Solutions of Introduction to Algorithms: A Creative Approach

Saman Saadi

Contents

1 Mathematical Induction **1**

1.1 Counting Regions in the Plane 1

1.2 Euler's Formula 2

1.3 Gray Codes 2

1.3.1 Implementing $\lfloor \log_2 n \rfloor$ and $\lceil \log_2 n \rceil$ 5

1.4 Website Questions 6

1.4.1 SRM 784 - Division II, Level One: Scissors 6

Chapter 1

Mathematical Induction

1.1 Counting Regions in the Plane

A set of lines in the plane is said to be in **general position** if no two lines are parallel and no three lines intersect at a common point.

Guess: Adding one more line to $n-1$ lines in general position in the plane **increases** the number of regions by n. In other words $T(n) = T(n-1) + n$.

The base cases is trivial

- $T(0) = 1$
- $T(1) = T(0) + 1 = 2$
- $T(2) = T(1) + 2 = 2 + 2 = 4$
- $T(3) = T(2) + 3 = 4 + 3 = 7$

So we assume $T(n)$ is correct, now we want to prove $T(n+1)$ is also correct. Let's remove line n^{th}. According to induction hypothesis Adding line $(n+1)^{th}$ add n new regions. If we add line n^{th} again, it intersect with line $(n+1)^{th}$ at exactly one point p. This point is located in region R.

In the absence of line n^{th}, line $(n+1)^{th}$ adds only one new region when it passes R. But in presence of line n^{th}, it adds 2 new regions when it passes R. For other regions line $(n+1)^{th}$ adds $n-1$ new regions with or without the presence of line n^{th}. So line $(n+1)^{th}$ adds $n-1+2 = n+1$ new regions when n^{th} is presented.

So instead of proving the number of regions by adding a new line, we proved how many new regions are added when we have line $(n+1)^{th}$. So It's easy to prove the number of regions. Starting with one line we have $2{+}2{+}3{+}4{+}\cdots{+}n = 1 + 1 + 2 + \cdots + n = 1 + \frac{n \times (n+1)}{2}$.

1.2 Euler's Formula

Consider a connected planar map with V vertices, E edges and F faces. A face is an enclosed region. The outside region is counted as one face. So for example, a square has four vertices, four edges and two faces.

Theorem The number of vertices (V), edges (E), and faces (F) in an arbitrary connected planar map are related by the formula $V + F = E + 2$.

Proof It's clear the formula doesn't hold if the planar map is not connected. So we cannot just simply remove an edge. So the base case should be a tree.

Theorem In a tree with V vertices, the number of edges E is $E = V - 1$.

Proof The base case is trivial. Suppose it's true for all trees with V vertices. Now consider a tree with $V+1$ vertices. There should be at least one vertex connected to only one edge. If we don't have such a vertex, then we can start from an arbitrary vertex v and try to visit other vertices. Since each vertex has at least 2 edges, we can easily enter and exit other vertices and visit edges at most once. Since the number of vertices are limited, then we should revisit a vertex. It implies a cycle which is a contradiction. So we have at least one vertex that is connected to only one edge. If we remove that vertex and that edge, the tree is still connected so we can use hypothesis so $E = V - 1$. So by adding one vertex and one edge, the formula is also correct for $V + 1$ vertices.

So the base case is tree. A tree only has one face. So we have $V + 1 = V - 1 + 2$.

Now consider a planar map which is not tree. In other words, it has at least one cycle. If we remove one edge from that cycle, It's still connected. By removing that edge, the inner face will be combined with outer face. So the number of faces also reduced by 1. So the formula is correct. The textbook choose faces as induction parameter but it actually remove an edge. So I don't see any different between choosing edge or face as induction parameter.

1.3 Gray Codes

Gray codes are strings of 0s and 1s in such a way that two neighbours only differ in one digit. For example 100 and 101. If the last string respect the condition with the first one, the code is closed; otherwise it's open. For example 00, 01, 11 and 10 is closed but 00, 01, 11 is open.

Hypothesis: There exists gray codes of length $\lceil \log_2 k \rceil$ for all values $k < n$. If k is even, then the code is closed; if k is odd, then the code is open.

Proof: There are two scenarios:

- If $n = 2m$ We use the hypothesis and assume $s_1, s_2, \ldots, s_m$ are gray codes. It can be open or closed. We can create gray codes of size n which is closed like $0s_1, 0s_2, \ldots, 0s_m, 1s_m, \ldots, 1s_2, 1s_1$.

  Based on the hypothesis the length of s_i $(1 \le i \le m)$ is $\lceil \log_2 m \rceil$. So the length of $0s_i$ is $\lceil \log_2 m \rceil + 1$:

$$\begin{aligned}\lceil \log_2 m \rceil + 1 &= \lceil log_2 \frac{n}{2} + 1 \rceil \\ &= \lceil \log_2 n - log_2 2 + 1 \rceil \\ &= \lceil \log_2 n \rceil\end{aligned}$$

- If $n = 2m+1$ We use the hypothesis and assume $s_1, s_2, \ldots, s_m, s_{m+1}$ are gray codes. It can be opne or closed. We can create gray codes of size n which is open like $0s_1, 0s_2, \ldots, 0s_m, 0s_{m+1}, 1s_{m+1}, 1s_m, \ldots, 1s_2$.

  Based on the hypothesis the length of s_i $(1 \le i \le m+1)$ is $\lceil \log_2 (m+1) \rceil$. So the length of $0s_i$ is $\lceil \log_2 (m+1) \rceil + 1$:

$$\begin{aligned}\lceil \log_2 (m+1) \rceil + 1 &= \lceil log_2 (\frac{n-1}{2} + 1) + 1 \rceil \\ &= \lceil \log_2 (\frac{n+1}{2}) + 1 \rceil \\ &= \lceil \log_2 (n+1) - log_2 2 + 1 \rceil \\ &= \lceil \log_2 (n+1) \rceil\end{aligned}$$

  Since n is odd we have $\lceil \log_2 (n) \rceil = \lceil \log_2 (n+1) \rceil$.

  Note that we cannot find another list that is closed. Because we need to find a list that stars with $0s_1$ and ends with $1s_1$. We generate s_{i+1} by flipping exactly one bit of s_i. So $0s_1$ bits are flipped in total m times to generate $0s_{m+1}$. Then we put $1s_{m+1}$ to the second list. We need to flip $1s_{m+1}$ bits $m-1$ times to generate the rest of the list. Therefore s_1 bits are flipped $m + m - 1 = 2m - 1$ which is an odd number. So it's impossible the list starts with $0s_1$ and ends with $1s_1$.

The implementation in C++:

```
string str;
void grayCodes(size_t index, size_t listLength)
{
  if (listLength == 1)
  {
    cout << str << endl;
    return;
  }
  const auto m = listLength / 2;
  if ((listLength % 2 ) == 0)
    grayCodes(index + 1, m);
  else
```

```
    grayCodes(index + 1, m + 1);
  str[index] = (str[index] == '0' ? '1' : '0');
  grayCodes(index + 1, m);
}

int main()
{
  const size_t n = 9;
  const size_t codeLen = [n]()
  {
    size_t res = 0;
    //ceil(lg(m)):
    for (size_t m = 1; m < n; m *= 2)
      ++res;
    return res;
  }();
  str = string(codeLen, '0');
  grayCodes(0, n);
}
```

Note that this is not the exact implementation of the proof. For example for $n = 6$ we have:

$$\begin{gathered} 0\ 00 \to 0s_1 \\ 0\ 01 \to 1s_2 \\ 0\ 11 \to 1s_3 \\ \vdots \\ 1\ 11 \to 1s_3 \\ 1\ 10 \to 1s_2' \\ 1\ 00 \to 1s_1 \end{gathered}$$

How this implementation always generate closed gray codes when n is even? Suppose $n = 2m$. The generated code should have the following structure to consider it as closed:

$$\begin{array}{l} 0s_1 \\ 0s_2 \\ \vdots \\ 0s_m \\ 1s_m \\ 1s'_{m-1} \\ \vdots \\ 1s'_2 \\ 1s_1 \end{array}$$

Suppose $s = (b_{k-1} \dots b_1 b_0)_2$ represents a code. Based on definition $k = \lceil \log_2 m \rceil$. For generating s_1 to s_m we flip s digits $m-1$ times. We define c_i as the number of times bit ith is flipped. It's obvious that:

$$m - 1 = \sum_{i=0}^{k-1} c_i$$

For the second half of the list which starts with $1s_m$ we use the same pattern to flip bits exactly $m-1$ times. So in total we have:

$$2 \times (m-1) = \sum_{i=0}^{k-1} 2 \times c_i$$

So the ith bit is flipped $2 \times c_i$ which is an even number. In other words if we flip the ith bit of s_0, c_i times for all $0 \le i \le k-1$ we get s_m. If we flip the ith bit of s_m, c_i times for all $0 \le i \le k-1$ we get s_0. So the list starts with $0s_1$ and ends with $1s_1$.

1.3.1 Implementing $\lfloor \log_2 n \rfloor$ and $\lceil \log_2 n \rceil$

Note that calculating $\lceil \log_2 n \rceil$ can be tricky. According to definition we have:

$$\lceil \log_2 n \rceil = r \implies 2^{r-1} < n \le 2^r$$
$$\lfloor \log_2 n \rfloor = r \implies 2^r \le n < 2^{r+1}$$

So for $\lceil \log_2 n \rceil$ we are looking for the maximum 2^i which is smaller than n. When we find it the answer is $i+1$. In each iteration as long as $2^i < n$, we can assume the result is at least $i+1$:

```
int ceilLogarithm(int n)
{
    int r = 0;
    for (int m = 1; m < n; m *= 2)
      ++r;
    return r;
}
```

For calculating $\lfloor \log_2 n \rfloor$ we are looking for the minimum 2^i which is bigger than n. When we find it the answer is $i - 1$. In each iteration as long as $2^i \leq n$, we can assume the result is at least i:

```
int floorLogarithm(int n)
{
    int r = 0;
    for (int m = 2; m <= n; m *= 2)
      ++r;
    return r;
}
```

1.4 Website Questions

1.4.1 SRM 784 - Division II, Level One: Scissors

You are in charge of N other people. You have a pair of scissors. But none of your N helpers do.

You purchased N pairs of scissors which are wrapped in plastic. Getting scissors out of the plastic wrap requires having another pair of scissors (that's not in plastic) and it takes 10 seconds. Assume that everything other than opening the packages happens instantly.

Calculate and return the shortest amount of time (in seconds) in which it is possible to release all the scissors from their plastic wraps.

For more information visit this website.

Solution There are two methods which are similar to each other. Note that how using different approaches to break down the problem can make it easier to understand.

Method 1 Suppose that we have k knives:

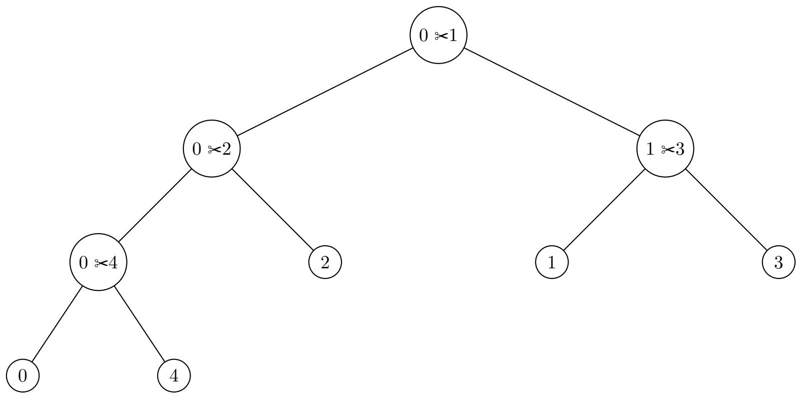
Hypothesis: We know the solution for $n < N$ and $1 \leq k \leq N + 1$

Proof: Note that we must prove that it also correct for $n = N$ and $1 \leq k \leq N + 1$. So if we have N unwrapped packages and k knives ($1 \leq k \leq N + 1$), we can unwrap at most k packages. After it we have at most $k + k$ knives.

$$T(n,k) = \begin{cases} T(n-k, 2k) + 10 & k \leq n \land n \neq 0 \\ 0 & n = 0 \\ T(0, k+n) + 10 & k > n \land n \neq 0 \end{cases}$$

The solution is $T(N, 1)$. Because at the beginning we only have one unwrapped knife.

Method 2 Intuitively we can use a binary tree. Each node has out-degree 0 (leaf) or 2 (node). For example for $N = 4$, we show the pair of scissors $1 \leq i \leq N$ as i. The first pair of scissors which are unwrapped is shown as 0:



As you can see at the beginning we only have a pair of scissors 0. Using it we unwrap the pair of scissors 1. In the next step we unwrap 2 with 0 and 3 with 1 simultaneously. So for $N = 4$, the answer is $h \times 10 = 3 \times 10 = 30$. h is the height of the tree. We can solve it using mathematical induction.

Hypothesis: We know the solution for $n < N$ pairs of wrapped scissors.

Proof: Suppose $n = N$. For an optimal solution we want to use as many pairs of scissors as possible to unwrap the remaining ones. Suppose $T(n)$ is the minimum time required to unwrap n pairs of scissors. Of course after we do that we have $n + 1$ pairs of scissors (remember you have a pair of unwrapped scissors). For edge cases we consider two possible scenarios:

1. $n = N = 2k+1$ We use the hypothesis and find the answer for the first k wrapped pairs of scissors ($T(k)$). Then using those k ones plus the first one we can unwrapped the remaining $k + 1$ ones:
$$T(2k + 1) = T(k) + 10$$
2. $n = N = 2k$ We use the hypothesis and unwrapped the first k pairs of scissors. Besides those k pairs of scissors, we have another one which is unwrapped from the beginning but we only have k pairs of wrapped scissors. We don't use one of those $k + 1$ ones:
$$T(2k) = T(k) + 10$$

Note that the following equation is not always correct:

$$T(2k) = T(k-1) + 10 \times 2$$

Because by unwrapping the first $k-1$ ones, we have k pairs of unwrapped scissors and $k+1$ wrapped ones. On the other hand, $T(k) = T(k-1) \vee T(k) = T(k-1) + 10$. So $T(2k) = T(k) + 10$ finds the optimal solution.

We can squeeze both equations into one:

$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + 10 & n > 0 \\ 0 & n = 0 \end{cases}$$