CS 577: Introduction to Algorithms

Final Exam Review Questions

Note: These problems are not representative of the difficultly of the Final Exam questions.

1 Graph/Greedy

1. **2-Colorable**

   Given a graph, determine whether the graph is 2-colorable. Specifically, can each node be assigned one of 2 colors so that no two neighboring nodes share a color?

   Solution:

   BFS, assigning "red" to the first layer, "blue" to the second layer, "red" to the third layer, etc. Then go over all the edges and check whether the two endpoints of this edge have different colors. This algorithm is $O(|V| + |E|)$ and the last step ensures its correctness.

2. **Interval Scheduling**

   Suppose you are given a set of n requests and each request i has a desired start and finish time pair $(s_i,\ f_i)$. We would like to determine a non-overlapping schedule for the requests with the largest number of requests scheduled. Show an optimal greedy algorithm to solve this problem and prove its optimality.

   Solution:

   Algorithm: Greedly select the the remaining request with the earliest finish time. After, remove all conflicting requests from the set.

   Optimality proof by "Stay Ahead" method: Let $A = i_1, ..., i_k$ be the requests selected by our greedy algorithm in the order they are added. Let $O = j_1, ..., j_m$ be the optimal solution ordered by finish time.

   Let $f(q)$ be the finish time of job q. Our first goal is to show that for all $r \leq k$, $f(i_r) \leq f(j_r)$. We will show this by induction.

   For the base case, let $r = 1$. Since the greedy algorithm selected the job with the earliest finish time first, then $f(i_1) \leq f(j_1)$.

For $t > 1$, assume that the statement is true for $t-1$ ($f(i_{t-1}) \leq f(j_{t-1})$) and prove it holds for t. Any jobs that are valid to add to the optimal solution are certainly valid to add to the greedy solution. Therefore, $f(i_r) \leq f(j_r)$.

From the inductive proof we showed that for all $r \leq k$, $f(i_r) \leq f(j_r)$. Now we must show that $m \leq k$. To show this is true, assume that $k < m$ and provide a contradiction. If $k < m$ then there is a job j_{k+1} in O that is not in A. This job must start after O's k-th job finishes and hence after A's k-th job finishes as well by our previous result. This job is then compatible with all the jobs in A and our greedy algorithm would have add it to A. That is our contradiction.

Together these steps prove that A is optimal.

2 Divide and Conquer

1. **Maximum Subarray Sum**

   Given a one dimensional array that may contain both positive and negative integers, find the sum of a contiguous subarray of numbers which has the largest sum.

   For example, if the given array is

   ```
   [-2, -5, 6, -2, -3, 1, 5, -6]
   ```

   then the maximum subarray sum is 7 using this subarray

   ```
   [6, -2, -3, 1, 5]
   ```

   Solution:

   Naive method: Using two loops try every subarray and compare to an overall maximum. This method has a $O(n^2)$ runtime.

   Divide and Conquer method: Run the following algorithm:

   (a) Divide the given array in two halves

   (b) Return the maximum of following three

   - Maximum subarray sum in left half (Make a recursive call)
   - Maximum subarray sum in right half (Make a recursive call)
   - Maximum subarray sum such that the subarray crosses the midpoint

   To find the maximum subarray that crosses the midpoint find the maximum sum starting from midpoint and ending at some point on left of mid, then find the maximum sum starting from mid + 1 and ending with some point on right of mid + 1. Adding

these sums gives the maximum sum that crosses the midpoint. This operation runs in $O(n)$ time.

Recurrence relation:

$$T(n) = 2T(n/2) + O(n)$$

Solving recurrence relation using unrolling or recursion tree should produce:

$$T(n) = O(nlogn)$$

3 Dynamic Programming

1. **Coin Change Problem**:

   You are given an integer array $D = [d_1...d_i]$ representing i coins of different denominations and an integer amount, A, representing a total amount of money.

   Return the fewest number of coins that you need to make up amount A. If that amount of money cannot be made up by any combination of the coins, return -1.

   You may assume that you have an infinite number of each kind of coin.

   Solution:

   Fill a 1D array $C[0...A]$ with $A+1$ values from index 0 to A using the following Bellman equation:

$$C[p] = \begin{cases} 0 & \text{if } p = 0 \\ min_{i:d_i \leq p}\{1 + C[p - d_i]\} & \text{if } p > 0 \end{cases}$$

   The $p-th$ element of C corresponds to the minimum number of coins needed to provide change for the amount p. Therefore the solution is held in $C[A]$.

   Runtime: Filling array C takes $O(A * i)$ time while retrieving the solution takes $O(1)$ time. Therefore, the overall time complexity is $O(A * i)$.

4 Network Flow

1. **The Escape Problem**

   You are given an $n \times n$ grid as an undirected graph with n^2 vertices organized in n rows and n columns. You are also given m distinct vertices called terminals. The Escape Problem asks you to determine whether there are m vertex-disjoint paths in the grid that connect the terminals to m distinct boundary vertices.

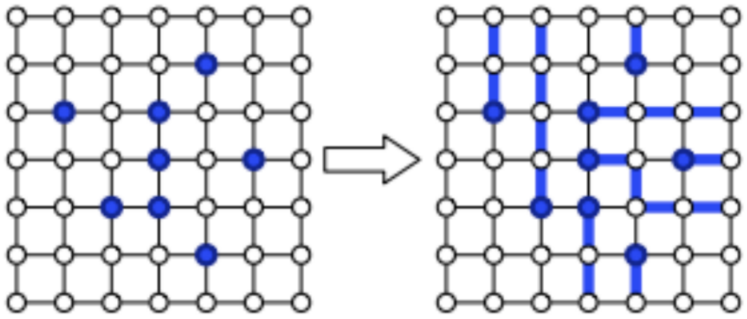
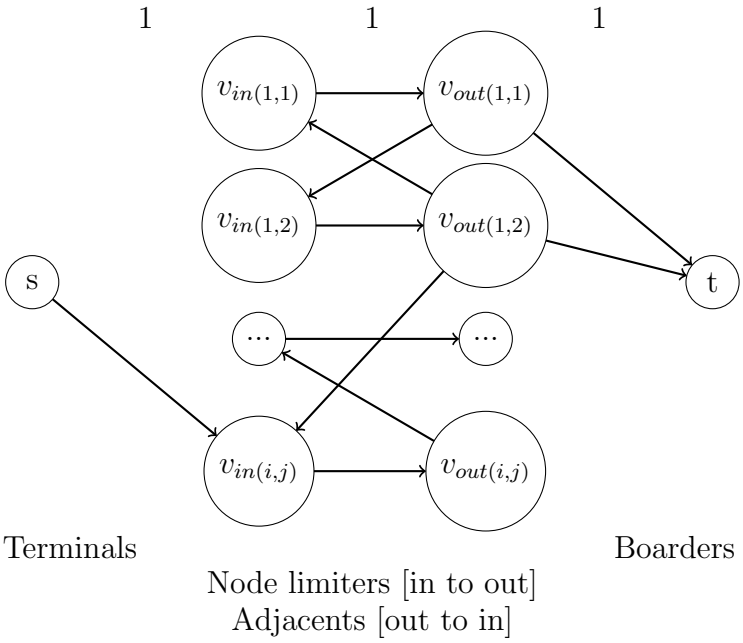


Figure 1: A positive instance of the Escape Problem

Solution:



Run Ford-Fulkerson on the above graph to determine the max-flow, f^*. There is a solution to the Escape Problem if the max-flow is equal to m.

Runtime: The time required for network construction is $O(|V|+|E|)$. The time required to run Ford-Fulkerson is $O(|E| * f^*)$. Therefore, the overall time complexity is $O(|V|+|E|) + O(|E| * f^*)$. This can be rewritten in terms of n and m. In this network $|V| = 2n^2 + 2 = O(n^2)$, $|E| = O(n^2) + O(n^2) + 5O(n^2) = O(n^2)$ and, $f^* = m$. The overall runtime is $O(n^2) + O(mn^2) = O(mn^2)$.

5 Intractability

1. **Independent Set**

Prove that the Independent Set decision problem is NP-Complete.

An independent set is a set of vertices such that no two vertices in S are adjacent to each other. For a given graph $G = (V, E)$ and integer k, the Independent Set decision problem is to find whether G contains an independent set of size $\geq k$.

Solution:

(a) Show Independent Set is in NP.

Given a certificate $V' \subseteq V$, we can verify if V' is a solution in polynomial time. This can be done by checking if an edge exits between any two vertices in V'. If an edge exists then it is not a valid solution. This can be done in $O(|V| + |E|)$ time.

(b) Show Independent Set is in NP-Hard.

The Clique decision problem is in NP-Hard. A clique is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent and therefore form a complete subgraph. For a given graph $G = (V, E)$ and integer k, the Clique decision problem is to find whether G contains an clique of size $\geq k$.

We will show a reduction from Clique to Independent Set (Clique $\leq_p$ Independent Set) to prove that Independent Set is in NP-Hard.

To reduce the Clique problem to Independent Set for a given graph $G = (V, E)$, construct a complementary graph $G' = (V', E')$ such that:

- $V' = V$
- E' is the complement of E. That is G' has all of the edges not in G.

We now must prove that there exists an independent set of size $\geq k$ in G' $\iff$ there exists a clique of size $\geq k$ in G.

$\implies$ If there is an independent set of size k in G', it implies that no two vertices share an edge in G' which further implies all of those vertices share an edge with all others in G forming a clique. Therefore, there is a a clique of size k in G.

$\impliedby$ If there is a clique of size k in G, it implies that all vertices share an edge with all others in G which further implies that no two of these vertices share an edge in G' forming an independent set. Therefore, there is an independent set of size k in G'.