Discussion 7: Disjoint set union, Graph, and more

CS101 Fall 2024

CS101 Course Team

Nov 2024

Disjoint set union

Motivation: union-find data structure + merge–find set $\rightarrow$ Disjoint set union.

Operation:

1. find(x): find which set x belongs to.
   Time complexity: $\Theta(h)$
2. set_union(x,y): merge the set x and y belongs to into one set.
   Time complexity: $\Theta(h)$

Disjoint set union

The height of the tree:

1. Worst case: $h = \Theta(\log(n))$
2. Average case: $h = o(\log(n)))$
3. Best case: $h = \Theta(1)$

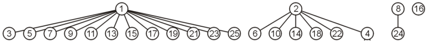


Figure: best case

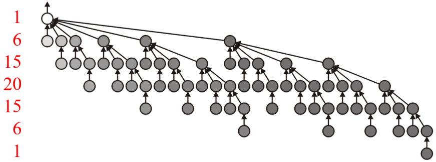


Figure: worst case

Disjoint set union

Optimization: Merge by rank + Path Compression $\Rightarrow h = \Theta(\alpha(n))$
Where $\alpha(n)$ is the inverse Ackermann function, which could be treated as constant when implementation(usually regard $\alpha(n) \approx 4$).
Which could be quite easy to implement. (one line of function for find operation!)

$$\alpha(n) = \min\{i \mid A(i, i) \geq n\}$$

where $A(i, j)$ is the Ackermann function:

$$A(i,j) = \begin{cases} j+1 & \text{if } i = 0 \\ A(i-1, 1) & \text{if } i > 0 \text{ and } j = 0 \\ A(i-1, A(i, j-1)) & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

Graph

Some typical concepts of graphs $G = (V, E)$:

1. degree: the number of adjacent vertices
2. sub-graph
3. path: an ordered sequence of vertices $(v_0, v_1, \ldots, v_k)$
4. simple path: path with no repetitions
5. simple circle: the simple path of at least two vertices with the first and last vertices equal
6. Connectedness: Two vertices v_i, v_j are said to be connected if there exists a path from v_i to v_j
7. strong connected, weakly connected
8. Weighted graph
9. tre, forest
10. directed graph, undirected graph
11. in, out-degree for directed graph
12. source node, sink node

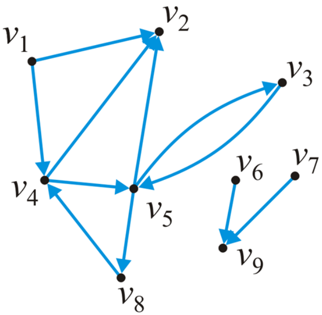
Graph

Ways to store graphs $G = (V, E)$:

1. adjacency matrix Space complexity: $\Theta(|V|^2)$
2. adjacency list Space complexity: $\Theta(|V| + |E|)$

$1 \quad \bullet \rightarrow 2 \rightarrow 4$
$2 \quad \bullet$
$3 \quad \bullet \rightarrow 5$
$4 \quad \bullet \rightarrow 2 \rightarrow 5$
$5 \quad \bullet \rightarrow 2 \rightarrow 3 \rightarrow 8$
$6 \quad \bullet \rightarrow 9$
$7 \quad \bullet \rightarrow 9$
$8 \quad \bullet \rightarrow 4$
$9 \quad \bullet$

	1	2	3	4	5	6	7	8	9
1		T		T					
2									
3					T				
4		T			T				
5		T	T					T	
6									T
7									T
8				T					
9									

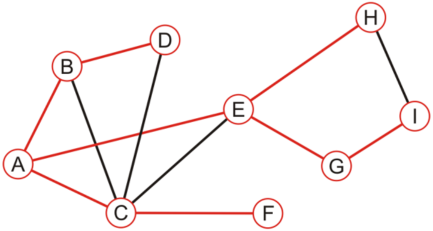


Graph

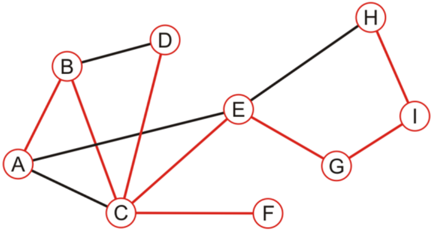
Graph traversal

1. BFS
2. DFS

A, B, C, E, D, F, G, H, I

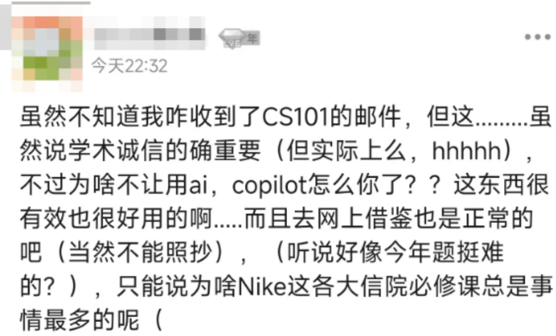


A, B, C, D, E, G, I, H, F



Academic Integrity

1. Spread code
2. generative AI



First

First

Figure: During Grading

First

Figure: During Grading

Figure: After grading

Multiple Choices (a)

Which of the following implementations do/does not affect the time complexity of any stack/queue operation?

- A When we implement a stack by an array, we put `stack.top()` at the first element of the array.
- B When we implement a stack by a singly linked-list with maintaining tail pointer, we put `stack.top()` at the tail of the linked-list.
- C When we implement a queue by a singly linked-list with maintaining tail pointer, we put `queue.back()` at the head of the linked-list and `queue.front()` at the tail.
- D When we implement a queue by a doubly linked-list with maintaining tail pointer, we put `queue.back()` at the head of the linked-list and `queue.front()` at the tail.

Mutliple Choices (c)

Which of the following statements is/are **TRUE**?

- A. The time complexity of bubble sort (no matter which optimization) is always not lower than insertion sort because it always performs not fewer swaps than insertion sort.
- B. The worst-case time complexity of counting the number of swaps of insertion sort on an array must be $\Omega(n^2)$.
- C. Insertion sort is more suitable for sorting small arrays compared to quick sort.
- D. Quick sort is more suitable for sorting large arrays than merge sort in all cases, especially for distributed data.

Fill in the blanks

- If we implement a stack by array and when the array is full, we move elements to another double-sized array, then for consecutively n `push`es to the empty stack, the time complexity of each `push` in general is $O(n)$, but the amortized time complexity is $\Theta(1)$.
- Consider an array of length n holding an uncommon type of elements, whose comparison take $\Theta(\log(n))$ time. Any in-place sorting algorithm based on comparison will have a worst-case time complexity of $\Omega(n \log^2 n)$.
- Array $[a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}]$ represents a binary min-heap containing 10 items, where the key of each item is a distinct integer. a_3, a_4, a_{10} could be the fourth smallest integer, if a_5 is the third smallest one.

Hash-Table insertions

There are a series of tuples (a, b) to be stored in a hash table using

- Quadratic probing. The probing function is $H_i(a, b) = (a + b + 0.5i + 0.5i^2) \mod 11$.
- Lazy erasing. A lazy-erased element is marked as E.

There is a hash table T which looks like

Index	0	1	2	3	4	5	6	7	8	9	10
Key Value			$(6, 7)$	$(1, 2)$		$(4, 9)$	E		E		

1. If we search for $(1, 1)$ in T, the probing sequence is $[2, 3, 5, 8, 1]$.
2. If we want to insert $(4, 1)$ into T (but we are not sure if it is in T), the probing sequence is $[5, 6, 8, 0]$, and finally it will be inserted at Index $[6]$.

Hash-Table insertions

There are a series of tuples (a, b) to be stored in a hash table using

- Quadratic probing. The probing function is $H_i(a, b) = (a + b + 0.5i + 0.5i^2) \mod 11$.
- Lazy erasing. A lazy-erased element is marked as E.

There is a hash table T which looks like

Index	0	1	2	3	4	5	6	7	8	9	10
Key Value			$(6,7)$	$(1,2)$		$(4,9)$	E		E		

1. If we create another hash table using lazy erasing but **linear probing**, then is it possible that starting from an empty hash table and after some insert and erase operations, the hash table looks the same as T?

Hash-Table insertions

There are a series of tuples (a, b) to be stored in a hash table using

- Quadratic probing. The probing function is $H_i(a, b) = (a + b + 0.5i + 0.5i^2) \mod 11$.
- Lazy erasing. A lazy-erased element is marked as E.

There is a hash table T which looks like

Index	0	1	2	3	4	5	6	7	8	9	10
Key Value			$(6,7)$	$(1,2)$		$(4,9)$	E		E		

1. If we create another hash table using lazy erasing but **linear probing**, then is it possible that starting from an empty hash table and after some insert and erase operations, the hash table looks the same as T?
   The probing sequence of searching $(4,9)$ is $2, 3, 4$ and we find that Index 4 is empty.

Hash-Table insertions

There are a series of tuples (a, b) to be stored in a hash table using

- Quadratic probing. The probing function is $H_i(a, b) = (a + b + 0.5i + 0.5i^2) \mod 11$.
- Lazy erasing. A lazy-erased element is marked as E.

There is a hash table T which looks like

Index	0	1	2	3	4	5	6	7	8	9	10
Key Value			$(6, 7)$	$(1, 2)$		$(4, 9)$	E		E		

1. If we create another hash table using lazy erasing but **linear probing**, then is it possible that starting from an empty hash table and after some insert and erase operations, the hash table looks the same as T?
   The probing sequence of searching $(4, 9)$ is $2, 3, 4$ and we find that Index 4 is empty.
   Therefore $(4, 9)$ is in T but we can't find it after searching, which contradicts.

Merge sort on Linked-lists

(a) Fill in the table according to what you know about merge sort and doubly linked-lists:

Data Structure \ Operation	Divide	Sub-problem(s)	Merge
Array	$\Theta(1)$	$2T(\frac{n}{2})$	$\Theta(\mathrm{n})$
Doubly Linked-list	$\Theta(\mathrm{n})$	$2T(\frac{n}{2})$	$\Theta(n)$

(c) We merge two sorted linked-lists $(a_1, a_2, a_3, a_4, a_5)$ and $(b_1, b_2, b_3, b_4, b_5)$ into one. Assume that these elements are distinct **except** $a_4 = b_3$. Suppose the result is

$$(b_1, b_2, a_1, a_2, a_3, b_3, a_4, b_4, a_5, b_5).$$

From this, you can infer that the number of inversions in the original array is at least 12.

Core code

```cpp
List merge(const List &x, const List &y) {
    if (x.empty())
        return y;
    if (y.empty())
        return x;
    auto xh = x.head(), yh = y.head();
    if (xh < yh)
        return con(xh, merge(x.nohead(), y));
    else
        return con(yh, merge(y.nohead(), x));
}
```

(d) In the above parts we discuss situations of doubly linked-lists, if `merge` is used to singly linked-lists, will the time complexity change? (Write "Yes" or "No".)

Karatsuba

Let A, B are two polynomials where $A(x) = \sum_{i=0}^{n} a_i x^i$, $B(x) = \sum_{i=0}^{n} b_i x^i$.

Recall the vertical multiplication on polynomials. Rewrite $A(x) = A_1(x)x^{\frac{n}{2}} + A_2(x)$ and $B(x) = B_1(x)x^{\frac{n}{2}} + B_2(x)$, where A_i, B_i are polynomials of $\frac{n}{2}$ or less degree :

$$
\begin{array}{rrr}
 & A_1 x^{\frac{n}{2}} & +A_2 \\
 & \times B_1 x^{\frac{n}{2}} & +B_2 \\
\hline
 & A_1 B_2 x^{\frac{n}{2}} & +A_2 B_2 \\
A_1 B_1 x^n & +A_2 B_1 x^{\frac{n}{2}} & \\
\hline
A_1 B_1 x^n & +(A_1 B_2 + A_2 B_1) x^{\frac{n}{2}} & +A_2 B_2.
\end{array}
$$

Write down the recurrence relation of this algorithm's time complexity and calculate it:

Karatsuba

Let A, B are two polynomials where $A(x) = \sum_{i=0}^{n} a_i x^i, B(x) = \sum_{i=0}^{n} b_i x^i$.

Recall the vertical multiplication on polynomials. Rewrite $A(x) = A_1(x)x^{\frac{n}{2}} + A_2(x)$ and $B(x) = B_1(x)x^{\frac{n}{2}} + B_2(x)$, where A_i, B_i are polynomials of $\frac{n}{2}$ or less degree :

$$\begin{array}{rrr}
 & A_1 x^{\frac{n}{2}} & +A_2 \\
 & \times B_1 x^{\frac{n}{2}} & +B_2 \\
\hline
 & A_1 B_2 x^{\frac{n}{2}} & +A_2 B_2 \\
A_1 B_1 x^n & +A_2 B_1 x^{\frac{n}{2}} & \\
\hline
A_1 B_1 x^n & +(A_1 B_2 + A_2 B_1) x^{\frac{n}{2}} & +A_2 B_2.
\end{array}$$

Write down the recurrence relation of this algorithm's time complexity and calculate it: $T(n) = 4T(\frac{n}{2}) + \Theta(n) = \Theta(n^2)$.

Karatsuba

Recall the vertical multiplication on polynomials. Rewrite $A(x) = A_1(x)x^{\frac{n}{2}} + A_2(x)$ and $B(x) = B_1(x)x^{\frac{n}{2}} + B_2(x)$, where A_i, B_i are polynomials of $\frac{n}{2}$ or less degree :

$$
\begin{array}{rrr}
 & A_1 x^{\frac{n}{2}} & +A_2 \\
 & \times B_1 x^{\frac{n}{2}} & +B_2 \\
\hline
 & A_1 B_2 x^{\frac{n}{2}} & +A_2 B_2 \\
A_1 B_1 x^n & +A_2 B_1 x^{\frac{n}{2}} & \\
\hline
A_1 B_1 x^n & +(A_1 B_2 + A_2 B_1)x^{\frac{n}{2}} & +A_2 B_2.
\end{array}
$$

Recall how Strassen's algorithm works for matrix multiplication: It decreases one time of multiplication (from 8 to 7), then its time complexity turns into $\Theta(n^{\log_2 7})$ from $\Theta(n^3)$. Consider how to reduce one multiplication, we wonder if $A_1 B_2 + A_2 B_1$ can be calculated with 1 multiplication with proper polynomial calculation:

Karatsuba

Recall the vertical multiplication on polynomials. Rewrite $A(x) = A_1(x)x^{\frac{n}{2}} + A_2(x)$ and $B(x) = B_1(x)x^{\frac{n}{2}} + B_2(x)$, where A_i, B_i are polynomials of $\frac{n}{2}$ or less degree :

$$
\begin{array}{rrr}
 & A_1 x^{\frac{n}{2}} & +A_2 \\
 & \times B_1 x^{\frac{n}{2}} & +B_2 \\
\hline
 & A_1 B_2 x^{\frac{n}{2}} & +A_2 B_2 \\
A_1 B_1 x^n & +A_2 B_1 x^{\frac{n}{2}} & \\
\hline
A_1 B_1 x^n & +(A_1 B_2 + A_2 B_1) x^{\frac{n}{2}} & +A_2 B_2.
\end{array}
$$

Recall how Strassen's algorithm works for matrix multiplication: It decreases one time of multiplication (from 8 to 7), then its time complexity turns into $\Theta(n^{\log_2 7})$ from $\Theta(n^3)$.
Consider how to reduce one multiplication, we wonder if $A_1 B_2 + A_2 B_1$ can be calculated with 1 multiplication with proper polynomial calculation:
$A_1 B_2 + A_2 B_1 = (A_1 + A_2)(B_1 + B_2) - A_1 B_1 - A_2 B_2$

n-choose-m

If we want to enumerate all ways of choosing m numbers from $\{1, 2, \cdots, n\}$, we can implement it by m `for` loops:

```
int main() {
    for (a[1] = 1; a[1] <= n - m + 1; ++a[1])
        for (a[2] = a[1] + 1; a[2] <= n - m + 2; ++a[2])
            ...
                for (a[m] = a[m - 1] + 1; a[m] <= n; ++a[m])
                    for (int j = 1; j <= m; ++j)
                        printf("%d%c", a[j], j < m ? ' ' : '\n');
}
```

n-choose-m

```
void loop(int i, int start, int end) {
    for (a[i] = start; a[i] <= end; ++a[i])
        if (i < m)
            loop(____, ____, ____);
        else
            for (int j = 1; j <= m; ++j)
                printf("%d%c", a[j], j < m ? ' ' : '\n');
}

int main() {
    loop(1, 1, n - m + 1);
}
```

n-choose-m

It is difficult to directly evaluate the time complexity of `loop(1, 1, n - m + 1)`. However, by the well-known fact that this algorithm enumerates all ways of choosing m numbers from n elements, we can simply derive the time complexity using the Binomial Coefficient:

$$T(n, m) = \Theta\left(m\binom{n}{m}\right) = \Theta\left(\frac{m \cdot n!}{m!(n-m)!}\right)$$

Now you need to prove mathematically that if m is a constant c, then the time complexity should be $T(n, c) = \Theta\left(n^c\right)$. **[Hint: you can check the Stirling Formula on the HINTS page.]**

n-choose-m

Prove by showing that $\frac{T(n,c)}{n^c} = \Theta(1)$.

$$\begin{aligned}
\frac{T(n,c)}{n^c} &= \frac{c \cdot n!}{c!(n-c)!n^c} = \Theta\left(\frac{n!}{(n-c)!n^c}\right) \\
&= \Theta\left(\frac{n^{n+\frac{1}{2}}e^{-n}}{(n-c)^{n-c+\frac{1}{2}}e^{-n+c}n^c}\right) \\
&= \Theta\left(\frac{n^{n-c+\frac{1}{2}}}{(n-c)^{n-c+\frac{1}{2}}}\right) \\
&= \Theta\left(\left(\frac{u+c}{u}\right)^{u+\frac{1}{2}}\right) && (u \leftarrow n-c) \\
&= \Theta\left(\left(1+\frac{c}{u}\right)^{u} \cdot \left(1+\frac{c}{u}\right)^{\frac{1}{2}}\right) = \Theta\left(e^c \cdot 1\right) = \Theta(1)
\end{aligned}$$

n-choose-m

$$\frac{n!}{(n-c)!} = n(n-1)(n-2)\cdots(n-c+1) = \Theta(n \cdot n \cdot n \cdots n) = \Theta(n^c)$$

n-choose-m

Now you need to show that if $m = \frac{n}{2}$ instead of a constant, then the time complexity should be $T\left(n, \frac{n}{2}\right) = o\left(n^{\frac{n}{2}}\right)$.

We can prove by $T\left(n, \frac{n}{2}\right) = \Theta\left(n^{\frac{1}{2}}2^n\right) = o\left(n^{\frac{n}{2}}\right)$.

$$
\begin{aligned}
T\left(n, \frac{n}{2}\right) =& \frac{\frac{n}{2} \cdot n!}{\left(\frac{n}{2}\right)!\left(\frac{n}{2}\right)!} \\
=& \Theta\left(\frac{n \cdot n^{n+\frac{1}{2}}e^{-n}}{\left(\frac{n}{2}\right)^{\frac{n}{2}+\frac{1}{2}}e^{-\frac{n}{2}}\left(\frac{n}{2}\right)^{\frac{n}{2}+\frac{1}{2}}e^{-\frac{n}{2}}}\right) && \text{(Stirling Formula)} \\
=& \Theta\left(\frac{n^{n+\frac{3}{2}}e^{-n}}{\left(\frac{n}{2}\right)^{n+1}e^{-n}}\right) = \Theta\left(\frac{n^{n+\frac{3}{2}}2^{n+1}}{n^{n+1}}\right)s = \Theta\left(n^{\frac{1}{2}}2^n\right)
\end{aligned}
$$

n-choose-m

And then

$$\begin{aligned}
&n^{\frac{1}{2}}2^n = o\left(n^{\frac{n}{2}}\right) \\
\iff &\lim_{n\to\infty} \frac{n^{\frac{1}{2}}2^n}{n^{\frac{n}{2}}} = 0 \\
\iff &\lim_{n\to\infty} 2^{\left(\frac{1}{2}\log n + n - \frac{n}{2}\log n\right)} = 2^{-\infty} = 0 \\
\Longleftarrow\ &\frac{1}{2}\log n + n = o\left(\frac{n}{2}\log n\right)
\end{aligned}$$