副作用をもつ演算

元のコード	順方向	逆方向	破壊的代入になる場合
++(前置)	++(前置)	--(後置)	桁あふれ
++(後置)	++(後置)	--(前置)	桁あふれ
--(前置)	--(前置)	++(後置)	桁あふれ
--(後置)	--(後置)	++(前置)	桁あふれ
+=	+=	-=	左辺と同じ変数が右辺に出現 桁あふれ
-=	-=	+=	左辺と同じ変数が右辺に出現 桁あふれ
*=	*=	/=	左辺と同じ変数が右辺に出現 桁あふれ 右辺が0
/=	/=	*=	左辺と同じ変数が右辺に出現 余りが出る 右辺が0
^=	^=	^=	左辺と同じ変数が右辺に出現
x=e; (破壊的代入の例)	SAVE(x); x=e;	RESTORE(x);	

関数呼出し

元のコード	順方向	逆方向
f()	f()	rev_f()

	元のコード	順方向	逆方向
旧	v = f();	SAVE(v); v = f();	RESTORE(v);
新	v = f();	SAVE(v); v = f();	rev_f(); RESTORE(v);

goto文

	元のコード	順方向	逆方向

旧	{
	  s1
	  goto label;
	  s2
	  goto label;
	  s3
	label:
	  s4
	}
		{
		  char c;
		  T[s1]
		  {c=1; SAVE(c);}
		  goto label;
		  T[s2]
		  {c=2; SAVE(c);}
		  goto label;
		  T[s3]
		  {c=0; SAVE(c);}
		label:
		  T[s4]
		}
			{
			  char c;
			  R[s4]
			  RESTORE(c);
			  switch(c)
			  {
			  case 0: break;
			  case 1: goto rlabel1;
			  case 2: goto rlabel2;
			  }
			  R[s3]
			rlabel2:
			  R[s2]
			rlabel1:
			  R[s1]
			}

新	{
	  s1
	  goto label;
	  s2
	  goto label;
	  s3
	label:
	  s4
	}
		{
		  unsigned char c;
		  T[s1]
		  {c=1; SAVE(c);}
		  goto label;
		  T[s2]
		  {c=2; SAVE(c);}
		  goto label;
		  T[s3]
		  {c=0; SAVE(c);}
		label:
		  T[s4]
		}
		（同一のラベルに対するgoto文は255個以下）
			{
			  unsigned char c;
			  R[s4]
			  RESTORE(c);
			  switch(c)
			  {
			  case 0: break;
			  case 1: goto rlabel1;
			  case 2: goto rlabel2;
			  }
			  R[s3]
			rlabel2:
			  R[s2]
			rlabel1:
			  R[s1]
			}

複合文

元のコード	順方向	逆方向

{
  int x,y;
  s
}
	{
	  int x,y;
	  T[s]
	  SAVE(x);
	  SAVE(y);
	}
		{
		  int x,y;
		  RESTORE(y);
		  RESTORE(x);
		  R[s]
		}

	元のコード	順方向	逆方向

旧	int f(int arg)
	{
	  int x, y;
	  s
	  return(x);
	}
		int f(int arg)
		{
		  int x, y;
		  T[s]
		  return(x);
		  SAVE(arg);
		  SAVE(x);
		  SAVE(y);
		}
			int rev_f(int arg)
			{
			  int x, y;
			  RESTORE(y);
			  RESTORE(x);
			  RESTORE(arg);
			  R[s]
			  return(x);
			}

新	int f(int arg)
	{
	  int x, y;
	  s
	  return(x);
	}
		int f(int arg)
		{
		  int x, y;
		  T[s]
		  SAVE(arg);
		  SAVE(x);
		  SAVE(y);
		  return(x);
		}
			int rev_f(int arg)
			{
			  int x, y;
			  RESTORE(y);
			  RESTORE(x);
			  RESTORE(arg);
			  R[s]
			  return(x);
			}

if文

元のコード	順方向	逆方向

if(test)
  s1
else
  s2
	{
	  char c=!!test;
	  if(c)
	  T[s1]
	  else
	  T[s2]
	  SAVE(c);
	}
		{
		  char c;
		  RESTORE(c);
		  if(c)
		  R[s1]
		  else
		  R[s2]
		}

元のコード	順方向	逆方向

if(test)
  s1
else
  s2
	if(test)
	  T[s1]
	else
	  T[s2]
	（条件式が実行前後で不変）
		if(test)
		  R[s1]
		else
		  R[s2]

	元のコード	順方向	逆方向

旧	if(test1)
	  s1
	else if(test2)
	  s2
	else if(test3)
	  s3
		{
		  char c1=!!test1;
		  char c2=!!test2;
		  char c3=!!test3;
		  if(c1)
		  T[s1]
		  else if(c2)
		  T[s2]
		  else if(c3)
		  T[s3]
		  SAVE(c1);
		  SAVE(c2);
		  SAVE(c3);
		}
			{
			  char c1;
			  char c2;
			  char c3;
			  RESTORE(c3);
			  RESTORE(c2);
			  RESTORE(c1);
			  if(c1)
			  R[s1]
			  else if(c2)
			  R[s2]
			  else if(c3)
			  R[s3]
			}

新	if(test1)
	  s1
	else if(test2)
	  s2
	else if(test3)
	  s3
		{
		  unsigned char c;
		  if(test1)
		  {
		  T[s1]
		  c = 0;
		  }
		  else if(test2)
		  {
		  T[s2]
		  c = 1;
		  }
		  else if(test3)
		  {
		  T[s3]
		  c = 2;
		  }
		  SAVE(c);
		}
		（cはfresh）
			{
			  unsigned char c;
			  RESTORE(c);
			  if(c==0)
			  R[s1]
			  else if(c==1)
			  R[s2]
			  else if(c==2)
			  R[s3]
			}

switch文

	元のコード	順方向	逆方向

旧	switch(e)
	{
	  case c1:
	  s1
	  case c2:
	  s2
	  break;
	  default:
	  s3
		{
		  char c=0, d=0;
		  switch(e)
		  {
		  case c1: c=1; d++;
		  T[s1]
		  case c2: c=2; d++;
		  T[s2]
		  break;
			{
			  char c, d;
			  RESTORE(c);
			  RESTORE(d);
			  switch(c)
			  {
			  case 0:
			  R[s3]
			  if(--d <= 0) break;

	break;
	}
		default: c=0; d++;
		T[s3]
		break;
		}
		SAVE(c);
		SAVE(d);
		}
			case 2:
			R[s2]
			if(--d <= 0) break;
			case 1:
			R[s1]
			}
			}

新	switch(e)
	{
	case c1:
	s1
	case c2:
	s2
	break;
	default:
	s3
	break;
	}
		{
		unsigned char c=0, d=0;
		switch(e)
		{
		case c1: c=1; d++;
		T[s1]
		case c2: c=2; d++;
		T[s2]
		break;
		default: c=0; d++;
		T[s3]
		break;
		}
		SAVE(c);
		SAVE(d);
		}
		（case文は255個以下）
			{
			unsigned char c, d;
			RESTORE(c);
			RESTORE(d);
			switch(c)
			{
			case 0:
			R[s3]
			if(--d <= 0) break;
			case 2:
			R[s2]
			if(--d <= 0) break;
			case 1:
			R[s1]
			}
			}

while文

	元のコード	順方向	逆方向

旧	while(test)
	s
		{
		int c=0;
		while(test)
		{
		c++;
		T[s]
		}
		SAVE(c);
		}
			{
			int c;
			RESTORE(c)
			while(c)
			{
			R[s]
			--c;
			}
			}

新	while(test)
	s
		{
		unsigned int c=0;
		while(test)
		{
		c++;
		T[s]
		}
			{
			unsigned int c;
			RESTORE(c)
			while(c)
			{
			R[s]
			--c;

		SAVE(c); } （cはfresh） （繰り返しは2^32 −1以下）	} } （cはfresh）

	元のコード	順方向	逆方向

旧	{
	int i;
	i=0;
	s1
	while(i<n)
	{
	s2
	i++;
	}
	s3
	}
		{
		int i;
		int c=0;
		i=0;
		T[s1]
		while(i<n)
		{
		c++;
		i++;
		T[s2]
		}
		SAVE(c);
		T[s3]
		SAVE(i);
		}
			{
			int i;
			int c;
			RESTORE(i)
			RESTORE(c)
			R[s3]
			while(c)
			{
			--i;
			R[s2]
			--c;
			}
			R[s1]
			}

新	{
	int i;
	i=0;
	s1
	while(i<n)
	{
	s2
	i++;
	}
	s3
	}
		{
		int i;
		i=0;
		T[s1]
		while(i<n)
		{
		i++;
		T[s2]
		}
		T[s3]
		SAVE(i);
		}
		（s1, s2でiが変更されない）
		（繰り返しは2^31−1以下）
			{
			int i;
			RESTORE(i)
			R[s3]
			while(i)
			{
			R[s2]
			--i;
			}
			R[s1]
			}

再帰関数がn回再帰する場合，記憶する情報はlog(n)で済む.
switchの最適化