Valgrind e Simulação de Cache: **Cachegrind**

Este laboratório apresenta o uso da ferramenta cachegrind do ambiente Valgrind, para maiores informações consulte aqui

Importante:

- A primeira execução do Cachegrind irá fazer a instalação da ferramenta e pode demorar um pouco mais.
- Os laboratorios usam uma multiplicação de matrizes como exemplo. O tamanho da matriz cresce com $O(N^2)$ e o tempo de execução com $O(N^3)$.
- Os exemplos estão em C. Mas o Cachegrind trabalha sobre o executável e pode ser usado em qualquer binário.
- Fique a vontade para contribuir.

Inicialização

Primeiro, configurar o laboratório.

```
!pip install git+https://github.com/lesc-ufv/cad4u >& /dev/null
!git clone https://github.com/lesc-ufv/cad4u >& /dev/null
%load_ext plugin
```

```
%%datacache
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char const *argv[]) {

  int n = 100;
  char a[n][n], b[n][n], c[n][n];

  int s = 0;
  for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
        a[i][j] = i + j;
        b[i][j] = i*2 + j;
      }
  }

  int temp;
  for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
          temp = 0;
```

```
            for (int k = 0; k < n; ++k) {
                temp += a[i][k] * b[k][j];
            }
            c[i][j] = temp;
        }
    }

    return 0;
}
```

```
Installing. Please wait... done!
```

Data...

Size ... 1

Asso... 1

Line... 32

Star...

Specify all cache parameters

A extensão **%%cachegrind** é semelhante a linha de comando, importante que os tamanhos de cache devem ser potência de 2, a linha além de potência de 2 começa com 32 bytes. A ordem dos parametros é tamanho da cache, associatividade e tamanho da linha. Os flags para cache de dados, de instruções e de último nível são **D1**, **I1**, and **LL**, respectivamente.

```
%%cachegrind --D1=32768,8,32 --I1=32768,2,32 --LL=65536,2,32 --file
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char const *argv[]) {

    int n = 100;
    int a[n][n], b[n][n], c[n][n];

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            a[i][j] = i + j;
            b[i][j] = i*2 + j;
        }
    }

    int temp;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            temp = 0;
            for (int k = 0; k < n; ++k) {
                temp += a[i][k] * b[k][j];
```

```
            temp += a[i][k] * b[k][j];
        }
        c[i][j] = temp;
      }
  }
  return 0;
}
```

```
    LLi miss rate:       0.01%

    D   refs:      12,324,469  (12,261,210 rd   + 63,259 wr)
    D1  misses:       133,435  (   128,603 rd   +  4,832 wr)
    LLd misses:         9,388  (     4,585 rd   +  4,803 wr)
    D1  miss rate:        1.1% (       1.0%     +    7.6%  )
    LLd miss rate:        0.1% (       0.0%     +    7.6%  )

    LL refs:          135,484  (   130,652 rd   +  4,832 wr)
    LL misses:         11,417  (     6,614 rd   +  4,803 wr)
    LL miss rate:         0.0% (       0.0%     +    7.6%  )
```

Atenção aos resultados

Valgrind quando simula a cache ele também simula a inicialização do sistema. Portanto, quando for utilizar o valgrind esteja ciente que se o seu código for muito simples será mascarado pela inicialização do sistema.

```
%%cachegrind --D1=1024,8,32 --I1=32768,2,32 --LL=65536,2,32 --file

int main(int argc, char const *argv[]) {
  //# empty code
}
```

```
    LLi miss rate:    1.11%

    D   refs:       52,820  (39,891 rd   + 12,929 wr)
    D1  misses:     14,079  (11,428 rd   +  2,651 wr)
    LLd misses:      3,511  ( 2,456 rd   +  1,055 wr)
    D1  miss rate:    26.7% (  28.6%     +   20.5%  )
    LLd miss rate:     6.6% (   6.2%     +    8.2%  )

    LL refs:        16,063  (13,412 rd   +  2,651 wr)
    LL misses:       5,475  ( 4,420 rd   +  1,055 wr)
    LL miss rate:      2.4% (   2.0%     +    8.2%  )
```

Exemplo de código mascarado

Abaixo é apresentado um código de **transposição de matrizes** sendo mascarado pela inicialização do sistema.

```
%%cachegrind --D1=1024,8,32 --I1=32768,2,32 --LL=65536,2,32 --file
#include <stdio.h>
#include <stdlib.h>

#define n 32
int A[n][n], B[n][n];

void trans(int M, int N) {
    int i, j, tmp;
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++) {
            tmp = A[i][j];
            B[j][i] = tmp;
        }
}

int main(int argc, char const *argv[]) {
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      A[i][j] = i + j;

  trans(n, n); //# transposição de matrizes
  return 0;
}
```

```
    LLi miss rate:    0.91%

    D   refs:       71,583  (55,512 rd   + 16,071 wr)
    D1  misses:     15,369  (11,561 rd   +  3,808 wr)
    LLd misses:      3,774  ( 2,463 rd   +  1,311 wr)
    D1  miss rate:    21.5% (  20.8%     +   23.7%  )
    LLd miss rate:     5.3% (   4.4%     +    8.2%  )

    LL refs:        17,360  (13,552 rd   +  3,808 wr)
    LL misses:       5,745  ( 4,434 rd   +  1,311 wr)
    LL miss rate:      2.0% (   1.6%     +    8.2%  )
```

Resultados **somente inicialização X transposição de matrizes**

Somente Inicialização:

- D refs: 1,203,059 (771,626 rd + 431,433 wr)
- D1 misses: 284,860 (219,936 rd + 64,924 wr)

Transposição de matrizes:

- D refs: 1,221,822 (787,247 rd + 434,575 wr)
- D1 misses: 286,147 (220,069 rd + 66,078 wr)

Note que a diferença é pequena, sendo para a cache dados L1:

- D refs: 1221822 - 1203059 = 18763

- D1 refs: 1221822 - 1208659 = 13163
- D1 misses: 286147 - 284860 = 1287

Solução: Mais trabalho para o algoritmo

Uma solução é fazer com que o seu código der mais trabalho para cache de dados, assim a inicialização não irá mascarar os resultados.

Tranposição simples

```
%%cachegrind --D1=1024,8,32 --I1=32768,2,32 --LL=65536,2,32 --file
#include <stdio.h>
#include <stdlib.h>

#define n 32
int A[n][n], B[n][n];

void trans(int M, int N) {
    int i, j, tmp;
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++) {
            tmp = A[i][j];
            B[j][i] = tmp;
        }
}

int main(int argc, char const *argv[]) {
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      A[i][j] = i + j;

  for (int i; i < 2000; ++i)
    trans(n, n); //# transposição de matrizes
  return 0;
}
```

```
    LLi miss rate:       0.00%

    D   refs:      22,994,119  (18,810,133 rd   + 4,183,986 wr)
    D1  misses:     2,320,216  (   269,433 rd   + 2,050,783 wr)
    LLd misses:         3,774  (     2,463 rd   +     1,311 wr)
    D1  miss rate:       10.1% (       1.4%     +      49.0%  )
    LLd miss rate:        0.0% (       0.0%     +       0.0%  )

    LL refs:        2,322,208  (   271,425 rd   + 2,050,783 wr)
    LL misses:          5,746  (     4,435 rd   +     1,311 wr)
    LL miss rate:         0.0% (       0.0%     +       0.0%  )
```

```
%%cachegrind --D1=1024,8,32 --I1=32768,2,32 --LL=65536,2,32 --file
```

```
#include <stdio.h>
#include <stdlib.h>

#define n 32
int A[n][n], B[n][n];

void transpose_32_32(int M, int N) {
    int BLOCK_SIZE, rowIndex, colIndex, blockedRowIndex, blockedColIndex, eBloc
    BLOCK_SIZE = 8;
    for (colIndex = 0; colIndex < M; colIndex += BLOCK_SIZE) {
        for (rowIndex = 0; rowIndex < N; rowIndex += BLOCK_SIZE) {
            for (blockedRowIndex = rowIndex; blockedRowIndex < rowIndex + BLOCK
                for (blockedColIndex = colIndex; blockedColIndex < colIndex + B
                    if (blockedRowIndex != blockedColIndex)
                        B[blockedColIndex][blockedRowIndex] = A[blockedRowIndex
                    else {
                        eBlockDiagl = A[blockedRowIndex][blockedColIndex];
                        iBlockDiagl = blockedRowIndex;
                    }
                }
                if (colIndex == rowIndex)
                    B[iBlockDiagl][iBlockDiagl] = eBlockDiagl;
            }
        }
    }
}

int main(int argc, char const *argv[]) {
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      A[i][j] = i + j;
  for (int i; i < 2000; ++i)
    transpose_32_32(n, n); //# transposição de matrizes 32x32
  return 0;
}
```

```
    LLi miss rate:       0.00%

    D   refs:      28,012,119  (25,514,133 rd   + 2,497,986 wr)
    D1  misses:     1,572,216  (   269,433 rd   + 1,302,783 wr)
    LLd misses:         3,774  (     2,463 rd   +     1,311 wr)
    D1  miss rate:        5.6% (       1.1%     +      52.2%  )
    LLd miss rate:        0.0% (       0.0%     +       0.1%  )

    LL refs:        1,574,213  (   271,430 rd   + 1,302,783 wr)
    LL misses:          5,751  (     4,440 rd   +     1,311 wr)
    LL miss rate:         0.0% (       0.0%     +       0.1%  )
```

Processando os dados: **Transposição simples X Transposição 32x32**

Note que a cache de dados teve maiores valores na transposição 32x32:

- Transposição simples: 24.144.358

- Transposição simples: 24,144,000
- Transposição 32x32: 29,162,358

Contudo as falhas na transposição 32x32 foram menores:

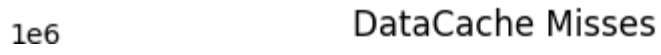
- Transposição simples: 10.7%
- Transposição 32x32: 6.3%

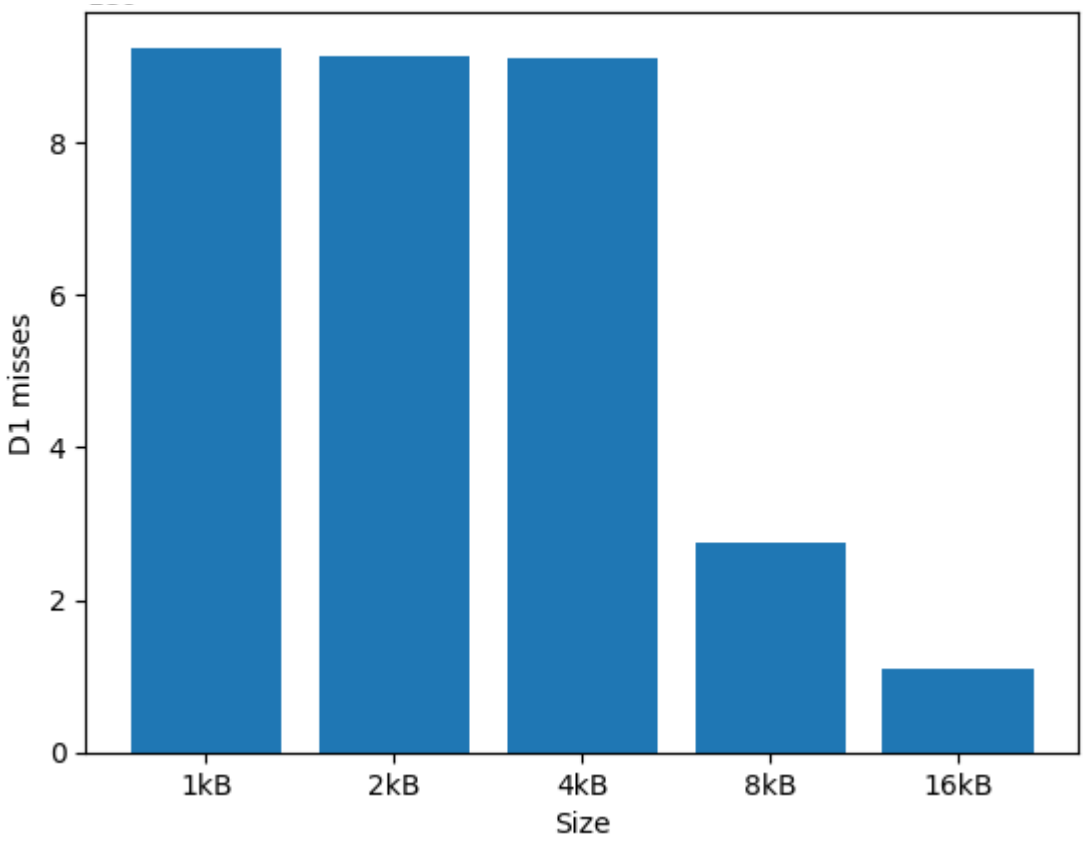
Logo, quanto menos falha na cache L1 melhor.

Variando o tamanho da Cache e visualizando falhas e taxa de falhas

A extensão **%%rangecachegrind** executa várias vezes com tamanhos de cache especificados pela lista **datacache=(4,8,16,32)**, em Kbytes. O usuário especifica a associatividade (**ways**) e o tamanho do linha (**line**), os gráficos são gerados de forma automática.

```
%%rangecachegrind datacache=(1,2,4,8,16); ways=2; line=32; bargraph=(misses)

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char const *argv[]) {

  int n = 200;
  int a[n][n], b[n][n], c[n][n];

  for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
        a[i][j] = i + j;
        b[i][j] = i*2 + j;
      }
  }

  int temp;
  for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
          temp = 0;
          for (int k = 0; k < n; ++k) {
              temp += a[i][k] * b[k][j];
          }
          c[i][j] = temp;
      }
  }
  return 0;
}
```





Tarefa

Variando os valores da cache de dados, ways e a lines (Utilize criatividade para mostrar)

```
%%rangecachegrind datacache=(8,16,16,32,32); ways=4; line=16; bargraph=(misses)
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>

void mul_matriz_coluna(int n, int m, int constante, int Matriz[n][m]){
    //Linhas = n
    //Colunas = m
    for(int coluna = 0; coluna < m; coluna++){
        for(int linha = 0; linha < n; linha++){
            Matriz[linha][coluna] = Matriz[linha][coluna] * constante;
        }
    }
}

void mul_matriz_linha(int n, int m, int constante, int Matriz[n][m]){
    //Linhas = n
    //Colunas = m
    for(int linha = 0; linha < n; linha++){
        for(int coluna = 0; coluna < m; coluna++){
            Matriz[linha][coluna] = Matriz[linha][coluna] * constante;
        }
    }
```

```
    }
}

void print_matrix(int n, int m, int Matrix[n][m]){
    for(int linha = 0; linha < n; linha++){
        printf("[");
        for(int coluna = 0; coluna < m; coluna++){
            if (coluna == 0){
                printf("%d", Matrix[linha][coluna]);
            }
            printf(", %d", Matrix[linha][coluna]);
        }
        printf("]\n");
    }
    printf("\n\n");
}

void preenche_matrix(int n, int m, int Matrix[n][m]){
    for(int linha = 0; linha < n; linha++){
        for(int coluna = 0; coluna < m; coluna++){
            Matrix[linha][coluna] = rand() % 64;
        }
    }
}

void copia_matrix(int n, int m, int Matrix[n][m], int Matrix2[n][m]){
    for(int linha = 0; linha < n; linha++){
        for(int coluna = 0; coluna < m; coluna++){
            Matrix2[linha][coluna] = Matrix[linha][coluna];
        }
    }
}

void menu_matriz(int n, int m, int Matrix[n][m]){
    int opcao;
    int constante;
    int controle = 1;

    printf("Digite o valor do esacalar: ");
    scanf("%d", &constante);

    while(controle) {
        printf("Escolha a operação que deseja realizar:\n");
        printf("1 - Multiplicar linha a linha por um escalar\n");
        printf("2 - Multiplicar coluna a coluna por um escalar\n");
        scanf("%d", &opcao);

        switch(opcao){
            case 1:
                controle = 0;
                mul_matriz_linha(n, m, constante, Matrix);
                print_matrix(n, m, Matrix);
                break;
            case 2:
                controle = 0;
```

```
                controle = 0;
                mul_matriz_coluna(n, m, constante, Matrix);
                print_matrix(n, m, Matrix);
                break;
            default:
                printf("Opção inválida\n");
                break;
        }
    }
}

int main(){
    printf("\n");
    int n = 1000;
    int m = 1000;
    int Matrix[n][m];

    preenche_matrix(n, m, Matrix);
    print_matrix(n, m, Matrix);

    menu_matriz(n, m, Matrix);
}
```

```
/content/valgrind_code.cpp:6:65: error: use of parameter outside function b
    6 | void mul_matriz_coluna(int n, int m, int constante, int Matriz[n][m
      |                                                                 ^
/content/valgrind_code.cpp:6:68: error: use of parameter outside function b
    6 | void mul_matriz_coluna(int n, int m, int constante, int Matriz[n][m
      |
/content/valgrind_code.cpp: In function ‘void mul_matriz_coluna(...)’:
/content/valgrind_code.cpp:9:34: error: ‘m’ was not declared in this scope;
    9 |     for(int coluna = 0; coluna < m; coluna++){
      |                                  ^
      |                                  tm
/content/valgrind_code.cpp:10:36: error: ‘n’ was not declared in this scope
   10 |         for(int linha = 0; linha < n; linha++){
      |                                    ^
/content/valgrind_code.cpp:11:13: error: ‘Matriz’ was not declared in this
   11 |             Matriz[linha][coluna] = Matriz[linha][coluna] * constan
      |             ^~~~~~
/content/valgrind_code.cpp:11:61: error: ‘constante’ was not declared in th
   11 |             Matriz[linha][coluna] = Matriz[linha][coluna] * constan
      |                                                             ^~~~~~~
/content/valgrind_code.cpp: At global scope:
/content/valgrind_code.cpp:16:64: error: use of parameter outside function
   16 | void mul_matriz_linha(int n, int m, int constante, int Matriz[n][m]
      |                                                                ^
/content/valgrind_code.cpp:16:67: error: use of parameter outside function
   16 | void mul_matriz_linha(int n, int m, int constante, int Matriz[n][m]
      |                                                                   ^
/content/valgrind_code.cpp: In function ‘void mul_matriz_linha(...)’:
/content/valgrind_code.cpp:19:32: error: ‘n’ was not declared in this scope
   19 |     for(int linha = 0; linha < n; linha++){
      |                                ^
/content/valgrind_code.cpp:20:38: error: ‘m’ was not declared in this scope
   20 |         for(int coluna = 0; coluna < m; coluna++){
      |                                      ^
      |                                      tm
```

```
      |                                                 tm
/content/valgrind_code.cpp:21:13: error: ‘Matriz’ was not declared in this
   21 |             Matriz[linha][coluna] = Matriz[linha][coluna] * constan
      |             ^~~~~~
/content/valgrind_code.cpp:21:61: error: ‘constante’ was not declared in th
   21 |             Matriz[linha][coluna] = Matriz[linha][coluna] * constan
      |                                                             ^~~~~~~
/content/valgrind_code.cpp: At global scope:
/content/valgrind_code.cpp:26:45: error: use of parameter outside function
   26 | void print_matrix(int n, int m, int Matrix[n][m]){
      |                                             ^
/content/valgrind_code.cpp:26:48: error: use of parameter outside function
   26 | void print_matrix(int n, int m, int Matrix[n][m]){
      |                                                ^
/content/valgrind_code.cpp: In function ‘void print_matrix(...)’:
/content/valgrind_code.cpp:27:32: error: ‘n’ was not declared in this scope
   27 |     for(int linha = 0; linha < n; linha++){
      |                                ^
/content/valgrind_code.cpp:29:38: error: ‘m’ was not declared in this scope
   29 |         for(int coluna = 0; coluna < m; coluna++){
      |                                       ^
      |                                       tm
/content/valgrind_code.cpp:31:30: error: ‘Matrix’ was not declared in this
   31 |                 printf("%d", Matrix[linha][coluna]);
```

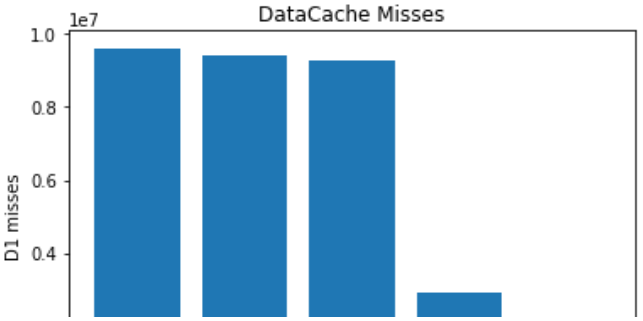
Explicação dos resultados

TODO

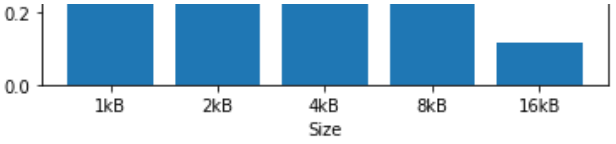
Utilizando os seus dados com gráficos, e
significados deles e o que acontece quar
associatividade (**ways**) e o tamanho d
utilize os gráficos gerados acima na exp
imagens junto ao texto, basta copiar a f
código da imagem).

Exemplo:

```
![image.png](
iVBORw0KGgoAAAANSUhEUgAAAYIAAAEWCAYAAABı
igOoCTwnLOMUBwIGE4PGiagxQY2liWgEgmWUWDGl
d/evfuefe989+5z/
bvfu1VX3nal3n15r797rnj7nngeoiAExIAbEgBgC
iAExIAbEgBgQA2JADIgBMSAGxIAYEANiQAyIATEg
CsD/APgZgP8C8GkATwaw85KduRWArQB2XbJ+qvZc
f2a9NMSAGxEC1DFAIHmC9vwGAIwF8DcDrlkTkEYJ
F7tsALgcgIciI0aoYEAN1M5ALQUJyNwD/B+COtuM
wTwIwBvAXBDM6DYsN7DUwM9S57/M3aFwquFkwHsl
eAeCNdoVzMYC7ZHY3t8T+QxO7p2fHuqu8IvgHa/
9GdvAhduXyqUwIKAjcZtkJAK8YKHLk6sKMw8MBXG
bEgBiYhIE+IWDDTPBPsTPcF8CdbLqISYrJ9/
ftWN8VwW0APBDA7gBuCuATAF5q9Q8D8L8bTCWtAb
```



```
++52jELwSwBMursAeCGAc+wYp7q
+AOC5Jir7A7gSwIPteHdBIXgBgFdnPFBkHmWJP10
WATDmJgnx8HgD4hd7cz4aWAsZALiqyKGCiOgddaI
4bYnffPi5zL4BLWGmKjuQgXlCwAR64pzzMqnTlyx
+rILisB7rTITcWqra08h+Gi280C7/8FdTLQUt7yc
0kANfPOwDgFQD+trOP3/zvA4CCyqsLTt1x
+kxFDBTLwL3t280yQpCDeBoAiojKahmYJwScAkpX
ZlManCLhVBDbY1nmiuC2AE41waD9LwB80uyfDeBc
J8D7F6+xY/OmhniY0028KuCUGK8mUuK/
K4D3A7gawMdtCo31eX7iS33iktsUPBbeVOf5aEdC
QGW1DPQJAZMW7xFwOojlCgDPsKkYbvOKICXcfXue
DynfVh4RZDm4Zkgv2j7u4tFQsB7F1/tGizYzoWA7
+v9mTyySMr1pw5ZXa4pKCQk6TAOfHtC4GimCgKwC
htkjcO88LkwTlTzomqrJaBXAiYbHjjkomfN1xT4f
1fx7CwBnA0hCwDqc30/
3GWjP5Pi7AF5kDXzO5vI5z84vEZwqSUKQ7hFQHDi
ovEnL/rDMEwK2xXgnHj4NxS9Dz7d7EpwSowiyPBH
+W8wr6DWanhRgojoFcCK5n87LpXgCXvNmXFw7Gl+
+LJE+5JNm3EaQ3Oj/OpHk6PcEqI/mfiz4WAYJkY+
+YPa8o2X4SAlZhAueXDU6X0PZ4s1t0RcAqnFbht2
wwiVMUeE5OdX2+85sJTpdxH/
njF6R3mhDwxjxFkbzzCSlyr6mhrje0XQwDuRDwWy
+n58R5ufubGQ5e6vNbaLrMzg5pVQyIATEgBmpkgJ
+RMNnt1Ph5fvfpw0txYAYEANiQAyIATEgBsSAGNg
gYKMfffEn+Xz7IwWBT318dplO7bXXXlvX1tb0EQe
o5ifgj1rSLxm5n89tp3ek5PXWrVMEVMSAGBADYmA
bo9Lp8ulkbi4SAzyfn74LnM9r5WxvzPh5jIM7dsm
HfwNzUrKVEKwrfO6IlBEiwExIAaGM1CqEGhqaLgv
+rH3DoisCVwzISAyIgcYZWJUQ9P3oi/+Xlh8WPi3
ayMf5XpXn3B6z6bCEhaDyaBV8MiAEXA6sSgnUJfK
yIgcYZkBA0HgCCLwbEgBiQEFgM7HvcqVujfBTWYk
FdMSAGIjEgITBvSggihbWwiAExMIQBCYGx1boQRM
+D3bZiIHWGZAQWARESYQSgtaHtPCLgeEMSAiMMwl
+CfBwJptYDEgIzJ8SAglBhBiIlZ6EZrMYkBBICK5
+z1CLgl1C4PG+bMiAhMDioPVkEAW/
Z1hHwS4h8HhfNmRAQmBx0HoyiILfM6yjYJcQeLwv
ouD3DOso2CUEHu/LhgxICCwOWk8GUfB7hnUU7BIC
uyIQMSAouD1pNBFPyeYR0Fu4TA433ZkAEJgcVB68
HweF82ZEBCYHHQejKIgt8zrKNglxB4vC8bMiAhsD
jYJQQe78uGDEgILA5aTwZR8HuGdRTsEgKP92VDBi
J5hHQW7hMDjfdmQAQmBxUHrySAKfs
+wjoJdQuDxvmzIgITA4qD1ZBAFv2dYR8EuIfB4Xz
TB7xnWUbBLCDzelw0ZkBBYHLSeDKLg9wzrKNglBE
+BKACwAcvn2V9XvW1tbcXm09GUTB7wmAKNglBB7v
qnN8u00JwalbvckgSjL0DOso2L2
+93Amm1gMrEoIDgVwepbJTwDAT15eBeA428H6n84
2zdV9AFwI4FsArgaw1jmeNo8xEOdu2bLF7ZvWk0E
fUlcEuiLwDAIJgYc12URiYFVCsMzU0MUAbpkl/
CsB7J1tb7cqIZAQeAanhMDDmmwiMbAqIdgVABP7f
+3fXcA8B0AO3XqrNuUEEgIPINTQuBhTTaRGFiVED
+zISGQEHgGp4TAw5psIjGwSiHoSeXjdkkIJASewS
Za68kgCn7PIIiCXULg8b5syICEwOKg9WQQBb9nWE
AhsDhoPRlEwe8Z1lGwSwg83pcNGZAQWBy0ngyi4F
VDBiQEFgetJ4Mo+D3DOgp2CYHH+7IhAxICi4PWk0
```

```
+wjoJdQuDxvmzIgITA4qD1ZBAFv2dYR8EuIfB4Xz
TB7xnWUbBLCDzelw0ZkBBYHLSeDKLg9wzrKNglBE
3ZUMGJAQWB60ngyj4PcM6CnYJgcf7siEDEgKLg9a
+z7COgl1C4PG+bMiAhMDioPVkEAW/
Z1hHwS4h8HhfNmRAQmBx0HoyiILfM6yjYJcQeLwv
ouD3DOso2CUEHu/LhgxICCwOWk8GUfB7hnUU7BIC
uyIQMSAouD1pNBFPyeYR0Fu4TA433ZkAEJgcVB68
HweF82ZEBCYHHQejKIgt8zrKNglxB4vC8bMiAhsD
jYJQQe78uGDEgILA5aTwZR8HuGdRTsEgKP92VDBi
J5hHQW7hMDjfdmQAQmBxUHrySAKfs
+wjoJdQuDxvmzIwJRCsDOA62P5chiASwFcDuD4OW
Qe78uGDIwVAiZnJv/rWsL+FoC/2paZ56/sAuAKAF
dOb7dpoTg1K3eZBAlGXqGdRTsXt97OJNNLAbGCsF
+MnLiwD8ab5jo3UJgYTAMzwlBB7WZBOJgbFCwCkb
+j4AFIOzAZwDgFNJC4uEQELgGZwSAg9rsonEwFgh
Xs4DJCcCqA95rQ7AfgmwBu2NP2MQbi3C1btrh903
sBPCGzOwPAXbPt7VZ1RaArAs8gkBB4WJNNJAbGCs
rpZvFBnWqcCnqD7buJXRHs1amzblNCICHwDE4JgY
43AAl9nTQydatZMAHGnrnGp6iT2NxDYfOb
+p2REJgYTAMzglBB7WZBOJgbFC8HlLznzMM5X0JF
h4APr5pmb9zIgmBhMAzOCUEHtZkE4mBsUJwiD3e
+RNbcqrnzp38vGmbEgIJgWdwSgg8rMkmEgNjhYBJ
glBB4WJNNJAbGCsHDAexpCfk5AN4DgFcJKykSAgm
IJASewSkh8LAmm0gMjBWC9LTQCwE82gQg7dt0PZA
+7ZbwokBMfNEvNmJxpPgG52H3fU
+VrGTk5VxICHgbFCsAeAhwHgK6NZ9gHwIFvf9IWu
1/diuE0RBQmBhMAzpFeRsHfUOT34ZSMGxgoBf0XM
+opLyKdj34ZSMGxgpB
+kXxswE8zRKzbhav6N5ASjyesE62tS9bxk7fqYgE
/nvwy0YMjBUCvnb6ZSYGTP58rfRxK1EB/Y7gmv9X
+qKQFcEniHdTaY1b3vwy0YMeIXgHZaJ+X8C+Ovi9
+HsBFv6P4r6PHd7chYRAQuAZ0t1kWvO2B79sxIBX
+37x78shEDY4XgSQC+B+AqAF+zD/8X8UqKhEBC4E
5u3z34ZSMGxgrBhwDwfUNFFAmBhMAzpLvJtOZtD3
/bdw9+2YiBsULwOQAvAfAEAI/
LPhKCFb5mwhPW3YRS63bL2OkzFTHgYWCsEKzsvUJ
Q6DHR1d4FZAnBU9Y5/Y1r7eMnX5TEQMeBsYKQXpk
nnfW8YuIfB4XzZkYKwQ9M3QrGyfpoY0NeQZ1rmQ1
feEdW5f83rL2Ok3FTHgYUBCYKzVnPy6ffcEQreNw
+lrHTXypiwMPAjhCC26/qJoHuEegegWcQ1Ch48/r
+bVb7LWMnxypiwMOAVwjSqyS6y5cD+KmEQEKwWYm
ex7PIOi2UfO2B79sxIBXCH5mPyTLXyuR1n8kIZAC
6B6B7hF4hvSOSMiratODXzZiwCsEfJ3E2NdPHwbg
+pcc0hCICHwDOlVJe0dcV4PftmIAa8Q5Dn5pgD4G
HdEUlpFmy1jJ98qYsDDgFcIdgLwPAC8H/BjAFcD+
+uuWlAI4AcJaEQEKwjLB4BsEy7dZSx4NfNmLAKwT
1EADvtn0SggEvsvOEdS2JbqN+toyd3KiIAQ8DXiH
+BvtcfrZs2eLh4BqbjRJMTcc9JNSEb1FfW8ZOXlT
BPCdjaaHdLNYN4s9g2CRsNR2zINfNmLAKwRfTBm9
+3wJOLaWbxQelgz1LTQ1pamjrMknZM6SXabeWOh7
+daBVOAnBkT2UJgYRAQrBEDCiliQEPA14h6MnVq9
bMSAhMBiYJlBVksdT1jXgm2jfraMndyoiAEPAxIC
dswTCLVhnNfflrGTExUx4GFAQmCszUssNe73BEKN
+TyDUirXb75axkwsVMeBhQEJgrHUTSs3bnkCoGW/
```

```
e95axkwcVMeBhQEJgrOXJpPZ1TyDUjjn1v2Xs5EE
EwPzZejKIgt8zPKNglxB4vC8bMiAhsDhoPRlEwe8
GDEgILA5aTwZR8HuGdRTsEgKP92VDBiQEFgetJ4M
J5hHQW7hMDjfdmQAQmBxUHrySAKfs
+wjoJdQuDxvmzIgITA4qD1ZBAFv2dYR8EuIfB4Xz
TB7xnWUbBLCDzelw0ZkBBYHLSeDKLg9wzrKNi9Qh
+X72v65r52G452WRGJAQGBO1D4K8/8m5Q5a5fc3r
D39Xu9i5fb6munxn3LoV1fq0ac8/q8HtlyW/Paqm
+pvn6M32reovZqObYSz73hN+Dbqax++jfZt1GYtx
UgOuZfs4H+X8I8u2XXq9+Qh1ZCMGJATGUOlBPqR/
dh077lGJAQGE+lBrenX8u5fn0tz3lKtFmParmtEr
+jMkAFLdMecryTbhGbIsqf9j
+zIEd6o79pyl2Cc8Wg5nQEJgnJUSzFP0Y3gYbN06
nuAupe9j+9EydnLnKWM5L8Xeg102MwYkBBYJpQTz
TBg102MwYkBBYJUwRiKW14gruUvo/tR8vYyZ2njC
j2lBL9N0QcPdtnMGJAQWCRMEYiltOEJ7lL6PrYfL
RD09wT3HeEtpoGTv595QS/DZFHzzYZTNjQEJgkTE
+n72H60jJ3cecpYzkux92CXzYyBVQrBYQAuBXA5g
a2prbr6UE8xT98JAwxXlLaKNl7OTfU0rw2xR98GC
gwPVpHfcDsIftewqAt3eOb7cpIZj9gxFPcE8xEEt
ZVw4GcPa8g2m/
hEBC4BnYUyShUtpoGb8Hu2xmDKxKCI4CcEpK4ACC
VrKQJ6iHx4SpjhvCW20jJ38e0oJfpuiDx7sspkxU
MtmxsCqhGDZqaEHAPgygL23ZfsFKxICCYFnYJeSx
+AN5QP6OyfuykhkBB4BvYUCbiUNlrG78EumxkDqx
+YqgB2QEEgIPAO7lCQ+RT9axu/BLpsZA6sUgo3y
+uDjEgIJgWdgT5GAS2mjZfwe7LKZMSAhsEgoZSBF
+feUEvw2RR882GUzY0BCYJEwRSCW0oYnuEvp+9h
+tIyd3HnKWM5Lsfdgl82MAQmBRUIpwTxFPzzBPcV
gLqXvY/vRMnZy5yljOS/F3oNdNjMGJAQWCaUE8xT
TynBb1P0wYNdNjMGJAQWCVMEYilteIK7lL6P7UfL
TDE9xTnLeENlrGTv49pQS/TdEHD3bZzBiQEFgkTE
+j62Hy1jJ3eeMpbzUuw92GUzY0BCYJFQSjBP0Q9F
+feUEvw2RR882GUzY0BCYJEwRSCW0oYnuEvp+9h
+tIyd3HnKWM5Lsfdgl82MAQmBRUIpwTxFPzzBPcV
gLqXvY/vRMnZy5yljOS/F3oNdNjMGJAQWCaUE8xT
TynBb1P0oWXsXt8nziQExsQUgVhKG8m5Q5al9H1s
pyxDMqe6Y85VmmzB5lhICY600p47pjycQxpyvJNu
8b0pWXsXt8nziQExsSYACzNNjl3yLI0DN7+DMGc6
+GYE51Pecp1SZh8iwlBMZaqc719MsTCJ7zlGjTMr
+cSYhMCY8gVeqTXLukGWpWIb2awjmVHfoOUqunzA
PIEwpP2S67aMnX7xlJL9OaRvLWP3+j5xJiEwJoYE
hmBOdZdtu4Z6CdOQZQ24lunjEMyp7jLt1lInYfIs
9X3iTEJgTCwTaLXUSc4dsqwF20b9HII51d2ozZqC
TyAsaq+mYy1jp588pSb/Lupry9i9vk
+cSQiMiUUBVtux5Nwhy9owzuvvEMyp7ry2atyfMA
raqXFfy9jpL0+p0c99fW4Zu9f3iTMJgTHRF1i17k
d9COZUN7evfT1h8iwlBMZa7UGQ998TCLl9zestY6
eJMwmBMZEHVO3ryblDlrVjTv0fgjnVTbYRlgnTkG
EnziQExkTrAd5UJE8AAAetSURBVBEFfwrsIcso2L
+FNgD1lGwe5NBlHwD/F5qhsFu9f3iQcJgTHRekBE
+TzxICIyJ1gMiCv4U2EOWUbB7k0EU/EN8nupGwe7
i81Q3Cnav7xMPEgJjovWAiII/
BfaQZRTs3mQQBf8Qn6e6UbB7fZ94WKUQHAbgUgCX
```

```
+z9raWsI1eNl6QETBP9jxW7dujYLdmwyi4JfvPQz
+gAPXp3X8OYBX2r5Hmih0qqzflBCcek1S84SDksG
+vvaqhOBQAKdnafwEAPzkhcdZj2VXAD8CsJNt9y4
+lq1Y079X49qua1kG2G5HOL+WqsSgqMAnJJl86ME
i5qtB+bQZXwl5mTMr3O9YvNcT9D1NC3czllEKwmf
+oc93yNTQwjOWc7DloBD2cuJws3si32824xWcj3F
Y7mWsZf8vYNxw1hwO4zJ4eOtFqnwTgSFu/NoB32u
+l7IxYAYEANiQAyIATEgBsSAGBADYkAMiAExIAbE
+w3EAnzWQDukjayJZ8tvhDAebZ8aHasxtVbAjgTw
8Rf0XAdyzRod3+vwM8zt///M2ALzHFxl/
31gnJU8D8BXj4kXG0eN7fifFQ88D8G2LBdq8AsDC
+XiHevMv7GHbu29MeBOBrQxYJQST8xH0sgLcuIQC
H1j/X4APgqA70lj2duWi4TgWVaHAvApAGxDJQADf
DATAii4
+ev4M8AcP8eIWCCPA3An5lj80T4cADvq9zhFIJvA
CuCJAS8guJjpr/VY69dFTLQDQ4KwT3scjk9JktYF
7sArAG4b0cIyAW/Jf5xRlaaGuJ0wE/MLjtc5SqnA
u5Y5zTv8wHwbcm8yuMXPJYkBH8A4JMAbmT786mhc
+1q0g5pUTsD3eCgEHAuOBcBYsyvCG5t29erHTwAY
kVAX9lz3sqC1+o2LEvbZOJ7WMAbgrgWnaF81iL6c
5VcE5I1X0XzTskoABrrBwUTIm0D8VsjLv1RyIeA+
37ArvzbZNgXhTJ9HnQkCevp/NJyfealpyeus1WYc
MGRXAhI31OyLxQZnVqtkYFucDAR0vlMjh+wOVTiy
+zuFQFviL+sM8BzIbi9vWKdTxzVWu5uSW4PE7w32
+O9ScD4NsRWG5r903SFQHfrkxf88ogvU4nFwLW45
DGKQdJenyUUwN/UjFudv1eALYCuMBufHO+lK8RSU
DkB6nDDdIyBHnDrhDcTaC2Ob9zw4PcIrID45k4Qg
ssVkTg74WDAfHGBJ9wi4frCJAaeD83sEzAFsMz15
MSAGxIAYEANiQAyIATEgBsSAGBADYkAMiAExIAbE
+Wz9acA4Iv3VMSAGBADYiA4A3xNBF8klt46yR+R3
niMn0vtHTSsy5fU8eVkfDcTX83BV3mriAExIAbEC
+i4fE/AsB/dqIiBsSAGBADlTLAdwXx9RJ81QJfPM
+Gl2pcDXjXzYjmkhBsSAGBADlTNwlE0V5ULAf1jy
0LHt00mIdjX7g3wbZWp8AVllwPgjWYWvsI7vZXSc
+crhvn46HvsXxEmIfgbe9V0umH8QQPGf0v4CXsDK
Kshbc6qcYEANiQAyIATEgBsSAGBADYkAMiAExIAb
DYkAMiAExIAbEgBgQA2JADIgBMSAGxEAgBv4fVtC
```