Graph Theory

ian.mcloughlin@gmit.ie

Topics

Fundamentals

Seven Bridges of Königsberg

Is it possible to walk through the city crossing each of the seven bridges once and only once?

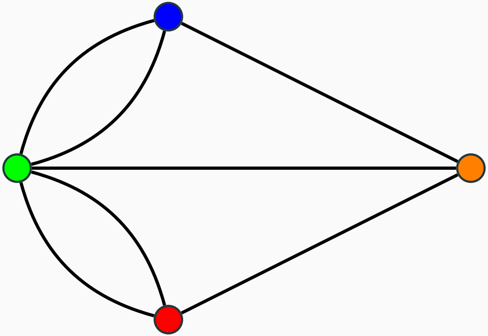
www.nature.com/nbt/journal/v29/n11

Leonhard Euler

- Born 1707 in Basel, Switzerland.
- Euler's identity: $e^{i\pi} + 1 = 0$.
- Solved the Bridges of Königsberg problem.
- It's not possible to cross all bridges once and once only.

https://en.wikipedia.org/wiki/Leonhard_Euler

Graph of Königsberg



Graph definition

Definition

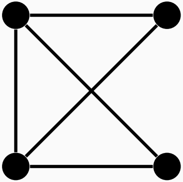
A *graph* consists of a finite set V and a set E of 2-subsets of V.

Vertices – the elements of the set V are called vertices.

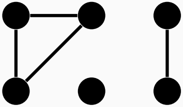
Edges – the elements of E are called edges.

$G = (V, E)$ – this is the way we write the graph G consists of the vertex set V and the edge set E.

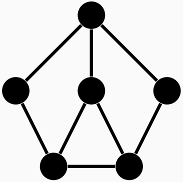
Example graphs



4 vertices, 5 edges



6 vertices, 4 edges



6 vertices, 8 edges

1 vertex, 0 edges

Sets of Königsberg

$$V = \{Green, Blue, Orange, Red\}$$

$$E = \{\{Green, Blue\}, \{Green, Blue\}, \{Green, Red\}, \{Green, Red\}, \{Blue, Orange\}, \{Green, Orange\}, \{Red, Orange\}\}$$

Not a graph by our definition

Note that the Bridges of Königsberg graph above is not a graph, due to the repeated edges. It's a multi-graph.

Adjanceny list

Green	Blue	Orange	Red
Blue	Green	Blue	Green
Orange	Orange	Green	Orange
Red		Red	

Our definition of a graph

The definition given above for a graph is not consistent with looped edges, directed edges or repeated edges. We only need to make small changes to the definition of a graph to allow for directed edges and repeated edges.

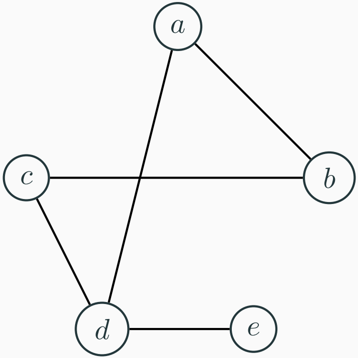
Repeated edges are edges that start and end at the same vertices.

Directed edges are edges where a direction is added.

Looped edges begin and end at the same vertex.

The application will determine the definition we want to use.

A better example

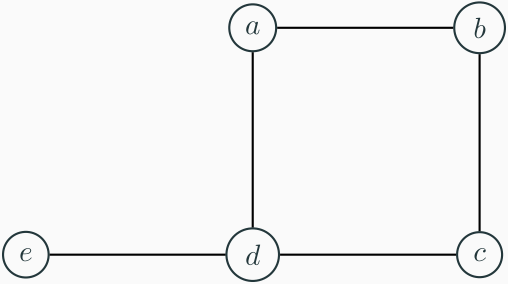


Exercise

Determine the vertex set, edge set and adjacency list of this graph.

global.oup.com/booksites/content/9780198507185/

Another better example

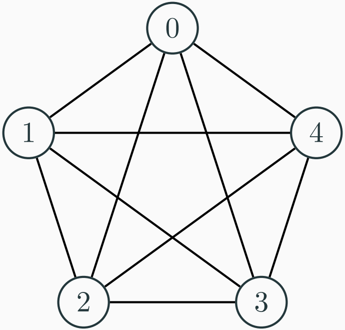


Exercise

Determine the vertex set, edge set and adjacency list of this graph.

global.oup.com/booksites/content/9780198507185/

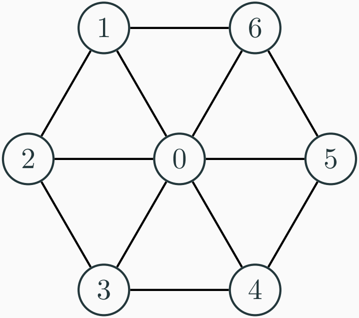
Complete graph K_n



Exercise

Determine the vertex set, edge set and adjacency list of K_5.

Wheel graph W_n



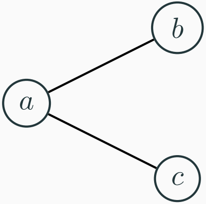
Exercise

Determine the vertex set, edge set and adjacency list of W_6.

Degree of a vertex

Definition

The degree of a vertex is the number of edges that contain it.



The degree of the vertex a is 2.

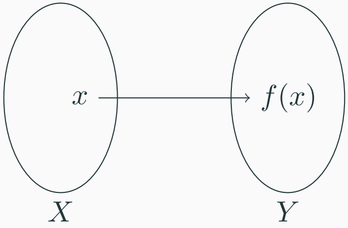
Exercise

For each of the vertices on the previous slide, determine its degree.

Functions

Definition

Suppose that X and Y are sets. We say we have a function f from X to Y if for each x in X we can specify a unique element in Y, which we denote by $f(x)$.

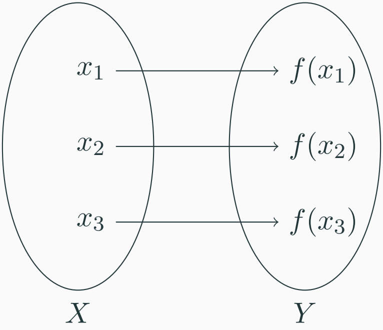


Bijections

Definition

A bijection is function f from a set X to a set Y where both of the following are true:

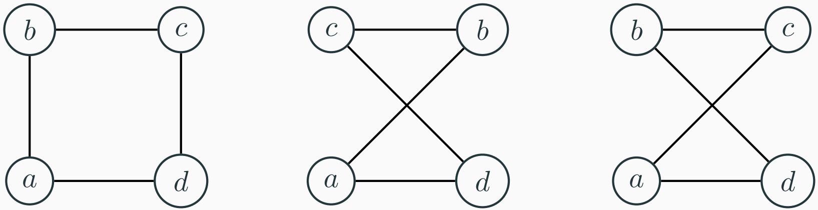
- every y in Y is a value $f(x)$ for at most one x in X.
- every y in Y is a value $f(x)$ for at least one x in X.



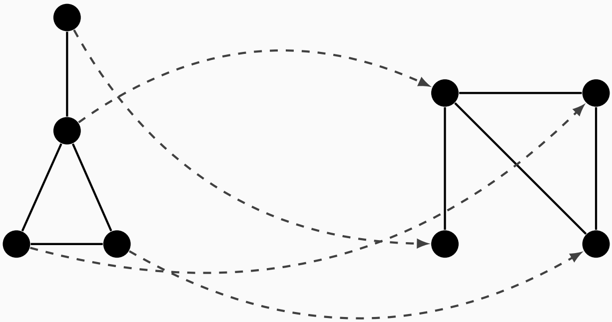
Isomorphism

Definition

Two graphs G_1 and G_2 are said to be isomorphic when there is a bijection α for the vertex set V_1 of G_1 to the vertex set V_2 of G_2 such that $\{\alpha(x), \alpha(y)\}$ is an edge of G_2 if and only if (x, y) is an edge of G_1.



Isomorphism example

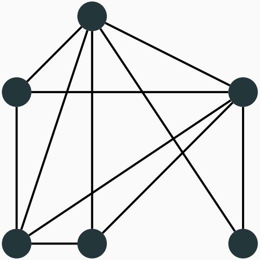


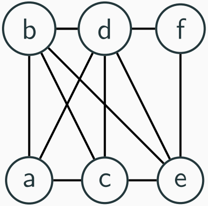
$$G_1 \xrightarrow[f]{} G_2$$

Isomorphism: degrees

Exercise

Determine if these two graphs are isomorphic.





Sum of degrees

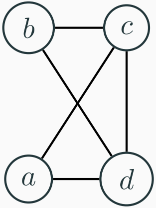
Theorem

The sum of the degrees of the vertices of a graph $G = (V, E)$ is equal to twice the number of edges:

$$\sum_{\mathrm{v} \in V} \delta(\mathrm{v}) = 2|E|$$

Proof.

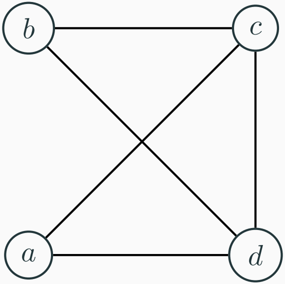
The degree $\delta(\mathrm{v})$ of a vertex v is equal to the number of edges incident on it. Every edge is incident on two vertices. So every edge contributes 1 to the degrees of two distinct vertices. Therefore every edge contributes 2 to the sum total of the degrees of all the vertices. □



Handshaking lemma

Definition

A vertex is an odd vertex if its degree is odd, and it is an even vertex if its degree is even. The set of all odd vertices is denoted V_o and the set of all even vertices is denoted V_e.



Exercise

Which of the above vertices are even, and which are odd?

Handshaking lemma

Lemma

The number of odd vertices $|V_o|$ in a graph is even.

Proof.

The sets V_o and V_e are disjoint (i.e. they don't have any elements in common.) Also, every vertex is either in V_o or V_e. Therefore $V = V_o \cup V_e$ and $|V| = |V_o| + |V_e|$.

Furthermore:

$$\sum_{\mathrm{v} \in V_o} \delta(\mathrm{v}) + \sum_{\mathrm{v} \in V_e} \delta(\mathrm{v}) = 2|E|$$

Both $2|E|$ and $\sum_{\mathrm{v} \in V_e} \delta(\mathrm{v})$ are even, so $\sum_{\mathrm{v} \in V_o} \delta(\mathrm{v})$ must be. Since $\delta(\mathrm{v})$ is odd for every v in V_o, this must mean that $|V_o|$ is even. □

Directed graph definition

Definition

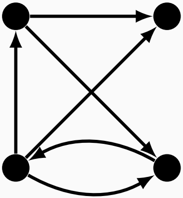
A *directed graph* consists of a finite set V and a set E of 2-tuples (ordered pairs of elements) from V.

Looped edges are allowed in this definition. A single one per vertex.

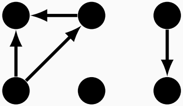
Multiple edges between the same start and end vertices are not allowed. However, two edges are allowed between every pair of vertices so long as they have opposite directions.

Direct edges use round brackets rather than curly braces: $E = \{(a, b) \mid a, b \in V\}$.

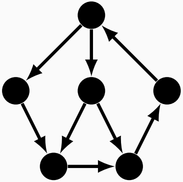
Example directed graphs



4 vertices, 6 directed edges



6 vertices, 4 directed edges



6 vertices, 8 directed edges

1 vertex, 0 directed edges

Multigraph definition

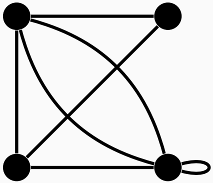
Definition

A *multigraph* consists of a finite set V and a multiset E of 2-multisubsets from V.

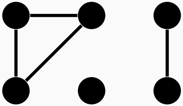
Multisets are like sets, but the same element can be in the set more than once.

Directed multigraphs are similar, but E is a set of 2-tuples of elements in V.

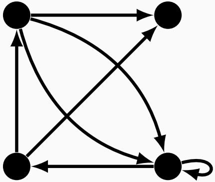
Example multigraphs



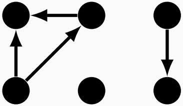
Multigraph



Still a multigraph



Directed multigraph



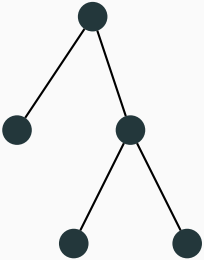
Still a directed multigraph

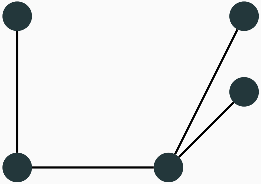
Trees

Definition

Tree

A *tree* is a graph where every pair of vertices has a path between them, and there are no cycles.





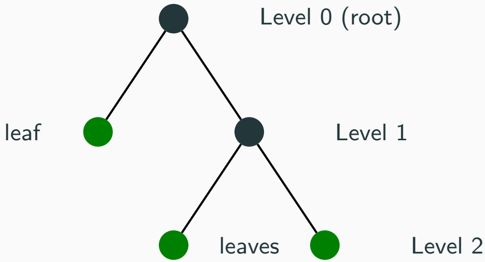
Rooted trees

Any vertex of a tree can be called its root.

Levels Root is at level 0, neighbours of the root are at level 1, their other neighbours at level 2, and so on.

Height of a tree is h, where there's vertex at level h but not at level $h+1$.

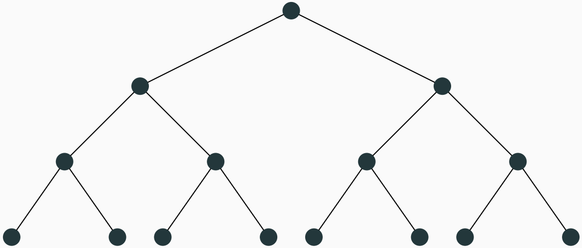
Leaf Vertex at level i not connected to a vertex at level $i+1$.



m-ary Rooted Tree

Definition

When a vertex at level i is connected to a vertex at level $i + 1$ it's common to call the former the *parent* and the latter the *child*. A rooted tree is m-ary if every parent has the same number of children. A 2-ary rooted tree is called a *binary tree*.



Isomorphic Rooted Trees

Definition
Two rooted trees are said to be *isomorphic* if there is a graph isomorphism between them which takes the root of one tree to the root of the other.

Logarithms

We define $\log$ in the following way:

$$m^h = l \Leftrightarrow \log_m l = h$$

What does log mean?

Suppose we have two numbers m and h and we ask the question "what is m to the power of h?" Let's call the answer l, so $l = m^h$.

The $\log$ function asks the inverse question: "what do we need to raise m to the power of to get l?" The answer is h.

For example, $10^2 = 100$ so $\log_{10} 100 = 2$. The subscript 10 is called the *base*.

Heights and leaves of m-ary rooted trees

Theorem

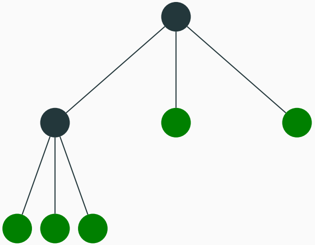
The height h of an m-ary rooted tree with l leaves is at least $\log_m l$. That is: $h \geq \log_m l$.

Proof.

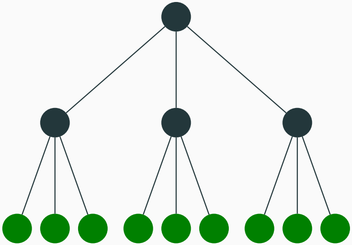
Note: $h \geq log_m l \Leftrightarrow m^h \geq m^{log_m l} \Leftrightarrow m^h \geq l$. So we'll just show l is at most m^h. For a tree of height 0, $l = 1$ and $m^0 = 1$ so $m^h \geq l$. Now assume the theorem is true for trees of height $i - 1$. Consider a tree of height i with l leaves. We can create m trees of height $i - 1$ from it by deleting the root. Each of these smaller trees has at most m^{h-1} leaves by assumption. There are m of these, so the big tree has at most $m \times m^{h-1} = m^h$ leaves. □

Examples of heights and leaves of m-ary trees

$m = 3$, $h = 2$, $l = 5$

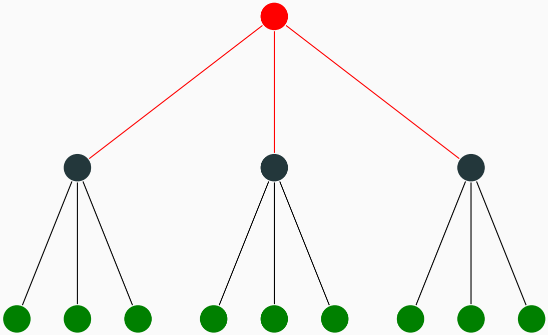


$m = 3$, $h = 2$, $l = 9$



Deleting the root of an m-ary tree

m smaller trees of height $h-1$



Spanning trees

Subgraph

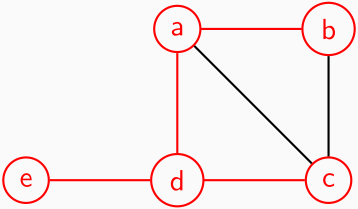
A *subgraph* $H = (V_H, E_H)$ of a graph $G = (V, E)$ is a graph such that V_H is a subset of V, E_H is a subset of E, and no edge in E_H contains a vertex not in V_H.

Spanning Tree

A *spanning tree* T of a connected graph G is a subgraph of G such that:

- the vertex set of T is the vertex set of G and
- T is a tree.

Spanning tree example



Spanning tree in red.

Graph databases

Digraph definition

Definition

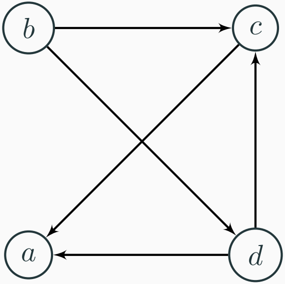
A *digraph* (short for directional graph) consists of a finite set V and a set E of ordered pairs of elements of V.

Degrees of vertices can now be split into in-degrees and out-degrees.

Walks, paths, cycles must be redefined.

Loops are allowed in the above definition, unless we rule them out.

Digraph example



Neo4j

- Neo4j is an open-source NoSQL graph database implemented in Java and Scala.
- Development started in 2003, it has been publicly available since 2007
- Available on GitHub.
- A graph is composed of two elements: a node and a relationship.

http://neo4j.com/developer/graph-database/

Cypher

- Cypher is a declarative graph query language.
- What to retrieve from a graph, not on how to retrieve it.
- Allows for expressive and efficient querying and updating of the graph store.
- Cypher borrows its structure from SQL.

http://neo4j.com/docs/stable/cypher-introduction.html

Cypher: Nodes

Create a node with the label User, and two properties:

```
CREATE (user:User { Id: 123, Name: "Jim" });
```

Find the node(s) with label User and their Id property being 123:

```
MATCH (user:User)
WHERE user.Id = 123
RETURN user;
```

neo4j.com/docs/stable/cypher-query-lang.html

Cypher: Relationships

Create a relationship with label FOLLOWS from user(s) with Id 123 to user(s) with Id 456:

```
MATCH (user1:User), (user2:User)
WHERE user1.Id = 123 AND user2.Id = 456
CREATE user1-[:FOLLOWS]->user2;
```

neo4j.com/docs/stable/cypher-query-lang.html

Create a relationship with label INVITED from user(s) with Id 123 to a new user with Id 789 and Name Jack:

```
MATCH (invitee:User)
WHERE invitee.Id = 123
CREATE invitee-[:INVITED]->(invited:User {Id: 789,
                                          Name: "Jack"});
```

neo4j.com/docs/stable/cypher-query-lang.html

Cypher: DELETE

Delete all nodes:

```
MATCH (x)
DELETE x;
```

neo4j.com/docs/stable/cypher-query-lang.html

Labels and properties

- Suppose we have nodes representing people.
- We give them the label People.
- We also want to identify each person as either Male or Female.
- Should we use Male and Female labels, or a Gender property?
- If you are going to use the person's gender in a lot of queries, a normal property will be relatively slow, so you should use a label.
- However, you can also index some of your properties to high-light them as important.

neo4j.com/docs/stable/cypher-query-lang.html

Cypher: shortestPath

Find the minimum number of hops between two nodes.

```
MATCH p=shortestPath(
  (a:Actor {id: 1})-[*]-(b:Actor {id: 10})
  )
RETURN p;
```

neo4j.com/docs/stable/cypher-query-lang.html

Algorithms

Decision tree

A decision tree is a rooted tree where each vertex represents a decision.

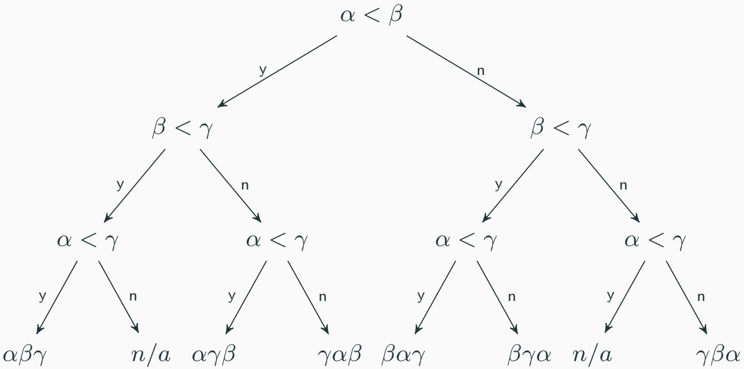
The results of a decision are represented by the edges connecting the vertex to the vertices at the next level down.

Decisions can connect to other decisions further down the tree.

Final outcomes of the procedure are represented by the leaves.

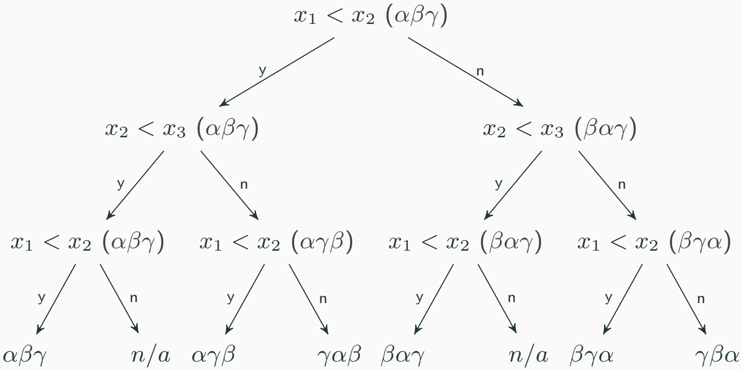
Decision tree for sorting three items

Three items – (α, β, γ)



Decision tree for bubble sort with three items

Three items – (α, β, γ)



Heap sort

Heap sort uses a tree as part of the algorithm.

This tree is not a decision tree, it is for another purpose.

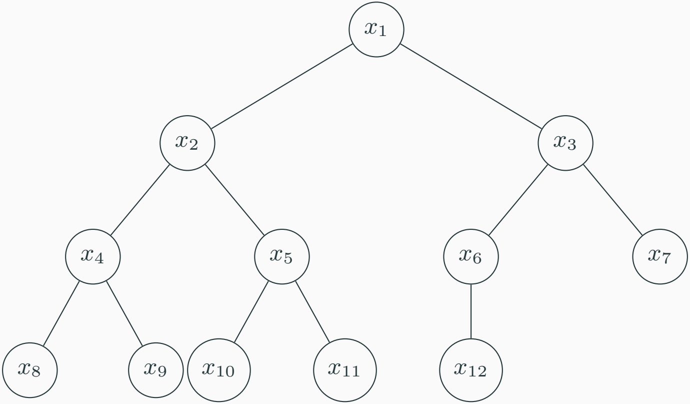
Different steps in the algorithm manipulate the tree.

Worst case – performs better than quick sort.

They have the same average performance though.

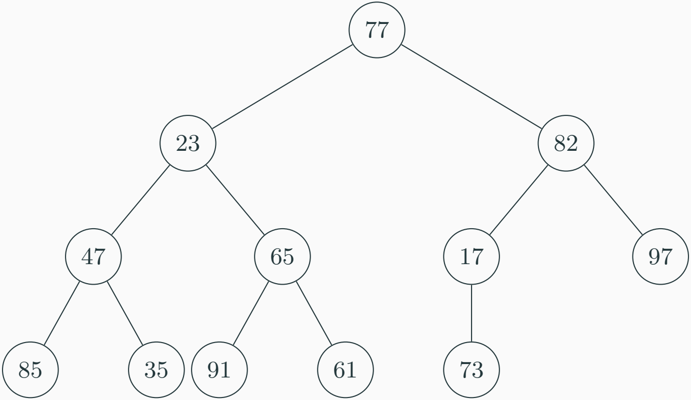
Heapsort initial tree

$$(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12})$$



Heapsort initial tree

$$(77, 23, 82, 47, 65, 17, 97, 85, 35, 91, 61, 73)$$



Transforming to a heap

Suppose the trees at x_{2r} and x_{2r+1} are already heaps.

The vertex x_r is their parent.

Compare x_r to x_{2r} and x_{2r+1}.

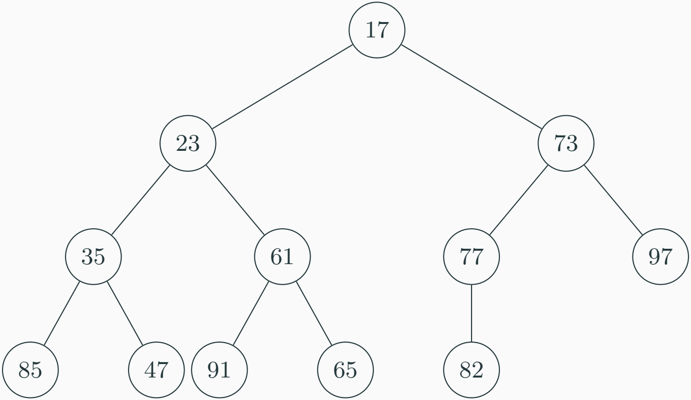
If x_r is smaller, do nothing.

Otherwise replace x_r with the smaller of it's children and fill the new vacancy with the smaller of x_r and the two children, and repeat if necessary.

Start this procedure at the last parent, and move backwards through the other parents.

Heapsort - the heap

$$(17, 23, 73, 35, 61, 77, 97, 85, 47, 91, 65, 82)$$



$x_r < x_{2r}$ and $x_r < x_{2r+1}$

Transforming to a sorted list

Start with a new empty list.

Place the root of the heap at the end of the list.

Remove the last leaf and place it at the root.

Transform the tree to a heap again. This is relatively easy since the subtrees at x_2 and x_3 are already heaps.

Repeat from step 2.

$$(17, 23, 35, 47, 61, 65, 73, 77, 82, 85, 91, 97)$$