Sets maps dict

Set in C++ Standard Template Library (STL)

- begin() – Returns an iterator to the first element in the set.
-
- end() – Returns an iterator to the theoretical element that follows last element in the set.
-
- size() – Returns the number of elements in the set.
-
- max_size() – Returns the maximum number of elements that the set can hold.
-
- empty() – Returns whether the set is empty.
-
-
-
-
-
-
-
- rbegin()– Returns a reverse iterator pointing to the last element in the container.
-
-
-
- rend()– Returns a reverse iterator pointing to the theoretical element right before the first element in the set container.
-
-
-
-

- crbegin()– Returns a constant iterator pointing to the last element in the container.
- crend() – Returns a constant iterator pointing to the position just before the first element in the container.
- cbegin()– Returns a constant iterator pointing to the first element in the container.
- cend() – Returns a constant iterator pointing to the position past the last element in the container.

- insert(const g) – Adds a new element ‘g’ to the set.
- erase(iterator position) – Removes the element at the position pointed by the iterator.
- erase(const g)– Removes the value ‘g’ from the set.
- clear() – Removes all the elements from the set.

The Dictionary ADT

As an ADT, a (unordered) dictionary D supports the following functions:

size(): Return the number of entries in D.

empty(): Return true if D is empty and false otherwise.

find(k): If D contains an entry with key equal to k, then return an iterator p referring any such entry, else return the special iterator end.

findAll(k): Return a pair of iterators (b,e), such that all the entries with key value k lie in the range from b up to, but not including, e.

insert(k,v): Insert an entry with key k and value v into D, returning an iterator referring to the newly created entry.

erase(k): Remove from D an arbitrary entry with key equal to k; an error condition occurs if D has no such entry.

erase(p): Remove from D the entry referenced by iterator p; an error condition occurs if p points to the end sentinel.

begin(): Return an iterator to the first entry of D.

end(): Return an iterator to a position just beyond the end of D.

Example

Operation	*Output*	*Dictionary*
empty()	**true**	$\emptyset$
insert(5,A)	$p_1 : [(5,A)]$	$\{(5,A)\}$
insert(7,B)	$p_2 : [(7,B)$	$\{(5,A),(7,B)\}$
insert(2,C)	$p_3 : [(2,C)$	$\{(5,A),(7,B),(2,C)\}$
insert(8,D)	$p_4 : [(8,D)$	$\{(5,A),(7,B),(2,C),(8,D)\}$
insert(2,E)	$p_5 : [(2,E)$	$\{(5,A),(7,B),(2,C),(2,E),(8,D)\}$
find(7)	$p_2 : [(7,B)$	$\{(5,A),(7,B),(2,C),(2,E),(8,D)\}$
find(4)	end	$\{(5,A),(7,B),(2,C),(2,E),(8,D)\}$
find(2)	$p_3 : [(2,C)$	$\{(5,A),(7,B),(2,C),(2,E),(8,D)\}$
findAll(2)	(p_3,p_4)	$\{(5,A),(7,B),(2,C),(2,E),(8,D)\}$
size()	5	$\{(5,A),(7,B),(2,C),(2,E),(8,D)\}$
erase(5)	–	$\{(7,B),(2,C),(2,E),(8,D)\}$
erase(p_3)	–	$\{(7,B),(2,E),(8,D)\}$
find(2)	$p_5 : [(2,E)]$	$\{(7,B),(2,E),(8,D)\}$

MAP ADT Functions

Some basic functions associated with Map:

- begin() – Returns an iterator to the first element in the map
- end() – Returns an iterator to the theoretical element that follows last element in the map
- size() – Returns the number of elements in the map
- max_size() – Returns the maximum number of elements that the map can hold
- empty() – Returns whether the map is empty
- pair insert(keyvalue, mapvalue) – Adds a new element to the map
- erase(iterator position) – Removes the element at the position pointed by the iterator
- clear() – Removes all the elements from the map

Courtesy:https://www.geeksforgeeks.org/map-associative-containers-the-c-standard-template-library-stl/

begin() function

- Syntax :

Mapname.begin()

- **Parameters** : No parameters are passed.
- **Returns** : This function returns a bidirectional iterator pointing to the first element.

erase()

- A built-in function in C++ STL which is used to erase element from the container.
- It can be used to **erase keys, elements** at any specified position or a given range.

Syntax :

- map_name.erase(key)

Parameters:

- The function accepts **one mandatory parameter *key*** which specifies the key to be erased in the map container.

Return Value:

- The function **returns 1 if the key element is found** in the map else returns 0.

size() function

- In C++, **size()** function is used to return the total number of elements present in the map.

Syntax:

- **map_name.size()**

Return Value: It returns the number of elements present in the map.

clear()

- clear() function is used to remove all the elements from the map container and thus leaving it's size 0.

Syntax:
map1.clear() where map1 is the name of the map.

Parameters:
No parameters are passed.

Return Value:
None

bits/stdc++.h

- In programming contests, people do focus more on finding the algorithm to solve a problem than on software engineering.
    - – From, software engineering perspective, it is a good idea to minimize the include
    - – If you use it actually includes a lot of files, which your program may not need, thus increases both compile time and program size unnecessarily.

- **Standard Template Library (STL)**

is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays,.

- It is a library of container classes, algorithms, and iterators.

container?

- container is a holder object that stores a collection of other objects (its elements).

Postlab

Comparison of Open Addressing Techniques-

	Linear Probing	Quadratic Probing	Double Hashing
Primary Clustering	Yes	No	No
Secondary Clustering	Yes	Yes	No
Number of Probe Sequence (m = size of table)	m	m	m^2
Cache performance	Best	Lies between the two	Poor

Sorting Algorithms	Special Input Condition	Time Complexity			Space Complexity
		Best Case	Average Case	Worst Case	Worst Case
Counting Sort	Each input element is an integer in the range 0- K	Ω(N + K)	Θ(N + K)	O(N + K)	O(K)
Radix Sort	Given n digit number in which each digit can take on up to K possible values	Ω(NK)	Θ(NK)	O(NK)	O(N + K)
Bucket Sort	Input is generated by the random process that distributes elements uniformly and independently over the interval [0, 1)	Ω(N + K)	Θ(N + K)	O(N^2)	O(N)

Set applicstion

Kruskals minimal spanning tree algo. Which in turn has many appplications like generating mazes of all kinds etc

It generate randoms trees

Map as ADT:

Consist of key value pairs

Each key must occur only once in the map

Values can be retrived from the keys

Values can be modified

Key value pair can be deleted

// value def

Abstract typedef< int value , char key, node*next> node

//Operator def :CREATE

Abstract void createNode(value , key)

Precondition :none postcondition: none

Put precondition: key is not alrealdey present

If present then update NODE->VALUE = VALUE

Key	BFS	DFS
Definition	BFS, stands for Breadth First Search.	DFS, stands for Depth First Search.
Data structure	BFS uses Queue to find the shortest path.	DFS uses Stack to find the shortest path.
Source	BFS is better when target is closer to Source.	DFS is better when target is far from source.
Suitablity for decision tree	As BFS considers all neighbour so it is not suitable for decision tree used in puzzle games.	DFS is more suitable for decision tree. As with one decision, we need to traverse further to augment the decision. If we reach the conclusion, we won.
Speed	BFS is slower than DFS.	DFS is faster than BFS.
Time Complexity	Time Complexity of BFS = O(V+E) where V is vertices and E is edges.	Time Complexity of DFS is also O(V+E) where V is vertices and E is edges.

1. Write the differences between linked list and linear array

Linear Array	Linked List
An array is a collection of elements of a similar data type.	A linked list is a collection of objects known as a node where node consists of two parts, i.e., data and address.
Array works with a static memory. Here static memory means that the memory size is fixed and cannot be changed at the run time.	The Linked list works with dynamic memory. Here, dynamic memory means that the memory size can be changed at the run time according to our requirements.
Array takes more time while performing any operation like insertion ,deletion, etc.	Linked list takes less time while performing any operation like insertion, deletion, etc.
In the case of an array, memory is allocated at compile-time.	In the case of a linked list, memory is allocated at run time.
Memory utilization is inefficient in an array	Memory utilization is inefficient in an array

LL APPLICATION: IMAGE VIEWR ..NEXT IMAGE IS LNKED TO PREVIOUS.

Backtracing using stack?

Backtracking can be defined as a general algorithmic technique that considers searching every possible combination in order to solve a computational problem.

RAT IN THE MAZE PROBLEM:

Q] A Maze is given as N*M binary matrix of blocks and there is a rat initially at (0, 0) ie. maze[0][0] and the rat wants to eat food which is present at some given block in the maze (fx, fy). In a maze matrix, 0 means that the block is a dead end and 1 means that the block can be used in the path from source to destination. The rat can move in any direction (not diagonally) to any block provided the block is not a dead end.

The task is to check if there exists any path so that the rat can reach the food or not. It is not needed to print the path.

Input : maze[4][5] = {
{1, 0, 1, 1, 0},
{1, 1, 1, 0, 1},
{0, 1, 0, 1, 1},
{1, 1, 1, 1, 1}
}
fx = 2, fy=3
Output : Path Found!
The path can be: (0, 0) -> (1, 0) -> (1, 1) -> (2, 1) -> (3, 1) -> (3, 2) -> (3, 3) -> (2, 3)

APPROACH

1. Initially, we will push a node with indexes i=0, j=0 and dir=0 into the stack.
2. We will move to all the direction of the topmost node one by one in an anti-clockwise manner and each time as we try out a new path we will push that node (block of the maze) in the stack.
3. We will increase *dir* variable of the topmost node each time so that we can try a new direction each time unless all the directions are explored ie. dir=4.
4. If dir equals to 4 we will pop that node from the stack that means we are retracting one step back to the path where we came from.
5. We will also maintain a visited matrix which will maintain which blocks of the maze are already used in the path or in other words present in the stack.
6. While trying out any direction we will also check if the block of the maze is not a dead end and is not out of the maze too.
7. We will do this while either the topmost node coordinates become equal to the food's coordinates that means we have reached the food or the stack becomes empty which means that there is no possible path to reach the food.

CALL STACK IN RECURSION

Recursive f uses call stack

When prog calla a f , that f goes on top of the call stack. Recursive calll will remain in the stack umrill the end of its execution

SINGLY LINKED LIST	DOUBLY LINKED LIST
The Singly linked list has two segments: data and link.	The doubly linked list has three segments. First is data and second, third are the

	pointers.
It permits traversal components only in one way.	It permits two-way traversal.
We mostly prefer a singly linked list for the execution of stacks.	We can use a doubly linked list to execute binary trees, heaps and stacks.
When we want to save memory and do not need to perform searching, we prefer a singly linked list.	In case of better implementation, while searching, we prefer a doubly linked list.
A singly linked list consumes less memory as compared to the doubly linked list.	The doubly linked list consumes more memory as compared to the singly linked list.