**Структуры, использующие идею бинарного поиска**

1. **Дерево двоичного поиска**

Библиотека: <set>

Объявление: set <\*тип данных\*> st;

Объявление итератора: set <\*тип данных\*>::iterator it;

Методы: insert(X); begin(); end(); size(); empty(); find(x); count(x),remove(x), erase(it1, it2);

Функционал:

1.Быстрая сортировка за *O*(*logn*).

2.Быстрый поиск за *O*(*logn*).

3.Нахождение количества разных элементов за О(1)

4. Удаление элементов за *O*(*logn* + *distance*(*first*, *last*)).

1. **Полное бинарное дерево**

**Функционал:**

1. **Быстрый поиск элемента**

struct b\_tree\_T

{

int tree[1<<MAX\_LVL];

int pred(int x)

{

return (x/2);

}

int potom(int x, int lr)

{

return (2\*x + lr);

}

int lvl\_count(int x)

{

int level=1, ch=1;

while(ch<x)

{

ch\*=2;

level++;

}

lvl\_c=level;

f\_leave=ch;

return level;

}

void make\_btree(int a)

{

int lvl = lvl\_count(a);

int cn = (1<<(lvl))-1;

length = cn;

}

void L\_iniz()

{

for (int i=f\_leave; i<f\_leave+n; i++)

{

tree[i]=1;

}

}

void sum(int len)

{

for (int i=len-f\_leave; i>=1; i--)

{

tree[i]= tree[potom(i, 0)] + tree[potom(i, 1)];

}

}

void t\_update(int ind)

{

for (int i=ind/2; i>=1; i/=2)

{

tree[i]=tree[potom(i, 0)] + tree[potom(i, 1)];

}

}

int k\_find(int ind\_f\_leave, int element)

{

int ind=1;

while (ind<ind\_f\_leave)

{

if (tree[potom(ind,0)]>=element) {ind=potom(ind,0);}

else {element-=tree[potom(ind,0)]; ind=potom(ind,1);}

}

return ind;

}

} BT;

**3.Дерево интервалов**

Функционал:

1. Очень быстро выполняет операции на отрезках. Поиск и модификации за O(logN);

struct IT\_T

{

struct vertex

{

int val;

int Tagged;

int Left;

int Right;

};

vertex tree[1<<MAX\_LVL];

int potom(int x, int lr)

{

return (2\*x + lr);

}

int pred(int x)

{

return (x/2);

}

int lvl\_count(int x)

{

int level=1, ch=1;

while(ch<x)

{

ch\*=2;

level++;

}

return level;

}

int ln\_count(int b)

{

lvl = lvl\_count(b);

return (1<<(lvl))-1;

}

void push(int v)

{

//cout << v <<" " <<tree[v].val <<"\n";

int l\_p = v<<1;

tree[v].Tagged = 0;

tree[l\_p].Tagged = 1;

tree[l\_p+1].Tagged = 1;

tree[l\_p].val = tree[l\_p+1].val = (tree[v].val)>>1;

//cout << potom(v,0).val << " " << potom(v,1).val << " " << tree[v].val/2 <<"\n";

}

void assign(int v, int new\_val, int sL, int sR)

{

if( (tree[v].Left != sL) || (tree[v].Right != sR) )

{

if (tree[v].Tagged) push(v);

int pot = (v << 1);

int M = tree[pot].Right;

if(sR <= M) assign(pot, new\_val, sL, sR);

else if(M < sL) assign(pot + 1, new\_val, sL, sR);

else {assign(pot, new\_val, sL, M); assign(pot + 1, new\_val, M + 1, sR);}

tree[v].val = potom(vr/2,0).val + potom(vr/2,1).val;

}

else{

//cout << tree[v].val << "\n\n";

tree[v].Tagged = 1;

tree[v].val = new\_val\*(sR-sL+1);

}

}

int Search(int v, int sL, int sR)

{

//cout << v << " " <<sL << " " <<sR << "\n";

if( (tree[v].Left != sL) || (tree[v].Right != sR) )

{

if (tree[v].Tagged) push(v);

int l\_pot = potom(v,0);

int M = tree[l\_pot].Right;

if(sR <= M) return Search(l\_pot, sL, sR);

else if(M < sL)return Search(l\_pot + 1, sL, sR);

else return Search(l\_pot, sL, M) + Search(l\_pot + 1, M + 1, sR);

}

else{

//cout << tree[v].val << "\n\n";

return tree[v].val;

}

}

void iniz\_t()

{

for (int i = l\_count; i <= ln; ++i)

{

IT.tree[i].Left = IT.tree[i].Right = i-l\_count+1;

}

for (int i = ((ln+1)>>1)-1; i >=1; --i)

{

int l\_p = potom(i,0), r\_p = potom(i,1);

tree[i].val = tree[l\_p].val + tree[r\_p].val;

tree[i].Left = tree[l\_p].Left;

tree[i].Right = tree[r\_p].Right;

}

}

} IT;