Worksheet #5: Computation Graphs for Async-Finish and Future Constructs

1) Can you write pseudocode with `async-finish` constructs that generates a Computation Graph with the same ordering constraints as the graph on the right? If so, provide a sketch of the program.

No. Finish cannot be used to ensure that D waits for both B and C, while E waits only for C.

2) Can you write pseudocode with `future async-get` constructs that generates a Computation Graph with the same ordering constraints as the graph on the right? If so, provide a sketch of the program.

Yes, see program sketch with void futures. A dummy return value can also be used.
1. HjFuture<String> A = future(() -> {
    return "A"; });
2. HjFuture<String> B = future(() -> {
    A.get(); return "B"; });
3. HjFuture<String> C = future(() -> {
    A.get(); return "C"; });
4. HjFuture<String> D = future(() -> {
    // Order of B.get() & C.get() doesn’t matter
    B.get(); C.get(); return "D"; });
5. HjFuture<String> E = future(() -> {
    C.get(); return "E"; });
6. HjFuture<String> F = future(() -> {
    D.get(); E.get(); return "F"; });
7. F.get();
Example: Binomial Coefficient

- The coefficient of the $x^k$ term in the polynomial expansion of the binomial power $(1 + x)^n$
- Number of sets of $k$ items that can be chosen from $n$ items
- Indexed by $n$ and $k$
  - written as $C(n, k)$
  - read as “$n$ choose $k$”
- Factorial Formula: $C(n, k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$
- Recursive Formula
  $$C(n, k) = C(n - 1, k - 1) + C(n - 1, k)$$
  Base cases: $C(n, n) = C(n, 0) = C(0, k) = 1$


---

Example: Binomial Coefficient
(Recursive Sequential version)

```c
1. int choose(int N, int K) {
2.     if (N == 0 || K == 0 || N == K) {
3.         return 1;
4.     }
5.     int left = choose (N-1, K - 1);
6.     int right = choose (N-1, K);
7.     return left + right;
8. }
```
Example: Binomial Coefficient
(Parallel Recursive Pseudocode)

1. Integer choose(int N, int K) {
2.     if (N == 0 || K == 0 || N == K) {
3.         return 1;
4.     }
5.     future<Integer> left  =
6.     future { return choose (N-1, K-1); }
7.     future<Integer> right =
8.     future { return choose (N-1, K); }
9.     return left.get() + right.get();
10. }

• Use of futures supports incremental parallelization with low developer effort

What inefficiencies do you see in the recursive Binomial Coefficient algorithm?

![Binomial Coefficient Tree]

C(4, 2) = 6
C(3, 1) = 3
C(3, 2) = 3
C(2, 0) = 1
C(2, 1) = 2
C(2, 2) = 1
C(1, 0) = 1
C(1, 1) = 1
Memoization

- Memoization - saving and reusing previously computed values of a function rather than recomputing them
  - A optimization technique with space-time tradeoff
  - A function can only be memoized if it is referentially transparent, i.e. functional
- Related to caching
  - memoized function "remembers" the results corresponding to some set of specific inputs
  - memoized function populates its cache of results transparently on the fly, as needed, rather than in advance

Helpful Link: [http://en.wikipedia.org/wiki/Memoization](http://en.wikipedia.org/wiki/Memoization)

Pascal’s Triangle is an example of Memoization

\[ C(n, k) = C(n - 1, k - 1) + C(n - 1, k) \]
Example: Binomial Coefficient
(sequential memoized version)

1. final Map<Pair<Int, Int>, Int> cache = new ...;

2. int choose(int N, int K) {
3. Pair<Int, Int> key = Pair.factory(N, K);
4. if (cache.contains(key)) {
5. return cache.get(key);
6. }
7. if (N == 0 || K == 0 || N == K) {
8. return 1;
9. }
10. int left = choose (N - 1, K - 1);
11. int right = choose (N - 1, K);
12. int result = left + right;
13. cache.put(key, result);
14. return result;
15. }

Example: Binomial Coefficient
(parallel memoized version w/ futures)

1. final Map<Pair<Int, Int>, future<Integer>> cache = new ...;
2. Integer choose(final int N, final int K) {
3. final Pair<Int, Int> key = Pair.factory(N, K);
4. if (cache.contains(key)) {
5. return cache.get(key).get();
6. }
7. future<Integer> f = future {
8. if (N == 0 || K == 0 || N == K) return 1;
9. future<Integer> left = future { return choose (N-1, K-1); }
10. future<Integer> right = future { return choose (N-1, K); }
11. return left.get() + right.get();
12. }
13. cache.put(key, f);
14. return f.get();
15. }

• Assumes availability of a “thread-safe” cache library, e.g., ConcurrentHashMap
Announcements & Reminders

• **IMPORTANT:**
  — Watch video & read handout for topic 2.3 for next lecture on Wednesday, Jan 25th

• HW1 was posted on the course web site ([http://comp322.rice.edu](http://comp322.rice.edu)) on Jan 11th, and is due on Jan 25th

• Quiz for Unit 1 (topics 1.1 - 1.5) is due by Jan 27th on Canvas

• See course web site for all work assignments and due dates

• Use Piazza (public or private posts, as appropriate) for all communications re. COMP 322

• See **Office Hours** link on course web site for latest office hours schedule. Group office hours are now scheduled during 3pm - 4pm on MWF in DH 3092 (default room but alternate room may need to be used on some days — an announcement will be made in the lecture on those days)