(m + n \log n)$...

Корневые деревья: решение

Решение:

- Организуем каждое множество как корневое дерево:
  - корень — *представитель* множества;
  - из любой вершины, кроме корня, выходит дуга;
  - из любой вершины по дугам можно дойти до корня.

Корневые деревья: решение

Решение:

- Организуем каждое множество как корневое дерево:
  - корень — *представитель* множества;
  - из любой вершины, кроме корня, выходит дуга;
  - из любой вершины по дугам можно дойти до корня.
  - Для единообразия: пусть из корня идёт дуга в себя.

Корневые деревья: решение

Решение:

- Организуем каждое множество как корневое дерево:
  - корень — *представитель* множества;
  - из любой вершины, кроме корня, выходит дуга;
  - из любой вершины по дугам можно дойти до корня.
  - Для единообразия: пусть из корня идёт дуга в себя.
- Изначально из каждой вершины i дуга ведёт в i.

Корневые деревья: решение

Решение:

- Организуем каждое множество как корневое дерево:
  - корень — *представитель* множества;
  - из любой вершины, кроме корня, выходит дуга;
  - из любой вершины по дугам можно дойти до корня.
  - Для единообразия: пусть из корня идёт дуга в себя.
- Изначально из каждой вершины i дуга ведёт в i.
- Чтобы выяснить, лежат ли вершины u и v в одном множестве, найдём их представителей:
  $u' := \text{root}(u)$, $v' := \text{root}(v)$.

Корневые деревья: решение

Решение:

- Организуем каждое множество как корневое дерево:
  - корень — *представитель* множества;
  - из любой вершины, кроме корня, выходит дуга;
  - из любой вершины по дугам можно дойти до корня.
  - Для единообразия: пусть из корня идёт дуга в себя.
- Изначально из каждой вершины i дуга ведёт в i.
- Чтобы выяснить, лежат ли вершины u и v в одном множестве, найдём их представителей:
  $u' := \text{root}(u)$, $v' := \text{root}(v)$.
- Если $u' = v'$, ничего не делаем.

Корневые деревья: решение

Решение:

- Организуем каждое множество как корневое дерево:
  - корень — *представитель* множества;
  - из любой вершины, кроме корня, выходит дуга;
  - из любой вершины по дугам можно дойти до корня.
  - Для единообразия: пусть из корня идёт дуга в себя.
- Изначально из каждой вершины i дуга ведёт в i.
- Чтобы выяснить, лежат ли вершины u и v в одном множестве, найдём их представителей:
  $u' := \text{root}(u)$, $v' := \text{root}(v)$.
- Если $u' = v'$, ничего не делаем.
- Если $u' \neq v'$, дугу из u' пускаем в v', или наоборот.

Корневые деревья: код

```
struct Dsu {
    vector <int> p;
    int n;

    Dsu (int n_) {
        n = n_;
        p = vector <int> (n);
        iota (p.begin (), p.end (), 0);
    }

    /* ... */
};
```

```
int root (int v) {
    if (p[v] == v)
        return v;
    return root (p[v]);
}

bool unite (int u, int v) {
    u = root (u);
    v = root (v);
    if (u == v)
        return false;
    p[u] = v;
    return true;
}
```

Наивная реализация: асимптотика

Дополнительная память:

Время работы:

- Узнать цвет вершины:
- Объединить два множества:
- Общее время работы:

Наивная реализация: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины:
- Объединить два множества:
- Общее время работы:

Наивная реализация: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(n)$.
- Объединить два множества:
- Общее время работы:

Наивная реализация: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(n)$.
- Объединить два множества: $O(n)$.
- Общее время работы:

Наивная реализация: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(n)$.
- Объединить два множества: $O(n)$.
- Общее время работы: $O(m + mn)$.

Эвристика о сжатии путей

Идея:

- Почему получилось медленно?
- В функции `root` мы проходим слишком длинные пути.

Эвристика о сжатии путей

Идея:

- Почему получилось медленно?
- В функции `root` мы проходим слишком длинные пути.
- Давайте, пройдя по пути, при возвращении переткнём все дуги сразу в корень.
- Теперь изо всех вершин бывшего пути можно дойти до корня за один шаг.

Эвристика о сжатии путей: код

```
struct Dsu {
    vector <int> p;
    int n;

    Dsu (int n_) {
        n = n_;
        p = vector <int> (n);
        iota (p.begin (), p.end (), 0);
    }

    /* ... */
};
```

```
int root (int v) {
    if (p[v] == v)
        return v;
    return root (p[v]);
}

bool unite (int u, int v) {
    u = root (u);
    v = root (v);
    if (u == v)
        return false;
    p[u] = v;
    return true;
}
```

Эвристика о сжатии путей: код

```
struct Dsu {
    vector <int> p;
    int n;

    Dsu (int n_) {
        n = n_;
        p = vector <int> (n);
        iota (p.begin (), p.end (), 0);
    }

    /* ... */
};
```

```
int root (int v) {
    if (p[v] == v)
        return v;
    return p[v] = root (p[v]);
}

bool unite (int u, int v) {
    u = root (u);
    v = root (v);
    if (u == v)
        return false;
    p[u] = v;
    return true;
}
```

Эвристика о сжатии путей: асимптотика

Дополнительная память:

Время работы:

- Узнать цвет вершины:
- Объединить два множества:
- Общее время работы:

Эвристика о сжатии путей: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины:
- Объединить два множества:
- Общее время работы:

Эвристика о сжатии путей: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(n)$.
- Объединить два множества:
- Общее время работы:

Эвристика о сжатии путей: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(n)$.
- Объединить два множества: $O(n)$.
- Общее время работы:

Эвристика о сжатии путей: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(n)$.
- Объединить два множества: $O(n)$.
- Общее время работы: $O(m + n \log n)$.

Эвристика о сжатии путей: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(n)$.
- Объединить два множества: $O(n)$.
- Общее время работы: $O(m + n \log n)$.
- Идея доказательства:
  - Проходя по пути, по всем дугам $u \to p(u)$, кроме последней, посмотрим на размеры поддеревьев: $s(u)$ и $s(p(u))$.
  - Если на каком-то шаге размер увеличился хотя бы вдвое, это могло произойти не более $\log_2 n$ раз.
  - В противном случае, когда мы переткнули дугу $u \to root$, размер $s(p(u))$ уменьшился хотя бы вдвое.
  - Поскольку $p(u)$ уже не корень, этот размер может только уменьшаться, а значит, для каждого $p(u)$ это также могло произойти не более $\log_2 n$ раз.

Эвристика о рангах

Идея:

- Почему получилось медленно?
- В функции `root` мы проходим слишком длинные пути.

Эвристика о рангах

Идея:

- Почему получилось медленно?
- В функции `root` мы проходим слишком длинные пути.
- Давайте для каждого множества помнить его *ранг*: максимальную длину пути в нём.
- При слиянии двух множеств будем привешивать множество меньшего ранга ко множеству большего ранга.

Эвристика о рангах

Идея:

- Почему получилось медленно?
- В функции `root` мы проходим слишком длинные пути.
- Давайте для каждого множества помнить его *ранг*: максимальную длину пути в нём.
- При слиянии двух множеств будем привешивать множество меньшего ранга ко множеству большего ранга.
- Если ранги были различны, итоговый ранг равен большему.
- Если ранги были равны, итоговый ранг увеличился на единицу.

Эвристика о рангах: код

```
struct Dsu {
    vector <int> p;
    int n;

    Dsu (int n_) {
        n = n_;
        p = vector <int> (n);
        iota (p.begin (), p.end (), 0);
    }

    /* ... */
};
```

```
int root (int v) {
    if (p[v] == v)
        return v;
    return root (p[v]);
}

bool unite (int u, int v) {
    u = root (u);
    v = root (v);
    if (u == v)
        return false;
    p[u] = v;
    return true;
}
```

Эвристика о рангах: код

```
struct Dsu {
    vector <int> p;
    vector <int> r;
    int n;

    Dsu (int n_) {
        n = n_;
        p = vector <int> (n);
        iota (p.begin (), p.end (), 0);
        r = vector <int> (n, 0);
    }

    /* ... */
};
```

```
int root (int v) {
    if (p[v] == v)
        return v;
    return root (p[v]);
}

bool unite (int u, int v) {
    u = root (u);
    v = root (v);
    if (u == v)
        return false;
    if (r[u] > r[v])
        swap (u, v);
    p[u] = v;
    if (r[u] == r[v])
        r[v] += 1;
    return true;
}
```

Эвристика о рангах: асимптотика

Дополнительная память:

Время работы:

- Узнать цвет вершины:
- Объединить два множества:
- Общее время работы:

Эвристика о рангах: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины:
- Объединить два множества:
- Общее время работы:

Эвристика о рангах: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества:
- Общее время работы:

Эвристика о рангах: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества: $O(\log n)$.
- Общее время работы:

Эвристика о рангах: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества: $O(\log n)$.
- Общее время работы: $O((m+n)\log n)$.

Эвристика о рангах: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества: $O(\log n)$.
- Общее время работы: $O((m+n)\log n)$.
- Идея доказательства:
  - Время работы `root` равно рангу множества.
  - Чтобы получить множество ранга k, нужно слить два множества ранга $k-1$.
  - Значит, во множестве ранга k не меньше 2^k элементов.

Эвристика о размерах

Идея:

- Почему получилось медленно?
- В функции `root` мы проходим слишком длинные пути.

Эвристика о размерах

Идея:

- Почему получилось медленно?
- В функции `root` мы проходим слишком длинные пути.
- Давайте для каждого множества помнить его *размер*: количество вершин в нём.
- При слиянии двух множеств будем привешивать множество меньшего размера ко множеству большего размера.
- Размер итогового множества равен сумме старых размеров.

Эвристика о размерах: код

```
struct Dsu {
    vector <int> p;
    int n;

    Dsu (int n_) {
        n = n_;
        p = vector <int> (n);
        iota (p.begin (), p.end (), 0);
    }

    /* ... */
};
```

```
int root (int v) {
    if (p[v] == v)
        return v;
    return root (p[v]);
}

bool unite (int u, int v) {
    u = root (u);
    v = root (v);
    if (u == v)
        return false;
    p[u] = v;
    return true;
}
```

Эвристика о размерах: код

```
struct Dsu {
    vector <int> p;
    vector <int> s;
    int n;

    Dsu (int n_) {
        n = n_;
        p = vector <int> (n);
        iota (p.begin (), p.end (), 0);
        s = vector <int> (n, 1);
    }

    /* ... */
};
```

```
int root (int v) {
    if (p[v] == v)
        return v;
    return root (p[v]);
}

bool unite (int u, int v) {
    u = root (u);
    v = root (v);
    if (u == v)
        return false;
    if (s[u] > s[v])
        swap (u, v);
    p[u] = v;
    s[v] += s[u];
    return true;
}
```

Эвристика о размерах: асимптотика

Дополнительная память:

Время работы:

- Узнать цвет вершины:
- Объединить два множества:
- Общее время работы:

Эвристика о размерах: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины:
- Объединить два множества:
- Общее время работы:

Эвристика о размерах: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества:
- Общее время работы:

Эвристика о размерах: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества: $O(\log n)$.
- Общее время работы:

Эвристика о размерах: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества: $O(\log n)$.
- Общее время работы: $O((m+n)\log n)$.

Эвристика о размерах: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества: $O(\log n)$.
- Общее время работы: $O((m+n)\log n)$.
- Идея доказательства:
  - Рассмотрим вершину v. Будем следить за длиной пути от v до корня.
  - Пока представитель v не меняется, длина пути до корня не меняется.
  - Каждый раз, когда представитель v меняется, длина пути до корня увеличивается на 1.
  - Но, если представитель поменялся — значит, множество стало хотя бы в два раза больше!
  - Поэтому это могло произойти не более $\log_2 n$ раз.

Две эвристики

Идея:

- Можно применить одновременно эвристику в `root` и любую эвристику в `unite`.

Две эвристики: код

```
struct Dsu {
    vector <int> p;
    int n;

    Dsu (int n_) {
        n = n_;
        p = vector <int> (n);
        iota (p.begin (), p.end (), 0);
    }

    /* ... */
};
```

```
int root (int v) {
    if (p[v] == v)
        return v;
    return root (p[v]);
}

bool unite (int u, int v) {
    u = root (u);
    v = root (v);
    if (u == v)
        return false;
    p[u] = v;
    return true;
}
```

Две эвристики: код

```
struct Dsu {
    vector <int> p;
    vector <int> r;
    int n;

    Dsu (int n_) {
        n = n_;
        p = vector <int> (n);
        iota (p.begin (), p.end (), 0);
        r = vector <int> (n, 0);
    }

    /* ... */
};
```

```
int root (int v) {
    if (p[v] == v)
        return v;
    return p[v] = root (p[v]);
}

bool unite (int u, int v) {
    u = root (u);
    v = root (v);
    if (u == v)
        return false;
    if (r[u] > r[v])
        swap (u, v);
    p[u] = v;
    if (r[u] == r[v])
        r[v] += 1;
    return true;
}
```

Две эвристики: асимптотика

Дополнительная память:

Время работы:

- Узнать цвет вершины:
- Объединить два множества:
- Общее время работы:

Две эвристики: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины:
- Объединить два множества:
- Общее время работы:

Две эвристики: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества:
- Общее время работы:

Две эвристики: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества: $O(\log n)$.
- Общее время работы:

Две эвристики: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества: $O(\log n)$.
- Общее время работы: $O((m+n)\log n)$.

Две эвристики: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества: $O(\log n)$.
- Общее время работы: $O((m+n)\log n)$.
- Оценка лучше: $O((m+n)\log^* n)$.

Две эвристики: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества: $O(\log n)$.
- Общее время работы: $O((m+n)\log n)$.
- Оценка лучше: $O((m+n)\log^* n)$.
- Оценка ещё лучше: $O((m+n)\alpha^{-1}(m+n))$.

Две эвристики: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества: $O(\log n)$.
- Общее время работы: $O((m+n)\log n)$.
- Оценка лучше: $O((m+n)\log^* n)$.
- Оценка ещё лучше: $O((m+n)\alpha^{-1}(m+n))$.
- Доказательство: Robert Endre Tarjan, «Efficiency of a Good But Not Linear Set Union Algorithm», 1975.

Две эвристики: асимптотика

Дополнительная память: $O(n)$.
Время работы:

- Узнать цвет вершины: $O(\log n)$.
- Объединить два множества: $O(\log n)$.
- Общее время работы: $O((m+n)\log n)$.
- Оценка лучше: $O((m+n)\log^* n)$.
- Оценка ещё лучше: $O((m+n)\alpha^{-1}(m+n))$.
- Доказательство: Robert Endre Tarjan, «Efficiency of a Good But Not Linear Set Union Algorithm», 1975.
- Такие оценки доказываются для разных пар эвристики в `root` и эвристики в `unite`.

Содержание

Задача

Задача:

- Дана последовательность чисел $a_1, a_2, \ldots, a_n$.
- Мы хотим узнавать суммы на отрезках с lo-го по hi-й элемент.

Задача

Задача:

- Дана последовательность чисел $a_1, a_2, \ldots, a_n$.
- Мы хотим узнавать суммы на отрезках с lo-го по hi-й элемент.

Наивное решение:

- Когда спрашивают сумму, считаем её циклом.
- Время: $O(hi - lo)$.

Задача

Задача:

- Дана последовательность чисел $a_1, a_2, \ldots, a_n$.
- Мы хотим узнавать суммы на отрезках с lo-го по hi-й элемент.

Наивное решение:

- Когда спрашивают сумму, считаем её циклом.
- Время: $O(hi - lo)$.

Более эффективное решение:

- Посчитаем суммы на всех префиксах: $s(k) = a_1 + a_2 + \ldots + a_k$.
- Быстро: $s(i) = s(i-1) + a_i$.
- Теперь $a_{lo} + \ldots + a_{hi} = s(hi) - s(lo-1)$.
- Время: $O(n)$ на подготовку и $O(1)$ на запрос.

Задача посложнее

Задача:

- Дана последовательность чисел $a_1, a_2, \ldots, a_n$.
- Мы хотим узнавать суммы на отрезках с lo-го по hi-й элемент.
- И хотим делать $a_i \mathrel{+}= x$ для любых i и x.

Задача посложнее

Задача:

- Дана последовательность чисел $a_1, a_2, \ldots, a_n$.
- Мы хотим узнавать суммы на отрезках с lo-го по hi-й элемент.
- И хотим делать $a_i \mathrel{+}= x$ для любых i и x.

Наивное решение:

- Когда нужно прибавление, просто делаем его.
- Когда спрашивают сумму, считаем её циклом.
- Время: $O(1)$ на изменение, $O(hi - lo)$ на запрос.

Задача посложнее

Задача:

- Дана последовательность чисел $a_1, a_2, \ldots, a_n$.
- Мы хотим узнавать суммы на отрезках с lo-го по hi-й элемент.
- И хотим делать $a_i \mathrel{+}= x$ для любых i и x.

Решение с префиксными суммами:

- Посчитаем суммы на всех префиксах: $s(0) = 0$, $s(i) = s(i-1) + a_i$.
- Теперь $a_{lo} + \ldots + a_{hi} = s(hi) - s(lo-1)$.
- А при прибавлении $a_i \mathrel{+}= x$ нужно пересчитать $s(i)$, $s(i+1)$, …, $s(n)$.
- Время: $O(n)$ на подготовку, $O(n)$ на изменение и $O(1)$ на запрос.

Простые решения: код

```
 1  int main () {
 2      int n, m;
 3      cin >> n >> m;
 4      vector <int> a (n);
 5      for (int i = 0; i < n; i++)
 6          cin >> a[i];
 7      Array z (a, n);
 8      for (int j = 0; j < m; j++) {
 9          string type;
10          cin >> type;
11          if (type == "add") {
12              int i, x;
13              cin >> i >> x;
14              z.add (i - 1, x);
15          }
16          if (type == "sum") {
17              int lo, hi;
18              cin >> lo >> hi;
19              cout << z.sum (lo - 1, hi) << endl;
20          }
21      }
22      return 0;
23  }
```

Ввод:

```
5 6
1 2 3 4 5
sum 1 5
add 1 4
sum 2 4
add 3 6
sum 1 3
sum 4 5
```

Вывод:

```
15
9
16
9
```

Простые решения: код

```
1   struct Array {
2       vector <int> a;
3       int n;
4
5       Array (vector <int> & a_, int n_) {
6           a = a_;
7           n = n_;
8       }
9
10      void add (int pos, int val) {
11          a[pos] += val;
12      }
13
14      int sum (int lo, int hi) {
15          int res = 0;
16          for (int i = lo; i < hi; i++)
17              res += a[i];
18          return res;
19      }
20  };
```

Ввод:

```
5 6
1 2 3 4 5
sum 1 5
add 1 4
sum 2 4
add 3 6
sum 1 3
sum 4 5
```

Вывод:

```
15
9
16
9
```

Простые решения: код

```
1   struct Array {
2       vector <int> a;
3       vector <int> s;
4       int n;
5
6       Array (vector <int> & a_, int n_) {
7           a = a_;
8           n = n_;
9           s = vector <int> (n + 1);
10          for (int i = 0; i < n; i++)
11              s[i + 1] = s[i] + a[i];
12      }
13
14      void add (int pos, int val) {
15          a[pos] += val;
16          for (int i = pos; i < n; i++)
17              s[i + 1] = s[i] + a[i];
18      }
19
20      int sum (int lo, int hi) {
21          return s[hi] - s[lo];
22      }
23  };
```

Ввод:

```
5 6
1 2 3 4 5
sum 1 5
add 1 4
sum 2 4
add 3 6
sum 1 3
sum 4 5
```

Вывод:

```
15
9
16
9
```

Корневая эвристика

Идея:

- Разобьём последовательность длины n на $\sqrt{n}$ отрезков длины $\sqrt{n}$.
- Например, последовательность длины 10 000 на 100 отрезков длины 100.
- На каждом отрезке k будем хранить сумму b_k.

Корневая эвристика

Идея:

- Разобьём последовательность длины n на $\sqrt{n}$ отрезков длины $\sqrt{n}$.
- Например, последовательность длины 10 000 на 100 отрезков длины 100.
- На каждом отрезке k будем хранить сумму b_k.
- При прибавлении $a_i \mathrel{+}= x$:
  - прибавим к элементу: $a_i \mathrel{+}= x$;
  - прибавим к сумме: $b_{\frac{i}{\sqrt{n}}} \mathrel{+}= x$.
  - Время: $O(1)$.

Корневая эвристика

Идея:

- Разобьём последовательность длины n на $\sqrt{n}$ отрезков длины $\sqrt{n}$.
- Например, последовательность длины 10 000 на 100 отрезков длины 100.
- На каждом отрезке k будем хранить сумму b_k.
- При суммировании идём слева направо:
  - прибавляем элементы по одному, пока не дойдём до границы отрезка;
  - прибавляем целые отрезки, пока это возможно;
  - прибавляем оставшиеся элементы по одному.
  - Например, чтобы найти сумму элементов от $lo = 498$ до $hi = 701$, просуммируем $a_{498} + a_{499} + a_{500} + b_6 + b_7 + a_{701}$.
  - Время: $O(\sqrt{n})$.

Двухуровневое разбиение

Идея:

- Разобьём последовательность длины n на $\sqrt[3]{n}$ отрезков длины $(\sqrt[3]{n})^2$.
- А каждый из них, в свою очередь, на $\sqrt[3]{n}$ отрезков длины $\sqrt[3]{n}$.
- Например, последовательность длины 1000 разбиваем на 10 больших отрезков длины 100, а каждый большой отрезок — на 10 маленьких отрезков длины 10.
- На каждом большом и маленьком отрезке храним сумму.

Двухуровневое разбиение

Идея:

- Разобьём последовательность длины n на $\sqrt[3]{n}$ отрезков длины $(\sqrt[3]{n})^2$.
- А каждый из них, в свою очередь, на $\sqrt[3]{n}$ отрезков длины $\sqrt[3]{n}$.
- Например, последовательность длины 1000 разбиваем на 10 больших отрезков длины 100, а каждый большой отрезок — на 10 маленьких отрезков длины 10.
- На каждом большом и маленьком отрезке храним сумму.
- При прибавлении поменяются три числа:
  - сам элемент,
  - сумма на маленьком отрезке,
  - сумма на большом отрезке.
  - Время: $O(1)$.

Двухуровневое разбиение

Идея:

- Разобьём последовательность длины n на $\sqrt[3]{n}$ отрезков длины $(\sqrt[3]{n})^2$.
- А каждый из них, в свою очередь, на $\sqrt[3]{n}$ отрезков длины $\sqrt[3]{n}$.
- Например, последовательность длины 1000 разбиваем на 10 больших отрезков длины 100, а каждый большой отрезок — на 10 маленьких отрезков длины 10.
- На каждом большом и маленьком отрезке храним сумму.
- При суммировании сложим не более $5 \cdot \sqrt[3]{n}$ чисел.
- Время: $O(\sqrt[3]{n})$.

Дерево отрезков

Что получается в пределе? Двоичное дерево.

- Каждый отрезок разбивается на два отрезка поменьше.
- На самом нижнем уровне — отдельные элементы последовательности.
- Удобно, когда длина отрезка — степень двойки.
- Если нет — можно дополнить нулями до ближайшей.
- На каждом отрезке храним сумму.

Дерево отрезков

Что получается в пределе? Двоичное дерево.

- Каждый отрезок разбивается на два отрезка поменьше.
- На самом нижнем уровне — отдельные элементы последовательности.
- Удобно, когда длина отрезка — степень двойки.
- Если нет — можно дополнить нулями до ближайшей.
- На каждом отрезке храним сумму.
- При прибавлении поменяются $\log_2 n$ чисел: сам элемент и суммы на всех отрезках, содержащих его.

Дерево отрезков

Что получается в пределе? Двоичное дерево.

- Каждый отрезок разбивается на два отрезка поменьше.
- На самом нижнем уровне — отдельные элементы последовательности.
- Удобно, когда длина отрезка — степень двойки.
- Если нет — можно дополнить нулями до ближайшей.
- На каждом отрезке храним сумму.
- При прибавлении поменяются $\log_2 n$ чисел: сам элемент и суммы на всех отрезках, содержащих его.
- При суммировании с каждой стороны — слева и справа — на каждом уровне прибавим не более одного числа. Время: $O(\log_2 n)$.

Дерево отрезков: реализация

Как хранить такое дерево?
Так же, как двоичную кучу.

- Пронумеруем вершины дерева.
- Корень дерева имеет в массиве номер 1 и хранит сумму всей исходной последовательности.
- Вершина с номером k — сумма вершин с номерами $2k$ и $2k + 1$.
- Будем хранить дерево в массиве, индексируя его номерами вершин.

Дерево отрезков: код

```
struct SegmentTree {
    vector <int> t;
    int half;

    SegmentTree (vector <int> & a, int n) {
        half = 1;
        while (half < n) half *= 2;
        t = vector <int> (half * 2);
        for (int i = 0; i < n; i++)
            t[i + half] = a[i];
        for (int i = half - 1; i >= 1; i--)
            t[i] = t[i * 2] + t[i * 2 + 1];
    }

    /* ... */
};
```

```
void add (int pos, int val) {
    for (pos += half; pos > 0;
         pos /= 2)
        t[pos] += val;
}

int sum (int lo, int hi) {
    int res = 0;
    for (lo += half, hi += half;
         lo < hi;
         lo /= 2, hi /= 2) {
        if (lo & 1)
            res += t[lo++];
        if (hi & 1)
            res += t[--hi];
    }
    return res;
}
```

Смежная задача

Задача:

- Дана последовательность чисел $a_1, a_2, \ldots, a_n$.
- Мы хотим одновременно прибавлять число x ко всем элементам с lo-го по hi-й.
- И хотим узнавать, чему равно a_i.

Смежная задача

Задача:

- Дана последовательность чисел $a_1, a_2, \ldots, a_n$.
- Мы хотим одновременно прибавлять число x ко всем элементам с lo-го по hi-й.
- И хотим узнавать, чему равно a_i.

Решение: заведём похожее дерево.

- Но нужно быстро прибавлять x на целом отрезке!
- Мы уже умеем разбивать отрезок любой длины на $O(\log n)$ отрезков, хранящихся в дереве.
- Давайте в вершины дерева, соответствующие этим отрезкам, добавлять x.
- Получится, что значение в a_i — это сумма значений в дереве на пути от a_i до корня.

Смежная задача: код

```
 1  int main () {
 2      int n, m;
 3      cin >> n >> m;
 4      vector <int> a (n);
 5      for (int i = 0; i < n; i++)
 6          cin >> a[i];
 7      AddTree z (a, n);
 8      for (int j = 0; j < m; j++) {
 9          string type;
10          cin >> type;
11          if (type == "add") {
12              int lo, hi, x;
13              cin >> lo >> hi >> x;
14              z.add (lo - 1, hi, x);
15          }
16          if (type == "ask") {
17              int i;
18              cin >> i;
19              cout << z.ask (i - 1) << endl;
20          }
21      }
22      return 0;
23  }
```

Ввод:

```
5 6
1 2 3 4 5
ask 3
add 1 5 1
ask 2
add 2 4 2
ask 3
ask 5
```

Вывод:

```
3
3
6
6
```

Смежная задача: код

```
struct AddTree {
    vector <int> t;
    int half;

    AddTree (vector <int> & a, int n) {
        half = 1;
        while (half < n) half *= 2;
        t = vector <int> (half * 2);
        for (int i = 0; i < n; i++)
            t[i + half] = a[i];
    }

    /* ... */
};
```

```
int ask (int pos) {
    int res = 0;
    for (pos += half; pos > 0;
         pos /= 2)
        res += t[pos];
    return res;
}

void add (int lo, int hi, int x) {
    for (lo += half, hi += half;
         lo < hi;
         lo /= 2, hi /= 2) {
        if (lo & 1)
            t[lo++] += x;
        if (hi & 1)
            t[--hi] += x;
    }
}
```

Дерево отрезков: повтор для сравнения

```
struct SegmentTree {
    vector <int> t;
    int half;

    SegmentTree (vector <int> & a, int n) {
        half = 1;
        while (half < n) half *= 2;
        t = vector <int> (half * 2);
        for (int i = 0; i < n; i++)
            t[i + half] = a[i];
        for (int i = half - 1; i >= 1; i--)
            t[i] = t[i * 2] + t[i * 2 + 1];
    }

    /* ... */
};
```

```
void add (int pos, int val) {
    for (pos += half; pos > 0;
         pos /= 2)
        t[pos] += val;
}

int sum (int lo, int hi) {
    int res = 0;
    for (lo += half, hi += half;
         lo < hi;
         lo /= 2, hi /= 2) {
        if (lo & 1)
            res += t[lo++];
        if (hi & 1)
            res += t[--hi];
    }
    return res;
}
```

Вопросы?

Вопросы?