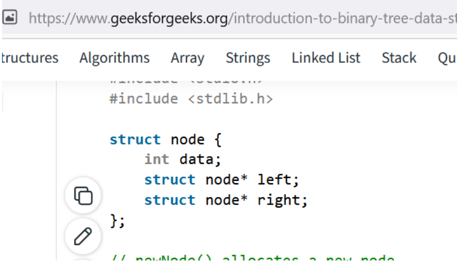
You are doing trees wrong!
(and graphs too)

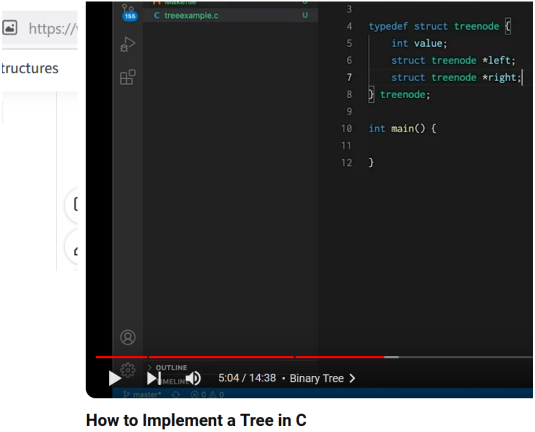
by me

December 29, 2023

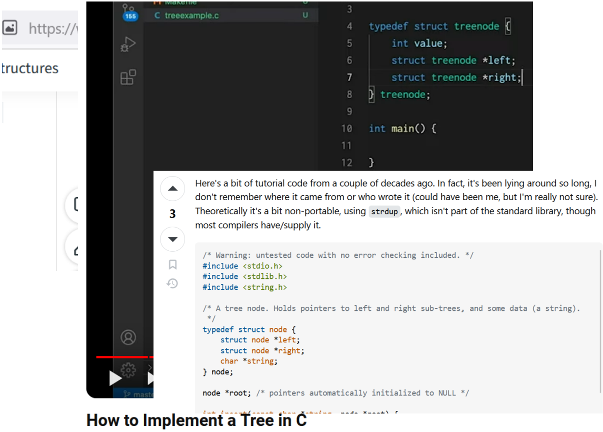
You are doing trees wrong! – Motivation



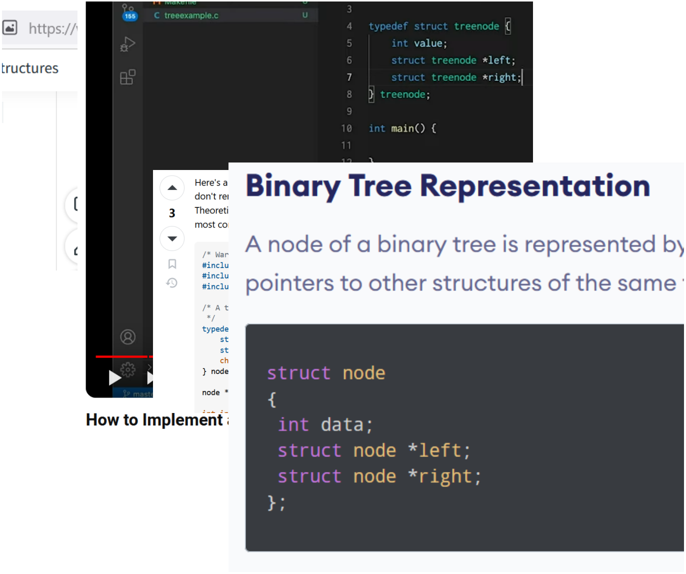
You are doing trees wrong! – Motivation



You are doing trees wrong! – Motivation



You are doing trees wrong! – Motivation



You are doing trees wrong!

Naive Way

```
struct node {
  struct node* left;
  struct node* right;
};
```

You are doing trees wrong!

Naive Way

```
struct node {
  struct node* left;
  struct node* right;
};
```

Better™ Way

```
struct node {
  int left;
  int right;
};
```

You are doing trees wrong!

Naive Way

```c
struct node {
  struct node* left;
  struct node* right;
};
```

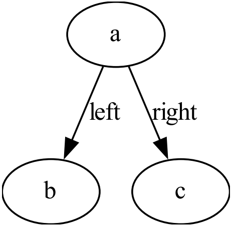
Better^{TM} Way

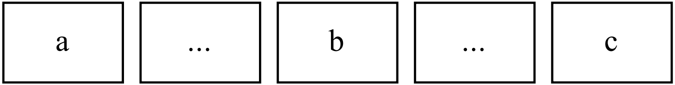
```c
struct node {
  int left;
  int right;
};

struct tree {
  struct node* nodes;
  int count, capacity;
};
```

You are doing trees wrong! – Memory Layout

Naive Way

```
struct node {
  struct node* left;
  struct node* right;
};
```

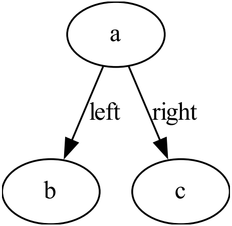


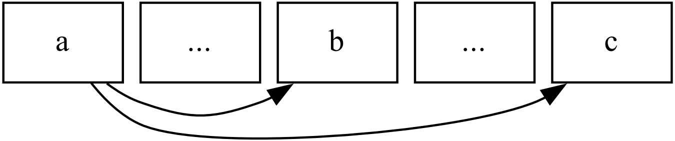


You are doing trees wrong! – Memory Layout

Naive Way

```
struct node {
  struct node* left;
  struct node* right;
};
```

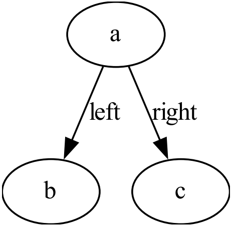


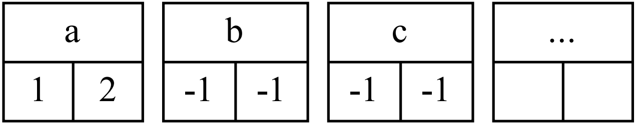


You are doing trees wrong! – Memory Layout

Better™ Way

```c
struct node {
  int left;
  int right;
};

struct tree {
  struct node* nodes;
  int count, capacity;
};
```

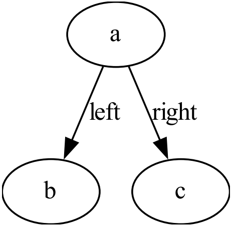


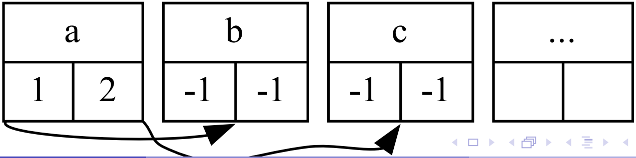


You are doing trees wrong! – Memory Layout

Better™ Way

```
struct node {
  int left;
  int right;
};

struct tree {
  struct node* nodes;
  int count, capacity;
};
```





You are doing trees wrong! – Insert

Naive Way

```
struct node* make_node(void) {
  struct node* parent =
      malloc(...);
  parent->left = NULL;
  parent->right = NULL;
  return parent;
}
```

You are doing trees wrong! – Insert

Naive Way

```c
struct node* make_node(void) {
  struct node* parent =
      malloc(...);
  parent->left = NULL;
  parent->right = NULL;
  return parent;
}
```

Better™ Way

```c
int make_node(struct tree* t) {
  reserve(t);
  t->nodes[t->count].left = -1;
  t->nodes[t->count].right = -1;

  return t->count++;
}
```

You are doing trees wrong! – Insert

Naive Way

```c
struct node* make_node(void) {
  struct node* parent =
      malloc(...);
  parent->left = NULL;
  parent->right = NULL;
  return parent;
}
```

Better™ Way

```c
int make_node(struct tree* t) {
  reserve(t);
  t->nodes[t->count].left = -1;
  t->nodes[t->count].right = -1;

  return t->count++;
}

void reserve(struct tree* t) {
  if (t->count == t->capacity) {
    t->capacity = t->capacity ?
      t->capacity*2 : 8;
    t->nodes = realloc(
      t->nodes, t->capacity);
  }
}
```

You are doing trees wrong! – Insert

Naive Way

```c
struct node* make_node(void) {
  struct node* parent =
      malloc(...);
  parent->left = NULL;
  parent->right = NULL;
  return parent;
}

struct node* a = make_node();
struct node* b = make_node();
struct node* c = make_node();
a->left = b;
a->right = c;
```

Better™ Way

```c
int make_node(struct tree* t) {
  reserve(t);
  t->nodes[t->count].left = -1;
  t->nodes[t->count].right = -1;

  return t->count++;
}

int a = make_node(tree);
int b = make_node(tree);
int c = make_node(tree);
tree->nodes[a].left = b;
tree->nodes[a].right = c;
```

You are doing trees wrong! – Insert Performance

```
Run on (16 X 2496 MHz CPU s)
CPU Caches:
  L1 Data 48 KiB (x8)
  L1 Instruction 32 KiB (x8)
  L2 Unified 1280 KiB (x8)
  L3 Unified 18432 KiB (x1)
--------------------------------------------------------------------
Benchmark                             Time             CPU   Iterations
--------------------------------------------------------------------
BM_malloc_n/1000                  41436 ns         35714 ns        28000
BM_malloc_n/100000              4249057 ns       4087936 ns          172
BM_malloc_n/1000000            44248594 ns      42968750 ns           16
BM_malloc_logn/1000                1346 ns          1367 ns       560000
BM_malloc_logn/100000            122387 ns        122070 ns         8960
BM_malloc_logn/1000000          1372260 ns       1196289 ns          640
BM_malloc_realloc/1000           260472 ns        250000 ns        10000
BM_malloc_realloc/100000       25432239 ns      25468750 ns          100
BM_malloc_realloc/1000000     246381140 ns     242187500 ns           10
```

You are doing trees wrong! – Find

Naive Way

```
struct node* find(
    struct node* r,
    const void* data) {
  struct node* n;

  if (compare(r->data, data))
    return r;

  if (r->left)
    if (n = find(r->left, data))
      return n;
  if (r->right)
    if (n = find(r->right, data))
      return n;

  return NULL;
}
```

You are doing trees wrong! – Find

Naive Way

```
struct node* find(
    struct node* r,
    const void* data) {
  struct node* n;

  if (compare(r->data, data))
    return r;

  if (r->left)
    if (n = find(r->left, data))
      return n;
  if (r->right)
    if (n = find(r->right, data))
      return n;

  return NULL;
}
```

Better™ Way

```
int find(
    struct tree* t,
    int r,
    const void* data) {
  int n;
  if (compare(t->nodes[r].data,
      data))
    return r;

  if (t->nodes[r].left >= 0)
    if (n = find(t,
        t->nodes[r].left, data))
      return n;
  if (t->nodes[r].right >= 0)
    if (n = find(t,
        t->nodes[r].right, data))
      return n;

  return -1;
}
```

You are doing trees wrong! – Find

Naive Way

```c
struct node* find(
    struct node* r,
    const void* data) {
  struct node* n;

  if (compare(r->data, data))
    return r;

  if (r->left)
    if (n = find(r->left, data))
      return n;
  if (r->right)
    if (n = find(r->right, data))
      return n;

  return NULL;
}
```

Better™ Way

```c
int find(
    struct tree* t,
    const void* data) {
  for (int n = 0; n != t->count;
      ++n) {
    if (compare(
        t->nodes[n].data, data))
      return n;
  }
  return -1;
}
```

You are doing trees wrong! – Delete Node

Naive Way

```c
void delete(struct node* n) {
  struct node* p = find_parent(n);

  if (p->left == p)
    p->left = NULL;
  if (p->right == p)
    p->right = NULL;

  delete_r(n);
}

void delete_r(struct node* n) {
  if (n->left)
    delete_r(n->left);
  if (n->right)
    delete_r(n->right);

  free_node(n);
}
```

You are doing trees wrong! – Delete Node

Naive Way

```
void delete(struct node* n) {
  struct node* p = find_parent(n);

  if (p->left == p)
    p->left = NULL;
  if (p->right == p)
    p->right = NULL;

  delete_r(n);
}

void delete_r(struct node* n) {
  if (n->left)
    delete_r(n->left);
  if (n->right)
    delete_r(n->right);

  free_node(n);
}
```

Better™ Way

```
void delete(
    struct tree* t, int n) {
  int p = find_parent(t, n);

  if (t->nodes[p].left == n)
    t->nodes[p].left = -1;
  if (t->nodes[p].right == n)
    t->nodes[p].right = -1;

  // does not alter structure!
  swap_nodes(n, --t->count);
}
```

You are doing trees wrong! – Delete Tree

Naive Way

```
void delete(struct node* n) {
  struct node* p = find_parent(n);

  if (p->left == p)
    p->left = NULL;
  if (p->right == p)
    p->right = NULL;

  delete_r(n);
}

void delete_r(struct node* n) {
  if (n->left)
    delete_r(n->left);
  if (n->right)
    delete_r(n->right);

  free_node(n);
}
```

You are doing trees wrong! – Delete Tree

Naive Way

```c
void delete(struct node* n) {
  struct node* p = find_parent(n);

  if (p->left == p)
    p->left = NULL;
  if (p->right == p)
    p->right = NULL;

  delete_r(n);
}

void delete_r(struct node* n) {
  if (n->left)
    delete_r(n->left);
  if (n->right)
    delete_r(n->right);

  free_node(n);
}
```

Better™ Way

```c
void delete_tree(struct tree* t) {
    free(t->nodes);
}
```

You are doing trees wrong! – (De-)Serialization

Naive Way

```
void serialize(
    struct node* n) {
  FILE* fp = fopen(...);
  /* (create hashmap of indices <-> pointers
       -- left as an exercise to the user) */
  fwrite(&hashmap->count, sizeof(int), 1, fp);
  serialize_r(fp, n, &hashmap);
  fclose(fp);
}

void serialize_r(
    FILE* fp,
    struct node* n,
    hashmap* hm) {
  int left_idx = hm_find(&hm, n->left);
  int right_idx = hm_find(&hm,n->right);
  fwrite(&left_idx, sizeof(int), 1, fp);
  fwrite(&right_idx, sizeof(int), 1, fp);

  if (n->left)
    serialize_r(fp, n->left, &hm);
  if (n->right)
    serialize_r(fp, n->right, &hm);
}
```

You are doing trees wrong! – (De-)Serialization

Naive Way

```
void serialize(
    struct node* n) {
  FILE* fp = fopen(...);
  /* (create hashmap of indices <-> pointers
       -- left as an exercise to the user) */
  fwrite(&hashmap->count, sizeof(int), 1, fp);
  serialize_r(fp, n, &hashmap);
  fclose(fp);
}

void serialize_r(
    FILE* fp,
    struct node* n,
    hashmap* hm) {
  int left_idx = hm_find(&hm, n->left);
  int right_idx = hm_find(&hm,n->right);
  fwrite(&left_idx, sizeof(int), 1, fp);
  fwrite(&right_idx, sizeof(int), 1, fp);

  if (n->left)
    serialize_r(fp, n->left, &hm);
  if (n->right)
    serialize_r(fp, n->right, &hm);
}
```

Better™ Way

```
void serialize(struct tree* t) {
  FILE* fp = fopen(...);
  fwrite(&t->count, sizeof(int),
      1, fp);
  fwrite(t->nodes, 1,
      sizeof(*t->nodes),
      t->count);
  fclose(fp);
}
```

You are doing trees wrong! – Summary

Pros

Cons

- More memory management overhead

You are doing trees wrong! – Summary

Pros

- More opportunities for tuning performance

Cons

- More memory management overhead

You are doing trees wrong! – Summary

Pros

- More opportunities for tuning performance
- Faster (fewer memory allocations)

Cons

- More memory management overhead

You are doing trees wrong! – Summary

Pros

- More opportunities for tuning performance
- Faster (fewer memory allocations)
- Less memory consumption

Cons

- More memory management overhead

You are doing trees wrong! – Summary

Pros

- More opportunities for tuning performance
- Faster (fewer memory allocations)
- Less memory consumption
- Higher cache locality

Cons

- More memory management overhead

You are doing trees wrong! – Summary

Pros

- More opportunities for tuning performance
- Faster (fewer memory allocations)
- Less memory consumption
- Higher cache locality
- Recursion is not always necessary! Opportunity for more efficient algorithms

Cons

- More memory management overhead

Thank you!

`https://github.com/TheComet/do-trees-right`