Contents

Chapter 1

Greedy algorithm

1.1 Problem 5.3

Run DFS on the graph to detect a cycle edge. Return YES as soon as a cycle edge is found. Else, if there is no cycle edge, return NO.
This algorithm has $O(|V|)$ runtime because we note that G is either a tree (in which case $|E| = |V| - 1$) or it is not (in which case $|E| > |V| - 1$).

- If G is a tree then we will not be able to detect any back edge. DFS will traverse the entire graph, which takes $O(|V| + |E|)$, but because $|E| = |V| - 1$, this is $O(|V|)$.
- If G is not a tree then we can find a back edge after traversing at most $|V|$ edges because the edges picked by DFS form a tree, and any tree in the original graph can have at most $|V|$ vertices.

1.2 Problem 5.4

We note that a connect component with m vertices must have at least $m - 1$ edges[1].

Let the number of vertices in component i be $m_i,\ i = 1, 2, \ldots, k$. We have $\sum_{i=1}^{k} m_i = n$.

The number of edges in the graph is the total number of edges in all components, which is at least

$$\sum_{i=1}^{k} (m_i - 1) = n - k.$$

1.3 Problem 5.5

(a) We follow Kruskal's algorithm to build the minimum spanning tree: at each step, pick the edge with the least weight that does not create a cycle. Because all edge weights are increased by 1, the weight of any edge relative to all other edges is the same, so Kruskal's will produce the same result.

(b) The shortest path will change. Consider the quadrilateral $ABCD$ with

$$AB = BC = CD = 2, AD = 7.$$

Currently the shortest path from A to D is $A \to B \to C \to D$. If we increase the weight of all edges by 1 then $AB = BC = CD = 3, DA = 8$, so the shortest path from A to D is the $A \to D$.

[1]since the component is connected, we can build a minimum spanning tree in it; this tree has m vertices and $m - 1$ edges, so the number of edges in the component must be at least $m - 1$.

1.4 Problem 5.6

Sort the edge by their weights

$$e_1 < e_2 < \ldots < e_n.$$

Kruskal's algorithm will iterate n times and, at time i, pick edge e_i, or the $i-$th smallest edge in the graph. Because the edge weights are distinct, the $i-th$ smallest edge in the graph is distinct for all $1 \leq i \leq n$. Therefore the minimum spanning tree is unique.

1.5 Problem 5.7

Negate all the edge weights in the input graph and use Kruskal's algorithm to find the minimum spanning tree in the new graph. This tree is the maximum spanning tree in the original graph.

1.6 Problem 5.8

We use Prim's algorithm to build a minimum spanning tree in G.
Start building the MST from a vertex that is not S. Now assume we have reached the step where we have the tree current tree T and we are about to add S to $T(V)$. In other words, the next edge to be added to $T(E)$ is the edge SA connecting S to a vertex $A \in T(V)$. $(*)$
We prove that SA is also the shortest path from S to A and therefore T and the tree of shortest paths from SA share the same edge SA.
Assume otherwise, the shortest path from S to A is not SA. Then there exists a vertex $B \neq A$ such that the shortest path from S to A consists of the shortest path from S to B and BA. So

$$d(S, B) + BA < SA \Rightarrow BA < SA.$$

We consider two cases:

- If $B \notin T(V)$ then because $BA < SA$, SA is not the lightest edge that connects T to a vertex outside of it.
- If $B \in T(V)$, call I and K two vertices on the path from S to B such that $I \in T(V), K \notin T(V)$ (I can be B and K can be S). Then $SA > d(S, B) = d(S, K) + IK + d(I, B) \geq IK$, so again SA is not the lightest edge.

Both cases contradict $(*)$. Thus we have proven that the minimum spanning tree and the tree of shortest paths from a vertex S always share an edge.

1.7 Problem 5.10

Start with a MST $T_H \in MST_H$ and $T_G \in MST_G$.
While there is an edge $e \in T_G \bigcap H$ such that $e \notin T_H$ do:

1. Add e to T_H to create a cycle C.
2. We see that for all $e' \in C, e' \neq e$ we have $w(e') \leq w(e)$. Otherwise if $w(e') > w(e)$ we should have picked e, not e' when building the MST T_H, according to Kruskal's algorithm.
3. Let $e = (u, v)$, so $u, v \in H$. We see that $e \in T_G$ so it connects two previously separate connected components, which we call U and V, and assume that $u \in U, v \in V$. Because u and v are in H and T_H is the MST of H, there exists an edge $e'' \in C \bigcap T_H$ that connects U and V.
4. From Step 2 we get $w(e'') \leq w(e)$. If $w(e'') < w(e)$ then when we built T_G we should have picked e'' instead of e to connect U and V. Therefore $w(e'') = w(e)$.

5. Let $T'_H = T_H \bigcup \{e\} - \{e''\}$ then T_H is also a MST in H.
6. Rename $T'_H \to T_H$ and check the loop condition.

After the loop we have $T_G \bigcap H \subset T_H$.

Chapter 2

Dynamic programming

2.1 Problem 6.2

Let $b[i]$ be the minimum total penalty for stopping at hotel a_i, $1 \leq i \leq n$. We have

$$b[i] = \min_{1 \leq j < i} \{b[j] + (200 - (a[i] - a[j]))^2\}.$$

Also record the value j which yields $\min_{1 \leq j < i} \{b[j] + (200 - (a[i] - a[j]))^2\}$ and set `b[i].prev = b[j]`. Backtrack from $b[n]$ to get the sequence of hotels to stop by.

2.2 Problem 6.3

Let $S[i]$ be the maximum total profit we get from building some restaurants in $\{m_1, m_2, \ldots, m_i\}$. Consider 2 cases:

(1) If restaurant m_i should not be built, then $S[i] = S[i-1]$.

(2) If restaurant m_i should be built, then let c_i be the maximum index j which yields $m_i - m_j \geq k$. We then have $S[i] = p_i + S[c_i]$.

Therefore in general,

$$S_i = \max\{S[i-1], p_i + S[c_i]\}.$$

To get the sequence of restaurants, keep an array $R[n]$ such that $R[i] = 1$ if restaurant i is built in the optimal solution, and $R[i] = 0$ otherwise. When we calculate $S[i]$, if the max falls to case (1), $R[i] = 0$. Else, $R[i] = 1$. Output all the $R[i]$s that are 1.

2.3 Problem 6.4

Consider an array $S[n]$ where $S[i] =$ `true` if the substring $s_1 s_2 \ldots s_i$ is a valid string, and `false` otherwise. We have $S[1] =$ `dict(s[1])` and

$$\begin{aligned} S[i] = (&S[1] \;\&\&\; \texttt{dict(s[2 .. i])} \\ &||(S[2] \;\&\&\; \texttt{dict(s[3..i])} \\ &|| \ldots \\ &||(S[i-1] \;\&\&\; \texttt{dict(s[i..i])})), \end{aligned}$$

where $s[j..i]$ is $s_j s_{j+1} \ldots s_i$.

2.4 Problem 6.6

Let the input string be $x_1 x_2 \ldots x_n$.
Let $Z = \{a, b, c\}$ and let $T[i, j] \subset Z$ be the set of the possible values that the product $x_i x_{i+1} \ldots x_j$ can yield with all possible parenthesizations.
We see that $T[i, i] = x_i$ for all $1 \leq i \leq n$. We need to compute $T[1, n]$.
Define $A \times B$ as $\{a \cdot b | a \in A, b \in B\}$.
We note that $T[i, i+1] = T[i, i] \bigcup T[i+1, i+1]$ and $T[i, i+2] = (T[i, i] \times T[i+1, i+2]) \bigcup (T[i, i+1] \times T[i+2, i+2])$ (to put it another way, abc can be written as $(a)(bc)$ or $(ab)(c)$).
We therefore see that we already have

$$T[1, 1], T[2, 2], T[3, 3], \ldots,$$

from which we can calculate

$$T[1, 2], T[2, 3], T[3, 4], \ldots,$$

from which we can calculate

$$T[1, 3], T[2, 4], T[3, 5], \ldots$$

and eventually we can expand to $T[1, n]$, which is what we need to find.
In other words,

$$T[i, i+s] = \bigcup_{i \leq k < i+s} (T[i, k] \times T[k+1, i+s]).$$

The algorithm is as follows:

```
for i = 1 to n: T[i,i] = x[i].
for s = 1 to n-1:
  for i = 1 to n - s:
    T[i, i + s] = empty
    for k = 1 to i + s - 1:
      T[i, i + s] = T[i, i + s] UNION (T[i, k] * T[k+1,s])
If a is in T[1,n] return true. Else, return false.
```

2.5 Problem 6.7

Let the input string be $x_1 x_2 \ldots x_n$. Let $T[i, j]$ be the length of the longest palindromic subsequence in $x[i..j]$. We have

$$\begin{cases} T[i.i] = 1 \\ T[i, i+1] = 2 \text{ if } x[i] = x[i+1] \text{ and } 0 \text{ if } x[i] \neq x[i+1] \\ T[i, j] = T[i+1, j-1] + 2 \text{ if } x[i] = x[j] \text{ and } \max\{T[i+1, j], T[i, j-1]\} \text{ else} \end{cases}$$

2.6 Problem 6.8

Let $E[i, j]$ be the length of the largest common substring of $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$ such that $x_i = y_j$.
We see that $E[1, j] = 1$ if $x_1 = y_j$ and 0 otherwise. Similarly, $E[j, 1] = 1$ if $y_1 = x_j$ and 0 otherwise.
In general, we have

$$E[i, j] = \begin{cases} E[i-1, j-1] + 1 \text{ if } x_i = y_j \\ 0 \text{ if } x_i \neq y_j \end{cases}$$

2.7 Problem 6.9

Let the input string be $x[0..n-1]$ and the input breakpoint array be $y[1..m]$. Convert y to $y[0..m+1]$ and let $y[0] = -1, y[m+1] = n-1$.
Let $M(i,j)$ be

$$\begin{cases} M(i,i) = 0, \forall i : 0 \le i \le m+1 \\ M(i,i+1) = 0, \forall i : 0 \le i \le m+1 \\ M(i,j) = (y[j] - y[i]) + \min\limits_{l: i<l<j} \{M(i,l) + M(l,j)\} \end{cases}$$

2.8 Problem 6.10

Let $E[i,j]$ be the probability of obtaining exactly i heads when j coins $c_1, c_2, \ldots, c_j$ with head-probability $p_1, p_2, \ldots, p_j$ are tossed. We have

$$\begin{cases} E[0,0] = 1 \\ E[0,j] = E[0,j-1] \cdot (1-p_j) \ \forall 1 \le j \le n \\ E[i,0] = 0 \ \forall 1 \le i \le k \\ E[i,j] = p_j \cdot E[i-1,j-1] + (1-p_j) \cdot E[i,j-1] \end{cases}$$

2.9 Problem 6.11

Let $E[i,j]$ be the longest common subsequence of $x_1x_2\ldots x_j$ and $y_1y_2\ldots y_j$.
We have $E[1,j] = 1$ if $x_1 = y_j$, for all $1 \le j \le m$. Similarly, $E[j,1] = 1$ if $x_j = y_1$, for all $1 \le j \le n$.
In general we have

$$E[i,j] = \begin{cases} E[i-1,j-1] + 1 \text{ if } x_i = y_j \\ \max\{E[i-1,j] + E[i,j-1]\} \text{ if } x_i \neq y_j. \end{cases}$$

2.10 Problem 6.12

Let $d[i,j]$ be the distance between point i and j. We have

$$\begin{cases} A[i,i] = 0 \\ A[i,i+1] = 0 \\ A[i,i+2] = 0 \\ A[i,j] = \min\limits_{i<k<j} \{A[i,k] + A[k,j] + d[i,k] + d[k,j]\} \end{cases}$$

2.11 Problem 6.13

A sequence where a greedy approach would fail is

$$1000 \quad 2000 \quad 1.$$

Let $E[i,j]$ be the maximum value the first player can have by picking cards from the set of cards $s_i, s_{i+1}, \ldots, s_j$.
We see that $E[i,j] = 0$ if $i \le j$. In general,

$$\begin{aligned} E[i,j] = \max\{&v_i + \min\{E[i+2,j] - v_{i+1}, E[i+1,j-1] - v_j\}, \\ &v_j + \min\{E[i+1,j-2] - v_{j-1}, E[i+2,j-1] - v_{i+1}\}\}. \end{aligned}$$