Generating Indecomposable Permutations

Andrew King
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada

Abstract

An indecomposable permutation π on $[n]$ is one such that $\pi([m]) = [m]$ for no $m < n$. We consider indecomposable permutations and give a new, inclusive enumerative recurrence for them. This recurrence allows us to generate all indecomposable permutations of length n in transposition Gray code order, in constant amortized time (CAT). We also present a CAT generation algorithm which is based on the Steinhaus-Johnson-Trotter algorithm for generating all permutations of length n. The question of whether or not there exists an adjacent transposition Gray code for indecomposable permutations remains open.

1 Introduction

A permutation π on the interval $[n]$ is indecomposable if and only if $\pi([m]) = [m]$ for no $m < n$. In other words, if and only if it has no proper prefix which is itself a permutation. It is easy to see that there is one indecomposable permutation of length 1, one such permutation of length 2, and three such permutations of length 3.

Indecomposable permutations (sometimes called irreducible permutations) were introduced by Comtet [1, 2], who enumerated them and discussed them in the more general context of permutations with a given number of components (see [2], Exercise 6.14). They have since been investigated in several

contexts, mostly combinatorial and algebraic. We seek to generate them quickly and in a meaningful order.

2 Combinatorial Issues

Let the set of indecomposable permutations of length n be denoted I_n. Comtet [1, 2] noted that $|I_n|$, the number of indecomposable permutations of length n, has the generating function

$$f(t) = 1 - \frac{1}{\sum_{n=1}^{\infty} n! t^n}. \tag{1}$$

The well-known recurrence for $|I_n|$ considers cases of permutations which are not indecomposable. Consider the number of decomposable permutations π on $[n]$ having i as the smallest integer such that $\pi([i]) = [i]$. It is easy to see that there are $|I_i|$ such prefixes, and the other $n-i$ elements can be permuted arbitrarily, therefore there are $|I_i|(n-i)!$ such sequences. The prefix length i lies between 1 and $n-1$, and there are $n!$ permutations in total. This yields the recurrence

$$|I_n| = n! - \sum_{i=1}^{n-1} |I_i|(n-i)! \tag{2}$$

This recurrence is simple, but not particularly useful, as it uses exclusion and is therefore unlikely to help us in finding a Gray code. The more useful recurrence follows:

Theorem 1.

$$|I_n| = \sum_{r=2}^{n} \sum_{j=0}^{r-2} |I_{n-j-1}| j! \tag{3}$$

$$= \sum_{j=0}^{n-2} (n-j-1) |I_{n-j-1}| j! \tag{4}$$

Proof. Consider the number of indecomposable permutations on $[n]$ with first element r. Remove the first element and subtract 1 from any element greater than r. The result is a permutation π on $[n-1]$. Consider the

j	$r=2$	$r=3$	$r=4$	$r=5$
3				**51234**
2			**41253**	51243
3				**51324**
1		**31452**	41352	51342
1		**31524**	41523	51423
1		**31542**	41532	51432
3				**52134**
2			**42153**	52143
3				**52314**
0	**23451**	32451	42351	52341
0	**23514**	32514	42513	52413
0	**23541**	32541	42531	52431
3				**53124**
0	**24153**	34152	43152	53142
3				**53214**
0	**24351**	34251	43251	53241
0	**24513**	34512	43512	53412
0	**24531**	34521	43521	53421
0	**25134**	35124	45123	54123
0	**25143**	35142	45132	54132
0	**25314**	35214	45213	54213
0	**25341**	35241	45231	54231
0	**25413**	35412	45312	54312
0	**25431**	35421	45321	54321

Table 1: Indecomposable permutations with parameters r and j for $n = 5$

largest $m < n$ such that $\pi([j]) = [j]$ (let $j = 0$ if none exists). Remove this prefix and subtract j from each element. The result is a permutation on $[n-j-1]$, and this must be indecomposable since decomposability would imply that j was chosen incorrectly. So there are $|I_{n-j-1}|$ such suffixes, and for it there are $j!$ possible prefixes of length j. Since the original permutation is indecomposable, $0 \leq j \leq r-2$. This yields the first identity. The second follows by simple arithmetic. See Table 1 as an example of the parameters r and j. □

This parameterization is essential to generating the permutations in Gray

code order in Section 4.

3 Generation in SJT Order

The Steinhaus-Johnson-Trotter (SJT) algorithm (see [5], p. 136) is a CAT generation algorithm for all permutations on $[n]$ in adjacent transposition Gray code order. Algorithm 1 generates all permutations in the same order as the SJT algorithm, put outputs only those which are indeomposable.

Algorithm 1 can be turned into the SJT algorithm by removing lines 9 to 16 and changing the condition on line 4 to be merely $m > n$. Initially, $spp[i] = 1$ and $p[i] = i$ for all i. As with the SJT algorithm, the initial call is Perm(1).

Algorithm 1 Generate indecomposable permutations in SJT order

```
 1: procedure Perm ( int m )
 2: local int i, j, t
 3: begin
 4:   if m > n and spp[n] = n then
 5:     Printit;
 6:   else
 7:     Perm(m + 1);
 8:     for i := 1 to m − 1 do
 9:       p[m] := p[m] + dir[m];
10:       for j := m to n do
11:         if p[j] ≤ s[j − 1] then
12:           spp[j] := j;
13:         else
14:           spp[j] := spp[j − 1];
15:         end if
16:       end for
17:       t := π^{-1}[m];   π[t] := π[t + dir[m]];   π[t + dir[m]] := n;
18:       π^{-1}[m] := t + dir[m];   π^{-1}[π[t]] := t;
19:       Perm(n + 1);
20:     end for
21:   end if
22:   dir[m] := −dir[m];
23: end
```

$p[i]$ represents the position that i would hold in the permutation if all numbers greater than i were to be removed. For example, in the permutation 635142, p would be [1, 2, 1, 3, 2, 1]. This explains line 9, because as a property of the SJT algorithm, m is always transposed with a smaller number, so $p[m]$ will always change by 1, and no other entry of p will change.

$spp[i]$ is the length of the smallest prefix which is itself a permutation, once all numbers greater than i have been removed. For example, in the permutation 635142, spp would be [1, 1, 3, 4, 5, 6]. Note that $spp[1] = 1$ always, and for $i > 1$, $spp[i] = i$ if and only if removing all numbers greater than i leaves an indecomposable permutation. In this example, 1, 312, 3142, 35142, and 635142 are all indecomposable.

Lemma 2. *Let π be a permutation on $[n]$. For $1 < i \leq n$,*

$$spp[i] = \begin{cases} i & \text{if } p[i] \leq spp[i-1] \\ spp[i-1] & \text{otherwise} \end{cases} \tag{5}$$

This follows from the fact that, when inserting i into a permutation on $[i-1]$, if i comes before the end of the smallest prefix which is itself a permutation, the result will be an indecomposable permutation. If i comes after the end of this prefix, then the existing prefix will remain the shortest such prefix.

Theorem 3. *Algorithm 1 generates all indecomposable permutations, and no others.*

Proof. It is clear that a permutation on $[n]$ is indecomposable if and only if $spp[n] = n$, so it remains only to show that lines 9 through 16 maintain p and spp correctly. We have already illustrated that line 9 increments or decrements $p[m]$ appropriately.

Lemma 2 establishes the appropriate value for $spp[n]$ when n is inserted into a permutation on $[n-1]$. Moving m in π in the algorithm will never change $spp[i]$ for $i < m$; that much is clear. However, it can change $spp[i]$ for $i > m$. By the claim, we can correctly maintain spp by recomputing $spp[m], spp[m+1], \ldots, spp[n]$ in that order. Hence lines 9 through 16 correctly maintain p and spp.

Therefore, adding to the SJT algorithm the specification that a permutation is printed if and only if $spp[n] = n$ ensures that the algorithm generates exactly all indecomposable permutations. □

Theorem 4. *Algorithm 1 runs in constant amortized time.*

Proof. We know from Comtet ([1]) that $\lim_{n\to\infty} |I_n|/(n!) = 1$, and we know further that $n!/|I_n| \leq 2$. Therefore it suffices to show that the algorithm's running time is bounded by a constant factor with respect to the running time of the SJT algorithm, since the SJT algorithm is itself CAT.

Modifying line 4 and adding line 9 clearly adhere to this constraint (because they do no more than add a constant amount of work to each node in the computation tree), so we need only be concerned about the **for** loop beginning at line 10. This loop does a constant amount of work $n - m$ times at each node at distance m from the root of the computation tree, for $m = 0, 1, 2, \ldots, n$. There are $m!$ nodes at distance m from the root, so let us bound the total amount of work done in the tree.

We take as granted that for $n > 0$, $\sum_{i=0}^{n} i! \leq 2 \cdot n!$. This can be proven trivially by induction. Let $W(n)$ be the number of times the inner **for** loop (line 10) is run through in total. Now consider the computation tree for $W(n+1)$. It is the same as the tree for $W(n)$ but with $n+1$ children added to each leaf, and with the inner **for** loop run through one extra time in every node at distance $\leq n$ from the root. There are $\sum_{i=0}^{n} i!$ such nodes, so $W(n+1) = W(n) + \sum_{i=0}^{n} i! \leq W(n) + 2 \cdot n!$. The sequence $\{W(n)\}_{n=1}^{\infty}$ begins $1, 3, 7, 17, 51, \ldots$, so we will show that for $n \geq 4$, $W(n) < n!$, using the basis $W(4) = 17$. Take $n \geq 4$ and suppose that $W(n) < n!$.

$$\begin{aligned} W(n+1) &\leq W(n) + 2 \cdot n! \\ &< 3 \cdot n! \\ &< (n+1)! \end{aligned} \tag{6}$$

So $W(n)$ is bounded by a constant times the number of nodes in the computation tree for $\mathsf{Perm}(n)$. Therefore Algorithm 1 runs in constant time amortized over the number of permutations output by the SJT algorithm, and therefore over the number of its own output permutations. □

4 Generation in Gray Code Order

In this section we first present the theory behind the Gray code, then present the Gray code generation algorithm.

4.1 Existence of a Gray Code

There is a transposition Gray code for indecomposable permutations which uses the partitioning of the set of indecomposable permutations induced jointly by the parameters r and j, discussed in Section 2. We must introduce several terms.

Terminology 1. *We shall denote the transposition Gray graph of indecomposable permutations G_n. Let the subgraph of G_n induced by those permutations which begin with r be denoted $G_{n,r}$. Let the subgraph of $G_{n,r}$ induced by those permutations with parameter j (as in Section 2) be denoted $G_{n,r,j}$. Let the vertex sets of these graphs be denoted I_n, $I_{n,r}$, and $I_{n,r,j}$ respectively.*

Terminology 2. *Consider two finite sets S_1, $S_2 \subseteq \mathbb{Z}^+$ such that $|S_1| = |S_2| = n > 0$. Now consider two bijections (permutations), $\pi_1 : S_1 \to [n]$ and $\pi_2 : S_2 \to [n]$. We say that π_1 and π_2 are* equivalent *permutations if and only if for any i and j such that $0 < i, j \leq n$, we have that $\pi_1^{-1}(i) < \pi_1^{-1}(j) \iff \pi_2^{-1}(i) < \pi_2^{-1}(j)$. When an underlying set S is implied or known, and π is a permutation on a set of size $|S|$, let $E(\pi)$ denote the permutation on S which is equivalent to π.*

Lemma 5. *Let j satisfy $0 \leq j \leq n-2$. Then for any two r_1 and r_2 satisfying $j+2 \leq r \leq n$, $G_{n,r_1,j} \cong G_{n,r_2,j}$.*

Proof. Consider the bijection $f : I_{n,r_1,j} \to I_{n,r_2,j}$ which maps $\pi_1 \in I_{n,r_1,j}$ to $\pi_2 \in I_{n,r_2,j}$ if and only if when the first elements of each permutation (i.e. the prefixes of length 1) are removed, the remaining permutations are equivalent. The image of π_1 is unique, as there can only be one permutation on $[n] \setminus r_2$ equivalent to a given permutation on $[n] \setminus r_1$.

Since r_1 and r_2 are both greater than $j+1$, it is easy to see that $f(\pi_1) \in I_{n,r_2,j}$. Now consider a permutation $\pi_1' \in I_{n,r_1,j}$ which is reached from π_1 by a single transposition. $f(\pi_1')$ is reached from π_2 by transposing the same two positions. By generality, the same rule applies in the other direction, so f is a graph isomorphism. □

At this point, we must define special vertices in the Gray graph.

Terminology 3. *Let the* top *vertices of $G_{n,r,j}$ and G_n be denoted* $\text{top}_{n,r,j}$ *and* top_n *respectively. Let the* bottom *vertices of $G_{n,r,j}$ and G_n be denoted*

$\mathrm{bot}_{n,r,j}$ *and* bot_n *respectively. Let them be defined as follows:*

$$\begin{aligned}
\mathrm{top}_n &= 2, 3, 4, \ldots, n, 1 \\
\mathrm{bot}_n &= n, 1, 2, 3, \ldots, n-1 \\
\mathrm{top}_{n,n,j} &= \begin{cases} n, 2, 3, \ldots, n-1, 1 & \text{if } j = 0 \\ n, 1, 3, 4, \ldots, n-1, 2 & \text{if } j = 1 \\ n, 1, 2, \ldots, j-2, j, j-1, n-1, j+1, j+2, \ldots, n-2 & \text{otherwise} \end{cases} \\
\mathrm{bot}_{n,n,j} &= n, 1, 2, \ldots, j-2, j-1, j, n-1, j+1, j+2, \ldots, n-2
\end{aligned}$$

For $r \neq n$, $\mathrm{top}_{n,r,j}$ *and* $\mathrm{bot}_{n,r,j}$ *are those vertices in* $G_{n,r,j}$ *which are isomorphic to* $\mathrm{top}_{n,n,j}$ *and* $\mathrm{bot}_{n,n,j}$ *respectively, under the isomorphism described in the proof of Lemma 5.*

The following facts are to be noted:

- $\mathrm{top}_n = \mathrm{top}_{n,2,0} = 2, E(\mathrm{top}_{n-1})$.
- $\mathrm{bot}_n = \mathrm{bot}_{n,n,n-2}$.
- For $j \geq 2$, $\mathrm{top}_{n,n,j} = n, 1, 2, \ldots, j-2, j, j-1, E(\mathrm{bot}_{n-j-1})$.
- $\mathrm{bot}_{n,n,j} = n, 1, 2, \ldots, j-2, j-1, j, E(\mathrm{bot}_{n-j-1})$.
- Transposing positions $j+2$ and $j+4$ in $\mathrm{bot}_{n,n,j}$ results in $\mathrm{top}_{n,n,j+2}$.
- Transposing positions 2 and n in $\mathrm{top}_{n,n,0}$ results in $\mathrm{top}_{n,n,1}$.
- Because of isomorphism, the above apply to all values of r, not just n.
- For $2 \leq r < n$, transposing elements (not positions) r and $r+1$ in $\mathrm{bot}_{n,r,j}$ results in $\mathrm{bot}_{n,r+1,j}$.

Let P_n be the transposition Gray graph of all permutations on $[n]$. Note, then, that $G_{n,r,j} \cong G_{n-j-1} \times P_j$. To see this, consider the explanation of Theorem 1.

Lemma 6. *To show that there is a Gray code for* I_n*, it suffices to show that for* $0 \leq j \leq n-2$*, there is a Gray code for* $I_{n,n,j}$ *that begins at* $\mathrm{top}_{n,n,j}$ *and ends at* $\mathrm{bot}_{n,n,j}$.

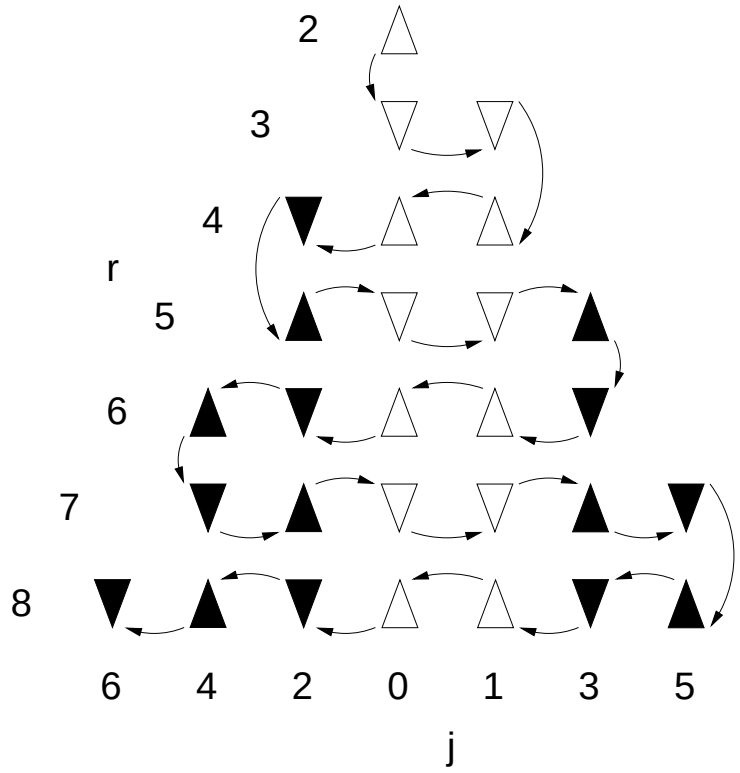

Figure 1: The traversals of $G_{n,r,j}$ graphs, combined to traverse G_n. Empty triangles indicate traversal from top to bottom, or bottom to top. Filled triangles indicate traversal from middle to bottom, or bottom to middle.

Proof. Suppose there are such Gray codes. Then by isomorphism, there is a top-to-bottom Gray code for $I_{n,2,0}$. We can traverse it, then make a transposition to reach $\text{bot}_{n,3,0}$. We can then traverse the previous Gray code in reverse to reach $\text{top}_{n,3,0}$ and make a transposition to reach $\text{top}_{n,3,1}$. Since there is a Gray code for $I_{n,n,1}$, we can traverse $G_{n,3,1}$ in Gray code order, then make a transposition to jump from $\text{bot}_{n,3,1}$ to $\text{bot}_{n,4,1}$.

Again, we can traverse from $\text{bot}_{n,4,1}$ to $\text{bot}_{n,4,2}$ by making the transpositions we used to traverse $r = 3$, only in reverse order. We can then jump from $\text{bot}_{n,4,0}$ to $\text{top}_{n,4,2}$. At this point, we can traverse $G_{n,4,2}$ from top to bottom, since there is a Gray code isomorphic to that for $I_{n,n,2}$.

In this fashion, we can continue jumping between values of r and j in the order shown in Figure 1 until we have traversed I_n completely. □

Theorem 7. *There is a transposition Gray code for I_n.*

Proof. We will use strong induction to prove the theorem.

BASIS: $n = 1$. $|I_1| = 1$, so there is a trivial Gray code from top to bottom.

INDUCTION: Assume that for all $0 < i < n$, there is a Gray code for I_i which runs from top_i to bot_i.

$\text{top}_{n,n,0} = n, E(\text{top}_{n-1})$. $\text{bot}_{n,n,0} = n, E(\text{bot}_{n-1})$. So by traversing the Gray code for I_{n-1} in the last $n-1$ positions of the permutation, we can traverse $G_{n,n,0}$ from top to bottom in Gray code order. Similarly, $\text{top}_{n,n,1} = n, 1, E(\text{top}_{n-2})$ and $\text{bot}_{n,n,1} = n, 1, E(\text{bot}_{n-2})$, so we can traverse $G_{n,n,1}$ from top to bottom in Gray code order using the Gray code for I_{n-2}.

We must now address the case where $j \leq 2$, i.e. the top to bottom traversals, using the fact that $G_{n,r,j} \cong G_{n-j-1} \times P_j$. First note that we have a transposition Gray code for P_j whose final permutation is the same as its initial permutation, but with the final two positions transposed: left-to-right SJT. This is the same as the Steinhaus-Johnson-Trotter algorithm, but the element in the leftmost position is moved first, not the element in the rightmost position. In order to traverse $G_{n,n,j}$, we must traverse G_{n-j-1} in the final $n-j-1$ positions once for every permutation of length j.

Recall that $\text{top}_{n,n,j} = n, 1, 2, \ldots, j-2, j, j-1, E(\text{bot}_{n-j-1})$, and $\text{bot}_{n,n,j} = n, 1, 2, \ldots, j-2, j-1, j, E(\text{bot}_{n-j-1})$, and note that we have an even number ($j!$) of permutations of length j. Therefore we can simply traverse G_{n-j-1} from bottom to top, advance the permutation of length j, traverse G_{n-j-1} from top to bottom, advance the permutation, etc., until every permutation

in $I_{n,n,j}$ has been exhausted. Because $j!$ is even and we are using left-to-right SJT, by beginning at $\text{mid}_{n,n,j}$ we ensure that we will end the traversal of $G_{n,n,j}$ at $\text{bot}_{n,n,j}$.

We have now established the necessary conditions given Lemma 6, so there is a Gray code for I_n. □

4.2 Generating the Gray Code

In this section we present a recursive algorithm for generating I_n in Gray code order. The initial call is PrintIt; Indec$(n, 0)$. RevIndec$(n, 0)$ generates I_n in reverse order (from bottom to top). To generate I_n with Indec, we must initialize the permutation p to top_n. Swap simply transposes the two positions, given as parameters, in p.

Indec and RevIndec call Permute, which in turn calls Indec and RevIndec. Every time Permute$(n, j, depth, m,$ true$)$ is called, we must allocate three vectors of length j, as in Algorithm 1: π, π^{-1}, and $dirArr$. To perform the left-to-right SJT traversal of permutations, we initialize π and π^{-1} to $j, j-1, j-2, \ldots, 1$, and we initialize $dirArr[i]$ to 1 for all i.

Indec jumps between the $I_{n,r,j}$ graphs in the order shown in Figure 1, traversing each subgraph by calling Permute. The algorithm uses the subgraph isomorphism described earlier in this section. NextJ and its inverse RevNextJ select the next value of j by simple case analysis. The values of j are changed in Indec, lines 18–24, and in RevIndec, lines 14–20, choosing the positions to transpose by the difference in j. The values of r are changed in Indec, lines 27–31, and in RevIndec, lines 23–27, this time choosing between two transpositions, depending on the structure of the current permutation. We will now prove that this algorithm is CAT, but first we will prove two technical lemmas.

Lemma 8. *Let $W_I(n)$ be the number of times a single call to Indec$(n, 0)$ results in a call to Indec$(i, 0)$ for $i \leq 2$. $W_I(n) = |I_n|$.*

Proof. $W_I(1) = 1$. For $n > 1$,

$$W_I(n) = \sum_{r=2}^{n} \sum_{j=0}^{r-2} W_I(n-j-1)j!. \tag{7}$$

This is because for a given value of j, Indec calls Indec$(n-j-1, depth)$ $j!$ times, and each of these calls contains $W_i(n-j-1)$ calls to Indec$(i, 0)$ for

Algorithm 2 Traverse G_n in Gray code order

```
 1: procedure Indec ( int n, depth )
 2: local int r, j, j'; boolean dir
 3: begin
 4:   if n > 2 then
 5:     for r := 2 to n do
 6:       if r > 2 then
 7:         j := r − 3;
 8:       else
 9:         j := 0;
10:       end if
11:       while true do
12:         dir := (r % 2 = j);
13:         Permute(n, j, depth, 1, dir, true);
14:         j' := NextJ(r, j);
15:         if j' > r − 2 then
16:           break;
17:         end if
18:         if j' = j + 2 then
19:           Swap(depth + j + 2, depth + j + 4); PrintIt;
20:         else if j' = j − 2 then
21:           Swap(depth + j, depth + j + 2); PrintIt;
22:         else
23:           Swap(depth + 2, depth + n); PrintIt;
24:         end if
25:         j := j';
26:       end while
27:       if r < n − 1 then
28:         Swap(depth + 1, depth + r + 2); PrintIt;
29:       else if r = n − 1 then
30:         Swap(depth + 1, depth + r); PrintIt;
31:       end if
32:     end for
33:   end if
34: end
```

Algorithm 3 Traverse G_n in reverse Gray code order

```
 1: procedure RevIndec ( int n, depth )
 2: local int r, j, j'; boolean dir
 3: begin
 4:   if n > 2 then
 5:     for r := n down to 2 do
 6:       j := r − 2;
 7:       while true do
 8:         dir := (r % 2 ≠ j);
 9:         Permute(n, j, depth, 1, dir, true);
10:         j' := RevNextJ(r, j);
11:         if j' > r − 2 then
12:           break;
13:         end if
14:         if j' = j + 2 then
15:           Swap(depth + j + 2, depth + j + 4); PrintIt;
16:         else if j' = j − 2 then
17:           Swap(depth + j, depth + j + 2); PrintIt;
18:         else
19:           Swap(depth + 2, depth + n); PrintIt;
20:         end if
21:         j := j';
22:       end while
23:       if r = n then
24:         Swap(depth + 1, depth + r − 1); PrintIt;
25:       else if r > 2 then
26:         Swap(depth + 1, depth + r + 1); PrintIt;
27:       end if
28:     end for
29:   end if
30: end
```

Algorithm 4 Traverse $G_{n,r,j}$

```
1:  procedure Permute ( int n, j, depth, m; boolean first )
2:  local int i, t
3:  begin
4:    if m > j then
5:      if ¬first then
6:        Printit;
7:      end if
8:      if dir then
9:        Indec(n − j − 1, depth + j + 1)
10:     else
11:       RevIndec(n − j − 1, depth + j + 1)
12:     end if
13:     dir := ¬dir;
14:   else
15:     Permute(n, j, depth, m + 1, first);
16:     first := false;
17:     for i := 1 to m − 1 do
18:       t := π^{-1}[m];   π[t] := π[t + dirArr[m]];   π[t + dirArr[m]] := n;
19:       π^{-1}[m] := t + dirArr[m];   π^{-1}[π[t]] := t;
20:       Permute(n, j, depth, m + 1, false);
21:     end for
22:   end if
23:   dirArr[m] := −dirArr[m];
24: end
```

Algorithm 5 Determine the next value of j to traverse

```
1:  procedure NextJ ( int r, j )
2:  begin
3:    if j = 0 and r is even then
4:      return j + 1;
5:    else if j = 1 and r is odd then
6:      return j − 1;
7:    else if j % 2 = r % 2 then
8:      return j + 2;
9:    else
10:     return j − 2;
11:   end if
12: end
13:
14: procedure RevNextJ ( int r, j )
15: begin
16:   if j = 0 and r is odd then
17:     return j + 1;
18:   else if j = 1 and r is even then
19:     return j − 1;
20:   else if j % 2 = r % 2 then
21:     return j − 2;
22:   else
23:     return j + 2;
24:   end if
25: end
```

$i \leq 2$. Recall that this is the same recurrence as in Equation 3, and since $W_I(1) = 1$, $W_I(n) = |I_n|$. □

Lemma 9. *Let $W_P(n)$ be the number of times a single call to* Indec$(n, 0)$ *results in a call to* Permute(n, j) *for $j \leq 1$. $W_P(n) \leq n!$.*

Proof. $W_I(1) = 0$ and $W_I(2) = 0$. For $n > 2$, Indec$(n, 0)$ calls Permute$(n, 0)$ $n - 1$ times, Permute$(n, 1)$ $n - 2$ times. These are the only such calls that happen in the calls at the top level. Beyond that, we have the familiar recurrence for calls to Permute(n, j) for $j \leq 1$ by recursive calls. This gives us

$$W_P(n) = 2n - 3 + \sum_{j=0}^{n-2} (n - j - 1) \cdot W_P(n - j - 1) \cdot j! \tag{8}$$

$$= 2n - 3 + \sum_{j=0}^{n-4} (n - j - 1) \cdot W_P(n - j - 1) \cdot j!, \tag{9}$$

the second formulation coming from the knowledge that $W_I(i) = 0$ for $i < 2$. The sequence $\{W_P(n)\}_{n=1}^{\infty}$ begins $0, 0, 3, 14, 72, 443, \ldots$. We claim that $W_P(n) \leq |I_n|$ for $n \geq 6$, and will prove it by induction.

Basis: $|I_6| = 461$, $W_P(6) = 443$.

INDUCTION: Let $n \geq 7$. Assume that all $6 \leq i \leq n-1$, $W_P(i) < |I_i|$.

$$\begin{aligned}
W_P(n) &= 2n - 3 + \sum_{j=0}^{n-7} ((n-j-1) \cdot W_P(n-j-1) \cdot j!) \\
&\quad + \sum_{j=n-6}^{n-2} ((n-j-1) \cdot W_P(n-j-1) \cdot j!) && (10) \\
&\leq 2n - 3 + \sum_{j=0}^{n-7} ((n-j-1) \cdot |I_{n-j-1}| \cdot j!) \\
&\quad + \sum_{j=n-6}^{n-2} ((n-j-1) \cdot W_P(n-j-1) \cdot j!) && (11) \\
&= 2n - 3 + |I_n| - \sum_{j=n-6}^{n-2} ((n-j-1) \cdot |I_{n-j-1}| \cdot j!) \\
&\quad + \sum_{j=n-6}^{n-2} ((n-j-1) \cdot W_P(n-j-1) \cdot j!) && (12) \\
&= |I_n| + (2n + 5(n-6)! + 4(n-5)!) \\
&\quad - (3 + 2(n-3)! + (n-2)!) && (13) \\
&\leq |I_n| + 2n - 43(n-6)! - 60(n-5)! && (14) \\
&\leq |I_n| && (15)
\end{aligned}$$

These steps are reasonably straightforward. (13) recalls Equation 4, and the steps that follow are arithmetical facts which rely on n being greater than 6. Since $|I_n| \leq n!$, the lemma is proved. □

These bounds on the call counts help us prove that the algorithm is efficient (CAT).

Theorem 10. *Indec*$(n, 0)$ *generates* I_n *in constant amortized time.*

Proof. Each call counted by $W_I(n)$ and $W_P(n)$ runs in constant time and outputs no permutations (though Permute will still call Indec once), so we can omit them from our analysis, since both $W_I(n)$ and $W_P(n)$ are bounded by $2|I_n|$; the total time is constant per indecomposable permutation generated by the main call, so it suffices to show that the rest of the algorithm is CAT.

Not counting the time taken during the recursive calls to Permute, each call to Indec$(n, depth)$ outputs $(n^2 - n - 2)/2$ permutations (one for each jump between a $G_{n,r,j}$ subgraph) and makes $(n^2 - n)/2$ calls to Permute (one for each $G_{n,r,j}$) in $O(n^2)$ time. Since we can assume $n \geq 3$, $(n^2 - n - 2)/2$ and $(n^2 - n)/2$ are each greater than $n^2/5$. Therefore Indec$(n, depth)$ itself does a constant amount of work per output and a constant amount of work per call to Permute.

Not counting the time taken during the recursive calls, each call to Permute(n, j) outputs $j! - 1$ permutations and makes $j!$ calls to Indec in $O(j!)$ time, which we can see because the SJT algorithm is CAT, and we have added only a constant amount of work per node. Because we can assume that $j \geq 2$, we know that $j! - 1 \geq j!/2$. Therefore we know that Permute(n, j) itself does a constant amount of work per output and a constant amount of work per recursive call made.

We have now shown that each of Indec and Permute does a constant amount of work per permutation output (recall that NextJ runs in constant time), so the entire algorithm is CAT. □

5 Conclusions

We have given two CAT algorithms for generating indecomposable permutations: one generates them by selecting them efficiently in the Steinhaus-Johnson-Trotter algorithm, and one generates them in transposition Gray code order. The Gray code was developed by using a new parameterization of indecomposable permutations. The question of whether or not there is an adjacent transposition Gray code for these permutations remains open.

6 Acknowledgements

I would like to thank Frank Ruskey for presenting the problem to me and helping me along the way, and Joe Sawada for his helpful comments on CAT algorithms.

References

[1] Comtet, Louis. *Sur les coefficients de l'inverse de la série formelle* $\sum n!t^n$. Comptes Rend. Acad. Sci. Paris, A 275, pp 569-572, 1972.

[2] Comtet, Louis. *Advanced Combinatorics*. Dordrecht, Holland: D. Reidel Publ. Co., 1974.

[3] Hertel, Alex and Philip. The stonecarver's Hamilton cycle algorithm. Private communication.

[4] King, Andrew D. *Transposition Gray codes for indecomposable permutations*, B.Sc. honours thesis, University of Victoria, Canada, 2002.

[5] Ruskey, Frank. *Combinatorial Generation, Working Version (1j)*. Victoria, Canada: University of Victoria, 2001.

[6] Sloane, N. J. A. The On-Line Encyclopedia of Integer Sequences. `http://www.research.att.com/~njas/sequences`.

[7] Wilf, Herbert S. *Generatingfunctionology*. San Diego: Academic Press, Inc., 1990.