Advanced Data Structures
Summary

January 2021

We discussed, among others, about the following data structures and operations on them:

Binary search trees

1. What information must be stored in the nodes of a binary search tree?
2. Answer: the key of the node, a pointer to the parent node, a pointer to the left child, a pointer to the right child, and satellite data.
3. What operations are of interest for binary search trees?

   Answer:

   (a) `search(r,k)`: search the node with key `k` in the binary search tree with root `r`, and return a pointer to it. If there is no node with key `k`, return `null`.

   (b) `minimum(r)`: return a pointer to the node with minimum key in the binary search tree with root `r`. If the tree is empty (that is, `r` is `null`), return `null`.

   (c) `maximum(r)`: return a pointer to the node with maximum key in the binary search tree with root `r`. If the tree is empty (that is, `r` is `null`), return `null`.

   (d) `successor(r,x)`: return a pointer to the successor of node `x` in the binary search tree with root `r`. If node `x` has no successor, return `null`.

   (e) `predecessor(r,x)`: return a pointer to the predecessor of node `x` in the binary search tree with root `r`. If node `x` has no predecessor, return `null`.

   (f) `delete(r,z)`: delete node `z` from the binary search tree with root `r` and return a pointer to the root of the final binary search tree.

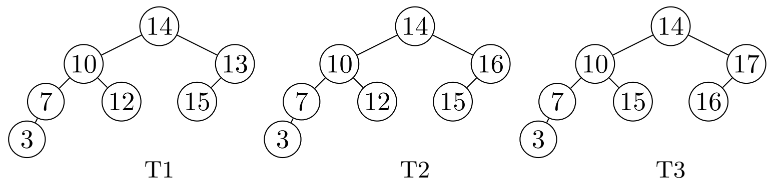
   (g) `insert(r,x)`: insert node `x` in the binary search tree with root `r` and return a pointer to the root of the final binary search tree.. It is assumed that node `p` has no children.

4. What are the average and worst case runtime complexities of these operations when they are performed on a binary search tree with n nodes?

   Answer: All operations have average runtime complexity $O(\log n)$ and worst-case runtime complexity $O(n)$.

5. How are the keys distributed in the nodes of a binary search tree?
   ANSWER: For every node n is the tree, the keys of the nodes of the left subtree of n are smaller than the key of n, and the keys of the nodes of the right subtree of n are larger than the key of n.

6. Which of the following binary trees is a binary search tree?



   ANSWER: T2

7. Consider the binary search tree T2 from the previous exercise.
   (a) Indicate the binary seach tree produced by the insertion of a node with key 11 in T2.
   (b) Indicate the binary serch tree produced by the deletion of the node with key 10 from T2.

8. Consider binary search trees whose nodes are instances of the `C++` class

```
struct Node {
   int key;      // key
   Node *p;      // pointer to parent
   Node *left;   // pointer to left child
   Node *right;  // pointer to right child
   ...           // satellite data
}
```

   You should be able to write down efficient implementations of the main operations on binary search trees, for example:

```
Node* search(Node* r, int k) {
  while(r!=null&&r->key!=k)
    r=(k<r->key)?r->left:r->right;
  return r;
}

Node* minimum(Node* r) {
  if (r == null) return r;
  while (r->left!=null) r=r->left;
  return r;
}
```

```
Node* maximum(Node* r) {
  if (r == null) return r;
  while (r->right!=null) r=r->right;
  return r;
}

Node* insert(Node* r, Node* x) {
  Node* u=r;
  Node* v=null;
  while(u!=null) {
    v=u;
    u=(x->key<u->key)?u->left:u->right;
  }
  x->p=v;
  if(v==null) { // r was null
    return x;
  if(x->key<v->key) v->left=x;
  else v->right=x;
  return r;
}

Node* delete(Node* r, Node* z) {
  Node* newroot = r;
  Node* y=(z->left==null||z->right==null)?z:successor(r,z);
  Node* x=(y->left!=null)?y->left:y->right;
  if(x!=null) x->p=y->p;
  if(y->p==null)
    newroot=x;
  else if(y==y->p->left)
    y->p->left=x;
  else
    y->p->right=x;
  if(y!=z) {
    z->key=y->key;
    // copy y's satellite data into z
  }
  return newroot;
}

Node* successor(Node* r) {
  if(r==null) return r;
  if(r->right!=null) return minimum(r->right);
  Node* y=x->p;
  while((y!=null)&&(x==y->right)) {
    x=y;
```

```
    y=y->p;
  }
  return y;
}
```

Red-black trees

1. Indicate the general structure of a balanced red-black tree, and the red-black properties.

   ANSWER: the nodes of a balanced red-black tree have the same structure as the nodes of a binary search tree, but they have an additional field which indicates the color of the node: red or black.

   A balanced red-black tree has five red-black properties: (1) every node is either red or black, (2) the root is black, (3) every root Nil is black, (4) the children of a red node are black, and (5) every path from a node to a descendant leaf contains the same number of black nodes.

2. What are the main operations on red-black trees, and their worst case runtime complexity?

   ANSWER: They are the same as for binary search trees. Their worst case runtime complexity is $O(\log n)$

3. What is the purpose and effect of the operations LEFTROTATE and RIGHTROTATE on the nodes of red-black trees? You should be able to write down the pseudocode for these operations.

   ANSWER:

4. What are the height and black height of a node x in a balanced red-black tree? You should be able to write down the pseudocode for these operations.

   ANSWER: The height of x is the number of edges of a longest path from x to a leaf node. The black-height is the number of black nodes (including `Nil`) on the path from x to a leaf, not counting x.

   Suppose the nodes of the tree are instances of the class `Node` where

```
enum Color {RED, BLACK};
struct Node {
    int key;
    Node *l, *r, *p; // pointers to left child, right child, and parent
    Color c;         // color
    ...              // satellite data
}
```

Then the height and black height of a node `x` in the tree can be computed efficiently as follows:

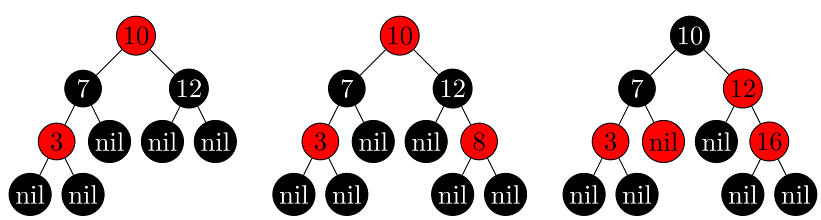
```
int h(Node* x) {
  if (x==nil) return 0;
  int h1=h(x->left), h2=h(x->right);
  return 1+(h1>h2)?h1:h2;
}

int bh(Node* x) {
  int blacks=0;
  if (x==nil) return 0;
  if(x->color==BLACK) blacks--;
  while(true) {
    if (x->color==BLACK) blacks++;
    x=x->left;
    if (x==nil) break;
  }
  return blacks+1;
}
```

If the tree with root `x` has n nodes then

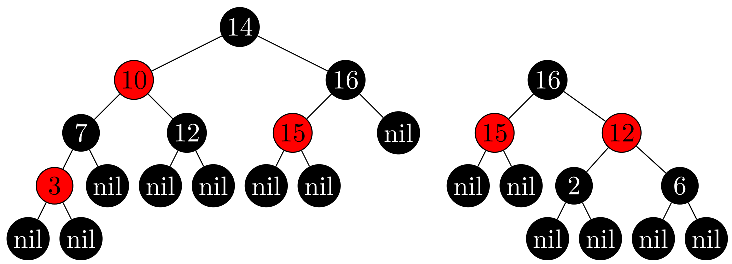
(a) `h(x)` has worst case runtime complexity $O(n)$

(b) `bh(x)` has worst case runtime complexity $O(\log n)$

5. Consider the following trees:



(a) (b) (c)



(d) (e)

(a) Which of them are balanced red-black trees?

(b) For the trees which are not balanced red-black trees, indicate the conditions which are not satisfied.

(c) For the balanced red-black trees, indicate their black-height.

6. Draw the balanced red-black tree produced by the insertion of a node with key `11` in the balanced red-black tree (d) from the previous exercise.

7. Draw the balanced red-black tree produced by the removing the node with key `10` from the balanced red-black tree (d) from the previous exercise.

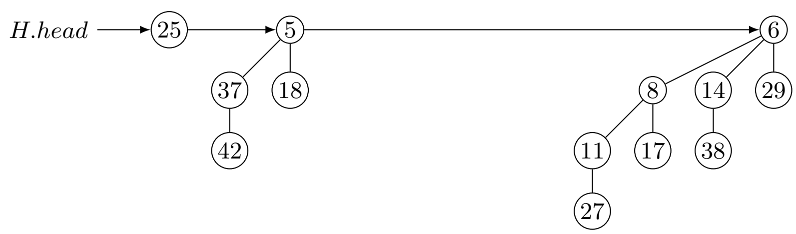
Binomial heaps

1. What is a binomial tree?

2. Indicate a typical `C++` class for the left-child, right-sibling representation of binomial trees. (Indicate the meanings of the fields of this class.) Indicate the additional condition that must be fulfilled by a heap-ordered binomial tree.

3. What is a binomial heap, and how can it be represented in C++?

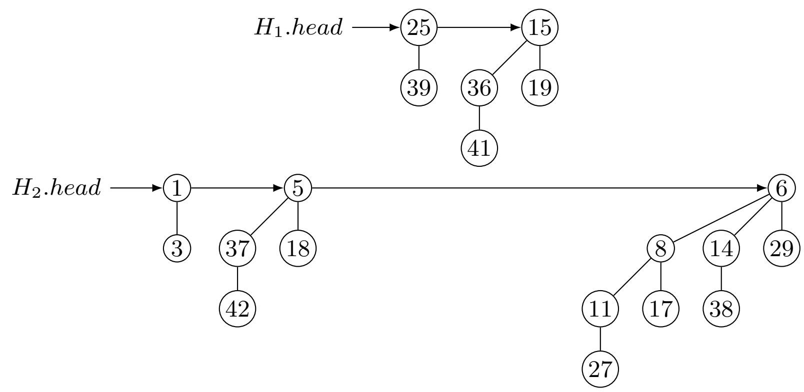
4. Consider the binomial heap H depicted below:



(a) Draw the binomial heap that results after deleting the node with minimum key from H.

(b) Draw the binomial heap that results after decreasing key 27 to 7.

5. Draw the binomial heap that results by merging the binomial heaps H_1 and H_2 depicted below:



6. Write down the pseudocode for the operation $\texttt{merge}(H_1, H_2)$ which computes a binomial heap H by merging the binomial heaps H_1 and H_2 (which are possibly destroyed).

   What is the time-complexity of this operation, if we assume that m and n are the lengths of the root lists of H_1 and H_2?

7. Write down the pseudocode for the operation $\texttt{min}(H)$ which computes the minimum key in a binomial heap H with n key. What is the worst-case time complexity of this operation?

Disjoint-set structures

1. What operations are well-supported by a disjoint-set structure?

2. Indicate how can we compute the connected components of an undirected graph G with a disjoint-set structure.

3. Indicate a `C++` structure for the linked-list representation of a disjoint-set structure, together with its main operations.

4. Indicate a `C++` structure for the rooted-tree representation of a disjoint-set structure, together with its main operations.

Amortised Analysis

1. What is amortised analysis and how does it differ from average-case time analysis?

2. Indicate the most common techniques for amortised analysis.

3. A k-digit ternary counter counts upward from 0, by keeping the base-3 representation of a number in an array $A[0..k-1]$ of digits between 0 and 2. Assume that

- ▶ the lowest-order digit is stored in $A[0]$ and the highest-order digit is stored in $A[k-1]$, thus it represents the number $x = \sum_{i=0}^{k-1} A[i] \cdot 3^i$.
- ▶ The initial number stored in the k-digit ternary counter is 0, thus $A[0] = \ldots = A[k-1] = 0$.

(a) Write down the pseudocode for the operation INCREMENT(A) which adds 1 modulo 3 to the counter.

(b) Use amortised analysis to show that the amortised cost of a sequence of n increment operations on an initially 0 counter of this kind takes $O(1)$ time.

Computational geometry

1. Suppose points in plane are represented in `C++` as instances of the class

```
struct Point { float x, y; };
```

(a) Define the function

```
float area(Point A, Point B, Point C)
```

which takes as inputs three distinct points `A`, `B`, `C` and returns the area of triangle `ABC`.

(b) Suppose we have two rectangles whose edges are parallel with the axes of coordinates. Let

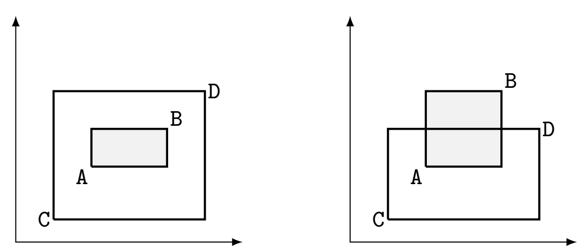
`A`, `B` be the lower-left and upper-right corners of the first rectangle, and

`C`, `D` be the lower-left and upper-right corners of the second rectangle.

Define the function

```
bool inside(Point A, Point B,Point C, Point D)
```

which returns `true` if the first rectangle is inside the second rectangle, and `false` otherwise. For example, (A) depicts a situation when the function call should return `true`, and (B) indicates a situation when the function call should return `false`.



(A) (B)

2. Let $P_1, \ldots, P_n$ be the enumeration of the points of a convex polygon, in clockwise order, and P another point.

(a) Write down an algorithm that tests if P is on any of the bordering segments of this polygon.

(b) Write down an algorithm which tests if P is in the interior of the convex polygon.

String matching

1. What is a keyword tree for a set P of patterns? How can it be used to find fast the occurrences of a pattern from P in a text T?

2. You should be able to compute and draw the keyword tree for a set of patterns, together with its failure links.

3. Let $P = \texttt{abcab}$, and $\delta : \{0,1,2,3,4,5\} \times \{\texttt{a},\texttt{b},\texttt{c}\} \to \{\texttt{0},\texttt{1},\texttt{2},\texttt{3},\texttt{4},\texttt{5}\}$ the transition function of the matching automaton $\mathcal{A}_P$. Remember that, for every $0 \leq i \leq 5$ and $x \in \{\texttt{a},\texttt{b},\texttt{c}\}$, $\delta(i,x) = j$ if and only if $P[1..j]$ is the longest prefix of P that is a suffix of $P[1..j]x$.

   (a) Fill in the following table with the values of $\delta(i,x)$:

$\delta(i,x)$	$x = \texttt{a}$	$x = \texttt{b}$	$x = \texttt{c}$
$i = 0$			
$i = 1$			
$i = 2$			
$i = 3$			
$i = 4$			
$i = 5$			

   (b) What is the value of $\delta^*(\texttt{ababc})$?

Answer:

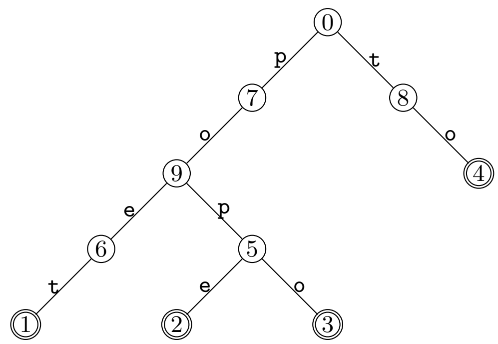
(a) $0 \xrightarrow{\texttt{a}} 1 \xrightarrow{\texttt{b}} 2 \xrightarrow{\texttt{c}} 3 \xrightarrow{\texttt{a}} 4 \xrightarrow{\texttt{b}} 5$

$\delta(i,x)$	$x = \texttt{a}$	$x = \texttt{b}$	$x = \texttt{c}$
$i = 0$	1	0	0
$i = 1$	1	2	0
$i = 2$	1	0	3
$i = 3$	4	0	0
$i = 4$	1	5	0
$i = 5$	1	0	3

(b) Remember that $\delta^*(\epsilon) = 0$ and $\delta^*(wx) = \delta(\delta^*(w), x)$. Thus

$$\delta^*(\texttt{a}) = \delta(\delta^*(\epsilon), \texttt{a}) = \delta(\texttt{0}, \texttt{a}) = \texttt{1},$$
$$\delta^*(\texttt{ab}) = \delta(\delta^*(\texttt{a}), \texttt{b}) = \delta(\texttt{1}, \texttt{b}) = \texttt{2},$$
$$\delta^*(\texttt{aba}) = \delta(\delta^*(\texttt{ab}), \texttt{a}) = \delta(\texttt{2}, \texttt{a}) = \texttt{1},$$
$$\delta^*(\texttt{abab}) = \delta(\delta^*(\texttt{aba}), \texttt{b}) = \delta(\texttt{1}, \texttt{b}) = \texttt{2},$$
$$\delta^*(\texttt{ababc}) = \delta(\delta^*(\texttt{abab}), \texttt{c}) = \delta(\texttt{2}, \texttt{c}) = \texttt{3}.$$

4. The following is a keyword tree without failure links for thee set of patterns {`poet`, `popo`, `pope`, `to`}.

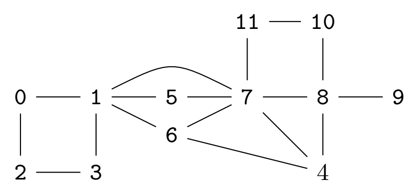


Indicate the failure links that should be added to this keyword tree,

ANSWER: $0 \to 0, 7 \to 0, 8 \to 0, 9 \to 0, 4 \to 0, 6 \to 0, 5 \to 7,\ 1 \to 8,\ 2 \to 0,\ 3 \to 9$.

Tree traversals

1. Let G be the graph depicted below:



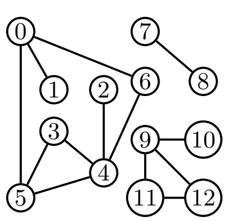
and represented with the adjacency lists

`adj`$[0] = [1, 2]$	`adj`$[1] = [0, 3, 5, 6, 7]$	`adj`$[2] = [0, 3]$	`adj`$[3] = [1, 2]$
`adj`$[4] = [6, 7, 8]$	`adj`$[5] = [1, 7]$	`adj`$[6] = [1, 4, 7]$	`adj`$[7] = [1, 4, 5, 6, 8, 11]$
`adj`$[8] = [4, 7, 9, 10]$	`adj`$[9] = [8]$	`adj`$[10] = [8, 11]$	`adj`$[11] = [7, 10]$

(a) Draw the search tree produced by breadth-first traversal from source node 11. Fill in the table below with the representation with predecessors of this breadth-first search tree:

Node x	0	1	2	3	4	5	6	7	8	9	10	11
`p`$[x]$												`nil`

2. Consider the following graph represented with adjacency lists



`adj`$[0] = [1, 5, 6]$, `adj`$[1] = [0]$, `adj`$[2] = [4]$,
`adj`$[3] = [4, 5]$, `adj`$[4] = [2, 3, 5, 6]$, `adj`$[5] = [0, 3, 4]$,
`adj`$[6] = [0, 4]$, `adj`$[7] = [8]$, `adj`$[8] = [7]$,
`adj`$[9] = [10, 11, 12]$, `adj`$[10] = [9]$, `adj`$[11] = [9, 12]$,
`adj`$[12] = [9, 11]$.

(a) Draw the search trees produced by the breadth-first traversal of all nodes of this graph. We assume that the nodes are enumerated in the order

$$[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12].$$

(b) Draw the search trees produced by the breadth-first traversal of all nodes of this graph. We assume that the nodes are enumerated in the order

$$[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12].$$

(c) Write the lists that enumerate the nodes of the graph in the preorder/postorder and reverse postorder produced by the depth-first traversal performed before.

3. Consider the digraph G represented with the adjacency lists

$\texttt{adj}[0] = [1]$	$\texttt{adj}[1] = [2]$	$\texttt{adj}[2] = [3, 4]$	$\texttt{adj}[3] = [0]$
$\texttt{adj}[4] = [5]$	$\texttt{adj}[5] = [6]$	$\texttt{adj}[6] = [4, 7]$	$\texttt{adj}[7] = []$

(a) What are the values of x, y, z, t if the following is a representation with adjacency lists of the reverse digraph G^r:

$\texttt{adj}[0] = [x]$	$\texttt{adj}[1] = [0]$	$\texttt{adj}[2] = [1]$	$\texttt{adj}[3] = [2]$
$\texttt{adj}[4] = [2, y]$	$\texttt{adj}[5] = [4]$	$\texttt{adj}[6] = [z]$	$\texttt{adj}[7] = [t]$

(b) Draw the trees produced by the depth first traversal of all nodes in G^r. We assume that the nodes of the tree are enumerated in the order

$$[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12].$$

(c) Write the lists L that enumerate the nodes of the graph in the reverse postorder produced by the previous depth-first tree traversal of all nodes of G^r.

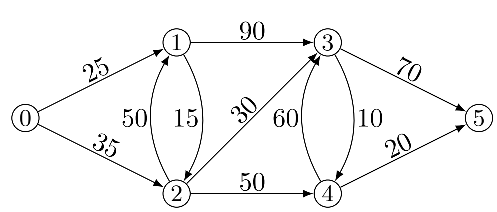
(d) Draw search trees produced by the depth-first traversal of all nodes of G, if we assume that the nodes are enumerated in the order given by the previously computed list L.

(e) How many strong components has G? Indicate them.

Dijkstra algorithm

Example:

1. Consider the following weighted digraph:



Indicate the representation with predecessors of the tree of paths with minimum weight computed by Dijkstra algorithm from source node 0, and draw that tree.

Answer: Remember that Dijkstra algorithm computes progressively the values of 2 arrays:

- `d`$[x]$: the weighted distance from source node (which, in this example is 0) to node x
- `p`$[x]$: the predecessor (or parent) of x in the tree of paths with minimum weight from source node to x.

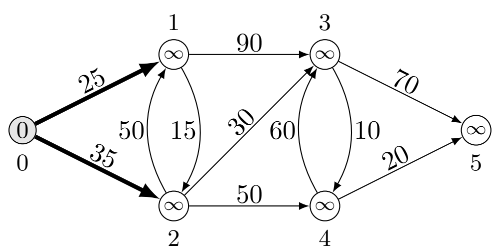
It works as follows.

1. Arrays `p` and `d` are initialized as follows:

Node x	0	1	2	3	4	5
`p`$[x]$	`null`	0	0	0	0	0
`d`$[x]$	0	∞	∞	∞	∞	∞

Tee initialization step is followed by 5 rounds of relaxation steps (In general, number of rounds = number of nodes in the graph). Every relaxation round selects an unmarked node, marks it, and relaxes all edges from the selected node to unmarked nodes.

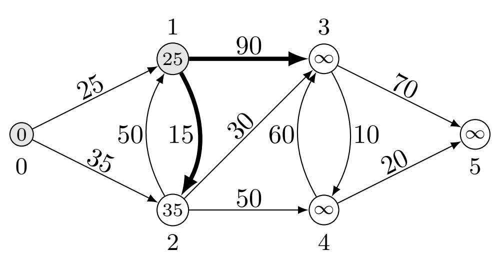
The **first relaxation round** selects node 0, because `d`$[x]$ has minimum value for the unmarked node $x = 0$. We relax the edges $0 \xrightarrow{25} 1$ and $0 \xrightarrow{35} 2$:



After this round, arrays `p` and `d` have values

Node x	0*	1	2	3	4	5
`p`$[x]$	`null`	0	0	0	0	0
`d`$[x]$	0	**25**	**35**	∞	∞	∞

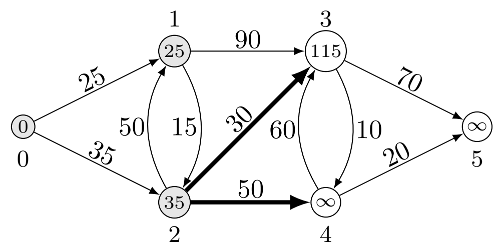
The **second relaxation round** selects node 1, because `d`$[x]$ has minimum value for the unmarked node $x = 1$. We relax the edges $1 \xrightarrow{15} 2$ and $1 \xrightarrow{90} 3$:



After this round, `p` and `d` have values

Node x	0^*	1^*	2	3	4	5
`p`$[x]$	`null`	0	0	1	0	0
`d`$[x]$	0	25	35	**115**	∞	∞

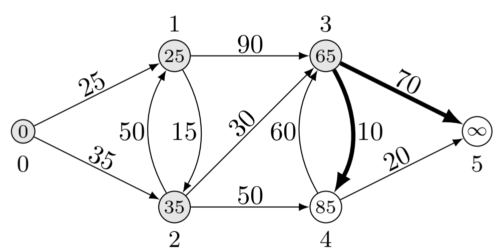
The **third relaxation round** selects node 2, because `d`$[x]$ is minimum for the unmarked node $x = 2$. We relax the edges $2 \xrightarrow{30} 3$ and $2 \xrightarrow{50} 4$:



After this round, `p` and `d` have values

Node x	0^*	1^*	2^*	3	4	5
`p`$[x]$	`null`	0	0	2	2	0
`d`$[x]$	0	25	35	**65**	**85**	∞

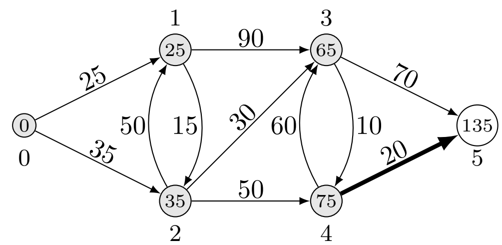
The **fourth relaxation round** selects node 3, because `d`$[x]$ is minimum for the unmarked node $x = 3$. We relax the edges $3 \xrightarrow{10} 4$ and $3 \xrightarrow{70} 5$:



After this round, `p` and `d` have values

Node x	0^*	1^*	2^*	3^*	4	5
`p`$[x]$	`null`	0	0	2	3	3
`d`$[x]$	0	25	35	65	**75**	**135**

The **fifth relaxation round** selects node 4, because `d`$[x]$ is minimum for the unmarked node $x = 4$. We relax the edge $4 \xrightarrow{20} 5$:



After this round, `p` and `d` have values

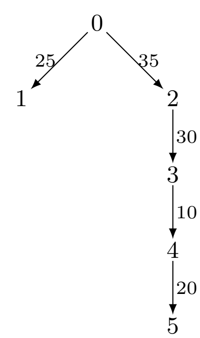
Node x	0^*	1^*	2^*	3^*	4^*	5
`p`$[x]$	`null`	0	0	2	3	4
`d`$[x]$	0	25	35	65	75	**95**

These are the final values of arrays `p` and `d`.

2. The representation with predecessors of the tree of paths with minimum weights from source node 0, computed with Dijkstra algorithm is

Node x	0^*	1^*	2^*	3^*	4^*	5
`p`$[x]$	`null`	0	0	2	3	4

This tree looks as follows:



Working with polynomials

1. Suppose $a = (a_0, a_1, \ldots, a_{n-1})$ is the coefficient representation of a polynomial

$$A(x) = a_0 + a_1\,x + \ldots + a_{n-1}x^{n-1}$$

with degree-bound n.

Write the pseudocode of the operation `deriv`(a) which computes the coefficient representation of the polynomial

$$A'(x) = a_1 + 2\,a_2\,x + 3\,a_3\,x^2 + \ldots + (n-1)\,a_{n-1}x^{n-2}$$

and indicate the runtime complexity of your implementation.