Zadania domowe. Blok 4. Zestaw 2

Maciej Poleski

28 maja 2012

1 Statystyka

Drzewo licznikowe jak w zadaniu R.

```
static uint32_t nextpow2(uint32_t v)
{
    uint32_t r = 1;
    while(r < v)
        r *= 2;
    return r;
}

class SumTree
{
public:
    SumTree(std::size_t size) :
        _size(nextpow2(size)), _tree(new int[_size * 2])
    {
        memset(_tree,0,_size*2*sizeof(int));
    }

    ~SumTree()
    {
        delete [] _tree;
    }

    void update(int ix, int value);
    int difference(int ix, int k)
    {
        return sum(ix,ix+k-1)-sum(ix+k,ix+k+k-1);
    }

private:
    int sum(uint32_t a, uint32_t b);
    int value(uint32_t index)
    {
        return _tree[index + _size];
    }
    uint32_t size() const
    {
        return _size;
```

```
    }

private:
    std::size_t _size;
    int *_tree;
};

void SumTree::update(int index, int v)
{
    int diff=v-_tree[_size+index];
    for(size_t i = index + _size; i > 0; i /= 2)
    {
        _tree[i] += diff;
    }
}

int SumTree::sum(uint32_t a, uint32_t b)
{
    if(a == 0 && b == _size - 1)
        return _tree[1];
    uint_fast32_t left = a + _size;
    uint_fast32_t right = b + _size;
    int result = 0;
    uint_fast8_t height = 0;
    uint_fast32_t i = left;
    while(true)
    {
        if(left > right)
            break;
        while((i << height) < left || (((i + 1) << height) - 1) > right)
        {
            i *= 2;
            --height;
        }
        while((((i / 2) << (height + 1)) >= left) &&
                (((i / 2 + 1) << (height + 1)) - 1 <= right))
        {
            i /= 2;
            ++height;
        }
        result += _tree[i];
```

```
        left = (i + 1) << height;
        ++i;
    }
    return result;
}
```

Zamiast funkcji `init(int)` istnieje konstruktor.

2 Minimax

Drzewo licznikowe jak w zadaniu R.

```
static uint32_t nextpow2(uint32_t v)
{
    uint32_t r = 1;
    while(r < v)
        r *= 2;
    return r;
}

struct Node
{
    int min;
    int max;
};

class SumTree
{
public:
    SumTree(std::size_t size) :
        _size(nextpow2(size)), _tree(new Node[_size * 2])
    {
    }

    ~SumTree()
    {
        delete [] _tree;
    }

    void update(int ix, int value);
    int max(int a, int b);
```

```
    int min(int a, int b);

private:
    uint32_t size() const
    {
        return _size;
    }

private:
    std::size_t _size;
    Node *_tree;
};

void SumTree::update(int index, int v)
{
    _tree[_size+index].min=_tree[_size+index].max=v;
    for(size_t i = (index + _size)/2; i > 0; i /= 2)
    {
        _tree[i].min=std::min(_tree[i*2].min,_tree[i*2+1].min);
        _tree[i].max=std::max(_tree[i*2].max,_tree[i*2+1].max);
    }
}

int SumTree::min(int a, int b)
{
    if(a == 0 && b == _size - 1)
        return _tree[1].min;
    uint_fast32_t left = a + _size;
    uint_fast32_t right = b + _size;
    int result = std::numeric_limits<int>::max();
    uint_fast8_t height = 0;
    uint_fast32_t i = left;
    while(true)
    {
        if(left > right)
            break;
        while((i << height) < left || (((i + 1) << height) - 1) > right)
        {
            i *= 2;
            --height;
        }
```

```
        while((((i / 2) << (height + 1)) >= left) &&
                (((i / 2 + 1) << (height + 1)) - 1 <= right))
        {
            i /= 2;
            ++height;
        }
        result = std::min(result, _tree[i].min);
        left = (i + 1) << height;
        ++i;
    }
    return result;
}

int SumTree::max(int a, int b)
{
    if(a == 0 && b == _size - 1)
        return _tree[1].max;
    uint_fast32_t left = a + _size;
    uint_fast32_t right = b + _size;
    int result = std::numeric_limits<int>::min();
    uint_fast8_t height = 0;
    uint_fast32_t i = left;
    while(true)
    {
        if(left > right)
            break;
        while((i << height) < left || (((i + 1) << height) - 1) > right)
        {
            i *= 2;
            --height;
        }
        while((((i / 2) << (height + 1)) >= left) &&
                (((i / 2 + 1) << (height + 1)) - 1 <= right))
        {
            i /= 2;
            ++height;
        }
        result = std::max(result, _tree[i].max);
        left = (i + 1) << height;
        ++i;
    }
```

```
    return result;
}
```

Zamiast funkcji `init(int)` istnieje konstruktor.

3 Multizbiór

Drzewo licznikowe jak w zadaniu R + funkcja `kthelem(int)`.

```
static uint32_t nextpow2(uint32_t v)
{
    uint32_t r = 1;
    while(r < v)
        r *= 2;
    return r;
}

class SumTree
{
public:
    SumTree(std::size_t size) :
        _size(nextpow2(size)), _tree(new size_t[_size * 2])
    {
        memset(_tree,0,_size*2*sizeof(size_t));
    }

    ~SumTree()
    {
        delete [] _tree;
    }

    void insert(int a, int k);
    void delete(int index)
    {
        insert(index,-_tree[_size+index]);
    }
    int kthelem(int k);

private:
    uint32_t size() const
    {
```

```
        return _size;
    }

private:
    std::size_t _size;
    size_t *_tree;
};

void SumTree::insert(int index, int v)
{
    for(size_t i = index + _size; i > 0; i /= 2)
    {
        _tree[i] += v;
    }
}

int SumTree::kthelem(int k)
{
    size_t i=1;
    while(i<_size)
    {
        if(k<=_tree[i*2])
        {
            i*=2;
        }
        else
        {
            k-=_tree[i*2];
            i=i*2+1;
        }
    }
    return i-_size;
}
```

Zamiast funkcji `init(int)` istnieje konstruktor. Zakładam że `kthelem(1)` ma zwrócić najmniejszy element należący do zbioru oraz że poszukiwany element istnieje.