COMP 345 Fall 18 Week 3

Haotao Lai (Eric)
h_lai@encs.concordia.ca

Lab Instructor

Section: B-X 9999 --W---- 20:30 22:20 H929

Name: Haotao Lai (Eric)

Office: EV 8.241

Email: h_lai@encs.concordia

Website: http://laihaotao.me/ta

Assignment 1 (Dr. Paquet's section)

Reminder:

- Assignment 1 is out since Sep 13
- Assignment 1 will be due on Oct 12 at 23:59
- You need to submit your assignment via EAS (one submission per team)
- Don't try to finish it one day before the due (probably you can't make it on time)

Contents

- parameter passing
- vector
- graph traversal algorithm

Parameter-Passing

Parameter-Passing

- pass by value: copy the value, and pass the new copied value;
- pass by reference: create a new alias for that parameter and pass the alias;
- pass by pointer: get the address of the parameter and pass that address;

Parameter-Passing

```cpp
#include <iostream>

using std::cout;
using std::endl;

int main() {
    int n = 100;
    cout << "= = = = = = = = = = = = = = = = =" << endl;
    cout << "integer n: " << n << endl;
    cout << "= = = = = = = = = = = = = = = = =" << endl;

    pass_by_value(n);
    pass_by_reference(n);
    pass_by_pointer(&n);

    return 0;
}
```

Parameter-Passing

```
void pass_by_value(int n) {
    cout << "= = = = = = = = = = = = = = = = =" << endl;
    cout << "pass by value" << endl;
    cout << "value of n: " << n << endl;
    cout << "address of n: " << &n << endl;
}

void pass_by_reference(int &n) {
    cout << "= = = = = = = = = = = = = = = = =" << endl;
    cout << "pass by reference" << endl;
    cout << "value of n: " << n << endl;
    cout << "address of n: " << &n << endl;
}

void pass_by_pointer(int *n) {
    cout << "= = = = = = = = = = = = = = = = =" << endl;
    cout << "pass by pointer" << endl;
    cout << "value of n: " << *n << endl;
    cout << "address of n: " << n << endl;
}
```

Parameter-Passing

Output from the program

```
= = = = = = = = = = = = = = = = =
integer n: 100
address of n in main: 0x7ffeee9f14c8
= = = = = = = = = = = = = = = = =
pass by value
value of n: 100
address of n: 0x7ffeee9f145c
= = = = = = = = = = = = = = = = =
pass by reference
value of n: 100
address of n: 0x7ffeee9f14c8
= = = = = = = = = = = = = = = = =
pass by pointer
value of n: 100
address of n: 0x7ffeee9f14c8
```

Difference between reference and pointer

1. A pointer can be re-assigned any number of times while a reference cannot be re-seated after binding.
2. Pointers can point nowhere (`NULL`), whereas reference always refer to an object.
3. You can't take the address of a reference like you can with pointers.
4. There's no "reference arithmetics" (but you can take the address of an object pointed by a reference and do pointer arithmetics on it as in `&obj + 5`).

—— from stackoverflow, know more click here

How to write a function that can swap two integers?

Parameter-passing

```
int main() {
    int i = 10;
    int j = 20;

    cout << "= = = = = = = = = = =" << endl;
    cout << "before swap" << endl;
    cout << "value of i: " << i << endl;    // 10
    cout << "value of j: " << j << endl;    // 20

    swap(i, j);
    swap(&i, &j);

    cout << "= = = = = = = = = = =" << endl;
    cout << "after swap" << endl;
    cout << "value of i: " << i << endl;    // expecting 20
    cout << "value of j: " << j << endl;    // expecting 10
}
```

Straight forward

```
void swap1(int x, int y) {
    int tmp = x;
    x = y;
    y = tmp;
}

void swap2(int &x, int &y) {
    int tmp = x;
    x = y;
    y = tmp;
}
```

Which one is correct ?

```c
void swap3(int *x, int *y) {
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

void swap4(int *x, int *y) {
    int *tmp = x;
    x = y;
    y = tmp;
}
```

Vector

Vector

- vector<T> in cpp likes List<T> in Java
- Vectors are sequence containers representing arrays that can change in size.
- know more about vector, go here

vector

```cpp
#include <iostream>
#include <vector>

using std::vector;
using std::cout;
using std::endl;

int main() {
    vector<int> vecInt;
    // add elements into the vector
    for (int i = 0; i < 10; i++) {
        vecInt.push_back(i);
    }
    // traverse the vector
    for (auto it = vecInt.begin(); it != vecInt.end(); it++) {
        cout << it.operator*() << " ";
    }
    cout << endl;
    // another way to traverse
    for (auto &vec : vecInt) {
        cout << vec << " ";
    }
    cout << endl;
    // access via index
    cout << "the 2nd element in the vector is -> " << vecInt[1] << endl;
    // access the first and last element
    cout << "the 1st element in the vector is -> " << vecInt.front() << endl;
    cout << "the last element in the vector is -> " << vecInt.back() << endl;
    // ..... try to discover more APIs by yourself
    return 0;
}
```

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Element access:

operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)
data C++11	Access data (public member function)

Modifiers:

assign	Assign vector content (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_back C++11	Construct and insert element at the end (public member function)

vector

```
// erasing from vector
#include <iostream>
#include <vector>

int main () {
  std::vector<int> myvector;

  for (int i=1; i<=10; i++) {
    myvector.push_back(i);
  }

  myvector.erase(myvector.begin() + 5);
  myvector.erase (myvector.begin(), myvector.begin() + 3);

  std::cout << "myvector contains:";
  for (unsigned i=0; i<myvector.size(); ++i) {
    std::cout << ' ' << myvector[i];
  }
  std::cout << '\n';

  return 0;
}
```

vector

```cpp
// vector::emplace
#include <iostream>
#include <vector>

int main () {
  std::vector<int> myvector = {10,20,30};

  auto it = myvector.emplace(myvector.begin()+1, 100);
  myvector.emplace (it, 200);
  myvector.emplace (myvector.end(), 300);

  std::cout << "myvector contains:";
  for (auto& x: myvector) {
    std::cout << ' ' << x;
  }
  std::cout << '\n';

  return 0;
}
```

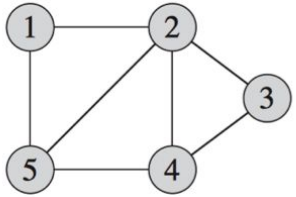
beyond vector

C++ container library reference (you may need them for dfs or bfs implementation or you assignment):

https://en.cppreference.com/w/cpp/container

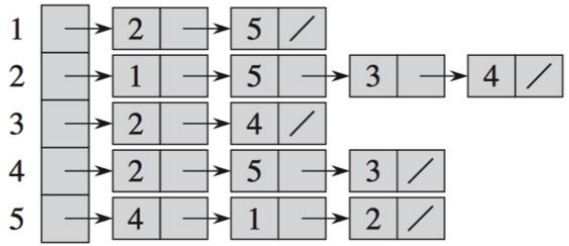
Graph

How can we represent a graph



(a)

real graph



(b)

adjacency linked list

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

adjacency matrix

How to traverse a graph

There are a lot of ways to do it, the most common two is DFS and BFS.

You are not restricted in this two ways, during your demo ! ! !

Take DFS as an Example

Recursion

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)
```

white means the vertex hasn't been discovered yet

time just for timestamp

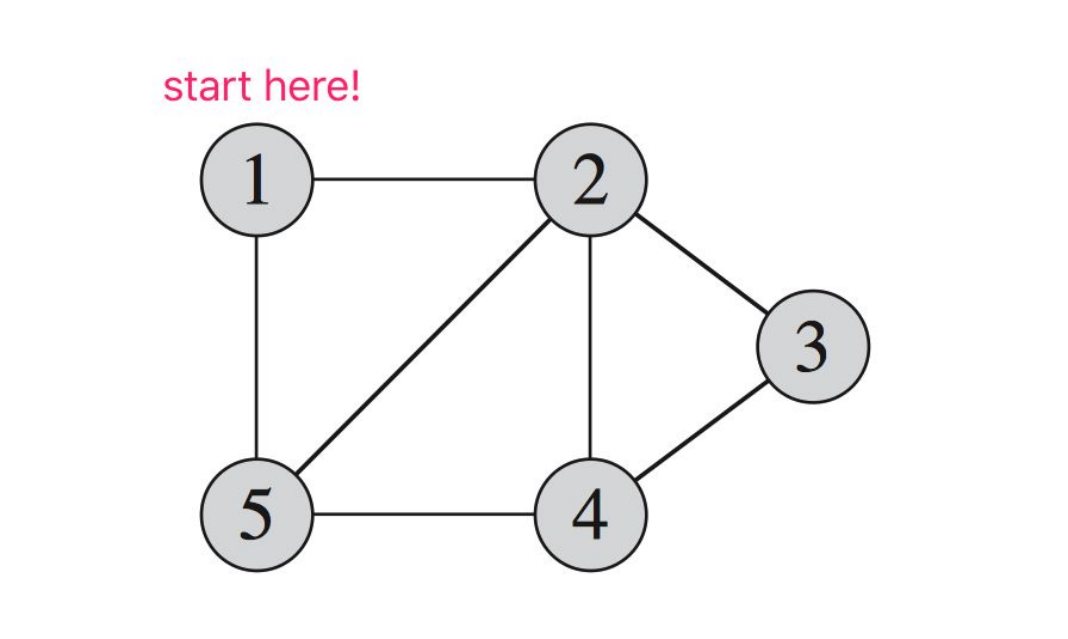
Recursion (continue)

```
DFS-VISIT(G, u)
 1  time = time + 1                  // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]            // explore edge (u, v)
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK                  // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

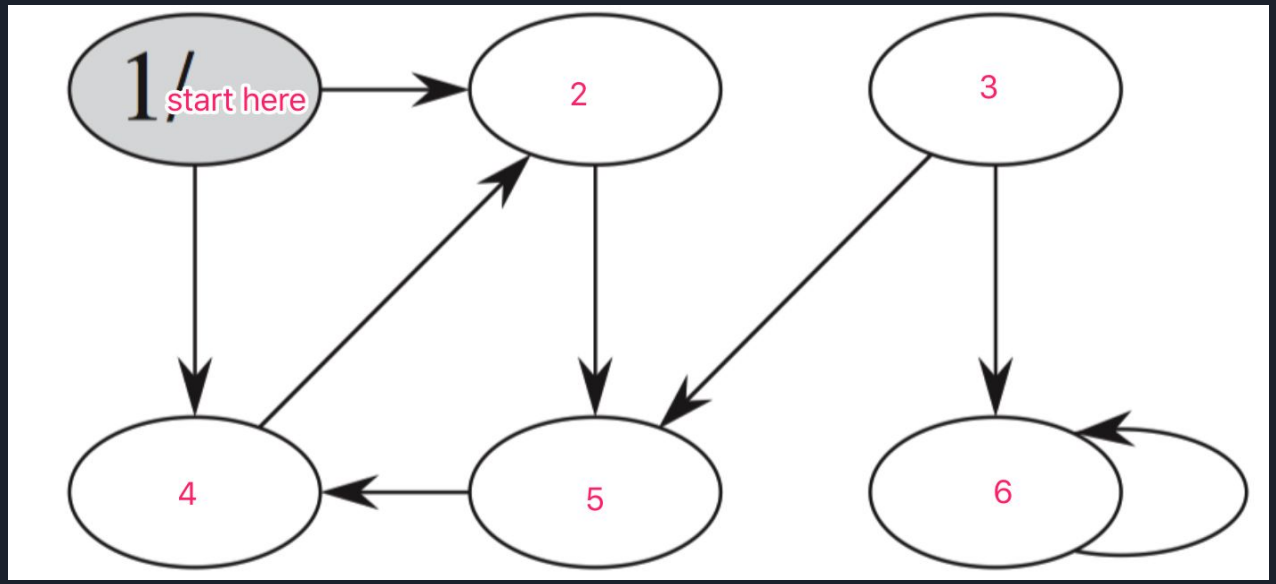
Loop

```
dfs(G, v)
    Set visited
    Stack Stack
    stack.push(v)

    while stack is not empty
        Stack s
        tmp = stack.pop()
        visited.add(tmp)

        for all vertex u in G.Adj[tmp]
            if u is not in visited
                AND u is not in stack
                    s.push(u)

        while s is not empty
            stack.push(s.pop())
```

Example 1 Undirected Graph



Example 2 Directed Graph



Question?