




POLITECNICO
DI TORINO

Recursion (cont'ed)

Tecniche di Programmazione – A.A. 2014/2015



Summary

1. Definition and divide-and-conquer strategies
2. Simple recursive algorithms
 1. Fibonacci numbers
 2. Dicothomic search
 3. X-Expansion
 4. Proposed exercises
3. Recursive vs Iterative strategies
4. More complex examples of recursive algorithms
 1. Knight's Tour
 2. Proposed exercises

Recursion

▶ **Divide et Impera**

- ▶ Split a problem \mathcal{P} into $\{ \mathcal{Z}_i \}$ where \mathcal{Z}_i are still complex, yet *simpler* instances of the same problem.
- ▶ Solve $\{ \mathcal{Z}_i \}$, then merge the solutions
- ▶ Merge & split must be “simple”
- ▶ A.k.a., *Divide n' Conquer*

▶ **Exploration**

- ▶ Systematic procedure to enumerate all possible solutions
- ▶ Solutions \leftrightarrow Paths
- ▶ Similar to D+I with $\{ \mathcal{S}, \mathcal{P}' \}$

▶ 3

Tecniche di programmazione A.A. 2014/2015

Complessità



Tecniche di programmazione A.A. 2014/2015

Divide et Impera

- ▶ Solution = Solve (Problem) ;
- ▶ **Solve** (Problem) {
 - ▶ List<SubProblem> subProblems = **Divide** (Problem) ;
 - ▶ For (each subP[i] in subProblems) {
 - ▶ SubSolution[i] = **Solve** (subP[i]) ;
 - ▶ }
 - ▶ Solution = **Combine** (SubSolution[1..N]) ;
 - ▶ return Solution ;
- ▶ }

▶ 5

Tecniche di programmazione A.A. 2014/2015

Divide et Impera – Divide and Conquer

- ▶ Solution = Solve (Problem) ;
 - ▶ **Solve** (Problem) {
 - ▶ List<SubProblem> subProblems = **Divide** (Problem) ;
 - ▶ For (each subP[i] in subProblems) {
 - ▶ SubSolution[i] = **Solve** (subP[i]) ;
 - ▶ }
 - ▶ Solution = **Combine** (SubSolution[1..N]) ;
 - ▶ return Solution ;
 - ▶ }
- recursive call
- “a” sub-problems, each “b” times smaller than the initial problem

▶ 6

Tecniche di programmazione A.A. 2014/2015

Divide et Impera – Divide and Conquer

```

▶ Solve ( Problem ) {
  ▶ if( problem is trivial )
    ▶ Solution = Solve_trivial ( Problem ) ;
  ▶ else {
    ▶ List<SubProblem> subProblems = Divide ( Problem ) ;
    ▶ For ( each subP[i] in subProblems ) {
      □ SubSolution[i] = Solve ( subP[i] ) ;
    }
    ▶ Solution = Combine ( SubSolution[ 1..N ] ) ;
  }
  ▶ return Solution ;
}

```

do recursion

▶ 7

Tecniche di programmazione A.A. 2014/2015

What about complexity?

- ▶ a = number of sub-problems for a problem
- ▶ b = how smaller sub-problems are than the original one
- ▶ n = size of the original problem
- ▶ $T(n)$ = complexity of **Solve**
 - ▶ ...our unknown complexity function
- ▶ $\Theta(1)$ = complexity of **Solve_trivial**
 - ▶ ...otherwise it wouldn't be trivial
- ▶ $D(n)$ = complexity of **Divide**
- ▶ $C(n)$ = complexity of **Combine**

▶ 8

Tecniche di programmazione A.A. 2014/2015

What about complexity?

- ▶ $\Theta(I)$ = complexity of **Solve_trivial**
 - ▶ ...otherwise it wouldn't be trivial
- ▶ $D(n)$ = complexity of **Divide**
- ▶ $C(n)$ = complexity of **Combine**

- ▶ $T(n) = \Theta(n \log n)$

▶ 9

Tecniche di programmazione A.A. 2014/2015

Exploration

- ▶ **Explore** () {
 - ▶ List<Step> steps = **PossibleSteps** (Problem) ;
 - ▶ For (each s in steps) {
 - ▶ **Do** (s)
 - ▶ **Explore** () ;
 - ▶ **Undo** (s)
 - ▶ }
- ▶ }

▶ 10

Tecniche di programmazione A.A. 2014/2015

Exploration

```

▶ Explore ( ) {
  ▶ List<Step> steps = PossibleSteps ( Problem ) ;
  ▶ For ( each s in steps ) {
    ▶ Do ( s )
    ▶ Explore ( ) ;
    ▶ Undo ( s )
  ▶ }
▶ }

```

Local variable

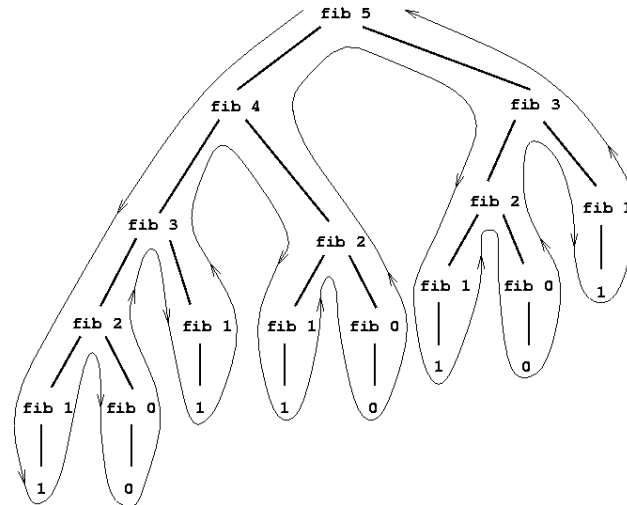
Update "global" status

Backtrack!

What about complexity?

- ▶ (Almost always)
- ▶ $T(n) = \Theta(e^n)$

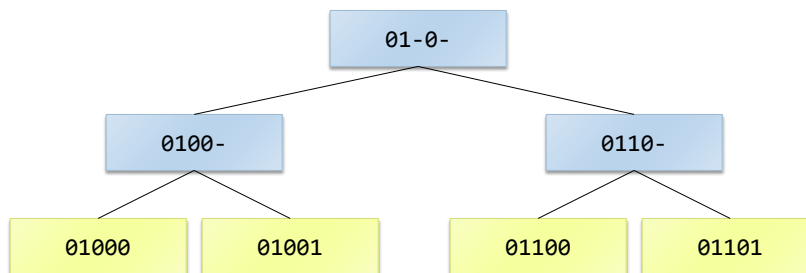
Recursion Tree (exploration)



▶ 13

Tecniche di programmazione A.A. 2014/2015

Don't care expansion



▶ 14

Tecniche di programmazione A.A. 2014/2015

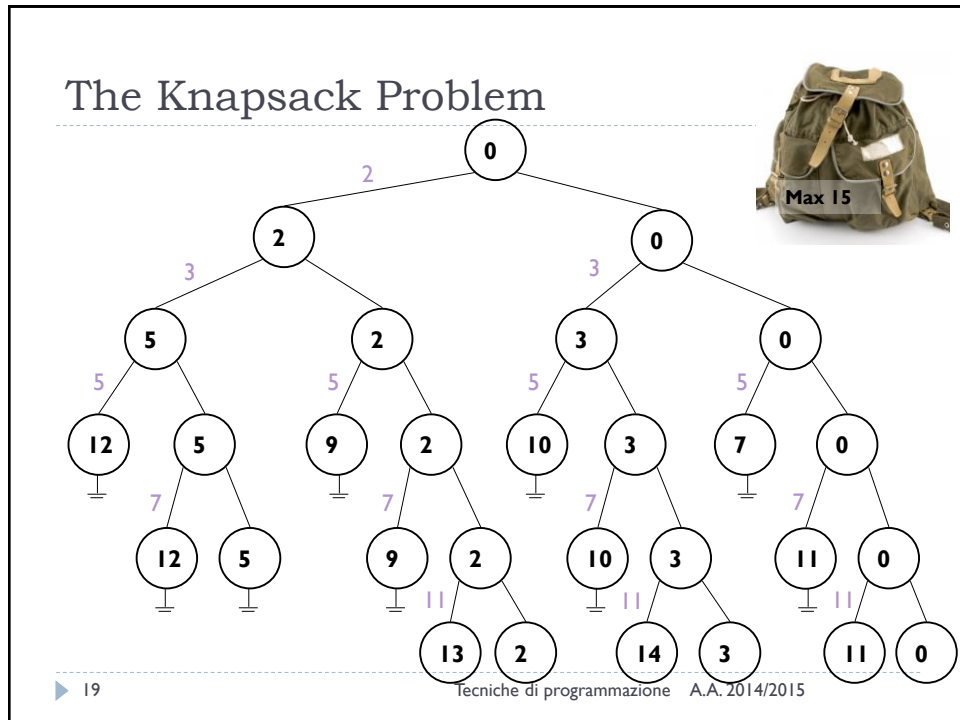


Recursive vs Iterative strategies

Recursion

Recursion and iteration

- ▶ Every **recursive** program can **always** be implemented in an **iterative** manner
- ▶ The best solution, in terms of efficiency and code clarity, depends on the problem



Licenza d'uso

- ▶ Queste diapositive sono distribuite con licenza Creative Commons "Attribuzione - Non commerciale - Condividi allo stesso modo (CC BY-NC-SA)"
- ▶ Sei libero:
 - ▶ di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera
 - ▶ di modificare quest'opera
- ▶ Alle seguenti condizioni:
 - ▶ **Attribuzione** — Devi attribuire la paternità dell'opera agli autori originali e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera.
 - ▶ **Non commerciale** — Non puoi usare quest'opera per fini commerciali.
 - ▶ **Condividi allo stesso modo** — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.
- ▶ <http://creativecommons.org/licenses/by-nc-sa/3.0/>

20

Tecniche di programmazione A.A. 2014/2015