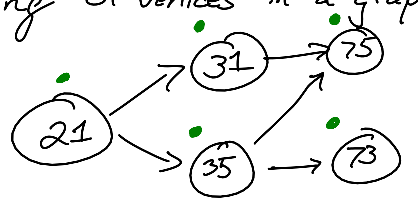
CS 35 Review Session

Topological Sort

An ordering of vertices in a graph s.t. all predecessors of a vertex appear before that vertex.

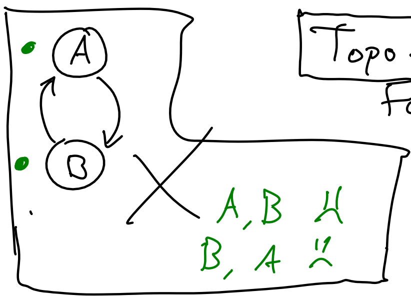


Unvisited

Visiting (in-progress)

Visited

Answer: 21, 35, 73, 31, 75



$\boxed{\text{Topo Sort}}$:

For each vertex v:

Look at v

$\boxed{\text{Look}}$ at vertex:

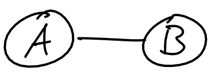
If vertex is unvisited:

- mark as visiting
- look at all vertices it can directly reach
- mark as visited
- add this vertex to front

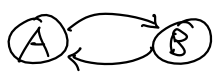
Else If vertex is visiting

Cycle !!

"Undirected" — if A→B, then B→A



≡



Big-O

"My algorithm is $O(n^3)$ steps."

For some variable n, # of steps I take is proportional to n^3.

"$\underbrace{f(n)}_{\text{how many steps?}}$ is $O(\underbrace{g(n)}_{\text{some mathematical expression}})$" $\equiv$ "$\exists c > 0, k \geq 1. \forall n \geq k. \ f(n) \leq c g(n)$"

Big-O Consequences

- Constants don't really matter: $O(n^2) = O(2n^2)$
- Smaller terms don't matter: $O(n^2 + n) = O(n^2)$
- All logarithms are the same: $O(\log_2 n) = O(\log_4 n)$

$O(n)$ steps
```
For i in 1 to n:
    Constant
End
```

```
For i in 1 to n:
    Constant          n times
End
For i in 1 to n:
    Constant          n times
End
```

$O(n) + O(n) = O(n)$

2n steps

$O(n)$ steps

```
For i in 1 to n:              run n times
    For j in 1 to n:          each full run of loop:
        print(i, j)           n prints
    End
End
```

n times that I print n things

n^2 prints

nesting $\Rightarrow$ *

$O(n) * O(n) = O(n^2)$

```
Set a to 1                    O(1)
For i In 1 to n:              +
    Multiply a by 2           O(n)
End For                       +
For j In 1 to a:              O(2^n)
    Constant
End
```

$$\underbrace{2 \cdot 2 \cdot 2 \cdot \ldots \cdot 2}_{2^n}$$

$= O(2^n) + O(n) + O(1) = O(2^n)$

```
Set a to n
While a > 1:
    Set a to ⌊a/2⌋
End
```

$\log_2(n)$

Inductive Proofs

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$$

$$\boxed{\forall k \geq 1. \quad P(k) \text{ is true}}$$ Inductive proof result

$$\longrightarrow P(n) \equiv \sum_{i=1}^{n} i = \frac{n^2+n}{2}$$

- $P(n)$: statement
- Base case: prove $P(1)$ is true
- Inductive case: prove, if $P(k)$ is true, then $P(k+1)$ true

$P(1) \Rightarrow P(2) \Rightarrow P(3) \Rightarrow \cdots \Rightarrow P(702345\,18)$

Base case:

show

$$P(1) \equiv \boxed{\sum_{i=1}^{1} i = \frac{1^2+1}{2}}$$

is true

$$1 = \frac{1+1}{2}$$

$$1 = \frac{2}{2}$$

$$1 = 1 \checkmark$$

Inductive case:

Assume $P(k)$ is true.

$$\sum_{i=1}^{k} i = \frac{k^2+k}{2}$$

show

$$P(k+1) \equiv \boxed{\sum_{i=1}^{k+1} i = \frac{(k+1)^2+k+1}{2}}$$

$$\sum_{i=1}^{k} i = \frac{k^2+k}{2}$$

$$\left(\sum_{i=1}^{k} i\right) + k + 1 = \frac{k^2+k}{2} + k + 1$$

$$1 + 2 + \cdots + k + (k+1) = \frac{k^2+k}{2} + k + 1$$

$$\left(\sum_{i=1}^{k+1} i\right) = \frac{k^2+k}{2} + k + 1$$

$$= \frac{k^2+k+2k+2}{2}$$

$$= \frac{(k^2+2k+1)+(k+1)}{2}$$

$$P(k+1) \equiv \left(\sum_{i=1}^{k+1} i\right) = \frac{(k+1)^2+(k+1)}{2}$$

Proving correctness

```
Function addUp(n):
    If n = 0 Then
        Return n
    Else
        Return 1 + addUp(n-1)
    EndIf
EndFunction
```

actual behavior

$P(k) \equiv$ addUp(k) returns k (for any $k \geq 0$)

expected behavior

Base case: $P(0) \equiv$ addUp(0) returns 0

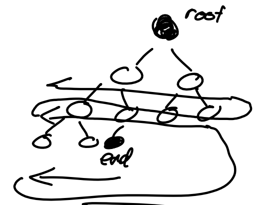
Because $n=0$, we enter the Then block. We return n, which is 0; therefore: addUp(0) returns 0.

Inductive case: If $P(k)$ is true, $P(k+1)$ is true

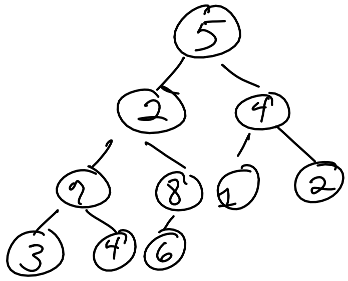
Because $k \geq 0$, $k+1 \geq 1$. So, addUp$(k+1)$ enters the Else block. We call addUp$(k+1-1)$ (the same as addUp(k)). By inductive hypotheses, addUp(k) returns k. So we return $1+k$. $\square$

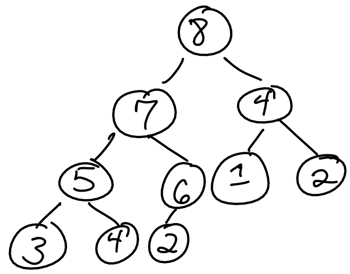
Heapify

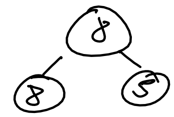
```
Function heapify(tree):
    For each index in tree from end to root:
        bubbleDown(index)      sink(index)
    EndFor
EndFunction
```



max-heap
min-heap

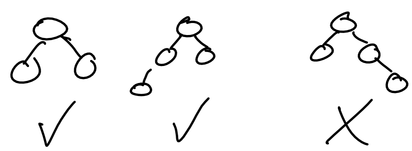






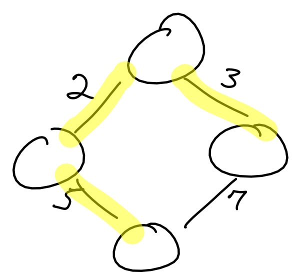
A max-heap is a complete binary tree s.t. each node ≥ its children

A complete binary tree is a binary tree s.t. all levels except the last are full and last level is packed to the left.



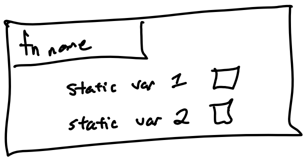
Minimum Spanning Trees:

NOT ON EXAM

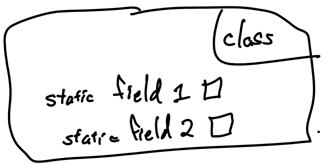


Stack Diagrams

Stack diagram captures a moment in time during program execution.



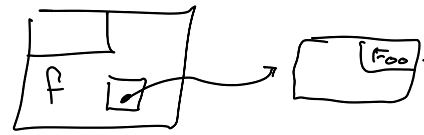
stack frame — a function is being called



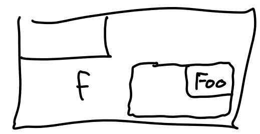
object — allocated

pointer

Foo* f = new Foo();

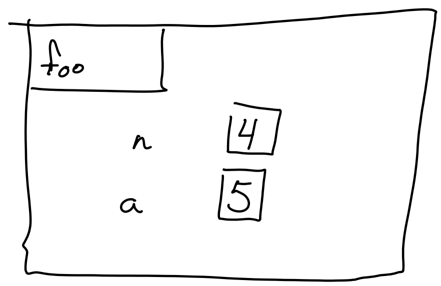


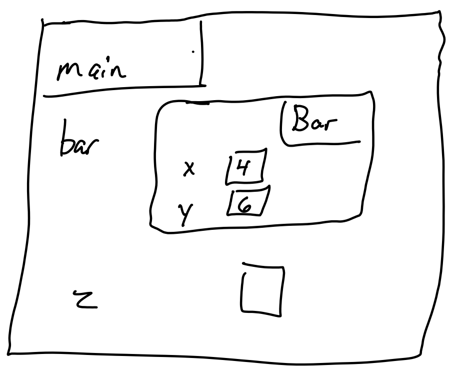
Foo f;

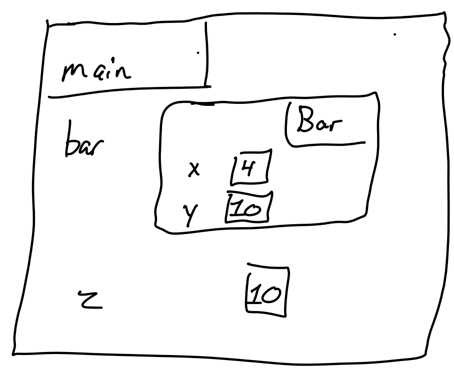



```
class Bar {
  public:
    int x = 4;
    int y = 6;
};

int foo(int n) {
    int a = n + 1;
    int b = a * 2;
    return b;
}

int main() {
    Bar bar;
    int z = foo(bar.x);
    bar.y = z;
    return 0;
}
```

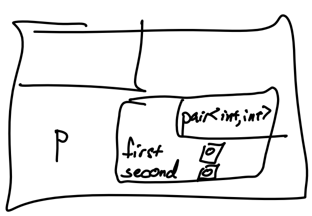
Show stack diagram







pair<int, int> p(0,0);



Function allLengthBFS(graph, src): $O(|E|)$

$O(1)$ ~~~ Initialization
~~~

While frontier is not empty: $\leftarrow O(|V|)$

current ← frontier.remove()

$*$

For each neighbor of current

If ~~~

$O(1) \rightarrow$ ~~~ Accounting

$O(|V|)$

EndIf

$=$

EndFor

$O(|V|^2)$

EndWhile

$O(|E|)$

Return ~~~

EndFunction

Dijkstra's: Same argument, but If block takes $O(\log |V|)$

$O(|E| \cdot \log |V|)$

insert into PQ

$$O(|E| \log |E|) \leq O(|E| \log |V|^2)$$
$$= O(|E|\ 2 \cdot \log |V|)$$
$$= O(|E| \cdot \log |V|)$$
~~~

thing | thing thing

thing | thing | thing | thing thing

0 1
0 2
2
3
0
4
4