# Algorithms and data structures 3: Naive Matricies, Graphs-1

Boris Kirikov

1.10.2015

# Outline

# Defining matricies

**Def**: Let $I, J, X$ – some sets, $A\colon I \times J \to X$ is called matrix, $a_{ij} = a(i, j)$ is matrix entry, $I$ and $J$ are index sets.

**Def**: Let $M(I, J, X) = \{I \times J \to X\}$ set of all such matricies.

# Defining matricies

**Def**: Let $I, J, X$ – some sets, $A\colon I \times J \to X$ is called matrix, $a_{ij} = a(i, j)$ is matrix entry, $I$ and $J$ are index sets.

**Def**: Let $M(I, J, X) = \{I \times J \to X\}$ set of all such matricies.

**Note**: often $I = \{1, \dots, m\}$ and $J = \{1, \dots, n\}$.

**Note**: often matricies are represented as tables:

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ & \dots & \\ a_{m1} & \dots & a_{mn} \end{pmatrix}$$

**Def**: $M(I, I, X)$ are called square matricies.

# Some common examples

- empty matrix
- column: $|J| = 1$ *(we'll call this vector)*
- row: $|I| = 1$ *(we'll call this covector)*
- submatrix: $I' \subset I$, $J' \subset J$
- diagonal matrix: $diag(a_1, \ldots, a_n)$
- $e = diag(1, \ldots, 1)$
- $0 = diag(0, \ldots, 0)$

*(vector and covector are **NOT** defined like this)*

# Matrix operations

## 1. Sum

Let $X$ be field. (To be precise: $R$-module over assoc. ring with 1)

**Def**: $A, B \in M(I, J, X)$

$$A + B = (a)_{ij} + (b)_{ij} = (a + b)_{ij}$$

**L**: $(M(I, J, X), +)$ is *commutative monoid*:

1. $(A + B) + C = A + (B + C)$
2. $\exists 0 \colon 0 + A = A = A + 0$
3. $\forall A \exists (-A) \colon A + (-A) = (-A) + A = 0$
4. Commutative: $A + B = B + A$

# Matrix operations

## 2. Multiply by scalar

**Def**: $A \in M(I, J, X)$, $\lambda \in X$

$$\lambda A = \lambda (a)_{ij} = (\lambda a)_{ij}$$

**L**: $(M(I, J, X), +, \cdot)$ is left $R$-module:

1. $(\lambda \mu) A = \lambda(\mu A)$
2. $(\lambda + \mu) A = \lambda A + \mu A$
3. $\lambda(A + B) = \lambda A + \lambda B$
4. $\forall A \quad 1A = A$

# Matrix operations

## 3. Matrix multiplication (Kelly)

**Def**: If $A \in M(I, J, X)$, $B \in M(J, K, X)$, then $AB \in M(I, K, X)$ is defined as follow:

$$(AB)(i, k) = \sum_{j \in J} a_{ij} b_{jk}$$

**Th**:

1. $A(BC) = (AB)C$
2. $(A + B)C = AC + BC$
3. $AB \neq BA$

# Matrix operations

## 3. Matrix multiplication (Kelly)

**Def**: If $A \in M(I, J, X)$, $B \in M(J, K, X)$, then $AB \in M(I, K, X)$ is defined as follow:

$$(AB)(i, k) = \sum_{j \in J} a_{ij} b_{jk}$$

**Th**:

1. $A(BC) = (AB)C$
2. $(A + B)C = AC + BC$
3. $AB \neq BA$

## 4. Matrix multiplication (Adamar)

**Def**: $A, B \in M(I, J, X)$

# Matricies and linear equations

Kelly's definition of multiplication lets us to write system of linear equations in matrix form:

$$\begin{cases} a_{11}x_1 & + & \dots & + & a_{1n}x_n & = & b_1 \\ & + & \dots & + & & = & \\ a_{m1}x_1 & + & \dots & + & a_{mn}x_n & = & b_m \end{cases}$$

$$Ax = b$$

# More examples

- shift matrix
- cycle matrix (Coxeter)
- Vandermond
- Jordan cell
- Calculus: Jacobian, Wronskian, Hessian

# Defining graphs

**Def**: $G = (V, E)$ is graph, iff $E \subset \{\{u, v\}: u, v \in V\}$. $V$ is set of verticies, $E$ is set of edges.

**Def**: $G = (V, E)$ is directed graph, iff $E \subset \{(u, v): u, v \in V\}$.

**Def**: $H$ is subgraph of $G$, iff $V(H) \subset V(G)$ and $E(H) \subset E(G)$.

# Defining graphs

**Def**: $G = (V, E)$ is graph, iff $E \subset \{\{u, v\} \colon u, v \in V\}$. $V$ is set of verticies, $E$ is set of edges.

**Def**: $G = (V, E)$ is directed graph, iff $E \subset \{(u, v) \colon u, v \in V\}$.

**Def**: $H$ is subgraph of $G$, iff $V(H) \subset V(G)$ and $E(H) \subset E(G)$.

**Note**: There can be other definitions, adding or removing some properties:

- multiple edges
- loops
- weights for edges or verticies
- ...

# Representations 0: graphic

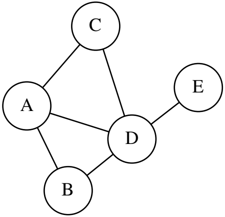


Figure 1: Some random graph

# Representations 0: graphic

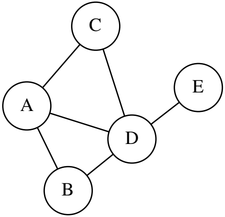


Figure 1: Some random graph

$$V = \{A, B, C, D, E\}$$

$$E = \{\{A, C\}, \{C, D\}, \{A, D\}, \{A, B\}, \{B, D\}, \{D, E\}\}.$$

# Representations 1: edge list

$$V = \{0, 1, 2, 3, 4\}$$

$$E = \{\{0, 2\}, \{2, 3\}, \{0, 3\}, \{0, 1\}, \{1, 3\}, \{3, 4\}\}.$$

```
struct edge {
  int from, to
} G[N];
```

# Representations 1: edge list

$V = \{0, 1, 2, 3, 4\}$

$E = \{\{0, 2\}, \{2, 3\}, \{0, 3\}, \{0, 1\}, \{1, 3\}, \{3, 4\}\}.$

```
struct edge {
  int from, to
} G[N];
```

$O(2E)$ memory.

# Representations 2: Incidence matrix

**Def**: Incidence matrix $Inc(G) \in M(E, V, \{0,1\})$ of graph $G = (V, E)$ is:

$$Inc(G)(e,v) = \begin{cases} 1, & e = (u,w), (u = v \wedge w = v) \\ 0 & \end{cases}$$

**Q**: How many edges can be in graph with $n$ verticies?

# Representations 2: Incidence matrix

**Def**: Incidence matrix $Inc(G) \in M(E, V, \{0, 1\})$ of graph $G = (V, E)$ is:

$$Inc(G)(e, v) = \begin{cases} 1, & e = (u, w), (u = v \wedge w = v) \\ 0 & \end{cases}$$

**Q**: How many edges can be in graph with $n$ verticies?

```
int G[MAX*MAX][MAX];
```

$O(V \cdot E)$ memory.

# Representation 3: Adjacency matrix

**Def**: Adjacency matrix $Adj(G) \in M(V, V, \{0, 1\})$ of graph $G = (V, E)$ is:

$$Adj(G)(u, v) = \begin{cases} 1, & \exists e \in E \colon e = (u, v) \\ 0 \end{cases}$$

**Note**: often *Adj* matrix is defined for weighted graphs and keeps weights of its' edges.

```
int G[MAX][MAX];
```

# Representation 3: Adjacency matrix

**Def**: Adjacency matrix $Adj(G) \in M(V, V, \{0, 1\})$ of graph $G = (V, E)$ is:

$$Adj(G)(u, v) = \begin{cases} 1, & \exists e \in E \colon e = (u, v) \\ 0 & \end{cases}$$

**Note**: often *Adj* matrix is defined for weighted graphs and keeps weights of its' edges.

```
int G[MAX][MAX];
```

$O(V^2)$ memory.

# Representation 4: Adjacency lists

The most compact representation. For each vertex store pointers to all adjanced ones:

```
std::vector<int> G[MAX];
// ... or ...
std::vector<std::vector<int>> G;
```

# Representation 4: Adjacency lists

The most compact representation. For each vertex store pointers to all adjanced ones:

```
std::vector<int> G[MAX];
// ... or ...
std::vector<std::vector<int>> G;
```

$O(\max(V, E))$ memory.

# Graph anatomy

**Def**: Subgraph $P$ of graph $G$ is a path, iff it is simple (no loops, no multiple edges) and its verticies can be ordered so that two verticies are adjacent iff they are cosecutive in the ordering.

**Def**: Subgraph $C$ of graph $G$ is a cycle, iff it can be represented as path $P$ plus edge from the last to the first vertex in ordering.

**Def**: Graph $G$ is connected, iff for any $u, v \in V(G)$ exists path $P(u, v)$ starting from $u$ and ending in $v$. Otherwise it is called disconnected.

**Def**: Maximal connected subgraphs are called (adjacency) components.

**Def**: A walk is $((v_1, \ldots, v_n), (e_1, \ldots, e_{n-1}))$ such that for $1 \leq i \leq n$ the $e_i = (v_{i-1}, v_i)$. If $\forall i, j$ is $e_i \neq e_j$ than it is a trail. If also $v_1 = v_n$, it is a circuit.

# Examples

**Task**: Proove that if $A = Adj(G)$, than $(a_{uv})^k$ is number of paths from $u$ to $v$ of length $k$.

# Examples

**Task**: Proove that if $A = Adj(G)$, than $(a_{uv})^k$ is number of paths from $u$ to $v$ of length $k$.

**Def**: $G = (V, E)$, $deg\colon V \to \mathbb{N}$, $deg(v) = |\{e \in E\colon e = (v, u)\}|$.

**Task**: $G = (V, E)$, $\forall v \in V \quad deg(v) \not\vdots 2$. Proove that $|V| \vdots 2$.

# Sapnning trees

**Def**: Graph $G$ is called a tree, iff it is connected and $|E| = |V| - 1$.

**Def**: Subgraph $H \leq G$ is called spanning tree, iff it is tree and $V(G) = V(G)$.

# Constructing minimal spanning trees: Prim

**Algo**:

1. Let $T = \{v\}$ be spanning tree.
2. At every step we take the lighters edge from $T$ to $G \setminus T$.
3. When no such edges exist, we are done

# Constructing minimal spanning trees: Prim

**Algo**:

1. Let $T = \{v\}$ be spanning tree.
2. At every step we take the lighters edge from $T$ to $G \setminus T$.
3. When no such edges exist, we are done

Let's do it faster:

1. Store for every vertex pointer to lightest edge to $T$
2. Now we select next edge in $O(n)$.
3. When the vertex is added, recalculate all adjacent vertexes for $O(n)$.

This gives $O(n^2)$ performance.

# Constructing minimal spanning trees: Kruskal

**Idea**: Start from $|V|$ trees. At every step the lightest edge between different trees and add it.

**Algo**:

1. Sort all edges
2. *Somehow* make sets of verticies
3. Take the lightest unused edge, merge sets

We'll talk more about this algo and storing sets later.

# Traversing graphs

1. DFS
   - Stack
   - Call stack or array

# Traversing graphs

1. DFS
   - Stack
   - Call stack or array
2. BFS
   - Queue
   - List queue, array queue