Answer Key for Algorithms Design ITU 230109

1a

On Sample input 2, Gordon's algorithm finds a solution of size 4, maybe `1 2 3 4`.

2a

Pip Price

2b

Sort the tiles (a_i, b_i) by the sum $s_i = a_i + b_i$ of their pip values in ascending order. Initialise a counter $c = 0$. Iterate over the sorted list $[s_0, \ldots, s_{T-1}]$ of sums, smallest first, incrementing the counter with the current s_i until all sums are processed or $c + s_i > k$. Return i. Pseudocode:

```
T, k = parse first line
s[0], ..., s[T - 1] = sums of each tile, sorted such that s[i] <= s[i+1]
c = i = 0
while i < T and c + s[i] <= k
    c += s[i]
    i += 1
print(i)
```

2c

$O(T \log T)$, dominated by the sorting step.

3a

Line

3b

Construct an undirected graph G with one vertex for every pip value appearing in the input; there are at most $2T$ such vertices. For each tile (a, b), add the edge between a and b; there are exactly T such edges. Compute the connected component C containing 1, for instance using BFS or DFS in G from vertex 1. Return $\max C$.

3c

Using BFS: $O(2T + T) = O(T)$

4a

Stack

4b

The answer is $\max_{1 \leq i \leq T} \operatorname{opt}(i)$, where

$$\operatorname{opt}(i) = 1 + \max_{1 \leq j \leq T} \{\operatorname{opt}(j) \colon j \neq i, a_j \geq a_i, b_j \geq b_i\}$$

with the convention that $\max \emptyset = 0$.

4c

Time $O(T^2)$, space $O(T)$.

5a

Red and White

5b

This is Maximum Bipartite Matching. There is a vertex in the left (right) part for every red (white) tile; two tiles from different parts share an edge if they have a value in common. The standard reduction turns this into a flow problem on $T + 2$ nodes with $O(T^2)$ arcs of unit capacity. To precise, the node set is $\{s, t\} \cup L \cup R$, there is an arc from s to every node in L, an arc to t from every node in R, and an arc from (the node corresponding to) a red tile (a, b) to (the node corresponding to) a white tile (a', b') if $\{a, b\} \cap \{a', b'\} \neq \emptyset$. All arcs have capacity 1. The size of a maximum flow from s to t is the answer.

5c

Using, say, Ford–Fulkerson's algorithm, the algorithm runs in time $O(T^3)$ (it performs at most T iterations of a path-finding algorithm in a graph with $O(T^2)$ edges.) If you want to be clever, Hopcroft–Karp would reduce this to $O(T^{5/2})$.

6a

Sale

6b

Vertex Cover

6d

Given an instance G, k to (the decision version of) Vertex Cover, construct an instance to Sale as follows. Let $T = |E(G)|$, and for every edge $\{u, v\}$ create a tile (u, v). Let k' be the first integer output by the hypothetical algorithm for Sale. Then the answer to the Vertex Cover instance is "yes" if and only if $k' \leq k$.

Comments on Dynamic Programming

The problem can be viewed as the Longest Path problem in DAG of the partial order defined by the stackability relation, and thus has a well-known solution.

To be more concrete, for a tile t, let t_a and t_b denote the pip-values on its two sides, with $t_a \leq t_b$. Define $\mathrm{opt}(t)$ to be the maximum stack that can be built with t as the base. Then the answer to *Stack* is

$$\max_t \mathrm{opt}(t)\,.$$

The recurrence is

$$\mathrm{opt}(t) = 1 + \begin{cases} \max_{t' \in S(t)} \mathrm{opt}(t') & \text{if } S(t) \neq \emptyset \\ 0 & \text{otherwise}\,, \end{cases}$$

where $S(t)$ are the successors of t in the partial order defined by the stackability relation, *i.e.*, the set of tiles t' such that

$$t_a \leq t'_a \text{ and } t_b \leq t'_b\,.$$

For each tile t it takes linear time to go through all the other tiles t' to check if $t' \in S(t)$. Thus, the algorithm runs in time $O(T^2)$, and takes space $O(T)$.

Pseudopolynomial solution

A slower alternative to 4b is pseudopylnomial and builds a table with entries $\mathrm{opt}(a, b)$ for $0 \leq a \leq b \leq M$, where M is the largest pip value on any tile. The idea is that $\mathrm{opt}(a, b)$ is the height of a stack that can be built on top of a tile with values (a, b) (no matter whether (a, b) is part of the input or not.) Then the answer to *Stack* is $\mathrm{opt}(0, 0)$, and we have the recurrence

$$\mathrm{opt}(a, b) = \mathrm{opt}(a + 1, b) + \mathrm{opt}(a, b + 1) + \begin{cases} 1\,, & \text{if } (a, b) \in S\,; \\ 0\,, & \text{if } (a, b) \notin S\,; \end{cases}$$

where S denotes the set of all tiles in the input.