CS1510

web.mst.edu/~ricardom/

section A

CS340 - office

T/Th 3:30 - 5:00 in CS212 (computer lab)

50% tests 3 and 1 final
50% homework (lowest dropped)

graded on correctness, not length

no-compile — 10% not-io score=10 (/100)
crashing — 25%

LEAD
CS212
Tu/Th 3:30-5:30

max 5 ~~misses~~ absences

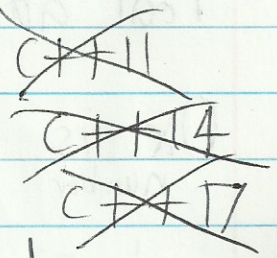
price@mst.edu

advisors are informed of bad academic performance

Feb 17 – Test #1
March 24 — Test #2
April 28 — Test #3
Final (date published somewhere) (comprehensive)

Lecture Notes/Sample Code

C++03



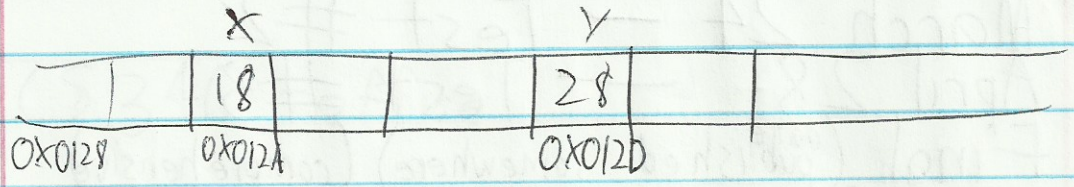
do not use jpico!

1/20/2017

Tre is in Google

Minerama 4-6 today Havener

C++ Pointers



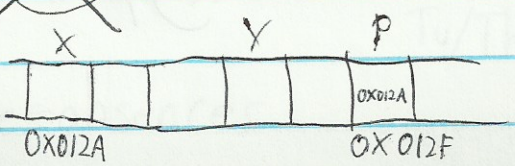
& — address-of operator

cout << &x; 0X012A

real addresses are many more digits

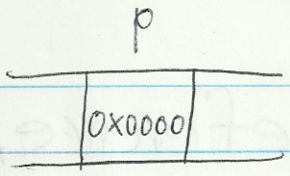
0x is a prefix indicating a hexadecimal number

int p;
p = &x; fails



int *p;
p = &x;

```
int *p = NULL;
```

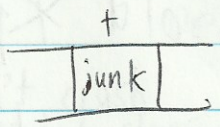


↑ a constant

pointers carry type

```
int x = 74;
char c = 'r';
float z = 3.14;

int *p = NULL;
p = &x;
p = &c;  ⊗
p = &z;  ⊗

char *q = NULL;
q = &c;
float *r = NULL;
r = &z;
int *s = p;
```

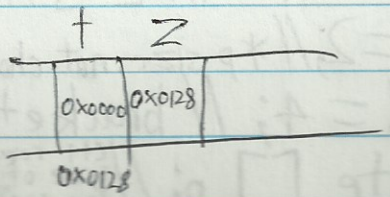
& — address of




```
char *t;

t = &t;  ⊗
char*  char**
```

with operator=, both sides must be the same type

```
int a[5];
int *v = &a[3];

char *t = NULL;
char **z = &t;
```



dereference, *

```
cout << *q;        r
cout << *p;        74
*q = 'm';
*p = 3 * *p;
```

~~new <type> operator~~

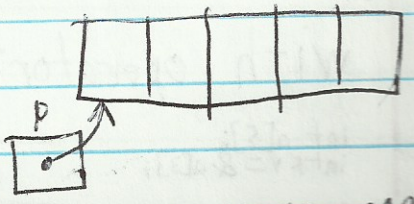
- new
- delete

~~new~~ new <type> -creates anonymous variable
returns pointer (dynamic memory)

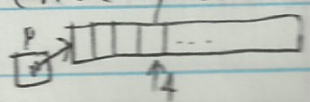
```
p = new int;
q = new float;

p = new int; // old *p lost :(
```

delete p; -deletes what p points to
p = NULL; -good practice

```
p = new int[5];
```



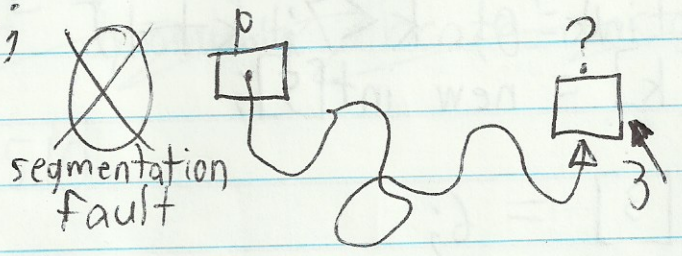

```
p = new int[x]; // dynamic array variable size!  0<x<999999
x *= 2; // *p size not changed
p[3] = 4; // brackets dereference pointer
delete [] p; // ctt remembers the size of the p array. You can't access it.
p = NULL; // good practice
```



Errors
— Dangling pointer
— Memory leak

Dangling pointer — when a pointer with an invalid address is used

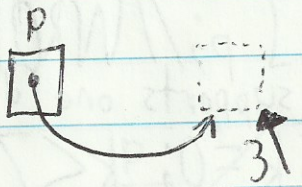
```
int *p;
*p = 3;
```




```
p = new int;
delete p;
*p = 3; // doesn't crash
```

very difficult to trace

wild pointer



Memory leak — when a dynamic variable becomes unreachable

```
q = new float;
q = &y; // doesn't crash
```

if repeated, slows computer

programs that slow to a crawl after running for long periods have memory leaks

1/25/2016

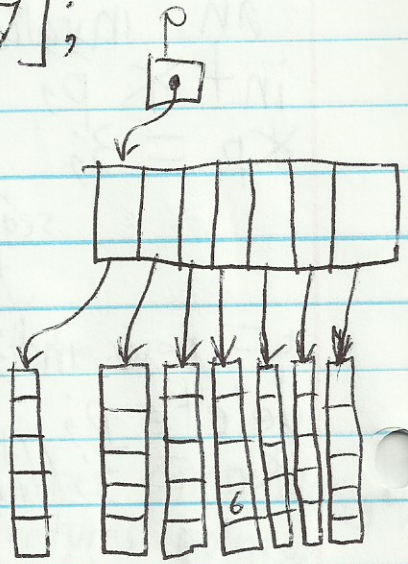
c++ pointers

ACM
- SIG-COMP 6-7 we cs204
- SIG-GAME 3:30-4:30th cs207

- new only supports one dimension

fails → `int* p = new int[7][5];`

```
int** p = new int*[7];
for( int k=0; k<7; k++)
  p[k] = new int[5];

p[3][3] = 6;
```



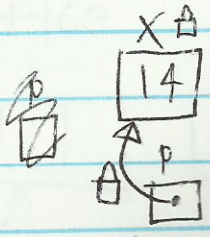

```
delete [] p;//WRONG
```

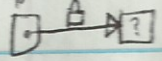
- delete only supports one dimension

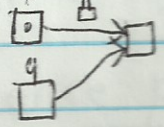
```
for( int k=0; k<7; k++)
   delete [] p[k];
delete [] p;
p = NULL;
```

- const

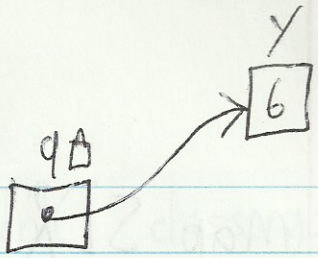
```
const int x = 14;
int* p =&x;
const int* p = &x;
p = new int;
int *q = p;
```



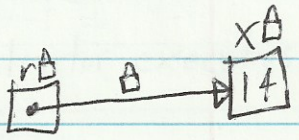




```
const int x = 14;
int y = 6;
```



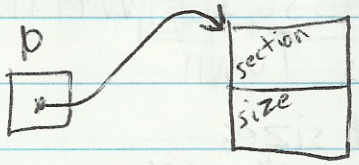

```
int* const q = &y;
*q = 7;
```




```
const int* const r = &x;
```

~~You cannot dynamically allocate constants~~

```
class cs1510
{ public:
    char section;
    int size;
};

cs1510* p;
p = new cs1510;
```



~~*p.section = 'a';~~
*p.section = 'a';

```
(*p).section = 'a';
p->section = 'a';
```

Map SDRIVE on personal computer
— see cs1001 presentation 1

emacs save cx cs
exit cx cc

- Homework #1 Posted
- LEAD Tu/Th 3:30–5:00 CS213

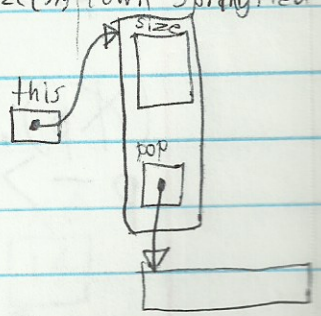
Classes with Pointers

class Town
{
  int size;
  Person* pop;
  void setsize(~~int n~~) hidden Town* const this, int n)
  { this->size = n; // works
  }
  Town(){} //hidden
  ~Town(){} //hidden
  Town(const Town& rhs){...} //hidden
  const Town& operator=(const Town& rhs)
  {....} //hidden

override with pointer members

class Person

Springfield.setsize(5); Town Springfield



destructor is executed when a class goes out of scope

```
{
  Town Rolla;
}// destructor executes
```

copy constructor
- declaration with initialization
- passing objects by value

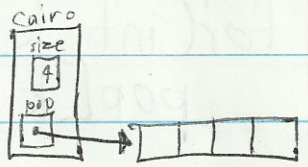
```
foo(Town x)
{
}
```

```
{
  Town Rolla;
  Town Cuba = Rolla; // copy constructor executes
  Town Sedalia(Rolla); // again; this is more proper
  foo(Rolla); // called again

  Town Salem;
  Salem = Rolla; // operator=
```

operator=
- whenever explicitly called

```
class Town
{
  int size;
  Person* pop;
  Town()
  {
    size = 4;
    pop = new People[size];
  }
```

```
{
  Town Cairo;
}// memory leak, *pop lost
```

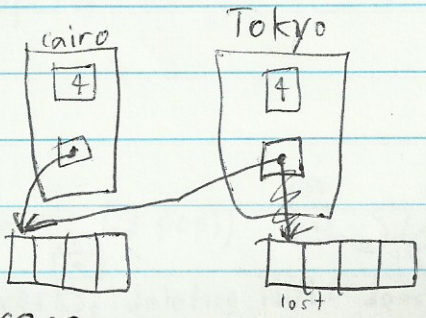


solution: destructor

```
~Town()
{
  delete [] pop;
}
```

```
{
  Town Tokyo;
  Town Cairo
  Tokyo = Cairo; // *Tokyo.pop lost
```

shallow copy




```
}// deletes []*Tokyo.pop twice
```

"double free" followed by pages of binary numbers

Default operator=

```
const Town& operator= (const Town& rhs)
{
   if ( this != &rhs) // alias test
   {
      size = rhs.size;
      pop = rhs.pop; // change this line
   }
   return *this;
}
```

to this

```
delete [] pop;
pop = new Person[size];
for(int k=0; k<size; k++)
   pop[k] = rhs.pop[k];
```

1/30/2017

Hw1 due Friday

./a.out < in.txt

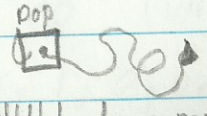
Classes with Pointers

Big-3 (new C++ has 5 things you must override)

- Destructor
- Copy Constructor
- Op= operator=

```
class Town
{
  int size;
  People* pop;
  const Town& operator=(const Town& rhs)
  {
    if( this != rhs) //alias test
    {
      this->size = rhs.size;
      delete [] pop;
      pop = new People[size];
      for( int k=0; k<size; k++)
        pop[k] = rhs.pop[k];
    }
    return *this;
  }

  Town(const Town& rhs)
  {
    *this = rhs; (delete [] pop;)
    this->pop = NULL;
    *this = rhs; // works, deleting NULL does nothing
  }
```

makes copy more explicit (as opposed to pointer copy)



Abstract Data Types & Data Structures

Planning
Organizing Data

ADT — a mathematical object together with operations

Data Structures — a particular implementation of an ADT

ADT	DS
integer	int
real number	float
set	
string	string
function	
graph	
list	
stack	
queue	
tree	
map	

what is the sum of the digits of 100!

Building Blocks

classes
arrays
pointers

a List of size n is a sequence of elements (usually of the same type)
$\langle a_0, a_1, a_2, \ldots a_{n-1} \rangle$
if n=0, empty list $\langle\rangle$

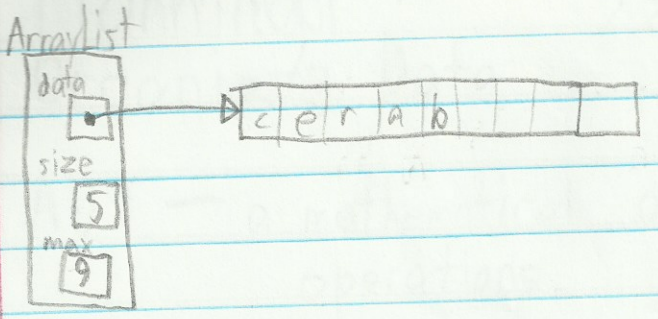
the elements are linearly ordered
- every element has a position
- every element has a predecessor (except a_0)
- every element has a succesor (except a_{n-1})

operations $l_1 = \langle c, e, r, a, b \rangle$
- size(l_1) = 5
- front(l_1) = c
- back(l_1) = b
- at(l_1, 3) = a
- find(l_1, a) = true

How many sig figures do float and double have

- insert(l_1, 2, 'k') = $\langle c, e, k, r, a, b \rangle$
- remove(l_1, 3) = $\langle c, e, r, b \rangle$
- insert_back(l_1, 'p') = $\langle c, e, r, a, b, p \rangle$

an array is a Data Structure
a list is a mathematical object



You cannot dynamically resize an array

```
template<typename T>
class ArrayList
{
  T *m_data;
  int m_size;
  int m_max;

  ArrayList()
  {
    m_data = new T[4];
    m_size = 0;
    m_max = 4;
  }

  void insert_back( const T& x )
  {
    if( m_size != m_Max)
    { m_data[m_size] = x;
      m_size++;
    }
    return;
  }

  int size() const
  {
    return m_size;
  }

  const T& at( const int pos) const
  {
    if ( 0 <= pos && pos < m_size)
      return m_data;
  }

  const T& front() const
  {
    if ( m_size > 0)
      return m_data[0];
  }
```

2/3/2017

```
ArrayList<char> l;
l.insert_back('g');
l.insert_back('r');
cout << l.front();
```

max = 4
size = 0
data = new char[4]
size++

homework #2 posted

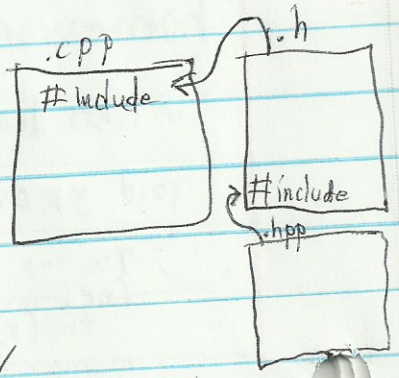
in class ArrayList

```
void grow()
{
  T* tmp = new T[m_max * 2];
  for(int k=0; k<m_size; k++)
    tmp[k] = m_data[k];
  m_max *= 2;
  delete [] m_data;
  m_data = tmp;
  //delete tmp; //WRONG
  return;
}

void insert_back(const T& x)
{
  if(m_size == m_max)
    grow();
  m_data[m_size] = x;
  m_size++;
  return;
}

void insert(const T& x, const int pos)
{
  if(m_size == m_max)
    grow();
  if(pos <= m_size && 0 <= pos)
  {
    for(int k=m_size-1; k >= pos; k--)
      m_data[k+1] = m_data[k];
    m_data[pos] = x;
    m_size++;
  }
  return;
}
```

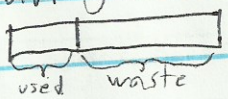
```
if (m_size < m_max/4)
  shrink(); // halves size, leaving extra space
```

"separation of concerns"

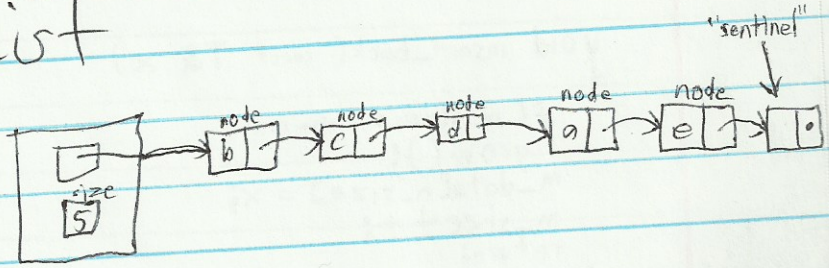
2/6/2017



ArrayList drawbacks

- inserting and deleting is inefficient
- having to get a new array
-  alot of wasted memory

D.S. LinkedList




```
class Node
{public:
  T m_data;
  Node* m_next;
}

class LinkedList
{
  Node* m_head;
  int size;
  LinkedList()
  {
    size = 0;
    m_head = new Node;
    m_head->m_next = NULL;
  }
  void insert_front( const T& x )
  {
    Node* tmp = new Node;
    tmp->m_next = m_head;
    tmp->m_data = x;
    m_head = tmp;
    tmp = NULL;
    m_size++;
  }
```

2/10/2017

valgrind - check for memory leaks
$ valgrind [program to execute]

$$\int \arcsin x\,dx = x\arcsin x + \sqrt{1-x^2} + C$$
$$\int \arccos x\,dx = x\arccos x - \sqrt{1-x^2} + C$$
$$\int \arctan x\,dx = x\arctan x - \frac{1}{2}\ln(1+x^2) + C$$

Next Friday Test #1

D.S. LinkedList

```
const T& at(int pos)
{
  if(pos < m_size)
  {
    for
    Node* p = m_head;
    for(int k=0; k<pos; k++)
      p = p->m_next;
    return p->m_data;
  }
}

Node* at_ptr(int pos)
{
  if(pos < m_size)
  {
    Node* p = m_head;
    for(int k=0; k<m_size; k++)
      p= p->m_next;
    return p;
  }
}
```

```
void insert( Node* p, const T& x)
{
  Node* tmp = new Node(*p); //or, with no constructor, Node* tmp = new Node; tmp->m_data = p->m_data; tmp->m_next = p->m_next;
  p->m_data = x;
  p->m_next = tmp;
  m_size++;
}
```

1) create new Node
2) copy Node pointed by p to new node
3) insert x and chain new node
4) increment size

```
bool find( const T& x)
{
  Node* p = m_head;
  while(p->m_next != NULL)
  {
    if(p->m_data == x)
      return true;
    else
      p = p->m_next;
  }
  return false;
}
```

What if it's the last element (before the sentinel)?

```
Node      //here's an option
{
  T m_data;
  Node* m_next;
  Node* m_prev;
```

D.D. DoublyLinkedList (4 pointers to take care of with insert)

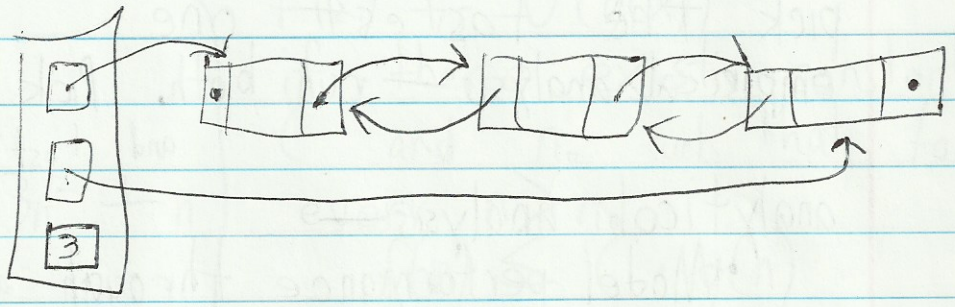
```
void insert( Node* p, const T& x )
{
  Node* tmp = new Node;
  if(0 Back
  for(
  p->m_prev = tmp;
  tmp->m_data = x;
  tmp->m_next = p;
  tmp->m_prev = p->m_prev;
  Fab
  if((tmp->m_prev)).m_next = tmp;
  tmp->m_prev->m_next = tmp;
  p->m_prev = tmp;
}
```

2/10/2017

Test #1 — 1 week from today
50 min

- class notes from website
- know about homework

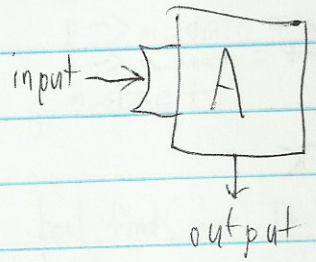
D.S. Doubly Linked List

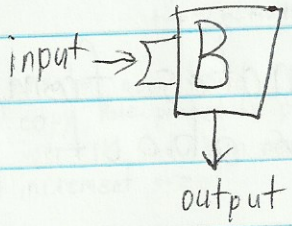



```
void insert_back( const T& x)
{
  Node* tmp = new Node;
  m_back->m_next = tmp;
  m_back = tmp;
  m_back->m_next = NULL;
  m_back->m_data = x;
  m_size++;
}
```

// fails for empty list

Algorithm Complexity (an intro)





pick the fastest one

empirical analysis — run both. Pick "bench marks" and test both programs

analytical analysis —

- Model performance through a mathematical object

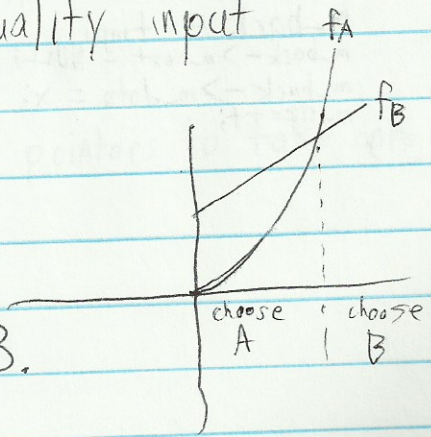
runtime function — returns time

$f(\text{size of input}) \rightarrow \text{time}$

- assume worst quality input

$f_A(n) = \frac{1}{2}n^2 + 3$

$f_B(n) = 20n + 123$



But, in general, choose B.

we're not interested in values, but the R.O.G.

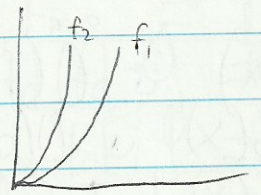
Test #1 this Friday

runtime function
- rate of growth: Big-O

Big-O — given functions $f(x)$ and $g(x)$ we say that $f(x)$ is $O(g(x))$ if there exists constants C and n_0 such that for every $n > n_0$

$$f(n) \leq C \cdot g(n)$$

$f_1(n) = n^2$
$f_2(n) = 3n^2 + n$



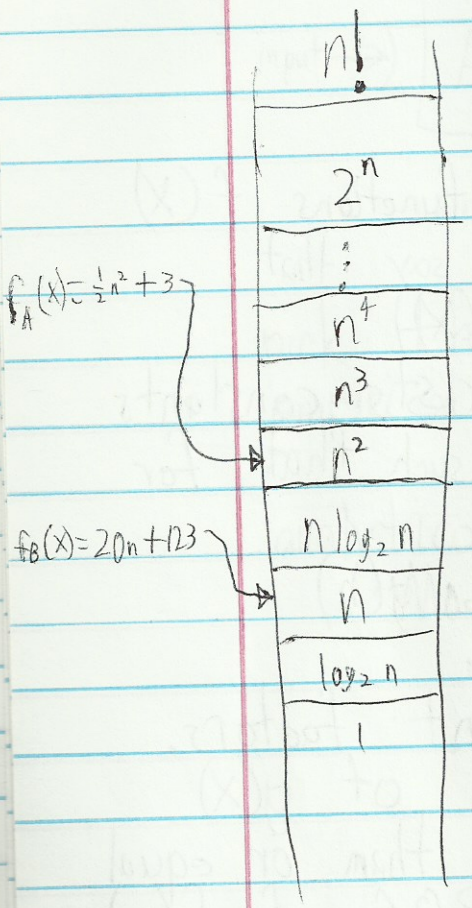
ignoring constant factors, the R.O.G. of $g(x)$ is greater than or equal to the R.O.G. of $f(x)$

f_1 is $O(f_2)$
f_2 is $O(f_1)$

Big-Θ
$f(x)$ is $\Theta(g(x))$ if
$f(x)$ is $O(g(x))$ and
$g(x)$ is $O(f(x))$

R.O.G.

~~ladder~~ hierarchy



Rule 1
if $T_1(x)$ is $O(f(x))$
and $T_2(x)$ is $O(g(x))$
then $T_1(x) + T_2(x)$
is $O(f(x)+g(x))$

Rule 2
if $T_1(x)$ is $O(f(x))$
and $T_2(x)$ is $O(g(x))$
then $T_1(x) \cdot T_2(x)$
is $O(f(x) \cdot g(x))$

Rule 3
if $T_1(x)$ is $O(f(x))$
and $T_2(x)$ is $O(g(x))$
then $T_1(x) + T_2(x)$
is $O(\max(f(x), g(x)))$

Rule 4
A polynomial of degree k is $O(n^k)$

Getting runtime functions
— count operations

0) The cost of a statement is the # of operations in the statement

```
void Arraylist::swap(int i, int j)
{
   T tmp;
   tmp = data[i];
   data[i] = data[j];
   data[j] = tmp;

   return;
}
```

cost: 7 = O(1)

```
foo(int n, int k)
   int x;
   if(n==0)
      x=0;
   else {
      x = k*k;
      x = x/n;
   }
   return x;
}
```

assume worst quality input
cost = 5 = O(1)

```
sum(int a[], int n)
    int s=0;
    for(int k=0; k<n; k++){
        s = s + a[k];
    }
    return s;
}
```

Cost of for loop

$$\text{init} + \sum_{k=0}^{n-1} (\text{check} + \text{update} + \text{body})$$

1 1 3

$$\text{for loop} = 1 + \sum_{k=0}^{n-1} (5) = 1 + 5n$$

- while loops

```
while(test)
    body
```

1) assume it terminates
2) estimate # of iterations

```
LinkedList * LinkedList::find(T x)
{
    LinkedList * p = this;
    while( p->next != NULL){
        if(p->data == x){
            return p;
        }
        p = p->m_next;
    }
    return NULL;
}
```

still reachable — memory leaks

iostream makes memory leaks

"no leaks are possible" — no memory leaks

concentrate on "definitely lost"

runtime of a while loop = n(body + test)
↑ # of iterations

n: size of LinkedList
in worst case, find repeats n times

```
while( p->next != NULL) {        cost of while loop
  if(p->data == x)               6n+1
    return p;
  }
  p = p->m_next;
}
return NULL;
```

```
for( int k=0; k<n; k++) ——  6n^2+3n+1
{
  for( int j=0; j<n; j++)
  {
    s = s + a[k][j];  -4        6n+1
  }
}
```

$6n^2+3n+2$

$O(n^2)$

Test #1 review

- C++ pointers & * ->
  new delete
  - destructor
  - copy constructor
  - operator=
  - memory leaks
  - dangling pointers

- Abstract Data Types & Data Structures

- ADT List

- D.S. ArrayList

- LinkedList (D.S.)
  - DoublyLinkedList
    (remove fn)

- Algorithm Complexity
  - Definition of Big-O

    if $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$
    then $f(x)$ is $\Theta(g(x))$

  - Complexity Hierarchy
  - Place functions in Hierarchy

— Extract a runtime fn from code

Study guides in website

Algorithm Complexity

Complexity of a Problem

"the problem has a complexity of f(n)"
— the best program we know of to solve the problem has a complexity of f(n)"

$n!$
2^n
⋮
n^4
n^3
n^2
$n \log_2 n$
n
$\log_2 n$
1

Moore's law

Wirth's law — software gets slower faster than hardware gets faster

	1000 $\frac{op}{min}$	10,000 $\frac{op}{min}$	x10
$33n$	30	300	10x
$6n^2$	12	40	4x
$2n^3$	7	17	2x
$n!$	6	7	1.15x

m_back — last element, NOT sentinel

2/22/2017

Re-download ~~Linf~~ linkedlist.h

const Node<T> * getFirstPtr() const;

Inheritance

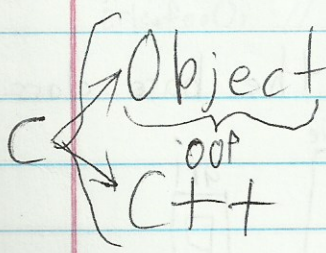
in Object Oriented Programming

C was for Unix

Simula

Alan Kay: Smalltalk

- encapsulation <public: private:
- inheritance
- polymorphism



Inheritance — the ability to declare a class as an extension of another class

```
class A
{
  int x;
public:
  int y;
  void foo();
};
```

"Base class" 2 variables

```
class B : public A
{
public:
  int z;
  void bar();
  void foo();
};

void B::bar()
{ x=0; }   x is private
```

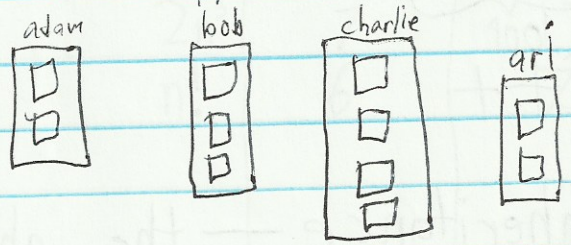
"derived class"

3 variables

```
class C : public B
{
public:
  int w;
};
```

4 variables

- A derived class has the type of the base class




```
A adam;
B bob;
C charlie;
A ari;

ari = adam;
ari = bob;
bob = ari;
zap(A& a)
{
}

zap(charlie);
zap(B& b)
{
}
zap(charlie); //calls zap-B
```

```
A* p;
p = &bob;
p = &charlie;
p->y = 6;
p->z = 7;
```

bob has 2 foo() fns

```
bob.foo(); // foo-B
p->foo(); // foo-A
bob.A::foo(); // foo-A
```

```
bob.bar();
```

solution

```
class A
{
  protected:
    int x; // accessible in B and C, not accessible elsewhere
```

```
class Platypus : public Duck, public Otter, public Spy
{
  :
};
```

```
Platypus Eric;
Eric.foo(); // 1. Platypus foo() 2. Duck foo() 3. Otter foo() 4. Spy foo()
```

searcer in this order

hw3

throw an error if you are given a swap with the same IDs

C++ Inheritance

- Constructors and Destructor are NOT inherited

```
class A
{
protected:
  int x;
  int y;
  A() {
    x=0;
    y=0;
  }
  A(int i, int j)
  {
    x=i;
    y=j;
  }
```

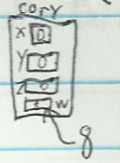
```
class B : public A
{
  int z;
  B() { // calls A() before execution
    z=0;
  }
```

same

```
  B(): A(0,0)
  {
    z=0;
  }
```

```
class C : public B
{
  int w;
  C() { //calls A(0,0), B(), etc. before execution
    w=8;
  }
```

```
B bob(3, 7);
C cory;
```



- the constructor of a derived class calls automatically a constructor of a base class

- the Destructor of a derived class automatically calls the dotructor of a base class

```
class A
{
protected:
  int * x;
~A(){
  delete [] x;
}
```

```
class B: public A
{
  int * z;
~B() // calls ~A(), probably after execution
{
  delete [] z;
}
```

```
class C: public B
{
  int a;
};
// when a C is destructed,
// ~B() and ~A() are called
```

Polymorphism

```
B bob;
A * p = &bob;
p->foo(); // calls A::foo()
          not B::foo()
```

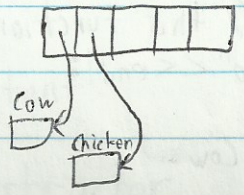
static type: A
dynamic type: B

static vs. dynamic type — only appears with inheritance and pointers

poly morphism — use the dynamic type of objects rather than their static type

class Animal class Chicken : public Animal class Cow: public Animal

```
Animal Farm[5]; // DOES NOT ALLOW CHICKENS AND COWS
Animal * Ranch[5]; // ALLOWS CHICKENS AND COWS

Ranch[0] = new Cow;
Ranch[1] = new Chicken
```




```
for(int k=0; k<2; k++)
  Ranch[k]->speak();
```

⊗ Animal does not have a speak function

hw3

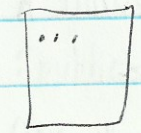
alias test

operator=

if we give Animal a speak function

```
class Animal
{
  void speak()
  { cout << "..." << endl;
  }
```

Ranch[0]->speak();



virtual

```
class Animal
{
 virtual void speak()
 {
   cout << "..." << endl;
 }
 //you cannot have virtual variables
```

Ranch[0]->speak(); Moo static type: Animal

Ranch[1]->Animal::speak(); ...

static — explicitly named

dynamic — anonymous

in Java, all functions are virtual

"virtual" is inherited

```
class Cow:public Animal
{
  void speak() // this function is virtual
  {
    cout<< "Moo" <<endl;
  }
};
class madCow:public Cow
{
 void speak()
 { cout << "HaHa!" << endl; }
```

```
Ranch[2] = new mad Cow;
Ranch[2]->speak();
```


```
class Animal { // "Abstract class"
  virtual void speak() = 0; // "pure virtual"

Animal Bob;

class Fox: public Animal
{
  // no one knows what the Fox says
}

Ranch[4] = new Fox;
```

if a class only has pure virtual functions, it is called an interface

```
class Cow : public Animal
{
  virtual void speak()
  {
    cout << "Moo" << endl;
  }
```

good practice

Make Destructors Virtual

```
class Animal
{
  virtual ~Animal()
```

very important

you cannot have a virtual constructor

3/1/2017

A.D.T. Stack

a stack is a sequence

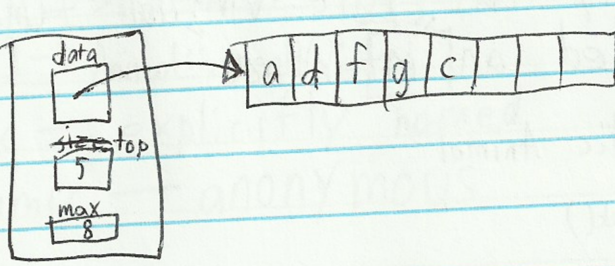
one end is the "top" $s_1 = \langle a, d, f, g, c \rangle$ (top → c)

operations
- top(s)
- push(s, x)
- pop(s)

these are the only operations

top(s_1) = 'c'
push(s_1, 'e') = ⟨a, d, f, g, c, e⟩
pop(s_1) = ⟨a, d, f, g⟩

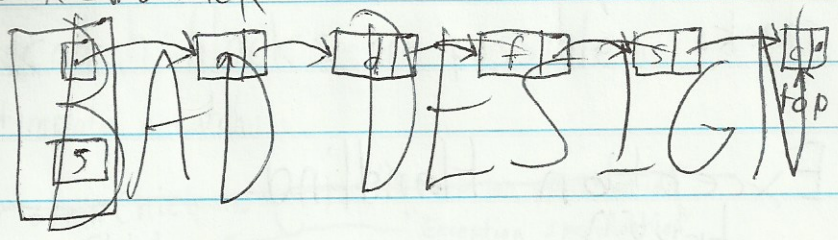
D.S. ArrayStack

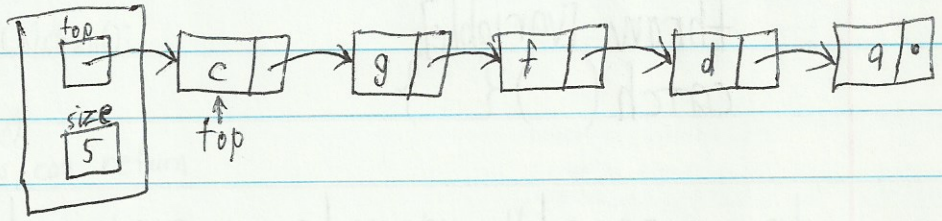



```
T& top()
{
  if(m_top != 0)
    return m_data[m_top];
  else
    error();
}
```

```
void push( const T& x)
{
  if( m_max == m_top)
    grow();
  m_data[ m_top ] = x;
  m_top ++;
}
```

D.S. LinkedStack



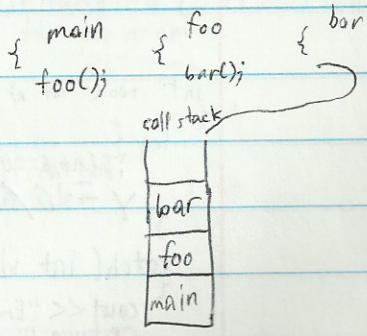


~~top~~

```
void top()
{
  if( m_head != NULL)
    return m_head->m_data;
  else
    error();
}
```

```
void push(const T& x) // LinkedList::insert_front
{
  Node<T>* tmp = new Node<T>(x, m_head);
  m_head = tmp;
  m_size++;
}
```

```
void pop() // LinkedList::remove_front
{
  if( m_head != NULL)
  {
    Node<T>* tmp = m_head;
    m_head = tmp->m_next;
    delete tmp;
    m_size--;
  }
  else
    error();
  return;
}
```



- balanced brackets (application of stack)
  ([] ✗
  [)] ✗
- function call stack (application of stack)

hw4 posted

stacks

Exception Handling

try{ }
throw [variable]
catch( ){ }

to separate regular execution from exceptional execution

```
int foo( int n)
{
  =
  if((n-6) != 0)
      y = a/(n-6);
  else
  {
    cout << "Error!"; } exceptional code
    return 0;
  }
  =
}
```

```
int foo( int n)
{
  try{
      if(n-6==0) throw n;
      y = a/(n-6)
  }
  catch( int x)
  {
    cout << "Error!";
    return 0;
  }
```

multiple catch blocks are possible

you cannot template a catch

~~// throws string~~ Chicken ← ~~"Exception specification"~~

foo() throw Chicken ← "Exception specification"

```
{
  throw Chicken();
}
catch(int x)
{ //catches can return
}
// no Chicken catcher, throw goes to caller
//
```

```
bar()
{
  try
  {
    foo(); // throws Chicken, if foo() throws, it does not return
  }
  catch( Chicken c) // catch( Animal a) would also catch Chickens
  {
  }
```

// if ~~no c~~ a throw is not caught, an unhandled exception crashes the program

int zap() throw string, int, grenade ← Exception specification ignored by compiler

catch(...) // syntax "master catch" "default catch"

```
{
  :
}
```

exception handling makes execution jump around, so only use it for error handling

```
#include <stdexcept>
```

- logic_error
  - invalid_argument
  - domain_error
  - length_error
  - out_of_range_error
- runtime_error
  - range_error
  - overflow_error
  - underflow_error

A.D.T. Queue

A queue is a sequence

$$\langle q_0, q_1, q_2, \ldots q_n \rangle$$

Designate one end the "front" and the other end the "back"

- front()
- ~~end~~
- enqueue()
- dequeue()..

front ↘ $q_2 = \langle c, b, f, e, d \rangle$ ← back

~~enqueue(q_2) = $\langle c, b, f, e,$~~

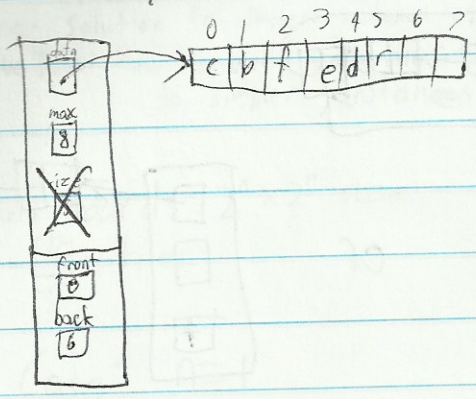
enqueue(q_2, 'r') = $\langle c, b, f, e, d, r \rangle$ (insert_back)

dequeue(q_2) = $\langle b, f, e, d \rangle$ (remove_front)

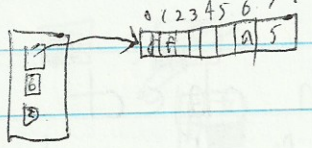
front(q_2) = c

"FIFO"

D.S. Array Queue



circular array



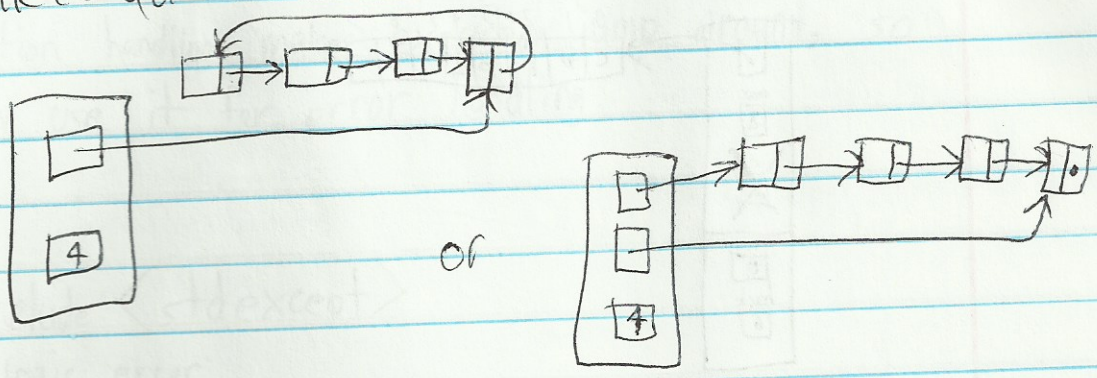
enqueue
{
m_back = (m_back + 1) % m_max;

enqueue
{ m_back = (m_back + 1) % m_max;
if (m_back == m_front)
grow();

++ is no longer faster than = +1
(compilers can detect the situation)

void dequeue()
{ if (m_front != m_back)
m_front = (m_front + 1) % m_max;

linked Queue



3/8/2017 # Recursion

recursive object — an object which ~~is~~ consists or is defined in terms of itself

— with Mathematical Objects
— Base Case (simplest object or smallest object)
— Recursive Rule — How to obtain complex objets from simpler ones

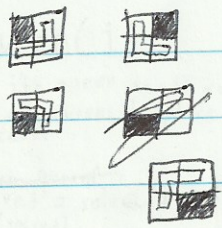
$P = \{2, 4, 8, 16, 32, \ldots\}$
Base Case: 2 is in P
Recursive Rule: $n \in P \Rightarrow 2n \in P$

Solutions to Problems (Algorithms)
- Base Case: solution to simplest instance of problem
- Recursive Rule: How to build solutions to complex instances from solutions to simpler instances

cover chess board $2^n \times 2^n$ size
by  tiles
2

Base Case: $n=1$

2×2



Recursive Rule:

Suppose a $2^n \times 2^n$ board with one hole can be solved

consider a $2^{n+1} \times 2^{n+1}$ board

the hole is in one of it's four quadrants

that quadrant is $2^n \times 2^n$, so it can be covered by tiles

place a tile in the center of the $2^{n+1} \times 2^{n+1}$ board across the three quadrants without holes

now, those quadrants can be solved by the hypothesis

recursive function — a function that calls itself

```
void foo()
{
  int x; // local only to this call.  Each call gets its own x.
  foo();
}
```

Fibonacci
f(0) = 1
f(1) = 1
f(n) = f(n-1) + f(n-2)

```
int fibo( int n)
{
  if( n==0 || n==1)
    return 1;
  else
    return fibo(n-1) + fibo(n-2);
}
```

any recusive function can also be written with iteration

```
int exp(b, p)
{
  int x = 1;
  for(int k=0; k<p; k++)
    x *= b;
  return x;
}
```

O(p)

```
pow( int b, int p)
{
  if (p==1)
    return b;
  else
    if p is even
      return 2*pow(b, p/2)
```

$O(\log_2 p)$

Recursive Backtracking

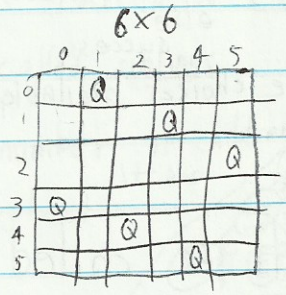
- Some problems cannot be solved with a fixed rule computation

n-queens problem

in a chess board of nxn
place n queens so that they do not attack each other

strategy: decompose into sequence of trial & error tasks

```
solve_queen(i)
repeat
  place ith queen in ith row (in valid spot)
  if no more queens to place
    success!
  else
    ~~solve_queen~~
    solved = solve_queen(i+1)
    if(solved)
      success!
    ~~else~~
while places available for i queen
fail
```



Board D.S. (array of size n)

row	0	1	2	3	4	5
col	1	3	5	0	2	4

a place is invalid if it is the same col as a previous queen or
b[i]-i == b[j]-j
or
b[i]+i == b[j]+j

```
bool solve_queen( int row, Board b, int n )
{
  for( int col=0; col < n; col++)
  {
    if( valid( row, col, b ) )
    {
      record( row, col, b )
      if( row == n )
        return true;
      else
      {
        solved = solve_queen( row+1, b, n)
        if( solved )
          return true;
        else
          undo( row, col, b)
      }
    }
  }
  return false;
}
```

~~HW 4~~

Recursive Backtracking

```
solve
    initialize choices
    do
        select choice
        if choice is valid
            record choice
        ~~if solution incomplete~~
            if solution incomplete
                s = solve next step
                if not s
                    undo recording
            ~~else~~ else
                success
        else
            success
    while choices available
    fail
```

used to be called "AI" — not anymore

practice — code n-queens problem before exam

A.D.T. Tree

- A tree is a collection of elements with a hierarchical relationship between the elements

- a single element ("node") is a tree
- if n is an element ("node") and $T_1, T_2, T_3 \ldots T_k$ are trees then n related to $T_1, T_2, T_3, T_4 \ldots T_k$ is a tree (recursive)
  n is called the root and $T_1 \ldots T_k$ are called subtrees of n

every tree has ~~a root~~ exactly one root

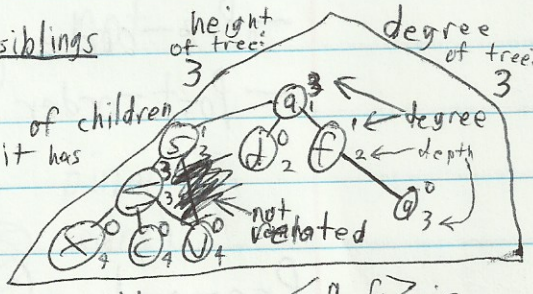
the root of each subtree of n is called a child of n and n is called the parent

nodes that have the same parent are called siblings

a leaf is a node with no children

the degree of a node is the number of children it has

the degree of a tree is the highest degree of a node



tree has degree 5 — a node has at most 5 children

<q, f> is a path

a path is a sequence of nodes: $< n_0, n_1, n_2, \ldots, n_k >$
where n_{i+1} is the parent of n_i $0 \leq i < k$
(so we start from the bottom)
Note: there is only one path from any node to the root

the depth of a node n is the number of nodes in the path between n and the root

the height of a tree is the length of the longest path in the tree

the root of a tree has no parent

3/15/2017

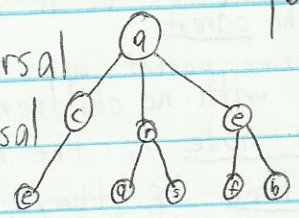
Homework #4 due today

Test #2 Next Friday

A.D.T. Tree

Tree traversal

- Pre-order traversal
- Post-order traversal



pre-order output
acerqsefb

post-order
ecqsrfbea

```
Pre-order ( tree t)
    process(root of t)
    for every subtree t' of t
      pre-order(t')
Post-order( tree t)
    for every subtree t' of t
      post-order( t')
    process( root of t)
```

~~iter~~

```
iter_pre_order( tree t)
    stack s
    push root of t
    while s is not empty
      x = top of s
      process x
      pop s
      for every child y of x
        push y into stack
```

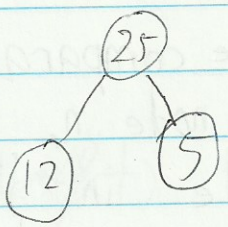
what would happen if queue is used rather than stack
(it's not post-order)
(it's "breadth first")

ADT BinarySearchTree

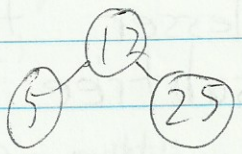
degree 2 – "binary"

search – elements are comparable

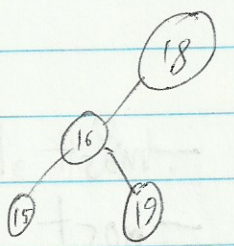
– for every node in BST, n is greater than any node in its left subtree and n is lesser than any node in its right subtree



binary, not search



binary search



binary not search

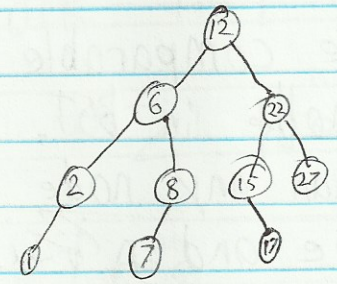
3/20/2017 Monday

Test #2 Friday

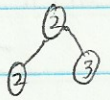
Tomorrow 9:45 AM CS203 – new CS chair candidate (refreshments)

Binary Search Tree

BST



not BST



search
- elements are comparable
- for every node n, n is greater than any node in its left subtree and n is lesser than any node in its right subtree

operations
- getMin(t) — most left ~~element~~ node in BST
- getMax(t) — most right node in BST
- insert(t, x) — there is always only one place to insert a node of a certain comparable value
- remove(t, x)
- find(~~x~~ t, x) — similar to insert

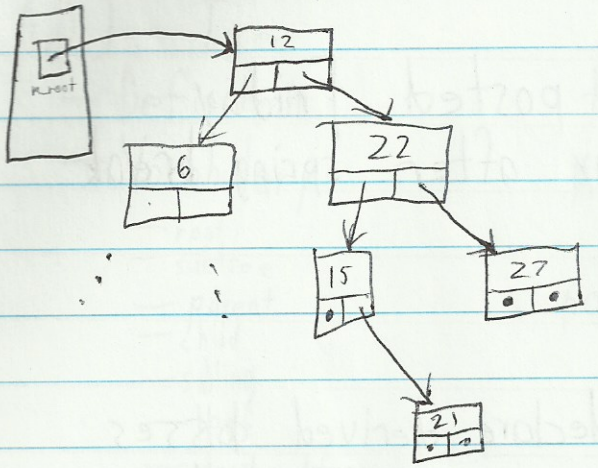
remove(t, x)

Case 1, x is a leaf
delete x from the tree

Case 2, x has 1 child ~~and A parent~~
graft x's child to it's parent

Case 3, x has 2 subtrees
take the max of the left ~~subtree~~ and the min of the right subtree, ~~and find which one works in place~~ pick one, remove that one of x, and insert it in place of x

D.S. BinarySearchTree




```
class TreeNode
{
  T m_data;
  TreeNode* m_left;
  TreeNode* m_right;
};
```

```
class BSTree
{
  TreeNode* m_root;
  int m_size;
};
```

TreeNode* t

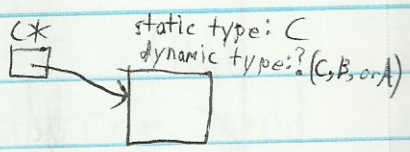
```
getMax( TreeNode*& t)
{
  if( t->m_right == NULL)
    return t->m_data;
  else
  return getMax(t->m_right);
}
```

```
insert( TreeNode *& t, const T& x)
{
  m_size++;
  if( t == NULL)
  {
    t = new TreeNode;
    t->m_data = x;
    t->m_right = t->m_left = NULL;
  }
  else if( x < t->m_data)
    insert( t->m_left, x );
  else
    insert( t->m_right, x);
  else // x == m_data
    m_size--;
}
```

Test #2 Friday

HW 5 will be posted Friday
due Friday after spring break

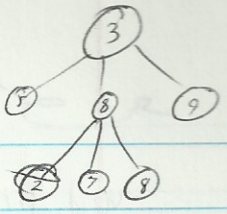
C++
- inheritance — declare derived classes
  - protected:
- polymorphism
  - static vs dynamic



  - virtual
  - pure virtual
  - abstract class
- Exception handling
  try, throw, catch

Recursion
- write recursive functions
  Algorithm vs Code

A.D.T. Stack
D.S. ArrayStack
D.S. LinkedStack
write operation
A.D.T. Queue
D.S. ArrayQueue
D.S. LinkedQueue



A.D.T. Tree

- Definition
- Vocabulary
  - root
  - subtree
  - parent
  - child
  - sibling
  - leaf
  - degree of node
  - degree of tree
  - path – goes up
  - depth of a node
  - height of a tree

traversals pre 3582789
post 5278893

pre-order(tree t)
process root of t
for each subtree t' of t
pre-order(t')

post-order

No BS trees! (in test)

```
class A
{
  ~A() // either written or generated
  {
    =
  }
};
```

```
class B : public A
{
  ~B() // either written or generated
  {
    =
  } // calls A::~A()
};
```

```
void ArrayStack<T>::pop()
{
    if( m_top )
      M_top--;
    return;
}
```

4/3/2017

- Homework 5
  don't copy/paste samples from website
  download
- Google Code Jam (same day as MegaMiner)
- MegaMiner

D.S. BinarySearchTree

```
class Node
{
    T m_data;
    Node * m_right;
    Node * m_left;
};

getMax()
getMin()
find()
insert()

remove()
```

- if no children, simply delete and set pointer to NULL
- if one child, point pointer to child and delete
- if two children, replace with min of right subtree or max of left subtree, removing that and deleting this

```
void remove(const T& x, Node* &t)
{
  if( t == NULL)
    return;
  if( x < t->m_data)
    remove( x, t->m_left);
  else if( x > t->m_data)
    remove( x, t->m_right);
  else // x == t->m_data
  {
    if( t->m_left == NULL
        && t->m_right == NULL)
    { delete t; t=NULL; }
    else if( t->m_left == NULL
      || t->m_right == NULL)
    {
      Node *child = t->m_left;
      if( child == NULL)
        child = t->m_right;
      delete t;
      t = child;
    }
    else // two children
    {
      t->m_data = getMax( t->m_left);
      remove( t->m_data, t->m_left);
    }
```

}
}

4/5/2017
Homework #5 due Friday

MegaMinerAI
siggame.io

ADT BSTree

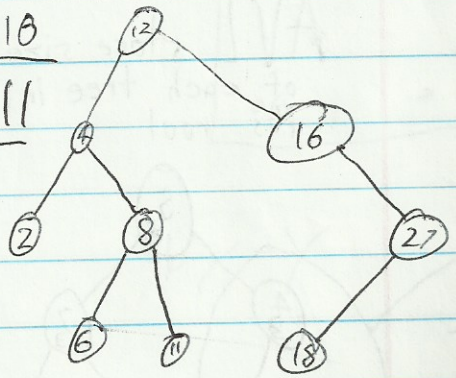
DS Node

DS ArrayBSTree

0	1	2	3	4	5	6	7	8	9	10
12	4	16	2	8	-1	27	\	\	6	11

or

12 something to show absence of an element

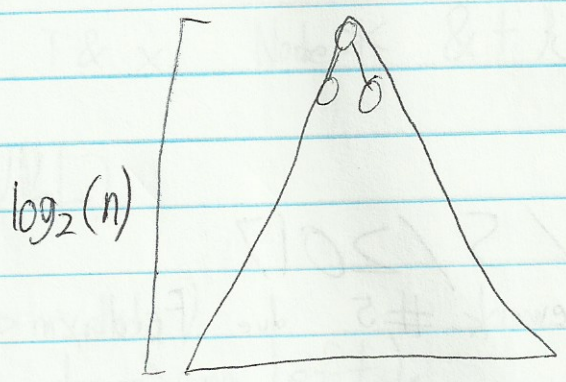


11	12	13	14
/	/	18	/

left(i) = 2i + 1
right(i) = 2i + 2
parent(i) = ⌊(i − 1)/2⌋

n elements in full tree

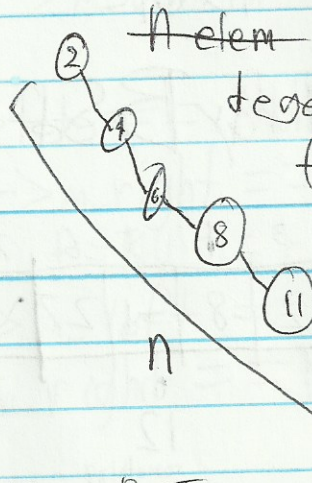
insert()
remove()
find()



Advanced Binary Search Trees
- AVL Tree
- Red-Black Tree
- Splay Tree

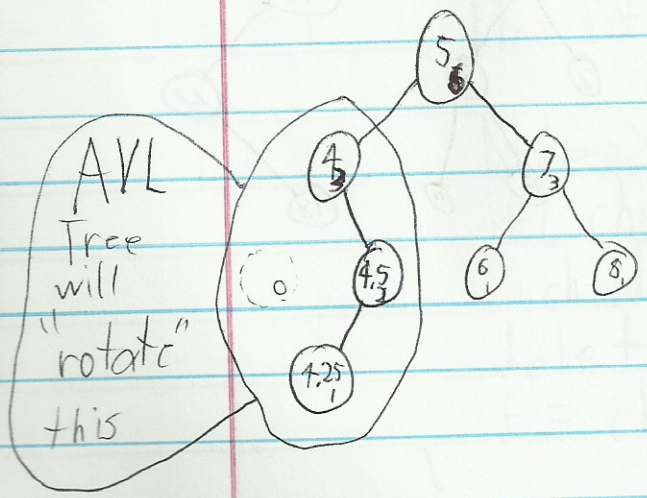
NodeBSTree::find() is $\log_2(n)$

~~n elem~~ degenerate tree



NodeBSTree::find() is n

AVL store size of each tree in its root



AVL Tree will "rotate" this

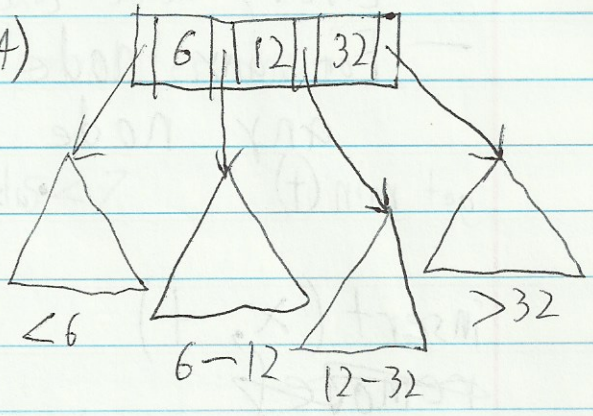
Red-Black "colors" each node and uses the color to prevent degeneracy

Splay Tree rearranges a tree when find() is called (if degenerate)

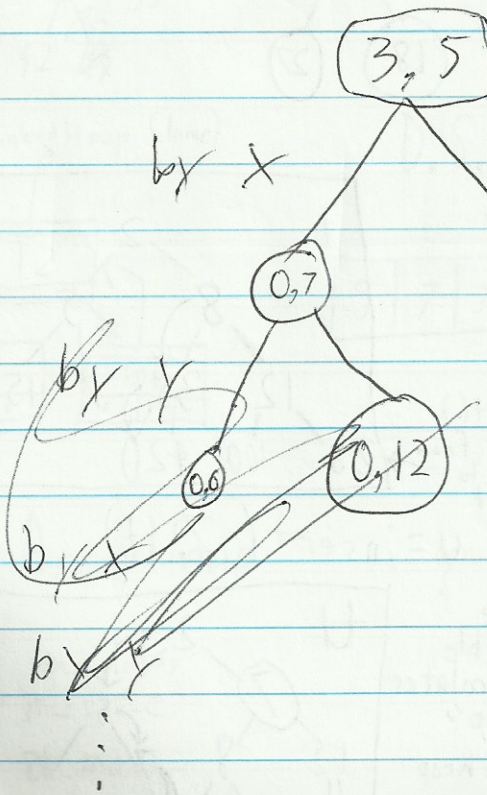
other trees

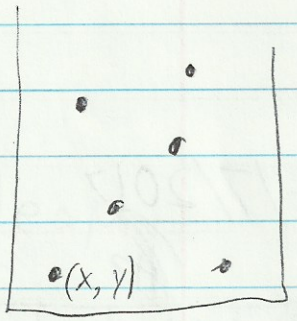
- B-tree

(degree 4)



- KD Tree





by y

by x
by y
by x

ADT Heap

- Binary Tree (not search)
- Every level except bottom is full
- For any node x, x is lesser than (optionally, or equal to) any node in its subtrees

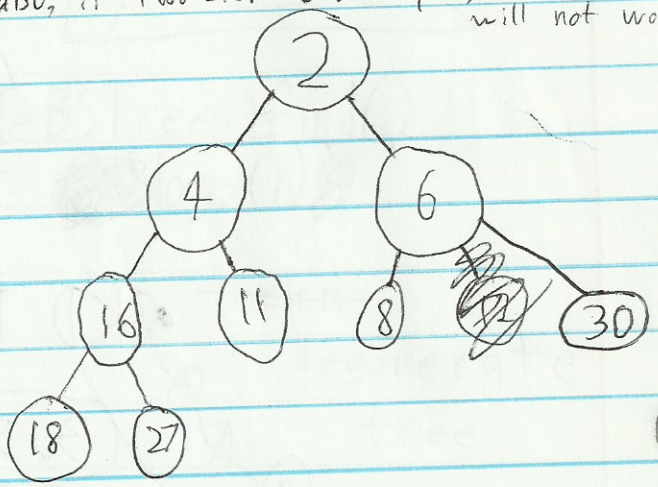
get min(t)

insert(x, t)

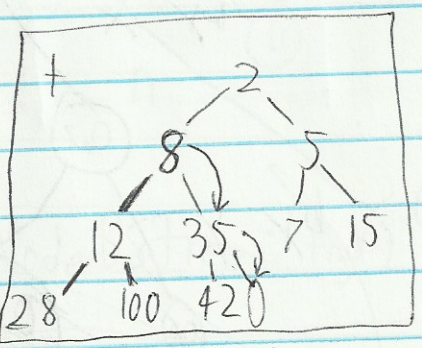
~~remove(t)~~

remove_min(t)

• also, if two elements are equal, the operations will not work

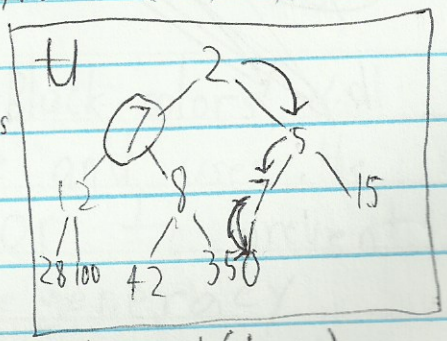


4/7/2017

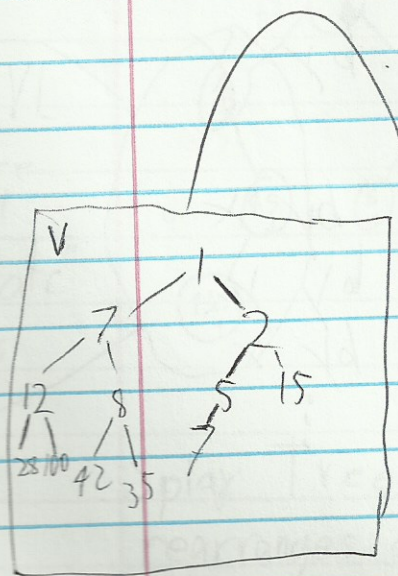


insert() fills from left to right

u = insert(7, t)

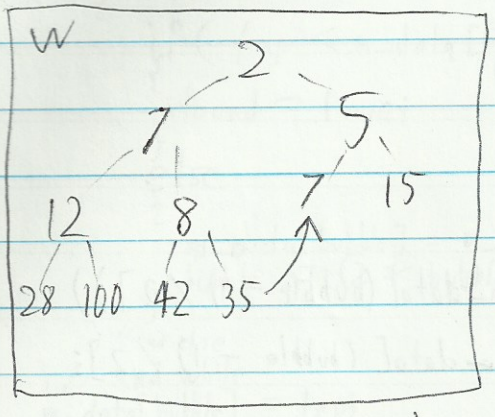


the bubble "percolates up" the heap



v = insert(1, u)

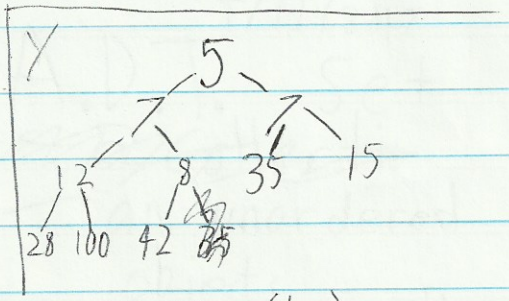
w = remove_min(w)

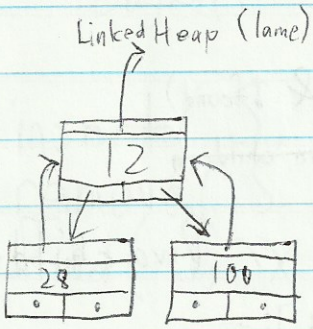


the bubble "percolates down"

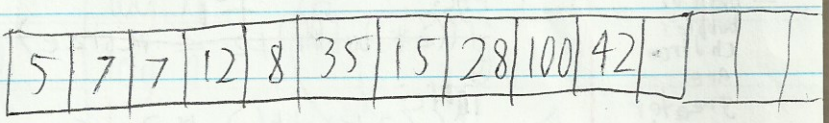
5 < 7

Y = remove_min(w)





D.S. Array Heap




```
class ArrayHeap
{
    T* m_data;
    int m_size;
    int m_max;
};
```

```
const T& getMin()
{
    return
    if(m_size != 0)
        return m_data[0];
    else
        throw error;
}
```

Array Heap

```
void insert(const T& x)
{
  if(m_size == m_max)
    grow();
  int bubble = m_size;
  while((bubble > 1) && (x < m_data[(bubble - 1) / 2]))
  {
    m_data[bubble - 1] = m_data[(bubble - 1) / 2];
    bubble = (bubble - 1) / 2;
  }
  m_data[bubble - 1] = x;
  m_size++;
}
```

```
void remove_min()
{ bool found = false;
  int bubble = 0;
  T& tmp = m_data[m_size - 1];
  m_size--;
  while((bubble * 2) < m_size && !found)
  { int fc;
    if((2*bubble)+2 < m_size) // two children
    {
    int fc;
    if((2*bubble) + 2 < m_size) // two children
    {
      fc = m_data[(2*bubble) + 1];
      if(m_data[(2*bubble) + 2] < fc)
        fc = m_data[(2*bu

      int fav;
      fc = (bubble * 2) + 1;
      if(m_data[(bubble * 2) + 2] < m_data[fc])
        fc++;
      if(m_data[fc] > tmp)
        found = true;
      else
      {
        m_data[bubble] = m_data[fc];
        bubble = fc;
      }
    }
    else
```

while conditions

1. bubble reaches bottom

or

2. both of bubble's children are greater than tmp

equal to tmp

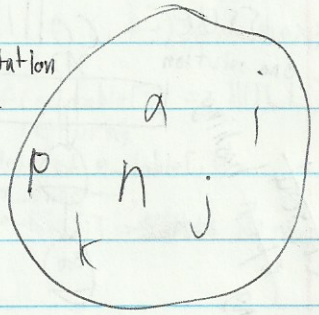
```
{
    if( tmp < m_data[(2 * bubble) + 1])
    {
      found = true;
    }
    else
    {
      m_data[bubble] = m_data[(2 * bubble) + 1];
      bubble = (2 * bubble) + 1;
    }
  } // end else
}// end while
m_data[bubble] = tmp;
}
```

4/12/2017

A.D.T. Set

~~Collection~~

— an unordered collection of non-repeating objects



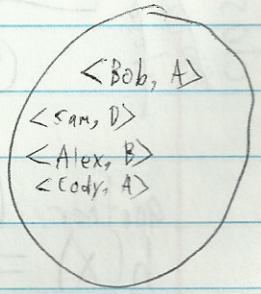
	tree implementation complexity
• insert(S, x)	log n
• remove(S, x)	log n
• find(S, x)	log n

A.D.T. Map

— a set of pairs <k, v>

k = key

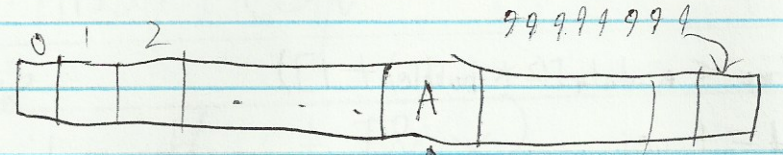
v = value



- all keys must be unique
- values may appear multiple times

	tree implementation complexity	array implementation complexity (impractical due to memory requirements)
• insert(M, k, v)	log(n)	
• remove(M, k)	log(n)	
• findValue(M, k)	log(n)	

ID Number array

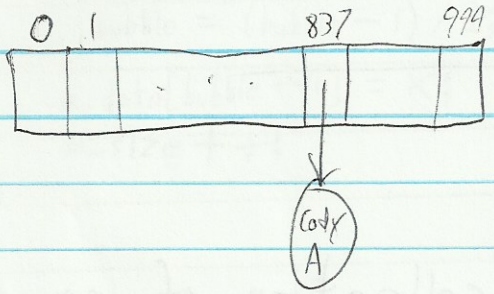


ID #37912837, Cody

better ArrayMap

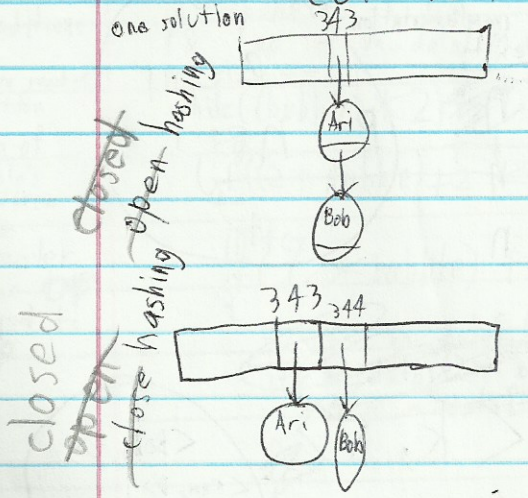
hash function

h(id) = id % 1000



issue: collision

one solution



closed hashing

open hashing

closed

open

close hashing

another hash function

h(x) = middle 3 digits of x^2

```
class student
{
  int m_id;
  string m_name;
  char m_grade;
};

class studentHashTable
{
  student ** m_data;
  int m_size;
  int m_max;
  studentHashTable()
  {
    m_data = new student*[100];
    for(long k=0; k<100; k++)
      m_data[k] = NULL;
    m_size = 0;
  }
```

```
void insert(student & x)
{
  int i = hash(x.m_id);
  while(m_table[i] != NULL && m_table[i]->m_id != x.m_id)
    i = (i+1) % m_max;
  delete m_table[i];
  m_table[i] = new student(x);
  if(m_size > m_max / 4)
    grow();
```

4/14/2017

D.S. Hashtable

```
class student          ADT Map
{
  int m_id;      <---  key
  string m_name;
  char m_grade   <=>   value
};

class studentTable
{
  student **m_table;
  int m_max;
  int m_size;
  student m_dead;
  void insert(student &x)
```

```
int hash(int id)
{
  return id % m_max;
}
```

```
void delete(student & x)
{
  int i = hash(x.m_id);
  while(m_table[i] != NULL &&
        m_table[i]->m_id != x.m_id)
  {
    i = (i+1) % m_max;
  }
  if(m_table[i] == NULL)
    return;
  delete m_table[i];
  m_size--;
  m_table[i] = & m_dead;
```

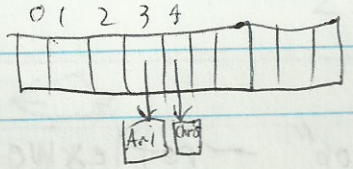
linear probing

walk right until either

- target is found

or

- NULL is found



this has problems

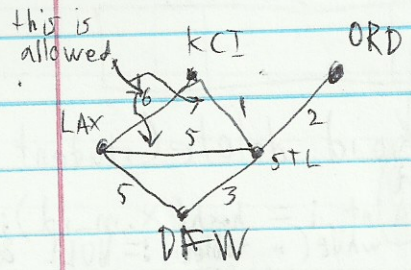
4/17/2017

Homework #7 posted
due next Wednesday
Test #3 next week
Miner LAN — ACM LAN party

Open hashing — tombstones
close hashing — Linked List

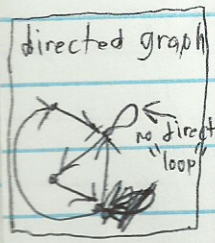
A.D.T. Graph

— a collection of objects together with a relationship among the objects



— a pair $G = \langle V, E \rangle$
V = set of objects (nodes, vertices)
E = set of pairs of objects $\langle v, w \rangle$ where $v \in V$ and $w \in V$

$V = \{ LAX, KCI, STL, ORD, DFW \}$
$E = \{ \langle LAX, STL \rangle, \langle STL, LAX \rangle, \langle STL, ORD \rangle, \langle ORD, STL \rangle, \langle KCI, STL \rangle, \langle STL\ KCI \rangle, \ldots \}$



if E is symmetric, the graph is called "undirected"
otherwise the graph is called "directed"

$\langle w, w \rangle$ "loop" — reflexive relation

if a graph does not contain a loop, it is

called "simple"

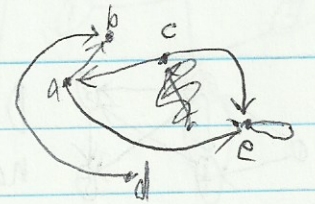
w is "adjacent" to v if $\langle v, w \rangle \in E$
w is a "neighbor" of v is $\langle v, w \rangle \in E$
note order

the "degree" of a node is the number of nodes adjacent to the node

degree of a = 2

a "path" is a sequence of nodes
$\langle n_0, n_1, n_2, n_3, \ldots n_k \rangle$
such that n_{i+1} is adjacent to n_i; $0 \leq i < k$

A path that does not "contains" a loop is called "simple"



example path: $\langle c, a, e, e, e \rangle$
not simple not cycle

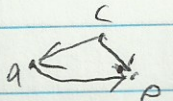
a cycle is a path where $n_0 = n_k$

$\langle c, a, e \rangle$ simple not cycle

$\langle e, e \rangle$ cycle

$\langle stL, kCI, stL \rangle$ cycle

DAG - Directed Acyclic Graph
- a directed graph with no cycles



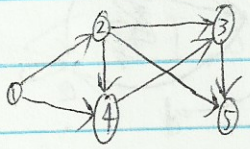
DAG

if $G = \langle V, E \rangle$ + weigh function
"weighted graph"

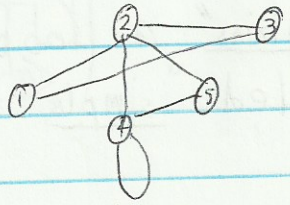
operations
- addVertex(G, v)
- addEdge(G, v, w)
- neighbors(G, v) → list of neighbors of v

4/19/2017 MyStringMap 256 ~~chars~~ char possibilities

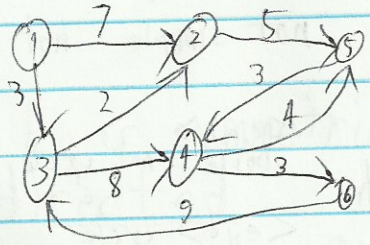
MAX CHAR VALUE
use sum of chars % max for hash



DAG
not simple



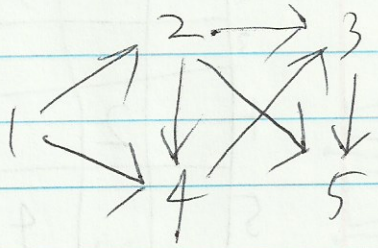
cyclic
not simple
not directed



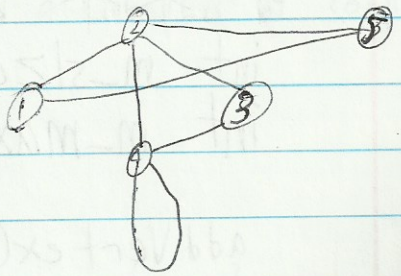
directed
weighted
cyclic

D.S. Adjacency Matrix

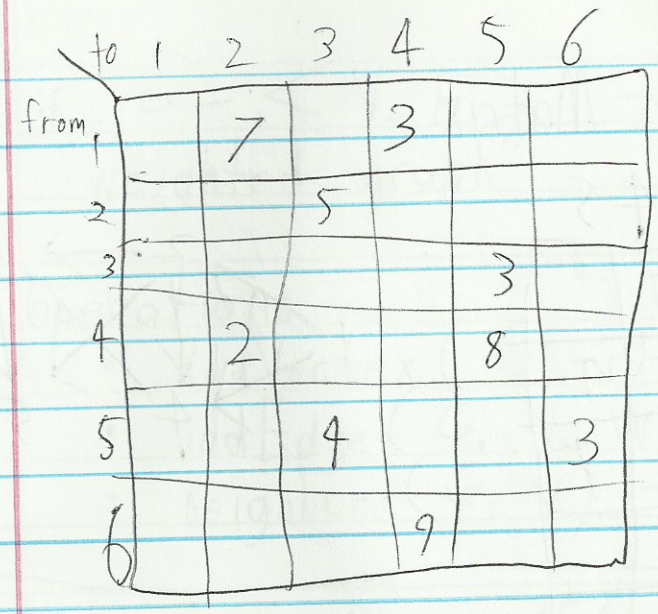
from \ to	1	2	3	4	5
1		T		T	
2			T	T	T
3					T
4			T		
5					

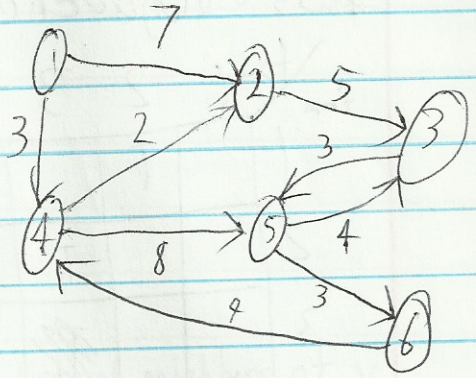


from \ to	1	2	3	4	5
1		T			T
2	T		T	T	T
3		T		T	
4		T	T	T	
5	T	T			



"symmetric matrix"






```
class AdjacencyMatrix
{
    T ** m_matrix;
    int m_size;
    int m_max;

    addVertex()                                   n^2
    {
        m_size++;
        if( m_size == m_max)
            grow();
    }
    addEdge(int v, int w)                         1
    {
        // suppose the edge does not already exist
        m_matrix[v][w] = true;
    }

    neighbors()                                   n
```

$$\int \frac{\ln(3x-5)}{\sin(-x+2)\sec(x+3)}\,dx$$

$$-\int \frac{\ln(3x-5)}{\sin(x-2)\sec(x+3)}\,dx$$

$$\cosh^2(x) - \sinh^2(x) = 1$$

$$\frac{d}{dx}(\operatorname{sech} x) = -\operatorname{sech} x \tanh x$$

$$\frac{d}{dx}\left(\frac{2}{e^x + e^{-x}}\right) = -\frac{2}{(e^x+e^{-x})}\cdot($$

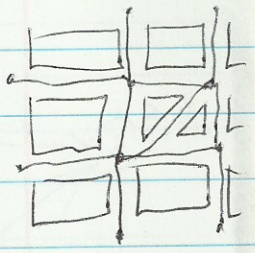
Adj Matrix

Cons:

Memory use

addVertex() inefficient

neighbors() very inefficient (consider street map)

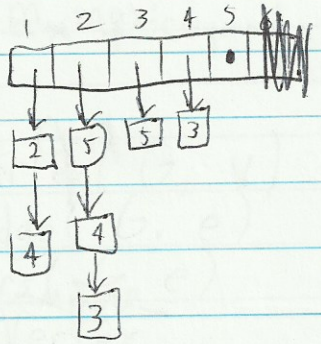


nodes: 1000
degree: ≤ 8

"sparse graphs" — many nodes, low node degree

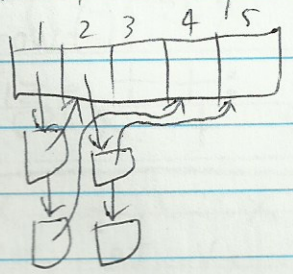
D.S. Adjacency List

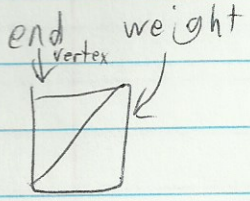
- Array of nodes, each with a LinkedList of connections

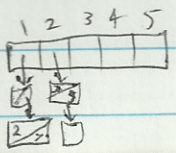


or

- Array of pointers to pointers to nodes







for weights, use structs

$$\int \csc x \, dx$$

$$\int \frac{\csc^2 x + \csc x \cot x}{\csc x + \cot x} \, dx$$

let $u = \csc x + \cot x$

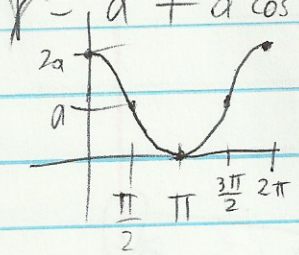
$du = (-\csc x \cot x - \csc^2 x) \, dx$

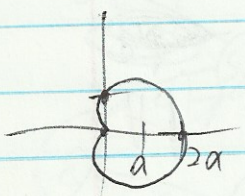
$dx = -\frac{du}{\csc x \cot x + \csc^2 x}$

$$-\int \frac{1}{u} \, du$$

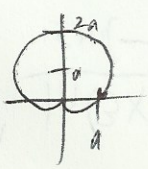
$$-\ln|\csc x + \cot x| + C$$

$r = a + a\cos\theta$ cardioid





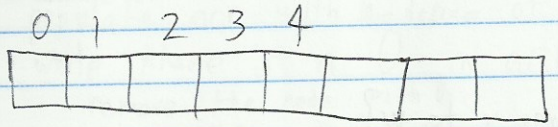
$r = a + a\sin\theta$

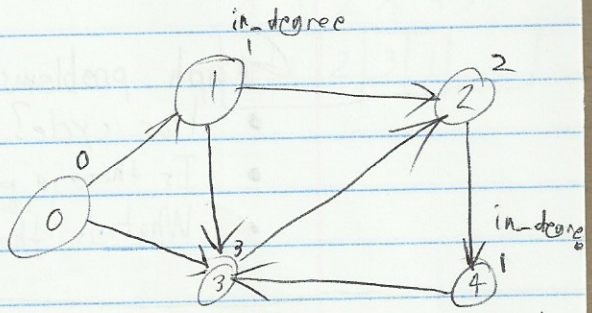


Homework #7 due Wednesday
Test #3 next Friday

A.D.T. Graph

D.S. AdjacencyList





(no path from 2 to 1)

```
class edge
{ int m_to;
};
class AdjacencyList
{
   LinkedList<edge> * m_nodes;
   int m_size;
   int m_max;
```

	complexity
• AddVertex(G, v)	n
• AddEdge(G, e)	1
• Neighbors(G, e)	1

• areNeighbors(G, v, w)
true if ⟨v, w⟩ is in G, false otherwise

AdjList complexity: n
AdjMatrix complexity: 1

```
   AddVertex
   {
      m_size++;
      if(m_size == m_max)
        grow();
      m_size++;
   }

   addEdge( int v, int w)
   {
      m_nodes[v].insert_back( w);
```

```
neighbors(int v)
{
    return m_nodes[v]; // that's a LinkedList
}
```

Graph problems.

- Has cycle?
- Is there a path from v to w?
- What is the shortest path from v to w?

~~Has~~ there a cycle?

Topological sort — sort nodes of graph

$< n_0, n_1, n_2, n_3, \ldots n_k >$

such that

if there is a path from n_i to n_j then $j > i$

if you fail, there is a cycle

in_degree of a node w

the number number of edges $\langle x, w \rangle$

(directed graphs only)

A node with in_degree 0 cannot be part of a cycle

to search for cycles, ~~etim~~ ~~find cycle()~~ remove nodes with in_degree 0

algorithm:

sort = <>

Look at graph for a node with in_degree == 0
  add such node to sort
  remove node from graph
repeat until graph is empty (no cycles) or no such node exists (cycle)

0	1	2	3	4	5
0	1	2	3	1	

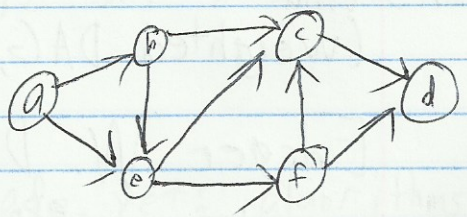
version 2
allocate sort
populate arr with in_degree of nodes
while there is a 0 in arr
  remove the node with 0
  append the removed node to sort
if arr empty, no cycle
else, cycle

Euler circuit — a ~~path~~ circuit using each edge exactly once ~~and returning to start~~

4/24/2017

Is there a path from v to w?

```
// Pre: the graph is a DAG
path( node v, node w)
  if( v == w)
    return true
  for each neighbor u of v
    success = path( u, w)
    if( success)
      return true
  return false
// this is called "depth first search"
```

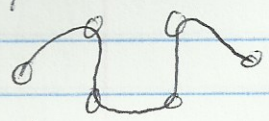


DAG

~~to~~ to get the path, create a predecessor table

	a	b	c	d	e	f
pred	a	a	~~f~~	c	b	e

to find the path from pred, start at end and construct the path backward



```
path(node v, node w)
  stack S
  push v into S
  while S is not empty
    x = top of S
    pop S
    if (x == w)
      return true
    for every neighbor y of x
      pred[y] = x
      push y into S
  return false
```

"Breadth first search"

```
path(node v, node w)
  queue Q
  enqueue v into Q
  while Q is not empty
    x = front of Q
    dequeue Q
    if (v == w)
      return true
    for every neighbor y of x
      pred[y] = x
      push y into S
  return false
```

4/26/2017

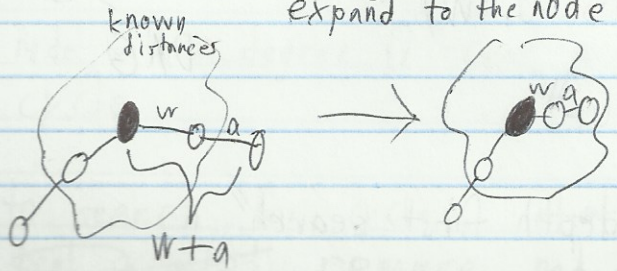
breadth first
depth first

which neighbor to explore next

problem 3 What is the shortest path between u and w? (weighted DAG)

Edsger W. Dijksta

expand to the node with the smallest w to a (distance)



```
shortest_path( node v, node w)
    heap H // the heap contains "shortest distance candidates"
    insert <v, 0> into H   // v-start node  0-distance
    pred[v] = v
    dist[v] = 0
    while H is not empty
        <u, du> = getMin(H)
        if( du < dist[u])
            dist[u] = du
            for every neighbor y of u
                insert <y, du + weight(u, y)>
                pred[y] = u
    return fail
```

remove_min(H)
if u == w
return du

this can be applied to ~~AI~~ graphs

Test #3

~~D.S.~~

ADT Binary Search Tree
DS TreeNode
DS ArrayTree

ADT Heap
DS ArrayHeap — no code, yet; algorithms

ADT Map & Set
DS HashTable

ADT Graph
Terminology Traversals
DS Adj Matrix
DS Adj List

Final Exam

either
- Final Exam (comprehensive)
- or
- Final Project (not a homework, cannot be dropped)

how to choose

diverse test scores — choose exam
consistent test scores — choose project

STL

Standard Template Library
— classes and functions for D.S.

#include <vector> (ArrayList)

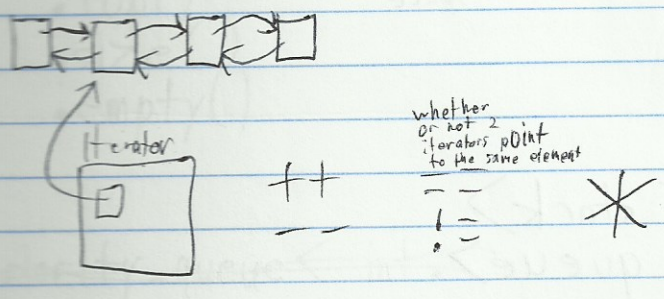
```
vector<int> v1;
v1.size();
v1.push_back();
v1.push_front(); //expensive
v1.clear();
v1.front();
vector<int> v2;
v1 = v2
vector<int> v3(v1);
v1 == v2
v1 != v2
v1[k]  // seg fault
```

#include <list> (DoublyLinkedList)

```
list<int> l1;
l1.size();
l1.push_front(); //cheap
l1.push_back();
```

ll.clear() ll.front()

ll[] ⊗ — no bracket operator for list



ll.begin() → the first element in an iterator
ll.end() → the sentinel iterator (NOT last element)

```
list<int>::iterator p;
p = ll.begin();
while( p != ll.end() )
{
   cout << *p << endl;
   p++;
}
```

vectors also have iterators (which is dumb)

```
for( vector<int>::iterator p = vl.begin(); p != vl.end(); p++)
   cout << *p << endl;
```

5/3/2017

put name in f project

STL

nesting

list<vector<int> > ll;
vector<vector<int> > ml;

(space) (space)

ml[a][b]

#include <stack>
<queue>

stack<int> s;
s.size()
(bool) s.empty()
s.push(3)
s.pop()
s.top()

queue<int> q;
q.size()
.empty()
.push(3)
.pop()
.front()
.back()

stacks and queues have iterators

#include <set>

set<int> s; ← type must have operator<

implemented as BinarySearchTree
(red and black tree)

s.size()
(bool) s.empty()
.insert(E)
.find(E)
.erase(E)

#include <queue>

priority_queue<int> q; // Max heap

(int> required)

- .push(3)
- .pop()
- .top()
- .size()
- .empty()

~~priority_queue< int,~~

class widget

{

int x

float y

};

class widgetComparator

{

operator()

~~compareWidget~~(widget a, widget b)

{

=

=

}

};

min_heap

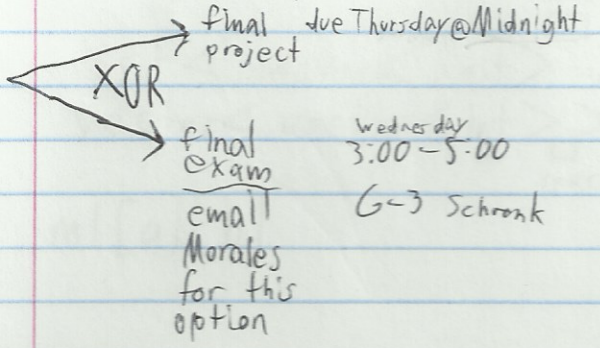
like greater<int>

or (default) less<int>

max_heap

priority_queue< widget, vector<widget>, widgetComparator>

for final, Dr. Morales has a super tester




```
#include <map>

map<string, char> grades;

grades.size()
bool grades.empty() const

access grades: []

grades["Alice"] = 'A';
grades["Bob"] = 'B';

cout << grades["Bob"] << endl;

grades.erase("Bob");
grades.find("Zebra");
```

in later classes, STL is allowed

in later classes, ~~Python is allowed~~, other languages are allowed

```
#include <algorithm> // very useful, worth a read
vector<string> v;
find( v.begin(), v.end(), "Bob"); ==> iterator
  if not found, v.end() is returned

count( v.begin(), v.end(), "cat");
equal( v1.begin(), v1.end(), v2.begin(), v2.end())

reverse( v.begin(), v.end()) // not compatible with list
random_shuffle( v.begin(), v.end())
sort( v.begin(), v.end())

make_heap( v.begin(), v.end(), cmp)
push_heap( /  ,  /  ,  /  )
pop_heap( /  ,  /  ,  /  )
```

STL is not in final exam

recruiters like:

- CONM "ninjas" — Price
- AI — Tavritz
- E/C "Evolutionary Computing" — ? (probably Tavritz)

next semester, Moraler will teach "Deep learning"