CS106B

Programming abstractions

Sopiko Kurdadze

Free University of Tbilisi  School of Mathematics And Computer Science

#include არის import-ის მსგავსი (იგივე) #include <iostream> (in/out-put-ის ხელმისაწვდომობისთვისაა საჭირო)

#include “console.h”

< something\_strange> C++ სისტემური ბიბლიოთეკა (stl)

“libraryName.h “ -ლოკალური პროექტის ბიბლიოთეკა

**using namespace std;**

**int main()**{

**return 0;**

}

**cout << „**სტრინგების**“** კონკატენაციისთვის გამოიყენება არა „+“ ოპერატორი, არამედ „<<“ -სისტემიდან კონსოლში დაბეჭდვა;

**endl-**ერთი ცალი მაინც უნდა იყოს;

**cin >>-** კონსოლიდან ინფორმაციის მიღება; მაგ: int age; cin >> age; ამ ცვლადს ვანიჭებთ წაკითხულ მნიშვნელობას; თუმცა არც ისე კარგი გზაა ეს. ჯობია:

**#include “simpio.h”-**stanford-ის ბიბლიოთეკიდან

getInteger();- int

getReal();- double

getLine();- string, line of text

getYesOrNo(); - bool

**ფუნქციის პროტოტიპი** გვაძლევს საშუალებას გამოვიყენოთ ეს ფუნქცია მანამდეც, ვიდრე მისი დეფინიცია წერია. რეკომენდირებულია მათი ჩამოწერა. იწერება main()- ფუნქციამდე.

**#include <cmath>**

abs(value); -რიცხვის მოდული

ceil(value);-დამრგვალება ზემოთ

floor(value);-დამრგვალება ქვემოთ

log10(value);-ლოგარითმი 10ის ფუძით

max(value1, value2);

min(value1, value2);

pow(base, exp);-ახარისხება base ადის exp ხარისხში

round(value);- უახლოეს მთელამდე დამრგვალება

sqrt(value);-კვადრატული ფესვი

sin(value); cos(value); tan(value);

ფუნქციის პარამეტრები:

* value-ს გადაცემა- კოპირდება ცვლადის, ობიექტის **მნიშვნელობები** f(int a)
* reference-ის გადაცემა- გადაეცემა თავად ცვლადი, ობიექტი f(int **&** a)

**Strings**

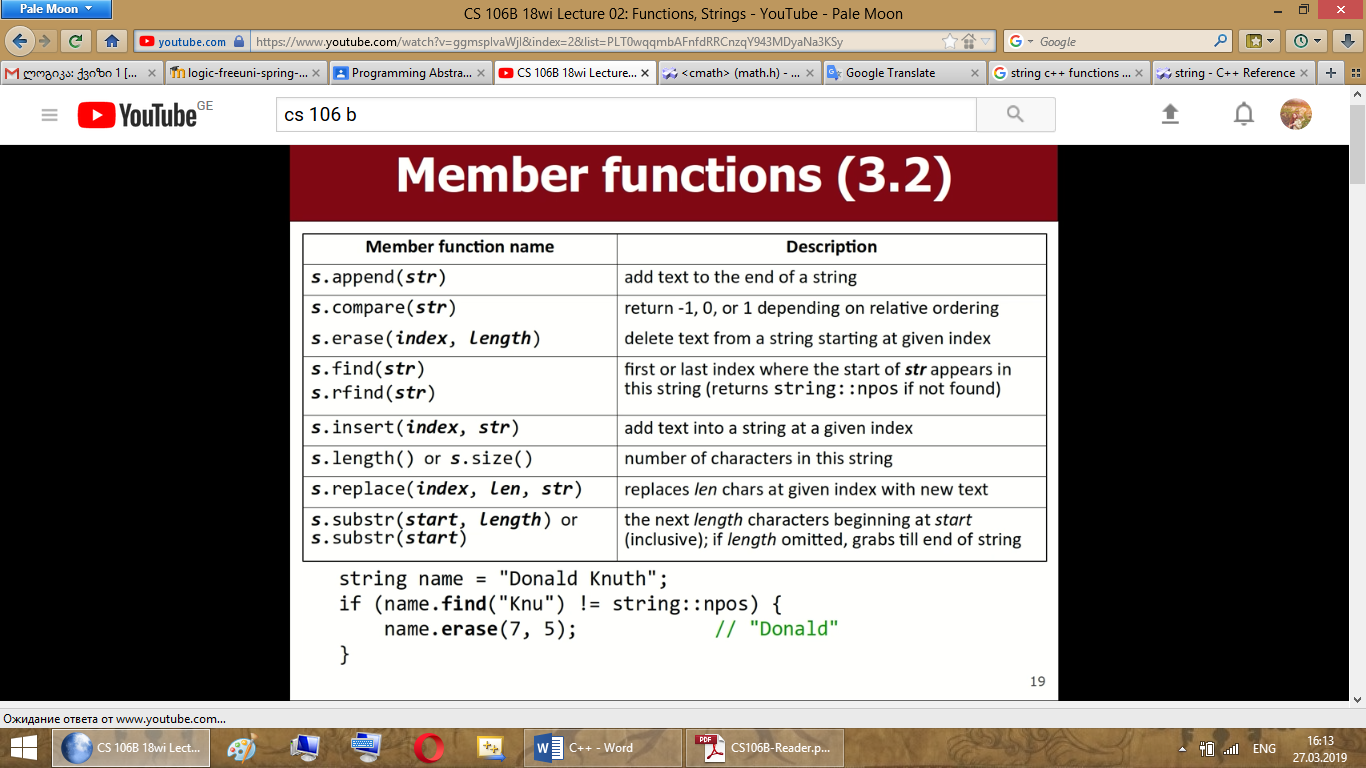
#include <string>

char c=str[index]; || char c=str.at(index);

სტრინგების კონკატენაცია „+“ ოპერატორით

სტრინგების შედარება s1==s2 ოპერატორით. (!=, < ,>, <=, >=) -ადარებს ASCII-ის მნიშვნელობას

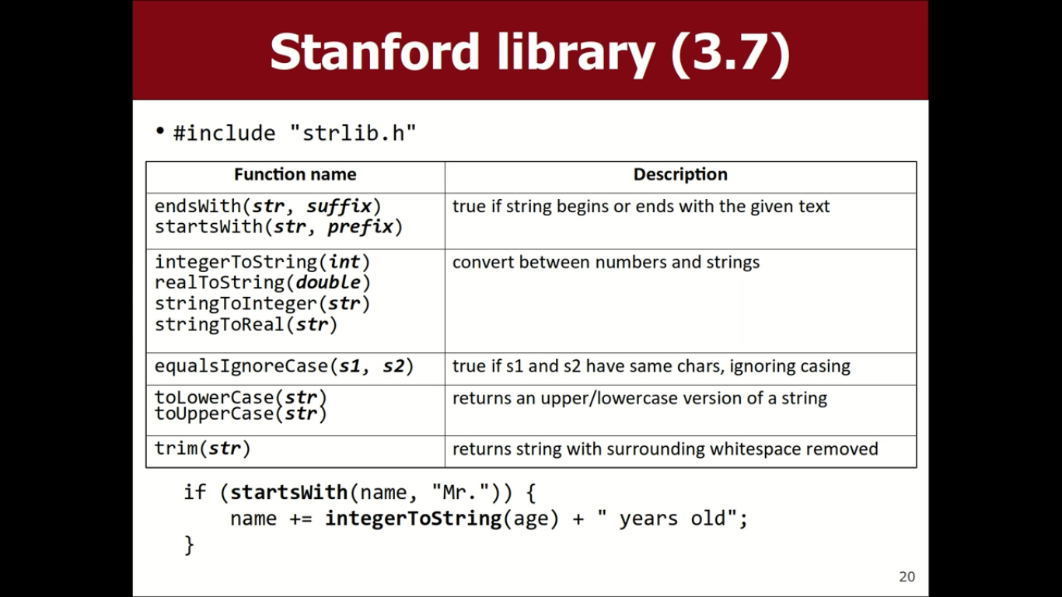
**mutable-**შესაძლებელია შეცვლა

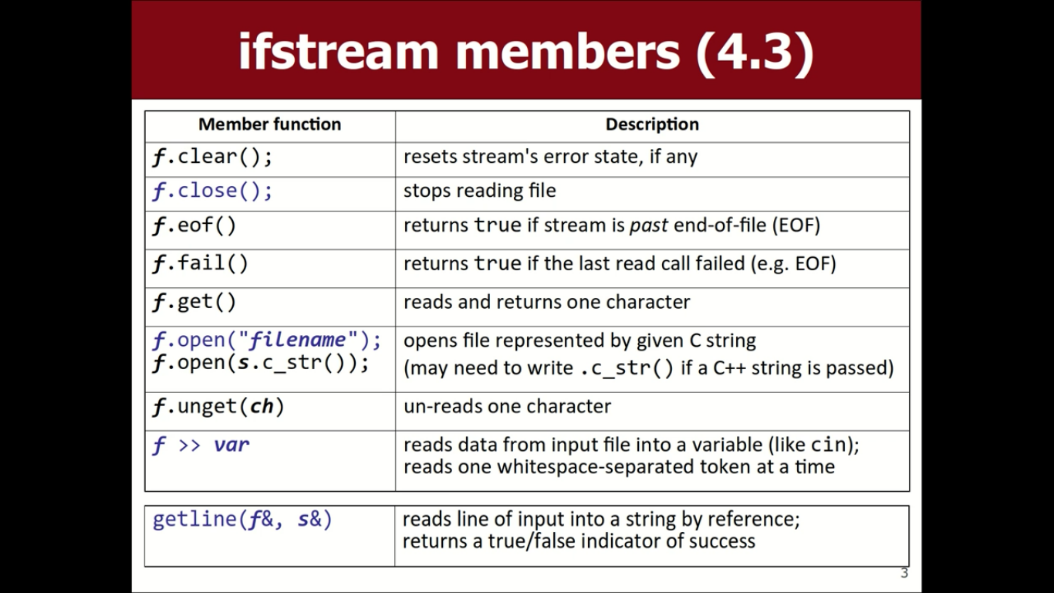


არსებობს ორი ტიპის სტრინგ: c string და c++ string. **„**C string-ების**“** კონკატენაცია არ შეიძლება + ოპერატორით.

C string +int/char გვაძლევს garbage-ს, bug-ს.

გამოსწორება შესაძლებელია C++ string-ად კონვერტაციით. მაგ: string s=string(“Hi”)+”Hello”.



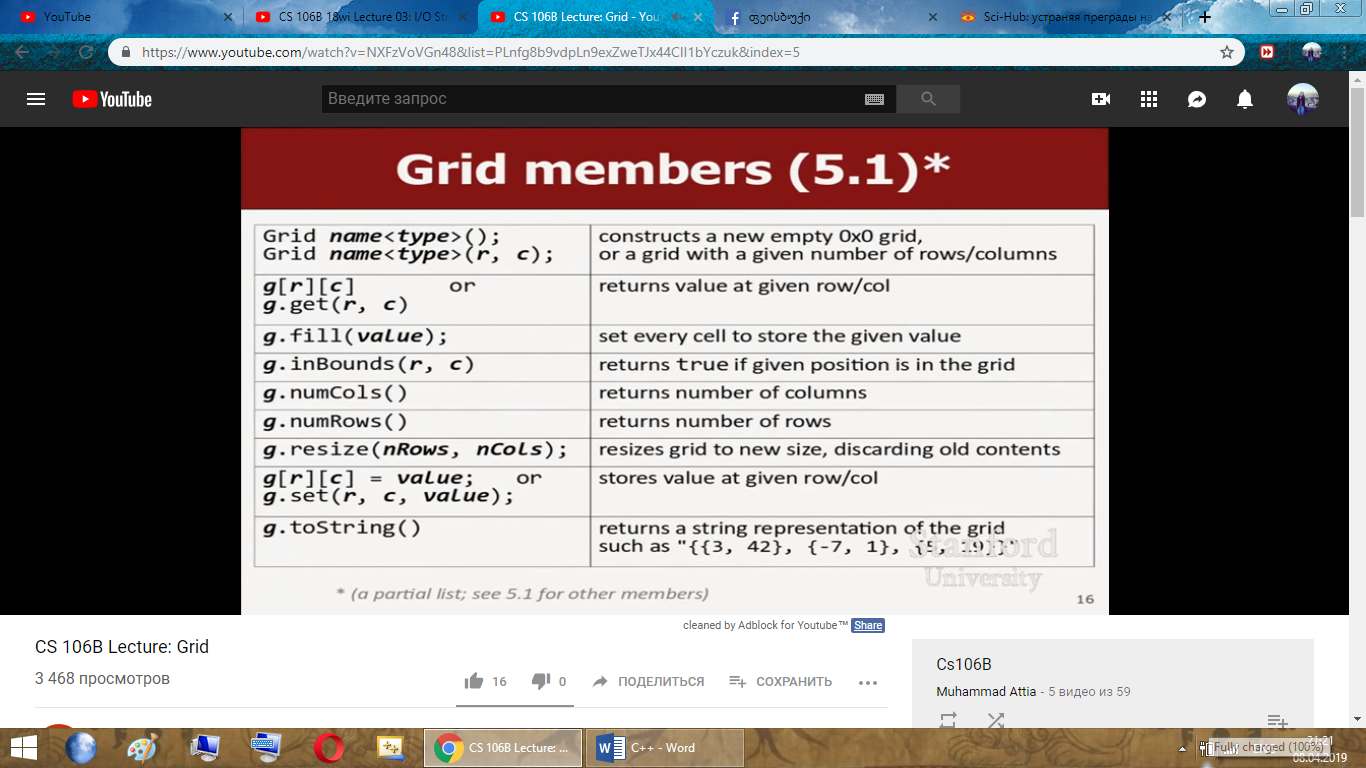
**ფაილიდან კითხვა:**

**#include <fstream>**

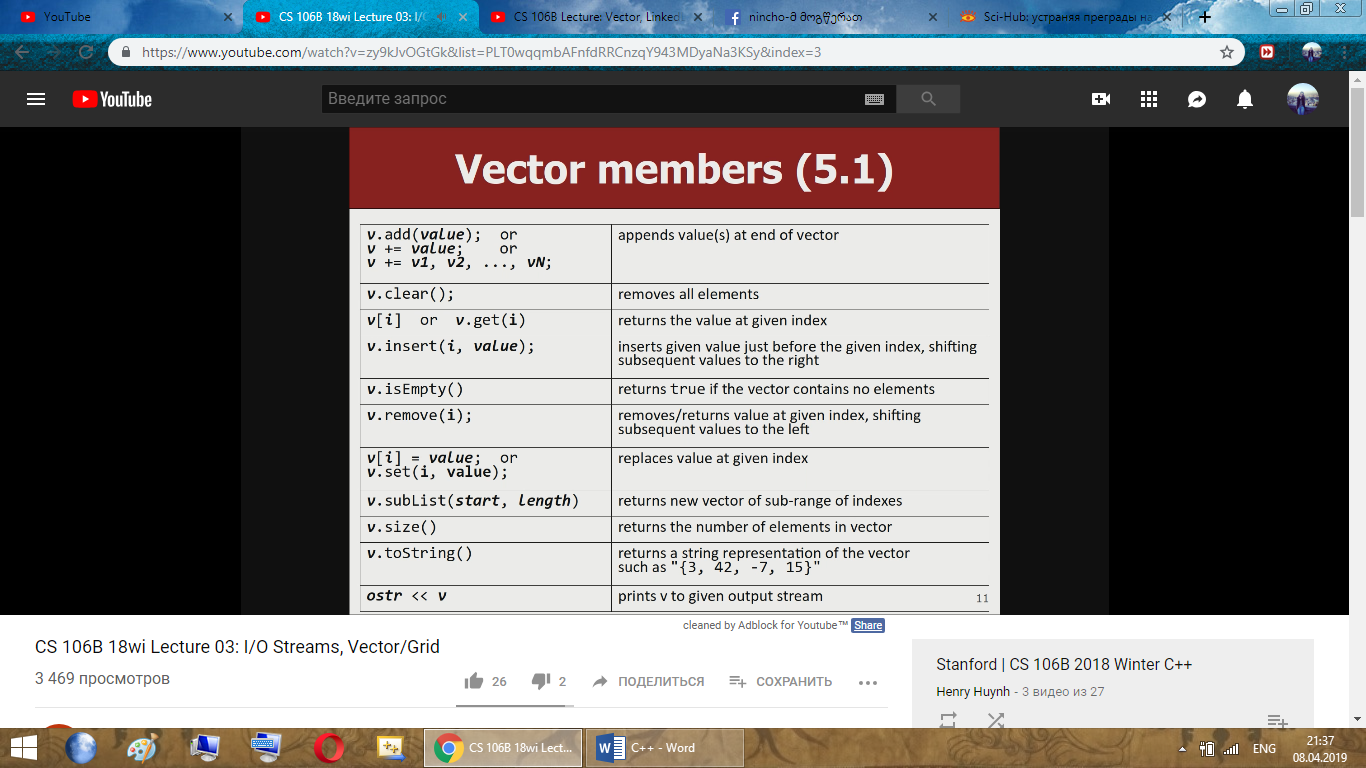
**Standard Template Library (STL)-** for using collections

Grid: #include “grid.h” (two dimensional array)

Grid <*type*> name(0,0);

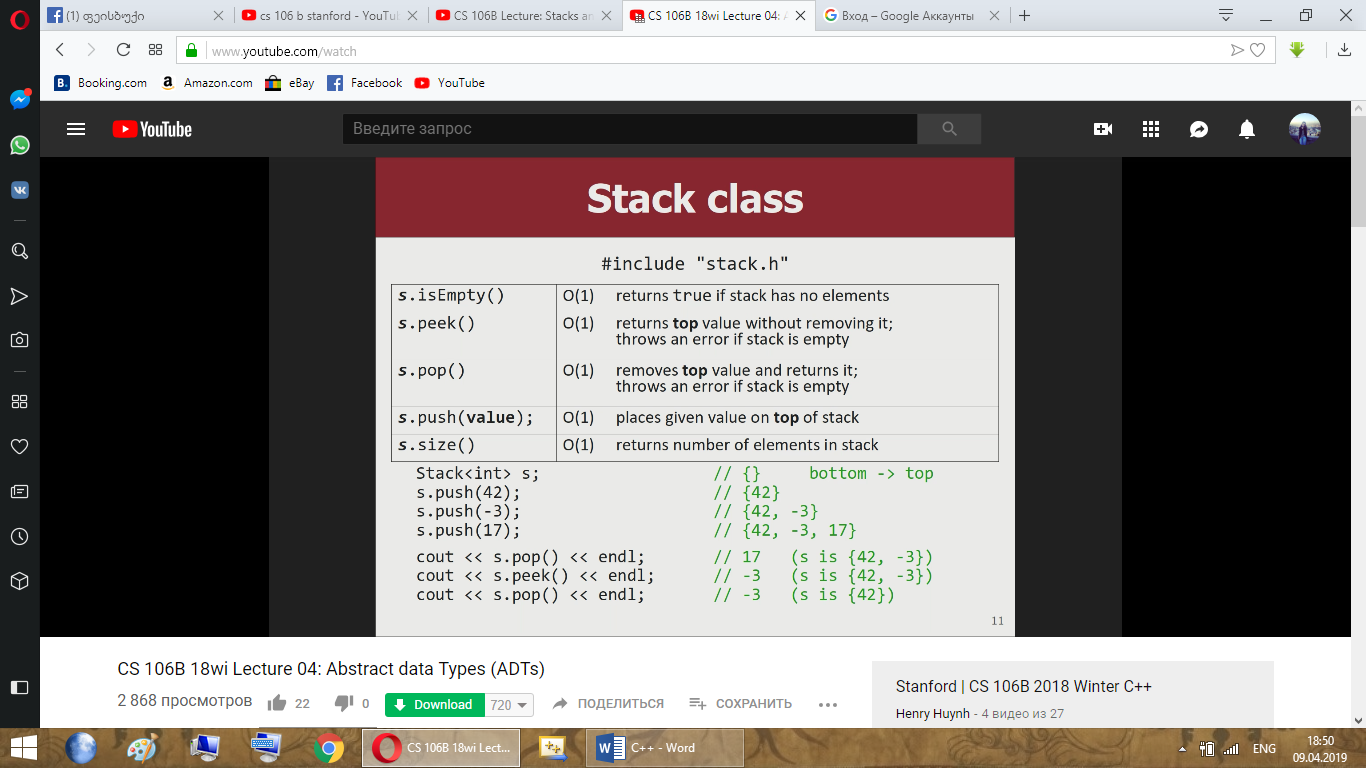


**Vector**: #include “vector.h” (like arrayList)

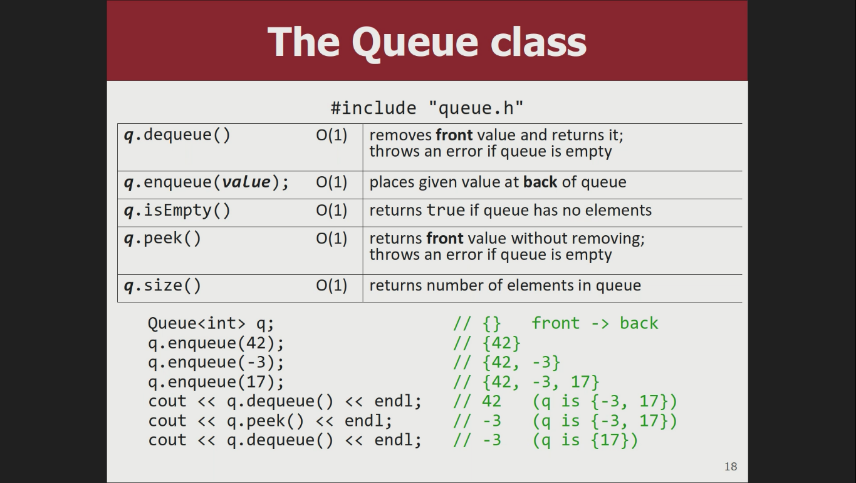
Vector<*type*> name;

|  |
| --- |
| **Methods** |
| [**size()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Vector-class.html#Method:size) | Returns the number of elements in this vector. |
| [**isEmpty()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Vector-class.html#Method:isEmpty) | Returns **true** if this vector contains no elements. |
| [**clear()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Vector-class.html#Method:clear) | Removes all elements from this vector. |
| [**get(index)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Vector-class.html#Method:get) | Returns the element at the specified index in this vector. |
| [**set(index, value)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Vector-class.html#Method:set) | Replaces the element at the specified index in this vector with a new value. |
| [**insert(0, value)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Vector-class.html#Method:insert) | Inserts the element into this vector before the specified index. |
| [**remove(index)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Vector-class.html#Method:remove) | Removes the element at the specified index from this vector. |
| [**add(value)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Vector-class.html#Method:add) | Adds a new value to the end of this vector. |
| **Operators** |  |
| [**vec[index]**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Vector-class.html#Operator:[]) | Overloads **[]** to select elements from this vector. |
| [**v1 + v2**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Vector-class.html#Operator:+) | Concatenates two vectors. |
| [**v1 += v2;**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Vector-class.html#Operator:+=) | Adds all of the elements from **v2** (or the single specified value) to **v1**. |

Stack #include “stack.h”

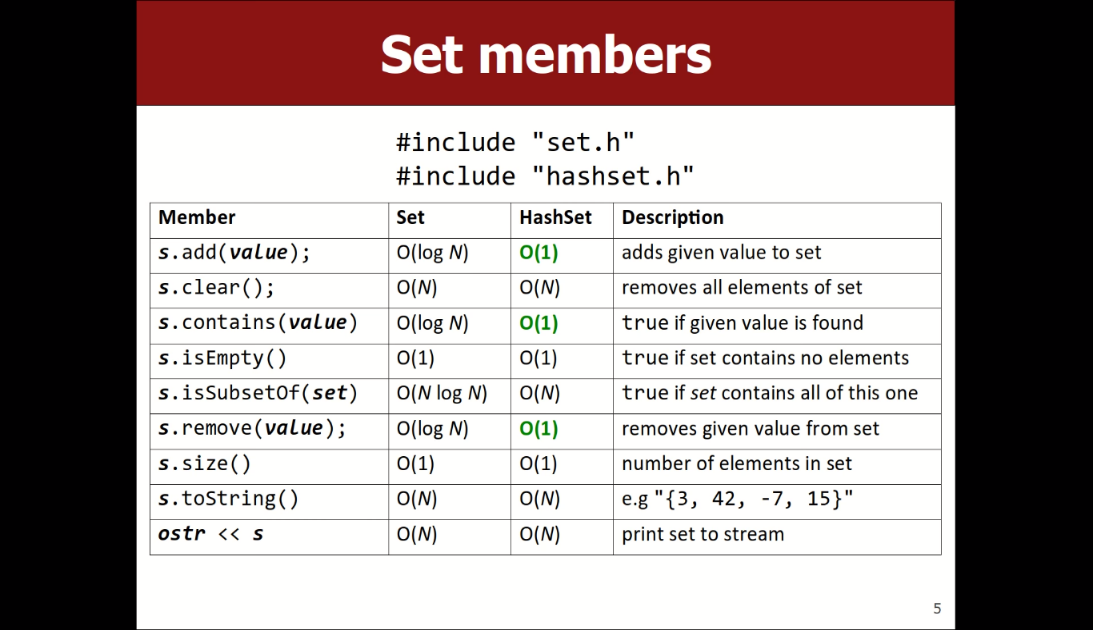


|  |  |  |
| --- | --- | --- |
| [clear()](https://stanford.edu/~stepp/cppdoc/Stack-class.html#Method:clear) | O(1) | Removes all elements from this stack. |
| [equals(***stack***)](https://stanford.edu/~stepp/cppdoc/Stack-class.html#Method:equals) | O(N) | Returns **true** if the two stacks contain the same elements in the same order. |
| [isEmpty()](https://stanford.edu/~stepp/cppdoc/Stack-class.html#Method:isEmpty) | O(1) | Returns **true** if this stack contains no elements. |
| [peek()](https://stanford.edu/~stepp/cppdoc/Stack-class.html#Method:peek) | O(1) | Returns the value of top element from this stack, without removing it. |
| [pop()](https://stanford.edu/~stepp/cppdoc/Stack-class.html#Method:pop) | O(1) | Removes the top element from this stack and returns it. |
| [push(***value***)](https://stanford.edu/~stepp/cppdoc/Stack-class.html#Method:push) | O(1) | Pushes the specified value onto this stack. |
| [size()](https://stanford.edu/~stepp/cppdoc/Stack-class.html#Method:size) | O(1) | Returns the number of values in this stack. |
| [toString()](https://stanford.edu/~stepp/cppdoc/Stack-class.html#Method:toString) | O(N) | Converts the stack to a printable string representation. |

Queue #include “queue.h”

|  |  |
| --- | --- |
| [**size()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Queue-class.html#Method:size) | Returns the number of values in the queue. |
| [**isEmpty()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Queue-class.html#Method:isEmpty) | Returns **true** if the queue contains no elements. |
| [**clear()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Queue-class.html#Method:clear) | Removes all elements from the queue. |
| [**enqueue(value)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Queue-class.html#Method:enqueue) | Adds **value** to the end of the queue. |
| [**dequeue()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Queue-class.html#Method:dequeue) | Removes and returns the first item in the queue. |
| [**peek()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Queue-class.html#Method:peek) | Returns the first value in the queue, without removing it. |
| [**front()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Queue-class.html#Method:front) | Returns the first value in the queue by reference. |
| [**back()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Queue-class.html#Method:back) | Returns the last value in the queue by reference. |

Set #include “set.h”



|  |  |  |
| --- | --- | --- |
|  | | |
| [add(***value***)](https://stanford.edu/~stepp/cppdoc/Set-class.html#Method:add) | O(log N) | Adds an element to this set, if it was not already there. |
| [clear()](https://stanford.edu/~stepp/cppdoc/Set-class.html#Method:clear) | O(N) | Removes all elements from this set. |
| [contains(***value***)](https://stanford.edu/~stepp/cppdoc/Set-class.html#Method:contains) | O(log N) | Returns **true** if the specified value is in this set. |
| [equals(***set***)](https://stanford.edu/~stepp/cppdoc/Set-class.html#Method:equals) | O(N) | Returns **true** if the two sets contain the same elements. |
| [first()](https://stanford.edu/~stepp/cppdoc/Set-class.html#Method:first) | O(log N) | Returns the first value in the set in the order established by a for-each loop. |
| [isEmpty()](https://stanford.edu/~stepp/cppdoc/Set-class.html#Method:isEmpty) | O(1) | Returns **true** if this set contains no elements. |
| [isSubsetOf(***set2***)](https://stanford.edu/~stepp/cppdoc/Set-class.html#Method:isSubsetOf) | O(N) | Implements the subset relation on sets. |
| [mapAll(***fn***)](https://stanford.edu/~stepp/cppdoc/Set-class.html#Method:mapAll) | O(N) | Iterates through the elements of the set and calls **fn(value)** for each one. |
| [remove(***value***)](https://stanford.edu/~stepp/cppdoc/Set-class.html#Method:remove) | O(log N) | Removes an element from this set. |
| [size()](https://stanford.edu/~stepp/cppdoc/Set-class.html#Method:size) | O(1) | Returns the number of elements in this set. |
| [toString()](https://stanford.edu/~stepp/cppdoc/Set-class.html#Method:toString) | O(N) | Converts the set to a printable string representation. |

|  |  |  |
| --- | --- | --- |
| [***set1*** == ***set2***](https://stanford.edu/~stepp/cppdoc/Set-class.html#Operator:==) | O(N) | Returns **true** if **set1** and **set2** contain the same elements. |
| [***set1*** != ***set2***](https://stanford.edu/~stepp/cppdoc/Set-class.html#Operator:!=) | O(N) | Returns **true** if **set1** and **set2** are different. |
| [***set1*** + ***set2***](https://stanford.edu/~stepp/cppdoc/Set-class.html#Operator:+) | O(N) | Returns the union of sets **set1** and **set2**, which is the set of elements that appear in at least one of the two sets. |
| [***set*** + ***value***](https://stanford.edu/~stepp/cppdoc/Set-class.html#Operator:+) | O(N) | Returns the union of set **set1** and individual value **value**. |
| [***set1*** += ***set2***;](https://stanford.edu/~stepp/cppdoc/Set-class.html#Operator:+=) | O(N) | Adds all of the elements from **set2** (or the single specified value) to **set1**. |
| [***set*** += ***value***;](https://stanford.edu/~stepp/cppdoc/Set-class.html#Operator:+=) | O(log N) | Adds the single specified value to the set. |
| [***set1*** - ***set2***](https://stanford.edu/~stepp/cppdoc/Set-class.html#Operator:-) | O(N) | Returns the difference of sets **set1** and **set2**, which is all of the elements that appear in **set1** but not **set2**. |
| [***set*** - ***value***](https://stanford.edu/~stepp/cppdoc/Set-class.html#Operator:-) | O(N) | Returns the set **set** with **value** removed. |
| [***set1*** -= ***set2***;](https://stanford.edu/~stepp/cppdoc/Set-class.html#Operator:-=) | O(N) | Removes the elements from **set2** (or the single specified value) from **set1**. |
| [***set*** -= ***value***;](https://stanford.edu/~stepp/cppdoc/Set-class.html#Operator:-=) | O(log N) | Removes the single specified value from the set. |
| [***set1*** \* ***set2***](https://stanford.edu/~stepp/cppdoc/Set-class.html#Operator:*) | O(N) | Returns the intersection of sets **set1** and **set2**, which is the set of all elements that appear in both. |
| [***set1*** \*= ***set2***;](https://stanford.edu/~stepp/cppdoc/Set-class.html#Operator:*=) | O(N) | Removes any elements from **set1** that are not present in **set2**. |
| ***ostream*** << ***set*** | O(N) | Outputs the contents of the set to the given output stream. |
| ***istream*** >> ***set*** | O(N log N) | Reads the contents of the given input stream into the set. |

Lexicon “lexicon.h”

|  |  |
| --- | --- |
| [**size()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Lexicon-class.html#Method:size) | Returns the number of words contained in the lexicon. |
| [**isEmpty()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Lexicon-class.html#Method:isEmpty) | Returns **true** if the lexicon contains no words. |
| [**clear()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Lexicon-class.html#Method:clear) | Removes all words from the lexicon. |
| [**add(word)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Lexicon-class.html#Method:add) | Adds the specified word to the lexicon. |
| [**addWordsFromFile(filename)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Lexicon-class.html#Method:addWordsFromFile) | Reads the file and adds all of its words to the lexicon. |
| [**contains(word)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Lexicon-class.html#Method:contains) | Returns **true** if **word** is contained in the lexicon. |
| [**containsPrefix(prefix)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Lexicon-class.html#Method:containsPrefix) | Returns true if any words in the lexicon begin with **prefix**. |

Map #include “map.h”

|  |  |
| --- | --- |
| [**size()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Map-class.html#Method:size) | Returns the number of entries in this map. |
| [**isEmpty()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Map-class.html#Method:isEmpty) | Returns **true** if this map contains no entries. |
| [**put(key, value)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Map-class.html#Method:put) | Associates **key** with **value** in this map. |
| [**get(key)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Map-class.html#Method:get) | Returns the value associated with **key** in this map. |
| [**containsKey(key)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Map-class.html#Method:containsKey) | Returns **true** if there is an entry for **key** in this map. |
| [**remove(key)**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Map-class.html#Method:remove) | Removes any entry for **key** from this map. |
| [**clear()**](http://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/materials/cppdoc/Map-class.html#Method:clear) | Removes all entries from this map. |

|  |  |
| --- | --- |
| **Constructor** | |
| [Timer()](https://stanford.edu/~stepp/cppdoc/Timer-class.html#Constructor:Timer) | Creates a timer object that has not yet been started. |
| [Timer(***autostart***)](https://stanford.edu/~stepp/cppdoc/Timer-class.html#Constructor:Timer2) | Creates a timer object and optionally starts it. |
|  |  |

**Timer** [#include "timer.h"](https://stanford.edu/~stepp/cppdoc/timer.html)

|  |  |
| --- | --- |
| [elapsed()](https://stanford.edu/~stepp/cppdoc/Timer-class.html#Method:elapsed) | Returns the number of milliseconds since the timer was started. |
| [isStarted()](https://stanford.edu/~stepp/cppdoc/Timer-class.html#Method:isStarted) | Returns whether the timer has been started. |
| [start()](https://stanford.edu/~stepp/cppdoc/Timer-class.html#Method:start) | Starts the timer. |
| [stop()](https://stanford.edu/~stepp/cppdoc/Timer-class.html#Method:stop) | Stops the timer. |

[#include “gwindow.h”](https://stanford.edu/~stepp/cppdoc/doxygen/classGWindow.html#details)

**Pointer-მიმთითებელი**

int a=1;

int\* p = &a; -ამპერსანტი გვაძლევს ინფორმაციას მეხსიერებაში ადგილის შესახებ, რომელზეც ცვლადი მდებარეობს, ხოლო ფოინთერ ტიპის ცვლადი ინახავს ამ მისამართს.

cout << p << endl; //ბეჭდავს მისამართს

dereferencing-დერეფერენსი

cout<< \*p << endl; //ბეჭდავს ცვლადის მნიშვნელობას, რომელზეც უთითებს (follow the pointer)

**null pointer:** int\* p1 = nullptr; points at 0 location; - 0x0 (points to nothing)

cout<< \*p1 << endl; //ვერაფერს დაბეჭდავს (და-crush-ავს), რადგან არაფერია შენახული ამ მისამართზე.

ასევე მოხდება არაინიციალიზებული ფოინთერის დერეფერენსით დაბეჭვდის შემთხვევაშიც;

**uninitialized pointer:** points at a random address.

declaring object:

1. **non-pointer way:**

Date d1;

d1.month=7;

d1.day =13;

1. **pointer way:**

Date\* d2=new Date();

d2->month =7; // similar to: (\*d2).month

d2->day =13;

დადებით-უარყოფითი მხარეები:

1-ს თუ ვიყენებთ რაიმე ფუქციაში, მოცემული ობიექტი d1 „იცოცხლებს“ მხოლოდ {-დან }-მდე, წაიშლება მეხსიერებიდან **stack memory**

2-ს შემთხვევაში, d2 არ იშლება მეხსიერებიდან ფუნქციის მერე; helps to manage memory. თუმცა ეს ფაქტი არ ნიშნავს იმას, რომ გლობალური ხდება d2, არამედ უბრალოდ არ იშლება მეხსიერებიდან მანამ, სანამ თავად არ წავშლით. **heap memory**

**Linked Lists - ბმული სია**

შედგება სტრუქტურებისადაგან, მაგალითად, ჩვენთვის კარგად ნაცნობი Node, რომლებიც ინახავენ შემდეგი ელემენტის (სტრუქტურის) ადგილმდებარეობის ინფორმაციას.

**ორმაგი ბმული სია** გულისხმობს, რომ სტრუქტურა ინახავს არა მარტო შემდეგ, არამედ წინა ელემენტის (სტრუქტურის) მისამართს.

struct Node{

int value;

Node\* next;

Node\* prev;

}**;**

**ორობითი ძებნის ხე- Binary Search Tree (BST)**

მონაცემთა სტრუქტურა.

ელემენტთა შესანახავად იყენებს შედარების ფუნქციას. შესაბამისად, შეუძლია ისეთი ტიპის ელემენტები შეინახოს, რომელსაც გააჩნია შედარების ოპერატორი.

BST არის Node-ების კოლექცია. თითოეულ კვანძს ჰყავს ორი შვილი- მარცხენა და მარჯვენა:

struct Node{

type name;

Node\* left;

Node\* right;

};

შეთანხმებისამებრ- მარცხნივ ინახება ისეთი ელემენტი, რომელიც ნაკლებია თავის მშობელზე, ხოლო მარჯვნივ- მეტი ან ტოლი.

ელემენტები ხეში არ მეორდება, ყველა უნიკალურია.

საწყის ელემენტს ხეში ეწოდება root ელემენტი.

თუ Node-ს შვილები არ ჰყავს, ანუ მისი მარცხენა და მარჯვენა შვილები Null-ებია, მას ფოთოლი ჰქვია.

ხის სიმაღლე- height- არის მანძილი root-დან ყველაზე დაშორებულ leaf-მდე.

ხეში ელემენტის ჩამატების დროა: O(h), სადაც h=height; ხის მაქსიმალური სიმაღლეა n-ელემენტთა რაოდენობა. მინიმალური: O(log *n*);

თავისთავად-ბალანსირებადი ხე უზრუნველყოფს სიმაღლის მინიმალურობას;

ხეში ელემენტთა დათვალიერების 3 ძირითადი გზაა:

* **Preorder**: ვნახოთ ჯერ Node, შემდეგ მისი შვილები
* **Inorder**: ვნახოთ მარცხენა შვილი, შემდეგ Node, ბოლოს მარჯვენა შვილი
* **Postorder**: ვნახოთ ჯერ შვილები, შემდეგ Node

მას შემდეგ, რაც ხე გამოვიყენეთ, მისგან უნდა გავათავისუფლოთ მეხსიერება, მსგავსად ბმული სიებისა.

ორობითი ძებნის ხეში Node-ის წაშლა არც ისე მარტივი ოპერაციაა. მას შემდეგ, რაც ძებნის ფუნქციით წასაშლელ Node-ს მივადგებით, იგი უნდა წაიშალოს ისე, რომ არ დაირღვეს ხის ძირითადი პრინციპი. ფოთლის წაშლა მარტივია, რადგან მას შვილები არ ჰყავს და არ სჭირდებათ ახალი მშობელი. სხვა შემთხვევაში, Node-ის წაშლის დროს გვრჩება ორი განშტოება, მისი ორი შვილის შთამომავლობა. **ხის პრინციპი არ დაირღვევა იმ შემთხვევაში, თუ ავირჩევთ მარცხენა ნაწილის უკიდურეს მარჯვენა წევრს, ან მარჯვენა ნაწილის უკიდურეს მარცხენა წევრს.**

დაბალანსებული ორობითი ხე გულისხმობს, რომ ყოველ ჯერზე სხვაობა მარცხენასა და მარჯვენა შვილების სიმაღლეებს შორის 0 ან 1-ია. ხის დასაბალანსებლად AVL- ალგორითმის გამოყენება შეიძლება, თუმცა ბოჭორას არ აუხსნია (წიგნში 683 გვ.-დან იწყება).

**გრაფები**

შესაძლოა გამოისახოს შემდეგნაირად:

**Map<*Node\**, Vector<*Node\**> >,** სადაც Key იქნება წვერო, ხოლო value-ში ჩაწერილი იქნება ის წვეროები, რომელთანაც დაკავშირებულია (მეზობლები).

ბმული კომპონენტი: ერთმანეთთან დაკავშირებული წვეროები ქმნის ბმულ კომპონენტს.

* თითოეული წვეროთა წყვილისთვის არსებობს რაიმე გზა.
* არც ერთი წვერო ამ ქვესიმრავლეში არ არის დაკავშირებული ისეთ წვეროსთან, რომელიც არაა ამ ბმულ კომპონენტში.

იმისათვის, რომ აღმოვაჩინოთ ბმული კომპონენტი გრაფში, საჭიროა ყველა წვეროზე იტერაცია (შემოვლა).

გრაფზე იტერაციის ბევრი გზა არსებობს, თუმცა ყველა მათგანი მოიცავს შემდეგ საფეხურებს:

* სიმრავლე იმ წვეროებისა, რომლების უკვე მოვინახულეთ;
* სიმრავლე იმ წვეროებისა, რომელთაც მოვინახულებთ შემდეგში;
* დანარჩენი წვეროები;

წვეროებზე იტერაციის მეთოდები განსხვავდება იმის მიხედვით, თუ რომელ წვეროს ავირჩევთ შემდეგში მოსანახულებლად.

რეკურსიულად **სიღრმეში ძებნა (DFS- depth first search):**

* რომ აღმოვაჩინოთ **ბმული კომპონენტი**, გვჭირდება ვიცოდეთ წვეროებს შორის გზის არსებობა (ამ შემთხვევაში, არაა აუცილებელი უმოკლესი იყოს გზა);
* ერთ-ერთი საშუალება ბმულობის აღმოჩენისათვის იქნება დინების მიმართულების მიყოლა, ანუ ავირჩიოთ მიმართულება და გავყვეთ მას;
  + როდესაც წვეროს შემდეგ აღარ იქნება გზა, უბრალოდ დაბრუნდეს უკან ერთი ნაბიჯით და იქიდან განაგრძოს, თუ არც იქ იქნება სხვა გზა, ისევ უკან დაბრუნდეს და ა.შ. სანამ (თუ) არ მივა საწყისამდე;

**წვერო u-დან რეკურსიული DFS:**

* თუ u უკვე მონიშნულია, break;
* მოვნიშნოთ u;
* u-ს თითოეული მეზობელი v -წვეროსთვის ვიძახებთ რეკურსიას:
  + რეკურსიული DFS v-დან;

ყველაზე ხშირად DFS იმპლემენტირებულია **stack**-ის საშუალებით:

Set-ში აღებული წვეროები, Stack-ში მეზობელი წვეროები.

**სიგანეში ძებნა (BFS –breadth first search):**

* საზოგადოდ ძებნის ალგორითმი, როდესაც შემდეგი წვეროები მწკრივში **queue**-ში იყრება;
* ეძებს წვეროებს საწყისი წვეროდან ჯერ ერთის დაშორებით, მერე ორის და ა.შ.;
* ეძებეს გზას უმცირესი წიბოთა რაოდენობით საწყისი წვეროდან ნებისმიერ სხვა წვერომდე;

სიგანეში გზის ძებნის ალგორითმი ოპტიმალურია იმ შემთხვევაში, როდესაც „უწონო“ (ან ერთნაირი წონის, მაგ. ყველა- 1) წიბოები გვაქვს.

ამ შემთხვევაში შესაბამისი “უიაფესი“ გზის ალგორითმის აღსაწერად ვაკეთებთ შემდეგს:

* წვეროები იყოფა სამ ჯგუფად:
  + მწვანე - რომლის უიაფესი მისასვლელი გზა ვიცით უკვე
  + ყვითელი - სავარაუდო გზის სიგრძე
  + ნაცრისფერი - აზრზე არ ვართ, რა სიგრძის გზა მიდის
* ეტაპობრივად მოვაშოროთ უმცირესი გზის სავარაუდო ყვითელი სიგრძე და ვაქციოთ მწვანე წვეროდ.

***Dijkstra's algorithm* - დეიქსტრას ალგორითმი**

***O(e + vlog v)* -** მუშაობის დრო, რომელშიც v-წვეროთა რაოდენობაა, ხოლო e-წიბოების.

ალგორითმი ეძებს უმოკლეს გზას იმ შემთხვევაში, თუ წიბოების წონა არაუარყოფითი რიცხვებია.

* მოვნიშნოთ ყველა წვერო ნაცრისფრად, გარდა საწყისი წვეროსა, რომელიც იქნება ყვითელი და 0 გზის სიგრძით;
* მანამ სანამ ყვითელი წვეროები არსებობს:
  + ავირჩიოთ ყვითელი წვერო მისასვლელი გზების სიგრძეთა უმცირესი ჯამით
  + მოვნიშნოთ არჩეული წვერო მწვანედ
  + მოვნიშნოთ არჩეული წვეროს ყველა მეზობელი წვერო ყვითლად სავარაუდო გზის სიგრძეებით
  + თუ მეზობელი წვერო ყვითელი იყო არჩეული წვეროს მიხედვით შევცვალოთ მისი ახალი გზის სიგრძის ღირებულება

დეიქსტრას ალგორითმი იყენებს **priority queue-ს decrease key()-ფუნქციით,** რომელიც პრიორიტეტის შეცვლის საშუალებას იძლევა.

**როგორ მუშაობს ალგორითმი:**

* დეიქსტრას ალგორითმი ეტაპობრივად ითვლის უიაფეს გზებს და ამ გზებს მიჰყვება. მიმართულება არ აქვს განსაზღვრული და სწორედ ამიტომ სჭირდება რაღაც მინიშნება, თუ საით განაგრძოს ძებნა. უმრავლესობა იმ წვეროებისა, რომლებსაც ამოწმებს და იყენებს ალგორითმი, გამოუსადეგარია.
* ჰოდა, **Heuristic function** გააკეთებს ამ საქმეს.

ჰეროისტიკა ითვლის (ცდილობს გამოიცნოს) მანძილი ცნობილი წვეროდან დანიშნულების წვერომდე. რა თქმა უნდა, ფუნქცია აუცილებლად გზის სიგრძის სწორ მნიშვნელობას არ დააბრუნებს, თუმცა გამოთვლები საჭიროა მაქსიმალურად ზუსტი იყოს.

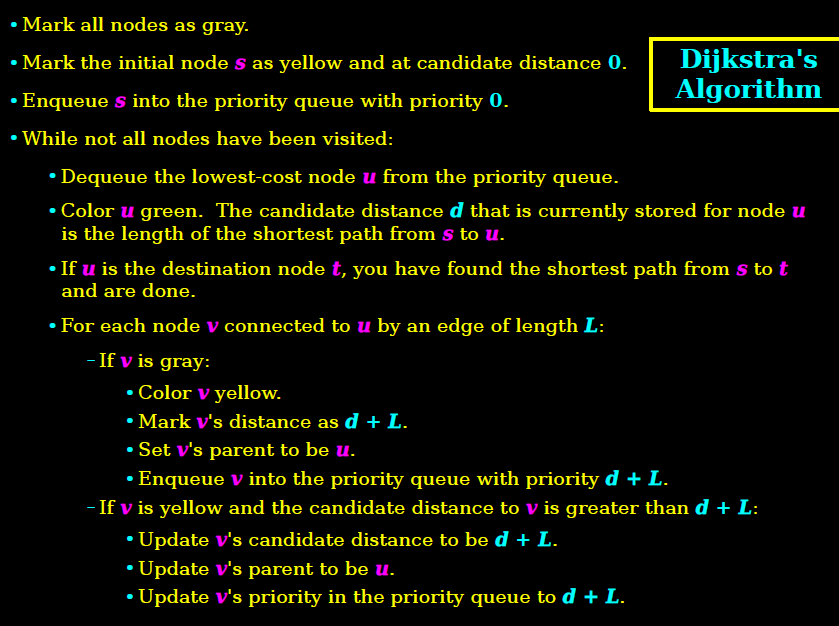
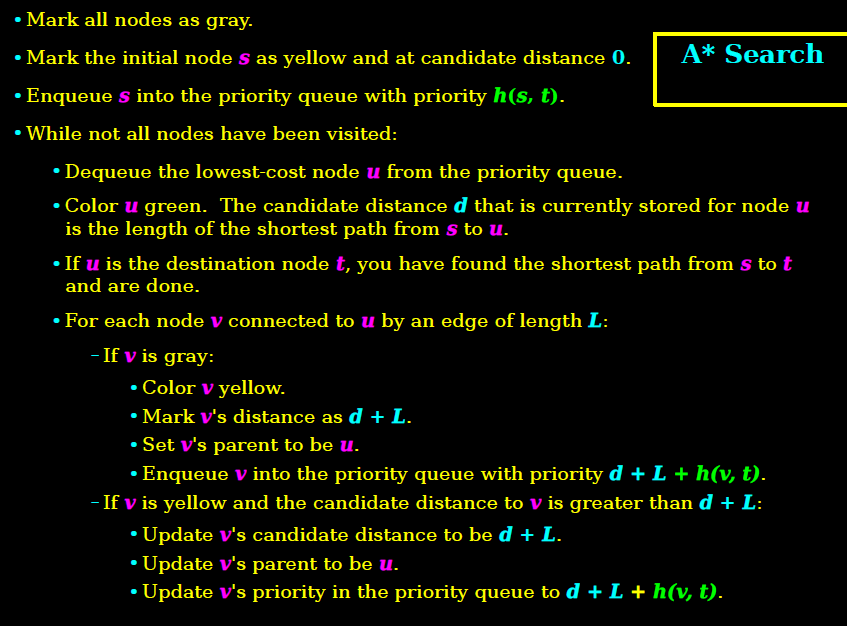
* **admissible heuristic -** დასაშვები, მისაღები ჰეროისტიკა ჰქვია ფუნქციას, თუკი ის არასდროს არ აჭარბებს გამოთვლებს საპოვნელ გზის სიგრძესთან დაკავშირებით. ანუ თუ სრულდება შემდეგი **უტოლობა:**
* *predicted-distance* ***≤*** *actual-distance*

***A\* search*** *algorithm:*

ჩვენ შეგვიძლია შევცვალოთ დეიქსტრას ალგორითმი ჰეროისტიკის დახმარებით:

ნებისმიერი მოცემული ***u*** წვეროსთვის არსებობს ორი გზის ღირებულება: ***s***საწყისი წვეროდან გზის ღირებულება და ჰეროისტიკული ფუნქციით გამოთვლილი გზის სავარაუდო ღირებულება ***u****-დან* საბოლოო ***t****-მდე.*

* **იდეა:** გავუშვათ ისევ დეიქსტრას ალგორითმი, თუმცა პრიორიტეტულ რიგში გამოვიყენოთ შემდეგი პრიორიტეტი: **priority(u) = distance(s, u) + heuristic(u, t)**



მინიმალური დამფარავი ხე **Minimum Spanning Trees (MST)**

დამფარავი ხე არამიმართულ გრაფში იმ წიბოთა სიმრავლეა, რომელიც არ შეიცავს ციკლს და რომელშიც არსებობს გზა ნებისმიერი წვეროდან სხვა წვერომდე მოცემულ გრაფში.

მინიმალური დამფარავი ხე დამფარავი ხეა, რომელიც შეიცავს ყველაზე ნაკლებს საერთო ღირებულებას გრაფის წიბოებისა.

**Kruskal's algorithm -** კრასკალის ალგორითმი MST-ის საპოვნელად გამოდგება

იდეა შემდეგია:

* ყველა წიბო „ამოიშლება“ გრაფიდან
* წიბოები **pqueue-ში** ლაგდება, პრიორიტეტებად- წიბოს წონა.
* სანამ pqueue არ არის ცარიელი:
  + dequeue() ავიღოთ წიბო პრიორიტეტული რიგიდან
  + თუ წიბოს ბოლოები ერთმანეთ რამენაირად უკვე არ უკავშირდება, დავამატოთ ეს წიბო.
  + სხვა შემთხვევაში გამოვტოვოთ ეს წიბო

ყველაზე რთული ნაწილი ამ ალგორითმში წიბოების ბოლოების ერთმანეთთან კავშირის დადგენაა. ამ პრობლემის გადაწყვეტაა წვეროების ჯგუფებად, **clusters**-ად, დაყოფა.

* თავიდან თითოეული წვეროსათვის გვაქვს დამოუკიდებელი ჯგუფი, კლასთერი.
* მას შემდეგ, რაც ვამატებთ წიბოს, ერთიანდება ის ორი ჯგუფი, რომლებშიც შედის წიბოს ბოლოები, წვეროები. ამგვარად, ახალ ჯგუფში ჩაყრილი წვეროები ერთმანეთს უკავშირდებიან, მათ შორის არსებობს რაიმე გზა.

**კრასკალი კლასთერებით:**

* მოვათავსოთ თითოეული წვერო თავის სიმრავლეში , კლასთერში
* მოვათავსოთ თითოეული წიბო **pqueue-ში**
* მანამ, სანამ 2 ან მეტი კლასტერი არსებობს
  + dequeue() ამოვიღოთ წიბო პრიორიტეტული რიგიდან
  + თუ მისი ბოლოები არ არიან ერთსა და იმავე კლასტერში:
    - გავაერთიანოთ წიბოს ბოლოების შემცველი კლასტერები
    - დავამატოთ წიბო დამფარავი ხეში
* დავაბრუნოთ დამფარავი ხე