	.	-	_		. .		-	-			_		
Wei	ahte	d ai	rant	ns a	nd t	heir	im	hlen	nent	atio	ns i	n C	++
	gine	u gi	upi	15 U		non			10III	auo	110 1		

Q: How does a weighted graph differ from an ordinary graph?

A: In a weighted graph each edge (x,y) has associated with it a weight w((x,y)), which is a number.

C	Comm	on in	terpre	tation	ns of	weiah	ts
ų v			icipie	FLALIO	13 01	weigi	13

- graph represents a road map and nodes are locations or intersections on the map: w((x,y)) can be the distance from x to y or the time required to get from x to y

- graph represents a project and the nodes are different tasks in the project: w((x,y)) is the time it takes to complete the task x

 graph represents a pipe network under construction and the nodes represent junctions in the pipe network: w((x,y)) can be the cost of constructing the pipe from x to y

Common use

- find the shortest/fastest/cheapest path from some node to some other node

- BFS assumes w((x,y)) = 1 for all edges

Goal: shortest path from <i>F</i> to <i>H</i> .	
One BFS solution: <i><f, d,="" h=""></f,></i>	2 5 H
True solution: <i><f, a,="" b,="" c,="" h=""></f,></i>	F 0 3 7

or each	node <i>y</i> adjac	ent to <i>x</i> , st	ore pair	<y, th="" w((x,y)<=""><th>)) ></th><th></th><th></th><th></th></y,>)) >			
struct No	ode							
۲ // All t	he data store	d in the no	de					
int id;								
Std::St	ring name;							
std::ve	ector <std::pai< td=""><td>r<node*,c< td=""><td>ost>> to</td><td>_neighboi</td><td>urs;</td><td></td><td></td><td></td></node*,c<></td></std::pai<>	r <node*,c< td=""><td>ost>> to</td><td>_neighboi</td><td>urs;</td><td></td><td></td><td></td></node*,c<>	ost>> to	_neighboi	urs;			
, 1								
Suitable v	when							
only nee	ed to move fo	rward alon	a edaes					
edges a	dded or delet	ed infrequ	ently	waight				
edges a only dat	dded or delet a associated	ed infreque with edge	ently is cost (weight)				
edges a only dat	dded or delet a associated	ed infreque with edge	ently is cost (weight)				
edges a only dat <u>Impler</u>	dded or delet a associated nentation 2	ed infreque with edge	ently is cost (weight)				
edges a only dat	dded or delet a associated nentation 2	ed infreque with edge	ently is cost (weight)				
edges a only dat <u>Impler</u>	dded or delet a associated nentation 2	ed infreque with edge	ently is cost (weight)				
edges a only dat <u>Impler</u> struct	dded or delet a associated nentation 2	ed infreque with edge	ently is cost (weight)				
edges a only dat <u>Impler</u> struct { // Al	dded or delet a associated nentation 2 Node I the data sto	red infrequences	ently is cost (weight)				
edges a only dat <u>Impler</u> struct { // Al int id std::	dded or delet a associated mentation 2 Node I the data sto d; string name;	red infrequences	ently is cost (node	weight)				
edges a only dat <u>Impler</u> struct { // Al int id std:: //	dded or delet a associated mentation 2 Node I the data sto d; string name;	red infrequences	ently is cost (node	weight)				
edges a only dat Impler struct { // Al int id std:: //	dded or delet a associated nentation 2 Node I the data sto d; string name; map, not set!	red infrequences	node	weight)				
edges a only dat Impler struct { { // Al int id std:: // std::	dded or delet a associated mentation 2 Node I the data sto d; string name; map, not set!	red infrequences with edge	node	veight)	ours;			
edges a only dat Impler struct { { // Al int id std:: // std:: };	dded or delet a associated mentation 2 Node I the data sto d; string name; map, not set!	red infrequences with edge	node	veight)	ours;			
edges a only dat Impler struct { // Al int id std:: // std:: };	dded or delet a associated mentation 2 Node I the data sto d; string name; map, not set!	red infrequences with edge	node	veight)	ours;			
edges a only dat <u>Impler</u> struct { { // Al int id std:: // std:: };	dded or delet a associated <u>mentation 2</u> Node I the data sto d; string name; map, not set! unordered_m	red infrequences with edge	node	veight)	ours;			
edges a only dat Impler struct { // Al int ic std:: // std:: }; Suitable ·	dded or delet a associated nentation 2 Node I the data sto d; string name; map, not set! unordered_m when	red infrequences with edge	node	veight)	Durs;			

Implementation	3					
struct Edge						
int cost; string name:						
//						
};						
struct Node						
// All the data	stored in th	e node				
std::string na	me;					
//						
//map, not	set!					
std::unordere ب	ed_map <noc< th=""><th>le*,Edge> to</th><th>_neighbours</th><th>3;</th><th></th><th></th></noc<>	le*,Edge> to	_neighbours	3;		
<u>/</u> ,						
Suitable when						
only need to mov	e forward al	ong edges				
edges added or o	deleted frequ	uently				
	la associate	u with edge				

_															
<u>Imp</u>	lementat	<u>ion 4</u>													
	In undirec	ted gra	phs, e	gde (data	can b	be sh	ared	oetwe	en di	rectio	ns		_	
st	ruct Edge														
1	int cost:														
	string nar	me;													
	Hugedata	a data;													
	// too n	nuch da	ata or o	chang	ging	data									
};															
ct	uct Node														
{															
<u> </u>	// All the	data sto	ored in	the I	node	;									
	int id;														
	std::string	g name;	;												
	//														
	//map.	not set	!												
	std::unor	dered_r	map <n< td=""><td>lode'</td><td>*,std</td><td>::shai</td><td>red_p</td><td>tr<ed< td=""><td>ge>></td><td>to_ne</td><td>eighb</td><td>ours;</td><td></td><td></td><td></td></ed<></td></n<>	lode'	*,std	::shai	red_p	tr <ed< td=""><td>ge>></td><td>to_ne</td><td>eighb</td><td>ours;</td><td></td><td></td><td></td></ed<>	ge>>	to_ne	eighb	ours;			
};															
Sullar	ble when														
- only	need to n	nove fo	rward	alond	a edo	aes									
- edge	es added	or delet	ted free	quen	tly										
there	e is much	data as	ssociat	ed w	ith e	dge									
the g	graph is u	ndirecte	ed and	we	do no	ot wis	sh to k	ceep t	wo co	pies	of the	e sam	e edge	e	
							1								

