Greedy algorithms and optimization

In an optimization problem we have a cost function c, a (normally very large) set of possible solutions and the goal is to find a solution with the best possible cost.

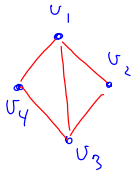
Examples

1. Coin change:
   What is the smallest number of coins and money bills we need to pay a given amount A? (e.g. in the danish system with 'coins' 500, 200, 100, 50, 20, 10, 5, 2, 1)
   Here the cost function c counts one for each coin we use and we want the smallest cost

2. Activity selection
   We are given k activites, each with a start and finish time so $A_i = [s_i, f_i]$ eg [9.00, 12.00]
   We have one room that we can use and the rule is that no two chosen activities may take place at the same time.
   We want to schedule as many of the activities as possible.
   Here c is one for each activity we schedule and we want the largest possible cost.

Some definitions and more problems.

A graph $G=(V,E)$ consists of a set V of vertices and a set $E \subseteq V \times V$ of edges



$V = \{v_1, v_2, v_3, v_4\}$

$E = \{v_1v_2, v_1v_3, v_1v_4, v_2v_3, v_3v_4\}$

A path in a graph $G=(V,E)$ is a sequence

$u_1 e_1 u_2 e_2 u_3 e_3 \ldots, u_{k-1} e_{k-1} u_k$

such that $u_1, u_2, \ldots u_k$ are distinct vertices of G

$e_1, e_2, \ldots e_{k-1}$ are distinct edges of G

and e_i connects u_i to u_{i+1} for $i = 1, 2, \ldots, k-1$

A cycle in G is like a path, but with $u_1 = u_k$

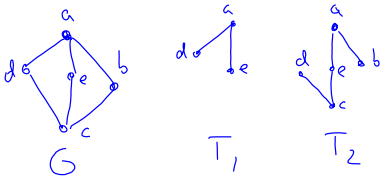
path from u to v

cycle

A graph $G=(V,E)$ is connected if it has a path from u to v for every choice of $u, v \in V$

A subgraph of $G=(V,E)$ is a graph $H=(V',E')$ where $V' \subseteq V$ and $E' \subseteq E$

A tree in a graph G is a connected subgraph $T=(V',E')$ of G with no cycle. It is a spanning tree if $V' = V$.



T_1, T_2 are both trees

T_1 is not spanning while T_2 is spanning

The complete graph K_n on n vertices has n vertices and an edge between any pair of these

Theorem $G=(V,E)$ is connected if and only if G has a spanning tree

proof: exercise

Further optimization problems

③ Longest path in a graph

Input: a graph $G=(V,E)$

Find: a path P with the maximum number of vertices

Here the cost $c(P)$ of a path is its number of vertices and we want to maxime $c(P)$

[very difficult problem, see DM553]

④ Minimum spanning tree (MST) problem

Input: A graph $G=(V,E)$ and $c: E \to \mathbb{R}_+$ so $c(e)>0$ for all $e \in E$

Find: a spanning tree T of minimum cost with respect to c, that is

$c(T) \leq c(T')$

for all spanning trees T' of G

Here $c(T)=\sum_{e \text{ in } T} c(e)$

["easy" problem, see below]

Greedy algorithms

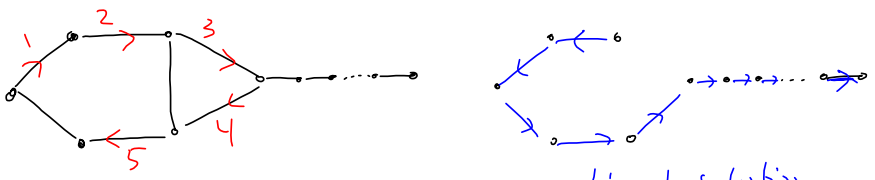
A greedy algorithm G for an optimization problem P constructs a solution by making a number of choices $a_1, a_2, \ldots a_k, \ldots$ such that

1. the k'th choice, a_k only depends on choices, $a_1, a_2, \ldots a_{k-1}$ made so far
2. Given choices $a_1, a_2, \ldots a_{k-1}$ a_k is the best choice according to G's strategy

Examples

③ Longest path problem

G: start in an arbitrary vertex and always take an edge to a vertex, not on the current path



Greedy solution optimal solution

Solution can be arbitrarily bad.

① Coin changing

General problem: we have k denominations of coins

$n_1 > n_2 > n_3 > \ldots > n_{k-1} > n_k = 1$ ⇒ any amount can be paid!

want to pay amount n

G:
1. $W := n$
2. For $i := 1$ to k do
   use $m_i = \lfloor \frac{W}{n_i} \rfloor$ coins of type i
   $W := W - m_i \cdot n_i$

Q: Does G always find an optimal solution?

Consider two coinsystems S_1, S_2 with coins

$S_1 = \{25, 10, 5, 1\}$ $S_2 = \{4, 3, 1\}$

Paying $n = 41$ in S_1 using G

$48 - 25 = 23$ used 1 · (25)
$23 - 2 \cdot 10 = 3$ used 2 · (10)
$3 - 3 \cdot 1 = 0$ used 3 · (1)

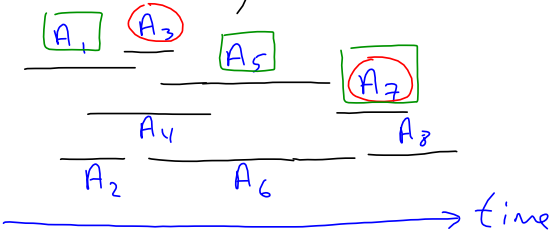
used 6 coins and this is best possible (see below)

Paying 6 in S_2 using G

$6 - 4 = 2$ used 1 · (4)
$2 - 2 \cdot 1 = 0$ used 2 · (1)

used 3 coins. Not optimal as $6 = 3 + 3$

② Activity selection



activities numbered according to finish time

Visually: the maximum # of activities we can schedule is 3, e.g. A_2, A_5, A_8

Consider two greedy strategies

G_1: always take the activity which is non-overlapping with the ones chosen so far and has the shortest duration → A_3, A_7 not optimal

G_2: always take the activity that ends first among those that do not overlap any of the activites chosen so far → A_1, A_5, A_7 optimal

In fact G_2 always finds an optimal solution

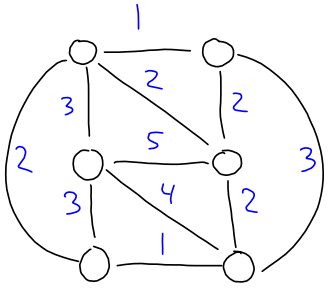
Q: How can we prove this?

See later :)

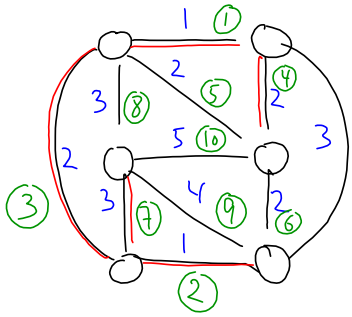
(4) A greedy algorithm for the minimum spanning tree problem in $G=(V,E)$ (connected) with edge cost $c(e)>0$

G:
1. sort the edges according to their cost
$c(e_1) \le c(e_2) \le \dots \le c(e_{m-1}) \le c(e_m)$ $m=|E|$
2. Set $X := \emptyset$
3. For $i := 1$ to m do
If $H=(V, X \cup \{e_i\})$ has no cycle
then $X := X \cup \{e_i\}$
4. Return $T=(V,X)$

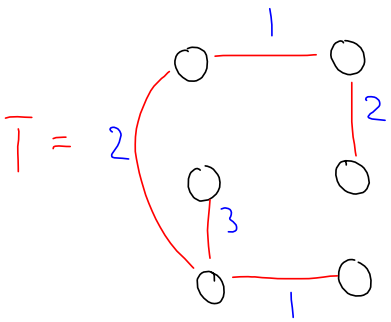
Example



G with costs c



① = i'th edge considered by G



cost of T is $c(T)=9$

Theorem The greedy algorithm G always produces a minimum spanning tree when the input is a connected graph. (DM507)

Properties that must hold for a greedy algorithm G if it should always solve the problem optimally:

a. Greedy choice property:
A globally optimal solution can be obtained by a series of locally optimal (greedy) choices

b. Optimal substructure property:
Every optimal solution contains optimal solutions to its subproblems.
After making a greedy choice a new (smaller problem of the same type arises)

These two are sufficient to guarantee that G produces an optimal solution.

Back to ① coin changing

ad b: coin changing has optimal substructure property, since after reducing the remaining amount to pay from n to $n' < n$ what remains is a new coin changing problem for n' in the same coin system.

If the optimal solution used r_1 coins to go from n to n' and then r_2 coins to pay n', then n' cannot be paid with less than r_2 coins or $r_1 + r_2$ would not be the best possible # of coins to pay n.

ad a: When $S = \{25, 10, 5, 1\}$
We (just) need to show that there is always some optimal solution when we used the greedy choice, that is, as many coins as possible from the largest denomination of a coin we could use
e.g. if $n = 62$ we need to show that some optimal solution uses two 25's

proof that the greedy choice property (gcp) holds for

$S = \{25, 10, 5, 1\}$

assume $n \geq 25$ and that $X = a_1 + a_2 + \dots + a_k$ where $a_1 \geq a_2 \geq \dots \geq a_k$, $a_i \in S$ and k is smallest # of coins to pay n from S.

If $a_1 = \dots = a_r = 25$ and $a_{r+1} < 25$, then let $n' = n - (a_1 + \dots + a_r)$

If $n' < 25$, then $X = a_1 + a_2 + \dots + a_k$ satisfies gcp

So assume $n' \geq 25$ and $n' = a_1' + a_2' + \dots + a_{k'}'$ with $a_j' \in S \setminus \{25\}$

If $a_1', a_2', a_3' = 10$ then $n' = 25 + 5 + a_4' + \dots + a_{k'}'$ is better ↯

so $a_3' < 10$ and now $\exists$ an index $2 < p \leq k'$

$a_1' + a_2' + \dots + a_p' = 25$ and we can get a better

solution $25 + a_{p+1}' + \dots + a_{k'}'$ ↯

Similar proof works for $n < 25$ (exercise)

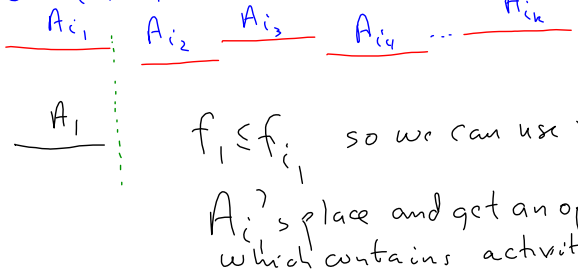
Proof that the greedy algorithm G_2 always finds an optimal solution for activity selection:

Let $A = \{A_1, A_2, \ldots, A_n\}$ be the set of activities numbered so that $f_1 \le f_2 \le \ldots \le f_n$ when $A_i = [s_i, f_i]$ for $i = 1, 2, \ldots, n$.

For a given activity A_j we denote by $\mathcal{A}_j$ the set of activities A_p s.t. $s_p \ge f_j$

G_2 has gcp: We need to show that there is some optimal solution which contains A_1 (the greedy choice)

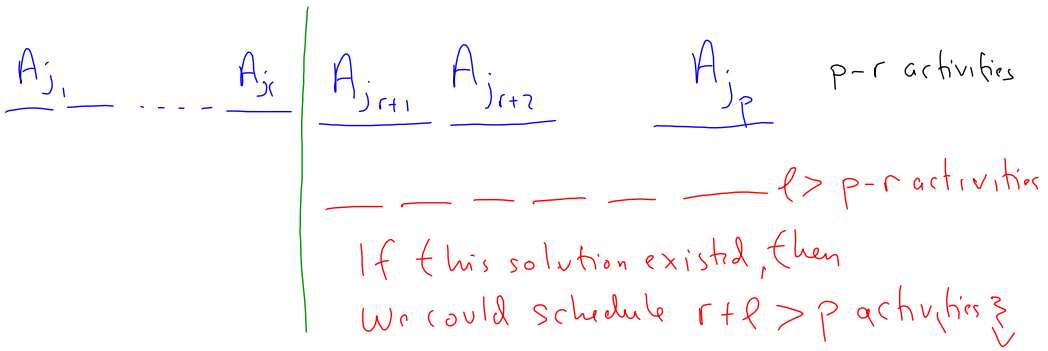
Let $A_{i_1}, A_{i_2}, \ldots, A_{i_k}$ be an opt sol (k activities) so it looks like the red activities below



$f_1 \le f_{i_1}$ so we can use A_1 in A_{i_1}'s place and get an opt. solution which contains activity A_1

G has optimal substructure:

Enough to show that if $A_{j_1}, A_{j_2}, \ldots, A_{j_p}$ is an optimal solution, then for every $1 < r < p$ the activities $A_{j_{r+1}}, A_{j_{r+2}}, \ldots, A_{j_p}$ form an optimal sol to $\mathcal{A}_{j_r}$



If this solution existed, then we could schedule $r + \ell > p$ activities

Further optimization problems.

⑤ Travelling Salesman Problem (TSP)

Input : A complete graph K_n with a cost function C on its edges

Find : A cycle on n vertices of minimum cost with respect to C

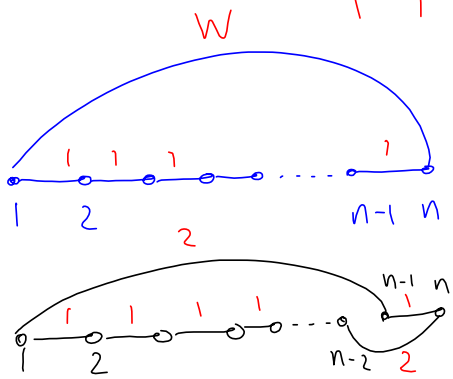
[Very difficult (DM553)]

Greedy strategy: 1. number the vertices $1, 2, \ldots, n$
2. start in 1 and always go to the the closest vertex not included yet

Example Suppose cost function is

$$c_{ij} = \begin{cases} W & \text{if } i=1 \text{ and } j=n \\ 2 & \text{if } j \neq i+1 \\ 1 & \text{if } j = i+1 \end{cases}$$

W large



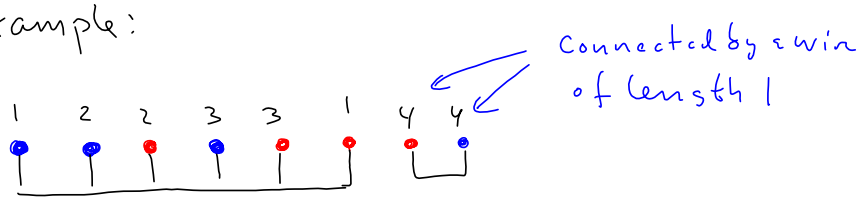
greedy solution
cost $(n-1)+W$

cost $n+2$

Conclusion: Greedy alg. can be arbitrarily bad as W can be anything.

(6) Connecting Wires

- There are n blue and n red dots, equally spaced in a line
- Goal: connect each blue dot to a red one so that no two red (blue) dots are connected to the same blue (red) dot and we use the minimum amount of wire.
- Example:

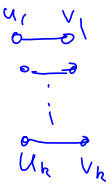


total wire length $5+1+1+1=8$

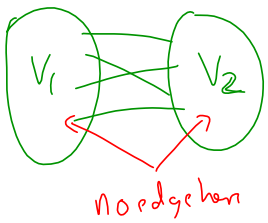
Suggest greedy strategies for solving this.
Will they work?

(6) is a special case of the following graph problem.

Notation: A matching in a graph $G=(V,E)$ is a collection of edges $u_1v_1, u_2v_2, \ldots u_kv_k$ that share no vertices:



A graph $G=(V,E)$ is bipartite if we can split V into two sets V_1, V_2 s.t all edges of G have one end in V_1 and the other in V_2:



A complete bipartite graph is a bipartite graph $G=(V_1 \cup V_2, E)$ s.t $\forall u \in V_1, v \in V_2 : uv \in E$.

A perfect matching in a graph $G=(V,E)$ is a matching that covers all vertices of G (so $|V|$ is even if there is a perfect matching)

(7) Minimum cost perfect matching in bipartite graphs.

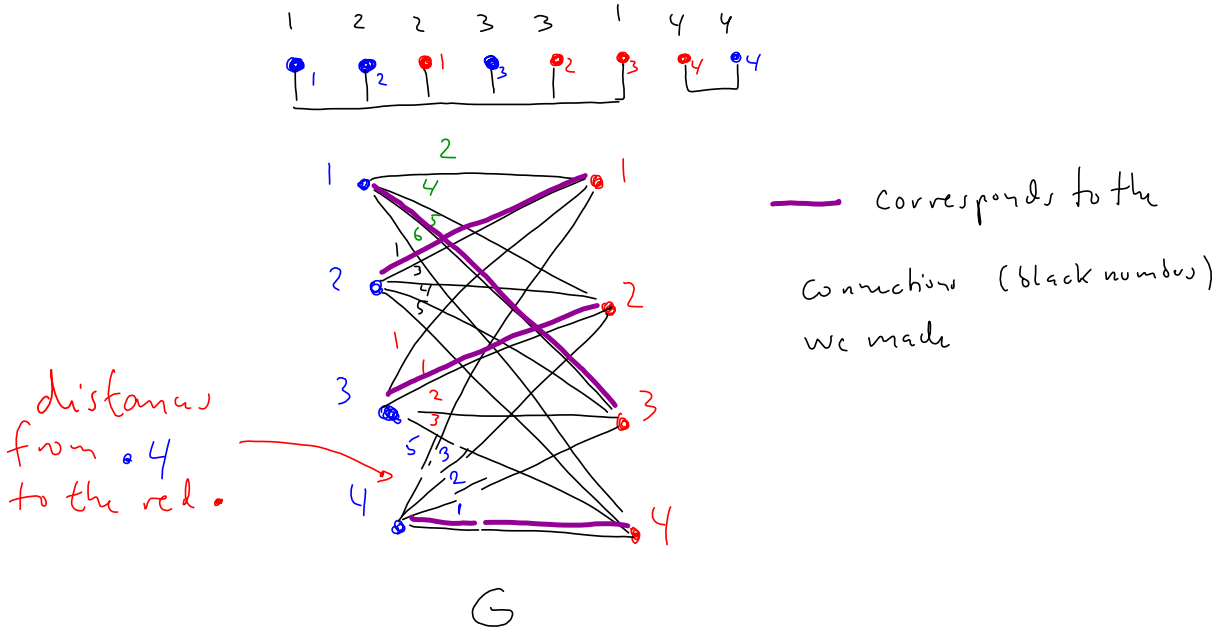
Input: A complete bipartite graph $G=(V_1,V_2,E)$ with $|V_1|=|V_2|=k$ and a cost function c on E

Find a perfect matching $\{e_1,e_2,\dots,e_k\}=M$ s.t. $c(M)=\sum_{i=1}^{k} c(e_i)$ is minimized, that is, $c(M)\le c(M')$ for all perfect matchings M'

See DM817 for how to solve this

Claim: (6) is a special case of (7)

recall example



So each solution S to the wire problem corresponds to a perfect matching in G whose total cost is the wirelength used in S.

Conversely, every perfect matching in G corresponds to a solution to the wire problem with total wirelength equal to the cost of the matching

(8) optimal prefix codes (Data compression)

Suppose we have a long text file that we want to store compactly so that we can easily (fast) retrieve the text again

We want to do this by coding each character as a binary string so that all occurrences of the same character get the same code

Easy method: use a fixed length binary string for each character and use different strings for different characters. Then if there are N characters in the file and each character is coded using k bits, we use Nk bits to code the file.

Can we do better?

prefix codes: assign a binary string $b(x)$ to each of the different characters of the text in such a way that no code is a prefix of another, that is if x, y are distinct characters then there is no binary string b_1 s.t. $b(y) = b(x)b_1$

and no binary string b_2 s.t. $b(x) = b(y)b_2$

Ex: we cannot have $b(x) = 110$ and $b(y) = 11$

since then we can take $b_2 = 0$

Note that we want characters that occur often in the text to have short codes, whereas than that occur seldom can have longer codes

Suppose the file has 100000 characters, all from $\{a, b, c, d, e, f\}$ with the following frequencies:

a: 45%, b: 13%, c: 12%, d: 16%, e: 9%, f: 5%

and used the following two codes: a fixed length code with 3 bits per character and a prefix code with variable lengths as shown next page

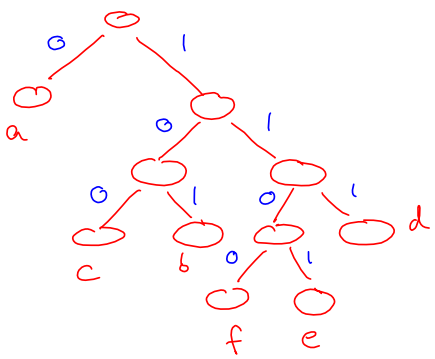
	a	b	c	d	e	f
Frequency	45	13	12	16	9	5
Fixed length codeword	000	001	010	011	100	101
Variable length codeword	0	101	100	111	1101	1100

The blue code is a prefix code and uses

$$\frac{45 \cdot 1 + (13+12+16) \cdot 3 + (9+5) \cdot 4}{100} = 2{,}24 \text{ bits pr character}$$

So it uses 224000 bits to code the text, when the fixed length code used 300000 !!

Prefix codes $\Leftrightarrow$ binary trees: If we draw a binary tree with the characters as leaves, then we can construct a prefix code by labelling the edges 0, 1 as shown below



Code for e is obtained by reading the bits on the path from the root to the leaf e:

1101

This gives the blue code in the table.

Greedy algorithm for making a prefix code: Huffman codes

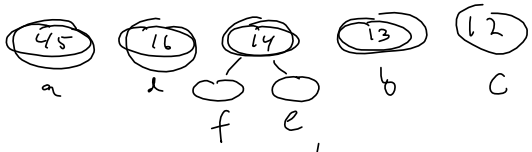
1. Make a vertex for each character in the text and label each vertex x with the name of the character and its frequency (call this the value $v(x)$ of x)

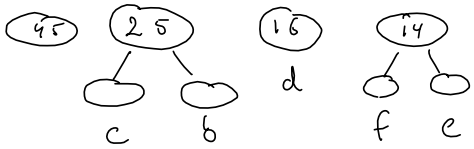
2. As long as there are at least two vertices a, b with $v(a), v(b) > 0$ do
   - pick the two vertices x, y s.t $v(x) \leq v(y)$ and $v(y) \leq v(w)$ for all $w \notin x, y$
   - Add a new vertex z with x, y as children and set $v(x) = v(y) = 0$, $v(z) = v(x) + v(y)$

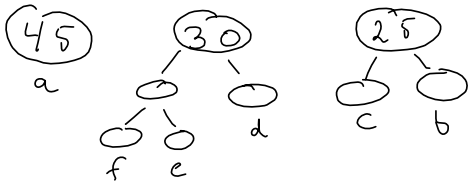
3. Label the edges of the final binary tree T by $0, 1$ as above and return the corresponding prefix code.

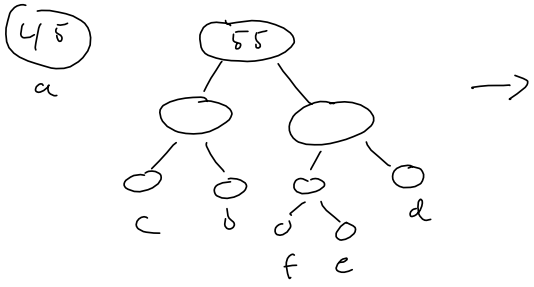
Example

	a	b	c	d	e	f
Frequency	45	13	12	16	9	5
Fixed length codeword	000	001	010	011	100	101
Variable length codeword	0	101	100	111	1101	1100

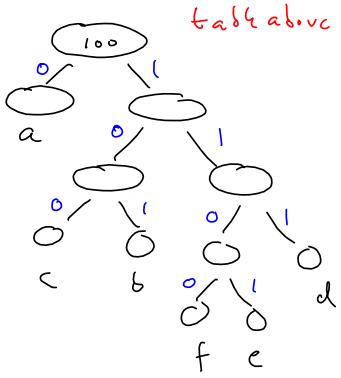








Gives the same code as the second in the table above



Theorem The greedy algorithm always produces an optimal prefix code for the given text, that is, the total number of bits used is minimum among all prefix codes.

See DM507