Lezione 12
Bioinformatica

Mauro Ceccanti‡ e Alberto Paoluzzi†

† Dip. Informatica e Automazione – Università “Roma Tre”
‡ Dip. Medicina Clinica – Università “La Sapienza”

Forward computation
Example

Basic tools

```python
def colortable (n):
    """To build a color lookup table with n elements
    Return a list of GLcolor
    ""
    return [GLcolor(array([1.0/n, 0, 0, 0]))*k.tolist() for k in range(1, n+1)]

colors = colortable(10)
size = 5
coords = [[i, j] for i in range(size) for j in range(size)]

for i in range(size):
    table = STRUCT(map(item, coords))
    numbers = STRUCT(map(mass, coords))
```
Basic tools

```python
def item((i,j)):
    '''To generate a single colored cell of the table.
    Return a colored unit square translated in (i,j).'''
    transl = T([1,2])([i,j])
    if a[i,j] > 0: color = colors[a[i,j]]
    else: color = WHITE
    return COLOR(color)(transl(CUBOID([1,1])))
```

```python
def mass((i,j)):
    '''To write the field value in position (i,j)
    Return a text suitably positioned in 2D space-
    'value' with attributes and adjusted position.'''
    transl = T([1,2])([i+0.5,j])
    m = TEXTWITHATTRIBUTES('centre',0.0,0.3,0.5,0.0) (str(abs(a[i,j])))
    return transl(T(2)(0.25) {m})
```

```python
goal = (0,(2,2))
update (goal)
VIEW(show(table))
boundary = [goal]
while boundary != []:
    c = boundary.pop() [1]
    new = cellforward(c)
    for e in new: insort(boundary, e)
    VIEW(show(table))
```

Forward computation

Algorithm **FORWARD**($A$, $h$)

Require: a (weighted) matrix $A$; a goal cell ($h$)
Ensure: $A$ filled with minimal distances from goal.

1: boundary = $h$
2: $A[h] = 0$
3: while boundary $\neq \emptyset$ do
4: pop(c, boundary)
5: insort(boundary, CELLFORWARD(c))
6: end while

Algorithm **CELLFORWARD**($A$, $k$)

Require: a matrix $A$; a current boundary cell ($k$)
Ensure: a subset of boundary cells.

1: adjacent ($k$) = \{ $i$ for $i \in \text{NEIGHBOR}(k)$ if $a[i] \geq 0.0$ \}
2: for $i \in \text{adjacent (k)}$ do
3: \quad $a[i] = -(a[i] + a[k])$
4: end for
5: return adjacent
a $d$-cell $\gamma$ is represented as a multi-index $k = (k_0, ..., k_{d-1}) \in \mathbb{Z}^d$

Algorithm \textsc{Neighbor}($k$)

\textbf{Require:} an $d$-cell ($k$)

\textbf{Ensure:} the set of $d$-cells $\delta \partial k - \{k\}$.

1: \textbf{return} $\delta \partial k - \{k\}$

Laplace-Beltrami operator $\delta \partial$

Examples in several dimensions

\begin{verbatim}
>>> print neighbor([8,7])
[[7, 7], [8, 6], [9, 7], [8, 8]]

>>> print neighbor([7])
[[6], [8]]

>>> print neighbor([1,2,3])
[[0, 2, 3], [1, 1, 3], [1, 2, 2], [2, 2, 3], [1, 3, 3], [1, 2, 4]]

>>> print neighbor([1,2,3,4])
[[0, 2, 3, 4], [1, 1, 3, 4], [1, 2, 2, 4], [1, 2, 3, 3], [2, 2, 3, 4], [1, 3, 3, 4], [1, 2, 4, 4], [1, 2, 3, 5]]

>>> print neighbor([0,0])
[[1, 0], [0, 1]]
\end{verbatim}
```python
def maxClimb(a, element):
    coboundary = neighbor(element)
    maxelement = coboundary.pop()
    while coboundary != []:
        temp = coboundary.pop()
        if a[tuple(temp)] > a[tuple(maxelement)]:
            maxelement = temp
    return maxelement
```
```python
def backPath(a, List):
    path = List
    if a[tuple(path[-1])] != 0:
        path += [maxClimb(a, path[-1])]
        backPath(a, path)
    return map(tuple, path)
```

```python
def backward(a, points):
    tree = set([])
    for point in points:
        tree = tree.union(backPath(a, [point]))
    return list(tree)
```

```python
def item((i,j)):
    """ to generate a single colored cell of the table. ""
    transl = T([i,2])([[i,j]])
    if a[i,j] > 0:
        color = colors[a[i,j]]
    else:
        color = WHITE
    return COLOR(color)(transl(CUBOID([1])))
```

```python
def item(index):
    """ to generate a single colored d-cell of the table 
    Return a colored cuboid translated in (i,j,...,k)""
    coords = range(1, len(index)+1)
    transl = T(coords)(index)
    if a[tuple(index)] > 0:
        color = colors[a[tuple(index)]]
    else:
        color = WHITE
    return COLOR(color)(transl(CUBOID([1]*len(index))))
```

Multidimensional update
def mass ((i, j)):
    """To write the field value in position (i, j)
    Return a text suitably positioned in 2D space"""
    transl = T((1, 2))((i + 0.5, j))
    m = TEXTWITHATTRIBUTES('centre', 0.0, 0.3, 0.5, 0.0) (str(abs(a[i, j])))
    return transl(T(2)(0.25){m})

def mass (index):
    """To write the field value in position (i, j)
    Return a text suitably positioned in space"""
    coords = range(1, len(index)+1)
    tpar = [0]*len(index)
    tpar[0] = 0.5
    transl = T(coords)(array(index) + array(tpar))
    m = TEXTWITHATTRIBUTES('centre', 0.0, 0.3, 0.5, 0.0) (str(abs(a[tuple(index)])))
    return transl(T(2)(0.25){m})