The class **P**: polynomial time

- Theorems 1 and 2 illustrate an important distinction.
- On the one hand, we demonstrated at most a square or polynomial difference between the time complexity of problems measured on deterministic single tape and multi-tape Turing machines.
- On the other hand, we showed at most an exponential difference between the time complexity of the problems on deterministic and non-deterministic Turing machines.
- For our purpose, polynomial difference in running time are considered to be small, whereas exponential differences are considered to be large.
- Polynomial time algorithms are fast enough for many purposes, but exponential time algorithms rarely are useful. (For $n=1000$, n^3 = 1 billion (still manageable number), 2^n is much larger than the number of atoms in the universe.)
- All reasonable deterministic computational models are polynomially equivalent. Any one of them can simulate another with only a polynomial increase in running time.
- From here on we focus on aspects of time complexity theory that are unaffected by polynomial difference in running time. We consider such differences to be insignificant and ignore them.
- ***The Question is*** whether a given problem is polynomial or non-polynomial.
- So we came to an important definition in the complexity theory, **P** class.

The class **P**: definition

• ***Definition***: **P** is the lass of languages that are decidable in polynomial time on a deterministic single tape Turing machine. That is

$$P = \bigcup_k TIME(n^k).$$

• The class **P** plays an important role in our theory and is important because

  • **P** is invariant for all models of computation that are polynomially equivalent to the deterministic single tape TM, and

  • **P** roughly corresponds to the class of problems that are realistically solvable on a computer.

• When we analyze an algorithm to show that it runs in polynomial time, we need to do two things

  • First, give a polynomial upper bound (usually in big-O notation) on the number of stages that the algorithm uses when it runs on input of length *n*.

  • Then, examine the individual stages in the description of the algorithm to be sure that each can be implemented in polynomial time on a reasonable deterministic model.

• When both tasks have been done, we can conclude that it runs in polynomial time because we have demonstrated that it runs for a polynomial number of stages, each of which can be done in polynomial time, and the composition of polynomials is a polynomial.

Examples of problems in **P**

- We had: the problem whether *w* is a member of the language $A = \{0^k 1^k : k \geq 0\}$ is in **P.**
- Fortunately, there are many problems that are in **P.**
- The *PATH* problem is to determine whether a directed path exists from *s* to *t.*

$$PATH(G,s,t) = \{< G,s,t >: G \text{ is a directed graph that has a directed path from } s \text{ to } t\}.$$

Theorem: $PATH \in P.$

- we use *breadth first search* and successively mark all nodes in *G* that are reachable from *s* by directed paths of length 1, then 2, then 3, through $m=|V|$.

M = "on <*G*,*s*,*t*>: where *G* is a directed graph with nodes *s* and *t.*

1. Place a mark on node *s.*
2. Repeat the following until no additional nodes get marked.
3. Scan all the edges of *G*. If an edge *(a,b)* is found going from marked node *a* to an unmarked node *b*, mark *b.*
4. If *t* is marked, *accept*; otherwise *reject*."

- ***Stages 1 ,4*** are executed only once. **Stage 3** runs at most $m=|V|$ times because each time except the last it marks an additional node in *G.* Hence, the **total number of stages** is *1+1+m,* giving a polynomial in the size of G.
- ***Stages 1,4*** easily implemented in polynomial time on any reasonable deterministic model. **Stage 3** involves a scan of the input and a test whether certain nodes are marked, which also is easily implemented in polynomial time.
- Hence, *M* is a polynomial time algorithm for *PATH.*