

## 辅学 lesson5

Zhejiang University, Advanced Data Structure and Algorithm Analysis

赵一帆

2025 年 12 月 27 日



# 目录

P and NP

Approximation Algorithm

Local Search

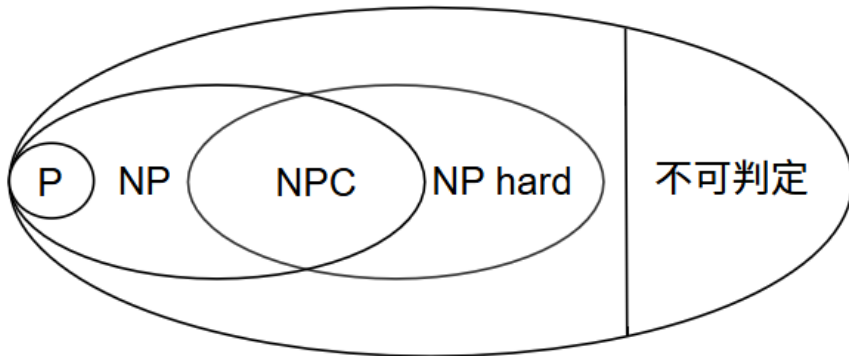
Randomized Algorithm

Others



# P and NP

What should we know?



decision problem.

# P and NP

- Halting Problem.
- When Turing Machine get a input(string), it must go into one of three situations: loop, halt and accept, halt and reject.
- Every Turing Machine can be encoded as a string.
- We assume there is a Turing Machine  $T_1$  can decide whether a Turing Machine  $M$  getting itself as input can halt
  - If  $M$  halts and accepts itself or halts and rejects itself,  $T_1$  halts and accepts  $M$
  - If  $M$  loops,  $T_1$  halts and rejects  $M$
- Then we can construct a Turing Machine  $T_2$ , when it get a input  $M$ 
  - Run  $T_1$  on  $M$ , if  $T_1$  halts and accepts  $M$ , then  $T_2$  loops forever.
  - If  $T_1$  halts and rejects  $M$ , then  $T_2$  halts and accepts  $M$
- What if  $T_2$  get itself as input?

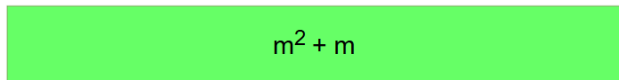
# Approximation Algorithm

- What should we know?
- Bin packing: NF(2), FF(1.7), BF(1.7).
  - Offline Bin packing can not be  $\alpha$ -approximation for any  $\alpha < \frac{3}{2}$  unless  $P \neq NP$
  - Online bin packing problem can not be  $\alpha$ -approximation for any  $\alpha < \frac{5}{3}$  unless  $P \neq NP$
- Knapsack: 2-approximation, FPTAS
- K-center problem: 2-approx, two different versions(in metric graph and in Euclidean space).
  - If distances are unbounded, no approximation algorithm exists.
  - If we can only select given nodes in metric graph, 2-approximation algorithm can not be improved.
  - If we can select any nodes in 2-dimension Euclidean space, 2-approximation algorithm is known and no 1.82-approximation algorithm exists.
- TSP tour: 2-approximation, 1.5-approximation
- PTAS, EPTAS, FPTAS.

# Online Problem

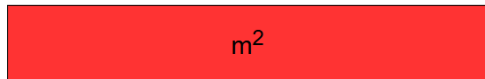
Online subset sum problem.

Subset sum is a special case of knapsack problem that the size and the value of a item are always the same.


$$m^2 + m$$

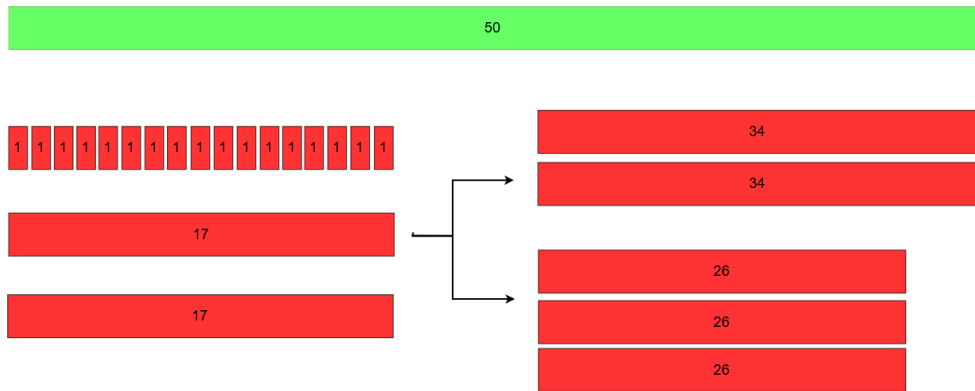

$$1$$


$$m$$


$$m^2$$

# Online Bin packing

Online Bin packing problem.



# Local Search

## 判断题

We are given a set of sites  $S = \{s_1, s_2, \dots, s_k\}$  in the plane, and we want to choose a set of  $k$  centers  $C = \{c_1, c_2, \dots, c_k\}$  so that the maximum distance from a site to the nearest center is minimized. Here  $c_i$  can be an arbitrary point in the plane. A local search algorithm arbitrarily choose  $k$  points in the plane to be the centers, then (1) divide  $S$  into  $k$  sets, where  $S_i$  is the set of all sites for which  $c_i$  is the nearest center; and (2) for each  $S_i$ , compute the central position as a new center for all the sites in  $S_i$ . If steps (1) and (2) cause the covering radius to strictly decrease, we perform another iteration, otherwise the algorithm stops. When the above local search algorithm terminates, the covering radius of its solution is at most 2 times the optimal covering radius.



# Local Search

## 判断题

We are given a set of sites  $S = \{s_1, s_2, \dots, s_k\}$  in the plane, and we want to choose a set of  $k$  centers  $C = \{c_1, c_2, \dots, c_k\}$  so that the maximum distance from a site to the nearest center is minimized. Here  $c_i$  can be an arbitrary point in the plane. A local search algorithm arbitrarily choose  $k$  points in the plane to be the centers, then (1) divide  $S$  into  $k$  sets, where  $S_i$  is the set of all sites for which  $c_i$  is the nearest center; and (2) for each  $S_i$ , compute the central position as a new center for all the sites in  $S_i$ . If steps (1) and (2) cause the covering radius to strictly decrease, we perform another iteration, otherwise the algorithm stops. When the above local search algorithm terminates, the covering radius of its solution is at most 2 times the optimal covering radius.

We can give a counterexample.

# Local Search

## 单选题

- (1) Max Leaf Spanning Tree: find a spanning tree  $T \in F$  with a maximum number of leaves.
  - (2) Minimum Spanning Tree: find a spanning tree  $T \in F$  with a minimum total weight of all the edges in  $T$ .
  - (3) Minimum Degree Spanning Tree: find a spanning tree  $T \in F$  such that its maximum degree of all the vertices is the smallest.
- A. The local search always return an optimal solution for Max Leaf Spanning Tree.
  - B. The local search always return an optimal solution for Minimum Spanning Tree.
  - C. The local search always return an optimal solution for Minimum Degree Spanning Tree.
  - D. For neither of the problems that this local search always return an optimal solution.

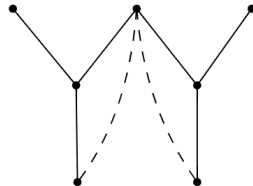
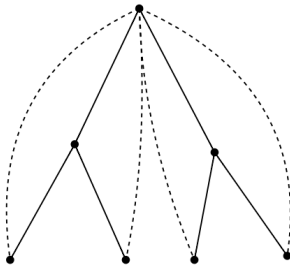
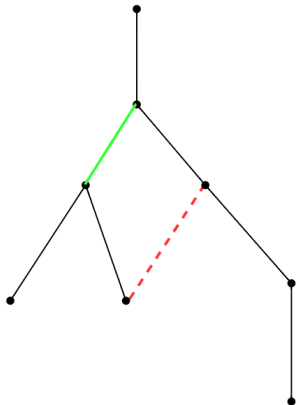


# Local Search

## 单选题

- (1) Max Leaf Spanning Tree: find a spanning tree  $T \in F$  with a maximum number of leaves.
  - (2) Minimum Spanning Tree: find a spanning tree  $T \in F$  with a minimum total weight of all the edges in  $T$ .
  - (3) Minimum Degree Spanning Tree: find a spanning tree  $T \in F$  such that its maximum degree of all the vertices is the smallest.
- A. The local search always return an optimal solution for Max Leaf Spanning Tree.
  - B. The local search always return an optimal solution for Minimum Spanning Tree.
  - C. The local search always return an optimal solution for Minimum Degree Spanning Tree.
  - D. For neither of the problems that this local search always return an optimal solution.

# Local Search



# Local Search

Or you can solve this problem from the NP complexity.

- Minimum Spanning tree is P problem, because we can solve it using Prim or Kruskal algorithm.
- Max Leaf Spanning Tree is NP problem, because we can use reduction chain Vertex Cover  $\rightarrow$  Connected Dominating Set Problem  $\rightarrow$  Max Leaf Spanning Tree to prove its NP-complete.
- Minimum Degree Spanning Tree is also NP problem, because we can use reduction Hamilton Path  $\rightarrow$  Minimum Degree Spanning Tree to prove its NP-complete.

If we can use local search to solve Max Leaf Spanning Tree or Minimum Degree Spanning Tree, we can solve all NP problem in polynomial time, which indicating  $P = NP$ .

# Local Search

## 单选题

Max-cut problem: Given an undirected graph  $G = (V, E)$  with positive integer edge weights  $w_e$ , find a node partition  $(A, B)$  such that  $w(A, B)$ , the total weight of edges crossing the cut, is maximized. Let us define  $S'$  be the neighbor of  $S$  such that  $S'$ , can be obtained from  $S$  by moving one node from  $A$  to  $B$ , or one from  $B$  to  $A$ . We only choose a node which, when flipped, increases the cut value by at least  $\frac{w(A, B)}{|V|}$ . Then which of the following is true?

- A. Upon the termination of the algorithm, the algorithm returns a cut  $(A, B)$  so that  $2.5w(A, B) \geq w(A^*, B^*)$  where  $(A^*, B^*)$  is an optimal partition.
- B. The algorithm terminates after at most  $O(\log |V| \log W)$  flips, where  $W$  is the total weight of edges.
- C. Upon the termination of the algorithm, the algorithm returns a cut  $(A, B)$  so that  $2s(A, B) \geq w(A^*, B^*)$ .
- D. The algorithm terminates after at most  $O(|V|^2)$  flips.

# Local Search

## 单选题

Max-cut problem: Given an undirected graph  $G = (V, E)$  with positive integer edge weights  $w_e$ , find a node partition  $(A, B)$  such that  $w(A, B)$ , the total weight of edges crossing the cut, is maximized. Let us define  $S'$  be the neighbor of  $S$  such that  $S'$ , can be obtained from  $S$  by moving one node from  $A$  to  $B$ , or one from  $B$  to  $A$ . We only choose a node which, when flipped, increases the cut value by at least  $\frac{w(A, B)}{|V|}$ . Then which of the following is true?

- A. Upon the termination of the algorithm, the algorithm returns a cut  $(A, B)$  so that  $2.5w(A, B) \geq w(A^*, B^*)$  where  $(A^*, B^*)$  is an optimal partition.
- B. The algorithm terminates after at most  $O(\log |V| \log W)$  flips, where  $W$  is the total weight of edges.
- C. Upon the termination of the algorithm, the algorithm returns a cut  $(A, B)$  so that  $2s(A, B) \geq w(A^*, B^*)$ .
- D. The algorithm terminates after at most  $O(|V|^2)$  flips.

# Local Search

- May NOT in polynomial time

↓ stop the algorithm when there are no "*big enough*" improvements.

**Big-improvement-flip:** Only choose a node which, when flipped, increases the cut value by at least

$$\frac{2\varepsilon}{|V|} w(A, B)$$

**Claim:** Upon termination, the big-improvement-flip algorithm returns a cut  $(A, B)$  so that

$$(2 + \varepsilon) w(A, B) \geq w(A^*, B^*)$$

**Claim:** The big-improvement-flip algorithm terminates after at most  $O(n/\varepsilon \log W)$  flips.





# Local Search

- $\forall u \in A, \sum_{v \in A} w_{uv} < \sum_{v \in B} w_{uv} + \frac{2\epsilon}{|V|} w(A, B)$
- $2 \sum_{u \in A} \sum_{v \in A} w_{uv} < \sum_{u \in A} \sum_{v \in B} w_{uv} + |A| \frac{2\epsilon}{|V|} w(A, B)$
- $2 \sum_{u \in B} \sum_{v \in B} w_{uv} < \sum_{u \in B} \sum_{v \in A} w_{uv} + |B| \frac{2\epsilon}{|V|} w(A, B)$
- $\sum_{u \in A} \sum_{v \in A} w_{uv} + \sum_{u \in B} \sum_{v \in B} w_{uv} < \sum_{u \in B} \sum_{v \in A} w_{uv} + \epsilon w(A, B) = (1 + \epsilon) w(A, B)$
- $w(A^*, B^*) \leq \sum_{u \in A} \sum_{v \in A} w_{uv} + \sum_{u \in B} \sum_{v \in B} w_{uv} + w(A, B) < (2 + \epsilon) w(A, B)$
- $\log_{1 + \frac{2\epsilon}{|V|}} W = \frac{\log W}{\log(1 + \frac{2\epsilon}{|V|})} \leq \frac{|V| \log W}{2\epsilon} = O\left(\frac{n \log W}{\epsilon}\right)$



# Local Search

## 多选题

Recall the on-line Hiring Problem we introduced in class. Consider the following algorithm for this problem.

For  $n$  interviewees with distinct ratings, we always set an integer  $k$  with  $0 \leq k < n$ , and reject first  $k$  interviewees we encounter. In the remaining interviewees, we hire the first one we encounter with better rating than the best of the first  $k$  ones encountered. If none of the remaining interviewees is better than the best of the first  $k$  ones encountered, the last one encountered is hired.

Assume that the ratings of interviewees are uniformly random. Among the following statements about this algorithm, which is/are correct?

- A. This algorithm hires the best of all the  $n$  interviewees if and only if this interviewee is not one of the first  $k$  ones encountered.
- B. This algorithm hires the last interviewee encountered if and only if the best of all the  $n$  interviewees is in the first  $k$  ones encountered.
- C. If  $k = 1$ , the probability of hiring the best of all the  $n$  interviewees is  $1/n$ .
- D. If  $k = n - 1$ , the probability of hiring the best of all the  $n$  interviewees is  $1/n$ .



# Local Search

## 多选题

Recall the on-line Hiring Problem we introduced in class. Consider the following algorithm for this problem.

**For  $n$  interviewees with distinct ratings, we always set an integer  $k$  with  $0 \leq k < n$ , and reject first  $k$  interviewees we encounter. In the remaining interviewees, we hire the first one we encounter with better rating than the best of the first  $k$  ones encountered. If none of the remaining interviewees is better than the best of the first  $k$  ones encountered, the last one encountered is hired.**

Assume that the ratings of interviewees are uniformly random. Among the following statements about this algorithm, which is/are correct?

- A. This algorithm hires the best of all the  $n$  interviewees if and only if this interviewee is not one of the first  $k$  ones encountered.
- B. This algorithm hires the last interviewee encountered if and only if the best of all the  $n$  interviewees is in the first  $k$  ones encountered.
- C. If  $k = 1$ , the probability of hiring the best of all the  $n$  interviewees is  $1/n$ .
- D. If  $k = n - 1$ , the probability of hiring the best of all the  $n$  interviewees is  $1/n$ .

## 单选题

Consider the Minimum Degree Spanning Tree problem: Given a connected undirected graph  $G(V, E)$ , find a spanning tree  $T$  whose maximum degree over its vertices is minimized over all spanning trees of  $G$ . The problem can be shown to be NP-hard by reduction from the Hamiltonian Path Problem. On the other hand, we can use local search to design approximating algorithms. Denote  $d(u)$  as the degree of vertex  $u$  on a tree  $T$ . Consider the following algorithm: (1) Find an arbitrary spanning tree  $T$  of  $G$ . (2) If there's some edge  $e \in E(G) \setminus E(T)$  with endpoints  $u, v$ , and there's some other vertex  $w$  on the path between  $u, v$  on  $T$  such that  $\max\{d(u), d(v)\} + 1 < d(w)$ , then we replace an edge  $e'$  incident to  $w$  on  $T$  with  $e$ , i.e.  $T := T \cup \{e\} \setminus \{e'\}$ . (3) Repeat Step (2) until there's no edge to replace. It can be shown that this algorithm will terminate at a solution with maximum vertex degree  $OPT + O(\log |V|)$ . To show the algorithm will terminate in finite steps, a useful technique is to define a nonnegative potential function  $\phi(T)$  and to show  $\phi(T)$  is strictly decreasing after each step. Which of the following potential functions below satisfies the above requirements?

A.  $\phi(T) = \sum_{v \in V} d(v)$ . B.  $\phi(T) = \sum_{u, v \in E(T)} \max\{d(u), d(v)\}$ .

C.  $\phi(T) = \sum_{u \in V} \sum_{v \in V, v \neq u} \sum_{w \in V, w \neq u, v} \max\{d(u), d(v), d(w)\}$ . D.  $\phi(T) = \sum_{v \in V} 3^{d(v)}$

## 单选题

Consider the Minimum Degree Spanning Tree problem: Given a connected undirected graph  $G(V, E)$ , find a spanning tree  $T$  whose maximum degree over its vertices is minimized over all spanning trees of  $G$ . The problem can be shown to be NP-hard by reduction from the Hamiltonian Path Problem. On the other hand, we can use local search to design approximating algorithms. Denote  $d(u)$  as the degree of vertex  $u$  on a tree  $T$ . Consider the following algorithm: (1) Find an arbitrary spanning tree  $T$  of  $G$ . (2) If there's some edge  $e \in E(G) \setminus E(T)$  with endpoints  $u, v$ , and there's some other vertex  $w$  on the path between  $u, v$  on  $T$  such that  $\max\{d(u), d(v)\} + 1 < d(w)$ , then we replace an edge  $e'$  incident to  $w$  on  $T$  with  $e$ , i.e.  $T := T \cup \{e\} \setminus \{e'\}$ . (3) Repeat Step (2) until there's no edge to replace. It can be shown that this algorithm will terminate at a solution with maximum vertex degree  $OPT + O(\log |V|)$ . To show the algorithm will terminate in finite steps, a useful technique is to define a nonnegative potential function  $\phi(T)$  and to show  $\phi(T)$  is strictly decreasing after each step. Which of the following potential functions below satisfies the above requirements?

$$\text{A. } \phi(T) = \sum_{v \in V} d(v). \quad \text{B. } \phi(T) = \sum_{u, v \in E(T)} \max\{d(u), d(v)\}.$$

$$\text{C. } \phi(T) = \sum_{u \in V} \sum_{v \in V, v \neq u} \sum_{w \in V, w \neq u, v} \max\{d(u), d(v), d(w)\}. \quad \text{D. } \phi(T) = \sum_{v \in V} 3^{d(v)}$$

# Local Search

- What should we know?
- LOCAL, SEARCH, always a feasible solution
- Hopfield Neural Networks
- Maximum Cut Problem(Upper bound 1.1382, lower bound 1.0625)
- State-flipping Algorithm
  - For Hopfield Neural Networks, no approximation-ratio is guaranteed, but a stable configuration must be found.
  - For Maximum Cut Problem, it is 2-approx.
- Big-improvement-flip Algorithm
  - It is only for Maximum Cut Problem,  $(2 + \epsilon)$ -approx.
  - Its running time is polynomials.
- Simulated Annealing Algorithm, K-L heuristic Algorithm.

# Randomized Algorithm

## 判断题

Let  $a = \{a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_n\}$  denote the list of elements we want to sort. In the quicksort algorithm, if the pivot is selected uniformly at random. Then any two elements get compared at most once and the probability of  $a_i$  and  $a_j$  being compared is  $\frac{2}{j-i+1}$  for  $j > i$ , given that  $a_i$  or  $a_j$  is selected as the pivot.

# Randomized Algorithm

## 判断题

Let  $a = \{a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_n\}$  denote the list of elements we want to sort. In the quicksort algorithm, if the pivot is selected uniformly at random. Then any two elements get compared at most once and the probability of  $a_i$  and  $a_j$  being compared is  $\frac{2}{j-i+1}$  for  $j > i$ , given that  $a_i$  or  $a_j$  is selected as the pivot.

If array  $a$  is sorted, then it is a conclusion.



# Randomized Algorithm

## 判断题

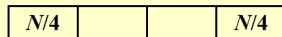
Let  $a = \{a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_n\}$  denote the list of elements we want to sort. In the quicksort algorithm, if the pivot is selected uniformly at random. Then any two elements get compared at most once and the probability of  $a_i$  and  $a_j$  being compared is  $\frac{2}{j-i+1}$  for  $j > i$ , given that  $a_i$  or  $a_j$  is selected as the pivot.

If array  $a$  is sorted, then it is a conclusion.

$$\sum_{i=1}^n \sum_{j>i} \frac{2}{j-i+1} = O(n \log n)$$

# Randomized Algorithm

**Claim:** The expected number of iterations needed until we find a central splitter is at most 2.



$N/2$  central splitters

$$\Pr[\text{find a central splitter}] = 1/2 \quad \checkmark$$

**Type  $j$ :** the subproblem  $S$  is of **type  $j$**  if  $N\left(\frac{3}{4}\right)^{j+1} \leq |S| \leq N\left(\frac{3}{4}\right)^j$

**Claim:** There are at most  $\left(\frac{4}{3}\right)^{j+1}$  subproblems of **type  $j$** .

$$E[T_{\text{type } j}] = O\left(N\left(\frac{3}{4}\right)^j\right) \times \left(\frac{4}{3}\right)^{j+1} = O(N)$$

$$\text{Number of different types} = \log_{4/3} N = O(\log N)$$

$\left. \begin{array}{l} E[T_{\text{type } j}] = O(N) \\ \text{Number of different types} = O(\log N) \end{array} \right\} O(N \log N)$

- Type  $j$  is a definition of a subproblem according to its length.
- The expected process time of a type is linear of the sum of elements in these subproblems of this type.



# Randomized Algorithm

## 判断题

Reviewing the randomized QuickSort in our course, we always select a central splitter as a pivot before recursions, make sure that each side contains at least  $n/4$  elements. Hence, differing from the deterministic QuickSort, the worst case expected running time of the randomized QuickSort is  $\Theta(n \log n)$ .



# Randomized Algorithm

## 判断题

Reviewing the randomized QuickSort in our course, we always select a central splitter as a pivot before recursions, make sure that each side contains at least  $n/4$  elements. Hence, differing from the deterministic QuickSort, the worst case expected running time of the randomized QuickSort is  $\Theta(n \log n)$ .

For any instance of Quick Sort, the expected running time is  $\Theta(n \log n)$



# Randomized Problem

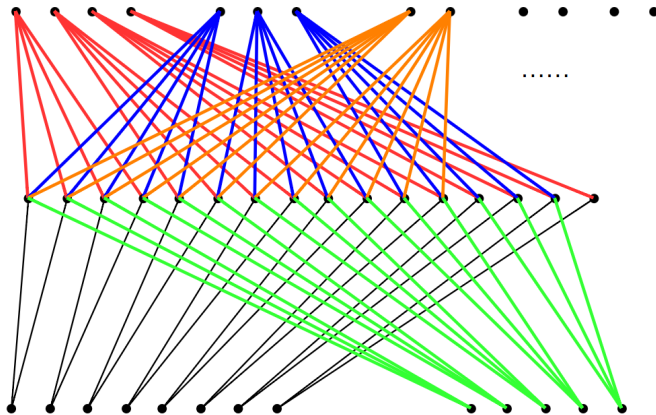
- What should we know?
- Monte Carlo algorithms: must terminate in polynomial time but may be wrong.
- Las Vegas algorithms: must get correct answer but its running time may be not polynomials.
- Hiring Algorithm
- Quicksort: Type j. Las Vegas algorithm.

# Problems

- Knapsack Problem 背包问题 FPTAS
- Bin Packing Problem 装箱问题 1.5-approximation
- Vertex Cover Problem 顶点覆盖问题 2-approximation
- Dominating Set Problem 支配集问题  $O(\log n)$ -approximation
- Maximum Independent Set Problem 最大独立集问题  $O(n)$ -approximation
- Maximum Clique Problem 最大团问题  $O(n)$ -approximation



# Vertex Cover



Degree greedy can be as bad as  $O(\log n)$ -approx.

# Dynamic programming

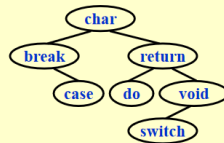
$$c_{ij} = \min_{i < l \leq j} \{w_{ij} + c_{i,l-1} + c_{l+1,j}\}$$

Dynamic Programming

word	break	case	char	do	return	switch	void
probability	0.22	0.18	0.20	0.05	0.25	0.02	0.08

break.. break	case..case	char.. char	do..do	return..return	switch..switch	void.. void
0.22   break	0.18   <b>case</b>	0.20   char	0.05   <b>do</b>	0.25   return	0.02   <b>switch</b>	0.08   void
break.. case	case.. char	char..do	do.. return	return..switch	switch.. void	
0.58   <b>break</b>	0.56   char	0.30   char	0.35   return	0.29   return	0.12   <b>void</b>	
break.. char	case..do	char.. return	do.. switch	return.. void		
1.02   case	0.66   char	0.80   return	0.39   return	0.47   return		
break..do	case.. return	char.. switch	do.. void			
1.17   case	1.21   char	0.84   return	0.57   <b>return</b>			
break.. return	case.. switch	char.. void				
1.83   char	1.27   char	1.02   return				
break.. switch	case.. void					
1.89   char	1.53   char					
break.. void						
2.15   <b>char</b>						

$$T(N) = O(N^3)$$



Please read 10.33 on p.419 for an  $O(N^2)$  algorithm.



# Dynamic programming

- How can we optimize the time complexity to  $O(N^2)$ ?
- previous method
  - first loop, the number of different span length  $k$  is  $O(N)$
  - second loop, the number of subproblems of fixed  $k$  is  $O(N)$
  - recursive function, the time of updating one entry is  $O(N)$
- Can we optimize one of them to  $O(1)$ ?
- Actually we can combine the latter two steps, and optimize them to  $O(N)$  as a whole.
- $root(i, j) \leq root(i, j+1)$  (Donald Ervin Knuth, Turing Award in 1974)
- So is we only need to seek root of  $(i, j)$  between  $(i, j-1)$  and  $(i+1, j)$ , which can complete the optimization if we view it layer by layer. (A layer indicates a fixed span length  $k$ )