CSE 332
Summer 2018
Final Review

1 Parallel Code

Explain the steps you would use to perform the following tasks in parallel. Your algorithm should have the best possible $O()$ span, but you need not worry about constant factors in this problem. You may assume you have access to already allocated auxiliary arrays as needed, and may alter the input array. Use the following parallel code patterns discussed in class:

- `out = map(f, arr)` Applies `f` to every element of `arr`, storing the results in `out`.
- `out = reduce(baseFn, combineFn, arr)` Given `baseFn` on a single element and `combineFn` on two arrays (one of which being `arr`, the other the output of `baseFn`), `reduce` stores the results in `out`.
- `out = parallelPrefixSum(arr)` Runs ParallelPrefixSum on `arr`, storing the results in `out`.
- `out = pack(condition,arr)` Given `condition`, performs a pack/filter on `arr`, storing the results in `out`.

1. **Input**: An array of integers, `arr`
   Output:: An array where index i is the sum of the prime numbers in `arr` at indices $0, ..., i$.

   Solution: Run map(CompTo0, arr, out) where CompTo0 outputs 0 if the input is composite and the input itself if it is prime. Then run ParallelPrefixSum(out).

2. **Input**: `arr`, an array of integers
   Output: True if `arr` is a valid (0-indexed) representation of a binary min-heap.

   Solution: Run `map(f, arr, out)` where `f` is `arr[i]` $\geq$ `arr[(i-1)/2]`$||i == 0$
   Run `reduce(identity, AND, out)`.

3. **Input**: an array `arr` **Output**: two arrays, one with elements less than k the other with elements greater than k.

Solution: `Pack(` $< k$`, arr` for the first array, `Pack(`$\geq k$`, arr` for the second.

4. **Input:** `arr` an array of integers
   Output: an array containing **peak** elements of `arr`. An element i is a peak element if `arr[i-1]` $<$ `arr[i]` and `arr[i+1]` $<$ `arr[i]`.

   Solution: This is just a `Pack(f, arr` with `f` being `arr[i-1]` $<$ `arr[i]` and `arr[i+1]` $<$ `arr[i]`.

2 Graph problems

1. You and your trusty Dragonite have just finished your training outside the Pokémon League. You now feel prepared to take down the Elite Four. There's just one problem – you've earned none of your badges.

   Your goal is to visit each of the n cities with gyms, crush all the gym leaders, and return to the Pokémon League as quickly as possible.

   Describe a graph representation of this problem (For example, what are the vertices and edges? Is the graph weighted?).

   Can you design an algorithm to quickly determine the best possible route? If so describe it (you may use any algorithm discussed in class as a black box). If not, informally justify why such an algorithm isn't likely.

   Solution: One possible graph representation is this one: have a vertex for each of the cities with a gym (and one for the Pokémon League). Edges will represent routes between the cities. Edges should be weighted with the time it would take to travel across that route. You could probably leave the graph undirected (as long as all routes take the same time to walk in each direction).
   An efficient algorithm isn't likely. The question is asking for the optimal tour, i.e. this is the optimization version of the traveling salesperson problem.
   A formal argument (which you'll see if you take CSE 421) would show that any instance of TSP could be turned into an instance of this problem.

2. Having just finished summer quarter, you and your friends celebrate with a trip to Disney Land. You decide to have a contest. Your goal is: starting from the entrance

   (a) Ride at least two distinct rides

(b) Make it to Splash Mountain

Whoever arrives first is the winner. You have an encyclopedic knowledge of Disney Land, thus you know how long it takes to walk from any ride to any other, and moreover you know how much time it takes to go on any ride.

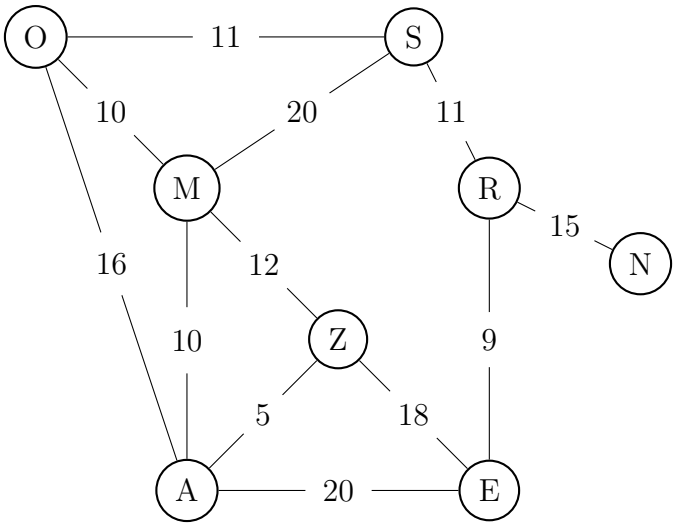
Describe a way to represent this problem as a graph. Then either describe an algorithm to run to solve the problem, or informally justify that such an algorithm isn't likely.

Solution: Make a graph representing a map of Disney Land, with a vertex for each ride, and the walking time as the weight of the edge. If you can walk the same speed in either direction, you can leave this portion of the graph unweighted. Now make $n+2$ copies of this graph. The first of these copies represents before you have ridden any rides. The next n represent when you have taken exactly one ride (and record which one you rode), and the final one represents when you have ridden two (distinct) rides.
Add (directed!) edges as follows: from each ride in the first copy, add an edge to that ride in one of the n single-ride copies. Weight these edges with the time it takes to go on that ride. Now from each of the n copies, add an edge for every ride **except** the one you've already ridden in that graph, to the corresponding vertex in the final copy. Weight all edges with the ride time.
To solve the problem you can run Dijkstra's from the entrance in copy 1, with a target of Splash Mountain in the final copy.
To find the rides to take, just look for the edge that leaves copy 1 and the edge that enters the final copy. Similarly, by following predecessors we can also find the route to follow.

3 Running Graph Algorithms

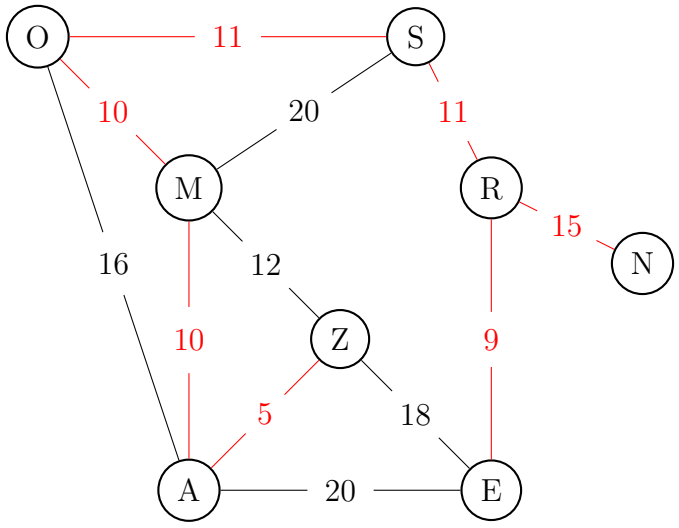
3.1 MST

Consider the following graph:



Find an MST of this graph using both of the two algorithms we've discussed in lecture. Make sure you say which algorithm you're using and show your work.

Solution:



- Using Prim's algorithm:

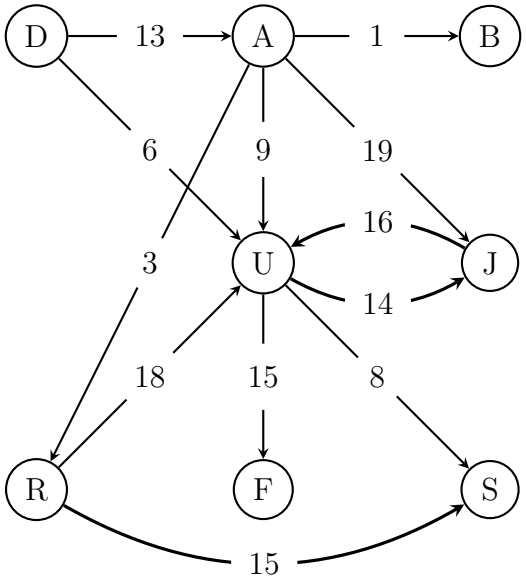
Vertex	Distance	Best Edge	Processed
A	–	–	Yes
E	~~20~~ ~~18~~ 9	~~(A, E)~~ ~~(E, Z)~~ (E, R)	Yes
M	10	(A, M)	Yes
N	15	(N, R)	Yes
O	~~16~~ 10	~~(A, O)~~ (M, O)	Yes
R	11	(R, S)	Yes
S	~~20~~ 11	~~(M, S)~~ (O, S)	Yes
Z	5	(A, Z)	Yes

- Using Kruskal's algorithm:

Edge	Include?	Reason
(A, Z)	Yes	–
(E, R)	Yes	–
(A, M)	Yes	–
(M, O)	Yes	–
(O, S)	Yes	–
(R, S)	Yes	–
(M, Z)	No	Cycle (M, A, Z, M)
(N, R)	Yes	–
(A, O)	No	Cycle (A, M, O, A)
(E, Z)	No	Cycle (E, R, S, O, M, A, Z, E)
(A, E)	No	Cycle (A, M, O, S, R, E, A)
(M, S)	No	Cycle (M, O, S, M)

3.2 Dijkstra

Consider the following graph:



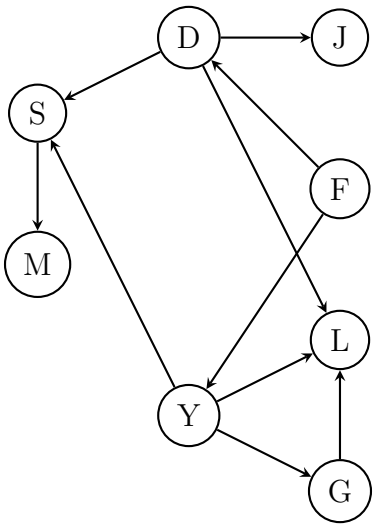
Use Dijkstra's Algorithm to find the lengths of the shortest paths from D to each of the other vertices. For full credit, you must show your work at every step. Break ties alphabetically.

Solution:

Vertex	Distance	Predecessor	Processed
A	13	D	Yes
B	14	A	Yes
D	0	–	Yes
F	21	U	Yes
J	20	U	Yes
R	16	A	Yes
S	14	U	Yes
U	6	D	Yes

3.3 Topo Sort

Consider the following graph:



Find a topological sort of this graph.

Solution: F, Y, D, G, L, S, J, M is one, there are many more valid solutions.

4 Hash table

1. Suppose we have a hash table that uses separate chaining and has an internal capacity of 10 (do NOT worry about resizing for this problem). Assume that each bucket is a linked list where new elements are added to the front of the list.

   Insert the following elements in the EXACT order given using the hash function $h(x) = x$:

   98, 18, 68, 21, 38, 8, 9, 11

   Solution: The state of the internal array will be

/	$11 \to 21$	/	/	/	/	/	/	$8 \to 38 \to 68 \to 18 \to 98$	9

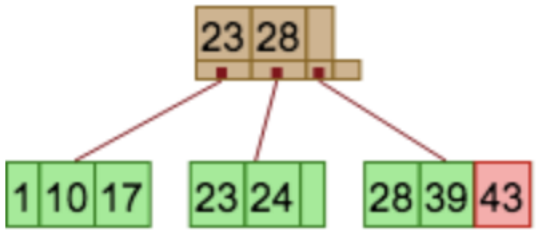
2. Repeat the same insertions with quadratic probing.

Solution: The state of the internal array will be

9	21	68	/	8	11	/	38	98	18

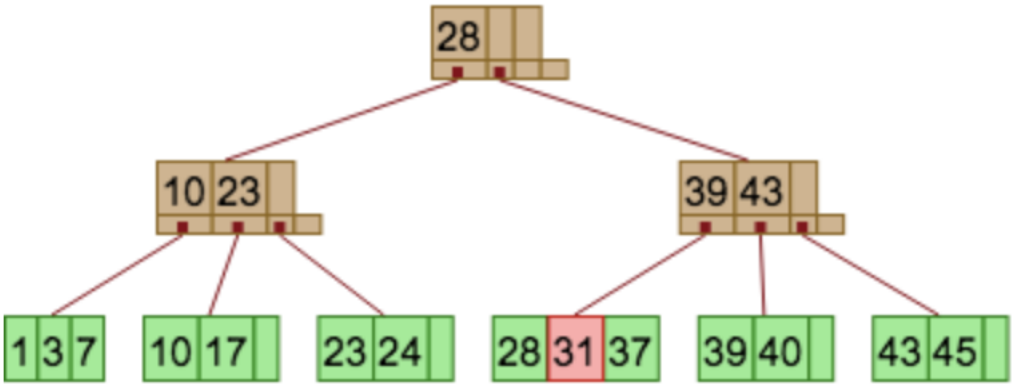
5 B-tree insert and math

Given the following existing B-tree (values not shown):



1. Insert the following keys into the B-tree, showing work:
   $3, 37, 45, 7, 40, 31$

Solution:



2. Delete the following keys from *the initial* B-tree:
   $10, 28, 24, 1, 39, 43$

Solution: 

3. Given the following parameters for a B-Tree with $M = 16$ and $L = 13$:

   - Key Size = 4 bytes
   - Pointer Size = 2 bytes
   - Data Size = 12 bytes per record (includes the key)

   Assuming that M and L were chosen appropriately, what is the likely page size on the machine where this implementation will be deployed? Give a numeric answer based on two equations using the parameter values above.

   p is the page size in bytes, k is key size in bytes, t is pointer size in bytes, and v is value size in bytes.

   $$\begin{aligned} p &\geq Mt + (M-1)k \\ &\geq (16)(2) + (16-1)(4) \\ &\geq 92 \\ p &\geq L(k+v) \\ &\geq 13(12) \\ &\geq 156 \end{aligned}$$

   Page size must be at least 156 bytes.

6 Concurrency

Your friend has written the following (somewhat strange) code for a stack.

```
1
2  public class WeirdConcurrentStack{
3      Object[] arr;
4      int capacity; int size;
5
6      public void push(Object o){
7              if(size == capacity)
8                  resize();
9              arr[size++] = o;
10     }
11
12     public Object pop(){
13         if(size == 0)
14             throw new NoSuchElementException();
15         Object retVal = arr[size-1];
16         size--;
17
18     }
19
20     public Object peek(){
21         Object retVal = this.pop();
22         this.push(retVal);
23         return retVal;
24     }
25
26     private void resize(){ /* details omitted */ }
27 }
```

Give a bad interleaving of this code.

Solution: One possible interleaving (there are many) Two threads call push(). Thread A executes the `arr[size] = o` portion of the code (before incrementing size). Thread B then executes the same line, overwriting `arr[size]`, and incrementing size. Thread A then increments size.
This is a bad interleaving because one of the two inserts is lost.

Your friend decides to fix the code by adding some re-entrant locks.

```
1
2  public class WeirdConcurrentStack{
3      Object[] arr;
4      int capacity; int size;
5      Lock lk;
6
7      public void push(Object o){
8              lk.acquire();
9              if(size == capacity)
10                 resize();
11             arr[size++] = o;
12             lk.release();
13     }
14
15     public Object pop(){
16         lk.acquire();
17         if(size == 0)
18             throw new NoSuchElementException();
19         Object retVal = arr[size-1];
20         size--;
21         lk.relsease();
22         return retVal;
23
24     }
25
26     public Object peek(){
27         Object retVal = this.pop();
28         this.push(retVal);
29         return retVal;
30     }
31
32     private void resize(){ /* details omitted */ }
33 }
```

Does the code above have a bad interleaving? If so give a bad interleaving. If not informally justify why there won't be any.

Solution: There is still a bad interleaving. Consider the following one:
Thread A calls peek, and successfully pops, and releases the lock. Thread B calls push, and acquires the lock before thread A can acquire it in the pop call. B pushes a new element on the top of the stack, and releases the lock. Thread A then replaces the element it popped on top of the stack.
But now the stack is in the wrong state. The most recently added element is not the one we just added, but the thing thread A popped off the top!

Does the code above have potential for deadlock? If so describe an interleaving to cause deadlock. If not informally justify why deadlock will not occur.

Solution: No. Since there is only one lock, there is no possiblity of deadlock.

Tell your friend a way to improve their code. If you found errors in the previous parts, your alterations should fix them. If you did not find errors, you should still find a way to improve the use of synchronization.

Solution: To fix the bad interleaving, peek should also acquire the lock at the top of the method and release it at the end. Note that the lock must be re-entrant for this fix to work.
There is another bug to fix – if an exception is thrown in pop, then we will never release the lock. Putting `lk.release()` in a finally block is one way to fix this error.

7 Sort

1. Robbie's first phone from middle school didn't have a lot of memory on it. How should his phone contacts be sorted to put them in last-name, first-name order?

   Solution: Since we don't have much memory, we should use an in-place sort. Without any further information, quicksort is a good default choice for in-place sorts (since it's usually the fastest in practice).

2. MatLab (a mathematical programming language) implements some of its functions very quickly, on the assumption that the data is mostly sorted already. What kind of sort are they probably using?

   Solution: Insertion sort. It has an $O(n)$ runtime on arrays that are nearly sorted.

3. Caitlin wants to run Radix sort on her extensive collection of books, sorting them alphabetically with their whole book contents as the input She wants to know not only if she has multiple copies of something, but also to see if differing versions have different cotents. Is this a good idea? Explain why either way. If it's not a good idea, recommend an input or sort that might work better.

Solution: This might not be a great idea. Radix sort depends on the length of the inputs, and if we're defining contents as "every word in the book" that's going to take a long time.
A better solution would be to sort by titles first (using any of the $O(n \log n)$ sorts or even radix sort), and then separately check whether different versions are identical with a more detailed check (like a linear scan). This would avoid having to look at every single word in every single book in the worst case.

8 Amdahl

1. Your company has a program which is 1/8 sequential and 7/8 parallelized. At a minimum, how many processors do you need for a 4x speedup? For full credit, your answer must be a simplified fraction or an integer.

Solution: We need $\frac{1}{1/8+\frac{7/8}{P}} \geq 4$ where P is the number of processors. Multiplying through by the denominator we get: $1 \geq \frac{1}{2} + \frac{7/2}{P}$ Simplifying further: $\frac{1}{2} \geq \frac{7}{2P}$ or $P \geq 7$.
So we will require at least 7 processors.

2. How much of the program would have to be parallelized if you only have four processors?

Solution: Using Amdahl's law we note $4 \leq \frac{1}{S+(1-S)/P}$ But then we can simplify that to:

$$4S + 4\frac{1-S}{P} \leq 1$$

Plugging in $P = 4$ we have

$$4S + (1-S) = 1 + 3S \leq 1$$

Thus S must be 0.

9 P vs. NP

For each of the following problems, circle all classes which the problem is **known** to belong.

Determine if there is any pair of vertices in a graph that are at distance at least k from each other.	P	NP	NP-complete	NP-hard
Find the shortest tour of a weighted graph	P	NP	NP-complete	NP-hard
Determine if a graph has a topological sort	P	NP	NP-complete	NP-hard

Solution: The first problem is in P and NP. It is in P because it is a decision problem, and can be solved by running Dijkstra from every vertex. Every problem in P is in NP. Since it is known to be in P, we can be confident it's neither NP-complete or NP-hard (or else we just solved P vs. NP in our exam).
The second problem is only NP-hard. It is not a decision problem, thus it cannot be any of the first three, but it is a version of TSP, so it's NP-hard.
The third problem is in P and NP. It is a decision problem with a polynomial time algorithm (you can find the SCCs and ensure they are all single vertices, or modify the topological sort algorithm seen in class to check that there is a valid vertex to add to the sort). As with the first problem, we can rule out NP-completeness and NP-hardness.

For each of the following statements, say whether it is true or false, and justify your answer in 1-2 sentences.

If we found an efficient algorithm for 2-SAT, we would have a polynomial time algorithm for every problem in NP.

Solution: False. 2-SAT is not NP-hard. We saw a polynomial time algorithm in class.

If we find an efficient algorithm for TSP, we would have a polynomial time algorithm for every NP-hard problem.

Solution: False. The claim is true if you replace "NP-hard" with "NP" or "NP-complete" but NP-hard problems include things like $n \times n$ chess (which we know has no polynomial time algorithm) and the Halting Problem (which we know has no algorithms of any kind).