HW2

ZONGQI CUI

September 28, 2024

1 Ploblem 1

1(a) To show that b is surjective, we need to demonstrate that for every node $i \in T$, there exists some vertex $v \in V$ such that $b(v) = i$.

- Suppose there is a node $i \in T$ that is not mapped to by any vertex in V, i.e., $b(v) \neq i$ for all $v \in V$.
- This would imply that the corresponding bag X_i contains no vertices from V. However, this contradicts the definition of tree decomposition, which states that every vertex of G must appear in at least one bag.
- Thus, there must exist some vertex $v \in X_i$ such that $b(v) = i$.

Hence, the function b is surjective, meaning that every node of the tree T is mapped to by some vertex from V. This completes the argument for surjectivity.

1(b)
- S is the intersection of X_i and X_j, so S contains the vertices that are shared between the two parts.
- By the properties of a tree decomposition, every edge in G is contained in some bag. Thus, any path from A to B must pass through the vertices on this path that appear in both X_i and X_j. Hence, the path must include a vertex in S, ensuring that S separates A and B.

1(c) Procedure:

- For each node i in T, if $|X_i| < k+1$, add additional vertices to X_i (if possible) to ensure that $|X_i| = k+1$. These additional vertices should be from the bigger neighboring bags.
- For each edge $\{i, j\}$, if $|X_i \cap X_j| < k$, bulid a new bag between the original bags and the vertices in the new bag should from the intersection of the original bags.
- take first step to make new bag bigger.

2 Problem 2

2(a)

$$B(s, i, j) = A(s^1, i^1) \wedge A(s^2, i^2) \wedge \dots$$

i: all nodes whose parent is j
$s = s^i \cap X_j$

2(b) the formula is:

$$A(s, i) = B(s^1, j^1, i) \wedge B(s^2, j^2, i) \wedge \dots$$

if i has no children, in other words it is a leaf node and then
$A(s, i) = true$

2(c) the number of subproblem is at most 2^k set of S,so the total subproblem will be $O(2^k n)$

3 Problem 3

3(a) n=8:

$$1 \Rightarrow 4 \Rightarrow 3 \Rightarrow 9 \Rightarrow 5 \Rightarrow 10 \Rightarrow 7 \Rightarrow 0$$

3(b) I want to write a function which can solve the problem:

```
int F(int x,int y,int p)
    {
        if(x==y)return -a[x];
        if(p)
        return max(F(x+1,y,0)+a[x],F(x,y-1,0)+a[y]);
```

```
        else
        return max(F(x+1,y,1)-a[x],F(x,y-1,1)-a[y]);
}
```

$p = 1$ means the player1, $p = 0$ means the player2.
$F(x, y, 1)$ means the player1 can get the maximum value from x to y and right now it is player1 to choose.
$F(x, y, 0)$ means the player1 can get the maximum value from x to y and right now it is player2 to choose.
when player1 choose, he will choose the maximum value, that's why we plus $a[x]$ or $a[y]$
when player2 choose, he will also choose the maximum value. we can assume that everytime player2 plus his own score equals to player1 minus his score.
so we can get the function above. when everything is done, we print the details of the function process, and we can know how to choose the best for player2

3(c)
- Each coin can be treated as a polynomial $(1 - p_i) + p_i \times x$,where $p_i$is the probability of heads
- Multiplying these polynomials for all n coins will give you the coefficients that represent the probabilities of getting exactly $0, 1, 2, \ldots, n$heads.
- Using FFT allows you to multiply the polynomials efficiently, reducing the time complexity to $O(nlog^2n)$

4 Problem 4

4(a) don't need to do anything

4(b) When a new edge added to the exiting tree, there must be a cycle appear in the tree. We just need to remove the edge that have the largest value.

4(c) don't need to do anything

4(d) Remove e from the MST, which will disconnect the tree into two components. Find the lightest edge e that reconnects these two components

and add it to the MST.

5 Problem 5

1. we can first use disjoint-set deta structure to manage each node. First we set each node $father[a] = a$ which help us to find its father node.

2. input each edge and sort them by their color(red first). If the number of red edge is less than k, we can announce that no such tree exists.

3. use algorithm which similiar to Kruskal that add the ordered edge to the tree if the two nodes are not in the same set. Using disjoint-set to find whether they were in the same set.

4. if k red edges are added to the tree, we can announce that we have found the tree, else we announce that no such tree exists.

Time complexity: Sorting edges: Sorting red and blue edges takes $O(E)$ and Union-find operations: Union-find takes $O(V)$.
so the total time complexity is $O(E + V)$