* **Segment tree**

**// summation  
struct segmenttree {  
 int n;  
 vector<int> st;**

**void init(int \_n) {  
 this->n = \_n;  
 st.resize(4 \* n, 0);  
 }  
  
 void build(int start, int ending, int node, vector<int> &v) {  
 // leaf node base case  
 if (start == ending) {  
 st[node] = v[start];  
 return;  
 }  
  
 int mid = (start + ending) / 2;  
  
 // left subtree is (start,mid)  
 build(start, mid, 2 \* node + 1, v);  
  
 // right subtree is (mid+1,ending)  
 build(mid + 1, ending, 2 \* node + 2, v);  
  
 st[node] = st[node \* 2 + 1] + st[node \* 2 + 2];  
 }  
  
 int query(int start, int ending, int l, int r, int node) {  
 // non overlapping case  
 if (start > r || ending < l) {  
 return 0;  
 }  
  
 // complete overlap  
 if (start >= l && ending <= r) {  
 return st[node];  
 }  
  
 // partial case  
 int mid = (start + ending) / 2;  
  
 int q1 = query(start, mid, l, r, 2 \* node + 1);  
 int q2 = query(mid + 1, ending, l, r, 2 \* node + 2);  
  
 return q1 + q2;  
 }  
  
 void update(int start, int ending, int node, int index, int value) {  
 // base case  
 if (start == ending) {  
 st[node] = value;  
 return;  
 }  
  
 int mid = (start + ending) / 2;  
 if (index <= mid) {  
 // left subtree  
 update(start, mid, 2 \* node + 1, index, value);  
 } else {  
 // right  
 update(mid + 1, ending, 2 \* node + 2, index, value);  
 }  
  
 st[node] = st[node \* 2 + 1] + st[node \* 2 + 2];  
  
 return;  
 }  
  
 void build(vector<int> &v) {  
 build(0, n - 1, 0, v);  
 }  
  
 int query(int l, int r) {  
 return query(0, n - 1, l, r, 0);  
 }  
  
 void update(int x, int y) {  
 update(0, n - 1, 0, x, y);  
 }  
};  
  
int main() {  
 vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8};  
 // cout << v.size();  
  
 segmenttree tree;  
  
 tree.init(v.size());  
  
 tree.build(v);  
  
 cout << tree.query(0, 4) << '\n';  
  
 tree.update(4, 10);  
  
 cout << tree.query(2, 6) << '\n';  
  
 tree.update(2, 20);  
  
 cout << tree.query(0, 4) << '\n';  
  
 return 0;  
}**

* **segment tree with lazy propagation.**

**// summation**

**struct segmenttree {**

**int n;**

**vector<int> st, lazy;**

**void init(int \_n) {**

**this->n = \_n;**

**st.resize(4 \* n, 0);**

**lazy.resize(4 \* n, 0);**

**}**

**void build(int start, int ending, int node, vector<int> &v){**

**// leaf node base case**

**if (start == ending) {**

**st[node] = v[start];**

**return;**

**}**

**int mid = (start + ending) / 2;**

**// left subtree is (start,mid)**

**build(start, mid, 2 \* node + 1, v);**

**// right subtree is (mid+1,ending)**

**build(mid + 1, ending, 2 \* node + 2, v);**

**st[node] = st[node \* 2 + 1] + st[node \* 2 + 2];**

**}**

**int query(int start, int ending, int l, int r, int node) {**

**// non overlapping case**

**if (start > r || ending < l) {**

**return 0;**

**}**

**// lazy propagation / clear the lazy update**

**if (lazy[node] != 0) {**

**// pending updates**

**// update the segment tree node**

**st[node] += lazy[node] \* (ending - start + 1);**

**if (start != ending) {**

**// propagate the updated value**

**lazy[2 \* node + 1] += lazy[node];**

**lazy[2 \* node + 2] += lazy[node];**

**}**

**lazy[node] = 0;**

**}**

**// complete overlap**

**if (start >= l && ending <= r) {**

**return st[node];**

**}**

**// partial case**

**int mid = (start + ending) / 2;**

**int q1 = query(start, mid, l, r, 2 \* node + 1);**

**int q2 = query(mid + 1, ending, l, r, 2 \* node + 2);**

**return q1 + q2;**

**}**

**void update(int start, int ending, int node, int l, int r, int value) {**

**// non overlapping case**

**if (start > r || ending < l) {**

**return ;**

**}**

**// lazy propagation / clear the lazy update**

**if (lazy[node] != 0) {**

**// pending updates**

**// update the segment tree node**

**st[node] += lazy[node] \* (ending - start + 1);**

**if (start != ending) {**

**// propagate the updated value**

**lazy[2 \* node + 1] += lazy[node];**

**lazy[2 \* node + 2] += lazy[node];**

**}**

**lazy[node] = 0;**

**}**

**// complete overlap**

**if (start >= l && ending <= r) {**

**st[node] += value \* (ending - start + 1);**

**if (start != ending) {**

**lazy[2 \* node + 1] += value;**

**lazy[2 \* node + 2] += value;**

**}**

**return;**

**}**

**// partial case**

**int mid = (start + ending) / 2;**

**update(start, mid, 2 \* node + 1, l, r, value);**

**update(mid + 1, ending, 2 \* node + 2, l, r, value);**

**st[node] = st[node \* 2 + 1] + st[node \* 2 + 2];**

**return;**

**}**

**void build(vector<int> &v) {**

**build(0, n - 1, 0, v);**

**}**

**int query(int l, int r) {**

**return query(0, n - 1, l, r, 0);**

**}**

**void update(int l, int r, int x) {**

**update(0, n - 1, 0, l, r, x);**

**}**

**};**

**int main(){**

**vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8};**

**// cout << v.size();**

**segmenttree tree;**

**tree.init(v.size());**

**tree.build(v);**

**cout << tree.query(0, 4) << '\n';**

**tree.update(0, 1, 10);**

**cout << tree.query(0, 4) << '\n';**

**return 0;**

**}**