Problem Set 7 Solutions

Calvin Walker

Problem 1:

(a) This problem is in NP. Consider the following non-deterministic algorithm A:

1. Choose a random path p from $s \to t$
2. Output YES if p uses at most 1 vertex from each Z_i for all $i \in [k]$, otherwise output NO.

If there is no valid st path in G, A always outputs NO regardless of the guess p. If there is a valid st path, then A has a nonzero chance of making this guess and outputting YES. So A is a non-deterministic polynomial time algorithm and this problem is in NP.

(b) This problem is not in NP. To verify a YES answer to this problem, we must check all of the possible subsets of G when removing $\leq k$ edges. There are $\sum_{j=1}^{k} \binom{n}{n-j}$ such subsets, each of which take linear time to verify. So there is no polynomial time verifier for this problem, and it is not in NP.

(c) This problem is in NP. Consider the following non-deterministic algorithm A:

1. Randomly partition the verticies into subsets L and R
2. Count the edges between L and R, and output YES if this number is k, otherwise output NO.

If there is no such cut of G, then A always outputs NO. If there is a valid cut C, then there is a nonzero chace A's random cut equals C and outputs YES. The edges crossing (L, R) can be counted in $O(n^2)$ time so A is a non-deterministic polynomial time algorithm and this problem is in NP.

(d) This problem is not in NP. To verify a YES answer, we must check all the sugraphs with $\geq k$ verticies to ensure there is a clique of size k, and check if there is a clique of size greater than k. There are $\sum_{j=k}^{n} \binom{n}{j}$ such subgraphs which each take polynomial time to verify. So there is no polynomial time verifier for this problem, and it is not in NP.

Problem 2:

Algorithm: Initialize a graph $G = (V, E)$ with a vertex for each space $i \in [n]$. For each each vertex create a directed edge to each of the spaces reachable from it in the path. Add a vertex v_0 with directed edges to $v_1, v_2 \ldots v_5$. Weight the edges in G equal to the negative value of the space represented by the vertex they are directed towards. Then, use Bellman-Ford to find the shortest path using k edges from v_0 to v_n, given by the subproblem $d_k(v_n)$. Return the absolute value of $d_k(v_n)$, which is the maximum total number of points.

Runtime: The graph can be initialized in linear time. Each vertex except the starting space has *indegree* 5, and there are n verticies in V, so Bellman-Ford will terminate in $O(5n \cdot n) = O(n^2)$ time, giving the algorithm a runtime of $O(n^2)$

Correctness: There are no negative cycles in G, since edges are only directed towards $\{v_{i+1}, v_{i+2}, \ldots v_{i+5}\}$ for vertex v_i, so we can assume the correctness of Bellman-Ford in finding the shortest, or most negative path from v_0 to v_n. The most negative path will be the maximum point sequence of jumps since a higher value for a space corresponds to a more negative edge.

Problem 3:

Algorithm: Given an $n \times n$ grid of verticies $G = (V, E)$ with undirected edges from each vertex to its neighbors in the grid, and m distinct points in V, do the following:

1. Split each vertex $v \in V$ into two verticies, v_{in} and v_{out}, with an edge of capacity 1 from v_{in} to v_{out}.
2. Convert each undirected edge $e \in E$ into two directed edges each with capacity 1 such that $e = (u, v)$ becomes (u_{out}, v_{in}) and (v_{out}, u_{in}).
3. Add a vertex s to G with directed edges of capacity 1 to each of the m starting points, and a vertex t with a directed edge of capacity 1 for each of the verticies on the boundary into t

Then, run Ford-Fulkerson to obtain an s to t integer valued max-flow F. If the value of F is equal to m, return YES, otherwise return NO.

Correctness: By taking the paths given by $f(e) = 1$, we can recover the escape paths for each person given F. Since each of these escape paths corresponds to a flow path with integer value 1, the total value of F must be m for there to be a vertex-disjoint escape path for each of the m people. The value of F can never be greater than m, since we can always take the min-cut $C = (L, R)$ where only s is in L, giving a max-flow of m. No vertex can be used in two different flow paths since we assign each vertex v a capacity of 1 by splitting it into v_{in} and v_{out} with an edge of capacity 1 from v_{in} to v_{out}. So all of the escape paths recoverable from F will be vertex disjoint.

Runtime: We can make the required modifications to G in linear time. Ford-Fulkerson will terminante in $O(F|E|)$ time which in this case is $O(mn^2)$, giving the algorithm a runtime of $O(mn^2)$.

Problem 4:

The picky eaters problem is in NP since given a pizza, it is poly-time verifiable if k people will be willing to eat it by checking each person and their corresponding preferences. We will show that picky eaters is in NP-hard by giving a poly-time reduction from Independent Set, which is NP-Complete, to picky eaters. Given an instance of the independent set problem $G = (V, E)$, assign each vertex a different topping, and a picky eater who's favorite topping is this vertex. For the verticies adjacent to each vertex $v \in V$, assign these as the toppings $v's$ picky eater dislikes. Letting $n = |V|$, this process takes $O(n^2)$ time, as we check the neighbors of each vertex. So we have a polytime reduction from Independent Set to picky eaters. Thus, picky eaters is in NP-hard, and we have already shown that it is in NP, so picky eaters is NP-complete.