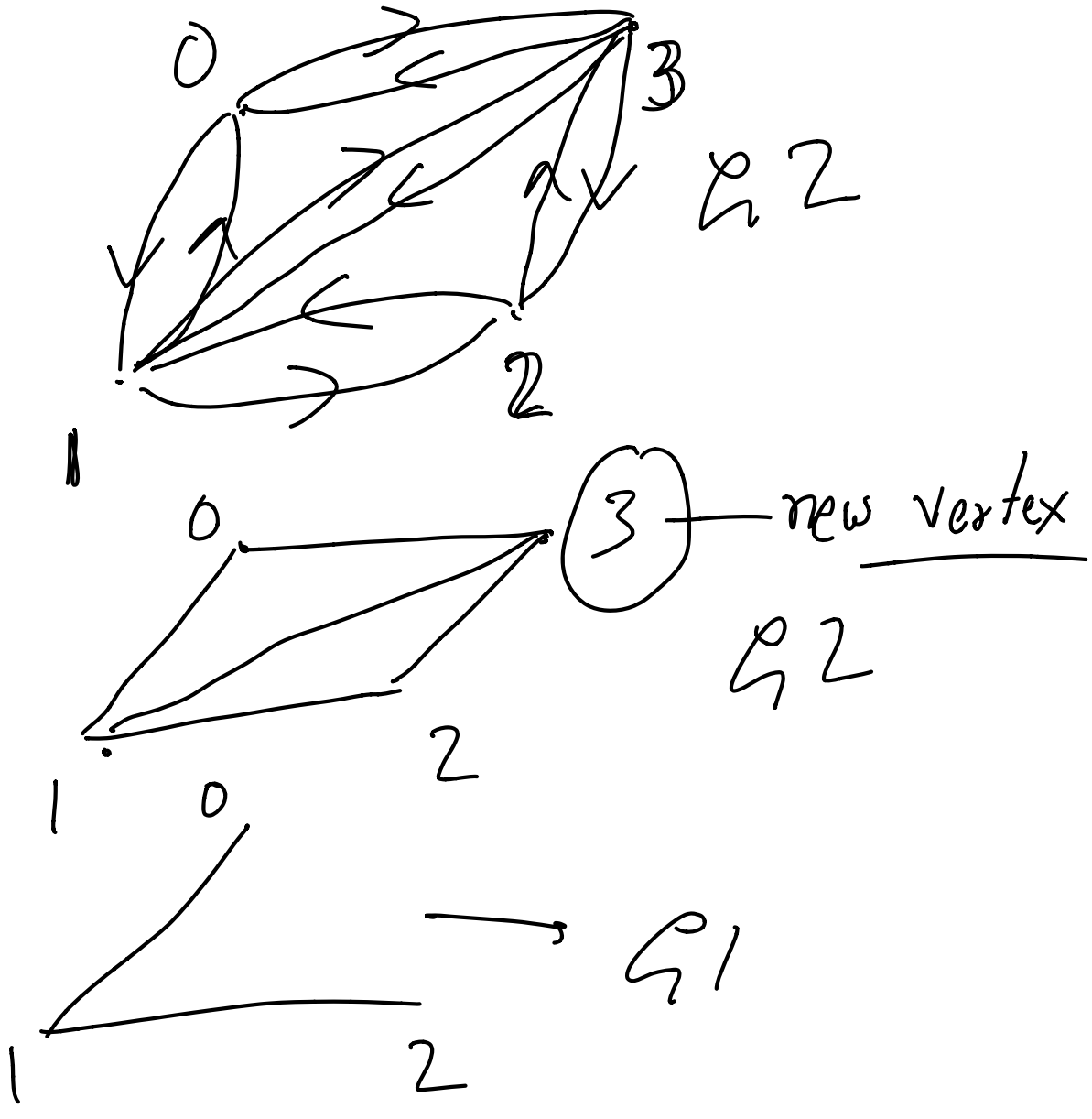
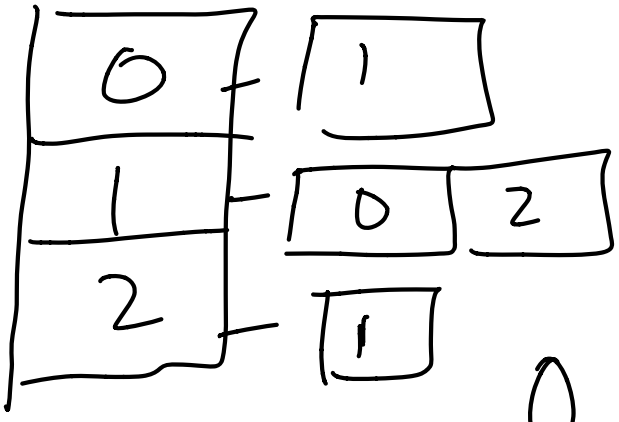
HW 11

Problem 2

Example



Input G1

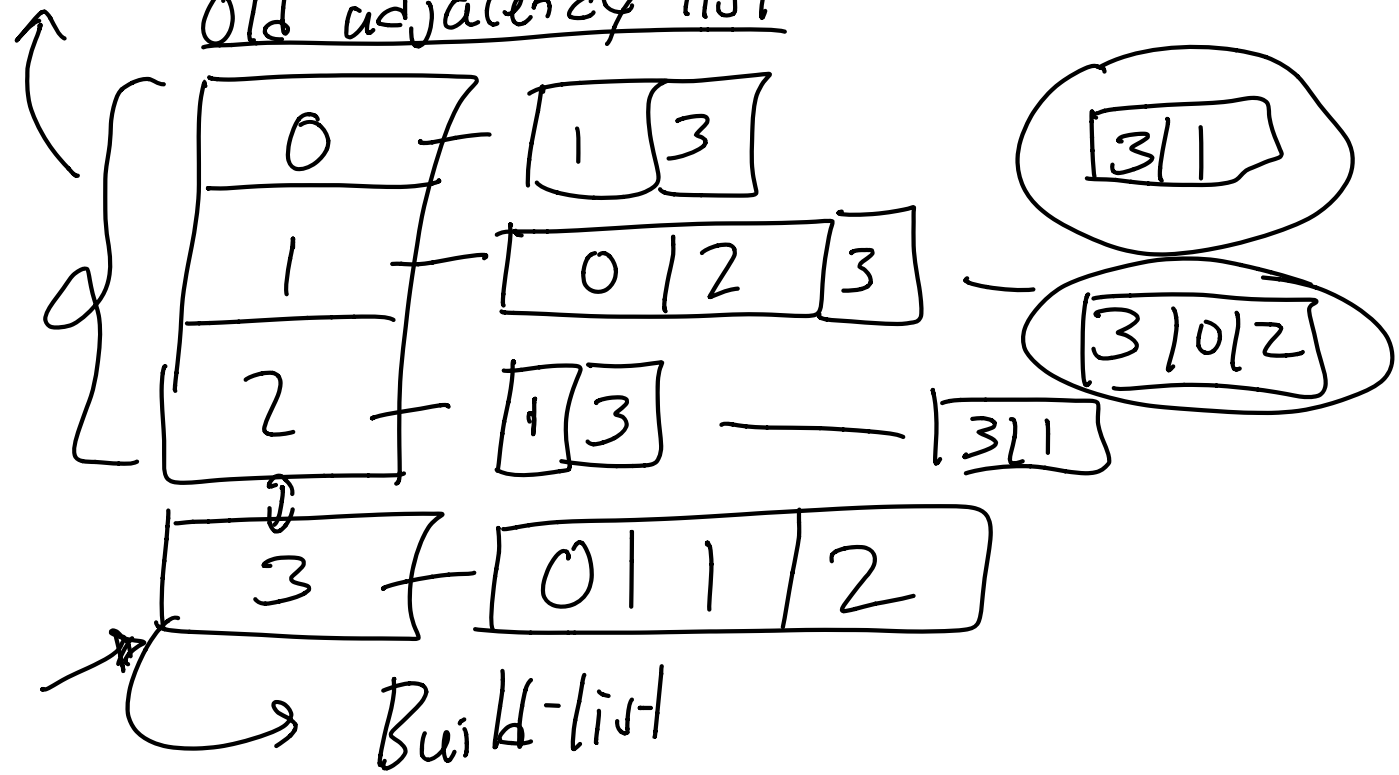


— G1

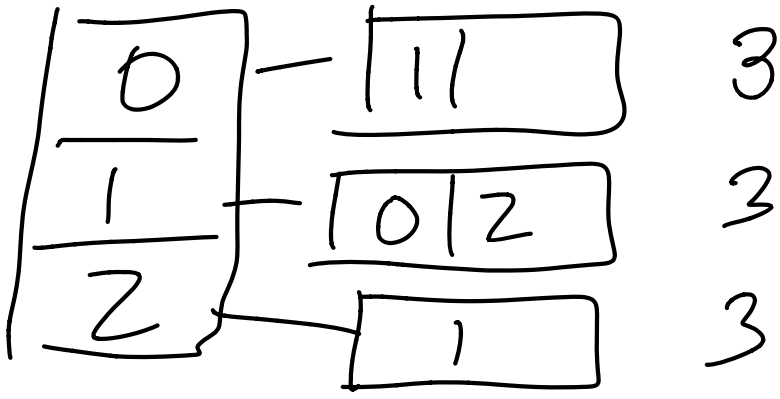
Vector map.

Output G2

Old adjacency list

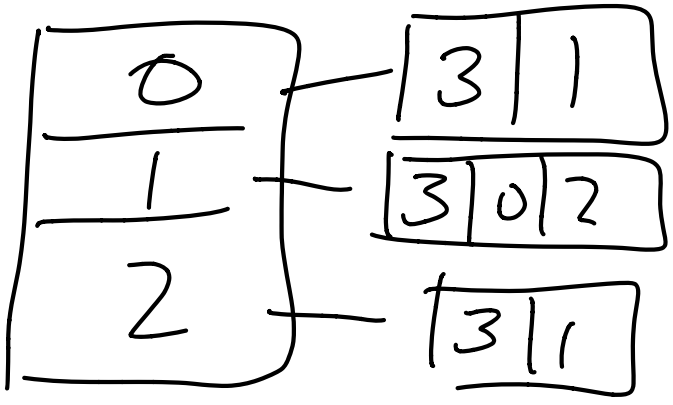


Take my old adjacency list



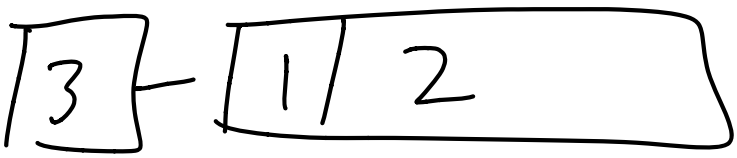
I have a vector of List and I want an element

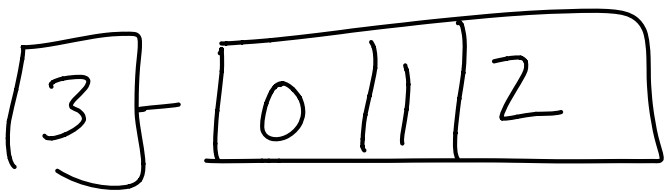
to every element (list) of this vector

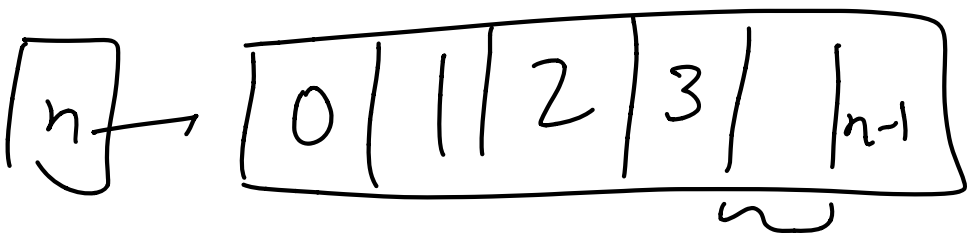


```
(vector-map (lambda ([nbrs : (Listof Vertex)])
              (cons n nbrs))
```

new vertex








```
(build-list n f)
```

[f(0), f(1), f(2) , f(n-1)].

[0, 1, 2, 3, (n-1)]

What should be f (identity function)

```
(build-list n (lambda ([i : Integer])
                i))
```

```
(vector (build-list n (lambda
                        ([i : Integer]) i)))
```

Just append the two parts

```
(: add-vertex : Graph -> Graph)

(define (add-vertex g)
  (match g
    [(Graph n adj)
     (Graph (+ n 1)
            (vec-append (vector-map
                         (lambda ([nbrs : (Listof Vertex)])
                           (cons n nbrs)) adj)
                        (vector (build-list n (lambda
                                                ([i : Integer]) i)))))]))
```

number of vertices

vector

Time complexity

Cons takes constant time

My lambda takes constant time

O(n)

Build list - O(n)

$O(n) + O(n) + O(n+1)$

↑ vector - append

$= O(n+1) = O(n)$

3.

b). Create a list of random events.

Convert this list to a BST.

lte?

```
;;
(: singleton : (All (A) (A A → Boolean)
                → (BST A))
(define (singleton val lte)
  (BST lte (Node val 'none
                 'none)))
```

Insert-all

Input1: BST

Input2: a list

Output: BST will all the elements in the list.

Idea use recursion

Insert the first element

Recursively the rest list

```
(: insert-all : (All (A) (List of A)
                 (BST A) -> (BST A)))
(define (insert-all my-list my-bst)
  (cond
```

remains unchanged

↓

```
[(empty? my-list) my-bst]

[else (insert-all (rest my-list)
        (BST-insert (first my-list)
          my-bst))]))
```

list → BST

Input 1: List
Input 2: Ite?

Output: BST containing elements of the list.

```
(: list->BST : (All (A) (Listof A)
                (AA -> Boolean) -> (BST A)))

(define (list->BST my-list lte)
  (insert-all (rest my-list) (singleton (first my-list) lte)))
```

Insert remaining elements

This creates a BST with one element

```
(define-struct Activity
  ([desc : String]
   [day-of-year : Integer]
   [loc- string]))

(define-type Calendar (BST
                       Activity))

(: activity-lte? : Activity Activity
   → Boolean)
(define (activity-lte? thing1 thing2)

(<= (Activity-day-of-year
     thing1) (Activity-day-of-year
              thing2)))
```

Function which converts a date to an activity

Day of year

```
(: day->act : Integer --> Activity)

(define (day->act day)

(Activity "" day "")
```

Then use build-list to create a list of activity.

Input: Integer.
Input: Function : Integer → Activity
Output: List of Activity

Suppose my function is f

[f(0), f(1), f(2) .... f(n-1)]

```
(: my-list-act : (Listof Activity))

(define my-list-act (build-list 5000
 (lambda ([n : Integer]) (day->act
 (+ (random 365) 1)))))

(define my-cal : Calendar)
(define my-cal (list-BST
          my-list-act activity-lte?))
```

Remark: This is not the ideal way to solve this problem.

The bst may not be balanced

Ideal way is to create a sorted vector/list and convert that vector to a balanced BST.

Exercise: Convert a sorted vector/list to a balanced BST

Note:

Recursion

Input: BST
Input: Start
Input: end.

Idea
Check if there is an element
with start date
Then delete it.
Then do recursion on start, end

If there is no element
with the start date
Then do recursion on start+1, end.

If start == end
then return the input.

```
(: go-on-vacation : Calendar Integer
                    Integer -> Calendar)

(define (go-on-vacation cal start end)
 (cond
   [(> start end) (error "Error")]
 [else
 (local
   [
    (: search-result : (U 'none Calendar))
(define search-result (BST-search
                       cal (day->act start)))]
 (match search-result
   ['none (if (start==end)
              cal
```

(go-on-vacation cal (+ 1 start)
end))]

new BST after deletion.

[(BST _ _)
(go-on-vacation (BST-delete
(day->uct start) (cal)
start date)]))]))