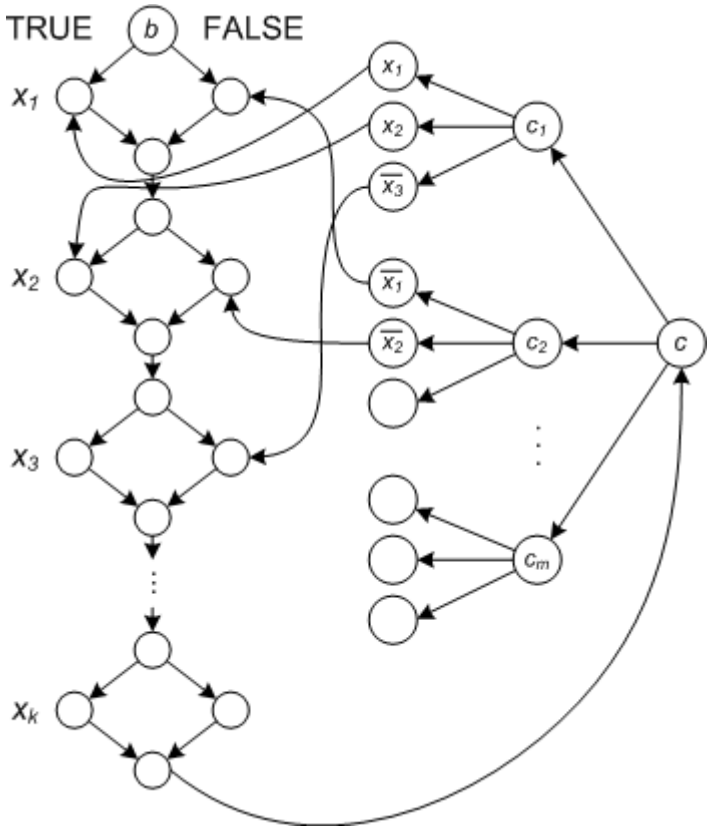
CS240 Homework #5

Zhiqiang Xie 77892769

Problem 1: Geography Game

1. GEOGRAPHY ∈ PSPACE.
    - The same as other two-player games, the search space can be represented in a polynomial-depth tree. Therefore, a stack in polynomial size is sufficient.
2. QSAT $\leq_p$ GEOGRAPHY
    - Construction:
        - Given instance $\Phi(x_1, \ldots, x_k) = C_1 \wedge C_2 \wedge \ldots C_m$ of QSAT, here we assume the k is odd.



(Image is from Wikipedia for convenience)

        - Here we construct a graph, where Each diamond structure corresponds to a quantified variable, the left side is the true assignment and the right side is the false assignment. And the matching literals are connected by a directed edge.

        - We start from node b and we assign $x_1 = T$ if the first player visits the left side, $x_1 = F$ if the first player visits the right side. Then the second player will decide which side of the second diamond structure to visit.
        - Based on the rule of the game, two players will alternately pick the truth value for x_i. Finally, the second player will visit node c (k is odd), and the first player will pick a clause, the second player will pick a literal.
    - Proof:

      We now need to prove that the first player can force a win if and only if QSAT formula is false.
        - $(\Rightarrow)$ Suppose that QSAT formula is false, and we can see that the first player is able to pick false clause with all literals are false. Therefore, no matter how the second player pick the literal, the first player can visit the last node which is not visited. Thus, the first player can force a win.
        - $(\Leftarrow)$ Suppose that the first player can force a win, here is at least one clause with all literals are false since the second player can pick any one of them from the specific clause. Thus, we must have a false clause which leads the QSAT formula to be false. Therefore, QSAT formula is false.

GEOGRAPHY problem is PSPACE-complete.

Problem 2: Geography in DAG

Topologically sort the nodes and denote they as $v_1, v_2, \ldots, v_n$, where v_n has zero out-degree and v_1 has zero in-degree. This procedure costs $O(m+n)$ time.

Now we start from v_n and set $w_n = 0$ which means the players can't force a win if he is looking for a next target at this node. From $i = n-1$ to 1:

- If the out-degree of v_i is 0, then $w_i = 0$
- $\forall k > i$, $w_i = 1$ if and only if there exists an edge (v_i, v_k) and $w_k = 0$
- Otherwise, $w_i = 0$
- This sub-process costs $O(n)$

Finally, for any designated start node v_i, we can simply check w_i for it, which indicates whether a player has a forced win. The time complexity
$T(n) = O(m+n) + n * O(n) \in O(n^2)$

Problem 3: Dominating set in tree

Firstly, we introduce three notations:

- $OPT_{in}(u)$: the dominant set with minimum cost for the u rooted subtree, which includes u.
- $OPT_{ex}(u)$: the dominant set with minimum cost for the u rooted subtree, which excludes u.
- $OPT_{un}(u)$: the dominant set with minimum cost for the u rooted subtree.

Start from the leaf nodes to the root (post-order traversal):

- If u is a leaf node, we initialize it as $OPT_{in}(u) = c(u), OPT_{ex}(u) = \infty, OPT_{un}(u) = 0$.
- Otherwise:
  - $OPT_{in}(u) = c(u) + \sum_{v \in children(u)} OPT_{un}(v)$
  - $OPT_{un}(u) = \sum_{v \in children(u)} min[OPT_{in}(v), OPT_{ex}(v)]$
  - $OPT_{ex}(u) = min_{v \in children(u)}[OPT_{in}(v) + \sum_{w \in children(u), w \neq v} min[OPT_{in}(w), OPT_{ex}(w$

Finally, one we process the dynamic algorithm to the root r, $min[OPT_{in}(r), OPT_{ex}(r)]$ is our minimum cost. And we can get the dominating set from an auxiliary table which are widely used in dynamic programming. The time complexity $T(n) = O(\sum_u n_u) \in O(n)$

Problem 4: Approximation for 3-Dimensional Matching

If A and B are two maximal matchings (M is a maximal matching if it is not a subset of any other matching in graph G):

- Each edge (x_i, y_j, z_k) in $B \backslash A$ can be adjacent to at most three edges in $A \backslash B$ because A is a matching, and the three adjacent edges could be $(x_i, \_, \_), (\_, y_j, \_), (\_, \_, z_k)$.
- Each edge in $A \backslash B$ is adjacent to an edge in $B \backslash A$ since B is a maximal matching.

Thus, $|A \backslash B| \leq 3|B \backslash A|$

Therefore, we have $|A| = |A \cap B| + |A \backslash B| \leq 3|B \cap A| + 3|B \backslash A| = 3|B|$

Here we showed: any maximal matching is a 3-approximation of a minimum maximal matching. In other word, any maximal matching is a $\frac{1}{3}$-approximation of a maximum maximal matching.

Therefore, any algorithm which can collect a maximal matching is correct, such as traversing all the candidates, comparing and picking the edges one by one. The time complexity could be $T(n) = n * O(n) \in O(n^2)$

Problem 5: Unit-size bins

Start from object with size of s_1, put it into the bin b_1:

- In the i^{th} step, we have bins $b_1, b_2, \ldots, b_k$ which are partially packed. If possible, put the object with size of s_i into any one of them. If it does not fit into any of these bins, open a new bin b_{k+1} and put the object in it.

Proof of approximation ratio:

- $OPT \geq \sum_{i=1}^{n} s_i = \sum_{i=1}^{k} w_i$, where w_i is the used amount of bin b_i and k is the number of used bins.
- $\sum_{i=1}^{n} w_i > \frac{1}{2} * (k-1)$, since here is at most 1 bin was packed with size $\leq \frac{1}{2}$

Thus, $OPT > \frac{1}{2} * (k-1), 2 * OPT + 1 > k$ or $2 * OPT \geq k$ because OPT and k are integers.

The time complexity $T(n) = n * O(n) \in O(n^2)$, this is a polynomial-time 2-approximation algorithm.

Problem 6: Approximation for 3-coloring

Algorithm:

- Uniform randomly (IID) color each of the vertices of V by each of the three colors.

Proof:

- The probability of whether an edge is satisfied $P = 1 - C_3^1 * \frac{1}{3} * \frac{1}{3} = \frac{2}{3}$
- The expected number of satisfied edges $N = \frac{2}{3} * |E|$
- Since $OPT \leq |E|$, the expected number $N \geq \frac{2}{3} OPT$. It's $\frac{2}{3}$-approximation